ENCODER-DECODER

NEURAL NETWORKS

Nal Kalchbrenner

Encoder-Decoder Neural Networks

Nal Kalchbrenner
New College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2017

To my parents Metka and Bojan

Acknowledgements

I would like to express my gratitude to my advisor Phil Blunsom who upheld my pursuit of neural networks from the start of my DPhil in 2012 when neural networks, especially those applied to language, felt like a wild gamble and a dream. I would like to thank those professors who accompanied me through the first two years of my DPhil: Bob Coecke, Stephen Pulman, Nando de Freitas. I am greatly indebted to the intellectual masters who have instilled in me the fundamentals of the science of learning and cognition in the broadest sense during my earlier years of study and have shaped my thinking about the matter ever since: Grisha Mints, Solomon Feferman, Patrick Suppes, Michael Ramscar, Michael Friedman, Dagfinn Føllesdal, Ronald de Wolf, Dick de Jongh, Johan van Benthem, Jay McClelland. I would like to sincerely thank those figures whom I have encountered over the past five years on various occasions around the world, at conferences and during internships, and who have been a special source of scientific and technological inspiration: Richard Socher, Ilya Sutskever, Wolfgang Macherey, Oriol Vinyals, Alex Graves, Jürgen Schmidhuber, Sepp Hochreiter, Taco Cohen, Geoffrey Hinton, Wojciech Zaremba and Elon Musk. I would like to thank in particular Tomas Mikolov for releasing his `rnnlm` code implemented in `C++` that has been a vast source of learning at the beginning of my DPhil. I am indebted to DeepMind for accepting me into their family as a research scientist early on in my DPhil. Sincere thanks go to Demis Hassabis, Mustafa Suleyman, Shane Legg,

Koray Kavukcuoglu and Ivo Danihelka. I would like to thank my close friends from my undegraduate and graduate times and from DeepMind for being there for me in science and life throughout these years: Anuk Arudpragrasam, Benjamin Hersh, Nathaniel Thomas, Dāvis Ozols, Fabio Zanasi, Giovanni Cinà, Biagio Giugliano, Lasse Espeholt, Michelle Yeo, Karen Simonyan, Aäron van den Oord, Sander Dieleman. Last, words fail to convey the gratitude that I feel towards my parents, Metka and Bojan, and my siblings, Lana, Gala and Boris, who have done the impossible to make this possible.

Abstract

This thesis introduces the concept of an *encoder-decoder neural network* and develops architectures for the construction of such networks. Encoder-decoder neural networks are probabilistic conditional generative models of high-dimensional structured items such as natural language utterances and natural images. Encoder-decoder neural networks estimate a probability distribution over structured items belonging to a target set conditioned on structured items belonging to a source set. The distribution over structured items is factorized into a product of tractable conditional distributions over individual elements that compose the items. The networks estimate these conditional factors explicitly.

We develop encoder-decoder neural networks for core tasks in natural language processing and natural image and video modelling. In Part I, we tackle the problem of sentence modelling and develop deep convolutional encoders to classify sentences; we extend these encoders to models of discourse. In Part II, we go beyond encoders to study the longstanding problem of translating from one human language to another. We lay the foundations of neural machine translation, a novel approach that views the entire translation process as a single encoder-decoder neural network. We propose a beam search procedure to search over the outputs of the decoder to produce a likely translation in the target language. Besides known recurrent decoders, we also propose a decoder architecture based solely on convolutional layers. Since the publication of these new foundations

for machine translation in 2013, encoder-decoder translation models have been richly developed and have displaced traditional translation systems both in academic research and in large-scale industrial deployment. In services such as Google Translate these models process in the order of 10^9 translation queries a day. In Part III, we shift from the linguistic domain to the visual one to study distributions over natural images and videos. We describe two- and three- dimensional recurrent and convolutional decoder architectures and address the longstanding problem of learning a tractable distribution over high-dimensional natural images and videos, where the likely samples from the distribution are visually coherent.

The empirical validation of encoder-decoder neural networks as state-of-the-art models of tasks ranging from machine translation to video prediction has a two-fold significance. On the one hand, it validates the notions of assigning probabilities to sentences or images and of learning a distribution over a natural language or a domain of natural images; it shows that a probabilistic principle of compositionality, whereby a high-dimensional item is composed from individual elements at the encoder side and whereby a corresponding item is decomposed into conditional factors over individual elements at the decoder side, is a general method for modelling cognition involving high-dimensional items; and it suggests that the relations between the elements are best learnt in an end-to-end fashion as non-linear functions in distributed space. On the other hand, the empirical success of the networks on the tasks characterizes the underlying cognitive processes themselves: a cognitive process as complex as translating from one language to another that takes a human a few seconds to perform correctly can be accurately modelled via a learnt non-linear deterministic function of distributed vectors in high-dimensional space.

Contents

List of Figures

Chapter 1

Framework

This thesis presents a framework for the computational modelling of human natural language and other high-dimensional tensors. The framework covers two aspects of natural language: the first is language *comprehension* and the second is language *generation*. Each of these aspects is viewed as a process or function modelled as a series of non-linear transformations, called *neural network layers*, applied to a set of inputs to produce a set of outputs. The outputs of intermediate transformations are high-dimensional real-valued vectors. These vectors are *distributed representations* of the linguistic content at that stage in the process. The entire function is a neural network and is approximated directly from pairs of inputs and outputs as they occur in empirical data. The same framework extends by analogy also to items other than natural language sequences, such as natural images and videos. In this overview chapter we expand on the fundamentals of the framework and present a setting in which to place the remaining chapters of the thesis.

1.1 Aspects of Natural Language

We start by defining the two central aspects of natural language and corresponding tasks that the framework aims at modelling.

1.1.1 Natural Language Comprehension

A basic form of natural language comprehension is the classification of linguistic utterances into categories. Example categories are the positive or negative sentiment of an utterance, or a topic that the utterance is about. Given an utterance $\mathbf{s} = s_0, ..., s_n$ of n tokens, e.g. words or characters, and a possible category y, a comprehension task has the following form:

$$p(y|\mathbf{s}) = \pi(f(s_0, ..., s_n)) \tag{1.1}$$

where f is the function that processes the utterance and computes a distributed representation for it, while π maps the representation to a probability distribution over the categories y. For reasons that will become clear below, we shall call such a function f an *encoder*.

1.1.1.1 Multi-Utterance Comprehension

Comprehension extends beyond an isolated utterance. An extension of the previous task is the classification of an utterance within the context of a paragraph or a dialogue. In this case the category that an utterance belongs to is influenced by past context. We can frame the multi-utterance comprehension task as follows:

$$p(y|\mathbf{s}_k, \mathbf{s}_{<k}, \mathbf{y}_{<k}) = \pi(f(\mathbf{s}_k, \mathbf{s}_{<k}, \mathbf{y}_{<k})) \tag{1.2}$$

where $\mathbf{s}_{<k}$ are utterances from past context and, optionally, $\mathbf{y}_{<k}$ are categories of the past utterances. The function f needs to encode not just the current utterance $\mathbf{s}_k$ but also previous utterances and it needs a form that can capture such context. Part I of the thesis is dedicated to single-utterance and multi-utterance comprehension (Ch. 2 and Ch. 3).

1.1.2 Natural Language Generation

The second aspect that is core to the framework is the generation of natural language utterances. Generation of a novel, structured item is a complex task that requires intelligent decision making and an understanding of the generated item. The generation process in our framework is defined as a generalization of the comprehension process. Instead of producing an individual category, the process produces a first token, then a second one conditioned on the first, and it continues in this way until it produces a token that signals the end of the generation process. A generation process estimates a distribution over language utterances $\mathbf{t} = t_0, ..., t_m$ as follows:

$$p(\mathbf{t}) = \prod_i p(t_i|\mathbf{t}_{<i}) \tag{1.3}$$

A generation process that follows a sequence of estimated conditional factors as in Eq. 1.3 is called *autoregressive*. Although other ways of generating natural language utterances are possible, this framework focuses on the described process that turns out to be robust and yield by far the best results for natural language generation. When $\mathbf{t}$ are utterances of natural language, the function estimating $p(\mathbf{t})$ is called a *language model*. When such a function is modeled by a neural network as is the case for all functions in the present framework, it is called a *neural language model*.

1.1.2.1 Neural Machine Translation

Machine translation, the ability of a computer to translate from one human language to another, is a problem of scientific and social importance. A solution can yield insights into the nature of human languages and make communication easy across borders. A central tenet of the present framework and a core contribution of the proposed thesis is that the process of translating from one human language to another can be entirely modeled as an autoregressive function defined by a series of neural

network layers. Given an utterance $\mathbf{s} = s_1, ..., s_m$ of m tokens in a source language $\mathbf{S}$ and an utterance $\mathbf{t} = t_1, ..., t_n$ of n tokens in a target language $\mathbf{T}$, a machine translation model estimates a distribution over the target language:

$$p(\mathbf{t}|\mathbf{s}) = \prod_i p(t_i|\mathbf{t}_{<i}, \mathbf{s}) \tag{1.4}$$

Each component $p(t_i|\mathbf{t}_{<i}, \mathbf{s})$ is modelled as a neural network function g:

$$p(t_i|\mathbf{t}_{<i}, \mathbf{s}) = \pi(g(t_i|\mathbf{t}_{<i}, f(\mathbf{s}))) \tag{1.5}$$

The function f is the encoder for the source utterance $\mathbf{s}$ yielding a distributed representation $f(\mathbf{s})$ of the source. The function g is called the *decoder* and takes the distributed representation of the source as well as previous context words from the target to create a distributed representation and by way of π a distribution over the next word in the target. The combination of g and f that compute representations to estimate the distribution $p(\mathbf{t}|\mathbf{s})$ for a pair of languages $\mathbf{S}$ and $\mathbf{T}$ is called a *neural translation model*. A direct estimation of the joint distribution $p(\mathbf{t}|\mathbf{s})$ is generally not tractable due to the rich structure of the item $\mathbf{t}$ and it requires factorizing the distribution into a product of conditional factors over lower-dimensional elements as stated in Eq. 1.4.

A neural translation model is an instance of an *encoder-decoder* neural network. Part II of this thesis is dedicated to encoder-decoder networks and neural machine translation (Ch. 4 and Ch. 5).

1.2 Architectures for Encoders and Decoders

The previous sections formulate comprehension and generation tasks as probabilistic and, in the case of structured outputs, autoregressive models. The main subsequent

challenge is to define an appropriate form for the function underlying the model, that is, an appropriate architecture for the neural network. In this section we describe the major types of architectures for individual encoders and decoders and how these architectures are combined into an encoder-decoder neural network.

1.2.1 Distributed Representations

Linguistic content during a comprehension or generation process is encoded as a distributed representation. Distributed representations are vectors of many dimensions where content is represented by a subset of these continuous-valued dimensions. For a word such as *apple* there may exist multiple concepts such as *fruit*, *red*, *green*, *sweet* that are related to *apple* in subtly different ways. In the corresponding distributed representation of the concept *apple*, each other concept that is related to *apple* is encoded by multiple such values in the vector; in turn, each dimension of the vector may represent the relatedness of more than one concept. This contrasts a distributed representation with a discrete, symbolic one and makes it possible to represent a large number of concepts via a small number of dimensions in a noise-tolerant and over-complete way. At the core of encoder-decoder neural networks as models of linguistic comprehension and generation processes is the hypothesis that such processes are usefully modelled as transformations of distributed representations of linguistic content.

1.2.2 Token Embeddings

The inputs to the neural networks are utterances that come as a sequence of tokens $\mathbf{u} = u_1, ..., u_n$. The tokens that we use are either words or characters. Words have the advantage that the encoded sequences are shorter and words are semantically more meaningful units than characters; but words lead to large vocabularies and the presence of unknown words not seen during training. By contrast, characters

	Encoder	*Decoder*
(Dilated) Convolutional	Whole kernel	Masked kernel
Recurrent	Bidirectional	Unidirectional
Max/Self-Att Pooling	Whole sequence	Past sequence

Figure 1.1: Types of layers applicable in neural encoders and decoders.

	Rec	*Conv*	*Dilated Conv*	*Pool*
Receptive Field Growth Rate	∞	$O(l)$	$O(2^l)$	∞
Sequential Steps per Layer (training / encoder inference)	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Sequential Steps per Layer (decoder inference)	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Figure 1.2: Properties of encoders and decoders with different types of processing layers.

yield longer sequences and make both long-range dependencies and the extraction of semantic units harder; they however give models that have open vocabularies and are fully general (see Ch. 5 for more discussion of token types).

When used as inputs to encoders or decoders, words and characters are mapped to arbitrary integers. The latter in turn are mapped to *learnt embeddings* $\mathbf{e}(u_i)$ via a lookup table; these embeddings correspond to the learnt distributed representations for the tokens. Once we obtain the embeddings for the tokens we concatenate them together into a *sentence matrix* $\mathbf{e}(\mathbf{u})$ with dimension $n \times d$ where n is the length of the utterance and d the dimension of the embeddings (Sect. 2.2).

1.2.3 Encoder Architectures

The individual token embeddings in the sentence matrix need to be processed to extract phrase-level and sentence-level representations. There are three types of processing layers that can be used in encoders, where the utterance u is wholly available during training and inference (Fig. 1.1)

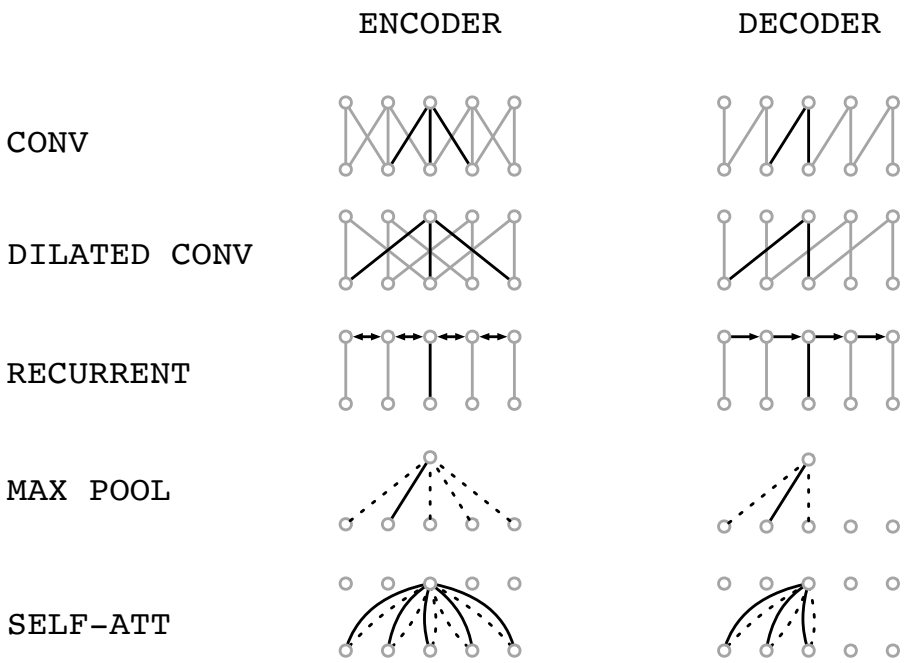

Figure 1.3: Diagram of layer types in neural encoders and decoders.

1.2.3.1 Convolutional Layers

The first processing method applies a convolution with a kernel of size k to the sentence matrix along the dimension of the tokens. Each window of k tokens is transformed via the kernel into an initial representation for the k tokens (Sect. 2.2). This is followed by a point-wise non-linear operation and can be embedded within residual blocks [He et al., 2016]. Encoders made of l convolutional layers have the following two properties. First, the total receptive field $r(l, k)$ grows linearly in the number of layers l:

$$r(l, k) = k + (l - 1)(k - 1) \tag{1.6}$$

Second, for each layer, the convolution can be applied to all tokens simultaneously; thus, the *number* of distinct matrix multiplication operations needed to process a whole sequence of tokens is just one and it is independent of the length of the sequence. This allows for parallelization and makes convolutional encoders particularly efficient

to compute on hardware such as GPUs both during training and inference (Fig. 1.2).

1.2.3.2 Dilated Convolutional Layers

The representation computed by a convolutional layer has a receptive field ranging over a finite number of tokens that is the size of the convolutional kernels. Stacking convolutional layers increases the receptive field as a linear function of the kernel size and the number of layers. This linear growth rate might be insufficient to include in the receptive field tokens that are far apart from each other. One mechanism to address this limitation is to use exponentially increasing factors of dilation in the convolutional kernels from one layer to the next (Sect. 5.3.3). The dilation factors may follow a series such as 1, 2, 4, 8, 16, etc. This ensures that the receptive field grows exponentially in the number of layers l and that no tokens within the size of the receptive field are left out. Dilated convolutional encoders thus have the following two properties. First, the total receptive has an exponential growth rate in terms of the number of layers. Secondly, since the underlying operation is still a convolution, the operation requires a single matrix multiplication to compute the output of a layer for the entire sequence; the same parallelization properties as in the vanilla convolution are kept in this case too.

1.2.3.3 Bidirectional Recurrent Layers

Both convolutional and dilated convolutional encoders ultimately have a principally bounded receptive field. An unbounded receptive field can be achieved with a recurrent neural network (RNN) within a single layer. At each step the RNN is given as input one of the tokens and combines it with the previous state to produce an output and the next state (Sect. 4.2). The state of the RNN can carry information for an unbounded number of steps. Since for encoders the entire sequence of tokens is always available, it is possible to encode the sequence both left-to-right and right-to-left

using a bidirectional RNN. This layer has the following two properties. On the one hand, the receptive field is immediately unbounded even after a single RNN layer. However, on the other hand, the computation cannot be parallelized as in the case of convolutional encoders. To encode a sequence of n tokens with an RNN, one needs to apply $O(n)$ matrix multiplication operations in sequence.

1.2.3.4 Pooling Layers

The final type of layer that is admissible in an encoder is a layer that simply combines information along the sequence of tokens using a limited parametrization. There are multiple ways of combining such information. Simple types of pooling involve point-wise addition or maximum of the token embeddings in the sentence matrix (Sect. 2.3). These simple types of pooling layers can be useful in classification tasks for the purposes of creating a fixed size representation from the variable-size sentence matrix.

Another type of pooling is self-attentional pooling [Vaswani et al., 2017]. For each token embedding $\mathbf{e_i}$, one first computes a key vector $\mathbf{c}_i$, a query vector $\mathbf{q}_i$ and a value vector $\mathbf{v}_i$. For each other token u_j, the key and query vectors are combined to obtain a similarity scalar $z_{i,j} = h(\mathbf{c}_i, \mathbf{q}_j)$ and a normalized corresponding scalar $a_{i,j} = \frac{e^{z_{i,j}}}{\sum_k^n e^{z_{i,k}}}$. The resulting pooled representation $\mathbf{a}_i$ is obtained as follows:

$$\mathbf{a}_i = \sum_j^n a_{i,j} \mathbf{v}_j \tag{1.7}$$

This type of pooling constructs a different representation for each token in the sequence. Multiple such layers can be stacked. This operation has the following properties. First, like for RNNs, the receptive field of self-attentional pooling is unbounded and can cover the whole sequence at once. Second, the computation can be parallelized along the sequence. But the computation is quadratic in sequence length, since

the key for each position is compared with the values at all other positions. The main types of encoder layers and their properties are summarized in Fig. 1.1 and Fig. 1.2 and are illustrated in Fig. 1.3

1.2.4 Decoder Architectures

Instead of encoding a sentence matrix, decoders generate a sequence of output tokens. The generation of each token is conditioned on previous tokens in an autoregressive manner (Eq. 1.4). For this reason the representation that generates a given token can only depend on past representations in the decoder, not on future representations pertaining to tokens that are yet to be generated. This restriction induces particular architectural modifications to the types of layers that can be used in decoders.

1.2.4.1 Masked Convolutional Layers

Convolutional layers in decoders make use of *masked kernels*. A standard kernel of size k processes a window of $(k-1)/2$ past and $(k-1)/2$ future tokens and the current token. A masked (also called causal) kernel of size k, by contrast, only processes a window of k past tokens. This is often implemented by setting to zero the weights that connect to future tokens in the window. Since masked convolutional layers are still just convolutions, they have the same efficiency properties during training as standard convolutional layers in encoders. But during inference, due to the inevitably sequential nature of decoders, masked convolutional layers need to be processed one window of past inputs at a time.

1.2.4.2 Masked Dilated Convolutional Layers

Dilation can also be applied to masked kernels. The receptive field grows exponentially only into the past. The same efficiency properties apply as for masked convolutional kernels.

1.2.4.3 Unidirectional Recurrent Layers

Recurrent layers in encoders have a bidirectional structure in order to capture both the past and the future context. In decoders, by contrast, due to the left-to-right generation order, decoders are bound to only capture the past structure. This is simply achieved dropping one side of the bidirectional RNN and using a standard unidirectional recurrent layer. Both training and inference in recurrent decoders naturally requires $O(n)$ sequential matrix multiplication operations.

1.2.4.4 Masked Pooling Layers

Just as convolutional and recurrent layers can be used to capture past context only, so can pooling layers be restricted to past information. In particular, self-attention layers can be masked to attend to past representations only; this can be obtained, for instance, by setting the logit scalars $z_{i,j} = -\infty$ for the values of future tokens. During training one can still compute masked self-attention layers at once for an entire sequence. Like for other layers, decoding is a sequential process that will require $O(n)$ separate matrix multiplication steps. Figure 1.3 illustrates the various types of layers admissible in decoders.

1.2.5 Encoder-Decoder Neural Networks

We have described various types of layers that can be used to build individual encoders and decoders. The combination of the two in an encoder-decoder architecture used for generation may require some care, especially when the input and output sequences have varying lengths. Even in comprehension tasks where one classifies the input utterance into a single category it is necessary to obtain a fixed-size representation from the encoder that captures the variable-length input utterance. In this subsection we look at ways of addressing these issues.

1.2.5.1 Fixed-Length Encoders

In a comprehension task the variable length utterance representation needs to be encoded into a fixed-size representation that is used to predict the class. The fixed size for the representation can be achieved in one of two ways. The first is by consistently selecting a specific part of the variable-length representation produced by the encoder. For example, one can encode an utterance with a bidirectional RNN and use the first and last states of the RNN as the fixed size representation for the entire utterance. The second way is to apply a pooling operation across the length of the encoder representation. A max-pool operator following one or more convolutional layers can be used for this (Ch. 2). Another option is to use a self-attentional pooling operator. Just like max-pool, the weighted sum provided by the operator produces a fixed size representation independently of the size of the encoder representation.

1.2.5.2 Fixed-Length Encoder-Decoders

The length problem is even more pronounced in generation tasks since input and output sequences can vary in length. We can use the fixed-size encodings from the previous subsection to provide a solution to this issue too. That is, after the fixed-size vector is computed by way of one of the two methods from the previous subsection, the vector is then used to bias the decoder (see Ch. 3 for a specific construction). The vector is transformed and added to one or more of the decoder layers, be these convolutional, recurrent or based on pooling.

1.2.5.3 Variable-Length Encoder-Decoders

Fixed-length encodings may require remembering and compressing a particularly large amount of information into a single vector. For some tasks such as machine translation keeping a variable length and resolution preserving encoding of the input utterance, i.e. learning an encoded representation for each token in the utterance as opposed

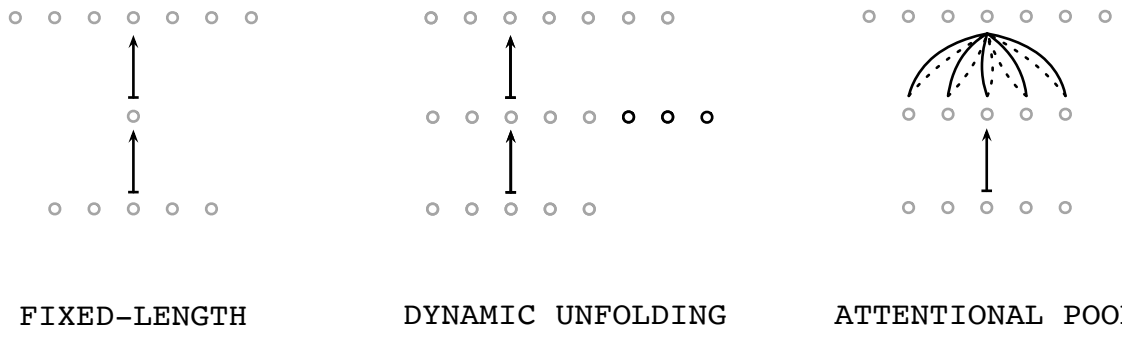

Figure 1.4: Types of connection in encoder-decoder networks.

to a single one for the utterance as a whole, can prove beneficial. We describe two methods to preserve the variable lengths of the representations in encoder-decoders.

The first method is called *dynamic unfolding* (see Ch. 5). Dynamic unfolding involves three components. The first is a simple (linear) estimate $|\hat{\mathbf{t}}| = d(|\mathbf{s}|)$ of a length $|\mathbf{t}|$ for the output sequence as a function of the length $|\mathbf{s}|$ of the input sequence such that the estimated length $|\hat{\mathbf{t}}|$ is a tight upper bound on the length of *most* output sequences that have input sequences of the given length $|\mathbf{s}|$. One way is to choose the empirical mean for $d(|\mathbf{s}|)$ and then choose a length that is a few standard deviations greater than the mean. The second part is an architecture that encodes the input sequence of length $|\mathbf{s}|$ into a representation of the estimated length $|\hat{\mathbf{t}}|$. The third part is the decoder for $\mathbf{t}$ that simply takes one slice of the encoded representation at each step as an additional input and produces the target sequence $\mathbf{t}$ one step at a time until it generates an end-of-sequence token. If the decoders spills over the length estimated by the encoder, it simply takes as additional input a vector of zeros (or a learnt bias) as the conditioning slice until the end-of-sequence token is produced. This method comes closest to the standard practice of stacking neural network layers for the case where the sizes of the layers differ.

The second method is to simply use a pooling operator such as attentional pooling [Bahdanau et al., 2014]. In the definition of self-attention layers we see that the

number of produced representations $\mathbf{a}_{i,j}$ is independent of the number of value vectors that are attended to. So it is easy to connect using attentional pooling encoders and decoders that operate on sequences with different lengths. As for self-attention, vanilla attentional pooling performs n dot products per decoding step, which for large n can turn out to be computationally expensive. Figure 1.4 represents the various methods of connecting encoders and decoders.

1.2.6 Output Layers

Finally, for each position in the output sequence the final layer produces a probability distribution over as many outputs as there are possible categories or tokens. A softmax layer is applied to the logits z_i in order to ensure that they sum to 1.0:

$$p(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{1.8}$$

Softmax layers over discrete output spaces are used throughout the current thesis (see e.g. Sect. 4.2 and Sect. 6.3.2).

1.3 Approximating the Function

We have surveyed different types of layers and mechanisms that can be used to construct neural networks that have an encoder structure for comprehension tasks and an encoder-decoder structure for generation tasks. The models are trained to maximize the likelihood of the data and use a cross-entropy loss. Training proceeds by back-propagation [Rumelhart et al., 1986a]: gradients of the cross-entropy loss with respect to the parameters in the transformations are efficiently computed by a multivariate application of the chain rule. Stochastic gradient descent is used to optimize the loss.

1.3.1 Beam Search for Decoders

Encoder-decoder networks produce a distribution over sequences of target tokens. One may extract a target sequence simply by sampling one token at each step from each conditional factor. Another simple way is to apply a *greedy decoding* algorithm: simply choose the most likely token at each step by maximizing each factor individually. In general these two methods lead to suboptimal results as outputs that are more likely than the ones obtained by sampling or greedy decoding are not considered. For tasks such as machine translation, it is important to find a high likelihood sample:

$$\mathbf{t}^* = \arg\max_{\mathbf{t}} p(\mathbf{t}|\mathbf{s}) \tag{1.9}$$

To achieve this it is possible to apply a *beam search* algorithm to the decoder (Sect. 4.5.3 and Sect. 5.6). One always keeps at most c candidate translations. Starting from the empty translation, one starts by producing the c tokens that are most likely to be the first token of the output, ranked by their log-likelihood. For each of the c tokens, one decodes the c most likely second tokens, ranked by the total log-likelihood, to obtain c^2 candidates. The entire list of two-token candidates is reranked and only the first c candidates are kept before proceeding to the third token. When an end-of-sequence token is encountered after pruning to c candidates for a given step, that candidate is set aside into a final list and the number of candidates kept in the search is reduced by 1. When the final list contains all c candidates, the most likely one is selected as the chosen translation.

This completes our high-level overview of encoder and encoder-decoder architectures.

1.4 Beyond Natural Language

In Part III of the thesis we extend the above ideas to the visual domain of images and videos (Ch. 6 and Ch. 7). We use the presented methods to learn a distribution over images and videos and generate samples from the distribution. Tokens are now pixels, or RGB color channels of pixels. Images are treated as two-dimensional sequences of pixels; videos are three-dimensional sequences of pixels. Just like characters, pixel values are treated as discrete tokens. This gives rise to autoregressive models of visual data and generalizes the framework: autoregressive decoders conditioned on encoded information are powerful and general estimators of data distributions across both linguistic and visual domains.

1.5 General Outline

The thesis is divided in three parts. The first part describes the construction of specific neural encoders for sentences and discourse and demonstrates then state-of-the-art results for tasks such as sentiment analysis and dialogue act classification. This part is based on the following two publications:

- Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom. *A Convolutional Neural Network for Modelling Sentences.* Association of Computational Linguistics, 2014

- Nal Kalchbrenner, Phil Blunsom. *Recurrent Convolutional Neural Networks for Discourse Compositionality.* Association of Computational Linguistics, Workshop, 2013

The second part deals with the foundations and architectures for encoder-decoders and neural machine translation. This part upholds our claim that language processes such as machine translation can be modelled by a series of non-linear transformations

of high-dimensional distributed representations. Translation models are studied both at the word and at the character level with state-of-the-art results. Machine translation and many other language processing tasks are nowadays modelled as encoder-decoder networks (e.g. [Wu et al., 2016a]). This part relies on the following two publications:

- Nal Kalchbrenner, Phil Blunsom. *Recurrent Continuous Translation Models.* Empirical Methods in Natural Language Processing, 2013
- Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, Koray Kavukcuoglu. *Neural Machine Translation in Linear Time.* 2016

Finally, the third part extends the framework to visual data. It shows that autoregressive decoders over pixels viewed as discrete variables obtain state-of-the-art performance in terms of the measured log-likelihood. It also shows that, despite optimizing simply for the likelihood of natural images and videos, these models are able to generate extremely precise and crisp images. The final part of the thesis relies on the following two publications:

- Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu. *Pixel Recurrent Neural Networks.* International Conference on Machine Learning, 2016. **Best Paper Award**.
- Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, Koray Kavukcuoglu. *Video Pixel Networks.* International Conference on Machine Learning, 2017.

Part I

Neural Encoders of Natural Language

Chapter 2

Convolutional Neural Networks for Modelling Sentences

2.1 Summary

We begin in Part I by studying neural encoders of linguistic utterances and their architectures. We describe a convolutional architecture dubbed the Dynamic Convolutional Neural Network (DCNN) for encoding sentences. The network uses Dynamic k-Max Pooling, a global pooling operation over linear sequences. The network handles input sentences of varying length and induces a feature graph over the sentence that is capable of explicitly capturing short and long-range relations. The network does not rely on a parse tree and is easily applicable to any language. We test the DCNN in four experiments: small scale binary and multi-class sentiment prediction, six-way question classification and Twitter sentiment prediction by distant supervision. The network achieves excellent performance in the first three tasks and a greater than 25% error reduction in the last task with respect to the strongest baseline.

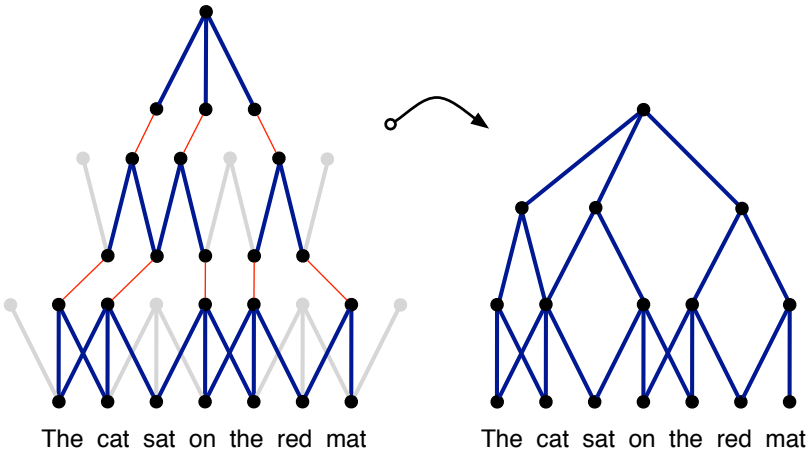

Figure 2.1: Subgraph of a feature graph induced over an input sentence in a Dynamic Convolutional Neural Network. The full induced graph has multiple subgraphs of this kind with a distinct set of edges; subgraphs may merge at different layers. The left diagram emphasises the pooled nodes. The width of the convolutional filters is 3 and 2 respectively. With dynamic pooling, a filter with small width at the higher layers can relate phrases far apart in the input sentence.

2.2 Introduction

The aim of a sentence model is to analyse and represent the semantic content of a sentence for purposes of classification or generation. The sentence modelling problem is at the core of many tasks involving a degree of natural language comprehension. These tasks include sentiment analysis, paraphrase detection, entailment recognition, summarisation, discourse analysis, machine translation, grounded language learning and image retrieval. Since individual sentences are rarely observed or not observed at all, one must represent a sentence in terms of features that depend on the words and short n-grams in the sentence that are frequently observed. The core of a sentence model involves a feature function that defines the process by which the features of the sentence are extracted from the features of the words or n-grams. As we have seen in Ch. 1, a sentence model is equivalent to a neural encoder for variable sized sequences.

