CONVOLUTIONAL NEURAL NETWORK LANGUAGE MODELS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF UNIVERSITY OF TRENTO
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Ngoc-Quan Pham
August 2016

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Master of Science.

(Marco Baroni) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Master of Science.

(Gemma Boleda)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Master of Science.

(German Kruzhevsky)

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) are the family of neural network models that feature a type of layer known as the convolutional layer. This layer can extract features by convolving a learnable filter (or kernel) along different positions of a vectorial input.

CNNs have been successfully applied in Computer Vision in many different tasks, including object recognition, scene parsing, action recognition, among others [13], but they have received less attention in NLP. Mainly, they have been somewhat explored in *static classification tasks* where the model is provided with a full linguistic unit as input (e.g. a sentence) and classes are treated as independent of each other. Examples of this are sentence or document classification for tasks such as Sentiment Analysis or Topic Categorization [22, 23], sentence matching [20], and relation extraction [38]. However, their application to *sequential prediction tasks*, where the input is construed to be part of a sequence (for example, language modeling or POS tagging), has been rather limited (with exceptions, such as Collobert et al. [8]). The main contribution of this paper is a systematic evaluation of CNNs in the context of a prominent sequential prediction task, namely, language modeling.

Statistical language models are a crucial component in many Natural Language Processing applications such as Automatic Speech Recognition, Machine Translation, and Information Retrieval. Here, we study the problem under the standard formulation of learning to predict the upcoming token given its previous context. One successful approach to this problem relies on counting the number of occurrences of n-grams while using smoothing and back-off techniques to estimate the probability of an upcoming word [25]. However, since each individual word is treated independently of the others, n-gram models fail to capture semantic relations between words. In contrast, neural network language models [3] learn to predict the upcoming word given the previous context while embedding the vocabulary in a continuous space that can represent the similarity structure between words. Both feed-forward [46] and recurrent neural networks [33] have been shown to outperform n-gram models [14, 33] in various setups. These two types of neural networks make different architectural decisions. Recurrent networks take one token at a time together with a hidden "memory" vector as input and produce a prediction and an updated hidden vector for the next time step. In contrast, feed-forward language models take as input the last n tokens, where n is a fixed window size, and use them jointly to predict the upcoming word.

In the thesis, we define and explore the feed-forward neural network language models a ugmented with convolutional layers and compare them with original feed-forward and recurrent neural networks. Our results show a 13$\%$ perplexity improvement of the CNN with respect to the feed-forward language model, while comparable or higher performance compared to similarly-sized recurrent models, and lower performance with respect to larger, state-of-the-art recurrent language models (LSTMs as trained in Zaremba et al. [53]).

Our second contribution is an analysis of the kind of information learned by the CNN, showing that the network learns to extract a combination of grammatical, semantic, and topical information from tokens of all across the input window, even those that are the farthest from the target.

In brief, this thesis is organised as follows. Chapter 2 is dedicated to provide background knowledge about the motivation, architecture and progress of neural language models by covering the details of the original feed-forward and recurrent language models with most techniques applied in state-of-the-art systems. In chapter 3, we describe the proposed model with the addition of convolutional layers. Subsequently, details about the experiments will be delivered in Chapter 4. Finally, the analysis will be presented in Chapter 5.

Chapter 2

Literature Review

This chapter describes the basic knowledge of Statistical Language Modeling together with the prominent approaches. The research context is drawn in order to show the necessity of the neural network-based approaches.

2.1 Language Modeling Overview

2.1.1 About language modeling

In general, language models aim at measuring the fluency of any text, showing how much it makes sense. Artificial systems that generate text, therefore, requires the aid of language models in order to produce textual outputs that are comprehensible.

From the statistical point of view, a word string is considered as a stochastic process, thus the language model is formulated as the probability estimation over all possible sequences of words. The sequence length can be arbitrary, while the words are taken from a limited vocabulary. For example, the likeliness of ”the end of our world” is much higher than ”tea end of our word”, because the latter string is much less likely to be found in available English text.

