On Architectures and Training Techniques for Neural Networks

Sheheryar Zaidi

Somerville College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2023

Abstract

The progress in deep learning over the past decade has led to the creation of models that have had a significant impact on various fields as widely ranging as biology and language. Powering this progress are methodological advances in the design of neural networks and their training procedures, alongside the availability of more data and better software and hardware.

This thesis presents four pieces of research devoted to understanding and improving the generalization of neural networks via the lenses of neural architectures and their training techniques. On the architecture design front, we introduce a new equivariant architecture based on attention, and we develop algorithms for automatically designing architectures for use in an ensemble. On the training techniques front, we propose a denoising technique for self-supervised pre-training of molecular property prediction models, and we study an optimization technique for improving generalization via periodic re-initialization of the neural network during training. Each work touches on various aspects of the overall workflow for training a neural network.

Acknowledgements

First, I would like to express my tremendous gratitude to Yee Whye. Since introducing me to deep learning in 2018, Yee Whye's guidance has been instrumental to my growth as a researcher. His breadth of knowledge often taught me where to look, his encouragement to pursue one's curiosity allowed me to develop my interests, and, above all, his optimism and kindness kept me going through the peaks and troughs of research.

There are so many others in the research community from whom I have learned a lot. I would like to thank Arnaud Doucet, Chris Holmes, Dan Ciubotaru, Frank Hutter, James Martens, Jasper Snoek, Jonathan Godwin, Jonathan Marchini, Mason Porter, Peter Battaglia, Razvan Pascanu, Tom Rainforth and various others at Oxford and DeepMind for their mentorship and support. On the personal side, I would like to thank Arber Zela, Bobby He, Bryn Elesedy, Emilien Dupont, Hyunjik Kim, Jean-Francois Ton, Jin Xu, Jonathan Tam, Michael Hutchinson and Michael Schaarschmidt for their friendship in research and life.

Thank you to Aker Scholarship, particularly Bjørn Blindheim for encouraging me to pursue a doctorate in the first place, and the Google PhD Fellowship for their support in realizing this project and for connecting me to two incredible communities of researchers.

Finally, I owe enormous gratitude to my family – to my parents Arshad and Wajeeha, for constant love, encouragement and always being in my corner, and to Rija and Mashaal, for advice, camaraderie and humour (although often at my expense). Thank you to Gaia, for years of being loved, heard and understood, for clarity and candour, and, most importantly, for introducing me to Ailynn.

Contents

1 Introduction

Over the past decade, the field of deep learning has produced a series of astonishingly capable neural network models that have impacted a wide range of fields. Neural networks have transformed computer vision to the extent that they are now the only class of machine learning models capable of achieving the current state-of-the-art at multiple tasks. In the game of Go, neural networks combined with reinforcement learning have reached super-human performance, which was long considered a major challenge for artificial intelligence [Silver et al., 2016]. Neural networks have revolutionized generative modelling, emerging, for instance, as the only class of models capable of generating realistic high-resolution images [Rombach et al., 2021]. The unprecedented accuracy in the prediction of protein structures has led neural networks to transform structural biology [Jumper et al., 2021]. Through demonstrating competence at a diverse array of tasks, neural networks applied to language modelling have even been likened to prototypes of artificial general intelligence [Bubeck et al., 2023].

One lesson this incredible progress has taught us is that the details matter. The details of the models and the training procedures used to create them have a large impact on the generalization performance and utility of the resulting systems. For neural networks, the two methodological ingredients that largely determine the learned model and its properties are the neural network's architecture and the precise procedure used to train it. The architecture of a neural network is how

the model's parameters are combined with its input to produce its output. The training procedure is the recipe that converts a dataset of training examples into the learned parameters of the architecture. More or less every breakthrough due to deep learning (in particular, each one mentioned above) has relied on advances in neural network architectures or techniques for training them. These range from simple but crucial details like residual connections in the architecture to intricate denoising schedules for training diffusion models. It is difficult to imagine any of the remarkable progress of the field without such advances.

This thesis is about understanding and improving generalization in neural networks through a focus on architectures and their training techniques. We present four works of research on the topics of pre-training, re-initialization during optimization, equivariant architectures and architectural diversity in ensembles. Together, these works cover a broad range of application areas as we empirically assess different methodologies for their benefits and limitations. The thesis is outlined as follows.

Chapter 2 begins by introducing fundamental concepts in machine learning that appear throughout this thesis. We discuss the motivation for deep learning within machine learning and conclude with a whistle-stop tour of some of the key methodological advances in deep learning—along the two themes of architectures and training techniques—over the past decade.

Chapter 3 introduces a technique for pre-training neural networks for the task of molecular property prediction from 3D structures. The technique relies on denoising applied to an unlabelled dataset of molecular structures, which is proven to be related to learning atomic forces via the framework of score-matching. We empirically evaluate the method on multiple molecular property prediction benchmarks and demonstrate improved performance as a result of pre-training, mirroring the progress of large-scale pre-training in the vision and language domains.

Chapter 4 studies an optimization technique that involves periodically re-initializing the neural network's weights during training. We study when re-initialization methods improve generalization in the context of image classification,

finding surprising settings when such methods have a regularizing effect on learning that leads to better performance.

Chapter 5 focuses on the topic of equivariant neural network architectures. A new architecture based on the transformer model is introduced that is by construction equivariant to the action of certain groups. We empirically evaluate the architecture on point cloud classification, modelling physical systems and molecular property prediction, demonstrating the generalization benefit of both equivariance and the architecture.

Chapter 6 is about neural network ensembles, a common technique for improving generalization, uncertainty estimation and robustness in deep learning. We develop a framework for exploiting the predictive diversity offered by different neural network architectures to build ensembles with varying architectures. We demonstrate the empirical benefits of this approach in vision, which include both better generalization and improved robustness.

Chapters 3 to 6 were published as the following papers respectively. My contributions to each paper are described at the end of the respective chapter, except for Chapter 5, where they are covered in Section 5.7.1. Asterisks (*) indicate equal contribution.

- **Pre-training via Denoising for Molecular Property Prediction**. Sheheryar Zaidi*, Michael Schaarschmidt*, James Martens, Hyunjik Kim, Yee Whye Teh, Alvaro Sanchez-Gonzalez, Peter W. Battaglia, Razvan Pascanu and Jonathan Godwin. *International Conference on Learning Representations*, 2023.
- **When Does Re-initialization Work?** Sheheryar Zaidi*, Tudor Berariu*, Hyunjik Kim, Jorg Bornschein, Claudia Clopath, Yee Whye Teh and Razvan Pascanu. *Proceedings on "I Can't Believe It's Not Better! - Understanding Deep Learning Through Empirical Falsification" at NeurIPS 2022 Workshops*, PMLR, 2023.[1]
- **LieTransformer: Equivariant Self-Attention for Lie Groups**. Michael Hutchinson*, Charline Le Lan*, Sheheryar Zaidi*, Emilien Dupont, Yee Whye Teh and Hyunjik Kim. *International Conference on Machine Learning*, 2021.
- **Neural Ensemble Search for Uncertainty Estimation and Dataset Shift**. Sheheryar Zaidi*, Arber Zela*, Thomas Elsken, Chris Holmes, Frank Hutter and Yee Whye Teh. *Neural Information Processing Systems*, 2021.

[1]Originally presented at a workshop, this paper was subsequently selected for publication in a special edition of PMLR.

2
Background

Contents

2.1 Fundamentals of Machine Learning

The field of *machine learning* (ML) is broadly occupied with the goal of creating algorithms that can learn models from data. The ultimate goal of these models is to complete *tasks* through learning directly from data, rather than through explicit instruction. One example of a task is predicting an object's identity from its image. Another example is predicting the energy of a molecule from its atomic structure.

Models can broadly be categorized as being *discriminative* or *generative*. Informally, discriminative models are learned in order to make predictions – given an input, the model predicts the corresponding output. Generative models aim to learn the data-generating distribution, the purpose of which can, for example, be to generate new examples from the distribution. Although the distinction between discriminative and generative models is useful, it is important to note that it is not a strict dichotomy. For example, a discriminative model that predicts a distribution over the outputs can be viewed as an input-conditioned generative

model. Whereas, many generative models are trained using discriminative loss objectives but are used for generation.

Data used to train models, called the *training dataset*, comes either in the form of a set of pairs $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ – which are considered as pairs of an input $\mathbf{x}_i$ with output or label y_i – or simply a set of unlabeled examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Learning algorithms are then functions that map a given training dataset to a model from the set of all permissible models, called the *hypothesis class*.

2.1.1 Empirical Risk Minimization

The primary focus of this thesis is on discriminative models trained using labeled datasets, which is formalized as follows. Notationally, let

$$\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$$

be the training dataset, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output spaces respectively. We assume that each $(\mathbf{x}_i, y_i)$ has been generated by identically and independently (i.i.d.) sampling an unknown probability distribution P on $\mathcal{X} \times \mathcal{Y}$. Let $\mathcal{H} \subseteq \{f : \mathcal{X} \to \mathcal{Y}\}$ denote the hypothesis class, which contains functions $\mathcal{X} \to \mathcal{Y}$. Denote by $\ell(y, y') \in \mathbb{R}^{\geq 0}$ the loss associated with predicting the label y when the true label is y', which is zero when $y = y'$. We focus on learning algorithms that pick a function $f^* \in \mathcal{H}$ by *empirical risk minimization* (ERM):

$$f^* \in \arg\min_{f \in \mathcal{H}} R_{\text{train}}(f), \tag{2.1}$$
$$\text{where } R_{\text{train}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i).$$

The objective function R_{train} minimized in Equation (2.1) is called the *training loss* or *empirical risk*, as it is computed over the training data $\mathcal{D}_{\text{train}}$. However, a central goal is that the learned function f^* does not only have a small loss over the training examples $(\mathbf{x}_i, y_i)$ but also new, unseen examples $(\mathbf{x}, y) \sim P$. In other words, the learned function should *generalize* to new examples. Formally, what we actually want is to achieve low *population risk*

$$R_{\text{pop}}(f) = \mathbb{E}\left[\ell(f(\mathbf{x}), y)\right] = \int \ell(f(\mathbf{x}), y) \, \mathrm{d}P(\mathbf{x}, y).$$

However, computing R_{pop} for an arbitrary $f \in \mathcal{H}$ is intractable for various reasons, foremost of which is that the data generating distribution P is typically unknown such that computing the expectation is not possible. We now observe that ERM minimizes a tractable objective function that is a Monte Carlo approximation of the population risk that we care about, requiring only access to the samples in $\mathcal{D}_{\text{train}}$ rather than knowledge of P:

$$\frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) \approx R_{\text{pop}}(f), \tag{2.2}$$

with equality as $n \to \infty$ by the law of large numbers [Çinlar, 2011, Section 6 in Chapter 3], since $(\mathbf{x}_i, y_i)$ are samples from P. Overall, one hopes that minimizing $R_{\text{train}}(f)$ goes at least some of the ways towards minimizing $R_{\text{pop}}(f)$. Theoretically quantifying when this happens is the subject of study in the field of learning theory [Vapnik, 1999].

2.1.2 Overfitting

It is key that the result of learning is a model that generalizes to unseen data. This raises the question of how we measure whether ERM has been successful – is $R_{\text{pop}}(f^*)$ going to be small? The learned function f^* of course depends on the training data $\mathcal{D}_{\text{train}}$ due to Equation (2.1). Therefore, its training loss $R_{\text{train}}(f^*)$ is not necessarily a good estimate of $R_{\text{pop}}(f^*)$. Indeed, the law of large numbers cannot be applied as in Equation (2.2), because f^* depends on all $(\mathbf{x}_i, y_i)$ and the training loss is also computed over $(\mathbf{x}_i, y_i)$. Usually, $R_{\text{train}}(f^*) \leq R_{\text{pop}}(f^*)$. Loosely speaking, when the gap between these two is large, we say that learning has *overfitted* to the training data.

Cross validation is a widely used method for estimating $R_{\text{pop}}(f^*)$. Using a validation dataset $\mathcal{D}_{\text{val}} = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_{n'}, y'_{n'})\}$ of new i.i.d. samples from P, we note that

$$R_{\text{val}}(f^*) = \frac{1}{n'}\sum_{i=1}^{n'} \ell(f^*(\mathbf{x}'_i), y'_i)$$

is an unbiased estimate of $R_{\text{pop}}(f^*)$, because f^* does not depend on $\mathcal{D}_{\text{val}}$. In practice, $\mathcal{D}_{\text{val}}$ is constructed by randomly splitting all available training data into two sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$. To summarize the discussion so far, learning using ERM in practice breaks down into the following steps:

1. Select a hypothesis class $\mathcal{H}$, loss function ℓ and split all available training data for the task into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$.

2. Perform the optimization in Equation (2.1) to pick $f^* \in \mathcal{H}$.

3. Report $R_{\text{val}}(f^*)$ as an estimate of how well f^* generalizes on the task.

2.1.3 Regularization

A crucial part of learning is choosing the hypothesis class $\mathcal{H}$. A trade-off exists when making this choice between the expressivity of $\mathcal{H}$ and the potential for overfitting. If $\mathcal{H}$ is a large set of expressive functions such that there exist $f \in \mathcal{H}$ with $R_{\text{train}}(f) \approx 0$, ERM may result in learned functions that do not generalize beyond the training data. More explicitly, suppose $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ and consider $\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} y_i \varphi(\mathbf{x} - \mathbf{x}_i)$ where $\varphi : \mathbb{R} \to \mathbb{R}$ is some smooth "bump" with $\varphi(0) = 1$ and compact support on a small neighborhood of 0. Provided the bump function has sufficiently small support, $\hat{f}(\mathbf{x}_i) = y_i$ for all $i = 1, \ldots, n$, hence $R_{\text{train}}(\hat{f}) = 0$. If functions such as $\hat{f}$ are contained in $\mathcal{H}$ and returned by the optimization, then for any realistic task and dataset, ERM is unlikely to generalize as it will result in the learned model only memorizing the training examples. On the other extreme, if $\mathcal{H}$ only contains, say, constant functions, it will lack the flexibility necessary for ERM to select a model that can even minimize the training loss, let alone generalize to unseen data. Generally, it is desirable that $\mathcal{H}$ is chosen to be expressive enough to model the task while avoiding the risk of overfitting, which is the crux of applying machine learning in practice. In deep learning, one reason why architecture choice is crucial is because it determines the hypothesis class.

The best way to reduce overfitting is to increase the amount of training data – use a sufficiently expressive hypothesis class and let the data inform the model

choice. When increasing the data is expensive or otherwise infeasible, overfitting can be avoided for expressive choices of $\mathcal{H}$ by applying *regularization*, which encourages the learned model to be "simple" in the hope of better generalization (an instance of applying Occam's razor). Let $\mathcal{R} : \mathcal{H} \to \mathbb{R}$ be a regularization function that measures some notion of how simple a function $f \in \mathcal{H}$ is such that smaller values of $\mathcal{R}$ are preferred (all else being equal). The optimization in Equation (2.1) can be modified as follows to explicitly encourage the learning algorithm to select simpler functions from the hypothesis class:

$$f^* \in \arg\min_{f \in \mathcal{H}} R_{\text{train}}(f) + \lambda \mathcal{R}(f). \tag{2.3}$$

$\lambda > 0$ is the regularization coefficient, which is a hyperparameter that determines the strength of regularization. Returning to our example of $\hat{f}$ above, suppose $\mathcal{H}$ is some set of smooth functions that also contains $\hat{f}$ and $\mathcal{R}(f) = \sup f'$, which encourages the learned function to vary more slowly.[1] Then, functions constructed with bump functions φ that have narrow support sets would incur large costs via $\mathcal{R}(f)$, hence they would be avoided by ERM with regularization.

The choice of $\mathcal{R}$ is an important area of research in machine learning. Whether a given $\mathcal{R}$ improves generalization is linked to the task and dataset as well. For example, when $\mathcal{H}$ is parameterized by some $\boldsymbol{\theta} \in \mathbb{R}^p$, as in the case of deep learning, a common choice of regularizer is the ℓ_2-norm of $\boldsymbol{\theta}$, called *weight decay*. However, whether weight decay actually improves generalization when training neural networks in practice often depends on the data, task and network architecture.

Equation (2.3) regularizes learning *explicitly*, because the user directly defines $\mathcal{R}$ and chooses λ. Learning can also be regularized *implicitly*, which is especially the case when training neural networks with gradient-based methods (see *e.g.* Neyshabur [2017]). Such optimization methods implicitly bias the solution of ERM, which itself is believed to be a source of regularization that is not controlled explicitly. Understanding the role of implicit regularization in training neural networks is a highly active line of research. Various directions have been studied, such as the

[1] Assuming $\sup f'$ exists for all $f \in \mathcal{H}$.

curvature of the minima in parameter space of $\boldsymbol{\theta}$ obtained with gradient-based methods [*e.g.* Hochreiter and Schmidhuber, 1994, Dinh et al., 2017, Keskar et al., 2017, Jiang* et al., 2020, Foret et al., 2021] and the properties of the learned mapping in function space [*e.g.* Rahaman et al., 2018, Basri et al., 2019, Xu et al., 2019, Chizat and Bach, 2020].

2.1.4 Loss Functions and Probabilistic Modelling

Loss functions ℓ usually result from adopting a probabilistic viewpoint of learning. In many practical settings, the ERM optimization in Equation (2.1) is equivalent to computing a *maximum likelihood estimator* (MLE) under some probabilistic model of the data, which then determines the loss function. Suppose we assume the following model for the data:

$$Y \mid \mathbf{X} = \mathbf{x} \sim P_{\boldsymbol{\theta}}(Y \mid \mathbf{X} = \mathbf{x}),$$

where $\boldsymbol{\theta} \in \Theta$ is an unknown parameter and $P_{\boldsymbol{\theta}}(\cdot \mid \mathbf{X} = \mathbf{x})$ is a distribution on $\mathcal{Y}$ parameterized by $\mathbf{x}$. Given observations $\mathcal{D}_{\text{train}}$, the MLE corresponds to the following objective:

$$\boldsymbol{\theta}_{\text{MLE}} \in \arg\max_{\boldsymbol{\theta}\in\Theta} \prod_{i=1}^{n} P_{\boldsymbol{\theta}}(y_i \mid \mathbf{X} = \mathbf{x}_i) = \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{n} \log P_{\boldsymbol{\theta}}(y_i \mid \mathbf{X} = \mathbf{x}_i). \tag{2.4}$$

Viewing $f_{\boldsymbol{\theta}}(\mathbf{x}) = P_{\boldsymbol{\theta}}(\cdot \mid \mathbf{X} = \mathbf{x})$ (the output is now a distribution on $\mathcal{Y}$ rather than a point estimate) and letting $\ell(f_{\boldsymbol{\theta}}(\mathbf{x}), y) = -\log P_{\boldsymbol{\theta}}(y \mid \mathbf{X} = \mathbf{x})$, we observe that Equation (2.4) is an equivalent optimization to Equation (2.1) over $\mathcal{H} = \{f_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta\}$. In regression tasks where Y is continuous, mean squared error (MSE) is a common loss function, which is the result of assuming the model $P_{\boldsymbol{\theta}}(y \mid \mathbf{X} = \mathbf{x}) = \mathcal{N}(y; \mu_{\boldsymbol{\theta}}(\mathbf{x}), \sigma^2)$, where the mean is a function of $\mathbf{x}$ parameterized by $\boldsymbol{\theta}$ and the variance is fixed. For classification tasks where Y is discrete taking one of C values, $P_{\boldsymbol{\theta}}(y \mid \mathbf{X} = \mathbf{x})$ is a categorical distribution with probability mass function $\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^C$, which results in the cross-entropy loss. These are two of the simplest and most common losses used in practice.

Recently, in the deep learning literature, various works have however found that augmenting these traditional loss functions with auxiliary terms can significantly

improve generalization. For example, Santoro et al. [2018] used an auxiliary loss that encourages representation learning in the context of abstract reasoning with neural networks, Jumper et al. [2021] utilized multiple auxiliary losses for protein structure prediction and Godwin et al. [2022] used an auxiliary denoising loss in the context of molecular property regression, which we build on in Chapter 3.

2.1.5 Ensembles

Given multiple learned functions $f_1, \ldots, f_m$, *ensemble learning* or *ensembling* is a technique for producing a single learned function f by combining $f_1, \ldots, f_m$, with the goal of f generalizing better than any of its constituents f_i. Ensembling is employed in many real-world applications of machine learning, because of its generalization advantages and ease of implementation for a variety of model types [Zhou, 2012].

The constituents of an ensemble, called the *base learners*, can a priori be any type of function. However, usually, they tend to come from some class of *randomized* learning algorithms (*i.e. homogeneous* ensembles), where the outcome of learning is random. This ensures that each time the learning algorithm is called, a different base learner f_i is returned. For deterministic learning algorithms, where all f_i would be the same, combining them into an ensemble (*e.g.* through averaging the outputs or voting) would yield a model identical to and no better than each f_i. This also relates to the intuition behind the generalization benefit of ensembling: provided that the base learners are each sufficiently accurate and mutually diverse, their errors will be different and can therefore cancel out. Various works have made this idea precise theoretically for specific types of ensembles [*e.g.* Hansen and Salamon, 1990, Breiman, 2001a, Bian and Chen, 2019, Goodfellow et al., 2016].

Randomization can be achieved in a variety of ways, and the optimal approach for generalization tends to depend on multiple factors, including the hypothesis class or model type and the dataset. Arguably, the most popular usage of ensembling is for building random forests [Breiman, 2001a], which are ensembles of decision trees [Breiman et al., 1984]. One source of randomization used in ensembles of decision trees is bootstrap aggregation, wherein the training data for each base

learner is constructed by sampling with replacement from the full training data. For neural networks, which are trained using gradient descent, a natural source of randomization is the initialization of the parameters. This method is usually called *deep ensembles* [Lakshminarayanan et al., 2017]. Due to the non-convex structure of the loss landscape, different initializations lead to different minima that represent different functions. While bootstrapping can also be applied to ensembles of neural networks, deep ensembles empirically outperform bootstrapping [Nixon et al., 2020].

In deep learning, aside from improved generalization accuracy, one major advantage of ensembling is better *uncertainty estimation*, which is one of the reasons they have been explored in recent literature. Chapter 6 builds upon and improves deep ensembles, both in terms of generalization and uncertainty estimation, by exploiting another source of diversity: the network architecture.

2.1.6 Faces of Generalization

In Section 2.1.1, we discussed that the goal of learning is to generalize on unseen data – the learned model should exhibit a small population risk. Population risk is measured over data sampled from the same underlying distribution as the training data. In practice, however, the desiderata for useful models extend beyond small population risk. Ideally, a model should generalize, to some extent, on examples outside the distribution of the training data, and it should give reliable estimates of its uncertainty when making predictions.

Recall that P is the distribution from which the training data is sampled. *Out-of-distribution generalization* refers to generalization on examples sampled from another distribution P'. For example, within vision, P may be a distribution over images taken on sunny days, while P' is a distribution of images taken on rainy days. Image classification models trained on examples from P should ideally generalize on examples from P'. As another example, P may be a distribution over molecular structures whose center of mass is zero, whereas P' is a distribution over similar structures that are instead translated and have a non-zero center of mass. Depending on the specific application, P' may not be known at training time. In the

vast majority of cases, both P and P' are also not known directly but only through their samples. Quantifying how generalization depends on the formal relationship between these two distributions is an area of active research [Redko et al., 2020].

One approach for building models that generalize well both in-distribution and are robust to some degree of distributional shift at test time is via incorporating *inductive biases* in the learning algorithm. The (somewhat overloaded) term inductive biases typically refers to incorporating into the model prior knowledge known about the target input-output mapping that we wish to learn. In deep learning, one way to achieve this is through the choice of network architecture (which determines the hypothesis class $\mathcal{H}$). An example of such an inductive bias is constructing architectures that are *equivariant* to the action of a group on the input and output spaces, in line with the target input-output mapping. Equivariant neural networks provably guarantee robustness to particular distributional shifts. Consider the example of a rotation-invariant model used for classifying images of cats and dogs. Such a model trained on only upright images of cats and dogs will generalize equally well to other distributions where the images are in any other orientation. Equivariance forms the subject of Chapter 5.

When models are deployed in the real world, their utility also depends on whether they can provide reliable estimates of their uncertainty. For data from distributions that are sufficiently distinct from the training data, models will inevitably make poorer predictions. Predictive uncertainty estimates that can indicate this, therefore, make models more useful and trustworthy, which is key for applications to critical domains. In classification tasks, where the model predicts a categorical probability distribution over the classes $\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^C$, one common measure of the quality of a model's predictive uncertainty is *expected calibration error* (ECE). ECE measures the mismatch between a model's confidence and its accuracy. For example, whenever a model predicts that an input belongs to a specific class with probability 70%, is it actually right 70% of the times? In regression tasks where a model returns a point estimate with a prediction interval, the attained coverage of the intervals is a measure of their quality. We note that conformal prediction is a general

methodology for constructing such prediction intervals (for classification as well) that come with theoretical, distribution-free guarantees. Due to this, conformal methods have recently gained popularity (see Angelopoulos and Bates [2021] for a recent introduction and review).

2.2 Deep Learning: Past & Present

In this section, we focus our attention on deep learning, which forms the subject of this thesis. Fundamentally, a neural network is simply a function $f_{\boldsymbol{\theta}}$ parameterized by a real vector $\boldsymbol{\theta} \in \mathbb{R}^p$. The input and output spaces of a neural network depend on the task, but when implemented computationally, these spaces are either finite-dimensional real vector spaces or unions thereof (in cases such as graph neural networks, where the input size can vary). Neural networks are usually non-linear functions of the input and span a hypothesis class $\mathcal{H} = \{f_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \mathbb{R}^p\}$ that is typically expressive and satisfies some version of a universal approximation result. The functions $f_{\boldsymbol{\theta}}$ are usually a composition of multiple *layers*, which tend to be simpler functions like affine or point-wise non-linear functions.

ERM in Equation (2.1) applied to neural networks leads to a generally non-convex optimization problem over $\boldsymbol{\theta} \in \mathbb{R}^p$. Almost all methods used to solve this in practice are variants of gradient descent. In its simplest form, starting with an *initialization* $\boldsymbol{\theta}_0$, we improve the parameters by iterating through $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla R_{\text{train}}(f_{\boldsymbol{\theta}_t})$, where we recall that $\boldsymbol{\theta} \mapsto R_{\text{train}}(f_{\boldsymbol{\theta}})$ maps parameters to their training loss and the step-size $\alpha \in \mathbb{R}$ is called the *learning rate*. For large datasets where computing the gradient $\nabla R_{\text{train}}(f_{\boldsymbol{\theta}_t})$ is expensive, we replace the gradient of $R_{\text{train}}(f_{\boldsymbol{\theta}_t})$ with the gradient of the training loss over a random subset (or *batch*) of the full data. This replacement is an unbiased estimate of the full gradient, so it constitutes a version of stochastic gradient descent (SGD). There is a rich literature on optimization methods for neural networks [Schmidt et al., 2021].

Gradient descent will not necessarily find a global minimum of the training loss, although, in various practical instances, it does, despite which the model often continues to generalize well to unseen data [Zhang et al., 2021]. A complete

theoretical understanding of the interplay been optimization and generalization in neural networks remains a key open problem in the field. However, there is abundant empirical evidence that how well a deep learning model generalizes on a given task is sensitively determined by:

1. *What* the neural network's architecture is – what is the mapping $\boldsymbol{\theta} \mapsto f_{\boldsymbol{\theta}}$?

2. *How* the network is trained – what is the learning protocol and the data?

Answering these two questions in specific settings is the bread and butter of applying deep learning. In the following sections, we first briefly review the challenges that motivated the development of deep learning. Then, we give a high-level overview of the key methodological advances in deep learning from the past decade – connecting them back to the above two themes of architecture design and training techniques. Our discussion focuses on specific methods within these themes that form the basis of the remaining chapters in this thesis.

2.2.1 Why Deep Learning

The development of deep learning was partly motivated by two growing needs. The first was a desire to reduce manual feature engineering in machine learning. The majority of machine learning methods prior to deep learning required hand-crafted features to be extracted from raw data examples and provided as input to the models. For example, image classification models using decision trees or support vector machines are usually given a Gabor-filtered version of the input image as an input feature instead of the raw pixel values. Hand-crafted features do not necessarily generalize across tasks, require domain expertise to design and may be suboptimal. Instead, neural networks used for image classification are provided the raw pixel values (possibly standardized) as input and the features extracted from the image are *learned* end-to-end. Similarly, early models for speech recognition relied on feeding handcrafted features to Hidden Markov models (HMMs) and Gaussian mixture models (GMMs). Eventually, this was also replaced by neural networks learned end-to-end which were found to outperform prior approaches. On the

whole, the use of deep learning circumvents the expensive procedure of constructing task-specific features and renders the methodology more applicable across domains.

The second was a need for methods where the computation required for learning scales favorably with the dataset size, enabling them to be applied to large datasets. For example, the computational complexity of kernel regression scales super-linearly with the dataset size, due to the need for inverting the $n \times n$ Gram matrix, where n is the size of the dataset. Similarly, fitting a decision tree to a dataset of size n also scales super-linearly, because deciding a splitting point for a node in the tree involves sorting the n examples by one of their (numerical) features, which is a super-linear operation. This restricts the applicability of such methods to large datasets (without further approximations).

The cost of training a neural network can be chosen to scale linearly with n in practice. When applying batched gradient descent with batch size m, n/m steps of gradient descent are needed for one full pass over the dataset, which is called an *epoch* of training. If the number of epochs is kept fixed, the cost of training scales linearly with n. Generally, observe that *batched* gradient-based optimization even allows some extent of decoupling between the dataset size and the computational cost of training. Indeed, as usually done in practice, the batch size is chosen independently of the dataset size, instead being chosen as the largest possible size supported by the available hardware [Godbole et al., 2023], which means that the overall cost of training depends on the total number of gradient steps. This is then chosen by taking into account the desired total training time and possibly the dataset size among other factors. Deep learning has so far remained unique in its successful application to very large datasets, such as trillions of tokens of natural text or billions of images [Chowdhery et al., 2022, OpenAI, 2023, Schuhmann et al., 2022]

2.2.2 Progress of the Last Decade

The origins of deep learning date back to the 1940s, and the field has since undergone multiple waves of development under different names (for a historical overview, see Goodfellow et al. [2016, Section 1.2]). The latest wave of development followed the

dramatic success of convolutional neural networks (CNNs) at the annual object recognition contest ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. Krizhevsky et al. [2012] applied a convolutional architecture known as AlexNet to the task of classifying images as belonging to one of 1,000 classes, achieving a significantly lower error rate than the best models in the previous years (and also compared to the runner-up in 2012). Key to their success was using a sufficiently *deep* convolutional architecture whose training was made computationally feasible through the use of graphics processing units (GPUs).

Since then, many methodological advances have been made through the community's focus on improving performance on benchmarks for different tasks and application domains. This progress has been powered by increases in the availability of large amounts of data and efficient computational hardware and software. We give an incomplete, whistle-stop tour of influential methodological advances, categorized by the application domains where deep learning has been highly successful.

In computer vision, convolutional neural networks (CNNs) [LeCun et al., 1989, Lecun et al., 1998] have been a dominant architecture. For classification tasks, these models take as input an array consisting of raw pixel values and map it to a discrete probability distribution over the class labels using a sequence of learned convolution layers. Following AlexNet in 2012, various new architectures were proposed that led to winning entries of subsequent ILSVRCs. An early common theme was scaling up the size of the CNN by increasing its number of layers [Simonyan and Zisserman, 2015, Szegedy et al., 2015] (which were some combination of linear convolutional layers, point-wise non-linearities, pooling layers and fully-connected layers). New types of layers were also proposed, such as batch normalization [Szegedy et al., 2015] that improved training speed and stability. He et al. [2016c] proposed residual connections that allowed scaling CNNs to over a hundred layers deep, further improving their performance on vision tasks. Normalization layers and skip connections have since become ubiquitous in modern architectures for vision and beyond. Increasing model size also increases the risk of overfitting, for which techniques like dropout [Srivastava et al., 2014] during training were proposed.

Modern SOTA vision models almost always rely on transfer learning – these models are initially pre-trained on billions of images (*e.g.* using a classification or contrastive loss) and then fine-tuned on downstream tasks such as image segmentation, video classification and image captioning [Dehghani et al., 2023].

In natural language processing (NLP) and speech, earlier successes of deep learning were in speech recognition. Following a similar trajectory to vision, CNN architectures were adapted to operate on log-Mel spectrograms allowing them to be applied to speech recognition and went through a similar cycle of architectural improvements and depth scaling resulting in better performance on benchmarks [Sainath et al., 2013] Meanwhile, other NLP tasks such as language modelling and machine translation require architectures that can operate on sequences of tokens (*e.g.* words or characters). Architectures based on recurrent neural networks (RNNs), such as long short-term memory (LSTM) networks and gated recurrent units (GRUs), were constructed to enable sequence-to-sequence learning [Hochreiter and Schmidhuber, 1997, Cho et al., 2014, Sutskever et al., 2014]. The major architectural breakthrough was due to Vaswani et al. [2017] who proposed a recurrence-free, attention-based model called the *transformer* and demonstrated its SOTA performance on machine translation and other tasks. Soon after, large transformers were applied to autoregressive language modelling on enormous internet-sized datasets of text. The motivating factor was empirical scaling laws which showed clear power law trends of improving model performance as model and data sizes were increased. This resulted in several, increasingly capable large language models (LLMs) [Devlin et al., 2019, Brown et al., 2020b, OpenAI, 2023, Chowdhery et al., 2023, Google, 2023] that for the first time demonstrated strong few-shot, multi-task learning abilities, outperformed task-specific models and are currently considered by many as prototypes of artificial general intelligence. Training large transformers for language modelling itself required innovations at various levels, such as optimization-related techniques, tokenization and positional embeddings, efficient variations and implementations of self-attention, and model parallelization

strategies. Recently, much focus has been devoted to methods for instruction tuning with human data in order to better align pre-trained LLMs with the user's intent.

In machine learning for scientific applications, the biggest catalyst of deep learning and the natural sciences was the second generation of AlphaFold [Jumper et al., 2021] which is a neural network that is capable of accurately predicting the 3D structure of a protein from its amino acid sequence – an important open problem in biology. Central to AlphaFold's design was a new architectural block called the Evoformer for embedding the multiple sequence alignment and pair-wise features using an attention-based mechanism. The architecture was trained with a novel procedure involving a combination of loss functions and distillation techniques as well as an additional module for predictive uncertainty estimation. Similarly, deep learning applied to other domains often also involves architectural innovation specific to the domain and task. Recent examples include DM21 for quantum chemistry [Kirkpatrick et al., 2021], the Learned Interpolation model for computational fluid dynamics [Kochkov et al., 2021], and Enformer for genetic expression prediction [Avsec et al., 2021]. More generally, graph neural networks (GNNs) that take as input featurized graphs are a popular family of architectures that are used for various scientific applications, because inputs in scientific domains (*e.g.* molecules or the states of a physical system) naturally lend themselves to a graph-based representation [Battaglia et al., 2018]. Most GNNs rely on variations of message passing layers where the features of a given node are updated based on the features of adjacent nodes and edges. Equivariance in neural networks has also been motivated by scientific applications where the coordinate system of the input is arbitrary.

Orthogonal to application areas, certain modelling paradigms have been successful across areas. A prominent case is transfer learning, which is a training technique that involves exploiting the knowledge learned from one task and dataset to improve performance on another task. In deep learning, this is usually done through pre-training, where a neural network is first trained on a typically large dataset with a generic task. The pre-trained model can then be fine-tuned on a related target task where data may be limited. Pre-training tasks may or may not require labelled

data. Language modelling in NLP and contrastive learning in vision are popular examples of pre-training tasks that require only unlabelled data. In Chapter 3, we introduce an analogue for pre-training models for molecular property prediction.

Generative modelling, which consists of learning the data-generating distribution, has also been transformed with the use of deep learning. High-dimensional distributions, such as high-resolution images, that were previously impossible to model have been successfully learned by large-scale neural networks combined with powerful modelling techniques. Early generative modelling approaches were generative adversarial networks [Goodfellow et al., 2014], variational auto-encoders [Kingma and Welling, 2014] and normalizing flows [Papamakarios et al., 2021]. Each of these techniques is a modelling framework that is most often implemented with neural networks used as learnable function approximators. They have been applied to multiple domains, particularly image generation. More recently, diffusion modelling [Ho et al., 2020, Song et al., 2021, Sohl-Dickstein et al., 2015] based on denoising score-matching has become the dominant approach for generative modelling, largely replacing other methods. Conditional generation, where the model learns to sample from the data distribution conditioned on some variable, has been highly successful with diffusion models, making generation more controllable and hence useful. Diffusion models have been used to build text-to-image systems, generate proteins and even model language. Note that each approach may be viewed as a method for formulating the learning problem that results in its unique training and inference procedures.

Finally, in the past few years, multimodality has been an important theme within deep learning, blurring the boundaries between application areas. Multimodal models combine different data modalities. The most common example currently is visual-language models that combine image and text data such that the inputs and/or outputs can be a combination of images and text. Dealing with multimodal data brings its own challenges, principally in picking a suitable parameterization and architecture for jointly embedding different modalities and an associated training objective. Current approaches tend to rely on convolutional and attention modules to

embed images and text respectively followed by modules that combine them. Various research efforts across academia and industry are currently working to extend the set of modalities towards video, audio, robotics and beyond. There are a number of other domains we have not discussed in this section that have also been transformed by the advent of deep learning, such as reinforcement learning and robotics.

3

Pre-training via Denoising for Molecular Property Prediction

Abstract: Many important problems involving molecular property prediction from 3D structures have limited data, posing a generalization challenge for neural networks. In this paper, we describe a pre-training technique based on denoising that achieves a new state-of-the-art in molecular property prediction by utilizing large datasets of 3D molecular structures at equilibrium to learn meaningful representations for downstream tasks. Relying on the well-known link between denoising autoencoders and score-matching, we show that the denoising objective corresponds to learning a molecular force field – arising from approximating the Boltzmann distribution with a mixture of Gaussians – directly from equilibrium structures. Our experiments demonstrate that using this pre-training objective significantly improves performance on multiple benchmarks, achieving a new state-of-the-art on the majority of targets in the widely used QM9 dataset. Our analysis then provides practical insights into the effects of different factors – dataset sizes, model size and architecture, and the choice of upstream and downstream datasets – on pre-training.

Contents

3.1 Introduction

The success of the best performing neural networks in vision and natural language processing (NLP) relies on pre-training the models on large datasets to learn meaningful features for downstream tasks [Dai and Le, 2015, Simonyan and Zisserman, 2014, Devlin et al., 2018, Brown et al., 2020a, Dosovitskiy et al., 2020b]. For molecular property prediction from 3D structures (a point cloud of atomic nuclei in $\mathbb{R}^3$), the problem of how to similarly learn such representations remains open. For example, none of the best models on the widely used QM9 benchmark use any form of pre-training [*e.g.* Klicpera et al., 2020a, Liu et al., 2022b, Schütt et al., 2021, Thölke and De Fabritiis, 2022], in stark contrast with vision and NLP. Effective methods for pre-training could have a significant impact on fields such as drug discovery and material science.

In this work, we focus on the problem of how large datasets of 3D molecular structures can be utilized to improve performance on downstream molecular property prediction tasks that also rely on 3D structures as input. We address the question: how can one exploit large datasets like PCQM4Mv2,[1] that contain over 3 million structures, to improve performance on datasets such as DES15K that are orders of magnitude smaller? Our answer is a form of self-supervised pre-training that generates useful representations for downstream prediction tasks, leading to state-of-the-art (SOTA) results.

Inspired by recent advances in noise regularization for graph neural networks (GNNs) [Godwin et al., 2022], our pre-training objective is based on denoising in the space of structures (and is hence self-supervised). Unlike existing pre-training methods, which largely focus on 2D graphs, our approach targets the setting where the downstream task involves 3D point clouds defining the molecular structure. Relying on the well-known connection between denoising and score-matching [Vincent, 2011, Song and Ermon, 2019, Ho et al., 2020], we show that the denoising objective is equivalent to learning a particular force field, adding

[1]Note that PCQM4Mv2 is a new version of PCQM4M that now offers 3D structures.

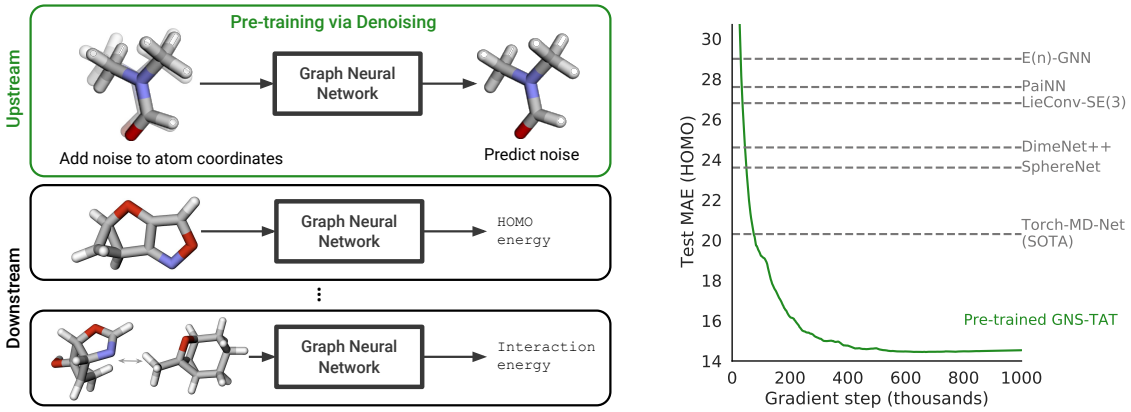


Figure 3.1: GNS-TAT pre-trained via denoising on PCQM4Mv2 outperforms prior work on QM9.

a new interpretation of denoising in the context of molecules and shedding light on how it aids representation learning.

The contributions of our work are summarized as follows:

- We investigate a simple and effective method for pre-training via denoising in the space of 3D structures with the aim of improving downstream molecular property prediction from such 3D structures. Our denoising objective is shown to be related to learning a specific force field.

- Our experiments demonstrate that pre-training via denoising significantly improves performance on multiple challenging datasets that vary in size, nature of task, and molecular composition. This establishes that denoising over structures successfully transfers to molecular property prediction, setting, in particular, a new state-of-the-art on 10 out of 12 targets in the widely used QM9 dataset. Figure 3.1 illustrates performance on one of the targets in QM9.

- We make improvements to a common GNN, in particular showing how to apply Tailored Activation Transformation (TAT) [Zhang et al., 2022] to Graph Network Simulators (GNS) [Sanchez-Gonzalez et al., 2020], which is complementary to pre-training and further boosts performance.

- We analyze the benefits of pre-training by gaining insights into the effects of dataset size, model size and architecture, and the relationship between the upstream and downstream datasets.

3.2 Related Work

Pre-training of GNNs. Various recent works have formulated methods for pre-training using graph data [Liu et al., 2021b, Hu et al., 2020a, Xie et al., 2021, Kipf and Welling, 2016], rather than 3D point clouds of atom nuclei as in this paper. Approaches based on contrastive methods rely on learning representations by contrasting different *views* of the input graph [Sun et al., 2019, Veličković et al., 2019, You et al., 2020, Liu et al., 2021a], or bootstrapping [Thakoor et al., 2021]. Autoregressive or reconstruction-based approaches, such as ours, learn representations by requiring the model to predict aspects of the input graph [Hu et al., 2020a,b, Rong et al., 2020, Liu et al., 2019b]. Most methods in the current literature are not designed to handle 3D structural information, focusing instead on 2D graphs. The closest work to ours is GraphMVP [Liu et al., 2021a], where 3D structure is treated as one view of a 2D molecule for the purpose of upstream contrastive learning. Their work focuses on downstream tasks that only involve 2D information, while our aim is to improve downstream models for molecular property prediction from 3D structures. After the release of this pre-print, similar ideas have been studied by Jiao et al. [2022] and Liu et al. [2022a].

Denoising, representation learning and score-matching. Noise has long been known to improve generalization in machine learning [Sietsma and Dow, 1991, Bishop, 1995]. Denoising autoencoders have been used to effectively learn representations by mapping corrupted inputs to original inputs [Vincent et al., 2008, 2010]. Specific to GNNs [Battaglia et al., 2018, Scarselli et al., 2009, Bronstein et al., 2017], randomizing input graph features has been shown to improve performance [Hu et al., 2020a, Sato et al., 2021]. Applications to physical simulation also involve corrupting the state with Gaussian noise [Sanchez-Gonzalez et al., 2018, 2020, Pfaff et al., 2020]. Our work builds on Noisy Nodes [Godwin et al., 2022], which

incorporates denoising as an auxiliary task to improve performance, indicating the effectiveness of denoising for molecular property prediction (*cf.* Section 3.3.2.2). Denoising is also closely connected to score-matching [Vincent, 2011], which has become popular for generative modelling [Song and Ermon, 2019, 2020, Ho et al., 2020, Hoogeboom et al., 2022, Xu et al., 2022, Shi et al., 2021]. We also rely on this connection to show that denoising structures corresponds to learning a force field.

Equivariant neural networks for 3D molecular property prediction. Recently, the dominant approach for improving models for molecular property prediction from 3D structures has been through the design of architectures that incorporate roto-translational inductive biases into the model, such that the outputs are invariant to translating and rotating the input atomic positions. A simple way to achieve this is to use roto-translation invariant features as inputs, such as inter-atomic distances [Schütt et al., 2017a, Unke and Meuwly, 2019], angles [Klicpera et al., 2020c,a, Shuaibi et al., 2021, Liu et al., 2022b], or the principal axes of inertia [Godwin et al., 2022]. There is also broad literature on equivariant neural networks, whose intermediate activations transform accordingly with roto-translations of inputs thereby naturally preserving inter-atomic distance and orientation information. Such models can be broadly categorized into those that are specifically designed for molecular property prediction [Thölke and De Fabritiis, 2022, Schütt et al., 2021, Batzner et al., 2021, Anderson et al., 2019, Miller et al., 2020] and general-purpose architectures [Satorras et al., 2021, Finzi et al., 2020, Hutchinson et al., 2021, Thomas et al., 2018b, Kondor et al., 2018b, Brandstetter et al., 2021]. Our pre-training technique is architecture-agnostic, and we show that it can be applied to enhance performance in both a GNN-based architecture [Sanchez-Gonzalez et al., 2020] and a Transformer-based one [Thölke and De Fabritiis, 2022]. We conjecture that similar improvements will hold for other models.

3.3 Methodology

3.3.1 Problem Setup

Molecular property prediction consists of predicting scalar quantities given the structure of one or more molecules as input. Each data example is a labelled set specified as follows: we are provided with a set of atoms $S = \{(a_1, \mathbf{p}_1), \ldots, (a_{|S|}, \mathbf{p}_{|S|})\}$, where $a_i \in \{1, \ldots, 118\}$ and $\mathbf{p}_i \in \mathbb{R}^3$ are the atomic number and 3D position respectively of atom i in the molecule, alongside a label $y \in \mathbb{R}$. We assume that the model, which takes S as input, is any architecture consisting of a backbone, which first processes S to build a latent representation of it, followed by a vertex-level or graph-level "decoder", that returns per-vertex predictions or a single prediction for the input respectively.

3.3.2 Pre-training via Denoising

Given a dataset of molecular structures, we pre-train the network by denoising the structures, which operates as follows. Let $\mathcal{D}_{\text{structures}} = \{S_1, \ldots, S_n\}$ denote the upstream dataset of equilibrium structures, and let GNN_θ denote a graph neural network with parameters θ which takes $S \in \mathcal{D}_{\text{structures}}$ as input and returns per-vertex predictions $\text{GNN}_\theta(S) = (\hat{\boldsymbol{\epsilon}}_1, \ldots, \hat{\boldsymbol{\epsilon}}_{|S|})$. The precise parameterization of the models we consider in this work is described in Sections 3.3.3 and 3.8.1.

Starting with an input molecule $S \in \mathcal{D}_{\text{structures}}$, we perturb it by adding i.i.d. Gaussian noise to its atomic positions $\mathbf{p}_i$. That is, we create a noisy version of the molecule:

$$\tilde{S} = \{(a_1, \tilde{\mathbf{p}}_1), \ldots, (a_{|S|}, \tilde{\mathbf{p}}_{|S|})\}, \text{ where } \tilde{\mathbf{p}}_i = \mathbf{p}_i + \sigma\boldsymbol{\epsilon}_i \text{ and } \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, I_3), \quad (3.1)$$

The noise scale σ is a tuneable hyperparameter (an interpretation of which is given in Section 3.3.2.1). We train the model as a denoising autoencoder by minimizing the following loss with respect to θ:

$$\mathbb{E}_{p(\tilde{S},S)}\left[\left\|\text{GNN}_\theta(\tilde{S}) - (\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_{|S|})\right\|^2\right]. \quad (3.2)$$

The distribution $p(\tilde{S}, S)$ corresponds to sampling a structure S from $\mathcal{D}_{\text{structures}}$ and adding noise to it according to Equation (3.1). Note that the model predicts the noise, not the original coordinates. Next, we motivate denoising as our pre-training objective for molecular modelling.

3.3.2.1 Denoising as Learning a Force Field

Datasets in quantum chemistry are typically generated by minimizing expensive-to-compute interatomic forces with methods such as density functional theory (DFT) [Parr and Weitao, 1994]. We speculate that learning this *force field* would give rise to useful representations for downstream tasks, since molecular properties vary with forces and energy. Therefore, a reasonable pre-training objective would be one that involves learning the force field. Unfortunately, this force field is either unknown or expensive to evaluate, and hence it cannot be used directly for pre-training. An alternative is to approximate the data-generating force field with one that can be cheaply evaluated and use it to learn good representations – an approach we outline in this section. Using the well-known link between denoising autoencoders and score-matching [Vincent, 2011, Song and Ermon, 2019, 2020], we can show that the denoising objective in Equation (3.2) is equivalent to learning a particular force field directly from equilibrium structures with some desirable properties. For clarity, in this subsection we condition on and suppress the atom types and molecule size in our notation, specifying a molecular structure by its coordinates $\mathbf{x} \in \mathbb{R}^{3N}$ (with N as the size of the molecule).

From the perspective of statistical physics, a structure $\mathbf{x}$ can be treated as a random quantity sampled from the Boltzmann distribution $p_{\text{physical}}(\mathbf{x}) \propto \exp(-E(\mathbf{x}))$, where $E(\mathbf{x})$ is the (potential) energy of $\mathbf{x}$. According to p_{physical}, low energy structures have a high probability of occurring. Moreover, the per-atom forces are given by $\nabla_{\mathbf{x}} \log p_{\text{physical}}(\mathbf{x}) = -\nabla_{\mathbf{x}} E(\mathbf{x})$, which is referred to as the force field. Our goal is to learn this force field. However both the energy function E and distribution p_{physical} are unknown, and we only have access to a set of equilibrium structures $\mathbf{x}_1, \ldots, \mathbf{x}_n$ that locally minimize the energy E. Since $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are then

local maxima of the distribution p_{physical}, our main approximation is to replace p_{physical} with a mixture of Gaussians centered at the data:

$$p_{\text{physical}}(\tilde{\mathbf{x}}) \approx q_\sigma(\tilde{\mathbf{x}}) := \frac{1}{n}\sum_{i=1}^{n} q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_i),$$

where we define $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_i) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}_i, \sigma^2 I_{3N})$. This approximation captures the fact that p_{physical} will have local maxima at the equilibrium structures, vary smoothly with $\mathbf{x}$ and is computationally convenient. Learning the force field corresponding to $q_\sigma(\tilde{\mathbf{x}})$ now yields a score-matching objective:

$$\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\left\|\text{GNN}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})\right\|^2\right]. \tag{3.3}$$

As shown by Vincent [2011], and recently applied to generative modelling [Song and Ermon, 2019, 2020, Ho et al., 2020, Shi et al., 2021, Xu et al., 2022], this objective is equivalent to the denoising objective. Specifically, defining $q_0(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta(\mathbf{x} = \mathbf{x}_i)$ to be the empirical distribution and $q_\sigma(\tilde{\mathbf{x}}, \mathbf{x}) = q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) q_0(\mathbf{x})$, the objective in Equation (3.3) is equivalent to:

$$\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}},\mathbf{x})}\left[\left\|\text{GNN}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\right\|^2\right] = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}},\mathbf{x})}\left[\left\|\text{GNN}_\theta(\tilde{\mathbf{x}}) - \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2}\right\|^2\right]. \tag{3.4}$$

We notice that the RHS corresponds to the earlier denoising loss in Equation (3.2) (up to a constant factor of $1/\sigma$ applied to GNN_θ that can be absorbed into the network). To summarize, denoising equilibrium structures corresponds to learning the force field that arises from approximating the distribution p_{physical} with a mixture of Gaussians. Note that we can interpret the noise scale σ as being related to the sharpness of p_{physical} or E around the local maxima $\mathbf{x}_i$. We also remark that the equivalence between Equation (3.3) and the LHS of Equation (3.4) does not require $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_i)$ to be a Gaussian distribution [Vincent, 2011], and other choices will lead to different denoising objectives, which we leave as future work. See Section 3.8.2 for technical caveats and details.

3.3.2.2 Noisy Nodes: Denoising as an Auxiliary Loss

Recently, Godwin et al. [2022] also applied denoising as an *auxiliary* loss to molecular property prediction, achieving significant improvements on a variety of molecular datasets. In particular, their approach, called Noisy Nodes, consisted of augmenting the usual optimization objective for predicting y with an auxiliary denoising loss. They suggested two explanations for why Noisy Nodes improves performance. First, the presence of a vertex-level loss discourages *oversmoothing* [Chen et al., 2019, Cai and Wang, 2020] of vertex/edge features after multiple message-passing layers – a common problem plaguing GNNs – because successful denoising requires diversity amongst vertex features in order to match the diversity in the noise targets $\boldsymbol{\epsilon}_i$. Second, they argued that denoising can aid representation learning by encouraging the network to learn aspects of the input distribution.

The empirical success of Noisy Nodes indicates that denoising can indeed result in meaningful representations. Since Noisy Nodes incorporates denoising only as an auxiliary task, the representation learning benefits of denoising are limited to the downstream dataset on which it is used as an auxiliary task. Our approach is to apply denoising as a pre-training objective on another large (unlabelled) dataset of structures to learn higher-quality representations, which results in better performance.