Various types of models of meaning have been proposed. Composition based methods have been applied to vector representations of word meaning obtained from

co-occurrence statistics to obtain vectors for longer phrases. In some cases, composition is defined by algebraic operations over word meaning vectors to produce sentence meaning vectors [Erk and Padó, 2008, Mitchell and Lapata, 2008, Mitchell and Lapata, 2010, Turney, 2012, Erk, 2012, Clarke, 2012]. In other cases, a composition function is learned and either tied to particular syntactic relations [Guevara, 2010, Zanzotto et al., 2010] or to particular word types [Baroni and Zamparelli, 2010, Coecke et al., 2010, Grefenstette and Sadrzadeh, 2011, Kartsaklis and Sadrzadeh, 2013, Grefenstette, 2013]. Another approach represents the meaning of sentences by way of automatically extracted logical forms [Zettlemoyer and Collins, 2005].

A central class of models are those based on neural networks. These range from basic neural bag-of-words or bag-of-n-grams models to the more structured recursive neural networks and to time-delay neural networks based on convolutional operations [Collobert and Weston, 2008, Socher et al., 2011b] (see also Ch. 3). Neural sentence models have a number of advantages. They can be trained to obtain generic vectors for words and phrases by predicting, for instance, the contexts in which the words and phrases occur. Through supervised training, neural sentence models can fine-tune these vectors to information that is specific to a certain task. Besides comprising powerful classifiers as part of their architecture, neural sentence models can be used to condition a neural language model to generate sentences word by word [Schwenk, 2012, Mikolov and Zweig, 2012] (Ch. 4 offers an extensive treatment of this).

We define a convolutional neural network architecture and apply it to the semantic modelling of sentences. The network handles input sequences of varying length. The layers in the network interleave one-dimensional convolutional layers and dynamic k-max pooling layers. Dynamic k-max pooling is a generalisation of the max pooling operator. The max pooling operator is a non-linear subsampling function that returns the maximum of a set of values [LeCun et al., 1998a]. The operator is generalised

in two respects. First, k-max pooling over a linear sequence of values returns the subsequence of k maximum values in the sequence, instead of the single maximum value. Secondly, the pooling parameter k can be dynamically chosen by making k a function of other aspects of the network or the input.

The convolutional layers apply one-dimensional filters across each row of features in the sentence matrix. Convolving the same filter with the n-gram at every position in the sentence allows the features to be extracted independently of their position in the sentence. A convolutional layer followed by a dynamic pooling layer and a non-linearity form a feature map. Like in the convolutional networks for object recognition [LeCun et al., 1998a], we enrich the representation in the first layer by computing multiple feature maps with different filters applied to the input sentence. Subsequent layers also have multiple feature maps computed by convolving filters with all the maps from the layer below. The weights at these layers form an order-4 tensor. The resulting architecture is dubbed a Dynamic Convolutional Neural Network.

Multiple layers of convolutional and dynamic pooling operations induce a structured feature graph over the input sentence. Figure 1 illustrates such a graph. Small filters at higher layers can capture syntactic or semantic relations between non-continuous phrases that are far apart in the input sentence. The feature graph induces a hierarchical structure somewhat akin to that in a syntactic parse tree. The structure is not tied to purely syntactic relations and is internal to the neural network.

We experiment with the network in four settings. The first two experiments involve predicting the sentiment of movie reviews [Socher et al., 2013b]. The network outperforms other approaches in both the binary and the multi-class experiments. The third experiment involves the categorisation of questions in six question types in the TREC dataset [Li and Roth, 2002]. The network matches the accuracy of other state-of-the-art methods that are based on large sets of engineered features and hand-coded knowledge resources. The fourth experiment involves predicting the sentiment

of Twitter posts using distant supervision [Go et al., 2009]. The network is trained on 1.6 million tweets labelled automatically according to the emoticon that occurs in them. On the hand-labelled test set, the network achieves a greater than 25% reduction in the prediction error with respect to the strongest unigram and bigram baseline reported in [Go et al., 2009].

The outline of the chapter is as follows. Section 2.3 describes the background to the DCNN including central concepts and related neural sentence models. Section 2.4 defines the relevant operators and the layers of the network. Section 2.5 treats of the induced feature graph and other properties of the network. Section 2.6 discusses the experiments and inspects the learnt feature detectors.

2.3 Background

The layers of the DCNN are formed by a convolution operation followed by a pooling operation. We begin with a review of related neural sentence models. Then we describe the operation of *one-dimensional convolution* and the classical Time-Delay Neural Network (TDNN) [Hinton, 1989, Waibel et al., 1990]. By adding a max pooling layer to the network, the TDNN can be adopted as a sentence model [Collobert and Weston, 2008].

2.3.1 Related Neural Sentence Models

Various neural sentence models have been described. A general class of basic sentence models is that of Neural Bag-of-Words (NBoW) models. These generally consist of a projection layer that maps words, sub-word units or n-grams to high dimensional embeddings; the latter are then combined component-wise with an operation such as summation. The resulting combined vector is classified through one or more fully connected layers.

A model that adopts a more general structure provided by an external parse tree is the Recursive Neural Network (RecNN) [Pollack, 1990, Küchler and Goller, 1996, Socher et al., 2011b, Hermann and Blunsom, 2013]. At every node in the tree the contexts at the left and right children of the node are combined by a classical layer. The weights of the layer are shared across all nodes in the tree. The layer computed at the top node gives a representation for the sentence. The Recurrent Neural Network (RNN) is a special case of the recursive network where the structure that is followed is a simple linear chain [Gers and Schmidhuber, 2001, Mikolov et al., 2011]. The RNN is primarily used as a language model, but may also be viewed as a sentence model with a linear structure. The layer computed at the last word represents the sentence.

Finally, a further class of neural sentence models is based on the convolution operation and the TDNN architecture [Collobert and Weston, 2008] (Ch. 3). Certain concepts used in these models are central to the DCNN and we describe them next.

2.3.2 Convolution

The *one-dimensional convolution* is an operation between a vector of weights $\mathbf{m} \in \mathbb{R}^m$ and a vector of inputs viewed as a sequence $\mathbf{s} \in \mathbb{R}^s$. The vector $\mathbf{m}$ is the *filter* of the convolution. Concretely, we think of $\mathbf{s}$ as the input sentence and $\mathbf{s}_i \in \mathbb{R}$ is a single feature value associated with the i-th word in the sentence. The idea behind the one-dimensional convolution is to take the dot product of the vector $\mathbf{m}$ with each m-gram in the sentence $\mathbf{s}$ to obtain another sequence $\mathbf{c}$:

$$\mathbf{c}_j = \mathbf{m}^\intercal \mathbf{s}_{j-m+1:j} \tag{2.1}$$

Equation 2.1 gives rise to two types of convolution depending on the range of the index j. The *narrow* type of convolution requires that $s \geq m$ and yields a sequence $\mathbf{c} \in \mathbb{R}^{s-m+1}$ with j ranging from m to s. The *wide* type of convolution does not have

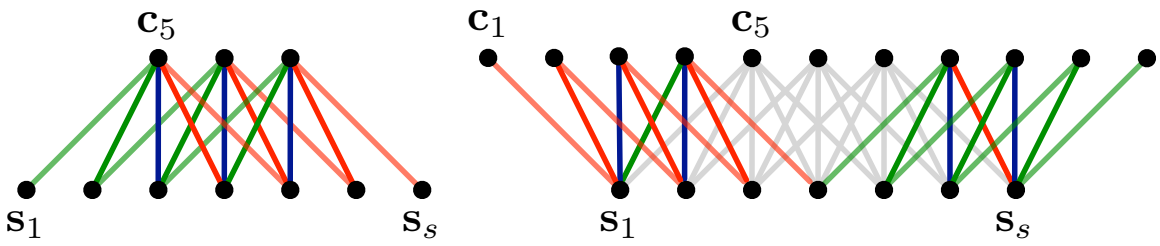

Figure 2.2: Narrow and wide types of convolution. The filter $\mathbf{m}$ has size $m = 5$.

requirements on s or m and yields a sequence $\mathbf{c} \in \mathbb{R}^{s+m-1}$ where the index j ranges from 1 to $s+m-1$. Out-of-range input values $\mathbf{s}_i$ where $i < 1$ or $i > s$ are taken to be zero. The result of the narrow convolution is a subsequence of the result of the wide convolution. The two types of one-dimensional convolution are illustrated in Fig. 2.

The trained weights in the filter $\mathbf{m}$ correspond to a linguistic feature detector that learns to recognise a specific class of n-grams. These n-grams have size $n \leq m$, where m is the width of the filter. Applying the weights $\mathbf{m}$ in a wide convolution has some advantages over applying them in a narrow one. A wide convolution ensures that all weights in the filter reach the entire sentence, including the words at the margins. This is particularly significant when m is set to a relatively large value such as 8 or 10. In addition, a wide convolution guarantees that the application of the filter $\mathbf{m}$ to the input sentence $\mathbf{s}$ always produces a valid non-empty result $\mathbf{c}$, independently of the width m and the sentence length s. We next describe the classical convolutional layer of a TDNN.

2.3.3 Time-Delay Neural Networks

A TDNN convolves a sequence of inputs $\mathbf{s}$ with a set of weights $\mathbf{m}$. As in the TDNN for phoneme recognition [Waibel et al., 1990], the sequence $\mathbf{s}$ is viewed as having a time dimension and the convolution is applied over the time dimension. Each $\mathbf{s}_j$ is often not just a single value, but a vector of d values so that $\mathbf{s} \in \mathbb{R}^{d \times s}$. Likewise, $\mathbf{m}$ is a matrix of weights of size $d \times m$. Each row of $\mathbf{m}$ is convolved with the corresponding

row of $\mathbf{s}$ and the convolution is usually of the narrow type. Multiple convolutional layers may be stacked by taking the resulting sequence $\mathbf{c}$ as input to the next layer.

The Max-TDNN sentence model is based on the architecture of a TDNN [Collobert and Weston, 2008]. In the model, a convolutional layer of the narrow type is applied to the sentence matrix $\mathbf{s}$, where each column corresponds to the feature vector $\mathbf{w}_i \in \mathbb{R}^d$ of a word in the sentence:

$$\mathbf{s} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \dots & \mathbf{w}_s \\ | & | & | \end{bmatrix} \tag{2.2}$$

To address the problem of varying sentence lengths, the Max-TDNN takes the maximum of each row in the resulting matrix $\mathbf{c}$ yielding a vector of d values:

$$\mathbf{c}_{max} = \begin{bmatrix} \max(\mathbf{c}_{1,:}) \\ \vdots \\ \max(\mathbf{c}_{d,:}) \end{bmatrix} \tag{2.3}$$

The aim is to capture the most relevant feature, i.e. the one with the highest value, for each of the d rows of the resulting matrix $\mathbf{c}$. The fixed-sized vector $\mathbf{c}_{max}$ is then used as input to a fully connected layer for classification.

The Max-TDNN model has many desirable properties. It is sensitive to the order of the words in the sentence and it does not depend on external language-specific features such as dependency or constituency parse trees. It also gives largely uniform importance to the signal coming from each of the words in the sentence, with the exception of words at the margins that are considered fewer times in the computation of the narrow convolution. But the model also has some limiting aspects. The range of the feature detectors is limited to the span m of the weights. Increasing m or stacking multiple convolutional layers of the narrow type makes the range of the feature detectors larger; at the same time it also exacerbates the neglect of the

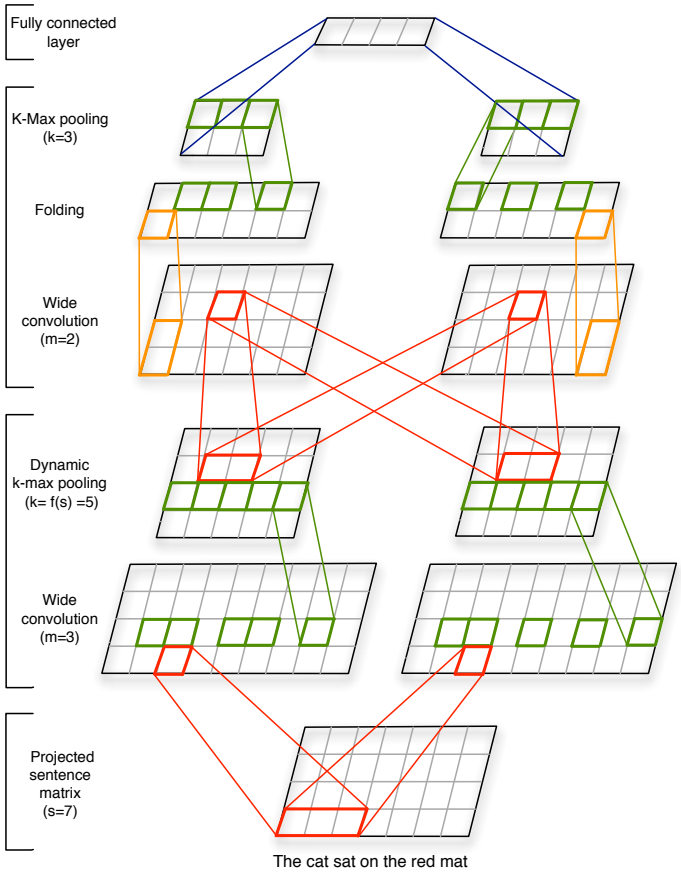

Figure 2.3: A DCNN for the seven word input sentence. Word embeddings have size $d = 4$. The network has two convolutional layers with two feature maps each. The widths of the filters at the two layers are respectively 3 and 2. The (dynamic) k-max pooling layers have values k of 5 and 3.

margins of the sentence and increases the minimum size s of the input sentence required by the convolution. For this reason higher-order and long-range feature detectors cannot be easily incorporated into the model. The max pooling operation has some disadvantages too. It cannot distinguish whether a relevant feature in one of the rows occurs just one or multiple times and it forgets the order in which the features occur. More generally, the pooling factor by which the signal of the matrix is reduced at once corresponds to $s - m + 1$; even for moderate values of s the pooling factor can be excessive. The aim of the next section is to address these limitations while preserving the advantages.

2.4 Convolutional Neural Networks with Dynamic k-Max Pooling

We model sentences using a convolutional architecture that alternates wide convolutional layers with dynamic pooling layers given by *dynamic k-max pooling*. In the network the width of a feature map at an intermediate layer varies depending on the length of the input sentence; the resulting architecture is the Dynamic Convolutional Neural Network. Figure 3 represents a DCNN. We proceed to describe the network in detail.

2.4.1 Wide Convolution

Given an input sentence, to obtain the first layer of the DCNN we take the embedding $\mathbf{w}_i \in \mathbb{R}^d$ for each word in the sentence and construct the sentence matrix $\mathbf{s} \in \mathbb{R}^{d \times s}$ as in Eq. 2.3. The values in the embeddings $\mathbf{w}_i$ are parameters that are optimised during training. A convolutional layer in the network is obtained by convolving a matrix of weights $\mathbf{m} \in \mathbb{R}^{d \times m}$ with the matrix of activations at the layer below. For example, the second layer is obtained by applying a convolution to the sentence matrix $\mathbf{s}$ itself. Dimension d and filter width m are hyper-parameters of the network. We let the operations be *wide* one-dimensional convolutions as described in Sect. 2.3.2. The resulting matrix $\mathbf{c}$ has dimensions $d \times (s + m - 1)$.

2.4.2 k-Max Pooling

We next describe a pooling operation that is a generalisation of the max pooling over the time dimension used in the Max-TDNN sentence model and different from the local max pooling operations applied in a convolutional network for object recognition [LeCun et al., 1998a]. Given a value k and a sequence $\mathbf{p} \in \mathbb{R}^p$ of length $p \geq k$, *k-max pooling* selects the subsequence $\mathbf{p}^k_{max}$ of the k highest values of $\mathbf{p}$. The order of the

values in $\mathbf{p}^k_{max}$ corresponds to their original order in $\mathbf{p}$.

The k-max pooling operation makes it possible to pool the k most active features in $\mathbf{p}$ that may be a number of positions apart; it preserves the order of the features, but is insensitive to their specific positions. It can also discern more finely the number of times the feature is highly activated in $\mathbf{p}$ and the progression by which the high activations of the feature change across $\mathbf{p}$. The k-max pooling operator is applied in the network after the topmost convolutional layer. This guarantees that the input to the fully connected layers is independent of the length of the input sentence. But, as we see next, at intermediate convolutional layers the pooling parameter k is not fixed, but is dynamically selected in order to allow for a smooth extraction of higher-order and longer-range features.

2.4.3 Dynamic k-Max Pooling

A *dynamic k-max pooling* operation is a k-max pooling operation where we let k be a function of the length of the sentence and the depth of the network. Although many functions are possible, we simply model the pooling parameter as follows:

$$k_l = \max(\ k_{top},\ \lceil \frac{L-l}{L} s \rceil\ ) \tag{2.4}$$

where l is the number of the current convolutional layer to which the pooling is applied and L is the total number of convolutional layers in the network; k_{top} is the fixed pooling parameter for the topmost convolutional layer (Sect. 2.4.2). For instance, in a network with three convolutional layers and $k_{top} = 3$, for an input sentence of length $s = 18$, the pooling parameter at the first layer is $k_1 = 12$ and the pooling parameter at the second layer is $k_2 = 6$; the third layer has the fixed pooling parameter $k_3 = k_{top} = 3$. Equation 2.4 is a model of the number of values needed to describe the relevant parts of the progression of an l-th order feature over a sentence

of length s. For an example in sentiment prediction, according to the equation a first order feature such as a positive word occurs *at most* k_1 times in a sentence of length s, whereas a second order feature such as a negated phrase or clause occurs at most k_2 times.

2.4.4 Non-linear Feature Function

After (dynamic) k-max pooling is applied to the result of a convolution, a bias $\mathbf{b} \in \mathbb{R}^d$ and a non-linear function g are applied component-wise to the pooled matrix. There is a single bias value for each row of the pooled matrix.

If we temporarily ignore the pooling layer, we may state how one computes each d-dimensional column a in the matrix $\mathbf{a}$ resulting after the convolutional and non-linear layers. Define $\mathbf{M}$ to be the matrix of diagonals:

$$\mathbf{M} = [\text{diag}(\mathbf{m}_{:,1}), \ldots, \text{diag}(\mathbf{m}_{:,m})] \tag{2.5}$$

where $\mathbf{m}$ are the weights of the d filters of the wide convolution. Then after the first pair of a convolutional and a non-linear layer, each column a in the matrix $\mathbf{a}$ is obtained as follows, for some index j:

$$a = g\left(\mathbf{M}\begin{bmatrix} \mathbf{w}_j \\ \vdots \\ \mathbf{w}_{j+m-1} \end{bmatrix} + \mathbf{b}\right) \tag{2.6}$$

Here a is a column of first order features. Second order features are similarly obtained by applying Eq. 2.6 to a sequence of first order features $a_j, ..., a_{j+m'-1}$ with another weight matrix $\mathbf{M}'$. Barring pooling, Eq. 2.6 represents a core aspect of the feature extraction function and has a rather general form that we return to below. Together with pooling, the feature function induces position invariance and makes the range of higher-order features variable.

2.4.5 Multiple Feature Maps

So far we have described how one applies a wide convolution, a (dynamic) k-max pooling layer and a non-linear function to the input sentence matrix to obtain a first order *feature map*. The three operations can be repeated to yield feature maps of increasing order and a network of increasing depth. We denote a feature map of the i-th order by $\mathbf{F}^i$. As in convolutional networks for object recognition, to increase the number of learnt feature detectors of a certain order, multiple feature maps $\mathbf{F}^i_1, \ldots, \mathbf{F}^i_n$ may be computed in parallel at the same layer. Each feature map $\mathbf{F}^i_j$ is computed by convolving a distinct set of filters arranged in a matrix $\mathbf{m}^i_{j,k}$ with each feature map $\mathbf{F}^{i-1}_k$ of the lower order $i-1$ and summing the results:

$$\mathbf{F}^i_j = \sum_{k=1}^{n} \mathbf{m}^i_{j,k} * \mathbf{F}^{i-1}_k \tag{2.7}$$

where $*$ indicates the wide convolution. The weights $\mathbf{m}^i_{j,k}$ form an order-4 tensor. After the wide convolution, first dynamic k-max pooling and then the non-linear function are applied individually to each map.

2.4.6 Folding

In the formulation of the network so far, feature detectors applied to an individual row of the sentence matrix $\mathbf{s}$ can have many orders and create complex dependencies across the same rows in multiple feature maps. Feature detectors in different rows, however, are independent of each other until the top fully connected layer. Full dependence between different rows could be achieved by making $\mathbf{M}$ in Eq. 2.5 a full matrix instead of a sparse matrix of diagonals. Here we explore a simpler method called *folding* that does not introduce any additional parameters. After a convolutional layer and before (dynamic) k-max pooling, one just sums every two rows in a feature map component-wise. For a map of d rows, folding returns a map of $d/2$ rows, thus halving the size of

the representation. With a folding layer, a feature detector of the i-th order depends now on two rows of feature values in the lower maps of order $i - 1$. This ends the description of the DCNN.

2.5 Properties of the Sentence Model

We describe some of the properties of the sentence model based on the DCNN. We describe the notion of the *feature graph* induced over a sentence by the succession of convolutional and pooling layers. We briefly relate the properties to those of other neural sentence models.

2.5.1 Word and n-Gram Order

One of the basic properties is sensitivity to the order of the words in the input sentence. For most applications and in order to learn fine-grained feature detectors, it is beneficial for a model to be able to discriminate whether a specific n-gram occurs in the input. Likewise, it is beneficial for a model to be able to tell the *relative* position of the most relevant n-grams. The network is designed to capture these two aspects. The filters $\mathbf{m}$ of the wide convolution in the first layer can learn to recognise specific n-grams that have size less or equal to the filter width m; as we see in the experiments, m in the first layer is often set to a relatively large value such as 10. The subsequence of n-grams extracted by the generalised pooling operation induces invariance to absolute positions, but maintains their order and relative positions.

As regards the other neural sentence models, the class of NBoW models is by definition insensitive to word order. A sentence model based on a recurrent neural network is sensitive to word order, but it has a bias towards the latest words that it takes as input [Mikolov et al., 2011]. This gives the RNN excellent performance at language modelling, but it is suboptimal for remembering at once the n-grams

further back in the input sentence. Similarly, a recursive neural network is sensitive to word order but has a bias towards the topmost nodes in the tree; shallower trees mitigate this effect to some extent [Socher et al., 2013a]. As seen in Sect. 2.3.3, the Max-TDNN is sensitive to word order, but max pooling only picks out a single n-gram feature in each row of the sentence matrix.

2.5.2 Induced Feature Graph

Some sentence models use internal or external structure to compute the representation for the input sentence. In a DCNN, the convolution and pooling layers induce an internal feature graph over the input. A node from a layer is connected to a node from the next higher layer if the lower node is involved in the convolution that computes the value of the higher node. Nodes that are not selected by the pooling operation at a layer are dropped from the graph. After the last pooling layer, the remaining nodes connect to a single topmost root. The induced graph is a connected, directed acyclic graph with weighted edges and a root node; two equivalent representations of an induced graph are given in Fig. 2.1. In a DCNN without folding layers, each of the d rows of the sentence matrix induces a subgraph that joins the other subgraphs only at the root node. Each subgraph may have a different shape that reflects the kind of relations that are detected in that subgraph. The effect of folding layers is to join pairs of subgraphs at lower layers before the top root node.

Convolutional networks for object recognition also induce a feature graph over the input image. What makes the feature graph of a DCNN peculiar is the global range of the pooling operations. The (dynamic) k-max pooling operator can draw together features that correspond to words that are many positions apart in the sentence. Higher-order features have highly variable ranges that can be either short and focused or global and long as the input sentence. Likewise, the edges of a subgraph in the induced graph reflect these varying ranges. The subgraphs can either be localised to

one or more parts of the sentence or spread more widely across the sentence. This structure is internal to the network and is defined by the forward propagation of the input through the network.

Of the other sentence models, the NBoW is a shallow model and the RNN has a linear chain structure. The subgraphs induced in the Max-TDNN model have a single fixed-range feature obtained through max pooling. The recursive neural network follows the structure of an external parse tree. Features of variable range are computed at each node of the tree combining one or more of the children of the tree. Unlike in a DCNN, where one learns a clear hierarchy of feature orders, in a RecNN low order features like those of single words can be directly combined with higher order features computed from entire clauses. A DCNN generalises many of the structural aspects of a RecNN. The feature extraction function as stated in Eq. 2.6 has a more general form than that in a RecNN, where the value of m is generally 2. Likewise, the induced graph structure in a DCNN is more general than a parse tree in that it is not limited to syntactically dictated phrases; the graph structure can capture short or long-range semantic relations between words that do not necessarily correspond to the syntactic relations in a parse tree. The DCNN has internal input-dependent structure and does not rely on externally provided parse trees, which makes the DCNN directly applicable to hard-to-parse sentences such as tweets and to sentences from any language.

2.6 Experiments

We test the network on four different experiments. We begin by specifying aspects of the implementation and the training of the network. We then relate the results of the experiments and we inspect the learnt feature detectors.

Classifier	Fine-grained (%)	Binary (%)
NB	41.0	81.8
BiNB	41.9	83.1
SVM	40.7	79.4
RecNTN	45.7	85.4
Max-TDNN	37.4	77.1
NBoW	42.4	80.5
DCNN	**48.5**	**86.8**

Table 2.1: Accuracy of sentiment prediction in the movie reviews dataset. The first four results are reported from [Socher et al., 2013b]. The baselines NB and BiNB are Naive Bayes classifiers with, respectively, unigram features and unigram and bigram features. SVM is a support vector machine with unigram and bigram features. RecNTN is a recursive neural network with a tensor-based feature function, which relies on external structural features given by a parse tree and performs best among the RecNNs.

2.6.1 Training

In each of the experiments, the top layer of the network has a fully connected layer followed by a softmax non-linearity that predicts the probability distribution over classes given the input sentence. The network is trained to minimise the cross-entropy of the predicted and true distributions; the objective includes an L_2 regularisation term over the parameters. The set of parameters comprises the word embeddings, the filter weights and the weights from the fully connected layers. The network is trained with mini-batches by backpropagation and the gradient-based optimisation is performed using the Adagrad update rule [Duchi et al., 2011]. Using the well-known convolution theorem, we can compute fast one-dimensional linear convolutions at all rows of an input matrix by using Fast Fourier Transforms. To exploit the parallelism of the operations, we train the network on a GPU. A Matlab implementation processes multiple millions of input sentences per hour on one GPU, depending primarily on the number of layers used in the network.

Classifier	Features	Acc. (%)
Hier	unigram, POS, head chunks NE, semantic relations	91.0
MaxEnt	unigram, bigram, trigram POS, chunks, NE, supertags CCG parser, WordNet	92.6
MaxEnt	unigram, bigram, trigram POS, wh-word, head word word shape, parser hypernyms, WordNet	93.6
SVM	unigram, POS, wh-word head word, parser hypernyms, WordNet 60 hand-coded rules	95.0
Max-TDNN	unsupervised vectors	84.4
NBoW	unsupervised vectors	88.2
DCNN	unsupervised vectors	93.0

Table 2.2: Accuracy of six-way question classification on the TREC questions dataset. The second column details the external features used in the various approaches. The first four results are respectively from [Li and Roth, 2002], [Blunsom et al., 2006], [Huang et al., 2008] and [Silva et al., 2011].

2.6.2 Sentiment Prediction in Movie Reviews

The first two experiments concern the prediction of the sentiment of movie reviews in the Stanford Sentiment Treebank [Socher et al., 2013b]. The output variable is binary in one experiment and can have five possible outcomes in the other: negative, somewhat negative, neutral, somewhat positive, positive. In the binary case, we use the given splits of 6920 training, 872 development and 1821 test sentences. Likewise, in the fine-grained case, we use the standard 8544/1101/2210 splits. Labelled phrases that occur as subparts of the training sentences are treated as independent training instances. The size of the vocabulary is 15448.

Table 2.1 details the results of the experiments. In the three neural sentence models—the Max-TDNN, the NBoW and the DCNN—the word vectors are param-

Classifier	Accuracy (%)
SVM	81.6
BiNB	82.7
MaxEnt	83.0
Max-TDNN	78.8
NBoW	80.9
DCNN	**87.4**

Table 2.3: Accuracy on the Twitter sentiment dataset. The three non-neural classifiers are based on unigram and bigram features; the results are reported from [Go et al., 2009].

eters of the models that are randomly initialised; their dimension d is set to 48. The Max-TDNN has a filter of width 6 in its narrow convolution at the first layer; shorter phrases are padded with zero vectors. The convolutional layer is followed by a non-linearity, a max-pooling layer and a softmax classification layer. The NBoW sums the word vectors and applies a non-linearity followed by a softmax classification layer. The adopted non-linearity is the tanh function. The hyper parameters of the DCNN are as follows. The binary result is based on a DCNN that has a wide convolutional layer followed by a folding layer, a dynamic k-max pooling layer and a non-linearity; it has a second wide convolutional layer followed by a folding layer, a k-max pooling layer and a non-linearity. The width of the convolutional filters is 7 and 5, respectively. The value of k for the top k-max pooling is 4. The number of feature maps at the first convolutional layer is 6; the number of maps at the second convolutional layer is 14. The network is topped by a softmax classification layer. The DCNN for the fine-grained result has the same architecture, but the filters have size 10 and 7, the top pooling parameter k is 5 and the number of maps is, respectively, 6 and 12. The networks use the tanh non-linear function. At training time we apply dropout to the penultimate layer after the last tanh non-linearity [Hinton et al., 2012].

POSITIVE							'NOT'						
lovely	comedic	moments	and	several	fine	performances	n't	have	any	huge	laughs	in	its
good	script	,	good	dialogue	,	funny	no	movement	,	no	,	not	much
sustains	throughout	is	daring	,	inventive	and	n't	stop	me	from	enjoying	much	of
well	written	,	nicely	acted	and	beautifully	not	that	kung	pow	is	n't	funny
remarkably	solid	and	subtly	satirical	tour	de	not	a	moment	that	is	not	false
NEGATIVE							'TOO'						
,	nonexistent	plot	and	pretentious	visual	style	,	too	dull	and	pretentious	to	be
it	fails	the	most	basic	test	as	either	too	serious	or	too	lighthearted	,
so	stupid	,	so	ill	conceived	,	too	slow	,	too	long	and	too
,	too	dull	and	pretentious	to	be	feels	too	formulaic	and	too	familiar	to
hood	rats	butt	their	ugly	heads	in	is	too	predictable	and	too	self	conscious

Figure 2.4: Top five 7-grams at four feature detectors in the first layer of the network.

We see that the DCNN significantly outperforms the other neural and non-neural models. The NBoW performs similarly to the non-neural n-gram based classifiers. The Max-TDNN performs worse than the NBoW likely due to the excessive pooling of the max pooling operation; the latter discards most of the sentiment features of the words in the input sentence. Besides the RecNN that uses an external parser to produce structural features for the model, the other models use n-gram based or neural features that do not require external resources or additional annotations. In the next experiment we compare the performance of the DCNN with those of methods that use heavily engineered resources.

2.6.3 Question Type Classification

As an aid to question answering, a question may be classified as belonging to one of many question types. The TREC questions dataset involves six different question types, e.g. whether the question is about a location, about a person or about some numeric information [Li and Roth, 2002]. The training dataset consists of 5452 labelled questions whereas the test dataset consists of 500 questions.

The results are reported in Tab. 2.2. The non-neural approaches use a classifier over a large number of manually engineered features and hand-coded resources. For instance, [Blunsom et al., 2006] present a Maximum Entropy model that relies on 26 sets of syntactic and semantic features including unigrams, bigrams, trigrams, POS tags, named entity tags, structural relations from a CCG parse and WordNet synsets. We evaluate the three neural models on this dataset with mostly the same hyper-parameters as in the binary sentiment experiment of Sect. 2.6.2. As the dataset is rather small, we use lower-dimensional word vectors with $d = 32$ that are initialised with embeddings trained in an unsupervised way to predict contexts of occurrence [Turian et al., 2010]. The DCNN uses a single convolutional layer with filters of size 8 and 5 feature maps. The difference between the performance of the DCNN and that

of the other high-performing methods in Tab. 2 is not significant ($p < 0.09$). Given that the only labelled information used to train the network is the training set itself, it is notable that the network matches the performance of state-of-the-art classifiers that rely on large amounts of engineered features and rules and hand-coded resources.

2.6.4 Twitter Sentiment Prediction with Distant Supervision

In our final experiment, we train the models on a large dataset of tweets, where a tweet is automatically labelled as positive or negative depending on the emoticon that occurs in it. The training set consists of 1.6 million tweets with emoticon-based labels and the test set of about 400 hand-annotated tweets. We preprocess the tweets minimally following the procedure described in [Go et al., 2009]; in addition, we also lowercase all the tokens. This results in a vocabulary of 76643 word types. The architecture of the DCNN and of the other neural models is the same as the one used in the binary experiment of Sect. 2.6.2. The randomly initialised word embeddings are increased in length to a dimension of $d = 60$. Table 2.3 reports the results of the experiments. We see a significant increase in the performance of the DCNN with respect to the non-neural n-gram based classifiers; in the presence of large amounts of training data these classifiers constitute particularly strong baselines. We see that the ability to train a sentiment classifier on automatically extracted emoticon-based labels extends to the DCNN and results in highly accurate performance. The difference in performance between the DCNN and the NBoW further suggests that the ability of the DCNN to both capture features based on long n-grams and to hierarchically combine these features is highly beneficial.

2.6.5 Visualising Feature Detectors

A filter in the DCNN is associated with a feature detector or neuron that learns during training to be particularly active when presented with a specific sequence of

input words. In the first layer, the sequence is a continuous n-gram from the input sentence; in higher layers, sequences can be made of multiple separate n-grams. We visualise the feature detectors in the first layer of the network trained on the binary sentiment task (Sect. 2.6.2). Since the filters have width 7, for each of the 288 feature detectors we rank all 7-grams occurring in the validation and test sets according to their activation of the detector. Figure 2.4 presents the top five 7-grams for four feature detectors. Besides the expected detectors for positive and negative sentiment, we find detectors for particles such as 'not' that negate sentiment and such as 'too' that potentiate sentiment. We find detectors for multiple other notable constructs including 'all', 'or', 'with...that', 'as...as'. The feature detectors learn to recognise not just single n-grams, but patterns within n-grams that have syntactic, semantic or structural significance.

