Towards Human-Like Natural Language Understanding with Language Models

Yordan Yordanov

Keble College

University of Oxford

Supervisor: Prof. Thomas Lukasiewicz

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2024

Contents

Acknowledgements

I would like to thank my supervisor Thomas Lukasiewicz and my co-supervisor Oana-Maria Camburu for always being available and helpful in my work.

I would like to thank everyone who contributed with advice, as well as all of my colleagues who collaborated with me. In particular, I would like to thank: Oana-Maria Camburu, Tommaso Salvatori, Vid Kocijan, Luca Pinchetti, Yuhang Song, and Ana-Maria Cretu. I would also like to thank Patrick Hohenecker and Sri Vishnu Kumar Karlapati for collaborating on work that did not make it to this thesis but helped me develop as a researcher. I would like to thank my Transfer and Confirmation of Status examiners Phil Blunsom and Emanuel Sallinger whose valuable feedback helped me improve my thesis.

I would also like to thank the staff at the Department of Computer Science who are always quick and responsive, in particular the highly professional IT team that made sure that our servers and computers are always online, and Keble College for providing support with non-academic matters.

I would also like to thank my funder for the Department of Computer Science Scholarship, and the EPSRC-funded Tier 2 facility JADE (EP/P020275/1) and Scan Computers International Ltd for providing GPU computing resources.

I would especially like to thank my family and my girlfriend and partner in life Yue He for always being there for me.

Abstract

In recent years, language models (LMs) have been established as the highest-capability models for most natural language tasks. However, beyond the invention of the transformer architecture, most progress has been made through scaling the model and data size (Radford et al., 2018, 2019; Brown et al., 2020; OpenAI, 2023). The scaling made it possible for these models to be on par or better than humans in standard natural language benchmarks. However, version after version of those models have remained different and inferior to humans in their reasoning capabilities, explainability and learning capabilities. Research in natural language explanations (NLEs) (Hendricks et al., 2016) has been behind the research on neural-network-based LMs (Bengio et al., 2003), in part due to its much later start. Furthermore, LMs are still trained via backpropagation, which is less efficient and fundamentally different from how the human brain works. In this thesis, I present my progress in making LMs more human-like, both in terms of natural language understanding and in terms of their biological plausibility. First, I explore a very challenging set of problems that test natural language understanding, namely, hard cases of pronoun resolution such as the Winograd Schema Challenge. In particular, I introduce improvements to training LMs for pronoun resolution via synthetic training datasets, specialised loss functions and by comparing task reformulations. Second, I use LMs to generate NLEs on commonsense reasoning tasks such as hard cases of pronoun resolution and commonsense validation. I demonstrate that LMs can be used for the efficient transfer of NLEs between domains, while obtaining high downstream accuracy. Finally, I explore using more biologically plausible predictive-coding-based training methods for LMs, which may be the future of deep learning beyond backpropagation (Millidge et al., 2022). I show the first-ever application of these methods to training LMs. I show the best ways to implement them, study their scalability, determine the best method to use, and show competitive results with backpropagation for small LMs.

Notations

List of commonly used abbreviations and mathematical notations in alphabetical order.

Abbreviations

BP – Backpropagation.

iPC – Incremental Predictive Coding – same as PPC.

KL – The KL-divergence modification by Pinchetti et al. (2022).

LayerNorm/layernorm – Layer Normalisation.

LM – Language Model.

LSTM – Long Short-Term Memory mechanism (Hochreiter and Schmidhuber, 1997).

NLE – Natural Language Explanation.

NLP – Natural Language Processing.

no-LayerNorm – Layer normalisation is disabled.

PC – The Predictive Coding method (Whittington and Bogacz, 2017).

PC layer/PC Layer – A special predictive coding layer that calculates the energy

PPC – Parallel Predictive Coding (Song, 2021).

PPC_{KL} – PPC with the KL-divergence modification by Pinchetti et al. (2022).

ppl – Perplexity.

RNN – Recurrent Neural Network.

SOTA – State-of-The-Art.

Mathematical Notations

h – Hidden size of the model.

lr – Learning Rate (optimisation parameter).

T – Number of inference steps for predictive coding.

Chapter 1

Introduction

Human-like natural language understanding is a broad term that encompasses the abilities of natural language models (LMs) to predict, explain, and learn like humans. In this thesis, I present progress on all three of these aspects: 1) human-like commonsense reasoning on hard cases of pronoun resolution (the Winograd Schema Challenge (Levesque et al., 2012)), 2) human-line natural language explanations (NLEs) that explain the commonsense reasoning process, and 3) brain-inspired learning via predictive coding (Clark, 2013). To achieve this, I fine-tune LMs for commonsense reasoning (hard cases of pronoun resolution) and for producing NLEs. Then, I consider training LMs by using biologically plausible predictive coding alternatives to backpropagation (BP). These methods are based on the predictive coding theory of the brain, and only rely on local computation (similar to the brain) which can lead to future efficiency gains with specialised analogue hardware (Millidge et al., 2022).

Commonsense Reasoning Human-like natural language understanding includes the model's capability for human-like commonsense reasoning. Hard cases of pronoun resolution, in particular the Winograd Schema Challenge (WSC), are a famous and challenging commonsense reasoning test. In this thesis, I present how data augmentation and large LMs have become the keys to high performance in hard cases of pronoun resolution. In particular, I specify my contribution to creating synthetic pronoun resolution datasets and training LMs on them. This work inspired two impactful works that I co-authored: (Kocijan et al., 2019b) and (Kocijan et al., 2019a). I also study and compare the different ways to represent the pronoun resolution task as an objective function for training LMs, which is based on my published work (Yordanov et al., 2020).

Natural Language Explanations Human-like natural language understanding also includes the model's ability to generate natural language explanations (NLEs). These are human-like textual explanations of the decisions of a model. Producing NLEs enables black-box model interpretability and improves the trust in the model's performance. A particular challenge with NLEs is data scarcity, because most NLP tasks do not have available NLEs. Furthermore, NLEs are difficult and expensive to collect, because they require high-quality human annotation. In this thesis, I demonstrate how one can leverage large datasets with NLEs such as e-SNLI (Camburu et al., 2018) to transfer the explanation-generation ability to tasks with few available NLEs such as hard cases of pronoun resolution (WinoGrande (Sakaguchi et al., 2020)) and commonsense validation (ComVE (Wang et al., 2020)), and I find the best training methods to achieve this. This is based on my published work (Yordanov et al., 2022).

Brain-Inspired Learning via Predictive Coding The predictive coding theory of the brain provides more biologically plausible alternatives to the dominant back-propagation paradigm for training neural networks (NNs). The neuroscience theory states that the brain operates with multiple levels of hierarchy, where the neurons of each level predict the input from the level below and adjust based on the error between prediction and input (Clark, 2013).

In machine learning, there are many predictive coding methods for training NNs such that each neuron is modelled to predict the state of the neuron below it at the next time step. In particular, there are two predictive coding methods used in this thesis: Predictive Coding (PC) (Whittington and Bogacz, 2017) – which uses multiple inference updates and a single weight update per training batch; and Parallel Predictive Coding (PPC) (Song, 2021) – a more biologically-plausible version of PC for which the inference and weight updates are done simultaneously at each time step of the algorithm. I use these methods because they are more biologically plausible than BP, as they are grounded in a leading theory of how the brain works (predictive coding) and rely only on local computations (Song et al., 2020; Song, 2021).

In this thesis, I show the first application of biologically plausible predictive coding methods to training LMs, and the first application of predictive coding to transformer LMs. I explore what architectural modifications are needed to implement the PPC method for RNN and transformer LMs. I explore using the KL-modification (Pinchetti et al., 2022) for predictive coding methods for training transformer LMs, which is based on my contribution to our published work (Pinchetti et al., 2022). I also determine what training method (PC vs PPC) works best for transformer LMs,

which is based on my contribution to our published work (Salvatori et al., 2024). Finally, I investigate how the RNN and transformer LMs trained with predictive coding scale in terms of model and training data size.

1.1 Research Questions

In this thesis, I formulate and make progress on research questions about the capabilities of LMs for human-like natural language understanding. The research questions are listed here and answered in each of the three core chapters (3, 4, and 5). While the background helpful to understanding each chapter is contained within it, I also include background on common topics like transfer learning and LMs in Chapter 2.

Hard Cases of Pronoun Resolution Commonsense reasoning capabilities are essential for human-like natural language understanding. In this thesis, I focus on hard cases of pronoun resolution as a famous commonsense reasoning test (Levesque et al., 2012).

First, I answer questions about the feasibility of using a synthetic pronoun resolution training dataset to improve the performance of LMs on hard cases of pronoun resolution. In particular, can a synthetic pronoun resolution dataset of sufficient quality be constructed, and what data sources are the best for this? Can the resulting dataset be used to improve the performance of LMs on hard cases of pronoun resolution? Are there any task-specific loss functions that can be used, and if so, which of them would lead to the best performance?

Second, I answer questions about the correct choice of the objective function for pronoun resolution, which I do by fine-tuning LMs on a large, high-quality dataset of hard cases of pronoun resolution (WinoGrande (Sakaguchi et al., 2020)). In particular, does the objective function affect pronoun resolution performance in a significant way? What is its reliability to random variations (seed choice)? Is the best objective function different depending on whether the evaluation dataset is the same (in-domain), or different (out-of-domain)?

Natural Language Explanations The ability of models to understand language goes beyond answering commonsense questions but also means that the models should be able to explain their answers. In this thesis, I train LMs to generate NLEs for commonsense reasoning tasks. I answer questions for the setting where only a few NLEs are available, but where abundant NLEs are available for another task, and

where abundant training labels are available for both training tasks. In particular, can few-shot out-of-domain transfer learning improve NLE quality, as verified by human annotators? How to use fine-tuning and multi-task learning for transfer learning of NLEs in this setting, and which method performs the best? Can human and automatic evaluation provide additional insights into the quality of NLEs produced by the models? Do the best methods scale in terms of data and model size? Do the answers to these questions depend on the choice of task: hard cases of pronoun resolution (WinoGrande (Sakaguchi et al., 2020)) or commonsense validation (ComVE (Wang et al., 2020))?

Predictive Coding for LMs While backpropagation (BP) is the training method that current LMs rely on, it is not human-like because it is not biologically plausible. To achieve greater biological plausibility, I explore using predictive coding methods to train LMs, which, to the best of my knowledge, I am the first to make progress on. Predictive coding methods are more biologically plausible than BP because they are based on the predictive coding theory of the brain, and work via local computations (Song et al., 2020; Song, 2021) like the brain. I answer several questions about applying predictive coding methods to recurrent neural network-based (RNN-based) and transformer-based LMs. In particular, how to apply predictive coding to LMs – what architectural modifications are beneficial, and what hyperparameter values to choose? What energy function and what predictive coding method work best for transformer LMs? Can LMs trained via predictive coding perform on par with the ones trained via BP? How does the training method affect how LMs scale in terms of data and model size? Do the answers to these questions depend on the transformer LM variant: conditional or masked?

1.2 Results

The research questions listed above are answered by the results of each of the three core chapters (3, 4, and 5). Here, I summarise the results of each chapter. The general conclusions and outlook are presented in Chapter 6.

Hard Cases of Pronoun Resolution In Chapter 3, I fine-tune LMs for commonsense reasoning, specifically, for solving hard cases of pronoun resolution.

I show that a large synthetic pronoun resolution dataset can be constructed automatically and that the resulting dataset is of sufficient quality for a training dataset.

Furthermore, I show that Wikipedia is the best data source for producing such a dataset. I show that the resulting dataset significantly improves the performance of LMs on hard cases of pronoun resolution (the Winograd Schema Challenge ((Levesque et al., 2012))). I also consider existing task-specific loss functions and introduce new variants that improve the pronoun resolution performance. Using the best dataset and loss configuration, I achieve results on par with the then-SOTA model while using ≈100 times fewer parameters.

Another question is the choice of the objective function, i.e. how to rephrase the pronoun resolution task for training LMs. I show that the objective function does matter for performance. I show that there is a clear best objective function for in-domain performance (on the test portion of the training dataset) that differs from the best objective function for out-of-domain performance (on datasets different from the training dataset).

Natural Language Explanations After my work on training LMs for commonsense reasoning in Chapter 3, in Chapter 4, I fine-tune LMs for joint prediction and NLE generation in the commonsense reasoning domain.

I show successful few-shot out-of-domain transfer of NLEs in the label-abundant setup. I identify four ways to combine few-shot and multi-task learning for this setup and determine which method performs the best. I evaluate all methods both in terms of the gold-standard human evaluation and in terms of automatic metrics. I show that the best method depends on the task, but that the best-performing methods do a separate fine-tuning on the few available NLEs. I also show that granular human evaluation results and automatic metrics can provide additional insights into the quality and type of NLEs produced by the models. Finally, I explore the scalability of the best methods for each task and show that they scale overall well in terms of data and model size.

Predictive Coding for LMs In chapters 3 and 4, I fine-tune LMs that were trained via BP. This raises the question of using more biologically plausible (human-like) training algorithms for LMs. In Chapter 5, I show my results on applying predictive coding methods inspired by the brain to train LMs.

I show that predictive coding, in particular, the PPC method (Song, 2021; Salvatori et al., 2022b), can be successfully applied to training RNN-based and transformer-based LMs. I show that some model choices such as the use of layer normalisation can affect performance depending on the LM type (masked or conditional), and I find the

best hyperparameter choices for PPC. For transformer LMs, I also determine the best predictive coding method to use, namely, PPC. I show that the predictive coding energy function via the KL-modification (Pinchetti et al., 2022) improves performance for the PPC method. Furthermore, I compare the performance of PPC versus BP and show that PPC outperforms BP for simple RNN LMs and is on par with BP for small transformer LMs.

Finally, I study the scalability of the PPC method. I show that simple RNN LMs trained with PPC scale well in terms of data and model size. On the other hand, transformer LMs, especially conditional transformer LMs, do not scale well for PPC, and underperform BP for larger models and longer training.

1.3 Publications

This thesis includes fully or partially a total of four publications. The following two publications are fully included in the thesis:

> Yordan Yordanov, Oana-Maria Camburu, Vid Kocijan, and Thomas Lukasiewicz. 2020. Does the objective matter? Comparing training objectives for pronoun resolution. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, November 16–20, 2020*

> Yordan Yordanov, Vid Kocijan, Thomas Lukasiewicz, and Oana-Maria Camburu. 2022. Few-shot out-of-domain transfer learning of natural language explanations in a label-abundant setup. In *Findings of the Association for Computational Linguistics: EMNLP 2022*. Association for Computational Linguistics

The thesis also includes Section "4.3 Transformer Language Models" from:

> Luca Pinchetti, Tommaso Salvatori, Yordan Yordanov, Beren Millidge, Yuhang Song, and Thomas Lukasiewicz. 2022. Predictive coding beyond Gaussian distributions. In *Advances in Neural Information Processing Systems*

and Section 5 "Language Model Experiments" from:

Tommaso Salvatori, Yuhang Song, Yordan Yordanov, Beren Millidge, Zhen ghua Xu, Lei Sha, Cornelius Emde, Rafal Bogacz, and Thomas Lukasiewicz. 2024. A stable, fast, and fully automatic learning algorithm for predictive coding networks. In *International Conference on Learning Representations*

The following publications that I have co-authored are not included in this thesis, but are inspired by my proof-of-concept work that is presented in this thesis as Section 3.2: *Solving the Winograd Schema Challenge*:

Vid Kocijan, Ana-Maria Cretu, Oana-Maria Camburu, Yordan Yordanov, and Thomas Lukasiewicz. 2019b. A surprisingly robust trick for the Winograd Schema Challenge. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4837–4842. Association for Computational Linguistics

Vid Kocijan, Oana-Maria Camburu, Ana-Maria Cretu, Yordan Yordanov, Phil Blunsom, and Thomas Lukasiewicz. 2019a. WikiCREM: A large unsupervised corpus for coreference resolution. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4303–4312. Association for Computational Linguistics

Chapter 2

Background and Literature Review: Language Models and Transfer Learning

This chapter contains the background and literature reviews on topics shared amongst the main chapters such as language models and transfer learning. Chapter-specific literature reviews and backgrounds can be found in each of the main chapters (3, 4, and 5).

2.1 Language Models

In recent years, language models (LMs) have become the foundation of state-of-the-art results in natural language processing (Brown et al., 2020; OpenAI, 2023; Wang et al., 2018, 2019a). These models consume vast quantities of text during training, which enables them to achieve impressive results with only a few task-specific examples (Devlin et al., 2019; Raffel et al., 2020; Brown et al., 2020; OpenAI, 2023).

2.1.1 Literature Review

The first NN-based LM was proposed by (Bengio et al., 2003). Initially, word embeddings were the dominating paradigm in NN-based language modelling (Mikolov et al., 2013; Pennington et al., 2014; Peters et al., 2018), where models would be trained to produce word embeddings that would serve as a building block for task-specific models. The first promising architecture for deep LMs is recurrent neural networks (RNNs) (Rumelhart et al., 1988; Sutskever et al., 2014), later enhanced with attention (Bahdanau et al., 2014). Convolutional NNs were also adapted for

LMs (Kalchbrenner et al., 2014), and for text classification (Zhang et al., 2015), but were later displaced by transformers (Vaswani et al., 2017; Radford et al., 2018).

The word embedding paradigm eventually shifted with the introduction of large language models (LLMs) based on the transformer architecture. The transformer model architecture was originally invented for natural language translation (Vaswani et al., 2017) where it achieved state-of-the-art performance in machine translation. Subsequently, it was adapted for language modelling by Radford et al. (2018), which constitutes the first generation of the GPT model family. The transformer architecture enables the GPT LM to achieve significant improvements in transfer learning performance on almost all NLP tasks in the GLUE benchmark (Wang et al., 2018).

Other notable directions of LM development include knowledge enhancement, e.g. ERNIE (Zhang et al., 2019), and enabling long contexts, e.g. TransformerXL (Dai et al., 2019) and XLNet (Yang et al., 2019).

Types of LMs: conditional and masked There are two main types of LMs depending on how the language modelling task is phrased: conditional and masked. Conditional LMs predict the next word (token) based only on the previous words (tokens), allowing for unidirectional context only. On the other hand, masked LMs mask several tokens in a sentence and are trained to reconstruct them based on all unmasked tokens, allowing for bidirectional context.

The conditional LM training objective of next-word-prediction has been shown to work the best for text generation tasks, e.g. GPT-1,2,3, and 4 (Radford et al., 2018, 2019; Brown et al., 2020; OpenAI, 2023), and T5 (Raffel et al., 2020). On the other hand, masked language modelling works better for classification, e.g. BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019b), and ALBERT (Lan et al., 2020).

There are other types of LMs that do not fit into the masked or conditional LM categories. Denoising auto-encoders are similar to masked LMs, but instead of masking the input, they corrupt the input by randomly replacing tokens with other (plausible) alternatives and then train the model to reconstruct the original text. E.g. ELECTRA (Clark et al., 2020) uses a small neural network to corrupt the input with plausible alternatives prior to training the main model.

Types of LMs based on the encoder Transformer LMs can also be either encoder-decoder (Lample and Conneau, 2019; Raffel et al., 2020) or decoder-only (Radford et al., 2018; Devlin et al., 2019). Encoder-decoder LMs follow the original

encoder-decoder transformer architecture by (Vaswani et al., 2017). Examples include XLM (Lample and Conneau, 2019), BART (Lewis et al., 2019) and T5 (Raffel et al., 2020). Decoder-only LMs simplify the transformer architecture by uniting the encoder and decoder into a single uniform block (e.g. see Figure 5.4). This architecture originated with the GPT-1 model (Radford et al., 2018), and was later used in all newer GPT models (Radford et al., 2019; Brown et al., 2020), as well as in other LMs including the family of masked LMs such as BERT (Devlin et al., 2019).

Modern-day LMs Modern-day LMs (e.g. GPT-4 (OpenAI, 2023)) achieve greater scalability and more efficient predictions via mixture-of-experts (MoE) Jacobs et al. (1991) – a technique that allows for parts of the NN to be disabled at prediction time. The Switch Transformer architecture (Fedus et al., 2022) integrates and adapts MoE inside the transformer block to create the first successful trillion-parameter LM.

In recent years, RNNs have seen a resurgence with the advent of Structured State Space models (SSMs) (Gu et al., 2022), and further improved by the Mamba LM (Gu and Dao, 2023), which shows superior efficiency to transformers.

LMs used in this thesis There are three pre-trained LMs used in this thesis: GPT-1 (Radford et al., 2018), RoBERTa (Liu et al., 2019b), and T5 (Raffel et al., 2020). Each LM is selected based on the best reported general-purpose benchmark performance at the time, while still being suitable for the task at hand. As discussed above, conditional LMs work best for text generation tasks, whereas masked LMs work best for classification tasks. This is why I use the conditional LM T5 for NLE generation in Section 4.2 and the masked LM RoBERTa for pronoun resolution classification in Section 3.3. The only exception is the conditional LM GPT-1 for pronoun resolution classification in Section 3.2 because masked LMs were not available at the time. Below I provide additional details for each LM.

GPT-1 is a decoder-only adaptation (Radford et al., 2018) of the original transformer architecture (Vaswani et al., 2017) with 117 million parameters trained on 7,000 books.

RoBERTa (Liu et al., 2019b) is an improvement to the BERT decoder-only masked LM (Devlin et al., 2019) by training it further – on 10 times more data (160GB), and by introducing slight modifications to the training procedure, while using the same model size – 355 million parameters.

T5 (Raffel et al., 2020) has an encoder-decoder transformer LM architecture similar to the original transformer architecture (Vaswani et al., 2017). T5 is trained

on a massive dataset (Colossal Clean Crawled Corpus (C4)) consisting of 750GB of text from the Web, together with a mixture of supervised tasks rephrased in text-to-text format. T5 has multiple variations based on its size (in number of parameters): T5-small: 60M; T5-base: 220M; T5-large: 770M, T5-3B: 2.8B; and T5-11B: 11B parameters.

2.1.2 Background

In the simplest terms, conditional LMs are trained to predict the next token in a sequence, given the previous tokens: $\mathrm{P}(w_i \mid w_0 \dots w_{i-1})$. On the other hand, masked LMs are trained to reconstruct randomly masked tokens (e.g. w_k) in the input (e.g. $w_0, \dots, w_{k-1}, <mask>, w_{k+1}, \dots, w_n$).

Recurrent neural networks (RNNs) (Rumelhart et al., 1988) are the first successful architecture for LMs (Bengio et al., 2003). RNNs use a cell $f(x_i, h_{i-1}) = h_i$ to autoregressively generate representations h_i of each token x_i of the input sequence, given the representation h_{i-1} of the previous token. Thus, RNNs require sequential, rather than parallel generation of representations, which incurs a computational overhead. RNNs also suffer from vanishing gradients (Hochreiter, 1991), which Hochreiter and Schmidhuber (1997) address with the creation of the LSTM (long short-term memory) cell.

Transformers (Vaswani et al., 2017; Devlin et al., 2019), on the other hand, are designed so that internal representations can be generated in parallel across the length of the sequence. This is achieved by using two components: positional embeddings and self-attention. Self-attention can be described by a parametrized function $f(x_i, x)$ that reweighs the input sequence x by the attention g of each of its elements x_i to all elements of x:

$$f(x_i, x) \coloneqq \sum_{j=1}^{n} x_j \frac{\exp(g(x_i, x_j))}{\sum_{k=1}^{n} \exp(g(x_i, x_k))}$$

Positional embeddings z are added to the input x to capture its sequential nature. They are introduced because the self-attention mechanism would otherwise not have information about the order of the input, which would make the model equivalent to bag-of-words. The resulting input is of the form $x'_i \coloneqq \mathrm{emb}(x_i, z_i)$, where $z_1, ..., z_n$ are the distinct embeddings for each position.

When evaluating the performance of LMs, a common metric to use is *perplexity* (Jelinek et al., 1977). More specifically, perplexity is the industry standard when evaluating the fit of the LMs to text corpora (Devlin et al., 2019; Radford et al.,

2018, 2019; Brown et al., 2020; OpenAI, 2023). It is a measure of how well the probability distribution at the output of the model approximates the observed targets. Perplexity is defined as the *exponential of the average negative log-likelihood* of the target sequence (Bengio et al., 2003):

$$exp\left(\frac{1}{N}\sum_{i=1}^{N} -\log P(w_1,\ldots,w_n)\right) = exp\left(\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n} -\log P(w_j|w_1,\ldots,w_{j-1})\right),$$

where N is the batch size, and n is the sequence length. For conditional LMs, the right-hand side of this formula is e^L, where L is the cross-entropy loss. For masked LMs, perplexity, as defined above, cannot be computed directly, but pseudo-perplexity can be computed by $e^{\mathscr{L}}$, where $\mathscr{L}$ is the cross-entropy loss of the masked tokens, as done by Devlin et al. (2019). The full formula is:

$$e^{\mathscr{L}} = exp\left(\frac{1}{N}\sum_{i=1}^{N}\sum_{j\in\mathscr{M}_i} -\log P(w_j|w_k : k \notin \mathscr{M}_i)\right),$$

where $\mathscr{M}_i$ is the set of positions of the masked tokens in sentence i.

One can also consider using accuracy as an evaluation metric for predicting the words in the text. E.g. Klakow and Peters (2002) show that accuracy and perplexity are highly correlated. However, to my knowledge, accuracy is not used for evaluating the LMs' fit to textual corpora (e.g. (Devlin et al., 2019; Radford et al., 2018, 2019; Brown et al., 2020; OpenAI, 2023)), but is only used to evaluate the LMs on downstream classification tasks. This may be because accuracy does not account for the degree of confidence in predictions, which is important for having a good fit to the training data distribution. Therefore, in this thesis, I do not use accuracy to evaluate LMs on text corpora.

For language generation applications such as machine translation, alternatives to perplexity are commonly used. These metrics test how close the LM-generated text is to human-generated text. Examples include BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), METEOR (Banerjee and Lavie, 2005), and BERTScore (Zhang et al., 2020b). Meister and Cotterell (2021) also propose a metric to test how well the LM-generated text matches the statistical tendencies of human language.

2.2 Transfer Learning

Transfer learning is the ability of machine learning models to retain and use knowledge from training on a *parent* dataset(s) (so-called pre-training) to training on a

child dataset(s) (so-called fine-tuning). Work on transfer learning in neural networks started as early as the 1970s, as documented by Bozinovski (2020). In recent years, transfer learning in NLP has been predominantly done via LMs (Raffel et al., 2020; Brown et al., 2020). These models are pre-trained on very large text corpora from the internet such as CommonCrawl (Rana, 2010), and subsequently fine-tuned for downstream (*child*) tasks, e.g. those comprising the GLUE benchmark (Wang et al., 2018). Successful transfer learning is often measured by comparing the performance of the fine-tuned model versus training on the child dataset only. Fine-tuning can be done in more than one step, e.g. in Chapter 4 I use up to three fine-tuning steps for transfer learning of natural language explanations.

For the rest of this section, I present the main categories of transfer learning: zero- vs few- vs multi-shot; in-domain vs out-of-domain; and in-context learning.

Zero- vs few- vs multi-shot Transfer learning can either be *zero-shot*, *few-shot*, or *multi-shot*, depending on the amount of data available for the downstream (*child*) task.

Multi-shot transfer learning (or simply *transfer learning*) is when there is abundant training data for the child task, so, oftentimes, no special training procedures are necessary to achieve transfer learning (Raffel et al., 2020).

Few-shot transfer learning is when there are only a few training data points for the child dataset, which can vary from 1 (sometimes called one-shot (Brown et al., 2020)) up to around 50 (Yordanov et al., 2020; Marasović et al., 2022), depending on the context. This often means that special care needs to be taken when training on those few examples to retain the knowledge from the parent task and prevent catastrophic forgetting (McCloskey and Cohen, 1989). One common method is to use learning rate schedulers (LeCun et al., 2002), where the learning rate is gradually increased and decreased to enable smoother fine-tuning. Another common method is gradual unfreezing (Howard and Ruder, 2018), where the network parameters are gradually unfrozen during training, starting from the most task-specific (the top) parts of the NN such as the prediction head.

Zero-shot transfer is when there are no training data points available from the child dataset. For example, modern-day large language models (LLMs) in chat assistants such as GPT-4, Gemini and Claude are capable of zero-shot transfer by only specifying the task description. For zero-shot transfer, since there is no training data, the performance is much more sensitive to the task specification via the choice of prompt (Liu et al., 2023).

In-domain vs out-of-domain Transfer learning can also be *in-domain* or *out-of-domain*, depending on whether the transfer learning is to the same task, or to a different task.

In-domain transfer learning deals with transferring performance from a parent dataset to a child dataset of the same task. For example, Sakaguchi et al. (2020) demonstrate that models trained on the WinoGrande dataset of hard cases of pronoun resolution achieve good performance when evaluated on the Winograd Schema Challenge (WSC) (Levesque et al., 2012) dataset of the same type. This is an example of zero-shot in-domain transfer learning.

Out-of-domain transfer learning means that the model learns to transfer performance between a parent task and a child task that are different (in different *domains*). All modern-day large LMs can be used effectively in out-of-domain transfer learning, which can be evaluated on benchmarks such as GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019a). My work in Chapter 4 (Yordanov et al., 2022) is also about out-of-domain transfer learning, for natural language explanation (NLE) generation.

In-context learning *In-context learning* (or few-shot demonstration) refers to when models are not trained on the few child-dataset examples but are provided with them at test time. In this setting, the model needs to be provided with all of the "training" instances in the context window (as part of the input) every time it needs to make a new prediction. An example of this is GPT-3, where Brown et al. (2020) write that they evaluate the model via: "in-context learning where we allow as many demonstrations as will fit into the model's context window (typically 10 to 100)".

Chapter 3

Hard Cases of Pronoun Resolution

Achieving human-like natural language understanding in LMs requires LMs to be able to do human-like commonsense reasoning. One such commonsense reasoning task is hard cases of pronoun resolution such as "*the trophy* does not fit in *the suitcase* because it is too small". What is special about hard cases of pronoun resolution is that it is one of the most challenging textual commonsense reasoning tasks (Levesque et al., 2012; Sakaguchi et al., 2020) for automated systems while being easy for humans. In this chapter, I focus on fine-tuning LMs for hard cases of pronoun resolution. I develop and compare pronoun-resolution-specific methods for data creation and model training. While focused on hard cases of pronoun resolution, these advancements can also help with understanding what commonsense reasoning knowledge is encoded in pre-trained LMs.

In this chapter, I aim to address two problems with prior work in this field. Firstly, in Section 3.2, I address the issue of the lack of large pronoun resolution datasets to serve for training models at the time, which has limited the development in the area (Opitz and Frank, 2018; Trinh and Le, 2018). For this, I demonstrate supervised learning of LMs with large new automatically generated pronoun resolution datasets. Secondly, in Section 3.3, I address the issue of having many ways to reformulate the pronoun resolution task for training (training objectives) (Opitz and Frank, 2018; Kocijan et al., 2019b,a; He et al., 2019; Wang et al., 2019b; He et al., 2019; Klein and Nabi, 2019; Sakaguchi et al., 2020), but no comparison in existing literature at the time that determines which one is the best. For this, I classify all categories of training objectives and compare them on equal grounds.

The first part of this chapter (Section 3.2) is based on my proof-of-concept work on the Winograd Schema Challenge (Levesque et al., 2012), which inspired two works that I co-authored: *A Surprisingly Robust Trick for the Winograd Schema Challenge* (Kocijan et al., 2019b) published at the ACL 2019 conference and *WikiCREM: A*

Large Unsupervised Corpus for Coreference Resolution (Kocijan et al., 2019a) published at the EMNLP 2019 conference. In this work, I create large automatically-generated datasets for pronoun resolution and introduce and compare loss functions for training conditional LMs on those datasets. I demonstrate that the constructed datasets and the loss functions can be used to fine-tune LMs to match the state-of-the-art results on the Winograd Schema Challenge while using ≈100 times fewer parameters.

The second part of this chapter (Section 3.3) is based on my work: *Does the Objective Matter? Comparing Training Objectives for Pronoun Resolution* (Yordanov et al., 2020) published at the EMNLP 2020 conference. In this work, I consider all known reformulations of the pronoun resolution problem, then I create the training objectives that correspond to them, and determine which one produces the most reliable and best results. I show that the best training objective depends on whether we aim for good in-domain performance, or good generalisability to other datasets.

3.1 Literature Review

The structure of this literature review is as follows: first, I introduce the Winograd Schema Challenge, which is the focus of this chapter, then I overview the two main categories of approaches to solving it: non-NN (knowledge or rule-based) and NN approaches, and finally, I list the main datasets and evaluations of pronoun resolution.

Introduction of the WSC The Winograd Schema Challenge (WSC) (Levesque et al., 2012) for hard cases of pronoun resolution was introduced as a modern benchmark for evaluating the natural language understanding of automated systems. It was presented as an alternative to the Turing test, and remained a hard challenge for many years (Sakaguchi et al., 2020; Kocijan et al., 2020, 2023). While theoretically, the average human should be able to correctly answer nearly 100% of the examples in the WSC dataset, in practice, some researchers show that the real number is lower – between 92.1% (Bender, 2015) and 96.5% (Sakaguchi et al., 2020).

There have been numerous attempts at this challenge, with the focus shifting over the years from using linguistic features and external knowledge (Rahman and Ng, 2012; Peng et al., 2015; Sharma et al., 2015; Emami et al., 2018) to using deep neural networks (Trinh and Le, 2018; Kocijan et al., 2019b; Sakaguchi et al., 2020).

Non-Neural-Network Approaches The first approaches to solving the WSC are knowledge-, rule- and feature-based, and do not involve NNs.

Rahman and Ng (2012) use many linguistic features to create feature vectors for each candidate substitution and rank them with support vector machines (SVMs) (Joachims, 2002). To train their model, they construct the DPR pronoun resolution dataset (detailed in Section 3.2.3).

Peng et al. (2015) use external knowledge from multiple sources to obtain scores that indicate how likely it is for the pronoun to refer to a given candidate. They improve the then-SOTA performance on the DPR (Rahman and Ng, 2012) dataset.

Sharma et al. (2015) combine semantic parsing, commonsense knowledge extraction and graph-based reasoning to a limited set (71 out of 282) from the WSC.

A good illustration of the limitations of formal reasoning is provided by Bailey et al. (2015), who make partial progress on WSC by showing that some examples of WSC can be solved via their proposed "correlation calculus". As the authors admit, many Winograd Schemas may be able to be solved via correlation calculus with "appropriate axioms", but that "obtaining such axioms is a major problem, of course" (Bailey et al., 2015).

Isaak and Michael (2016) use knowledge retrieved from Wikipedia and a modified formal reasoner to display confidence levels, which correctly resolved 107 out of 286 examples from WSC.

Emami et al. (2018) improve the then-SOTA on WSC by extracting relevant knowledge from the Web to form "evidence sentences", and resolving them via a coreference resolver, thus resolving the original schema.

Neural Network Approaches There are many neural network (NN) approaches for the WSC, with the more recent ones incorporating pre-trained LMs such as GPT, BERT, RoBERTa, and T5 (Kocijan et al., 2019b,a; Klein and Nabi, 2019; He et al., 2019; Sakaguchi et al., 2020; Khashabi et al., 2020; Lin et al., 2020; Lourie et al., 2021).

The first attempt to use NNs to the WSC is by Liu et al. (2017), who use knowledge-enhanced embeddings (KEE) in two networks: one uses an unsupervised semantic similarity method (USSM), whereas the other is supervised and uses an NN trained on OntoNotes (Pradhan and Ramshaw, 2017). They achieve only slightly above-chance results on WSC (52.8% accuracy). Opitz and Frank (2018) introduce the sequence ranking training objective which ranks which coreference candidate

yields a better-substituted sentence, and use the DPR dataset (Rahman and Ng, 2012) to train models to obtain slightly above-chance accuracy on WSC (54%).

The first successful NN approach for the WSC is via LSTM RNN LM ensemble by Trinh and Le (2018) (Google Brain). This is also the SOTA approach on WSC (63.7% accuracy) at the time of my preliminary work on solving the WSC (Section 3.2). Trinh and Le (2018) take a direct approach to the problem by replacing the pronoun in question with all of the alternative coreferent phrases, and picking the one that yields the highest sentence probability according to their massive pre-trained ensemble of LMs.

Wang et al. (2019b) propose the UDSSM model, which uses a bidirectional LSTM RNN to encode each statement as a combination of a contextual representation of the candidate and a contextual representation of the pronoun. They collect synthetic pronoun resolution training data from Project Gutenberg with a similar procedure to Kocijan et al. (2019b) (concurrent work), and show improved performance on PDP and WSC.

Kocijan et al. (2019b) set a new SOTA on WSC by fine-tuning the BERT (Devlin et al., 2019) LM on synthetic pronoun resolution data from Wikipedia, and on the high-quality DPR (Rahman and Ng, 2012) dataset. They show that the DPR dataset is large enough to achieve successful transfer learning with BERT, and that it is much more impactful than the synthetic dataset. Still, combining the two training datasets produces the best results.

Kocijan et al. (2019a) improve the synthetic pronoun resolution procedure by Kocijan et al. (2019b), and set a new SOTA in six pronoun resolution datasets, while slightly under-performing the SOTA on WSC.

Klein and Nabi (2019) propose the Maximum Attention Score (MAS) which utilises the attention in the pre-trained BERT model to achieve more accurate zero-shot classification on the WSC and PDP (Morgenstern et al., 2016) datasets.

He et al. (2019) propose the Hybrid Neural Network (HNN) model which jointly optimises two objectives: the probability of substituting the correct candidate according to the masked LM (BERT); and the semantic similarity between the candidate and the pronoun in the same BERT model. They train the HNN model on the DPR (Rahman and Ng, 2012) dataset and improve the SOTA results on WSC, WNLI and PDP.

Sakaguchi et al. (2020) improve the SOTA on WSC by introducing and training on the WinoGrande dataset – a large, high-quality pronoun resolution dataset. They

fine-tune the RoBERTa (Liu et al., 2019b) LM on WinoGrande to achieve SOTA results on WSC, PDP, DPR and other pronoun resolution datasets.

My work in Section 3.3 (Yordanov et al., 2020) uses the WinoGrande training dataset to compare the existing training objectives for pronoun resolution. These are: binary candidate prediction (Kocijan et al., 2019b,a; He et al., 2019); co-reference via semantic similarity (Wang et al., 2019b; He et al., 2019); sequence ranking after pronoun substitution (Opitz and Frank, 2018; Sakaguchi et al., 2020); and using transformer attentions (Klein and Nabi, 2019).

The WinoGrande (Sakaguchi et al., 2020) dataset establishes a new benchmark for hard cases of pronoun resolution, and many subsequent works choose WinoGrande rather than WSC as their benchmark. Khashabi et al. (2020) propose the UnifiedQA method, which involves joint training on many tasks rephrased in a unified question-answering format. They show $\approx 1\%$ accuracy improvement on WinoGrande versus fine-tuning the base LM (T5 (Raffel et al., 2020) or BART (Lewis et al., 2019)) on WinoGrande only, thus improving the SOTA. Lin et al. (2020) reformulate the pronoun resolution problem as natural language inference (NLI), and use the T5-3B (Raffel et al., 2020) LM to obtain a new SOTA on the WinoGrande dataset. Lourie et al. (2021) propose the universal commonsense reasoning model (UNICORN) which fine-tunes the T5-11B (Raffel et al., 2020) LM in a multi-task manner on five tasks from their proposed Rainbow benchmark to obtain a new SOTA on WinoGrande. Brown et al. (2020) report that the GPT-3 model achieves impressive zero-shot, one-shot and few-shot results on WSC and WinoGrande. GPT-4 (OpenAI, 2023) also improves the 5-shot SOTA on WinoGrande.

Newer approaches do not centre around WSC or WinoGrande, but rather evaluate their models on benchmarks with a wide range of tasks such as SuperGLUE (Wang et al., 2019a), which includes the WSC as a task. Some of the better-performing works on WSC in SuperGLUE include: a method for unsupervised training data generation (UDG) (Wang et al., 2021) from GPT-3; the knowledge-enhanced ERNIE 3.0 LM (Sun et al., 2021); METRO (Bajaj et al., 2022) – an architectural improvement to the ELECTRA (Clark et al., 2020) denoising LM; switch transformers (Fedus et al., 2022) in a mixture of experts (ST-MoE) (Zoph, 2022); and the Vega v2 LM (Zhong et al., 2022), which masks the most important tokens for pre-training the masked LM.

The Winograd Schema Challenge has been pronounced "defeated" (Kocijan et al., 2023), because state-of-the-art supervised models match or outperform humans on it.

However, WinoGrande has been proven to be more challenging and is still being used for few-shot LM evaluation, e.g. for GPT-4 (OpenAI, 2023).

Other Datasets and Evaluations Following the introduction of the WSC dataset (Levesque et al., 2012), many datasets of hard cases of pronoun resolution have been introduced, both for training and evaluation.

The *PDP* evaluation dataset consists of 122 (formed of two smaller datasets of size 60 and 62) pronoun resolution examples extracted from literature and news sources by the organisers of the Winograd Schema Challenge (Morgenstern et al., 2016).

The *DPR* (Rahman and Ng, 2012) dataset is a pronoun resolution dataset consisting of 943 high-quality human-engineered WSC-like schemas, and it has both a training and evaluation portion.

The *GAP* (Webster et al., 2018) dataset consists of 8,908 examples from Wikipedia, for which the gender pronouns (he, him, his / she, her, hers) are disambiguated by human annotators, and the dataset is balanced gender-wise.

The *WinoBias* (Zhao et al., 2018) dataset is an evaluation dataset of gender and stereotype bias in pre-trained models by constructing 3,160 pronoun resolution examples that include 40 job occupations. Similarly, the *Winogender* (Rudinger et al., 2018) dataset is a gender bias evaluation dataset consisting of 720 Winograd-style examples that mention 60 job occupations.

The *WNLI* (Wang et al., 2018) dataset (part of the GLUE benchmark) is a training and evaluation dataset of hard cases of pronoun resolution rephrased as natural language inference (NLI) problems. The training dataset of WNLI is derived from WSC, whereas the test set was provided by the authors of WSC (Levesque et al., 2012).

The *WikiCREM* (Kocijan et al., 2019a) training dataset is a synthetic pronoun resolution dataset consisting of 2,438,897 pronoun resolution examples extracted from Wikipedia. It improves the dataset collection procedure by Kocijan et al. (2019b), yielding much higher dataset quality.