3.3.3 GNS and GNS-TAT

The main two models we consider in this work are Graph Net Simulator (GNS) [Sanchez-Gonzalez et al., 2020], which is a type of GNN, and a better-performing variant we contribute called GNS-TAT. GNS-TAT makes use of a recently published network transformation method called Tailored Activation Transforms (TAT) [Zhang et al., 2022], which has been shown to prevent certain degenerate behaviors at initialization in deep MLPs/convnets that are reminiscent of oversmoothing in GNNs (and are also associated with training difficulties). While GNS is not by default compatible with the assumptions of TAT, we propose a novel GNN initialization scheme called "Edge-Delta" that makes it compatible by initializing to zero the

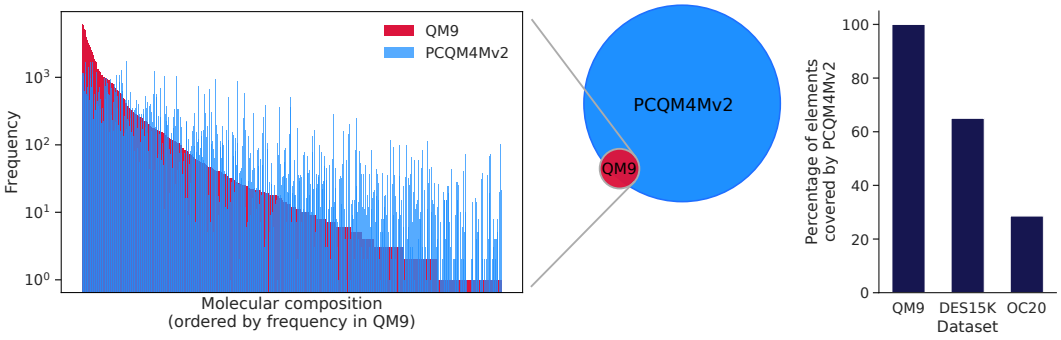


Figure 3.2: Left: Frequency of compositions of molecules appearing in QM9 overlayed with the corresponding frequency in PCQM4Mv2. Each bar represents one molecular composition (*e.g.* one carbon atom, two oxygen atoms). **Right:** Percentage of elements appearing in QM9, DES15K, OC20 that also appear in PCQM4Mv2.

weights that carry "messages" from vertices to edges. This marks the first application of TAT to any applied problem in the literature. See Section 3.8.1 for details.

3.4 Experiments

The goal of our experimental evaluation in this section is to answer the following questions. First, does pre-training a neural network via denoising improve performance on the downstream task compared to training from a random initialization? Second, how does the benefit of pre-training depend on the relationship between the upstream and downstream datasets? Our evaluation involves four realistic and challenging molecular datasets, which vary in size, compound compositions (organic or inorganic) and labelling methodology (DFT- or CCSD(T)-generated), as described below.

3.4.1 Datasets and Training Setup

Datasets. First, the main dataset we use for pre-training is **PCQM4Mv2** [Nakata and Shimazaki, 2017], which contains 3.4 million organic molecules, specified by their 3D structures at equilibrium calculated using DFT.[2] The molecules in PCQM4Mv2 only contain one label, however the labels are not used as denoising only requires the structures. The large scale and diversity of PCQM4Mv2 makes it well-suited

[2]An earlier version of this dataset without any 3D structures, called PCQM4M, was used for supervised pre-training [Ying et al., 2021], but to our knowledge, this is the first time the 3D structures from v2 have been used and in a self-supervised manner.

for pre-training via denoising. Second, as a dataset for fine-tuning, we use **QM9** [Ramakrishnan et al., 2014a], which contains around 130,000 small organic molecules and is widely used as a molecular property prediction benchmark [Klicpera et al., 2020a, Fuchs et al., 2020a, Satorras et al., 2021, Finzi et al., 2020, Hutchinson et al., 2021, Schütt et al., 2021, Thölke and De Fabritiis, 2022, Godwin et al., 2022]. Each molecule is specified by its structure alongside 12 associated molecular property labels. Third, **Open Catalyst 2020 (OC20)** [Chanussot* et al., 2021] is a recent large benchmark of interacting surfaces and adsorbates relevant to catalyst discovery. OC20 contains various tasks, such as predicting the relaxed state energy from an initial high-energy structure (IS2RE). We explore different combinations of upstream and downstream tasks as described in Section 3.4.3. Lastly, **DES15K** [Donchev et al., 2021] is a small dataset we use for fine-tuning, which contains around 15,000 dimer geometries (*i.e.* molecule pairs) with non-covalent molecular interactions. Each pair is labelled with its interaction energy computed using the gold-standard CCSD(T) method [Bartlett and Musiał, 2007]. CCSD(T) is usually both more expensive and accurate than DFT, which is used for all aforementioned datasets. See Section 3.8.4 for further details and a discussion about the choice of using DFT-generated structures for pre-training.

Figure 3.2 (right) shows what percentage of elements appearing in each of QM9, OC20 and DES15K also appear in PCQM4Mv2. Whereas QM9 is fully covered by PCQM4Mv2, we observe that DES15K has less element overlap with PCQM4Mv2 and less than $< 30\%$ of elements in OC20 are contained in PCQM4Mv2. This is owing to the fact that surface molecules in OC20 are inorganic lattices, none of which appear in PCQM4Mv2. This suggests that we can expect least transfer from PCQM4Mv2 to OC20. We also compare PCQM4Mv2 and QM9 in terms of the molecular compositions, *i.e.* the number of atoms of each element, that appear in each. Due to presence of isomers, both datasets contain multiple molecules with the same composition. For each molecular composition in QM9, Figure 3.2 (left) shows its frequency in both QM9 and PCQM4Mv2. We observe that most molecular

Table 3.1: Results on QM9 comparing the performance of GNS-TAT + Noisy Nodes (NN) with and without pre-training on PCQM4Mv2 (averaged over three seeds) with other baselines.

Target	Unit	SchNet	E(n)-GNN	DimeNet++	SphereNet	PaiNN	TorchMD-NET	GNS + NN	GNS-TAT + NN	Pre-trained GNS-TAT + NN
μ	D	0.033	0.029	0.030	0.027	0.012	**0.011**	0.025	0.021	0.016
α	a_0^3	0.235	0.071	0.043	0.047	0.045	0.059	0.052	0.047	**0.040**
ϵ_{HOMO}	meV	41.0	29.0	24.6	23.6	27.6	20.3	20.4	17.3	**14.9**
ϵ_{LUMO}	meV	34.0	25.0	19.5	18.9	20.4	18.6	17.5	17.1	**14.7**
$\Delta\epsilon$	meV	63.0	48.0	32.6	32.3	45.7	36.1	28.6	25.7	**22.0**
$\langle R^2 \rangle$	a_0^2	0.07	0.11	0.33	0.29	0.07	**0.033**	0.70	0.65	0.44
ZPVE	meV	1.700	1.550	1.210	1.120	1.280	1.840	1.160	1.080	**1.018**
U_0	meV	14.00	11.00	6.32	6.26	5.85	6.15	7.30	6.39	**5.76**
U	meV	19.00	12.00	6.28	7.33	5.83	6.38	7.57	6.39	**5.76**
H	meV	14.00	12.00	6.53	6.40	5.98	6.16	7.43	6.42	**5.79**
G	meV	14.00	12.00	7.56	8.00	7.35	7.62	8.30	7.41	**6.90**
c_{v}	$\frac{\text{cal}}{\text{mol K}}$	0.033	0.031	0.023	0.022	0.024	0.026	0.025	0.022	**0.020**

compositions in QM9 also appear in PCQM4Mv2. We also remark that since pre-training is self-supervised using only unlabelled structures, test set contamination is not possible – in fact, PCQM4Mv2 does not have most of the labels in QM9.

Training setup. GNS/GNS-TAT were implemented in JAX [Bradbury et al., 2018] using Haiku and Jraph [Hennigan et al., 2020, Godwin* et al., 2020]. All experiments were averaged over 3 seeds. Detailed hyperparameter and hardware settings can be found in Sections 3.8.5 and 3.8.6.

3.4.2 Results on QM9

We evaluate two variants of our model on QM9 in Table 3.1, GNS-TAT with Noisy Nodes trained from a random initialization versus pre-trained parameters. Pre-training is done on PCQM4Mv2 via denoising. For best performance on QM9, we found that using atom type masking and prediction during pre-training additionally helped [Hu et al., 2020a]. We fine-tune a separate model for each of the 12 targets, as usually done on QM9, using a single pre-trained model. This is repeated for three seeds (including pre-training). Following customary practice, hyperparameters, including the noise scale for denoising during pre-training and fine-tuning, are tuned on the HOMO target and then kept fixed for all other targets. We first observe that GNS-TAT with Noisy Nodes performs competitively with other models and

significantly improves upon GNS with Noisy Nodes, revealing the benefit of the TAT modifications. Utilizing pre-training then further improves performance across all targets, achieving a new state-of-the-art compared to prior work for 10 out of 12 targets. Interestingly, for the electronic spatial extent target $\langle R^2 \rangle$, we found GNS-TAT to perform worse than other models, which may be due to the optimal noise scale being different from that of other targets.

3.4.3 Results on OC20

Next, we consider the Open Catalyst 2020 benchmark focusing on the downstream task of predicting the relaxed energy from the initial structure (IS2RE). We compared GNS with Noisy Nodes trained from scratch versus using pre-trained parameters. We experimented with two options for pre-training: (1) pre-training via denoising on PCQM4Mv2, and (2) pre-training via denoising on OC20 itself. For the latter, we follow Godwin et al.'s [2022] approach of letting the denoising target be the relaxed structure, while the perturbed input is a random interpolation between the initial and relaxed structures with added Gaussian noise – this corresponds to the IS2RS task with additional noise. As shown in Figure 3.3 (left), pre-training on PCQM4Mv2 offers no benefit for validation performance on IS2RE, however pre-training on OC20 leads to considerably faster convergence but the same final performance. The lack of transfer from PCQM4Mv2 to OC20 is likely due to the difference in nature of the two datasets and the small element overlap as discussed in Section 3.4.1 and Figure 3.2 (right). On the other hand, faster convergence from using parameters pre-trained on OC20 suggests that denoising learned meaningful features. Unsurprisingly, the final performance is unchanged since the upstream and downstream datasets are the same in this case, so pre-training with denoising is identical to the auxiliary task of applying Noisy Nodes. The performance achieved is also competitive with other models in the literature as shown in Table 3.9.

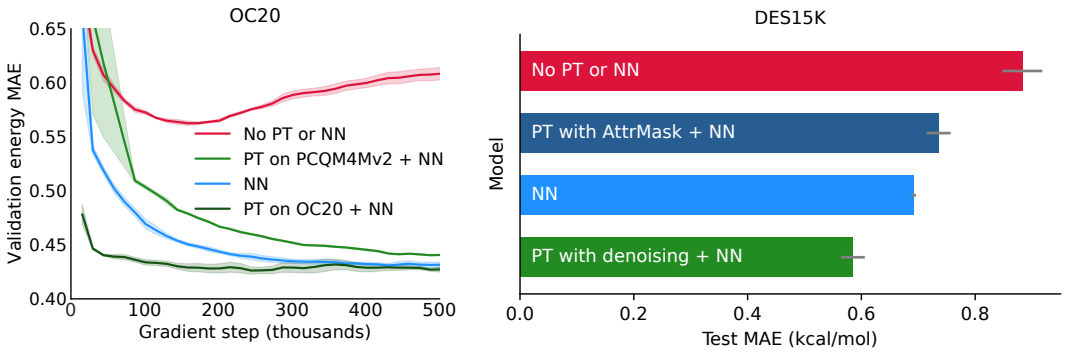


Figure 3.3: Left: Validation performance curves on the OC20 IS2RE task (`ood_both` split) with GNS. See Table 3.9 for a comparison to other models in the literature. **Right:** Test performance curves for predicting interaction energies of dimer geometries in the DES15K dataset with GNS-TAT. "PT" and "NN" stand for pre-training and Noisy Nodes respectively.

3.4.4 Results on DES15K

In our experiments so far, all downstream tasks were based on DFT-generated datasets. While DFT calculations are more expensive than using neural networks, they are relatively cheap compared to even higher quality methods such as CCSD(T) [Bartlett and Musiał, 2007]. In this section, we evaluate how useful pre-training on DFT-generated structures from PCQM4Mv2 is when fine-tuning on the recent DES15K dataset which contains higher quality CCSD(T)-generated interaction energies. Moreover, unlike QM9, inputs from DES15K are systems of two interacting molecules and the dataset contains only around 15,000 examples, rendering it more challenging. We compare the test performance on DES15K achieved by GNS-TAT with Noisy Nodes when trained from scratch versus using pre-trained parameters from PCQM4Mv2. As a baseline, we also include pre-training on PCQM4Mv2 using 2D-based AttrMask [Hu et al., 2020a] by masking and predicting atomic numbers. Figure 3.3 (right) shows that using Noisy Nodes significantly improves performance compared to training from scratch, with a further improvement resulting from using pre-training via denoising. AttrMask underperforms denoising since it likely does not fully exploit the 3D structural information. Importantly, this shows that pre-training by denoising structures obtained through relatively cheap methods

such as DFT can even be beneficial when fine-tuning on more expensive and smaller downstream datasets. See Section 3.8.7.1 for similar results on another architecture.

3.5 Analysis

3.5.1 Pre-training a Different Architecture

To explore whether pre-training is beneficial beyond GNS/GNS-TAT, we applied pre-training via denoising to the TorchMD-NET architecture [Thölke and De Fabritiis, 2022]. TorchMD-NET is a transformer-based architecture whose layers maintain per-atom scalar features $x_i \in \mathbb{R}^F$ and vector features $v_i \in \mathbb{R}^{3\times F}$, where F is the feature dimension, that are updated in each layer using a self-attention mechanism. We implemented denoising by using gated equivariant blocks [Weiler et al., 2018a, Schütt et al., 2021] applied to the processed scalar and vector features. The resulting vector features are then used as the noise prediction.

In Table 3.2, we evaluate the effect of adding Noisy Nodes and pre-training to the architecture on the HOMO and LUMO targets in QM9. Pre-training yields a boost in performance, allowing the model to achieve SOTA results. Note that the results shown for TorchMD-NET are from our runs using Thölke and De Fabritiis's [2022] open-source code, which led to slightly worse results than their published ones (our pre-training results still outperform their published results). Our code for experiments on TorchMD-NET is open source[3].

3.5.2 Varying Dataset Sizes

We also investigate how downstream test performance on the HOMO target in QM9 varies as a function of the number of upstream and downstream training examples. First, we compare the performance of GNS-TAT with Noisy Nodes either trained from scratch or using pre-trained parameters for different numbers of training examples from QM9; we also include the performance of just GNS-TAT. As shown in Figure 3.4 (left), pre-training improves the downstream performance for all dataset sizes. The difference in test MAE also grows as the downstream training

[3] `https://github.com/shehzaidi/pre-training-via-denoising`

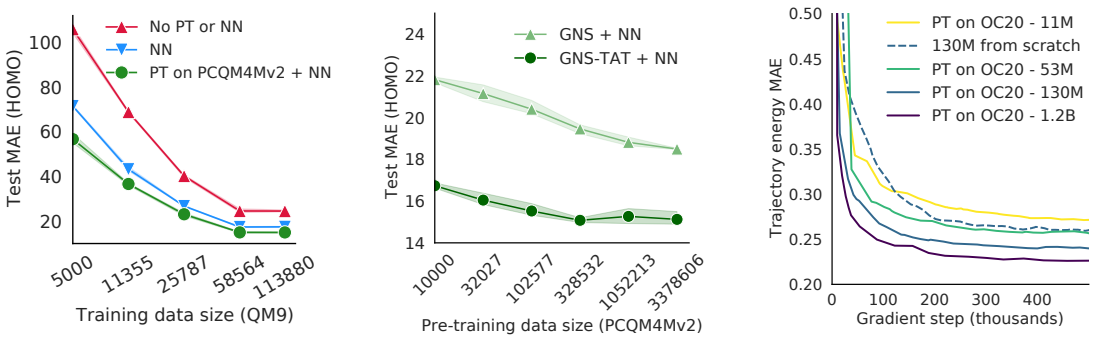


Figure 3.4: Left: Impact of varying the downstream dataset size for the HOMO target in QM9 with GNS-TAT. **Middle:** Impact of varying the upstream dataset size for the HOMO target in QM9. **Right:** Validation performance curves on the OC20 S2EF task (`ood_both` split) for different GNS model sizes. "PT" and "NN" stand for pre-training and Noisy Nodes respectively.

Table 3.2: Performance of TorchMD-NET with Noisy Nodes and pre-training on PCQM4Mv2.

Method	ϵ_{HOMO}	ϵ_{LUMO}
TorchMD-NET	22.0 ± 0.6	18.7 ± 0.4
+ Noisy Nodes	18.1 ± 0.1	15.6 ± 0.1
+ Pre-training	$\mathbf{15.6} \pm 0.1$	$\mathbf{13.2} \pm 0.2$

data reduces. Second, we assess the effect of varying the amount of pre-training data while fixing the downstream dataset size for both GNS and GNS-TAT as shown in Figure 3.4 (middle). For both models, we find that downstream performance generally improves as upstream data increases, with saturating performance for GNS-TAT. More upstream data can yield better-quality representations.

3.5.3 Varying Model Size

We study the benefit of pre-training as models are scaled up on large downstream datasets. Recall that the S2EF dataset in OC20 contains around 130 million DFT evaluations for catalytic systems, providing three orders of magnitude more training data than QM9. We compare the performance of four GNS models with sizes ranging from 10 million to 1.2 billion parameters scaled up by increasing the hidden layer sizes in the MLPs. Each is pre-trained via denoising using the trajectories provided for the IS2RE/IS2RS tasks as described in Section 3.4.3. We also compare

Table 3.3: Performance of TorchMD-NET for force prediction on MD17 (aspirin).

Method	Test MAE
TorchMD-NET	0.268 ± 0.003
+ Pre-training	$\mathbf{0.222} \pm 0.003$

this to a 130 million parameter variant of GNS trained from scratch. As shown in Figure 3.4 (right), the pre-trained models continue to benefit from larger model sizes.We also observe that pre-training is beneficial, as the model trained from scratch underperforms in comparison: the 130 million parameters model trained from scratch is outperformed by a pre-trained model of less than half the size.

3.5.4 Pre-training Improves Force Prediction

As shown in Section 3.3.2.1, denoising structures corresponds to learning an approximate force field directly from equilibrium structures. We explore whether pre-training via denoising would therefore also improve models trained to predict atomic forces. We compare the performance of TorchMD-NET for force prediction on the MD17 (aspirin) dataset with and without pre-training on PCQM4Mv2. Table 3.3 shows that pre-training improves force prediction. We also assess the effect of pre-training on the OC20 dataset for force prediction in Section 3.8.7.3, similarly finding an improvement due to pre-training.

3.5.5 Freezing Pre-trained Parameters

Finally, we perform an experiment to assess how useful the features learned by pre-training are if they are not fine-tuned for the downstream task but kept fixed instead. Specifically, on the HOMO target in QM9, we freeze the backbone of the model and fine-tune only the decoder (*cf.* Section 3.8.1). To evaluate this, we compare it to using random parameters from initialization for the model's backbone, which allows us to isolate how useful the pre-trained features are. As described in Section 3.8.1, the decoder is a simple module involving no message-passing. Figure 3.5 shows that only training the decoder while keeping the pre-trained parameters fixed results in test MAE of 40 meV, which is worse than fine-tuning

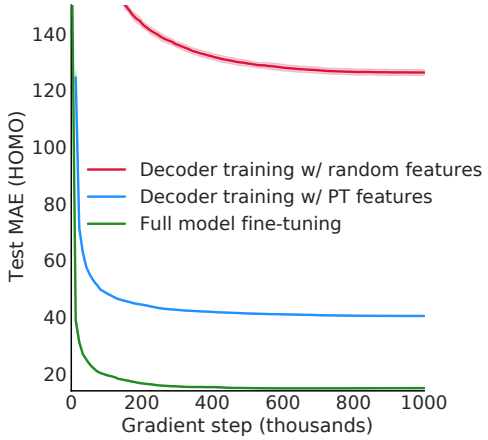


Figure 3.5: Training only the decoder results in significantly better performance with pre-trained instead of random features with GNS-TAT.

the entire model but substantially better than the performance of >100 meV in test MAE resulting from training the decoder when the remaining parameters are randomly initialized. This suggests that the features learned by denoising are more discriminative for downstream prediction than random features. We note that training only the decoder is also substantially faster than training the entire network – one batch on a single V100 GPU takes 15ms, which is 50× faster than one batch using 16 TPUs for the full network.

3.6 Limitations & Future Work

We have shown that pre-training can significantly improve performance for various tasks. One additional advantage of pre-trained models is that they can be shared in the community, allowing practitioners to fine-tune models on their datasets. However, unlike vision and NLP, molecular networks vary widely and the community has not yet settled on a "standard" architecture, making pre-trained weights less reusable. Moreover, the success of pre-training inevitably depends on the relationship between the upstream and downstream datasets. In the context of molecular property prediction, understanding what aspects of the upstream data distribution must match the downstream data distribution for transfer is an important direction for future work.

More generally, pre-training models on large datasets incurs a computational cost. However, our results show that pre-training for 3D molecular prediction does not require the same scale as large NLP and vision models. We discuss considerations on the use of compute and broader impact in Section 3.8.3.

3.7 Conclusion

We investigated pre-training neural networks by denoising in the space of 3D molecular structures. We showed that denoising in this context is equivalent to learning a force field, motivating its ability to learn useful representations and shedding light on successful applications of denoising in other works [Godwin et al., 2022]. This technique enabled us to utilize existing large datasets of 3D structures for improving performance on various downstream molecular property prediction tasks, setting a new SOTA in some cases such as QM9. More broadly, this bridges the gap between the utility of pre-training in vision/NLP and molecular property prediction from structures. We hope that this approach will be particularly impactful for applications of deep learning to scientific problems.

3.8 Supplementary Material

3.8.1 Architectural Details

3.8.1.1 Standard GNS

As our base model architecture we chose a Graph Net Simulator (GNS) [Sanchez-Gonzalez et al., 2020], which consists of an ENCODER which constructs a graph representation from the input S, a PROCESSOR of repeated message passing blocks that update the latent graph representation, and a DECODER which produces predictions. Our implementation follows Godwin et al.'s [2022] modifications to enable molecular and graph-level property predictions which has been shown to achieve strong results across different molecular prediction tasks without relying on problem-specific features.

In the ENCODER, we represent the set of atoms $S = \{(a_1, \mathbf{p}_1), (a_2, \mathbf{p}_2), \ldots, (a_{|S|}, \mathbf{p}_{|S|})\}$ as a directed graph $G = (V, E)$ where $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{|S|}\}$ and $E = \{\mathbf{e}_{i,j}\}_{i,j}$ are the sets of "featurized" vertices and edges, respectively. Edges $\mathbf{e}_{i,j} \in E$ are constructed whenever the distance between the i-th and j-th atoms is less than the connectivity radius R_{cut} (given in Section 3.8.6), in which case we connect $\mathbf{v}_i$ and $\mathbf{v}_j$ with a *directed* edge $\mathbf{e}_{i,j}$ from i to j that is a featurization of the displacement vector $\mathbf{p}_j - \mathbf{p}_i$. Meanwhile, for the i-th atom, $\mathbf{v}_i$ is given by a learnable vector embedding of the atomic number a_i.

The PROCESSOR consists of L message-passing steps that produce intermediate graphs $G_1, \ldots, G_L$ (with the same connectivity structure as the initial one). Each of these steps computes the sum of a shortcut connection from the previous graph, and the application of an Interaction Network [Battaglia et al., 2016]. Interaction Networks first update each edge feature by applying an "edge update function" to a combination of the existing feature and the features of the two connected vertices. They then update each vertex feature by applying a "vertex update function" to a combination of the existing feature and the (new) edge features of incoming edges. In GNS, edge update functions are 3 hidden layer fully-connected MLPs, using a "shifted softplus" ($\text{ssp}(x) = \log(0.5e^x + 0.5)$) activation function, applied

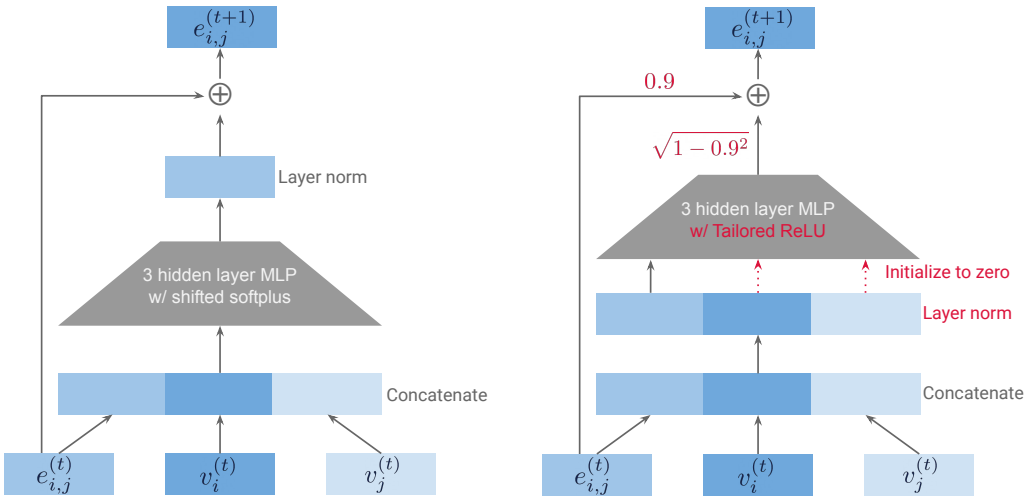


Figure 3.6: Diagram showing the edge update for a single step t of the PROCESSOR. **Left:** Edge update for GNS. **Right:** Edge update for GNS-TAT (with modifications shown in red).

to the concatenation of the relevant edge and vertex features, followed by a layer normalization layer. Vertex update functions are similar, but are applied to the concatenation of the relevant vertex feature and *sum* over relevant edge features.

In our implementation of GNS we applied the same PROCESSOR in sequence three times (with shared parameters), with the output of each being decoded to produce a prediction and corresponding loss value. The loss for the whole model is then given by the average of these. (Test-time predictions are meanwhile computed using only the output of the final PROCESSOR.)

The DECODER is responsible for computing graph-level and vertex-level predictions from the output of each PROCESSOR. Vertex-level predictions, such as noise as described in Section 3.3.2, are decoded using an MLP applied to each vertex feature. Graph-level predictions (*e.g.* energies) are produced by applying an MLP to each vertex feature, aggregating the result over vertices (via a sum), and then applying another MLP to the result.

3.8.1.2 GNS with Tailored Activation Transformation (GNS-TAT)

Tailored Activation Transformation (TAT) [Zhang et al., 2022] is a method for initializing and transforming neural networks to make them easier to train, and

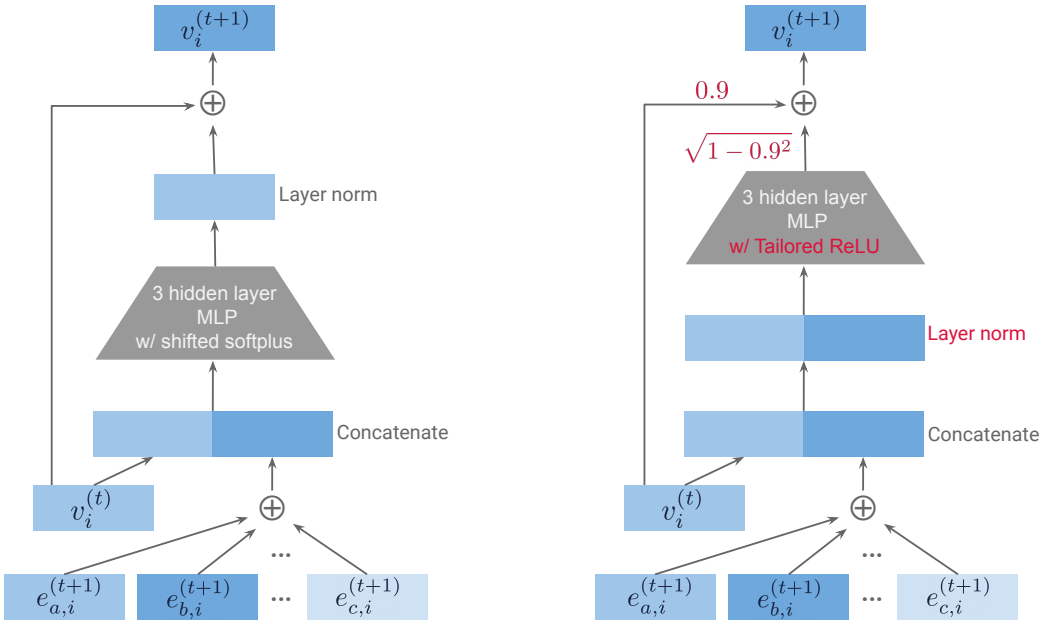


Figure 3.7: Diagram showing the vertex update for a single step t of the PROCESSOR. **Left:** Vertex update for GNS. **Right:** Vertex update for GNS-TAT (with modifications shown in red).

is based on a similar method called Deep Kernel Shaping (DKS) [Martens et al., 2021]. TAT controls the propagation of "q values", which are initialization-time approximations to dimension-normalized squared norms of the network's layer-wise activation vectors, and "c values", which are cosine similarities between such vectors (for different inputs). In other words, q values approximate $\|z(x)\|^2/\dim(z(x))$, where $z(x)$ denotes a layer's output as a function of the network's input x, and c values approximate $z(x)^\top z(x')/(\|z(x)\|\|z(x')\|)$, where x' is another possible network input. In standard deep networks, c values will converge to a constant value $c_\infty \in [0, 1]$, so that "geometric information" is lost, which leads to training difficulties [Martens et al., 2021]. DKS/TAT prevents this convergence through a combination of careful weight initialization, and transformations to the network's activation functions and sum/average layers.

Oversmoothing [Chen et al., 2019, Cai and Wang, 2020, Rong et al., 2019, Zhou et al., 2020, Yang et al., 2020b, Zhao and Akoglu, 2020, Do et al., 2021] is a phenomenon observed in GNN architectures where vertex/edge features all

converge to approximately the same value with depth, and is associated with training difficulties. It is reminiscent of how, when $c_\infty = 1$, feature vectors will converge with depth to a constant input-independent vector in standard deep networks. It therefore seems plausible that applying TAT to GNNs may help with the oversmoothing problem and thus improve training performance.

Unfortunately, the GNS architecture violates two key assumptions of TAT. Firstly, the sums over edge features (performed in the vertex update functions) violate the assumption that all sum operations must be between the outputs of linear layers with *independently sampled* initial weights. Secondly, GNS networks have multiple inputs for which information needs to be independently preserved and propagated to the output, while DKS/TAT assumes a single input (or multiple inputs whose representations evolve independently in the network).

To address these issues we introduce a new initialization scheme called "Edge-Delta", which initializes to zero the weights that multiply incoming vertex features in the edge update functions (and treats these weights as absent for the purpose of computing the initial weight variance). This approach is inspired by the use of the "Delta initialization" [Balduzzi et al., 2017, Xiao et al., 2018] for convolutional networks in DKS/TAT, which initializes filter weights of the non-central locations to zero, thus allowing geometric information, in the form of c values, to propagate independently for each location in the feature map. When using the Edge-Delta initialization, edge features propagate independently of each other (and of vertex features), through what is essentially a standard deep residual network (with edge update functions acting as the residual branches), which we will refer to as the "edge network".

Given the use of Edge-Delta we can then apply TAT to GNS as follows[4]. First, we replace GNS's activation functions with TAT's transformed Leaky-ReLU activation functions (or "Tailored ReLUs"), which we compute with TAT's η parameter set

[4]Note that Edge-Delta initialization is compatible with TAT, since for the purposes of q/c value propagation, zero-initialized connections in the network can be treated as absent.

to 0.8, and its "subnetwork maximizing function" defined on the edge network[5]. We also replace each sum involving shortcut connections with weighted sums, whose weights are 0.9 and $\sqrt{1-0.9^2}$ for the shortcut and non-shortcut branches respectively. We retain the use of layer normalization layers in the edge/vertex update functions, but move them to before the first fully-connected layer, as this seems to give the best performance. As required by TAT, we use a standard Gaussian fan-in initialization for the weights, and a zero initialization for the biases, with Edge-Delta used only for the first linear layer of the edge update functions. Finally, we replace the sum used to aggregate vertex features in the DECODER with an average. See Figures 3.6 and 3.7 for an illustration of these changes.

We experimented with an analogous "Vertex-Delta" initialization, which initializes to zero weights in the vertex update functions that multiply summed edge features, but found that Edge-Delta gave the best results. This might be because the edge features, which encode distances between vertices (and are best preserved with the Edge-Delta approach), are generally much more informative than the vertex features in molecular property prediction tasks. We also ran informal ablation studies, and found that each of our changes to the original GNS model contributed to improved results, with the use of Edge-Delta and weighted shortcut sums being especially important.

GNS vs. GNS-TAT on OC20. Our preliminary experiments used to determine appropriate choices for the use of Edge-Delta/Vertex-Delta initialization, weighting for shortcut sums and TAT hyperparameters such as η were conducted on QM9. This led to a parameterization of GNS-TAT which gave improved performance on QM9/DES15K as shown earlier in Sections 3.4.2 and 3.4.4. However, we kept the original parameterization for GNS from Godwin et al. [2022] on OC20, as the model size was different from QM9/DES15K (see Section 3.8.6) and re-tuning

[5]For the purposes of computing the subnetwork maximizing function we ignore the rest of the network and just consider the edge network. While the layer normalization layer (which we move before the MLP) technically depends on the vertex features, this dependency can be ignored as long as the q values of these features is 1 (which will be true given the complete set of changes we make to the GNS architecture).

hyperparameters on OC20 would be computationally expensive. Using the same TAT settings as QM9 on OC20 results in approximately the same performance as the original GNS model as shown in Table 3.4.

Table 3.4: GNS vs. GNS-TAT energy prediction MAE on OC20. TAT hyperparameters were tuned with experiments on QM9 and then kept fixed for OC20.

Validation Dataset	Model		
	GNS	GNS + NN	GNS-TAT + NN
ID	0.5233	0.4196	0.4199
OOD Adsorbate	0.6295	0.4900	0.5013
OOD Catalyst	0.5202	0.4316	0.4319
OOD Both	0.5617	0.4282	0.4373

3.8.2 Denoising as Learning a Force Field

We specify a molecular structure as $\mathbf{x} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}) \in \mathbb{R}^{3N}$, where $\mathbf{x}^{(i)} \in \mathbb{R}^3$ is the coordinate of atom i. Let $E(\mathbf{x})$ denote the total (potential) energy of $\mathbf{x}$, such that $-\nabla_{\mathbf{x}} E(\mathbf{x})$ are the forces on the atoms. As discussed in Section 3.3.2.1, learning the force field, *i.e.* the mapping $\mathbf{x} \mapsto -\nabla_{\mathbf{x}} E(\mathbf{x})$, is a reasonable pre-training objective. Furthermore, learning the force field can be viewed as score-matching if we define the distribution $p_{\text{physical}}(\mathbf{x}) \propto \exp(-E(\mathbf{x}))$ and observe that the *score* of p_{physical} is the force field: $\nabla_{\mathbf{x}} \log p_{\text{physical}}(\mathbf{x}) = -\nabla_{\mathbf{x}} E(\mathbf{x})$.

However, a technical caveat is that p_{physical} is an *improper* probability density, because it cannot be normalized due to the translation invariance of E. Writing the translation of a structure as $\mathbf{x} + \mathbf{t} := (\mathbf{x}^{(1)} + \mathbf{t}, \ldots, \mathbf{x}^{(N)} + \mathbf{t})$ where $\mathbf{t} \in \mathbb{R}^3$ is a constant vector, we have $E(\mathbf{x} + \mathbf{t}) = E(\mathbf{x})$. This implies that the normalizing constant $\int_{\mathbb{R}^{3N}} p_{\text{physical}}(\mathbf{x})\, \mathrm{d}\mathbf{x}$ diverges to infinity. To remedy this, we can restrict ourselves to the $(3N-3)$-dimensional subspace $V := \{\mathbf{x} \in \mathbb{R}^{3N} \mid \sum_i \mathbf{x}^{(i)} = 0\} \subseteq \mathbb{R}^{3N}$ consisting of the mean-centered structures, over which p_{physical} can be defined as a normalizable distribution.

Proceeding similarly as Section 3.3.2.1, let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in V$ be a set of *mean-centered* equilibrium structures. For any $\tilde{\mathbf{x}} \in V$, we now approximate

$$p_{\text{physical}}(\tilde{\mathbf{x}}) \approx q_\sigma(\tilde{\mathbf{x}}) := \frac{1}{n} \sum_{i=1}^{n} q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_i),$$

where the Gaussian distributions $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_i)$ are defined on V as:

$$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_i) = \frac{1}{(2\pi\sigma)^{(3N-3)/2}} \exp\left(-\frac{1}{2\sigma^2} \left\|\tilde{\mathbf{x}} - \mathbf{x}_i\right\|^2\right).$$

For convenience, we have expressed structures as vectors in the ambient space $\mathbb{R}^{3N}$, however they are restricted to lie in the smaller space V. Note that the normalizing constant accounts for the fact that V is $(3N-3)$-dimensional. As before, we define $q_0(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta(\mathbf{x} = \mathbf{x}_i)$ to be the empirical distribution and $q_\sigma(\tilde{\mathbf{x}}, \mathbf{x}) = q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) q_0(\mathbf{x})$. The score-matching objective is given by:

$$J_1(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\left\|\text{GNN}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})\right\|^2\right], \tag{3.5}$$

where the expectation is now over V. As shown by Vincent [2011], minimizing the objective above is equivalent to the minimizing the following objective:

$$J_2(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}, \mathbf{x})}\left[\left\|\text{GNN}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\right\|^2\right]. \tag{3.6}$$

This is recognized as a denoising objective, because $\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = (\mathbf{x} - \tilde{\mathbf{x}})/\sigma^2$. A practical implication of this analysis is that the noise $(\mathbf{x} - \tilde{\mathbf{x}})/\sigma^2 \in V$ should be mean-centered, which is intuitive since it is impossible to predict a translational component in the noise.

We include a proof of the equivalence between Equations (3.5) and (3.6) for completeness:

Proposition 1 (Vincent [2011])**.** *The minimization objectives $J_1(\theta)$ and $J_2(\theta)$ are equivalent.*

Proof. We first observe:

$$J_1(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\left\|\text{GNN}_\theta(\tilde{\mathbf{x}})\right\|^2\right] - 2\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\langle \text{GNN}_\theta(\tilde{\mathbf{x}}), \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})\rangle\right] + C_1$$
$$J_2(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\left\|\text{GNN}_\theta(\tilde{\mathbf{x}})\right\|^2\right] - 2\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}, \mathbf{x})}\left[\langle \text{GNN}_\theta(\tilde{\mathbf{x}}), \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\rangle\right] + C_2,$$

where C_1, C_2 are constants independent of θ. Therefore, it suffices to show that the middle terms on the RHS are equal. Since expectations over $q_\sigma(\tilde{\mathbf{x}})$ and $q_\sigma(\tilde{\mathbf{x}}, \mathbf{x})$ are restricted to $V \subseteq \mathbb{R}^{3N}$, we apply a change of basis to write them as integrals against the $(3N-3)$-dimensional Lebesgue measure. Pick an orthonormal basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{3N-3}\} \subseteq \mathbb{R}^{3N}$ for V and let $P_V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{3N-3}] \in \mathbb{R}^{3N \times 3(N-1)}$ be the projection matrix, so $\mathbf{z} = P_V^\intercal \mathbf{x}$ expresses a mean-centered structure $\mathbf{x}$ in terms of the coordinates of the chosen basis for V. Noting that P_V has orthonormal columns and that it yields a bijection between V and $\mathbb{R}^{3N-3}$, we calculate:

$$
\begin{aligned}
&\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\langle \mathrm{GNN}_\theta(\tilde{\mathbf{x}}), \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})\rangle\right] \\
&= \int_{\mathbb{R}^{3N-3}} q_\sigma(P_V\tilde{\mathbf{z}}) \langle \mathrm{GNN}_\theta(P_V\tilde{\mathbf{z}}), \nabla \log q_\sigma(P_V\tilde{\mathbf{z}})\rangle \,\mathrm{d}\tilde{\mathbf{z}} \\
&= \int_{\mathbb{R}^{3N-3}} q_\sigma(P_V\tilde{\mathbf{z}}) \left\langle \mathrm{GNN}_\theta(P_V\tilde{\mathbf{z}}), \frac{\nabla q_\sigma(P_V\tilde{\mathbf{z}})}{q_\sigma(P_V\tilde{\mathbf{z}})} \right\rangle \mathrm{d}\tilde{\mathbf{z}} \\
&= \int_{\mathbb{R}^{3N-3}} \langle \mathrm{GNN}_\theta(P_V\tilde{\mathbf{z}}), \nabla q_\sigma(P_V\tilde{\mathbf{z}})\rangle \,\mathrm{d}\tilde{\mathbf{z}} \\
&= \int_{\mathbb{R}^{3N-3}} \left\langle \mathrm{GNN}_\theta(P_V\tilde{\mathbf{z}}), \frac{1}{n}\sum_{i=1}^{n} \nabla q_\sigma(P_V\tilde{\mathbf{z}} \mid \mathbf{x}_i) \right\rangle \mathrm{d}\tilde{\mathbf{z}} \\
&= \int_{\mathbb{R}^{3N-3}} \left\langle \mathrm{GNN}_\theta(P_V\tilde{\mathbf{z}}), \frac{1}{n}\sum_{i=1}^{n} q_\sigma(P_V\tilde{\mathbf{z}} \mid \mathbf{x}_i) \nabla \log q_\sigma(P_V\tilde{\mathbf{z}} \mid \mathbf{x}_i) \right\rangle \mathrm{d}\tilde{\mathbf{z}} \\
&= \int_{\mathbb{R}^{3N-3}} \frac{1}{n}\sum_{i=1}^{n} q_\sigma(P_V\tilde{\mathbf{z}} \mid \mathbf{x}_i) \left\langle \mathrm{GNN}_\theta(P_V\tilde{\mathbf{z}}), \nabla \log q_\sigma(P_V\tilde{\mathbf{z}} \mid \mathbf{x}_i)\right\rangle \mathrm{d}\tilde{\mathbf{z}} \\
&= \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}, \mathbf{x})}\left[\langle \mathrm{GNN}_\theta(\tilde{\mathbf{x}}), \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\rangle\right]
\end{aligned}
$$

□

3.8.2.1 Differences Between Denoising for Generative Modelling vs. Learning Forces

Denoising has recently seen widespread use as a means for generative score-based modelling [*e.g.* Song and Ermon, 2019, 2020, Ho et al., 2020] due to the equivalence between denoising and score-matching [Vincent, 2011]. In this section, we elaborate on the differences between the equivalence of denoising and score-matching for generative modelling versus the equivalence of denoising and force learning. Both

of these equivalences rely on the result of Vincent [2011] although with different assumptions and different aims in practice.

As background, generative score-based modelling consists of modelling a target distribution $p(\mathbf{x})$ by learning an approximation of its score function $\nabla \log p(\mathbf{x})$ using a neural network given a training set of samples $\mathbf{x}_1, \dots \mathbf{x}_n \sim p(\mathbf{x})$. Using the learned approximation of the score function, new samples can be generated using score-based MCMC techniques, such as Langevin MCMC. Learning the score function corresponds to the optimization objective

$$\min_{\theta} \mathbb{E}_{p(\mathbf{x})} \left[\left\| \varphi_\theta(\mathbf{x}) - \nabla \log p(\mathbf{x}) \right\|^2 \right]. \tag{3.7}$$

Although the expectation over $p(\mathbf{x})$ can be approximated using the samples $\mathbf{x}_i$, the true score $\nabla \log p(\mathbf{x})$ is unknown, rendering the objective function intractable as is. *Denoising score-matching* [Vincent, 2011] approaches this by replacing $p(\mathbf{x})$ with an approximation that leads to a tractable objective. We can replace $p(\mathbf{x})$ with a *noisy approximation* $q_\sigma(\tilde{\mathbf{x}}) \coloneqq \int q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) p(\mathbf{x}) \, \mathrm{d}\mathbf{x}$ where $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \coloneqq \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 I)$ is the *noising distribution*. The advantage of doing so is that score-matching with $q_\sigma(\tilde{\mathbf{x}})$ is equivalent to the following tractable denoising objective by Proposition 1:

$$\min_{\theta} \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p(\mathbf{x})} \left[\left\| \varphi_\theta(\tilde{\mathbf{x}}) - \nabla \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \right\|^2 \right].$$

However, in order for this approximation to be accurate, it is *crucial that the noise level σ is as small as possible*, ideally zero in which case the approximation is exact. As shown by Song and Ermon [2019], choosing a value for σ close to zero leads to unreliable learned estimates of the score away from the regions of $p(\mathbf{x})$ with high probability density. Therefore, they proposed multi-scale denoising where the model simultaneously learns to approximate the scores of q_{σ_i} for $i = 1, \dots L$, where $\sigma_1 > \sigma_2 > \cdots > \sigma_L$ is a sequence of increasing noise levels such that σ_L is close to zero and $q_{\sigma_L} \approx p$. In practice, *this then leads to multi-scale denoising.*

In the context of learning forces, we first observe that the score function of the Boltzmann distribution $p(\mathbf{x}) = p_{\text{physical}}(\mathbf{x}) \propto \exp(-E(\mathbf{x}))$ is equal to the forces

$\nabla \log p_{\text{physical}}(\mathbf{x}) = -\nabla E(\mathbf{x})$. Therefore, learning the forces corresponds to the following score-matching objective:

$$\min_{\theta} \mathbb{E}_{p_{\text{physical}}(\mathbf{x})} \left[\left\| \varphi_\theta(\mathbf{x}) - \nabla \log p_{\text{physical}}(\mathbf{x}) \right\|^2 \right].$$

Once again, the score function $\nabla \log p_{\text{physical}}(\mathbf{x})$ is unknown and must be approximated. However, unlike previously, we now have access to a set of equilibrium, energy-minimizing structures $\mathbf{x}_1, \ldots, \mathbf{x}_n$; these examples are local maxima of the distribution $p_{\text{physical}}(\mathbf{x})$ rather than samples from it as in the case before. Let $q_\sigma(\tilde{\mathbf{x}})$ denote a mixture of Gaussians with noise scale σ centered at $\mathbf{x}_1, \ldots, \mathbf{x}_n$. In order to capture the fact that $p_{\text{physical}}(\mathbf{x})$ has local maxima at $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we can approximate $p_{\text{physical}}(\mathbf{x})$ with $q_\sigma(\tilde{\mathbf{x}})$. This distribution $q_\sigma(\tilde{\mathbf{x}})$ can also be written as $q_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) q_0(\mathbf{x}) \, \mathrm{d}\mathbf{x}$, where $q_0(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \delta(\mathbf{x} = \mathbf{x}_i)$, hence we can apply Proposition 1 again and reduce the optimization objective to denoising as explained in Section 3.3.2.1.

In contrast with before, in this case, a desirable noise scale σ is not zero, since $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are local maxima of p_{physical} and, intuitively, the approximation should contain some non-zero mass around each local maxima. As a result, we do not use multi-scale denoising in practice. It is also worth noting that in contrast with generative modelling, our aim is to learn the score function itself (*i.e.* forces) rather than producing new samples by using the learned score function with MCMC approaches.

3.8.3 Broader Impact

Who may benefit from this work? Molecular property prediction works towards a range of applications in materials design, chemistry, and drug discovery. Wider use of pre-trained models may accelerate progress in a similar manner to how pre-trained language or image models have enabled practitioners to avoid training on large datasets from scratch. Pre-training via denoising is simple to implement and can be immediately adopted to improve performance on a wide range of molecular property

prediction tasks. As research converges on more standardized architectures, we expect shared pre-trained weights will become more common across the community.

Potential negative impact and ethical considerations. Pre-training models on large structure datasets incurs additional computational cost when compared to training a potentially smaller model with less capacity from scratch. Environmental mitigation should be taken into account when pre-training large models [Patterson et al., 2021]. However, the computational cost of pre-training can and should be offset by sharing pre-trained embeddings when possible. Moreover, in our ablations of upstream dataset sizes for GNS-TAT, we observed that training on a subset of PCQM4Mv2 was sufficient for strong downstream performance. In future work, we plan to investigate how smaller subsets with sufficient diversity can be used to minimize computational requirements, *e.g.* by requiring fewer gradient steps.

3.8.4 Datasets

PCQM4Mv2. The main dataset we use for pre-training is PCQM4Mv2 [Nakata and Shimazaki, 2017] (license: CC BY 4.0), which contains 3,378,606 organic molecules, specified by their 3D structures at equilibrium (atom types and coordinates) calculated using DFT. Molecules in PCQM4Mv2 have around 30 atoms on average and vary in terms of their composition, with the dataset containing 22 unique elements in total. The molecules in PCQM4Mv2 only contain one label, unlike *e.g.* QM9, which contains 12 labels per molecule, however we do not use these labels as denoising only requires the structures.

QM9. QM9 is a dataset [Ramakrishnan et al., 2014a] (license: CCBY 4.0) with approximately 130,000 small organic molecules containing up to nine heavy C, N, O, F atoms, specified by their structures. Each molecule has 12 different labels corresponding to different molecular properties, such as highest occupied molecular orbital (HOMO) energy and internal energy, which we use for fine-tuning. Following prior work, we randomly split the dataset into 114k examples for training, 10k examples for validation and 10k examples for testing.

OC20. Open Catalyst 2020 [Chanussot* et al., 2021] (OC20, license: CC Attribution 4.0) is a recent large benchmark containing trajectories of interacting surfaces and adsorbates that are relevant to catalyst discovery and optimization. This dataset contains three tasks: predicting the relaxed state energy from the initial structure (IS2RE), predicting the relaxed structure from the initial structure (IS2RS) and predicting the energy and forces given the structure at any point in the trajectory (S2EF). For IS2RE and IS2RS, there are 460,000 training examples, where each data point is a trajectory of a surface-adsorbate molecule pair starting with a high-energy initial structure that is relaxed towards a low-energy, equilibrium structure. For S2EF, there are 113 million examples of (non-equilibrium) structures with their associated energies and per-atom forces. We evaluate the models on the provided validation sets, some of which include out-of-distribution data relative to the training data, as detailed in Chanussot* et al. [2021].

DES15K. DES15K [Donchev et al., 2021] (license: CC0 1.0) is a small dataset containing around 15,000 interacting molecule pairs, specifically dimer geometries with non-covalent molecular interactions. Each pair is labelled with the associated interaction energy computed using the coupled-cluster method with single, double, and perturbative triple excitations (CCSD(T)) [Bartlett and Musiał, 2007], which is widely regarded as the gold-standard method in electronic structure theory. We randomly split the dataset into 13k training examples, 1k validation examples and 1k testing examples.

Usage of DFT-generated structures for pre-training. The structures in PCQM4Mv2 are obtained using DFT calculations, which *in principle* could have also been used to generate labels for molecular properties in DFT-generated downstream datasets, such as QM9. However, there are multiple reasons why denoising remains a desirable pre-training objective in such settings. First, although there is a computational cost for generating datasets such as PCQM4Mv2, it is now openly available and part of our aim is to understand how to leverage such datasets for tasks on other datasets (analogous to how ImageNet is expensive to build, but once it is available, it is important to understand how it can improve downstream

performance on other datasets). Second, even if PCQM4Mv2 contained all the labels in QM9, pre-training via denoising structures allows one to pre-train a single, *label-agnostic* model which can be individually fine-tuned on any of the targets in QM9. This is substantially cheaper than per-target pre-training, and the resulting pre-trained model is also re-useable for differing needs and downstream tasks.

In other settings where the downstream dataset is generated using more expensive methods than DFT, pre-training via denoising DFT-relaxed structures can also be helpful and has a clear benefit, as shown by our experiment on the CCSD(T)-generated dataset DES15K where denoising on PCQM4Mv2 improves performance for a downstream task involving a "higher" level of theory such as CCSD(T). Generally, we emphasize that the methodology of denoising structures can be applied to any dataset of structures (regardless of whether they are computed using DFT or not). We hope that denoising will be useful in the future for learning representations by pre-training on structures obtained through other methods such as experimental data (in which case labeling may be expensive) and databases generated by other models such as AlphaFold (where only structures are available) [Jumper et al., 2021].

3.8.5 Experiment Setup and Compute Resources

Below, we list details on our experiment setup and hardware resources used.

GNS & GNS-TAT. GNS-TAT training for QM9, PCQM4Mv2 and DES15K was done on a cluster of 16 TPU v3 devices and evaluation on a single V100 device. GNS training for OC20 was done on 8 TPU v4 devices, with the exception of the 1.2 billion parameters variant of the model, which was trained on 64 TPU v4 devices. Pre-training on PCQM4Mv2 was executed for $3 \cdot 10^5$ gradient updates (approximately 1.5 days of training). Fine-tuning experiments were run until convergence for QM9 (10^6 gradient updates taking approximately 2 days) and DES15K (10^5 gradient updates taking approximately 4 hours) and stopped after $5 \cdot 10^5$ gradient updates on OC20 (2.5 days) to minimize hardware use (the larger models keep benefiting from additional gradient updates).

TorchMD-NET. We implemented denoising for TorchMD-NET on top of Thölke and De Fabritiis's [2022] open-source code.[6] Models were trained on QM9 using data parallelism over two NVIDIA RTX 2080Ti GPUs. Pre-training on PCQM4Mv2 was done using three GPUs to accommodate the larger molecules while keeping the batch size approximately the same as QM9. All hyperparameters except the learning rate schedule were kept fixed at the defaults. Pre-training took roughly 24 hours, whereas fine-tuning took around 16 hours.

Hyperparameter optimization. We note that effective pre-training via denoising requires sweeping noise values, as well as loss co-efficients for denoising and atom type recovery. For GNS/GNS-TAT, we relied on the hyperparameters published by Godwin et al. [2022] but determined new noise values for pre-training and fine-tuning by tuning over the set of values $\{0.005, 0.01, 0.02, 0.05, 0.1\}$ for each of PCQM4Mv2 and QM9 (on the HOMO energy target). We used the same values for DES15K without modification. We also ran a similar number of experiments to determine cosine cycle parameters for learning rates.

3.8.6 Hyperparameters

We report the main hyperparameters used for GNS and GNS-TAT below.

3.8.7 Additional Experimental Results

3.8.7.1 Performance of TorchMD-NET on DES15K

In addition to the experiments involving GNS-TAT on DES15K in Section 3.4.4, we also consider the performance of TorchMD-NET on DES15K with and without pre-training on PCQM4Mv2. As shown in Table 3.8, TorchMD-NET outperforms GNS-TAT when trained from scratch, and pre-training then yields a further boost in performance as with GNS-TAT.

[6] Available on GitHub at: `https://github.com/torchmd/torchmd-net`.