2.7 Discussion

Convolutional encoders like the DCNN were originally developed against the backdrop of Recursive Neural Networks that require an explicit syntactic parse tree for the sentence. It remains valid to this day that structure, syntactic and otherwise, can be captured extremely well *implicitly* in the distributed representation of neural encoders. Following an explicit structure may in fact limit what an encoder can access at a given point in the structure and what the encoder is able to learn; the structure of distributed spaces is more expressive than the limited structure of a syntactic tree that only provides an *ordering* according to which the tokens should be embedded. Convolutional encoders like the DCNN and other types of non-structured encoders such as those based on self-attention (Sect. 1.2.3.4) (and to some extent simple recurrent encoders) remain practical and popular in the current state of the field due to their computational efficiency, simplicity and effectiveness.

Chapter 3

Recurrent Convolutional Encoders for Discourse Compositionality

3.1 Summary

The previous section specified an architecture for encoding individual sentences. But the compositionality of meaning extends beyond the single sentence. Just as words combine to form the meaning of sentences, so do sentences combine to form the meaning of paragraphs, dialogues and general discourse. We introduce both a variation of the sentence model from Ch. 2 and a discourse model corresponding to the two levels of compositionality. The sentence model adopts convolution as the central operation for composing semantic vectors and is based on a novel hierarchical convolutional neural network. The discourse model extends the sentence model and is based on a recurrent neural network that is conditioned in a novel way both on the current sentence and on the current speaker. The discourse model is able to capture both the sequentiality of sentences and the interaction between different speakers. Without feature engineering or pretraining and with simple greedy decoding, the discourse model coupled to the sentence model obtains state of the art performance on a dialogue act

classification experiment.

3.2 Background

There are at least two levels at which the meaning of smaller linguistic units is composed to form the meaning of larger linguistic units. The first level is that of sentential compositionality, where the meaning of words composes to form the meaning of the sentence or utterance that contains them [Frege, 1892]. The second level extends beyond the first and involves general discourse compositionality, where the meaning of multiple sentences or utterances composes to form the meaning of the paragraph, document or dialogue that comprises them [Korta and Perry, 2012, Potts, 2011]. The problem of discourse compositionality is the problem of modelling how the meaning of general discourse composes from the meaning of the sentences involved and, since the latter in turn stems from the meaning of the words, how the meaning of discourse composes from the words themselves.

Tackling the problem of discourse compositionality promises to be central to a number of different applications. These include sentiment or topic classification of single sentences within the context of a longer discourse, the recognition of dialogue acts within a conversation, the classification of a discourse as a whole and the attainment of general unsupervised or semi-supervised representations of a discourse for potential use in dialogue tracking and question answering systems and machine translation, among others.

To this end much work has been done on modelling the meaning of single words by way of semantic vectors [Turney and Pantel, 2010, Collobert and Weston, 2008] and the latter have found applicability in areas such as information retrieval [Jones et al., 2006]. With regard to modelling the meaning of sentences and sentential compositionality, recent proposals have included simple additive and multiplicative models

that do not take into account sentential features such as word order or syntactic structure [Mitchell and Lapata, 2010], matrix-vector based models that do take into account such features but are limited to phrases of a specific syntactic type [Baroni and Zamparelli, 2010] and structured models that fully capture such features [Grefenstette et al., 2011] and are embedded within a deep neural architecture [Socher et al., 2012, Hermann and Blunsom, 2013]. It is notable that the additive and multiplicative models as well as simple, non-compositional bag of n-grams and word vector averaging models have equalled or outperformed the structured models at certain phrase similarity [Blacoe and Lapata, 2012] and sentiment classification tasks [Scheible and Schütze, 2013, Wang and Manning, 2012].

With regard to discourse compositionality, most of the proposals aimed at capturing semantic aspects of paragraphs or longer texts have focused on bag of n-grams or sentence vector averaging approaches [Wang and Manning, 2012, Socher et al., 2012]. In addition, the recognition of dialogue acts within dialogues has largely been treated in non-compositional ways by way of language models coupled to hidden Markov sequence models [Stolcke et al., 2000]. Principled approaches to discourse compositionality have largely been unexplored.

We introduce a novel model for sentential compositionality. The composition operation is based on a hierarchy of one dimensional convolutions. The convolutions are applied feature-wise, that is they are applied across each feature of the word vectors in the sentence. The weights adopted in each convolution are different for each feature, but do not depend on the different words being composed. The hierarchy of convolution operations involves a sequence of convolution kernels of increasing sizes (Fig. 3.1). This allows for the composition operation to be applied to sentences of any length, while keeping the model at a depth of roughly $\sqrt{2l}$ where l is the length of the sentence. The hierarchy of feature-wise convolution operations followed by sigmoid non-linear activation functions results in a hierarchical convolutional neural

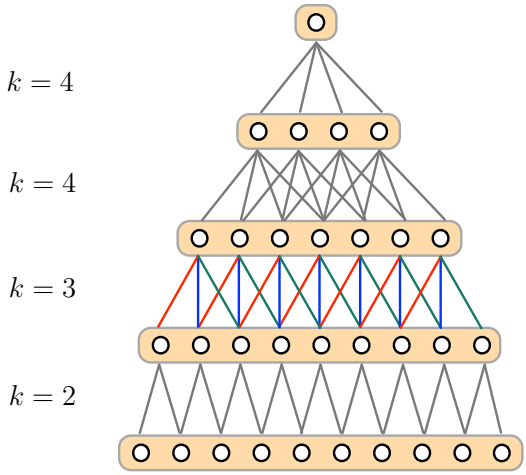

Figure 3.1: A hierarchical convolutional neural network for sentential compositionality. The bottom layer represents a single feature across all the word vectors in the sentence. The top layer is the value for that feature in the resulting sentence vector. Lines represent single weights and color coded lines indicate sharing of weights. The parameter k indicates the size of the convolution kernel at the corresponding layer.

network (HCNN) based on a convolutional architecture [LeCun et al., 2001]. The HCNN shares with the structured models the aspect that it is sensitive to word order and adopts a hierarchical architecture, although it is not based on explicit syntactic structure.

We also introduce a novel model for discourse compositionality. The discourse model is based on a recurrent neural network (RNN) architecture that is a powerful model for sequences [Sutskever et al., 2011a, Mikolov et al., 2010]. The model aims at capturing two central aspects of discourse and its meaning: the sequentiality of the sentences or utterances in the discourse and, where applicable, the interactions between the different speakers. The underlying RNN has its recurrent and output weights conditioned on the respective speaker, while simultaneously taking as input at every turn the sentence vector for the current sentence generated through the sentence model (Fig. 3.2).

We experiment with the discourse model coupled to the sentence model on the task of recognizing dialogue acts of utterances within a conversation. The dataset is given

by 1134 transcribed and annotated telephone conversations amounting to about 200K utterances from the Switchboard Dialogue Act Corpus [Calhoun et al., 2010].[1] The model is trained in a supervised setting without previous pretraining; word vectors are also randomly initialised. The model learns a probability distribution over the dialogue acts at step i given the sequence of utterances up to step i, the sequence of acts up to the previous step $i-1$ and the binary sequence of agents up to the current step i. Predicting the sequence of dialogue acts is performed in a greedy fashion.

We proceed as follows. In Sect. 3.3 we give the motivation and the definition for the HCNN sentence model. In Sect. 3.4 we do the same for the RCNN discourse model. In Sect. 3.5 we describe the dialogue act classification experiment and the training procedure. We also inspect the discourse vector representations produced by the model.

3.3 Sentence Model

The general aim of the sentence model is to compute a vector for a sentence s given the sequence of words in s and a vector for each of the words. The computation captures certain general considerations regarding sentential compositionality. We first relate such considerations and we then proceed to give a definition of the model.

3.3.1 Sentential compositionality

There are three main aspects of sentential compositionality that the model aims at capturing. To relate these, it is useful to note the following basic property of the model: a sentence s is paired to the matrix $\mathbf{M}^s$ whose columns are given sequentially by the vectors of the words in s. A row in $\mathbf{M}^s$ corresponds to the values of the corresponding feature across all the word vectors. The first layer of the network in

[1]The dataset is available at compprag.christopherpotts.net/swda.html

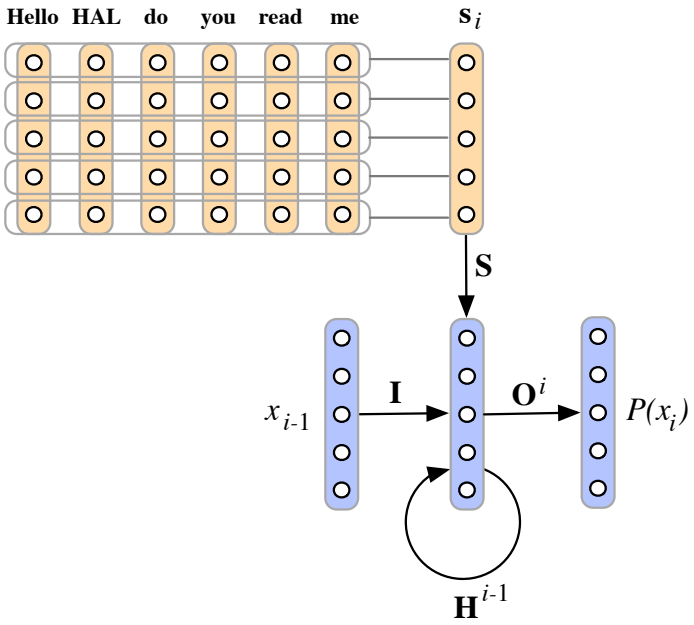

Figure 3.2: Recurrent convolutional neural network (RCNN) discourse model based on a RNN architecture. At each step the RCNN takes as input the current sentence vector $\mathbf{s}_i$ generated through the HCNN sentence model and the previous label x_{i-1} to predict a probability distribution over the current label $P(x_i)$. The recurrent weights $\mathbf{H}^{i-1}$ are conditioned on the previous agent a_{i-1} and the output weights are conditioned on the current agent a_i. Note also the sentence matrix $\mathbf{M}_s$ of the sentence model and the hierarchy of convolutions applied to each feature that is a row in $\mathbf{M}_s$ to produce the corresponding feature in $\mathbf{s}_i$.

Fig. 3.1 represents one such row of $\mathbf{M}^s$, whereas the whole matrix $\mathbf{M}^s$ is depicted in Fig. 3.2. The three considerations are as follows.

First, at the initial stage of the composition, the value of a feature in the sentence vector is a function of the values of the same feature in the word vectors. That is, the m-th value in the sentence vector of s is a function of the m-th row of $\mathbf{M}^s$. This aspect is preserved in the additive and multiplicative models where the composition operations are, respectively, addition + and component-wise multiplication $\odot$. The current model preserves the aspect up to the computation of the sentence vector $\mathbf{s}$ by adopting one-dimensional, feature-wise convolution operations. Subsequently, the discourse model that uses the sentence vector $\mathbf{s}$ includes transformations across the features of $\mathbf{s}$ (the transformation $\mathbf{S}$ in Fig. 3.2).

The second consideration concerns the hierarchical aspect of the composition operation. We take the compositionality of meaning to initially yield local effects across neighbouring words and then yield increasingly more global effects across all the words in the sentence. Composition operations like those in the structured models that are guided by the syntactic parse tree of the sentence capture this trait. The sentence model preserves this aspect not by way of syntactic structure, but by adopting convolution kernels of gradually increasing sizes that span an increasing number of words and ultimately the entire sentence.

The third aspect concerns the dependence of the composition operation. The operation is taken to depend on the different features, but not on the different words. Word specific parameters are introduced only by way of the learnt word vectors, but no word specific operations are learnt. We achieve this by using a single convolution kernel across a feature, and by utilizing different convolution kernels for different features. Given these three aspects of sentential compositionality, we now proceed to describe the sentence model in detail.

3.3.2 Hierarchical Convolutional Neural Network

The sentence model is taken to be a CNN where the convolution operation is applied one dimensionally across a single feature and in a hierarchical manner. To describe it in more detail, we first recall the convolution operation that is central to the model. Then we describe how we compute the sequence of kernel sizes and how we determine the hierarchy of layers in the network.

3.3.2.1 Kernel and One-dimensional Convolution

We recall the definition of sentence matrix and one-dimensional convolution applied to the sentence matrix (Sect. 2.3); the reader familiar with these notions may skip to the next subsection. Given a sentence s and its paired matrix $\mathbf{M}^s$, let $\mathbf{m}$ be a feature that is a row in $\mathbf{M}^s$. Before defining kernels and the convolution operation, let us consider the underlying operation of *local weighted addition*. Let $w_1, ..., w_k$ be a sequence of k weights; given the feature $\mathbf{m}$, local weighted addition over the *first* k values of $\mathbf{m}$ gives:

$$y = w_1\mathbf{m}_1 + ... + w_k\mathbf{m}_k \tag{3.1}$$

Then, a kernel simply defines the value of k by specifying the sequence of weights $w_1, ..., w_k$ and the one-dimensional convolution applies local weighted addition with the k weights to each subsequence of values of $\mathbf{m}$.

More precisely, let a one-dimensional *kernel* $\mathbf{k}$ be a vector of weights and assume $|\mathbf{k}| \leq |\mathbf{m}|$, where $|\cdot|$ is the number of elements in a vector. Then we define the discrete, valid, *one-dimensional convolution* $(\mathbf{k} * \mathbf{m})$ of kernel $\mathbf{k}$ and feature $\mathbf{m}$ by:

$$(\mathbf{k} * \mathbf{m})_i := \sum_{j=1}^{k} \mathbf{k}_j \cdot \mathbf{m}_{k+i-j} \tag{3.2}$$

where $k = |\mathbf{k}|$ and $|\mathbf{k} * \mathbf{m}| = |\mathbf{m}| - k + 1$. Each value in $\mathbf{k} * \mathbf{m}$ is a sum of k values of $\mathbf{m}$ weighted by values in $\mathbf{k}$ (Fig. 3.3). To define the hierarchical architecture of the

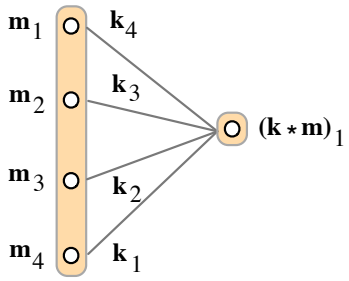

Figure 3.3: Convolution of a vector $\mathbf{m}$ with a kernel $\mathbf{k}$ of size 4.

model, we need to define a sequence of kernel sizes and associated weights. To this we turn next.

3.3.2.2 Sequence of Kernel Sizes

Let l be the number of words in the sentence s. The sequence of kernel sizes $\langle k_i^l \rangle_{i \leq t}$ depends only on the length of s and itself has length $t = \lceil \sqrt{2l} \rceil - 1$. It is given recursively by:

$$k_1^l = 2, \quad k_{i+1}^l = k_i^l + 1, \quad k_t^l = l - \sum_{j=1}^{t-1} (k_j^l - 1) \tag{3.3}$$

That is, kernel sizes increase by one until the resulting convolved vector is smaller or equal to the last kernel size; see for example the kernel sizes in Fig. 3.1. Note that, for a sentence of length l, the number of layers in the HCNN including the input layer will be $t + 1$ as convolution with the corresponding kernel is applied at every layer of the model. Let us now proceed to define the hierarchy of layers in the HCNN.

3.3.2.3 Composition Operation in a HCNN

Given a sentence s, its length l and a sequence of kernel sizes $\langle k_i^l \rangle_{i \leq t}$, we may now give the recursive definition that yields the hierarchy of one-dimensional convolution operations applied to each feature f that is a row in $\mathbf{M}^s$. Specifically, for each feature f, let $\mathbf{K}_i^f$ be a sequence of t kernels, where the size of the kernel $|\mathbf{K}_i^f| = k_i^l$. Then we

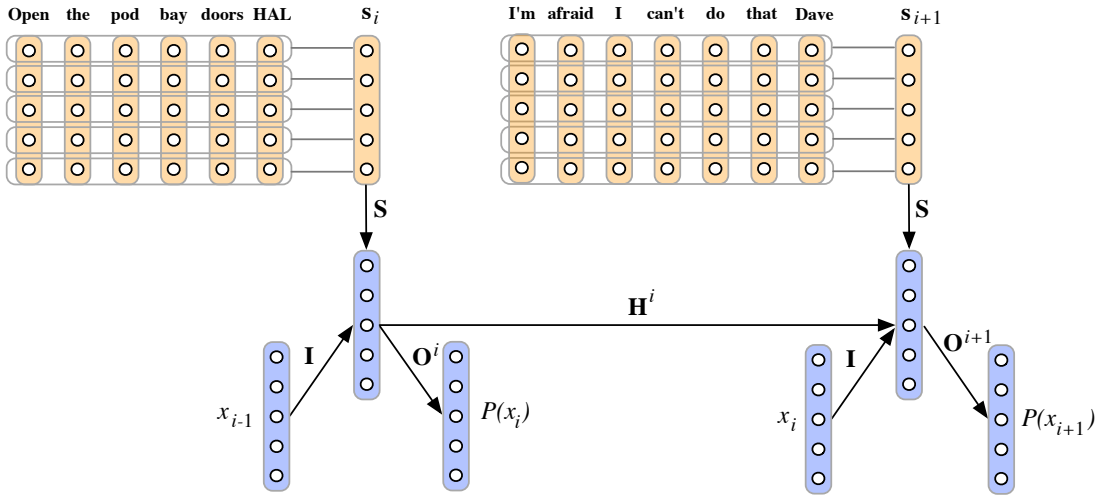

Figure 3.4: Unravelling of a RCNN discourse model to depth $d = 2$. The recurrent $\mathbf{H}^i$ and output $\mathbf{O}^i$ weights are conditioned on the respective agents a_i.

have the hierarchy of matrices and corresponding features as follows:

$$\mathbf{M}^1_{f,:} = \mathbf{M}^s_{f,:} \tag{3.4}$$

$$\mathbf{M}^{i+1}_{f,:} = \sigma(\ \mathbf{K}^f_i * \mathbf{M}^i_{f,:} + b^i_f\ ) \tag{3.5}$$

for some non-linear sigmoid function σ and bias b^i_f, where i ranges over $1, ..., t$. In sum, one-dimensional convolution is applied feature-wise to each feature of a matrix at a certain layer, where the kernel weights depend both on the layer and the feature at hand (Fig. 3.1). A hierarchy of matrices is thus generated with the top matrix being a single vector for the sentence.

3.3.2.4 Multiple merged HCNNs

Optionally one may consider multiple parallel HCNNs that are merged according to different strategies either at the top sentence vector layer or at intermediate layers. The weights in the word vectors may be tied across different HCNNs. Although potentially useful, multiple merged HCNNs are not used in the experiment below.

This concludes the description of the sentence model. Let us now proceed to the discourse model.

3.4 Discourse Model

The discourse model adapts a RNN architecture in order to capture central properties of discourse. We here first describe such properties and then define the model itself.

3.4.1 Discourse Compositionality

The meaning of discourse - and of words and utterances within it - is often a result of a rich ensemble of context, of speakers' intentions and actions and of other relevant surrounding circumstances [Korta and Perry, 2012, Potts, 2011]. Far from capturing all aspects of discourse meaning, we aim at capturing in the model at least two of the most prominent ones: the sequentiality of the utterances and the interactions between the speakers.

Concerning sequentiality, just the way the meaning of a sentence generally changes if words in it are permuted, so does the meaning of a paragraph or dialogue change if one permutes the sentences or utterances within. The change of meaning is more marked the larger the shift in the order of the sentences. Especially in tasks where one is concerned with a specific sentence within the context of the previous discourse, capturing the order of the sentences preceding the one at hand may be particularly crucial.

Concerning the speakers' interactions, the meaning of a speaker's utterance within a discourse is differentially affected by the speaker's previous utterances as opposed to *other* speakers' previous utterances. Where applicable we aim at making the computed meaning vectors reflect the current speaker and the sequence of interactions with the previous speakers. With these two aims in mind, let us now proceed to define

Dialogue Act Label	Example	Train (%)	Test (%)
Statement	*And, uh, it's a legal firm office.*	36.9	31.5
Backchannel/Acknowledge	*Yeah, anything could happen.*	18.8	18.2
Opinion	*I think that would be great.*	12.7	17.1
Abandoned/Uninterpretable	*So, -*	7.6	8.6
Agreement/Accept	*Yes, exactly.*	5.5	5.0
Appreciation	*Wow.*	2.3	2.2
Yes−No−Question	*Is that what you do?*	2.3	2.0
Non−Verbal	*[Laughter], [Throat-clearing]*	1.7	1.9
Other labels (34)		12.2	13.5
Total number of utterances		196258	4186
Total number of dialogues		1115	19

Table 3.1: Most frequent dialogue act labels with examples and frequencies in train and test data.

the model.

3.4.2 Recurrent Convolutional Neural Network

The discourse model coupled to the sentence model is based on a RNN architecture with inputs from a HCNN and with the recurrent and output weights conditioned on the respective speakers.

We take as given a sequence of sentences or utterances $s_1, ..., s_T$, each in turn being a sequence of words $s_i = y_1^i ... y_l^i$, a sequence of labels $x_1, ..., x_T$ and a sequence of speakers or agents $a_1, ..., a_T$, in such way that the i-th utterance is performed by the i-th agent and has label x_i. We denote by $\mathbf{s}_i$ the sentence vector computed by way of the sentence model for the sentence s_i. The RCNN computes probability distributions p_i for the label at step i by iterating the following equations:

$$\mathbf{h}_i = \sigma(\ \mathbf{I}x_{i-1} + \mathbf{H}^{i-1}\mathbf{h}_{i-1} + \mathbf{S}\mathbf{s}_i + b_h) \tag{3.6}$$

$$p_i = softmax(\mathbf{O}^i\mathbf{h}_i + b_o) \tag{3.7}$$

where $\mathbf{I}, \mathbf{H}^i, \mathbf{O}^i$ are corresponding weight matrices for each agent a_i and $softmax(y)_k =$

Center Dialogue	A: *Do you repair your own car?* B: *I try to, whenever I can.*	A: *– I guess we can start.* B: *Okay.*	A: *Did you use to live around here?* B: *Uh, Redwood City.*
First NN	A: *Do you do it every day?* B: *I try to every day.*	A: *I think for serial murder –* B: *Uh-huh.*	A: *Can you stand up in it?* B: *Uh, in parts.*
Second NN	A: *Well, do you have any children?* B: *I've got one.*	A: *The USSR – wouldn't do it* B: *Uh-huh.*	A: *[Laughter] Do you have any kids that you take fishing?* B: *Uh, got a stepdaughter.*
Third NN	A: *Do you manage the money?* B: *Well, I, we talk about it.*	A: *It seems to me there needs to be some ground, you know, some rules –* B: *Uh-huh.*	A: *Is our five minutes up?* B: *Uh, pretty close to it.*
Fourth NN	A: *Um, do you watch it every Sunday?* B: *[Breathing] Uh, when I can.*	A: *It sounds to me like, uh, you are doing well.* B: *My husband's retired.*	A: *Do you usually go out, uh, with the children or without them?* B: *Well, a variety.*

Table 3.2: Short dialogues and nearest neighbours (NN).

$\frac{e^{y_k}}{\sum_j e^{y_j}}$ returns a probability distribution. Thus p_i is taken to model the following predictive distribution:

$$p_i = P(x_i | x_{<i}, s_{\leq i}, a_{\leq i}) \tag{3.8}$$

An RCNN and the unravelling to depth $d = 2$ are depicted respectively in Fig. 3.2 and Fig. 3.4. With regards to vector representations of discourse, we take the hidden layer $\mathbf{h}_i$ as the vector representing the discourse up to step i. This concludes the description of the discourse model. Let us now consider the experiment.

3.5 Predicting Dialogue Acts

We experiment with the prediction of dialogue acts within a conversation. A dialogue act specifies the pragmatic role of an utterance and helps identifying the speaker's intentions [Austin, 1962, Korta and Perry, 2012]. The automated recognition of dialogue acts is crucial for dialogue state tracking within spoken dialogue systems [Williams, 2012]. We first describe the Switchboard Dialogue Act (SwDA) corpus [Calhoun et al., 2010] that serves as the dataset in the experiment. We report on the training procedure and the results and we make some qualitative observations regarding the discourse representations produced by the model.

3.5.1 SwDA Corpus

The SwDA corpus contains audio recordings and transcripts of telephone conversations between multiple speakers that do not know each other and are given a topic for discussion. For a given utterance we use the transcript of the utterance, the dialogue act label and the speaker's label; no other annotations are used in the model. Overall there are 42 distinct dialogue act labels such as Statement and Opinion (Tab. 3.1). We adopt the same data split of 1115 train dialogues and 19 test dialogues as used in [Stolcke et al., 2000].

3.5.2 Objective Function and Training

We minimise the cross-entropy error of the predicted and the true distributions and include an l_2 regularisation parameter. The RCNN is truncated to a depth $d = 2$ so that the prediction of a dialogue act depends on the previous two utterances, speakers and dialogue acts; adopting depths > 2 has not yielded improvements in the experiment. The derivatives are efficiently computed by back-propagation [Rumelhart et al., 1986b]. The word vectors are initialised to random vectors of length 25 and no pretraining procedure is performed. We minimise the objective using L-BFGS in mini-batch mode; the minimisation converges smoothly.

3.5.3 Prediction Method and Results

The prediction of a dialogue act is performed in a greedy fashion. Given the two previously predicted acts $\hat{x}_{i-1}, \hat{x}_{i-2}$, one chooses the act $\hat{x}_i$ that has the maximal probability in the predicted distribution $P(x_i)$. The LM-HMM model of [Stolcke et al., 2000] learns a language model for each dialogue act and a Hidden Markov Model for the sequence of dialogue acts and it requires all the utterances in a dialogue in order to predict the dialogue act of any one of the utterances. The RCNN makes the

	Accuracy (%)
RCNN	**73.9**
LM-HMM trigram	71.0
LM-HMM bigram	70.6
LM-HMM unigram	68.2
Majority baseline	31.5
Random baseline	2.4

Table 3.3: SwDA dialogue act tagging accuracies. The LM-HMM results are from [Stolcke et al., 2000]. Inter-annotator agreement and theoretical maximum is 84%.

weaker assumption that only the utterances up to utterance i are available to predict the dialogue act $\hat{x}_i$. The accuracy results of the models are compared in Tab. 3.3.

3.5.4 Discourse Vector Representations

We inspect the discourse vector representations that the model generates. After a dialogue is processed, the hidden layer $\mathbf{h}$ of the RCNN is taken to be the vector representation for the dialogue (Sect. 3.2). Table 3.2 includes three randomly chosen dialogues composed of two utterances each; for each dialogue the table reports the four nearest neighbours. As the word vectors and weights are initialised randomly without pretraining, the word vectors and the weights are induced during training only through the dialogue act labels attached to the utterances. The distance between two word, sentence or discourse vectors reflects a notion of pragmatic similarity: two words, sentences or discourses are similar if they contribute in a similar way to the pragmatic role of the utterance signalled by the associated dialogue act. This is suggested by the examples in Tab. 3.2, where a centre dialogue and a nearest neighbour may have some semantically different components (e.g. "repair your own car" and "manage the money"), but be pragmatically similar and the latter similarity is captured by the representations. In the examples, the meaning of the relevant words in the utterances, the speakers' interactions and the sequence of pragmatic roles are well preserved across the nearest neighbours.

3.6 Discussion

Motivated by the compositionality of meaning both in sentences and in general discourse, this chapter introduced a sentence model based on a convolutional architecture and a discourse model based on a novel use of recurrent networks. To this day this type of architecture remains compelling. The sentence encoders may be improved with recent convolutional or attentional ideas, e.g. residual connections, but the general structure of the word-level parallelizable convolutional encoder and the sentence-level recurrent encoder for the context are often used today. We use an analogous structure in the visual domain in the Video Pixel Network model that encodes single frames with a convolutional network and frame-to-frame transitions with a recurrent one (Sect. 7.4). The general study of discourse still remains somewhat unexplored; for example, neural machine translation is still largely performed at a sentence level. Nevertheless, simply using the added discourse-level context is bound to eventually give rise to significant improvements in many language comprehension and generation tasks. This ends Part I of the thesis devoted to neural encoders for single and multi-utterance sequences. We next turn to neural decoders and encoder-decoders in Part II.

Part II

Encoder-Decoder Neural Networks

Chapter 4

Recurrent Continuous Translation Models

4.1 Summary

The coming chapter introduces the key notion of encoder-decoder neural networks and their instantiation as neural translation models. The chapter describes two constructions of such networks, one that uses a fixed-length encoding and one that uses a variable-length encoding. We introduce a class of probabilistic continuous translation models called Recurrent Continuous Translation Models that are purely based on continuous representations for words, phrases and sentences and do not rely on alignments or phrasal translation units, as is common in traditional systems. RCTMs are neural translation models that are specific instances of encoder-decoder neural networks. The models have a generation and a conditioning aspect. The generation of the translation is modelled with a target Recurrent Language Model, whereas the conditioning on the source sentence is modelled with a Convolutional Sentence Model similar to the one describe in Ch. 2 but with some translation specific features. Through various experiments, we show first that our models obtain a perplexity with respect to

gold translations that is $> 43\%$ lower than that of state-of-the-art alignment-based translation models. Secondly, we show that they are remarkably sensitive to the word order, syntax, and meaning of the source sentence despite lacking alignments. Finally we show that they match a state-of-the-art system when rescoring n-best lists of translations.

4.2 Background

In most statistical approaches to machine translation the basic units of translation are phrases that are composed of one or more words. A crucial component of translation systems are models that estimate translation probabilities for pairs of phrases, one phrase being from the source language and the other from the target language. Such models count phrase pairs and their occurrences as distinct if the surface forms of the phrases are distinct. Although distinct phrase pairs often share significant similarities, linguistic or otherwise, they do not share statistical weight in the models' estimation of their translation probabilities. Besides ignoring the similarity of phrase pairs, this leads to general sparsity issues. The estimation is sparse or skewed for the large number of rare or unseen phrase pairs, which grows exponentially in the length of the phrases, and the generalisation to other domains is often limited.

Continuous representations have shown promise at tackling these issues. Continuous representations for words are able to capture their morphological, syntactic and semantic similarity [Collobert and Weston, 2008]. They have been applied in continuous language models demonstrating the ability to overcome sparsity issues and to achieve state-of-the-art performance [Bengio et al., 2003a, Mikolov et al., 2010]. Word representations have also shown a marked sensitivity to conditioning information [Mikolov and Zweig, 2012]. Continuous representations for characters have been deployed in character-level language models demonstrating notable language gener-

ation capabilities [Sutskever et al., 2011a]. As we have done in Ch. 2, continuous representations have also been constructed for phrases and sentences. The representations are able to carry similarity and task dependent information, e.g. sentiment, paraphrase or dialogue labels, significantly beyond the word level and to accurately predict labels for a highly diverse range of unseen phrases and sentences [Grefenstette et al., 2011, Socher et al., 2011a, Socher et al., 2012, Hermann and Blunsom, 2013].

Phrase-based continuous translation models were first proposed in [Schwenk et al., 2006] and recently further developed in [Schwenk, 2012, Le et al., 2012]. The models incorporate a principled way of estimating translation probabilities that robustly extends to rare and unseen phrases. They achieve significant Bleu score improvements and yield semantically more suggestive translations. Although wide-reaching in their scope, these models are limited to fixed-size source and target phrases and simplify the dependencies between the target words taking into account restricted target language modelling information.

We describe a class of continuous translation models called Recurrent Continuous Translation Models (RCTM) that map without loss of generality a sentence from the source language to a probability distribution over the sentences in the target language. We define two specific RCTM architectures. Both models adopt a recurrent language model for the generation of the target translation [Mikolov et al., 2010]. In contrast to other n-gram approaches, the recurrent language model makes no Markov assumptions about the dependencies of the words in the target sentence.

The two RCTMs differ in the way they condition the target language model on the source sentence. The first RCTM uses the convolutional sentence model from Ch. 3 to transform the source word representations into a representation for the source sentence. The source sentence representation in turn constraints the generation of each target word. The second RCTM introduces an intermediate representation. It uses a truncated variant of the convolutional sentence model to first transform the

source word representations into representations for the target words; the latter then constrain the generation of the target sentence. In both cases, the convolutional layers are used to generate combined representations for the phrases in a sentence from the representations of the words in the sentence.

An advantage of RCTMs is the lack of latent alignment segmentations and the sparsity associated with them. Connections between source and target words, phrases and sentences are learnt only implicitly as mappings between their continuous representations. As we see in Sect. 4.6, these mappings often carry remarkably precise morphological, syntactic and semantic information. Another advantage is that the probability of a translation under the models is efficiently computable requiring a small number of matrix-vector products that is linear in the length of the source and the target sentence. Further, translations can be generated directly from the probability distribution of the RCTM without any external resources.

We evaluate the performance of the models in four experiments. Since the translation probabilities of the RCTMs are tractable, we can measure the perplexity of the models with respect to the reference translations. The perplexity of the models is significantly lower than that of IBM Model 1 and is $> 43\%$ lower than the perplexity of a state-of-the-art variant of the IBM Model 2 [Brown et al., 1993, Dyer et al., 2013]. The second and third experiments aim to show the sensitivity of the output of the RCTM II to the linguistic information in the source sentence. The second experiment shows that under a random permutation of the words in the source sentences, the perplexity of the model with respect to the reference translations becomes significantly worse, suggesting that the model is highly sensitive to word position and order. The third experiment inspects the translations generated by the RCTM II. The generated translations demonstrate remarkable morphological, syntactic and semantic agreement with the source sentence. Finally, we test the RCTMs on the task of rescoring n-best lists of translations. The performance of the RCTM prob-

abilities joined with a single word penalty feature matches the performance of the state-of-the-art translation system `cdec` that makes use of twelve features including five alignment-based translation models [Dyer et al., 2010].

We proceed as follows. We begin in Sect. 4.3 by describing the general modelling framework underlying the RCTMs. In Sect. 4.4 we describe the RCTM I and in Sect. 4.5 the RCTM II. Section 4.6 is dedicated to the four experiments and we conclude in Sect. 4.7.