Since direction estimation for that probability distribution is intractable, the probability of a sentence $P(w_1^L)$ is factorised using the chain rule:

$$P(w_1^L) = P(w_1|\text{<s>}) \prod_{i=2}^{L} P(w_i|\text{<s>}w_1^{i-1}) = \prod_{i=l}^{L} P(w_l|h_l) \tag{2.1}$$

in which, <s> is used to denote beginning token of the sentence/string. The history h_i represents the string before the current word w_i. Instead of directly modeling the original distribution, the statistical models estimate the constituent probabilities, which usually results in browsing or storing the statistics of millions of possible word strings since each language may consist of several thousands to millions of words.

In terms of application, language models are often placed

2.1.2 Evaluation Metrics

Quality measurement of language models is often carried out using two different ways. First, statistical language models can be evaluated by the capability to predict a new corpus. The perplexity (PPL) of a word sequence w_1^L is computed as follows.

$$PPL = \exp\left(\frac{\sum_{l=1}^{L} - \ln P(w_i|h_i)}{L}\right) \tag{2.2}$$

It is notable that the exponential term is the average of the negative log-likelihood of every word in the data, while $\log_2 \text{PPL}$ is the **Entropy** of the model. A low perplexity value corresponds to the fact that the language model is able to fits better the data, since the distribution of the model is closer to the unknown distribution of the test data.

Second, the quality of language models can also be evaluated through their impacts on other applications such as Automatic Speech Recognition (ASR) or Statistical Machine Translation (SMT), by reducing the errors in the output of such systems. Specifically, in ASR, the metric used to evaluate the contribution of language models is Word Error Rate (WER) which is the distance between the decoder hypothesis and the reference. Similarly in SMT, the effect of language models can be reflected by the BLEU score [41] or even human judgement.

The main advantage of perplexity is that it is fast to perform and independent to other complex systems. The crediablilty of perplexity, however, depends on the validation/test data as well as the underlying vocabulary. It is also unusable for models that provide unnormalised distributions (sum of the distributions does not equal to 1). More importantly, an improvement in terms of perplexity does not always result in the application improvement. For example, the improvement is required to be at least 10% to be noteworthy for an ASR system [43]

2.2 Standard Neural Network Language Models

2.2.1 Research Context

The language modeling problem has experienced many approaches over the last two decades, most of which aiming at maximising the log-likelihood of the data (minimising the perplexity at the same time), while using some constraints and techniques to generalise to unseen data. The most widely used models rely on n-grams based on the Markov assumption that each word depends on $n-1$ previous words.

$$P(w_i|h_i) \approx P(w_i|w_{i-n+1}^{i-1}) \tag{2.3}$$

Count-based models estimate the probability of each n-gram based on simple counting. The method is unreliable since it struggles at estimating the distribution for rare and unseen n-grams, many of which actually make sense in natural language. Many techniques have been proposed to overcome this weakness, including the smoothing techniques (Witten-Bell, Good-Turing or Knesey-Ney) with back-off and interpolation (estimation based on lower order n-grams) [7, 11, 17, 25, 37, 52]. However, even with complicated smoothing techniques, the main weaknesses of n-gram language models are still exposed.

First, each word in the vocabulary is treated as a totally discrete random variable without any linguistic feature associated. The model relies on statistical occurrences and ignores morphological, syntactic and semantic relationship, by which the lexicon is formulated. There are several attempts to incorporate the word similarity in n-gram based language models. Notably, the class-based language models [5, 39] introduced word clustering and assumed that the distribution of unseen or rate words can be achieved by using the richer statistics from the corresponding class, which is less sparse than the word itself. Also, the structured language models [6, 12] tries to filter out irrelevant context words and focuses on important counterparts by using parse trees, which compensates the lack of syntactic information in n-gram models. Despite such efforts, the language modeling results were still unreliable compared to the Knesey-Ney smoothing technique. However, those works in the literature also suggests that the syntactic and semantic properties of words need to be automatically learned from the data.

Second, n-gram language models struggle to model a long range dependency between the predicted word and the context. Due to the fact that each word in the vocabulary is a separated random variable, the number of parameters to be estimated (statistics of n-grams) grow exponentially with the size of context. The *the curse of dimensionality* refers to the fact that one needs more amount of training data in order to reliably estimate the model when the number of learned parameters increases. For example, if the vocabulary size is 10000, the total number of n-grams for $n =$ is $10^1 6$ theoretically, which is also the number of parameters to be estimated accordingly. Also, the Zipf's law [24] indicates that only a small subset of the vocabulary accounts for most occurrences in the training data, thus it is almost impossible to have a training data that covers all possible n-grams.