Table 3.5: GNS-TAT hyperparameters for pre-training on PCQM4Mv2.

Parameter	Value or description
Gradient steps	$3 \cdot 10^5$
Optimizer	Adam with warm up and 1-cycle cosine decay schedule
β_1	0.9
β_2	0.95
Warm up steps	10^4
Warm up start learning rate	10^{-5}
Warm up max learning rate	10^{-4}
Cosine min learning rate	10^{-7}
Cosine cycle length	$5 \cdot 10^5$
Loss type	Mean squared error
Batch size	Dynamic to max edge/vertex/graph count
Max vertices in batch	256
Max edges in batch	9216
Max graphs in batch	8
Distance featurization	Bessel first kind ($r_{\min} = 0, \sigma = 1.0$)
Max edges per vertex	20
R_{cut}	10
MLP number of layers	3
MLP hidden sizes	1024
Activation	Tailored ReLU (with negative slope chosen using TAT)
message passing layers	10
Block iterations	3
Vertex/edge latent vector sizes	512
Decoder aggregation	Mean
Position noise	Gaussian ($\mu = 0, \sigma = 0.02$)
Parameter update	Exponentially moving average (EMA) smoothing
EMA decay	0.9999
Position loss coefficient	1.0
Atom type mask probability	0.75
Atom type loss coefficient	4.0

3.8.7.2 Comparison of OC20 IS2RE Pre-training Performance with Other Architectures

Table 3.9 compares the performance of our models with various other architectures proposed in prior work. GNS yields SOTA performance on each of the four validation sets for the direct IS2RE task. As discussed in Section 3.4.3 and shown in Figure 3.3 (left), the model pre-trained on OC20 itself achieves SOTA performance with faster convergence than all other GNS variants. Note that since we pre-train on OC20 itself, the pre-trained model performs equally well at convergence as the model

Table 3.6: GNS-TAT hyperparameters for fine-tuning on QM9 and DES15K.

Parameter	Value or description
Gradient steps	10^6 QM9 / 10^5 DES15K
Optimizer	Adam with warm up and 1-cycle cosine decay schedule
β_1	0.9
β_2	0.95
Warm up steps	10^4
Warm up start learning rate	10^{-5}
Warm up max learning rate	10^{-4}
Cosine min learning rate	$3 \cdot 10^{-7}$
Cosine cycle length	10^6 QM9 / 10^5 DES15K
Loss type	Mean squared error
Batch size	Dynamic to max edge/vertex/graph count
Max vertices in batch	256
Max edges in batch	3072
Max graphs in batch	8
Distance featurization	Bessel first kind ($r_{\min} = 0, \sigma = 1.0$)
Max edges per vertex	20
R_{cut}	10
MLP number of layers	3
MLP hidden sizes	1024
Activation	Tailored ReLU (with negative slope chosen using TAT)
message passing layers	10
Block iterations	3
Vertex/edge latent vector sizes	512
Decoder aggregation	Mean
Position noise	Gaussian ($\mu = 0, \sigma = 0.05$)
Parameter update	Exponentially moving average (EMA) smoothing
EMA decay	0.9999
Position loss coefficient	0.01
Atom type mask probability	0.0
Atom type loss coefficient	0.0

trained from scratch with noisy nodes (*cf.* Section 3.4.3).

3.8.7.3 Pre-training via Denoising for Force Prediction on OC20

Recall that in Section 3.5.4 we explored whether pre-training also improves force prediction models, given the link between denoising and learning forces as described in Section 3.3.2.1. In this section, we consider a second experiment for force models. The OC20 dataset contains a force prediction task (S2EF), where a model is trained to predict point-wise energy and forces (each point being a single DFT evaluation

Table 3.7: GNS hyperparameters for OC20.

Parameter	Value or description
Gradient steps	$5 \cdot 10^5$
Optimizer	Adam with warm up and 1-cycle cosine decay schedule
β_1	0.9
β_2	0.95
Warm up steps	$5 \cdot 10^5$
Warm up start learning rate	10^{-5}
Warm up max learning rate	10^{-4}
Cosine min learning rate	$5 \cdot 10^{-6}$
Cosine cycle length	$5 \cdot 10^6$
Loss type	Mean squared error
Batch size	Dynamic to max edge/vertex/graph count
Max vertices in batch	1024
Max edges in batch	12800
Max graphs in batch	10
Distance featurization	Gaussian ($\mu = 0, \sigma = 0.5$)
Max edges per vertex	20
R_{cut}	6
MLP number of layers	3
MLP hidden sizes	1024
Activation	shifted softplus
message passing layers	5
Block iterations	5
Vertex/edge latent vector sizes	512
Decoder aggregation	Sum
Position noise	Gaussian ($\mu = 0, \sigma = 0.2$)
Parameter update	Exponentially moving average (EMA) smoothing
EMA decay	0.9999
Position loss coefficient	1.0
Atom type mask probability	0.0
Atom type loss coefficient	0.0

Table 3.8: Performance of TorchMD-NET with and without pre-training for interaction energy prediction on DES15K.

Model	Test MAE (kcal/mol)
TorchMD-NET	0.721
+ Pre-training on PCQM4Mv2	**0.406**

during a relaxation trajectory) from a given 3D structure. Tables 3.10 and 3.11 show the performance of GNS when trained from scratch vs. pre-trained via denoising on equilibrium structures in OC20, as described in Section 3.4.3. We show two metrics

Table 3.9: Comparison of different variants of GNS with other baseline architectures on IS2RE prediction for OC20.

Model	Validation MAE for IS2RE				
	ID	OOD Adsorbate	OOD Catalyst	OOD Both	Average
DimeNet ++	0.5636	0.7127	0.5612	0.6492	0.6217
GemNet	0.5561	0.7342	0.5659	0.6964	0.6382
SphereNet	0.5632	0.6682	0.5590	0.6190	0.6024
SEGNN	0.5310	0.6432	0.5341	0.5777	0.5715
GNS	0.5233	0.6295	0.5202	0.5617	0.5587
GNS + NN	0.4196	0.4900	0.4316	0.4282	0.4424
GNS + NN (PT on PCQM4Mv2)	**0.4135**	0.4856	**0.4245**	0.4245	**0.4370**
GNS + NN (PT on OC20)	0.4164	**0.4836**	0.4267	**0.4237**	0.4376

for measuring force prediction performance on each of the four validation datasets: mean absolute error (lower is better) and cosine similarity (higher is better). We observe that the pre-trained model improves upon the model trained from scratch for both metrics and all four validation datasets, with improvements up to 15%.

Table 3.10: Force prediction on OC20 by MAE (lower is better).

Validation Dataset	Model	
	GNS	Pre-trained GNS
ID	0.0332	**0.0282**
OOD Adsorbate	0.0366	**0.0314**
OOD Catalyst	0.0360	**0.0335**
OOD Both	0.0406	**0.0382**

Table 3.11: Force prediction on OC20 by cosine similarity (higher is better).

Validation Dataset	Model	
	GNS	Pre-trained GNS
ID	0.4845	**0.5517**
OOD Adsorbate	0.4730	**0.5414**
OOD Catalyst	0.4553	**0.4983**
OOD Both	0.4417	**0.4849**

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Pre-training via Denoising for Molecular Property Prediction
Publication Status	☒Published ☐ Accepted for Publication ☐Submitted for Publication ☐Unpublished and unsubmitted work written in a manuscript style
Publication Details	Sheheryar Zaidi*, Michael Schaarschmidt*, James Martens, Hyunjik Kim, Yee Whye Teh, Alvaro Sanchez-Gonzalez, Peter Battaglia, Razvan Pascanu, Jonathan Godwin. "Pre-training via Denoising for Molecular Property Prediction." *International Conference on Learning Representations,* 2023.

Student Confirmation

Student Name:	Sheheryar Zaidi		
Contribution to the Paper	☐ Hyunjik, Razvan and I initiated the project. ☐ I formulated the idea of exploring denoising as a form for pre-training. ☐ I proved the results connecting denoising with force learning via score-matching, with advice from Jonathan on background papers. ☐ I did all the dataset analysis. ☐ I designed and ran most of the experiments in the paper, with help from Michael (who reran multiple results for the final draft) and James (who developed DKS/TAT for GNS with some help from me for implementing the delta initialization schemes). ☐ Michael and I produced the plots in the paper. ☐ I wrote most of the paper with help from Michael, James and Jonathan for certain sections. ☐ All authors contributed to the development of the project through discussions and reviewed the final draft.		
Signature		Date	01/02/2023

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Prof Yee Whye Teh		
Supervisor comments	Looks good to me.		
Signature		Date	31/10/2023

This completed form should be included in the thesis, at the end of the relevant chapter.

4

When Does Re-initialization Work?

Abstract: Re-initializing a neural network during training has been observed to improve generalization in recent works. Yet it is neither widely adopted in deep learning practice nor is it often used in state-of-the-art training protocols. This raises the question of when re-initialization works, and whether it should be used together with regularization techniques such as data augmentation, weight decay and learning rate schedules. In this work, we conduct an extensive empirical comparison of standard training with a selection of re-initialization methods to answer this question, training over 15,000 models on a variety of image classification benchmarks. We first establish that such methods are consistently beneficial for generalization in the absence of any other regularization. However, when deployed alongside other carefully tuned regularization techniques, re-initialization methods offer little to no added benefit for generalization, although optimal generalization performance becomes less sensitive to the choice of learning rate and weight decay hyperparameters. To investigate the impact of re-initialization methods on noisy data, we also consider learning under label noise. Surprisingly, in this case, re-initialization significantly improves upon standard training, even in the presence of other carefully tuned regularization techniques.

Contents

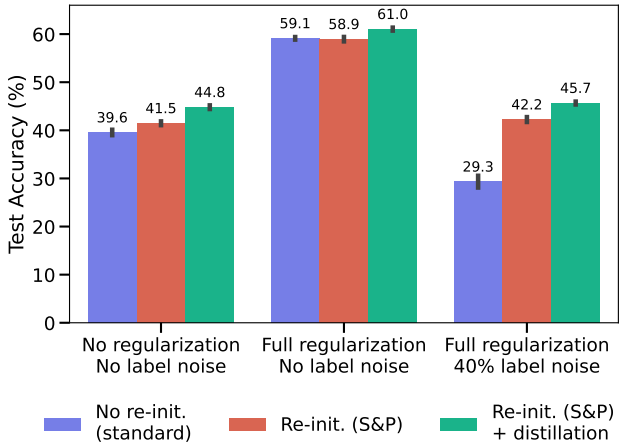


Figure 4.1: Comparison of standard training (*i.e.* no re-initialization) with re-initialization using Shrink & Perturb in three scenarios on Tiny ImageNet with PreAct-ResNet-18. Full regularization refers to the usage of carefully tuned data augmentation, weight decay and a learning rate schedule, whereas no regularization is without. Without label noise, Shrink & Perturb helps in the absence of other regularization but has limited benefit otherwise. With label noise, Shrink & Perturb helps significantly even alongside other regularization. Adding distillation further improves performance. All training protocols have approximately equal computational cost.

4.1 Introduction

Recent works (*e.g.* [Alabdulmohsin et al., 2021] and [Ash and Adams, 2020]) have proposed a set of techniques for training neural networks based on *re-initialization*. These methods, which we collectively refer to as *re-initialization methods*, involve re-initializing and transforming a part or all of the parameters of a neural network periodically throughout learning. Studies have shown that re-initialization can help in certain settings, such as small-data regimes [Alabdulmohsin et al., 2021, Taha et al., 2021] and online learning [Ash and Adams, 2020]. However, despite having no overhead in terms of computation cost and small implementation overhead, such re-initialization techniques have not yet been adopted as common deep learning practice. Indeed, most state-of-the-art (SOTA) training protocols do not incorporate re-initialization techniques, relying instead on advances in *e.g.* optimization [Foret et al., 2020], architectures [Liu et al., 2021c, Tan and Le, 2019, Brock et al., 2021], data augmentation [Cubuk et al., 2018, 2020] and pre-training [Chen et al., 2020].

However, prior work suggests that re-initialization can improve the generalization

performance of neural networks. For example, in the context of online learning, Ash and Adams [2020] studied a scenario in which training data arrives sequentially in "chunks" over time such that at any point in time the training dataset consists of the union of all chunks arrived so far. They compared training the network from scratch each time a new chunk arrives to *warm-starting*, where we continuously train (*i.e.* fine-tune) the model, finding that warm-starting significantly underperformed. In order to remedy this, they proposed Shrink & Perturb (defined in Section 4.3.2), a re-initialization technique applied each time a new chunk arrives. Interestingly, applying Shrink & Perturb not only closed the performance gap between warm-starting and training from scratch, but *improved upon the model trained from scratch*. Does this mean that the standard approach to training neural networks without re-initialization is sub-optimal?

The motivation of our work is to systematically understand the benefits and limitations of re-initialization methods in training neural networks. To that end, we trained over 15,000 models to identify the settings under which re-initialization methods are helpful. We study the interaction between re-initialization and other widely used regularization and optimization techniques, including data augmentation, weight decay and learning rate schedules. We seek to answer a fundamental, practical question: are the benefits of re-initialization additive with those of common, existing techniques for improving generalization? Should re-initialization be present in the arsenal of deep learning practices? Our experiments therefore include careful ablations in which we consider settings ranging from vanilla training to SOTA protocols which yield top performance for a given architecture. In addition to varying the training protocol, we also investigate the impact of re-initialization methods when learning on noisy data by studying what happens under the presence of label noise, leading to some surprising results.

Our empirical study centers around two re-initialization methods. First, as our primary focus, we consider *Shrink & Perturb*, which was proposed by Ash and Adams [2020] in the context of online learning and warm-starting. To the

best of our knowledge, the regularization benefits of Shrink & Perturb as a re-initialization method for standard i.i.d. supervised learning have not been studied before, and we find that an adaptation of Shrink & Perturb can sometimes lead to significant gains in this setting. Second, we consider the recently proposed *Layer-wise Re-initialization* [Alabdulmohsin et al., 2021] that has been shown to outperform prior re-initialization schemes on a variety of datasets. Furthermore, we show that re-initialization methods can naturally be adapted to incorporate self-distillation [Furlanello et al., 2018] which typically leads to improvement, at negligible cost. Figure 4.1 summarizes our results on Tiny ImageNet. From here on, we will use *standard training* to refer to training *without* re-initialization.

Our contributions and findings are as follows:

- **Shrink & Perturb can benefit i.i.d. learning.** We investigate the benefits of *Shrink & Perturb*, previously proposed for online learning, and its variant with distillation as re-initialization methods that can be used for i.i.d supervised learning. We show that they are helpful and outperform techniques such as Layer-wise Re-initialization in certain settings.

- **Re-initialization improves generalization in the absence of other regularization (Section 4.4).** There is a consistent advantage in periodically re-initializing a neural network—even up to 25 times—during training, pointing to the inherent regularization benefit of re-initialization.

- **Re-initialization has limited advantage over standard training in a SOTA setting (Section 4.5).** When data augmentation, weight decay and learning rate schedules are carefully tuned, re-initialization performs at par with standard training. However, the optimal performance becomes *more robust* to the choice of learning rate and weight decay, which is desirable in practice.

- **Under label noise, re-initialization methods lead to significant improvement in generalization (Section 4.6).** This improvement appears

on top of other well-tuned regularization, revealing a setting where the effects of re-initialization and other regularization techniques do not overlap.

4.2 Related Work

Various re-initialization methods have been proposed in the literature, differing usually by motivation and choice of weights that are re-initialized. For example, motivated by sparsity, Han et al. [2016] introduced dense-sparse-dense training, where after an initial stage of training, the smallest weights are pruned to induce sparsity, the network is trained and in the third and last stage, the pruned weights are re-initialized from zero. Taha et al. [2021] also proposed "knowledge evolution" which partitions the weights of a network into two parts, one of which is continuously training and the other repeatedly re-initialized. For transfer learning, Li et al. [2020] proposed to periodically re-initialize only the final layer, optionally with ensembling [Zhao et al., 2018]. Recently, Alabdulmohsin et al. [2021] compared a number of these methods with their proposed Layer-wise Re-initialization. They found it outperformed prior approaches, which is why we chose to study it in this work. Their empirical evaluation focused on the small-data regime, where one set of hyperparameters common across multiple datasets and architectures are used. They also did not study how re-initialization methods interact with other regularizers. Also, in the spirit of re-initialization, various approaches for "resetting" or "restarting" training can also be found in the literature [Loshchilov and Hutter, 2016, Huang et al., 2017b, Smith, 2017, Izmailov et al., 2018].

Re-initialization also recently appeared in the context of online learning and warm-starting neural networks, where data arrives sequentially [Ash and Adams, 2020, Caccia et al., 2021]. Ash and Adams [2020] showed that *warm-starting* neural network training worsens generalization. They proposed Shrink & Perturb, a simple technique to improve performance in their online learning setup. The benefits of Shrink & Perturb as a generic re-initialization method for i.i.d. learning have not been studied before. Caccia et al. [2021] studied techniques for expanding capacity by initializing *new* parameters as more data arrives. Igl et al. [2021] observed a

similar phenomenon in reinforcement learning and proposed distillation as a remedy. Generally, self-distillation [Furlanello et al., 2018, Hinton et al., 2015] itself closely relates to re-initialization (see Section 4.3.1).

4.3 Background on Re-initialization Methods

Let $f_{\boldsymbol{\theta}}$ be a neural network with parameters $\boldsymbol{\theta} \in \mathbb{R}^p$. We will assume $f_{\boldsymbol{\theta}}$ is trained to minimize some loss function $L(\boldsymbol{\theta})$. Moreover, let p_{init} be a probability distribution over $\mathbb{R}^p$ from which we sample the *initialization parameters* $\boldsymbol{\theta}_0 \sim p_{\text{init}}$, where optimization begins. Then, we define a *re-initialization* of $f_{\boldsymbol{\theta}}$ to be the function $f_{\boldsymbol{\theta}_{\text{RI}}}$, where we have replaced the current parameters $\boldsymbol{\theta}$ with re-initialized parameters $\boldsymbol{\theta}_{\text{RI}} = R(\boldsymbol{\theta}_{\text{init}}, \boldsymbol{\theta})$. The function R computes the re-initialized parameters using freshly initialized parameters $\boldsymbol{\theta}_{\text{init}} \sim p_{\text{init}}$ and the current parameters $\boldsymbol{\theta}$. Different re-initialization methods will differ in terms of R.

Re-initialization methods will modify standard training as follows. Assuming a computational budget corresponding to N epochs over the training data, we train the model in T *stages*. Each stage will consist of training the network for $\lfloor N/T \rfloor$ epochs. Between any two stages, we apply re-initialization as described earlier. This ensures that the computational cost in terms of the total number of gradient steps is approximately equal to the cost of standard training for N epochs. This is key to ensure that the performance of different methods can be compared fairly. Explicitly, let $\boldsymbol{\theta}_{t-1}^{\text{end}}$ denote the parameters at the end of the previous stage $t-1$. Stage t then consists of training the model starting from initial parameters $\boldsymbol{\theta}_t^{\text{start}} = R(\boldsymbol{\theta}_{\text{init}}, \boldsymbol{\theta}_{t-1}^{\text{end}})$. The initial parameters for stage 1 are sampled directly from p_{init}. See Algorithm 1 for a pseudo-code description.

4.3.1 Incorporating Self-Distillation With Re-initialization

Our experiments will also consider the effect of incorporating *self-distillation*, which is a natural extension of re-initialization methods because training is split into multiple sequential stages. Specifically, when using distillation, we train the model in stage t by minimizing a combination of the training loss and a distillation loss

between the current model (the student) and the model at the end of the previous stage $t-1$ (the teacher), that is, we minimize the loss

$$L(\boldsymbol{\theta}) + \beta_{\text{distill}}\,\text{KL}(f_{\boldsymbol{\theta}_{t-1}^{\text{end}}} \parallel f_{\boldsymbol{\theta}}), \tag{4.1}$$

with respect to $\boldsymbol{\theta}$. The second term measures the KL-divergence between the predicted class probabilities of the teacher and the student on the training data, and β_{distill} is a hyperparameter determining the strength of distillation. In stage $t = 1$, we only optimize the loss $L(\boldsymbol{\theta})$. Note that this requires generating the predictions of the teacher network at the end of the previous stage and caching them for use during the optimization of the student in the current stage. The added computational cost is negligible, and we found that the training time with and without distillation did not vary much. We also remark that the setting in which re-initialization simply consists of replacing all the old model parameters with a new initialization, *i.e.* $\boldsymbol{\theta}_{\text{RI}} = \boldsymbol{\theta}_{\text{init}}$, combined with distillation is equivalent to Born-Again Networks (BANs) [Furlanello et al., 2018] supervised by the true and teacher labels, with a *fixed* computational budget.[1]

4.3.2 Shrink & Perturb

Our primary focus in this work will be Shrink & Perturb, a method proposed by Ash and Adams [2020] in the context of online learning and warm-starting neural network training. Ash and Adams [2020] studied a scenario in which training data arrives sequentially in "chunks" over time such that at any point in time the training dataset consists of the union of all chunks arrived so far. They compared re-training the network from scratch each time a new chunk arrives to *warm-starting*, where we continuously train (*i.e.* fine-tune) the model. They found that the warm-started network significantly underperformed. In order to remedy this, they proposed Shrink & Perturb, a re-initialization technique applied

[1] In the original BANs, the computational cost scaled proportionally with the number self-distillation iterations. For a fair comparison, we consider BANs under a fixed total budget.

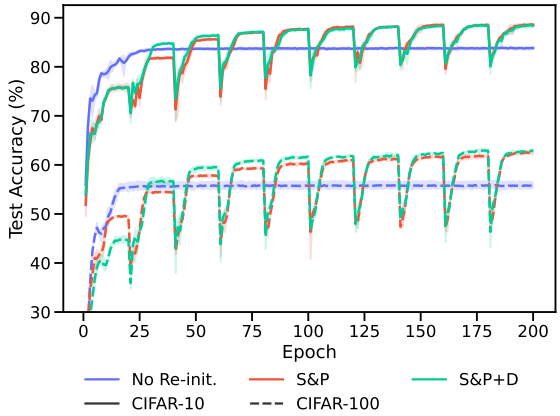


Figure 4.2: Example test accuracy curves on the CIFAR datasets with ResNet-18 in setting ∅. Shrink & Perturb involves 10 stages. For each method and dataset, the learning rate is tuned separately – the optimal learning rates are on average 10 times smaller for re-initialization than standard training here.

to the previous model each time a new chunk arrives. Shrink & Perturb consists of defining the re-initialized parameters to be:

$$\boldsymbol{\theta}_{\mathrm{RI}} = \lambda\boldsymbol{\theta} + \gamma\boldsymbol{\theta}_{\mathrm{init}},$$

where $\lambda, \gamma \in [0, 1]$ are hyperparameters. We *shrink* the current parameters by a multiplicative factor of λ and *perturb* them using a fresh initialization scaled by γ. In our initial experiments, we tuned these hyperparameters which then remain fixed at $\lambda = 0.4, \gamma = 0.1$ across all architectures and datasets and are similar to the optimal values in the setup of Ash and Adams [2020]. The number of stages T is a hyperparameter we vary in our experiments.

Shrink & Perturb was demonstrated to improve upon re-training from scratch in the online learning setup. Ash and Adams [2020] suggested that it may have a helpful regularization effect, but this effect has not been studied in any detail in the usual i.i.d. learning setting. Contrary to their setup where Shrink & Perturb is applied for online learning, we will view Shrink & Perturb as a re-initialization method for training on a stationary dataset as described earlier.

4.3.3 Layer-wise Re-initialization

Alabdulmohsin et al. [2021] recently proposed *Layer-wise Re-initialization*, which re-initializes the architecture block-by-block during training. Assuming that there

Table 4.1: Test accuracy (%) of different methods in settings ranging from basic to SOTA protocols on CIFAR-10 with ResNet-18.

Setting Abbrev.	Data Aug.	Cosine Anneal.	Weight Decay	No Re-initialization (standard training)	Self-distillation (fixed-budget BAN)	Layer-wise Re-initialization		Shrink & Perturb	
						w/o dist.	w/ dist.	w/o dist.	w/ dist.
∅	✗	✗	✗	83.8 ±0.3	84.0 ±0.4	87.6 ±0.4	87.3 ±0.2	**88.8** ±0.2	**88.6** ±0.4
D	✓	✗	✗	92.5 ±0.0	92.6 ±0.2	92.6 ±0.3	93.2 ±0.1	93.1 ±0.2	**94.1** ±0.0
DC	✓	✓	✗	92.8 ±0.2	92.6 ±0.1	93.4 ±0.1	93.4 ±0.2	**94.1** ±0.2	**94.2** ±0.1
DCW	✓	✓	✓	95.0 ±0.0	94.8 ±0.2	94.7 ±0.2	95.0 ±0.3	94.7 ±0.2	94.7 ±0.1

are K "blocks" in the architecture, we have a total of $T = K$ stages during training. At the end of stage t, we re-initialize all layers after the t-th block, that is, we set:

$$\boldsymbol{\theta}_{\text{RI}} = \boldsymbol{\theta} \odot \boldsymbol{m}^{(t)} + \boldsymbol{\theta}_{\text{init}} \odot (1 - \boldsymbol{m}^{(t)}).$$

Here $\boldsymbol{m}^{(t)} \in \{0, 1\}^p$ is a vector which masks all layers after the first t blocks, *i.e.* $\boldsymbol{m}_i^{(t)} = 1$ if the i-th parameters belongs to the first t blocks and is otherwise zero. $\odot$ denotes an element-wise product. More generally, note that we can have $T = KM$ total stages, where at the end of each of the first M stages $t = 1, \ldots, M$, we re-initialize all layers after the first block. At the end of each of the second M stages $t = M + 1, \ldots, 2M$, we re-initialize all layers after the second block and so forth. We note that Alabdulmohsin et al. [2021] set $M = 1$ in their main experiments.

Moreover, in addition to re-initializing all blocks after the t-th block, Layer-wise Re-initialization also (1) re-scales the norm of the first t blocks to their norm at initialization and (2) adds a normalization layer (without trainable parameters) after block t whose mean and standard deviation are computed by forward passing a batch of training data. Note that unlike Shrink & Perturb, Layer-wise Re-initialization has a somewhat architecture-specific definition, since it requires picking *a priori* what a block corresponds to in the architecture. In our experiments with a ResNet-18 architecture, we considered four cases: $T = K = 2$ where the network is split between the second and third residual blocks, $T = K = 5$ where we split by each residual block and lastly $T = 10$ and 20 by letting $M = 2$ and 4 respectively in the previous case.

Table 4.2: Test accuracy (%) of different methods in settings ranging from basic to SOTA protocols on CIFAR-100 with ResNet-18.

Setting Abbrev.	Data Aug.	Cosine Anneal.	Weight Decay	No Re-initialization (standard training)	Self-distillation (fixed-budget BAN)	SGDR	Layer-wise Re-initialization		Shrink & Perturb	
							w/o dist.	w/ dist.	w/o dist.	w/ dist.
∅	✗	✗	✗	55.5 ±0.6	56.4 ±0.5	N/A	61.0 ±0.6	62.5 ±0.2	63.1 ±0.6	**63.5** ±0.3
D	✓	✗	✗	70.8 ±0.1	70.5 ±0.5	N/A	72.1 ±0.3	**74.7** ±0.2	71.9 ±0.1	74.0 ±0.6
DC	✓	✓	✗	71.2 ±0.2	70.9 ±0.4	71.0 ±0.6	74.6 ±0.5	**75.4** ±0.2	**75.4** ±0.3	**75.4** ±0.4
DCW	✓	✓	✓	**77.9** ±0.2	77.2 ±0.1	77.5 ±0.2	77.5 ±0.1	77.3 ±0.3	77.5 ±0.2	77.0 ±0.3

Table 4.3: Test accuracy (%) of standard training and Shrink & Perturb on Tiny ImageNet with PreAct-ResNet-18.

Setting Abbrev.	Data Aug.	Cosine Anneal.	Weight Decay	No Re-initialization (standard training)	Shrink & Perturb	
					w/o dist.	w/ dist.
∅	✗	✗	✗	39.55±0.69	41.47±0.48	44.81±0.46
DCW	✓	✓	✓	59.12±0.33	58.95±0.56	61.03±0.38

4.4 The Regularizing Effect of Re-initialization

In this section, we investigate the effect of re-initialization when no other regularization is deployed. We show that re-intiliazation provides a consistent and considerable advantage over standard training, even beyond the small-data regime [Alabdulmohsin et al., 2021, Taha et al., 2021], as is the case of Tiny ImageNet. This highlights their effectiveness as a regularization technique. Additionally, we find that Shrink & Perturb outperforms Layer-wise Re-initialization in this setting.

4.4.1 Experimental Setup

For each architecture (ResNet-18, PreAct-ResNet-18, MobileNetV2) [He et al., 2016c,b, Sandler et al., 2018] and dataset (CIFAR-10, CIFAR-100, Tiny ImageNet), we used SGD with momentum 0.9 for training with early stopping by validation accuracy. For the CIFAR datasets, we separately tuned the learning rate and weight decay hyperparameters for each method by cross validation. Note that this is different to the setup of Alabdulmohsin et al. [2021] where the learning rate and weight decay hyperparameters were fixed for all combinations of datasets and architectures in their image classification experiments. For the larger and more computationally costly Tiny ImageNet dataset, we tuned the learning rate and weight decay hyperparameters for standard training (that is, no re-initialization) and

kept them fixed for the re-initialization methods. This simulates a practical scenario: given tuned hyperparameters for standard training on an expensive and large dataset, does it help to use re-initialization without re-tuning the hyperparameters?

For re-initialization methods, we note that the number of stages T is an additional hyperparameter which we tune (*cf.* Section 4.5.5). Moreover, all models are optimized for the same number of gradient steps for all methods. All other hyperparameters such as batch size and number of epochs are the same for all methods and specified in Tables 4.4 and 4.5 of Section 4.8.2. Our code is provided for reproducibility and will be open-sourced.

4.4.2 Re-initialization Improves Performance in the Absence of Other Regularization

The different settings we consider are labeled ∅, D, DC and DCW, as described on the left of Table 4.1. Setting ∅ is the basic setting where we use a constant learning rate, no weight decay and no data augmentation. As shown in the first rows of Tables 4.1 and 4.2, both Shrink & Perturb and Layer-wise Re-initialization improve generalization performance over standard training for both CIFAR datasets. Moreover, Shrink & Perturb outperforms Layer-wise Re-initialization by a margin of 1-2 percentage points in both cases. Both re-initialization methods also benefit from incorporating distillation for CIFAR-10, although distillation matters less for CIFAR-100. Next, we consider training in setting ∅ for Tiny ImageNet, focusing only on Shrink & Perturb with and without distillation in this case, as shown in Table 4.3. As before, re-initialization is beneficial compared to standard training, particularly when we include distillation. Indeed, these results indicate that re-initialization yields a notable generalization boost in setting ∅ without additional computational cost.

Examples of the test accuracy learning curves are shown in Figure 4.2 for Shrink & Perturb with 10 stages. Notice that each re-initialization causes a sudden drop in performance from which there is a quick recovery. Whereas the test accuracy for standard training stagnates early on, training with Shrink & Perturb keeps improving test accuracy with each successive re-initialization and already outperforms standard

training by epoch 50. All models shown in this plot achieved approximately 100% training accuracy and close to zero training negative log-likelihood. See Figures 4.8 and 4.9 for examples of training curves.

4.5 Re-initialization Alongside Other Regularization

Section 4.4 demonstrates that re-initialization methods have a regularizing effect on learning that improves generalization. Next, we consider how beneficial re-initialization is when deployed alongside other regularization techniques that are commonly used to achieve high performance in SOTA training protocols. In particular, we consider three common regularization techniques: data augmentation, cosine annealing and weight decay. Note that while cosine annealing is not an explicit regularizer, the choice of learning rates affects regularization [Li et al., 2019, Li and Arora, 2019] and is an important ingredient for well-performing models. For re-initialization methods, we apply cosine annealing *per stage*, yielding a cyclical learning rate [Loshchilov and Hutter, 2016]. Our motivation is to explore whether the effect of these different techniques is *additive* with the benefit of re-initialization; indeed, regularizers that are helpful each on their own are not necessarily helpful in the presence of one another.

4.5.1 Re-initialization Offers Little Benefit in SOTA Settings

Using a similar setup as before, we compare standard training with various re-initialization methods across these settings on the CIFAR datasets in Tables 4.1 and 4.2. Beginning with the simple setting explored in Section 4.4, we gradually add **d**ata augmentation (setting D), **c**osine annealing (setting DC) and **w**eight decay (setting DCW), achieving around SOTA performance in setting DCW. First, we observe that all re-initialization methods except fixed-budget BAN noticeably improve upon standard training in settings ∅, D and DC. In most cases, the best performing method is Shrink & Perturb with distillation, improving accuracy over

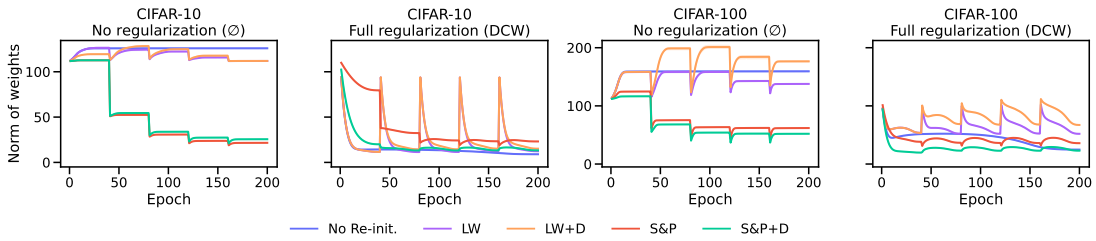


Figure 4.3: Weight norms during training vary across methods more in setting ∅ than setting DCW. Curves show how the norm of ResNet-18 weights varies through training for re-initialization methods (with 5 stages) and standard training. In setting ∅, Shrink & Perturb yields a model with smaller weight norm than standard training and Layer-wise Re-initialization, but in setting DCW, all methods yield models with similar weight norms.

standard training by upto 5 percentage points on CIFAR-10 and 8 percentage points on CIFAR-100.

However, a key finding here is that *when all regularization techniques are used and carefully tuned (setting DCW), the best re-initialization methods do not improve generalization performance over standard training.* Therefore, stacking re-initialization on top of other regularization techniques results in a *sub-additive* effect on performance. Note that this setting is arguably more important and common in practice than previous settings. We speculate that Shrink & Perturb affects the norm of the weights during training which may overlap with the effect of weight decay. As shown in Figure 4.3, in the absence of weight decay, the norm of the weights tends to increase monotonically without re-initialization, whereas Shrink & Perturb periodically reduces this weight norm. In setting ∅, this results in the learned models' weight norms varying across methods, while, in setting DCW, all methods learn models with similar final norms due to weight decay. Moreover, in setting ∅, Shrink & Perturb results in a model with norm similar to that of standard training in setting DCW. Therefore, both weight decay and Shrink & Perturb encourage the final learned model to have a small weight norm, a quantity that is known to be linked to generalization performance [Neyshabur et al., 2018, Bartlett et al., 2017]. Nevertheless, this does not fully explain why Shrink & Perturb works, as shown later in Section 4.6. We also show the impact of re-initialization in setting DCW for Tiny ImageNet in the last row of Table 4.3. Similar to before, the improvement in performance from using Shrink & Perturb is much smaller here

than in setting ∅. Shrink & Perturb has no impact on performance, whereas Shrink & Perturb with distillation slightly improves performance over standard training.

4.5.2 The Role of Self-Distillation

Another observation from Tables 4.1 and 4.2 is that fixed-budget BANs do not improve performance compared to standard training in most cases, sometimes leading to worse test accuracies, and are outperformed by Shrink & Perturb and Layer-wise Re-initialization in all settings except DCW. Recall from Section 4.3.1 that BANs can be viewed as a simple re-initialization method combined with distillation: we re-initialize the full network *from scratch* in each stage. Therefore, BANs constitute an important baseline for more sophisticated re-initialization methods. Our results indicate that while distillation can boost performance, it is computationally sub-optimal to *completely* re-initialize each network in the sequence. Under a fixed-budget setting as shown here, too many stages will then lead to too few gradient steps per stage, whereas too few stages do not reap the full benefits of distillation, both of which can negatively impact performance. On the other hand, Layer-wise Re-initialization and Shrink & Perturb partly "re-use" the model from the previous stage, circumventing the need for a large number of gradient steps per stage and improving learning efficiency. Therefore, re-initialization is itself crucial, and distillation does not suffice on its own. Figure 4.7 shows how complete re-initialization in BANs leads to large drops in performance from which it is more difficult to recover quickly.

4.5.3 The Role of Cosine Annealing

Recall that cosine annealing is applied per stage for re-initialization methods, yielding a cyclical learning rate schedule as used in SGDR [Loshchilov and Hutter, 2016]. Such a learning rate schedule can have its own benefits, therefore we compared re-initialization in settings DC and DCW with SGDR for CIFAR-100. As shown in Table 4.2, SGDR does not match the performance of the re-initialization methods

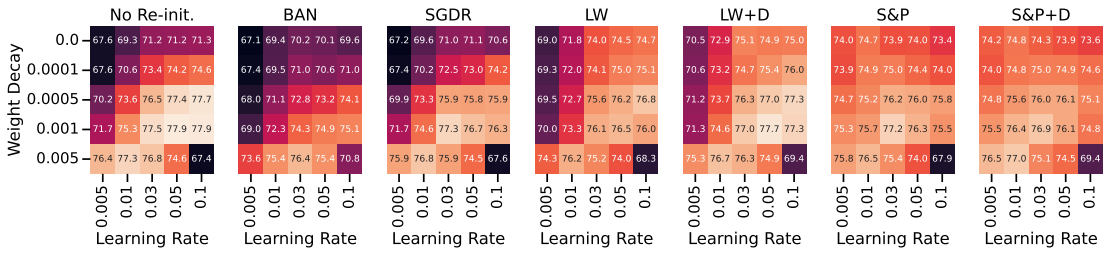


Figure 4.4: Re-initialization can make performance less sensitive to the choice of learning rate and weight decay. Performance of different methods over a grid search of learning rate and weight decay combinations on CIFAR-100 with ResNet-18 in setting DCW. All re-initialization methods use 5 stages. Shrink & Perturb stands out most, as its performance varies much less over the grid than other methods, especially standard training (no re-initialization).

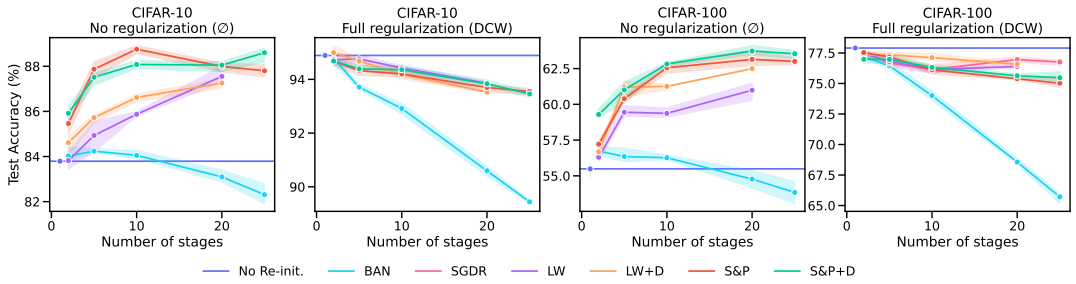


Figure 4.5: Test accuracy as a function of the number of re-initialization stages in settings ∅ and DCW. In setting ∅, *any* number of stages improves upon standard training for Shrink & Perturb and Layer-wise Re-initialization, though the optimal number tends to be 5-20 stages. In setting DCW, adding more stages monotonically worsens generalization. Recall that for Layer-wise Re-initialization, the number of stages is tied to the architecture itself, and $T \in \{2, 5, 10, 20\}$.

in setting DC and performs similar in setting DCW. This indicates that when re-initialization is helpful, it is not an effect of the cyclical schedule.

4.5.4 Re-initialization Makes Optimal Performance More Robust to Hyperparameters

Although re-initialization methods do not offer much benefit in terms of generalization for optimal hyperparameters in setting DCW, we found that they can make performance less sensitive to the choice of learning rate and weight decay hyperparameters – particularly Shrink & Perturb. Figure 4.4 shows the test accuracy achieved by different methods over a grid of learning rate and weight decay choices for CIFAR-100 in setting DCW. We observe that the performance of both Layer-wise Re-initialization and Shrink & Perturb varies less over the grid compared

to no re-initialization. This is especially prominent for Shrink & Perturb. For example, with learning rate 0.005 and weight decay 0, performance of standard training degrades to 67.6%, whereas the performance of Shrink & Perturb remains above 73% throughout. Therefore, re-initialization methods can be beneficial in regimes where thoroughly hyperparameter tuning is infeasible and robustness becomes crucial. This complies with the findings of Alabdulmohsin et al. [2021] who operated under the setup of using a common set of hyperparameters for all datasets and architectures, which may be sub-optimal in certain cases for standard training, and found re-initialization methods outperform standard training.

4.5.5 Impact of the Number of Re-initialization Stages

Recall that the number of stages T used for re-initialization is an additional hyperparameter that we tuned in our experiments. Note that $T = 1$ trivially corresponds to standard training (*i.e.* no re-initialization). We show how test accuracy varies as a function of T in Figure 4.5 for the different methods over each of the CIFAR datasets in settings $\varnothing$ and DCW. In setting $\varnothing$, typically a large number of stages, in the range 5-20, is most beneficial for the re-initialization methods. In fact, for Shrink & Perturb and Layer-wise Re-initialization, *any* number of stages in the range shown (2-25) improves upon standard training. However, in setting DCW, the profile of the curves changes drastically. Increasing the number of stages monotonically lowers the test accuracy, which is consistent with our finding that in this setting, re-initialization methods offer little advantage. See Figure 4.10 in Section 4.8.1 for further experiments with MobileNetV2 on the CIFAR datasets.

4.6 Re-initialization Under Label Noise

Having studied how different components of the training protocol interact with re-initialization, we next focus on learning under noisy data. In particular, we add label noise to the data which makes learning more challenging. We consider the impact on performance when a randomly-chosen fraction q of the training data points have random labels, whereas the test set remains the same. The addition of noise

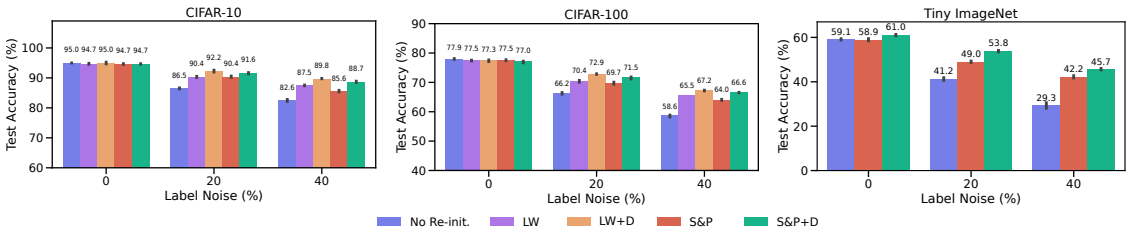


Figure 4.6: Re-initialization is beneficial for learning under label noise. Both re-initialization methods improve performance, with distillation leading to a further improvement. Moreover, the benefit of re-initialization compared to standard training increases with more label noise.

makes learning more difficult as models would require more regularization to prevent them from overfitting to the noise. Therefore, learning under label noise can help distinguish different regularization techniques. We remain in setting DCW, where data augmentation, weight decay and learning rate schedules are used. Moreover, we follow the setup described in Section 4.4.1; on the CIFAR datasets, learning rate and weight decay are tuned separately for each method, while on Tiny ImageNet, we use the hyperparameters obtained from tuning on standard training only.

Figure 4.6 shows the performance of different methods as we add $q = 20\%$ and 40% label noise on the CIFAR and Tiny ImageNet datasets. Surprisingly, re-initialization methods consistently improve upon standard training here. For CIFAR datasets, both Shrink & Perturb and Layer-wise Re-initialization are beneficial each on their own, and performance further improves with the addition of distillation. The results are similar for Tiny ImageNet, where the difference between standard training and re-initialization is even starker: Shrink & Perturb with distillation improves upon standard training by over 15 percentage points in test accuracy when we have 40% label noise.

We can therefore conclude that the effects of re-initialization do not completely overlap with those of data augmentation, weight decay and learning rate schedules. Indeed, even though re-initialization does not improve generalization in setting DCW without label noise, it shows a substantial improvement on generalization with label noise on top of standard regularization techniques. We suspect that one reason re-initialization helps in this scenario relates to the order in which neural networks

learn training examples. "Difficult examples" (such as mislabelled inputs) tend to be learned *later* during training by *deeper* layers [Baldock et al., 2021]. Re-initialization might be exploiting this property of learning by naturally selecting the clean data: since periodically re-initializing "resets" training to an extent, the network might be prevented from confidently fitting to the noise (which is learned later) to instead focus on learning the correctly labelled examples (which are learned earlier). This may also explain why Layer-wise Re-initialization outperforms Shrink & Perturb here due to its layer-wise structure that re-initializes deeper layers more. This also raises the question of whether the performance gap between re-initialization methods and standard training can be closed by tuning the epoch budget to reduce overfitting. As shown in Figure 4.11, this is *not* sufficient to close to gap.

4.7 Conclusion, Limitations & Future Work

We have investigated when re-initialization methods, a set of techniques that are simple to implement with almost no computational overhead, improve generalization compared to standard training. We found that such methods are beneficial for improving performance in the absence of other regularization. However, in a setting resembling SOTA training protocols (*i.e.* including data augmentation, weight decay and learning rate schedules), which is arguably more important and common, re-initialization methods do not improve over standard training, although optimal performance becomes more robust to the choice of hyperparameters. Finally, under label noise, re-initialization does have a significant and helpful effect on learning that other regularization techniques are not able to offer.

In future work, we hope to study the implications of our work on online learning, where Shrink & Perturb was first proposed and shown to be helpful. Does the generalization gap due to warm-starting still appear if models are trained with sufficient regularization? Does Shrink & Perturb continue to be beneficial? One limitation of our work is that, although we observe clear empirical trends in *when* re-initialization works, a deeper understanding of *why* it does or does not work is missing and would be very desirable to gain in future work. Another limitation of our

study is that we restrict ourselves to CIFAR-10/100 and Tiny ImageNet datasets and convolution-based network architectures for classification. Although this restriction allowed us to thoroughly explore the interaction between re-initialization and other techniques on these standard benchmarks and carefully tune hyperparameters given a limited compute budget, it would be interesting to extend the scope of our study to other architectures, tasks and data modalities as future work.

4.8 Supplementary Material

4.8.1 Additional Results

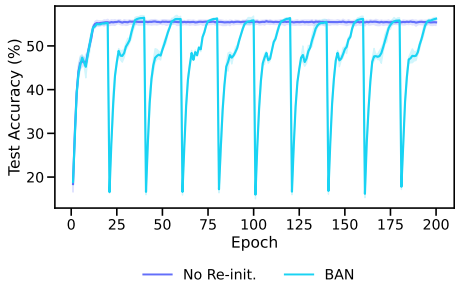


Figure 4.7: Example test accuracy curves on the CIFAR-100 dataset with ResNet-18 in setting ∅. BAN involves 10 stages. For each method and dataset, the learning rate is tuned separately. Notice that each re-initialization causes a large drop in performance in contrast with *e.g.* Shrink & Perturb as shown in Figure 4.2.

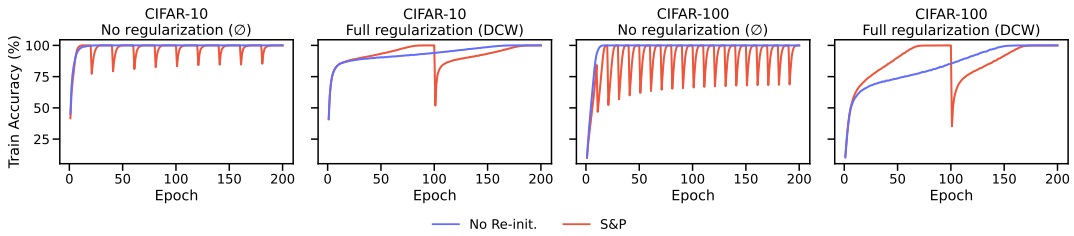


Figure 4.8: Example train accuracy curves for standard training and Shrink & Perturb. In each plot, Shrink & Perturb is shown with its optimal number of stages. Observe that in all cases the models achieve close to 100% accuracy.

4.8.2 Hyperparameters & Pseudo-code of Re-initialization Methods

Hyperparameters for our experiments are described in Tables 4.4 and 4.5, and the pseudo-code for training with re-initialization is shown in Algorithm 1.

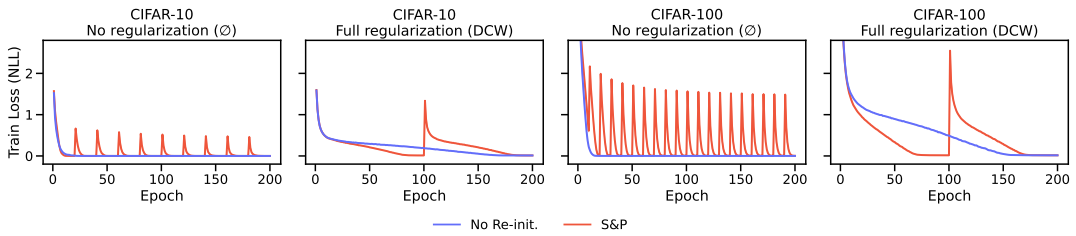


Figure 4.9: Example train negative log-likelihood curves for standard training and Shrink & Perturb. In each plot, Shrink & Perturb is shown with its optimal number of stages. Observe that in all cases the models achieve close to zero loss.

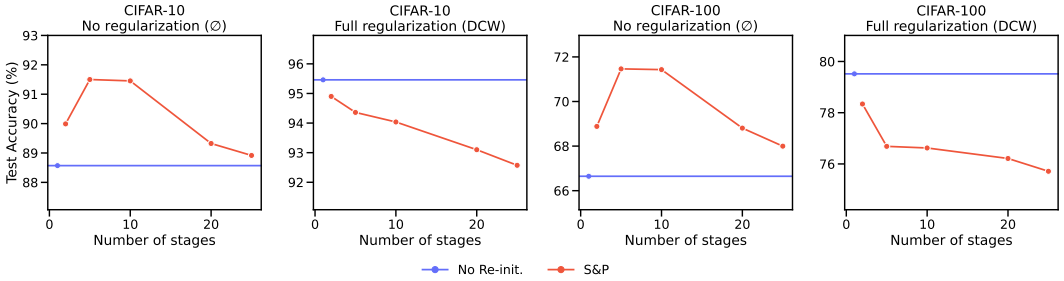


Figure 4.10: Effect of Shrink & Perturb on the CIFAR datasets in settings ∅ and DCW with a MobileNetV2 architecture. Our results in this figure indicate that the overall trend observed in our experiments with ResNet-18 also appears for MobileNetV2. In particular, in setting ∅, Shrink & Perturb improves performance for any number of stages shown, whereas in setting DCW, it provides no benefit over standard training (no re-initialization).

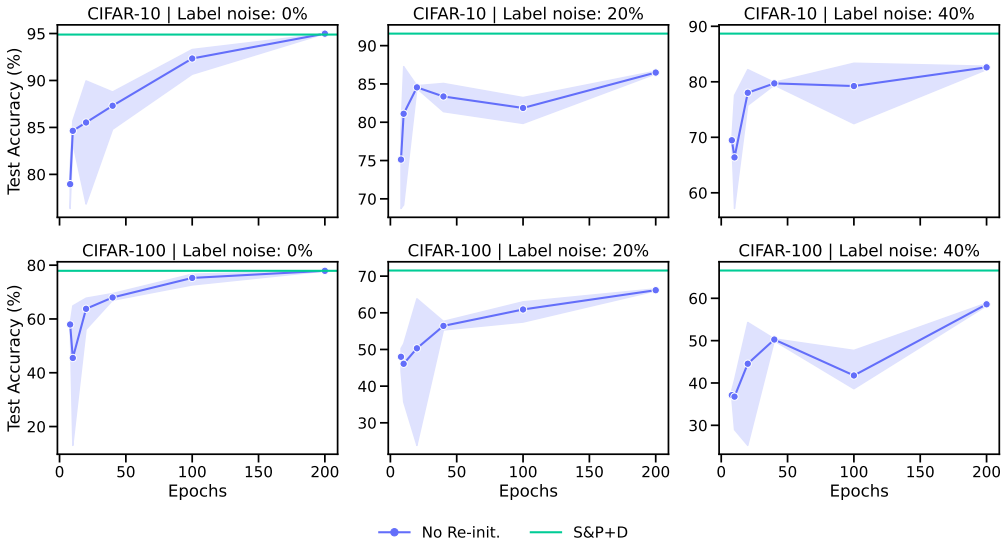


Figure 4.11: Under label noise, tuning the epoch budget for standard training does not close the gap with re-initialization methods. See the discussion in Section 4.6.

Algorithm 1: Training with re-initialization function R.

Input: Training data $\mathcal{D}$, number of re-initialization stages T, initialization distribution p_{init}, total epochs N.

$P \leftarrow \lfloor N/T \rfloor$ ▷ `epochs per stage`
for $t \leftarrow 1$ **to** T **do**
  if $t == 1$ **then**
    $\boldsymbol{\theta}^{\text{start}} \sim p_{\text{init}}$ ▷ `initialize training`
  else
    $\boldsymbol{\theta}^{\text{start}} \leftarrow R(\boldsymbol{\theta}_{\text{init}}, \boldsymbol{\theta}^{\text{end}})$, where $\boldsymbol{\theta}_{\text{init}} \sim p_{\text{init}}$ is an i.i.d. initialization. ▷ `re-initialize network`
  end
  $\boldsymbol{\theta}^{\text{end}} \leftarrow$ parameters after training for P epochs on $\mathcal{D}$, starting from $\boldsymbol{\theta}^{\text{start}}$, optionally with distillation from teacher network $f_{\boldsymbol{\theta}^{\text{end}}}$ if $t > 1$. ▷ `optimization objective is defined in Equation (4.1)`
end
return $\boldsymbol{\theta}^{\text{end}}$

General	
Total training epochs	200
Stages × epochs per stage	$\{2 \times 100, 5 \times 40, 10 \times 20, 20 \times 10, 25 \times 8\}$
Batch size	125
Momentum	0.9
Learning rate grid	$\{0.005, 0.01, 0.03, 0.05, 0.1\}$
Weight decay grid	$\{0, 0.0001, 0.0005, 0.001, 0.005\}$
Distillation strength β_{distill} (if used)	1
Data augmentation (if used)	Random horizontal flips and size-preserving crops after padding 4 pixels.
Data normalization	$\mu = (0.485, 0.456, 0.406)$ $\sigma = (0.229, 0.224, 0.225)$
Shrink & Perturb	
Shrink λ (default unless explicitly defined otherwise)	0.4
Perturb γ (default unless explicitly defined otherwise)	0.1
Layer-wise Re-initialization	
(K, M)	$\{(2,1), (5,1), (5,2)\}$

Table 4.4: Hyperparameters common to all models trained on CIFAR-10 and CIFAR-100 (32×32 images) [Krizhevsky et al., 2009].