4.3 Framework

We recall some notions from Ch. 1 and describe the encoder-decoder framework underling the specific RCTM models in detail. An RCTM estimates the probability $P(\mathsf{f}|\mathsf{e})$ of a target sentence $\mathsf{f} = \mathsf{f}_1, ..., \mathsf{f}_m$ being a translation of a source sentence $\mathsf{e} = \mathsf{e}_1, ..., \mathsf{e}_k$. Let us denote by $\mathsf{f}_{i:j}$ the substring of words $\mathsf{f}_i, ..., \mathsf{f}_j$. Using the following identity,

$$P(\mathsf{f}|\mathsf{e}) = \prod_{i=1}^{m} P(\mathsf{f}_i|\mathsf{f}_{1:i-1}, \mathsf{e}) \tag{4.1}$$

an RCTM estimates $P(\mathsf{f}|\mathsf{e})$ by directly computing for each target position i the conditional probability $P(\mathsf{f}_i|\mathsf{f}_{1:i-1}, \mathsf{e})$ of the target word f_i occurring in the translation at position i, given the preceding target words $\mathsf{f}_{1:i-1}$ and the source sentence e. We see that an RCTM is sensitive not just to the source sentence e but also to the preceding words $\mathsf{f}_{1:i-1}$ in the target sentence; by doing so it incorporates a model of the target language itself.

To model the conditional probability $P(\mathsf{f}|\mathsf{e})$, an RCTM comprises both a generative architecture for the target sentence and an architecture for conditioning the latter on the source sentence. To fully capture Eq. 4.1, we model the generative architecture with a recurrent language model (RLM) based on a recurrent neural network [Mikolov et al., 2010]. The prediction of the i-th word f_i in a RLM depends on all

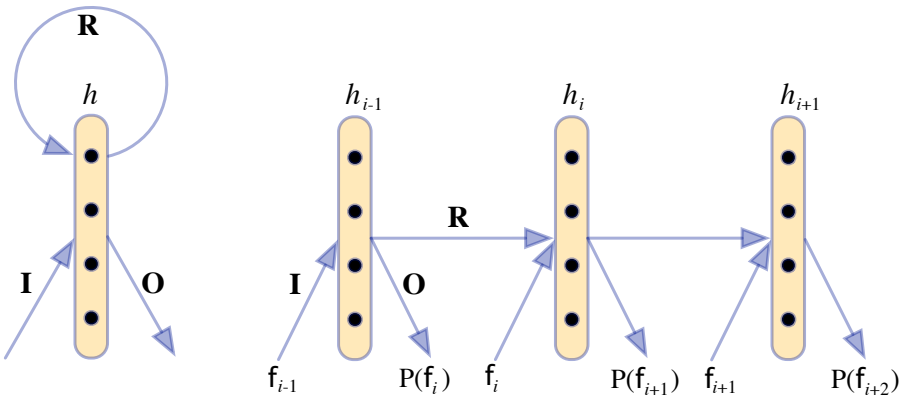

Figure 4.1: A RLM (left) and its unravelling to depth 3 (right). The recurrent transformation is applied to the hidden layer h_{i-1} and the result is summed to the representation for the current word f_i. After a non-linear transformation, a probability distribution over the next word f_{i+1} is predicted.

the preceding words $\mathsf{f}_{1:i-1}$ in the target sentence ensuring that conditional independence assumptions are not introduced in Eq. 4.1. Although the prediction is most strongly influenced by words closely preceding f_i, long-range dependencies from across the whole sentence can also be exhibited. The conditioning architectures are model specific and are treated in Sect. 4.4-4.5. Both the generative and conditioning aspects of the models deploy continuous representations for the constituents and are trained as a single joint architecture. Given the modelling framework underlying RCTMs, we now proceed to describe in detail the recurrent language model underlying the generative aspect.

4.3.1 Recurrent Language Model

A RLM models the probability $P(\mathsf{f})$ that the sequence of words f occurs in a given language. Let $\mathsf{f} = \mathsf{f}_1, ..., \mathsf{f}_m$ be a sequence of m words, e.g. a sentence in the target language. Analogously to Eq. 4.1, using the identity,

$$P(\mathsf{f}) = \prod_{i=1}^{m} P(\mathsf{f}_i | \mathsf{f}_{1:i-1}) \tag{4.2}$$

the model explicitly computes without simplifying assumptions the conditional distributions $P(\mathsf{f}_i|\mathsf{f}_{1:i-1})$. The architecture of a RLM comprises a vocabulary V that contains the words f_i of the language as well as three transformations: an input vocabulary transformation $\mathbf{I} \in \mathbb{R}^{q\times|V|}$, a recurrent transformation $\mathbf{R} \in \mathbb{R}^{q\times q}$ and an output vocabulary transformation $\mathbf{O} \in \mathbb{R}^{|V|\times q}$. For each word $\mathsf{f}_k \in V$, we indicate by $\mathsf{i}(\mathsf{f}_k)$ its index in V and by $\mathsf{v}(\mathsf{f}_k) \in \mathbb{R}^{|V|\times 1}$ an all zero vector with only $\mathsf{v}(\mathsf{f}_k)_{\mathsf{i}(\mathsf{f}_k)} = 1$.

For a word f_i, the result of $\mathbf{I} \cdot \mathsf{v}(\mathsf{f}_i) \in \mathbb{R}^{q\times 1}$ is the input continuous representation of f_i. The parameter q governs the size of the word representation. The prediction proceeds by successively applying the recurrent transformation $\mathbf{R}$ to the word representations and predicting the next word at each step. In detail, the computation of each $P(\mathsf{f}_i|\mathsf{f}_{1:i-1})$ proceeds recursively. For $1 < i < m$,

$$h_1 = \sigma(\mathbf{I} \cdot \mathsf{v}(\mathsf{f}_1)) \tag{4.3a}$$

$$h_{i+1} = \sigma(\mathbf{R} \cdot h_i + \mathbf{I} \cdot \mathsf{v}(\mathsf{f}_{i+1})) \tag{4.3b}$$

$$o_{i+1} = \mathbf{O} \cdot h_i \tag{4.3c}$$

and the conditional distribution is given by,

$$P(\mathsf{f}_i = v|\mathsf{f}_{1:i-1}) = \frac{\exp(o_{i,v})}{\sum_{v=1}^{V} \exp(o_{i,v})} \tag{4.4}$$

In Eq. 4.3, σ is a nonlinear function such as tanh. Bias values b_h and b_o are included in the computation. An illustration of the RLM is given in Fig. 4.1.

The RLM is trained by backpropagation through time [Mikolov et al., 2010]. The error in the predicted distribution calculated at the output layer is backpropagated through the recurrent layers and cumulatively added to the errors of the previous predictions for a given number d of steps. The procedure is equivalent to standard backpropagation over a RLM that is unravelled to depth d as in Fig. 4.1.

RCTMs may be thought of as RLMs, in which the predicted distributions for each

word f_i are conditioned on the source sentence e. We next define two conditioning architectures each giving rise to a specific RCTM.

4.4 Recurrent Continuous Translation Model I

The RCTM I uses a convolutional sentence model (CSM) in the conditioning architecture; this is similar to the ones described in Ch. 2 and Ch. 3, but it uses no pooling and details about kernel sizes differ. The CSM creates a representation for a sentence that is progressively built up from representations of the n-grams in the sentence. The CSM embodies a hierarchical structure. Although it does not make use of an explicit parse tree, the operations that generate the representations act locally on small n-grams in the lower layers of the model and act increasingly more globally on the whole sentence in the upper layers of the model. The lack of the need for a parse tree yields two central advantages over sentence models that require it [Grefenstette et al., 2011, Socher et al., 2012]. First, it makes the model robustly applicable to a large number of languages for which accurate parsers are not available. Secondly, the translation probability distribution over the target sentences does not depend on the chosen parse tree.

The RCTM I conditions the probability of each target word f_i on the continuous representation of the source sentence e generated through the CSM. This is accomplished by adding the sentence representation to each hidden layer h_i in the target recurrent language model. We next describe the procedure in more detail, starting with the CSM itself.

4.4.1 Convolutional Sentence Model

The CSM models the continuous representation of a sentence based on the continuous representations of the words in the sentence. Let $\mathsf{e} = \mathsf{e}_1 ... \mathsf{e}_k$ be a sentence in a

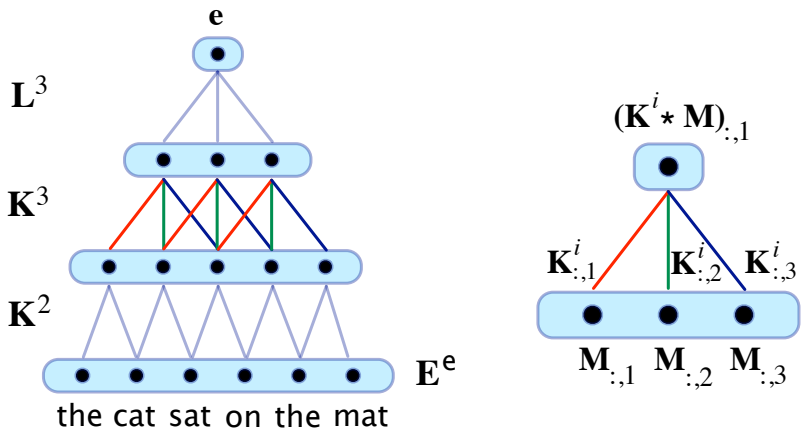

Figure 4.2: A CSM for a six word source sentence e and the computed sentence representation $\mathbf{e}$. $\mathbf{K}^2, \mathbf{K}^3$ are weight matrices and $\mathbf{L}^3$ is a top weight matrix. To the right, an instance of a one-dimensional convolution between some weight matrix $\mathbf{K}^i$ and a generic matrix $\mathbf{M}$ that could for instance correspond to $\mathbf{E}^{\mathsf{e}}_2$. The color coding of weights indicates weight sharing.

language and let $\mathsf{v}(\mathsf{e}_i) \in \mathbb{R}^{q \times 1}$ be the continuous representation of the word e_i. Let $\mathbf{E}^{\mathsf{e}} \in \mathbb{R}^{q \times k}$ be the *sentence matrix* for e defined by,

$$\mathbf{E}^{\mathsf{e}}_{:,i} = \mathsf{v}(\mathsf{e}_i) \tag{4.5}$$

The main component of the architecture of the CSM is a sequence of *weight* matrices $(\mathbf{K}^i)_{2 \leq i \leq r}$ that correspond to the kernels or filters of the convolution and can be thought of as learnt feature detectors. From the sentence matrix $\mathbf{E}^{\mathsf{e}}$ the CSM computes a continuous vector representation $\mathbf{e} \in \mathbb{R}^{q \times 1}$ for the sentence e by applying a sequence of convolutions to $\mathbf{E}^{\mathsf{e}}$ whose weights are given by the weight matrices. The weight matrices and the sequence of convolutions are defined next.

We denote by $(\mathbf{K}^i)_{2 \leq i \leq r}$ a sequence of weight matrices where each $\mathbf{K}^i \in \mathbb{R}^{q \times i}$ is a matrix of i columns and $r = \lceil \sqrt{2N} \rceil$, where N is the length of the longest source sentence in the training set. Each row of $\mathbf{K}^i$ is a vector of i weights that is treated as the kernel or filter of a *one-dimensional* convolution. Given for instance a matrix $\mathbf{M} \in \mathbb{R}^{q \times j}$ where the number of columns $j \geq i$, each row of $\mathbf{K}^i$ can be convolved

with the corresponding row in $\mathbf{M}$, resulting in a matrix $\mathbf{K}^i * \mathbf{M}$, where $*$ indicates the convolution operation and $(\mathbf{K}^i * \mathbf{M}) \in \mathbb{R}^{q \times (j-i+1)}$. For $i = 3$, the value $(\mathbf{K}^i * \mathbf{M})_{:,a}$ is computed by:

$$\mathbf{K}^i_{:,1} \odot \mathbf{M}_{:,a} + \mathbf{K}^i_{:,2} \odot \mathbf{M}_{:,a+1} + \mathbf{K}^i_{:,3} \odot \mathbf{M}_{:,a+2} \tag{4.6}$$

where $\odot$ is component-wise vector product. Applying the convolution kernel $\mathbf{K}^i$ yields a matrix $(\mathbf{K}^i * \mathbf{M})$ that has $i - 1$ columns less than the original matrix $\mathbf{M}$.

Given a source sentence of length k, the CSM convolves successively with the sentence matrix $\mathbf{E}^{\mathsf{e}}$ the sequence of weight matrices $(\mathbf{K}^i)_{2 \leq i \leq r}$, one after the other starting with $\mathbf{K}^2$ as follows:

$$\mathbf{E}^{\mathsf{e}}_1 = \mathbf{E}^{\mathsf{e}} \tag{4.7a}$$

$$\mathbf{E}^{\mathsf{e}}_{i+1} = \sigma(\mathbf{K}^{i+1} * \mathbf{E}^{\mathsf{e}}_i) \tag{4.7b}$$

After a few convolution operations, $\mathbf{E}^{\mathsf{e}}_i$ is either a vector in $\mathbb{R}^{q \times 1}$, in which case we obtained the desired representation, or the number of columns in $\mathbf{E}^{\mathsf{e}}_i$ is smaller than the number $i + 1$ of columns in the next weight matrix $\mathbf{K}^{i+1}$. In the latter case, we equally obtain a vector in $\mathbb{R}^{q \times 1}$ by simply applying a top weight matrix $\mathbf{L}^j$ that has the same number of columns as $\mathbf{E}^{\mathsf{e}}_i$. We thus obtain a sentence representation $\mathbf{e} \in \mathbb{R}^{q \times 1}$ for the source sentence e. Note that the convolution operations in Eq. 4.7b are interleaved with non-linear functions σ. Note also that, given the different levels at which the weight matrices $\mathbf{K}^i$ and $\mathbf{L}^i$ are applied, the top weight matrix $\mathbf{L}^j$ comes from an additional sequence of weight matrices $(\mathbf{L}^i)_{2 \leq i \leq r}$ distinct from $(\mathbf{K}^i)_{2 \leq i \leq r}$. Fig. 4.2 depicts an instance of the CSM and of a one-dimensional convolution.[1]

[1]For a formal treatment of the construction, see Ch. 3.

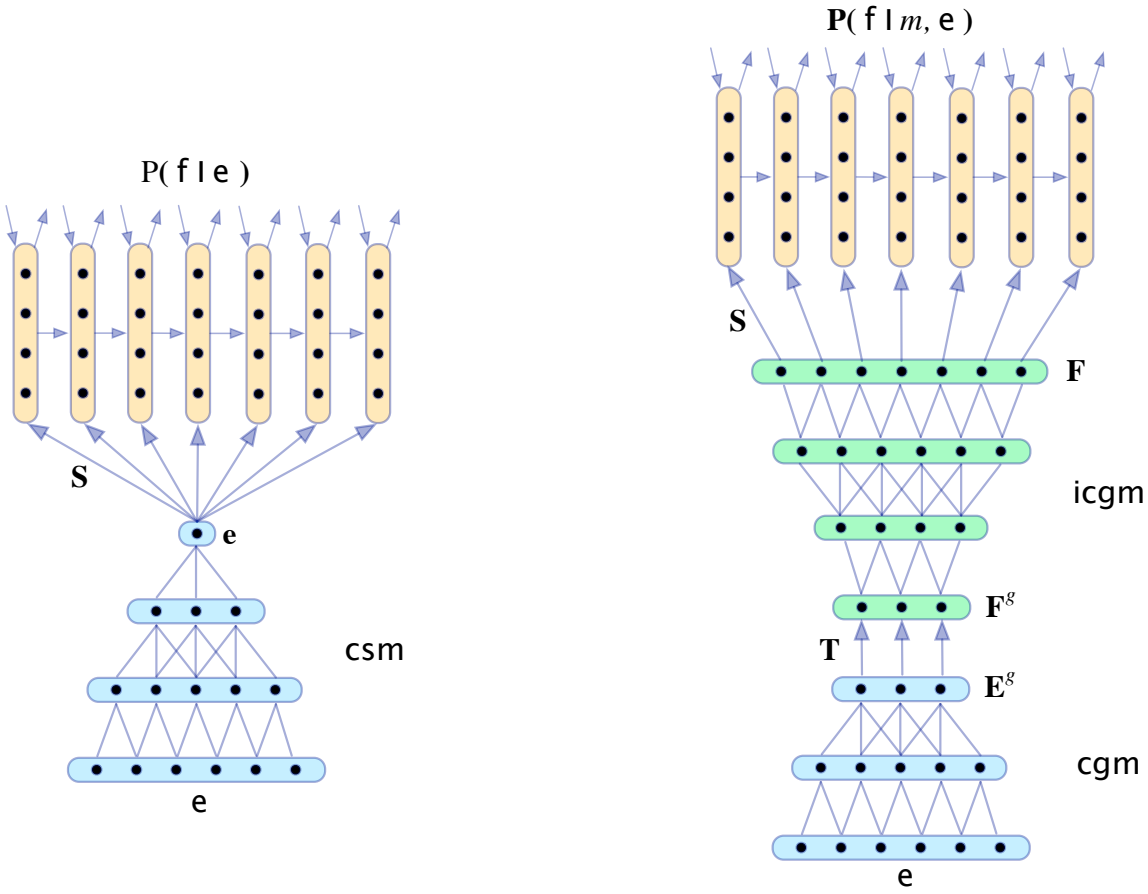

Figure 4.3: A graphical depiction of the two RCTMs. Arrows represent full matrix transformations while lines are vector transformations corresponding to columns of weight matrices.

4.4.2 RCTM I

As defined in Sect. 4.3, the RCTM I models the conditional probability $P(\mathsf{f}|\mathsf{e})$ of a sentence $\mathsf{f} = \mathsf{f}_1, ..., \mathsf{f}_m$ in a target language F being the translation of a sentence $\mathsf{e} = \mathsf{e}_1, ..., \mathsf{e}_k$ in a source language E. According to Eq. 4.1, the RCTM I explicitly computes the conditional distributions $P(\mathsf{f}_i|\mathsf{f}_{1:i-1}, \mathsf{e})$. The architecture of the RCTM I comprises a source vocabulary V^{E} and a target vocabulary V^{F}, two sequences of weight matrices $(\mathbf{K}^i)_{2\leq i\leq r}$ and $(\mathbf{L}^i)_{2\leq i\leq r}$ that are part of the constituent CSM, transformations $\mathbf{I} \in \mathbb{R}^{q\times|V^{\mathsf{F}}|}$, $\mathbf{R} \in \mathbb{R}^{q\times q}$ and $\mathbf{O} \in \mathbb{R}^{|V^{\mathsf{F}}|\times q}$ that are part of the constituent RLM and a sentence transformation $\mathbf{S} \in \mathbb{R}^{q\times q}$. We write $\mathbf{e} = \mathsf{csm}(\mathsf{e})$ for the output of the CSM with e as the input sentence.

The computation of the RCTM I is a simple modification to the computation of the RLM described in Eq. 4.3. It proceeds recursively as follows:

$$\mathbf{s} = \mathbf{S} \cdot \mathsf{csm}(\mathsf{e}) \tag{4.8a}$$

$$h_1 = \sigma(\mathbf{I} \cdot \mathsf{v}(\mathsf{f}_1) + \mathbf{s}) \tag{4.8b}$$

$$h_{i+1} = \sigma(\mathbf{R} \cdot h_i + \mathbf{I} \cdot \mathsf{v}(\mathsf{f}_{i+1}) + \mathbf{s}) \tag{4.8c}$$

$$o_{i+1} = \mathbf{O} \cdot h_i \tag{4.8d}$$

and the conditional distributions $P(\mathsf{f}_{i+1}|\mathsf{f}_{1:i}, \mathsf{e})$ are obtained from o_i as in Eq. 4.4. σ is a nonlinear function and bias values are included throughout the computation. Fig. 4.3 illustrates an RCTM I.

Two aspects of the RCTM I are to be remarked. First, the length of the target sentence is predicted by the target RLM itself that by its architecture has a bias towards shorter sentences. Secondly, the representation of the source sentence e constraints uniformly all the target words, contrary to the fact that the target words depend more strongly on certain parts of the source sentence and less on other parts. The next model proposes an alternative formulation of these aspects.

4.5 Recurrent Continuous Translation Model II

The central idea behind the RCTM II is to first estimate the length m of the target sentence independently of the main architecture. Given m and the source sentence e, the model constructs a representation for the n-grams in e, where n is set to 4. Note that each level of the CSM yields n-gram representations of e for a specific value of n. The 4-gram representation of e is thus constructed by truncating the CSM at the level that corresponds to $n = 4$. The procedure is then inverted. From the 4-gram representation of the source sentence e, the model builds a representation of a sentence that has the predicted length m of the target. This is similarly accomplished by truncating the *inverted* CSM for a sentence of length m.

We next describe in detail the Convolutional n-gram Model (CGM). Then we return to specify the RCTM II.

4.5.1 Convolutional n-gram model

The CGM is obtained by truncating the CSM at the level where n-grams are represented for the chosen value of n. A column $\mathbf{g}$ of a matrix $\mathbf{E}_i^{\mathsf{e}}$ obtained according to Eq. 4.7a represents an n-gram from the source sentence e. The value of n corresponds to the number of word vectors from which the n-gram representation $\mathbf{g}$ is constructed; equivalently, n is the span of the weights in the CSM underneath $\mathbf{g}$ (see Fig. 4.2-4.3). Note that any column in a matrix $\mathbf{E}_i^{\mathsf{e}}$ represents an n-gram with the same span value n. We denote by $\mathsf{gram}(\mathbf{E}_i^{\mathsf{e}})$ the size of the n-grams represented by $\mathbf{E}_i^{\mathsf{e}}$. For example, for a sufficiently long sentence e, $\mathsf{gram}(\mathbf{E}_2^{\mathsf{e}}) = 2$, $\mathsf{gram}(\mathbf{E}_3^{\mathsf{e}}) = 4$, $\mathsf{gram}(\mathbf{E}_4^{\mathsf{e}}) = 7$. We denote by $\mathsf{cgm}(\mathsf{e}, n)$ that matrix $\mathbf{E}_i^{\mathsf{e}}$ from the CSM that represents the n-grams of the source sentence e.

The CGM can also be inverted to obtain a representation for a sentence from the representation of its n-grams. We denote by icgm the inverse CGM, which depends

on the size of the n-gram representation $\mathsf{cgm}(\mathsf{e}, n)$ and on the target sentence length m. The transformation icgm unfolds the n-gram representation onto a representation of a target sentence with m words. The architecture corresponds to an inverted CGM or, equivalently, to an inverted truncated CSM (Fig. 4.3). Given the transformations cgm and icgm, we now detail the computation of the RCTM II.

4.5.2 RCTM II

The RCTM II models the conditional probability $P(\mathsf{f}|\mathsf{e})$ by factoring it as follows:

$$P(\mathsf{f}|\mathsf{e}) = P(\mathsf{f}|m, \mathsf{e}) \cdot P(m|\mathsf{e}) \tag{4.9a}$$

$$= \prod_{i=1}^{m} P(\mathsf{f}_{i+1}|\mathsf{f}_{1:i}, m, \mathsf{e}) \cdot P(m|\mathsf{e}) \tag{4.9b}$$

and computing the distributions $P(\mathsf{f}_{i+1}|\mathsf{f}_{1:i}, m, \mathsf{e})$ and $P(m|\mathsf{e})$. The architecture of the RCTM II comprises all the elements of the RCTM I together with the following additional elements: a translation transformation $\mathbf{T}^{q \times q}$ and two sequences of weight matrices $(\mathbf{J}^i)_{2 \leq i \leq s}$ and $(\mathbf{H}^i)_{2 \leq i \leq s}$ that are part of the icgm[2].

The computation of the RCTM II proceeds recursively as follows:

$$\mathbf{E}^g = \mathsf{cgm}(\mathsf{e}, 4) \tag{4.10a}$$

$$\mathbf{F}^g_{:,j} = \sigma(\mathbf{T} \cdot \mathbf{E}^g_{:,j}) \tag{4.10b}$$

$$\mathbf{F} = \mathsf{icgm}(\mathbf{F}^g, m) \tag{4.10c}$$

$$h_1 = \sigma(\mathbf{I} \cdot \mathsf{v}(\mathsf{f}_1) + \mathbf{S} \cdot \mathbf{F}_{:,1}) \tag{4.10d}$$

$$h_{i+1} = \sigma(\mathbf{R} \cdot h_i + \mathbf{I} \cdot \mathsf{v}(\mathsf{f}_{i+1}) + \mathbf{S} \cdot \mathbf{F}_{:,i+1}) \tag{4.10e}$$

$$o_{i+1} = \mathbf{O} \cdot h_i \tag{4.10f}$$

[2]Just like r the value s is small and depends on the length of the source and target sentences in the training set. See Sect. 4.6

and the conditional distributions $P(\mathsf{f}_{i+1}|\mathsf{f}_{1:i}, \mathsf{e})$ are obtained from o_i as in Eq. 4.4. Note how each reconstructed vector $\mathbf{F}_{:,i}$ is added successively to the corresponding layer h_i that predicts the target word f_i. The RCTM II is illustrated in Fig. 4.3.

For the separate estimation of the length of the translation, we estimate the conditional probability $P(m|\mathsf{e})$ by letting,

$$P(m|\mathsf{e}) = P(m|k) = \mathsf{Poisson}(\lambda_k) \tag{4.11}$$

where k is the length of the source sentence e and $\mathsf{Poisson}(\lambda)$ is a Poisson distribution with mean λ. This concludes the description of the RCTM II. We now turn to the experiments.

4.6 Experiments

We report on four experiments. The first experiment considers the perplexities of the models with respect to reference translations. The second and third experiments test the sensitivity of the RCTM II to the linguistic aspects of the source sentences. The final experiment tests the rescoring performance of the two models.

4.6.1 Training

Before turning to the experiments, we describe the data sets, hyper parameters and optimisation algorithms used for the training of the RCTMs.

4.6.1.1 Data sets

The training set used for all the experiments comprises a bilingual corpus of 144953 pairs of sentences less than 80 words in length from the news commentary section of the Eighth Workshop on Machine Translation (WMT) 2013 training data. The source language is English and the target language is French. The English sentences

contain about 4.1M words and the French ones about 4.5M words. Words in both the English and French sentences that occur twice or less are substituted with the $\langle unknown \rangle$ token. The resulting vocabularies V^{E} and V^{F} contain, respectively, 25403 English words and 34831 French words.

For the experiments we use four different test sets comprised of the Workshop on Machine Translation News Test (WMT-NT) sets for the years 2009, 2010, 2011 and 2012. They contain, respectively, 2525, 2489, 3003 and 3003 pairs of English-French sentences. For the perplexity experiments unknown words occurring in these data sets are replaced with the $\langle unknown \rangle$ token. The respective 2008 WMT-NT set containing 2051 pairs of English-French sentences is used as the validation set throughout.

4.6.1.2 Model hyperparameters

The parameter q that defines the size of the English vectors $\mathsf{v}(\mathsf{e}_i)$ for $\mathsf{e}_i \in V^{\mathsf{E}}$, the size of the hidden layer h_i and the size of the French vectors $\mathsf{v}(\mathsf{f}_i)$ for $\mathsf{v}(\mathsf{f}_i) \in V^{\mathsf{F}}$ is set to $q = 256$. This yields a relatively small recurrent matrix and corresponding models. To speed up training, we factorize the target vocabulary V^{F} into 256 classes following the procedure in [Mikolov et al., 2011].

The RCTM II uses a convolutional n-gram model CGM where n is set to 4. For the RCTM I, the number of weight matrices r for the CSM is 15, whereas in the RCTM II the number r of weight matrices for the CGM is 7 and the number s of weight matrices for the inverse CGM is 9. If a test sentence is longer than all training sentences and a larger weight matrix is required by the model, the larger weight matrix is easily factorized into two smaller weight matrices whose weights have been trained. For instance, if a weight matrix of 10 weights is required, but weight matrices have been trained only up to weight 9, then one can factorize the matrix of 10 weights with one of 9 and one of 2. Across all test sets the proportion of sentence pairs that

require larger weight matrices to be factorized into smaller ones is $< 0.1\%$.

4.6.1.3 Objective and optimisation

The objective function is the average of the sum of the cross-entropy errors of the predicted words and the true words in the French sentences. The English sentences are taken as input in the prediction of the French sentences, but they are not themselves ever predicted. An l_2 regularisation term is added to the objective. The training of the model proceeds by back-propagation through time. The cross-entropy error calculated at the output layer at each step is back-propagated through the recurrent structure for a number d of steps; for all models we let $d = 6$. The error accumulated at the hidden layers is then further back-propagated through the transformation $\mathbf{S}$ and the CSM/CGM to the input vectors $\mathsf{v}(\mathsf{e}_i)$ of the English input sentence e. All weights, including the English vectors, are randomly initialised and inferred during training.

The objective is minimised using mini-batch adaptive gradient descent (Adagrad) [Duchi et al., 2011]. The training of an RCTM takes about 15 hours on 3 multicore CPUs. While our experiments are relatively small, we note that in principle our models should scale similarly to RLMs which have been applied to hundreds of millions of words.

4.6.2 Perplexity of gold translations

Since the computation of the probability of a translation under one of the RCTMs is efficient, we can compute the perplexities of the RCTMs with respect to the reference translations in the test sets. The perplexity measure is an indication of the quality that a model assigns to a translation. We compare the perplexities of the RCTMs with the perplexity of the IBM Model 1 [Brown et al., 1993] and of the Fast-Aligner (FA-IBM 2) model that is a state-of-the-art variant of IBM Model 2 [Dyer et al., 2013].

WMT-NT	*2009*	*2010*	*2011*	*2012*
KN-5	218	213	222	225
RLM	178	169	178	181
IBM 1	207	200	188	197
FA-IBM 2	153	146	135	144
RCTM I	143	134	140	142
RCTM II	**86**	**77**	**76**	**77**

Table 4.1: Perplexity results on the WMT-NT sets.

WMT-NT PERM	*2009*	*2010*	*2011*	*2012*
RCTM II	174	168	175	178

Table 4.2: Perplexity results of the RCTM II on the WMT-NT sets where the words in the English source sentences are randomly permuted.

We add as baselines the unconditional target RLM and a 5-gram target language model with modified Kneser-Nay smoothing (KN-5). The results are reported in Tab. 4.1. The RCTM II obtains a perplexity that is $> 43\%$ lower than that of the alignment based models and that is 40% lower than the perplexity of the RCTM I. The low perplexity of the RCTMs suggests that continuous representations and the transformations between them make up well for the lack of explicit alignments. Further, the difference in perplexity between the RCTMs themselves demonstrates the importance of the conditioning architecture and suggests that the localised 4-gram conditioning in the RCTM II is superior to the conditioning with the whole source sentence of the RCTM I.

4.6.3 Sensitivity to source sentence structure

The second experiment aims at showing the sensitivity of the RCTM II to the order and position of words in the English source sentence. To this end, we randomly permute in the training and testing sets the words in the English source sentence. The results on the permuted data are reported in Tab. 4.2. If the RCTM II were

roughly comparable to a bag-of-words approach, there would be no difference under the permutation of the words. By contrast, the difference of the results reported in Tab. 4.2 with those reported in Tab. 4.1 is very significant, clearly indicating the sensitivity to word order and position of the translation model.

4.6.3.1 Generating from the RCTM II

To show that the RCTM II is sensitive not only to word order, but also to other syntactic and semantic traits of the sentence, we generate and inspect candidate translations for various English source sentences. The generation proceeds by sampling from the probability distribution of the RCTM II itself and does not depend on any other external resources. Given an English source sentence e, we let m be the length of the gold translation and we search the distribution computed by the RCTM II over all sentences of length m. The number of possible target sentences of length m amounts to $|V|^m = 34831^m$ where $V = V^{\mathsf{F}}$ is the French vocabulary; directly considering all possible translations is intractable. We proceed as follows: we sample with replacement 2000 sentences from the distribution of the RCTM II, each obtained by predicting one word at a time. We start by predicting a distribution for the first target word, restricting that distribution to the top 5 most probable words and sampling the first word of a candidate translation from the restricted distribution of 5 words. We proceed similarly for the remaining words. Each sampled sentence has a well-defined probability assigned by the model and can thus be ranked. Table 4.3 gives various English source sentences and some candidate French translations generated by the RCTM II together with their ranks.

English source sentence	**French gold translation**	**RCTM II candidate translation**	**Rank**
the patient is sick .	le patient est malade .	le patient est insuffisante .	1
		le patient est mort .	4
		la patient est insuffisante .	23
the patient is dead .	le patient est mort .	le patient est mort .	1
		le patient est dépassé .	4
the patient is ill .	le patient est malade .	le patient est mal .	3
the patients are sick .	les patients sont malades .	les patients sont confrontés .	2
		les patients sont corrompus .	5
the patients are dead .	les patients sont morts .	les patients sont morts .	1
the patients are ill .	les patients sont malades .	les patients sont confrontés .	5
the patient was ill .	le patient était malade .	le patient était mal .	2
the patients are not dead .	les patients ne sont pas morts .	les patients ne sont pas morts .	1
the patients are not sick .	les patients ne sont pas malades .	les patients ne sont pas $\langle unknown \rangle$.	1
		les patients ne sont pas mal .	6
the patients were saved .	les patients ont été sauvés .	les patients ont été sauvées .	6

Table 4.3: English source sentences, respective translations in French and candidate translations generated from the RCTM II and ranked out of 2000 samples according to their decreasing probability. Note that end of sentence dots (.) are generated as part of the translation.

WMT-NT	*2009*	*2010*	*2011*	*2012*
RCTM I + WP	19.7	21.1	22.5	21.5
RCTM II + WP	19.8	21.1	22.5	21.7
`cdec` (12 features)	19.9	21.2	22.6	21.8

Table 4.4: Bleu scores on the WMT-NT sets of each RCTM linearly interpolated with a word penalty WP. The `cdec` system includes WP as well as five translation models and two language modelling features, among others.