The *KnowRef* (Emami et al., 2019) dataset consists of 8,724 pronoun resolution examples (7,455 train; 1,269 test) generated from several unstructured text sources. They achieve good data quality via a multi-step dataset creation process which finishes in quality control via human annotators.

The *WinoGrande* (Sakaguchi et al., 2020) dataset consists of 9,248 training, 1,267 development, and 1,767 test high-quality crowd-sourced examples of hard cases of

pronoun resolution. The dataset difficulty is achieved by filtering out the easier examples via a RoBERTa (Liu et al., 2019b) pre-trained LM.

The *Winventor* (Isaak and Michael, 2020) dataset consists of 848 high-quality Winograd-like schemas of hard cases of pronoun resolution constructed by an automated system from Wikipedia.

Elazar et al. (2021) introduce a zero-shot evaluation for LMs derived from the WSC, which is harder than supervision-enabled benchmarks like WinoGrande.

3.2 Solving the Winograd Schema Challenge

This section is a complete rework of my proof-of-concept work on the Winograd Schema Challenge (WSC) supervised by Oana-Maria Camburu, Ana-Maria Cretu, and Thomas Lukasiewicz, which was later developed into the two works by Kocijan et al. (2019b,a). The original proof-of-concept work was developed largely based on the idea outlined (but not implemented) by Cretu (2018) for creating a synthetic pronoun resolution dataset and using it as a training dataset for pronoun resolution (see Section 3.2.2 for more details). However, the proof-of-concept work had limited hyperparameter search and suboptimal experimental setup.

In this section, I rework that proof-of-concept work, by substantially expanding and improving its quality, ideas and results, while keeping within the tools and parameters at the time: same synthetic datasets (Wikipedia Synthetic and Gutenberg Synthetic), same LM (GPT-1), same optimiser (AdamW), same base loss function (max-margin loss (Mao et al., 2016)), and without using any ideas from subsequent works. In this rework, I introduce new ideas and modifications that substantially improve performance, the most important of which are the introduction of partial loss, the change of the model selection and early-stopping criteria, and the change of the dev evaluation dataset. For illustration of the performance uplift, I also include the best model configuration from the original proof-of-concept work as an additional baseline when reporting the final results. Furthermore, I compare against the strongest variant of the SOTA model that was available at the time, namely the 14-model ensemble version, instead of the 10-model ensemble version.

I show that the newly-trained models perform on par with the 14-model ensemble, whereas the original models only match the performance of the 10-model ensemble version on the Winograd Schema Challenge (WSC). While this work only matches the performance of the then-SOTA model (Trinh and Le, 2018) on WSC, it does so

with a much smaller model (with ≈100 times fewer parameters), which shows the utility of the synthetic pronoun resolution dataset.

3.2.1 Introduction

The Winograd Schema Challenge (WSC) (Levesque et al., 2012) is a set of hard cases of pronoun resolution that test the commonsense reasoning abilities of an NLP model. Many methods have been proposed to solve the WSC (Rahman and Ng, 2012; Liu et al., 2017; Peng et al., 2015; Trinh and Le, 2018), with the most recent method (Trinh and Le, 2018) being the SOTA (at the time) on WSC given by an ensemble of LSTM RNN LMs trained in an unsupervised manner on unstructured textual data.

In this work, I propose a method to solve the WSC that differs from previous works in four main ways:

- it trains on synthetic pronoun resolution datasets, which I generate from unstructured text (Wikipedia and Project Gutenberg) following the idea by Cretu (2018);
- it uses a superior LM architecture – the transformer (Vaswani et al., 2017), more specifically the pre-trained GPT-1 model (Radford et al., 2018);
- it uses the *max-margin* loss (Mao et al., 2016), which is used to distinguish correct from incorrect pronoun resolutions;
- apart from the standard LM loss, it uses a novel *partial loss* which optimises directly for partial scoring (a term introduced by Trinh and Le (2018))

The experiments in this work include a comparison of all loss configurations to establish the best configuration for fine-tuning GPT-1 on the synthetic datasets. I also compare the data sources (Wikipedia and Project Gutenberg) to establish which data source leads to the best performance. Furthermore, I propose a simple data filtration procedure via the pre-trained GPT model to improve the dataset quality and evaluate the benefits of using it. Finally, I compare the best model and data configurations against the then-SOTA method by Trinh and Le (2018).

3.2.2 Related Works

Related Ideas Cretu (2018) proposes the outlines of the dataset creation idea implemented in this work. Cretu (2018) further suggests numerous data sources, amongst which I have identified Wikipedia and Project Gutenberg as the most promising. Finally, Cretu (2018) proposes to use this dataset to train a "suitable sentence representation and classification method". In this work, I choose to, instead, start from an entirely pre-trained LM (GPT-1) (Radford et al., 2018).

Works Inspired by The Proof-Of-Concept This work is the proof-of-concept for two published works that I co-authored, but which are not a part of this thesis: (Kocijan et al., 2019b) and (Kocijan et al., 2019a). These works are related to my proof-of-concept work in three ways: first, the successful proof-of-concept directly motivated their creation, second, via the similar synthetic dataset creation procedure, and third, via the same base loss function (max-margin loss) used in training.

The dataset creation procedure described in this work is used to create the MaskedWiki (Kocijan et al., 2019b) dataset and adapted to create the WikiCREM (Kocijan et al., 2019a) dataset. The WikiCREM dataset is an improvement over MaskedWiki in terms of dataset quality (Kocijan et al., 2019a), which was achieved in three ways: by restricting the candidates to personal names, by extracting full sentences, and by ensuring that both candidates appear within the same sentence. As for the loss functions, Kocijan et al. (2019b) and Kocijan et al. (2019a) both use the max-margin loss function used in this work, but applied to the BERT masked LM at the output corresponding to the position of the masked pronoun.

The main difference between those two works (Kocijan et al., 2019b,a) and my proof-of-concept work is in realising that one can use the available training datasets such as DPR (Rahman and Ng, 2012) and more powerful LM such as BERT (Devlin et al., 2019) to improve performance. Moreover, those training datasets are more impactful for performance than even the higher-quality synthetic WikiCREM dataset (Kocijan et al., 2019a). Furthermore, (Kocijan et al., 2019b,a) use the more powerful BERT (Devlin et al., 2019) masked LM versus my choice of the GPT-1 (Radford et al., 2018) conditional LM. This also changes how the LM loss function is computed because the masked LM can directly provide the probability of each candidate given the full context, whereas the conditional LM can only model the conditional probability from left to right, which gives rise to the *full* and *partial* pronoun resolution losses as defined in this work.

The SOTA Method The then-state-of-the-art approach is given by Trinh and Le (2018), where they achieve 61.5% accuracy on WSC-273 (Levesque et al., 2012) via an ensemble of 10 LSTM RNN LMs. The best reported number by Trinh and Le (2018) is 63.7% accuracy on WSC-273, but it was obtained by using the sentences (minus the answers) in WSC-273 for creating the training data (they created the STORIES dataset by matching the questions in WSC-273 to web data from CommonCrawl.[1]) I consider this use of the test data in the training process to be unfair, even if it does not include the labels, but I still include it as an additional baseline. In this experimental setup, I assume that WSC-273 is a test-only dataset, which does not participate in the training in any way. The 10-model ensemble by Trinh and Le (2018) significantly outperforms their corresponding single-model result of 56.4% accuracy on WSC-273, which shows the advantage of using ensembles of models in this setting.

Other Works Concurrently to Kocijan et al. (2019b), Wang et al. (2019b) improve the SOTA set by Trinh and Le (2018) by using a similar synthetic data collection procedure as used by Kocijan et al. (2019b). However, the differences between the work by Kocijan et al. (2019b) and by Wang et al. (2019b) are many and respectively are: the different synthetic data source – Wikipedia vs Project Gutenberg; the additional use of the DPR (Rahman and Ng, 2012) training dataset vs no additional supervision data; the use of a powerful pre-trained model – BERT (Devlin et al., 2019) vs a bidirectional LSTM with pre-trained GloVe (Pennington et al., 2014) embeddings; and the use of a different loss function – max-margin loss (Mao et al., 2016) vs binary cross-entropy loss.

Loss Functions for Pronoun Resolution The only loss function that was used by others for pronoun resolution prior to this work was the default LM loss (cross-entropy loss), e.g. in the work by Trinh and Le (2018) (used as the SOTA baseline), which is the first term ($loss_{correct}$) of the max-margin loss (Mao et al., 2016) that I use (Equation 3.3). One might also consider the loss function used in the pronoun resolution work by Liu et al. (2017) who encode external knowledge by representing cause-effect pairs and encoding their Pointwise Mutual Information (PMI) (denoted sim below) via a regularisation term to the main loss function. Their formula is $h(sim(k,g) - sim(i,j)) = max(0, sim(k,g) - sim(i,j))$, which resembles the second term of the max-margin loss that I use: $max(0, M + loss_{correct} - loss_{incorrect})$ for $M = 0$, although the context is very different. I have included an ablation study

[1] `https://commoncrawl.org/`

for each term of the max-margin loss function that I use, so, in particular, in the model selection experiment (Table 3.2) I have compared max-margin loss against all pronoun resolution-related loss functions available prior to this work.

3.2.3 Method

Here, I introduce the proposed method, including dataset construction, model choice and training procedure. First, I introduce all datasets, including those from which I extract synthetic pronoun resolution data (Project Gutenberg and Wikipedia). I then detail the procedure for creating the synthetic pronoun resolution datasets that I use for training. Finally, I justify the model choice and explain the training and evaluation procedures.

Available Datasets Here I introduce all datasets that are used in this work. The WSC-273, PDP-60, and PDP-62 are used as test datasets, whereas DPR is used as both a dev (DPR-train) and test (DPR-test) dataset. The Wikipedia and Project Gutenberg text corpora are used to construct the training data for the models.

WSC-273 This is a pronoun resolution dataset from the Winograd Schema Challenge (Levesque et al., 2012) which can be found on the web as the first 273 examples on this webpage[2]. Most examples come in pairs, where the two sentences differ slightly, but have different correct answers for what the pronoun refers to. E.g.:

> *Lily spoke to Donna, breaking her silence.* Answer: *Lily.*,
>
> *Lily spoke to Donna, breaking her concentration.* Answer: *Donna.*

DPR (Definite Pronoun Resolution Dataset) This is a pronoun resolution dataset by Rahman and Ng (2012) that follows the format of the Winograd Schema Challenge (Levesque et al., 2012). It consists of 943 sentence pairs split into 1322 train and 564 test examples. In this work, I use the training dataset as a dev set (to select the best model), and use the testing dataset to report the final (test) accuracy of the selected model(s). Example sentence pair from the dataset:

> *Michael Phelps was fined by Judge Louis because he was caught with illegal drugs.* Answer: *Michael Phelps.*
>
> *Michael Phelps was fined by Judge Louis because he has to uphold federal law.* Answer: *Judge Louis*

[2] `https://cs.nyu.edu/~davise/papers/WinogradSchemas/WSCollection.xml`

PDP-60 and PDP-62[3] *(Pronoun Disambiguation Problems)* These datasets consist of pronoun resolution examples collected from various text sources and presented in the first round of the Winograd Schema Challenge (Morgenstern et al., 2016). Further description and historical context of these datasets are provided by Kocijan et al. (2020). As their names suggest, these datasets are of size 60 and 62, respectively. Example:

> *Even before they reached town, they could hear a sound like corn popping. Dora asked what it was, and Dad said it was firecrackers.* Options: *town, sound, corn.* Answer: *sound.*

Wikipedia This is raw text data from the English Wikipedia extracted from Wikimedia dumps[4][5] dated up to mid-2018. The downloaded dataset consists of more than 12GB of raw text files.

Project Gutenberg This dataset consists of all English books from the Project Gutenberg[6] repository. The downloaded dataset consists of more than 40GB of raw text files.

Synthetic Training Dataset Constructing a pronoun resolution dataset usually requires human annotation, which can be very expensive. In practice, this means that the available pronoun resolution datasets (especially at the time of creating this dataset, i.e. pre-2019) were relatively small – up to a few hundred to a thousand entries, e.g. DPR (Rahman and Ng, 2012) has 943 pairs of examples, and WSC (Levesque et al., 2012) has 273 examples. On the other hand, the available text corpora such as Wikipedia and Project Gutenberg have several orders of magnitude more unstructured data. This raised the question as to whether there is an automated procedure to create a pronoun-resolution-like dataset to help with training models to disambiguate pronouns. One such procedure (the general idea was suggested by Cretu (2018)) takes advantage of repeated nouns in the text to create synthetic examples of pronoun resolution. This procedure exploits two facts: first, it is somewhat common in a sentence to have repeated (non-pronoun) mentions of the same noun or name, e.g. "Josh and Bryan played the game where Josh gave Bryan a head start."; second, replacing a repeated mention with another noun that appears nearby can create a

[3] `https://commonsensereasoning.org/disambiguation.html`
[4] `https://dumps.wikimedia.org/`
[5] `https://dumps.wikimedia.org/enwiki/latest/`
[6] `https://www.gutenberg.org/`

sentence which is still valid, but logically false. E.g. the sentence above can become: "Josh and Bryan played the game where Bryan gave Bryan a head start.".

The full data generation procedure is as follows:

1. I clean the text so that it can be tokenised for GPT (Radford et al., 2018) – the model of choice. I also remove headers and footers from Gutenberg books, and HTML code from Wikipedia articles.

2. I tag the words as parts of speech via the Stanford POS tagger,[7] so that I can find which words are nouns and what types of nouns they are.

3. For each noun X (including proper noun, i.e. name) occurring in the dataset, I look for repetitions (X′) of it that precede it, within a certain number of word positions (window size of 15). If there is a noun repetition within the given window, a random other noun Y of the same type is chosen from the window as a possible replacement. That way I obtain two versions of the text window: the first has the original repeated noun (X′, X), while in the second I replace X with the randomly chosen alternative noun Y.

4. I extend each window with some context on both sides: up to 5 words before the start, and up to 12 words after the end, stopping if the sentence is complete.

   Example (from Wikipedia) of the process in steps 2-4. The window for noun repetitions (defined in step 3) is shown in square brackets [.]:

   Original text: "boundary is Second [Street on the north and First Street on the south. The western boundary is Pasadena Street] and the eastern boundary is Pomeroy Street"

   Constructed text: "boundary is Second [Street on the north and First Street on the south. The western boundary is Pasadena north] and the eastern boundary is Pomeroy Street"

The end result is a dataset consisting of sentence pairs, with exactly one sentence in each pair being correct.

I choose the dataset parameters after looking at the generated data. I select the values for window size (15) based on a trade-off between data size and data quality. Increasing the window size increases the dataset size because this adds examples with larger spaces between noun repetitions. However, these new examples can lead to

[7] `https://nlp.stanford.edu/software/tagger.shtml`

Model	Evaluation mode	Gut Synth	Wiki Synth	WSC-273
GPT	full	**0.89**	**0.88**	**0.56**
	partial	0.76	0.77	0.546

Table 3.1: Data quality of the synthetic datasets in terms of the accuracy obtained by the pre-trained GPT model.

lower dataset quality for the task at hand because the target WSC and DPR sentences are relatively short. I select the values for context extension before the window (up to 5) and context extension after the window (up to 12) based on a trade-off between the sentence length and the context necessary to disambiguate the pronoun. The empirical impact of these dataset parameter choices on dataset quality and model performance is very small and can be found in Appendix A.1, along with additional details. The only exception is a drop in performance when the context is shifted towards the front of the sentence – via a longer context extension before the window or a shorter context extension after the window. This is most likely because this context shift is unnatural for the evaluation dataset (DPR) where the vast majority of examples start with a candidate mention (i.e. no context before the window).

Using the procedure above, I construct one synthetic dataset from Project Gutenberg and one from Wikipedia. The resulting data sizes are listed below:

Gutenberg Synthetic: 5,504,323 pairs; *Wikipedia Synthetic:* 8,460,325 pairs

The quality of the synthetic datasets can be measured by the accuracy of the pre-trained GPT model (Radford et al., 2018) on them. Table 3.1 shows that both synthetic datasets are classified with at least 88% accuracy (via full scoring as defined in Eq. 3.1), which shows that the quality is relatively low. In comparison, the accuracy of GPT on WSC-273 (the main test dataset) is only 56%. In Appendix A I show five random examples from each synthetic dataset to further illustrate the data quality.

I also obtain higher-quality versions of the Gutenberg Synthetic and Wikipedia Synthetic datasets, by using the pre-trained GPT model to filter them. More specifically, I discard all correctly classified examples by the GPT model using full scoring (Eq. 3.1). In theory, this should leave the more difficult examples from both datasets. The sizes (below) of the resulting datasets are around $1/10^{th}$ of the original datasets:

Gutenberg Synth Filtered: 609,933 pairs; *Wikipedia Synth Filtered:* 991,302 pairs

In practice, the filtered data contains more high-quality examples, but it also includes a lot of ambiguous examples and examples with rare or foreign words which the GPT model seems to have difficulty with.

An example with relatively high quality from Gutenberg Synthetic Filtered:

but when the man had plucked the third, farmer Weatherbeard screamed so loudly that the [farmer/man] thought that brick and mortar would be rent in twain, but; *Answer:* man.

An example with foreign/rare words from Gutenberg Synthetic Filtered:

the true cross; with its main horizontal beam and the titulus, represented by a plain [beam/cross] in the cross of Lorraine. *Answer:* beam.

In Appendix A I show five random examples from each filtered dataset to further illustrate the data quality.

Model The model of choice is GPT-1 (Radford et al., 2018) which is a transformer conditional LM with 117 million parameters. While GPT-1 underperforms the then-SOTA on WSC-273 given by the Google Brain's ensemble of LSTM RNNs (Trinh and Le, 2018) on WSC-273 and PDP-60 (see Table 3.3), its architecture is much newer and has a much smaller model size than an equivalent RNN model. GPT-1 that I use has only 117 million parameters, whereas a single (not ensemble) model of the LSTM RNN has ≈1.4 billion parameters (2 layers of LSTMs, each with around 310M parameters, and an output layer of 820M parameters) i.e. more than 10 times more. The 10-model ensemble of LSTM RNNs, thus, has two orders of magnitude more parameters than the GPT-1 model that I use. Furthermore, the excellent transfer learning performance of GPT-1 (Radford et al., 2018) is particularly important for the proposed method because it involves fine-tuning an LM (GPT-1) on a (synthetic) pronoun resolution dataset.

Training and Evaluation Each training instance consists of a sentence with a gap (the pronoun) and two candidates to fill the gap: one correct and one incorrect. During training, the GPT-1 model takes as input a tokenised sentence $[w_1, \ldots, w_n]$ and produces a list of probabilities (after softmax) of each token given the previous tokens in the sentence: $[P(w_1), P(w_2|w_1), \ldots, P(w_n|w_1, \ldots, w_{n-1})]$ The probability of the sentence then becomes the product of the outputs: $P(w_1, \ldots, w_n) = P(w_1)P(w_2|w_1) \ldots P(w_n|w_1, \ldots, w_{n-1})$. The standard LM training is done by minimising the cross-entropy loss $-log(P(w_1, \ldots, w_n))$, but I make some modifications (below) to the loss function to train the model to distinguish between correctly and incorrectly substituted sentences.

I use either of two ways to predict the correct substitution: *full scoring* or *partial scoring*, as proposed by Trinh and Le (2018). Full scoring compares the probabilities

of the correctly and incorrectly substituted sentences and selects the substitution with the highest probability:

$$P(w_1, \ldots, w_{k-1}, o_{corr}, w_{k+1}, \ldots, w_n) \lesseqgtr P(w_1, \ldots, w_{k-1}, o_{incorr}, w_{k+1}, \ldots, w_n) \quad (3.1)$$

Partial scoring, on the other hand, selects the substitution that yields the highest probability of completing the remainder of the sentence that follows that substitution:

$$P(w_{k+1}, \ldots, w_n | w_1, \ldots, w_{k-1}, o_{corr}) \lesseqgtr P(w_{k+1}, \ldots, w_n | w_1, \ldots, w_{k-1}, o_{incorr}) \quad (3.2)$$

Trinh and Le (2018) have found in their experiments partial scoring often works better than the more straightforward full scoring, so I consider both in this work, at the evaluation phase. Multi-token options are treated similarly to single-token options, where all tokens of a given candidate option are inserted in the sentence instead of the pronoun, so, for simplicity, I write all options as single tokens.

I define two versions of the log-probability loss. The first is *full loss*:

$$loss^{full}(o) \coloneqq -\log(P(w_1, \ldots, w_{k-1}, o, w_{k+1}, \ldots, w_n))$$

which corresponds to the standard cross-entropy loss and the full scoring defined above (Eq. 3.1). The second is *partial loss*:

$$loss^{partial}(o) \coloneqq -\log(P(w_{k+1}, \ldots, w_n | w_1, \ldots, w_{k-1}, o))$$

which corresponds directly to the partial scoring defined above (Eq. 3.2). For each choice of $loss^{full}$ or $loss^{partial}$, I denote as $loss_{correct} \coloneqq loss^{full/partial}(o_{correct})$ the log-probability loss function for the correctly substituted sentence, and $loss_{incorrect} \coloneqq loss^{full/partial}(o_{incorrect})$ to be the log-probability loss function for the incorrectly substituted sentence.

The training loss function that I use is derived from the *max-margin loss* by Mao et al. (2016) as:

$$loss = loss_{correct} + \lambda * max(0, M + loss_{correct} - loss_{incorrect}) \quad (3.3)$$

where $\lambda > 0$ and $M > 0$ are constants that can be considered as hyperparameters. The first term of this loss function is the standard loss term which makes sure that sentences with correct substitutions have high probability $P(correct) \coloneqq P(w_1, \ldots, w_{k-1}, o_{correct}, w_{k+1}, \ldots, w_n)$. The second term, on the other hand, can

be shown to weakly enforce that the probability of the correctly substituted sentence ($P(correct)$) is greater or equal to the probability of the incorrectly substituted $P(incorrect) \coloneqq P(w_1, \ldots, w_{k-1}, o_{incorrect}, w_{k+1}, \ldots, w_n)$ times a constant factor $e^M > 1$. Here is the proof:

$$M + loss_{correct} - loss_{incorrect} \leq 0 \tag{3.4}$$

$$loss_{correct} \leq loss_{incorrect} - M \tag{3.5}$$

$$-\log(P(correct)) \leq -\log(P(incorrect)) - M \tag{3.6}$$

$$\log(P(correct)) \geq \log(P(incorrect)) + M \tag{3.7}$$

$$P(correct) \geq e^M P(incorrect) \tag{3.8}$$

In other words, the second term of Eq. 3.3 ensures that $P(correct)$ is sufficiently larger than $P(incorrect)$, which is why I use it.

The loss in Eq. 3.3 I name *single loss*. Its counterpart *double loss* is obtained by adding the cross-entropy loss of the incorrect substitution:

$$loss = loss_{correct} + loss_{incorrect} + \lambda * max(0, M + loss_{correct} - loss_{incorrect}) \tag{3.9}$$

The rationale for the $loss_{incorrect}$ term is that I want to make sure that the model does not lower $P(incorrect)$ too much because the incorrect substitution still gives a grammatically valid sentence, although logically false. This is important because I am fine-tuning the pre-trained GPT-1 model which has been pre-trained to maximise the probability of sentences.

In summary, I compare four loss functions: full single (FS), full double (FD), partial single (PS) and partial double (PD). For completeness, I also included the boundary cases $\lambda = 0$ (single cross-entropy loss) and $loss = max(0, M + loss_{correct} - loss_{incorrect})$, both with full and partial loss. Note that all losses (E.g. Eq. 3.3 and Eq. 3.9) are element-wise, not batch-wise. That is, I aggregate the batch after computing the $loss$ value for each element. That way the model can discriminate each sentence pair directly.

Training Procedure I use the pre-configured AdamW (Loshchilov and Hutter, 2017) optimiser implementation[8] that comes with the pre-trained GPT model, with a learning rate of 6.25e-5, unless specified otherwise. I use the default linear scheduler with warm-up with default settings (as used by Radford et al. (2018)), namely warm-up over 0.2% of the training data. I have also tried warm-up over 2% and 10% of the training data, but I observed similar or worse performance with those settings, hence I only consider 0.2% in the experiments here. I do a geometric hyperparameter search with factor 2 over the following intervals: $lr \in [1.95 \times 10^{-6}, 1.25 \times 10^{-4}]$, $\lambda \in [2.5, 20]$, and $M \in [6.25 \times 10^{-4}, 0.2]$. I also considered different batch sizes, but I settled on using only the batch size of 128. This is because initial results suggest that performance improves from 32 to 64 to 128 batch size, with diminishing returns, and that 64 and 128 give similar performance. The choice of loss function also introduces hyperparameters: single/double and full/partial loss. The number of training epochs is 10 for the model selection experiment and 5 for the final experiment on the large datasets.

The primary dev-evaluation dataset is DPR (train) rather than the synthetic dataset because by design DPR (train) (Rahman and Ng, 2012) is closer in distribution to the test datasets DPR (test), WSC and PDP. The metric for dev evaluation is chosen as the maximum of the full-scoring accuracy and the partial-scoring accuracy on the DPR (train) dataset because it is a-priori unclear which scoring mode is better for each loss selection. In practice, all experiments show that partial-scoring accuracy is higher than full-scoring accuracy, so this metric is equivalent to partial-scoring accuracy. Models are evaluated after every epoch based on this metric and only the model state corresponding to the best-performing epoch is reported in the final results. I do not cut short the training (i.e. early stopping) because initial experiments suggest that during training the dev performance can worsen (versus the zero-shot GPT baseline) before it gets better.

3.2.4 Experiments and Results

The experiments presented here aim to narrow down the space of loss functions and hyperparameters, and then compare the best model with the SOTA baseline by Google Brain (Trinh and Le, 2018). They also answer the question as to what extent the best models improve the performance over the pre-trained GPT model that I use as a starting point of the training procedure.

[8] `https://github.com/openai/finetune-transformer-lm`

Experimental Setup The experimental procedure is as follows:

First, I train the models with all loss configurations on a smaller version of the Gutenberg Synthetic dataset obtained by randomly subsampling it down to 200,000 training examples, which I name Gutenberg Synthetic Small. This is because of computation constraints – e.g., one training epoch for one hyperparameter combination on the full Gutenberg Synthetic dataset takes ≈37 hours with a batch size of 128 on one Nvidia Titan Xp GPU. I then discard all but the best-performing one, which I train on the large synthetic datasets (Wikipedia and Gutenberg) until the performance stops improving. The various model configurations are compared by evaluation on the DPR (train) dataset. For additional information, I report the performance on Gutenberg Synthetic (dev) – a random subset of 40,000 examples not including the 200,000 training instances. Once I have trained the models with the best configurations, I report the test accuracy on the DPR (test), WSC-273, PDP-60, and PDP-62 datasets. When testing on the WSC and PDP datasets I replace the pronoun with the given possible alternative nouns to obtain the sentence alternatives to be compared.

There are four baseline models that I compare the best model configurations against: the SOTA single-model, 10-model ensemble and 14-model ensemble by Trinh and Le (2018), and the pre-trained GPT-1 (Radford et al., 2018) model which is the starting point of training the models in this work. I also compare against the best models from the previous version of this work, which are GPT-FD-Gut-old and GPT-FD-Wiki-old.

I also construct the Wikipedia Synthetic Small dataset identically to Gutenberg Synthetic Small, with equal size, to compare the usefulness of the two datasets when accounting for data size.

I also train the best loss configuration (as selected on Gutenberg Synthetic Small) on Wikipedia Synthetic Filtered and Gutenberg Synthetic Filtered – the filtered versions of the big synthetic datasets. This aims to establish if these higher-quality filtered subsets of the training data yield good results, which can make filtration a useful procedure for large-scale low-quality data sources.

In the results below, I use the following naming scheme: first is the underlying pre-trained model, namely GPT(-1). Then two letters: F/P – full or partial loss, S/D/L/N – Single, Double, LM loss only, or No LM loss term. The final part of the name is the synthetic dataset on which the model was trained: Gut, Wiki, GutFilt, WikiFilt, GutSmall, or WikiSmall.

Results Here I present the model selection results and the final results versus the SOTA baseline. I also comment on the usefulness of the filtered datasets, and the relative usefulness of Gutenberg versus Wikipedia as a data source.

Model	Gutenberg Synthetic (dev)*	Evaluation mode	DPR (train)
GPT	–	full	0.545
GPT	–	partial	0.633
GPT-PS-GutSmall	0.871	partial	**0.655**
GPT-PD-GutSmall	0.887	partial	0.652
GPT-PL-GutSmall	0.780	partial	0.647
GPT-PN-GutSmall	0.781	partial	0.631
GPT-FS-GutSmall	**0.935**	partial	0.640
GPT-FD-GutSmall	0.920	partial	0.644
GPT-FL-GutSmall	0.920	partial	0.638
GPT-FN-GutSmall	0.928	partial	0.623
GPT-PS-WikiSmall	–	partial	**0.664**
GPT-PS-GutFilt	–	partial	0.612
GPT-PS-WikiFilt	–	partial	0.626

Table 3.2: Comparison in terms of accuracy of all model combinations trained on small versions of Gutenberg and Wikipedia Synthetic datasets. Some models trained on the filtered datasets are also included. The pre-trained GPT baseline is also included for reference. The best performance in each category is marked in bold. * - accuracy is obtained by choosing the option that gives the lower loss value.

Model Selection In Table 3.2, I compare all loss configurations on DPR (train) and, additionally, on Gutenberg Synthetic (dev) – a random subset of 40,000 examples not including the 200,000 training instances. The results show that the best-performing loss configuration on DPR (train) is partial single loss (PS) with 65.5% acc, which means that PS will be the choice for the WikiSmall, GutFilt, and WikiFilt experiments, as well as for the full-data experiments.

All partial-loss models outperform their full-loss counterparts. This makes sense since partial-loss models optimise directly for partial scoring, and the training is initiated from the GPT model which performs better with partial scoring vs full scoring: 63.3% vs 54.5%. On the other hand, the partial-loss models significantly underperform the full-loss models on Gutenberg Synthetic (dev), which shows that Gutenberg Synthetic (dev) is not a good evaluation dataset, which further justifies using DPR (train) for evaluation. For both full-loss and partial-loss models, partial scoring gives better accuracy than full scoring, which is why only partial scoring is reported in the results in Table 3.2.

Double-loss models (FD, PD) perform similarly to single-loss models (FS, PS): slightly worse for partial loss, with 65.2% acc vs 65.5% acc for PD vs PS; and slightly better for full loss, with 64.4% acc vs 64.0% acc for FD vs FS. This means that the single-loss is good enough as a default assumption for other works like (Kocijan et al., 2019b,a).

The no-LM-loss models PN and FN underperform the GPT baseline, which shows the importance of having the LM loss term. PN and FN were stopped early in the hyperparameter search since the best results are obtained after 1 epoch, and the best learning rate value converges to 0 with the hyperparameter search, i.e. no training achieves the best results.

The hyperparameter search shows that the partial-loss models overall need lower learning rates than the full-loss models: PS, PD, PL: 1.5625e-5, 1.5625e-5, 1.5625e-5; FS, FD, FL: 6.25e-5, 6.25e-5, 1.5625e-5. This is most likely because partial loss is not natural for the pre-trained GPT model, which is trained via full (LM-only) loss, hence partial-loss models need a smoother adaptation via a lower lr.

One potential limitation of max-margin loss (in PS, PD, FS, and FD) versus cross-entropy LM loss (PL and FL) is that max-margin loss introduces two additional hyperparameters: λ and M, which can increase computation costs with hyperparameter search. However, max-margin loss performs better than cross-entropy loss, and I have found the best values of those hyperparameters: $\lambda = 5$ and $M = 0.0125$, which can either be assumed or be a good starting point for hyperparameter search.

Training on Wikipedia Synthetic Small yields significantly better results on DPR (train) than the Gutenberg Synthetic Small: 66.4% vs 65.5%. This experiment accounts for the size difference between the Wikipedia Synthetic and Gutenberg Synthetic by subsampling to 200k examples, therefore, we can infer that the Wikipedia dataset is of higher quality than the Gutenberg dataset, and is of more use for training pronoun resolution models.

The models trained on the filtered data (Table 3.2) do not outperform the GPT baseline, yielding worse results than their counterparts trained on the small versions of Gutenberg Synthetic and Wikipedia Synthetic. This makes the filtered datasets not useful versus the original datasets. The bad performance can be attributed to two factors: first, the filtration procedure removes simple examples that can be resolved by the pre-trained GPT models, but this can lead to decreased performance on simple examples; and second, the filtered datasets have a high prevalence of rare words and ambiguous synthetic examples (i.e. examples that not only cannot be resolved by the

Model	Eval mode	DPR (test)	WSC-273	PDP-60	PDP-62
GGL-SM	full	–	0.538	0.6	–
	partial	–	0.564	0.53	–
GGL-E-5	full	–	–	**0.7**	–
GGL-E-10	partial	–	0.615	–	–
GGL-E-14	–	–	**0.637**	–	–
GPT	full	0.57	0.56	0.62	**0.62**
	partial	0.65	0.546	0.53	**0.62**
GPT-FD-Gut-old	full	0.576	0.56	0.58	0.607
	partial	0.626	0.59	**0.63**	0.574
GPT-FD-Wiki-old	full	0.573	0.553	0.58	0.59
	partial	**0.652**	**0.612**	0.6	0.508
GPT-PS-Gut	full	0.578	0.557	**0.617**	0.52
	partial	**0.651**	0.597	**0.617**	0.49
GPT-PS-Wiki	full	0.574	0.571	0.6	**0.59**
	partial	0.64	**0.634**	0.583	0.475

Table 3.3: Comparison in terms of accuracy on four test datasets, where each model is evaluated via full or partial scoring. GGL-SM and GGL-E are Google Brain's single-model and ensemble (SOTA) baselines (Trinh and Le, 2018). The best performance in each section is marked in bold. Some values are unavailable (-). The most important column is WSC-273.

GPT model but also cannot be resolved in general), which are not useful for DPR and WSC which have simple vocabulary.

Final Results After selecting the best model configuration, namely partial single loss (PS), I train it on the full Synthetic Gutenberg and Synthetic Wikipedia datasets and compare it against the SOTA GGL-E-10 (10-model ensemble) and GGL-E-14 (14-model ensemble) models and two other baselines (GGL-SM (single-model) and the GPT pre-trained model). To save computational resources, I also assume the best values for $\lambda = 5$ and $M = 0.0125$, and only search over the learning rate values.

The results in Table 3.3 show that PS-Wiki performs similarly (statistically insignificant difference) to GGL-E-10 and GGL-E-14 in terms of accuracy on WSC-273, which are the SOTA at the time. This means that this training approach is promising, and further improvements may outperform GGL-E, e.g. by using higher-quality data and/or training an ensemble of models.

PS-Gut and PS-Wiki outperform GPT on WSC-273, with GPT vs GPT-PS-Wiki being statistically significant ($p < 0.05$), which shows the successful transfer learning by using the synthetic Wikipedia dataset. PS-Wiki also outperformed (in terms of

accuracy on WSC-273) the PS-Gut model, which implies that the Wikipedia data is more suitable for constructing a synthetic pronoun resolution dataset than the Project Gutenberg data. This is confirmed by the results in Table 3.2, where the Wikipedia model outperforms the Gutenberg model when accounting for data size.

The new best models GPT-PS-Gut and GPT-PS-Wiki outperform the corresponding old best models: GPT-FD-Gut-old and GPT-FD-Wiki-old, but due to the small size of WSC-273, the comparison is statistically insignificant.

The PDP-60 and PDP-62 results are volatile likely due to the small data sizes. Overall, all models perform poorly on them ($< 65\%$ accuracy), which can be attributed to the sentence lengths of PDP – up to 64 words, which is much longer than DPR and WSC. Furthermore, the model selection procedure uses DPR, so these longer sentences are out-of-distribution when selecting the best model.

Although DPR (test) results do not correlate with WSC-273 results in Table 3.3 across models (the GPT baseline has similar performance to the fine-tuned models), they do seem to help when selecting the best hyperparameters for each model, as evident by the superior WSC-273 results when fine-tuning on Gut and Wiki versus the pre-trained GPT model.

Overall, partial scoring is better than full scoring for evaluating the fine-tuned GPT-based models, which matches the conclusions of the model selection experiment.

3.2.5 Conclusion

This work is a comprehensive rework of my proof-of-concept on creating a synthetic pronoun resolution training dataset for training LMs, which inspired two influential works that I co-authored (Kocijan et al., 2019b,a).

The main conclusions of this work are several. First, I have shown that extracting a large synthetic pronoun disambiguation set is feasible, although inspecting the data shows that it is of relatively low quality. Second, I have determined which data source works best for synthetic data generation, namely that Wikipedia is better than Project Gutenberg. Third, the resulting datasets are good enough as training data, as I have shown that training the GPT-1 model on it achieves similar results to the SOTA on WSC, while using a much smaller model. Finally, I have established which loss configuration achieves the best results, namely, my proposed *partial single* loss.

Additionally, I have shown that naive model-based data filtration may not be the best way to further improve data quality. Future work may develop higher-quality data collection procedures which improve upon the synthetic dataset construction

procedure in this work, similarly to the improvements introduced with the WikiCREM (Kocijan et al., 2019a) dataset. Larger data sources such as CommonCrawl (Rana, 2010) can also be considered, as well as newer and more powerful LMs.

3.3 Comparing Training Objectives for Pronoun Resolution

Here I present the results associated with my work *Does the Objective Matter? Comparing Training Objectives for Pronoun Resolution* (Yordanov et al., 2020) published at the EMNLP 2020 conference. After my proof-of-concept work in Section 3.2 and the subsequent work by Kocijan et al. (2019b,a), there have been several other proposed ways to rephrase the pronoun resolution task as a training objective (Wang et al., 2019b; Opitz and Frank, 2018), and no clear best way to do it. In this work, I compare all existing categories of training objectives by using the then-newly-released WinoGrande dataset (Sakaguchi et al., 2020), which provides a large amount of high-quality pronoun resolution data for training and evaluation. I show that the sequence ranking objective (WG-SR) (Opitz and Frank, 2018; Sakaguchi et al., 2020) outperforms all other objectives in-domain (on the same dataset distribution), but that the coreference semantic similarity objective (CSS) (Wang et al., 2019b; He et al., 2019) is the best in out-of-domain performance (on other pronoun resolution datasets).

The results in this section have been reworked from the original work (Yordanov et al., 2020) by introducing a true train/dev/test data split rather than using WG-dev as both a dev and a test dataset. Furthermore, I have expanded the hyperparameter search at the expense of using fewer seeds during the search, because previously the best hyperparameter values for all models were at the boundary of the search space. The results remain unchanged with small exceptions. The expanded hyperparameter search has led to 100% seed-wise stability for the WG-SR model (previously 10% of seeds did not perform well), the same 100% stability for the other three models, and overall lower standard deviation across seeds. Also, the results now show that both CSS and MAS outperform WG-SR out-of-domain (previously only CSS). This makes it more likely that the relatively poor out-of-domain performance of WG-SR is due to its lack of explicit candidate localization and candidate-pronoun matching that is present in CSS and MAS. Previously the reason was unclear.

3.3.1 Introduction and Related Works

Hard cases of pronoun resolution have been a long-standing problem in natural language processing, which has served as a performance benchmark for the research community (Levesque et al., 2012; Wang et al., 2018, 2019a). For example, the WinoGrande dataset (Sakaguchi et al., 2020) consists of pronoun resolution schemas that are constructed so that resolving them requires background knowledge and commonsense reasoning. In WinoGrande, the pronoun is obscured by "__" to remove gender and number cues. The task is to find the correct candidate for "__" out of two given candidates. For example:

> *John moved the couch from the garage to the backyard to create space. The __ is small.* Candidates: garage, backyard.

Recently, supervised learning on top of pre-trained LMs has been established as the main approach for pronoun resolution (Kocijan et al., 2019b,a; Sakaguchi et al., 2020). Under this type of approach, I identify four categories of objectives commonly used for pronoun resolution:

1. comparing the LM probabilities for each candidate (Kocijan et al., 2019b,a; He et al., 2019),

2. using semantic similarity between the pronoun and the candidates (Wang et al., 2019b; He et al., 2019),

3. using sequence ranking among the possible substituted sentences (Opitz and Frank, 2018; Sakaguchi et al., 2020), and

4. selecting a candidate based on the attentions of the pronoun in a transformer model (Klein and Nabi, 2019).

I list one representative model from each category. For 1, Kocijan et al. (2019b) use the BERT masked LM (Devlin et al., 2019) to produce the probabilities of the pronoun to be replaced with each of the two candidates. For 2, the Unsupervised Deep Structured Semantic Model (UDSSM-I) (Wang et al., 2019b) uses contextualized word embeddings produced by a bidirectional recurrent neural network (BiRNN), and then compares the word embedding of each candidate with the word embedding of the pronoun. For 3, RoBERTa-WinoGrande (Sakaguchi et al., 2020) encodes a pair of sentences (one for each candidate substituted in the input) by using RoBERTa (Liu et al., 2019b) to determine which substitution is the correct one. Finally, the

zero-shot Maximum Attention Score (MAS) model (Klein and Nabi, 2019) selects a candidate based on how much the pronoun attends to each candidate internally in BERT.

The problem with all these objectives is that they have not been introduced under the same circumstances. They use different LMs and word embeddings (e.g., BERT, RoBERTa, or BiRNN), and have been trained on different data (e.g., DPR (Rahman and Ng, 2012), WinoGrande, or no additional data). Therefore, it is unclear whether the choice of the objective function is essential for pronoun resolution tasks. Moreover, the seed-wise stability and the expected performance of these models have usually not been reported. However, seed-wise instability and performance variation are well-known problems when fine-tuning transformer-based models (Liu et al., 2020; Dodge et al., 2020).

In this work, I compare the performance and seed-wise stability of the four categories of training objectives for pronoun resolution on equal grounds. To do this, for category 4, I adapt to training the zero-shot MAS model. For category 2, I also introduce Coreference Semantic Similarity (CSS), which is a simplification and modification of UDSSM-I for transformer encoders. I select WinoGrande as the training and development dataset due to its large size (40,938 examples) and generalizability to other pronoun resolution tasks (Sakaguchi et al., 2020). For testing, I also use the following well-established datasets: the Winograd Schema Challenge dataset (WSC) (Levesque et al., 2012) and the Definite Pronoun Resolution dataset (DPR) (Rahman and Ng, 2012). I choose as LM RoBERTa (Liu et al., 2019b), as it significantly outperforms BERT on WinoGrande, WSC, and DPR (Sakaguchi et al., 2020).