General	
Total training epochs	500
Stages × epochs per stage	$\{2 \times 100, 5 \times 40, 10 \times 20, 20 \times 10, 25 \times 8\}$
Batch size	100
Momentum	0.9
Learning rate grid	$\{0.01, 0.05, 0.1\}$
Weight decay grid	$\{0, 0.0001, 0.0005, 0.005\}$
Distillation strength β_{distill} (if used)	2
Data augmentation (if used)	Random horizontal flips and size-preserving crops after padding 4 pixels.
Data normalization	$\mu = (0.485, 0.456, 0.406)$ $\sigma = (0.229, 0.224, 0.225)$
Shrink & Perturb	
Shrink λ (default unless explicitly defined otherwise)	0.4
Perturb γ (default unless explicitly defined otherwise)	0.1

Table 4.5: Hyperparameters common to all models trained on Tiny ImageNet (64×64 images) [Le and Yang, 2015].

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	When Does Re-initialization Work?
Publication Status	☒Published ☐ Accepted for Publication ☐Submitted for Publication ☐Unpublished and unsubmitted work written in a manuscript style
Publication Details	Sheheryar Zaidi*, Tudor Berariu*, Hyunjik Kim, Jorg Bornschein, Claudia Clopath, Yee Whye Teh, Razvan Pascanu. "When Does Re-initialization Work?" *Proceedings on "I Can't Believe It's Not Better! - Understanding Deep Learning Through Empirical Falsification" at NeurIPS 2022 Workshops*, PMLR 187:12-26, 2023.

Student Confirmation

Student Name:	Sheheryar Zaidi		
Contribution to the Paper	☐ Razvan and I initiated the project by beginning to study shrink & perturb for continual learning. ☐ I proposed studying the generalization properties of shrink & perturb for supervised learning after running the initial experiments exploring its benefits. ☐ I proposed exploring shrink & perturb under label noise and ran the initial experiments. ☐ Tudor and I wrote the code the code and designed and ran the experiments in the paper. Both of us produced the plots in the paper. ☐ I wrote the paper with feedback and advice from Tudor, Razvan and Hyunjik. ☐ All authors contributed to the development of the project through discussions and reviewed the final draft.		
Signature		Date	01/02/2023

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Prof Yee-Whye Teh		
Supervisor comments	Looks good to me.		
Signature		Date	31 Oct 2023

This completed form should be included in the thesis, at the end of the relevant chapter.

5

LieTransformer: Equivariant Self-Attention for Lie Groups

Abstract: Group equivariant neural networks are used as building blocks of group invariant neural networks, which have been shown to improve generalisation performance and data efficiency through principled parameter sharing. Such works have mostly focused on group equivariant convolutions, building on the result that group equivariant linear maps are necessarily convolutions. In this work, we extend the scope of the literature to *self-attention*, that is emerging as a prominent building block of deep learning models. We propose the `LieTransformer`, an architecture composed of `LieSelfAttention` layers that are equivariant to arbitrary Lie groups and their discrete subgroups. We demonstrate the generality of our approach by showing experimental results that are competitive to baseline methods on a wide range of tasks: shape counting on point clouds, molecular property regression and modelling particle trajectories under Hamiltonian dynamics.

Contents

5.1 Introduction

Group equivariant neural networks are useful architectures for problems with symmetries that can be described in terms of a group (in the mathematical sense). Convolutional neural networks (CNNs) are a special case that deal with translational symmetry, in that when the input to a convolutional layer is translated, the output is also translated. This property is known as *translation equivariance*, and offers a useful inductive bias for perception tasks which usually have translational symmetry. Constraining a linear layer to obey this symmetry, resulting in a covolutional layer, greatly reduces the number of parameters and computational cost. This has led to the success of CNNs in multiple domains such as computer vision [Krizhevsky et al., 2012] and audio [Graves and Jaitly, 2014]. Following on from this success, there has been a growing literature on the study of group equivariant CNNs (G-CNNs) that generalise CNNs to deal with other types of symmetries beyond translations, such as rotations and reflections.

Most works on group equivariant NNs deal with CNNs i.e. linear maps with shared weights composed with pointwise non-linearities, building on the result that group equivariant linear maps (with mild assumptions) are necessarily convolutions [Kondor and Trivedi, 2018, Cohen et al., 2019, Bekkers, 2020]. However there has been little work on non-linear group equivariant building blocks. In this paper we extend group equivariance to self-attention [Vaswani et al., 2017], a non-trivial non-linear map, that has become a prominent building block of deep learning models in various data modalities, such as natural-language processing [Vaswani et al., 2017, Brown et al., 2020b], computer vision [Zhang et al., 2019, Parmar et al., 2019a], reinforcement learning [Parisotto et al., 2020], and audio generation [Huang et al., 2019].

We thus propose `LieTransformer`, a group invariant Transformer built from group equivariant `LieSelfAttention` layers. It uses a lifting based approach, that relaxes constraints on the attention module compared to approaches without lifting. Our method is applicable to Lie groups and their discrete subgroups (e.g. cyclic

groups C_n and dihedral groups D_n) acting on homogeneous spaces. Our work is very much in the spirit of Finzi et al. [2020], our main baseline, but for group equivariant self-attention instead of convolutions. Among works that deal with equivariant self-attention, we are the first to propose a methodology for general groups and domains (unspecified to 2D images Romero et al. [2020], Romero and Cordonnier [2021] or 3D point clouds Fuchs et al. [2020b]). We demonstrate the generality of our approach through strong performance on a wide variety of tasks, namely shape counting on point clouds, molecular property regression and modelling particle trajectories under Hamiltonian dynamics.

5.2 Background

5.2.1 Group Equivariance

This section lays down some of the necessary definitions and notations in group theory and representation theory in an informal and intuitive manner. For a more formal presentation of definitions, see Section 5.7.2.

Loosely speaking, a **group** G is a set of symmetries, with each group element g corresponding to a symmetry transformation. These group elements ($g, g' \in G$) can be composed (gg') or inverted (g^{-1}), just like transformations. An example of a **discrete group** is C_n, the set of rotational symmetries of a regular n-gon. The group consists of n such rotations, including the identity. An example of a continuous (infinite) group is $SO(2)$, the set of all 2D rotations about the origin. C_n is a subset of $SO(2)$, hence we call C_n a **subgroup** of $SO(2)$. Note that $SO(2) = \{g_\theta : \theta \in [0, 2\pi)\}$ can be parameterised by the angle of rotation θ. Such groups that can be continuously parameterised by real values are called **Lie groups**.

A symmetry transformation of group element $g \in G$ on object $v \in V$ is referred to as the **group action** of G on V. If this action is linear on a vector space V, then we can represent the action as a linear map $\rho(g)$. We call ρ a **representation** of G, and $\rho(g)$ often takes the form of a matrix. For $SO(2)$, the standard rotation

matrix is an example of a representation that acts on $V = \mathbb{R}^2$:

$$\rho(g_\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.1}$$

Note that this is only one of many possible representations of $SO(2)$ acting on $\mathbb{R}^2$ (e.g. replacing θ with $n\theta$ yields another valid representation), and $SO(2)$ can act on spaces other than $\mathbb{R}^2$, e.g. $\mathbb{R}^d$ for arbitrary $d \geq 2$.

In the context of group equivariant neural networks, V is commonly defined to be the space of scalar-valued functions on some set S, so that $V = \{f \mid f : S \to \mathbb{R}\}$. This set could be a Euclidean input space e.g. a grey-scale image can be expressed as a feature map $f : \mathbb{R}^2 \to \mathbb{R}$ from pixel coordinate x_i to pixel intensity $\mathtt{f}_i$, supported on the grid of pixel coordinates. We may express the rotation of the image as a representation of $SO(2)$ by extending the action ρ on the pixel coordinates to a representation π that acts on the space of feature maps:

$$[\pi(g_\theta)(f)](x) \triangleq f(\rho(g_\theta^{-1})x). \tag{5.2}$$

Note that this is equivalent to the mapping $(x_i, \mathtt{f}_i)_{i=1}^n \mapsto (\rho(g_\theta)x_i, \mathtt{f}_i)_{i=1}^n$. As a special case, we can define $V = \{f | f : G \to \mathbb{R}\}$ to be the space of scalar-valued functions on the group G, for which we can define a representation π acting on V via the **regular representation**:

$$[\pi(g_\theta)(f)](g_\phi) \triangleq f(g_\theta^{-1} g_\phi). \tag{5.3}$$

Here the action ρ is replaced by the action of the group on itself. If we wish to handle multiple channels of data, e.g. RGB images, we can stack these feature maps together, transforming in a similar manner.

Now let us define the notion of G-**equivariance**.

Definition 1. *We say that a map $\Phi : V_1 \to V_2$ is G-**equivariant** with respect to actions ρ_1, ρ_2 of G acting on V_1, V_2 respectively if: $\Phi[\rho_1(g)f] = \rho_2(g)\Phi[f]$ for any $g \in G, f \in V_1$.*

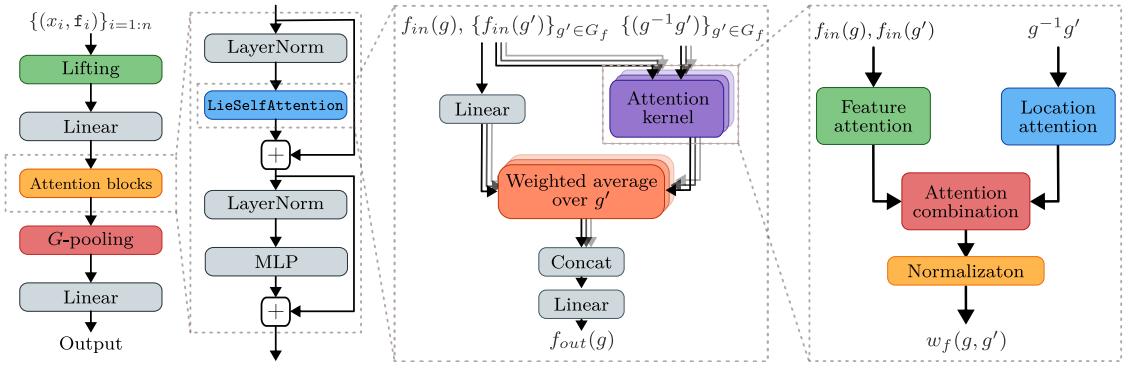


Figure 5.1: Architecture of the LieTransformer.

In the above example of rotating RGB images, we have $G = SO(2)$ and $\rho_1 = \rho_2 = \pi$. Hence the equivariance of Φ with respect to $SO(2)$ means that rotating an input image and then applying Φ yields the same result as first applying Φ to the original input image and then rotating the output, i.e. Φ *commutes* with the representation π.

The end goal for group equivariant neural networks is to design a neural network that obeys certain symmetries in the data. For example, we may want an image classifier to output the same classification when the input image is rotated. So in fact we want a G-**invariant** neural network, where the output is invariant to group actions on the input space. Note that G-invariance is a special case of G-equivariance, where ρ_2 is the **trivial representation** i.e. $\rho_2(g)$ is the identity map for any $g \in G$. Invariant maps are easy to design, by discarding information, e.g. pooling over spatial dimensions is invariant to rotations and translations. However, such maps are not expressive as they fail to extract high-level semantic features from the data. This is where equivariant neural networks become relevant; the standard recipe for constructing an expressive invariant neural network is to compose multiple equivariant layers with a final invariant layer. It is a standard result that such maps are invariant (e.g. Bloem-Reddy and Teh [2020]) and a proof is given in Section 5.7.3 for completeness.

5.2.2 Equivariant Maps on Homogeneous Input Spaces

Here we introduce the framework for G-equivariant maps, and provide group equivariant convolutions as an example. Suppose we have data in the form of a

set of input pairs $(x_i, \mathtt{f}_i)_{i=1}^n$ where $x_i \in \mathcal{X}$ are spatial coordinates and $\mathtt{f}_i \in \mathcal{F}$ are feature values. The data can be described as a single feature map $f_\mathcal{X} : x_i \mapsto \mathtt{f}_i$. We assume that a group G acts on the x-space $\mathcal{X}$ via action ρ, and that the action is **transitive** (also referred to as $\mathcal{X}$ being **homogeneous**). This means that all elements of $\mathcal{X}$ are connected by the action: $\forall x, x' \in \mathcal{X}$, $\exists g \in G : \rho(g)x = x'$. We often write gx instead of $\rho(g)x$ to reduce clutter. For example, the group of 2D translations $T(2)$ acts transitively on $\mathbb{R}^2$ since there is a translation connecting any two points in $\mathbb{R}^2$. On the other hand, the group of 2D rotations about the origin $SO(2)$ does not act transitively on $\mathbb{R}^2$, since points that have different distances to the origin cannot be mapped onto each other by rotations. However the group of 2D roto-translations $SE(2)$, whose elements can be written as a composition tR of $t \in T(2)$ and $R \in SO(2)$, acts transitively on $\mathbb{R}^2$ since $SE(2)$ contains $T(2)$.

For such homogeneous spaces $\mathcal{X}$, it can be shown that there is a natural partition of G into disjoint subsets such that there is a one-to-one correspondence between $\mathcal{X}$ and these subsets. Namely each $x \in \mathcal{X}$ corresponds to the **coset** $s(x)H = \{s(x)h | h \in H\}$, where the subgroup $H = \{g \in G | gx_0 = x_0\}$ is called the **stabiliser** of origin x_0, and $s(x) \in G$ is a group element that maps x_0 to x. It can be shown that the coset $s(x)H$ does not depend on the choice of $s(x)$, and that $s(x)H$ and $s(x')H$ are disjoint for $x \neq x'$. For $T(2)$ acting on $\mathbb{R}^2$, we have $H = \{e\}$, the identity, and $s(x) = t_x$, the group element describing the translation from x_0 to x, and so each x corresponds to $\{t_x\}$. For $SE(2)$ acting on $\mathbb{R}^2$, we have $H = SO(2)$ and $s(x) = t_x$, so each x corresponds to $\{t_x R | R \in SO(2)\}$. This correspondence is often written as an isomorphism $X \simeq G/H$, where G/H is the set of cosets of H.

Using this isomorphism, we can map each point in $\mathcal{X}$ to a set of group elements in G, i.e. mapping each data pair $(x_i, \mathtt{f}_i)$ to (possibly multiple) pairs $\{(g, \mathtt{f}_i) | g \in s(x_i)H\}$. This can be thought of as **lifting** the feature map $f_\mathcal{X} : x_i \mapsto \mathtt{f}_i$ defined on $\mathcal{X}$ to a feature map $\mathcal{L}[f_\mathcal{X}] : g \mapsto \mathtt{f}_i$ defined on G [Kondor and Trivedi, 2018]. Let $\mathcal{I}_U$ denote the space of such feature maps from G to $\mathcal{F}$. Subsequently, we may define group equivariant maps as functions from $\mathcal{I}_U$ to itself, which turns out to be a simpler task than defining equivariant maps directly on $\mathcal{X}$.

The **group equivariant convolution** [Cohen and Welling, 2016, Cohen et al., 2018, Finzi et al., 2020, Romero et al., 2020] is an example of such a group equivariant map that has been studied extensively. Specifically, the group equivariant convolution $\Psi : \mathcal{I}_U \to \mathcal{I}_U$ is defined as:

$$[\Psi f](g) \triangleq \int_G \psi(g'^{-1}g) f(g') dg' \tag{5.4}$$

where $\psi : G \to \mathbb{R}$ is the convolutional filter and the integral is defined with respect to the left Haar measure of G. Note that for discrete groups the integral amounts to a sum over the group. Hence the integral can be computed exactly for discrete groups [Cohen and Welling, 2016], and for Lie groups it can be approximated using Fast Fourier Transforms [Cohen et al., 2018] or Monte Carlo (MC) estimation [Finzi et al., 2020]. Given the regular representation π of G acting on $\mathcal{I}_U$ as in Equation (5.3), we can easily verify that Ψ is equivariant with respect to π (c.f. Section 5.7.3).

5.3 LieTransformer

We first outline the problem setting before describing our model, the `LieTransformer`. We tackle the problem of regression/classification for predicting a scalar/vector-valued target y given a set of input pairs $(x_i, \mathtt{f}_i)_{i=1}^n$ where $x_i \in \mathbb{R}^{d_x}$ are spatial locations and $\mathtt{f}_i \in \mathbb{R}^{d_f}$ are feature values at the spatial location. Hence the training data of size N is a set of tuples $((x_i, \mathtt{f}_i)_{i=1}^{n_j}, y_j)_{j=1}^N$. In some tasks such as point cloud classification, the feature values $\mathtt{f}_i$ may not be given. In this case the $\mathtt{f}_i$ can set to be a fixed constant or a (G-invariant) function of $(x_i)_{i=1}^n$.

`LieTransformer` is composed of a **lifting** layer followed by residual blocks of `LieSelfAttention` layers, `LayerNorm` and pointwise `MLP`s, all of which are equivariant with respect to the regular representation, followed by a final invariant `G-pooling` layer (see Section 5.7.8 for more details on these layers). We summarise the architecture in Figure 5.1 and describe its key components below.

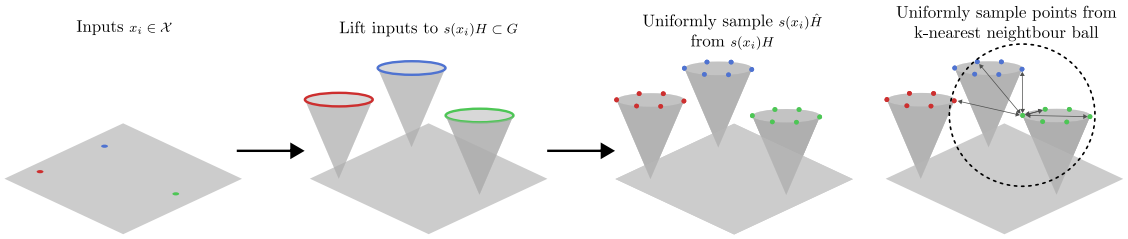


Figure 5.2: Visualisation of lifting, sampling $\hat{H}$, and subsampling in the local neighbourhood for $SE(2)$ acting on $\mathbb{R}^2$. Self-attention is performed on this subsampled neighbourhood.

5.3.1 Lifting

Recall from Section 5.2.2 that the **lifting** $\mathcal{L}$ maps $f_\mathcal{X}$ (supported on $\bigcup_{i=1}^n \{x_i\} \subset \mathcal{X}$) to $\mathcal{L}[f_\mathcal{X}]$ (supported on $\bigcup_{i=1}^n s(x_i)H \subset G$) such that:

$$\mathcal{L}[f_\mathcal{X}](g) \triangleq \mathtt{f}_i \quad \text{for} \quad g \in s(x_i)H. \tag{5.5}$$

This can be thought of as extending the domain of $f_\mathcal{X}$ from $\mathcal{X}$ to G while preserving the feature values $\mathtt{f_i}$, mapping $(x_i, \mathtt{f}_i) \mapsto (g, \mathtt{f}_i)$ for $g \in s(x_i)H$ (c.f. Figure 5.2). Subsequently we may design G-equivariant maps on the space of functions on G, which is a simpler task than desgining G-equivariant maps directly on $\mathcal{X}$ (e.g. Cohen et al. [2018]).

As in Equations (5.2) and (5.3), we define the representation π on $f_\mathcal{X}$ and $\mathcal{L}[f_\mathcal{X}]$ as:

$$[\pi(u)f_\mathcal{X}](x) = f_\mathcal{X}(u^{-1}x)$$
$$[\pi(u)\mathcal{L}[f_\mathcal{X}]](g) = \mathcal{L}[f_\mathcal{X}](u^{-1}g)$$

where $u \in G$. Note that the actions correspond to mappings $(x_i, \mathtt{f}_i) \mapsto (ux_i, \mathtt{f}_i)$ and $(g, \mathtt{f}_i) \mapsto (ug, \mathtt{f}_i)$ respectively.

We need to ensure that lifting preserves equivariance, which is why we need the space to be homogeneous with respect to the action of G on $\mathcal{X}$.

Proposition 2. *The lifting layer $\mathcal{L}$ is equivariant with respect to the representation π.*

Intuition for proof. When x_i is shifted by $u \in G$, the lifted coset $s(x_i)H$ is also shifted by u, i.e. $x_i \mapsto ux_i \Rightarrow s(x_i)H \mapsto us(x_i)H$. See Section 5.7.3 for full proof.

Algorithm 2: LieSelfAttention

Input: $(g, f(g))_{g \in G_f}$

for $g \in G_f$ **do**
  for $g' \in G_f$ (or $\texttt{nbhd}_\eta(g)$) **do**
    $k_c(f(g), f(g')), k_l(g^{-1}g')$ ▷ Compute content/location attention
    $\alpha_f(g, g') = F(k_c(f(g), f(g')), k_l(g^{-1}g'))$ ▷ Compute unnormalised weights
  end
  $\{w_f(g, g')\}_{g' \in G_f} = \texttt{norm}\{\alpha_f(g, g')\}_{g' \in G_f}$ ▷ Compute normalised weights and output
  $f_{\text{out}}(g) = \int_{G_f} w_f(g, g') W^V f(g') dg'$
end
return $(g, f_{\text{out}}(g))_{g \in G_f}$

5.3.2 LieSelfAttention

Let $f \triangleq \mathcal{L}[f_{\mathcal{X}}]$, hence f is defined on the set $G_f = \cup_{i=1}^n s(x_i)H$. We define the `LieSelfAttention` layer in Algorithm 2, where self-attention (see Section 5.7.4 for the original formulation) is defined across the elements of G_f. There are various choices for functions content-based attention k_c , location-based attention k_l , F that determines how to combine the two to form unnormalised weights, and the choice of normalisation of weights. See Section 5.7.5 for a non-exhaustive list of choices of the above, and also for the details of the multi-head generalisation of `LieSelfAttention`.

Proposition 3. *LieSelfAttention is equivariant with respect to the regular representation π.*

Intuition for proof. `LieSelfAttention` can be thought of as a map $\Phi : (g, f(g))_{g \in G_f} \mapsto (g, f_{\text{out}}(g))_{g \in G_f}$, and equivariance holds if $\forall u \in G$, Φ maps $(ug, f(g))_{g \in G_f}$ to $(ug, f_{\text{out}}(g))_{g \in G_f}$. Now note that Φ is a function of $g \in G_f$ only via $g^{-1}g'$ for $g' \in G_f$, and $g^{-1}g'$ is invariant to the group action $g \mapsto ug, g' \mapsto ug'$. This is enough to show that Φ satisfies the above condition for equivariance. See Section 5.7.3 for a full proof.

Generalisation to infinite G_f. For Lie Groups, G_f is usually infinite (it need not be if H is discrete e.g. for $T(n)$ acting on $\mathbb{R}^n$, we have $H = \{e\}$ hence G_f is finite). To deal with this case we resort to Monte Carlo (MC) estimation to approximate the integral in `LieSelfAttention`, following the approach of Finzi et al. [2020]:

1. Replace $G_f \triangleq \cup_{i=1}^{n} s(x_i)H$ with a finite subset $\hat{G}_f \triangleq \cup_{i=1}^{n} s(x_i)\hat{H}$ where $\hat{H}$ is a finite subset of H sampled uniformly. We refer to $|\hat{H}|$ as the number of *lift samples*.

2. (Optional, for computational efficiency) Further replace $\hat{G}_f$ with uniform samples from the neighbourhood $\texttt{nbhd}_\eta(g) \triangleq \{g' \in \hat{G}_f : d(g, g') \leq \eta\}$ for some threshold η where distance is measured by the log map $d(g, g') = ||\nu[\log(g^{-1}g')]||$ (c.f. Section 5.7.6).

See Figure 5.2 for a visualisation. Due to MC estimation we now have equivariance in expectation as Finzi et al. [2020]. For sampling within the neighbourhood, we can show that the resulting `LieSelfAttention` is still equivariant in expectation given that the distance is a function of $g^{-1}g'$ (c.f. Section 5.7.3).

5.4 Related Work

Equivariant maps with/without lifting. Equivariant neural networks can be broadly categorised by whether the input spatial data is *lifted* onto the space of functions on group G or not. Without lifting, the equivariant map is defined between the space of functions/features on the homogeneous input space X, with equivariance imposing a constraint on the parameterisation of the convolutional kernel or attention module [Cohen and Welling, 2017, Worrall et al., 2017, Thomas et al., 2018a, Kondor et al., 2018a, Weiler et al., 2018c,b, Weiler and Cesa, 2019, Esteves et al., 2020, Fuchs et al., 2020b]. In the case of convolutions, the kernel is expressed using a basis of equivariant functions such as circular or spherical harmonics. However with lifting, the equivariant map is defined between the space of functions/features on G, and aforementioned constraints on the convolutional kernel or attention module are relaxed at the cost of an increased dimensionality of the input to the neural network [Cohen and Welling, 2016, Cohen et al., 2018, Esteves et al., 2018, Finzi et al., 2020, Bekkers, 2020, Romero and Hoogendoorn, 2020, Romero et al., 2020, Hoogeboom et al., 2018]. Our method also uses lifting to define equivariant self-attention.

Equivariant self-attention. Most of the above works use equivariant convolutions as the core building block of their equivariant module, drawing from the result that bounded linear operators are group equivariant if and only if they are convolutions [Kondor and Trivedi, 2018, Cohen et al., 2019, Bekkers, 2020]. Such convolutions are used with pointwise non-linearities (applied independently to the features at each spatial location/group element) to form expressive equivariant maps. Exceptions to this are Romero et al. [2020] and Fuchs et al. [2020b] that explore equivariant attentive convolutions, reweighing convolutional kernels with attention weights. This gives non-linear equivariant maps with non-linear interactions across spatial locations/group elements. Instead, our work removes convolutions and investigates the use of equivariant self-attention only, inspired by works that use stand-alone self-attention on images to achieve competitive performance to convolutions [Parmar et al., 2019b, Dosovitskiy et al., 2020a]. Furthermore, Romero et al. [2020] focus on image applications (hence scalability) and discrete groups (p4, p4m), and Fuchs et al. [2020b] focus on 3D point cloud applications and the $SE(3)$ group with irreducible representations acting on functions on $\mathcal{X}$. Instead we use regular representations actingon functions on G, and give a general method for Lie groups acting on homogeneous spaces, with a wide range of applications from dealing with point cloud data to modelling Hamiltonian dynamics of particles. This is very much in the spirit of Finzi et al. [2020], except for self-attention instead of convolutions. In concurrent work, Romero and Cordonnier [2021] describe group equivariant self-attention also using lifting and regular representations. Their analogue of location-based attention are group invariant positional encodings. The main difference between the two works is that Romero and Cordonnier [2021] specify methodology for discrete groups applied to image classification only and it is not clear how to extend their approach to Lie groups. In contrast, our method provides a general formula for (unimodular) Lie groups and their discrete subgroups for the aforementioned applications.

Table 5.1: Mean and standard deviation of test accuracies on the shape counting task at convergence (over 8 random initialisations).

Training data	D_{train}	D_{train}	D_{train}	D^{T2}_{train}	D^{T2}_{train}	D^{SE2}_{train}
Test data	D_{test}	D^{T2}_{test}	D^{SE2}_{test}	D^{T2}_{test}	D^{SE2}_{test}	D^{SE2}_{test}
SetTransformer	0.58 ± 0.07	0.44 ± 0.02	0.44 ± 0.02	0.61 ± 0.02	0.51 ± 0.01	0.55 ± 0.01
LieTransformer-T2	**0.75** ± 0.03	**0.75** ± 0.03	0.63 ± 0.06	**0.75** ± 0.03	0.63 ± 0.06	0.70 ± 0.03
LieTransformer-SE2	0.71 ± 0.01	0.71 ± 0.01	**0.69** ± 0.02	0.71 ± 0.01	**0.69** ± 0.02	**0.72** ± 0.04

5.5 Experiments

We consider three different tasks that have certain symmetries, highlighting the benefits of the `LieTransformer`: (1) Counting shapes in 2D point cloud of constellations (2) Molecular property regression and (3) Modelling particle trajectories under Hamiltonian dynamics.[1]

5.5.1 Counting Shapes in 2D Point Clouds

We first consider the toy, synthetic task of counting shapes in a 2D point cloud $\{x_1, x_2, ..., x_K\}$ of constellations [Kosiorek et al., 2019], mainly to check that `LieTransformer` has the correct invariance properties. We use $\mathtt{f}_i = 1$ for all points. Each example consists of points in the plane that form the vertices of a pattern. There are four types of patterns: triangles, squares, pentagons and the 'L' shape, with varying sizes, orientation, and number of instances per pattern (see Figure 5.3 (right)). The task is to classify the number of instances of each pattern, hence is invariant to 2D roto-translations $SE(2)$.

We first create a fixed training set D_{train} and test set D_{test} of size 10,000 and 1,000 respectively. We then create augmented test sets D^{T2}_{test} and D^{SE2}_{test} that are copies of D_{test} with arbitrary transformations in $T(2)$ and $SE(2)$ respectively. In Table 5.1, we evaluate the test accuracy of `LieTransformer` at convergence with and without data augmentation during training time – D^{T2}_{train} and D^{SE2}_{train} indicate random $T(2)$ and $SE(2)$ augmentations respectively to each batch of D_{train} at every training iteration. We evaluate the test performance of `LieTransformer-T2` and

[1]The code for our experiments is available at: `https://github.com/oxcsml/lie-transformer`

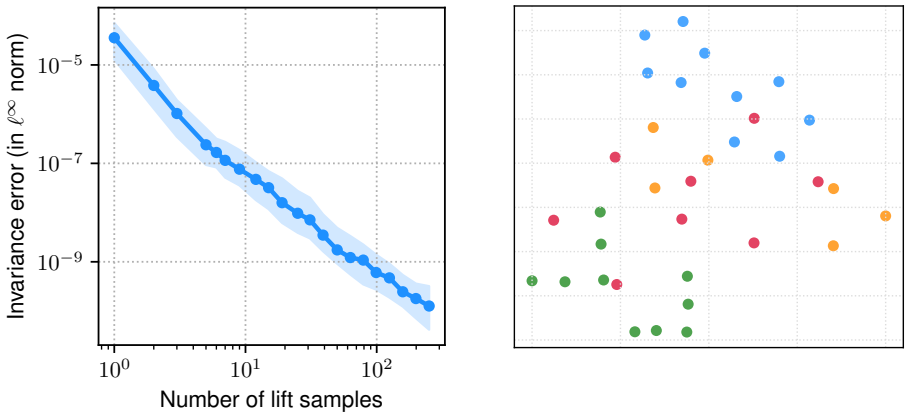


Figure 5.3: (Left) $SE(2)$ invariance error on output logits vs. number of lift samples for a single layer `LieTransformer-SE2`. Plot shows median and interquartile range across 100 runs, randomizing over model seed, input example and transformation applied to input. (Right) An example 2D point cloud from D_{train}. Each colour corresponds to a different pattern.

`LieTransformer-SE2` that are invariant to $T(2)$ and $SE(2)$ respectively, against the baseline `SetTransformer` [Lee et al., 2019], a Transformer-based model that is permutation invariant, but not invariant to rotations nor translations. We use a similar number of parameters for each model. See Section 5.7.9.1 for further details on the setup.

Note that the test accuracy of `LieTransformer-T2` and `LieTransformer-SE2` remains unchanged when the train/test set is augmented with $T(2)$. For `LieTransformer-SE2`, this is not quite true for $SE(2)$ augmentations because the model is only $SE(2)$ equivariant in expectation and not exactly equivariant given a finite number of lifts samples ($|\hat{H}|$). However the changes in accuracy for $SE(2)$ augmentation are much smaller compared to `LieTransformer-T2`. The test accuracy of `SetTransformer`, the non-invariant baseline, is always lower than `LieTransformer`. Note that `LieTransformer-T2` does slightly better than `LieTransformer-SE2` on D_{test} and D^{T2}_{test}. We suspect that the variance in the sampling of the lifting layer for `LieTransformer-SE2` is making optimisation more difficult, and will continue to explore these results.

In Figure 5.3 (left), we report the equivariance error of `LieTransformer-SE2` when increasing the number of lift samples ($|\hat{H}|$) used in the Monte Carlo ap-

Table 5.2: QM9 molecular property prediction mean absolute error. Bold indicates best performance in a given section, underlined indicates best overall performance. *These results are from our own runs of the Dimenet++ model. The original paper used different train/valid/test splits to the other papers listed here. †These results are from our owns runs of LieConv as these ablations were not present in the original paper.

Task	α	$\Delta\epsilon$	ϵ_{HOMO}	ϵ_{LUMO}	μ	C_ν	G	H	R^2	U	U_0	ZPVE
Units	bohr^3	meV	meV	meV	D	cal/mol K	meV	meV	bohr^2	meV	meV	meV
WaveScatt [Hirn et al., 2017]	.160	118	85	76	.340	.049	–	–	–	–	–	–
NMP [Gilmer et al., 2017]	**.092**	69	43	38	**.030**	.040	19	17	.180	20	20	**1.50**
SchNet [Schütt et al., 2017b]	.235	**63**	**41**	**34**	.033	**.033**	**14**	**14**	**.073**	**19**	**14**	1.70
Cormorant [Anderson et al., 2019]	.085	61	34	38	.038	.026	20	21	.961	21	22	2.03
DimeNet++ [Klicpera et al., 2020b] *	**.049**	**34**	**26**	**20**	**.033**	**.024**	**8**	**7**	.387	**7**	**7**	**1.23**
L1Net [Miller et al., 2020]	.088	68	45	35	.043	.031	14	14	**.354**	14	13	1.56
TFN [Thomas et al., 2018a]	.223	58	40	38	.064	.101	–	–	–	–	–	–
SE3-Transformer [Fuchs et al., 2020b]	.148	53	36	33	.053	.057	–	–	–	–	–	–
LieConv-T3 [Finzi et al., 2020] †	.125	60	36	32	.057	.046	35	37	1.54	36	35	3.62
LieConv-T3 + SO3 Aug [Finzi et al., 2020]	.084	49	30	**25**	**.032**	.038	22	24	.800	19	19	2.28
LieConv-SE3 [Finzi et al., 2020]†	.097	**45**	**27**	**25**	.039	.041	39	46	2.18	49	48	3.27
LieConv-SE3 + SO3 Aug [Finzi et al., 2020]†	.088	**45**	**27**	**25**	.038	.043	47	46	2.12	44	45	3.25
LieTransformer-T3 (Us)	.179	67	47	37	.063	.046	27	29	.717	27	28	2.75
LieTransformer-T3 + SO3 Aug (Us)	**.082**	51	33	27	.041	**.035**	**19**	**17**	**.448**	**16**	**17**	**2.10**
LieTransformer-SE3 (Us)	.104	52	33	29	.061	.041	23	27	2.29	26	26	3.55
LieTransformer-SE3 + SO3 Aug (Us)	.105	52	33	29	.062	.041	22	25	2.31	24	25	3.67

proximation of `LieSelfAttention`. As expected the invariance error decreases monotonically with the number of lift samples, and already with 3 lift samples, the error is small ($\approx 10^{-6}$).

5.5.2 QM9: Molecular Property Regression

We apply the `LieTransformer` to the QM9 molecule property prediction task [Ruddigkeit et al., 2012, Ramakrishnan et al., 2014b]. This dataset consists of 133,885 small inorganic molecules described by the location and charge of each atom in the molecule, along with the bonding structure of the molecule. The dataset includes 19 properties of each molecule, such as various rotational constants, energies and enthalpies, and 12 of these are used as regression tasks. We expect these molecular properties to be invariant to 3D roto-translations $SE(3)$. We follow the customary practice of performing hyperparameter search on the ϵ_{HOMO} task and use the same hyperparameters for training on the other 11 tasks. Further details of the exact experimental setup can be found in Section 5.7.9.2.

We trained four variants of both `LieTransformer` and `LieConv`, namely the $T(3)$ and $SE(3)$ invariant models with and without $SO(3)$ (rotation) data augmentation. We set x_i to be the atomic position and $\mathtt{f}_i$ to be the charge. Table 5.2 shows the test error of all models and baselines on the 12 tasks. The table is divided into 3 sections. **Upper**: non-invariant models specifically designed for the QM9 task. **Middle**: invariant models specifically designed for the QM9 task. **Lower**: invariant models that are general-purpose. We show very competitive results, and perform best of general-purpose models on 8/12 tasks. In particular when comparing against `LieConv`, we see better performance on the majority of tasks, suggesting that the attention framework is better suited to these tasks than convolutions.

As expected for both `LieTransformer` and `LieConv`, the $SE(3)$ models tend to outperform the $T(3)$ models without $SO(3)$ data augmentation (on 10/12 tasks and 7/12 tasks respectively), showing that being invariant to rotations improves generalisation. Moreover the $SE(3)$ models perform similarly with and without augmentation, whereas the $T(3)$ models greatly benefit from augmentation, showing evidence that the $SE(3)$ models are indeed invariant to rotations. However the $T(3)$ models with augmentation outperform the $SE(3)$ counterparts on most tasks for both `LieTransformer-SE3` and `LieConv-SE3`. As for the experiments in Section 5.5.1, we suspect that the variance in the sampling of the lifting layer of $SE(3)$ models, along with the $SE(3)$ log-map (Section 5.7.6) in the location attention is making optimisation more difficult, and plan to continue investigating the source of this discrepancy in performance. Note however that `LieTransformer-SE3` and `LieConv-SE3` tend to outperform the irreducible representation (irrep) based $SE(3)$-Transformer and TFN. This can be seen as further evidence that regular representation approaches tend to outperform irrep approaches, in line with the empirical observations of Weiler and Cesa [2019].

5.5.3 Modelling Particle Trajectories with Hamiltonian Dynamics

We also apply the `LieTransformer` to a physics simulation task in the context of Hamiltonian dynamics, a formalism for describing the evolution of a physical system using a single scalar function $H(q, p)$, called the Hamiltonian.

We consider the case of n particles in d dimensions, writing the position $\mathbf{q} \in \mathbb{R}^{nd}$ and momentum $\mathbf{p} \in \mathbb{R}^{nd}$ of all particles as a single state $\mathbf{z} = (\mathbf{q}, \mathbf{p})$. The Hamiltonian $H : \mathbb{R}^{2nd} \to \mathbb{R}$ takes as input $\mathbf{z}$ and returns its total (potential plus kinetic) energy. The time evolution of the particles is then given by *Hamilton's equations*:

$$\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}}, \quad \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = -\frac{\partial H}{\partial \mathbf{q}}. \tag{5.6}$$

Several recent works have shown that modelling physical systems by learning its Hamiltonian significantly outperforms approaches that learn the dynamics directly [Greydanus et al., 2019, Sanchez-Gonzalez et al., 2019, Zhong et al., 2020, Finzi et al., 2020]. Specifically, we can parameterise the Hamiltonian of a system by a neural network H_θ that is learned by ensuring that trajectories from the ground truth and learned system are close to each other. Given a learned H_θ, we can simulate the system for T timesteps by solving Equation (5.6) with a numerical ODE solver to obtain a trajectory $\{\hat{\mathbf{z}}_t(\theta)\}_{t=1}^T$ and minimize the ℓ^2-norm between this trajectory and the ground truth $\{\mathbf{z}_t\}_{t=1}^T$.

However we know a-priori that such physical systems have symmetries, namely conserved quantities such as linear and angular momentum. A notable result is Noether's theorem [Noether, 1918], which states that the system has a conserved quantity if and only if the Hamiltonian is group-invariant. For example, translation invariance of the Hamiltonian implies conservation of momentum and rotation invariance implies conservation of angular momentum. Hence in our experiments, we parameterise the Hamiltonian H_θ by a `LieTransformer` and endow it with the symmetries corresponding to the conservation laws of the physical system we are modelling. We test our model on the spring dynamics task proposed in Sanchez-Gonzalez et al. [2019] – we consider a system of 6 particles with randomly

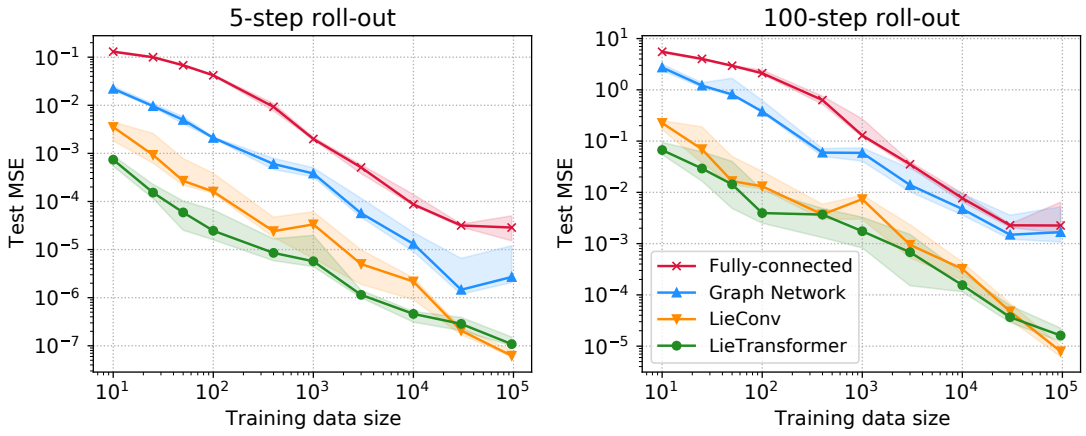


Figure 5.4: Data efficiency on Hamiltonian spring dynamics. All models are trained using 5-step roll-outs, with test performance on 5-step (left) and 100-step (right) roll-outs. Plots show median MSE and interquartile range (IQR) across 10 random seeds.

sampled massses in 2D, where each particle connected to all others by springs. This system conserves both linear and angular momentum, so the ground truth Hamiltonian will be both translationally and rotationally invariant, that is, $SE(2)$-invariant. We simulate this system for 500 timesteps from random initial conditions and use random subsets of length 5 from these roll-outs to train the model (see Section 5.7.9.3 for full experimental details).

We compare our method to different parameterisations of H_θ, namely Fully-connected network [Chen et al., 2018], Graph Network [Sanchez-Gonzalez et al., 2019] and `LieConv`. Only `LieTransformer` and `LieConv` incorporate invariance. In Figures 5.4 to 5.6, we use `LieTransformer-T2` and `LieConv-T2` since Finzi et al. [2020] report that there are numerical instabilities for `LieConv-SE2` on this task, due to which `LieConv-T2` is their default model and performs the best. However in Figure 5.7, we also consider $SE(2)$-invariant versions of both models with modifications to the lifting procedure, which fixed the instabilities as outlined in Section 5.7.6.

Figure 5.4 compares the performance of all methods as a function of the number of training examples. `LieTransformer` is highly data-efficient: the inductive bias from the symmetries of the Hamiltonian allow us to accurately learn the dynamics even from a small training set. Our method consistently outperforms non-invariant

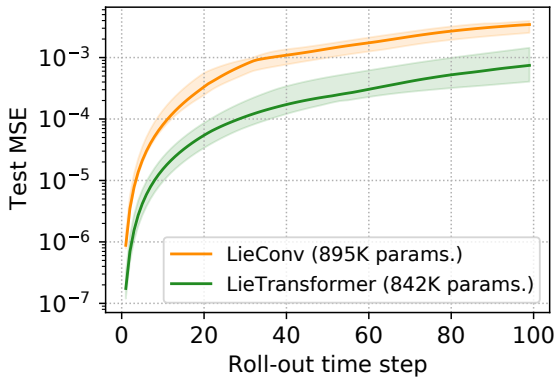


Figure 5.5: Test error vs. roll-out time step. Plots show median MSE and IQR across 10 random seeds.

methods (fully-connected and graph networks), typically by 1-3 orders of magnitude. Furthermore, our method outperforms `LieConv` for most data sizes except the largest sizes where the errors are similar, suggesting that the attention framework more suited for this task.

Figure 5.5 shows the test error as function of the roll-out time step for a training data size of 10,000 (corresponding plots for other training data sizes are included in Section 5.7.10.1). Here we show that the `LieTransformer` shows better generalisation than `LieConv` across all roll-out lengths, the error being low ($< 10^{-3}$) for 100 step-roll-outs even though we only train on 5-step roll-outs. We also include example trajectories of our model in Figure 5.6 (more examples can be found in the appendix, including ones where `LieConv` performs better than `LieTransformer`) illustrating the accuracy of our model on this task.

Lastly, Figure 5.7 compares `LieConv` and `LieTransformer` for different model sizes (number of parameters) and equivariance groups. We first note `LieTransformer` outperforms `LieConv` given a fixed model size and group. For $T(2)$-invariant models, our method benefits from a larger model, whereas `LieConv` deteriorates (`LieConv-T(2) (895K)` is their default architecture on this task). However, for both methods, the $SE(2)$-invariant models perform at par with or better than their $T(2)$-invariant counterparts despite having smaller model sizes. In particular, `LieTransformer-SE(2) (139K)` outperforms all other models in this comparison despite having the smallest number of parameters, which highlights the advantage

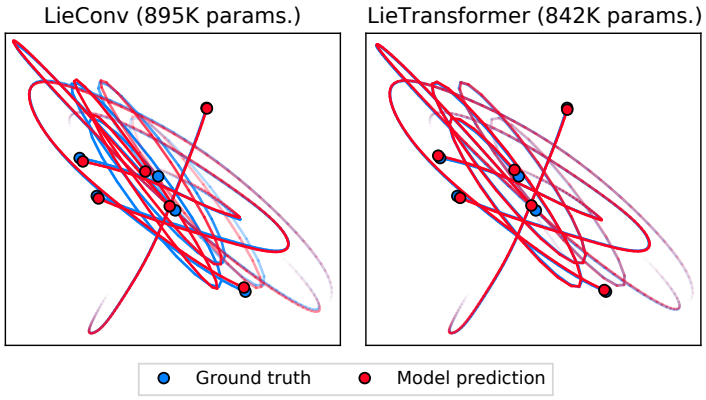


Figure 5.6: Example trajectory predictions on the spring dynamics task. LieTransformer closely follows the ground truth while LieConv diverges from the ground truth at later timesteps.

of incorporating the correct task symmetries into the architecture and the attention framework. Overall, we have shown that our model is suitable for use in a neural ODE setting that requires equivariant drift functions.

5.6 Limitations and Future Work

From the algorithmic perspective, `LieTransformer` shares the weakness of `LieConv` in being memory-expensive ($O(|\hat{G}_f||\texttt{nbhd}_\eta|)$ memory cost (Section 5.7.7) due to: 1. The lifting procedure that increases the number of inputs by $|\hat{H}|$, and 2. Quadratic complexity in the number of inputs from having to compute the kernel value at each pair of inputs. Although the first is a weakness shared by all lifting-based equivariant neural networks, the second can be addressed by incorporating works that study efficient variants of self-attention [Wang et al., 2020, Kitaev et al., 2020, Zaheer et al., 2020, Katharopoulos et al., 2020]. An alternative is to incorporate information about pairs of inputs (such as bonding information for the QM9 task) as masking in self-attention (c.f. Section 5.7.9.2).

From the methodological perspective, a key weakness of the `LieTransformer` that is also shared with `LieConv` is its approximate equivariance due to MC estimation of the integral in `LieSelfAttention` for the case where H is infinite. The aforementioned directions for memory-efficiency can help to reduce the ap-

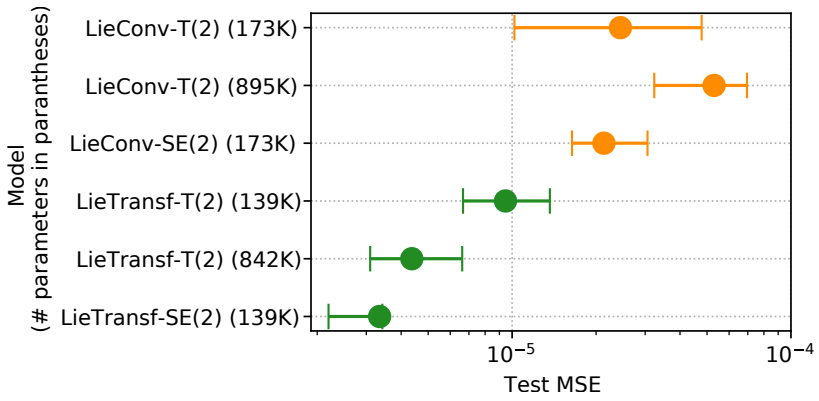


Figure 5.7: Median test MSE and IQR on 5-step trajectories, across 5 random seeds. Results for 100-step trajectories in Figure 5.8.

proximation error by allowing to use more lift samples ($|\hat{H}|$). Other directions include incorporating the notion of *steerability* [Cohen and Welling, 2017] to deal with vector fields in an equivariant manner (given inputs $(x_i, \mathtt{f}_i)$, the group acts non-trivially on $\mathtt{f}_i$ as well as x_i), and extending to non-homogeneous input spaces as outlined in Finzi et al. [2020].

5.7 Supplementary Material

5.7.1 Contributions

- Charline and Yee Whye conceived the project and Yee Whye initially came up with an equivariant form of self-attention.
- Through discussions between Michael, Charline, Hyunjik and Yee Whye, this was modified to the current `LieSelfAttention` layer, and Michael derived the equivariance of the `LieSelfAttention` layer.
- Michael, Sheheryar and Hyunjik simplified the proof of equivariance and further developed the methodology for the `LieTransformer` in its current state, and created links between `LieTransformer` and other related work.
- Michael wrote the initial framework of the `LieTransformer` codebase. Charline and Sheheryar wrote the code for the shape counting experiments, Michael

wrote the code for the QM9 experiments, Sheheryar wrote the code for the Hamiltonian dynamics experiments, after helpful discussions with Emilien.

- Charline carried out the experiments for Table 1, Michael carried out most of the experiments for Table 2 with some help from Hyunjik, Sheheryar carried out the experiments for Figure 3b and all the Hamiltonian dynamics experiments.

- Hyunjik wrote all sections of the paper except the experiment sections: the shape counting section was written by Charline, the QM9 section by Michael and the Hamiltonian dynamics section was written by Emilien and Sheheryar.

5.7.2 Formal definitions for Groups and Representation Theory

Definition 2. *A **group** G is a set endowed with a single operator $\cdot : G \times G \mapsto G$ such that*

1. *Associativity:* $\forall g, g', g'' \in G, (g \cdot g') \cdot g'' = g \cdot (g' \cdot g'')$
2. *Identity:* $\exists e \in G$, $\forall g \in G$ $g \cdot e = e \cdot g = g$
3. *Invertibility:* $\forall g \in G, \exists g^{-1} \in G, g \cdot g^{-1} = g^{-1} \cdot g = e$

Definition 3. *A **Lie group** is a finite-dimensional real smooth manifold, in which group multiplication and inversion are both smooth maps.*

The general linear group $GL(n, \mathbb{R})$ of invertible $n \times n$ matrices is an example of a Lie group.

Definition 4. *Let S be a set, and let* $\mathrm{Sym}(S)$ *denote the set of invertible functions from S to itself. We say that a group G **acts** on S via an action $\rho : G \to \mathrm{Sym}(S)$ when ρ is a group **homomorphism**: $\rho(g_1 g_2)(s) = (\rho(g_1) \circ \rho(g_2))(s)$ $\forall s \in S$.*

If S is a vector space V and this action is, in addition, a linear *function, i.e. $\rho : G \to GL(V)$, where $GL(V)$ is the set of linear invertible functions from V to itself, then we say that ρ is a **representation** of G.*

5.7.3 Proofs

Lemma 1. *The function composition $f \circ f_K \circ ... \circ f_1$ of several equivariant functions f_k, $k \in \{1, 2, ..., K\}$ followed by an invariant function f, is an invariant function.*

Proof. Consider group representations $\pi_1, \dots, \pi_K$ that act on $f_1, \dots f_K$ respectively, and representation π_0 that acts on the input space of f_1. If each f_k is equivariant with respect to π_k, π_{k-1} such that $f_k \circ \pi_{k-1} = \pi_k \circ f_k$, and f is invariant such that $f \circ \pi_k = f$, then we have

$$\begin{aligned} f \circ f_k \circ ... \circ f_1 \circ \pi_0 &= f \circ f_k \circ ... \circ \pi_1 \circ f_1 \\ &\vdots \\ &= f \circ \pi_k \circ f_k \circ ... \circ f_1 \\ &= f \circ f_k \circ ... \circ f_1, \end{aligned}$$

hence $f \circ f_K \circ ... \circ f_1$ is invariant. □

Lemma 2. *The group equivariant convolution $\Psi : \mathcal{I}_U \to \mathcal{I}_U$ defined as: $[\Psi f](g) \triangleq \int_G \psi(g'^{-1}g) f(g') dg'$ is equivariant with resepct to the regular representation π of G acting on $\mathcal{I}_U$ as $[\pi(u)f](g) \triangleq f(u^{-1}g)$.*

Proof.

$$\begin{aligned} \Psi[\pi(u)f](g) &= \int_G \psi(g'^{-1}g)[\pi(u)f](g')dg' \\ &= \int_{uG} \psi(g'^{-1}g) f(u^{-1}g')dg' \\ &= \int_G \psi(g'^{-1}u^{-1}g) f(g')dg' \\ &= [\Psi f](u^{-1}g \\ &= [\pi(u)[\Psi f]](g). \end{aligned}$$

The second equality holds by invariance of the left Haar measure. □

Proposition 2. *The lifting layer $\mathcal{L}$ is equivariant with respect to the representation π.*

Proof. Note $\mathcal{L}[\pi(u)f_{\mathcal{X}}](g) = \mathtt{f}_i$ for $g \in s(ux_i)H$ and $[\pi(u)\mathcal{L}[f_{\mathcal{X}}]](g) = \mathcal{L}[f_{\mathcal{X}}](u^{-1}g) = \mathtt{f}_i$ for $g \in us(x_i)H$. Hence $\mathcal{L}[\pi(u)f_{\mathcal{X}}] = \pi(u)\mathcal{L}[f_{\mathcal{X}}]$ because the two cosets are equal: $s(ux_i)H = us(x_i)H \ \forall u \in G$. Note that this holds because:

- If $g \in s(x_i)H = \{g \in G | gx_0 = x_i\}$, then g maps x_0 to x_i, hence ug maps x_0 to ux_i. So if $g \in s(x_i)H$ then $ug \in s(ux_i)H = \{g \in G | gx_0 = ux_i\}$, the set of all g that map x_0 to ux_i. In summary, $us(x_i)H \subset s(ux_i)H$
- Conversely, if $g \in s(ux_i)H$, then we know that $u^{-1}g$ maps x_0 to x_i, so $u^{-1}g \in s(x_i)H$, hence $g \in us(x_i)H$. In summary, $s(ux_i)H \subset us(x_i)H$
- We have shown $us(x_i)H \subset s(ux_i)H$ and $s(ux_i)H \subset us(x_i)H$, thus $s(ux_i)H = us(x_i)H$.