The results in Tab. 4.3 show the remarkable syntactic agreements of the candidate translations; the large majority of the candidate translations are fully well-formed French sentences. Further, subtle syntactic features such as the singular or plural ending of nouns and the present and past tense of verbs are well correlated between the English source and the French candidate targets. Finally, the meaning of the English source is well transferred on the French candidate targets; where a correlation is unlikely or the target word is not in the French vocabulary, a semantically related word or synonym is selected by the model. All of these traits suggest that the RCTM II is able to capture a significant amount of both syntactic and semantic information from the English source sentence and successfully transfer it onto the French translation.

4.6.4 Rescoring and BLEU Evaluation

The fourth experiment tests the ability of the RCTM I and the RCTM II to choose the best translation among a large number of candidate translations produced by another system. We use the `cdec` system to generate a list of 1000 best candidate translations for each English sentence in the four WMT-NT sets. We compare the rescoring performance of the RCTM I and the RCTM II with that of the `cdec` itself. `cdec` employs 12 engineered features including, among others, 5 translation models, 2 language model features and a word penalty feature (WP). For the RCTMs we simply interpolate the log probability assigned by the models to the candidate translations with the word penalty feature WP, tuned on the validation data. The results of the

experiment are reported in Tab. 4.4.

While there is little variance in the resulting Bleu scores, the performance of the RCTMs shows that their probabilities correlate with translation quality. Combining a monolingual RLM feature with the RCTMs does not improve the scores, while reducing `cdec` to just one core translation probability and language model features drops its score by two to five tenths. These results indicate that the RCTMs have been able to learn both translation and language modelling distributions.

4.7 Discussion

We have introduced Recurrent Continuous Translation Models that comprise a class of purely continuous neural sentence-level translation models. We have shown the translation capabilities of these models and the low perplexities that they obtain with respect to reference translations. We have shown the ability of these models at capturing syntactic and semantic information and at estimating during reranking the quality of candidate translations.

The RCTMs offer great modelling flexibility due to the sensitivity of the continuous representations to conditioning information. The models also suggest a wide range of potential advantages and extensions, from being able to include discourse representations beyond the single sentence and multilingual source representations, to being able to model morphologically rich languages through character-level recurrences.

RCTMs were the first sentence-level neural translation models that were developed in the literature. Remarkably, from a definitional perspective, little has changed between RCTMs and current state-of-the-art neural translation models that are published (Ch. 5 and e.g. [Wu et al., 2016a, Vaswani et al., 2017]) or deployed in large-scale online translation services [Wu et al., 2016a]: the models have a source encoder over

a sentence matrix and an autoregressive target decoder, the models are all trained by maximum likelihood estimation and beam search is used to decode the output space of the models. Furthermore, despite the good performance of bidirectional recurrent encoders [Bahdanau et al., 2014], recent state-of-the-art neural translation models use convolutional (Ch. 5) or parallelizable attentional encoders like in the RCTMs; the ability to parallelize the encoding along the length of the sentence speeds up both the training and the inference. In the next chapter, we look at purely convolutional neural translation models where not just the encoder, but also the decoder can be run in parallel during training.

Chapter 5

Convolutional Encoder-Decoders

5.1 Summary

In the previous chapter we have defined encoder-decoder neural networks and the specific instance of neural translation models RCTM I and RCTM II. This chapter continues the investigation into neural translation models by looking at decoders that are convolutional as opposed to recurrent and by applying the models to raw sequences of characters instead of words. This gives rise to the ByteNet architecture. The ByteNet is a one-dimensional convolutional neural network that is composed of two parts, one to encode the source sequence and the other to decode the target sequence. The two network parts are connected by stacking the decoder on top of the encoder and preserving the temporal resolution of the sequences. To address the differing lengths of the source and the target, we introduce an efficient mechanism by which the decoder is dynamically unfolded over the representation of the encoder. The ByteNet uses dilation in the convolutional layers to increase its receptive field. The resulting network has two core properties: it runs in time that is linear in the length of the sequences and it sidesteps the need for excessive memorization. The ByteNet decoder attains state-of-the-art performance on character-level language modelling and

outperforms the previous best results obtained with recurrent networks. The ByteNet also achieves state-of-the-art performance on character-to-character machine translation on the English-to-German WMT translation task, surpassing comparable neural translation models that are based on recurrent networks with attentional pooling and run in quadratic time. We find that the latent alignment structure contained in the representations reflects the expected alignment between the tokens.

5.2 Background

In neural language modelling, a neural network estimates a distribution over sequences of words or characters that belong to a given language [Bengio et al., 2003b]. In neural machine translation, the network estimates a distribution over sequences in the target language conditioned on a given sequence in the source language. The network can be thought of as composed of two parts: a *source network* (the encoder) that encodes the source sequence into a representation and a *target network* (the decoder) that uses the representation of the source encoder to generate the target sequence (Ch. 4).

Recurrent neural networks (RNN) are powerful sequence models [Hochreiter and Schmidhuber, 1997] and are widely used in language modelling [Mikolov et al., 2010], yet they have a potential drawback. RNNs have an inherently serial structure that prevents them from being run in parallel along the sequence length during training and evaluation. Forward and backward signals in a RNN also need to traverse the full distance of the serial path to reach from one token in the sequence to another. The larger the distance, the harder it is to learn the dependencies between the tokens [Hochreiter et al., 2001].

A number of neural architectures have been proposed for modelling translation, such as encoder-decoder networks (Ch. 4) [Sutskever et al., 2014, Cho et al., 2014, Kaiser and Bengio, 2016], networks with attentional pooling [Bahdanau et al., 2014]

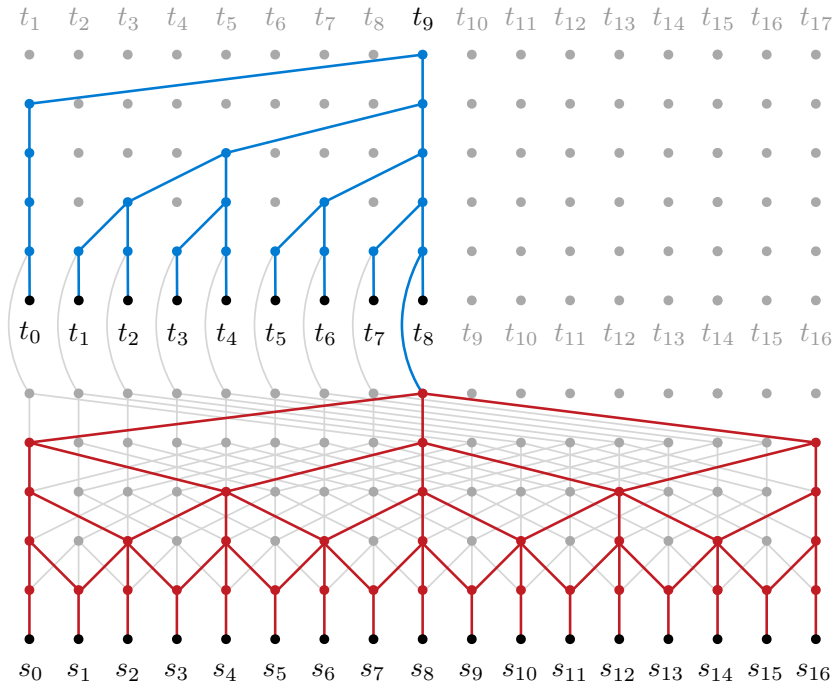

Figure 5.1: The architecture of the ByteNet. The target decoder (blue) is stacked on top of the source encoder (red). The decoder generates the variable-length target sequence using dynamic unfolding.

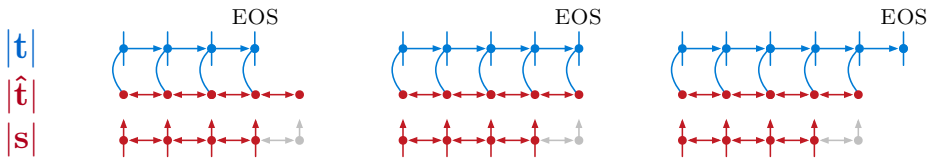

Figure 5.2: Dynamic unfolding in the ByteNet architecture. At each step the decoder is conditioned on the source representation produced by the encoder for that step, or simply on no representation for steps beyond the extended length $|\hat{\mathbf{t}}|$. The decoding ends when the target network produces an end-of-sequence (EOS) symbol.

and two-dimensional networks (see Ch. 6). Despite the generally good performance, the proposed models either have running time that is super-linear in the length of the source and target sequences, or they process the source sequence into a constant size representation, burdening the model with a memorization step. Both of these drawbacks grow more severe as the length of the sequences increases.

We present a family of encoder-decoder neural networks that are characterized by two architectural mechanisms aimed to address the drawbacks of the conventional approaches mentioned above. The first mechanism involves the *stacking* of the decoder on top of the representation of the encoder in a manner that preserves the temporal resolution of the sequences; this is in contrast with architectures that encode the source into a fixed-size representation (Ch. 4) [Sutskever et al., 2014]. The second mechanism is the *dynamic unfolding* mechanism that allows the network to process in a simple and efficient way source and target sequences of different lengths (Sect. 5.4.2).

The ByteNet is the instance within this family of models that uses one-dimensional convolutional neural networks of fixed depth for both the encoder and the decoder (Fig. 5.1). The two CNNs use increasing factors of dilation to rapidly grow the receptive fields; a similar technique was later used also in [van den Oord et al., 2016]. The convolutions in the decoder CNN are masked to prevent the network from seeing future tokens in the target sequence.

The network has beneficial computational and learning properties. From a computational perspective, the network has a running time that is *linear* in the length of the source and target sequences (up to a constant $c \approx \log d$ where d is the size of the desired dependency field). The computation in the encoder during both training and decoding, as well as in the decoder during training only, can also be run efficiently *in parallel* along the sequences (Sect. 5.3). From a learning perspective, the representation of the source sequence in the ByteNet is *resolution preserving*;

the representation sidesteps the need for memorization and allows for maximal bandwidth between encoder and decoder. In addition, the distance traversed by forward and backward signals between any input and output tokens corresponds to the fixed depth of the networks and is largely independent of the distance between the tokens. Dependencies over large distances are connected by short paths and can be learnt more easily.

We apply the ByteNet model to strings of characters for character-level language modelling and character-to-character machine translation. We evaluate the decoder network on the Hutter Prize Wikipedia task [Hutter, 2012] where it achieves the state-of-the-art performance of 1.31 bits/character. We further evaluate the encoder-decoder network on character-to-character machine translation on the English-to-German WMT benchmark where it achieves a state-of-the-art BLEU score of 22.85 (0.380 bits/character) and 25.53 (0.389 bits/character) on the 2014 and 2015 test sets, respectively. On the character-level machine translation task, ByteNet betters a comparable version of GNMT [Wu et al., 2016a] that is a state-of-the-art system. These results show that deep CNNs are simple, scalable and effective architectures for challenging linguistic processing tasks.

The chapter is organized as follows. Section 2 lays out the background and some desiderata for neural architectures underlying translation models. Section 3 defines the proposed family of architectures and the specific convolutional instance (ByteNet) used in the experiments. Section 4 analyses ByteNet as well as existing neural translation models based on the desiderata set out in Section 2. Section 5 reports the experiments on language modelling and Section 6 reports the experiments on character-to-character machine translation.

5.3 Neural Translation Model

In order to frame the discussion, we briefly recall the definition of a neural translation model. Given a string $\mathbf{s}$ from a source language, a neural translation model estimates a distribution $p(\mathbf{t}|\mathbf{s})$ over strings $\mathbf{t}$ of a target language. The distribution indicates the probability of a string $\mathbf{t}$ being a translation of $\mathbf{s}$. A product of conditionals over the tokens in the target $\mathbf{t} = t_0, ..., t_N$ leads to a tractable formulation of the distribution. Each conditional factor expresses complex and long-range dependencies among the source and target tokens. The strings are usually sentences of the respective languages; the tokens are words or, as in the our case, characters. The network that models $p(\mathbf{t}|\mathbf{s})$ is composed of two parts: a source network (the encoder) that processes the source string into a representation and a target network (the decoder) that uses the source representation to generate the target string (Ch. 4). The decoder functions as a language model for the target language.

A neural translation model has some basic properties. The decoder is autoregressive in the target tokens and the model is sensitive to the ordering of the tokens in the source and target strings. It is also useful for the model to be able to assign a non-zero probability to any string in the target language and retain an open vocabulary.

5.3.1 Desiderata

Beyond these basic properties the definition of a neural translation model does not determine a unique neural architecture, so we aim at identifying some desiderata.

First, the running time of the network should be *linear* in the length of the source and target strings. This ensures that the model is scalable to longer strings, which is the case when using characters as tokens.

The use of operations that run *in parallel* along the sequence length can also be beneficial for reducing computation time.

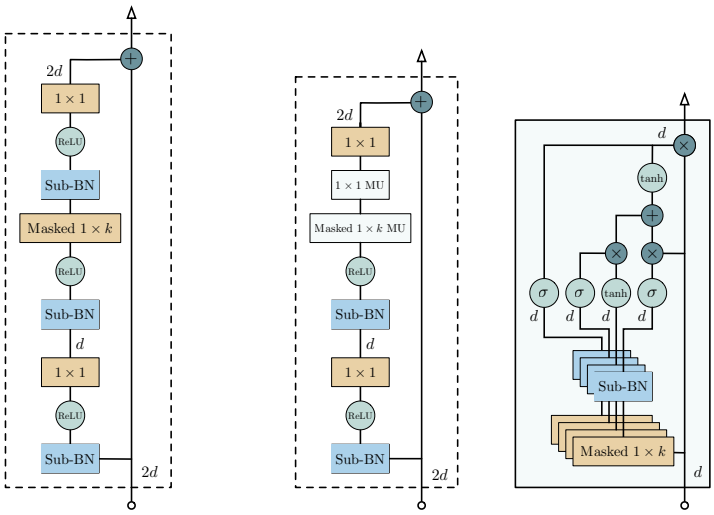

Figure 5.3: Left: Residual block with ReLUs [He et al., 2016] adapted for decoders. Right: Residual Multiplicative Block adapted for decoders and corresponding expansion of the MU (Ch. 7).

Second, the size of the source representation should be linear in the length of the source string, i.e. it should be *resolution preserving*, and not have constant size. This is to avoid burdening the model with an additional memorization step before translation. In more general terms, the size of a representation should be proportional to the amount of information it represents or predicts.

Third, the path traversed by forward and backward signals in the network (between input and ouput tokens) should be short. Shorter paths whose length is largely decoupled from the sequence distance between the two tokens have the potential to better propagate the signals [Hochreiter et al., 2001] and to let the network learn long-range dependencies more easily.

5.4 ByteNet

We aim at building neural language and translation models that capture the desiderata set out in Sect. 5.3.1. The proposed ByteNet architecture is composed of a decoder that is *stacked* on an encoder (Sect. 5.4.1) and generates variable-length outputs via

dynamic unfolding (Sect. 5.4.2). The decoder is a language model that is formed of one-dimensional convolutional layers that are masked (Sect. 5.4.4) and use dilation (Sect. 5.4.5). The encoder processes the source string into a representation and is formed of one-dimensional convolutional layers that use dilation but are *not* masked. Figure 5.1 depicts the two networks and their combination.

5.4.1 Encoder-Decoder Stacking

A notable feature of the proposed family of architectures is the way the encoder and the decoder are connected. To maximize the representational bandwidth between the encoder and the decoder, we place the decoder on top of the representation computed by the encoder. This is in contrast to models that compress the source representation into a fixed-size vector (Ch. 4) [Sutskever et al., 2014] or that pool over the source representation with a mechanism such as attentional pooling [Bahdanau et al., 2014].

5.4.2 Dynamic Unfolding

An encoder and a decoder network that process sequences of different lengths cannot be directly connected due to the different sizes of the computed representations. We circumvent this issue via a mechanism which we call dynamic unfolding, which works as follows.

Given source and target sequences $\mathbf{s}$ and $\mathbf{t}$ with respective lengths $|\mathbf{s}|$ and $|\mathbf{t}|$, one first chooses a sufficiently tight upper bound $|\hat{\mathbf{t}}|$ on the target length $|\mathbf{t}|$ as a linear function of the source length $|\mathbf{s}|$:

$$|\hat{\mathbf{t}}| = a|\mathbf{s}| + b \tag{5.1}$$

The tight upper bound $|\hat{\mathbf{t}}|$ is chosen in such a way that, on the one hand, it is greater than the actual length $|\mathbf{t}|$ in almost all cases and, on the other hand, it does not

increase excessively the amount of computation that is required. Once a linear relationship is chosen, one designs the source encoder so that, given a source sequence of length $|\mathbf{s}|$, the encoder outputs a representation of the established length $|\hat{\mathbf{t}}|$. In our case, we let $a = 1.20$ and $b = 0$ when translating from English into German, as German sentences tend to be somewhat longer than their English counterparts (Fig. 5.5). In this manner the representation produced by the encoder can be efficiently computed, while maintaining high bandwidth and being resolution-preserving. Once the encoder representation is computed, we let the decoder unfold step-by-step over the encoder representation until the decoder itself outputs an end-of-sequence symbol; the unfolding process may freely proceed beyond the estimated length $|\hat{\mathbf{t}}|$ of the encoder representation. Figure 5.2 gives an example of dynamic unfolding.

5.4.3 Input Embedding Tensor

Given the target sequence $\mathbf{t} = t_0, ..., t_n$ the ByteNet decoder embeds each of the first n tokens $t_0, ..., t_{n-1}$ via a look-up table (the n tokens $t_1, ..., t_n$ serve as targets for the predictions). The resulting embeddings are concatenated into a tensor of size $n \times 2d$ where d is the number of inner channels in the network.

5.4.4 Masked One-dimensional Convolutions

The decoder applies masked one-dimensional convolutions to the input embedding tensor that have a masked kernel of size k. The masking ensures that information from future tokens does not affect the prediction of the current token. The operation can be implemented either by zeroing out some of the weights of a wider kernel of size $2k - 1$ or by padding the input map.

Figure 5.4: Recurrent ByteNet variants of the ByteNet architecture. Left: Recurrent ByteNet with convolutional source network and recurrent target network. Right: Recurrent ByteNet with bidirectional recurrent source network and recurrent target network. The latter architecture is a strict generalization of the RNN Enc-Dec network.

5.4.5 Dilation

The masked convolutions use dilation to increase the receptive field of the target network [Chen et al., 2014, Yu and Koltun, 2015]. Dilation makes the receptive field grow exponentially in terms of the depth of the networks, as opposed to linearly. We use a dilation scheme whereby the dilation rates are doubled every layer up to a maximum rate r (for our experiments $r = 16$). The scheme is repeated multiple times in the network always starting from a dilation rate of 1 (see also Ch. 6 and Ch. 7 below for a similar construction in two dimensions).

5.4.6 Residual Blocks

Each layer is wrapped in a residual block that contains additional convolutional layers with filters of size 1×1 [He et al., 2016]. We adopt two variants of the residual blocks: one with ReLUs, which is used in the machine translation experiments, and one with Multiplicative Units, which is used in the language modelling experiments. Figure 5.3 diagrams the two variants of the blocks. In both cases, we use layer normalization [Ba et al., 2016] before the activation function, as it is well suited to sequence processing where computing the activation statistics over the following future tokens (as would be done by batch normalization) must be avoided. After a series of residual blocks of

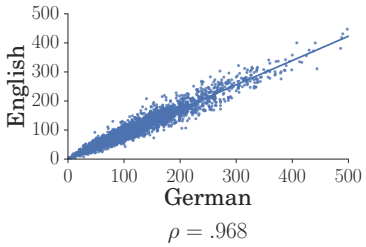

Figure 5.5: Lengths of sentences in characters and their correlation coefficient for the English-to-German WMT NewsTest-2013 validation data. The correlation coefficient is similarly high ($\rho > 0.96$) for all other language pairs that we inspected.

Model	$\mathbf{Net_S}$	$\mathbf{Net_T}$	**Time**	**RP**	$\mathbf{Path_S}$	$\mathbf{Path_T}$
RCTM 1	CNN	RNN	$\|S\|\|S\| + \|T\|$	no	$\|S\|$	$\|T\|$
RCTM 2	CNN	RNN	$\|S\|\|S\| + \|T\|$	yes	$\|S\|$	$\|T\|$
RNN Enc-Dec	RNN	RNN	$\|S\| + \|T\|$	no	$\|S\| + \|T\|$	$\|T\|$
RNN Enc-Dec Att	RNN	RNN	$\|S\|\|T\|$	yes	1	$\|T\|$
Grid LSTM	RNN	RNN	$\|S\|\|T\|$	yes	$\|S\| + \|T\|$	$\|S\| + \|T\|$
Extended Neural GPU	cRNN	cRNN	$\|S\|\|S\| + \|S\|\|T\|$	yes	$\|S\|$	$\|T\|$
Recurrent ByteNet	RNN	RNN	$\|S\| + \|T\|$	yes	$\max(\|S\|, \|T\|)$	$\|T\|$
Recurrent ByteNet	CNN	RNN	$c\|S\| + \|T\|$	yes	c	$\|T\|$
ByteNet	CNN	CNN	$c\|S\| + c\|T\|$	yes	c	c

Table 5.1: Properties of various neural translation models.

increased dilation, the network applies one more convolution and ReLU followed by a convolution and a final softmax layer.

5.5 Model Comparison

In this section we analyze the properties of various previously introduced neural translation models as well as the ByteNet family of models. For the sake of a more complete analysis, we include two recurrent ByteNet variants (which we do not evaluate in the experiments).

5.5.1 Recurrent ByteNets

The ByteNet is composed of two stacked encoder and decoder networks where the decoder network dynamically adapts to the output length. This way of combining the networks is not tied to the networks being strictly convolutional. We may consider two variants of the ByteNet that use recurrent networks for one or both of the networks (see Figure 5.4). The first variant replaces the convolutional decoder with a recurrent one that is similarly stacked and dynamically unfolded. The second variant also replaces the convolutional encoder with a recurrent encoder, e.g. a bidirectional RNN. The target RNN is then placed on top of the source RNN. Considering the latter Recurrent ByteNet, we can see that the RNN Enc-Dec network [Sutskever et al., 2014, Cho et al., 2014] is a Recurrent ByteNet where all connections between source and target – except for the first one that connects s_0 and t_0 – have been severed. The Recurrent ByteNet is a generalization of the RNN Enc-Dec and, modulo the type of weight-sharing scheme, so is the convolutional ByteNet.

5.5.2 Comparison of Properties

In our comparison we consider the following neural translation models: the Recurrent Continuous Translation Model (RCTM) 1 and 2 (Ch. 4); the RNN Enc-Dec [Sutskever et al., 2014, Cho et al., 2014]; the RNN Enc-Dec Att with the attentional pooling mechanism [Bahdanau et al., 2014] of which there are a few variations [Luong et al., 2015, Chung et al., 2016a]; the Grid LSTM translation model [Kalchbrenner et al., 2016] that uses a multi-dimensional architecture; the Extended Neural GPU model [Kaiser and Bengio, 2016] that has a convolutional RNN architecture; the ByteNet and the two Recurrent ByteNet variants.

Our comparison criteria reflect the desiderata set out in Sect. 5.3.1. We separate the first (computation time) desideratum into three columns. The first column indicates the time complexity of the network as a function of the length of the se-

quences and is denoted by **Time**. The other two columns $\mathbf{Net_S}$ and $\mathbf{Net_T}$ indicate, respectively, whether the source and the target network use a convolutional structure (CNN) or a recurrent one (RNN); a CNN structure has the advantage that it can be run in parallel along the length of the sequence. The second (resolution preservation) desideratum corresponds to the **RP** column, which indicates whether the source representation in the network is resolution preserving. Finally, the third desideratum (short forward and backward flow paths) is reflected by two columns. The $\mathbf{Path_S}$ column corresponds to the length in layer steps of the shortest path between a source token and any output target token. Similarly, the $\mathbf{Path_T}$ column corresponds to the length of the shortest path between an input target token and any output target token. Shorter paths lead to better forward and backward signal propagation.

Table 5.1 summarizes the properties of the models. The ByteNet, the Recurrent ByteNets and the RNN Enc-Dec are the only networks that have linear running time (up to the constant c). The RNN Enc-Dec, however, does not preserve the source sequence resolution, a feature that aggravates learning for long sequences such as those that appear in character-to-character machine translation [Luong and Manning, 2016]. The RCTM 2, the RNN Enc-Dec Att, the Grid LSTM and the Extended Neural GPU do preserve the resolution, but at a cost of a quadratic running time. The ByteNet stands out also for its **Path** properties. The dilated structure of the convolutions connects any two source or target tokens in the sequences by way of a small number of network layers corresponding to the depth of the source or target networks. For character sequences where learning long-range dependencies is important, paths that are sub-linear in the distance are advantageous.

Model	**Inputs**	**Outputs**	**WMT Test '14**	**WMT Test '15**
Phrase Based MT [Freitag et al., 2014, Williams et al., 2015]	phrases	phrases	20.7	24.0
RNN Enc-Dec [Luong et al., 2015]	words	words	11.3	
Reverse RNN Enc-Dec [Luong et al., 2015]	words	words	14.0	
RNN Enc-Dec Att [Zhou et al., 2016]	words	words	20.6	
RNN Enc-Dec Att [Luong et al., 2015]	words	words	20.9	
GNMT (RNN Enc-Dec Att) [Wu et al., 2016a]	word-pieces	word-pieces	**24.61**	
RNN Enc-Dec Att [Chung et al., 2016b]	BPE	BPE	19.98	21.72
RNN Enc-Dec Att [Chung et al., 2016b]	BPE	char	21.33	23.45
GNMT (RNN Enc-Dec Att) [Wu et al., 2016a]	char	char	22.62	
ByteNet	char	char	**23.75**	**26.26**

Table 5.2: BLEU scores on En-De WMT NewsTest 2014 and 2015 test sets.

Model	Test
Stacked LSTM [Graves, 2013]	1.67
GF-LSTM [Chung et al., 2015]	1.58
Grid-LSTM (Ch. 6)	1.47
Layer-normalized LSTM [Chung et al., 2016a]	1.46
MI-LSTM [Wu et al., 2016b]	1.44
Recurrent Memory Array Structures [Rocki, 2016]	1.40
HM-LSTM [Chung et al., 2016a]	1.40
Layer Norm HyperLSTM [Ha et al., 2016]	1.38
Large Layer Norm HyperLSTM [Ha et al., 2016]	1.34
Recurrent Highway Networks [Srivastava et al., 2015b]	1.32
ByteNet Decoder	**1.31**

Table 5.3: Negative log-likelihood results in bits/byte on the Hutter Prize Wikipedia benchmark.

5.6 Character Prediction

We first evaluate the ByteNet Decoder separately on a character-level language modelling benchmark. We use the Hutter Prize version of the Wikipedia dataset and follow the standard split where the first 90 million bytes are used for training, the next 5 million bytes are used for validation and the last 5 million bytes are used for testing [Chung et al., 2015]. The total number of characters in the vocabulary is 205.

The ByteNet Decoder that we use for the result has 30 residual blocks split into six sets of five blocks each; for the five blocks in each set the dilation rates are, respectively, $1, 2, 4, 8$ and 16. The masked kernel has size 3. This gives a receptive field of 315 characters. The number of hidden units d is 512. For this task we use residual multiplicative blocks (Fig. 5.3 Right). For the optimization we use Adam [Kingma and Ba, 2014] with a learning rate of 0.0003 and a weight decay term of 0.0001. We apply dropout to the last ReLU layer before the softmax dropping units with a probability of 0.1. We do not reduce the learning rate during training. At each step we sample a batch of sequences of 500 characters each, use the first 100 characters as the minimum context and predict the latter 400 characters.

	WMT Test '14	WMT Test '15
Bits/character	0.521	0.532
BLEU	23.75	26.26

Table 5.4: Bits/character with respective BLEU score achieved by the ByteNet translation model on the English-to-German WMT translation task.

Table 5.3 lists recent results of various neural sequence models on the Wikipedia dataset. All the results except for the ByteNet result are obtained using some variant of the LSTM recurrent neural network [Hochreiter and Schmidhuber, 1997]. The ByteNet decoder achieves 1.31 bits/character on the test set.

Director Jon Favreau, who is currently working on Disney's forthcoming Jungle Book film, told the website Hollywood Reporter: "I think times are changing."
Regisseur Jon Favreau, der derzeit an Disneys bald erscheinenden Dschungelbuch-Film arbeitet, sagte gegenber der Webseite Hollywood Reporter: "Ich glaube, die Zeiten ändern sich."
Regisseur Jon Favreau, der zur Zeit an Disneys kommendem Jungle Book Film arbeitet, hat der Website Hollywood Reporter gesagt: "Ich denke, die Zeiten ändern sich".
Matt Casaday, 25, a senior at Brigham Young University, says he had paid 42 cents on Amazon.com for a used copy of "Strategic Media Decisions: Understanding The Business End Of The Advertising Business."
Matt Casaday, 25, Abschlussstudent an der Brigham Young University, sagt, dass er auf Amazon.com 42 Cents ausgegeben hat für eine gebrauchte Ausgabe von "Strategic Media Decisions: Understanding The Business End Of The Advertising Business."
Matt Casaday, 25, ein Senior an der Brigham Young University, sagte, er habe 42 Cent auf Amazon.com für eine gebrauchte Kopie von "Strategic Media Decisions: Understanding The Business End Of The Advertising Business".

Table 5.5: Raw output translations generated from the ByteNet that highlight interesting reordering and transliteration phenomena. For each group, the first row is the English source, the second row is the ground truth German target, and the third row is the ByteNet translation.

5.7 Character-Level Machine Translation

We evaluate the full ByteNet on the WMT English to German translation task. We use NewsTest 2013 for validation and NewsTest 2014 and 2015 for testing. The English and German strings are encoded as sequences of characters; no explicit segmentation into words or morphemes is applied to the strings. The outputs of the network are strings of characters in the target language. We keep 323 characters in the German vocabulary and 296 in the English vocabulary.

The ByteNet used in the experiments has 30 residual blocks in the encoder and 30 residual blocks in the decoder. As in the ByteNet Decoder, the residual blocks are arranged in sets of five with corresponding dilation rates of $1, 2, 4, 8$ and 16. For this task we use the residual blocks with ReLUs (Fig. 5.3 Left). The number of hidden units d is 800. The size of the kernel in the source network is 3, whereas the size of the masked kernel in the target network is 3. For the optimization we use Adam with a learning rate of 0.0003.

Each sentence is padded with special characters to the nearest greater multiple of 50; 20% of further padding is applied to each source sentence as a part of dynamic unfolding (eq. 5.1). Each pair of sentences is mapped to a bucket based on the pair of padded lengths for efficient batching during training. We use *vanilla* beam search according to the total likelihood of the generated candidate and accept only candidates which end in a end-of-sentence token. We use a beam of size 12. We do not use length normalization, nor do we keep score of which parts of the source sentence have been translated [Wu et al., 2016a].

Table 5.2 and Table 5.4 contain the results of the experiments. On NewsTest 2014 the ByteNet achieves the highest performance in character-level and subword-level neural machine translation, and compared to the word-level systems it is second only to the version of GNMT that uses word-pieces. On NewsTest 2015, to our knowledge, ByteNet achieves the best published results to date.

Table 5.5 contains some of the unaltered generated translations from the ByteNet that highlight reordering and other phenomena such as transliteration. The character-level aspect of the model makes post-processing unnecessary in principle. We further visualize the sensitivity of the ByteNet's predictions to specific source and target inputs using gradient-based visualization [Simonyan et al., 2013]. Figure 5.6 represents a heatmap of the magnitude of the gradients of the generated outputs with respect to the source and target inputs. For visual clarity, we sum the gradients for all the characters that make up each word and normalize the values along each column. In contrast with the attentional pooling mechanism [Bahdanau et al., 2014], this general technique allows us to inspect not just dependencies of the outputs on the source inputs, but also dependencies of the outputs on previous target inputs, or on any other neural network layers.

This second part of the thesis concludes our treatment of encoder-decoder neural networks for learning to map structured linguistic items to other structured linguistic items. But structured items may not purely be linguistic; in fact, they might not be one-dimensional sequences at all. They could be general tensors of data. In Part III we extend these methods to the autoregressive generation of two-dimensional and three-dimensional tensors of natural image and video data.

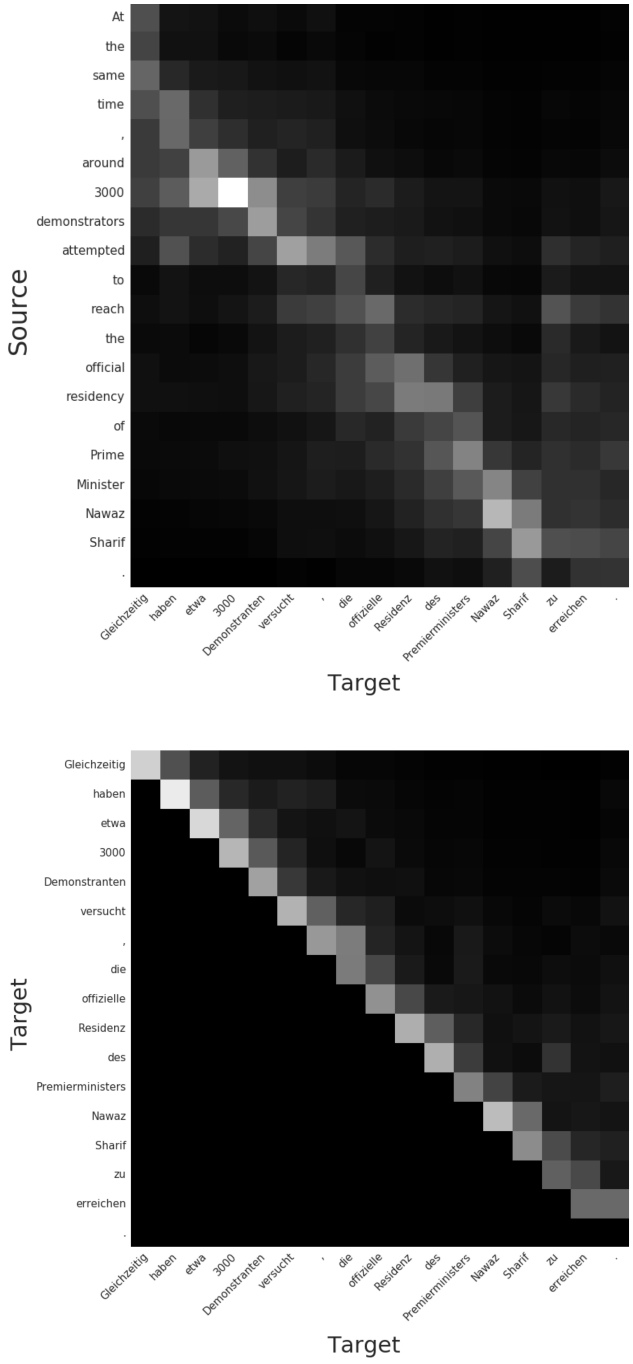

Figure 5.6: Magnitude of gradients of the predicted outputs with respect to the source and target inputs. The gradients are summed for all the characters in a given word. In the bottom heatmap the magnitudes are nonzero on the diagonal, since the prediction of a target character depends highly on the preceding target character in the same word.