Finally, the model comparison is done under an unprecedentedly large number of seeds – 20 (previously up to 5 seeds (Opitz and Frank, 2018)). This enables a more accurate assessment of the per-seed variation of results and the expected average performance of each model.

3.3.2 Method

This section presents the four training objectives and the models[9] that represent each of them.

All four models share the RoBERTa[10] contextualized word embeddings. RoBERTa has an identical transformer architecture to BERT (Devlin et al., 2019), with the

[9]The code is publicly available at: `https://github.com/YDYordanov/WS-training-objectives`.

[10]*roberta-large* (Wolf et al., 2019)

only difference being the training procedure. Hence, RoBERTa is a masked LM that outputs the probability distribution for filling a gap in the text (denoted by a "<mask>" token). Additionally, RoBERTa is a text encoder, with one output for each token of the input sentence. Three of the models (WG-SR, CSS, and MAS, as defined below) use a multi-layer perceptron (MLP) classification "head", which takes some part of the encoder as input.

All four models use binary cross-entropy loss with a pair of probabilities as input, and the following target labels: sentence correctness for WG-SR and candidate correctness for BWP, CSS, and MAS.

WinoGrande Sequence Ranking (WG-SR) I refer to the RoBERTa-WG model introduced by Sakaguchi et al. (2020) as WG-SR, since it has a sequence ranking objective. This model predicts which sentence of a pair of substituted sentences is more plausible. Each sentence of the pair of sentences in the input of WG-SR is split in two before the substituted candidate. For example,

> <s> The city councilmen refused the demonstrators a permit because </s> </s> __ feared violence. </s>,

where "__" is filled with each of the two candidates: "the city councilmen" or "the demonstrators".

The WG-SR code[11] is based on the RobertaForMultipleChoice model (Wolf et al., 2019), restricted to a binary choice. This model consists of the pre-trained RoBERTa encoder and an MLP head based on the <s> (first) token of RoBERTa's output. The MLP has one hidden layer with tanh activation, hidden size matching that of the encoder, and one-dimensional output. The pair of input sentences $(\mathrm{S}_1, \mathrm{S}_2)$ thus produces a pair of values, which are then passed through a softmax to obtain the two sentence probabilities $P(\mathrm{S}_1)$ and $P(\mathrm{S}_2)$.

Binary Word Prediction (BWP) I denote by Binary Word Prediction (BWP) the model suggested by Liu et al. (2019b) in their code repository[12] as a modification of the model from Kocijan et al. (2019b). Instead of using margin loss, BWP uses binary cross-entropy loss. I selected this modified version, because it is claimed to yield "slightly better (and more robust) results" by its authors (92.3% vs 90% accuracy on their development set), and it also has two fewer hyperparameters.

[11] `https://github.com/allenai/winogrande`
[12] `https://github.com/pytorch/fairseq/tree/master/examples/roberta/wsc`

For a given (unsubstituted) input sentence, the BWP model estimates which of the two candidates is more likely to fill the gap "__". The input format is like in the following example, where "__" is replaced by the "<mask>" token, to serve for the masked LM:

> <s> The city councilmen refused the demonstrators a permit because <mask> feared violence. </s>

With such an input, the RoBERTa masked LM returns the log-probability predictions at the "<mask>" token over the vocabulary. Of those predictions, only the ones corresponding to the two coreference candidates c_1 and c_2 are selected by BWP: $\log P_{\text{vocab}}(c_1)$ and $\log P_{\text{vocab}}(c_2)$. Here, the log-probability of each candidate is defined by averaging the log-probabilities of its tokens. Then, softmax is computed with inputs $\log P_{\text{vocab}}(c_1)$ and $\log P_{\text{vocab}}(c_2)$, which is how I define the pair of probabilities:

$$(P(c_1), P(c_2)) := \left(\frac{P_{\text{vocab}}(c_1)}{P_{\text{vocab}}(c_1) + P_{\text{vocab}}(c_2)}, \frac{P_{\text{vocab}}(c_2)}{P_{\text{vocab}}(c_1) + P_{\text{vocab}}(c_2)} \right)$$

Coreference Semantic Similarity (CSS) I propose Coreference Semantic Similarity (CSS), a modification of the training objective of the Unsupervised Deep Structured Semantic Model (UDSSM-I) (Wang et al., 2019b). Like UDSSM-I, the CSS objective works by comparison in the word embedding space, such that the candidate that is more similar to the embedding of the pronoun is selected. Unlike UDSSM-I, the CSS objective is simpler, with no attention weights on the tokens of the candidates. It also uses a transformer encoder instead of a recurrent neural network, which enables it to take advantage of state-of-the-art pre-trained LMs.

The input format for this model is the same as for BWP (above). This input is used by RoBERTa to produce contextualized word embeddings. For each candidate c, I define its contextualized word embedding $\text{emb}(c)$ by averaging the contextualized word embeddings of its tokens.

For classification, I compare the similarity scores of the embeddings of the <mask> token with each of the two candidates c_1 and c_2, i.e., I compare:

$$\text{sim}(\text{emb}(c_1), \text{emb}(\text{<mask>})) \text{ and } \text{sim}(\text{emb}(c_2), \text{emb}(\text{<mask>}))$$

and select the candidate with greater similarity.

For the similarity score function, I use *additive alignment* (Bahdanau et al., 2014), i.e., $\text{sim}(x, y) := v^\top \tanh(Wx + Uy)$, with the trainable parameters: vector v, and matrices W and U, with hidden size equal to that of RoBERTa and output size of

one. I choose additive alignment because it is more expressive (with more trainable parameters and an activation function) than all similarity measures used in prior pronoun resolution works, namely the ones by Wang et al. (2019b): cosine similarity ($\text{sim}(x, y) := \frac{x^T y}{\|x\|\|y\|}$), multiplicative similarity ($\text{sim}(x, y) := (Wx + b)^T y$), and additive similarity ($\text{sim}(x, y) := ax + by$). Furthermore, performing the same hyperparameter search as in Section 3.3.3 for all four similarity measures shows that additive alignment outperforms cosine similarity on the dev set, with 94.3% vs 93.4% accuracy, respectively, and slightly outperforms multiplicative similarity and additive similarity, with 94.3% vs 94.1% accuracy for both.

During training, $\text{sim}(\text{emb}(c_1), \text{emb}(<\text{mask}>))$ and $\text{sim}(\text{emb}(c_2), \text{emb}(<\text{mask}>))$ are fed to a binary softmax function to obtain $P(c_1)$ and $P(c_2)$.

Maximum Attention Score (MAS) The Maximum Attention Score (MAS) model was originally developed for zero-shot evaluation of transformer models on pronoun disambiguation (Klein and Nabi, 2019). It uses the attentions of all layers of a transformer model to produce a maximum attention score for each candidate that summarizes how much the pronoun attends to a candidate. The candidate that is most attended is selected. I adapt this objective to be trainable by replacing the summary of attentions with an MLP over the concatenated masked attention tensors, followed by a binary classifier.

The input of MAS is the same as for BWP (above). Then, similarly to Klein and Nabi (2019), I extract the two attention tensors A_{c_1} and A_{c_2} given by the multi-layer RoBERTa attentions of the "<mask>" token to each of the two candidates c_1 and c_2, respectively. For each candidate c, the attention tensor A_c is defined as the average of the attention tensors of all tokens that form c. The two corresponding max-masking tensors M_{c_1} and M_{c_2} are then derived as follows: for $i = 1, 2$ and for each multi-index j of the tensor A_{c_i}, I set $M_{c_i}(j) = 1$, if $A_{c_i}(j) \geq A_{c_{3-i}}(j)$, and $M_{c_i}(j) = 0$, otherwise. I obtain the two corresponding max-masked tensors by the element-wise products: $B_{c_1} = A_{c_1} \circ M_{c_1}$ and $B_{c_2} = A_{c_2} \circ M_{c_2}$.

Unlike Klein and Nabi (2019), I introduce an MLP on top of the concatenated tensor $B = [B_{c_1}, B_{c_2}]$ for binary classification. This is because the method by Klein and Nabi (2019) is an evaluation-only method for pre-trained LMs, and a classification head is usually used for fine-tuning LMs, e.g. (Liu et al., 2019b). The MLP has two hidden layers, tanh activation, hidden size the same as its input, and two-dimensional output. The choice of 2-layer MLP allows for sufficient expressivity of the model,

while adding negligible memory and computational overhead versus the pre-trained LM (which has 24 much larger hidden layers).

The MLP is followed by a binary softmax function to produce the two candidate probabilities $P(c_1)$ and $P(c_2)$.

3.3.3 Experiments

For all four models, I select the best hyperparameters via grid search and then train the models with the best hyperparameters on 20 seeds. For WinoGrande, I split WG-train-XL randomly into a train and dev dataset (39,130 and 1,268 examples, resp.), and use the original WG-dev dataset (1,267 examples) as a test set. This is because there are submission limitations (maximum one per week) of the WinoGrande leaderboard,[13] which prevent me from reporting all 80 trained models on the original test set (WG-test). I also report all models on the out-of-domain pronoun resolution datasets WSC (273 examples) and DPR (564 examples). The candidates provided in WSC were treated differently for the CSS and MAS models, as these models require precise candidate localization (see Appendix B.2).

For all four models, I do a grid search over the learning rate values $\{2.5e-6, 5e-6, 1e-5, 2e-5, 4e-5\}$, the number of training epochs $\{4, 6, 8, 10, 12\}$, and the batch-size values $\{8, 16, 32\}$. This hyperparameter space is selected based on the union of the grid search by the original WG-SR work (Sakaguchi et al., 2020) and my observations on the other three models. The best hyperparameters (in Appendix B.1) are selected based on the maximum dev accuracy across the hyperparameter space.

For all experiments, I use linear learning rate decay with warm-up over 10% of the training data, and the AdamW optimizer (Wolf et al., 2019), for which I only alter the learning rate.

3.3.4 Results

Table 3.4 shows the final seed-wise results for all four objectives. We see that the semantic similarity objective (CSS) is the best-performing training objective on out-of-domain testing, with 90.7% average accuracy on WSC and 93.1% average accuracy on DPR. CSS clearly outperforms both WG-SR and BWP on WSC and DPR, while performing on par with MAS on WSC (90.7% vs 90.8% average accuracy) and outperforming MAS on DPR (93.1% vs 92.6% accuracy). On the other hand, the sentence

[13] `https://leaderboard.allenai.org/winogrande/submissions/public`

Model	WG-dev	WSC	DPR
WG-SR	**78.2** (0.68)	89.3 (1.04)	92.1 (0.73)
BWP	76.0 (0.53)	89.9 (0.84)	92.0 (0.45)
CSS	76.6 (0.47)	90.7 (1.0)	**93.1** (0.43)
MAS	76.2 (0.7)	**90.8** (1.09)	92.6 (0.49)

Table 3.4: Seed-wise aggregated performance of models on WG-dev, WSC, and DPR, across all 20 seeds. The number format is: average accuracy in %, and standard deviation (in parentheses). The best performance is marked in bold.

Model	Maximum	Average	Standard deviation	Number of well-performing
WG-SR	94.2	92.4	1.04	46 out of 75
BWP	93.8	92.5	0.75	64 out of 75
CSS	**94.3**	**92.7**	3.33	63 out of 75
MAS	94.1	92.4	1.26	70 out of 75

Table 3.5: Performance of all four models on the dev set aggregated across all 75 hyperparameter combinations. The numbers in the first three columns are: maximum accuracy in %, average accuracy in %, and standard deviation. Only the well-performing models (with at least 60% accuracy) are reported, and their number is in the last column. The best performance is marked in bold.

ranking objective used by WG-SR clearly outperforms the other three objectives on in-domain testing, with 78.2% average accuracy on WG-dev.

In order to verify the statistical significance of the main results, I used the paired-samples t-test to compare the distributions of accuracy across the 20 seeds. Comparing the accuracies of CSS and WG-SR on WG-dev, WSC, and DPR, respectively, I get the following two-tailed p-values: $7.72e-8$, $8.95e-4$, and $2.17e-5$. For CSS vs BWP on WSC, and CSS vs MAS on DPR, respectively, I get the following two-tailed p-values: $6.75e-3$ and $2.08e-3$. All results are significant with $p < 0.05$.

All models perform well on all 20 seeds. However, during the hyperparameter search, I observed that all models were prone to not performing well for certain combinations of hyperparameters. The performance threshold that I used was selected as having $\leq 60\%$ accuracy on WG-dev, and its value was selected based on the performance distribution of all models. I observed that all models either perform around 50% accuracy (chance) or 70% accuracy or more on the dev set. 60% in this context is a good middle-ground threshold. Table 3.5 shows that MAS performed well most often, with 70 out of 75 hyperparameter combinations. Out of the four models, WG-SR performed well least often, for only 46 out of all 75 hyperparameter combinations.

CSS performs the best across the well-performing hyperparameter combinations, but has the most variation in performance, with a 3.33 standard deviation. Please note that the numbers in Table 3.5 are much higher than the results on WG-dev in Table 3.4 because the dev set that I use is subsampled from the original WinoGrande training dataset, which is easier to solve than WG-dev (WG-dev was constructed to be harder (Sakaguchi et al., 2020)).

WG-SR likely performs better in-domain than CSS, MAS, and BWP, since those three use existing properties of RoBERTa (such as the possibility to compare contextualized embeddings, the attention structure of the model, and its pre-trained LM prediction head, respectively) for a task that they were not originally designed for (pronoun resolution). WG-SR, on the other hand, only uses the output of RoBERTa at the 0-th token, which is not pre-trained.

I identify two possible reasons why WG-SR performs worse than CSS on out-of-domain examples. The first reason is the one mentioned above, namely, not explicitly exploiting the listed properties of the pre-trained model would lead to a better fit on a specific dataset, but worse "general knowledge". This reason is not the full explanation, since WG-SR has similar performance to BWP on DPR. The second possible reason is that CSS uses explicit candidate localization and candidate-pronoun matching (by comparing the embedding of the candidate and the pronoun), whereas in WG-SR these are achieved implicitly by feeding a pair of sentences to the model, one with the correct and one with the incorrect substitution. This reason is more likely because MAS also uses explicit candidate localization and candidate-pronoun matching, and it outperforms WG-SR out-of-domain.

3.3.5 Conclusion

In this work, I categorized four existing objectives for pronoun resolution and compared their performance and seed-wise stability and performance variation on equal grounds. The experiments showed that, on in-domain testing, the objective of sequence ranking based on the first token in RoBERTa outperforms the other three objectives. On out-of-domain testing, the objective of semantic similarity between the pronoun and each candidate performs the best among the four objectives.

Future work may investigate whether these results translate to other LMs besides RoBERTa as well as other training datasets besides WinoGrande. Also, one could analyze the strengths and weaknesses of each objective, and evaluate other variations of these objectives.

3.4 General Conclusion

In the first work in this chapter, I have demonstrated the creation and successful use of synthetic pronoun resolution datasets for training LMs. I have examined multiple loss function variations and determined the best one for pronoun resolution in this setting. The results show that my best model matches the then-SOTA performance while using a much smaller model size. The results further show that data quality may be the key to further improvements in performance and that model-based filtration may not be the way to do it.

In the second work in this chapter, I have identified all existing categories of training objectives for pronoun resolution, and have explored the benefits and downsides of each. To achieve this, I have trained all models on the large high-quality WinoGrande dataset for pronoun resolution, and have compared them both in terms of performance and stability. The results suggest that there is a clear best training objective for in-domain performance (evaluating on the test portion of the same dataset), which differs from the best objective for out-of-domain performance (evaluating on a different dataset).

Chapter 4

Transfer Learning of Natural Language Explanations

Achieving human-like natural language understanding in LMs also requires LMs to be able to explain their decisions like humans. Such explanations in human language are commonly referred to as *natural language explanations* (NLEs). In this chapter, I focus on the transfer learning of NLEs. One domain of interest is creating NLEs for hard cases of pronoun resolution (from Chapter 3), because this is both a hard task and lacks high-quality NLEs.

The core part of this chapter (Section 4.2) is based on my work: *Few-Shot Out-of-Domain Transfer Learning of Natural Language Explanations in a Label-Abundant Setup* (Yordanov et al., 2022) published at the Findings of the Association for Computational Linguistics: EMNLP 2022. The paper, as the name suggests, is focused on transferring NLEs from a domain with abundant labels and NLEs to a domain with abundant labels, but only a few NLEs. I have demonstrated that a successful transfer of NLEs is possible from the e-SNLI (Camburu et al., 2018) – a large natural language inference dataset with NLEs, to two other datasets. The first dataset is WinoGrande (Sakaguchi et al., 2020) – a dataset of hard cases of pronoun resolution, where I introduce a small dataset of high-quality NLEs. The second one is ComVE (Wang et al., 2020) – a dataset of commonsense validation (answering the question *Does a statement make sense or not?*), where I restrict the number of available NLEs. Apart from demonstrating the feasibility of NLE transfer learning, I study the performance of the different ways to use multi-task learning and fine-tuning for achieving NLE transfer. The results suggest that the best method depends on the underlying task (hard cases of pronoun resolution or commonsense validation), but that the few child-task NLEs should be on a separate training regime.

The last part of this chapter (Section 4.3) presents additional experiments on the scalability of the best methods found in the previous part (Section 4.2). The results show that those methods scale overall well in terms of data and model size, but that the models transition from relying on the parent task to relying on the child task when using more training NLEs.

4.1 Literature Review

In this section, I introduce the existing literature on NLEs. I present the three main directions of research: producing high-quality NLEs, NLE faithfulness and transfer learning of NLEs. I also summarise the literature on NLE evaluation, data annotation and the existing NLE datasets.

Introduction and First Methods Natural language explanations (NLEs) have two main types of applications. In commercial applications, NLEs are important for building appropriate trust with the users of the deployed system, which any production model can benefit from. In research and model evaluation, explanations provide reassurance that the model understands the task at hand and does not take shortcuts. There are two main desirable properties of the generated NLEs: perceived quality (as judged by humans) and NLE faithfulness (Jacovi and Goldberg, 2020). *Faithful* NLEs are those that are true to the reasoning process of the model, whereas perceived NLE quality is a surface-level characteristic.

The general field of black-box model interpretability (Simonyan et al., 2013) precedes the invention of models capable of generating NLEs (Hendricks et al., 2016). Examples of architecture-agnostic model interpretation techniques include integrated gradients (Sundararajan et al., 2017) which show the contribution (weights) of each part of the input to the model's prediction, and influence functions (Han et al., 2020) which "explain the decisions of a model by identifying influential training examples". This is still an active field of research because most approaches allow for the analysis of models without the need for additional model modifications or additional training data, which may be needed for generating NLEs. Models that use attention provide an additional interpretability mechanism given by what the model attends to in the input when predicting a given output – e.g. it can show the word alignment between two languages in machine translation (Bahdanau et al., 2014).

Rationales are a broad category of model explanations, but in this chapter, I only consider extractive rationales, i.e. the subsets of the input that best explain the

decisions of the model. E.g.: "Jacob won a game against Philip. Who scored the most points? *Answer*: Jacob. *Rationale*: Jacob won.". In comparison, NLEs are not a list of words or phrases, but rather, refer to complete sentences – the same way a person would explain their decision.

Most of the initial work on NLE generation was in the computer vision domain – e.g. Hendricks et al. (2016) generate NLEs for classifying bird species from photos. This is likely because the first GPU-powered NN breakthrough came to computer vision rather than NLP, with the entry to the ImageNet competition by AlexNet (Krizhevsky et al., 2012). On the other hand, the GPU-powered NN breakthroughs in NLP came later, with the CBOW and SkipGram word-embedding models (Mikolov et al., 2013). One of the first works on NLEs for natural language processing was by Ling et al. (2017a) who trained a model to answer simple mathematical problems and to provide step-by-step solutions in natural language.

There are three main focuses in NLE generation: quality improvement (Camburu et al., 2018; Narang et al., 2020; Valentino et al., 2022), NLE faithfulness (Kumar and Talukdar, 2020; Wiegreffe et al., 2021; Liu et al., 2019a; Latcinnik and Berant, 2020; Atanasova et al., 2023), and transfer learning of NLEs (Erliksson et al., 2021; Yordanov et al., 2022).

Generating High-Quality NLEs These are the most notable approaches to generating high-quality NLEs. These works usually assume that a training dataset with NLEs exists, and it is used to train models to generate NLEs on the dev/test portions of that dataset. High-quality NLEs refers to perceived quality, which is different from NLE faithfulness to the model's internal reasoning process. Faithfulness is presented as a separate category below.

In natural language inference (NLI), Camburu et al. (2018) introduce the large, high-quality e-SNLI dataset of NLEs for SNLI (Bowman et al., 2015), which provides training data for NLE generation. They modify the InferSent (Conneau et al., 2017) architecture to generate both labels and NLEs, and show that the e-SNLI dataset can be used to train the model to generate NLEs. They also show that e-SNLI can be used to train universal sentence representations (evaluated on SentEval (Conneau and Kiela, 2018)) and for the transfer of NLEs to the MNLI (Williams et al., 2018) dataset. Narang et al. (2020) show that large multi-task learning with NLEs (using the T5 (Raffel et al., 2020) LM) can be used to produce high-quality NLEs. They further show a successful zero-shot transfer of NLEs between tasks: from e-SNLI (Camburu et al., 2018) to CoS-E (Rajani et al., 2019). Chen et al. (2021) propose to generate

contrastive NLEs for natural language inference (NLI) – i.e. NLEs that show why the model made a given choice rather than any other choice. They achieve this by using external knowledge and by creating counterfactual examples (e.g. premise-hypothesis pairs) for the alternative labels, which enable the contrastive NLEs. Brahman et al. (2021) generate NLEs for the Defeasible Inference task (δ-NLI) (Rudinger et al., 2020) – the task of determining if a given natural language inference (NLI) relation is strengthened or weakened based on a piece of new evidence. They collect training NLEs from various available and model-generated sources and train BART (Lewis et al., 2019) and GPT-2 (Radford et al., 2019) LMs to produce NLEs.

In other applications, Ehsan et al. (2018) explain the actions of autonomous systems (experiments use the Frogger arcade game) by using their internal representations to generate NLEs via a machine-translation-like architecture. The training data and NLEs are generated by human players. Ni et al. (2019) propose a method to extract and generate justifications (NLEs) for user recommendation algorithms. They train models to produce personalised justifications in the form of NLEs by taking into account the user profile (the information known about the user) and the item profile (the available reviews from other users). Marasović et al. (2020) generate NLEs for visual-textual tasks by using the GPT-2 (Radford et al., 2019) LM and augmenting its input with the "output of external vision models that enable different levels of visual understanding".

In the low-resource scenario, Jang and Lukasiewicz (2021) show that the predict-then-explain method (doing prediction before generating an NLE) provides the best classification and NLE generation performance when compared to joint training and explain-then-predict. Zhou et al. (2023) present FLamE – "a two-stage few-shot learning framework that first generates explanations using GPT-3, and then fine-tunes a smaller model (e.g., RoBERTa) with generated explanations to effectively learn from explanations". Solano et al. (2023) show that one can fine-tune only a small portion (6.8%) of the parameters of the T5 (Raffel et al., 2020) LM to achieve competitive NLE generation results.

Rationale generation can also be used to improve NLE quality. Majumder et al. (2022) introduce the RExC framework which aims to improve NLE quality by first generating rationales, then extracting external knowledge w.r.t. those rationales, selecting the best knowledge samples, and finally, generating the NLEs based on them.

There are also some applications of reinforcement learning to NLE generation (Inoue et al., 2021; Suo et al., 2023). Inoue et al. (2021) produce NLEs (*abstractive rationales*) for context-based question answering. They use an iterative procedure via

reinforcement learning to improve the NLE's compactness, usefulness and readability. This NLE generation task is easier than others because the NLE is a partial summary of the provided context. In computer vision, Suo et al. (2023) use reinforcement learning with NLE generation by generating multiple NLEs per training example, and optimising for NLE quality by using the *answer score* provided by the model as a training signal in reinforcement learning.

Modern-day chat assistants such as Microsoft/OpenAI's ChatGPT[1], Google's Gemini[2], and Anthropic's Claude[3] can operate cross-domain: both with images and text, and can explain themselves (i.e. provide NLEs) if asked to do so, although with no faithfulness guarantees.

NLE Faithfulness NLE faithfulness is the extent to which the generated NLEs reflect the true reasoning process of the model. There are several notable works that aim to produce models with greater NLE faithfulness (Liu et al., 2019a; Kumar and Talukdar, 2020; Zhao and Vydiswaran, 2021; Wiegreffe et al., 2021).

Liu et al. (2019a) introduce a generative explanation framework that aims to "help build stronger connections between the explanations and predictions". They achieve this by minimising the so-called *explanation factor*, which is the "semantic distance of input texts, generated explanations and golden explanations".

Kumar and Talukdar (2020) introduce the NILE architecture, which aims to produce high-quality faithful NLEs for natural language inference (NLI). Their modular approach allows them to improve NLE faithfulness by first generating explanations for each potential label (entails/contradicts/neutral), and then predicting the correct label based on the data and generated explanations.

Zhao and Vydiswaran (2021) propose the LIREx framework for generating NLEs for natural language inference (NLI). LIREx consists of a module that extracts rationales, followed by a module that uses the rationales to generate NLEs for each possible label, and, finally, a classifier that predicts the label based on the premise, hypothesis and NLEs. Zhao and Vydiswaran (2021) evaluate comprehensiveness (removing the NLEs for classification) and sufficiency (classifying only based on NLEs) and show that LIREx produces faithful NLEs.

Wiegreffe et al. (2021) train models and evaluate the agreement between the predictions and the generated NLEs. To achieve this, they investigate whether the

[1] `https://openai.com/chatgpt`
[2] `https://gemini.google.com`
[3] `https://claude.ai`

predicted labels and NLEs share the following properties: first, if they are similarly robust to perturbations in the model's input, and, second, if they are attributed to similar extractive rationales in the model's input.

Transfer Learning of NLEs Transfer learning of NLEs is usually necessary when not enough data is available for the *child* task (i.e. the target task). Following the categorisation in Chapter 2.2, transfer learning of NLEs can be zero-shot, few-shot and multi-shot, depending on the number of NLEs on the child task. It can also be in-domain or out-of-domain depending on whether the parent and child task are in the same domain. There is another distinction, which is raised in my work (Yordanov et al., 2022), namely label-abundant or scarce-label transfer of NLEs, depending on whether one has abundant labels for the child task, or only a few.

Zero-shot in-domain transfer of NLEs (between datasets of the same task) has been done, e.g., by Camburu et al. (2018); Kumar and Talukdar (2020), and Narang et al. (2020). Narang et al. (2020) additionally consider zero-shot out-of-domain transfer of NLEs, while Erliksson et al. (2021) extend their work by showing that few-shot out-of-domain transfer of NLEs is possible in the abundant-label setup. Marasović et al. (2022) use prompt engineering for few-shot out-of-domain transfer of NLEs for the scarce-label setup. The prompt choice is less relevant in the abundant-label setup, because task adaptation can be done via the abundant training labels.

My work in this chapter (Yordanov et al., 2022) demonstrates successful few-shot out-of-domain transfer of NLEs between the (parent) e-SNLI dataset and the (child) WinoGrande (Sakaguchi et al., 2020) and ComVE (Wang et al., 2020) datasets.

NLE Evaluation NLE evaluation can be done in one of two ways: automatic evaluation and human evaluation. The use of automatic metrics from the machine translation domain to the NLE evaluation domain has been investigated by Kayser et al. (2021). They conclude that the automatic evaluation metrics only weakly correlate with human judgment of NLE quality, but that there are significantly better alternatives to the BLEU score (Papineni et al., 2002), such as METEOR (Banerjee and Lavie, 2005) and BERTScore (Zhang et al., 2020b).

Camburu et al. (2020) introduce a procedure to evaluate NLE inconsistency in natural language inference (NLI). They train a *reverse* model to generate the hypothesis from the NLE and premise, and use this model to generate adversarial hypotheses by feeding it contradicting NLEs. They show that existing models are prone to generate inconsistent NLEs by using these adversarial hypotheses.

The ERASER benchmark (DeYoung et al., 2020) includes a number of NLE generation tasks, along with metrics for automatic evaluation of NLEs. Apart from the standardization of NLE evaluation, the ERASER benchmark proposes metrics for measuring NLE faithfulness. Jacovi and Goldberg (2020) additionally propose a more comprehensive definition of faithful NLEs and procedures for evaluation. Hase et al. (2020) propose Leakage-Adjusted Simulatability (LAS) – an NLE metric that aims to improve NLE faithfulness evaluation by simulating the model's predictions based on its inputs and its generated NLEs. LAS "measures how well explanations help an observer predict a model's output, while controlling for how explanations can directly leak the output". Atanasova et al. (2023) propose two tests for faithfulness evaluation of model-generated NLEs. The first test fails the model if adding insignificant words to the input changes the predicted label and the generated NLE. The second one tests the sufficiency of the NLE by testing if restricting the model's input to what is contained in the NLE changes the predicted label. Although faithfulness is important for NLE generation, it falls beyond the scope of the current work, which is perceived NLE quality.

Data Annotation for NLEs There are three main types of NLE data generation: crowdsourcing (e.g. via Amazon Mechanical Turk) (Camburu et al., 2018; Rajani et al., 2019), generation by trusted people/experts (Aggarwal et al., 2021; Yordanov et al., 2022) and generation via large pre-trained LMs such as GPT-3 (Brown et al., 2020) (Wiegreffe et al., 2022). While the quality of expert-generated NLEs can be trusted, crowdsourced and model-generated NLEs are less reliable and may require additional processing.

Many works on NLE generation use Amazon Mechanical Turk because NLEs can be generated by workers relatively cheaply. The downside of this approach is that the generated NLEs may be of poor quality (Rajani et al., 2019) unless special care is taken. To address this issue, Camburu et al. (2018) propose a combination of rule-based (*in-browser*) verifications and guidance to the workers by requiring them to first provide extractive rationales.

Lee et al. (2020) create a pipeline that allows for the human generation of NLEs and uses them to label unlabelled data for named entity recognition, relation extraction, and sentiment analysis. They use the NLEs on the labelled instances to train LSTM-based RNN models to generate labels for unlabeled instances. Lee et al. (2020) ensure NLE quality by requiring the annotators to provide extractive rationales, and then to provide relations between them and thus form structured NLEs.

Other works use large pre-trained LMs such as GPT-3 (Brown et al., 2020) to generate NLE data. E.g. (Wiegreffe et al., 2022) propose a model-based NLE generation and filtration pipeline that improves NLE quality. First they prompt the GPT-3 (Brown et al., 2020) LM with some examples with NLEs for CommonsenseQA (Talmor et al., 2019) and SNLI (Bowman et al., 2015); GPT-3 then generates 5 NLEs for each, and the best NLE is selected via a T5 (Raffel et al., 2020) model, which has been fine-tuned on human judgment.

NLE Datasets Here I list some more notable datasets with NLEs. There are examples both for natural language processing and for computer vision tasks.

The *e-SNLI* (Camburu et al., 2018) dataset consists of high-quality explanations for the SNLI (Bowman et al., 2015) dataset: one NLE for each instance of the training dataset (550k examples), and three NLEs for each dev and test instance. The dataset is constructed by using Amazon Mechanical Turk and by using numerous procedures to ensure high NLE quality.

The *LIAR-PLUS* (Alhindi et al., 2018) dataset consists of NLEs for fact-checking claims. The dataset augments the LIAR (Wang, 2017) dataset (13k examples) with NLEs derived from the justifications that the human annotators have used in the LIAR dataset.

The Common Sense Explanations (CoS-E) dataset (Rajani et al., 2019) consists of human commonsense explanations for the Commonsense Question Answering (CommonsenseQA) dataset (Talmor et al., 2019). Unfortunately, CoS-E has overall poor NLE quality (most NLEs rephrase the question and the answer, rather than providing reasoning), which is why I do not use CoS-E in this work. The ECQA dataset (Aggarwal et al., 2021) is constructed similarly to CoS-E, but they partnered with a private firm, which helped with producing NLEs of substantially higher quality than CoS-E. Unfortunately, the ECQA dataset was published at a later stage of the work included in this chapter, which is why I did not include it.

The *ComVE* (Wang et al., 2020) dataset consists of three tasks: binary commonsense classification, i.e. "select the statement of the two that does not make sense"; multi-choice explanation, i.e. select the explanation corresponding to that choice; and explanation generation for the binary commonsense classification. The dataset provides three NLEs for each instance of the training dataset (10k), dev dataset (1k) and test dataset (1k) crowdsourced from Amazon Mechanical Turk. NLE quality is ensured via comprehensive guidelines and manual quality control.

The *SBIC* (Sap et al., 2020) societal bias dataset consists of ≈ 45k posts from the internet annotated with categories of offensiveness and short explanations generated via crowdsourcing from Amazon Mechanical Turk. The Few Explanations Benchmark (FEB) (Marasović et al., 2022) includes the SBIC dataset by converting the categories and the short explanations into full NLEs.

The Abductive Natural Language Inference dataset (AbductiveNLI) (Bhagavatula et al., 2020) consists of 20,000 examples of natural language statements A and C, where the task is to choose a statement B satisfying: $if(A\ and\ B)$, $then\ C$. This dataset is not directly applicable to my work because it cannot be formulated as an NLP task with explanations. This is because the correct statement B should be the NLE, but it determines the answer C, rather than explaining it.

In computer vision, there are many datasets with NLEs. Examples include the e-ViL (Kayser et al., 2021) benchmark consisting of three datasets of visual-language tasks with NLEs. As part of the benchmark, Kayser et al. (2021) created the e-SNLI-VE dataset with NLEs by adapting the e-SNLI dataset (Camburu et al., 2018) to the SNLI-VE visual-language task (Xie et al., 2019).

Other examples include the VCR (visual commonsense reasoning) dataset (Zellers et al., 2019), which consists of 290,000 questions about movie scenes, with multiple-choice answers, and multiple-choice NLEs to explain those answers; VQA-E (Li et al., 2018) and VQA-X (Kim et al., 2018) datasets, which consist of NLEs for the VQA (visual question answering) dataset (Antol et al., 2015); the action explanation dataset (ACT-X) (Park et al., 2018) with NLEs for images extracted from YouTube; and BDD-X (Kim et al., 2018), which is a dataset for training self-driving cars, consisting of over 77 hours of vehicle driving annotated with driver actions.

4.2 Few-Shot Out-of-Domain Transfer Learning of Natural Language Explanations in a Label-Abundant Setup

This work is based on my paper with the same name (Yordanov et al., 2022), published at the Findings of the Association for Computational Linguistics: EMNLP 2022. In this work, I address the problem that training models for NLE generation usually requires the acquisition of task-specific NLEs, which is time- and resource-consuming. I propose to use few-shot out-of-domain transfer learning of NLEs from a parent task with many NLEs to a child task. Specifically, I examine the setup in which the child task has few NLEs but abundant labels. I establish four few-shot transfer learning

methods that cover the possible fine-tuning combinations of the labels and NLEs for the parent and child tasks. I use these methods to transfer explainability from a large natural language inference dataset (e-SNLI) separately to two child tasks: (1) hard cases of pronoun resolution, where I introduce the small-e-WinoGrande dataset of NLEs on top of the WinoGrande dataset, and (2) commonsense validation (ComVE). The results demonstrate that the parent task helps with NLE generation and I establish the best methods for this setup.

4.2.1 Introduction

Recent developments have made it possible for AI models to learn from natural language explanations (NLEs) for the ground-truth labels at training time and generate such explanations for their decisions at deployment time (Hendricks et al., 2016; Ling et al., 2017b; Park et al., 2018; Camburu et al., 2018; Kim et al., 2018; Rajani et al., 2019; Camburu et al., 2020; Narang et al., 2020; Kumar and Talukdar, 2020; Marasović et al., 2022). To train a model to generate NLEs, existing works rely on human-annotated training datasets of NLEs. However, large datasets of NLEs, such as e-SNLI (Camburu et al., 2018), are time-consuming and expensive to gather. One approach is to transfer explanations from a different domain, via few-shot transfer learning. The usual setup for few-shot out-of-domain transfer learning consists of transfer learning from a *parent* task, with abundant training examples, to a *child* task that only has a few training examples (Thrun, 1996; Ravi and Larochelle, 2017).

In this work, I assume that both parent and child tasks have abundant labels, and the parent task has abundant NLEs, but the child task only has a few training NLEs. This can be characterised as a few-shot transfer learning of NLEs in a label-abundant setup. Given the advent of deep learning in recent years, this child-task scenario may be quite frequent, as one may already have a large dataset with labels on which they aim to train NLEs-generating models without having to annotate the entire dataset with NLEs. Vanilla few-shot learning approaches can be directly applied in this setting, but those approaches would give up a large proportion of the training labels just so that the few-shot regime applies equally to labels and NLEs. Therefore, alternative approaches are needed to generate few-shot NLEs, while using the abundant labels to obtain a high child-task accuracy.

In this work, I introduce three few-shot transfer learning methods for NLEs that utilize the abundant training labels for both the parent and child task, and I adapt for computational efficiency the method from Erliksson et al. (2021). Together, these

four methods are combinations of multi-task learning and fine-tuning between a parent and a child task with few training NLEs but abundant labels. I instantiate the few-shot learning approaches on e-SNLI (Camburu et al., 2018) as parent task, and WinoGrande (Sakaguchi et al., 2020) and ComVE (Wang et al., 2020) as child tasks. As the WinoGrande dataset does not contain NLEs, I introduce *small-e-WinoGrande*, which provides 100/50/100 NLEs for the training, development, and test sets, respectively. I show the extent to which few-shot out-of-domain transfer learning of NLEs is currently feasible and provide insight into which learning techniques work best in this setup. I perform human evaluation and do comprehensive evaluation against child-task-only and zero-shot baselines.[4]

4.2.2 Related Work

To my knowledge, at the time of writing (Yordanov et al., 2022), there is only one existing work for few-shot out-of-domain transfer learning of NLEs, that of Erliksson et al. (2021), who introduce a vanilla fine-tuning method on top of the zero-shot WT5 model (Narang et al., 2020). Their work follows that of Narang et al. (2020), who show a proof-of-concept for zero-shot NLE transfer across domains, without any training NLEs on the child task (zero-shot), but who use the largest T5 model (with 11B parameters) (Raffel et al., 2020) to obtain those results. Erliksson et al. (2021) adapt this approach to the more practical few-shot NLE setup via a simple fine-tuning method on top of a smaller zero-shot WT5 model (Narang et al., 2020). However, their work is limited by two factors: (1) they only use one of the four possible training methods that I identify for this setup, and (2) they use only automatic evaluation metrics, which do not necessarily align with human judgment (Camburu et al., 2018; Kayser et al., 2021).

Marasović et al. (2022) show that prompt engineering can help in few-shot out-of-domain transfer of NLEs, for the case where the training labels are also scarce. In that case, prompt engineering is relevant because their setting is few-shot learning of both NLEs and labels. In this work, we also have abundant labels on the child task, which makes prompt engineering less relevant. In practice, I have found that prompt choice does not significantly affect performance, and the prompt choices I make in this work (Table 4.1) follow the WT5 style (Narang et al., 2020).

In the more general area of natural language generation, few-shot learning is a growing topic (Chen et al., 2020), e.g., in dialogue generation (Peng et al., 2020;

[4]The code and the datasets are publicly available at: `https://github.com/YDYordanov/Few-shot-NLEs`.

Shalyminov et al., 2019). These approaches, however, do not directly apply to transfer learning of NLEs, which is a dual task of predicting both the label and generating an explanation.

Gradual unfreezing is one potential method to help with few-shot transfer learning which one may consider for this work. However, looking more closely at the original gradual unfreezing work (Howard and Ruder, 2018), it uses a random initialisation for a 2-layer classification head. Similarly, random initialisation is used by Kumar et al. (2022), where they do 1-step gradual unfreezing, which suggests that the need for gradual unfreezing is most likely due to the bad (random) starting initialisation of the head. Since in this work, we do not have a randomly initialised head (as I directly fine-tune a pre-trained model), gradual unfreezing may not be suitable for this setting. Furthermore, Raffel et al. (2020) show that gradual unfreezing does not help when applied to the T5 LM used in this work.

For the task of resolving hard cases of pronoun resolution, there is the WinoWhy (Zhang et al., 2020a) diagnostic dataset for assessing commonsense knowledge in generated NLEs. It is based on the Winograd Schema Challenge dataset (Levesque et al., 2012) and is phrased as a zero-shot NLE classification task. I decided not to use it in this work, because I am interested in measuring NLE generation rather than classification of predefined NLEs.

4.2.3 Method

In this section, I introduce the datasets that I use, the base model (T5) (Raffel et al., 2020) that I fine-tune, all baselines and all four few-shot NLE transfer learning methods (M1-M4). I also describe the training procedure and the evaluation procedure of the model-generated NLEs given by human evaluation on Amazon Mechanical Turk.

Datasets Here I introduce all datasets used in this work. The NLE-abundant e-SNLI dataset is the parent dataset of choice, whereas the WinoGrande and ComVE datasets are the child datasets for which I want the models to generate NLEs.

e-SNLI. Natural language inference (NLI) (Dagan et al., 2006) is the task of assigning a relation of *entailment*, *contradiction*, or *neutrality* between a *premise* and a *hypothesis*. The e-SNLI dataset (Camburu et al., 2018) consists of human-written NLEs on top of the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015). I select e-SNLI as parent dataset due to its large size (∼570K) and high-quality NLEs available for each training instance. Example from e-SNLI:

Premise: An adult dressed in black holds a stick. **Hypothesis:** An adult is walking away, empty-handed.
Label: contradiction
NLE: Holds a stick implies using hands so it is not empty-handed.

WinoGrande. The WinoGrande dataset (Sakaguchi et al., 2020) consists of 40,398 binary fill-in-the-gap instances of pronoun resolution that follow the Winograd Schema format (Levesque et al., 2012). I select WinoGrande as a child task, since it requires implicit knowledge, which I want to capture in the NLEs. I construct the *small-e-WinoGrande* dataset by manually creating NLEs for 100/50/100 training/dev/test instances. Example from *small-e-WinoGrande*:

The geese prefer to nest in the fields rather than the forests because in the __ predators are very visible.
Options: fields, forests. **Answer:** fields.
NLE: The fields are more open spaces than the forests, hence predators are more visible there.