□

Proposition 3. *LieSelfAttention is equivariant with respect to the regular representation π.*

Proof. Let $\mathcal{I}_U = \mathcal{L}(G, \mathbb{R}^D)$ be the space of unconstrained functions $f : G \to \mathbb{R}^D$. We can define the regular representation π of G acting on $\mathcal{I}_U$ as follows:

$$[\pi(u)f](g) = f(u^{-1}g) \tag{5.7}$$

f is defined on the set $G_f = \cup_{i=1}^{n} s(x_i)H$ (i.e. union of cosets corresponding to each x_i). Note $G_{\pi(u)f} = uG_f$, and G_f does not depend on the choice of section s.

Note that for all provided choices of k_c and k_l, we have:

$$k_c([\pi(u)f](g), [\pi(u)f](g')) = k_c(f(u^{-1}g), f(u^{-1}g')) \tag{5.8}$$

$$k_l(g^{-1}g') = k_l((u^{-1}g)^{-1}(u^{-1}g')) \tag{5.9}$$

Hence for all choices of F, we have that

$$\begin{aligned}\alpha_{\pi(u)f}(g, g') &= F(k_c([\pi(u)f](g), [\pi(u)f](g')), k_l(g^{-1}g')) \\ &= F(k_c(f(u^{-1}g), f(u^{-1}g')), k_l((u^{-1}g)^{-1}u^{-1}g')) \\ &= \alpha_f(u^{-1}g, u^{-1}g')\end{aligned} \tag{5.10}$$

We thus prove equivariance for the below choice of `LieSelfAttention` $\Phi : \mathcal{I}_U \to \mathcal{I}_U$ that uses softmax normalisation, but a similar proof holds for constant normalisation. Let $A_f(g, g') \triangleq \exp(\alpha_f(g, g'))$, hence Equation (5.10) also holds for A_f.

$$[\Phi f](g) = \int_{G_f} w_f(g, g') f(g') dg' \tag{5.11}$$

$$= \int_{G_f} \frac{A_f(g, g')}{\int_{G_f} A_f(g, g'') dg''} f(g') dg' \tag{5.12}$$

Hence:

$$\begin{aligned} w_{\pi(u)f}(g, g') &= \frac{A_{\pi(u)f}(g, g')}{\int_{G_{\pi(u)f}} A_{\pi(u)f}(g, g'') dg''} \\ &= \frac{A_f(u^{-1}g, u^{-1}g')}{\int_{uG_f} A_f(u^{-1}g, u^{-1}g'') dg''} \\ &= \frac{A_f(u^{-1}g, u^{-1}g')}{\int_{G_f} A_f(u^{-1}g, g'') dg''} \\ &= w_f(u^{-1}g, u^{-1}g') \end{aligned} \tag{5.13}$$

Then we can show that Φ is equivariant with respect to the representation π as follows:

$$\begin{aligned} \Phi[\pi(u)f](g) &= \int_{G_{\pi(u)f}} w_{\pi(u)f}(g, g')[\pi(u)f](g') dg' \\ &= \int_{uG_f} w_f(u^{-1}g, u^{-1}g') f(u^{-1}g') dg' \\ &= \int_{G_f} w_f(u^{-1}g, g') f(g') dg' \\ &= [\Phi f](u^{-1}g) \\ &= [\pi(u)[\Phi f]](g) \end{aligned} \tag{5.14}$$

□

Equivariance holds for any α_f that satisfies Equation (5.10). Multiplying α_f by an indicator function $\mathbb{1}\{d(g, g') < \lambda\}$ where $d(g, g')$ is some function of $g^{-1}g'$, we can show that *local* self-attention that restricts attention to points in a neighbourhood also satisfies equivariance. When approximating the integral with Monte Carlo samples (equivalent to replacing G_f with $\hat{G}_f$) we obtain a self-attention layer that is equivariant in expectation for constant normalisation of attention weights (i.e.

$\mathbb{E}[\hat{\Phi}[\pi(u)f](g)] = \Phi[\pi(u)f](g) = [\pi(u)[\Phi f]](g)$ where $\hat{\Phi}$ is the same as Φ but with $\hat{G}_f$ instead of G_f). However for softmax normalisation we obtain a biased estimate due to the nested MC estimate in the denominator's normalising constant.

5.7.4 Introduction to Self-Attention

Self-attention [Vaswani et al., 2017] is a mapping from an input set of N vectors $\{x_1, \ldots, x_N\}$, where $x_i \in \mathbb{R}^D$, to an output set of N vectors in $\mathbb{R}^D$. Let us represent the inputs as a matrix $X \in \mathbb{R}^{N \times D}$ such that the ith row $X_{i:}$ is x_i. **Multihead self-attention** (MSA) consists of M **heads** where M is chosen to divide D. The output of each head is a set of N vectors of dimension D/M, where each vector is obtained by taking a weighted average of the input vectors $\{x_1, \ldots, x_N\}$ with weights given by a weight matrix W, followed by a linear map $W^V \in \mathbb{R}^{D \times D/M}$. Using m to index the head ($m = 1, \ldots, M$), the output of the mth head can be written as:

$$f^m(X) \triangleq WXW^{V,m} \in \mathbb{R}^{N \times D/M}$$
$$\text{where} \quad W \triangleq \texttt{softmax}(XW^{Q,m}(XW^{K,m})^\top) \in \mathbb{R}^{N \times N}$$

where $W^{Q,m}, W^{K,m}, W^{V,m} \in \mathbb{R}^{D \times D/M}$ are learnable parameters, and the softmax normalisation is performed on each row of the matrix $XW^{Q,m}(XW^{K,m})^\top \in \mathbb{R}^{N \times N}$. Finally, the outputs of all heads are concatenated into a $N \times D$ matrix and then right multiplied by $W^O \in \mathbb{R}^{D \times D}$. Hence MSA is defined by:

$$MSA(X) \triangleq [f^1(X), \ldots, f^M(X)]W^O \in \mathbb{R}^{N \times D}. \tag{5.15}$$

Note $XW^Q(XW^K)^\top$ is the Gram matrix for the dot-product kernel, and softmax normalisation is a particular choice of normalisation. Hence MSA can be generalised to other choices of kernels and normalisation that are equally valid [Wang et al., 2018, Tsai et al., 2019].

5.7.5 LieSelfAttention: Details

We explore the following non-exhaustive list of choices for content-based attention, location-based attention, combining content and location attention and normalisation of weights:

Content-based attention $k_c(f(g), f(g'))$:

1. Dot-product: $\frac{1}{\sqrt{d_v}} \left(W^Q f(g)\right)^\top W^K f(g') \in \mathbb{R}$
   for $W^Q, W^K \in \mathbb{R}^{d_v \times d_v}$
2. Concat: $\texttt{Concat}[W^Q f(g), W^K f(g')] \in \mathbb{R}^{2d_v}$
3. Linear-Concat-linear: $W\texttt{Concat}[W^Q f(g), W^K f(g')] \in \mathbb{R}^{d_s}$
   for $W \in \mathbb{R}^{d_s \times 2d_v}$.

Location-based attention $k_l(g^{-1}g')$ for Lie groups G:

1. Plain: $\nu[\log(g^{-1}g')]$
2. MLP: $\texttt{MLP}(\nu[\log(g^{-1}g')])$

where $\log : G \to \mathfrak{g}$ is the log map from G to its Lie algebra $\mathfrak{g}$, and $\nu : \mathfrak{g} \to \mathbb{R}^d$ is the isomorphism that extracts the free parameters from the output of the log map [Finzi et al., 2020]. We can use the same log map for discrete subgroups of Lie groups (e.g. $C_n \leq SO(2), D_n \leq O(2)$). See Section 5.7.6 for an introduction to the Lie algebra and the exact form of $\nu \circ \log(g)$ for common Lie groups.

Combining content and location attention $\alpha_f(g, g')$:

1. Additive: $k_c(f(g), f(g')) + k_l(g^{-1}g')$
2. MLP: $\texttt{MLP}[\texttt{Concat}[k_c(f(g), f(g')), k_l(g^{-1}g')]]$
3. Multiplicative: $k_c(f(g), f(g')) \cdot k_l(g^{-1}g')$

Note that the MLP case is a strict generalisation of the additive combination, and for this option k_c and k_l need not be scalars.

Normalisation of weights $\{w_f(g, g')\}_{g' \in G_f}$:

1. Softmax: $\text{softmax}\left(\{\alpha_f(g, g')\}_{g' \in G_f}\right)$
2. Constant: $\{\frac{1}{|G_f|}\alpha_f(g, g')\}_{g' \in G_f}$

We also outline how to extend the single-head `LieSelfAttention` described in Algorithm 2 extends to **Multihead equivariant self-attention**. Let M be the number of heads, assuming it divides d_v, with m indexing the head. Then the output of each head is:

$$V^m(g) = \int_{G_f} w_f(g, g') W^{V,m} f(g') dg' \in \mathbb{R}^{d_v/M} \tag{5.16}$$

The only difference is that $W^{Q,m}, W^{K,m}, W^{V,m} \in \mathbb{R}^{d_v/M \times d_v}$. The multihead self-attention combines the heads using $W^O \in \mathbb{R}^{d_v \times d_v}$, to output:

$$f_{\text{out}}(g) = W^O \begin{bmatrix} V^1(g) \\ \vdots \\ V^M(g) \end{bmatrix} \tag{5.17}$$

5.7.6 Lie Algebras and Log maps

In this section we briefly introduce Lie algebras and log maps, mainly summarising relevant sections of Hall [2015]. See the reference for a formal and thorough treatment of Lie groups and Lie algebras.

Given a Lie group G, a smooth manifold, its **Lie algebra** $\mathfrak{g}$ is a vector space defined to be the tangent space at the identity element $e \in G$ (together with a bilinear operation called the Lie bracket $[x, y]$, whose details we omit as it is not necessary for understanding log maps). We most commonly deal with **matrix Lie groups**, namely subgroups of the general linear group $GL(n; \mathbb{C})$, the group of all $n \times n$ invertible matrices with entries in $\mathbb{C}$. This includes rotation/reflection groups $SO(n)$ and $O(n)$, as well as the group of translations $T(n)$ and roto-translations $SE(n)$, that are isomorphic to subgroups of $GL(n+1; \mathbb{C})$. For example, $SE(n)$ is isomorphic to the group of matrices of the form:

$$\begin{bmatrix} & & & | \\ & R & & x \\ & & & | \\ — & 0 & — & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

where $R \in SO(n)$ and $x \in \mathbb{R}^n$. For such matrix Lie Groups G, the Lie algebra is precisely the set of all matrices X such that $\exp(tX) \in G$ for all $t \in \mathbb{R}$, where exp is

the matrix exponential ($\exp(A) = I + A + A^2/2! + \ldots$). Hence the Lie algebra $\mathfrak{g}$ can be thought of as a set of matrices, and the matrix exponential exp can be thought of as a map from the Lie algebra $\mathfrak{g}$ to G. This map turns out to be surjective for all the groups mentioned below, and hence we may define the log map $\log : G \to \mathfrak{g}$ in the other direction. Since the effective dimension of Lie algebra, say d, is smaller than the number of entries of the $n \times n$ (or $(n+1) \times (n+1)$ in the case of $SE(n)$ and $T(n)$) matrix element of the Lie algebra, we use a map $\nu : \mathfrak{g} \to \mathbb{R}^d$ that extracts the free parameters from the Lie algebra element, to obtain a form that is suitable as an input to a neural network. See below for concrete examples.

- $G = T(n)$, $t \in \mathbb{R}^n$, $\nu[\log(t)] = t$
- $G = SO(2), R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \in \mathbb{R}^{2\times 2}$

$$\nu[\log(R)] = \theta = \arctan(R_{10}/R_{01}) \tag{5.18}$$

- $G = SE(2), R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \in \mathbb{R}^{2\times 2}, t \in \mathbb{R}^2$

$$\nu[\log(tR)] = \begin{bmatrix} t' \\ \theta \end{bmatrix} \tag{5.19}$$

  where $t' = V^{-1}t$, $V = \begin{bmatrix} a & -b \\ b & a \end{bmatrix}$, $a \triangleq \frac{\sin\theta}{\theta}$, $b \triangleq \frac{1-\cos\theta}{\theta}$

- $G = SO(3), R \in \mathbb{R}^{3\times 3}, t \in \mathbb{R}^3$:

$$\nu[\log(R)] = \nu\left[\frac{\theta}{2\sin\theta}(R - R^\top)\right] = \frac{\theta}{2\sin\theta}\begin{bmatrix} R_{21} - R_{12} \\ R_{02} - R_{20} \\ R_{10} - R_{01} \end{bmatrix} \tag{5.20}$$

  where $\cos\theta = \frac{\mathrm{Tr}(R)-1}{2}$. Note that the Taylor expansion of $\theta/\sin\theta$ should be used when θ is small.

- $G = SE(3), R \in \mathbb{R}^{3\times 3}, t \in \mathbb{R}^3$:

$$\nu[\log(tR)] = \begin{bmatrix} t' \\ r' \end{bmatrix} \tag{5.21}$$

  where $t' = V^{-1}t, r' = \nu[\log(R)]$, $V = I + \frac{1-\cos\theta}{\theta^2}(R - R^\top) + \frac{\theta-\sin\theta}{\theta^3}(R - R^\top)^2$.

Canonical lifting without Log map. Recall that location-based attention only requires a function $k_l(g^{-1}g')$ which we are free to parameterise in any way. Since various groups G can naturally be expressed in terms of real matrices (see above), $g^{-1}g' \in G$ can be expressed as a (flattened) real vector. For example, any $g \in SO(2)$ can simply be expressed as a vector $[t, \theta]^\intercal$ where $t \in \mathbb{R}^2$ and $\theta \in [0, 2\pi)$. Therefore, we can bypass the log map $\nu \circ \log$ and directly use this vector, which we found to be more numerically stable and sometimes resulted in better performance of `LieTransformer`. In particular, for `LieConv-SE2` and `LieTransformer-SE2` on the Hamiltonian spring dynamics task, we did not use the log map and instead opted for this "canonical" lift. We plan to also try this for `LieConv-SE3` and `LieTransformer-SE3` for the QM9 task.

5.7.7 Memory and Time Complexity Comparison with LieConv

5.7.7.1 LieConv

- Inputs: $\{g, f(g)\}_{g \in G_f}$ where
  - $f(g) \in \mathbb{R}^{d_v}$
  - G_f defined as in Section 5.3.1.
- Outputs: $\{g, \frac{1}{|\texttt{nbhd}(g)|} \sum_{g' \in \texttt{nbhd}(g)} k_L(g^{-1}g') f(g')\}_{g \in G_f}$ where
  - $\texttt{nbhd}(g) = \{g' \in G_f : \nu[\log(g)] < r\}$. Let us assume that $|\texttt{nbhd}(g)| \approx n \ \forall g$.
  - $k_L(g^{-1}g') = \texttt{MLP}_\theta(\nu[\log(g^{-1}g')]) \in \mathbb{R}^{d_{\text{out}} \times d_v}$.

There are (at least) two ways of computing `LieConv`: 1. Naive and 2. PointConv Trick.

1. **Naive**
   - **Memory**: Store $k_L(g^{-1}g') \in \mathbb{R}^{d_{\text{out}} \times d_v} \ \forall g \in G_f, g' \in \texttt{nbhd}(g)$. This requires $O(|G_f| n d_{\text{out}} d_v)$ memory.

– **Time**: Compute $k_L(g^{-1}g')f(g') \forall g \in G_f, g' \in \texttt{nbhd}(g)$. This requires $O(|G_f| n d_{\text{out}} d_v)$ flops.

2. **PointConv Trick**

One-line summary: instead of applying a shared linear map then summing across `nbhd`, first sum across `nbhd` then apply the linear map.

Details: $k_L(g^{-1}g') = \texttt{MLP}_\theta(\nu[\log(g^{-1}g')]) = \texttt{reshape}(HM(g^{-1}g'), [d_{\text{out}}, d_v])$ where

– $M(g^{-1}g') \in \mathbb{R}^{d_{\text{mid}}}$ are the final layer activations of $\texttt{MLP}_\theta$.

– $H \in \mathbb{R}^{d_{\text{out}} d_v \times d_{\text{mid}}}$ is the final linear layer of $\texttt{MLP}_\theta$.

The trick assumes $d_{\text{mid}} \ll d_{\text{out}} d_v$, and reorders the computation as:

$$\begin{aligned}&\sum_{g' \in \texttt{nbhd}(g)} \texttt{reshape}(HM(g^{-1}g'), [d_{\text{out}}, d_v]) f(g') \\ &= \texttt{reshape}(H, [d_{\text{out}}, d_v d_{\text{mid}}]) \sum_{g' \in \texttt{nbhd}(g)} M(g^{-1}g') \otimes f(g')\end{aligned}$$

where $\otimes$ is the Kronecker product: $x \otimes y = [x_1 y_1, \dots x_1 y_{d_y}, \dots, x_{d_x} y_1, \dots x_{d_x} y_{d_y}] \in \mathbb{R}^{d_x d_y}$. So $M(g^{-1}g') \otimes f(g') \in \mathbb{R}^{d_v d_{\text{mid}}}$.

– **Memory**: Store $M(g^{-1}g')\ \forall g \in G_f, g' \in \texttt{nbhd}(g)$, and store H. This requires $O(|G_f| n d_{\text{mid}} + d_{\text{out}} d_v d_{\text{mid}})$ memory.

– **Time**: Compute $\sum_{g' \in \texttt{nbhd}(g)} M(g^{-1}g') \otimes f(g')$ via matrix multiplication: $\begin{bmatrix} | & & | \\ M(g^{-1}g'_1) & \dots & M(g^{-1}g'_n) \\ | & & | \end{bmatrix} \begin{bmatrix} — & f(g'_1) & — \\ & \vdots & \\ — & f(g'_n) & — \end{bmatrix}$. This requires $O(d_v n d_{\text{mid}})$ flops.

Then multiply by H, requiring $O(d_v d_{\text{out}} d_{\text{mid}})$ flops.

This is done for each $g \in G_f$, so the total number of flops is $O(|G_f| d_v d_{\text{mid}}(n + d_{\text{out}}))$.

5.7.7.2 Equivariant Self-Attention

- Inputs: $\{g, f(g)\}_{g \in G_f}$ where
  - $f(g) \in \mathbb{R}^{d_v}$
  - G_f defined as in Section 5.3.1.
- Outputs: $\{g, f(g) + \sum_{g' \in \texttt{nbhd}(g)} w_f(g, g') W^V f(g')\}_{g \in G_f}$ where
  - $\texttt{nbhd}(g) = \{g' \in G_f : \nu[\log(g)] < r\}$. Let us assume that $|\texttt{nbhd}(g)| \approx n \ \forall g$.
  - $\{w_f(g, g')\}_{g' \in G_f} = \text{softmax}\left(\{\alpha_f(g, g')\}_{g' \in G_f}\right)$
  - $\alpha_f(g, g') = k_f(f(g), f(g')) + k_x(g^{-1}g')$
  - $k_f(f(g), f(g')) = \left(W^Q f(g)\right)^\top W^K f(g') \in \mathbb{R}$
  - $k_x(g) = \texttt{MLP}_\phi(\nu[\log(g)]) \in \mathbb{R}$
  - $W^Q, W^K, W^V \in \mathbb{R}^{d_v \times d_v}$.
- **Memory**: Store $\alpha_f(g, g')$ and $W^V f(g') \ \forall g \in G_f, g' \in \texttt{nbhd}(g)$. This requires $O(|G_f| n d_v)$ memory.
- **Time**: Compute $k_f(f(g), f(g'))$ and $w_f(g, g') \ \forall g \in G_f, g' \in \texttt{nbhd}(g)$. This requires $O(|G_f| n d_v^2)$ flops.

With multihead self-attention (M heads), the output is:

$$f(g) + W^O \begin{bmatrix} V^1 \\ \vdots \\ V^M \end{bmatrix}$$

where $W^O \in \mathbb{R}^{d_v \times d_v}$, $V^m = \sum_{g' \in \texttt{nbhd}(g)} w_f(g, g') W^{V,m} f(g')$ for $W^{K,m}, W^{Q,m}, W^{V,m} \in \mathbb{R}^{d_v/M \times d_v}$.

- **Memory**: Store $\alpha_f^m(g, g')$ and $W^{V,m} f(g') \ \forall g \in G_f, g' \in \texttt{nbhd}(g), m \in \{1, \dots, M\}$. This requires $O(M|G_f| n + M|G_f| n d_v/M) = O(|G_f| n (M + d_v))$ memory.
- **Time**: Compute $k_f^m(f(g), f(g'))$ and $w_f^m(g, g') \ \forall g \in G_f, g' \in \texttt{nbhd}(g), m \in \{1, \dots, M\}$. This requires $O(M|G_f| n d_v d_v/M) = O(|G_f| n d_v^2)$ flops.

5.7.8 Other Equivariant/Invariant building blocks

G-Pooling is simply averaging over the features across the group: Inputs: $\{f(g)\}_{g\in G_f}$ Output: $\bar{f}(g) \triangleq \frac{1}{|G_f|}\sum_{g\in G_f} f(g)$ Note that G-pooling is invariant with respect to the regular representation.

Pointwise MLPs are MLPs applied independently to each $f(g)$ for $g \in G_f$. It is easy to show that any such pointwise operations are equivariant with respect to the regular representation.

LayerNorm [Ba et al., 2016] is defined as follows:

Inputs: $\{g, f(g)\}_{g\in G_f}$ where

- $f(g) \in \mathbb{R}^{d_v}$
- G_f defined as in Section 5.3.1.

Outputs: $\{g, \beta \odot \frac{f(g)-m(g)}{\sqrt{v(g)+\epsilon}} + \gamma\}_{g\in G_f}$ where

- Division in fraction above is *scalar* division i.e. $\sqrt{v(g)+\epsilon} \in \mathbb{R}$.
- $m(g) = \text{Mean}_c f_c(g') \in \mathbb{R}$.
- $v(g) = \text{Var}_c f_c(g') \in \mathbb{R}$.
- $\beta, \gamma \in \mathbb{R}^D$ are learnable parameters.

BatchNorm We also describe BatchNorm [Ioffe and Szegedy, 2015] that is used in Finzi et al. [2020] for completeness:

Inputs: $\{g, f^b(g)\}_{g\in G_f, b\in\mathcal{B}}$ where

- $f(g) \in \mathbb{R}^{d_v}$
- G_f defined as in Section 5.3.1, $\mathcal{B}$ is the batch of examples.

Outputs: $\{g, \beta \odot \frac{f^b(g)-\mathbf{m}(g)}{\sqrt{\mathbf{v}(g)+\epsilon}} + \gamma\}_{g\in G_f, b\in\mathcal{B}}$ where

- Division in fraction above denotes *pointwise* division i.e. $\sqrt{\mathbf{v}(g)+\epsilon} \in \mathbb{R}^D$.
- $\mathbf{m}(g) = \text{Mean}_{g'\in \texttt{nbhd}(g), b\in\mathcal{B}} f^b(g') \in \mathbb{R}^D$ - Mean is taken for every channel.

- $\mathbf{v}(g) = \text{Var}_{g' \in \texttt{nbhd}(g), b \in \mathcal{B}} f^b(g') \in \mathbb{R}^D$ - Var is taken for every channel.
- $\texttt{nbhd}(g) = \{g' \in G_f : \nu[\log(g)] < r\}$.
- $\beta, \gamma \in \mathbb{R}^D$ are learnable parameters.

A moving average of $\mathbf{m}(g)$ and $\mathbf{v}(g)$ are tracked during training time for use at test time. It is easy to check that both BatchNorm and LayerNorm are equivariant wrt the action of the regular representation π on f (for BatchNorm, note $g' \in \texttt{nbhd}(g)$ iff $u^{-1}g' \in \texttt{nbhd}(u^{-1}g)$).

5.7.9 Experimental details

5.7.9.1 Counting shapes in 2D point clouds

Each training / test example consists of up to two instances of each of the following shapes: triangles, squares, pentagons and the "L" shape. The x_i are the 2D coordinates of each point and $\texttt{f}_i = 1$ for all points.

We performed an architecture search on the `LieTransformer` first and then set the architecture of the `SetTransformer` such that the models have a similar number of parameters (1075k for the `SetTransformer` and 1048k for both `LieTransformer-T2` and `LieTransformer-SE2`) and depth.

Model architecture. The architecture used for the `SetTransformer` [Lee et al., 2019] consists of 8 layers in the encoder, 8 layers in the decoder and 4 attention heads. No inducing points were used.

The architecture used for both `LieTransformer-T2` and `LieTransformer-SE2` is made of 10 layers, 8 heads and feature dimension $d_v = 128$. The dimension of the kernel used is 12. One lift sample was used for each point.

Training procedure. We use Adam [Kingma and Ba, 2015] with parameters $\beta_1 = 0.5$ and $\beta_2 = 0.9$ and a learning rate of $1e-4$. Models are trained with mini-batches of size 32 until convergence.

5.7.9.2 QM9

For the QM9 experiment setup we follow the approach of Anderson et al. [2019] for parameterising the inputs and for the train/validation/test split. The $\mathtt{f}_i$ is a learnable linear embeddings of the vector $[1, c_i, c_i^2]$ for charge c_i, with different linear maps for each atom type. We split the available data as follows: 100k samples for training, 10% for a test set and the rest used for validation.

In applying our model to this task, we ignore the bonding structure of the molecule. As noted in [Klicpera et al., 2019] this should not be needed to learn the task, although it may be helpful as auxiliary information. Given most methods compared against do not use such information, we follows this for a fair comparison (an exception is the $SE(3)$-Transformer [Fuchs et al., 2020b] that uses the bonding information). It would be possible to utilise the bonding structure both in the neighbourhood selection step and as model features by treating only atoms that are connected via a bond to another atom as in the neighbourhood of that atom.

We performed architecture and hyperparameter optimisation on the ϵ_{HOMO} task and then trained with the resulting hyperparameters on the other 11 tasks. `LieTransformer-T3` uses 13 layers of attention blocks (performance saturated at 13 layers), using 8 heads (M) in each layer and feature dimension $d_v = 848$. The attention kernel uses the $linear - concat - linear$ feature embedding, identity embedding of the Lie algebra elements, and an MLP to combine these embeddings into the final attention coefficients. The final part of the model used had minor differences to the one in Figure 5.1. Instead of a global pooling layer followed by a 3 layer MLP, a single linear layer followed by global pooling was used. A single lift sample was used (since $H = \{e\}$ for $T(3)$-invariant models), with the radius of the neighbourhood chosen such that $|\mathtt{nbhd}_\eta(g)| = 50 \ \forall g \in G$ and we uniformly sample 25 points from this neighbourhood. `LieTransformer-SE3` used a similar hyperparameter setting, except using 30 layers (performance saturated at 30 layers) and 2 lift samples were used for each input point ($|\hat{H}| = 2$), with the radius of the neighbourhood η chosen such that the $|\mathtt{nbhd}_\eta(g)| = 25 \ \forall g \in G$ and we uniformly sample 20 points from this neighbourhood. All models were trained using Adam,

with a learning rate of $3e-4$ and a batch size of 75 for 500 epochs. For the `LieConv` models we used the hyperparameter setting that was used in Finzi et al. [2020].

Training these models with $T(3)$ and $SE(3)$ equivariance took approximately 3 and 6 days respectively on a single Nvidia Tesla-V100.

5.7.9.3 Hamiltonian dynamics

Spring dynamics simulation. We exactly follow the setup described in Appendix C.4 of Finzi et al. [2020] for generating the trajectories used in the train and test data.

Model architecture. In all results shown except Figures 5.7 and 5.8, we used a `LieTransformer-T2` with 5 layers, 8 heads and feature dimension $d_v = 160$. The attention kernel uses dot-product for the content component, a 3-layer MLP with hidden layer width 16 for the location component and addition to combine the content and location attention components. (Also, see end of Section 5.7.6 for a relevant discussion about the use of the log map in location-attention k_l for this task.) We use constant normalisation of the weights. We observed a significant drop in performance when, instead of constant normalisation, we used softmax normalisation (which caused small gradients at initialization leading to optimization difficulties). The architecture had 842k parameters. Our small models (with 139k parameters) in Figures 5.7 and 5.8 use 3 layers and feature dimension $d_v = 80$, keeping all else fixed. `LieTransformer-SE(2)` and `LieConv-SE(2)` in Figures 5.7 and 5.8 used 2 lift samples, which were deterministically chosen to be $\hat{H} = C_2 < SO(2)$, where C_2 is the group of 180° rotations. In fact, this yields *exact* equivariance to $T(2) \rtimes C_2$. Note that the true Hamiltonian $H(\mathbf{q}, \mathbf{p})$ for the spring system separates as $H(\mathbf{q}, \mathbf{p}) = K(\mathbf{p}) + V(\mathbf{q})$ where K and V are the kinetic and potential energies of the system respectively. Following Finzi et al. [2020], our model parameterises the potential term V. In particular, x_i is particle i's position and $\mathtt{f}_i = (m_i, k_i)$ where m_i is its mass and k_i is used to define the spring constants: $k_i k_j$ is spring constant for the spring connecting particles i and j (see Appendix C.4 of Finzi et al. [2020] for details).

Training details. To train the `LieTransformer`, we used Adam with a learning rate of 0.001 with cosine annealing and a batch size of 100. For a training dataset of size n, we trained the model for $400\sqrt{3000/n}$ epochs (although we found model training usually converged with fewer epochs). When $n \leq 100$, we used the full dataset in each batch. For training the `LieConv` baseline, we used their default architecture (with 895k parameters) and hyperparameter settings for this task, except for the number of epochs which was $400\sqrt{3000/n}$ to match the setting used for training `LieTransformer`.[2] The small `LieConv` models (with 173k parameters) in Figures 5.7 and 5.8 use 3 layers and 192 channels (instead of the default 4 layers and 384 channels). Lastly, only for the data efficiency results in Figure 5.4, we used early stopping by validation loss and generated *nested* training datasets as the training size varies, keeping the test dataset fixed.

Loss computation. One small difference between our setup and that of Finzi et al. [2020] is in the way we compute the test loss. Since we compare models' losses not only over 5-step roll-outs but also longer 100-step roll-outs, we average the individual time step losses using a geometric mean rather than an arithmetic mean as in Finzi et al. [2020]. Since the losses for later time steps are typically orders of magnitude higher than for earlier time steps (see e.g. Figure 5.5), a geometric mean prevents the losses for later time steps from dominating over the losses for the earlier time steps. During training, we use an arithmetic mean across time steps to compute the loss for optimization, exactly as in Finzi et al. [2020]. This applies for both `LieTransformer` and `LieConv`.

5.7.10 Additional experimental results

5.7.10.1 Hamiltonian Dynamics

[2]This yields better results for `LieConv` compared to those reported by Finzi et al. [2020], where they use fewer total epochs.

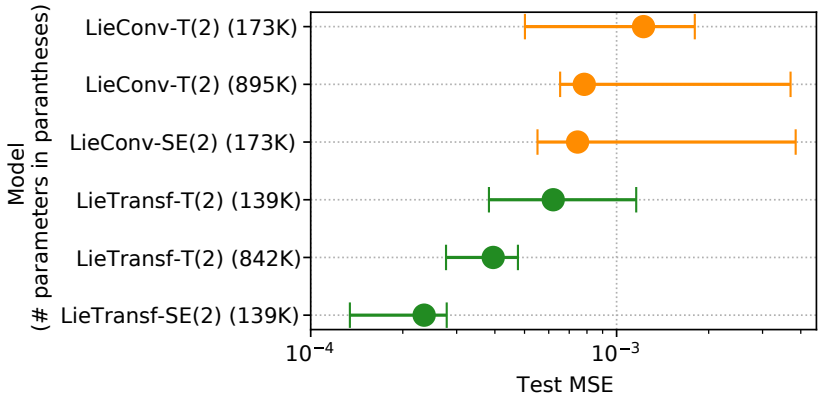


Figure 5.8: Comparison of models by group and number of parameters over 100-step trajectory roll-outs. Similar to the results of Figure 5.7, LieTransformer models outperform their LieConv counterparts when fixing the group and using approximately equal number of parameters. Moreover, models (approximately) equivariant to SE(2) outperform their T(2) counterparts, with LieTransformer-SE(2) again outperforming all other models despite having the smallest number of parameters. Plot shows the median across at least 5 random seeds with interquartile range.

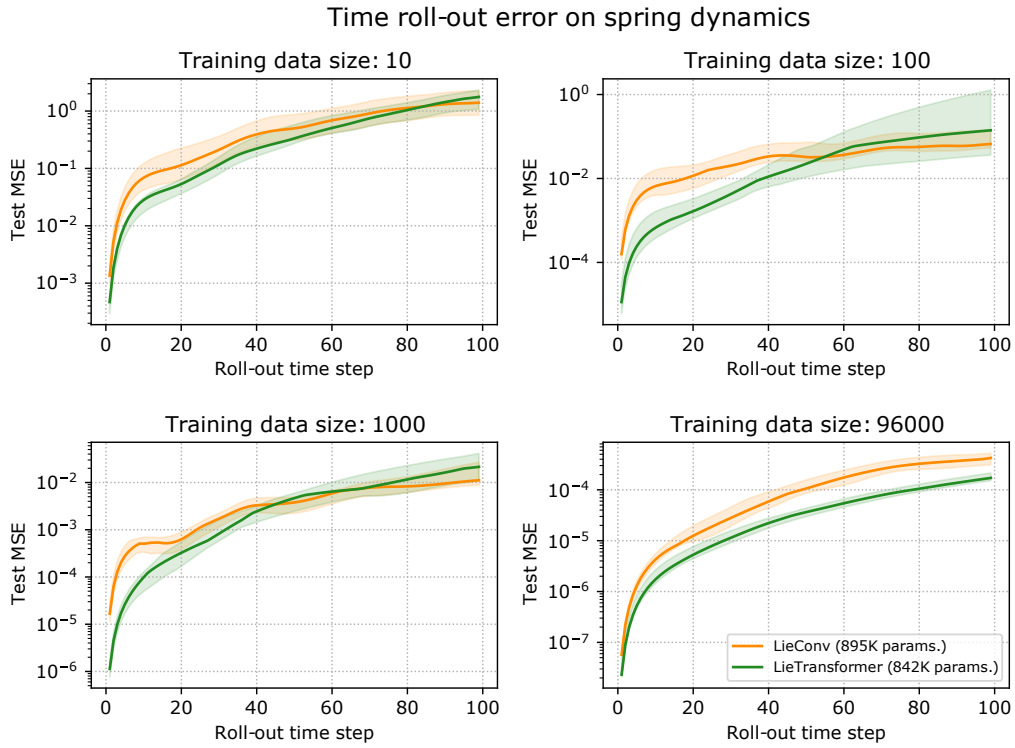


Figure 5.9: Plots of model error as a function of time step for various data sizes. As can be seen, the LieTransformer generally outperforms LieConv across various training data sizes.

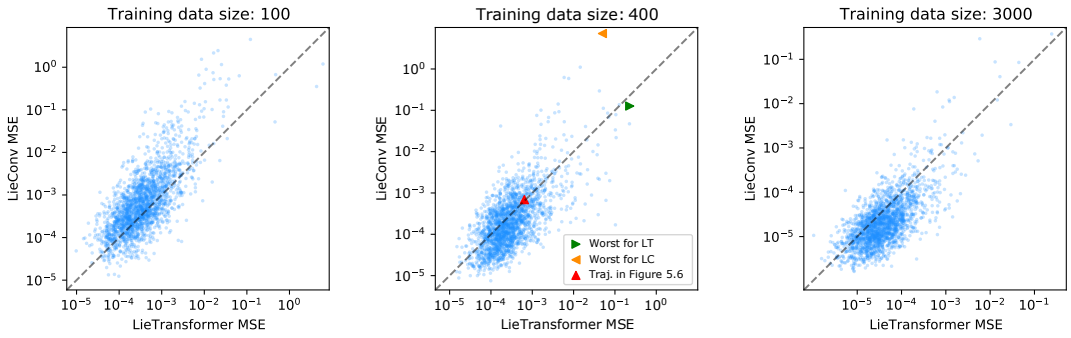
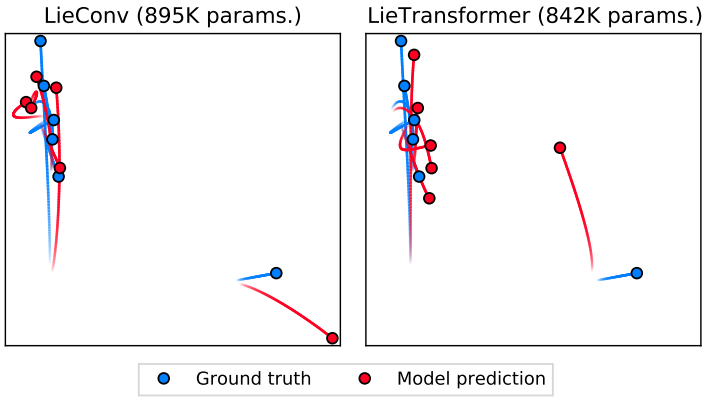
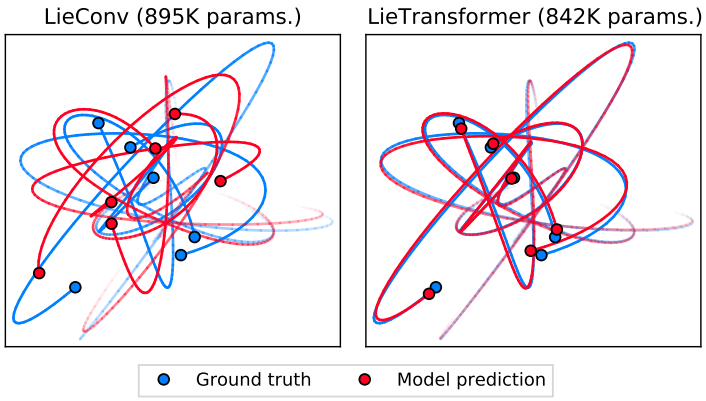


Figure 5.10: Scatter plots comparing the MSE of the LieTransformer against the MSE of LieConv for various training dataset sizes. Each point in a scatter plot corresponds to a 100-step test trajectory, indicating the losses achieved by both models on that trajectory. In the middle figure we have highlighted the MSEs corresponding to the trajectories shown in Figures 5.6 and 5.11.



(a) Test trajectory where LieTransformer has the highest error.



(b) Test trajectory where LieConv has the highest error.

Figure 5.11: Additional example trajectories comparing LieTransformer and LieConv. Both models are trained on a dataset of size 400. See Figure 5.10 for a scatter plot showing these test trajectories and the one in Figure 5.6 in relation to all other trajectories in the test dataset.

6

Neural Ensemble Search for Uncertainty Estimation and Dataset Shift

Abstract: Ensembles of neural networks achieve superior performance compared to stand-alone networks in terms of accuracy, uncertainty calibration and robustness to dataset shift. *Deep ensembles*, a state-of-the-art method for uncertainty estimation, only ensemble random initializations of a *fixed* architecture. Instead, we propose two methods for automatically constructing ensembles with *varying* architectures, which implicitly trade-off individual architectures' strengths against the ensemble's diversity and exploit architectural variation as a source of diversity. On a variety of classification tasks and modern architecture search spaces, we show that the resulting ensembles outperform deep ensembles not only in terms of accuracy but also uncertainty calibration and robustness to dataset shift. Our further analysis and ablation studies provide evidence of higher ensemble diversity due to architectural variation, resulting in ensembles that can outperform deep ensembles, even when having weaker average base learners. Our code is available at `https://github.com/automl/nes`.

Contents

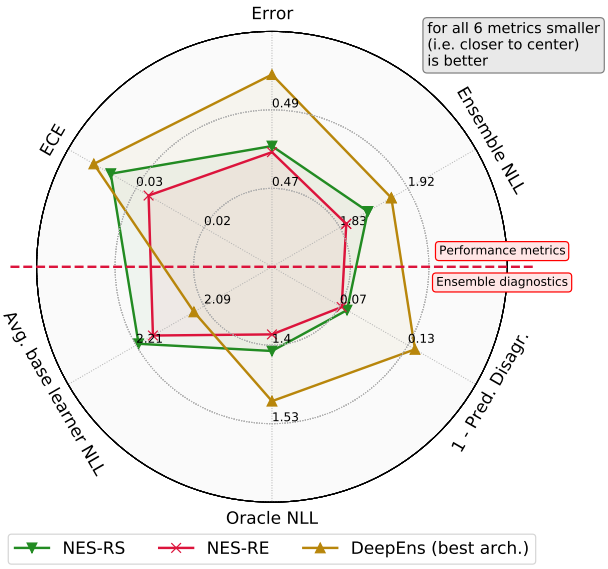


Figure 6.1: A comparison of a deep ensemble with the best architecture (out of 15,625 possible architectures) and ensembles constructed by our method NES on ImageNet-16-120 over NAS-Bench-201. Performance metrics (smaller is better): error, negative log-likelihood (NLL) and expected calibration error (ECE). We also measure average base learner NLL and two metrics for ensemble diversity (see Section 6.3): oracle NLL and [1−predictive disagreement]; small means more diversity for both metrics. NES ensembles outperform the deep ensemble, despite the latter having a significantly stronger average base learner.

6.1 Introduction

Some applications of deep learning rely only on point estimate predictions made by a neural network. However, many critical applications also require reliable predictive uncertainty estimates and robustness under the presence of dataset shift, that is, when the observed data distribution at deployment differs from the training data distribution. Examples include medical imaging [Esteva et al., 2017] and self-driving cars [Bojarski et al., 2016]. Unfortunately, several studies have shown that neural networks are not always robust to dataset shift [Ovadia et al., 2019, Hendrycks and Dietterich, 2019], nor do they exhibit calibrated predictive uncertainty, resulting in incorrect predictions made with high confidence [Guo et al., 2017].

Deep ensembles [Lakshminarayanan et al., 2017] achieve state-of-the-art results for predictive uncertainty calibration and robustness to dataset shift. Notably,

they have been shown to outperform various approximate Bayesian neural networks [Lakshminarayanan et al., 2017, Ovadia et al., 2019, Gustafsson et al., 2020]. Deep ensembles are constructed by training a *fixed* architecture multiple times with different random initializations. Due to the multi-modal loss landscape [Fort et al., 2019, Wilson and Izmailov, 2020], *randomization* by different initializations induces diversity among the base learners to yield a model with better uncertainty estimates than any of the individual base learners (i.e. ensemble members).

Our work focuses on *automatically* selecting *varying* base learner architectures in the ensemble, exploiting architectural variation as a beneficial source of diversity missing in deep ensembles due to their *fixed* architecture. Such architecture selection during ensemble construction allows a more "ensemble-aware" choice of architectures and is based on data rather than manual biases. As discussed in Section 6.2, while ensembles with varying architectures has already been explored in the literature, variation in architectures is typically limited to just varying depth and/or width, in contrast to more complex variations, such as changes in the topology of the connections and operations used, as considered in our work. More generally, automatic ensemble construction is well-explored in AutoML [Feurer et al., 2015, Lévesque et al., 2016, Olson and Moore, 2016, Mendoza et al., 2016]. Our work builds on this by demonstrating that, in the context of *uncertainty estimation*, automatically constructed ensembles with varying architectures outperform deep ensembles that use state-of-the-art, or even *optimal*, architectures (Figure 6.1). Studied under controlled settings, we assess the ensembles by various measures, including predictive performance, uncertainty estimation and calibration, base learner performance and two ensemble diversity metrics, showing that architectural variation is beneficial in ensembles.

Note that, *a priori*, it is not obvious how to find a set of diverse architectures that work well as an ensemble. On the one hand, optimizing the base learners' architectures in isolation may yield multiple base learners with similar architectures (like a deep ensemble). On the other hand, selecting the architectures randomly may yield numerous base learners with poor architectures harming the ensemble.

Moreover, as in neural architecture search (NAS), we face the challenge of needing to traverse vast architectural search spaces. We address these challenges in the problem of *Neural Ensemble Search* (NES), an extension of NAS that aims to find a *set* of complementary architectures that together form a strong ensemble. In summary, our contributions are as follows:

1. We present two NES algorithms for automatically constructing ensembles with varying base learner architectures. As a first step, we present NES with random search (NES-RS), which is simple and easily parallelizable. We further propose NES-RE inspired by regularized evolution [Real et al., 2019], which evolves a population of architectures yielding performant and robust ensembles.

2. This work is the first to apply automatic ensemble construction over architectures to *complex*, state-of-the-art neural architecture search spaces.

3. In the context of uncertainty estimation and robustness to dataset shift, we demonstrate that ensembles constructed by NES improve upon state-of-the-art deep ensembles. We validate our findings over five datasets and two architecture search spaces.

6.2 Related Work

Ensembles and uncertainty estimation. Ensembles of neural networks [Hansen and Salamon, 1990, Krogh and Vedelsby, 1995, Dietterich, 2000] are commonly used to boost performance. In practice, strategies for building ensembles include independently training multiple initializations of the same network, i.e. *deep ensembles* [Lakshminarayanan et al., 2017], training base learners on different bootstrap samples of the data [Zhou et al., 2002], training with diversity-encouraging losses [Liu and Yao, 1999, Lee et al., 2015, Zhou et al., 2018, Webb et al., 2019, Jain et al., 2020, Pearce et al., 2020] and using checkpoints during the training trajectory of a network [Huang et al., 2017a, Loshchilov and Hutter, 2017]. Despite

a variety of approaches, Ashukha et al. [2020] found many sophisticated ensembling techniques to be equivalent to a small-sized deep ensemble by test performance.

Much recent interest in ensembles has been due to their state-of-the-art predictive uncertainty estimates, with extensive empirical studies [Ovadia et al., 2019, Gustafsson et al., 2020] observing that deep ensembles outperform other approaches for uncertainty estimation, notably including Bayesian neural networks [Blundell et al., 2015, Gal and Ghahramani, 2016, Welling and Teh, 2011] and post-hoc calibration [Guo et al., 2017]. Although deep ensembles are not, technically speaking, equivalent to Bayesian neural networks and the relationship between the two is not well understood, diversity among base learners in a deep ensemble yields a model which is arguably closer to *exact* Bayesian model averaging than other approximate Bayesian methods that only capture a single posterior mode in a multi-modal landscape [Wilson and Izmailov, 2020, Fort et al., 2019]. Also, He et al. [2020] draw a rigorous link between Bayesian methods and deep ensembles for wide networks, and Pearce et al. [2020] propose a technique for approximately Bayesian ensembling. Our primary baseline is deep ensembles as they provide state-of-the-art results in uncertainty estimation.

AutoML and ensembles of varying architectures. Automatic ensemble construction is commonly used in AutoML [Feurer et al., 2015, Hutter et al., 2018]. Prior work includes use of Bayesian optimization to tune non-architectural hyperparameters of an ensemble's base learners [Lévesque et al., 2016], posthoc ensembling of fully-connected networks evaluated by Bayesian optimization [Mendoza et al., 2016] and building ensembles by iteratively adding (sub-)networks to improve ensemble performance [Cortes et al., 2017, Macko et al., 2019]. Various approaches, including ours, rely on ensemble selection [Caruana et al., 2004]. We also note that Simonyan and Zisserman [2015] and He et al. [2016a] employ ensembles with varying architectures but *without* automatic construction. Importantly, in contrast to our work, all aforementioned works limit architectural variation to only changing width/depth or fully-connected networks. Moreover, such ensembles have

not been considered before in terms of uncertainty estimation, with prior work focusing instead on predictive performance [Frachon et al., 2020]. Another important part of AutoML is neural architecture search (NAS), the process of automatically designing *single model* architectures [Elsken et al., 2019], using strategies such as reinforcement learning [Zoph and Le, 2017], evolutionary algorithms [Real et al., 2019] and gradient-based methods [Liu et al., 2019a]. We use the search spaces defined by Liu et al. [2019a] and Dong and Yang [2020], two of the most commonly used ones in recent literature.

Concurrent to our work, Wenzel et al. [2020] consider ensembles with base learners having varying hyperparameters using an approach similar to NES-RS. However, they focus on non-architectural hyperparameters such as L_2 regularization strength and dropout rates, keeping the architecture fixed. As in our work, they also consider predictive uncertainty calibration and robustness to shift, finding similar improvements over deep ensembles.

6.3 Visualizing Ensembles of Varying Architectures

In this section, we discuss diversity in ensembles with varying architectures and visualize base learner predictions to add empirical evidence to the intuition that architectural variation results in more diversity. We also define two metrics for measuring diversity used later in Section 6.5.

6.3.1 Definitions and Set-up

Let $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_i, y_i) : i = 1, \ldots, N\}$ be the training dataset, where the input $\boldsymbol{x}_i \in \mathbb{R}^D$ and, assuming a classification task, the output $y_i \in \{1, \ldots, C\}$. We use $\mathcal{D}_{\text{val}}$ and $\mathcal{D}_{\text{test}}$ for the validation and test datasets, respectively. Denote by f_θ a neural network with weights θ, so $f_\theta(\boldsymbol{x}) \in \mathbb{R}^C$ is the predicted probability vector over the classes for input $\boldsymbol{x}$. Let $\ell(f_\theta(\boldsymbol{x}), y)$ be the neural network's loss for data point $(\boldsymbol{x}, y)$. Given M networks $f_{\theta_1}, \ldots, f_{\theta_M}$, we construct the *ensemble* F of these networks by averaging the outputs, yielding $F(\boldsymbol{x}) = \frac{1}{M} \sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.

In addition to the ensemble's loss $\ell(F(\boldsymbol{x}), y)$, we will also consider the *average base learner* loss and the *oracle ensemble's* loss. The average base learner loss is simply defined as $\frac{1}{M}\sum_{i=1}^{M}\ell(f_{\theta_i}(\boldsymbol{x}), y)$; we use this to measure the *average base learner strength* later. Similar to prior work [Lee et al., 2015, Zhou et al., 2018], the oracle ensemble F_{OE} composed of base learners $f_{\theta_1}, \ldots, f_{\theta_M}$ is defined to be the function which, given an input $\boldsymbol{x}$, returns the prediction of the base learner with the smallest loss for $(\boldsymbol{x}, y)$, that is,

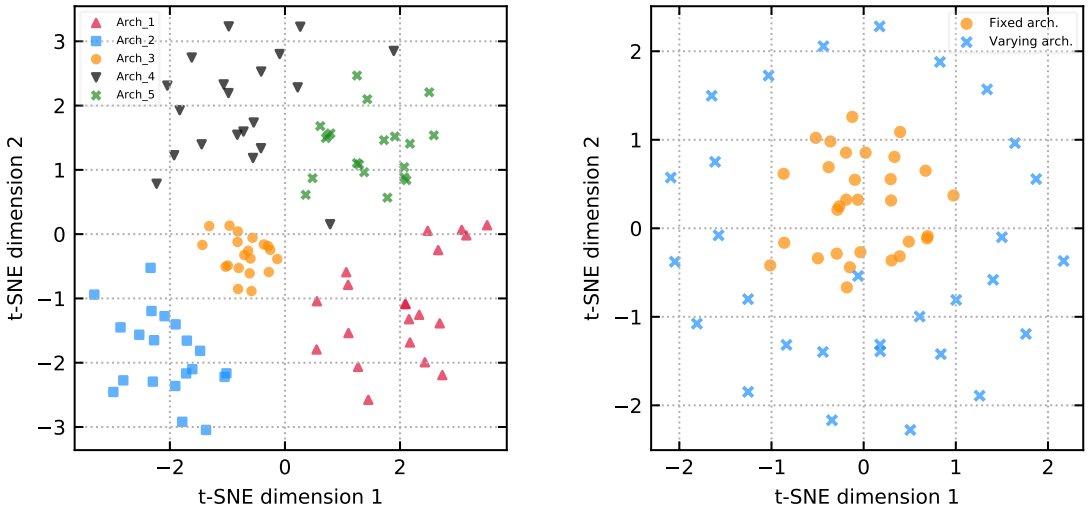
$$F_{\text{OE}}(\boldsymbol{x}) = f_{\theta_k}(\boldsymbol{x}), \quad \text{where} \quad k \in \arg\min_i \ell(f_{\theta_i}(\boldsymbol{x}), y).$$

The oracle ensemble can only be constructed if the true class y is known. We use the oracle ensemble loss as one of the measures of *diversity* in base learner predictions. Intuitively, if base learners make diverse predictions for $\boldsymbol{x}$, the oracle ensemble is more likely to find some base learner with a small loss, whereas if all base learners make identical predictions, the oracle ensemble yields the same output as any (and all) base learners. Therefore, as a rule of thumb, all else being equal, smaller oracle ensemble loss indicates more diverse base learner predictions.

Proposition 4. *Suppose ℓ is negative log-likelihood (NLL). Then, the oracle ensemble loss, ensemble loss, and average base learner loss satisfy the following inequality:*

$$\ell(F_{OE}(\boldsymbol{x}), y) \leq \ell(F(\boldsymbol{x}), y) \leq \frac{1}{M}\sum_{i=1}^{M}\ell(f_{\theta_i}(\boldsymbol{x}), y).$$

We refer to Appendix 6.7.1 for a proof. Proposition 4 suggests that it can be beneficial for ensembles to not only have strong average base learners (smaller upper bound), but also more diversity in their predictions (smaller lower bound). There is extensive theoretical work relating strong base learner performance and diversity with the generalization properties of ensembles [Hansen and Salamon, 1990, Zhou, 2012, Breiman, 2001b, Jiang et al., 2017, Bian and Chen, 2019, Goodfellow et al., 2016]. In Section 6.5, the two metrics we use for measuring diversity are oracle ensemble loss and (normalized) predictive disagreement, defined as the average pairwise predictive disagreement amongst the base learners, normalized by their average error [Fort et al., 2019].



(a) Five different architectures, each trained with 20 different initializations.

(b) Predictions of base learners in two ensembles, one with fixed architecture and one with varying architectures.

Figure 6.2: t-SNE visualization of base learner predictions. Each point corresponds to one network's (dimension-reduced) predictions.