Part III

Neural Decoders in the Visual Domain

Chapter 6

Pixel Recurrent Neural Networks

6.1 Summary

Part I and Part II have primarily dealt with one-dimensional sequences of natural language utterances. Few if any of the concepts are language specific. Conditional autoregressive modelling can be applied to other structured objects such as images with strong results. To this we turn in Part III.

Modelling the distribution of natural images is a landmark problem in unsupervised learning. This task requires an image model that is at once expressive, tractable and scalable. The present chapter describes a deep neural network that sequentially predicts the pixels in an image along the two spatial dimensions. The method models the discrete probability of the raw pixel values and encodes the complete set of dependencies in the image. Architectural novelties include fast two-dimensional recurrent layers and an effective use of residual connections in deep recurrent networks. We achieve log-likelihood scores on natural images that are considerably better than the previous state of the art. Our main results also provide benchmarks on the diverse ImageNet dataset. Samples generated from the model appear crisp, varied and globally coherent.

6.2 Background

Generative image Modelling is a central problem in unsupervised learning. Probabilistic density models can be used for a wide variety of tasks that range from image compression and forms of reconstruction such as image inpainting (e.g., see Figure 6.1) and deblurring, to generation of new images. When the model is conditioned on external information, possible applications also include creating images based on text descriptions or simulating future frames in a planning task. One of the great advantages in generative modelling is that there are practically endless amounts of image data available to learn from. However, because images are high dimensional and highly structured, estimating the distribution of natural images is extremely challenging.

One of the most important obstacles in generative modelling is building complex and expressive models that are also tractable and scalable. This trade-off has resulted in a large variety of generative models, each having their advantages. Most work focuses on stochastic latent variable models such as VAE's [Rezende et al., 2014, Kingma and Welling, 2013] that aim to extract meaningful representations, but often come with an intractable inference step that can hinder their performance.

One effective approach to *tractably* model a joint distribution of the pixels in the image is to cast it as a product of conditional distributions; this approach has been adopted in non-parametric patch-based sequential models [Efros and Leung, 1999], in autoregressive models such as NADE [Larochelle and Murray, 2011] and fully visible neural networks [Neal, 1992]. The factorization turns the joint modelling problem into a sequence problem, where one learns to predict the next pixel given all the previously generated pixels. But to model the highly nonlinear and long-range correlations between pixels and the complex conditional distributions that result, a highly expressive sequence model is necessary.

As we have seen above, RNNs are powerful models that offer a compact, shared

Figure 6.1: Image completions sampled from a PixelRNN.

parametrization of a series of conditional distributions. RNNs have been shown to excel at hard sequence problems ranging from handwriting generation [Graves, 2013], to character prediction [Sutskever et al., 2011b] and, as we have seen above, as decoders in neural machine translation (Ch. 4). A two-dimensional RNN has produced very promising results in modelling grayscale images and textures [Theis and Bethge, 2015].

In this chapter we advance two-dimensional RNNs and apply them to large-scale modelling of natural images. The resulting *PixelRNNs* are composed of up to twelve, fast two-dimensional Long Short-Term Memory (LSTM) layers. These layers use LSTM units in their state [Hochreiter and Schmidhuber, 1997, Graves and Schmidhuber, 2009] and adopt a convolution to compute at once all the states along one of the spatial dimensions of the data. We design two types of these layers. The first type is the *Row LSTM* layer where the convolution is applied along each row; a similar technique is described in [Stollenga et al., 2015]. The second type is the *Diagonal BiLSTM* layer where the convolution is applied in a novel fashion along the diagonals of the image. The networks also incorporate *residual connections* [He et al., 2016] around LSTM layers; we observe that this helps with training of the PixelRNN for up to twelve layers of depth.

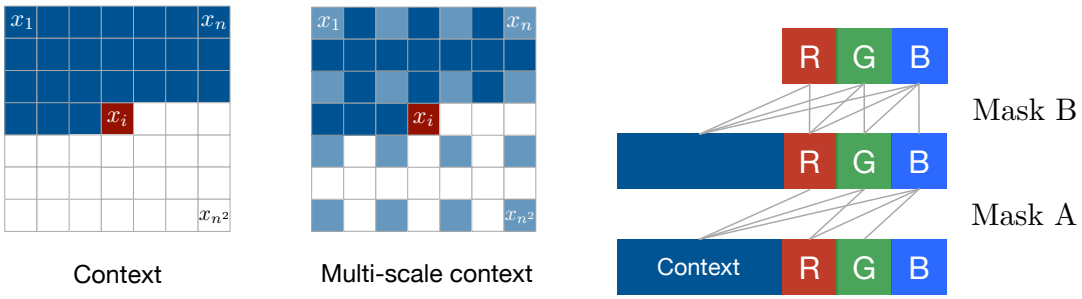

Figure 6.2: **Left**: To generate pixel x_i one conditions on all the previously generated pixels left and above of x_i. **Center**: To generate a pixel in the multi-scale case we can also condition on the subsampled image pixels (in light blue). **Right**: Diagram of the connectivity inside a masked convolution. In the first layer, each of the RGB channels is connected to previous channels and to the context, but is not connected to itself. In subsequent layers, the channels are also connected to themselves.

We also consider a second, simplified architecture which shares the same core components as the PixelRNN. We observe that two-dimensional CNNs can also be used as sequence models with a fixed dependency range, by using *masking* in a two-dimensional setting, in an analogous way to masked one-dimensional convolutional (Ch. 5). The *PixelCNN* architecture is a fully convolutional network of fifteen layers that preserves the spatial resolution of its input throughout the layers and outputs a conditional distribution at each location.

Both PixelRNN and PixelCNN capture the full generality of pixel inter-dependencies without introducing independence assumptions as in e.g., latent variable models. The dependencies are also maintained between the RGB color values within each individual pixel. Furthermore, in contrast to previous approaches that model the pixels as continuous values (e.g., [Theis and Bethge, 2015, Gregor et al., 2014]), we model the pixels as *discrete* values using a multinomial distribution implemented with a simple softmax layer. We observe that this approach gives both representational and training advantages for our models.

The contributions of the chapter are as follows. In Section 6.4 we design two types of PixelRNNs corresponding to the two types of LSTM layers; we describe the purely convolutional PixelCNN that is our fastest architecture; and we design a *Multi-Scale*

version of the PixelRNN. In Section 6.6 we show the relative benefits of using the discrete softmax distribution in our models and of adopting residual connections for the LSTM layers. Next we test the models on MNIST and on CIFAR-10 and show that they obtain log-likelihood scores that are considerably better than previous results. We also provide results for the large-scale ImageNet dataset resized to both 32×32 and 64×64 pixels; to our knowledge likelihood values from generative models have not previously been reported on this dataset. Finally, we give a qualitative evaluation of the samples generated from the PixelRNNs.

6.3 Model

Our aim is to estimate a distribution over natural images that can be used to tractably compute the likelihood of images and to generate new ones. The network scans the image one row at a time and one pixel at a time within each row. For each pixel it predicts the conditional distribution over the possible pixel values given the scanned context. Figure 6.2 illustrates this process. The joint distribution over the image pixels is factorized into a product of conditional distributions. The parameters used in the predictions are shared across all pixel positions in the image.

To capture the generation process, [Theis and Bethge, 2015] propose to use a two-dimensional LSTM network [Graves and Schmidhuber, 2009] that starts at the top left pixel and proceeds towards the bottom right pixel. The advantage of the LSTM network is that it effectively handles long-range dependencies that are central to object and scene understanding. The two-dimensional structure ensures that the signals are well propagated both in the left-to-right and top-to-bottom directions.

In this section we first focus on the form of the distribution, whereas the next section will be devoted to describing the architectural innovations inside PixelRNN.

6.3.1 Generating an Image Pixel by Pixel

The goal is to assign a probability $p(\mathbf{x})$ to each image $\mathbf{x}$ formed of $n \times n$ pixels. We can write the image $\mathbf{x}$ as a one-dimensional sequence $x_1, ..., x_{n^2}$ where pixels are taken from the image row by row. To estimate the joint distribution $p(\mathbf{x})$ we write it as the product of the conditional distributions over the pixels:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1}) \tag{6.1}$$

The value $p(x_i|x_1, ..., x_{i-1})$ is the probability of the i-th pixel x_i given all the previous pixels $x_1, ..., x_{i-1}$. The generation proceeds row by row and pixel by pixel. Figure 6.2 (Left) illustrates the conditioning scheme.

Each pixel x_i is in turn jointly determined by three values, one for each of the color channels Red, Green and Blue (RGB). We rewrite the distribution $p(x_i|\mathbf{x}_{<i})$ as the following product:

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G}) \tag{6.2}$$

Each of the colors is thus conditioned on the other channels as well as on all the previously generated pixels.

Note that during training and evaluation the distributions over the pixel values are computed *in parallel*, while the generation of an image is *sequential*.

6.3.2 Pixels as Discrete Variables

Previous approaches use a continuous distribution for the values of the pixels in the image (e.g. [Theis and Bethge, 2015, Uria et al., 2014]). By contrast we model $p(\mathbf{x})$ as a discrete distribution, with every conditional distribution in Equation 6.2 being a multinomial that is modeled with a softmax layer. Each channel variable $x_{i,*}$ simply

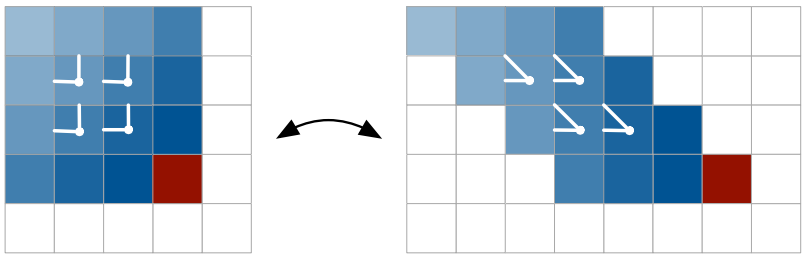

Figure 6.3: In the Diagonal BiLSTM, to allow for parallelization along the diagonals, the input map is skewed by offseting each row by one position with respect to the previous row. When the spatial layer is computed left to right and column by column, the output map is shifted back into the original size. The convolution uses a kernel of size 2×1.

takes one of 256 distinct values. The discrete distribution is representationally simple and has the advantage of being arbitrarily multimodal without prior on the shape (see Fig. 6.6). Experimentally we also find the discrete distribution to be easy to learn and to produce better performance compared to a continuous distribution (Section 6.6).

6.4 Pixel Recurrent Neural Networks

In this section we describe the architectural components that compose the PixelRNN. In Sections 6.4.1 and 6.4.2, we describe the two types of LSTM layers that use convolutions to compute at once the states along one of the spatial dimensions. In Section 6.4.3 we describe how to incorporate *residual* connections to improve the training of a PixelRNN with many LSTM layers. In Section 6.4.4 we describe the softmax layer that computes the discrete joint distribution of the colors and the masking technique that ensures the proper conditioning scheme. In Section 6.4.5 we describe the PixelCNN architecture. Finally in Section 6.4.6 we describe the multi-scale architecture.

6.4.1 Row LSTM

The Row LSTM is a unidirectional layer that processes the image row by row from top to bottom computing features for a whole row at once; the computation is performed with a one-dimensional convolution. For a pixel x_i the layer captures a roughly triangular context above the pixel as shown in Figure 6.4 (center). The kernel of the one-dimensional convolution has size $k \times 1$ where $k \geq 3$; the larger the value of k the broader the context that is captured. The weight sharing in the convolution ensures translation invariance of the computed features along each row.

The computation proceeds as follows. An LSTM layer has an input-to-state component and a recurrent state-to-state component that together determine the four gates inside the LSTM core. To enhance parallelization in the Row LSTM the input-to-state component is first computed for the entire two-dimensional input map; for this a $k \times 1$ convolution is used to follow the row-wise orientation of the LSTM itself. The convolution is *masked* to include only the valid context (see Section 6.4.4) and produces a tensor of size $4h \times n \times n$, representing the four gate vectors for each position in the input map, where h is the number of output feature maps.

To compute one step of the state-to-state component of the LSTM layer, one is given the previous hidden and cell states $\mathbf{h}_{i-1}$ and $\mathbf{c}_{i-1}$, each of size $h \times n \times 1$. The new hidden and cell states $\mathbf{h}_i$, $\mathbf{c}_i$ are obtained as follows:

$$\begin{aligned} [\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] &= \sigma(\mathbf{K}^{ss} \circledast \mathbf{h}_{i-1} + \mathbf{K}^{is} \circledast \mathbf{x}_i) \\ \mathbf{c}_i &= \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i \\ \mathbf{h}_i &= \mathbf{o}_i \odot \tanh(\mathbf{c}_i) \end{aligned} \tag{6.3}$$

where $\mathbf{x}_i$ of size $h \times n \times 1$ is row i of the input map, and $\circledast$ represents the convolution operation and $\odot$ the element-wise multiplication. The weights $\mathbf{K}^{ss}$ and $\mathbf{K}^{is}$ are the

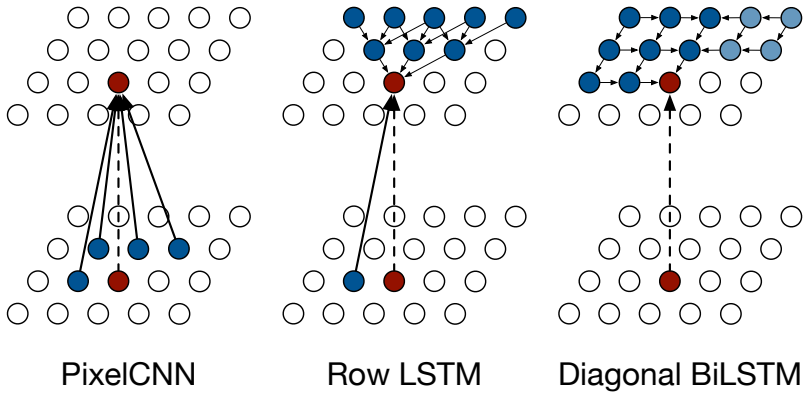

Figure 6.4: Visualization of the input-to-state and state-to-state mappings for the three proposed architectures.

kernel weights for the state-to-state and the input-to-state components, where the latter is precomputed as described above. In the case of the output, forget and input gates $\mathbf{o}_i$, $\mathbf{f}_i$ and $\mathbf{i}_i$, the activation σ is the logistic sigmoid function, whereas for the content gate $\mathbf{g}_i$, σ is the tanh function. Each step computes at once the new state for an entire row of the input map. Because the Row LSTM has a triangular receptive field (Figure 6.4), it is unable to capture the entire available context.

6.4.2 Diagonal BiLSTM

The Diagonal BiLSTM is designed to both parallelize the computation and to capture the entire available context for any image size. Each of the two directions of the layer scans the image in a diagonal fashion starting from a corner at the top and reaching the opposite corner at the bottom. Each step in the computation computes at once the LSTM state along a diagonal in the image. Figure 6.4 (right) illustrates the computation and the resulting receptive field.

The diagonal computation proceeds as follows. We first skew the input map into a space that makes it easy to apply convolutions along diagonals. The skewing operation offsets each row of the input map by one position with respect to the previous row, as illustrated in Figure 6.3; this results in a map of size $n \times (2n - 1)$.

At this point we can compute the input-to-state and state-to-state components of the Diagonal BiLSTM. For each of the two directions, the input-to-state component is simply a 1×1 convolution K^{is} that contributes to the four gates in the LSTM core; the operation generates a $4h \times n \times n$ tensor. The state-to-state recurrent component is then computed with a *column-wise* convolution K^{ss} that has a kernel of size 2×1. The step takes the previous hidden and cell states, combines the contribution of the input-to-state component and produces the next hidden and cell states, as defined in Equation 6.3. The output feature map is then skewed back into an $n \times n$ map by removing the offset positions. This computation is repeated for each of the two directions. Given the two output maps, to prevent the layer from seeing future pixels, the *right* output map is then shifted down by one row and added to the *left* output map.

Besides reaching the full dependency field, the Diagonal BiLSTM has the additional advantage that it uses a convolutional kernel of size 2×1 that processes a minimal amount of information at each step yielding a highly non-linear computation. Kernel sizes larger than 2×1 are not particularly useful as they do not broaden the already global receptive field of the Diagonal BiLSTM.

6.4.3 Residual Connections

We train PixelRNNs of up to twelve layers of depth. As a means to both increase convergence speed and propagate signals more directly through the network, we deploy *residual connections* from one LSTM layer to the next. Figure 6.5 shows a diagram of the residual blocks. The input map to the PixelRNN LSTM layer has $2h$ features. The input-to-state component reduces the number of features by producing h features per gate. After applying the recurrent layer, the output map is upsampled back to $2h$ features per position via a 1×1 convolution and the input map is added to the output map. This method is related to previous approaches that use gating along the

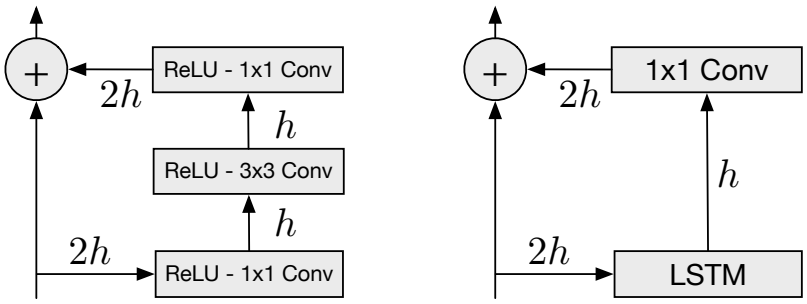

Figure 6.5: Residual blocks for a PixelCNN (left) and PixelRNNs.

depth of the recurrent network [Kalchbrenner et al., 2016, Zhang et al., 2016], but has the advantage of not requiring additional gates. Apart from residual connections, one can also use learnable skip connections from each layer to the output. In the experiments we evaluate the relative effectiveness of residual and layer-to-output skip connections.

6.4.4 Two-dimensional Masked Convolution

The h features for each input position at every layer in the network are split into three parts, each corresponding to one of the RGB channels. When predicting the R channel for the current pixel x_i, only the generated pixels left and above of x_i can be used as context. When predicting the G channel, the value of the R channel can also be used as context in addition to the previously generated pixels. Likewise, for the B channel, the values of both the R and G channels can be used. To restrict connections in the network to these dependencies, we apply a *mask* to the input-to-state convolutions and to other purely convolutional layers in a PixelRNN.

We use two types of masks that we indicate with *mask A* and *mask B*, as shown in Figure 6.2 (Right). Mask A is applied only to the first convolutional layer in a PixelRNN and restricts the connections to those neighboring pixels and to those colors in the current pixels that have already been predicted. On the other hand, mask B is applied to all the subsequent input-to-state convolutional transitions and relaxes

<table>
<tr><th>PixelCNN</th><th>Row LSTM</th><th>Diagonal BiLSTM</th></tr>
<tr><td colspan="3">7×7 conv mask A</td></tr>
<tr><td colspan="3">Multiple residual blocks: (see fig 6.5)</td></tr>
<tr><td>Conv
3×3 mask B</td><td>Row LSTM
i-s: 3×1 mask B
s-s: 3×1 no mask</td><td>Diagonal BiLSTM
i-s: 1×1 mask B
s-s: 1×2 no mask</td></tr>
<tr><td colspan="3">ReLU followed by 1×1 conv, mask B (2 layers)</td></tr>
<tr><td colspan="3">256-way Softmax for each RGB color (Natural images)
or Sigmoid (MNIST)</td></tr>
</table>

Table 6.1: Details of the architectures. In the LSTM architectures i-s and s-s stand for input-state and state-state convolutions.

the restrictions of mask A by also allowing the connection from a color to itself. The masks can be easily implemented by zeroing out the corresponding weights in the input-to-state convolutions after each update. Similar masks have also been used in variational autoencoders [Gregor et al., 2014].

6.4.5 PixelCNN

The Row and Diagonal LSTM layers have a potentially unbounded dependency range within their receptive field. This comes with a computational cost as each state needs to be computed sequentially. One simple workaround is to make the receptive field large, but not unbounded. We can use standard convolutional layers to capture a bounded receptive field and compute features for all pixel positions at once. The PixelCNN uses multiple convolutional layers that preserve the spatial resolution; pooling layers are not used. Masks are adopted in the convolutions to avoid seeing the future context; masks have previously also been used in non-convolutional models such as MADE [Germain et al., 2015]. Note that the advantage of parallelization of the PixelCNN over the PixelRNN is only available during training or during evaluating of test images. The image generation process is sequential for both kinds of networks,

as each sampled pixel needs to be given as input back into the network. The PixelCNN is from an architectural point of view a two-dimensional version of the ByteNet architecture in Ch. 5.

6.4.6 Multi-Scale PixelRNN

The Multi-Scale PixelRNN is composed of an *unconditional* PixelRNN and one or more *conditional* PixelRNNs. The unconditional network first generates in the standard way a smaller $s \times s$ image that is *subsampled* from the original image. The conditional network then takes the $s \times s$ image as an additional input and generates a larger $n \times n$ image, as shown in Figure 6.2 (Middle).

The conditional network is similar to a standard PixelRNN, but each of its layers is biased with an upsampled version of the small $s \times s$ image. The upsampling and biasing processes are defined as follows. In the upsampling process, one uses a convolutional network with deconvolutional layers to construct an enlarged feature map of size $c \times n \times n$, where c is the number of features in the output map of the upsampling network. Then, in the biasing process, for each layer in the conditional PixelRNN, one simply maps the $c \times n \times n$ conditioning map into a $4h \times n \times n$ map that is added to the input-to-state map of the corresponding layer; this is performed using a 1×1 unmasked convolution. The larger $n \times n$ image is then generated as usual.

6.5 Specifications of Models

In this section we give the specifications of the PixelRNNs used in the experiments. We have four types of networks: the PixelRNN based on Row LSTM, the one based on Diagonal BiLSTM, the fully convolutional one and the Multi-Scale one.

Table 6.1 specifies each layer in the single-scale networks. The first layer is a 7×7

convolution that uses the mask of type A. The two types of LSTM networks then use a variable number of recurrent layers. The input-to-state convolution in this layer uses a mask of type B, whereas the state-to-state convolution is not masked. The PixelCNN uses convolutions of size 3×3 with a mask of type B. The top feature map is then passed through a couple of layers consisting of a Rectified Linear Unit (ReLU) and a 1×1 convolution. For the CIFAR-10 and ImageNet experiments, these layers have 1024 feature maps; for the MNIST experiment, the layers have 32 feature maps. Residual and layer-to-output connections are used across the layers of all three networks.

The networks used in the experiments have the following hyperparameters. For MNIST we use a Diagonal BiLSTM with 7 layers and a value of $h = 16$ (Section 6.4.3 and Figure 6.5 right). For CIFAR-10 the Row and Diagonal BiLSTMs have 12 layers and a number of $h = 128$ units. The PixelCNN has 15 layers and $h = 128$. For 32×32 ImageNet we adopt a 12 layer Row LSTM with $h = 384$ units and for 64×64 ImageNet we use a 4 layer Row LSTM with $h = 512$ units; the latter model does not use residual connections.

6.6 Experiments

In this section we describe our experiments and results. We begin by describing the way we evaluate and compare our results. In Section 6.6.2 we give details about the training. Then we give results on the relative effectiveness of architectural components and our best results on the MNIST, CIFAR-10 and ImageNet datasets.

6.6.1 Evaluation

All our models are trained and evaluated on the log-likelihood loss function coming from a discrete distribution. Although natural image data is usually modeled with

continuous distributions using density functions, we can compare our results with previous art in the following way. In the literature it is currently best practice to add real-valued noise to the pixel values to dequantize the data when using density functions [Uria et al., 2013]. When uniform noise is added (with values in the interval [0, 1]), then the log-likelihoods of continuous and discrete models are directly comparable [Theis et al., 2015]. In our case, we can use the values from the discrete distribution as a piecewise-uniform continuous function that has a constant value for every interval $[i, i+1], i = 1, 2, \ldots 256$. This corresponding distribution will have the same log-likelihood (on data with added noise) as the original discrete distribution (on discrete data).

For MNIST we report the negative log-likelihood in *nats* as it is common practice in literature. For CIFAR-10 and ImageNet we report negative log-likelihoods in *bits* per dimension. The total discrete log-likelihood is normalized by the dimensionality of the images (e.g., $32 \times 32 \times 3 = 3072$ for CIFAR-10). These numbers are interpretable as the number of bits that a compression scheme based on this model would need to compress every RGB color value [van den Oord and Schrauwen, 2014b, Theis et al., 2015]; in practice there is also a small overhead due to arithmetic coding.

6.6.2 Training Details

Our models are trained on GPUs using the Torch toolbox. From the different parameter update rules tried, RMSProp gives best convergence performance and is used for all experiments. The learning rate schedules were manually set for every dataset to the highest values that allowed fast convergence. The batch sizes also vary for different datasets. For smaller datasets such as MNIST and CIFAR-10 we use smaller batch sizes of 16 images as this seems to regularize the models. For ImageNet we use as large a batch size as allowed by the GPU memory; this corresponds to 64 images/batch for 32×32 ImageNet, and 32 images/batch for 64×64 ImageNet. Apart

from scaling and centering the images at the input of the network, we don't use any other preprocessing or augmentation. For the multinomial loss function we use the raw pixel color values as categories. For all the PixelRNN models, we learn the initial recurrent state of the network.

6.6.3 Discrete Softmax Distribution

Apart from being intuitive and easy to implement, we find that using a softmax on discrete pixel values instead of a mixture density approach on continuous pixel values gives better results. For the Row LSTM model with a softmax output distribution we obtain 3.06 bits/dim on the CIFAR-10 validation set. For the same model with a Mixture of Conditional Gaussian Scale Mixtures (MCGSM) [Theis and Bethge, 2015] we obtain 3.22 bits/dim.

In Figure 6.6 we show a few softmax activations from the model. Although we don't embed prior information about the meaning or relations of the 256 color categories, e.g. that pixel values 51 and 52 are neighbors, the distributions predicted by the model are meaningful and can be multimodal, skewed, peaked or long tailed. Also note that values 0 and 255 often get a much higher probability as they are more frequent. Another advantage of the discrete distribution is that we do not worry about parts of the distribution mass lying outside the interval [0, 255], which is something that typically happens with continuous distributions.

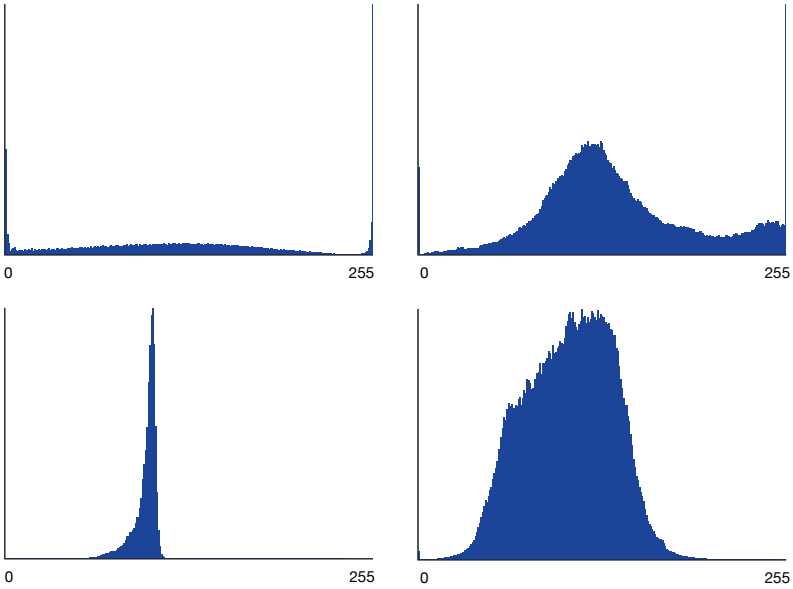

Figure 6.6: Example softmax activations from the model. The top left shows the distribution of the first pixel red value (first value to sample). The other graphs show distributions for a few other chosen positions in the image.

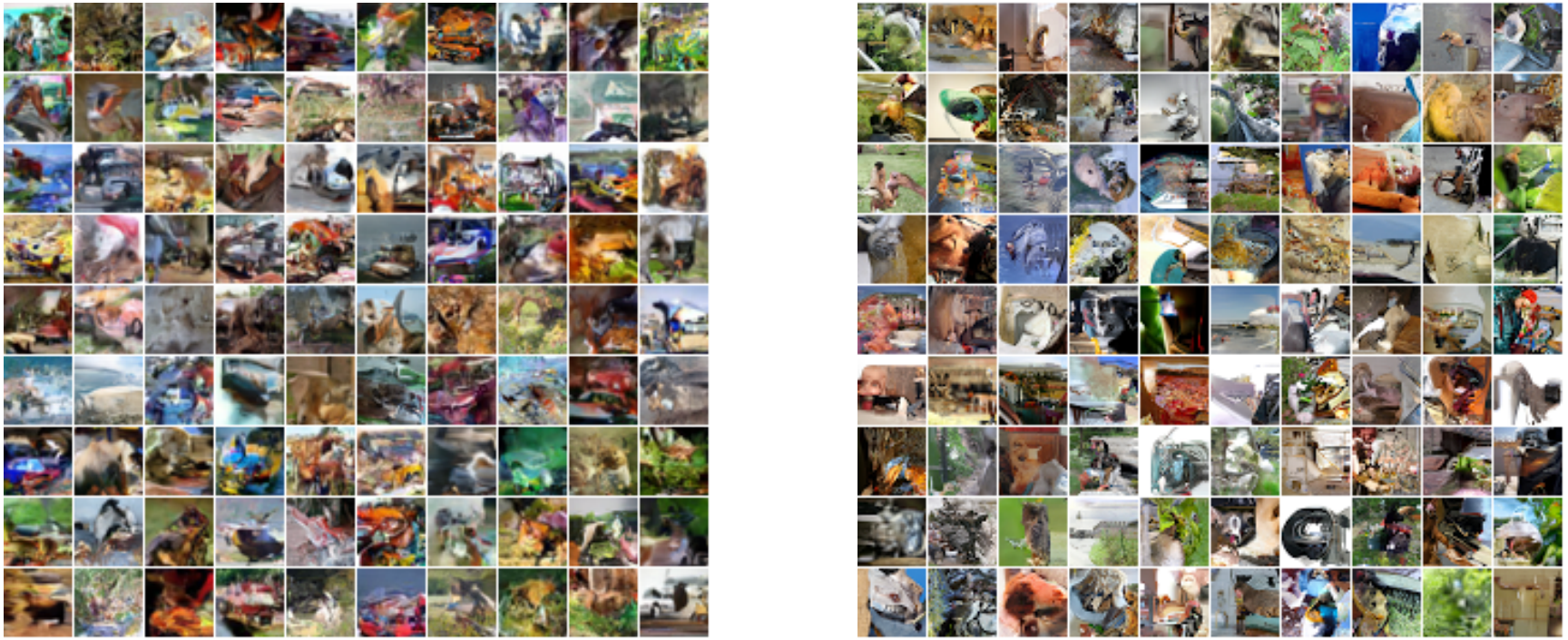

Figure 6.7: Samples from models trained on CIFAR-10 (left) and ImageNet 32x32 (right) images. In general we can see that the models capture local spatial dependencies relatively well. The ImageNet model seems to be better at capturing more global structures than the CIFAR-10 model. The ImageNet model was larger and trained on much more data, which explains the qualitative difference in samples.

6.6.4 Residual Connections

Another core component of the networks is residual connections. In Table 6.2 we show the results of having residual connections, having standard skip connections or having both, in the 12-layer CIFAR-10 Row LSTM model. We see that using residual connections is as effective as using skip connections; using both is also effective and preserves the advantage.

	No skip	**Skip**
No residual:	3.22	3.09
Residual:	3.07	3.06

Table 6.2: Effect of residual and skip connections in the Row LSTM network evaluated on the Cifar-10 validation set in bits/dim.

When using both the residual and skip connections, we see in Table 6.3 that performance of the Row LSTM improves with increased depth. This holds for up to the 12 LSTM layers that we tried.

# layers:	1	2	3	6	9	12
NLL:	3.30	3.20	3.17	3.09	3.08	3.06

Table 6.3: Effect of the number of layers on the negative log likelihood evaluated on the CIFAR-10 validation set (bits/dim).

6.6.5 MNIST

Although the goal of our work was to model natural images on a large scale, we also tried our model on the binary version [Salakhutdinov and Murray, 2008] of MNIST [LeCun et al., 1998b] as it is a good sanity check and there is a lot of previous art on this dataset to compare with. In Table 7.1 we report the performance of the Diagonal BiLSTM model and that of previous published results. To our knowledge this is the best reported result on MNIST so far.