ComVE. Commonsense Validation and Explanation (ComVE) (Wang et al., 2020), as reformulated by Majumder et al. (2022), is the task of jointly identifying which one of two statements contradicts commonsense and explaining why. The dataset consists of 10,000 training, 1,000 validation, and 1,000 test instances. I select ComVE as a child task, because it is a commonsense reasoning task for which there are good-quality human-written NLEs. Example from ComVE:

Statements: He drinks [milk/apple].
Answer: "apple" does not make sense.
NLE: An apple is a whole food and unable to be drunk without being juiced.

For more dataset details, see Appendix C.1.

Base Model Similarly to Narang et al. (2020), I use the T5 (Raffel et al., 2020) generative LM, in particular, the *Base* model with 220M parameters, due to its good trade-off of performance and computational requirements. For T5, tasks are distinguished only via their task-specific input/target formats. I follow the input/target format for e-SNLI by Narang et al. (2020): *premise: [premise] hypothesis: [hypothesis] / [relation] explanation: [explanation].* I obtain the input formats for WinoGrande

Task	Input Format	Target Format
e-SNLI	explain nli premise: [premise] hypothesis: [hypothesis]	[relation] explanation: [explanation]
W.G.	explain WinoGrande schema: [schema start] _ [schema end] options: [option 1], [option 2].	[correct option] explanation: [explanation]
ComVE	explain ComVE Sentence 1: [statement 1] Sentence 2: [statement 2]	[nonsensical statement id] explanation: [explanation]

Table 4.1: T5 input/target formats for each task: e-SNLI, WinoGrande (W.G.), and ComVE, used for all models. When training on examples without NLEs, "explain" and "explanation:" are not included in the input/target format.

Model name	Meaning
CD–fine-tune	fine-tune T5 on the child dataset, and then fine-tune on 50 NLEs
CD–union	fine-tune T5 on the union of the child dataset and 50 NLEs
WT5–fine-tune	fine-tune T5 on the union of e-SNLI and SNLI, and then fine-tune on the child dataset
WT5	fine-tune T5 on the union of e-SNLI, SNLI, and the child dataset
M1	fine-tune T5 on the union of e-SNLI, the child dataset, and 50 NLEs
M2	fine-tune T5 on the union of e-SNLI and the child dataset, and then fine-tune on 50 NLEs
M3	fine-tune T5 on e-SNLI, and then fine-tune on the union of the child dataset and 50 NLEs
M4	fine-tune T5 on e-SNLI, and then fine-tune on the child dataset, and, finally, on 50 NLEs

Table 4.2: Legend of the model names. The child dataset excludes the NLEs, unless specified. The 50 NLEs refer to the few (50) instances of the child task with NLEs.

and ComVE in a similar manner (see Table 4.1). I observed in early experiments that the exact choice of input/target formats does not significantly affect performance.

I choose the best practice for multi-task learning with T5, namely, via training on the union of the datasets in question (Raffel et al., 2020).

Few-Shot Transfer Learning Methods Table 4.2 shows all the models that I use, with further diagrams for M1-M4 in Figure 4.1. M1 to M4 are the four few-shot transfer learning methods for NLE generation, which I obtain by combining the

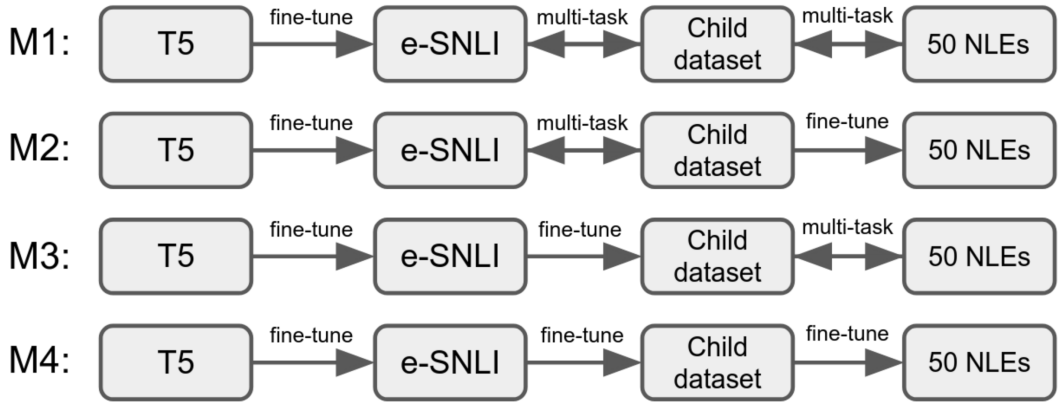


Figure 4.1: Diagram of the training procedures for M1-M4 models.

parent dataset with NLEs, the child dataset, and a few NLEs (I use 50 in this work) in all reasonable multi-task and fine-tuning combinations. These methods (M1-M4) are inspired by the zero-shot transfer of NLEs work by Narang et al. (2020) (WT5), where fine-tuning on the child dataset (with or without NLEs) is not explored. M1 extends the WT5 (Narang et al., 2020) zero-shot method by including the few NLEs of the child task in the unified training dataset, whereas M3 is similar to the method by Erliksson et al. (2021). However, the methods by Narang et al. (2020) and Erliksson et al. (2021) use the union of the parent dataset with and without explanations, which I do not, because in the few-shot NLE case, this is unnecessary and doubles the computation cost.

To understand the benefits and downsides of the few-shot transfer of NLEs methods (M1-M4), first, we need to understand the few-shot NLE generation (without parent task). In the few-shot NLE setting, to produce good NLEs, the model must understand the underlying task – i.e. have a good task accuracy because the number of NLEs is insufficient to train the model fully. For this, one can consider multi-task learning because it is better at fusing information between the tasks than fine-tuning (e.g. see (Aghajanyan et al., 2021)). However, when using multi-task learning with the union of the child dataset (without NLEs) and its few NLEs (similar to Erliksson et al. (2021)), there is a fixed training proportion between the labelled data and the NLE data, which may or may not provide the correct balance. On the other hand, when first training on the labelled data, and then fine-tuning on the few NLEs, the balance can be controlled by how much fine-tuning is done. Furthermore, when fine-tuning on the few NLEs, the last training step only trains for NLE generation, which is the target task. This can, in theory, benefit M2 and M4 over M1 and M3. There are similar considerations on the balance between the parent and child datasets depending on whether one uses multi-task learning (M1, M2) or fine-tuning (M3, M4).

I also consider four baseline methods. The two child-task baselines CD–fine-tune and CD-union serve to measure the contribution of the parent in NLE transfer. Two zero-shot NLE transfer learning baselines, WT5 (Narang et al., 2020) and WT5–fine-tune, serve to measure the contribution of the 50 training NLEs in the child task.

Training Details The training objective is given by cross-entropy loss with targets as described in Table 4.1. I use the AdamW optimizer (Loshchilov and Hutter, 2019) and linear learning rate scheduler with warm-up over 10% of the training. For all models, I fix the batch size to 16 and do a grid search over the learning rate values and the number of training epochs. The hyperparameters' ranges are described in

Appendix C.2. No early stopping is needed, because I use a learning rate scheduler and the number of training epochs is a hyperparameter. I do not use gradual unfreezing (Howard and Ruder, 2018), because it has been shown by Raffel et al. (2020) that it does not help when applied to the T5 LM.

At each stage of training, the best hyperparameter combinations are either selected by the perplexity relative to the target NLEs of the dev set of the child task (CT), by dev accuracy on CT, or by NLE perplexity on the e-SNLI dev set. At each stage of training, I would want the model to obtain good CT NLE performance. However, if no CT NLEs are used in training at that stage, one cannot expect good CT NLE generation performance. In that case, I choose to evaluate the models on their most relevant training dataset to CT NLE generation, where I assume that CT without NLEs is more relevant than the parent dataset (e-SNLI). The choice of evaluation metric depends on the choice of dev set: NLE perplexity if the dev set contains NLEs, and accuracy if not. Following Camburu et al. (2018), I use NLE perplexity for hyperparameter selection over language generation metrics like BLEU (Papineni et al., 2002) because perplexity measures how well the model predicts the dev data (e.g. see Section 2.1.2). Furthermore, language generation metrics require the models to generate NLEs, which for small dev NLE datasets like small-e-WinoGrande (50 dev NLEs), may mean a lot of randomness in the average score. The selection criteria for each model, along with the best hyperparameters are given in Table C.1 in Appendix C.2.

Model-generated explanations are obtained via beam search (Graves, 2012) with a beam width of 5. This beam width value is selected among the values $\{1, 2, 3, 4, 5, 6\}$ by manually examining the NLE generation quality of multiple models on random examples of the dev sets of both child tasks.

Human Evaluation I use Amazon Mechanical Turk to evaluate the model-generated NLEs, with three annotators per instance. The evaluation procedure for each instance is in three steps and follows existing works (Kayser et al., 2021; Majumder et al., 2022; Marasović et al., 2022). An example for WinoGrande is given in Figure 4.2. First, annotators have to predict the classification label for the example. Second, they have to select one of four options for whether the NLE is a valid and satisfactory explanation for the selected label: Yes, Weak Yes, Weak No, or No. Third, they have to select shortcomings of the explanation from the following: "does not make sense", "insufficient justification", "irrelevant to the task", "too trivial", and "none". These choices may not only provide insight into the problems that the NLEs

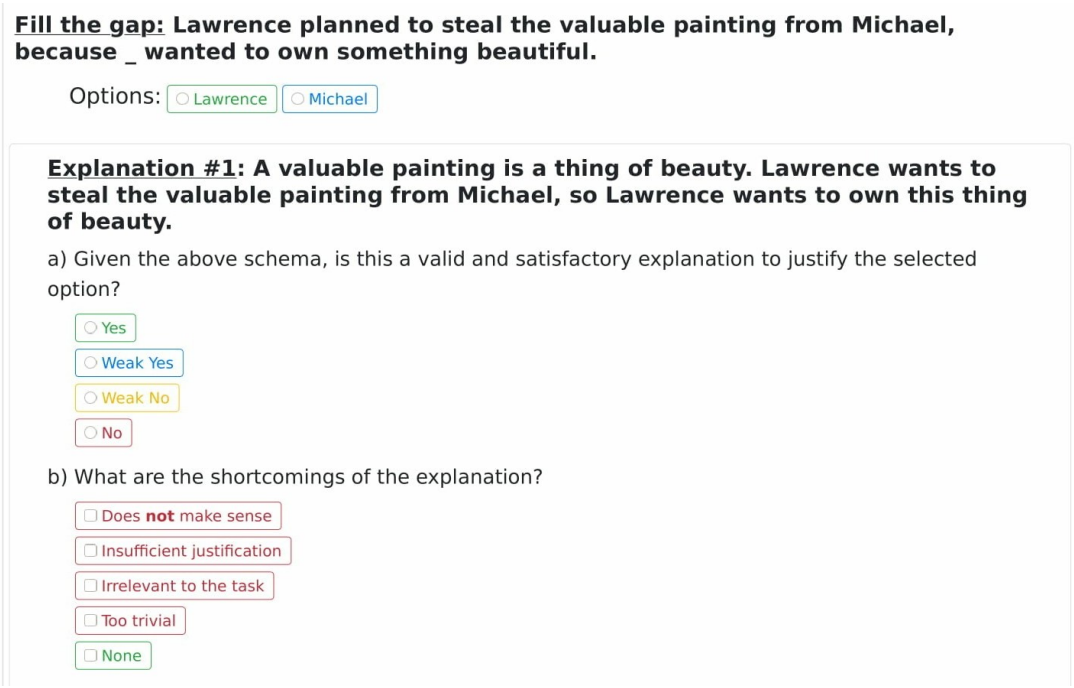


Figure 4.2: An example of what the human annotators see when they evaluate NLEs for WinoGrande.

may have, but also guide the annotators to carefully think about the answer to the main question about NLE quality.

As suggested by Kayser et al. (2021), for each example, the annotators are provided with two (shuffled) NLEs, one from a model and one ground-truth from the test set. This serves to mentally ground the annotator's score of the model-generated NLE.

All models are evaluated on 100 examples from the test dataset of each child task. Similarly to previous works (Camburu et al., 2018; Kayser et al., 2021; Majumder et al., 2022), the NLE evaluation is only done on correctly labelled (by the model) examples, as it is expected that an incorrect label is not supported by the model with a correct NLE. See Appendix C.3 for more details and for screenshots of the forms used to collect the annotations.

4.2.4 Experiments and Results

Experimental Setup Following Kayser et al. (2021), I use an aggregated score (I call *NLE score*) to evaluate the NLE generation quality, where each option selected by the annotators: Yes, Weak Yes, Weak No, or No is mapped to a value: 1, $\frac{2}{3}$, $\frac{1}{3}$, or 0, respectively. The exact formula is:

$$\text{NLE score} = \left(Y + \frac{2}{3}Y_{weak} + \frac{1}{3}N_{weak}\right) \frac{100}{Y + Y_{weak} + N_{weak} + N}$$

Model	WinoGrande		ComVE		ComVE Automatic NLE Metrics					
	Task acc%	NLE score	Task acc%	NLE score	B-1	B-2	B-3	B-4	MET	BERT-Score
CD–fine-tune	59.7	34.7	**87.8**	31.4	**45.2**	**29.5**	**19.5**	**13.1**	**21.5**	83.4
CD–union	57.2	35.9	83.1	27.7	27.4	16.6	10.2	6.4	19.1	81.8
WT5–fine-tune	**60.2**	8.7	85.7	28.9	24.6	15.1	9.7	6.5	13.5	74.8
WT5	58.0	8.3	76.2	23.9	22.8	12.0	6.4	3.6	12.7	71.5
M1	53.6	28.3	82.8	40.2	34.5	19.2	10.8	6.3	20.3	81.8
M2	56.0	**44.1***	80.6	40.6	43.5	26.3	16.5	10.6	20.0	83.1
M3	54.6	29.6	85.5	38.6	33.6	18.8	10.9	6.2	20.8	82.1
M4	58.2	41.9*	86.5	**48.5***	44.4	27.5	17.5	10.7	21.2	**83.6**

Table 4.3: Performance of models on WinoGrande and ComVE as child tasks. From the 100 test examples, only the correctly classified are given NLE scores. B-1,2,3,4 stand for BLEU-1,2,3,4; MET stands for METEOR. Best results are in bold; * denotes the statistically significant best results.

where Y, Y_{weak}, N_{weak}, and N denote the number of Yes, Weak Yes, Weak No, and No, respectively, and the last term rescales the NLE score to be between 0 and 100. This aggregation has two goals: (1) to provide a single metric to compare the methods, and (2) to account for the subjective nature of choosing between close labels such as Yes and Weak Yes.

For every model comparison, I report if it is statistically significant via the paired Student's t-test for equal variances (Yuen and Dixon, 1973), with single-tailed p-values and 0.05 statistical significance threshold.

I additionally evaluate all models on the full ComVE test set (1,000 examples with three NLEs per example) via automatic metrics (BLEU (Papineni et al., 2002), METEOR (Banerjee and Lavie, 2005), and BERTScore (Zhang et al., 2020b)), with METEOR and BERTScore having been shown to have the best correlation with human judgment of NLEs across several datasets (Kayser et al., 2021). Automatic evaluation provides additional insights into NLE quality since the human evaluation was done on only 100 examples, which is ten times fewer. I do not compute automatic metrics w.r.t. WinoGrande, since its NLE test set contains only 100 examples for which I already have the human evaluation, the gold standard for NLE evaluation.

Quantitative Results Here, I report the task accuracy, NLE score, and automatic NLE evaluation metrics (BLEU, METEOR and BERTScore).

The results are given in Table 4.3. We notice that the automatic metrics are not well aligned with the human evaluation (NLE score). This has also been previously observed in other studies (Camburu et al., 2018; Kayser et al., 2021). Therefore, I

will base my main conclusions only on the human evaluation (NLE score). However, I will also comment on the automatic metric results and what they show.

Main Conclusions: First, we notice that all methods (M1–M4) significantly outperform the zero-shot baselines (WT5–fine-tune and WT5) in terms of NLE quality for both datasets, which proves the utility of the 50 child-task NLEs.

Second, we see that not all methods outperform the child-task baselines. For example, on WinoGrande, both CD–fine-tune and CD-union outperform M1 and M3 in terms of the NLE quality. This shows that it is sometimes possible that fine-tuning on a large parent task of out-of-domain NLEs hurts NLE quality of a child task. However, for both datasets, the best performing method is among the M1–M4 methods (and for ComVE, all M1–M4 methods outperform the child-task baselines), suggesting that it is generally useful to use a large dataset of NLEs as a parent task even when out-of-domain.

Third, we see that the M1–M4 methods rank differently on different datasets, in particular, M2 and M4 are the significantly best methods on WinoGrande, and M4 is the significantly best method on ComVE. I believe that the main difference in method ranking is that the methods obtain much closer-to-chance accuracy on WinoGrande than on ComVE. This is important, because poor task understanding can lead to poor NLE generation. In particular, M2 and M4 obtain significantly better WinoGrande accuracy than M1 and M3, which might explain the significant gap in NLE score. M2 and M4 are the best NLE generation methods on both datasets, which could be because M2 and M4 use the 50 NLEs of the child task in a separate training regime, whereas M1 and M3 use a combination of the much larger child dataset with the 50 NLEs. This suggests that the 50 NLEs require their own training regime to obtain good NLE generation.

Finally, we see that the best task performances on both datasets are obtained by a baseline method: WT5–fine-tune (no child NLEs) for WinoGrande and CD–fine-tune (child-task only) for ComVE. Moreover, among the M1-M4 methods, there is no clear best method in terms of task performance, nor a correlation with the NLE quality. Hence, a trade-off between task performance and NLE quality is needed, which I observe is achieved by M4.

Automatic Metrics: The automatic evaluation results largely confirm the results from the human-annotated NLE scores. The best-scoring M4 model outperforms all three other main models (M1-M3) in terms of BLEU, METEOR, and BERTScore. However, the best-performing baseline, CD–fine-tune, outperforms M4 in terms of BLEU score but is similar in terms of METEOR and BERTScore. This suggests

WinoGrande Model	**NLE score**	**Yes%**	**Weak Yes%**	**Weak No%**	**No%**	**Non-sense%**	**Insuffi-cient%**	**Irrele-vant%**	**Trivial %**	**None%**
CD–fine-tune	34.7	17.5	20.1	11.6	50.8	32.0	37.0	4.0	7.5	19.5
CD–union	35.9	20.7	15.2	15.2	49.0	33.8	32.4	5.5	6.4	21.9
WT5–fine-tune	8.7	4.6	4.1	4.1	87.2	60.8	20.3	10.6	4.1	4.1
WT5	8.3	4.8	3.0	4.2	87.9	71.1	**12.8**	9.6	**2.1**	4.3
M1	28.3	14.3	14.3	13.6	57.8	**28.0**	39.5	8.9	4.5	19.1
M2	**44.1**	**25.9**	18.0	18.5	**37.6**	**28.1**	33.2	6.5	4.0	**28.1**
M3	29.6	15.4	14.8	13.0	56.8	43.7	29.3	6.9	2.3	17.8
M4	41.9	22.6	22.6	12.8	42.1	34.3	33.3	**2.5**	6.9	23.0
ComVE Model										
CD–fine-tune	31.4	25.4	7.2	3.8	63.6	26.9	32.3	12.5	3.6	24.7
CD–union	27.7	23.6	4.2	3.8	68.4	39.8	24.6	10.2	**2.7**	22.7
WT5–fine-tune	28.9	20.0	11.8	3.1	65.1	30.7	37.9	**8.9**	3.6	18.9
WT5	23.9	15.3	10.2	5.6	69.0	36.9	31.7	11.9	5.2	14.3
M1	40.2	28.5	14.6	6.1	50.8	22.1	29.0	18.1	4.7	26.1
M2	40.6	27.4	17.7	4.2	50.6	23.9	33.5	10.4	4.4	27.9
M3	38.6	30.3	8.8	7.5	53.5	32.5	**21.7**	12.0	4.4	29.3
M4	**48.5**	**36.7**	14.3	6.8	**42.2**	**18.8**	28.2	13.1	2.9	**37.1**

Table 4.4: Human annotations of the correctly classified NLEs generated by models with WinoGrande and ComVE as child tasks (CT). The columns Yes, Weak Yes, Weak No, and No present the percentages of NLE validity scores given by the human annotators. The last five columns present the shortcomings provided by the human annotators. The best results are in bold. I do not bold the Weak Yes and Weak No, since it is not clear that higher/lower is better.

that it produces NLEs that are closer to the test NLEs in terms of the low-level features that are captured by BLEU-1,2,3,4 (unigram, bigram, trigram, and four-grams). This can be explained by the fact that many training NLEs resemble one of the two statements in ComVE, because often the correct statement is a valid trivial NLE. Upon manual inspection, I noticed that the baseline CD–fine-tune does produce more trivial NLEs (that repeat part of the input statements).

Full Human Evaluation Results Table 4.4 presents the full human evaluation results table for all models which includes the separate Yes, Weak Yes, Weak No, and No scores. Table 4.4 also summarizes, for each model, the shortcomings that the human annotators found in the model-generated NLEs, which are informative of the issues with NLE generation.

Both for WG and ComVE, the best-performing model in terms of NLE score (M2 and M4, respectively) is also the highest in terms of Yes% and lowest in terms of No% scores, which shows that the NLE score is summarising correctly the more granular labels (Yes, Weak Yes, Weak No, No). This is also confirmed by the worse-performing model – WT5 which gets the lowest Yes score and highest No score.

For WinoGrande, better-performing models (M2 and M4) produce relatively low percentages of nonsensical (28.1% and 34.3%, respectively) and the highest percentages of NLEs without shortcomings (28.1% and 23.0%, respectively). On the other hand, the worse-performing models in terms of NLE score (such as WT5 and WT5–fine-tune) produce significantly more nonsensical NLEs (71.1% and 60.8%) when compared to the other models (maximum of 43.7%). In comparison, M1-M4 produce more partially correct NLEs ("insufficient justification"), with a minimum of 29.3% for M3 compared to 12.8% and 20.3% for the single-task baselines WT5 and WT5–fine-tune, respectively.

For ComVE, the best-scored model, M4, also has the lowest proportions of nonsensical (18.8%) and trivial (2.9%) explanations, and the highest percentage of NLEs without shortcomings (37.1%). This shows that M4 produces NLEs that make more sense (more coherent) and have fewer shortcomings than other methods.

Qualitative Examples Here, I present a short qualitative analysis of the results with examples generated by the trained models.

WinoGrande. While many of the correct NLEs that are generated by the models are trivial, e.g.:

> Fiction books were interesting and easy to read for Logan but not Brett because __ enjoyed real tales. **A:** Brett.
> **NLE:** Fiction books were interesting and easy to read for Logan but not Brett because Brett enjoyed real tales

there are some non-trivial generations, e.g.:

> Brett was browsing the internet while he found the information unlike Randy, __ prefers using books. **A:** Randy.
> **NLE:** If Brett is browsing the internet, then he is more likely to find information than Randy, who prefers to use books.

ComVE. By manual inspection, the correct model-generated NLEs are either a repetition of the correct statement or a negation of the nonsensical statement. Examples from each:

> **Statements:** The fire will [burn / cool] you if you touch it. **A:** burn.
> **NLE:** The fire will burn you if you touch it.

Statements: He inserts [his thumb / the keys] into the car ignition switch.
A: the keys. **NLE:** One cannot insert his thumb into the car ignition switch.

For more qualitative results, see Appendix C.4, where six non-cherry-picked NLEs from each model are presented.

4.2.5 Conclusion

In this work, I investigated four methods for few-shot out-of-domain transfer learning of NLEs for the abundant-label setting. I introduced *small-e-WinoGrande*, a dataset of NLEs on top of a small sample of instances from WinoGrande. I showed that out-of-domain few-shot learning can significantly help with NLE generation compared to zero-shot or child-task-only learning. Amongst the four NLE few-shot learning methods, I found that the most convincing NLEs are generated by the methods that provide separate training regimes for the child task and its few training NLEs. While the results indicate that few-shot out-of-domain transfer learning of NLEs is helpful, there is room for improvement both in the quality of the generated NLEs and in task performance. Thus, this work provides an essential foundation for future research into few-shot out-of-domain transfer learning of NLEs where label abundance is available.

Limitations The training methods in this work can apply to any language other than English, but a large parent task with NLEs is needed and a high-performance pre-trained generative LM may be needed for that language. Training one of the methods takes approximately three hours per e-SNLI epoch on one NVIDIA TITAN Xp GPU, which should be multiplied by the number of epochs and the hyperparameter combinations used. In total, the required time to reproduce all the results is approximately 45 GPU days. However, applying these methods in practice is much cheaper because one can select hyperparameter intervals informed by this work and use smarter hyperparameter search strategies than grid search.

Reproducing the human evaluation for all models in this work can be relatively expensive for researchers with a limited budget – the cost was $1 per 10 model-generated NLEs examples.

4.3 Scalability Experiments

This section is a follow-up to the main work in Section 4.2. Here, I explore the scalability of the best methods from Section 4.2 (Table 4.4) in terms of model and data size. I show that the methods scale overall well, but I observe a transitory drop in performance when the model transitions from transfer learning to in-domain learning as the number of training NLEs increases.

Experiments I explore the scalability of the best methods from Section 4.2, which are M2 for WG and M4 for ComVE. I use the same LM, datasets, hyperparameters and other details as in the main work (Section 4.2) unless specified otherwise.

For the model scalability experiment, in addition to the T5-Base model (220M parameters), I also train a larger model T5-Large (770M parameters) (Raffel et al., 2020). Both LM variants are trained on 50 NLEs from the child dataset – the same number as in Section 4.2. For the data scalability experiment, I train the best model configurations for WG and ComVE on 0, 25, 50 (the default), and 100 NLEs from the child task. For WG there are only 100 training examples with NLEs, but for ComVE there are many more available (10,000), which is why I train models on up to 200 training examples with NLEs for ComVE in this experiment.

Similarly to the main work (Section 4.2), I use both human evaluation (NLE score) for both WG and ComVE and automatic evaluation for ComVE only. In Section 4.2, I have observed that among the automatic metrics, BERTScore correlates the best with the NLE score, so I only report BERTScore here, as an additional verification.

Results The results in Table 4.5 show that the larger model performs better on WG as expected, both with better accuracy and NLE score. On the other hand, on ComVE the large model performs slightly worse than its small-model counterpart in terms of NLE score (45.8 vs 48.5). The ComVE result can be explained in two ways: first, looking closely at the per-category results, the large model produces more Weak Yes, but less Yes, whereas it has a similar number of No and Weak No as the small number, hence the NLEs produced by that model are not bad. Second, the large model obtains much better accuracy @100, so it has to produce more NLEs since I only evaluate NLEs for correctly classified examples.

The results in Table 4.5 show that the methods scale well from 0 to 25 to 50 training NLEs for both WG and ComVE, and both in terms of NLE score and BERTScore. The NLE score drops off from 50 to 100 NLEs for both WG and ComVE, but the

WG Model	Acc @100	NLE score	Yes%	Weak Yes%	Weak No%	No%	BERT Score
T5L–M2 (50)	**68**	**49.5**	**27.0**	27.9	11.8	**33.3**	–
M2 (0)	64	10.6	3.6	7.3	6.2	82.8	–
M2 (25)	64	40.5	19.3	25.5	12.5	42.7	–
M2 (50)	63	44.1	25.9	18.0	18.5	37.6	–
M2 (100)	63	40.4	20.6	22.2	14.8	42.3	–
ComVE Model	**Acc @100**	**NLE score**	**Yes%**	**Weak Yes%**	**Weak No%**	**No%**	**BERT Score**
T5L–M4 (50)	**87**	45.8	29.5	22.2	4.6	43.7	83.2
M4 (0)	83	25.4	10.4	18.9	7.2	63.5	78.5
M4 (25)	82	43.2	26.4	22.4	5.7	45.5	83.1
M4 (50)	79	48.5	36.7	14.3	6.8	42.2	83.6
M4 (100)	81	46.2	32.5	18.5	4.1	44.9	83.9
M4 (200)	79	**49.9**	**35.9**	18.6	5.1	**40.5**	**84.4**

Table 4.5: Scalability performance. The first line in each dataset section shows the results when replacing T5-Base with T5-Large in the best-performing models (M2 for WG and M4 for ComVE). The other models for each dataset show the scalability w.r.t. the number of NLEs for the child task (in brackets). The columns Yes, Weak Yes, Weak No, and No present the percentages of NLE validity scores given by the human annotators. Only correctly classified examples are included in these scores. The best results are in bold. I do not bold the Weak Yes and Weak No, since it is not clear that higher/lower is better.

results on ComVE suggest that this is transitory because the NLE score improves at 200 NLEs. This phenomenon may be due to the change from relying on the parent task (few-shot transfer learning mode) to relying more on the child-task NLEs (multi-shot learning mode). For ComVE, the drop in performance at 100 NLEs may also be insignificant due to the small number of human-evaluated NLEs ($\approx$80), and because the BERTScore on the full test dataset does not register such a drop in performance.

Conclusion In this section, I explored the scalability of the best few-shot out-of-domain transfer learning methods from Section 4.2 for NLE generation, both in terms of data and model size. I showed that the best methods for WinoGrande and ComVE scale similarly in data size, and that larger model sizes help with performance on the more difficult WinoGrande dataset, but not as much on ComVE. I also observed a temporary drop in performance with increasing the number of training NLEs likely due to shifting from relying on the parent task NLEs to relying more on the child task NLEs.

4.4 Conclusion

In this chapter, I explored the few-shot out-of-domain transfer of NLEs in the label-abundant setting. I showed that transfer learning in this setting can be achieved, as evaluated by the gold-standard human evaluation. I introduced transfer and multi-task learning methods for this setting and determined the best-performing methods. I showed that the best method depends on the task at hand (hard cases of pronoun resolution or commonsense validation), but that the few child-task NLEs should be on a separate training regime. Furthermore, I explored the scalability of the best methods in terms of data and model size, which I observed to be overall good. The results indicate that increasing the number of training NLEs shifts the model from relying on the NLEs of the parent task to relying on the NLEs of the child task.

In absolute terms, the best-performing models still achieve less-than-perfect NLE scores, which shows that there is room for improvement in terms of NLE quality. Furthermore, more powerful LMs may be needed for hard datasets such as WinoGrande, where the task accuracy could hold back the NLE quality.

Chapter 5

Predictive Coding for Language Models

Biological plausibility is an important aspect of creating human-like language models (LMs). In recent years, the predictive coding theory of the brain has been adapted to multiple training algorithms for machine learning with desirable properties (Song et al., 2020). These algorithms present a promising direction of research with biological plausibility (Song et al., 2020), theoretically faster training speeds with analogue hardware (Song, 2021), support for more complex architectures (Salvatori et al., 2022a), and more parameter-efficient models (Song, 2021). These characteristics can enable future applications to challenging tasks such as hard cases of pronoun resolution (Chapter 3) and producing high-quality NLEs (Chapter 4). In this chapter, I explore the application of these new algorithms for training recurrent-neural-network-based (RNN-based) LMs and for training transformer LM architectures such as BERT (Devlin et al., 2019) and GPT (Radford et al., 2018). This chapter includes the first application of biologically plausible predictive coding methods to training LMs (Section 5.3), and the first application of predictive coding to transformer models (Section 5.4). This chapter lays the groundwork for human-like LMs in terms of the biological plausibility of their training method, by using methods inspired by the predictive coding theory of the brain.

In Sections 5.3 and 5.6, I show how to apply predictive coding methods to train LMs – I show what architectural modifications are beneficial (such as layer normalisation) and explore the choice of hyperparameter values. In Section 5.4, I explore what energy function works best for predictive coding via the KL-modification (Pinchetti et al., 2022), whereas in Section 5.5, I show what method works best for transformer LMs, namely that PPC (Song, 2021; Salvatori et al., 2022b) is better than PC (Whittington and Bogacz, 2017).

I show that LMs trained with predictive coding (more specifically via PPC) can perform on par with BP: in Section 5.3, I show that PPC outperforms BP for simple RNN LMs, while in Sections 5.4 and 5.5, I show that PPC performs on par with BP for small transformer LMs. Furthermore, I show how predictive coding scales in terms of data and model size: in Section 5.3, I show that PPC scales similarly to BP for simple RNN LMs, whereas in Section 5.6, I show that PPC scales overall worse than BP for transformer LMs. However, the results in Sections 5.5 and 5.6 reveal that the choice of transformer LM variant matters because PPC performs and scales better with masked over conditional LMs.

The first part of this chapter (5.3) is based on my term paper titled *Predictive Coding for Language Models* (2020). In it, I explore various ways to apply Parallel Predictive Coding (PPC) (Song, 2021) to train RNN-based LMs. The main findings are that PPC significantly outperforms backpropagation (BP) for RNN LMs when trained on small datasets, with the gap between the two narrowing for large datasets. On the other hand, PPC scales similarly to BP in terms of model size.

The second part of this chapter (5.4) is based on a work that I co-authored: *Predictive Coding Beyond Gaussian Distributions* (Pinchetti et al., 2022) published at the NeurIPS 2022 conference, more specifically section "4.3 Transformer Language Models" in that paper. In this work, Pinchetti et al. (2022) diagnose and propose a solution (the *KL-modification*) to the problem of having *softmax* functions in deep neural networks when training via predictive coding. My specific contribution to this work is in making transformer LMs work with the new KL-modification method. I demonstrate that the KL-modified method performs better than the predictive coding baseline, and is on par with BP in terms of performance for small transformer LMs.

The third part of this chapter (5.5) is based on a work that I co-authored: *A Stable, Fast, and Fully Automatic Learning Algorithm for Predictive Coding Networks* (Salvatori et al., 2024) accepted at the ICLR 2024 conference, more specifically section 5 "Language Model Experiments" in that paper. It proposes the iPC method (named PPC in this thesis) as an improvement to the PC method (Whittington and Bogacz, 2017) that is more biologically plausible and computationally efficient. My contribution to this work is in implementing the method for transformer LMs and expanding the experiments to include masked and conditional LMs, thus establishing the best method to train transformer LMs. I demonstrate that the PPC method outperforms PC for both conditional and masked LMs, both in terms of performance and stability, and is on par with BP for small LMs.

The fourth part of this chapter (5.6) is a follow-up to the previous two parts (5.4 and 5.5) that aims to answer questions about whether larger LMs and other LM variants such as masked LMs can be trained effectively with predictive coding. I also find some key hyperparameters and architectural choices that are most suitable for predictive coding. The results show that in terms of longer training, PPC scales worse than BP for both conditional and masked LMs, whereas in terms of model size, PPC scales similarly to BP for masked LMs, but worse for conditional LMs. This means that large LMs are still out of reach and that further improvements are needed to the scalability of predictive coding methods.

5.1 Literature Review

In this literature review, I introduce predictive coding as a neuroscience theory, its history (first works), backpropagation alternatives, and applications of predictive coding in machine learning. I also explore the literature on energy-based models, which is closely related to predictive coding. Finally, I mention the methods and results from existing literature that I use in my work, and the contribution of my work to the field.

Predictive Coding Theory Predictive coding is a neuroscience theory of how the brain works that states that the brain processes information by minimising prediction errors. According to the theory, the brain operates with multiple levels of hierarchy, where the neurons of each level predict the input from the level below and adjust based on the error between prediction and input (Clark, 2013). Figure 5.1 illustrates this.

Bayesian predictive coding incorporates Bayesian principles on belief updates into the predictive coding theory (Knill and Pouget, 2004). In Bayesian predictive coding, the usual assumption is that the prediction error distributions are Gaussian (Friston, 2005), although some works have generalised this to non-Gaussian distributions (Pinchetti et al., 2022). Bayesian predictive coding is closely related to the free energy principle (Friston, 2010) because the latter is also based on Bayesian principles and it also minimises the prediction error (surprise) about the observations under the agent's model of the world.

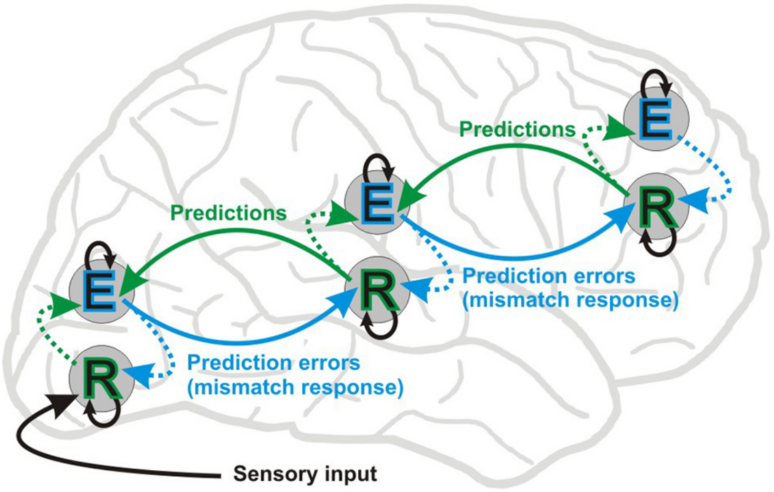


Figure 5.1: Diagram of predictive coding in the brain. The brain updates based on the mismatch between the representation units (R) and the error units (E) at each level of abstraction. Credit: Stefanics et al. (2014).

First Works The term *predictive coding* was popularised and formalised by Srinivasan et al. (1982) who describe it as removing predictable components from the signal, to enable sensory neurons to focus on important (non-predictable) observations. However, these predictive coding principles were implemented as early as the 1950s with works by Harrison (1952) on decorrelation and linear prediction of television signals and by Oliver (1952) who propose *predictive-subtractive coding* for quantising (discretising) messages.

Rao and Ballard (1999) propose a hierarchical predictive coding architecture for perception (computer vision), where lower-level neurons aggregate local information and pass it to higher-level neurons, which capture more global information. They show that their model identifies edges and contours of objects, which has been proven to occur in the visual cortex, as well as other effects observed in the brain.

Friston (2005) proposes a Bayesian predictive coding framework that captures several key aspects of brain organisation and function: it is hierarchical, it has forward and backward connections with different roles, it has associative plasticity (where connection strength is modified during learning), and responses to repeated stimuli are suppressed. Friston (2005) draws a clear distinction between inference – "finding the conditional expectation of the causes", and learning – identifying "the maximum likelihood value of the parameters".

More detailed overviews of the history of predictive coding are provided by Salvatori et al. (2023)and Millidge et al. (2021).

Backpropagation Alternatives There have been many proposed alternatives to backpropagation (BP): Difference Target Propagation (Lee et al., 2015), Online Alternating Minimisation with Auxiliary Variables (Choromanska et al., 2019), Decoupled Neural Interfaces using Synthetic Gradients (Jaderberg et al., 2017), Training Neural Networks with Local Error Signals (Nøkland and Eidnes, 2019), Fully-Autonomous Zero-Divergent Inference Learning (FaZIL) (Song et al., 2020), and many others. Some training methods are introduced to address the biological implausibility of BP (Song et al., 2020), whereas others focus on overcoming its computational limitations (Huo et al., 2018). All training methods are evaluated in the computer vision domain.

Predictive Coding in Machine Learning The biologically-inspired predictive coding (McClelland and Rumelhart, 1988) has been established as one of the most promising alternatives to BP (Whittington and Bogacz, 2017; Boutin et al., 2019; Millidge et al., 2020; Song et al., 2020; Song, 2021; Salvatori et al., 2022b). While almost all predictive coding methods have been developed for computer vision, there are very few methods for text generation (Oord et al., 2018; Pinchetti et al., 2022; Millidge et al., 2023c; Li et al., 2023; Salvatori et al., 2024). This is likely because sequence-to-sequence predictive coding models such as applications to RNNs (Millidge et al., 2023c; Li et al., 2023) and transformers (Pinchetti et al., 2022; Salvatori et al., 2024) lag in development to feed-forward architectures (Whittington and Bogacz, 2017; Song et al., 2020).

Whittington and Bogacz (2017) introduce a supervised predictive coding method (denoted as PC) with local synaptic plasticity, i.e. weight updates are based on a local learning rule without the need for global update such as backpropagation of errors. In this method, for each batch of data, inference is done until convergence, followed by a weight update. Whittington and Bogacz (2017) show that the PC method can converge to BP under certain conditions. Salvatori et al. (2022a) demonstrate that PC can be applied to neural networks with arbitrary structure given by *graph topologies*. These include complex structures such as networks with loop connections (connections that loop into themselves) and backward connections (the opposite of skip connections). Such structures cannot exist for BP, but are enabled in predictive coding by the existence of the time dimension via the inference steps. Millidge et al. (2020) demonstrate that predictive coding approximates BP, while only relying on local update rules, and demonstrate how predictive coding can be applied to any neural architecture.

Song et al. (2020) propose a predictive coding method (FaZIL) that enables fully parallel training across neurons and layers. This method demonstrates that a biologically plausible training method can approximate BP, with minimum computational overhead. (Song et al., 2020) also propose the Z-IL training method – a variation of the PC method, where the weight update of layer l happens at $t = l$, instead of updating together with all other layers at $t = T$. Salvatori et al. (2021) later prove that Z-IL is equivalent to BP for convolutional and recurrent neural networks. (Salvatori et al., 2022c) then further improve Z-IL to exactly replicate the parameter updates in BP for arbitrary computational graphs, and with more biological plausibility than BP. (Millidge et al., 2023b) prove theoretical results that deep predictive coding networks "can, in theory, achieve the same generalization performance as BP, while maintaining their unique advantages". Song et al. (2024) introduce *prospective configuration* – a framework for interpreting energy-based predictive coding methods that follow the inference-then-weight-update formula – e.g. PC (Whittington and Bogacz, 2017). They show that such predictive coding networks are equivalent to energy machines.

Song (2021) and Salvatori et al. (2022b) propose the parallel predictive coding method (PPC), which is also the method of choice in this thesis. PPC has a fully local update rule, where inference and weight updates happen simultaneously, and there is no need for a signal to switch between the two modes unlike PC (Whittington and Bogacz, 2017). Salvatori et al. (2022b) show that PPC is much faster than PC and is competitive to BP for small network architectures.