6.3.2 Visualizing Similarity in Base Learner Predictions

The fixed architecture used to build deep ensembles is typically chosen to be a strong stand-alone architecture, either hand-crafted or found by NAS. However, optimizing the base learner's architecture and *then* constructing a deep ensemble can neglect diversity in favor of strong base learner performance. Having base learner architectures vary allows more diversity in their predictions. We provide empirical evidence for this intuition by visualizing the base learners' predictions. Fort et al. [2019] found that base learners in a deep ensemble explore different parts of the function space by means of applying dimensionality reduction to their predictions. Building on this, we uniformly sample five architectures from the DARTS search space [Liu et al., 2019a], train 20 initializations of each architecture on CIFAR-10 and visualize the similarity among the networks' predictions on the test dataset using t-SNE [Van der Maaten and Hinton, 2008]. Experiment details are available in Section 6.5 and Appendix 6.7.2.

As shown in Figure 6.2a, we observe clustering of predictions made by different initializations of a fixed architecture, suggesting that base learners with varying architectures explore different parts of the function space. Moreover, we also

visualize the predictions of base learners of two ensembles, each of size $M = 30$, where one is a deep ensemble and the other has varying architectures (found by NES-RS as presented in Section 6.4). Figure 6.2b shows more diversity in the ensemble with varying architectures than in the deep ensemble. These qualitative findings can be quantified by measuring diversity: for the two ensembles shown in Figure 6.2b, we find the predictive disagreement to be 94.6% for the ensemble constructed by NES and 76.7% for the deep ensemble (this is consistent across independent runs). This indicates higher predictive diversity in the ensemble with varying architectures, in line with the t-SNE results.

6.4 Neural Ensemble Search

In this section, we define *neural ensemble search* (NES). In summary, a NES algorithm optimizes the architectures of base learners in an ensemble to minimize ensemble loss.

Given a network $f : \mathbb{R}^D \to \mathbb{R}^C$, let $\mathcal{L}(f, \mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \ell(f(\boldsymbol{x}), y)$ be the loss of f over dataset $\mathcal{D}$. Given a set of base learners $\{f_1, \ldots, f_M\}$, let `Ensemble` be the function which maps $\{f_1, \ldots, f_M\}$ to the ensemble $F = \frac{1}{M}\sum_{i=1}^{M} f_i$ as defined in Section 6.3 . To emphasize the architecture, we use the notation $f_{\theta,\alpha}$ to denote a network with architecture $\alpha \in \mathcal{A}$ and weights θ, where $\mathcal{A}$ is a search space (SS) of architectures. A NES algorithm aims to solve the following optimization problem:

$$\begin{aligned} &\min_{\alpha_1,\ldots,\alpha_M\in\mathcal{A}} \mathcal{L}\left(\texttt{Ensemble}(f_{\theta_1,\alpha_1}, \ldots, f_{\theta_M,\alpha_M}), \mathcal{D}_{\text{val}}\right) \\ &\text{s.t.} \quad \theta_i \in \arg\min_{\theta} \mathcal{L}(f_{\theta,\alpha_i}, \mathcal{D}_{\text{train}}) \qquad \text{for } i = 1, \ldots, M \end{aligned} \tag{6.1}$$

Eq. 6.1 is difficult to solve for at least two reasons. First, we are optimizing over M architectures, so the search space is effectively $\mathcal{A}^M$, compared to it being $\mathcal{A}$ in typical NAS, making it more difficult to explore fully. Second, a larger search space also increases the risk of overfitting the ensemble loss to $\mathcal{D}_{\text{val}}$. A possible approach here is to consider the ensemble as a single large network to which we apply NAS, but joint training of an ensemble through a single loss has been empirically observed to underperform training base learners independently,

specially for large networks [Webb et al., 2019]. Instead, our general approach to solve Eq. 6.1 consists of two steps:

1. **Pool building**: build a *pool* $\mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_K,\alpha_K}\}$ of size K consisting of potential base learners, where each f_{θ_i,α_i} is a network trained independently on $\mathcal{D}_{\text{train}}$.

2. **Ensemble selection**: select M base learners (without replacement as discussed below) $f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*}$ from $\mathcal{P}$ to form an ensemble which minimizes loss on $\mathcal{D}_{\text{val}}$. (We set $K \geq M$.)

Step 1 reduces the options for the base learner architectures, with the intention to make the search more feasible and focus on strong architectures; step 2 then selects a performant ensemble. This procedure also ensures that the ensemble's base learners are trained independently. We use ensemble selection without replacement [Caruana et al., 2004] for step 2. More specifically, this is forward step-wise selection; that is, given the set of networks $\mathcal{P}$, we start with an empty ensemble and add the network from $\mathcal{P}$ which minimizes ensemble loss on $\mathcal{D}_{\text{val}}$. We repeat this without replacement until the ensemble is of size M. `ForwardSelect`$(\mathcal{P}, \mathcal{D}_{\text{val}}, M)$ denotes the resulting set of M base learners selected from $\mathcal{P}$.

Note that selecting the ensemble from $\mathcal{P}$ is a combinatorial optimization problem; a greedy approach such as `ForwardSelect` is nevertheless effective as shown by Caruana et al. [2004], while keeping computational overhead low, given the predictions of the networks on $\mathcal{D}_{\text{val}}$. We also experimented with various other ensemble selection algorithms, including weighted averaging, as discussed in Section 6.5.2 and Appendix 6.7.3.11, finding `ForwardSelect` to perform best.

We have not yet discussed the algorithm for building the pool in step 1; we present two approaches, NES-RS (Section 6.4.1) and NES-RE (Section 6.4.2). NES-RS is a simple random search based algorithm, while NES-RE is based on regularized evolution [Real et al., 2019], a state-of-the-art NAS algorithm. Note that while gradient-based NAS methods have recently become popular, they are not naively

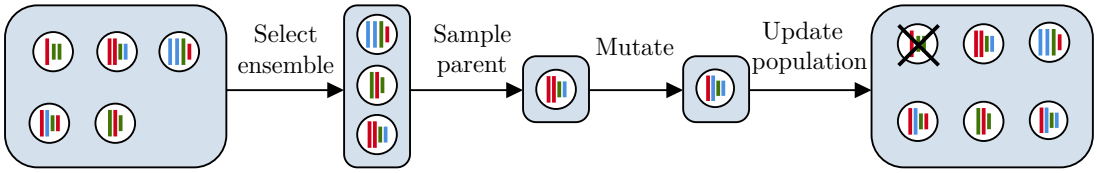


Figure 6.3: Illustration of one iteration of NES-RE. Network architectures are represented as colored bars of different lengths illustrating different layers and widths. Starting with the current population, ensemble selection is applied to select parent candidates, among which one is sampled as the parent. A mutated copy of the parent is added to the population, and the oldest member is removed.

applicable in our setting as the base learner selection component `ForwardSelect` is typically non-differentiable.

6.4.1 NES with Random Search

In NAS, random search (RS) is a competitive baseline on carefully designed architecture search spaces [Li and Talwalkar, 2019, Yang et al., 2020a, Yu et al., 2020]. Motivated by its success and simplicity, NES with random search (NES-RS) builds the pool $\mathcal{P}$ by independently sampling architectures uniformly with replacement from the search space $\mathcal{A}$ (and training them). Since the architectures of networks in $\mathcal{P}$ vary, applying ensemble selection is a simple way to exploit diversity, yielding a performant ensemble. Importantly, NES-RS is easy to parallelize, exactly like deep ensembles. See Algorithm 4 in Appendix 6.7.2.3 for pseudocode.

6.4.2 NES with Regularized Evolution

A more guided approach for building the pool $\mathcal{P}$ is using regularized evolution (RE) [Real et al., 2019]. While RS has the benefit of simplicity by sampling architectures uniformly, the resulting pool might contain many weak architectures, leaving few strong architectures for `ForwardSelect` to choose between. RE is an evolutionary algorithm used on NAS spaces. It explores the search space by evolving (via mutations) a *population* of architectures. We first briefly describe RE as background before NES-RE. RE starts with a randomly initialized fixed-size population of architectures. At each iteration, a subset of size m of the population

Algorithm 3: NES with Regularized Evolution

Input: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$; population size P; number of parent candidates m.

1 Sample P architectures $\alpha_1, \ldots, \alpha_P$ independently and uniformly from $\mathcal{A}$.
2 Train each architecture α_i using $\mathcal{D}_{\text{train}}$, and initialize $\mathfrak{p} = \mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_P,\alpha_P}\}$.
3 **while** $|\mathcal{P}| < K$ **do**
4 Select m parent candidates $\{f_{\widetilde{\theta}_1,\widetilde{\alpha}_1}, \ldots, f_{\widetilde{\theta}_m,\widetilde{\alpha}_m}\} = \texttt{ForwardSelect}(\mathfrak{p}, \mathcal{D}_{\text{val}}, m)$.
5 Sample uniformly a parent architecture α from $\{\widetilde{\alpha}_1, \ldots, \widetilde{\alpha}_m\}$. ▷ α `stays in` $\mathfrak{p}$.
6 Apply mutation to α, yielding child architecture β.
7 Train β using $\mathcal{D}_{\text{train}}$ and add the trained network $f_{\theta,\beta}$ to $\mathfrak{p}$ and $\mathcal{P}$.
8 Remove the oldest member in $\mathfrak{p}$. ▷ `as done in RE.`
9 **end**
10 Select base learners $\{f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*}\} = \texttt{ForwardSelect}(\mathcal{P}, \mathcal{D}_{\text{val}}, M)$ by forward step-wise selection without replacement.
11 **return** ensemble $\texttt{Ensemble}(f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*})$

is sampled, from which the best network by validation loss is selected as the parent. A mutated copy (e.g. changing an operation in the network, see Appendix 6.7.2.4 for examples of mutations) of the parent architecture, called the child, is trained and added to the population, and the oldest member of the population is removed, preserving the population size. This is iterated until the computational budget is reached, returning the *history*, i.e. all the networks evaluated during the search, from which the best model is chosen by validation loss.

Building on RE for NAS, we propose NES-RE to build the pool of potential base learners. NES-RE starts by randomly initializing a population $\mathfrak{p}$ of size P. At each iteration, we first apply `ForwardSelect` to the population to select an ensemble of size m, then we uniformly sample one base learner from this ensemble to be the parent. A mutated copy of the parent is added to $\mathfrak{p}$ and the oldest network is removed, as in usual regularized evolution. This process is repeated until the computational budget is reached, and the history is returned as the pool $\mathcal{P}$. See Algorithm 3 for pseudocode and Figure 6.3 for an illustration.

Also, note the distinction between the *population* and the *pool* in NES-RE: the population is evolved, whereas the pool is the set of all networks evaluated during evolution (i.e., the history) and is used post-hoc for selecting the ensemble. Moreover, `ForwardSelect` is used both for selecting m parent candidates (line 4 in NES-RE) and choosing the final ensemble of size M (line 9 in NES-RE). In general, $m \neq M$.

6.4.3 Ensemble Adaptation to Dataset Shift

Deep ensembles offer (some) robustness to dataset shift relative to training data. In general, one may not know the type of shift that occurs at test time. By using an ensemble, diversity in base learner predictions prevents the model from relying on one base learner's predictions which may not only be incorrect but also overconfident.

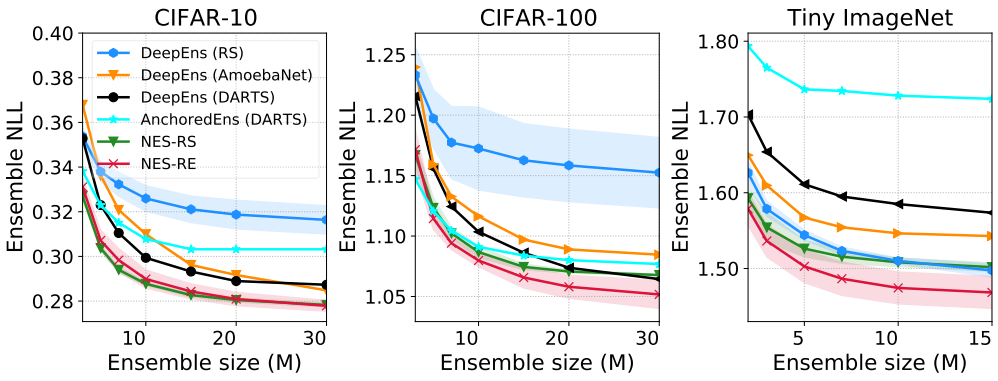
We assume that one does not have access to data points with test-time shift at training time, but one does have access to some validation data $\mathcal{D}_{\text{val}}^{\text{shift}}$ with a *validation* shift, which encapsulates one's belief about test-time shift. Crucially, test and validation shifts are disjoint. To adapt NES-RS and NES-RE to return ensembles robust to shift, we propose using $\mathcal{D}_{\text{val}}^{\text{shift}}$ instead of $\mathcal{D}_{\text{val}}$ whenever applying `ForwardSelect` to select the final ensemble. In Algorithms 3 and 4, this is in lines 9 and 3, respectively. Our experiments show that this is highly effective against shift.

Note that in line 4 of Algorithm 3, we can also replace $\mathcal{D}_{\text{val}}$ with $\mathcal{D}_{\text{val}}^{\text{shift}}$ when expecting test-time shift; we simply sample one of $\mathcal{D}_{\text{val}}, \mathcal{D}_{\text{val}}^{\text{shift}}$ uniformly at each iteration, in order to promote exploration of architectures that work well both in-distribution and during shift and reduce cost by avoiding running NES-RE once for each of $\mathcal{D}_{\text{val}}, \mathcal{D}_{\text{val}}^{\text{shift}}$. See Appendices 6.7.3.3 and 6.7.2.4 for further discussion.

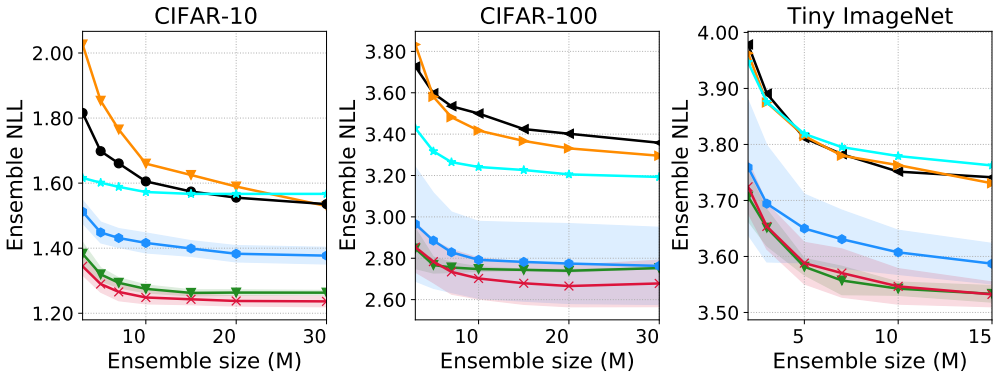
6.5 Experiments

6.5.1 Comparison to Baselines: Uncertainty Estimation & Robustness to Dataset Shift

We compare NES to deep ensembles on different choices of architecture search space (DARTS [Liu et al., 2019a] and NAS-Bench-201 [Dong and Yang, 2020] search spaces) and dataset (Fashion-MNIST, CIFAR-10, CIFAR-100, ImageNet-16-120 and Tiny ImageNet). The search spaces are *cell-based*, containing a rich variety of convolutional architectures with differing cell topologies, number of connections and operations (see Appendix 6.7.2.1 for visualizations). For CIFAR-10/100 and Tiny ImageNet, we also consider dataset shifts proposed by Hendrycks and Dietterich [2019]. We use NLL, classification error and expected calibration



(a) No data shift



(b) Dataset shift: severity 5 (out of 5)

Figure 6.4: NLL vs. ensemble sizes on CIFAR-10, CIFAR-100 and Tiny ImageNet with and without dataset shifts [Hendrycks and Dietterich, 2019] over the DARTS search space. Mean NLL shown with 95% confidence intervals.

error (ECE) [Guo et al., 2017, Naeini et al., 2015] as our metrics, for which small values are better. Note that NLL and ECE evaluate predictive uncertainty. Experimental/implementation details are in Appendix 6.7.2, additional experiments are in Appendix 6.7.3. All evaluations are on the test dataset. Each paragraph below highlights one of our main findings.

Baselines. One of our main baselines is deep ensembles with fixed, optimized architectures. We consider various optimized architectures, indicated as "DeepEns (`arch`)":

- On the DARTS search space, we consider the architectures found by:
  1. the DARTS algorithm (DeepEns (`DARTS`)),
  2. regularized evolution (DeepEns (`AmoebaNet`)),

3. random search, *with the same number of networks trained* as NES algorithms (DeepEns (`RS`)).

- On the NAS-Bench-201 search space, in addition to DeepEns (`RS`), we consider:

1. the architecture found by GDAS[1] [Dong and Yang, 2019] (DeepEns (`GDAS`)),
2. the best architecture in the search space by validation loss (DeepEns (`best arch.`)).[2]

We also compare to *anchored ensembles* [Pearce et al., 2020], a recent technique for approximately Bayesian ensembles, which explicitly regularizes each base learner towards a fresh initialization sample and aims to induce more ensemble diversity. We use the DARTS architecture, and our implementation is described in Appendix 6.7.2.5.

NES shows improved predictive uncertainty (NLL) and robustness to dataset shift (Figure 6.4). Figure 6.4a shows the NLL achieved by NES-RS, NES-RE and the baselines as functions of the ensemble size M without dataset shift. We find that NES algorithms consistently outperform deep ensembles, with NES-RE usually outperforming NES-RS. Next, we evaluate the robustness of the ensembles to dataset shift in Figure 6.4b. In our setup, all base learners are trained on $\mathcal{D}_{\text{train}}$ without data augmentation of shifted examples. However, as explained in Section 6.4.3, we use a shifted validation dataset, $\mathcal{D}_{\text{val}}^{\text{shift}}$, and evaluate on a shifted test dataset, $\mathcal{D}_{\text{test}}^{\text{shift}}$. The types of shifts appearing in $\mathcal{D}_{\text{val}}^{\text{shift}}$ are disjoint from those in $\mathcal{D}_{\text{test}}^{\text{shift}}$. The severity of the shift varies between 1-5. We refer to Appendix 6.7.2 and Hendrycks and Dietterich [2019] for details. The fixed architecture used in the baseline DeepEns (RS) is selected based on its loss over $\mathcal{D}_{\text{val}}^{\text{shift}}$, but the DARTS and AmoebaNet are architectures from the literature. As shown in Figure 6.4b, ensembles picked by NES-RS and NES-RE are significantly more

[1]We did not consider the DARTS algorithm on NAS-Bench-201, since it returns degenerate architectures with poor performance on this space [Dong and Yang, 2020]. Whereas, GDAS yields state-of-the-art performance on this space.

[2]This is feasible, because all architectures in this search space were evaluated and are available.

Table 6.1: Classification error of ensembles for different shift severities. Best values and all values within 95% confidence interval are bold faced. Note that NAS-Bench-201 comes with each architecture trained with *three* random initializations; therefore we set $M = 3$ in that case.

(a) $M = 10$; DARTS search space.

Dataset	Shift Severity	Classif. error (%), $\mathcal{A}$ = DARTS search space					
		DeepEns (RS)	DeepEns (Amoe.)	DeepEns (DARTS)	AnchoredEns (DARTS)	NES-RS	NES-RE
CIFAR-10	0	$10.8_{\pm 0.2}$	9.7	10.0	10.2	$\mathbf{9.4}_{\pm 0.1}$	$\mathbf{9.4}_{\pm 0.2}$
	3	$25.1_{\pm 0.7}$	25.6	26.3	28.6	$23.2_{\pm 0.2}$	$\mathbf{22.9}_{\pm 0.2}$
	5	$41.1_{\pm 0.9}$	42.7	42.9	44.9	$38.0_{\pm 0.2}$	$\mathbf{37.4}_{\pm 0.4}$
CIFAR-100	0	$33.2_{\pm 1.2}$	31.6	30.9	30.9	$\mathbf{30.7}_{\pm 0.1}$	$\mathbf{30.4}_{\pm 0.4}$
	3	$54.8_{\pm 1.6}$	54.2	55.1	55.2	$\mathbf{49.8}_{\pm 0.1}$	$\mathbf{49.1}_{\pm 1.0}$
	5	$64.3_{\pm 3.2}$	68.5	68.5	68.9	$\mathbf{62.4}_{\pm 0.2}$	$\mathbf{61.4}_{\pm 1.4}$
Tiny ImageNet	0	$\mathbf{37.5}_{\pm 0.3}$	38.5	39.1	42.8	$\mathbf{37.4}_{\pm 0.2}$	$\mathbf{37.0}_{\pm 0.6}$
	3	$53.8_{\pm 0.6}$	55.4	54.6	58.7	$53.0_{\pm 0.2}$	$\mathbf{52.7}_{\pm 0.2}$
	5	$70.5_{\pm 0.4}$	71.7	71.6	73.7	$\mathbf{69.9}_{\pm 0.2}$	$70.2_{\pm 0.1}$

(b) $M = 3$; NAS-Bench-201 search space.

Dataset	Shift Severity	Classif. error (%), $\mathcal{A}$ = NAS-Bench-201 search space				
		DeepEns (GDAS)	DeepEns (best arch.)	DeepEns (RS)	NES-RS	NES-RE
CIFAR-10	0	8.4	**7.2**	$7.8_{\pm 0.2}$	$7.7_{\pm 0.1}$	$7.6_{\pm 0.1}$
	3	28.7	27.1	$28.3_{\pm 0.3}$	$\mathbf{22.0}_{\pm 0.2}$	$22.5_{\pm 0.1}$
	5	47.8	46.3	$37.1_{\pm 0.0}$	$\mathbf{32.5}_{\pm 0.2}$	$33.0_{\pm 0.5}$
CIFAR-100	0	29.9	26.4	$26.3_{\pm 0.4}$	$\mathbf{23.3}_{\pm 0.3}$	$\mathbf{23.5}_{\pm 0.2}$
	3	60.3	54.5	$57.0_{\pm 0.9}$	$\mathbf{46.6}_{\pm 0.3}$	$\mathbf{46.7}_{\pm 0.5}$
	5	75.3	69.9	$64.5_{\pm 0.0}$	$\mathbf{59.7}_{\pm 0.2}$	$60.0_{\pm 0.6}$
ImageNet-16-120	0	49.9	49.9	$50.5_{\pm 0.6}$	$\mathbf{48.1}_{\pm 1.0}$	$\mathbf{47.9}_{\pm 0.4}$

robust to dataset shift than the baselines, highlighting the effectiveness of applying `ForwardSelect` with $\mathcal{D}_{\text{val}}^{\text{shift}}$. Unsurprisingly, AnchoredEns (DARTS) and DeepEns (DARTS/AmoebaNet) perform poorly compared to the other methods, as they are not optimized to deal with dataset shift here. The results of our experiments on Fashion-MNIST are in Appendix 6.7.3.1. In line with the results in this section, both NES algorithms outperform deep ensembles with NES-RE performing best.

Better uncertainty calibration versus dataset shift and classification

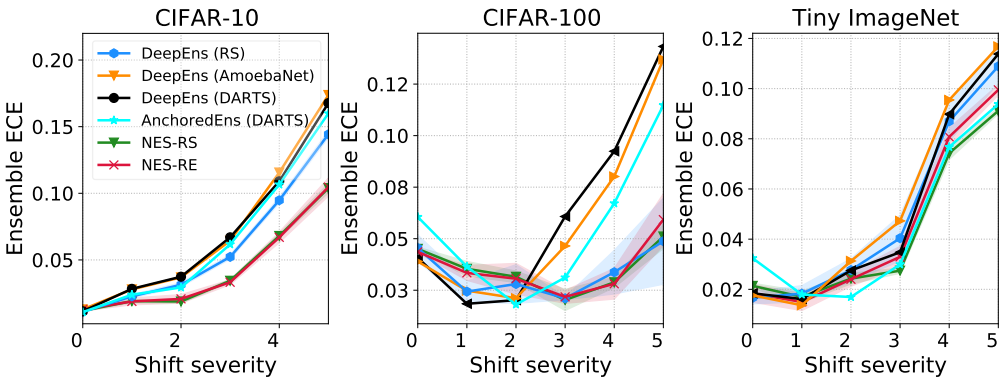


Figure 6.5: ECE vs. dataset shift severity on CIFAR-10, CIFAR-100 and Tiny ImageNet over the DARTS search space. No dataset shift is indicated as severity 0. Ensemble size is $M = 10$.

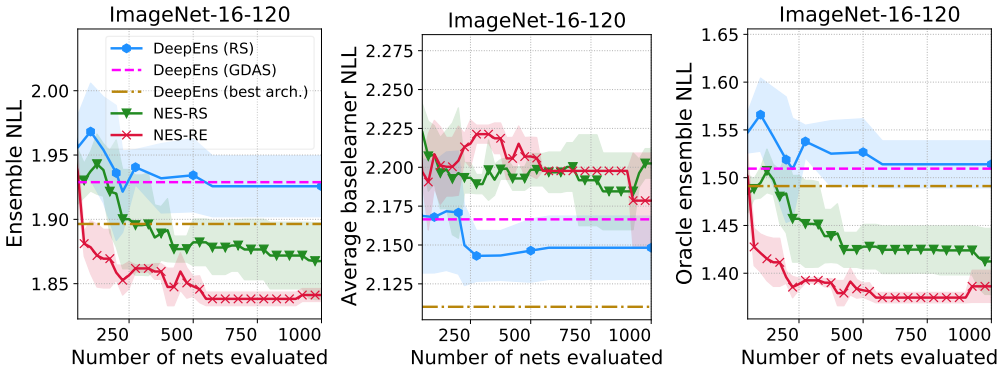


Figure 6.6: Ensemble, average base learner and oracle ensemble NLL versus budget K on ImageNet-16-120 over the NAS-Bench-201 search space. Ensemble size is $M = 3$.

error (Figure 6.5, Table 6.1). We also assess the ensembles by error and ECE. In short, ECE measures the mismatch between the model's confidence and accuracy. Figure 6.5 shows the ECE achieved by the ensembles at varying dataset shift severities, noting that uncertainty calibration is especially important when models are used during dataset shift. Ensembles found with NES tend to exhibit better uncertainty calibration and are either competitive with or outperform anchored and deep ensembles for most shift severities. Notably, on CIFAR-10, ECE is reduced by up to 40% relative to the baselines. In terms of classification error, we find that ensembles constructed by NES outperform deep ensembles, with reductions of up to 7 percentage points in error, shown in Table 6.1. As with NLL, NES-RE tends to outperforms NES-RS.

Ensembles found by NES tend to be more diverse (Table 6.2). We

measure ensemble diversity using the two metrics predictive disagreement (larger means more diverse) and oracle ensemble NLL (smaller means more diverse) as defined in Section 6.3 and average base learner NLL. Table 6.2 contrasts NES with the baselines in terms of these metrics. In terms of both diversity metrics, ensembles constructed by NES tend to be more diverse than anchored and deep ensembles. Ranking of the methods is largely consistent for different ensemble sizes M (see Appendix 6.7.3). Unsurprisingly, note that the average base learner in NES is not always the best: by optimizing the fixed architecture first and *then* ensembling it, deep ensembles end up with a strong average base learner at the expense of less ensemble diversity. Despite higher average base learner NLL, NES ensembles perform better (recall Figure 6.4), highlighting once again the importance of diversity.

NES outperforms the deep ensemble of the best architecture in the NAS-Bench-201 search space (Figure 6.6). Next, we compare NES to deep ensembles over the NAS-Bench-201 search space, which has two benefits: we demonstrate that our findings are not specific to the DARTS search space, and NAS-Bench-201 is an exhaustively evaluated search space for which all architectures' trained weights are available (three initializations per architecture), allowing us to compare NES to the deep ensemble of the *best* architecture by validation loss. Results shown in Figure 6.6 compare the losses of the ensemble, average base learner and oracle ensemble versus the number of networks evaluated K. Interestingly, although DeepEns (best arch.) has a significantly stronger average base learner than the other methods, its lack of diversity, as indicated by higher oracle ensemble loss (Figure 6.6) and lower predictive disagreement (Figure 6.1), yields a weaker ensemble than both NES algorithms. Also, NES-RE outperforms NES-RS with a 6.6x speedup as shown in Figure 6.6-left.

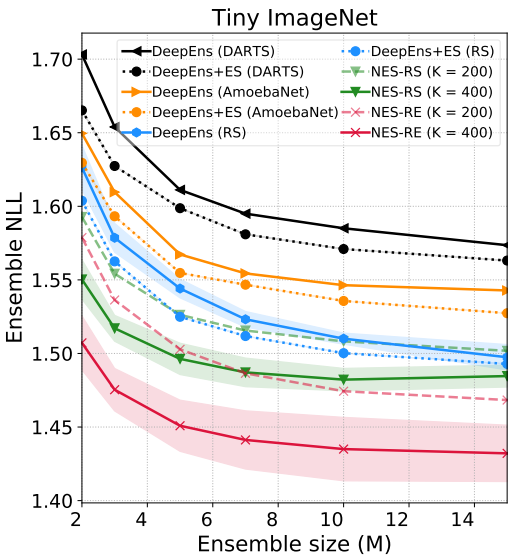
6.5.2 Analysis and Ablations

Why does NES work? What if deep ensembles use ensemble selection over initializations? (Figure 6.7). NES algorithms differ from deep ensembles in two important ways: the ensembles use varying architectures and NES utilizes

Table 6.2: Diversity and base learner strength. Predictive disagreement (larger means more diverse) and oracle ensemble NLL (smaller means more diverse) are defined in Section 6.3. Despite the stronger base learners, deep ensembles tend to be less diverse than ensembles constructed by NES. The results are consistent across datasets and shift severities (Appendix 6.7.3). Best values and all values within 95% confidence interval are bold faced.

Dataset	**Metric**	**Method** (with $M = 10$)					
		DeepEns (RS)	DeepEns (Amoe.)	DeepEns (DARTS)	AnchoredEns (DARTS)	NES-RS	NES-RE
CIFAR-10	Pred. Disagr.	$0.823_{\pm 0.023}$	**0.947**	0.932	0.842	**0.948**$_{\pm 0.004}$	$0.943_{\pm 0.009}$
	Oracle NLL	$0.125_{\pm 0.007}$	0.103	0.093	0.113	**0.086**$_{\pm 0.001}$	$0.088_{\pm 0.002}$
	Avg. bsl. NLL	$0.438_{\pm 0.005}$	0.552	0.513	**0.411**	$0.485_{\pm 0.003}$	$0.493_{\pm 0.010}$
CIFAR-100	Pred. Disagr.	$0.831_{\pm 0.027}$	**0.946**	0.935	0.839	$0.934_{\pm 0.006}$	$0.943_{\pm 0.014}$
	Oracle NLL	$0.635_{\pm 0.040}$	0.502	0.509	0.583	**0.498**$_{\pm 0.008}$	**0.487**$_{\pm 0.014}$
	Avg. bsl. NLL	$1.405_{\pm 0.028}$	1.552	1.491	**1.290**	$1.467_{\pm 0.022}$	$1.487_{\pm 0.036}$
Tiny ImageNet	Pred. Disagr.	$0.742_{\pm 0.008}$	0.737	0.749	0.662	**0.772**$_{\pm 0.005}$	**0.768**$_{\pm 0.005}$
	Oracle NLL	$0.956_{\pm 0.009}$	0.987	1.009	1.203	$0.929_{\pm 0.007}$	**0.910**$_{\pm 0.015}$
	Avg. bsl. NLL	$1.743_{\pm 0.005}$	1.764	1.813	1.871	$1.755_{\pm 0.007}$	**1.728**$_{\pm 0.012}$

ensemble selection (`ForwardSelect`) to pick the base learners. On Tiny ImageNet over the DARTS search space, we conduct an experiment to explore whether the improvement offered by NES over deep ensembles is only due to ensemble selection. The baselines "DeepEns + ES" operate as follows: we optimize a fixed architecture for the base learners, train K random initializations of it to form a pool and apply `ForwardSelect` to select an ensemble of size M. Figure 6.7-left shows that both NES algorithms (each shown for two computational budgets $K = 200, 400$) outperform all DeepEns + ES baselines. Figure 6.7-right contains the cost of each method in terms of the number of networks trained. Note that DeepEns + ES (RS) is the most competitive of the deep ensemble baselines, and, at an equal budget of 400, it is outperformed by both NES algorithms. However, even at half the cost (200), NES-RE outperforms DeepEns + ES (RS) while NES-RS performs competitively. As expected, deep ensembles *with* ensemble selection consistently perform better than *without* ensemble selection at the expense of higher computational cost, but do not close the gap with NES algorithms. We also re-emphasize that comparisons between NES and deep ensembles in our experiments always fix the base learner

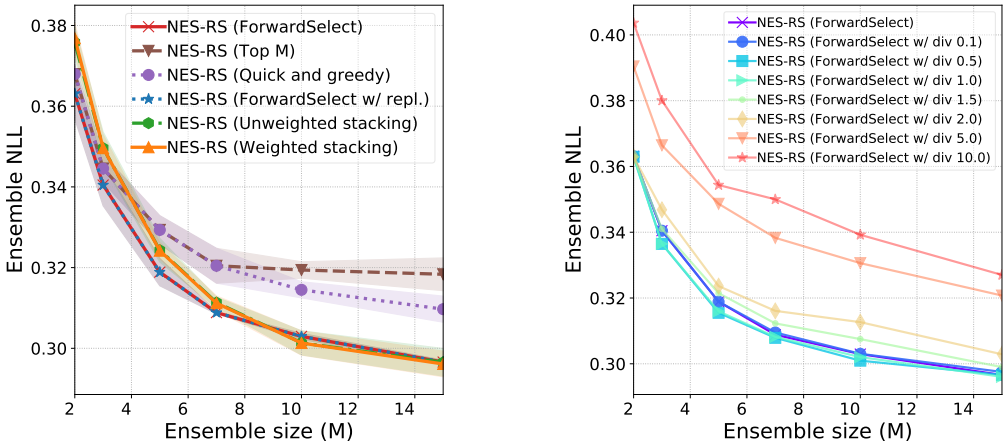


Method	# nets trained	
	Arch.	Ensemble
DeepEns (DARTS)	32	10
DeepEns + ES (DARTS)	32	200
DeepEns (AmoebaNet)	25200	10
DeepEns + ES (AmoebaNet)	25200	200
DeepEns (RS)	200	10
DeepEns + ES (RS)	200	200
NES-RS ($K = 200$)		200
NES-RS ($K = 400$)		400
NES-RE ($K = 200$)		200
NES-RE ($K = 400$)		400

Figure 6.7: Comparison of NES to deep ensembles with ensemble selection on Tiny ImageNet. LEFT: NLL vs ensemble size. RIGHT: Cost for Tiny ImageNet experiments reported in terms of the number of networks trained when $M = 10$. The "arch" column indicates the number of networks trained to first select an architecture, and the "ensemble" column contains the number of networks trained to build the ensemble.

training routine and method for composing the ensemble, so variations in ensemble performance are only due to the architecture choices. Therefore, to summarize, combined with the finding that NES outperforms deep ensembles even when NES' average base learner is weaker (as in e.g. Figure 6.6 and Table 6.2), we find architectural variation in ensembles to be important for NES' performance gains.

Computational cost and parallelizability. The primary computational cost involved in NES is training K networks to build the pool $\mathcal{P}$. In our experiments on the DARTS search space, we set K to be 400 for CIFAR-10/100 and 200 for Tiny ImageNet (except Figure 6.7 which additionally considers $K = 400$). Figure 6.7-right gives an example of costs for each method. The cost of a deep ensemble with an optimized architecture stems from the initial architecture search and the subsequent training of M random initializations to build ensemble. We refer to Appendix 6.7.2 for details and discussion of computation cost, including training times. We note that apart from DeepEns (DARTS) and AnchoredEns (DARTS), NES has a lower computational cost than the baselines in our experiments. Similar to deep ensembles, training the pool for NES-RS is embarrassingly parallel. Our



(a) Ensemble selection algorithms.

(b) Ensemble selection with explicit diversity regularization.

Figure 6.8: NES-RS on CIFAR-10.

implementation of NES-RE is also parallelized as described in Appendix 6.7.2.

Comparison of different ensemble selection algorithms. Both NES algorithms utilize `ForwardSelect` as the ensemble selection algorithm (ESA). We experimented with various other choices of ESAs, including weighted averaging and explicit diversity regularization, as shown in Figure 6.8. A detailed description of each ESA is in Appendix 6.7.3.11. In summary, our choice of `ForwardSelect` performs better than or at par with all ESAs considered. Moreover, weighted averaging using stacking and Bayesian model averaging has a very minor impact since the weights end up being close to uniform. Explicit diversity regularization, as shown in Figure 6.8b, appears to slightly improve performance in some cases, provided that the diversity regularization strength hyperparameter is appropriately tuned.

NES is insensitive to the size of validation dataset $\mathcal{D}_{\text{val}}$. We study the sensitivity of NES to the size of $\mathcal{D}_{\text{val}}$. Specifically, we measure test loss of the ensembles selected using $\mathcal{D}_{\text{val}}$ of different sizes (with as few as 10 validation samples). The results in Figure 6.9 indicate that NES is insensitive to validation size for different levels of dataset shift severity, achieving almost the same loss when using $10\times$ fewer validation data. We provide more details in Appendix 6.7.3.8 and we also discuss why overfitting to $\mathcal{D}_{\text{val}}$ is averted during ensemble selection.

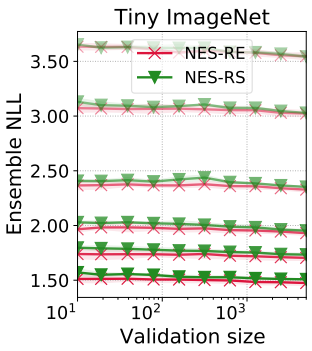


Figure 6.9: Test performance of NES algorithms with varying validation data sizes. Each curve corresponds to one particular dataset shift severity (0-5 with 0 being no shift). The more transparent curves correspond to higher shift severities.

Mutations of an architecture are similar in function space and by performance. To explore how NES-RE traverses the search space, we analyzed how mutations affect an architecture as follows. We sampled five random architectures from the DARTS space, and for each one, we applied a single random mutation twenty times, yielding a "family" of parent-children architectures, which are then trained on CIFAR-10. Figure 6.10-left shows the result of t-SNE applied to the test predictions of these architectures, where each color corresponds to one "family", of which the "star" is the parent architecture and the circles are the children architectures. The clustering demonstrates that architectures which differ by only a single mutation are similar in function space after training. Figure 6.10-right shows the NLL and error achieved by these architectures. Again, similar clustering shows that architectures differing by a single mutation also perform similarly w.r.t. NLL and error. This confirms that mutations allow for locally exploring the search space.

Further experiments. We also provide additional experiments in the Appendix. This includes a comparison of ensembles built by averaging logits vs. probabilities (Appendix 6.7.3.10), a comparison of NES to ensembles with other hyperparameters being varied (either width/depth or training hyperparameters similar to Wenzel et al. [2020]) (Appendix 6.7.3.7) and using a weight-sharing model [Bender et al., 2018] as a proxy to accelerate the search phase in NES (Appendix 6.7.3.12).

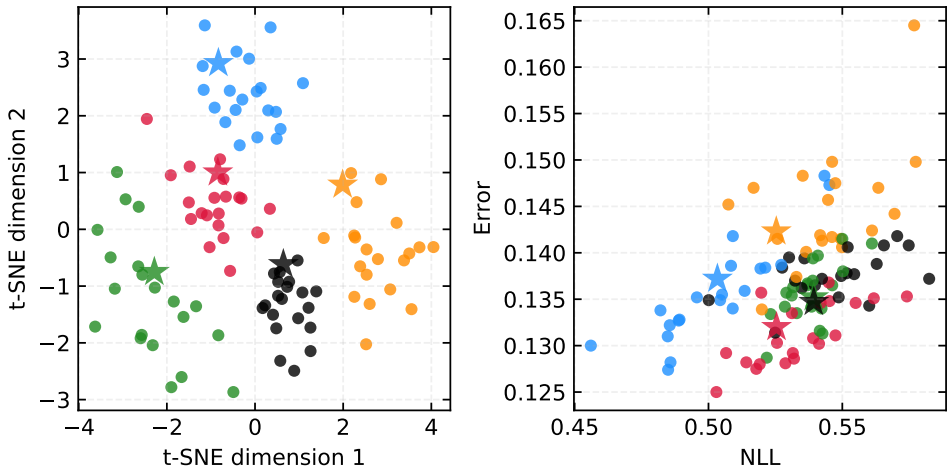


Figure 6.10: LEFT: t-SNE of the test predictions of 20 single random mutations (circles) from 5 parent architecture (stars). RIGHT: NLL and error achieved by these architectures.

6.6 Conclusion, Limitations & Broader Impact

We presented Neural Ensemble Search for automatically constructing ensembles with varying architectures and demonstrated that the resulting ensembles outperform state-of-the-art deep ensembles in terms of uncertainty estimation and robustness to dataset shift. Our work highlights the benefit of ensembling varying architectures. In future work, we aim to address the limitation posed by the computational cost of NES due to building the pool $\mathcal{P}$. An interesting approach in this direction could be the use of differentiable NAS methods to simultaneously optimize base learner architectures within a one-shot model and reduce cost [Liu et al., 2019a, Xu et al., 2020, Chen et al., 2021]. More generally, we also hope to explore what other hyperparameters can be varied to improve ensemble performance and how best to select them.

Our work can readily be applied to many existing ensemble-based deep learning systems to improve their predictive performance. This work also focuses on improving uncertainty estimation in neural networks, which is a key problem of growing importance with implications for safe deployment of such systems. We are not aware of any direct negative societal impacts of our work, since NES is task-agnostic and its impact depends on its applications.

6.7 Supplementary Material

6.7.1 Proof of Proposition 4

Taking the loss function to be NLL, we have $\ell(f(\boldsymbol{x}), y)) = -\log [f(\boldsymbol{x})]_y$, where $[f(\boldsymbol{x})]_y$ is the probability assigned by the network f of $\boldsymbol{x}$ belonging to the true class y, i.e. indexing the predicted probabilities $f(\boldsymbol{x})$ with the true target y. Note that $t \mapsto -\log t$ is a convex and decreasing function.

We first prove $\ell(F_{\text{OE}}(\boldsymbol{x}), y) \leq \ell(F(\boldsymbol{x}), y)$. Recall, by definition of F_{OE}, we have $F_{\text{OE}}(\boldsymbol{x}) = f_{\theta_k}(\boldsymbol{x})$ where $k \in \arg\min_i \ell(f_{\theta_i}(\boldsymbol{x}), y)$, therefore $[F_{\text{OE}}(\boldsymbol{x})]_y = [f_{\theta_k}(\boldsymbol{x})]_y \geq [f_{\theta_i}(\boldsymbol{x})]_y$ for all $i = 1, \ldots, C$. That is, f_{θ_k} assigns the highest probability to the correct class y for input $\boldsymbol{x}$. Since $-\log$ is a decreasing function, we have

$$\begin{aligned}\ell(F(\boldsymbol{x}), y) &= -\log\left(\frac{1}{M}\sum_{i=1}^{M}[f_{\theta_i}(\boldsymbol{x})]_y\right) \\ &\geq -\log\left([f_{\theta_k}(\boldsymbol{x})]_y\right) = \ell(F_{\text{OE}}(\boldsymbol{x}), y).\end{aligned}$$

We apply Jensen's inequality in its finite form for the second inequality. Jensen's inequality states that for a real-valued, convex function φ with its domain being a subset of $\mathbb{R}$ and numbers $t_1, \ldots, t_n$ in its domain, $\varphi(\frac{1}{n}\sum_{i=1}^{n} t_i) \leq \frac{1}{n}\sum_{i=1}^{n}\varphi(t_i)$. Noting that $-\log$ is a convex function, $\ell(F(\boldsymbol{x}), y) \leq \frac{1}{M}\sum_{i=1}^{M}\ell(f_{\theta_i}(\boldsymbol{x}), y)$ follows directly.

6.7.2 Experimental and Implementation Details

We describe details of the experiments shown in Section 6.5 and Appendix 6.7.3 and include Algorithms 4 and 5 describing NES-RS and `ForwardSelect`, respectively. Note that unless stated otherwise, all sampling over a discrete set is done uniformly in the discussion below.

Algorithm 4: NES with Random Search

Input: Search space $\mathcal{A}$; ensemble size M; comp. budget K; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$.

1 Sample K architectures $\alpha_1, \ldots, \alpha_K$ independently and uniformly from $\mathcal{A}$.
2 Train each architecture α_i using $\mathcal{D}_{\text{train}}$, yielding a pool of networks $\mathcal{P} = \{f_{\theta_1,\alpha_1}, \ldots, f_{\theta_K,\alpha_K}\}$.
3 Select base learners $\{f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*}\} = \texttt{ForwardSelect}(\mathcal{P}, \mathcal{D}_{\text{val}}, M)$ by forward step-wise selection without replacement.
4 **return** ensemble $\texttt{Ensemble}(f_{\theta_1^*,\alpha_1^*}, \ldots, f_{\theta_M^*,\alpha_M^*})$

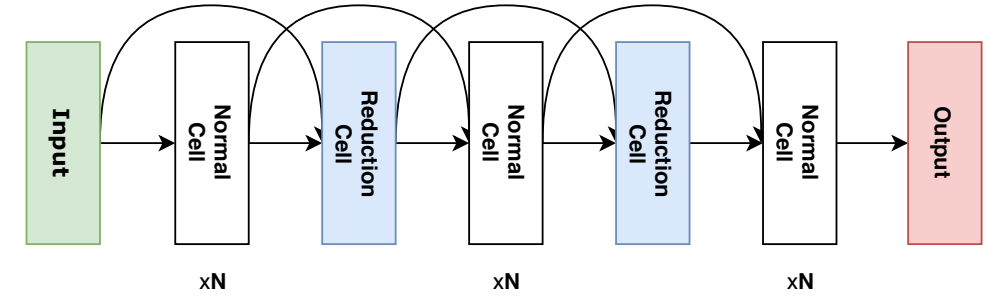
Algorithm 5: `ForwardSelect` (forward step-wise selection without replacement [Caruana et al., 2004])

Input: Pool of base learners $\mathcal{P}$; ensemble size M; $\mathcal{D}_{\text{val}}$. Assume $|\mathcal{P}| \geq M$.

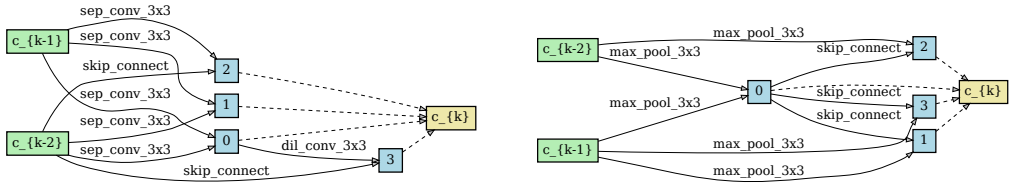
Initialize an empty set of base learners $E = \{\}$.
while $|E| < M$ **do**
  Add $f_{\theta,\alpha}$ to E, where $f_{\theta,\alpha} \in \arg\min_{f \in \mathcal{P}} \mathcal{L}(\texttt{Ensemble}(E \cup \{f\}), \mathcal{D}_{\text{val}})$
  Remove $f_{\theta,\alpha}$ from $\mathcal{P}$. ▷ `without replacement`
end
return E ▷ `set of selected base learners`

6.7.2.1 Architecture Search Spaces

DARTS search space. The first architecture search space we consider in our experiments is the one from DARTS [Liu et al., 2019a]. We search for two types of *cells*: *normal* cells, which preserve the spatial dimensions, and *reduction* cells, which reduce the spatial dimensions. See Figure 6.11b for an illustration. The cells are stacked in a macro architecture, where they are repeated and connected using skip connections (shown in Figure 6.11a). Each cell is a directed acyclic graph, where nodes represent feature maps in the computational graph and edges between them correspond to operation choices (e.g. a convolution operation). The cell parses inputs from the previous and previous-previous cells in its 2 input nodes. Afterwards it contains 5 nodes: 4 intermediate nodes that aggregate the information coming from 2 previous nodes in the cell and finally an output node that concatenates the output of all intermediate nodes across the channel dimension. AmoebaNet contains one more intermediate node, making that a deeper architecture. The set of possible operations (eight in total in DARTS) that we use for each edge in the cells is the same as DARTS, but we leave out the "zero" operation since that is not necessary for non-differentiable approaches such as random search and evolution. Specifically, this leaves us with the following set of seven operations: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 average pooling, 3×3 max pooling and identity. Randomly of architectures is done by sampling the structure of the cell and the operations at each edge. The total number of architectures contained in this space is $\approx 10^{18}$. We refer the reader to Liu et al. [2019a] for more details.



(a) Macro architecture.



(b) Normal (left) and reduction (right) cells.

Figure 6.11: Illustration of the DARTS search space: (a) The macro architecture is a stack of normal and reduction cells. Normal cells are repeated N times between reduction cells ($N = 2$ in our experiments, i.e. 8 cells in total). (b) The cells for the DARTS architecture are depicted as directed acyclic graphs. Briefly, cells are composed of 2 input nodes (green), 4 intermediate nodes (blue) and one output node (red). Each edge is an operation which is applied to the preceding node's tensor and the output is summed element-wise in the succeeding node (an intermediate node). The output node is a concatenation of all intermediate nodes.

NAS-Bench-201 search space. NAS-Bench-201 [Dong and Yang, 2020] is a tabular NAS benchmark, i.e. all architectures in the cell search space are trained and evaluated beforehand so one can query their performance (and weights) from a table quickly. Since this space is exhaustively evaluated, its size is also limited to only *normal* cells containing 4 nodes in total (1 input, 2 intermediate and 1 output node) and 5 operation choices on every edge connecting two nodes. This means that there are only 15,625 possible architecture configurations in this space. The networks are constructed by stacking 5 cells with in-between fixed residual blocks for reducing the spacial resolution. Each of them is trained for 200 epochs 3 times with 3 different seeds on 3 image classification datasets. For more details, please refer to Dong and Yang [2020].

6.7.2.2 Datasets

Fashion-MNIST [Xiao et al., 2017]. Fashion-MNIST consists of a training set of 60k 28×28 grayscale images and a test set of 10k images. The number of total labels is 10 classes. We split the 60k training set images to 50k used to train the networks and 10k used only for validation.

CIFAR-10/100 [Krizhevsky et al., 2009]. CIFAR-10 and CIFAR-100 both consist of 60k 32×32 colour images with 10 and 100 classes, respectively. We use 10k of the 60k training images as the validation set. We use the 10k original test set for final evaluation.

Tiny ImageNet [Le and Yang, 2015]. Tiny Imagenet has 200 classes and each class has 500 training, 50 validation and 50 test colour images with 64×64 resolution. Since the original test labels are not available, we split the 10k validation examples into 5k for testing and 5k for validation.

ImageNet-16-120 [Dong and Yang, 2020] This variant of the ImageNet-16-120 [Chrabaszcz et al., 2017] contains 151.7k train, 3k validation and 3k test ImageNet images downsampled to 16×16 and 120 classes.

Note that the test data points are only used for final evaluation. The data points for validation are used by the NES algorithms and DeepEns + ES baselines during ensemble selection and by DeepEns (RS) for picking the best architecture from the pool to use in the deep ensemble. Note that when considering dataset shift for CIFAR-10, CIFAR-100 and Tiny ImageNet, we also apply two disjoint sets of "corruptions" (following the terminology used by Hendrycks and Dietterich [2019]) to the validation and test sets. We never apply any corruption to the training data. More specifically, out of the 19 different corruptions provided by Hendrycks and Dietterich [2019], we randomly apply one from {`Speckle Noise`, `Gaussian Blur`, `Spatter`, `Saturate`} to each data point in the validation set and one from {`Gaussian Noise`, `Shot Noise`, `Impulse Noise`, `Defocus Blur`,

`Glass Blur`, `Motion Blur`, `Zoom Blur`, `Snow`, `Frost`, `Fog`, `Brightness`, `Contrast`, `Elastic Transform`, `Pixelate`, `JPEG compression`} to each data point in the test set. This choice of validation and test corruptions follows the recommendation of Hendrycks and Dietterich [2019]. Also, as mentioned in Section 6.5, each of these corruptions has 5 severity levels, which yields 5 corresponding severity levels for $\mathcal{D}_{\text{val}}^{\text{shift}}$ and $\mathcal{D}_{\text{test}}^{\text{shift}}$.

6.7.2.3 Training Routine & Time

The macro-architecture we use has 16 initial channels and 8 cells (6 normal and 2 reduction) and was trained using a batch size of 100 for 100 epochs for CIFAR-10/100 and 15 epochs for Fashion-MNIST. For Tiny ImageNet, we used 36 initial channels and a batch size of 128 for 100 epochs. Training a single network took roughly 40 minutes and 3 GPU hours[3] for CIFAR-10/100 and Tiny ImageNet, respectively. The networks are optimized using SGD with momentum set to 0.9. We used a learning rate of 0.025 for CIFAR-10/100 and Fashion-MNIST, and 0.1 for Tiny ImageNet. Unlike DARTS, we do not use any data augmentation procedure during training, nor any additional regularization such as ScheduledDropPath [Zoph et al., 2018] or auxiliary heads, except for the case of Tiny ImageNet, for which we used ScheduledDropPath, gradient clipping and standard data augmentation as default in DARTS. All other hyperparameter settings are exactly as in DARTS [Liu et al., 2019a].

All results containing error bars are averaged over multiple runs (at least five) with error bars indicating a 95% confidence interval. We used a budget $K = 400$ for CIFAR-10/100 (corresponding to 267 GPU hours). For Tiny ImageNet on the DARTS search space, unless otherwise stated, we used $K = 200$ (corresponding to 600 GPU hours). Note that only in Figure 6.7 and Table 6.30 we also tried NES algorithms using a higher budget of $K = 400$ on Tiny ImageNet for an equal-cost comparison with DeepEns + ES (RS). For ImageNet-16-120 on the NAS-Bench-201 search space, we used $K = 1000$ (compute costs are negligible since it is a tabular

[3]We used NVIDIA RTX 2080Ti GPUs for training.

benchmark). Note that NES algorithms can be parallelized, especially NES-RS which is embarrassingly parallel like deep ensembles, therefore training time can be reduced easily when using multiple GPUs.

6.7.2.4 Implementation Details of NES-RE

Parallization. Running NES-RE on a single GPU requires evaluating hundreds of networks sequentially, which is tedious. To circumvent this, we distribute the "while $|\mathcal{P}| < K$" loop in Algorithm 3 over multiple GPUs, called worker nodes. We use the parallelism scheme provided by the `hpbandster` [Falkner et al., 2018] codebase.[4] In brief, the master node keeps track of the population and history (lines 1, 4-6, 8 in Algorithm 3), and it distributes the training of the networks to the individual worker nodes (lines 2, 7 in Algorithm 3). In our experiments, we always use 20 worker nodes and evolve a population $\mathfrak{p}$ of size $P = 50$ when working over the DARTS search space. Over NAS-Bench-201, we used one worker since it is a tabular NAS benchmark and hence is quick to evaluate on. During iterations of evolution, we use an ensemble size of $m = 10$ to select parent candidates.