The neural network language model, as a consequence, is investigated in order to tackle both problems that traditional count-based models cannot solve.

2.2.2 Concept

In this section, we describe the neural language models [3] (also known as continuous space language models), which are designed to fight the curse of dimensionality in the conventional approaches, by utilising two properties as follows

- Words are no longer discrete variables without any relationship to each other. Alternatively, they are represented as vectors in a continuous space, where similar words are neighbors, such approach is referred as *Distributed word representation* or *Word embedding* [2]. With such representation, each n-gram representation is a combination of the word vectors, which no longer grows exponentially when n is increased and we can mathematically quantify word similarity. More importantly, the distributed representation has to be learned during the language model learning process.

- With the words mapped into continuous vectors and the n-gram context is the combination of the word vectors, the estimation of the probability distribution of each n-gram becomes learning a smoothing function over the context vector and produces the probability for all possible words in the vocabulary. Both the word space and the smoothing function are represented in neural networks, in which they are learnable parameters.

With such design, a neural language model, if successfully trained, can capture the linguistic regularities and generalise from the samples in the training data to similar n-grams. Ideally, the embeddings of similar words would have small distance between each other. Consequently, if we replace a word within an n-gram with a similar word, there is only a small change in the input of the network, which in turn does not make a great difference in the probability distribution at the output. For example, if the model has seen the phrase "The cat is running", it should be able to predict the probability of the phrase "A dog is running" even if the latter does not appear in the training data.

In the literature, distributional semantic models that approximate word meaning from the contextual information has been investigated for a long time [36]. Traditionally most models achieve word embeddings by collecting the occurrence statistics of context words and rely on re-weighting techniques for context informativeness. Distributed representation for generic discrete variables had also been used in neural network models [40], while attempts to solve language modeling with neural network were also observed [45]. However, the standard feed-forward neural language models described in [2] was the first attempt to combine word embeddings that are automatically learned in neural network language models.

2.2.3 Standard architecture

This section is dedicate to describe the very first neural language model, which is illustrated in Figure 2.1. The model takes the context words as inputs and outputs the probability

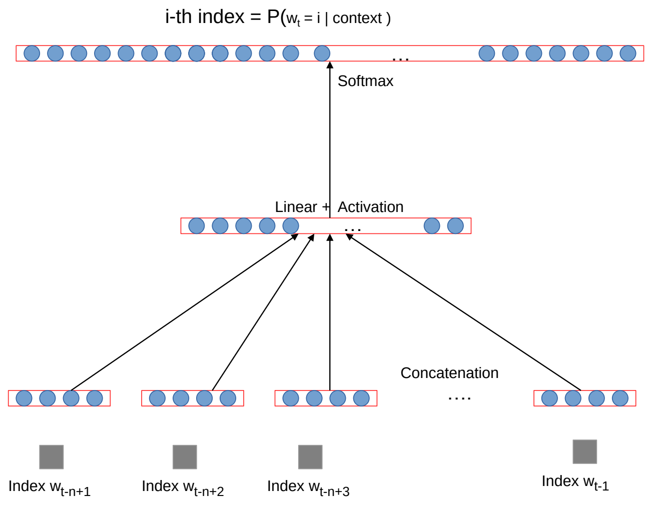

Figure 2.1: Neural Language Model by Bengio et al [2].

distribution for *all* words in the vocabulary. It consists of three basic components: the input layer, the hidden layer and the output layer. . The hidden layer can be extended to two or more layers which does not change the fundamental architecture.