Model	**NLL Test**
DBM 2hl [1]:	≈ 84.62
DBN 2hl [2]:	≈ 84.55
NADE [3]:	88.33
EoNADE 2hl (128 orderings) [3]:	85.10
EoNADE-5 2hl (128 orderings) [4]:	84.68
DLGM [5]:	≈ 86.60
DLGM 8 leapfrog steps [6]:	≈ 85.51
DARN 1hl [7]:	≈ 84.13
MADE 2hl (32 masks) [8]:	86.64
DRAW [9]:	≤ 80.97
PixelCNN:	81.30
Row LSTM:	80.54
Diagonal BiLSTM (1 layer, $h = 32$):	**80.75**
Diagonal BiLSTM (7 layers, $h = 16$):	**79.20**

Table 6.4: Test set performance of different models on MNIST in *nats* (negative log-likelihood). Prior results taken from [1] [Salakhutdinov and Hinton, 2009], [2] [Murray and Salakhutdinov, 2009], [3] [Uria et al., 2014], [4] [Raiko et al., 2014], [5] [Rezende et al., 2014], [6] [Salimans et al., 2015], [7] [Gregor et al., 2014], [8] [Germain et al., 2015], [9] [Gregor et al., 2015].

Model	**NLL Test (Train)**
Uniform Distribution:	8.00
Multivariate Gaussian:	4.70
NICE [1]:	4.48
Deep Diffusion [2]:	4.20
Deep GMMs [3]:	4.00
RIDE [4]:	3.47
PixelCNN:	3.14 (3.08)
Row LSTM:	3.07 (3.00)
Diagonal BiLSTM:	**3.00** (2.93)

Table 6.5: Test set performance of different models on CIFAR-10 in *bits/dim*. For our models we give training performance in brackets. [1] [Dinh et al., 2014], [2] [Sohl-Dickstein et al., 2015], [3] [van den Oord and Schrauwen, 2014a], [4] personal communication [Theis and Bethge, 2015].

Image size	NLL Validation (Train)
32x32:	3.86 (3.83)
64x64:	3.63 (3.57)

Table 6.6: Negative log-likelihood performance on 32×32 and 64×64 ImageNet in *bits/dim*.

6.6.6 CIFAR-10

Next we test our models on the CIFAR-10 dataset [Krizhevsky, 2009]. Table 6.5 lists the results of our models and that of previously published approaches. All our results were obtained without data augmentation. For the proposed networks, the Diagonal BiLSTM has the best performance, followed by the Row LSTM and the PixelCNN. This coincides with the size of the respective receptive fields: the Diagonal BiLSTM has a global view, the Row LSTM has a partially occluded view and the PixelCNN sees the fewest pixels in the context. This suggests that effectively capturing a large receptive field is important. Figure 6.7 (left) shows CIFAR-10 samples generated from the Diagonal BiLSTM.

6.6.7 ImageNet

Although to our knowledge the are no published results on the ILSVRC ImageNet dataset [Russakovsky et al., 2015] that we can compare our models with, we give our ImageNet log-likelihood performance in Table 6.6 (without data augmentation). On ImageNet the current PixelRNNs do not appear to overfit, as we saw that their validation performance improved with size and depth. The main constraint on model size are currently computation time and GPU memory.

Note that the ImageNet models are in general less compressible than the CIFAR-10 images. ImageNet has greater variety of images, and the CIFAR-10 images were most likely resized with a different algorithm than the one we used for ImageNet images. The ImageNet images are less blurry, which means neighboring pixels are less

correlated to each other and thus less predictable. Because the downsampling method can influence the compression performance, we have made the used downsampled images available[1].

Figure 6.7 (right) shows 32×32 samples drawn from our model trained on ImageNet. Figure 6.8 shows 64×64 samples from the same model with and without multi-scale conditioning. Finally, we also show image completions sampled from the model in Figure 6.9.

6.7 Discussion

In this chapter we significantly improve and build upon deep recurrent neural networks as generative models for natural images. We have described novel two-dimensional LSTM layers: the Row LSTM and the Diagonal BiLSTM, that scale more easily to larger datasets. The models were trained to model the raw RGB pixel values. We treated the pixel values as discrete random variables by using a softmax layer in the conditional distributions. We employed masked convolutions to allow PixelRNNs to model full dependencies between the color channels. We proposed and evaluated architectural improvements in these models resulting in PixelRNNs with up to 12 LSTM layers.

We have shown that the PixelRNNs significantly improve the state of the art on the MNIST and CIFAR-10 datasets. We also provide new benchmarks for generative image modelling on the ImageNet dataset. Based on the samples and completions drawn from the models we can conclude that the PixelRNNs are able to model both spatially local and long-range correlations and are able to produce images that are sharp and coherent. Given that these models improve as we make them larger and that there is practically unlimited data available to train on, more computation and larger models are likely to further improve the results.

[1] http://image-net.org/small/download.php

Figure 6.8: Samples from models trained on ImageNet 64x64 images. Left: normal model, right: multi-scale model. The single-scale model trained on 64x64 images is less able to capture global structure than the 32x32 model. The multi-scale model seems to resolve this problem. Although these models get similar performance in log-likelihood, the samples on the right do seem globally more coherent.

Figure 6.9: Image completions sampled from a model that was trained on 32x32 ImageNet images. Note that diversity of the completions is high, which can be attributed to the log-likelihood loss function used in this generative model, as it encourages models with high entropy. As these are sampled from the model, we can easily generate millions of different completions. It is also interesting to see that textures such as water, wood and shrubbery are also inputed relative well (see Figure 6.1).

Chapter 7

Video Pixel Networks

7.1 Summary

The final chapter of Part III and of the thesis extends the framework introduced in the previous chapter to the generation of videos. The videos studied here are some of the largest and most complex tensors that have been generated with the autoregressive method to date. We propose a probabilistic video model, the Video Pixel Network (VPN), that estimates the discrete joint distribution of the raw pixel values in a video. The model and the neural architecture reflect the time, space and color structure of video tensors and encode it as a four-dimensional dependency chain. The VPN approaches the best possible performance on the Moving MNIST benchmark, a leap over the previous state of the art, and the generated videos show only minor deviations from the ground truth. The VPN also produces detailed samples on the action-conditional Robotic Pushing benchmark and generalizes to the motion of novel objects.

7.2 Background

Video modelling has remained a challenging problem due to the complexity and ambiguity inherent in video data. Current approaches range from mean squared error models based on deep neural networks [Srivastava et al., 2015a, Oh et al., 2015], to models that predict quantized image patches [Ranzato et al., 2014], incorporate motion priors [Patraucean et al., 2015, Finn et al., 2016] or use adversarial losses [Mathieu et al., 2015, Vondrick et al., 2016]. Despite the wealth of approaches, future frame predictions that are free of systematic artifacts (e.g. blurring) have been out of reach even on relatively simple benchmarks like Moving MNIST [Srivastava et al., 2015a].

We propose the Video Pixel Network (VPN), a generative video model based on deep neural networks, that reflects the factorization of the joint distribution of the pixel values in a video. The model encodes the four-dimensional structure of video tensors and captures dependencies in the time dimension of the data, in the two space dimensions of each frame and in the color channels of a pixel. This makes it possible to model the stochastic transitions locally from one pixel to the next and more globally from one frame to the next without introducing independence assumptions in the conditional factors. The factorization further ensures that the model stays fully tractable; the likelihood that the model assigns to a video can be computed exactly. The model operates on pixels without preprocessing and predicts discrete multinomial distributions over raw pixel intensities, allowing the model to estimate distributions of any shape.

The architecture of the VPN consists of two parts: resolution preserving CNN encoders and PixelCNN decoders (Ch. 6). The CNN encoders preserve at all layers the spatial resolution of the input frames in order to maximize representational capacity. The outputs of the encoders are combined over time with a convolutional LSTM that also preserves the resolution [Hochreiter and Schmidhuber, 1997, Shi et al., 2015]. The PixelCNN decoders use masked convolutions to efficiently capture space and

color dependencies and use a softmax layer to model the multinomial distributions over raw pixel values. The network uses dilated convolutions in the encoders to achieve larger receptive fields and better capture global motion. The network also utilizes newly defined multiplicative units and corresponding residual blocks.

We evaluate VPNs on two benchmarks. The first is the Moving MNIST dataset [Srivastava et al., 2015a] where, given 10 frames of two moving digits, the task is to predict the following 10 frames. In Sect. 7.6 we show that the VPN achieves 87.6 nats/frame, a score that is near the lower bound on the loss (calculated to be 86.3 nats/frame); this constitutes a significant improvement over the previous best result of 179.8 nats/frame [Patraucean et al., 2015].

The second benchmark is the Robotic Pushing dataset [Finn et al., 2016] where, given two natural video frames showing a robotic arm pushing objects, the task is to predict the following 18 frames. We show that the VPN not only generalizes to new action sequences with objects seen during training, but also to new action sequences involving *novel* objects not seen during training. Random samples from the VPN preserve remarkable detail throughout the generated sequence. We also define a baseline model that lacks the space and color dependencies. This lets us see that the latter dependencies are crucial for avoiding systematic artifacts in generated videos.

7.3 Model

In this section we define the probabilistic model implemented by Video Pixel Networks. Let a video $\mathbf{x}$ be a four-dimensional tensor of pixel values $\mathbf{x}_{t,i,j,c}$, where the first (temporal) dimension $t \in \{0, ..., T\}$ corresponds to one of the frames in the video, the next two (spatial) dimensions $i, j \in \{0, ..., N\}$ index the pixel at row i and column j in frame t, and the last dimension $c \in \{R, G, B\}$ denotes one of the three RGB

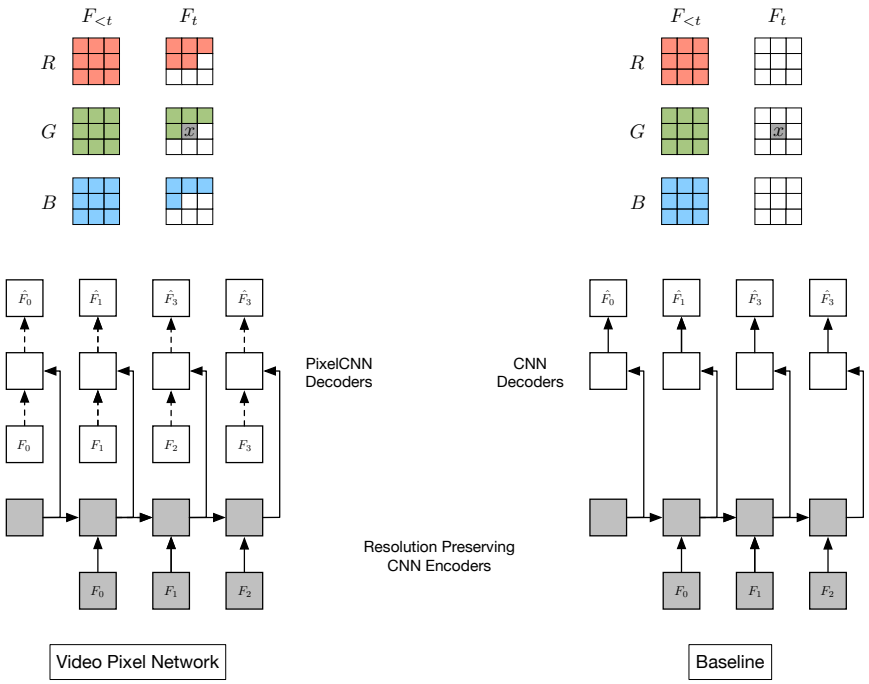

Figure 7.1: Dependency map (top) and neural network structure (bottom) for the VPN (left) and the baseline model (right).

channels of the pixel. We let each $\mathbf{x}_{t,i,j,c}$ be a random variable that takes values from the RGB color intensities of the pixel.

By applying the chain rule to factorize the video likelihood $p(\mathbf{x})$ as a product of conditional probabilities, we can model it in a tractable manner and without introducing independence assumptions:

$$p(\mathbf{x}) = \prod_{t=0}^{T} \prod_{i=0}^{N} \prod_{j=0}^{N} p(\mathbf{x}_{t,i,j,B}|\mathbf{x}_{<}, \mathbf{x}_{t,i,j,R}, \mathbf{x}_{t,i,j,G})\ p(\mathbf{x}_{t,i,j,G}|\mathbf{x}_{<}, \mathbf{x}_{t,i,j,R})\ p(\mathbf{x}_{t,i,j,R}|\mathbf{x}_{<}). \quad (7.1)$$

Here $\mathbf{x}_{<} = \mathbf{x}_{(t,<i,<j,:)} \cup \mathbf{x}_{(<t,:,:,:)}$ comprises the RGB values of all pixels to the left and above the pixel at position (i, j) in the current frame t, as well as the RGB values of pixels from all the previous frames. This represents a four-dimensional factorization of the joint probability.

Note that the factorization itself does not impose a unique ordering on the set

of variables. We choose an ordering according to two criteria. The first criterion is determined by the properties and uses of the data; frames in the video are predicted according to their temporal order. The second criterion favors orderings that can be computed efficiently; pixels are predicted starting from one corner of the frame (the top left corner) and ending in the opposite corner of the frame (the bottom right one) as this allows for the computation to be implemented efficiently (Ch. 6). The order for the prediction of the colors is chosen by convention as R, G and B.

The VPN models directly the four dimensions of video tensors. We use F_t to denote the t-th frame $\mathbf{x}_{t,:,:,:}$ in the video $\mathbf{x}$.

Figure 7.1 illustrates the fourfold dependency structure for the green color channel value of the pixel x in frame F_t, which depends on: (i) all pixels in all the previous frames $F_{<t}$; (ii) all three colors of the already generated pixels in F_t; (iii) the already generated red color value of the pixel x.

We follow the PixelRNN approach from Ch. 6 in modelling each conditional factor as a discrete multinomial distribution over 256 raw color values. This allows for the predicted distributions to be arbitrarily multimodal.

7.3.1 Baseline Model

We compare the VPN model with a baseline model that encodes the temporal dependencies in videos from previous frames to the next, but ignores the spatial dependencies between the pixels within a frame and the dependencies between the color channels. In this case the joint distribution is factorized by introducing independence assumptions:

$$p(\mathbf{x}) \approx \prod_{t=0}^{T} \prod_{i=0}^{N} \prod_{j=0}^{N} p(\mathbf{x}_{t,i,j,B} | \mathbf{x}_{<t,:,:,:}) \; p(\mathbf{x}_{t,i,j,G} | \mathbf{x}_{<t,:,:,:}) \; p(\mathbf{x}_{t,i,j,R} | \mathbf{x}_{<t,:,:,:}). \tag{7.2}$$

Figure 7.1 illustrates the conditioning structure in the baseline model. The green channel value of pixel $\mathbf{x}$ only depends on the values of pixels in previous frames. Various models have been proposed that are similar to our baseline model in that they capture the temporal dependencies only [Ranzato et al., 2014, Srivastava et al., 2015a, Oh et al., 2015]

7.3.2 Remarks on the Factorization

To illustrate the properties of the two factorizations, suppose that a model needs to predict the value of a pixel x and the value of the adjacent pixel y in a frame F, where the transition to the frame F from the previous frames $F_<$ is non-deterministic. For a simple example, suppose the previous frames $F_<$ depict a robotic arm and in the current frame F the robotic arm is about to move either left or right. The baseline model estimates $p(x|F_<)$ and $p(y|F_<)$ as distributions with two modes, one for the robot moving left and one for the robot moving right. Sampling independently from $p(x|F_<)$ and $p(y|F_<)$ can lead to two inconsistent pixel values coming from distinct modes, one pixel value depicting the robot moving left and the other depicting the robot moving right. The accumulation of these inconsistencies for a few frames leads to known artifacts such as blurring of video continuations. By contrast, in this example, the VPN estimates $p(x|F_<)$ as the same bimodal distribution, but then estimates $p(y|x, F_<)$ *conditioned on the selected value of* x. The conditioned distribution is unimodal and, if the value of x is sampled to depict the robot moving left, then the value of y is sampled accordingly to also depict the robot moving left.

Generating a video tensor requires sampling $T*N^2*3$ variables, which for a second of video with resolution 64×64 is in the order of 10^5 samples. This figure is in the order of 10^4 for generating a single image or for a second of audio signal [van den Oord et al., 2016], and it is in the order of 10^2 for language tasks such as machine translation (Ch. 4).

7.4 Architecture

In this section we construct a network architecture capable of computing efficiently the factorized distribution in Sect. 7.3. The architecture consists of two parts. The first part models the temporal dimension of the data and consists of Resolution Preserving CNN Encoders whose outputs are given to a Convolutional LSTM. The second part models the spatial and color dimensions of the video and consists of PixelCNN architectures (Ch. 6) that are conditioned on the outputs of the CNN Encoders.

7.4.1 Resolution Preserving CNN Encoders

Given a set of video frames $F_0, ..., F_T$, the VPN first encodes each of the first T frames $F_0, ..., F_{T-1}$ with a CNN Encoder. These frames form the histories that condition the generated frames. Each of the CNN Encoders is composed of k ($k = 8$ in the experiments) residual blocks (Sect. 7.5) and the spatial resolution of the input frames is preserved throughout the layers in all the blocks. Preserving the resolution is crucial as it allows the model to condition each pixel that needs to be generated without loss of representational capacity. The outputs of the T CNN Encoders, which are computed in parallel during training, are given as input to a Convolutional LSTM, which also preserves the resolution. This part of the VPN computes the temporal dependencies of the video tensor and is represented in Fig. 7.1 by the shaded blocks.

7.4.2 PixelCNN Decoders

The second part of the VPN architecture computes dependencies along the space and color dimensions. The T outputs of the first part of the architecture provide representations for the contexts that condition the generation of a portion of the $T + 1$ frames $F_0, ..., F_T$; if one generates all the $T + 1$ frames, then the first frame F_0 receives no context representation. These context representations are used to

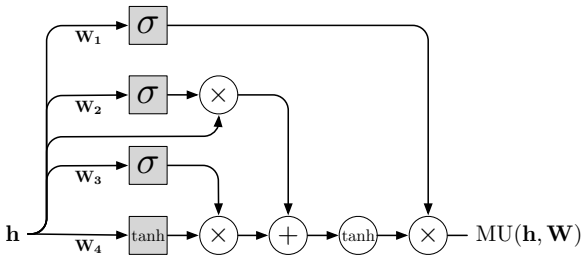

Figure 7.2: Structure of a multiplicative unit (MU). The squares represent the three gates and the update. The circles represent component-wise operations.

condition decoder neural networks that are PixelCNNs. PixelCNNs are composed of l resolution preserving residual blocks ($l = 12$ in the experiments), each in turn formed of *masked* convolutional layers. Since we treat the pixel values as discrete random variables, the final layer of the PixelCNN decoders is a softmax layer over 256 intensity values for each color channel in each pixel.

Figure 7.1 depicts the two parts of the architecture of the VPN. The decoder that generates pixel x of frame F_t sees the context representation for all the frames up to F_{t-1} coming from the preceding CNN encoders. The decoder also sees the pixel values above and left of the pixel x in the current frame F_t that is itself given as input to the decoder.

7.4.3 Architecture of Baseline Model

We implement the baseline model by using the same CNN encoders to build the context representations. In contrast with PixelCNNs, the decoders in the baseline model are CNNs that do not use masking on the weights; the frame to be predicted thus cannot be given as input. As shown in Fig. 7.1, the resulting neural network captures the temporal dependencies, but ignores spatial and color channel dependencies within the generated frames. Just like for VPNs, we make the neural architecture of the baseline model resolution preserving in all the layers.

7.5 Network Building Blocks

In this section we describe two basic operations that are used as the building blocks of the VPN. The first is the *Multiplicative Unit* (MU, Sect. 7.5.1) that contains multiplicative interactions inspired by LSTM [Hochreiter and Schmidhuber, 1997] gates. The second building block is the *Residual Multiplicative Block* (RMB, Sect. 7.5.2) that is composed of multiple layers of MUs.

7.5.1 Multiplicative Units

A multiplicative unit (Fig. 7.2) is constructed by incorporating LSTM-like gates into a convolutional layer. Given an input $\mathbf{h}$ of size $N \times N \times c$, where c corresponds to the number of channels, we first pass it through four convolutional layers to create an update $\mathbf{u}$ and three gates $\mathbf{g_{1-3}}$. The input, update, and gates are then combined in the following manner:

$$\begin{aligned} \mathbf{g_1} &= \sigma(\mathbf{W_1} * \mathbf{h}) \\ \mathbf{g_2} &= \sigma(\mathbf{W_2} * \mathbf{h}) \\ \mathbf{g_3} &= \sigma(\mathbf{W_3} * \mathbf{h}) \\ \mathbf{u} &= \tanh(\mathbf{W_4} * \mathbf{h}) \\ \mathrm{MU}(\mathbf{h}; \mathbf{W}) &= \mathbf{g_1} \odot \tanh(\mathbf{g_2} \odot \mathbf{h} + \mathbf{g_3} \odot \mathbf{u}) \end{aligned} \tag{7.3}$$

where σ is the sigmoid non-linearity and $\odot$ is component-wise multiplication. Biases are omitted for clarity. In our experiments the convolutional weights $\mathbf{W_{1-4}}$ use a kernel of size 3×3. Unlike LSTM networks, there is no distinction between *memory* and *hidden* states. Also, unlike Highway networks [Srivastava et al., 2015b] and Grid LSTM [Kalchbrenner et al., 2016], there is no setting of the gates such

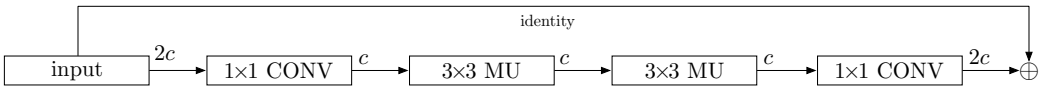

Figure 7.3: Structure of the residual multiplicative block (RMB) incorporating two multiplicative units (MUs).

that $\mathrm{MU}(\mathbf{h}; \mathbf{W})$ simply returns the input $\mathbf{h}$; the input is always processed with a non-linearity.

7.5.2 Residual Multiplicative Blocks

To allow for easy gradient propagation through many layers of the network, we stack two MU layers in a residual multiplicative block (Fig. 7.3) where the input has a residual (additive skip) connection to the output [He et al., 2016]. For computational efficiency, the number of channels is halved in MU layers inside the block. Namely, given an input layer $\mathbf{h}$ of size $N \times N \times 2c$ with $2c$ channels, we first apply a 1×1 convolutional layer that reduces the number of channels to c; no activation function is used for this layer, and it is followed by two successive MU layers each with a convolutional kernel of size 3×3. We then project the feature map back to $2c$ channels using another 1×1 convolutional layer. Finally, the input $\mathbf{h}$ is added to the overall output forming a residual connection. Such a layer structure is similar to the bottleneck residual unit of [He et al., 2016]. Formally, the Residual Multiplicative Block (RMB) is computed as follows:

$$
\begin{aligned}
\mathbf{h_1} &= \mathbf{W_1} * \mathbf{h} \\
\mathbf{h_2} &= \mathrm{MU}(\mathbf{h_1}; \mathbf{W_2}) \\
\mathbf{h_3} &= \mathrm{MU}(\mathbf{h_2}; \mathbf{W_3}) \\
\mathbf{h_4} &= \mathbf{W_4} * \mathbf{h_3} \\
\mathrm{RMB}(\mathbf{h}; \mathbf{W}) &= \mathbf{h} + \mathbf{h_4}
\end{aligned}
\tag{7.4}
$$

Model	Test
[Shi et al., 2015]	367.2
[Srivastava et al., 2015a]	341.2
[Brabandere et al., 2016]	285.2
[Patraucean et al., 2015]	179.8
Baseline model	110.1
VPN	**87.6**
Lower Bound	86.3

Table 7.1: Cross-entropy results in nats/frame on the Moving MNIST dataset.

We also experimented with a standard residual block of [He et al., 2016] which uses ReLU non-linearities – see Sect. 7.6 and 7.7 for details.

7.5.3 Dilated Convolutions

Having a large receptive field helps the model to capture the motion of larger objects. One way to increase the receptive field without much effect on the computational complexity is to use dilated convolutions [Chen et al., 2014, Yu and Koltun, 2015], which make the receptive field grow exponentially, as opposed to linearly, in the number of layers. In the variant of VPN that uses dilation, the dilation rates are the same within each RMB, but they double from one RMB to the next up to a chosen maximum size, and then repeat; this is analogous to the ByteNet construction in the Ch. 5. In particular, in the CNN encoders we use two repetitions of the dilation scheme $[1, 2, 4, 8]$, for a total of 8 RMBs. We do not use dilation in the decoders.

7.6 Moving MNIST

The Moving MNIST dataset consists of sequences of 20 frames of size 64×64, depicting two potentially overlapping MNIST digits moving with constant velocity and bouncing off walls. Training sequences are generated on-the-fly using digits from the

MNIST training set without a limit on the number of generated training sequences (our models observe 19.2M training sequences before convergence). The test set is fixed and consists of 10000 sequences that contain digits from the MNIST test set. 10 of the 20 frames are used as context and the remaining 10 frames are generated.

In order to make our results comparable, for this dataset only we use the same sigmoid cross-entropy loss as used in prior work [Srivastava et al., 2015a]. The loss is defined as:

$$H(z, y) = -\sum_i z_i \log y_i + (1 - z_i) \log(1 - y_i) \tag{7.5}$$

where z_i are the grayscale targets in the Moving MNIST frames that are interpreted as probabilities and y_i are the predictions. The lower bound on $H(z, y)$ may be non-zero. In fact, if we let $z_i = y_i$, for the 10 frames that are predicted in each sequence of the Moving MNIST test data, $H(z, y) = 86.3$ nats/frame.

7.6.1 Implementation Details

The VPNs with and without dilation, as well as the baseline model, have 8 RMBs in the encoders and 12 RMBs in the decoders; for the network variants that use ReLUs we double the number of residual blocks to 16 and 24, respectively, in order to equate the size of the receptive fields in the two model variants. The number of channels in the blocks is $c = 128$ while the convolutional LSTM has 256 channels. The topmost layer before the output has 768 channels. We train the models for 300000 steps with 20-frame sequences predicting the last 10 frames of each sequence. Each step corresponds to a batch of 64 sequences. We use RMSProp for the optimization with an initial learning rate of $3 \cdot 10^{-4}$ and multiply the learning rate by 0.3 when learning flatlines.

Model	Test
VPN (RMB, No Dilation)	89.2
VPN (Relu, No Dilation)	89.1
VPN (Relu, Dilation)	87.7
VPN (RMB, Dilation)	87.6
Lower Bound	86.3

Table 7.2: Cross-entropy results in nats/frame on the Moving MNIST dataset.

7.6.2 Results

Table 7.1 reports the results of various recent video models on the Moving MNIST test set. Our baseline model achieves 110.1 nats/frame, which is significantly better than the previous state of the art [Patraucean et al., 2015]. We attribute these gains to architectural features and, in particular, to the resolution preserving aspect of the network. Further, the VPN achieves 87.6 nats/frame, which approaches the lower bound of 86.3 nats/frame.

Table 7.2 reports results of architectural variants of the VPNs. The model with dilated convolutions improves over its non-dilated counterpart as it can more easily act on the relatively large digits moving in the 64×64 frames. In the case of Moving MNIST, MUs do not yield a significant improvement in performance over just using ReLUs, possibly due to the relatively low complexity of the task. A sizeable improvement is obtained from MUs on the Robotic Pushing dataset (Tab. 7.3).

A qualitative evaluation of video continuations produced by the models matches the quantitative results. Figure 7.4 shows random continuations produced by the VPN and the baseline model on the Moving MNIST test set. The frames generated by the VPN are consistently sharp even when they deviate from the ground truth.

By contrast, the continuations produced by the baseline model get progressively more blurred with time – as the uncertainty of the model grows with the number of generated frames, the lack of inter-frame spatial dependencies leads the model to take

the expectation over possible trajectories.

7.7 Robotic Pushing

The Robotic Pushing dataset consists of sequences of 20 frames of size 64×64 that represent camera recordings of a robotic arm pushing objects in a basket. The data consists of a training set of 50000 sequences, a validation set, and two test sets of 1500 sequences each, one involving a subset of the objects seen during training and the other one involving *novel* objects not seen during training. Each frame in the video sequence is paired with the state of the robot at that frame and with the desired action to reach the next frame. The transitions are non-deterministic as the robotic arm may not reach the desired state in a frame due to occlusion by the objects encountered on its trajectory. 2 frames, 2 states and 2 actions are used as context; the desired 18 actions in the future are also given. The 18 frames in the future are then generated conditioned on the 18 actions as well as on the 2 steps of context.

7.7.1 Implementation Details

For this dataset, both the VPN and the baseline model use the softmax cross-entropy loss, as defined in Sect. 7.3. As for Moving MNIST, the models have 8 RMBs in the encoders and 12 RMBs in the decoders; the ReLU variants have 16 residual blocks in the encoders and 24 in the decoders. The number of channels in the RMBs is $c = 128$, the convolutional LSTM has 256 channels and the topmost layer before the output has 1536 channels. We use RMSProp with an initial learning rate of 10^{-4}. We train for 275000 steps with a batch size of 64 sequences per step. Each training sequence is obtained by selecting a random subsequence of 12 frames together with the corresponding states and actions. We use the first 2 frames in the subsequence as context and predict the other 10 frames. States and actions come as vectors of 5 real values. For the 2 context frames, we condition each layer in the encoders and the

Model	Validation	Test (Seen)	Test (Novel)
Baseline model	2.06	2.08	2.07
VPN (Relu, Dilation)	0.73	0.72	0.75
VPN (Relu, No Dilation)	0.72	0.73	0.75
VPN (RMB, Dilation)	0.63	0.65	0.64
VPN (RMB, No Dilation)	**0.62**	**0.64**	**0.64**

Table 7.3: Negative log-likelihood in nats/dimension on the Robotic Pushing dataset.

decoders with the respective state and action vectors; the conditioning is performed by the result of a 1×1 convolution applied to the action and state vectors that are broadcast to all of the 64×64 positions. For the other 10 frames, we condition the encoders and decoders with the action vectors only. We discard the state vectors for the predicted frames during training and do not use them at generation. For generation we unroll the models for the entire sequence of 20 frames and generate 18 frames.

7.7.2 Results

Table 7.3 reports the results of the baseline model and variants of the VPN on the Robotic Pushing validation and test sets. The best variant of the VPN has a $> 65\%$ reduction in negative log-likelihood over the baseline model. This highlights the importance of space and color dependencies in non-deterministic environments. The results on the validation and test datasets with seen objects and on the test dataset with novel objects are similar. This shows that the models have learned to generalize well not just to new action sequences, but also to new objects. Furthermore, we see that using multiplicative interactions in the VPN gives a significant improvement over using ReLUs.

Figures 7.5–7.9 visualize the samples generated by our models. Figure 7.5 contains random samples of the VPN on the validation set with seen objects (together with the corresponding ground truth). The model is able to distinguish between the robotic

arm and the background, correctly handling occlusions and only pushing the objects when they come in contact with the robotic arm. The VPN generates the arm when it enters into the frame from one of the sides. The position of the arm in the samples is close to that in the ground truth, suggesting the VPN has learned to follow the actions. The generated videos remain detailed throughout the 18 frames and few artifacts are present. The samples remain good showing the ability of the VPN to generalize to new sequences of actions. Figure 7.6 evaluates an additional level of generalization, by showing samples from the test set with *novel* objects not seen during training. The VPN seems to identify the novel objects correctly and generates plausible movements for them. The samples do not appear visibly worse than in the datasets with seen objects. Figure 7.7 demonstrates the probabilistic nature of the VPN, by showing multiple different video continuations that start from the same context frames and are conditioned on the same sequence of 18 future actions. The continuations are plausible and varied, further suggesting the VPN's ability to generalize. Figure 7.8 shows samples from the baseline model. In contrast with the VPN samples, we see a form of high frequency noise appearing in the non-deterministic movements of the robotic arm. This can be attributed to the lack of space and color dependencies, as discussed in Sec. 7.3.2. Figure 7.9 shows a comparison of continuations of the baseline model and the VPN from the same context sequence. Besides artifacts, the baseline model also seems less responsive to the actions.

7.8 Discussion

We have introduced the Video Pixel Network, a deep generative model of video data that models the factorization of the joint likelihood of video. We have shown that, despite its lack of specific motion priors or surrogate losses, the VPN approaches the lower bound on the loss on the Moving MNIST benchmark that corresponds to a large improvement over the previous state of the art. On the Robotic Pushing

dataset, the VPN achieves significantly better likelihoods than the baseline model that lacks the fourfold dependency structure; the VPN generates videos that are free of artifacts and are highly detailed for many frames into the future. The fourfold dependency structure provides a robust and generic method for generating videos without systematic artifacts.