Mutations. We adapt the mutations used in RE to the DARTS search space. As in RE, we first pick a normal or reduction cell at random to mutate and then sample one of the following mutations:

- `identity`: no mutation is applied to the cell.
- `op mutation`: sample one edge in the cell and replace its operation with another operation sampled from the list of operations described in Appendix 6.7.2.1.
- `hidden state mutation`: sample one intermediate node in the cell, then sample one of its two incoming edges. Replace the input node of that edge with another sampled node, without altering the edge's operation.

[4] `https://github.com/automl/HpBandSter`

For example, one possible mutation would be changing the operation on an edge in the cell (as shown in e.g. Figure 6.16) from say 5×5 separable convolution to 3×3 max pooling. See Real et al. [2019] for further details and illustrations of these mutations. Note that for NAS-Bench-201, following Dong and Yang [2020] we only use `op mutation`.

Adaptation of NES-RE to dataset shifts. As described in Section 6.4.3, at each iteration of evolution, the validation set used in line 4 of Algorithm 3 is sampled uniformly between $\mathcal{D}_{\text{val}}$ and $\mathcal{D}_{\text{val}}^{\text{shift}}$ when dealing with dataset shift. In this case, we use shift severity level 5 for $\mathcal{D}_{\text{val}}^{\text{shift}}$. Once the evolution is complete and the pool $\mathcal{P}$ has been formed, then for each severity level $s \in \{0, 1, \ldots, 5\}$, we apply `ForwardSelect` with $\mathcal{D}_{\text{val}}^{\text{shift}}$ of severity s to select an ensemble from $\mathcal{P}$ (line 9 in Algorithm 3), which is then evaluated on $\mathcal{D}_{\text{test}}^{\text{shift}}$ of severity s. (Here $s = 0$ corresponds to no shift.) This only applies to CIFAR-10, CIFAR-100 and Tiny ImageNet, as we do not consider dataset shift for Fashion-MNIST and ImageNet-16-120 .

6.7.2.5 Implementation details of anchored ensembles.

Anchored ensembles [Pearce et al., 2020] are constructed by independently training M base learners, each regularized towards a new initialization sample (called the *anchor point*). Specifically, the k-th base learner f_{θ_k} is trained to minimize the following loss (using the notation in Section 6.3.1):

$$\frac{1}{N}\sum_{i=1}^{N} \ell(f_{\theta_k}(\boldsymbol{x}_i), y_i) + \frac{\lambda}{N}\|\Gamma^{1/2}(\theta_k - \mathring{\theta}_k)\|_2^2$$

where $\ell(f_{\theta_k}(\boldsymbol{x}_i), y_i) = -\log\left[f_{\theta_k}(\boldsymbol{x}_i)\right]_{y_i}$ is the cross-entropy loss for the i-th datapoint. Moreover, the anchor point $\mathring{\theta}_k$ is an independently sampled initialization, and λ is the regularization strength. Defining p to be the number of parameters, i.e. $\theta \in \mathbb{R}^p$, and letting σ_i^2 be the variance of the initialization for the i-th parameter $(\theta_k)_i$, $\Gamma \in \mathbb{R}^{p \times p}$ is the diagonal matrix with $\Gamma_{ii} = 1/(2\sigma_i^2)$.

Although Pearce et al. [2020] do not include a tune-able regularization parameter λ (they set $\lambda = 1$), we found anchored ensembles performed poorly without tuning λ. For CIFAR-10/100, we used $\lambda = 0.4$ and for Tiny ImageNet, we used $\lambda = 0.1$

(tuned by grid search). We followed the recommendation of Pearce et al. [2020] in using a different initialization for optimization than the anchor point $\mathring{\theta}_k$. Moreover, we turned off weight decay when training anchored ensembles, and we did not apply the regularization to the trainable parameters in batch normalization layers (which are initialized deterministically).

6.7.3 Additional Experiments

In this section we provide additional results for the experiments conducted in Section 6.5. Note that, as with all results shown in Section 6.5, all evaluations are made on test data unless stated otherwise.

6.7.3.1 Results on Fashion-MNIST

As shown in Figure 6.12, we see a similar trend on Fashion-MNIST as with other datasets: NES ensembles outperform deep ensembles with NES-RE outperforming NES-RS. To understand why NES algorithms outperform deep ensembles on Fashion-MNIST [Xiao et al., 2017], we compare the average base learner loss (Figure 6.13) and oracle ensemble loss (Figure 6.14) of NES-RS, NES-RE and DeepEns (RS). Notice that, apart from the case when ensemble size $M = 30$, NES-RS and NES-RE find ensembles with both stronger and more diverse base learners (smaller losses in Figures 6.13 and 6.14, respectively). While it is expected that the oracle ensemble loss is smaller for NES-RS and NES-RE compared to DeepEns (RS), it initially appears surprising that DeepEns (RS) has a larger average base learner loss considering that the architecture for the deep ensemble is chosen to minimize the base learner loss. We found that this is due to the loss having a sensitive dependence not only on the architecture but also the initialization of the base learner networks. Therefore, re-training the best architecture by validation loss to build the deep ensemble yields base learners with higher losses due to the use of different random initializations. Fortunately, NES algorithms are not affected by this, since they simply select the ensemble's base learners from the pool without having to re-train anything which allows them to exploit good architectures as

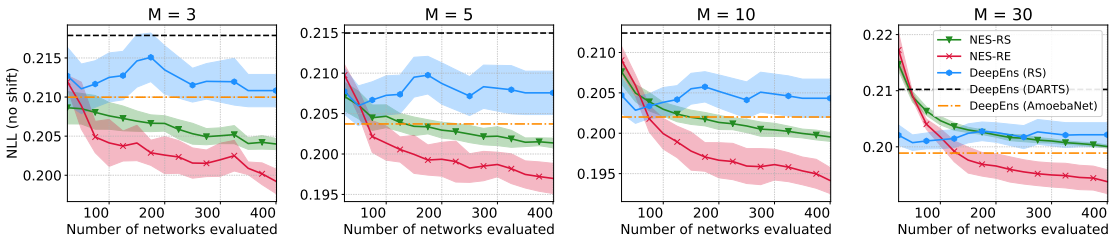


Figure 6.12: Results on Fashion-MNIST with varying ensembles sizes M. Lines show the mean NLL achieved by the ensembles with 95% confidence intervals.

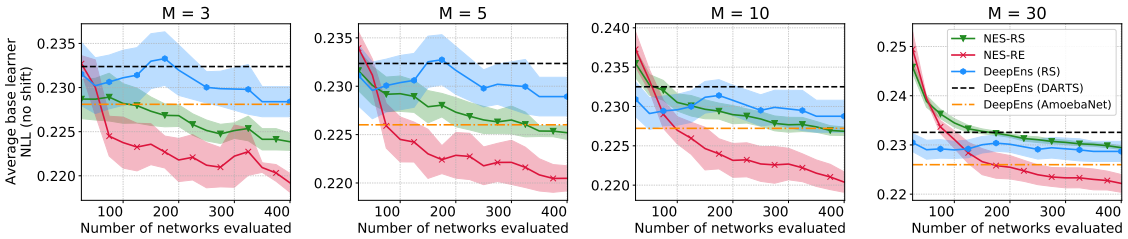


Figure 6.13: Average base learner loss for NES-RS, NES-RE and DeepEns (RS) on Fashion-MNIST. Lines show the mean NLL and 95% confidence intervals.

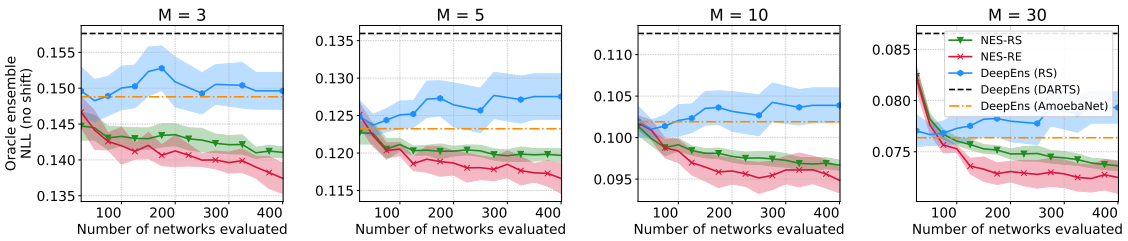


Figure 6.14: Oracle ensemble loss for NES-RS, NES-RE and DeepEns (RS) on Fashion-MNIST. Lines show the mean NLL and 95% confidence intervals.

well as initializations. Note that, for CIFAR-10-C experiments, this was not the case; base learner losses did not have as sensitive a dependence on the initialization as they did on the architecture.

In Table 6.3, we compare the classification error and expected calibration error (ECE) of NES algorithms with the deep ensembles baseline for various ensemble sizes on Fashion-MNIST. Similar to the loss, NES algorithms also achieve smaller errors, while ECE remains approximately the same for all methods.

6.7.3.2 Entropy on out-of-distribution inputs

To assess how well models respond to completely out-of-distribution (OOD) inputs (inputs which do not belong to one of the classes the model can predict), we

Table 6.3: Error and ECE of ensembles on Fashion-MNIST for different ensemble sizes M. Best values and all values within 95% confidence interval are bold faced.

M	Classification Error (%)					Expected Calibration Error (ECE)				
	NES-RS	NES-RE	DeepEns (RS)	DeepEns (DARTS)	DeepEns (AmoebaNet)	NES-RS	NES-RE	DeepEns (RS)	DeepEns (DARTS)	DeepEns (AmoebaNet)
3	$7.4_{\pm 0.1}$	$\mathbf{7.2}_{\pm 0.1}$	$7.6_{\pm 0.1}$	7.7	7.7	$0.007_{\pm 0.001}$	$0.007_{\pm 0.002}$	$0.008_{\pm 0.001}$	**0.003**	0.008
5	$\mathbf{7.3}_{\pm 0.1}$	$\mathbf{7.1}_{\pm 0.2}$	$7.5_{\pm 0.1}$	7.7	7.4	$\mathbf{0.005}_{\pm 0.001}$	$\mathbf{0.005}_{\pm 0.001}$	$\mathbf{0.006}_{\pm 0.001}$	**0.005**	**0.005**
10	$7.3_{\pm 0.1}$	$\mathbf{7.0}_{\pm 0.1}$	$7.5_{\pm 0.1}$	7.6	7.3	$\mathbf{0.004}_{\pm 0.001}$	$\mathbf{0.005}_{\pm 0.001}$	$\mathbf{0.005}_{\pm 0.001}$	0.006	**0.005**
30	$7.3_{\pm 0.1}$	$\mathbf{7.0}_{\pm 0.1}$	$7.4_{\pm 0.1}$	7.5	7.3	$\mathbf{0.004}_{\pm 0.001}$	$\mathbf{0.004}_{\pm 0.002}$	$\mathbf{0.004}_{\pm 0.001}$	0.008	**0.004**

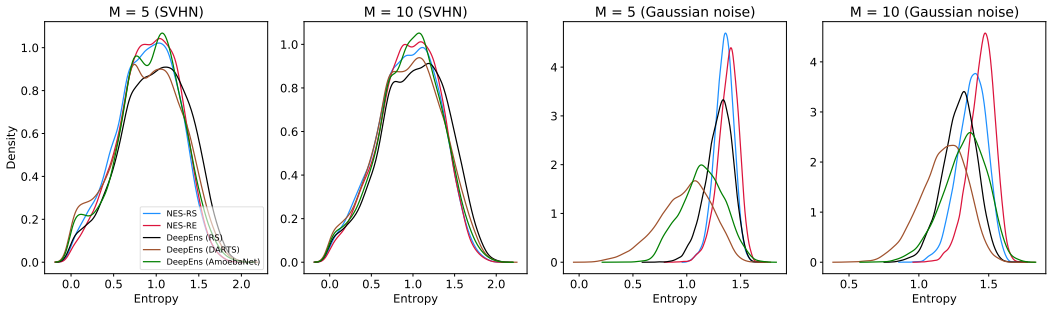


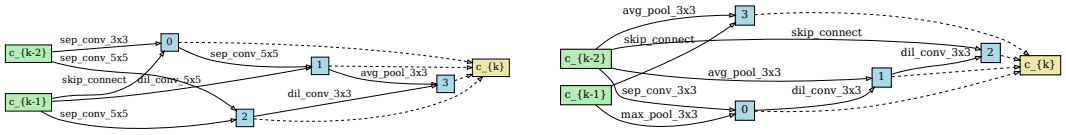
Figure 6.15: Entropy of predicted probabilities when trained on CIFAR-10 in the DARTS search space.

investigate the entropy of the predicted probability distribution over the classes when the input is OOD. Higher entropy of the predicted probabilities indicates more uncertainty in the model's output. For CIFAR-10 on the DARTS search space, we compare the entropy of the predictions made by NES ensembles with deep ensembles on two types of OOD inputs: images from the SVHN dataset and Gaussian noise. In Figure 6.15, we notice that NES ensembles indicate higher uncertainty when given inputs of Gaussian noise than deep ensembles but behave similarly to deep ensembles for inputs from SVHN.

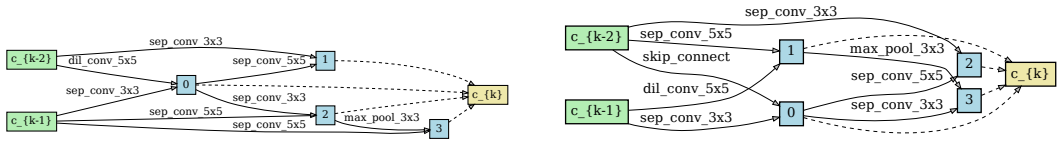
6.7.3.3 Additional Results on the DARTS search space

In this section, we provide additional experimental results on CIFAR-10, CIFAR-100 and Tiny ImageNet on the DARTS search space, complementing the results in Section 6.5 as shown in Figures 6.17-6.23. We also include examples of architectures chosen by NES-RE in Figure 6.16 for Tiny ImageNet, showcasing variation in architectures.

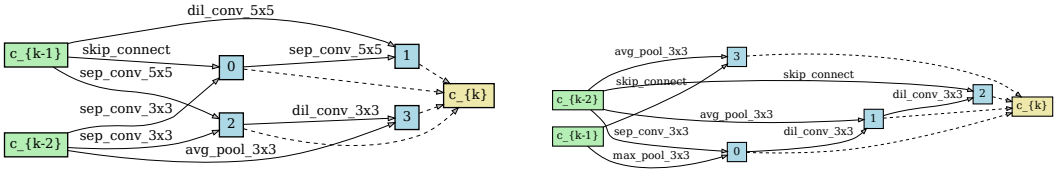
Results on CIFAR for larger models. In additional to the results on CIFAR-10 and CIFAR-100 on the DARTS search space using the settings described in Appendix 6.7.2.3, we also train larger models (around 3M parameters) by scaling up the number of stacks cells and initial channels in the network. We run NES and other baselines similarly as done before and plot results in Figure 6.27 and 6.28 for NLL and classification test error with budget $K = 90$. As shown, NES algorithms tend to outperform or be competitive with the baselines. Note, more runs are needed including error bars for conclusive results in this case.



(a) First base learner

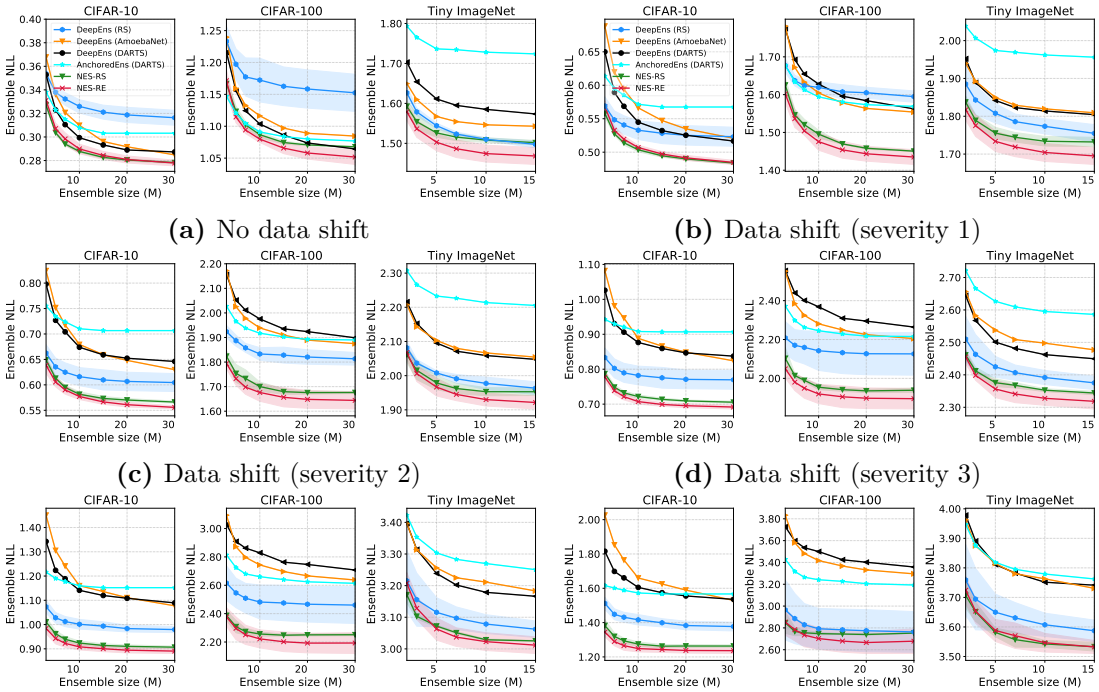


(b) Second base learner



(c) Third base learner

Figure 6.16: Illustration of example cells found by NES-RE for an ensemble of size $M = 3$ on Tiny ImageNet over the DARTS search space. In each sub-figure, left is the normal cell and right is the reduction cell. Note that the first and third base learners have the same reduction cell in this instance.

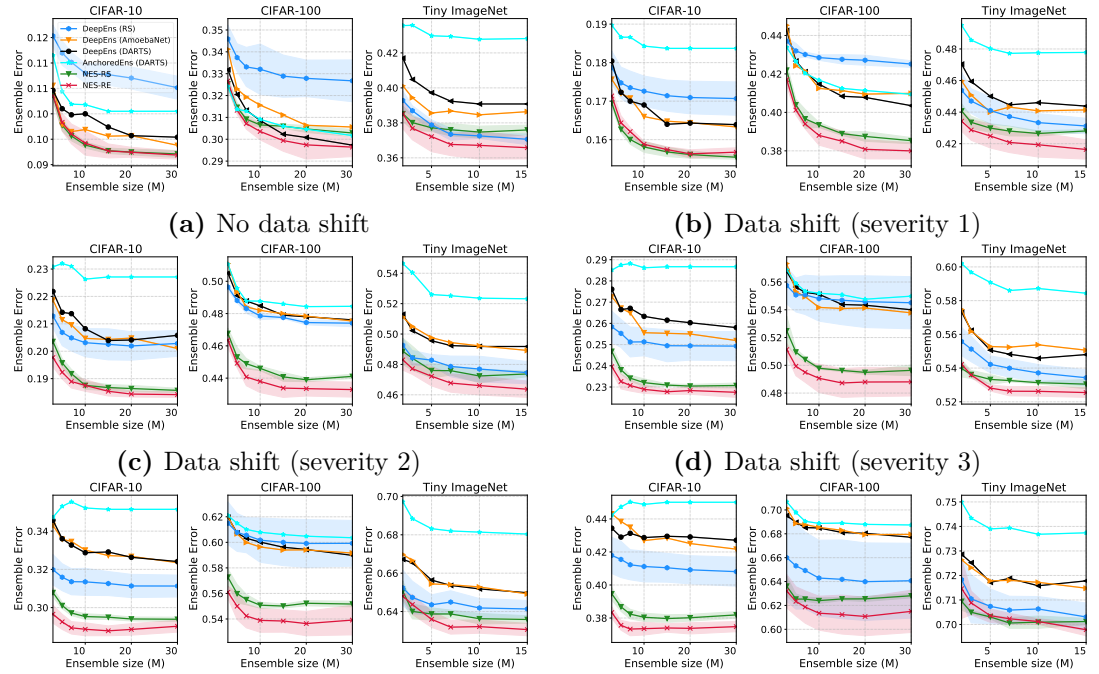


(a) No data shift **(b)** Data shift (severity 1)

(c) Data shift (severity 2) **(d)** Data shift (severity 3)

(e) Data shift (severity 4) **(f)** Data shift (severity 5)

Figure 6.17: NLL vs. ensemble sizes on CIFAR-10, CIFAR-100 and Tiny ImageNet with varying dataset shifts [Hendrycks and Dietterich, 2019] over DARTS space.

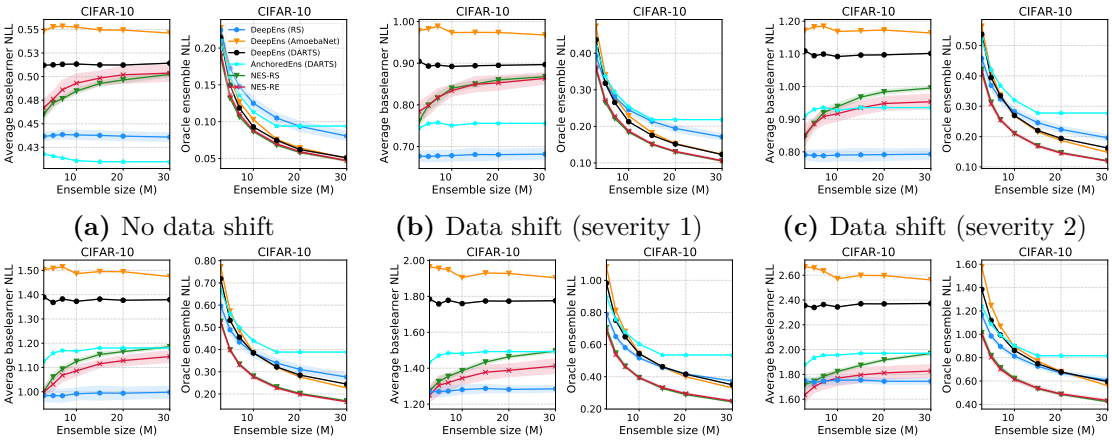


(a) No data shift **(b)** Data shift (severity 1)

(c) Data shift (severity 2) **(d)** Data shift (severity 3)

(e) Data shift (severity 4) **(f)** Data shift (severity 5)

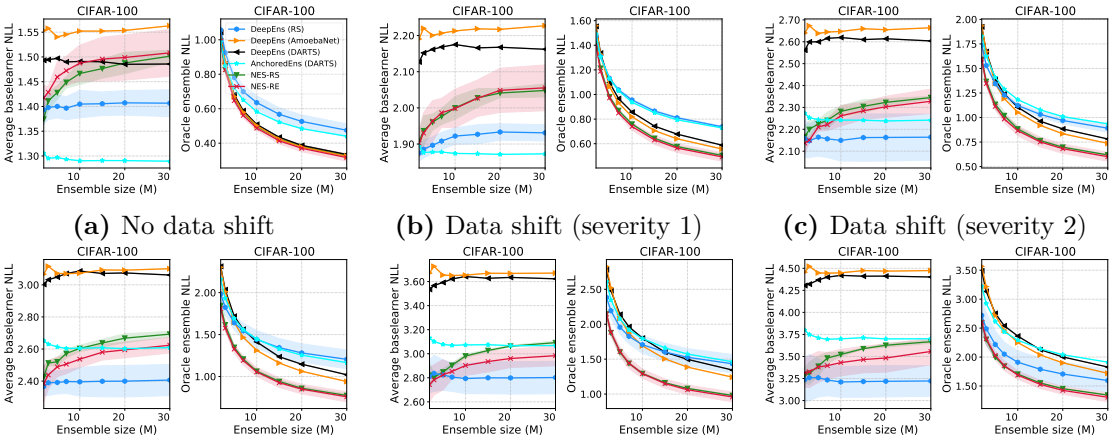
Figure 6.18: Classification error rate (between 0-1) vs. ensemble size on DARTS search space.



(a) No data shift **(b)** Data shift (severity 1) **(c)** Data shift (severity 2)

(d) Data shift (severity 3) **(e)** Data shift (severity 4) **(f)** Data shift (severity 5)

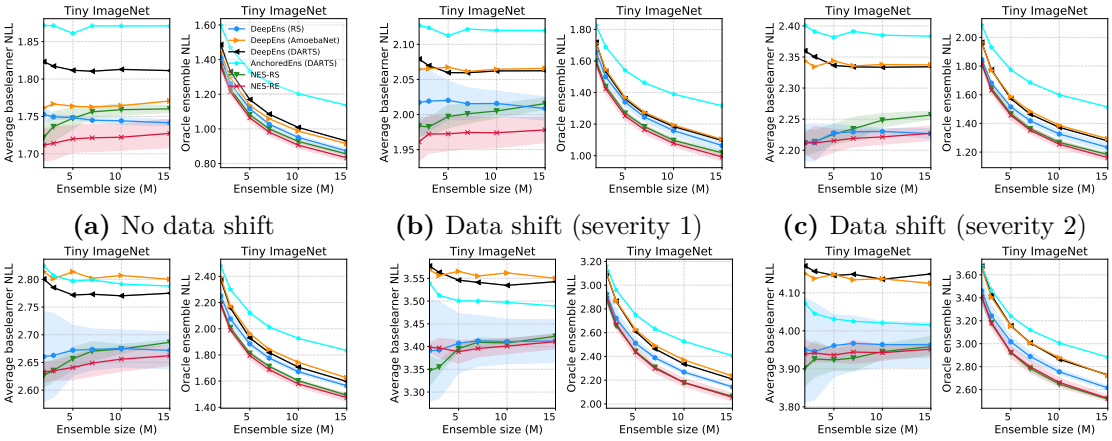
Figure 6.19: Average base learner and oracle ensemble NLL across ensemble sizes and shift severities on CIFAR-10 over DARTS search space.



(a) No data shift **(b)** Data shift (severity 1) **(c)** Data shift (severity 2)

(d) Data shift (severity 3) **(e)** Data shift (severity 4) **(f)** Data shift (severity 5)

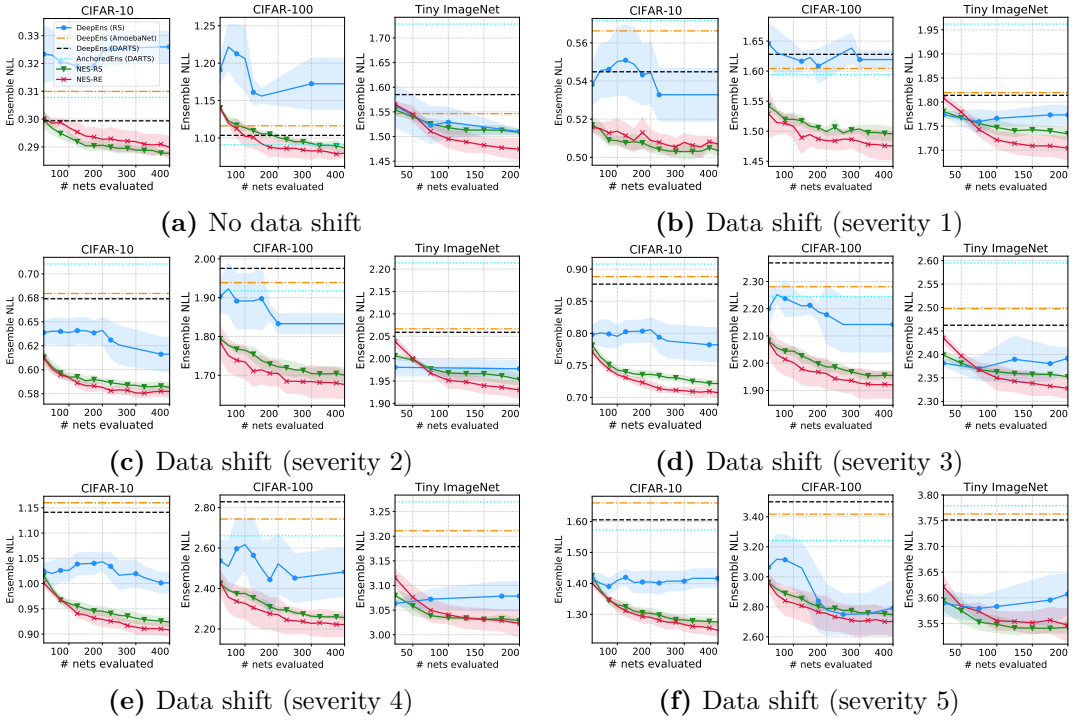
Figure 6.20: Average base learner and oracle ensemble NLL across ensemble sizes and shift severities on CIFAR-100 over DARTS search space.



(a) No data shift **(b)** Data shift (severity 1) **(c)** Data shift (severity 2)

(d) Data shift (severity 3) **(e)** Data shift (severity 4) **(f)** Data shift (severity 5)

Figure 6.21: Average base learner and oracle ensemble NLL across ensemble sizes and shift severities on Tiny ImageNet over DARTS search space.

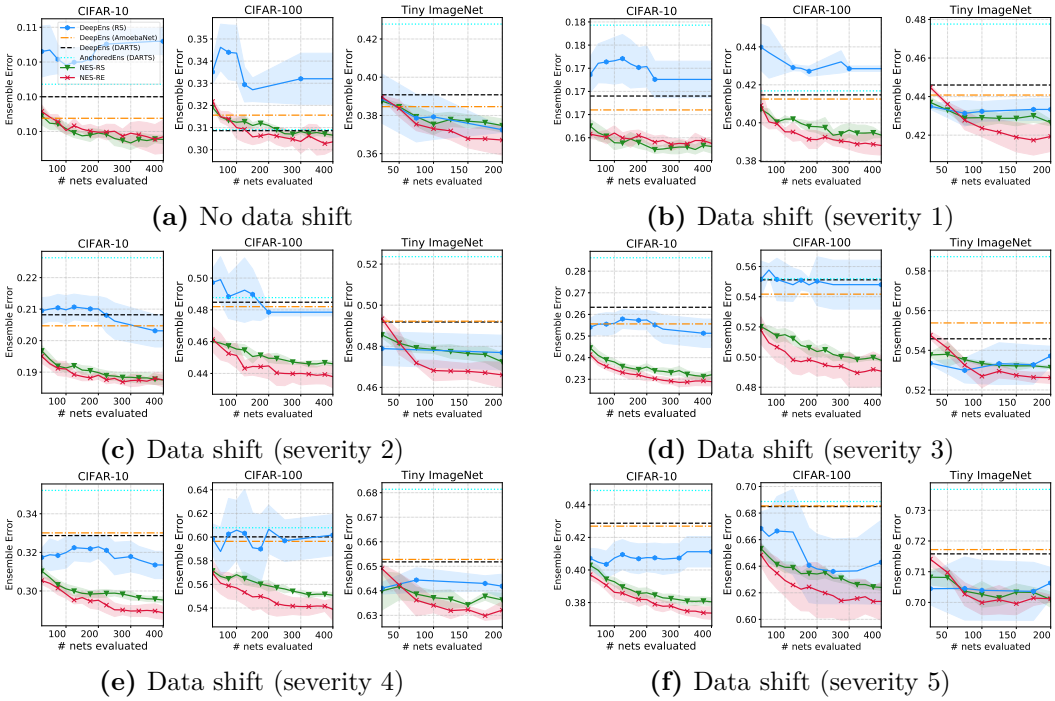


(a) No data shift **(b)** Data shift (severity 1)

(c) Data shift (severity 2) **(d)** Data shift (severity 3)

(e) Data shift (severity 4) **(f)** Data shift (severity 5)

Figure 6.22: Ensemble NLL vs. budget K. Ensemble size fixed at $M = 10$.

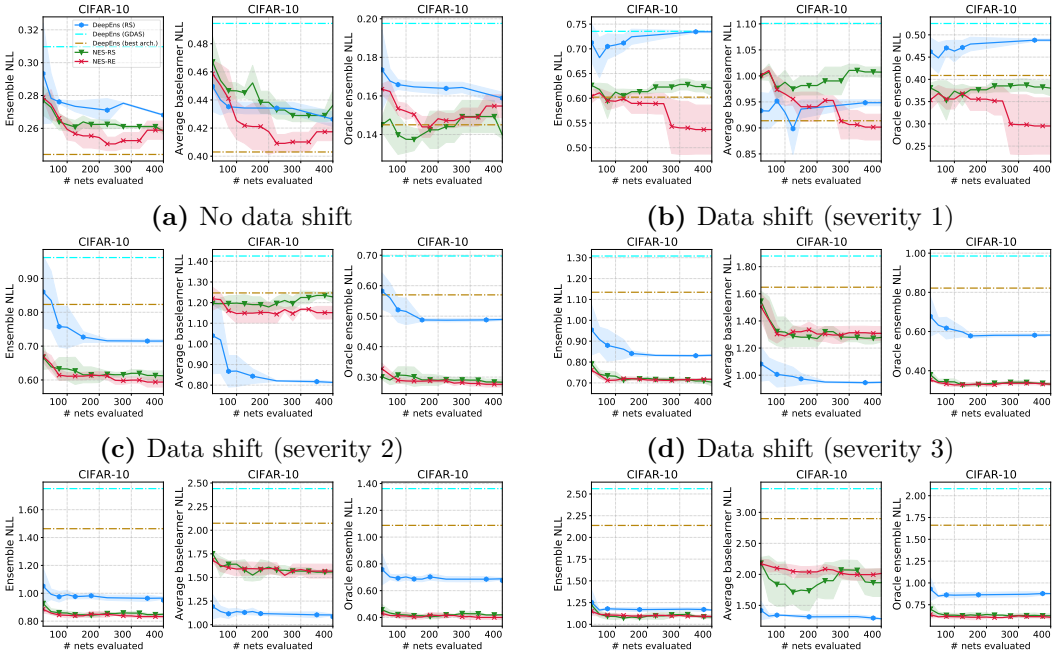


(a) No data shift **(b)** Data shift (severity 1)

(c) Data shift (severity 2) **(d)** Data shift (severity 3)

(e) Data shift (severity 4) **(f)** Data shift (severity 5)

Figure 6.23: Ensemble error vs. budget K. Ensemble size fixed at $M = 10$.

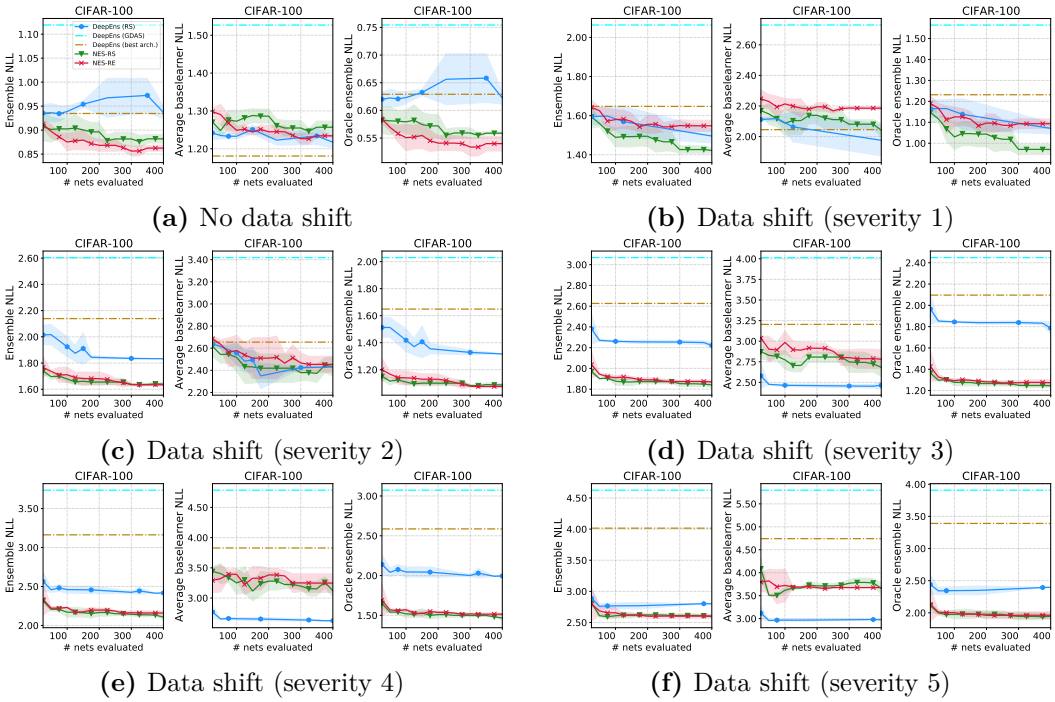


(a) No data shift **(b)** Data shift (severity 1)

(c) Data shift (severity 2) **(d)** Data shift (severity 3)

(e) Data shift (severity 4) **(f)** Data shift (severity 5)

Figure 6.24: NES and deep ensembles performance on the NAS-Bench-201 search space and CIFAR-10 dataset. Plots show ensemble NLL, average baselearner NLL and oracle ensemble NLL vs. budget K. Ensemble size fixed at $M = 3$.

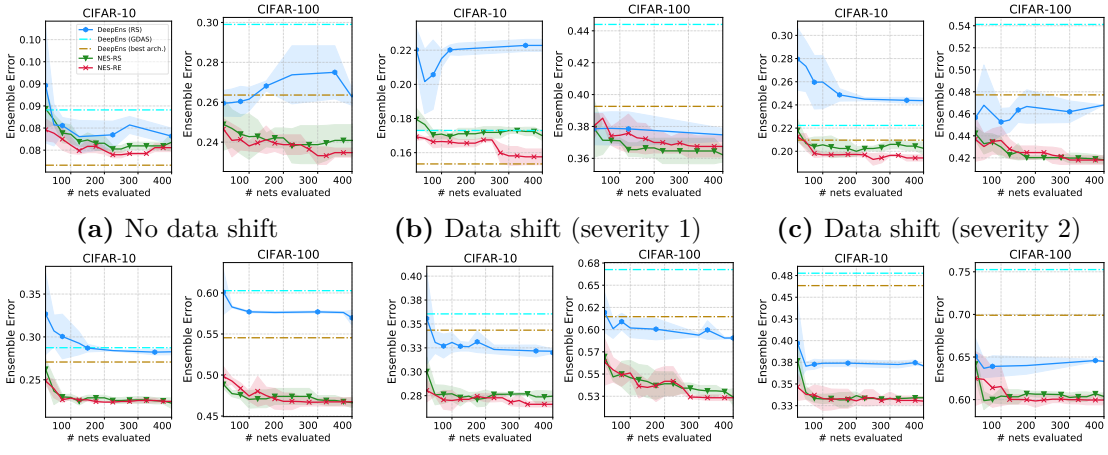


(a) No data shift **(b)** Data shift (severity 1)

(c) Data shift (severity 2) **(d)** Data shift (severity 3)

(e) Data shift (severity 4) **(f)** Data shift (severity 5)

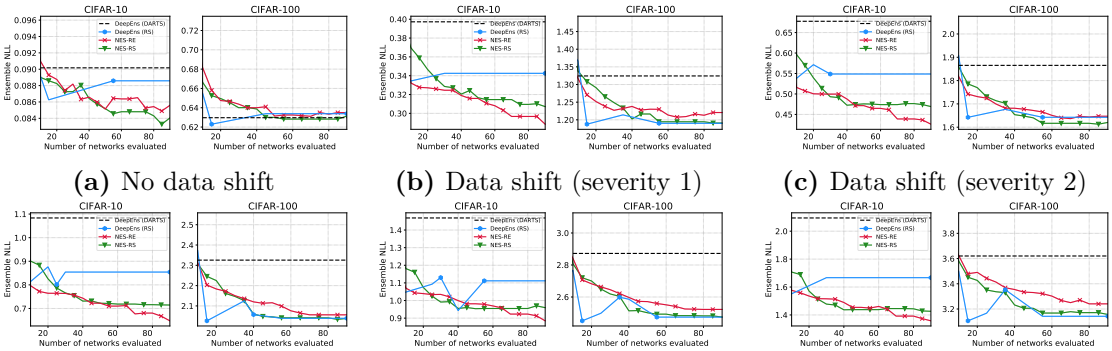
Figure 6.25: CIFAR-100 analogous to Figure 6.24.



(a) No data shift **(b)** Data shift (severity 1) **(c)** Data shift (severity 2)

(d) Data shift (severity 3) **(e)** Data shift (severity 4) **(f)** Data shift (severity 5)

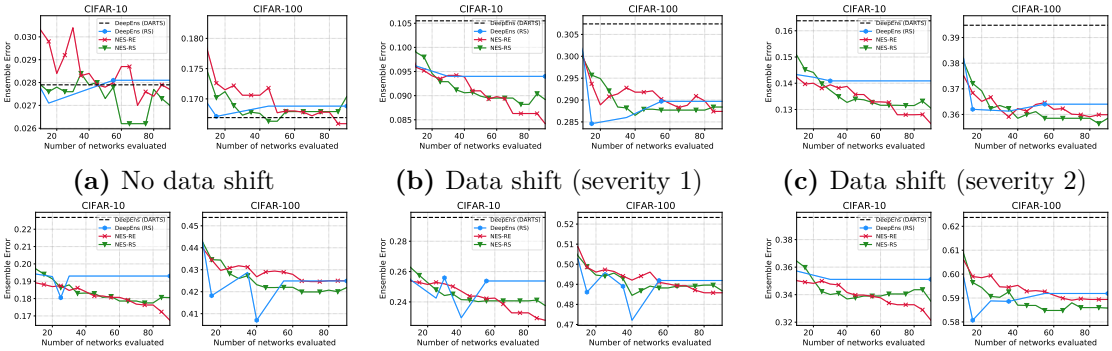
Figure 6.26: Ensemble error vs. budget K on the NAS-Bench-201 space (CIFAR-10 and CIFAR-100). Ensemble size fixed at $M = 3$.



(a) No data shift **(b)** Data shift (severity 1) **(c)** Data shift (severity 2)

(d) Data shift (severity 3) **(e)** Data shift (severity 4) **(f)** Data shift (severity 5)

Figure 6.27: High fidelity NLL vs. budget K on CIFAR-10 and CIFAR-100 with and without respective dataset shifts over the DARTS search space. Ensemble size is fixed at $M = 10$.



(a) No data shift **(b)** Data shift (severity 1) **(c)** Data shift (severity 2)

(d) Data shift (severity 3) **(e)** Data shift (severity 4) **(f)** Data shift (severity 5)

Figure 6.28: High fidelity classification error vs. budget K on CIFAR-10 and CIFAR-100 with and without respective dataset shifts over the DARTS search space. Ensemble size is fixed at $M = 10$.

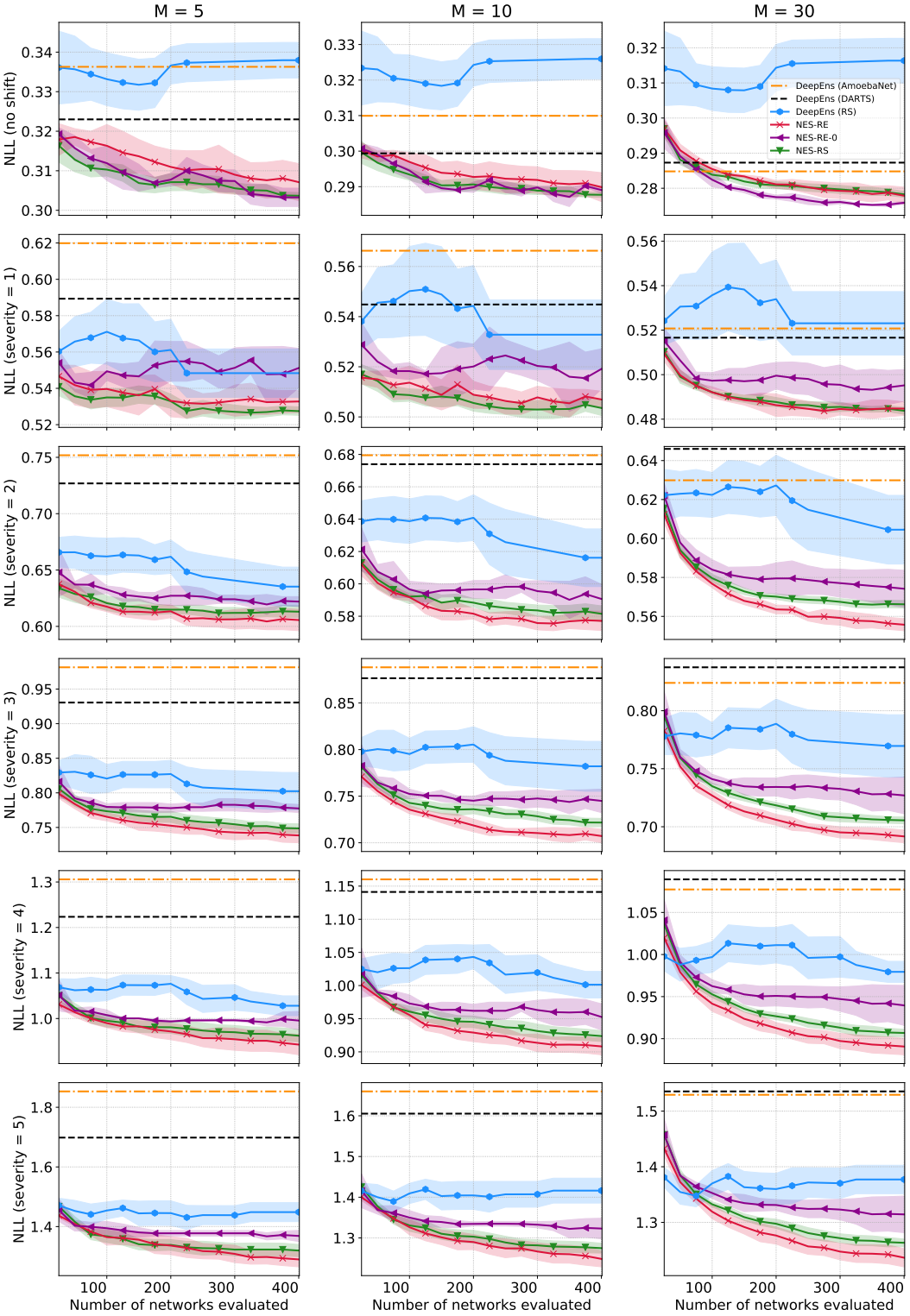


Figure 6.29: Results on CIFAR-10 [Hendrycks and Dietterich, 2019] with varying ensembles sizes M and shift severity. Lines show the mean NLL achieved by the ensembles with 95% confidence intervals. See Appendix 6.7.3.3 for the definition of NES-RE-0.

6.7.3.4 Additional Results on the NAS-Bench-201 search space

In this section, we provide additional experimental results on CIFAR-10, CIFAR-100 and ImageNet-16-120 on the NAS-Bench-201 search space, complementing the results in Section 6.5 as shown in Figures 6.24-6.26.

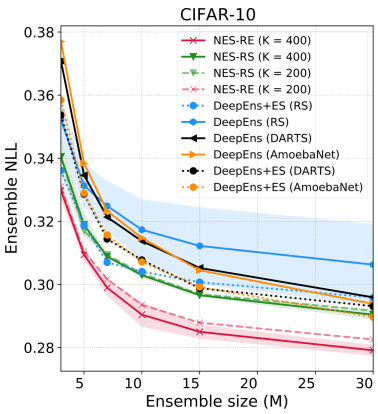
6.7.3.5 Ablation study: NES-RE optimizing only on $\mathcal{D}_{\text{val}}$

We also include a variant of NES-RE, called NES-RE-0, in Figure 6.29. NES-RE and NES-RE-0 are the same, except that NES-RE-0 uses the validation set $\mathcal{D}_{\text{val}}$ without any shift during iterations of evolution, as in line 4 of Algorithm 3. Following the discussion in Appendix 6.7.2.4, recall that this is unlike NES-RE, where we sample the validation set to be either $\mathcal{D}_{\text{val}}$ or $\mathcal{D}_{\text{val}}^{\text{shift}}$ at each iteration of evolution. Therefore, NES-RE-0 evolves the population without taking into account dataset shift, with $\mathcal{D}_{\text{val}}^{\text{shift}}$ only being used for the post-hoc ensemble selection step in line 9 of Algorithm 3.

As shown in the Figure 6.29, NES-RE-0 shows a minor improvement over NES-RE in terms of loss for ensemble size $M = 30$ in the absence of dataset shift. This is in line with expectations, because evolution in NES-RE-0 focuses on finding base learners which form strong ensembles for in-distribution data. On the other hand, when there is dataset shift, the performance of NES-RE-0 ensembles degrades, yielding higher loss and error than both NES-RS and NES-RE. Nonetheless, NES-RE-0 still manages to outperform the DeepEns baselines consistently. We draw two conclusions on the basis of these results: (1) NES-RE-0 can be a competitive option in the absence of dataset shift. (2) Sampling the validation set, as done in NES-RE, to be $\mathcal{D}_{\text{val}}$ or $\mathcal{D}_{\text{val}}^{\text{shift}}$ in line 4 of Algorithm 3 plays an important role is returning a final pool $\mathcal{P}$ of base learners from which `ForwardSelect` can select ensembles robust to dataset shift.

6.7.3.6 What if deep ensembles use ensemble selection over initializations?

We provide additional experimental results in this section for comparing NES to deep ensembles with ensemble selection building on those shown in Figure 6.7 in Section 6.5. The table in Figure 6.30-right compares the methods with respect to



Tiny ImageNet

Method	No dataset shift		Dataset shift: severity 5	
	NLL	**Error (%)**	**NLL**	**Error (%)**
DeepEns (DARTS)	1.59	39.08	3.75	71.58
DeepEns + ES (DARTS)	1.57	38.68	3.68	70.90
DeepEns (AmoebaNet)	1.55	38.46	3.76	71.72
DeepEns + ES (AmoebaNet)	1.54	38.12	3.70	71.68
DeepEns (RS)	$1.51_{\pm 0.00}$	$37.46_{\pm 0.27}$	$3.60_{\pm 0.03}$	$70.48_{\pm 0.38}$
DeepEns + ES (RS)	1.50	36.98	3.55	**70.10**
NES-RS ($K = 200$)	$1.51_{\pm 0.01}$	$37.42_{\pm 0.21}$	$3.53_{\pm 0.01}$	$\mathbf{69.93}_{\pm 0.23}$
NES-RS ($K = 400$)	$1.48_{\pm 0.01}$	$37.45_{\pm 0.25}$	$\mathbf{3.51}_{\pm 0.01}$	$\mathbf{70.15}_{\pm 0.34}$
NES-RE ($K = 200$)	$1.48_{\pm 0.01}$	$36.98_{\pm 0.57}$	$3.55_{\pm 0.02}$	$70.22_{\pm 0.13}$
NES-RE ($K = 400$)	$\mathbf{1.44}_{\pm 0.02}$	$\mathbf{36.55}_{\pm 0.17}$	$3.54_{\pm 0.02}$	$\mathbf{69.98}_{\pm 0.24}$

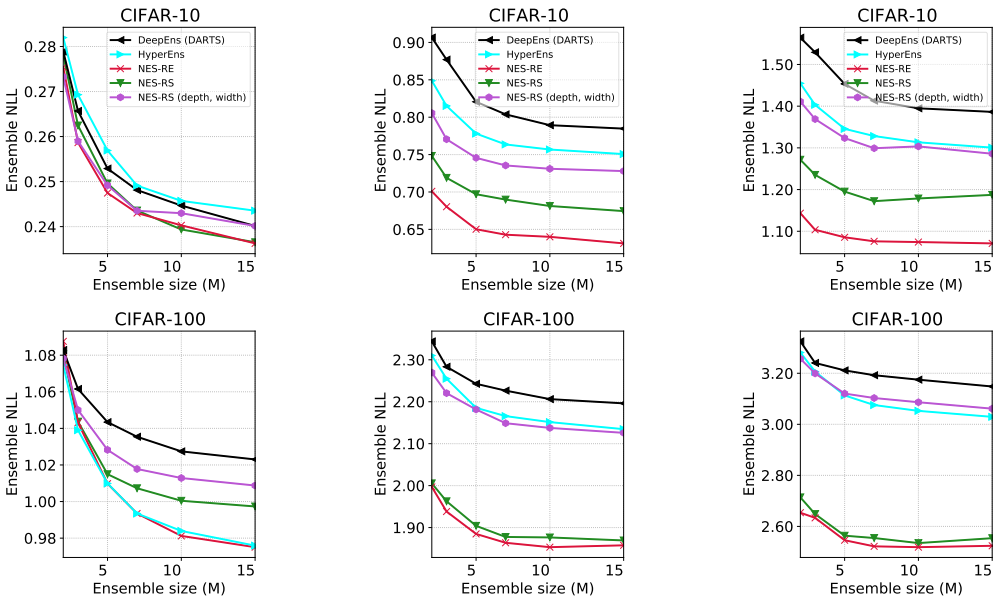
Figure 6.30: Comparison of NES to deep ensembles with ensemble selection over initializations over the DARTS search space. LEFT: NLL vs ensemble size on CIFAR-10 (similar to ablation in Figure 6.7). RIGHT: Test NLL and classification error NES vs. deep ensembles on Tiny ImageNet with ensemble size $M = 10$.

classification error as well as loss at different dataset shift severities. In the absence of any dataset shift, we find that NES-RE outperforms all baselines with respect to both metrics, specially at the higher budget of $K = 400$ and NES-RS performs competitively. At shift severity 5, NES algorithms also outperform the baselines, with NES-RS performing slightly better than NES-RE. The plot in Figure 6.30-left shows similar results on CIFAR-10 over the DARTS search space for the same experiment as the one conducted on Tiny ImageNet and shown in Figure 6.7.

6.7.3.7 Comparing NES to ensembles with other varying hyperparameters

Since varying the architecture in an ensemble improves predictive performance and uncertainty estimation as demonstrated in Section 6.5, it is natural to ask what other hyperparameters should be varied in an ensemble. It is also unclear which hyperparameters might be more important than others. Note that concurrent work by Wenzel et al. [2020] has shown that varying hyperparameters such as L_2 regularization strength, dropout rate and label smoothing parameter also improves upon deep ensembles. While these questions lie outside the scope of our work and are left for future work, we conduct preliminary experiments to address them.