Pinchetti et al. (2022) conjecture that the softmax function, common in many NN architectures, cannot be effectively captured by the standard Gaussian assumption in predictive coding (Friston, 2005). They propose a modification, which I denote as the *KL-modification*, which works for non-Gaussian distributions. Pinchetti et al. (2022) show that the KL-modification improves the performance on several architectures and tasks, and is particularly effective for transformer LMs, where the softmax function is not only at the network output, but also at the self-attention mechanism in each transformer block. These transformer experiments are included in Section 5.4 of this thesis. Salvatori et al. (2024) follow-up on the work by Pinchetti et al. (2022) by showing further progress on applying the PPC method to transformers (Section 5.5 in this thesis). They show that PPC outperforms PC (Whittington and Bogacz, 2017) for small-scale conditional and masked transformer LMs, and is on par with BP.

Millidge et al. (2023c) show that RNNs can be natively implemented within the predictive coding framework, and apply it for predicting noisy time series. Li et al. (2023) also propose a predictive coding method for RNNs based on the *mean-field*

dynamics of the model, and show improvement over the vanilla PC method in computer vision and NLP tasks. Tang et al. (2023) propose the temporal predictive coding (tPC) method, which efficiently memorises and retrieves large sequences, in a similar way to how this is done in the brain.

Ororbia (2019) propose a predictive coding method for spiking neural networks that uses a local synaptic update rule. They demonstrate promising results in image classification and online learning with video input. Ororbia and Kifer (2022) propose neural generative coding (NGC) – a generative implementation of predictive coding with local update rules. They also experiment with adding skip connections directly from the input, as well as recurrent connections (loops), both of which they show to improve performance on computer vision tasks. Zahid et al. (2023) introduce the Langevin Predictive Coding (LPC) generative method, which modifies the standard predictive coding framework by introducing noise in the inference process and an appropriate model initialisation. They show superior image generation results to Variational Autoencoders (VAEs) (Kingma and Welling, 2014), with much faster convergence.

Oord et al. (2018) propose the Contrastive Predictive Coding (CPC) method which applies the predictive coding principle of predicting future observations to unsupervised contrastive learning. The authors demonstrate the usefulness of CPC on audio, image, and textual tasks, as well as for reinforcement learning. Unfortunately, the CPC method is not biologically plausible because it does not have local update rules.

Energy-Based Models Energy-based models such as predictive coding (Rao and Ballard, 1999) and Hopfield networks (Hopfield, 1982) are models trained by minimising an energy function. Predictive coding (PC) is a type of energy-based method, where an energy function is minimised between predicted states and actual states (x and μ, respectively) (for more details see Section 5.2).

Millidge et al. (2023a) generalise the results by Millidge et al. (2020) and Song et al. (2020) that show that energy-based predictive coding methods such as PC (Whittington and Bogacz, 2017) and FaZIL (Song et al., 2020) approximate BP. Millidge et al. (2023a) show that these models approximate BP under more general conditions and that these conditions can, in turn, be used to derive new energy-based alternatives to BP with desirable properties.

An energy-based architectural alternative to the transformer has been proposed by Hoover et al. (2023) – The Energy Transformer, which "combines aspects of three

promising paradigms in machine learning, namely, attention mechanism, energy-based models, and associative memory".

My Work Here, I list the insights and methods from existing literature that I use in my work and I summarise the contribution of my work to the field.

I use two methods from existing works: the PC method Whittington and Bogacz (2017) and the PPC method (Song, 2021; Salvatori et al., 2022b), both introduced above in this Literature Review. Salvatori et al. (2022b) show that PPC is much faster than PC and matches BP for small NNs, which is why the main method I use is PPC. In this thesis, I use PC only to the extent that I compare PPC to it in Section 5.5. In Section 5.5 (based on my contribution to our work (Salvatori et al., 2024)), I show that for transformer-based LMs, PPC outperforms PC, and is more stable and faster.

I also use the KL-modification by Pinchetti et al. (2022), which I show in Section 5.4 (based on my contribution to our work (Pinchetti et al., 2022)) that it helps for training transformer LMs, and is on par with BP for small LMs. My work in Section 5.4 is also the first work on applying predictive coding for training transformer LMs. In Section 5.3, I present (to the best of my knowledge) the first application of a biologically plausible predictive coding method (PPC) to natural language processing. I explore the best way to apply PPC and show successful training of RNN LMs. I also show that PPC scales on par with BP and outperforms BP for simple RNN LMs. On the other hand, in Section 5.6, I show that the PPC method scales worse than BP for transformer LMs. I also show that PPC performs better for masked than conditional LMs.

In summary, this thesis shows the first results on training LMs with predictive coding methods and explores the best ways to implement them and their scalability. This can unlock the practical application of these methods to training LMs.

5.2 Background

This is the background for understanding predictive coding and the methods used in this thesis. In 5.2.1, I introduce the predictive coding methods used in this thesis. In 5.2.2 and 5.2.3, I go into detail on how predictive coding works.

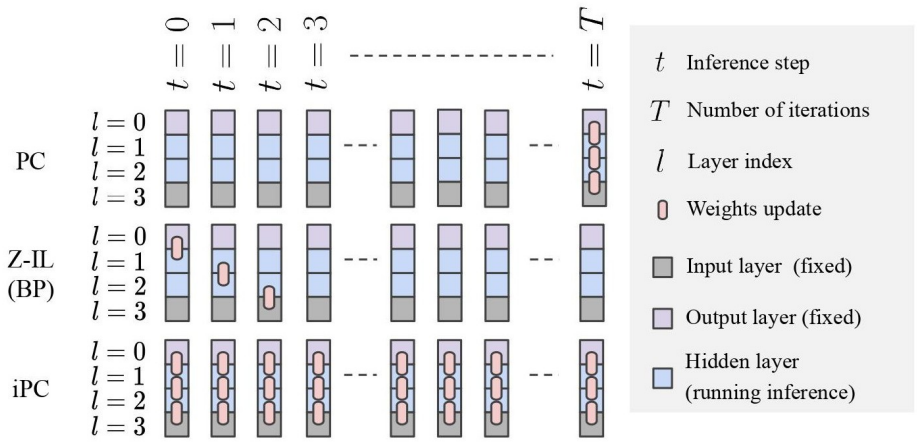


Figure 5.2: Top: PC; bottom: iPC (PPC). Credit: Salvatori et al. (2024).

5.2.1 Predictive Coding Methods

In this thesis, I use two predictive coding methods: PC (Whittington and Bogacz, 2017) and parallel predictive coding (PPC) (Song, 2021; Salvatori et al., 2022b). But to understand these predictive coding methods, we need to define weight and inference updates first. *Inference update* means updating the predictions of the node values at the next time step (denoted as μ) by updating the input to the next layer (denoted as x). Here, the time dimension is a consequence of the model looking at each batch T subsequent times. On the other hand, *weight update* means updating the node states by updating the underlying model weights.

Both in PC and PPC, there are multiple inference steps per batch, and their number is usually denoted by T. Where PC and PPC differ is that while PC does one weight update per batch (following all of the inference updates), PPC does T weight updates – simultaneously with the inference updates. This is illustrated in Figure 5.2.

The simultaneous (parallel) weight and inference updates in PPC motivate the notation *parallel predictive coding* introduced by Song (2021) which I use in this thesis.

5.2.2 The Predictive Coding Architecture

PC and PPC both require architectural modifications in the form of predictive coding layers (PC Layers). This is illustrated in Figure 5.3.

The PC Layer is a neural network layer used to define μ and x values for predictive coding and to compute the energy function between them: $E = (\mu - x)^2$, where μ is the output of the previous layer, and x is the input to the next layer. This is depicted in

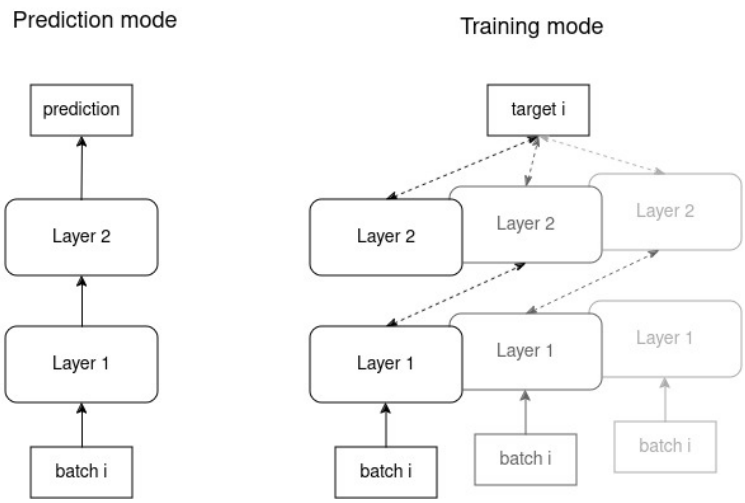


Figure 5.3: An illustration of the PC architecture in prediction (left) and training (right) modes. The PC model is disconnected during training and the signal is propagated only through the energy function (dotted lines) between nodes at successive time steps $t = 1, \ldots, T$. During prediction, the PC architecture is equivalent to its BP counterpart.

Figure 5.3 as dotted lines. The PC Layers are used between other network layers, e.g. after linear and before tanh activation: $output = \tanh(PCLayer(Linear(input)))$. PC Layers are only enabled during model training and are disabled during model evaluation by switching them to identity layers: $PCLayer(x) = x$, which allows for generating predictions. This makes PC Layers similar to layers such as dropout used for BP.

As visible from Figure 5.3, we need the number of inference steps T to be more or equal to the number of PC Layers for the signal to propagate from input to output. Figure 5.3 also shows that blocks can be executed in parallel at each time step.

5.2.3 Loss and Energy Functions

For PC and PPC, there are two types of errors that are optimized jointly: energy error (E) and NN loss (L). The NN loss (L) is the usual (cross-entropy) loss between the output of the final neuron(s) and the target value. The energy error (E) is the average across all PC Layers of the squared error between predicted and true value:

$$E := \frac{1}{\#\text{PC Layers}} \sum_{l \in \text{PC Layers}} (\mu_l - x_l)^2 \tag{5.1}$$

where x and μ are as defined above. The objective function is to minimize the total loss $L_{\text{total}} := E + L$. Conceptually, the loss function (L) can also be considered a part

of the energy function (E) as the energy at the PC Layer between the output and the target of the NN.

During training, PPC updates jointly the weights (*weight update*) and the energy parameters (*inference step*), T times per batch. PC, on the other hand, does T *inference steps*, followed by one *weight update* step. During evaluation, the PC Layers are disabled (replaced by the identity function) and the forward pass is thus equivalent to the forward pass in the original NN.

The KL-modification Pinchetti et al. (2022) propose the KL-divergence modification of the energy function of predictive coding methods. It computes the energy error identically to the standard PC/PPC methods as an MSE error: $E = (x - \mu)^2$, except when applied to the softmax function, where the formula changes to:

$$E = softmax(x) \log \left(\frac{softmax(x)}{softmax(\mu) + 10^{-8}} + 10^{-8} \right) \tag{5.2}$$

If the PC layer is applied at the softmax at the output of a neural network, then x is pinned to the 1-hot target so there is no need for softmax at x, and then Eq. 5.2 becomes:

$$E = x \log \left(\frac{x}{softmax(\mu) + 10^{-8}} + 10^{-8} \right) \tag{5.3}$$

When x is the target vector, Equation 5.3 gives equivalent gradients to using cross-entropy loss because the cross-entropy and the KL-divergence differ by a constant for a constant target distribution (Pinchetti et al., 2022).

5.3 Predictive Coding for RNN-Based Language Models

This work is based on my term paper titled: *Predictive Coding for Language Models* (2020). The term paper experiments were redone in this work.

This work is the first work that applies a biologically plausible predictive coding method to train LMs. I use the Parallel Predictive Coding (PPC) (Song, 2021; Salvatori et al., 2022b) training method which is a biologically plausible alternative to BP, because it does not require a synchronization node and has local updates. Furthermore, PPC has shown promising results on various tasks, with better data and parameter efficiency than BP (Song, 2021; Salvatori et al., 2022b). In this work, I make a fair comparison between BP and PPC on language modelling with recurrent neural networks (RNNs). I explore ways to implement PPC, do a comprehensive hyperparameter search, and test PPC's data and parameter scalability against BP. The experiments show that PPC significantly outperforms BP on small datasets, but the gap between the two narrows for larger datasets. On the other hand, the experiments show similar parameter scalability between PPC and BP. These results show the first indication that predictive coding methods can be an effective alternative to BP for training LMs.

5.3.1 Introduction

Backpropagation (BP) has been established as the default and best method to train deep learning models. And while BP has produced the best empirical results for training large models (Brown et al., 2020; Shoeybi et al., 2020), there have been doubts about its biological plausibility (Bengio et al., 2016; Whittington and Bogacz, 2017; Lillicrap et al., 2020), and its limited computation parallelizability (Huo et al., 2018).

There have been many other proposed training methods aimed at addressing some of these problems (Lee et al., 2015; Choromanska et al., 2019; Jaderberg et al., 2017; Nøkland and Eidnes, 2019). Predictive coding training methods are one of the most promising, with recent work by (Song et al., 2020) demonstrating good empirical results and improved parallelizability. As an improvement to the method proposed by Song et al. (2020), its authors have subsequently created the Parallel Predictive Coding (PPC) training method (Song, 2021; Salvatori et al., 2022b), which is the method of choice in this work.

While LMs are the most widely used machine learning models for natural language processing, evident by the existence of the research field of BERTology (Rogers et al., 2020; Devlin et al., 2019), there has been no comparison in language modelling between predictive coding methods (such as PPC) and BP. The problem with such a comparison is that none of the state-of-the-art LMs (Devlin et al., 2019; Liu et al., 2019b; Brown et al., 2020) has been adapted for PPC, whereas they have been adapted for BP with "tricks" like dropout (Srivastava et al., 2014), batch normalization (Ioffe and Szegedy, 2015) and layer normalization (Ba et al., 2016). Therefore, we cannot make a fair comparison between a new approach such as PPC and a mature approach such as BP by using any state-of-the-art LM architecture. In this work, I aim to address this problem by using a simplified (RNN) LM architecture to compare PPC and BP on an equal basis. I compare the performance and the data and parameter scalability when training with PPC versus BP. Since PPC is a new training approach, I also study the correct ways to implement PPC and do an extensive hyperparameter search to determine the importance of each hyperparameter choice for the performance of LMs.

5.3.2 Related Works

There are several closely related works to this one: (Whittington and Bogacz, 2017; Song, 2021; Salvatori et al., 2022b), as well as these two more recent works: (Pinchetti et al., 2022; Millidge et al., 2023c).

Whittington and Bogacz (2017) introduce a supervised predictive coding method which I refer to as *PC* in this thesis. PC works by minimising an MSE energy function, where for each batch, the inference is done until convergence, followed by a single weight update (see Section 5.2).

Song (2021) and Salvatori et al. (2022b) introduce the parallel predictive coding (PPC) method used in this work. Unlike PC, PPC uses only local computations without the need for a global signal. Furthermore, PPC updates the model weights at each inference step, rather than only at the end of the inference, which makes it more efficient.

Pinchetti et al. (2022) present the *KL-modification* for PPC (Song, 2021) that deals with problems created by the softmax function, and they show significantly better performance on a variety of architectures and tasks. In practice, the KL-modification is equivalent to using cross-entropy loss for NNs with only one softmax function (at the output) (see Section 5.2.3), which is what I use in this work.

Millidge et al. (2023c) show that RNNs can be implemented natively for predictive coding, which can lead to improved performance for noisy time series. This work is much more recent than the term paper comprising this chapter, so I only mention it as a future direction of research.

There is only one work prior to my work that applies a predictive coding-inspired method to natural language – the work by (Oord et al., 2018). However, as discussed in Section 5.1, their method is only partially based on predictive coding and is not biologically plausible because it does not have local update rules.

5.3.3 Method

The main comparison in this work is between two training methods: PPC (a predictive coding method), and backpropagation (BP). I have selected PPC (Song, 2021; Salvatori et al., 2022b) as the training method over PC (Whittington and Bogacz, 2017) because it is more biologically plausible and because Song (2021) and Salvatori et al. (2022b) have shown it outperforms PC for small NNs. Instead of using transformers (Vaswani et al., 2017), which is a BP-optimized architecture, I use vanilla RNNs which would provide a more fair comparison for PPC vs BP.

RNNs suffer in terms of efficiency due to their recurrence, where an unrolled RNN forms a sequence of identical layers. The use of PPC can, in theory, counter the lack of parallelisability of RNNs across their sequence length because PPC only uses local computations (see Section 5.2). Unfortunately, the PPC codebase (and ML libraries such as PyTorch) that I use does not support the parallelisation across layers needed to test this, hence I do not report training times and other computation metrics.

Model Choice For fairness of comparison, I use a simple architecture: RNN, without any of the modifications and tricks that have been developed with BP in mind. This is because no architecture has been optimized for PPC yet. More specifically, I use a uni-directional multi-layer RNN with a vanilla RNN cell and with a standard choice of a *tanh* activation function for the cell. The RNN cell is defined as: $s_i = tanh(Linear([s_{i-1}, x_i]))$, where s_i are the hidden states, and x_i are the inputs. For more details on (RNN) LMs, see Section 2.1.2.

I do not use LSTM cells because they were introduced to help BP with long-term dependencies, by storing relevant information in the memory cell, and by maintaining a memory channel through the unrolled RNN (Hochreiter and Schmidhuber, 1997). Also, I do not use dropout since it is a BP-specific model modification.

PPC details There are several important choices for implementing and training with PPC: the PC Layer placement and output clipping, the loss function, the number of training steps per batch (T), and the inference learning rate ($lr_{inference}$).

PC Layers: In PPC, predictive coding layers (PC Layers) can be placed at any neural connection (see Section 5.2.2). I am going to evaluate the two most sensible choices of PC Layer placement: either before or after all activation functions. Additionally, if choosing to place PC Layers before activations, one can consider clipping the PC Layer output to fit in the range of possible output values of the activation function (e.g. for tanh this is $[-1, 1]$).

Loss function: In language modelling it is commonplace to use the cross-entropy loss between output and targets. There can be a potential issue for PPC since in the total loss, the cross-entropy loss is added to the energy error, which is a mean-squared error (MSE) loss (see Section 5.2.3 for details). Song (2021); Salvatori et al. (2022b) have suggested using the MSE loss with PPC for classification, but in language modelling, cross-entropy loss optimises directly for perplexity (e^{loss}), which is why I choose cross-entropy loss for both PPC and BP.

Training steps per batch: PPC works by doing multiple (T) training steps per batch, in succession (see Section 5.2). This poses the question of what the appropriate values for T are, and whether PPC can benefit from large batch sizes, which will be addressed in the Experiments section (5.3.4).

Inference updates: In addition to the usual weight update (w-update), PPC also simultaneously does inference update (x-update) (see Section 5.2 for more details). This introduces the inference learning rate $lr_{inference}$ hyperparameter in addition to the usual weight learning rate $lr_{weights}$. This raises the question of the appropriate values for $lr_{inference}$, which will be addressed in the Experiments section (5.3.4).

BP details As a baseline, I use backpropagation (BP), as formalized by Rumelhart et al. (1988). Unlike PPC, BP has a global update rule, where the error is propagated backwards layer-by-layer. This limits the parallelizability and introduces issues with synchronization when splitting across large computational resources (Shoeybi et al., 2020).

I do a full hyperparameter search for BP, which includes the learning rate (lr) and the batch size (bs).

Comparison I will compare BP and PPC in terms of data scalability, by varying the training data size, and in terms of parameter scalability by varying the number

of layers and the hidden size of the model. Salvatori et al. (2022b) show good results with PPC for small data sizes, but LMs require vast amounts of training data, so my experiments will determine if PPC scales well for large datasets.

Tokenizer and Vocabulary I use the SentencePiece tokenizer[1] with byte-pair-encoding (BPE) (Sennrich et al., 2016) subword units implementation from torchtext[2]. I choose SentencePiece over WordPiece (Wu et al., 2016) since the latter is not publicly available, and since SentencePiece has more features (listed in their repository). I select this tokenization method because it is a good middle ground between character and word embeddings, which combines the expressivity of word embeddings and the computation efficiency of character embeddings. The tokenizer was trained on a part of the training data, with a vocabulary size of 8,000.

Default Hyperparameter Values I use the following default hyperparameters for all experiments (unless otherwise stated): the hidden size, the intermediate size and the token embedding size are all equal to 128; the number of RNN layers is 2; the input sequence length is equal to 12 (including $<sos>$ and $<eos>$ tokens).

Dataset I use the 1 Billion Word Benchmark (1BW) (Chelba et al., 2013), which can be found on their official website[3]. I randomly split a validation (dev) set of size 10,000 from the training part of the 1BW dataset. Out of the rest of the training dataset, I subsample six subsets: train-S, train-M, train-L, train-XL, train-2XL, and train-3XL of sizes: 100,000; 200,000; 400,000; 800,000; 1.6M; and 3.2M respectively. The test dataset is provided, of size 306,688. The default dataset used as training data in the experiments is train-M (200K examples).

Training All models are trained on two epochs of the training dataset. The optimiser of choice for BP is AdamW (Loshchilov and Hutter, 2017). For PPC, I use AdamW for weight updates and Stochastic Gradient Descent (SGD) for inference updates, which I choose based on initial experiments.

[1] `https://github.com/google/sentencepiece`
[2] `https://pytorch.org/text/`
[3] `http://www.statmt.org/lm-benchmark/`

Evaluation All results are reported in terms of perplexity. The hyperparameter search results are reported on the dev set, whereas the data and parameter scalability experiments are reported on the test set. All models are evaluated on sequences up to 10 – same as for training.

5.3.4 Experiments

In total, I run three experiments. The first experiment is designed to find the best hyperparameters for PPC, which has new hyperparameters that are not present for BP. The last two experiments are focused on comparing the performance between the two training methods, by looking at data and parameter scalability.

For each experiment, I do a grid search over the specified hyperparameter choices and values. Due to the combinatorial explosion and the computational constraints, for each experiment, I choose strategically the hyperparameters to be searched over, with their sets of values. The hyperparameter search for PPC is split into two grid searches (in two parts) for this exact reason.

I run two statistical significance tests across the seeds for all experiments that use multiple seeds (3 seeds): the two-sample t-test and the paired-samples t-test, and I report both one-tailed and two-tailed p-values. I deduce statistical significance based on the worst (highest) p-value obtained across these tests, with a significance threshold of 0.05. The full list of p-values is available in Table D.1 in Appendix D.

PPC: hyperparameter choice In this experiment, I investigate the following hyperparameter choices: the PC Layer location (before or after activation functions), the use of output clipping for PC Layers, the batch size, the number of inference steps (T), the inference learning rate ($lr_{inference}$), and the weight learning rate (lr_{weight}). Due to the large number of hyperparameters, this experiment is split into two parts. In the first part, I determine the best batch size, the best PC Layer placement and whether to use PC Layer output clipping. In the second part, given these hyperparameter values, I find the best values for T and the energy and weight learning rates. For each of the two parts, I do a single grid search over the following hyperparameters and ranges:

Batch size: For the first part of the hyperparameter search, I choose a range of batch sizes: {64, 128, 256}. For the second part of the hyperparameter search, I fix the best value of the batch size.

PC Layer placement: For the first part of the hyperparameter search, I try two options for the PC Layer placement, with all PC Layers either before or after all

activation functions of the RNN model. I also try PC Layer output clipping for the after-activation case. For the second part of the hyperparameter search, I fix the best options from the first part.

Learning rates: Initial experiments with a limited hyperparameter search and by searching over the values $lr_{inference} \in \{0.25, 0.5, 1.0\}$ suggest that $lr_{inference} = 0.5$ performs best, which I will use as the default value in the first part of the hyperparameter search. In the second part of the hyperparameter search, I confirm the best choice of $lr_{inference}$ by searching over the values $\{0.25, 0.5, 1.0\}$. For the weight learning rate lr_{weight}, I do a geometric search with factor 2 over the range $[2.5 \times 10^{-5}, 4 \times 10^{-4}]$, for both parts of the hyperparameter search.

Training Steps Per Batch (T): I choose the minimum value for the number of training steps per batch (T) based on the number of layers l and the maximum sequence length (s) as follows: $\mathrm{T}_{min} = l + s - 2 = 2 + 12 - 2 = 12$. This makes sure that the error has propagated through the entire computation graph of the RNN model. In practice, larger values than 12 are better, and for both parts of the hyperparameter search, I search for $T \in \{15, 13, \ldots, 30\}$.

PC Layer output clipping: In the case when the PC Layer is after an activation function, one may choose to clip its output to the range of the output of the activation function, so that there is no discrepancy between the training and evaluation behaviour of PPC. I do this in the first part of the hyperparameter search, and only when the PC Layers are chosen to be immediately after the activation functions.

BP: hyperparameter ranges For BP, I do a grid search over the batch size and the learning rate over the values below.

Batch Size: selected from: {32, 64, 128, 256, 512}.

Learning Rate: selected via geometric search with factor 2 over the interval: $[6.25 \times 10^{-6}, 6.4 \times 10^{-3}]$.

Data scalability In this experiment, I compare PPC and BP when training on each of the six datasets: train-S, train-M, train-L, train-XL, train-2XL, train-3XL. This will serve three purposes: first, how the performance scales with data size; second, whether the gap between BP and PPC widens or shrinks as the training data increases, and third, to test whether PPC approximates BP for large dataset sizes. To account for seed-wise variation, for the smaller dataset sizes (S-XL), I run all models with 3 seeds, and for the larger dataset sizes (2XL, 3XL), I only run 1 seed due to the computational constraints.

Batch size	**64**	**128**	**256**	**512**	**1024**
Perplexity	261.7	249.7	237.6	**236.1**	238.6

Table 5.1: Best perplexity for each value of the batch size. The aggregation is minimum across the values for the other three hyperparameters: $T \in [15, 30]$, $lr_{weight} \in [4 \times 10^{-4}, 8 \times 10^{-4}, \ldots, 64 \times 10^{-4}]$, and PC Layer placement (before/after activation/after activation clipped). The best value is in bold.

Parameter scalability In this experiment, I train models with different numbers of parameters, to compare the parameter scalability of PPC and BP. I vary the number of layers and the hidden size of the model, as follows: the hidden size in: {64, 128, 256}, while using 2 layers; and the number of layers is either 1, 2, or 3, while the hidden size is 128. For training data, I use the default training dataset – train-M. I run all models on 3 seeds to account for the per-seed variability of results.

5.3.5 Results

These are the hyperparameter search experiments for PPC, followed by the data and parameter scalability experiments for PPC and BP.

PPC Hyperparameters First I search over the batch size PC Layer placement and using PC Layer output clipping. With those best values, I then search over the values of T, $lr_{inference}$, and lr_{weight}.

First part of hyperparameter search

The results in Table 5.1 suggest that PPC favours large batch sizes (512) versus BP which I find to perform best with a batch size of around 64. The results in Table 5.2 also show that PPC favours the PC Layer location of before activation rather than after activation functions (236.1 vs 250 ppl). This can be explained by the issue with PC Layers potentially producing state values x that are outside of the output range (-1, 1) of the tanh activation function. While output clipping is initially appealing for fixing this issue, it seems to lead to significantly worse performance (250 vs 590.1 ppl), which can be explained by the fact that output clipping deletes information about values outside of the target interval (-1, 1).

In summary, the best hyperparameters from the first part of the hyperparameter search are: batch size of 512, PC Layer placement immediately before all activation functions, and not using PC Layer output clipping. I fix these values for the second part of the hyperparameter search.

Second part of hyperparameter search

PC Layer place	before	after	after and clipped
Perplexity	**236.1**	250.0	590.1

Table 5.2: Best perplexity for each PC Layer placement: before activation, after activation, and after activation with output clipping. The aggregation is minimum across the values for the other three hyperparameters: $T \in [15, 30]$, $lr_{weight} \in [4 \times 10^{-4}, 8 \times 10^{-4}, \ldots, 64 \times 10^{-4}]$, and batch size (64, 128, ..., 1024). The best value is in bold.

Inference lr	0.25	0.5	1.0
Perplexity	248.2	**236.1**	1165.5

Table 5.3: Best perplexity for each inference learning rate value: 0.25, 0.5, and 1.0. The aggregation is minimum across the values for the other two hyperparameters: $T \in [15, 30]$, and $lr_{weight} \in [4 \times 10^{-4}, 8 \times 10^{-4}, \ldots, 64 \times 10^{-4}]$. The best value is in bold.

With the hyperparameter values from the first part, I find the best values for T, the weight learning rate and the energy learning rate. The results in Table 5.3 show that the best value of $lr_{inference}$ is 0.5. While I have observed that $lr_{inference}$ is stable across multiple scenarios, I opt to leave T and lr_{weight} to be searched over in subsequent experiments, because their best values can depend on the network and data size.

Conclusion of hyperparameter search

In summary, I have identified the best hyperparameter values for PPC as follows: batch size of 512, PC Layer placement immediately before all activation functions, energy learning rate of 0.5, and no PC Layer output clipping. I fix these hyperparameters in the data scalability and parameter scalability experiments, which saves computational resources.

Data Scalability The data scalability results of PPC and BP in Table 5.4 show that PPC outperforms BP across all training data sizes (100,000, 200,000, 400,000,

Method	train-S	train-M	train-L	train-XL	train-2XL	train-3XL
PPC	**279.7***	**238.8***	**203.7***	**184.5***	**167.2**	**154.9**
BP	360.9	288.0	248.3	206.4	176.2	163.1

Table 5.4: Data scalability of the PPC vs BP training methods: (test) perplexity, lower is better. The dataset size doubles from S to M to ... to 3XL. Numbers are the average of 3 seeds for train-S to train-XL, and 1 seed for train-2XL and train-3XL. The best values are in bold; * denotes the statistically significant best results.

Training method	h=64	h=128	h=256
PPC	**262.2***	**238.8***	**226.4***
BP	329.0	288.0	267.9

Table 5.5: Parameter scalability of the PPC vs BP training methods: (test) perplexity for hidden sizes (64, 128, 256), lower is better. Numbers are the average of 3 seeds. The best values are in bold; * denotes the statistically significant best results.

Training method	1 layer	2 layers	3 layers
PPC	**204.2***	**238.8***	**362.6**
BP	242.2	288.0	371.5

Table 5.6: Parameter scalability of the PPC vs BP training methods: (test) perplexity for the number of layers (1, 2, 3), lower is better. Numbers are the average of 3 seeds. The best values are in bold; * denotes the statistically significant best results.

. . . , 3.2M examples, respectively). While both PPC and BP improve the performance when increasing the data size, the gap narrows as the data size increases. However, the gap remains in favour of PPC even when using 16 times more data (train-3XL) than the original training dataset (train-M). This is a promising sign for using PPC for training LMs for practical applications where large datasets are the norm.

Parameter Scalability I explore the model's parameter scalability in terms of hidden size and in terms of number of layers.

The results in Table 5.5 show that both PPC and BP scale well in terms of hidden size, from 64 to 128 to 256, but that PPC outperforms BP across the hidden sizes. The results in Table 5.6 show that both PPC and BP scale poorly in terms of number of layers, from 1 to 2 to 3. This suggests that the RNN model itself has layer scalability issues, most likely due to the simplified RNN architecture used (see Section 5.3.3).

Overall, PPC and BP scale similarly in terms of the number of model parameters, but PPC outperforms BP across the board for small datasets (200k examples in the train-M dataset).

All comparisons of PPC vs BP that use multiple seeds (3 seeds) are statistically significant with $p < 0.05$, apart from the 3-layer comparison. However, for 3 layers, both PPC and BP perform poorly, so statistical significance is less important.

5.3.6 Conclusion

In this work, I have explored how to implement the PPC (Song, 2021; Salvatori et al., 2022b) predictive coding method for recurrent neural network language models (RNN

LMs), and how it compares against backpropagation (BP). I have concluded that the placement of the predictive coding layers (PC Layers) affects the performance of the RNN LMs. I have also determined some key hyperparameters that work the best for PPC in this setting, and have compared the best configuration of PPC to BP. I have shown that PPC significantly outperforms BP for small data sizes, with a gap remaining even at large data sizes. I have also observed similar scalability between PPC and BP in terms of model size. These results may unlock the application of predictive coding methods as an effective alternative to BP for training LMs.

However, this work is focused on simple RNNs where the PPC and BP methods can be compared on equal footing. Further improvements to the PPC method and PPC-specific network architectural improvements may be needed to use the PPC method for training SOTA large LMs. Future works can also explore the applications of PPC to more complex architectures such as transformers, which I do in my subsequent work in Sections 5.4, 5.5, and 5.6.

5.4 Predictive Coding beyond Gaussian Distributions

This section is based on my contribution to the paper by Pinchetti et al. (2022) published at the NeurIPS 2022 conference, which is section "4.3 Transformer Language Models" in that paper, where I demonstrate that the newly-introduced *KL-modification* can be successfully applied to transformer LMs.

5.4.1 Introduction

Biologically plausible predictive coding methods such as PC (Whittington and Bogacz, 2017) and PPC (Song, 2021) have shown impressive performance for small feed-forward and convolutional NNs in the computer vision domain. However, more complex and recent architectures such as transformers (Vaswani et al., 2017) for language generation are missing in all existing[4] predictive coding works. This is most likely because previous methods such as PC and PPC fail to achieve good performance with transformers, which I confirm in this work. To address this problem, Pinchetti et al. (2022) introduce a modification (the *KL-modification*) of the PC and PPC methods via the "KL-divergence between the expected and actual distributions" in the predictive coding framework.

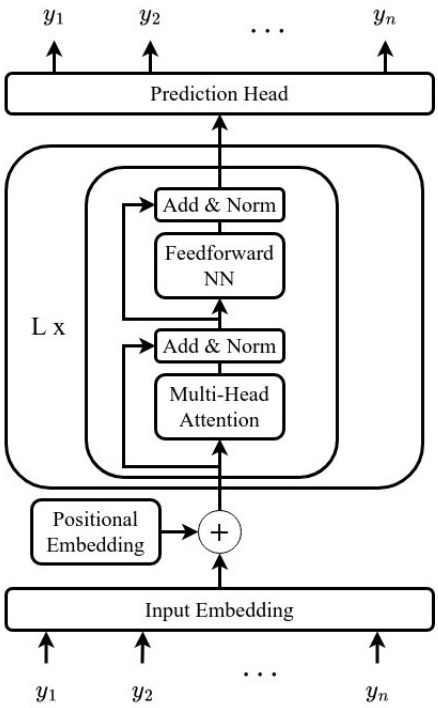
In this work, I conjecture that the softmax functions, both in the self-attention and at the model output (Equations 5.4 and 5.7), are the components of the transformer architecture that interfere with the original PC and PPC methods. I show that the KL-modification when applied to those softmax functions yields substantially better and more stable language generation performance, which is on par with backpropagation (BP) for a small transformer LM architecture. This method can unlock the application of predictive coding to transformer LMs.

5.4.2 Method

In this section, I present the model architecture, the predictive coding training details, and the training and evaluation procedures.

Architecture The LM architecture I use is based on the GPT-1 (Radford et al., 2018) transformer conditional LM. The choice of a conditional LM is because condi-

[4]before the work by Pinchetti et al. (2022) which this is based on



model with L transformer blocks.

Figure 5.4: Diagram of a transformer

tional LMs are trained for next-token prediction which makes them most suitable for natural language generation, which is the focus of this work.

The choice of GPT-1 is because it is a fairly simple architecture so it is easier to work with and to interpret the results, while still using the basic transformer architecture which is the basis of SOTA LMs. SOTA LMs like GPT-4 (OpenAI, 2023) are still largely based on the GPT-1 decoder-only conditional transformer LM architecture, with small, iterative architectural improvements from GPT-1 (Radford et al., 2018) to GPT-2 (Radford et al., 2019), GPT-3 (Brown et al., 2020), and GPT-4 (OpenAI, 2023). Therefore, it is likely that the main conclusions of this work are still valid for SOTA LM architecture variants. However, the predictive coding methods used in this work do not scale well to large complex architectures yet, as evidenced by the Additional Experiments (Section 5.6). Therefore, more advances to the predictive coding methods need to be done before my work and insights can be applied to SOTA large LMs.

The transformer architecture (Figure 5.4) that I use can be summarised mathematically as follows: let the input sequence of vocabulary tokens be $x_1, x_2, \ldots, x_n$. Let the vocabulary size be m, and let the vector X_i be the 1-hot encoding of x_i in the vocabulary space. The model then converts the input matrix $X \coloneqq [X_1, X_2, \ldots, X_n]$ to

token embeddings via a trainable embedding matrix $M_{m\times e}$ where e is the embedding dimension ($e = h$, the hidden size), as follows: $E_{n\times e} \coloneqq X_{n\times m}M_{m\times e}$. This is followed by adding a trainable positional embedding matrix $Pos_{128\times e}$: $E^{(2)}_{n\times e} \coloneqq E_{n\times e} + Pos_{n\times e}$, where 128 is the maximum sequence length supported by the model. Positional embeddings are required because the transformer shares all other parameters across the input dimension n. Layer normalisation (Ba et al., 2016) is applied to the resulting embeddings: $E^{(3)}_{n\times e} \coloneqq LayerNorm(E^{(2)}_{n\times e})$. Layer normalisation stabilises the training by normalising each neuron in a given layer by the values of all neurons in that layer.

After the embedding, the transformer architecture has a number of *transformer blocks*. Each block consists of a self-attention module, followed by a feed-forward neural network (FNN) module. The self-attention module applies self-attention across the input sequence dimension. Each input (query) attends over the input sequence via keys and obtains as output a weighted average of the values of those keys w.r.t. their attention weights. To do this, first, the input $I_{n\times h}$ is transformed into queries, keys, and values, via trainable matrices $Q_{h\times h}$, $K_{h\times h}$, and $V_{h\times h}$, respectively. The attention scores are obtained by multiplying together the queries and keys, and dividing by a normalising constant: $AttnScores_{n\times n} \coloneqq I_{n\times h}Q_{h\times h}(I_{n\times h}K_{h\times h})^T/\sqrt{h}$. To enforce the conditional LM, an attention mask $M_{n\times n}$ is added to $AttnScores_{n\times n}$, defined as: $M_{i,j} = 0$ if $j \leq i$ and $M_{i,j} = -\infty$ if $j > i$. The attention scores are transformed to attention weights via a softmax function applied across the second dimension:

$$AttnWeights_{n\times n} \coloneqq Softmax_{dim=2}(AttnScores_{n\times n} + M_{n\times n}) \tag{5.4}$$

The softmax causes the "$-\infty$" values of the attention mask to yield zero attention weights, thus preventing tokens from attending to future positions in the sequence (for conditional language modelling). The attention output is then the weighted average of the values matrix $V^{(2)}_{n\times h} \coloneqq I_{n\times h}V_{h\times h}$ w.r.t. the attention weights: $AttnOut_{n\times h} \coloneqq AttnWeights_{n\times n}V^{(2)}_{n\times h}$. Multi-head attention works similarly, but the attention hidden dimension h is split into several *heads* of dimension h' that independently apply self-attention, and the final results $AttnOut_{n\times h'}$ get summed together. Finally, a linear transformation is applied to the attention output:
$AttnOut^{(2)}_{n\times h} \coloneqq Linear_{h\times h}(AttnOut_{n\times h})$, followed by skip-connection, and layer normalisation: $AttnOut^{(3)}_{n\times h} \coloneqq LayerNorm(AttnOut^{(2)}_{n\times h} + I_{n\times h})$. The skip-connection and layer normalisation serve to speed up and stabilise the training.

The FNN module following the self-attention module consists of a linear transformation followed by Tanh activation and another linear layer:

$$FNNOut_{n\times h} \coloneqq Linear_{h\times h}(Tanh(Linear_{h\times h}(AttnOut^{(3)}_{n\times h}))) \tag{5.5}$$

The output is then obtained by applying a skip-connection and layer normalisation:

$$FNNOut^{(2)}_{n\times h} \coloneqq LayerNorm(FNNOut_{n\times h} + AttnOut^{(3)}_{n\times h})$$

The transformer blocks are followed by a LM *prediction head*. It consists of a single-layer neural network with Tanh activation:

$$H_{n\times h} \coloneqq Tanh(Linear_{h\times h}(FNNOut^{(2)}_{n\times h})) \tag{5.6}$$

followed by layer normalisation, projection to the vocabulary size dimension (v), and softmax over the vocabulary dimension to obtain probabilities over the vocabulary:

$$HOut_{n\times v} \coloneqq Softmax_{dim=2}(Linear_{h\times v}(Layernorm(H_{n\times h}))) \tag{5.7}$$

$HOut_{n\times v}$ is the final output of the transformer model, which is trained to predict the target tokens expressed as 1-hot vocabulary vectors.

Predictive Coding Training Having a softmax activation function is problematic for predictive coding, because the usual Gaussian assumption (Friston, 2005) does not hold because softmax introduces neuron inter-dependence which breaks the usual independence assumption. The KL-modification by Pinchetti et al. (2022) aims to solve the problem of having softmax activation functions and does this by only modifying the energy function for softmax activation, as described in Background Chapter 5.2. In this work, I compare using the KL-modification versus not using it, and I consider two predictive coding methods to apply it to: PC (Whittington and Bogacz, 2017) and PPC (Song, 2021; Salvatori et al., 2022b). PC works by multiple inference updates per batch, followed by a single weight update, whereas PPC does multiple updates per batch: one per inference step. For more details, see Section 5.2.

Training with predictive coding requires strategic placement of PC Layers (see Section 5.2) inside the LM architecture. These layers are only used during training and are disabled during prediction, where the original LM architecture is used. Initial experiments suggest that PC Layer placement does not significantly affect performance, whether it is before or after activation.

In this work, the PC Layers are located after each activation function – tanh or softmax, including the softmax at the network output, by modifying equations 5.4, 5.5, 5.6, and 5.7 appropriately. The only exception is for the KL-modification, where the energy function takes into consideration the values of x and μ before softmax, but still applies softmax to them. See Background Section 5.2 for more details.