Input layer The input layer constructs the embeddings of the words in the context, by using a shared word space and mapping each word in the context to a real-valued vector. Concretely, each word w_i is represented as an 1-of-V coding, which is a long vector with the size of vocabulary V with all zeros except for the element corresponding to the word's index in the dictionary. Using this form of sparse coding, the word space (also called projection matrix R) is a matrix that contains V rows and each word embedding v_i corresponds to one row of the matrix. The number of columns of the matrix is the size of the embedding, which is a tunable hyper-parameter. Notably, the values of the word vectors do not depend on the position of the word in the context. The context vector i is the concatenation of the word vectors.

$$i = \{R^T v_1; R^T v_2; ....; R^T v_{n-1}\} \tag{2.4}$$

Hidden layer In the hidden layer, the input (context vector) is transformed nonlinearly, where each layer activation values are defined by

$$\begin{aligned} h^j_{out} &= f(W^j h_{in} + b^j) \\ h^1_{in} &= i \end{aligned} \tag{2.5}$$

In equation 2.5, each hidden layer h^j has the corresponding weights W^j and b^j. The input of the first hidden layer is the context vector produced from the input (projection) layer. The size of the subsequent hidden layers are tunable hyper parameters. f denotes a nonlinear activation function. Popular choices for the activation function are Tangent Hyperbolic, Sigmoid or ReLU, expressed in equation 2.6.

$$f(x) = \begin{cases} \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)} \text{ if } f = \text{Tanh} \\ \frac{1}{1+\exp(-x)} \text{ if } f = \text{Sigmoid} \\ \max(0, x) \text{ if } f = \text{ReLU} \end{cases} \tag{2.6}$$

Output layer The final layer of the network produces the probability distribution for all words in the vocabulary, thus having totally V nodes. Each neuron in the layer is associated to the probability of one word, as shown in Figure 2.1. First, a linear transformation is used to obtain the unnormalised distribution:

$$o = W^o h + b^o \tag{2.7}$$

Notably, the values of o is unnormalised because each element in o is associate to the score of each output word given the context vector. W^o and b^o are the corresponding weights and biases of the layer. Importantly, W^o has the same form as the projection matrix at the input layer, since it also learns embedding for each word in the output vocabulary. Subsequently, the true probability distribution is estimated thanks to the *softmax* function:

$$p(w_i|h) = \frac{\exp(o_i)}{\sum_j \exp(o_j)} \tag{2.8}$$

In equation 2.8, the probability of each word w_i given the encoded context h is estimated by normalizing all values in o. Overall, the main tunable hyper-parameters of the network are the order of n-grams (the number of words in the context, which can range from 6 to 15), the sizes of hidden layers, and the word embedding size. The free parameters Θ that are algorithmically learned from the data includes the projection matrix R, the weight matrices W^h at the hidden layers, W^o at the output layer and the biases b^h, b^o.

2.2.4 Training method

From the machine learning perspective, the neural network language model has transformed the statistcal language modeling problem from a generative learning process to a discriminative classification problem. The free parameters of the network are trained by minimising the objective function, which is the log-likelihood $\mathcal{L}$ of the parameters Θ given the training samples. The parameters are updated after each iteration based on some optimization techniques, among which Stochastic Gradient Descent (SGD) is most commonly used in neural language models [14, 33, 53]. SGD and other variants such as Adadelta [54] or RMSProp [51] require the computation of the first order derivatives of the loss function with respected to the parameters, which can be performed efficiently with the back-propagation algorithm [29].

2.2.5 Optimisation Process

In this section, we describe the back-propagation flow in the standard feed forward neural language model - the core of the optimisation process. Back-propagation [44] involves using a dynamic programming strategy to compute the derivatives of the loss function with respect to the parameters layer by layer, based on the chain rules. In the standard network, the error derivatives are back-propagated from the output layer to the input (projection) layer.

Objective Function The network aims at maximising the probability of the label word given the context, hence the objective function is typically chosen as the Negative Log-Likelihood function. Assuming we have N samples, each of which is an n-gram, we can compute the loss function over the training data as follows:

$$\mathcal{L} = -\sum_{i}^{N} \log P(w_i | w_{i-n+1}^{i-1}) \tag{2.9}$$

For ease of understanding, we denote the derivative of the loss function $\mathcal{L}$ at *each* sample or batch of samples with respect to a variable $x \in \Theta$ by dx.