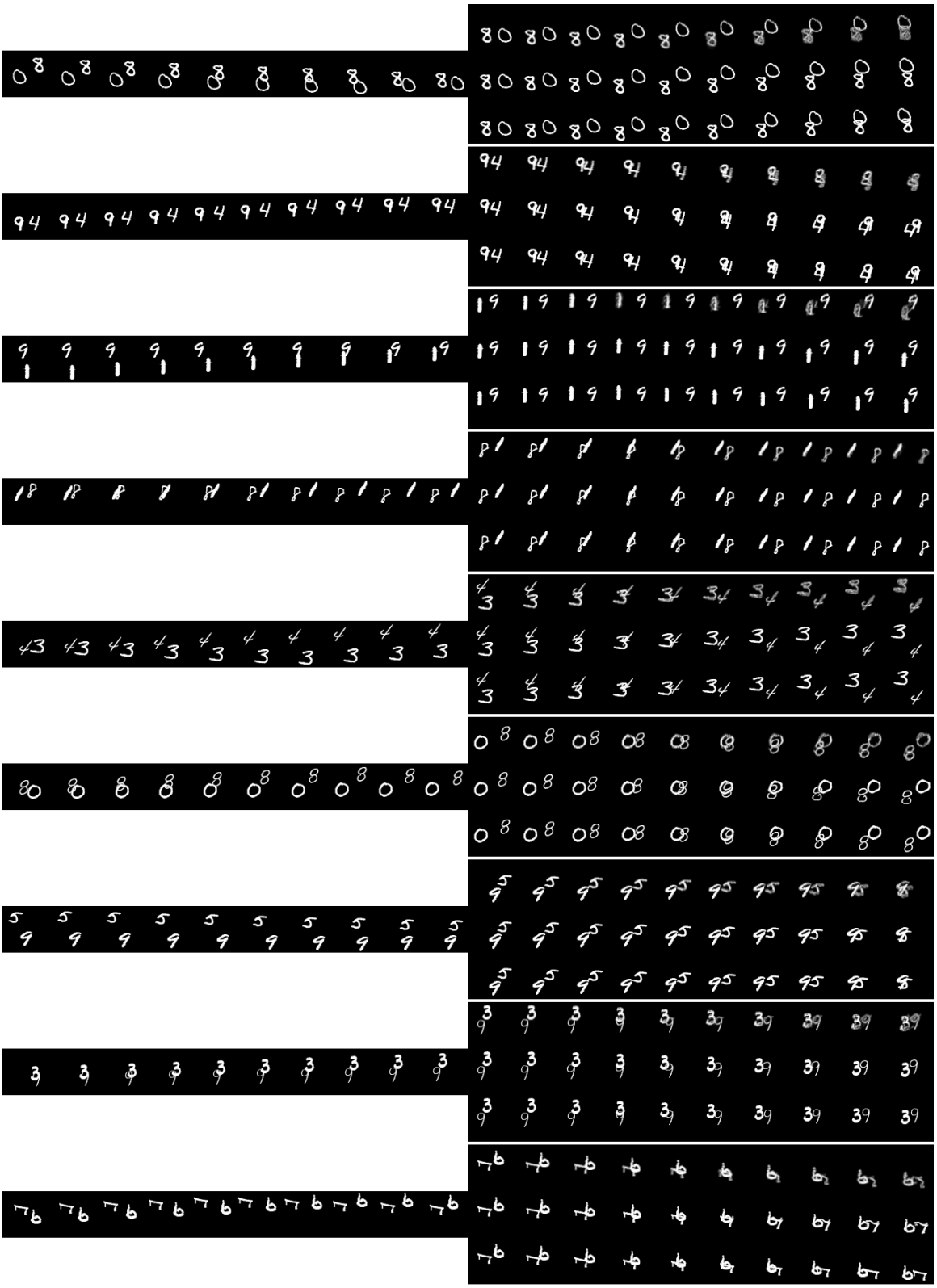

Figure 7.4: Randomly sampled continuations of videos from the Moving MNIST test set. For each set of three rows, the first 10 frames in the middle row are the given context frames. The next three rows of 10 frames each are as follows: frames generated from the baseline model (top row), frames generated from the VPN (middle row) and ground truth frames (bottom row).

Figure 7.5: Randomly sampled continuations of videos from the Robotic Pushing validation set (with seen objects). Each set of four rows corresponds to a sample of 2 given context frames and 18 generated frames. In each set of four rows, rows 1 and 3 are samples from the VPN. Rows 2 and 4 are the actual continuation in the data. Animated GIFs available at `nal.ai/vpn`

Figure 7.6: Randomly sampled continuations of videos from the Robotic Pushing test set with *novel* objects not seen during training. Each set of four rows is as in Fig. 7.5.

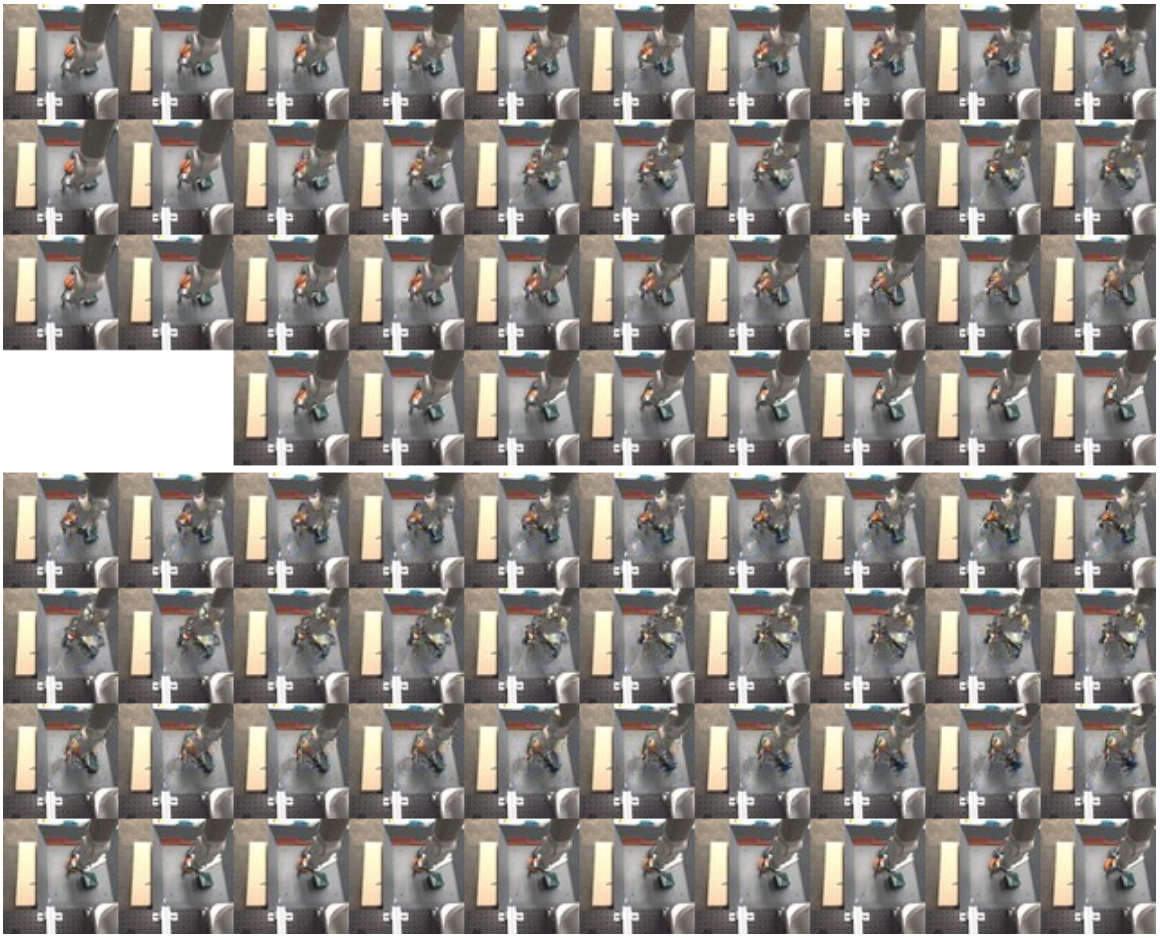

Figure 7.7: Three different samples from the VPN starting from the same 2 context frames on the Robotic Pushing validation set. For each set of four rows, top three rows are generated samples, the bottom row is the actual continuation in the data.

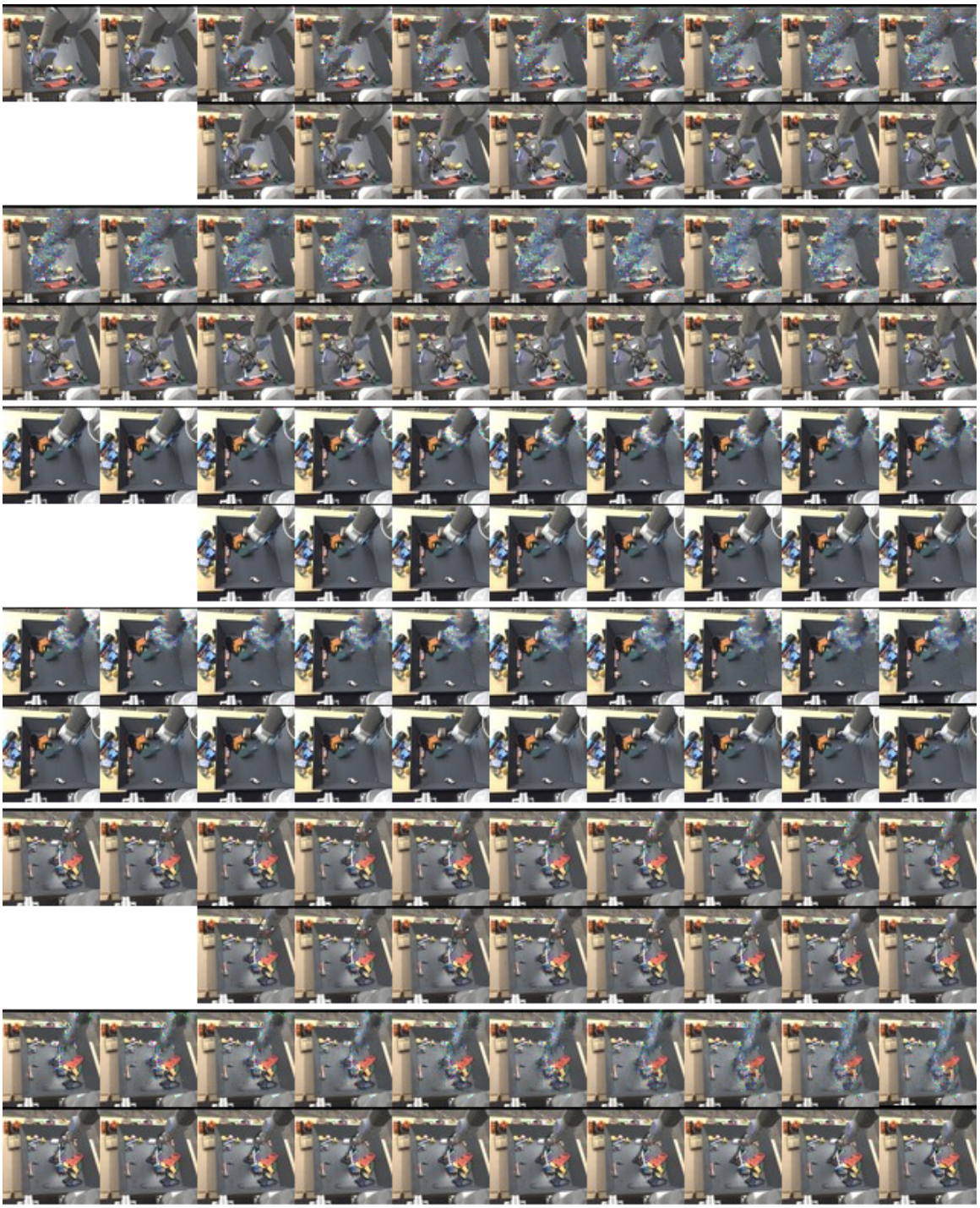

Figure 7.8: Randomly sampled continuations from the baseline model on the Robotic Pushing validation set (with seen objects). Each set of four rows is as in Fig. 7.5.

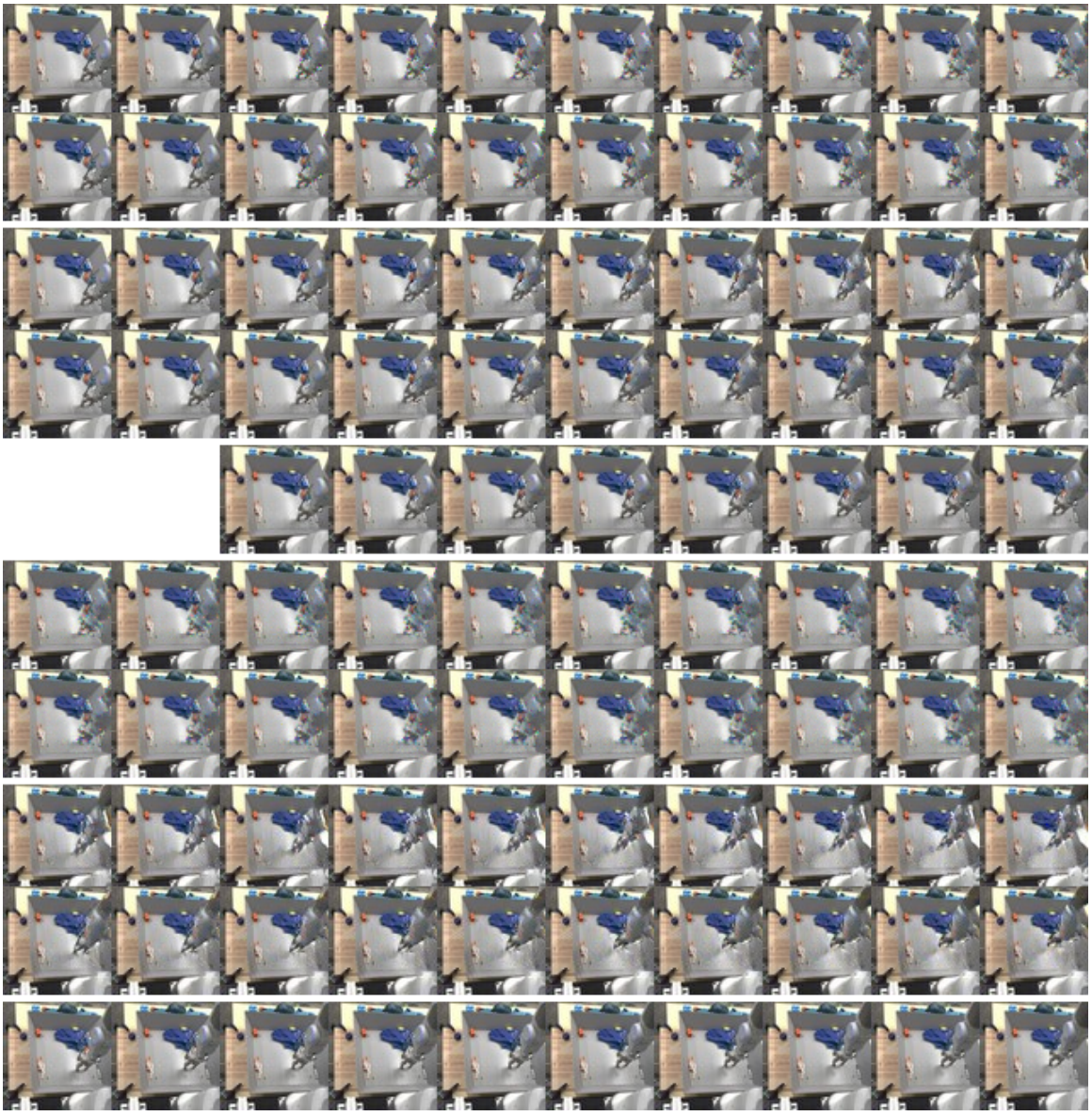

Figure 7.9: Comparison of continuations given the same 2 context frames from the Robotic Pushing validation set for the baseline model (rows 1 and 2, and 6 and 7), for the VPN (rows 3 and 4, and 8 and 9) and for the actual continuation in the data (rows 5 and 10).

Chapter 8

Afterword

This thesis introduces a powerful framework for the classification and generation of structured tensors. It contains the first deep convolutional models of sentences (Ch. 2) as well as the first deep neural networks for discourse classification (Ch. 3). The thesis also presents the first instance of encoder-decoder neural networks as a novel foundation for machine translation, which were initially publisheds in 2013 (Ch. 4), as well as the first fully convolutional encoder-decoders published in 2016 with state-of-the-art results on character-level translation (Ch. 5). The thesis further shows how autoregressive decoding is not limited to natural language or one-dimensional sequences. State-of-the-art results on image likelihood estimation (Ch. 6) and the first solution to artifact-free video generation (Ch. 7) give examples for how the framework generalizes to the visual domain and to multi-dimensional tensors. These results show the striking effectiveness of these methods. Yet, when a method is shown to be effective, just as many new questions are created as those that are answered. We conclude this thesis by summarizing a few of these open directions of inquiry:

- We have shown how it is possible to learn effective distributions over natural language and images. The learnt distributions however can sometimes have undesirable properties. For instance, searching for a most likely utterance from

an *unconditional* language model will often yield a short, repetitive and somewhat meaningless sentence putting into question the notion of *the probability of an utterance* itself. Note that somewhat mysteriously this phenomenon does not appear in the *conditional* case of a neural translation model where a likely output of the well-trained model is a high-quality translation of the source; the success of neural machine translation depends almost entirely on this mysterious property of conditional distributions over target languages. Similarly, optimizing for the likelihood of a generated image will often yield uniformly brightly colored images; and this can also happen for conditional distributions over video frames. These bizarre behaviors of learnt distributions can reduce their usefulness, but above all they call for an increased understanding and systematic characterization of the probabilistic objective.

- Another avenue of inquiry concerns the limits on the decoding speed for the networks. It is clear that careful implementations using (persistent) kernels for GPUs can greatly raise these limits in general. But increases in serial computation - which represents the ultimate bottleneck for sequential sampling - have stalled and over the recent years the largest computational gains have come not from speeding up individual cores, but from combining thousands of cores together on a single chip, as is done inside GPUs. Methods on how to render sequential decoding from encoder-decoder networks more parallel will need to be developed to overcome this fundamental bottleneck of sequential decoding.

- Improvements in the quality of the models themselves is expected to arise from the use of better neural architectures. A great advantage of encoder-decoder networks is that they are very sensitive to the properties of the underlying architecture. They thus lend themselves well to the discovery of new architectural principles with the goal of better capturing, for example, long-range dependen-

cies across all of the dimensions of the tensors.

- Especially in the visual (and audio) domains, deploying such powerful models directly in the input space of pixels can result in modelling explicitly highly redundant and compressible information. Large patches of an ordinary natural image are all of a similar color and frames in a video are highly correlated. As a result it seems reasonable to attempt to apply encoder-decoder networks directly in a space of compressed representations. Faithfully learning such compressed representations remains an open problem in unsupervised learning.

- To end with, from a more conceptual perspective, encoder-decoder networks and their successes can be enlightening as to the nature of the phenomena they attempt to model. For example, much can be said about the nature of language or the nature of the visual world around us from a cognitive and philosophical perspective by looking at the behavior of these models in these domains. Surprising philosophical conclusions may arise from a serious undertaking of this theoretical line of inquiry.

To these matters shall the scientist's wandering mind turn next.

Bibliography

[Austin, 1962] Austin, J. L. (1962). How to do things with words. *Oxford: Clarendon.*

[Ba et al., 2016] Ba, L. J., Kiros, R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, abs/1607.06450.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Baroni and Zamparelli, 2010] Baroni, M. and Zamparelli, R. (2010). Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *EMNLP*, pages 1183–1193. ACL.

[Bengio et al., 2003a] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003a). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[Bengio et al., 2003b] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003b). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[Blacoe and Lapata, 2012] Blacoe, W. and Lapata, M. (2012). A comparison of vector-based representations for semantic composition. In *EMNLP-CoNLL*, pages 546–556. ACL.

[Blunsom et al., 2006] Blunsom, P., Kocik, K., and Curran, J. R. (2006). Question classification with log-linear models. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 615–616, New York, NY, USA. ACM.

[Brabandere et al., 2016] Brabandere, B. D., Jia, X., Tuytelaars, T., and Gool, L. V. (2016). Dynamic filter networks. *CoRR*, abs/1605.09673.

[Brown et al., 1993] Brown, P. F., Pietra, V. J., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311.

[Calhoun et al., 2010] Calhoun, S., Carletta, J., Brenier, J., Mayo, N., Jurafsky, D., Steedman, M., and Beaver, D. (2010). The nxt-format switchboard corpus: a rich resource for investigating the syntax, semantics, pragmatics and prosody of dialogue. *Language Resources and Evaluation*, 44(4):387–419.

[Chen et al., 2014] Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[Chung et al., 2016a] Chung, J., Ahn, S., and Bengio, Y. (2016a). Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704.

[Chung et al., 2016b] Chung, J., Cho, K., and Bengio, Y. (2016b). A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*.

[Chung et al., 2015] Chung, J., Gülçehre, C., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. *CoRR, abs/1502.02367*.

[Clarke, 2012] Clarke, D. (2012). A context-theoretic framework for compositionality in distributional semantics. *Computational Linguistics*, 38(1):41–71.

[Coecke et al., 2010] Coecke, B., Sadrzadeh, M., and Clark, S. (2010). Mathematical Foundations for a Compositional Distributional Model of Meaning.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*.

[Dinh et al., 2014] Dinh, L., Krueger, D., and Bengio, Y. (2014). NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159.

[Dyer et al., 2013] Dyer, C., Chahuneau, V., and Smith, N. A. (2013). A simple, fast, and effective reparameterization of ibm model 2. In *Proc. of NAACL*.

[Dyer et al., 2010] Dyer, C., Weese, J., Setiawan, H., Lopez, A., Ture, F., Eidelman, V., Ganitkevitch, J., Blunsom, P., and Resnik, P. (2010). cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12. Association for Computational Linguistics.

[Efros and Leung, 1999] Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece.

[Erk, 2012] Erk, K. (2012). Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653.

[Erk and Padó, 2008] Erk, K. and Padó, S. (2008). A structured vector space model for word meaning in context. *Proceedings of the Conference on Empirical Methods in Natural Language Processing - EMNLP '08*, (October):897.

[Finn et al., 2016] Finn, C., Goodfellow, I. J., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction. *CoRR*, abs/1605.07157.

[Frege, 1892] Frege, G. (1892). Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, 100.

[Freitag et al., 2014] Freitag, M., Peitz, S., Wuebker, J., Ney, H., Huck, M., Sennrich, R., Durrani, N., Nadejde, M., Williams, P., Koehn, P., Herrmann, T., Cho, E., and Waibel, A. (2014). Eu-bridge mt: Combined machine translation. In *ACL 2014 Ninth Workshop on Statistical Machine Translation*.

[Germain et al., 2015] Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). MADE: Masked autoencoder for distribution estimation. *arXiv preprint arXiv:1502.03509*.

[Gers and Schmidhuber, 2001] Gers, F. A. and Schmidhuber, J. (2001). Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

[Go et al., 2009] Go, A., Bhayani, R., and Huang, L. (2009). Twitter sentiment classification using distant supervision. *Processing*, pages 1–6.

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

[Graves and Schmidhuber, 2009] Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*.

[Grefenstette, 2013] Grefenstette, E. (2013). Category-theoretic quantitative compositional distributional models of natural language semantics. *arXiv preprint arXiv:1311.1539*.

[Grefenstette and Sadrzadeh, 2011] Grefenstette, E. and Sadrzadeh, M. (2011). Experimental support for a categorical compositional distributional model of meaning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1394–1404. Association for Computational Linguistics.

[Grefenstette et al., 2011] Grefenstette, E., Sadrzadeh, M., Clark, S., Coecke, B., and Pulman, S. (2011). Concrete sentence spaces for compositional distributional models of meaning. *CoRR*, abs/1101.0309.

[Gregor et al., 2015] Gregor, K., Danihelka, I., Graves, A., and Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. *Proceedings of the 32nd International Conference on Machine Learning*.

[Gregor et al., 2014] Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep autoregressive networks. In *Proceedings of the 31st International Conference on Machine Learning*.

[Guevara, 2010] Guevara, E. (2010). Modelling Adjective-Noun Compositionality by Regression. *ESSLLI'10 Workshop on Compositionality and Distributional Semantic Models*.

[Ha et al., 2016] Ha, D., Dai, A., and Le, Q. V. (2016). HyperNetworks. *ArXiv e-prints*.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. *CoRR*, abs/1603.05027.

[Hermann and Blunsom, 2013] Hermann, K. M. and Blunsom, P. (2013). The Role of Syntax in Vector Space Models of Compositional Semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Sofia, Bulgaria. Association for Computational Linguistics. Forthcoming.

[Hinton, 1989] Hinton, G. E. (1989). Connectionist learning procedures. *Artif. Intell.*, 40(1-3):185–234.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[Hochreiter et al., 2001] Hochreiter, S., Bengio, Y., and Frasconi, P. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kolen, J. and Kremer, S., editors, *Field Guide to Dynamical Recurrent Networks*. IEEE Press.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

[Huang et al., 2008] Huang, Z., Thint, M., and Qin, Z. (2008). Question classification using head words and their hypernyms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 927–936, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Hutter, 2012] Hutter, M. (2012). The human knowledge compression contest. *http://prize.hutter1.net/*.

[Jones et al., 2006] Jones, R., Rey, B., Madani, O., and Greiner, W. (2006). Generating query substitutions. In Carr, L., Roure, D. D., Iyengar, A., Goble, C. A., and Dahlin, M., editors, *WWW*, pages 387–396. ACM.

[Kaiser and Bengio, 2016] Kaiser, Ł. and Bengio, S. (2016). Can active memory replace attention? *Advances in Neural Information Processing Systems*.

[Kalchbrenner et al., 2016] Kalchbrenner, N., Danihelka, I., and Graves, A. (2016). Grid long short-term memory. *International Conference on Learning Representations*.

[Kartsaklis and Sadrzadeh, 2013] Kartsaklis, D. and Sadrzadeh, M. (2013). Prior disambiguation of word tensors for constructing sentence vectors. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Seattle, USA.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Korta and Perry, 2012] Korta, K. and Perry, J. (2012). Pragmatics. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Winter 2012 edition.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[Küchler and Goller, 1996] Küchler, A. and Goller, C. (1996). Inductive learning in symbolic domains using structure-driven recurrent neural networks. In Görz, G. and Hölldobler, S., editors, *KI*, volume 1137 of *Lecture Notes in Computer Science*, pages 183–197. Springer.

[Larochelle and Murray, 2011] Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. *The Journal of Machine Learning Research*.

[Le et al., 2012] Le, H. S., Allauzen, A., and Yvon, F. (2012). Continuous space translation models with neural networks. In *HLT-NAACL*, pages 39–48.

[LeCun et al., 1998a] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun et al., 1998b] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998b). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

[LeCun et al., 2001] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (2001). Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press.

[Li and Roth, 2002] Li, X. and Roth, D. (2002). Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

[Luong and Manning, 2016] Luong, M. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. In *ACL*.

[Luong et al., 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *EMNLP*.

[Mathieu et al., 2015] Mathieu, M., Couprie, C., and LeCun, Y. (2015). Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In Kobayashi,

T., Hirose, K., and Nakamura, S., editors, *INTERSPEECH*, pages 1045–1048. ISCA.

[Mikolov et al., 2011] Mikolov, T., Kombrink, S., Burget, L., Cernocký, J., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *ICASSP*, pages 5528–5531. IEEE.

[Mikolov and Zweig, 2012] Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. In *SLT*, pages 234–239.

[Mitchell and Lapata, 2008] Mitchell, J. and Lapata, M. (2008). Vector-based models of semantic composition. In *Proceedings of ACL*, volume 8.

[Mitchell and Lapata, 2010] Mitchell, J. and Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429.

[Murray and Salakhutdinov, 2009] Murray, I. and Salakhutdinov, R. R. (2009). Evaluating probabilities under high-dimensional latent variable models. In *Advances in Neural Information Processing Systems*.

[Neal, 1992] Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial intelligence*.

[Oh et al., 2015] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. P. (2015). Action-conditional video prediction using deep networks in atari games. In *NIPS*, pages 2863–2871.

[Patraucean et al., 2015] Patraucean, V., Handa, A., and Cipolla, R. (2015). Spatio-temporal video autoencoder with differentiable memory. *CoRR*, abs/1511.06309.

[Pollack, 1990] Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46:77–105.

[Potts, 2011] Potts, C. (2011). Pragmatics. In Mitkov, R., editor, *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 2 edition.

[Raiko et al., 2014] Raiko, T., Li, Y., Cho, K., and Bengio, Y. (2014). Iterative neural autoregressive distribution estimator NADE-k. In *Advances in Neural Information Processing Systems*.

[Ranzato et al., 2014] Ranzato, M., Szlam, A., Bruna, J., Mathieu, M., Collobert, R., and Chopra, S. (2014). Video (language) modeling: a baseline for generative models of natural videos. *CoRR*, abs/1412.6604.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*.

[Rocki, 2016] Rocki, K. (2016). Recurrent memory array structures. *CoRR*, abs/1607.03085.

[Rumelhart et al., 1986a] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representations by error propagation. In Rumelhart, D. E. and Mcclelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA.

[Rumelhart et al., 1986b] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Learning internal representations by error propagation. *MIT Press Computational Models Of Cognition And Perception Series*, page 318362.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*.

[Salakhutdinov and Hinton, 2009] Salakhutdinov, R. and Hinton, G. E. (2009). Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*.

[Salakhutdinov and Murray, 2008] Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*.

[Salimans et al., 2015] Salimans, T., Kingma, D. P., and Welling, M. (2015). Markov chain monte carlo and variational inference: Bridging the gap. *Proceedings of the 32nd International Conference on Machine Learning*.

[Scheible and Schütze, 2013] Scheible, C. and Schütze, H. (2013). Cutting recursive autoencoder trees. *CoRR*, abs/1301.2811.

[Schwenk, 2012] Schwenk, H. (2012). Continuous space translation models for phrase-based statistical machine translation. In *COLING (Posters)*, pages 1071–1080.

[Schwenk et al., 2006] Schwenk, H., Déchelotte, D., and Gauvain, J.-L. (2006). Continuous space language models for statistical machine translation. In *ACL*.

[Shi et al., 2015] Shi, X., Chen, Z., Wang, H., Yeung, D., Wong, W., and Woo, W. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*, pages 802–810.

[Silva et al., 2011] Silva, J., Coheur, L., Mendes, A., and Wichert, A. (2011). From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, 35(2):137–154.

[Simonyan et al., 2013] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034.

[Socher et al., 2011a] Socher, R., Huang, E. H., Pennin, J., Ng, A. Y., and Manning, C. D. (2011a). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 24*, pages 801–809.

[Socher et al., 2012] Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (2012). Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Socher et al., 2013a] Socher, R., Le, Q. V., Manning, C. D., and Ng, A. Y. (2013a). Grounded Compositional Semantics for Finding and Describing Images with Sentences. In *Transactions of the Association for Computational Linguistics (TACL)*.

[Socher et al., 2011b] Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011b). Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Socher et al., 2013b] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013b). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Stroudsburg, PA. Association for Computational Linguistics.

[Sohl-Dickstein et al., 2015] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *Proceedings of the 32nd International Conference on Machine Learning*.

[Srivastava et al., 2015a] Srivastava, N., Mansimov, E., and Salakhutdinov, R. (2015a). Unsupervised learning of video representations using lstms. In *ICML*, volume 37, pages 843–852.

[Srivastava et al., 2015b] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015b). Highway networks. *CoRR*, abs/1505.00387.

[Stolcke et al., 2000] Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R. A., Jurafsky, D., Taylor, P., Martin, R., Ess-Dykema, C. V., and Meteer, M. (2000). Dialog act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3):339–373.

[Stollenga et al., 2015] Stollenga, M. F., Byeon, W., Liwicki, M., and Schmidhuber, J. (2015). Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation. In *Advances in Neural Information Processing Systems 28*.

[Sutskever et al., 2011a] Sutskever, I., Martens, J., and Hinton, G. E. (2011a). Generating text with recurrent neural networks. In Getoor, L. and Scheffer, T., editors, *ICML*, pages 1017–1024. Omnipress.

[Sutskever et al., 2011b] Sutskever, I., Martens, J., and Hinton, G. E. (2011b). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning*.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

[Theis and Bethge, 2015] Theis, L. and Bethge, M. (2015). Generative image modeling using spatial LSTMs. In *Advances in Neural Information Processing Systems*.

[Theis et al., 2015] Theis, L., van den Oord, A., and Bethge, M. (2015). A note on the evaluation of generative models. *CoRR*, abs/1511.01844.

[Turian et al., 2010] Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics.

[Turney, 2012] Turney, P. (2012). Domain and function: A dual-space model of semantic relations and compositions. *J. Artif. Intell. Res.(JAIR)*, 44:533–585.

[Turney and Pantel, 2010] Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *J. Artif. Intell. Res. (JAIR)*, 37:141–188.

[Uria et al., 2013] Uria, B., Murray, I., and Larochelle, H. (2013). RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*.

[Uria et al., 2014] Uria, B., Murray, I., and Larochelle, H. (2014). A deep and tractable density estimator. In *Proceedings of the 31st International Conference on Machine Learning*.

[van den Oord et al., 2016] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499.

[van den Oord and Schrauwen, 2014a] van den Oord, A. and Schrauwen, B. (2014a). Factoring variations in natural images with deep gaussian mixture models. In *Advances in Neural Information Processing Systems*.

[van den Oord and Schrauwen, 2014b] van den Oord, A. and Schrauwen, B. (2014b). The student-t mixture as a natural image patch prior with application to image compression. *The Journal of Machine Learning Research*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

[Vondrick et al., 2016] Vondrick, C., Pirsiavash, H., and Torralba, A. (2016). Generating videos with scene dynamics. *CoRR*, abs/1609.02612.

[Waibel et al., 1990] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1990). Readings in speech recognition. chapter Phoneme Recognition Using Time-delay Neural Networks, pages 393–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Wang and Manning, 2012] Wang, S. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL (2)*, pages 90–94. The Association for Computer Linguistics.

[Williams, 2012] Williams, J. D. (2012). A belief tracking challenge task for spoken dialog systems. In *NAACL-HLT Workshop on Future Directions and Needs in the Spoken Dialog Community: Tools and Data*, SDCTD '12, pages 23–24, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Williams et al., 2015] Williams, P., Sennrich, R., Nadejde, M., Huck, M., and Koehn, P. (2015). Edinburgh's syntax-based systems at WMT 2015. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*.

[Wu et al., 2016a] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., ?ukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens,

K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016a). Google?s neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

[Wu et al., 2016b] Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. (2016b). On multiplicative integration with recurrent neural networks. *CoRR*, abs/1606.06630.

[Yu and Koltun, 2015] Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122.

[Zanzotto et al., 2010] Zanzotto, F. M., Korkontzelos, I., Fallucchi, F., and Manandhar, S. (2010). Estimating linear models for compositional distributional semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1263–1271. Association for Computational Linguistics.

[Zettlemoyer and Collins, 2005] Zettlemoyer, L. S. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, pages 658–666. AUAI Press.

[Zhang et al., 2016] Zhang, Y., Chen, G., Yu, D., Yao, K., Khudanpur, S., and Glass, J. (2016). Highway long short-term memory RNNs for distant speech recognition. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*.

[Zhou et al., 2016] Zhou, J., Cao, Y., Wang, X., Li, P., and Xu, W. (2016). Deep recurrent models with fast-forward connections for neural machine translation. *CoRR*, abs/1606.04199.