Training and Evaluation The training objective for the conditional LM is next-token prediction. The input sequence is formed by tokenising a piece of text, cutting it to fit the context length, and adding a start-of-sentence token (sos), and an end-of-sentence token (eos) around it: $Input = [x_1 = <\text{sos}>, x_2 = w_1, x_3 = w_2, \dots x_{n-1} = w_{n-2}, x_n = <\text{eos}>]$. The target sequence is given by the input sequence shifted one position to the left, so that each token in the input is at the same position as the following sentence token is in the output: $Target = [x_1 = w_1, x_2 = w_2, \dots x_{n-1} = w_{n-1}, x_n = <\text{eos}>]$. The attention mask (Equation 5.4) together with this input and target format ensures that the training objective is conditional language modelling (i.e. predictions are only based on past tokens).

The training data is generated from the commonly-used One Billion Word Benchmark (Chelba et al., 2013), where each data point is a sentence. The training dataset of size $200,000$ and the dev datasets of size $10,000$ are disjoint randomly sampled subsets of the original training dataset. The test dataset is the original test dataset of the 1B Word Benchmark.

The vocabulary of size 8001 is automatically generated from a portion of the training text data via byte-pair-encoding with the SentencePiece tokenizer (Kudo and Richardson, 2018). The resulting tokens are sub-words, but most of them are multi-character. This allows for a balance between the semantic expressiveness of multi-character tokens, and the computational feasibility given by the limited vocabulary size.

The context size of the model is chosen to be 32 tokens, which equates to an input size of 34 including the <sos> and <eos> tokens. This allows for full sentences to be encoded in the input while limiting the model's computational requirements.

The training procedure does not involve early stopping because early stopping works poorly when the training curve is irregular. This is a particular problem for some predictive coding models, e.g. Figure 5.5. Instead, I run the full training while evaluating the model at regular intervals and saving the best model for test evaluation. The model's performance is measured as per-token perplexity (denoted *ppl*) as described in Section 2.1.2.

Both for BP and for $PC_{\mathcal{F}}$, I update the model weights via the AdamW (Loshchilov and Hutter, 2019) optimiser with the default (0.01) weight decay. For $PC_{\mathcal{F}}$, inference (x-update) (see Section 5.2) is done via stochastic gradient descent (SGD).

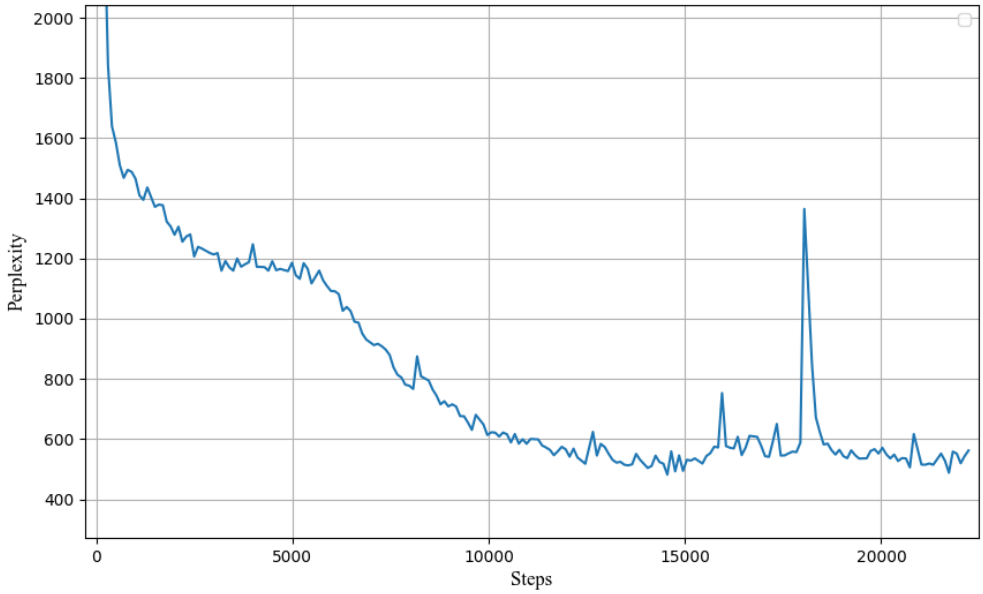


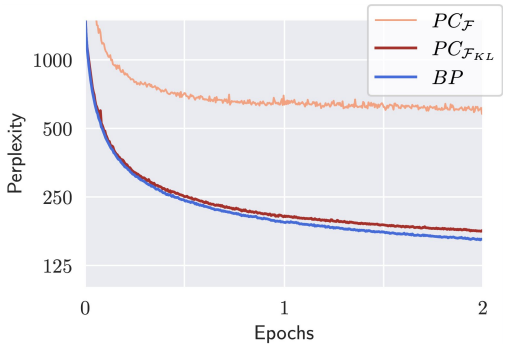
Figure 5.5: Irregular training curve. Model: transformer masked LM, 2-transformer-block, 1-head, trained via PC with the KL-modification.

5.4.3 Experiments and Results

For each model, I compare the three training methods: BP, $PC_{\mathcal{F}}$, and $PC_{\mathcal{F}_{KL}}$ for training a transformer LM. I run a hyperparameter search for each training method, select the best models, compare their training curves and their test performance, and show qualitative examples of model predictions.

Experimental Setup In the final results, I report three training methods: BP, $PC_{\mathcal{F}}$ (PC without KL), and $PC_{\mathcal{F}_{KL}}$ (PPC with KL). This choice is informed by initial experiments that suggest that PC outperforms PPC for the model in the non-KL case (582.2 vs 1086 ppl), whereas, PPC outperforms PC for the KL case (177.1 vs 208.6 ppl).

The three training methods are compared for training the transformer LM described in Section 5.4.2, with model dimensions: one transformer block, one head, and a hidden size of 128. All models are trained on two epochs with a batch size of 8. The hyperparameters being tuned are as follows: the weight learning rate lr_{weight} for BP, $PC_{\mathcal{F}}$ and $PC_{\mathcal{F}_{KL}}$, the inference learning rate $lr_{inference}$ for $PC_{\mathcal{F}}$ and $PC_{\mathcal{F}_{KL}}$, and the number of update steps per batch T for $PC_{\mathcal{F}}$ and $PC_{\mathcal{F}_{KL}}$. The best hyperparameters are selected via grid search over the ranges listed in Appendix E. The ranges



Training Method	Test Perplexity
BP	162.64 ± 0.76
$PC_{\mathcal{F}_{KL}}$	175.90 ± 1.74
$PC_{\mathcal{F}}$	590.08 ± 12.60

Figure 5.6: Left: Comparison of LMs trained with BP, $PC_{\mathcal{F}}$ and $PC_{\mathcal{F}_{KL}}$, as shown by dev perplexity. Right: Test perplexity achieved by the various training methods for transformer LMs. Average ($\pm\sigma$) of 10 seeds.

are selected so that the best hyperparameter combination is always in the interior of the ranges. After selecting the best hyperparameters, each model is trained with 9 additional seeds. Additional training details are provided in Appendix E.

Quantitative Results Figure 5.6 shows that $PC_{\mathcal{F}_{KL}}$ significantly outperforms $PC_{\mathcal{F}}$ in terms of test perplexity (175.9 vs 590.1), while having a more stable training curve (Figure 5.6 (Left)). I believe that this large gap in performance is because the *softmax* is not only in the output layer, but also in the attention mechanism of the transformer model. Furthermore, $PC_{\mathcal{F}_{KL}}$ is more stable than $PC_{\mathcal{F}}$ across seeds, with 1.74 vs 12.6 standard deviation. The performance of $PC_{\mathcal{F}_{KL}}$ is close to that of BP (175.9 vs 162.4) and both training curves look stable.

Qualitative Results Table 5.7 shows example sentence completions given by BP, $PC_{\mathcal{F}}$, and $PC_{\mathcal{F}_{KL}}$ along with the probabilities assigned to each prediction. The sentences were selected subjectively from the test dataset based on how interesting they are, and cut right before a subjectively interesting word to be predicted.

Table 5.7 shows that the LMs trained by BP and $PC_{\mathcal{F}_{KL}}$ do not differ significantly on the test set. In some cases, the predictions given by $PC_{\mathcal{F}_{KL}}$ are closer to the ground truth, e.g., for "Yet the bank and its executives are still ready to support specific Democratic [candidates]", $PC_{\mathcal{F}_{KL}}$ predicts "leaders" and "candidates" as top-2 choices. All models show failure in commonsense reasoning, e.g., for "I've been dreaming about this since I was a [child]" they fail to assign "child" with $>$ 1% probability, which shows the limitations of small LMs.

Input sentence	BP		$PC_{\mathcal{F}}$		$PC_{\mathcal{F}_{KL}}$	
Yet the bank and its executives are still ready to support specific Democratic [candidates]	leaders Party candidates	(7.5) (7.2) (3.8)	. , and	(1.0) (1.0) (0.6)	leaders candidates presidential	(12.1) (7.3) (4.8)
GMAC started out offering car [loans]	and sales ,	(2.2) (1.7) (1.7)	, and in	(5.3) (3.1) (2.5)	sales products services	(4.4) (2.3) (2.0)
I've been dreaming about this since I was a [child]	great " good	(1.6) (1.5) (1.2)	lot good very	(1.1) (1.1) (0.9)	" good year	(1.9) (1.2) (1.1)
Here is a breakdown of the seven taxes and fees that have been [collected]	a to in	(2.3) (2.0) (1.9)	a the in	(4.7) (2.3) (1.3)	a to the	(4.3) (2.6) (2.4)
Under the plan, Iceland will reimburse the [money]	first world same	(1.9) (1.0) (0.8)	best first most	(1.7) (1.6) (0.7)	first same world	(1.9) (1.2) (1.2)
Aniston and Pitt were still married when Pitt and Jolie made the 2005 [film]	, . World	(2.3) (1.9) (1.6)	, and .	(23.0) (5.6) (5.1)	. , and	(10.2) (10.0) (3.9)

Table 5.7: Top predictions of each model for completing a number of sentences. The ground-truth completion is given in [brackets]; the model prediction format is: <word> (<probability %>).

5.4.4 Conclusion

In this work, I implemented the KL predictive coding modification by (Pinchetti et al., 2022) to transformer LMs. I showed that using the KL-modification improves performance for predictive coding methods (PC and PPC (Whittington and Bogacz, 2017; Song, 2021)), and leads to greater training and seed-wise stability. Furthermore, I showed that predictive coding with the KL-modification performs similarly to BP for training small LMs. Future work can explore the scalability of this approach to larger LMs and more data (done in Section 5.6), as well as develop further improvements to the method.

5.5 A Stable, Fast, and Fully Automatic Learning Algorithm for Predictive Coding Networks

This section is based on my contribution to the work by Salvatori et al. (2024) accepted at the ICLR 2024 conference, namely, section 5 "Language Model Experiments". Salvatori et al. (2024) introduce the incremental predictive coding (iPC) method, which I refer to in this thesis as parallel predictive coding (PPC). My contribution is to apply

PPC to both conditional and masked LMs and to compare it against the PC (Whittington and Bogacz, 2017) predictive coding method and backpropagation (BP).

5.5.1 Introduction

The predictive coding theory of the brain offers promising directions for developing biologically plausible machine learning methods. One such method is Parallel Predictive Coding (PPC) – first proposed by Song (2021) as an improvement to the PC (Whittington and Bogacz, 2017) predictive coding method that is more efficient, biologically plausible and performs better on computer vision tasks (Salvatori et al., 2022b). In Section 5.4 (based on my work in (Pinchetti et al., 2022)) I showed how to apply PPC to transformers. However, I did not directly compare PPC to PC on transformer LMs. Furthermore, in my previous work (Pinchetti et al., 2022) I only consider single-block transformers, and only of the conditional LM type, which limits the scope of my conclusions.

In this work, I compare PPC and PC on both masked and conditional transformer LMs. Furthermore, I consider two-block rather than one-block models, which is a more practical transformer configuration. This is because using multiple transformer blocks (2-3) usually improves performance greatly while fitting within modest compute budgets (Kaplan et al., 2020). I show that PPC vastly outperforms PC both on masked and conditional LMs, and performs similarly to BP.

5.5.2 Method

Model A recent work has shown that it is possible to introduce a small modification to the training algorithm of PC to improve its performance on small LMs (Pinchetti et al., 2022), which I use in this work. Here, I test the performance of PC, PPC, and BP on models based on the GPT-1 (Radford et al., 2018) and BERT (Devlin et al., 2019) decoder-only transformer LM architectures. The GPT-1-based model is a conditional LM trained on next-token-prediction, following Radford et al. (2018), while the BERT-based model is a masked LM trained to reconstruct randomly masked tokens from the input, following Devlin et al. (2019). The GPT-1-based model is identical to the conditional LM architecture used by Pinchetti et al. (2022) and in the Additional Experiments (Section 5.6), whereas the BERT-based model is equivalent to removing GPT-1's triangular attention mask (Equation 5.4 in Section 5.4.2) so that the model can use bidirectional context as in BERT (Devlin et al., 2019).

Training Details The training and dev datasets are generated by randomly sampling, respectively, $200,000$ and $10,000$ instances from the One Billion Word Benchmark (Chelba et al., 2013). The test dataset is the original test dataset of the 1B Word Benchmark. For both models, I use two transformer blocks with one head and a hidden size of 128. The input length of the model is restricted to 32 tokens. The vocabulary is obtained via byte-pair-encoding with 8001 tokens, generated via the SentencePiece tokenizer (Kudo and Richardson, 2018).

For PC and PPC, I use the $\mathcal{F}_{KL}$ modification (Pinchetti et al., 2022), which replaces the Gaussian distribution for the layers with *softmax* activation with a non-Gaussian distribution. Initial experiments confirm that this helps both PC and PPC for both LMs. Initial experiments with the masked LMs also suggest that PC and PPC benefit from disabling the layer normalization (Ba et al., 2016) found in BERT, but BP does not. Therefore, for the masked LM, I assume layer normalization for BP only. For the conditional LMs, I observe that PPC and BP benefit from layer normalization, but PC does not. I further explore layer normalisation for PPC in Section 5.6.2.

I train each model on 16 epochs of the training dataset. I do early stopping every $100 * 128/batch\_size$ batches. For PPC, and PC, I have observed that a batch size of 128 works best, so I assume it throughout the experiments.

Based on early experiments, I assume the parameter (weight) optimizer to be AdamW (Loshchilov and Hutter, 2019), and the inference optimizer to be Stochastic Gradient Descent (SGD). I denote the parameter (weight) learning rate as lr, and the inference learning rate as $lr_{inference}$.

For PC, I also use an inference learning rate discount (between 0 and 1), which I use to multiply the learning rate by if the energy has not improved. I have observed that this can increase the performance and decrease the optimal T (the number of updates per batch), hence speeding up the training. I use 0.9 for the discount value, which I have found works best. For PPC, I assume the $lr_{inference}$ value to be 0.5, as suggested by my initial experiments. I do grid searches over the hyperparameter ranges listed in Appendix F.

5.5.3 Experiments and Results

Experimental Setup I compare the BP, PC and PPC training methods on both masked and conditional LMs. Here, PC and PPC are shorthand notations for the methods I refer to PC_{KL} and PPC_{KL} in Sections 5.4 and 5.6 because I use the KL-modification by Pinchetti et al. (2022), which they showed to improve results.

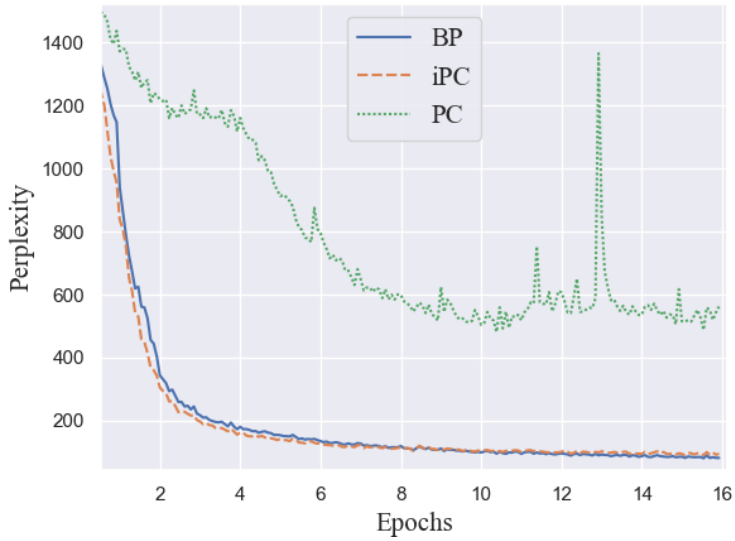


Figure 5.7: Dev perplexity during training of the best performing masked LMs. All three training methods are displayed: BP, PC and PPC (iPC).

For each model, I select the best hyperparameters via grid search, after which I run each model on 9 additional seeds for a total of 10 seeds. This allows us to compare the expected perplexity (ppl) performance and see the performance variation across seeds. I also use a convergence threshold to discard those models that do not converge. For PPC and BP, I define the convergence threshold at 200 test perplexity, and for PC I define it at 800. For complete per-seed results, see Appendix F.

Model	BP		PC		PPC	
	Perplexity	#Conv	Perplexity	#Conv	Perplexity	#Conv
MLM	120.02 ± 13.19	7	523.08 ± 12.3	3	**106.19** ± 10.54	**10**
CLM	**113.32** ± 0.36	**10**	206.34 ± 6.46	**10**	142.54 ± 4.23	7

Table 5.8: Perplexity of the three compared methods on 10 random seeds, for both the masked (MLM) and conditional (CLM) language models, with the number of seeds that each model converged with.

Results The results in Table 5.8 show that PPC significantly outperforms PC in both masked (MLM) and conditional (CLM) language models. For masked LMs, PPC also exhibits a much better convergence, with all 10 seeds converging, whereas PC has only 3 converging seeds. The poor performance of PC is due to its poor training stability, as evident by Figure 5.7, where we can also see that the training

curves of PPC and BP are similar. In fact, PPC performs similarly to BP in terms of test perplexity, with PPC performing better than BP on masked LMs with 106 vs. 120 ppl (where all 10 runs converged), and worse on conditional LMs with 113 vs. 143 ppl (where 3 runs have not converged). However, for masked LMs, BP exhibits worse convergence than PPC, with only 7 out of 10 seeds converging, and 3 seeds having perplexity of 350 or more. This means that the best hyperparameters for PPC are more stable across seeds than those for BP. On the other hand, if I focus on the original seed with a full hyperparameter search for both models I observe that BP outperforms PPC with 83.6 vs 94 test perplexity. This can be seen in Figure 5.7, where BP outperforms PPC in the second half of the training.

Further to its poor performance, the PC method is more computationally intensive than PPC because of two main reasons. First, for PC, there are multiple (T) inference steps needed for each weight update step, which adds a significant amount of computation when compared to PPC (with 1 inference step per weight update). The second reason is that PC requires a more extensive hyperparameter search because the best $lr_{inference}$ value is not clear (e.g. I got 0.03125 for MLM and 0.0078125 for CLM) as it is unclear how many inference steps are needed before the weight update (e.g. I got 8 for MLM and 10 for CLM). In contrast, PPC uses 1-step inference in conjunction with each weight update, and, in practice, the best $lr_{inference}$ value for PPC_{KL} seems to be 0.5, as confirmed in Section 5.6 Additional Experiments.

5.5.4 Conclusion

The experiments performed on masked and conditional transformer LMs show that PPC is significantly better than PC in terms of both performance and stability, obtaining results that are overall similar to those of BP. PPC is also a more computationally efficient method than PC by requiring only one inference step per weight update step and can benefit from a faster hyperparameter search. In summary, I have found that the PPC method is better-performing, more stable and faster than PC for training transformer-based LMs.

5.6 Additional Experiments

This section is a follow-up to Chapters 5.4 and 5.5. In Chapter 5.4 (and (Pinchetti et al., 2022)), I show that using the KL-divergence modification for parallel predictive coding (PPC_{KL}) outperforms the standard parallel predictive coding method (PPC) for transformer LMs. Furthermore, in Chapter 5.5, I show that the PPC (PPC_{KL})

method significantly outperforms PC (PC_{KL}) for both conditional and masked LMs. In this section, I present additional experiments that aim to determine whether this promising training method PPC_{KL} can outperform backpropagation (BP) for training transformer LMs across two architectures, with new hyperparameter choices, and whether it can scale well. More specifically, I also consider a masked transformer LM architecture (BERT) (Devlin et al., 2019) in addition to the GPT-1 (Radford et al., 2018) conditional transformer LM architecture used by Pinchetti et al. (2022). Furthermore, I investigate the effects of previously untested hyperparameters such as batch size and layer normalisation, and scalability to larger models and longer training. The scalability to larger models is both in terms of model depth via more transformer blocks, and model width via larger hidden size and more transformer self-attention heads.

These experiments provide a more complete understanding of the performance of the PPC_{KL} method when compared to BP. The results show the benefits of larger batch sizes, the importance of choosing whether or not to use layer normalisation, and how PPC_{KL} scales versus BP to larger transformer LMs and with more training.

5.6.1 Method

The method used in this section is parallel predictive coding with the KL-divergence modification, as detailed in Section 5.4.2. Unless otherwise specified, all implementation details match those described in Section 5.4.2.

In this section, I use both masked and conditional LM variants. The conditional LM (CLM) is based on the GPT-1 (Radford et al., 2018) transformer architecture (detailed in Section 5.4.2 and depicted in Figure 5.4), whereas the masked LM (MLM) is based on BERT, and is obtained by removing the conditional attention mask from the CLM. This means that Equation 5.4:

$$AttnWeights_{n\times n} \coloneqq Softmax_{dim=2}(AttnScores_{n\times n} + M_{n\times n})$$

simplifies as:

$$AttnWeights_{n\times n} \coloneqq Softmax_{dim=2}(AttnScores_{n\times n}) \tag{5.8}$$

The two models differ in their training objectives – the training objective of the CLM is next-token-prediction, whereas the MLM is based on the masking training objective of BERT. The masking training objective is given by reconstructing randomly masked tokens in a sentence. The particular procedure follows BERT (Devlin et al., 2019),

where 15% of the sentence tokens are randomly selected, and then replaced randomly as follows: 80% chance of <mask>, 10% chance of random token, and 10% chance of being unchanged. The target sequence is the same length as the input sequence, but it replaces all non-<mask> tokens with <mask>, and all <mask> tokens with the original tokens that were masked. Hence, predicting the target is equivalent to reconstructing the masked tokens in the input. Finally, the input and target sequences are prepended by an <sos> token and appended by an <eos> token.

The model's training data, input size (34), vocabulary and optimiser (AdamW) are the same as in Section 5.4.2. The loss function L is the cross-entropy loss between the output (as described in Section 5.4.2) and the target which depends on whether the LM is masked or conditional (see Section 2.1.2).

Layer normalisation is used by default, as described in Section 5.4.2, but it is unclear if it should be used when training via PPC. I will explore this in the hyperparameter experiments in Section 5.6.2 below.

5.6.2 Experiments and Results

There are several limitations of my work on conditional LMs (Pinchetti et al., 2022) which these experiments aim to address. I classify these limitations into three categories: 1) extended hyperparameter search, 2) other LM variants, and 3) scalability of the method.

The need for extended hyperparameter search comes from the assumption of the batch size being 8 (Pinchetti et al., 2022). In this section, I will explore the effect of larger batch sizes on the performance of the models. Another consideration is whether the default inclusion of layer normalisation (Ba et al., 2016) should be kept for predictive coding. Layer normalisation could be problematic because it introduces inter-dependencies across the input similar to the softmax function, which Pinchetti et al. (2022) have shown to need special modifications for predictive coding.

All hyperparameter search is done both for the conditional LM used by Pinchetti et al. (2022), and for the corresponding masked LM (Devlin et al., 2019). This allows for comparison between the two models in the context of predictive coding.

Finally, exploring the scalability is necessary because Pinchetti et al. (2022) use a transformer model with only one transformer block and one head, trained on only two epochs. Here I explore larger models with longer training times to see how the predictive coding method would scale versus BP.

Experimental Setup Here I give additional details about the experiments and models being used.

For the expanded hyperparameter search experiments, I search over the weight learning rate and the inference learning rate (as in Section 5.4), but also the batch size and the inclusion of layer normalisation (yes or no). Here I assume two transformer blocks and five training epochs unless specified otherwise. This is an increase from one transformer block and two training epochs in my previous work (Pinchetti et al., 2022) (see Section 5.4) because it also serves to inform the subsequent scalability experiments.

Following the expanded hyperparameter search and commentary, I run scalability experiments by assuming the best batch size and layer normalisation from the previous experiments, and by searching over the other hyperparameters. The only exception to this is the experiments with longer training times which is many times more computationally demanding. For that experiment, I assume the best hyperparameters from the experiments with fewer epochs. For all scalability experiments, I assume two transformer blocks, one attention head, a hidden size of 128 and five training epochs unless specified otherwise. I run 3 seeds for each parameter scalability experiment, to account for per-seed variations.

Initial results suggest that the transformer models by (Pinchetti et al., 2022) are under-trained when training on 5 instead of 2 epochs for 1-transformer-block conditional LM trained with PPC_{KL} vs BP, with the original batch size of 8. There is a significant improvement in performance from 165.1 ppl to 138 ppl, which highlights the need to use more training epochs. Five training epochs are still within my computational budget, so I am going to use 5 training epochs as default in all experiments. Similarly, I choose to use two instead of one transformer block by default, which also fits within the computational budget. This has the benefit of producing results that are more relevant for real-world scenarios where larger models are commonly used.

All hyperparameter search is done via grid search with wide enough hyperparameter ranges.

For PPC_{KL}, the hyperparameter search is over four hyperparameters: weight learning rate ($lr_{weights}$), inference learning rate ($lr_{inference}$), number of inference steps (T), and batch size. Unless otherwise specified, I search over the following values for $lr_{inference}$: $\{0.25, 0.5, 1.0\}$. The hyperparameter ranges for $lr_{weights}$ and batch size depend on the experiment, but are always a geometric progression with a ratio of 2. The hyperparameter range for T depends on the experiment but is always a range of consecutive positive numbers.

For BP, the hyperparameter search is over two hyperparameters: the learning rate (lr) and the batch size. For each experiment and variable, the range of values is always a geometric progression with a ratio of 2.

Hyperparameter Search Results These experiments explore the effects of larger batch sizes and the disabling of layer normalisation in transformers trained with PPC_{KL}. The default model depth is two transformer blocks, but I additionally do a wide hyperparameter search across batch size values for models with one transformer block. This is followed by a more precise hyperparameter search for models with two transformer blocks.

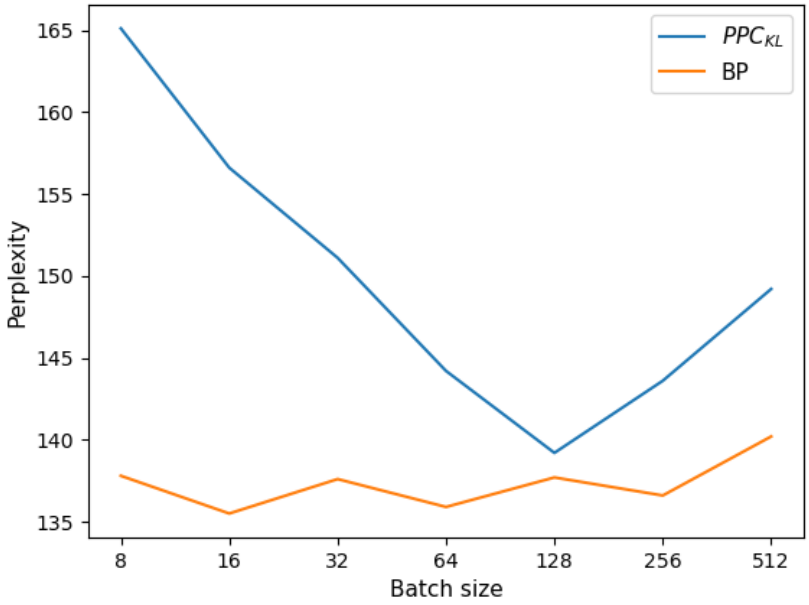


Figure 5.8: Results across batch sizes for 1-transformer-block conditional LM trained with PPC_{KL} (with layernorm) vs BP (with layernorm) on 5 epochs.

Larger batch sizes for conditional LMs: This experiment explores a wide range of batch sizes between 8 and 512, for training 1-transformer-block conditional LMs on 5 epochs with PPC_{KL} vs BP. Figure 5.8 illustrates the need for larger batch sizes than the batch size of 8 used by Pinchetti et al. (2022). The best results show similar performance between PPC_{KL} and BP, with 139.2 vs 135.5 ppl, but only when the batch size is 128 for PPC_{KL}. Comparing the curves across the batch sizes, the BP curve is flatter, whereas the PPC_{KL} achieves its significantly best results with a batch size of 128. This highlights the need for PPC_{KL} to do a hyperparameter search for the batch size around 128, which I explore in the layer normalisation experiments.

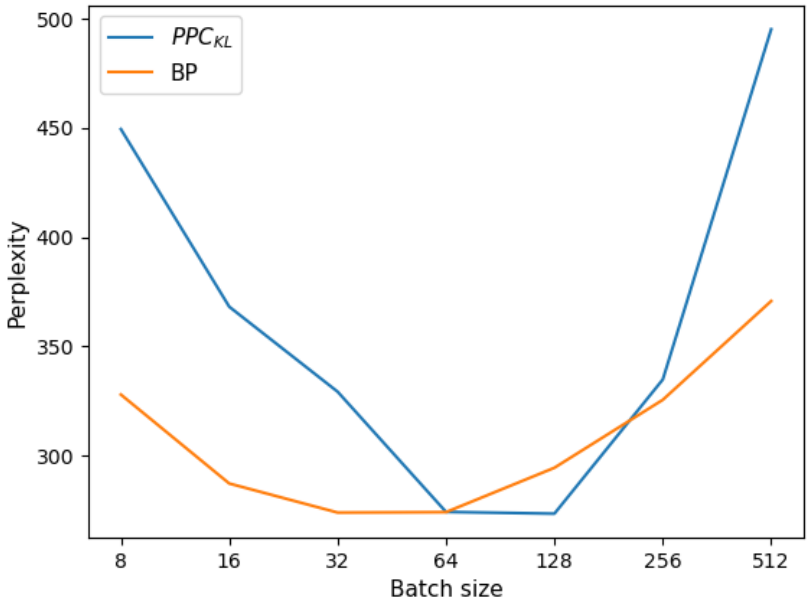


Figure 5.9: Results across batch sizes for 1-transformer-block masked LM trained with PPC_{KL} (with layernorm) vs BP (with layernorm) on 5 epochs.

Larger batch sizes for masked LMs: This experiment explores a wide range of batch sizes between 8 and 512, for training 1-transformer-block masked LMs on 5 epochs with PPC_{KL} vs BP. The results in Figure 5.9 show that larger (than 8) batch sizes shrink the difference between PPC_{KL} and BP. In the best batch size values, PPC_{KL} and BP perform similarly, with 273.5 vs 274.0 ppl. Here, the best batch sizes for PPC_{KL} are 64 and 128, while other batch sizes produce much worse results. In contrast, BP appears more stable across the batch size choice, with only small differences between batch size values of 16, 32, 64 and 128. I will investigate below to confirm if 128 is the best value for PPC_{KL} in the layer normalisation experiments below.

LayerNorm investigation for conditional LMs: This experiment explores LayerNorm vs no-LayerNorm for conditional LM (2-transformer-block), trained with batch sizes of 64, 128, and 256, on 5 epochs. The results in Table 5.9 show that layer normalisation helps for both PPC_{KL} and BP, with 153.8 vs 179.6 ppl and 131.9 vs 142.4 ppl, respectively. The results on BP are expected because LayerNorm was developed to improve performance for BP (Ba et al., 2016). Here, the best results for PPC_{KL} are obtained with a batch size of 128, which matches the best batch size value from the "larger batch sizes" experiment.

	LayerNorm	**No LayerNorm**
PPC_{KL}	**153.8**	179.6
BP	**131.9**	142.4

Table 5.9: Model: 2-transformer-block conditional LM, trained with 64, 128, and 256 batch sizes, on 5 epochs. Comparison between LayerNorm and No LayerNorm for PPC_{KL} and BP in terms of perplexity (lower is better). Note: BP uses a full hyperparameter search with more batch sizes.

LayerNorm investigation for masked LMs: This experiment explores LayerNorm vs no-LayerNorm for masked LM (2-transformer-block), trained with batch size = 64, 128 and 256, on 5 epochs. The results in Table 5.10 show that for PPC_{KL}, no LayerNorm is better than using LayerNorm, with 119.4 vs 278.5 ppl. In contrast, for BP, using LayerNorm is better than not using it, with 145.6 vs 251.7 ppl. This is expected because LayerNorm was developed to improve performance for BP (Ba et al., 2016). Surprisingly, the best PPC_{KL} model outperforms the best BP model with 119.4 vs 145.6 ppl. I explore whether these PPC_{KL} results scale with larger models and additional training time in the "Scalability Results" paragraphs below. Similarly to the conditional LM experiment, the best results for PPC_{KL} are obtained with a batch size of 128.

	LayerNorm	**No LayerNorm**
PPC_{KL}	278.5	**119.4**
BP	**145.6**	251.7

Table 5.10: Model: 2-transformer-block masked LM, trained with 64, 128 and 256 batch sizes, on 5 epochs. Comparison between LayerNorm and No LayerNorm for PPC_{KL} and BP in terms of perplexity (lower is better). Note: BP uses a full hyperparameter search with more batch sizes.

The best batch size value for PPC_{KL} is 128, both for masked and conditional LMs, regardless of whether one uses LayerNorm or no-LayerNorm. This repeats the conclusion of the batch sizes experiments, but also extends it to two transformer blocks instead of only one. Therefore, in subsequent experiments, I can and will assume a batch size of 128 for PPC_{KL}. This partially compensates for the computational increase in PPC_{KL} versus BP introduced by the additional hyperparameters of inference learning rate ($lr_{inference}$) and number of inference steps (T).

Furthermore, the hyperparameter search experiments also confirm that the best value for $lr_{inference}$ is 0.5. This is because all experiments have considered the values 0.25, 0.5 and 1.0, and the best performance of PPC_{KL} is achieved for 0.5 for all of

them. The subsequent experiments still explore all three values of $lr_{inference}$, but they also confirm that 0.5 is the best one. Therefore, $lr_{inference} = 0.5$ can be assumed in situations where computational resources are limited.

Conclusion of experiments In summary, I have established the best values for the training batch size (128) and whether to use LayerNorm when training conditional and masked LMs with PPC_{KL}. LayerNorm improves the performance for conditional LMs, but not for masked LMs, which informs the LayerNorm choice for the scalability experiments that follow. Additionally, the best value for the inference learning rate ($lr_{inference}$) in these experiments is 0.5, which can help in experiments with longer training times. In terms of performance, PPC_{KL} and BP are similar, with BP being better for conditional LMs, whereas PPC_{KL} is better for masked LMs. The performance is further examined in the scalability experiments below.

Scalability Results: Larger Models Here, I present the results of the experiments with multiple transformer blocks, with increased hidden size, and with multiple attention heads, for both masked and conditional LMs. These experiments aim to test how the PPC_{KL} method scales against BP for larger transformer models. I assume the best batch size and layer normalisation choice from the hyperparameter search, to save computational resources.

Multi-transformer-block experiments for conditional LMs: The results in Table 5.11 show that BP outperforms PPC_{KL} across all model depths (1, 2, and 3 transformer blocks). Furthermore, the performance of PPC_{KL} does not scale well with multiple transformer blocks as it decreases from 140.3 to 153.8 to 167.4 ppl. In contrast, BP improves in performance from 135.6 to 131.7 to 131.4 ppl, respectively. This means that additional modifications to the PPC_{KL} training method may be needed to improve the scaling of conditional LM performance with model depth.

	1-block	**2-block**	**3-block**
PPC_{KL}	140.3	153.8	167.4
BP	**135.6**	**131.7**	**131.4**

Table 5.11: Multi-transformer-block experiment for conditional LMs. Model: one-head conditional LM, with 128 hidden size, trained on five epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds.

Multi-transformer-block experiments for masked LMs: The results in Table 5.12 show that PPC_{KL} and BP are similar in performance for masked LMs with 1, 2 and

3 transformer blocks, and both scale well in terms of depth (number of transformer blocks).

	1-block	**2-block**	**3-block**
PPC_{KL}	286.6	150.4	140.7
BP	**277.1**	**144.7**	**138.5**

Table 5.12: Multi-transformer-block experiment for masked LMs. Model: one-head masked LM, with 128 hidden size, trained on five epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds.

Increased hidden size for conditional LMs: The results in Table 5.13 show that both PPC_{KL} and BP improve with increasing the hidden size (h=128 to h=256) for conditional LMs, from 153.8 to 143.4 ppl, and from 131.7 to 120.0 ppl. Unfortunately, PPC_{KL} is significantly worse than BP for all considered hidden sizes, and the gap in performance remains when increasing the hidden size.

	h=128	**h=256**
PPC_{KL}	153.8	143.4
BP	**131.7**	**120.0**

Table 5.13: Experiment: increased hidden size (128 and 256) for conditional LM. Model: two-block, one-head conditional LM, trained on five epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds.

Increased hidden size for masked LMs: The results in Table 5.14 show that PPC_{KL} outperforms BP across hidden sizes (h=128 and h=256) for masked LMs. Unfortunately, both PPC_{KL} and BP struggle to make significant improvements in performance from 128 to 256 hidden size. The performance of PPC_{KL} and BP degrades with increasing the hidden size, with 154 vs 156.3 ppl, and 144.7 vs 155.5 ppl, respectively. This could mean that for the transformer only some combinations of hidden size, number of heads and number of transformer blocks are optimal for scaling the model size.

Multi-head experiments for conditional LMs: The results in Table 5.15 show that BP outperforms PPC_{KL} for both 1 and 2-attention-head models. The performance of BP improves with the number of heads, from 131.7 to 116.5 ppl for 1 and 2 heads, respectively, which is expected from the higher capacity of the larger model. Unfortunately, the performance of PPC_{KL} does not improve from 1 to 2 attention heads, with 153.8 and 154.1 ppl, respectively. This repeats the negative conclusions

	h=128	**h=256**
PPC_{KL}	150.4	156.3
BP	**144.7**	**155.5**

Table 5.14: Experiment: increased hidden size (128 and 256) for masked LM. Model: two-block, one-head masked LM, trained on five epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds.

from other conditional LM scalability experiments and suggests that the PPC_{KL} method may need additional adaptations for conditional LM scalability.

	1-head	**2-head**
PPC_{KL}	153.8	154.1
BP	**131.7**	**116.5**

Table 5.15: Experiment: multiple attention heads (1 vs 2) for conditional LMs. Model: two-transformer-block conditional LM, with 128 hidden size, trained on five epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds.

Multi-head experiments for masked LMs: Table 5.16 shows that BP slightly outperforms PPC_{KL} for both 1 and 2 attention heads. Both PPC_{KL} and BP improve in performance from 1 to 2 attention heads: from 150.4 to 123.3 ppl, and from 144.7 to 119.7 ppl, respectively. This shows that for PPC_{KL} masked LMs scale better than conditional LMs in terms of the number of attention heads.

	1-head	**2-head**
PPC_{KL}	150.4	123.3
BP	**144.7**	**119.7**

Table 5.16: Experiment: multiple attention heads (1 vs 2) for masked LMs. Model: two-transformer-block masked LM, with 128 hidden size, trained on five epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds.

Conclusion of experiments In summary, PPC_{KL} scales similarly to BP for masked LMs, but worse for conditional LMs. This means that additional modifications to the PPC_{KL} training method may be needed to achieve better scalability for conditional LMs. Overall, PPC_{KL} performs similarly to BP in the masked LM experiments, but it underperforms BP in the conditional LM experiments. This together with the scalability conclusions means that PPC_{KL} does not work well for conditional LMs. This can be explained by interference between the conditional attention mask and

the PC layer located at the softmax function directly after the mask. Unfortunately, this cannot be fixed by moving the PC layers elsewhere because PPC_{KL} functions differently from PPC only if the PC layers are used at the softmax function (Pinchetti et al., 2022), and because initial experiments suggest that PPC (non-KL) does not perform well for transformers regardless of PC layer placement. Future research can look into other ways to solve the performance problem of PPC_{KL} for conditional LMs.

Scalability Results: Longer Training Here I present the results of the experiments with a larger number of training epochs, for both masked and conditional LMs. These experiments aim to determine whether the PPC_{KL} method scales well versus BP with longer training. These results are obtained via focused hyperparameter searches around the best hyperparameters for 2-transformer-block 1-head LMs trained on 5 epochs in the hyperparameter search experiments above.

Longer training for conditional LMs: The results in Table 5.17 show that BP outperforms PPC_{KL} both with 5 and 60 training epochs. Both PPC_{KL} and BP improve with more epochs, but the effect for BP is more significant, with 131.7 to 106.2 (19.4% improvement) ppl, versus 153.8 to 136.4 (11.3% improvement) ppl for PPC_{KL}. The graphs in Figure 5.10 show that BP produces much better results than PPC_{KL} throughout all 60 epochs. This means that further improvements may be needed to the PPC_{KL} method for training conditional LMs.

	5 epochs	**60 epochs**
PPC_{KL}	153.8	136.4
BP	**131.7**	**106.2**

Table 5.17: Experiment: longer training for conditional LMs. Model: two-transformer-block, one-head conditional LM, trained on 5 vs 60 epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds for 5 epochs and 1 seed for 60 epochs.