In this section, we consider two additional baselines working over the DARTS search space on CIFAR-10/100:



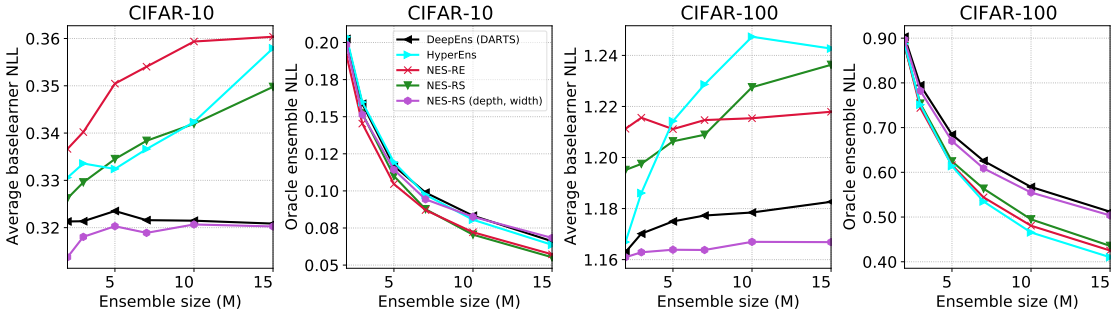
(a) No data shift **(b)** Data shift (severity 3) **(c)** Data shift (severity 5)

Figure 6.31: Plots show NLL vs. ensemble sizes comparing NES to the baselines introduced in Appendix 6.7.3.7 on CIFAR-10 and CIFAR-100 with and without respective dataset shifts [Hendrycks and Dietterich, 2019].

1. **HyperEns:** Optimize a fixed architecture, train K random initializations of it *where the learning rate and L_2 regularization strength are also sampled randomly* and select the final ensemble of size M from the pool using `ForwardSelect`. This is similar to `hyper ens` from Wenzel et al. [2020].

2. **NES-RS (depth, width):** As described in Appendix 6.7.2.1, NES navigates a complex (non-Euclidean) search space of architectures by varying the cell, which involves changing both the DAG structure of the cell and the operations at each edge of the DAG. We consider a baseline in which we keep the cell fixed (the optimized DARTS cell) and only vary the width and depth of the overall architecture. More specifically, we vary the number of *initial channels* $\in \{12, 14, 16, 18, 20\}$ (width) and the number of *layers* $\in \{5, 8, 11\}$ (depth). We apply NES-RS over this substantially simpler search space of architectures as usual: train K randomly sampled architectures (i.e. sampling only depth and width) to form a pool and select the ensemble from it.

The results shown in Figures 6.31 and Table 6.32b compare the two baselines



(a) Average base learner loss and oracle ensemble loss for NES and the baselines introduced in Appendix 6.7.3.7 on CIFAR-10 and CIFAR-100. Recall that small oracle ensemble loss generally corresponds to higher diversity.

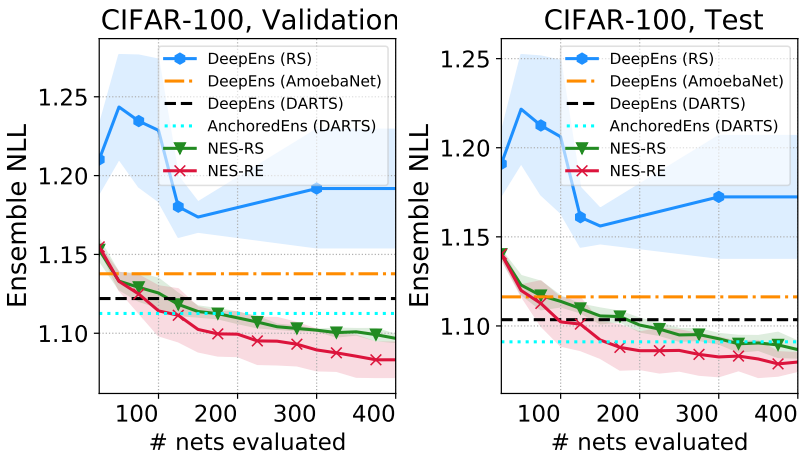
Dataset	Shift Severity	DARTS search space				
		DeepEns (DARTS)	HyperEns	NES-RS (depth, width)	NES-RS	NES-RE
C10	0	8.2	8.1	8.0	8.0	**7.7**
	3	25.9	25.0	24.1	22.5	**21.5**
	5	43.3	40.8	42.1	38.1	**34.9**
C100	0	28.8	**28.1**	28.8	28.4	28.4
	3	54.0	52.7	53.1	48.9	**48.5**
	5	68.4	67.2	67.9	61.3	**60.7**

(b) Classification errors comparing NES to the baselines introduced in Appendix 6.7.3.7 for different shift severities and $M = 10$. Best values are bold faced.

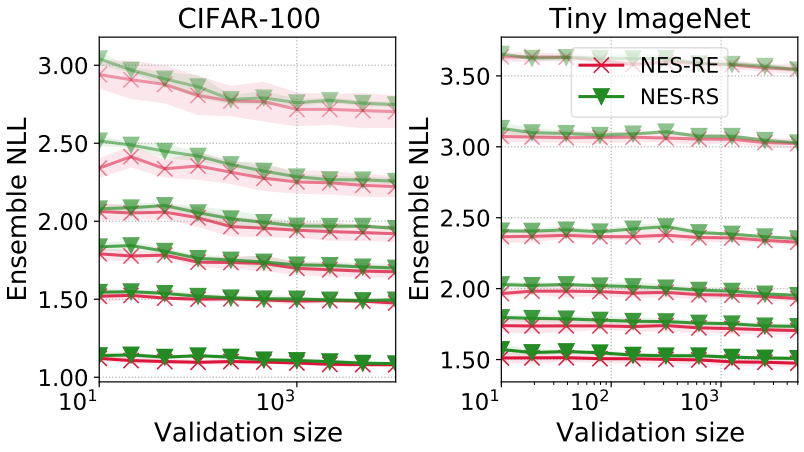
Figure 6.32: See Appendix 6.7.3.7 for details.

above to DeepEns (DARTS), NES-RS and NES-RE.[5] As shown in Figure 6.31, NES-RE tends to outperform the baselines, though is at par with HyperEns on CIFAR-100 without dataset shift (Figure 6.31a). Under the presence of dataset shift (Figures 6.31b and 6.31c), both NES algorithms substantially outperform all baselines. Note that both HyperEns and NES-RS (depth, width) follow the same protocol as NES-RS and NES-RE: ensemble selection uses a shifted validation dataset when evaluating on a shifted test dataset. In terms of classification error, the observations are similar as shown in Table 6.32b. Lastly, we view the diversity

[5]Note that runs of DeepEns (DARTS), NES-RE and NES-RS differ slightly in this section relative to Section 6.5, as we tune the learning rate and L_2 regularization strength for each dataset instead of using the defaults used in Liu et al. [2019a]. This yields a fair comparison: HyperEns varies the learning rate and L_2 regularization while using a fixed, optimized architecture (DARTS), whereas NES varies the architecture while using fixed, optimized learning rate and L_2 regularization strength.



(a) Ensemble loss over the validation and test datasets over time (i.e. with increasing budget K). These curves being similar indicates no overfitting to the validation dataset during ensemble selection.



(b) Test performance of NES algorithms with varying validation data sizes. Each curve corresponds to one particular dataset shift severity (0-5 with 0 being no shift), with more transparent curves corresponding to higher shift severities. The loss stays relatively constant even with 10× fewer validation data samples, indicating NES is not very sensitive to validation dataset size.

Figure 6.33: See Appendix 6.7.3.8 for details.

of the ensembles from the perspective of oracle ensemble loss in Figure 6.32a. As in Section 6.5, results here also suggest that NES agorithms tend to find more diverse ensembles despite having higher average base learner loss.

6.7.3.8 Sensitivity to the size of the validation dataset $\mathcal{D}_{\text{val}}$ and overfitting during ensemble selection

In this section, we consider how sensitive the performance of NES is to changes in the size of the validation dataset $\mathcal{D}_{\text{val}}$, and we discuss why overfitting to $\mathcal{D}_{\text{val}}$ is not a concern in our experiments. Recall that $\mathcal{D}_{\text{val}}$ is used by NES algorithms

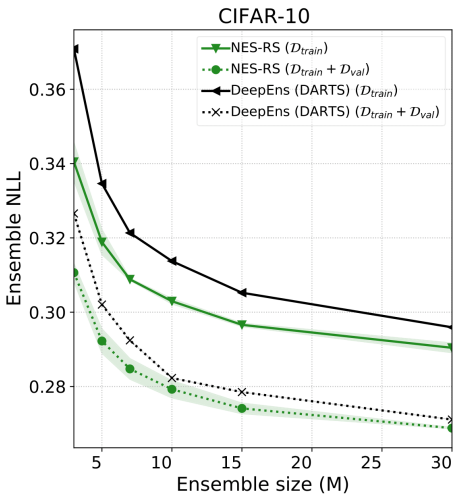


Figure 6.34: Re-training the selection architectures on $\mathcal{D}_{\text{train}} + \mathcal{D}_{\text{val}}$.

during ensemble selection from the pool of base learners as outlined in Algorithm 5, and we use $\mathcal{D}_{\text{val}}$ or $\mathcal{D}_{\text{val}}^{\text{shift}}$ depending on whether dataset shift is expected at test time as described in Section 6.4.3. Over the DARTS search space for CIFAR-100 and Tiny ImageNet, we measure the test loss of NES as a function of different validation dataset sizes. Specifically, we measure test loss of the ensembles selected using validation datasets of different sizes (with as few as 10 validation samples). Figure 6.33b shows this for different levels of dataset shift severity. The results indicate that NES is insensitive to the validation set size, achieving almost the same loss when using 10× fewer validation data.

NES also did not overfit to the validation data in our experiments. This can be seen in Figure 6.33a for CIFAR-100 over the DARTS search space, which shows that both the test and validation losses decrease over time (i.e. as the pool size K increases). This finding is consistent across datasets and search spaces in our experiments. We also remark that overfitting is unexpected, because ensemble selection "fits a small number of parameters" since it only selects which base learners are added to the ensemble.

6.7.3.9 Re-training the selected architectures on $\mathcal{D}_{\text{train}} + \mathcal{D}_{\text{val}}$

In practice, one might choose to re-trained the selected architectures from NES on the $\mathcal{D}_{\text{train}} + \mathcal{D}_{\text{val}}$ to make maximal use of the data available. We re-trained the

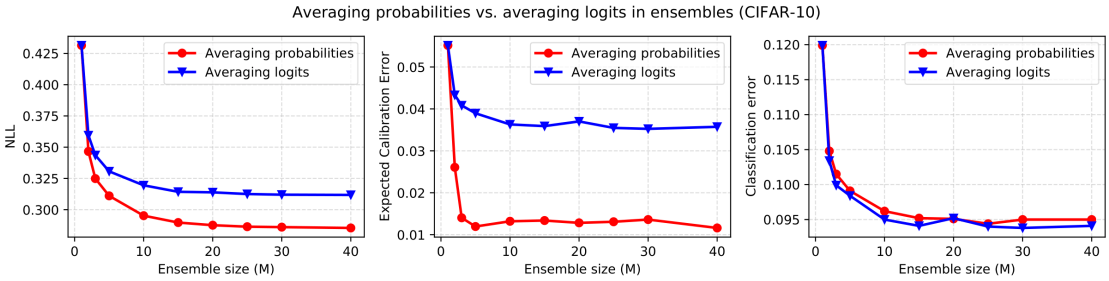


Figure 6.35: See Appendix 6.7.3.10 for details.

ensembles constructed by NES and the best performing deep ensemble baseline, DeepEns (DARTS), on $\mathcal{D}_{\text{train}} + \mathcal{D}_{\text{val}}$, finding that both ensembles improve since the base learners improve due to more training data (see Figure 6.34). Note that, as shown in Appendix 6.7.3.8, the performance of ensemble selection (which uses $\mathcal{D}_{\text{val}}$) is relatively insensitive to the size of $\mathcal{D}_{\text{val}}$. Therefore, one way to bypass the additional cost of having to re-train the ensembles on $\mathcal{D}_{\text{train}} + \mathcal{D}_{\text{val}}$ is to simply pick a very small $\mathcal{D}_{\text{val}}$, such that the performance of the models when trained on $\mathcal{D}_{\text{train}} + \mathcal{D}_{\text{val}}$ is approximately the same as when trained on $\mathcal{D}_{\text{train}}$.

6.7.3.10 Averaging logits vs. averaging probabilities in an ensemble

In Figure 6.35, we explore whether ensembles should be constructed by averaging probabilities (i.e. post-softmax) as done in our work or by averaging the logits (i.e. pre-softmax). We find that while classification error of the resulting ensembles is similar, ensembles with averaged probabilities perform notably better in terms of uncertainty estimation (NLL) and are better calibrated as well (ECE).

6.7.3.11 Comparison of ensemble selection algorithms

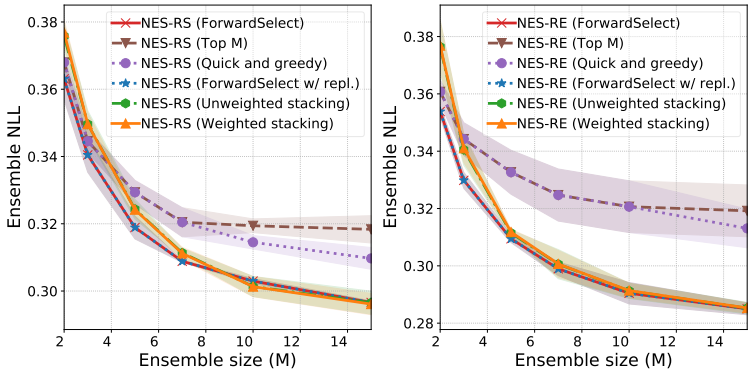
In Figure 6.8a, we compare the performance of NES for different choices of the ensemble selection algorithm (ESA) used in stage 2 (ensemble selection from pool). In addition to our default `ForwardSelect` without replacement, we experiment with the following seven choices:

- `TopM`: selects the top M (ensemble size) models by validation performance.

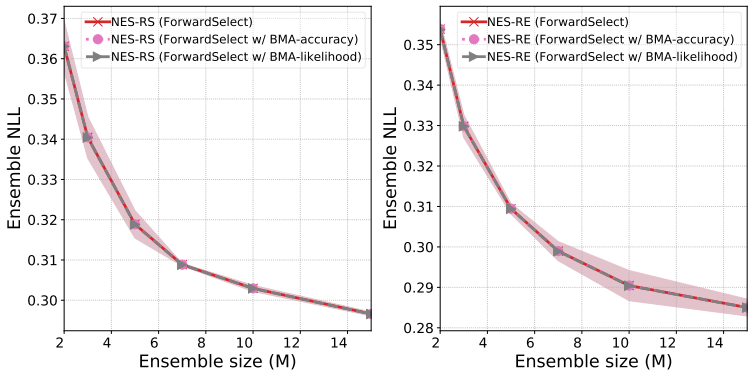
- `QuickAndGreedy`: Starting with the best network by validation performance, add the next best network to the ensemble only if it improves validation performance, iterating until the ensemble size is M or all models have been considered (returning an ensemble of size at most M).
- `ForwardSelect` with replacement: `ForwardSelect` but with replacement which allows repetitions of base learners.
- `Stacking` (Weighted): Linearly combines all base learners in the pool with learned stacking weights (which are positive and sum to 1). Then keeps only the base learners with the M largest stacking weights and weights them using the renormalized learned stacking weights.
- `Stacking` (Unweighted): Linearly combines all base learners in the pool with learned stacking weights (which are positive and sum to 1). Then keeps only the base learners with the M largest stacking weights and combines them by a simple, unweighted average.
- `ForwardSelect` with Bayesian model averaging by likelihood: Selects the base learners in the ensemble using `ForwardSelect` and then takes an average weighted by the (normalized) validation likelihoods of the base learners.
- `ForwardSelect` with Bayesian model averaging by accuracy: Selects the base learners in the ensemble using `ForwardSelect` and then takes an average weighted by the (normalized) validation accuracies of the base learners.

In Figure 6.36 we provide additional results including the one in Figure 6.8a. From the plots we can observe that:

- `ForwardSelect` performs better than or at par with all ESAs considered here.
- `Stacking` tends to perform competitively but still worse than `ForwardSelect`, and whether weighted averaging is used or not has a very minor impact on performance, since the learned weights tend to be close to uniform.



(a) Left: NES-RS. Right: NES-RE.



(b) Left: NES-RS. Right: NES-RE.

Figure 6.36: `ForwardSelect` compared to other ensemble selection algorithms. Experiments were conducted on CIFAR-10 over the DARTS search space. See Appendix 6.7.3.11 for details.

- Bayesian model averaging (BMA) also has a very minor impact (almost invisible in the plots in Figure 6.36b), because the likelihoods and accuracies of individual base learners are very similar (a consequence of multi-modal loss landscapes in neural networks with different models achieving similar losses) hence the BMA weights are close to uniform. The NLL achieved by the BMA ensemble only differed at the 4th or 5th decimal places compared to unweighted ensembles with the same base learners.

Lastly, we assess the impact of *explicitly* regularizing for diversity during ensemble selection as follows: we use `ForwardSelect` as before but instead use it to minimize the objective "validation loss - diversity strength $\times$ diversity" where diversity is defined as the average (across base learners and data points) L_2 distance between a base learner's predicted class probabilities and the ensemble's predicted class

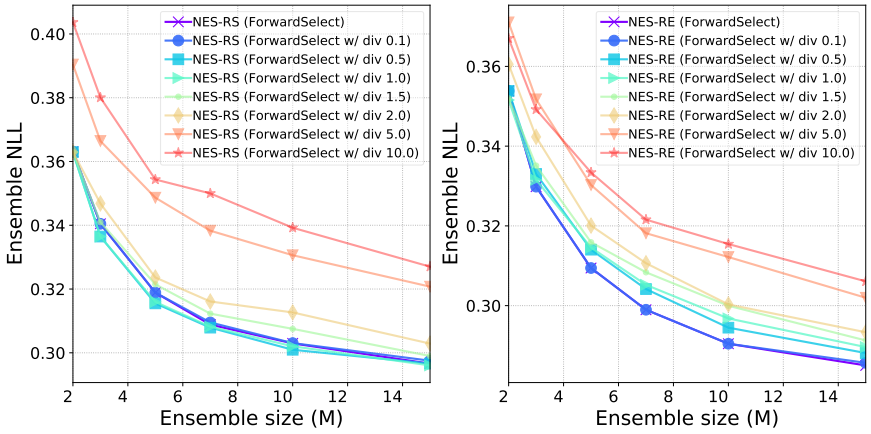


Figure 6.37: Ensemble selection with explicit diversity regularization. Experiments were conducted on CIFAR-10 over the DARTS search space.

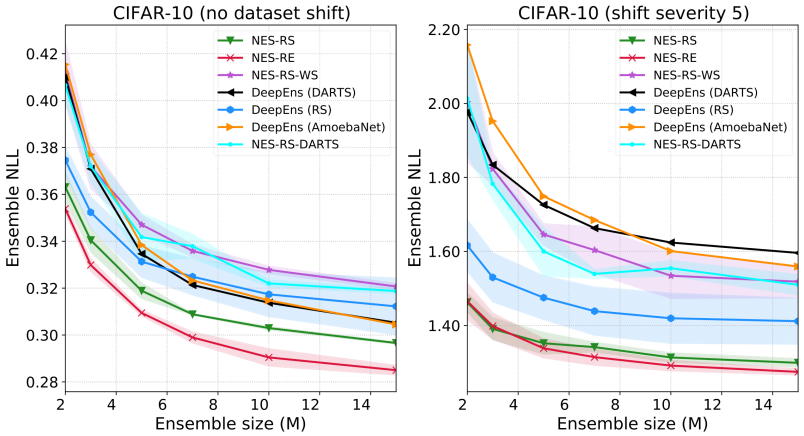


Figure 6.38: Comparison of NES-RS-WS and NES-RS-DARTS with the other baselines and NES-RS and NES-RE. The plots show the NLL vs. ensemble sizes on CIFAR-10 with and without dataset shifts over the DARTS search space. Mean NLL shown with 95 confidence intervals.

probabilities. The plots below show the results for different choices of the "diversity strength" hyperparameter. In summary, if appropriately tuned, `ForwardSelect` with diversity performs slightly better than usual `ForwardSelect` for NES-RS, though for NES-RE, the diversity term seems to harm performance as shown in Figure 6.37-right.

6.7.3.12 Generating the architecture pool using weight-sharing NAS algorithms

In order to accelerate the NES search phase, we generated the pool using the weight sharing schemes proposed by Random Search with Weight Sharing [Li and Talwalkar,

2019] and DARTS [Liu et al., 2019a]. Specifically, we trained one-shot weight-sharing models using each of these two algorithms, then we sampled architectures from the weight-shared models uniformly at random to build the pool. Next, we ran `ForwardSelect` as usual to select ensembles from these pools. Finally, we retrained the selected base learners from scratch using the same training pipeline as NES and the deep ensemble baselines. In Figure 6.38, we refer to the results of these methods as NES-RS-WS and NES-RS-DARTS, respectively. As we can see, their performance is worse than NES-RS and NES-RE, even though the computation cost is reduced significantly. This is likely due to the low correlation between the performance of the architectures when evaluated using the shared weights and the performance when re-trained in isolation. Prior work has shown this is caused by weight interference and co-adaptation which occurs during the weight-sharing model training [Yu et al., 2020, Zela et al., 2020]. Nonetheless, we believe this is a promising avenue for future research requiring more development, as it substantially reduces the cost of exploring the search space.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Neural Ensemble Search for Uncertainty Estimation and Dataset Shift
Publication Status	☒Published ☐ Accepted for Publication ☐Submitted for Publication ☐Unpublished and unsubmitted work written in a manuscript style
Publication Details	Sheheryar Zaidi*, Arber Zela*, Thomas Elsken, Chris C. Holmes, Frank Hutter, and Yee Whye Teh. "Neural Ensemble Search for Uncertainty Estimation and Dataset Shift." *Advances in Neural Information Processing Systems* 34 (2021): 7898-7911.

Student Confirmation

Student Name:	Sheheryar Zaidi		
Contribution to the Paper	☐ Yee Whye, Chris and I conceived the idea and initiated the project. ☐ I created the early versions of the NES algorithms in discussions with Yee Whye. I refined them to their current form after discussions with Arber. ☐ I designed the experiments with help from Arber and chose the metrics (e.g. oracle ensemble loss) to analyse the ensembles. ☐ Arber and I wrote the codebase, ran the experiments and produced the plots. ☐ I wrote the main paper and most of the appendix in consultation with Arber. ☐ All authors contributed to the development of the project through discussions and reviewed the final draft.		
Signature		Date	01/02/2023

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Yee Whye Teh		
Supervisor comments	Looks good to me.		
Signature		Date	31/10/2023

This completed form should be included in the thesis, at the end of the relevant chapter.

7
Conclusion

The recent rapid ascent of deep learning has led to neural networks powering a large and diverse range of systems in society today. Given that some of these systems are being likened to prototypes of artificial general intelligence, it is likely that neural networks will continue to become increasingly integrated with society. The utility of such systems is a function of models' abilities to generalize well, safely and robustly to unseen data, while also reliably indicating their uncertainty. This thesis contends that, methodologically, these properties are intrinsically shaped by a neural network's architecture and training procedure. Indeed, as we argued in Chapter 2, the architecture and its training procedure directly determine the model and its properties, *given* a fixed training dataset. We illustrated this proposition in four works aimed at understanding and improving generalization in neural networks, each examining the effect of a given aspect of a neural network's architecture or training procedure on generalization.

We began by considering the successful paradigm of pre-training in the context of molecular property prediction in Chapter 3. We argued that molecular property prediction from 3D structures is a key task where none of the best models before our work relied on pre-training, which is in strong contrast with the vision and language domains. We then devised a simple pre-training objective based on denoising unlabelled structures, which was shown to be theoretically linked to learning atomic forces via the framework of score-based modelling. The effectiveness

of our approach was demonstrated through improved generalization of models across various benchmarks, in some cases outperforming the state-of-the-art due to pre-training. One implication of this result is that denoising leads to learned representations that improve generalization on downstream tasks. For future work, this naturally raises the question of how representations learned by other applications of denoising, *e.g.* for diffusion models for image generation, can be further utilized downstream.

Next, in Chapter 4, we studied an optimization technique for training neural networks based on the counter-intuitive concept of re-initializing model parameters periodically. The learning dynamics induced by stochastic gradient descent algorithms are widely considered to play a crucial role in the generalization properties of models but are little understood. Our analysis was aimed at empirically uncovering the settings where re-initialization improves generalization. We explored the interplay between the use of re-initialization and other regularization techniques and discovered a surprising dependence of the effectiveness of re-initialization on the amount of label noise in the data distribution. Our work thus highlights how the interesting interaction between optimization and generalization in deep learning can be further complicated by the training data distribution.

In Chapter 5, we shifted our focus to architectures, in particular, studying equivariant architectures for application to tasks that have known symmetries. The principal example of such a task is modelling physical systems or molecules whose coordinate systems are arbitrary. Incorporating these symmetries into the neural network architecture makes generalization easier, and the resulting models become provably robust to the symmetries. We proposed a new architecture called LieTransformer which is an equivariant version of the prolific transformer architecture. We demonstrated its strong performance for modelling physical systems, point cloud classification and molecular property prediction. This work affirms the sensitive dependence of model generalization on architectural choices, particularly when they relate to inductive biases present in the data.

Continuing the study of architectures in Chapter 6, we considered generalization and predictive uncertainty estimation under distribution shifts, which is important for the safe and reliable deployment of models in the real world. Ensembles of neural networks are one of the dominant approaches for improving robustness to distribution shifts by exploiting the diversity amongst the ensemble's base learners. We proposed algorithms for constructing ensembles with varying architectures and showed that even small variations in the architectures lead to diversity in function space that yields ensembles that generalize more gracefully under distribution shifts. Our findings point once again to the impact of the architecture on the generalization of the resulting model. In particular, one key observation is that even when different architectures achieve similar test losses, the mistakes they make can differ. This has implications not just for ensembles but also sparse models such as mixture-of-experts that are extensively used for scaling language models.

Consistently in this thesis, we have seen how sometimes seemingly small details related to the architecture and training of a neural network have an outsized impact on the performance of the resulting model. Moreover, the space of possible choices that feed into designing and training these models is enormous, and a rigorous, theoretical understanding of the effects of these choices is currently limited at best, all the while that these models are becoming increasingly more capable and useful for society. This is what makes deep learning—and the careful investigation of how and why these details matter—exciting.

Bibliography

Ibrahim Alabdulmohsin, Hartmut Maennel, and Daniel Keysers. The Impact of Reinitialization on Generalization in Convolutional Neural Networks. *arXiv preprint arXiv:2109.00267*, 2021.

Brandon Anderson, Truong Son Hy, and Risi Kondor. Cormorant: Covariant Molecular Neural Networks. In *NeurIPS*, 2019.

Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

Jordan Ash and Ryan P Adams. On Warm-Starting Neural Network Training. In *Advances in Neural Information Processing Systems*, volume 33, pages 3884–3894. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/288cd2567953f06e460a33951f55daaf-Paper.pdf`.

Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of In-Domain Uncertainty Estimation and Ensembling in Deep Learning. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=BJxI5gHKDr`.

Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R Ledsam, Agnieszka Grabska-Barwinska, Kyle R Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature methods*, 18(10):1196–1203, 2021.

Jimmy Ba, J. Kiros, and Geoffrey E. Hinton. Layer Normalization. *ArXiv*, abs/1607.06450, 2016.

Robert John Nicholas Baldock, Hartmut Maennel, and Behnam Neyshabur. Deep Learning Through the Lens of Example Difficulty. In *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=WWRBHhH158K.

David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *International Conference on Machine Learning*, pages 342–350. PMLR, 2017.

Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-Normalized Margin Bounds for Neural Networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6241–6250, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Rodney J. Bartlett and Monika Musiał. Coupled-cluster theory in quantum chemistry. *Rev. Mod. Phys.*, 79:291–352, Feb 2007. URL https://link.aps.org/doi/10.1103/RevModPhys.79.291.

Ronen Basri, David W. Jacobs, Yoni Kasten, and Shira Kritchman. The Convergence Rate of Neural Networks for Learned Functions of Different Frequencies. In *Neural Information Processing Systems*, 2019. URL https://api.semanticscholar.org/CorpusID:173991140.

P. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and K. Kavukcuoglu. Interaction Networks for Learning about Objects, Relations and Physics. *ArXiv*, abs/1612.00222, 2016.

P. Battaglia, Jessica B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *ArXiv*, abs/1806.01261, 2018.

Simon Batzner, T. Smidt, L. Sun, J. Mailoa, M. Kornbluth, N. Molinari, and B. Kozinsky. SE(3)-Equivariant Graph Neural Networks for Data-Efficient and Accurate Interatomic Potentials. *ArXiv*, abs/2101.03164, 2021.

Erik J Bekkers. B-Spline CNNs on Lie groups. In *ICLR*, 2020.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and Simplifying One-Shot Architecture Search. In *International Conference on Machine Learning*, 2018.

Yijun Bian and Huanhuan Chen. When does Diversity Help Generalization in Classification Ensembles? *ArXiv*, abs/1903.06236, 2019.

Charles M. Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7:108–116, 1995.

Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetries and invariant neural networks. *Journal of Machine Learning Research*, 21(90):1–61, 2020.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Network. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1613–1622, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/blundell15.html`.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to End Learning for Self-Driving Cars. *CoRR*, abs/1604.07316, 2016.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik Bekkers, and Max Welling. Geometric and Physical Quantities improve E (3) Equivariant Message Passing. *arXiv preprint arXiv:2110.02905*, 2021.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001a.

Leo Breiman. Random Forests. In *Machine Learning*, pages 5–32, 2001b.

Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020a. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*, 2020b.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

Lucas Caccia, Jing Xu, Myle Ott, Marc'Aurelio Ranzato, and Ludovic Denoyer. On Anytime Learning at Macroscale. *arXiv preprint arXiv:2106.09563*, 2021.

Chen Cai and Yusu Wang. A Note on Over-Smoothing for Graph Neural Networks. *CoRR*, abs/2006.13318, 2020. URL `https://arxiv.org/abs/2006.13318`.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble Selection from Libraries of Models. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 18, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. URL `https://doi.org/10.1145/1015330.1015432`.

Lowik Chanussot*, Abhishek Das*, Siddharth Goyal*, Thibaut Lavril*, Muhammed Shuaibi*, Morgane Riviere, Kevin Tran, Javier Heras-Domingo, Caleb Ho, Weihua Hu, et al. Open Catalyst 2020 (OC20) Dataset and Community Challenges. *ACS Catalysis*, 2021.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *CoRR*, abs/1909.03211, 2019. URL `http://arxiv.org/abs/1909.03211`.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. DrNAS: Dirichlet Neural Architecture Search. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=9FWas6YbmB3`.

Lénaïc Chizat and Francis Bach. Implicit Bias of Gradient Descent for Wide Two-layer Neural Networks Trained with the Logistic Loss. In *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 1305–1338. PMLR, 09–12 Jul 2020. URL `https://proceedings.mlr.press/v125/chizat20a.html`.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. URL `https://doi.org/10.3115/v1/d14-1179`.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling Language Modeling with Pathways, 2022.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023. URL `http://jmlr.org/papers/v24/22-1144.html`.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Erhan Çinlar. *Probability and stochastics*, volume 261. Springer, 2011.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *ICML*, 2016.

Taco S Cohen and Max Welling. Steerable CNNs. In *ICLR*, 2017.

Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *ICLR*, 2018.

Taco S Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant CNNs on homogeneous spaces. In *NeurIPS*, 2019.

Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. AdaNet: Adaptive Structural Learning of Artificial Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 874–883, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL `http://proceedings.mlr.press/v70/cortes17a.html`.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

Andrew M Dai and Quoc V Le. Semi-supervised Sequence Learning. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL `https://proceedings.neurips.cc/paper/2015/file/7137debd45ae4d0ab9aa953017286b20-Paper.pdf`.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling Vision Transformers to 22 Billion Parameters. In *Proceedings of the 40th International Conference on Machine Learning*, volume

202 of *Proceedings of Machine Learning Research*, pages 7480–7512. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/dehghani23a.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. URL https://aclanthology.org/N19-1423.

Thomas G. Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45014-6.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp Minima Can Generalize for Deep Nets. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1019–1028. JMLR.org, 2017.

Tien Huu Do, Duc Minh Nguyen, Giannis Bekoulis, Adrian Munteanu, and N. Deligiannis. Graph Convolutional Neural Networks with Node Transition Probability-based Message Passing and DropNode Regularization. *Expert Syst. Appl.*, 174:114711, 2021.

Alexander G. Donchev, Andrew G. Taube, Elizabeth Decolvenaere, Cory Hargus, Robert T. McGibbon, Ka-Hei Law, Brent A. Gregersen, Je-Luen Li, Kim Palmo, Karthik Siva, et al. Quantum chemical benchmark databases of gold-standard dimer interaction energies. *Scientific Data*, 8, 2021.

Xuanyi Dong and Yi Yang. Searching for a Robust Neural Architecture in Four GPU Hours. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019.

Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=HJxyZkBKDr`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020a.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2020b. URL `https://arxiv.org/abs/2010.11929`.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542:115–, 2017.

Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) equivariant representations with spherical CNNs. In *ECCV*, 2018.

Carlos Esteves, Ameesh Makadia, and Kostas Daniilidis. Spin-Weighted Spherical CNNs. In *NeurIPS*, 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning*

Research, pages 1437–1446, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/falkner18a.html.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *International Conference on Machine Learning*, pages 3165–3176. PMLR, 2020.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware Minimization for Efficiently Improving Generalization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=6Tm1mposlrM.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep Ensembles: A Loss Landscape Perspective. *ArXiv*, abs/1912.02757, 2019.

Luc Frachon, Wei Pang, and George M. Coghill. ImmuNeCS: Neural Committee Search by an Artificial Immune System, 2020.

F. Fuchs, Daniel E. Worrall, Volker Fischer, and M. Welling. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. *ArXiv*, abs/2006.10503, 2020a.

Fabian B Fuchs, Daniel E Worrall, Volker Fischer, and Max Welling. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. In *NeurIPS*, 2020b.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born Again Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1607–1616. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/furlanello18a.html.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/gal16.html.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural Message Passing for Quantum Chemistry. In *ICML*, 2017.

Varun Godbole, George E. Dahl, Justin Gilmer, Christopher J. Shallue, and Zachary Nado. Deep Learning Tuning Playbook, 2023. URL http://github.com/google-research/tuning_playbook. Version 1.0.

Jonathan Godwin*, Thomas Keck*, Peter Battaglia, Victor Bapst, Thomas Kipf, Yujia Li, Kimberly Stachenfeld, Petar Veličković, and Alvaro Sanchez-Gonzalez. Jraph: A library for graph neural networks in jax., 2020. URL http://github.com/deepmind/jraph.

Jonathan Godwin, Michael Schaarschmidt, Alexander L Gaunt, Alvaro Sanchez-Gonzalez, Yulia Rubanova, Petar Veličković, James Kirkpatrick, and Peter Battaglia. Simple GNN Regularisation for 3D Molecular Property Prediction and Beyond. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=1wVvweK3oIb.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran

Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Google. Gemini: A Family of Highly Capable Multimodal Models, 2023.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, 2014.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In *NeurIPS*, 2019.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/guo17a.html.

Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schön. Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

Lars K. Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

Bobby He, Balaji Lakshminarayanan, and Yee Whye Teh. Bayesian Deep Ensembles via the Neural Tangent Kernel. In *Advances in Neural Information Processing Systems*, volume 33, pages 1010–1022. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/0b1ec366924b26fc98fa7b71a9c249cf-Paper.pdf`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. In *Computer Vision – ECCV 2016*, pages 630–645, Cham, 2016b. Springer International Publishing. ISBN 978-3-319-46493-0.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016c.

Dan Hendrycks and Thomas Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *International Conference on Learning Representations (ICLR)*, 2019.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Matthew Hirn, Stéphane Mallat, and Nicolas Poilvert. Wavelet scattering regression of quantum chemical energies. *Multiscale Modeling & Simulation*, 15(2):827–863, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. *arXiv preprint arxiv:2006.11239*, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. SIMPLIFYING NEURAL NETS BY DISCOVERING FLAT MINIMA. In *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994. URL https://proceedings.neurips.cc/paper/1994/file/01882513d5fa7c329e940dda99b12147-Paper.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Emiel Hoogeboom, Jorn WT Peters, Taco S Cohen, and Max Welling. HexaConv. In *ICLR*, 2018.

Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant Diffusion for Molecule Generation in 3D, 2022. URL https://arxiv.org/abs/2203.17003.

Weihua Hu, Bowen Liu, Joseph Gomes, M. Zitnik, Percy Liang, V. Pande, and J. Leskovec. Strategies for Pre-training Graph Neural Networks. *arXiv: Learning*, 2020a.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. GPT-GNN: Generative Pre-Training of Graph Neural Networks. *CoRR*, abs/2006.15437, 2020b. URL https://arxiv.org/abs/2006.15437.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music Transformer. In *ICLR*, 2019.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John Hopcroft, and Kilian Weinberger. Snapshot Ensembles: Train 1, get M for free. In *International Conference on Learning Representations (ICLR)*, 2017a.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot Ensembles: Train 1, Get M for Free. In *5th International*

Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017b. URL https://openreview.net/forum?id=BJYwwY9ll.

Michael J Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim. LieTransformer: Equivariant Self-Attention for Lie Groups. In *International Conference on Machine Learning*, pages 4533–4543. PMLR, 2021.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient Non-stationarity and Generalisation in Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Qun8fv4qSby.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. In *UAI*, pages 876–885. AUAI Press, 2018.

Siddhartha Jain, Ge Liu, Jonas Mueller, and David Gifford. Maximizing Overall Diversity for Improved Uncertainty Estimates in Deep Ensembles. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4264–4271, Apr. 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5849.

Yiding Jiang*, Behnam Neyshabur*, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic Generalization Measures and Where to Find Them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgIPJBFvH.

Zhengshen Jiang, Hongzhi Liu, Bin Fu, and Zhonghai Wu. Generalized Ambiguity Decompositions for Classification with Applications in Active Learning and Unsupervised Ensemble Pruning. In *AAAI*, pages 2073–2079. AAAI Press, 2017.

Rui Jiao, Jiaqi Han, Wenbing Huang, Yu Rong, and Yang Liu. Energy-Motivated Equivariant Pretraining for 3D Molecular Graphs. *arXiv preprint arXiv:2207.08824*, 2022.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, 2020.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=H1oyRlYgg`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Thomas Kipf and Max Welling. Variational Graph Auto-Encoders. *ArXiv*, abs/1611.07308, 2016.

James Kirkpatrick, Brendan McMorrow, David H. P. Turban, Alexander L. Gaunt, James S. Spencer, Alexander G. D. G. Matthews, Annette Obika, Louis Thiry, Meire Fortunato, David Pfau, et al. Pushing the frontiers of density functionals

by solving the fractional electron problem. *Science*, 374(6573):1385–1389, 2021. URL https://www.science.org/doi/abs/10.1126/science.abj6511.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. In *ICLR*, 2019.

Johannes Klicpera, Shankari Giri, Johannes T. Margraf, and Stephan Günnemann. Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules. *CoRR*, abs/2011.14115, 2020a. URL https://arxiv.org/abs/2011.14115.

Johannes Klicpera, Shankari Giri, Johannes T Margraf, and Stephan Günnemann. Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules. *arXiv preprint arXiv:2011.14115*, 2020b.

Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional Message Passing for Molecular Graphs. *ArXiv*, abs/2003.03123, 2020c.

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021. URL https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

Risi Kondor and Shubhendu Trivedi. On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. In *ICML*, 2018.

Risi Kondor, Zhen Lin, and Shubhendu Trivedi. Clebsch–Gordan nets: a fully Fourier space spherical convolutional neural network. In *NeurIPS*, 2018a.

Risi Kondor, Hy Truong Son, Horace Pan, Brandon M. Anderson, and Shubhendu Trivedi. Covariant Compositional Networks For Learning Graphs. *CoRR*, abs/1801.02144, 2018b. URL http://arxiv.org/abs/1801.02144.

Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. In *NeurIPS*, 2019.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian Institute for Advanced Research), 2009. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Anders Krogh and Jesper Vedelsby. Neural Network Ensembles, Cross Validation, and Active Learning. In *Advances in Neural Information Processing Systems 7*, pages 231–238. MIT Press, 1995.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems 30*, pages 6402–6413. Curran Associates, Inc., 2017.

Y. Le and X. Yang. Tiny ImageNet Visual Recognition Challenge, 2015.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989. ISSN 0899-7667. URL `https://doi.org/10.1162/neco.1989.1.4.541`.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *ICML*, 2019.

Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks. *arXiv e-prints*, art. arXiv:1511.06314, 2015.

Julien-Charles Lévesque, Christian Gagné, and Robert Sabourin. Bayesian Hyperparameter Optimization for Ensemble Learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'16, page 437–446, Arlington, Virginia, USA, 2016. AUAI Press. ISBN 9780996643115.

Liam Li and Ameet Talwalkar. Random Search and Reproducibility for Neural Architecture Search. In *UAI*, page 129. AUAI Press, 2019.

Xingjian Li, Haoyi Xiong, Haozhe An, Cheng-Zhong Xu, and Dejing Dou. RIFLE: Backpropagation in Depth for Deep Transfer Learning through Re-Initializing the Fully-connected LayEr. In *ICML*, 2020.

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *arXiv preprint arXiv:1907.04595*, 2019.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations*, 2019.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2019a.

Shengchao Liu, Mehmet F. Demirel, and Yingyu Liang. N-Gram Graph: Simple Unsupervised Representation for Graphs, with Applications to Molecules. In *NeurIPS*, 2019b.

Shengchao Liu, Hanchen Wang, Weiyang Liu, Joan Lasenby, Hongyu Guo, and Jian Tang. Pre-training Molecular Graph Representation with 3D Geometry. *CoRR*, abs/2110.07728, 2021a. URL `https://arxiv.org/abs/2110.07728`.

Shengchao Liu, Hongyu Guo, and Jian Tang. Molecular Geometry Pretraining with SE(3)-Invariant Denoising Distance Matching. *arXiv preprint arXiv:2206.13602*, 2022a.

Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural networks : the official journal of the International Neural Network Society*, 12 10:1399–1404, 1999.

Yi Liu, Limei Wang, Meng Liu, Yuchao Lin, Xuan Zhang, Bora Oztekin, and Shuiwang Ji. Spherical Message Passing for 3D Molecular Graphs. In *International Conference on Learning Representations*, 2022b. URL `https://openreview.net/forum?id=givsRXsOt9r`.

Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S. Yu. Graph Self-Supervised Learning: A Survey. *CoRR*, abs/2103.00111, 2021b. URL `https://arxiv.org/abs/2103.00111`.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021c.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

Vladimir Macko, Charles Weill, Hanna Mazzawi, and Javier Gonzalvo. Improving Neural Architecture Search Image Classifiers via Ensemble Learning. *ArXiv*, abs/1903.06236, 2019.

James Martens, Andy Ballard, Guillaume Desjardins, Grzegorz Swirszcz, Valentin Dalibard, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Rapid training of deep neural networks without skip connections or normalization layers using Deep Kernel Shaping, 2021. URL `https://arxiv.org/abs/2110.01765`.

Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards Automatically-Tuned Neural Networks. In *ICML 2016 AutoML Workshop*, 2016.

Benjamin Kurt Miller, Mario Geiger, Tess E Smidt, and Frank Noé. Relevance of rotationally equivariant convolutions for predicting molecular properties. *arXiv preprint arXiv:2008.08461*, 2020.

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining Well Calibrated Probabilities using Bayesian Binning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2901–2907. AAAI Press, 2015. ISBN 0262511290.

Maho Nakata and Tomomi Shimazaki. PubChemQC Project: A Large-Scale First-Principles Electronic Structure Database for Data-Driven Chemistry. *Journal of Chemical Information and Modeling*, 57(6):1300–1308, 2017. URL `https://doi.org/10.1021/acs.jcim.7b00083`. PMID: 28481528.

Behnam Neyshabur. Implicit Regularization in Deep Learning. *ArXiv*, abs/1709.01953, 2017. URL `https://api.semanticscholar.org/CorpusID:22111111`.

Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Skz_WfbCZ`.

Jeremy Nixon, Balaji Lakshminarayanan, and Dustin Tran. Why Are Bootstrapped Deep Ensembles Not Better? In *"I Can't Believe It's Not Better!" NeurIPS 2020 workshop*, 2020. URL `https://openreview.net/forum?id=dTCir0ceyv0`.

Emmy Noether. Invariant variation problems. *Zu Göttingen, Math-phys*, pages 235–257, 1918.

Randal S. Olson and Jason H. Moore. TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning. In *Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pages 66–74,

New York, New York, USA, 24 Jun 2016. PMLR. URL http://proceedings.mlr.press/v64/olson_tpot_2016.html.

OpenAI. GPT-4 Technical Report, 2023.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems 32*, pages 13991–14002. Curran Associates, Inc., 2019.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021. URL http://jmlr.org/papers/v22/19-1028.html.

Emilio Parisotto, H Francis Song, Jack W Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing Transformers for Reinforcement Learning. In *ICML*, 2020.

Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-Alone Self-Attention in Vision Models. In *NeurIPS*, 2019a.

Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-Alone Self-Attention in Vision Models. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 68–80, 2019b. URL https://proceedings.neurips.cc/paper/2019/hash/3416a75f4cea9109507cacd8e2f2aefc-Abstract.html.

Robert G. Parr and Yang Weitao. *Density-Functional Theory of Atoms and Molecules*. Oxford University Press, USA, 1994. ISBN 0195092767. URL http://www.amazon.com/

Density-Functional-Molecules-International-Monographs-Chemistry/dp/0195092767/ref=sr_1_1?ie=UTF8&s=books&qid=1279096906&sr=1-1.

David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. Carbon Emissions and Large Neural Network Training. *CoRR*, abs/2104.10350, 2021. URL https://arxiv.org/abs/2104.10350.

Tim Pearce, Felix Leibfried, and Alexandra Brintrup. Uncertainty in Neural Networks: Approximately Bayesian Ensembling. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 234–244, Online, 26–28 Aug 2020. PMLR. URL http://proceedings.mlr.press/v108/pearce20a.html.

T. Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and P. Battaglia. Learning Mesh-Based Simulation with Graph Networks. *ArXiv*, abs/2010.03409, 2020.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Dräxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron C. Courville. On the Spectral Bias of Neural Networks. In *International Conference on Machine Learning*, 2018.

R. Ramakrishnan, Pavlo O. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014a.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014b.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, pages 4780–4789. AAAI Press, 2019. ISBN 978-1-57735-809-1.

Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younès Bennani. A survey on domain adaptation theory. *CoRR*, abs/2004.11829, 2020. URL `https://arxiv.org/abs/2004.11829`.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

David W Romero and Jean-Baptiste Cordonnier. Group Equivariant Stand-Alone Self-Attention For Vision. In *ICLR*, 2021.

David W. Romero and Mark Hoogendoorn. Co-Attentive Equivariant Neural Networks: Focusing Equivariance On Transformations Co-Occurring in Data. In *ICLR*, 2020.

David W Romero, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Attentive Group Equivariant Convolutional Networks. In *ICML*, 2020.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. The Truly Deep Graph Convolutional Networks for Node Classification. *CoRR*, abs/1907.10903, 2019. URL `http://arxiv.org/abs/1907.10903`.

Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-Supervised Graph Transformer on Large-Scale Molecular Data, 2020. URL `https://arxiv.org/abs/2007.02835`.

Lars Ruddigkeit, Ruud van Deursen, Lorenz C. Blum, and Jean-Louis Reymond. Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17. *J. Chem. Inf. Model.*, 52(11):2864–2875, November 2012. ISSN 1549-9596. URL `https://doi.org/10.1021/ci300415d`. Publisher: American Chemical Society.

Tara N. Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8614–8618, 2013.

Alvaro Sanchez-Gonzalez, N. Heess, Jost Tobias Springenberg, J. Merel, Martin A. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. *ArXiv*, abs/1806.01242, 2018.

Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR, 13–18 Jul 2020. URL `http://proceedings.mlr.press/v119/sanchez-gonzalez20a.html`.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *CoRR*, abs/1801.04381, 2018. URL `http://arxiv.org/abs/1801.04381`.

Adam Santoro, Felix Hill, David G. T. Barrett, Ari S. Morcos, and Timothy P. Lillicrap. Measuring abstract reasoning in neural networks. *ArXiv*, abs/1807.04225, 2018.

R. Sato, Makoto Yamada, and Hisashi Kashima. Random Features Strengthen Graph Neural Networks. In *SDM*, 2021.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) Equivariant Graph Neural Networks, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR, 18–24 Jul 2021. URL `http://proceedings.mlr.press/v139/schmidt21a.html`.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. LAION-5B: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL `https://openreview.net/forum?id=M3Y74vmsMcY`.

Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, A. Tkatchenko, and K. Müller. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *NIPS*, 2017a.

Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *NeurIPS*, 2017b.

Kristof Schütt, Oliver Unke, and Michael Gastegger. Equivariant message passing for the prediction of tensorial properties and molecular spectra. In *International Conference on Machine Learning*, pages 9377–9388. PMLR, 2021.

Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. Learning Gradient Fields for Molecular Conformation Generation. In *International Conference on Machine Learning*, 2021.

Muhammed Shuaibi, Adeesh Kolluru, Abhishek Das, Aditya Grover, Anuroop Sriram, Zachary Ulissi, and C Lawrence Zitnick. Rotation invariant graph neural networks using spin convolutions. *arXiv preprint arXiv:2106.09575*, 2021.

J. Sietsma and Robert J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4:67–79, 1991.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. doi: 10.1038/nature16961.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014. URL `https://arxiv.org/abs/1409.1556`.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015.

Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/sohl-dickstein15.html`.

Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

Yang Song and Stefano Ermon. Improved Techniques for Training Score-Based Generative Models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=PxTIG12RRHS`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *International Conference on Learning Representations*, 2019.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

Ahmed Taha, Abhinav Shrivastava, and Larry S Davis. Knowledge Evolution in Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12843–12852, 2021.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Velickovic, and Michal Valko. Bootstrapped Representation Learning on Graphs. *CoRR*, abs/2102.06514, 2021. URL `https://arxiv.org/abs/2102.06514`.

Philipp Thölke and Gianni De Fabritiis. TorchMD-NET: Equivariant Transformers for Neural Network based Molecular Potentials. *arXiv preprint arXiv:2202.02541*, 2022.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3D point clouds. *arXiv preprint arXiv:1802.08219*, 2018a.

Nathaniel Thomas, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds. *CoRR*, abs/1802.08219, 2018b. URL `http://arxiv.org/abs/1802.08219`.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel. In *EMNLP-IJCNLP*, 2019.

Oliver T. Unke and Markus Meuwly. PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges. *Journal of Chemical Theory and Computation*, 15(6):3678–3693, May 2019. ISSN 1549-9626. URL `http://dx.doi.org/10.1021/acs.jctc.9b00181`.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *NeurIPS*, 2017.

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rklz9iAcKQ`.

Pascal Vincent. A Connection Between Score Matching and Denoising Autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.

Pascal Vincent, H. Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML '08*, 2008.

Pascal Vincent, H. Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.*, 11: 3371–3408, 2010.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-Attention with Linear Complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

Andrew M. Webb, Charles Reynolds, Dan-Andrei Iliescu, Henry W. J. Reeve, Mikel Luján, and Gavin Brown. Joint Training of Neural Network Ensembles. *CoRR*, abs/1902.04422, 2019.

Maurice Weiler and Gabriele Cesa. General E(2)-Equivariant Steerable CNNs. In *NeurIPS*, 2019.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data. In *NeurIPS*, 2018a.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3D steerable CNNs: Learning rotationally equivariant features in volumetric data. In *NeurIPS*, 2018b.

Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant CNNs. In *CVPR*, 2018c.

Max Welling and Yee Whye Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter Ensembles for Robustness and Uncertainty Quantification. In *Advances in Neural Information Processing Systems*, volume 33, pages 6514–6527. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/481fbfa59da2581098e841b7afc122f1-Paper.pdf`.

Andrew G Wilson and Pavel Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. In *Advances in Neural Information Processing Systems*, volume 33, pages 4697–4708. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/322f62469c5e3c7dc3e58f5a4d1ea399-Paper.pdf`.

Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *CVPR*, 2017.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv e-prints*, 2017.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402, 2018.

Yaochen Xie, Zhao Xu, Zhengyang Wang, and Shuiwang Ji. Self-Supervised Learning of Graph Neural Networks: A Unified Review. *CoRR*, abs/2102.10757, 2021. URL `https://arxiv.org/abs/2102.10757`.

Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. GeoDiff: A Geometric Diffusion Model for Molecular Conformation Generation. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=PzcvxEMzvQC`.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2020.

Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training Behavior of Deep Neural Network in Frequency Domain. In *Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part I*, page 264–274, Berlin, Heidelberg, 2019. Springer-Verlag. ISBN 978-3-030-36707-7. URL `https://doi.org/10.1007/978-3-030-36708-4_22`.

Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly hard. In *International Conference on Learning Representations (ICLR)*, 2020a.

Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting" over-smoothing" in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020b.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do Transformers Really Perform Bad for Graph Representation? In *NeurIPS*, 2021.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph Contrastive Learning with Augmentations. In *Advances in Neural Information Processing Systems*, volume 33, pages 5812–5823. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/3fe230348e9a12c13120749e3f9fa4cd-Paper.pdf`.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the Search Phase of Neural Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2020.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020.

Arber Zela, Julien Siems, and Frank Hutter. NAS-Bench-1Shot1: Benchmarking and Dissecting One-Shot Neural Architecture Search. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJx9ngStPH`.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Guodong Zhang, Aleksandar Botev, and James Martens. Deep Learning without Shortcuts: Shaping the Kernel with Tailored Rectifiers. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=U0k7XNTiFEq`.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-Attention Generative Adversarial Networks. In *ICML*, 2019.

Kaikai Zhao, Tetsu Matsukawa, and Einoshin Suzuki. Retraining: A Simple Way to Improve the Ensemble Accuracy of Deep Neural Networks for Image Classification. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 860–867, 2018.

L. Zhao and Leman Akoglu. PairNorm: Tackling Oversmoothing in GNNs. *ArXiv*, abs/1909.12223, 2020.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ryxmb1rKDS`.

Kuangqi Zhou, Yanfei Dong, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Effective Training Strategies for Deep Graph Neural Networks. *CoRR*, abs/2006.07107, 2020. URL `https://arxiv.org/abs/2006.07107`.

Tianyi Zhou, Shengjie Wang, and Jeff A Bilmes. Diverse Ensemble Evolution: Curriculum Data-Model Marriage. In *Advances in Neural Information Processing Systems 31*, pages 5905–5916. Curran Associates, Inc., 2018.

Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall, 1st edition, 2012. ISBN 1439830037.

Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1):239–263, 2002. ISSN 0004-3702.

Barret Zoph and Quoc V Le. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, 2018.