Longer training for masked LMs: The results in Table 5.18 show that PPC_{KL} outperforms BP for both 5 and 60 training epochs. Both methods improve significantly in performance from 5 to 60 epochs, with 55.5% improvement for PPC_{KL} and 64.1% improvement for BP. These performance improvements are larger than those for the conditional LM (above), which may be because the masked LM is an optimised architecture (BERT), whereas the conditional LM is a derivative of the masked LM obtained via a conditional LM mask, hence not necessarily optimised. The graphs in Figure 5.11 show that the results of PPC_{KL} diverge from those of BP when trained

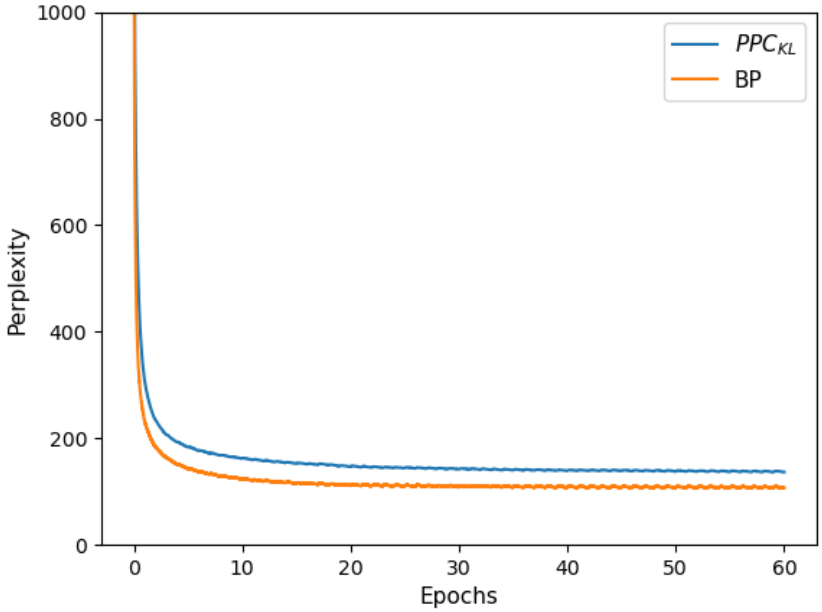


Figure 5.10: Experiment: longer training for conditional LMs. Model: two-transformer-block one-head conditional LM, trained on 60 epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better).

on up to 60 epochs. Further improvements to the PPC_{KL} method may be needed to improve the training performance and stabilise the training in order to scale to longer training.

	5 epochs	**60 epochs**
PPC_{KL}	150.4	67.0
BP	**144.7**	**52.0**

Table 5.18: Experiment: longer training for masked LMs. Model: two-transformer-block, one-head masked LM, trained on 5 vs 60 epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better). The results are the average of 3 seeds for 5 epochs and 1 seed for 60 epochs.

Figure 5.11 also shows that during training PPC_{KL} exhibits unstable behaviour beyond 35 epochs, with spikes above 500 perplexity. To investigate this phenomenon, I ran the same training configuration with three additional seeds. The results in Figure 5.12 show that the other seeds are much more stable during training, with fewer spikes and one seed having no spikes at all. Also, all seeds continue improving throughout the training and perform similarly, with 70.3, 65.8, and 67.6 ppl, respectively, versus 67.0 ppl for the original seed. This means the spiking phenomenon is

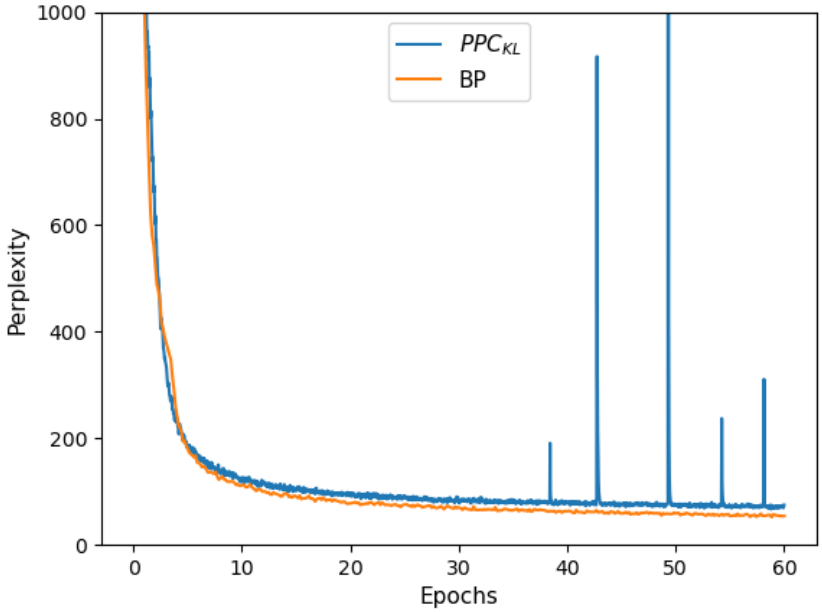


Figure 5.11: Experiment: longer training for masked LMs. Model: two-transformer-block one-head Masked LM, trained on 60 epochs. Comparison between PPC_{KL} and BP in terms of perplexity (lower is better).

less pronounced on average and does not affect the training process or outcome.

Conclusion of experiments In summary, training on 60 epochs highlights the problems of PPC_{KL} with scalability. BP scales better with more epochs than PPC_{KL} and performs better with 60 training epochs, both for conditional and masked LMs. The results also confirm the conclusions of the other scalability experiments, namely, that PPC_{KL} performs better against BP for masked LMs, than for conditional LMs.

5.6.3 Conclusion

In this work, I studied the scalability of the best predictive coding method (PPC_{KL}) for training conditional and masked transformer LMs, as determined by Sections 5.4 and 5.5. I showed that the PPC_{KL} method is more suitable for training masked transformer LMs like BERT (Devlin et al., 2019) than conditional transformer LMs like GPT (Radford et al., 2018). I compared PPC_{KL} to backpropagation (BP) when using more transformer blocks, more attention heads, a larger hidden size, or more training epochs. The model scalability results showed that PPC_{KL} scales similarly to BP for masked LMs, but worse for conditional LMs. On the other hand, the data scalability results showed that both LM variants scale worse for PPC_{KL} than for

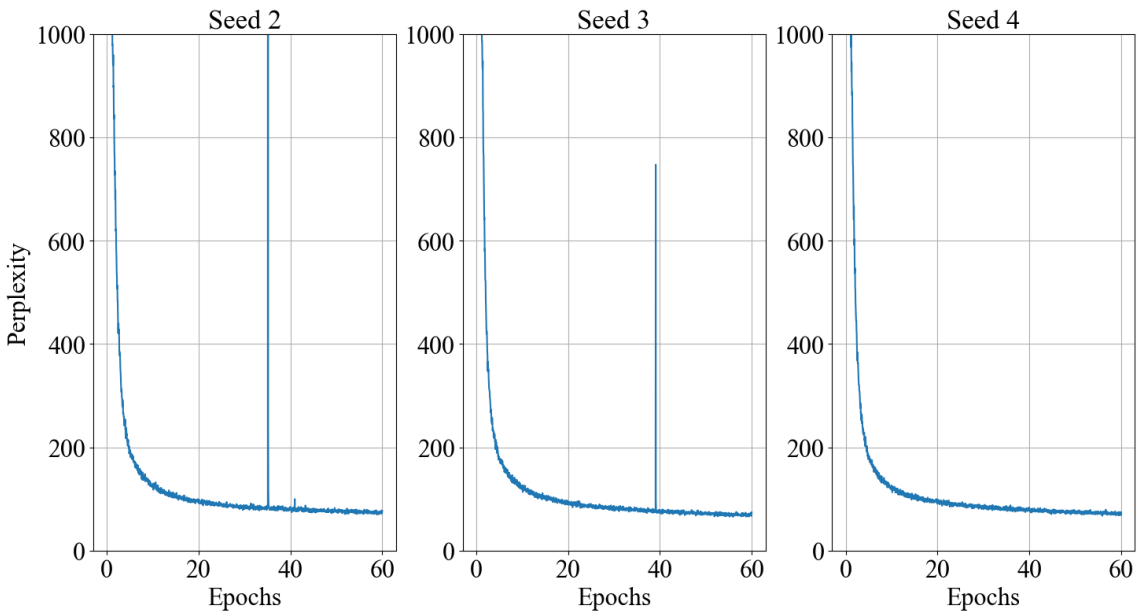


Figure 5.12: Experiment: three additional seeds for masked LMs. Model: two-transformer-block one-head Masked LM, trained on 60 epochs.

BP. Further improvements to the PPC_{KL} method may be needed to ensure better scalability, on par with BP.

The results also highlighted that the choice of some hyperparameters for PPC_{KL} is stable across experiments. These hyperparameters include the batch size, the inference learning rate, and the choice of layer normalisation. These choices can save valuable computational resources when using PPC_{KL} for training transformer LMs.

5.7 Conclusion

In this chapter, I explored training LMs by using alternative methods to BP which are more biologically plausible. These methods are based on the predictive coding theory of the brain and rely only on local computations, like the brain. I showed the first-ever results of applying biologically plausible predictive coding methods for training LMs, for both RNN and transformer architectures. I showed how to implement predictive coding for those architectures, what key hyperparameters and what predictive coding methods work best for them. In particular, I showed that predictive coding methods with the KL-modification (Pinchetti et al., 2022) significantly outperform the standard predictive coding methods for training transformer LMs. Furthermore, when using the KL-modification, the PPC method outperforms PC for transformer

LMs. When compared to BP, I showed that PPC is significantly better for simple RNN LMs, whereas PPC matches the performance of BP for small transformer LMs. Unfortunately, the data and model scalability results showed that PPC has poor scalability when compared to BP for transformer LMs, especially for conditional LMs. However, for simple RNN architectures, PPC scales similarly to BP with model and data size.

Future work can look into ways to improve the scalability of PPC and other predictive coding methods for complex architectures. Furthermore, future work can introduce architectures optimised for predictive coding, similar to Millidge et al. (2023c).

Chapter 6

Conclusion

In this thesis, I have made research contributions to the question of producing LMs that are more human-like in three important aspects. First, to understand natural language, by doing commonsense reasoning on the task of hard cases of pronoun resolution; second, to explain their decisions in natural language (producing NLEs) on commonsense reasoning tasks; and third, to be trained via more biologically plausible methods based on the predictive coding theory of the brain. These methods may be the future of deep learning beyond backpropagation by being more biologically plausible and performing better via specialised neuromorphic hardware (Millidge et al., 2022).

6.1 Main Contributions

The main contributions of this thesis are split into three chapters: 3. Hard Cases of Pronoun Resolution, 4. Transfer Learning of Natural Language Explanations, and 5. Predictive Coding for Language Models.

Hard Cases of Pronoun Resolution I have presented two important contributions to solving hard cases of pronoun resolution. My first work is based on my proof-of-concept work that inspired two influential works by Kocijan et al. (2019b,a). In the proof-of-concept work, I have constructed synthetic pronoun resolution datasets from available text corpora and have used them to train LMs to resolve pronouns on the Winograd Schema Challenge (WSC) (Levesque et al., 2012) dataset. In addition, I have presented improvements to the model selection procedure and the loss function for training conditional LMs for pronoun resolution. I have shown that my best model matches the state-of-the-art at the time while using ≈100 times fewer parameters.

In my second work (Yordanov et al., 2020), I have compared all existing categories of training objectives for pronoun resolution. I have answered the question of whether the training objective matters for performance and stability, and have shown that this depends on whether one targets the same or a different pronoun resolution dataset.

Transfer Learning of NLEs I have presented my contribution to transfer learning of NLEs, namely, my work on few-shot out-of-domain transfer learning of NLEs in a label-abundant setup (Yordanov et al., 2022). It focuses on transfer learning of NLEs between different domains (different tasks): from abundant NLEs on the *parent* task, to a *child* task where only a few training NLEs are available. Furthermore, it assumes abundant training labels for both parent and child tasks, which can help with task performance. I have identified four training methods in this setup, and have compared them on two child tasks: hard cases of pronoun resolution, in particular on WinoGrande (Sakaguchi et al., 2020), and commonsense validation, in particular on ComVE (Wang et al., 2020). I have comprehensively evaluated all methods via human evaluation, which is the gold standard of NLE evaluation. The results have demonstrated a successful transfer learning of NLEs in this setting. Furthermore, they have shown that the best training methods are those that do a separate fine-tuning on the few NLEs of the child task. Finally, I have explored the scalability of the best method for each task in terms of LM size and number of training NLEs, which I have shown to be overall good.

Predictive Coding for LMs I have presented my contributions to applying predictive coding methods for training RNN and transformer LMs, which include my transformer LM experiments that are part of two published works: (Pinchetti et al., 2022) and (Salvatori et al., 2024). In this thesis, I have shown the first-ever application of biologically plausible predictive coding methods to training LMs. I have explored ways to implement predictive coding methods for training LMs and have compared those methods to training with backpropagation (BP). I have found the best hyperparameters and ways to implement predictive coding for both RNN and transformer LMs. I have shown that the KL-modification by Pinchetti et al. (2022) improves the performance of predictive coding methods for transformer LMs (Pinchetti et al., 2022). I have further shown that the PPC (Salvatori et al., 2022b) method outperforms PC (Whittington and Bogacz, 2017) for transformer LMs. Finally, I have shown that PPC performs on par with BP for small transformer LMs, and better than BP for vanilla RNNs.

Additionally, I have studied the scalability of the PPC method in terms of data and model size, and have concluded that their scalability is similar for simple RNN LMs, but worse for transformer LMs, and especially for conditional transformer LMs. This means that further improvements to the PPC method are needed for more complex architectures, which would unlock applications for training state-of-the-art LMs.

6.2 Discussion

A natural question is why I do not combine the ideas of Chapters 3 and 4 with the predictive coding ideas of Chapter 5. The reason is the poor scalability of the results in Chapter 5 (Section 5.6.2), which means that medium-sized and large LMs are currently out of reach for predictive-coding-based training, as training those would lead to poor downstream task performance. For example, based on the poor perplexity that I obtained for LMs trained with predictive coding, I expect that the WSC pronoun resolution performance of such a model to be 50% (chance) (corresponding to Chapter 3) and the generated NLEs for it to not make sense (corresponding to Chapter 4) due to the poor task (WSC) accuracy of the model.

6.3 Outlook

Some of my research has already led to works by others. E.g. Kocijan et al. (2019b) and Kocijan et al. (2019a) improve upon the pronoun resolution results and data creation method of my proof-of-concept work on the Winograd Schema Challenge (WSC) presented in Section 3.2. Since the WSC is already solved (Kocijan et al., 2023), the new research focus would have to be on evaluating and improving models on higher-quality pronoun resolution benchmarks such as WinoGrande (Sakaguchi et al., 2020).

For transfer learning for NLE generation, future work can improve the few-shot transfer learning performance by improving its few-shot efficiency and NLE quality. Present-day SOTA LMs can already produce NLEs with few or zero training examples. Their massive pre-training can be viewed as a parent task with abundant explanations, and thus transfer learning of NLEs can be formulated for that setup. Future work can also look at the faithfulness of NLEs produced by those large LMs.

For predictive coding, future work can look into improvements in the PPC method for deeper and larger models, especially for more complex architectures such as transformers. This will then enable combining the ideas for pronoun resolution and NLE

generation in Chapters 3 and 4 with predictive coding in Chapter 5. Future research can also be done for using predictive coding for transfer learning in general, which I do not explore in this thesis. Finally, additional performance improvements can come from natively implementing existing architectures such as RNNs (Millidge et al., 2023c), or by creating new predictive-coding-optimised architectures.

On the hardware side, chips can be developed to power predictive coding methods such as PPC. As suggested by Song (2021) and Salvatori (2022), those chips would use a large number of low-performance (possibly analogue) computing units that take advantage of the local computation properties of predictive coding methods. Code libraries can be created to enable layer parallelization and localised computation on that hardware.

Appendix A

Appendix to Solving the Winograd Schema Challenge

A.1 Performance Across Dataset Parameters

These are the additional results on the effect of dataset parameters (window size, front-extension size, and back-extension size) on the quality of the data and the resulting performance of using the data for training for pronoun resolution.

Dataset Details The window size is the only parameter that affects the number of generated examples. The default window size 15 yields 5,50M pairs (as before). There is a noticeable effect when increasing the window size to 18, which yields 6,76M pairs (22.7% more); also, decreasing the window size to 15 yields 4,09M pairs (25.7% less).

Experimental Setup I consider the effect of increasing and decreasing each of the three dataset parameters. For this, I consider the following values: window size of 12, 15 (default), and 18; front context extension up to 3, 5 (default), and 7; and back context extension up to 9, 12 (default), and 15. There are seven datasets in total, as listed in Table A.1: one for the default parameter choices (15-5-12) and two for each parameter.

For quality evaluation, I test the pre-trained GPT model on each of the datasets, similar to Table 3.1, and report the maximum of the partial and full scoring accuracy.

For performance evaluation, I use Project Gutenberg as the data source and train on a random sample of 200,000 examples from each dataset as in the loss selection experiment (Table 3.2), due to computation constraints. To account for the different number of training examples with window size, for window sizes 12 and 18 I use

randomly sampled datasets of proportional size: 148,658 and 245,458, respectively. This should provide a fair representation of the performance differences to be expected with the choice of window size.

I train the models using the best loss configuration (PS – partial single) from the loss selection experiment (Table 3.2) and do a hyperparameter search over the learning rate values: $lr \in [7.8125 \times 10^{-6}, 3.125 \times 10^{-5}, 1.5625 \times 10^{-5}]$ informed by the loss selection experiment.

Window size	**Front context extension (up to)**	**Back context extension (up to)**	**GPT accuracy full/partial**	**DPR (train) accuracy**
15	5	12	0.890	0.655
12	5	12	**0.887**	0.651
18	5	12	0.893	0.650
15	3	12	0.888	**0.657**
15	7	12	0.890	0.638
15	5	9	0.889	0.637
15	5	15	0.890	0.650

Table A.1: Performance of the PS model configuration when trained on 200,000 examples from datasets with different parameters: window size, front-extension size, and back-extension size. The first line shows the dataset with default parameters. The best values are marked in bold: GPT accuracy (lower is better); and DPR accuracy (higher is better).

Results Dataset quality can be judged by the accuracy of the pre-trained GPT model, where lower accuracy means higher quality (more difficult dataset). The results in Table A.1 show that the datasets differ very little in terms of this quality measure. As expected, the examples with smaller window sizes yield slightly higher quality (88.7% vs 89.0% accuracy), and larger window sizes yield lower quality (89.3% vs 89.0% accuracy). The choices of front context extension and back context extension have less impact on quality in terms of GPT accuracy, and no effect for larger than the default values. At first glance, shorter context sizes (in dataset configurations 15-3-12 and 15-5-9) may seem to slightly improve quality (less GPT accuracy). However, the lower GPT accuracy may be due to insufficient context – with critical information missing to disambiguate the pronoun, rather than a higher quality of examples.

The results in Table A.1 also show the pronoun resolution performance when training on each dataset. Overall, the DPR (train) accuracy is insignificantly different

True:	agreed to suspend our evening lecture tours, and spent most of our time in wandering from store to *store*.
False:	agreed to suspend our evening lecture tours, and spent most of our time in wandering from store to *time*.
True:	experience logically required three divine persons, of one and the same divine essence. economic trinity required essential *trinity*.
False:	experience logically required three divine persons, of one and the same divine essence. economic trinity required essential *essence*.
True:	what did her action; her manner of living; even her existence; matter to any *living* soul?
False:	what did her action; her manner of living; even her existence; matter to any *existence* soul?
True:	and the grandfather of an emperor. As a father, he has a considerable pension from the *emperor* of Germany; and as a grandfather, he has been honour
False:	and the grandfather of an emperor. As a father, he has a considerable pension from the *pension* of Germany; and as a grandfather, he has been honour
True:	The Bible is infallible it has no authority. this we deny. Inspiration is not infallibility, but *inspiration* is authority.
False:	The Bible is infallible it has no authority. this we deny. Inspiration is not infallibility, but *bible* is authority.

Table A.2: Gutenberg Synthetic: 5 random examples

for the dataset configurations, except for 15-7-12 and 15-5-9, where the accuracy is significantly smaller (63.8% and 63.7% vs $\geq$ 65% for the others). This may be because these two combinations represent a context shift towards the front of the sentence, which is unnatural to the target dataset (DPR).

A.2 Random Examples From Each Dataset

Here are five random examples (sentence pairs) from each dataset. These random examples serve to illustrate the quality and type of the data from each dataset.

Gutenberg Synthetic See Table A.2.

Wikipedia Synthetic See Table A.3.

Gutenberg Synthetic Filtered See Table A.4.

Wikipedia Synthetic Filtered See Table A.5.

True:	Channel Islands and the Isle of Man. It is the national organisation of Quakers living in Britain. *Britain* yearly meeting refers to both the religious gathering and the organisation.
False:	Channel Islands and the Isle of Man. It is the national organisation of Quakers living in Britain. *Organisation* yearly meeting refers to both the religious gathering and the organisation.
True:	English legal system to refer to a junior barrister undertaking paid written work on behalf of a more senior *barrister*.
False:	English legal system to refer to a junior barrister undertaking paid written work on behalf of a more senior *undertaking*.
True:	Ah Quee Street was established when Kapitan Chung, Keng Quee donated his beach *Street* shophouse to be demolished to create the street that bears his name
False:	Ah Quee Street was established when Kapitan Chung, Keng Quee donated his beach *Chung* shophouse to be demolished to create the street that bears his name
True:	close into the core this counter jet is so much dimmer than the main *jet* that it is invisible in radio waves.
False:	close into the core this counter jet is so much dimmer than the main *counter* that it is invisible in radio waves.
True:	there are subcutaneous air sacs in the lower body and along the sides. Other air *sacs* are located between the sternum and the pectoral muscles and between the
False:	there are subcutaneous air sacs in the lower body and along the sides. Other air *sides* are located between the sternum and the pectoral muscles and between the

Table A.3: Wikipedia Synthetic: 5 random examples

True:	throwing out all the alfalfa roots and thorough hoeing during the growing season and keeping the *alfalfa* mowers from sawing off the tops of them, the trees may.
False:	throwing out all the alfalfa roots and thorough hoeing during the growing season and keeping the *hoeing* mowers from sawing off the tops of them, the trees may.
True:	soon we were among rocks and before us a line of breakers backed by frowning *rocks*, very dreadful to behold.
False:	soon we were among rocks and before us a line of breakers backed by frowning *breakers*, very dreadful to behold.
True:	he could have made a fortune. They crowded into his room and sat on his bed. The *bed* collapsed, and Cowan was hurled to the floor and killed again
False:	he could have made a fortune. They crowded into his room and sat on his bed. The *room* collapsed, and Cowan was hurled to the floor and killed again
True:	he set out with his host by way of the Campagna, towards the kingdom of Apulia, and *Campagna*; and very soon he had a large part thereof at his
False:	he set out with his host by way of the Campagna, towards the kingdom of Apulia, and *way*; and very soon he had a large part thereof at his
True:	enzymes contained in the seeds or fruit from which the oils are expressed, hence the necessity for separating *oils* and fats from adhering albuminous matters as quickly as possible.
False:	enzymes contained in the seeds or fruit from which the oils are expressed, hence the necessity for separating *seeds* and fats from adhering albuminous matters as quickly as possible.

Table A.4: Gutenberg Synthetic Filtered: 5 random examples

True:	both hairless and powderpuff varieties can appear in the same litter. The look of the *powderpuff* varies according to how it is groomed.
False:	both hairless and powderpuff varieties can appear in the same litter. The look of the *litter* varies according to how it is groomed.
True:	the program enabled users to divide documents into groups, display the groups on a disk and then the *documents* in the selected group, and set up a template for each
False:	the program enabled users to divide documents into groups, display the groups on a disk and then the *users* in the selected group, and set up a template for each
True:	formally set up within Cambridge assessment, with responsibility for developing and administering admissions tests. The thinking skills *admissions* testing, TSAT, unit was responsible for administering the thinking skills
False:	formally set up within Cambridge assessment, with responsibility for developing and administering admissions tests. The thinking skills *tests* testing, TSAT, unit was responsible for administering the thinking skills
True:	is a retired Canadian professional ice hockey forward who predominantly played his career in Europe, winning the elite ice hockey *league* title in his last season with the Nottingham Panthers.
False:	is a retired Canadian professional ice hockey forward who predominantly played his career in Europe, winning the elite hockey *hockey* league title in his last season with the Nottingham Panthers.
True:	at the University of Oxford, the School of Interdisciplinary Area Studies, SIAS, *School* of Interdiscplinary Area Studies, Oxford and St Antony's College specialise
False:	at the University of Oxford, the School of Interdisciplinary Area Studies, SIAS, *University* of Interdiscplinary Area Studies, Oxford and St Antony's College specialise

Table A.5: Wikipedia Synthetic Filtered: 5 random examples

Appendix B

Appendix to Comparing Training Objectives for Pronoun Resolution

B.1 Best Hyperparameters

See Table B.1 for the best hyperparameters for each model.

B.2 WSC Preprocessing

When evaluating the CSS and the MAS model on the WSC dataset, I noticed a problem with the dataset, which interfered with locating the candidates in the text. The problem is that, in some WSC examples, the given candidate options do not match word-by-word the candidates as they appear in the text. For example,

> *Madonna fired her trainer because __ couldn't stand her boyfriend.*
>
> Candidates: Madonna, The trainer.

In this example, I resolve this problem by manually replacing the candidate option "the trainer" with "her trainer", to match exactly the candidate as it appears in the text. By following this procedure, I manually modified all 88 problematic examples in WSC (out of 273 examples in total). Note that this problem does not exist for

Model	**epochs**	**batch size**	**learn. rate**
WG-SR	12	8	5e-6
BWP	8	16	5e-6
CSS	12	8	2.5e-6
MAS	12	16	5e-6

Table B.1: The best hyperparameters for every model.

WinoGrande and DPR. Furthermore, in real-world applications, such a problem does not exist, since the candidates are not provided and have to be extracted automatically from the text. Detected candidates thus match the spans in the text.

I use this modified version of WSC only for the CSS and MAS models, because they require precise candidate localization. For WG-SR and BWP, I use the unmodified WSC version. The edited dataset can be found in the code repository[1].

[1] `https://github.com/YDYordanov/WS-training-objectives`

Appendix C

Appendix to Transfer Learning of Natural Language Explanations

C.1 Datasets

WinoGrande. Because of the lack of a publicly available test set (testing happens through its leaderboard,[1] which has submission limitations), I do a random split of the original WinoGrande training dataset into 39,130 training instances (called WG-train) and 1,268 validation instances (called WG-dev). For testing, I use the original WinoGrande development set, which I denote by WG-test.

I created the small-e-WinoGrande dataset by manually constructing NLEs for 100 examples from WG-train, 50 examples from WG-dev, and 100 examples from WG-test. Example:

> The geese prefer to nest in the fields rather than the forests because in the __ predators are very visible.
> **Options:** fields, forests. **Answer:** fields.
> **NLE:** The fields are more open spaces than the forests, hence predators are more visible there.

ComVE. Originally, ComVE (Wang et al., 2020) consists of three tasks: A, B, and C, where only tasks A and C are relevant to this work. ComVE-A is the classification task of identifying which statement out of a pair of statements does not make sense. The ComVE-C task provides only the statement that does not make sense (from the pair) and requires the model to generate an NLE for why that is the case. To form a classification task with explanations, I merge tasks A and C by matching the

[1] `https://leaderboard.allenai.org/winogrande/submissions/public`

nonsensical statements, as done by Majumder et al. (2022). The resulting task can be described as "given a pair of sentences, identify which one does **not** make sense, and explain why", which I refer to simply as ComVE. The resulting ComVE dataset consists of 10,000 training, 1,000 validation, and 1,000 test instances. Each instance consists of a pair of statements, a label, and three human-generated NLEs. I use all three NLEs per example only in the full test set. For training, I use up to one NLE per example, assuming a strict few-shot regime where each NLE annotation is expensive to get. For human evaluation, I randomly sample the test dataset down to 100 instances, to save human-annotation costs.

C.2 Training Details

For all models, I fix the batch size to 16 and do a grid search over the learning rate values and the number of training epochs. For all WinoGrande models, I search over the learning rate values of 3e-4, 1e-4, and 3e-5, whereas for ComVE I search over 1e-3, 3e-4, 1e-4, and 3e-5. For e-SNLI, I train the models on 1, 2, 3, and 5 epochs. For WinoGrande, I train the models on 1, 2, 3, 5, 7, 9, and 11 epochs, and for ComVE, I train on 1, 2, 3, 5, 7, 10, and 13 epochs. When few-shot fine-tuning with NLEs, I train the models on 1, 2, 3, 5, 7, 10, 13, 17, 21, and 26 epochs. Multi-task learning always uses the hyperparameter range of the larger dataset.

The selection criteria for each model, along with the best hyperparameters are given in Table C.1. Note that the WG-dev accuracy in Table C.1 is much higher than the corresponding WG-test accuracy in Table 4.3, because WG-dev is sampled from the training dataset of WinoGrande, whereas WG-test is the original WinoGrande development set, which is filtered to increase its difficulty (Sakaguchi et al., 2020).

C.3 Human Evaluation

As suggested by Kayser et al. (2021), for each example, the annotators are provided with two (shuffled) NLEs, one from a model and one ground-truth from the test set. This serves to mentally ground the annotator's score of the model-generated NLE.

Additionally, there are multiple checks placed in the data collection form to ensure high-quality annotations. Most notably, in each group of 10 instances, at least 90% of the labels have to be answered correctly, and at least 90% of the ground-truth NLEs have to be annotated by Yes or Weak Yes. The final check requires that at most 80% of the model-generated NLEs should be annotated by Yes or Weak Yes. I

Models	Num epochs	Learning rate	Criterion	Best value
e-SNLI	3	3e-4	e-SNLI dev NLE ppl	2.192
(e-SNLI, SNLI)	3	3e-4	e-SNLI dev NLE ppl	2.199
WinoGrande Models				
(e-SNLI, WinoGrande)	5	1e-4	WG-dev acc	83.2%
e-SNLI–WinoGrande	7	3e-4	WG-dev acc	81.0%
WinoGrande	5	1e-4	WG-dev acc	85.1%
CD–fine-tune	21	3e-4	WG-dev NLE ppl	4.665
CD–union	5	1e-4	WG-dev NLE ppl	4.945
WT5–fine-tune	11	3e-4	WG-dev acc	80.8%
WT5	5	1e-4	WG-dev acc	83.4%
M1	3	3e-5	WG-dev NLE ppl	4.815
M2	10	3e-4	WG-dev NLE ppl	4.401
M3	5	1e-4	WG-dev NLE ppl	5.419
M4	17	3e-4	WG-dev NLE ppl	5.022
ComVE Models				
(e-SNLI, ComVE)	3	3e-4	ComVE dev acc	82.8%
e-SNLI–ComVE	7	3e-4	ComVE dev acc	86.8%
ComVE	5	3e-4	ComVE dev acc	88.4%
CD–fine-tune	13	3e-4	ComVE dev NLE ppl	5.170
CD–union	5	1e-4	ComVE dev NLE ppl*	9.294
WT5–fine-tune	10	3e-4	ComVE dev acc	87.0%
WT5	5	1e-4	ComVE dev acc	84.4%
M1	5	1e-4	ComVE dev NLE ppl	7.886
M2	5	1e-3	ComVE dev NLE ppl	4.958
M3	1	1e-3	ComVE dev NLE ppl	7.970
M4	5	1e-3	ComVE dev NLE ppl	5.002

Table C.1: Best hyperparameters for all trained models (including the intermediary models), along with the corresponding criterion used for model selection, and the best dev result value w.r.t. that criterion. The datasets in brackets denotes the model obtained by fine-tuning T5 on the union of those datasets; dataset1–dataset2 denotes subsequent fine-tuning on dataset1, then on dataset2. *–subject to the dev accuracy being large enough ($> 75\%$).

included this check to ensure that the annotators are more critical, and I estimated this threshold manually. These are reasonable assumptions for both WinoGrande and ComVE, judging by the quality of the ground-truth and model-generated NLEs.

I had 130 annotators for ComVE and 113 for WinoGrande. Most of the annotators annotated only ten model-generated NLEs each. To further ensure high-quality annotations, I re-annotated all the instances of the annotators who annotated many instances (more than 60 for WinoGrande and more than 100 for ComVE) but selected more than five wrong shortcomings from a sample of ten random instances, after manual inspection. I found two such annotators for ComVE and one for WinoGrande. The annotators were paid $1 per 10 pairs of NLEs.

Below are full-page screenshots of the data collection forms that I used for Wino-

Grande (Figure C.1) and ComVE (Figure C.2).

C.4 Examples of Model-Generated NLEs

In the twelve tables below Figure C.2 are the answers and NLEs for each child task (WinoGrande and ComVE) and for all eight compared models on the first six examples (out of the 100 that were evaluated). The first six tables present six examples for WinoGrande, whereas the second six tables are for ComVE.

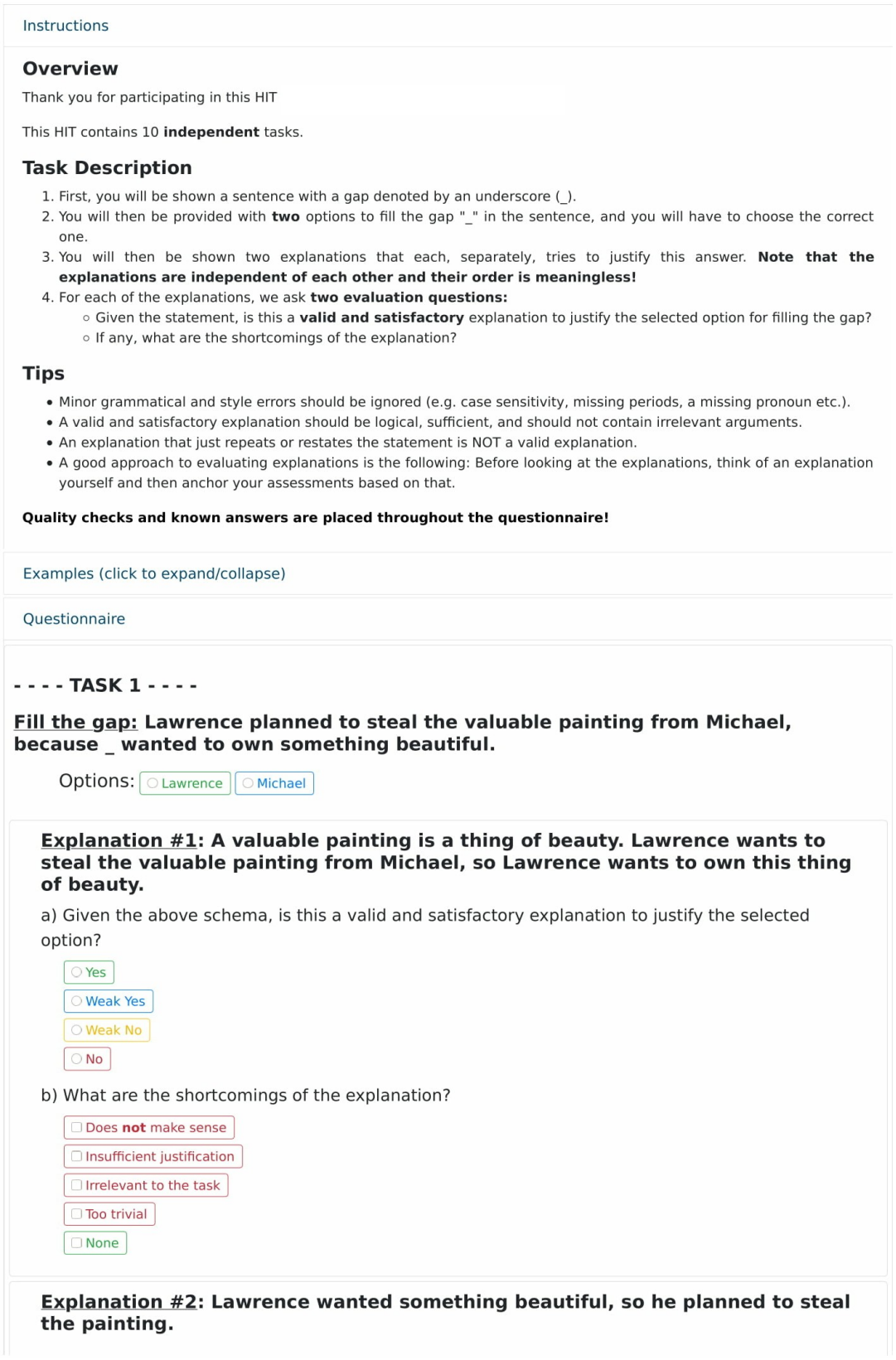


Figure C.1: WinoGrande data collection template. There are two explanations per task.

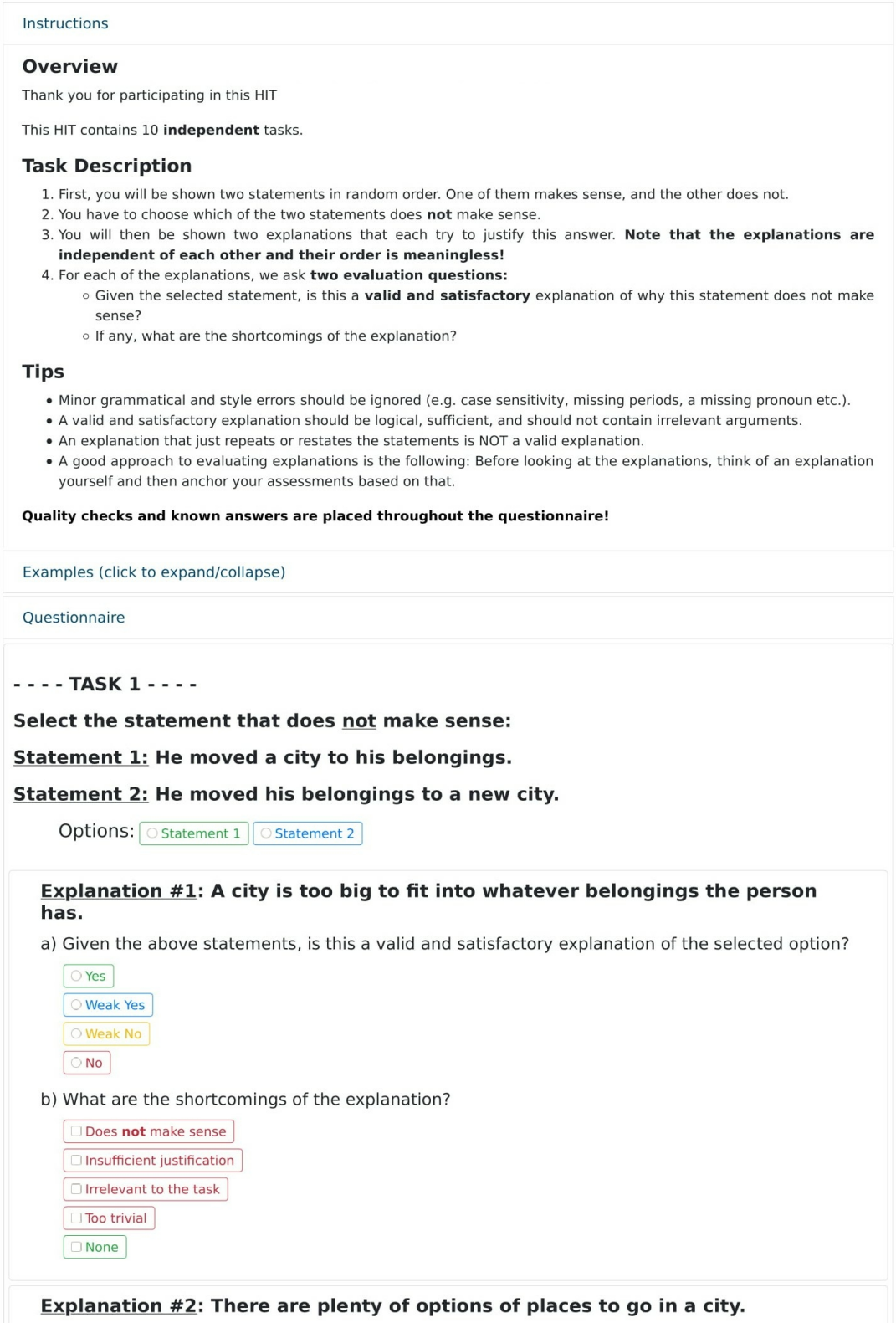


Figure C.2: ComVE data collection template. There are two explanations per task.

WinoGrande schema: The __ car was a pleasure to drive but the red car was not as the blue car was very expensive. **Options:** blue, red. **Ground-truth:** blue.	
Model	**Output**
CD–fine-tune	**Prediction:** red. **NLE:** The blue car is expensive and the blue car is more expensive.
CD–union	**Prediction:** red. **NLE:** The blue car is more expensive than the red car, hence it is more expensive to drive.
WT5–fine-tune	**Prediction:** red. **NLE:** The car is described in more detail than in the second sentence. Blue
WT5	**Prediction:** blue. **NLE:** Red is a different color than blue.
M1	**Prediction:** red. **NLE:** The car was a pleasure to drive but the blue car was not as the blue car was very expensive.
M2	**Prediction:** red. **NLE:** The red car is more expensive than the blue car.
M3	**Prediction:** blue. **NLE:** The blue car is a pleasure to drive and the red car is very expensive.
M4	**Prediction:** blue. **NLE:** The car is a pleasure to drive but the red car is red.

WinoGrande schema: Logan ended the one-side relationship they were having with Robert, which made __ feel relieved. **Options:** Logan, Robert. **Ground-truth:** Logan.	
Model	**Output**
CD–fine-tune	**Prediction:** Robert. **NLE:** Logan ended the one-side relationship they were having with Robert, which made Logan feel relieved.
CD–union	**Prediction:** Logan. **NLE:** Robert has a one-sided relationship with Logan, which makes him feel relieved.
WT5–fine-tune	**Prediction:** Logan. **NLE:** The Logan
WT5	**Prediction:** Logan. **NLE:** Robert
M1	**Prediction:** Robert. **NLE:** Logan ended the one-side relationship they were having with Robert, which made Logan feel relieved
M2	**Prediction:** Logan. **NLE:** The fact that Logan ended the one-sided relationship with Robert makes him feel relieved.
M3	**Prediction:** Robert. **NLE:** Logan's feelings are relieved because he ended the relationship with Robert.
M4	**Prediction:** Logan. **NLE:** Logan ending a one-side relationship makes Logan feel relieved.

WinoGrande schema: Fiction books were interesting and easy to read for Logan but not Brett because __ enjoyed real tales. **Options:** Logan, Brett. **Ground-truth:** Brett.	
Model	**Output**
CD–fine-tune	**Prediction:** Brett. **NLE:** Fiction books were interesting and easy to read for Logan, but Brett enjoyed real tales more than fiction books.
CD–union	**Prediction:** Brett. **NLE:** Fiction books were interesting and easy to read for Logan, while real tales were more difficult to read for Brett.
WT5–fine-tune	**Prediction:** Logan. **NLE:** There is nothing to indicate that he is enjoying real stories or that he has been referred to as 'Boy'
WT5	**Prediction:** Logan. **NLE:** The Logan may not be Logan. Brett may not be Brett. Logan may not be Logan. Brett may not be Brett. Logan may not be Logan. Brett may not be Brett. Logan may not be Logan. Brett may not be Brett. Loga [. . .]
M1	**Prediction:** Brett. **NLE:** Fiction books were interesting and easy to read for Logan but not Brett because Brett enjoyed real tales
M2	**Prediction:** Brett. **NLE:** Fiction books are not real tales.
M3	**Prediction:** Logan. **NLE:** Fiction books are interesting and easy to read for Logan, so Logan is more likely to enjoy real tales than Brett.
M4	**Prediction:** Logan. **NLE:** Fiction books are not real tales.

WinoGrande schema: Erin ordered a martini when Tanya only ordered a tonic water with lemon, because __ was pregnant. **Options:** Erin, Tanya. **Ground-truth:** Tanya.	
Model	**Output**
CD–fine-tune	**Prediction:** Tanya. **NLE:** Erin ordered a martini when Tanya ordered tonic water with lemon because she was pregnant.
CD–union	**Prediction:** Erin. **NLE:** Erin ordered a martini when Tanya ordered a tonic water with lemon, because Erin was pregnant.
WT5–fine-tune	**Prediction:** Erin. **NLE:** The Tanya is pregnant. Tanya is pregnant. Tanya
WT5	**Prediction:** Tanya. **NLE:** Erin and Tanya are two different people. Erin and Tanya are two different people. Erin and Tanya are two different people. Erin and Tanya are two different people. Erin and Tanya are two different peo [. . .]
M1	**Prediction:** Tanya. **NLE:** Erin ordered a martini when Tanya only ordered a tonic water with lemon, because _ was pregnant.
M2	**Prediction:** Tanya. **NLE:** Erin ordering a martini implies Erin ordering a martini, and Tanya ordering a tonic implies Erin is pregnant.
M3	**Prediction:** Tanya. **NLE:** Erin ordered a martini
M4	**Prediction:** Erin. **NLE:** Erin ordering a martini implies Erin is pregnant. Erin ordering a tonic water implies Erin is pregnant. Erin ordering a martini implies Erin ordering a martini. Erin ordering a tonic water implies Eri [. . .]

WinoGrande schema: The game on television was very important for Ian but not Dennis because __ was into music instead. **Options:** Ian, Dennis. **Ground-truth:** Dennis.	
Model	**Output**
CD–fine-tune	**Prediction:** Dennis. **NLE:** The game on television was very important for Ian but not for Dennis because he was into music instead.
CD–union	**Prediction:** Dennis. **NLE:** The game on television was very important for Ian, but not for Dennis, because he was into music instead.
WT5–fine-tune	**Prediction:** Dennis. **NLE:** He is playing music instead of playing music instead of playing music instead of playing music instead of playing music instead of playing music instead of playing music inste [. . .]
WT5	**Prediction:** Dennis. **NLE:** The Ian can't be Dennis. Dennis can't be Ian. Dennis can't be Ian. Dennis can't be Dennis. Dennis can't be Ian. Dennis can't be Ian. Dennis can't be Dennis. Dennis can't be Ian. Dennis can't be Dennis [. . .]
M1	**Prediction:** Dennis. **NLE:** The game on television was very important for Ian but not Dennis because he was into music instead.
M2	**Prediction:** Dennis. **NLE:** The game on television is more important for Ian than Dennis because Dennis is more likely to be into music instead.
M3	**Prediction:** Dennis. **NLE:** The game on television is very important for Ian
M4	**Prediction:** Dennis. **NLE:** The game on television is very important for Ian, so he is not into music.

WinoGrande schema: It was easier for the diner to follow their budget than the food truck because the __ had more money to spend. **Options:** diner, food truck. **Ground-truth:** diner.	
Model	**Output**
CD–fine-tune	**Prediction:** diner. **NLE:** The diner has more money to spend than the food truck.
CD–union	**Prediction:** diner. **NLE:** The diner has more money to spend than the food truck, so it is easier for them to follow their budget.
WT5–fine-tune	**Prediction:** diner. **NLE:** food truck
WT5	**Prediction:** food truck. **NLE:** diner
M1	**Prediction:** diner. **NLE:** The diner had more money to spend than the food truck.
M2	**Prediction:** diner. **NLE:** The diner has more money to spend than the food truck.
M3	**Prediction:** diner. **NLE:** The diner has more money to spend than the food truck.
M4	**Prediction:** diner. **NLE:** More money to spend is a rephrasing of more money to spend.

ComVE Statement 1: He inserts his thumb into the car ignition switch. **Statement 2:** He inserts the keys into the car ignition switch. **Nonsensical statement:** Statement 1.	
Model	**Output**
CD–fine-tune	**Prediction:** Statement 1. **NLE:** A thumb is not inserted into the car ignition switch.
CD–union	**Prediction:** Statement 1. **NLE:** It's not a car ignition switch, it's a car ignition switch, it's a car ignition switch, it's a car ignition switch, it's a car ignition switch, it's a car ignition switch, it's a car ignition switch, [. . .]
WT5–fine-tune	**Prediction:** Statement 1. **NLE:** SIDEWALK INSTALLS THE KEYS INTO THE CAR INFUSION SHUT.
WT5	**Prediction:** Statement 1. **NLE:** the keys are not the same as the thumb
M1	**Prediction:** Statement 1. **NLE:** One cannot insert his thumb into the car ignition switch.
M2	**Prediction:** Statement 1. **NLE:** One cannot insert his thumb into the car ignition switch.
M3	**Prediction:** Statement 1. **NLE:** You can't use your thumb to insert keys into the car ignition switch.
M4	**Prediction:** Statement 1. **NLE:** A thumb is not used to insert keys into a car ignition switch

ComVE Statement 1: She was met with an accident and was rushed to the hospital **Statement 2:** She was met with an accident and was rushed to the library **Nonsensical statement:** Statement 2.	
Model	**Output**
CD–fine-tune	**Prediction:** Statement 2. **NLE:** The library is closed to the public
CD–union	**Prediction:** Statement 2. **NLE:** She was met with an accident and was rushed to the library
WT5–fine-tune	**Prediction:** Statement 2. **NLE:** She was met with an accident and was rushed to the hospital 1 explanation:
WT5	**Prediction:** Statement 2. **NLE:** .
M1	**Prediction:** Statement 2. **NLE:** The library is not the only place where a person can have an accident.
M2	**Prediction:** Statement 2. **NLE:** A library is not a hospital.
M3	**Prediction:** Statement 2. **NLE:** A library is not a hospital.
M4	**Prediction:** Statement 2. **NLE:** A library is not a hospital.

ComVE Statement 1: She came off sailing across the pacific ocean **Statement 2:** She came off walking across the pacific ocean **Nonsensical statement:** Statement 2.	
Model	**Output**
CD–fine-tune	**Prediction:** Statement 2. **NLE:** She came off sailing across the pacific ocean
CD–union	**Prediction:** Statement 2. **NLE:** She came off sailing across the pacific ocean Sailing across the pacific ocean Sailing across the pacific ocean Sailing across the pacific ocean Sailing across the pacific ocean Sailing across the pac [. . .]
WT5–fine-tune	**Prediction:** Statement 2. **NLE:** She came off sailing across the pacific ocean Sentence 1:
WT5	**Prediction:** Statement 2. **NLE:** .
M1	**Prediction:** Statement 2. **NLE:** A person can't be sailing and walking at the same time.
M2	**Prediction:** Statement 2. **NLE:** sailing is not the same as walking
M3	**Prediction:** Statement 2. **NLE:** Sailing and walking are two different things.
M4	**Prediction:** Statement 2. **NLE:** sailing is not the same as walking

ComVE Statement 1: Leena daily eats two bananas which are good for health. **Statement 2:** Leena eats almost 1000 bananas daily as his morning breakfast. **Nonsensical statement:** Statement 2.	
Model	**Output**
CD–fine-tune	**Prediction:** Statement 2. **NLE:** Leena eats two bananas a day which are good for health.
CD–union	**Prediction:** Statement 2. **NLE:** Bananas are good for health and you can't eat them in the morning.
WT5–fine-tune	**Prediction:** Statement 2. **NLE:** Leena daily eats two bananas which are good for health 1 explanation:
WT5	**Prediction:** Statement 2. **NLE:** .
M1	**Prediction:** Statement 2. **NLE:** Leena daily eats two bananas which are good for health. is same as Leena eats almost 1000 bananas daily as his morning breakfast
M2	**Prediction:** Statement 2. **NLE:** The bananas are good for health.
M3	**Prediction:** Statement 2. **NLE:** The bananas are not good for health, they are good for health.
M4	**Prediction:** Statement 2. **NLE:** The bananas are good for health.

ComVE Statement 1: The fire will burn you if you touch it. **Statement 2:** The fire will cool you if you touch it. **Nonsensical statement:** Statement 2.	
Model	**Output**
CD–fine-tune	**Prediction:** Statement 2. **NLE:** The fire will burn you if you touch it.
CD–union	**Prediction:** Statement 2. **NLE:** The fire will burn you if you touch it and will cool you if you touch it
WT5–fine-tune	**Prediction:** Statement 2. **NLE:** The fire will burn you if you touch it.
WT5	**Prediction:** Statement 2. **NLE:** the fire will burn you if you touch it.
M1	**Prediction:** Statement 2. **NLE:** The fire will burn you is not the same as the fire that will cool you.
M2	**Prediction:** Statement 2. **NLE:** The fire will burn you if you touch it
M3	**Prediction:** Statement 1. **NLE:** Fire can't burn you
M4	**Prediction:** Statement 2. **NLE:** The fire will burn you.

ComVE Statement 1: Salsa is for hair **Statement 2:** Brushes are for hair **Nonsensical statement:** Statement 1.	
Model	**Output**
CD–fine-tune	**Prediction:** Statement 1. **NLE:** Salsa is for hair
CD–union	**Prediction:** Statement 1. **NLE:** Salsa is for hair
WT5–fine-tune	**Prediction:** Statement 1. **NLE:** SENTENCE 1:
WT5	**Prediction:** Statement 1. **NLE:** :
M1	**Prediction:** Statement 1. **NLE:** Brushes are for hair is not the same as a Salsa
M2	**Prediction:** Statement 1. **NLE:** Salsa is for hair
M3	**Prediction:** Statement 1. **NLE:** Brushes are for hair is a rephrasing of brushes are for hair.
M4	**Prediction:** Statement 1. **NLE:** Salsa is for hair

Appendix D

Appendix to Predictive Coding for RNN-Based Language Models

D.1 Statistical Significance

The results in Table D.1 show the statistical significance across the seeds of all comparisons of PPC vs BP that use multiple seeds (3 seeds). All tests show that all comparisons apart from the 3-layer comparison are statistically significant with $p < 0.05$.

Experiment: data sizes	**Two-sample t-test: one-/two-tailed p-values**	**Paired-samples t-test: one-/two-tailed p-values**
train-S	0.00006 / 0.00012	0.00369 / 0.00738
train-M	0.00244 / 0.00489	0.01574 / 0.03147
train-L	0.00051 / 0.00102	0.00635 / 0.01271
train-XL	0.00036 / 0.00071	0.00394 / 0.00788
Experiment: hidden sizes		
h=64	0.00070 / 0.00139	0.00845 / 0.01690
h=128	0.00244 / 0.00489	0.01574 / 0.03147
h=256	0.00011 / 0.00021	0.00324 / 0.00648
Experiment: num layers		
1 layer	0.00002 / 0.00004	0.00146 / 0.00292
2 layers	0.00244 / 0.00489	0.01574 / 0.03147
3 layers	0.18358 / 0.36717	0.25070 / 0.50141

Table D.1: Comparing PPC vs BP via statistical significance tests. Both one-tailed and two-tailed p-values are reported, for two statistical significance tests.

Appendix E

Appendix to Predictive Coding beyond Gaussian Distributions

E.1 Training Details

Here are the hyperparameter ranges and best values used for each model:

For BP: $lr_{weight} \in \{0.0004, 0.0008, 0.0016, 0.0032, 0.0064\}$.
Best value: $lr_{weight} = 0.0016$.

For $PC_{\mathcal{F}}$: $T \in \{4, 5, 6, 7, 8\}$,
$lr_{inference} \in \{0.001953125, 0.00390625, 0.0078125, 0.015625, 0.03125\}$,
$lr_{weight} \in \{0.0002, 0.0004, 0.0008, 0.0016, 0.0032, 0.0064, 0.0128\}$.
Best values: $T = 4$, $lr_{inference} = 0.015625$, $lr_{weight} = 0.0064$.

For $PC_{\mathcal{F}_{KL}}$: $T \in \{4, 5, 6, 7, 8\}$, $lr_{inference} \in \{0.25, 0.5, 1.0\}$,
$lr_{weight} \in \{0.000025, 0.00005, 0.0001, 0.0002, 0.0004, 0.0008, 0.0016\}$.
Best values: $T = 5$, $lr_{inference} = 0.5$, $lr_{weight} = 0.0008$.

The total training time of the hyperparameter search is approximately 94 hours on one Nvidia Titan RTX GPU.

Appendix F

Appendix to A Stable, Fast, and Fully Automatic Learning Algorithm for Predictive Coding Networks

F.1 Hyperparameters

Here are the hyperparameter ranges used for grid search for each model:

Masked LM:

- **BP:** $batch\_size \in \{32, 64, 128, 256, 512\}$,

  $lr \in \{0.0002, 0.0004, 0.0008, 0.0016, 0.0032, 0.0064, 0.0128\}$.

  Best combination: $batch\_size = 256$, $lr = 0.0032$.

- **PC:** $T \in \{5, 6, 7, 8, 9, 10, 11, 12\}$,

  $lr_{inference} \in \{0.015625, 0.03125, 0.0625\}$,

  $lr \in \{0.0001, 0.0002, 0.0004, 0.0008\}$.

  Best combination: $T = 8$, $lr_{inference} = 0.03125$, $lr = 0.0004$.

- **PPC:** $T \in \{4, 5, 6, 7, 8\}$,

  $lr \in \{0.0004, 0.0008, 0.0016, 0.0032\}$.

  Best combination: $T = 6$, $lr = 0.0008$.

Conditional LM:

- **BP:** $batch\_size \in \{32, 64, 128, 256, 512\}$,

  $lr \in \{0.0002, 0.0004, 0.0008, 0.0016, 0.0032, 0.0064, 0.0128\}$.

  Best combination: $batch\_size = 64$, $lr = 0.0016$.

- **PC:** $T \in \{7, 8, 9, 10, 11, 12\}$,

  $lr_{inference} \in \{0.00390625, 0.0078125, 0.015625, 0.03125\}$,

  $lr \in \{0.0001, 0.0002, 0.0004, 0.0008\}$.

  Best combination: $T = 10$, $lr_{inference} = 0.0078125$, $lr = 0.0004$.

- **PPC:** $T \in \{3, 4, 5, 6\}$,

  $lr \in \{0.0008, 0.0016, 0.0032, 0.0064, 0.0128\}$.

  Best combination: $T = 4$, $lr = 0.0032$.

F.2 Per-Seed Results

Conditional LM results:
PC: [201.096, 206.322, 198.798, 187.0, 214.385, 213.66, 218.577, 221.076, 219.609, 182.898]
PPC: [137.086, 138.435, 235.153, 160.08, 139.284, 140.68, 149.148, 237.234, 281.977, 133.098]
BP: [112.655, 113.454, 113.454, 113.235, 113.726, 115.153, 112.391, 112.877, 113.359, 112.921]
Masked LM results:
PC: [503.548, 1042.125, 1158.864, 909.209, 557.766, 1155.093, 1176.144, 901.295, 507.916, 1162.502]
PPC: [93.998, 93.899, 93.449, 137.027, 94.491, 108.241, 92.411, 97.724, 156.112, 94.504]
BP: [83.619, 147.775, 787.334, 133.073, 358.486, 455.535, 92.951, 95.322, 152.135, 135.3]

Bibliography

Shourya Aggarwal, Divyanshu Mandowara, Vishwajeet Agrawal, Dinesh Khandelwal, Parag Singla, and Dinesh Garg. 2021. Explanations for CommonsenseQA: New dataset and models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3050–3065. Association for Computational Linguistics.

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. *arXiv preprint arXiv:2101.11038*.

Tariq Alhindi, Savvas Petridis, and Smaranda Muresan. 2018. Where is your evidence: Improving fact-checking by justification modeling. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 85–90. Association for Computational Linguistics.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. 2015. VQA: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Pepa Atanasova, Oana-Maria Camburu, Christina Lioma, Thomas Lukasiewicz, Jakob Grue Simonsen, and Isabelle Augenstein. 2023. Faithfulness tests for natural language explanations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 283–294. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Daniel Bailey, Amelia Harrison, Yuliya Lierler, Vladimir Lifschitz, and Julian Michael. 2015. The Winograd Schema Challenge and reasoning about correlation. In *Working Notes of the Symposium on Logical Formalizations of Commonsense Reasoning*. AAAI Press.

Payal Bajaj, Chenyan Xiong, Guolin Ke, Xiaodong Liu, Di He, Saurabh Tiwary, Tie-Yan Liu, Paul Bennett, Xia Song, and Jianfeng Gao. 2022. Metro: Efficient denoising pretraining of large scale autoencoding language models with model generated signals. *arXiv preprint arXiv:2204.06644*.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72. Association for Computational Linguistics.

David Bender. 2015. Establishing a human baseline for the Winograd Schema Challenge. In *Midwest Artificial Intelligence and Cognitive Science Conference*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. 2016. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*.

Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Wen tau Yih, and Yejin Choi. 2020. Abductive commonsense reasoning. In *International Conference on Learning Representations*.

Victor Boutin, Angelo Franciosini, Frederic Chavane, Franck Ruffier, and Laurent Perrinet. 2019. Sparse deep predictive coding captures contour integration capabilities of the early visual system. *arXiv preprint arXiv:1902.07651*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642. Association for Computational Linguistics.

Stevo Bozinovski. 2020. Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica (Slovenia)*, 44.

Faeze Brahman, Vered Shwartz, Rachel Rudinger, and Yejin Choi. 2021. Learning to rationalize for nonmonotonic reasoning with distant supervision. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12592–12601.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. 2018. e-SNLI: Natural language inference with natural language explanations. In *Advances in Neural Information Processing Systems 31*, pages 9539–9549. Curran Associates, Inc.

Oana-Maria Camburu, Brendan Shillingford, Pasquale Minervini, Thomas Lukasiewicz, and Phil Blunsom. 2020. Make up your mind! Adversarial generation of inconsistent natural language explanations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4157–4165.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Qianglong Chen, Feng Ji, Xiangji Zeng, Feng-Lin Li, Ji Zhang, Haiqing Chen, and Yin Zhang. 2021. KACE: Generating knowledge aware contrastive explanations for natural language inference. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2516–2527. Association for Computational Linguistics.

Zhiyu Chen, Harini Eavani, Wenhu Chen, Yinyin Liu, and William Yang Wang. 2020. Few-shot NLG with pre-trained language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 183–190. Association for Computational Linguistics.

Anna Choromanska, Benjamin Cowen, Sadhana Kumaravel, Ronny Luss, Mattia Rigotti, Irina Rish, Brian Kingsbury, Paolo DiAchille, Viatcheslav Gurev, Ravi Tejwani, and Djallel Bouneffouf. 2019. Beyond backprop: Online alternating minimization with auxiliary variables. *arXiv preprint arXiv:1806.09077*.

Andy Clark. 2013. Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(3):181–204.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

Alexis Conneau and Douwe Kiela. 2018. SentEval: An evaluation toolkit for universal sentence representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA).

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680. Association for Computational Linguistics.

Ana-Maria Cretu. 2018. Group-sparse sentence representations. Master's thesis, EPFL University, Switzerland.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190. Springer.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for*

Computational Linguistics, pages 2978–2988. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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. Association for Computational Linguistics.

Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C. Wallace. 2020. ERASER: A benchmark to evaluate rationalized NLP models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4443–4458. Association for Computational Linguistics.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Upol Ehsan, Brent Harrison, Larry Chan, and Mark O. Riedl. 2018. Rationalization: A neural machine translation approach to generating natural language explanations. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '18, page 81–87. Association for Computing Machinery.

Yanai Elazar, Hongming Zhang, Yoav Goldberg, and Dan Roth. 2021. Back to square one: Artifact detection, training and commonsense disentanglement in the Winograd Schema. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10486–10500. Association for Computational Linguistics.

Ali Emami, Noelia De La Cruz, Adam Trischler, Kaheer Suleman, and Jackie Chi Kit Cheung. 2018. A knowledge hunting framework for common sense reasoning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1949–1958. Association for Computational Linguistics.

Ali Emami, Paul Trichelair, Adam Trischler, Kaheer Suleman, Hannes Schulz, and Jackie Chi Kit Cheung. 2019. The KnowRef coreference corpus: Removing gender and number cues for difficult pronominal anaphora resolution. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3952–3961. Association for Computational Linguistics.

Karl Fredrik Erliksson, Anders Arpteg, Mihhail Matskin, and Amir H. Payberah. 2021. Cross-domain transfer of generative explanations using text-to-text models. In *Natural Language Processing and Information Systems: 26th International Conference on Applications of Natural Language to Information Systems, NLDB 2021, June 23–25, 2021, Proceedings*, page 76–89. Springer-Verlag.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Karl Friston. 2005. A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836.

Karl Friston. 2010. The free-energy principle: A unified brain theory? *Nature reviews. Neuroscience*, 11:127–38.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Albert Gu, Karan Goel, and Christopher Re. 2022. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*.

Xiaochuang Han, Byron C. Wallace, and Yulia Tsvetkov. 2020. Explaining black box predictions and unveiling data artifacts through influence functions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5553–5563. Association for Computational Linguistics.

C. W. Harrison. 1952. Experiments with linear prediction in television. *Bell System Technical Journal*, 31(4):764–783.

Peter Hase, Shiyue Zhang, Harry Xie, and Mohit Bansal. 2020. Leakage-adjusted simulatability: Can models generate non-trivial explanations of their behavior in natural language? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4351–4367. Association for Computational Linguistics.

Pengcheng He, Xiaodong Liu, Weizhu Chen, and Jianfeng Gao. 2019. A hybrid neural network model for commonsense reasoning. In *Proceedings of the 1st Workshop on Commonsense Inference in Natural Language Processing*, pages 13–21. Association for Computational Linguistics.

Lisa Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. 2016. Generating visual explanations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 9908 of *LNCS*, pages 3–19.

Sepp Hochreiter. 1991. *Untersuchungen zu dynamischen neuronalen Netzen*. Ph.D. thesis, Technical University Munich, Institute of Computer Science.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Benjamin Hoover, Yuchen Liang, Bao Pham, Rameswar Panda, Hendrik Strobelt, Duen Horng Chau, Mohammed Zaki, and Dmitry Krotov. 2023. Energy transformer. *Advances in Neural Information Processing Systems*, 36.

John J. Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79 8:2554–8.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339. Association for Computational Linguistics.

Zhouyuan Huo, Bin Gu, Qian Yang, and Heng Huang. 2018. Decoupled parallel backpropagation with convergence guarantee. *arXiv preprint arXiv:1804.10574*.

Naoya Inoue, Harsh Trivedi, Steven Sinha, Niranjan Balasubramanian, and Kentaro Inui. 2021. Summarize-then-answer: Generating concise explanations for multi-hop reading comprehension. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6064–6080. Association for Computational Linguistics.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Nicos Isaak and Loizos Michael. 2016. Tackling the Winograd Schema Challenge through machine logical inferences. In *STAIRS 2016*, pages 75–86. IOS Press.

Nicos Isaak and Loizos Michael. 2020. Winventor: A machine-driven approach for the development of Winograd schemas. In *International Conference on Agents and Artificial Intelligence*.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Alon Jacovi and Yoav Goldberg. 2020. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4198–4205. Association for Computational Linguistics.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. 2017. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*.

Myeongjun Jang and Thomas Lukasiewicz. 2021. Are training resources insufficient? Predict first then explain! *arXiv preprint arXiv:2110.02056*.

Frederick Jelinek, Robert L. Mercer, Lalit R. Bahl, and Janet M. Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *Journal of the Acoustical Society of America*, 62.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeff Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Maxime Kayser, Oana-Maria Camburu, Leonard Salewski, Cornelius Emde, Virginie Do, Zeynep Akata, and Thomas Lukasiewicz. 2021. e-ViL: A dataset and benchmark for natural language explanations in vision-language tasks. In *2021*

IEEE/CVF International Conference on Computer Vision (ICCV), pages 1224–1234.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UnifiedQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907. Association for Computational Linguistics.

Jinkyu Kim, Anna Rohrbach, Trevor Darrell, John Canny, and Zeynep Akata. 2018. Textual explanations for self-driving vehicles. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part II*, page 577–593. Springer-Verlag.

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Conference Track Proceedings*.

Dietrich Klakow and Jochen Peters. 2002. Testing the correlation of word error rate and perplexity. *Speech Communication*, 38(1):19–28.

Tassilo Klein and Moin Nabi. 2019. Attention is (not) all you need for commonsense reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4831–4836. Association for Computational Linguistics.

David C. Knill and Alexandre Pouget. 2004. The Bayesian brain: The role of uncertainty in neural coding and computation. *Trends in Neurosciences*, 27(12):712–719.

Vid Kocijan, Oana-Maria Camburu, Ana-Maria Cretu, Yordan Yordanov, Phil Blunsom, and Thomas Lukasiewicz. 2019a. WikiCREM: A large unsupervised corpus for coreference resolution. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4303–4312. Association for Computational Linguistics.

Vid Kocijan, Ana-Maria Cretu, Oana-Maria Camburu, Yordan Yordanov, and Thomas Lukasiewicz. 2019b. A surprisingly robust trick for the Winograd Schema Challenge. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4837–4842. Association for Computational Linguistics.

Vid Kocijan, Ernest Davis, Thomas Lukasiewicz, Gary Marcus, and Leora Morgenstern. 2023. The defeat of the Winograd Schema Challenge. *Artificial Intelligence*, 325:103971.

Vid Kocijan, Thomas Lukasiewicz, Ernest Davis, Gary F. Marcus, and L. Morgenstern. 2020. A review of Winograd Schema Challenge datasets and approaches. *arXiv preprint arXiv:2004.13831*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, System Demonstrations*, pages 66–71.

Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. 2022. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations*.

Sawan Kumar and Partha Talukdar. 2020. NILE: Natural language inference with faithful natural language explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8730–8742. Association for Computational Linguistics.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Veronica Latcinnik and Jonathan Berant. 2020. Explaining question answering models through text generation. *arXiv preprint arXiv:2004.05569*.

Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2002. Efficient BackProp. In *Neural networks: Tricks of the trade*, pages 9–50. Springer.

Dong-Ho Lee, Rahul Khanna, Bill Yuchen Lin, Seyeon Lee, Qinyuan Ye, Elizabeth Boschee, Leonardo Neves, and Xiang Ren. 2020. LEAN-LIFE: A label-efficient annotation framework towards learning from explanation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 372–379. Association for Computational Linguistics.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. 2015. Difference target propagation. *arXiv preprint arXiv:1412.7525*.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The Winograd Schema Challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, page 552–561. AAAI Press.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Chan Li, Junbin Qiu, and Haiping Huang. 2023. Meta predictive learning model of natural languages. *arXiv preprint arXiv:2309.04106*.

Qing Li, Qingyi Tao, Shafiq Joty, Jianfei Cai, and Jiebo Luo. 2018. VQA-E: Explaining, elaborating, and enhancing your answers for visual questions. In *The European Conference on Computer Vision (ECCV)*.

Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. 2020. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Sheng-Chieh Lin, Jheng-Hong Yang, Rodrigo Nogueira, Ming-Feng Tsai, Chuan-Ju Wang, and Jimmy Lin. 2020. TTTTTackling WinoGrande schemas. *arXiv preprint arXiv:2003.08380*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017a. Program induction by rationale generation: Learning to solve and explain algebraic word

problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167. Association for Computational Linguistics.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017b. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 158–167.

Hui Liu, Qingyu Yin, and William Yang Wang. 2019a. Towards explainable NLP: A generative explanation framework for text classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5570–5581. Association for Computational Linguistics.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Quan Liu, Hui Jiang, Zhen-Hua Ling, Xiaodan Zhu, Si Wei, and Yu Hu. 2017. Combing context and commonsense knowledge through neural networks for solving Winograd Schema problems. In *2017 AAAI Spring Symposium Series*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in Adam. *arXiv preprint arXiv:1711.05101v1*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations*. OpenReview.net.

Nicholas Lourie, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Unicorn on rainbow: A universal commonsense reasoning model on a new multitask benchmark. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13480–13488.

Bodhisattwa Prasad Majumder, Oana-Maria Camburu, Thomas Lukasiewicz, and Julian McAuley. 2022. Knowledge-grounded self-rationalization via extractive and natural language explanations. *Proceedings of 39th International Conference on Machine Learning (ICML)*.

Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan Yuille, and Kevin Murphy. 2016. Generation and comprehension of unambiguous object descriptions. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11–20.

Ana Marasović, Chandra Bhagavatula, Jae sung Park, Ronan Le Bras, Noah A. Smith, and Yejin Choi. 2020. Natural language rationales with full-stack visual reasoning: From pixels to semantic frames to commonsense graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2810–2829. Association for Computational Linguistics.

Ana Marasović, Iz Beltagy, Doug Downey, and Matthew E. Peters. 2022. Few-shot self-rationalization with natural language prompts. In *Findings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

James L. McClelland and David E. Rumelhart. 1988. An interactive activation model of context effects in letter perception: Part I. an account of basic findings. In Allan Collins and Edward E. Smith, editors, *Readings in Cognitive Science*, pages 580–596. Morgan Kaufmann.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165.

Clara Meister and Ryan Cotterell. 2021. Language model evaluation beyond perplexity. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5328–5339. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

Beren Millidge, Tommaso Salvatori, Yuhang Song, Rafal Bogacz, and Thomas Lukasiewicz. 2022. Predictive coding: Towards a future of deep learning beyond backpropagation? In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5538–5545. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Beren Millidge, Anil Seth, and Christopher L Buckley. 2021. Predictive coding: a theoretical and experimental review. *arXiv preprint arXiv:2107.12979*.

Beren Millidge, Yuhang Song, Tommaso Salvatori, Thomas Lukasiewicz, and Rafal Bogacz. 2023a. Backpropagation at the infinitesimal inference limit of energy-based models: Unifying predictive coding, equilibrium propagation, and contrastive Hebbian learning. In *The Eleventh International Conference on Learning Representations*.

Beren Millidge, Yuhang Song, Tommaso Salvatori, Thomas Lukasiewicz, and Rafal Bogacz. 2023b. A theoretical framework for inference and learning in predictive coding networks. In *The Eleventh International Conference on Learning Representations*.

Beren Millidge, Mufeng Tang, Mahyar Osanlouy, and Rafal Bogacz. 2023c. Predictive coding networks for temporal prediction. *bioRxiv*.

Beren Millidge, Alexander Tschantz, and Christopher L. Buckley. 2020. Predictive coding approximates backprop along arbitrary computation graphs. *arXiv preprint arXiv:2006.04182*.

Leora Morgenstern, Ernest Davis, and Charles Ortiz. 2016. Planning, executing, and evaluating the winograd schema challenge. *AI Magazine*, 37:50–54.

Sharan Narang, Colin Raffel, Katherine Lee, A. Roberts, Noah Fiedel, and Karishma Malkan. 2020. WT5?! Training text-to-text models to explain their predictions. *arXiv preprint arXiv:2004.14546*.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197. Association for Computational Linguistics.

Arild Nøkland and Lars Hiller Eidnes. 2019. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*.

B. M. Oliver. 1952. Efficient coding. *Bell System Technical Journal*, 31(4):724–750.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Juri Opitz and Anette Frank. 2018. Addressing the Winograd Schema Challenge as a sequence ranking task. In *Proceedings of the 1st International Workshop on Language Cognition and Computational Models*, pages 41–52. Association for Computational Linguistics.

Alexander Ororbia. 2019. Spiking neural predictive coding for continual learning from data streams. *arXiv preprint arXiv:1908.08655*.

Alexander Ororbia and Daniel Kifer. 2022. The neural coding framework for learning generative models. *Nature communications*, 13(1):2064.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics.

Dong Huk Park, Lisa Anne Hendricks, Zeynep Akata, Anna Rohrbach, Bernt Schiele, Trevor Darrell, and Marcus Rohrbach. 2018. Multimodal explanations: Justifying decisions and pointing to the evidence. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8779–8788.

Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. 2020. Few-shot natural language generation for task-oriented dialog. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 172–182. Association for Computational Linguistics.

Haoruo Peng, Daniel Khashabi, and Dan Roth. 2015. Solving hard coreference problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 809–819. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Luca Pinchetti, Tommaso Salvatori, Yordan Yordanov, Beren Millidge, Yuhang Song, and Thomas Lukasiewicz. 2022. Predictive coding beyond Gaussian distributions. In *Advances in Neural Information Processing Systems*.

Sameer Pradhan and Lance Ramshaw. 2017. *OntoNotes: Large Scale Multi-Layer, Multi-Lingual, Distributed Annotation*, pages 521–554. Springer Netherlands.

A. Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

A. Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Altaf Rahman and Vincent Ng. 2012. Resolving complex cases of definite pronouns: The Winograd Schema Challenge. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 777–789. Association for Computational Linguistics.

Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain yourself! Leveraging language models for commonsense reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4932–4942. Association for Computational Linguistics.

Ahad Rana. 2010. Common Crawl – building an open web-scale crawl using Hadoop.

Rajesh Rao and Dana Ballard. 1999. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2:79–87.

Sachin Ravi and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Conference Track Proceedings*. OpenReview.net.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *arXiv preprint arXiv:2002.12327*.

Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. 2018. Gender bias in coreference resolution. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 8–14. Association for Computational Linguistics.

Rachel Rudinger, Vered Shwartz, Jena D. Hwang, Chandra Bhagavatula, Maxwell Forbes, Ronan Le Bras, Noah A. Smith, and Yejin Choi. 2020. Thinking like a skeptic: Defeasible inference in natural language. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4661–4675. Association for Computational Linguistics.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. *Learning representations by back-propagating errors*, page 696–699. MIT Press.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. WinoGrande: An adversarial Winograd Schema Challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740.

Tommaso Salvatori. 2022. *Learning and memorization via predictive coding*. Ph.D. thesis, University of Oxford.

Tommaso Salvatori, Ankur Mali, Christopher L Buckley, Thomas Lukasiewicz, Rajesh PN Rao, Karl Friston, and Alexander Ororbia. 2023. Brain-inspired computational intelligence via predictive coding. *arXiv preprint arXiv:2308.07870*.

Tommaso Salvatori, Luca Pinchetti, Beren Millidge, Yuhang Song, Tianyi Bao, Rafal Bogacz, and Thomas Lukasiewicz. 2022a. Learning on arbitrary graph topologies via predictive coding. In *Proceedings of the 36th Annual Conference on Neural Information Processing Systems NeurIPS 2022*.

Tommaso Salvatori, Yuhang Song, Thomas Lukasiewicz, Rafal Bogacz, and Zhenghua Xu. 2021. Predictive coding can do exact backpropagation on convolutional and recurrent neural networks. *arXiv preprint arXiv:2103.03725*.

Tommaso Salvatori, Yuhang Song, Beren Millidge, Zhenghua Xu, Lei Sha, Cornelius Emde, Rafal Bogacz, and Thomas Lukasiewicz. 2022b. Incremental predictive coding: A parallel and fully automatic learning algorithm. *arXiv preprint arXiv:2212.00720*.

Tommaso Salvatori, Yuhang Song, Zhenghua Xu, Thomas Lukasiewicz, and Rafal Bogacz. 2022c. Reverse differentiation via predictive coding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):8150–8158.

Tommaso Salvatori, Yuhang Song, Yordan Yordanov, Beren Millidge, Zhen ghua Xu, Lei Sha, Cornelius Emde, Rafal Bogacz, and Thomas Lukasiewicz. 2024. A stable, fast, and fully automatic learning algorithm for predictive coding networks. In *International Conference on Learning Representations*.

Maarten Sap, Saadia Gabriel, Lianhui Qin, Dan Jurafsky, Noah A. Smith, and Yejin Choi. 2020. Social bias frames: Reasoning about social and power implications of language. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5477–5490. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics.

Igor Shalyminov, Sungjin Lee, Arash Eshghi, and Oliver Lemon. 2019. Few-shot dialogue generation without annotated data: A transfer learning approach. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 32–39. Association for Computational Linguistics.

Arpit Sharma, Nguyen H. Vo, Somak Aditya, and Chitta Baral. 2015. Towards addressing the Winograd Schema Challenge: Building and using a semantic parser and a knowledge hunting module. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 1319–1325. AAAI Press.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Jesus Solano, Oana-Maria Camburu, and Pasquale Minervini. 2023. Sparsefit: Few-shot prompting with sparse fine-tuning for jointly generating predictions and natural language explanations. *arXiv preprint arXiv:2305.13235*.

Y Song. 2021. *Predictive coding inspires effective alternatives to backpropagation*. Ph.D. thesis, University of Oxford.

Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. 2020. Can the brain do backpropagation? — exact implementation of backpropagation in predictive coding networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 22566–22579. Curran Associates, Inc.

Yuhang Song, Beren Millidge, Tommaso Salvatori, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. 2024. Inferring neural activity before plasticity as a foundation for learning beyond backpropagation. *Nature Neuroscience*, pages 1–11.

M.V. Srinivasan, Simon Laughlin, and AT Dubs. 1982. Predictive coding: A fresh view of inhibition in the retina. *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, 216:427–59.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Gábor Stefanics, Jan Kremláček, and István Czigler. 2014. Visual mismatch negativity: a predictive coding view. *Frontiers in Human Neuroscience*, 8.

Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. 2021. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*.

Wei Suo, Mengyang Sun, Weisong Liu, Yiqi Gao, Peng Wang, Yanning Zhang, and Qi Wu. 2023. S3C: Semi-supervised VQA natural language explanation via self-critical learning. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2646–2656.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. 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 4149–4158. Association for Computational Linguistics.

Mufeng Tang, Helen Barron, and Rafal Bogacz. 2023. Sequential memory with temporal predictive coding. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Sebastian Thrun. 1996. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, volume 8, pages 640–646. MIT Press.

Trieu H. Trinh and Quoc V. Le. 2018. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*.

Marco Valentino, Mokanarangan Thayaparan, and André Freitas. 2022. Case-based abductive natural language inference. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1556–1568. International Committee on Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, L. Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019a. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32*, pages 3266–3280. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355. Association for Computational Linguistics.

Cunxiang Wang, Shuailong Liang, Yili Jin, Yilong Wang, Xiaodan Zhu, and Yue Zhang. 2020. SemEval-2020 Task 4: Commonsense validation and explanation. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 307–321. International Committee for Computational Linguistics.

Shuohang Wang, Sheng Zhang, Yelong Shen, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, and Jing Jiang. 2019b. Unsupervised deep structured semantic models for commonsense reasoning. 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 882–891. Association for Computational Linguistics.

William Yang Wang. 2017. "Liar, liar pants on fire": A new benchmark dataset for fake news detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 422–426. Association for Computational Linguistics.

Zirui Wang, Adams Wei Yu, Orhan Firat, and Yuan Cao. 2021. Towards zero-label language learning. *arXiv preprint arXiv:2109.09193*.

Kellie Webster, Marta Recasens, Vera Axelrod, and Jason Baldridge. 2018. Mind the GAP: A balanced corpus of gendered ambiguous pronouns. *Transactions of the Association for Computational Linguistics*, 6:605–617.

James Whittington and Rafal Bogacz. 2017. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, 29:1–34.

Sarah Wiegreffe, Jack Hessel, Swabha Swayamdipta, Mark Riedl, and Yejin Choi. 2022. Reframing human-AI collaboration for generating free-text explanations. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 632–658. Association for Computational Linguistics.

Sarah Wiegreffe, Ana Marasović, and Noah A. Smith. 2021. Measuring association between labels and free-text rationales. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10266–10284. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771. Version 4*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Ning Xie, Farley Lai, Derek Doran, and Asim Kadav. 2019. Visual entailment: A novel task for fine-grained image understanding. *arXiv preprint arXiv:1901.06706*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, 32.

Yordan Yordanov, Oana-Maria Camburu, Vid Kocijan, and Thomas Lukasiewicz. 2020. Does the objective matter? Comparing training objectives for pronoun resolution. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, November 16–20, 2020*.

Yordan Yordanov, Vid Kocijan, Thomas Lukasiewicz, and Oana-Maria Camburu. 2022. Few-shot out-of-domain transfer learning of natural language explanations in a label-abundant setup. In *Findings of the Association for Computational Linguistics: EMNLP 2022*. Association for Computational Linguistics.

Karen K. Yuen and W. J. Dixon. 1973. The approximate behaviour and performance of the two-sample trimmed t. *Biometrika*, 60(2):369–374.

Umais Zahid, Qinghai Guo, and Zafeirios Fountas. 2023. Sample as you infer: Predictive coding with Langevin dynamics. *arXiv preprint arXiv:2311.13664*.

Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. From recognition to cognition: Visual commonsense reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hongming Zhang, Xinran Zhao, and Yangqiu Song. 2020a. WinoWhy: A deep diagnosis of essential commonsense knowledge for answering Winograd Schema Challenge. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5736–5745. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020b. BERTScore: Evaluating text generation with BERT. In *Proceedings of the 8th International Conference on Learning Representations*. OpenReview.net.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 649–657. MIT Press.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451. Association for Computational Linguistics.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. 2018. Gender bias in coreference resolution: Evaluation and debiasing methods. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 15–20. Association for Computational Linguistics.

Xinyan Zhao and V.G.Vinod Vydiswaran. 2021. LIREx: Augmenting language inference with relevant explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14532–14539.

Qihuang Zhong, Liang Ding, Yibing Zhan, Yu Qiao, Yonggang Wen, Li Shen, Juhua Liu, Baosheng Yu, Bo Du, Yixin Chen, et al. 2022. Toward efficient language model pretraining and downstream adaptation via self-evolution: A case study on superglue. *arXiv preprint arXiv:2212.01853*.

Yangqiaoyu Zhou, Yiming Zhang, and Chenhao Tan. 2023. FLamE: Few-shot learning from natural language explanations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6743–6763. Association for Computational Linguistics.

Barret Zoph. 2022. Designing effective sparse expert models. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1044–1044.