For each sample, let's w be the index of the predicted word w_i in the vocabulary, we have:

$$\begin{aligned} -\log P(w_i | w_{i-n+1}^{i-1}) &= -\log P(w|h) \\ &= -log(\frac{\exp(o_w)}{\sum_i exp(o_i)}) \\ &= log(\sum_i \exp(o_i)) - o_w \end{aligned} \tag{2.10}$$

Subsequently, we compute the error derivatives dx given the parameters in each layer using back-propagation:

Output layer The derivatives at the output layer:

$$do_i = \begin{cases} 1 - p_i \text{ if } i == w \\ -p_i \text{ otherwise} \end{cases} \tag{2.11}$$

Notably, o_i denotes the i^{th} element of the vector o, which is the unnormalised distribution of the vocabulary given the encoded context h.

Hidden layers As a result, we can compute the derivatives with respect to the parameters and the previous hidden layer h, based on the original inference from Equation 2.7.

$$\begin{aligned} dW^o &= doh^T \\ db^o &= do \\ dh &= W^{oT} do \end{aligned} \tag{2.12}$$

For the inner hidden layers, the derivatives for the linear components are computed similarly, by replacing do with dh (the output of the previous hidden layer is the input of the next hidden layer). However, we have to take into account the non-linear activation functions, which are shown in the inference equation for the previous hidden layers:

$$h_{next} = f(W^h h_{prev} + b^h) \tag{2.13}$$

With W^h and b^h are the corresponding weights connecting h_prev to h_next. The derivatives are computed as follows:

$$\begin{aligned} d[W^h h_{prev} + b^h] &= f'(h_{next}) dh_{next} \\ db^h &= d[W^h h_{prev} + b^h] \\ dW^h &= d[W^h h_{prev} + b^h] h_{prev}^T \\ dh_{prev} &= {W^h}^T d[W^h h_{prev} + b^h] \end{aligned} \tag{2.14}$$

The derivative function of f, f':

$$f(x) = \begin{cases} 1 - \text{Tanh}(x)^2 \text{ if } f = \text{Tanh} \\ \text{Sigmoid}(x) - \text{Sigmoid}(x)^2 \text{ if } f = \text{Sigmoid} \\ 1 \text{ when } x > 0 \text{ and } 0 \text{ otherwise if } f = \text{ReLU} \end{cases} \tag{2.15}$$

Input Layer The derivative function of the input layer is identical to Equation 2.14 and Equation 2.15, considering that the input layer is the layer before the first hidden layer.

Parameter Update After obtaining the derivatives of the loss function with respect to all parameters in the network, we can update the parameters following Stochastic Gradient Descent. The method is based on the phenomenon that the gradient of a function always points towards the direction of maximal increase at any point. The update rule is as follows with the learning rate parameter $\alpha > 0$:

$$x = x - \alpha dx \tag{2.16}$$

The learning rate is also considered as a function of the number of samples trained in the data. Typically, the learning rate is updated after the model observes a number of training examples, with two most common ways. The first way is to exponentially decrease the learning rate after some training samples with a *learning rate decay*, normally an epoch (training all samples in the training data). The second way is to reduce the learning rate based on a validation data. After each epoch, if the perplexity on the validation data is decreased, the learning rate is kept the same, otherwise it is multiplied by the learning rate decay.

2.3 Recurrent Neural Language Models

2.3.1 Recurrent Structure

Compared to the statistical n-gram models, the feed-forward neural language models had created a considerable leap in representation by combining distributed representation of words with a robust classifier to generalise from observed sequences. As can be seen from various works [14, 46], the feed-forward language model significantly outperformed the traditional Knesey-Ney back-off models. However, the feed-forward models require a fixed input size, thus still rely on the Markov Assumption which limits the context to a particular number of words, even when the context size can be huge up to 10. In order to model long sentences, or even paragraphs with long-term dependencies, it is beneficial to investigate in models that can be flexible in terms of input size. For example, if the distance between the open bracket and the closed counterpart is further than the n-gram input size, the feed forward model may forget to close the brackets after seeing the initial one. Ideally, as the learning process of human is associated with a memory that keeps the current information (such as topics), similar structure should be simulated and integrated in the network.

Recurrent Neural Networks RNN [10] are a class of neural networks that can efficiently model sequences by using a dynamic memory structure. While the feed-forward network can only receive one input and compute the corresponding output without any relation with other inputs, the recurrent counterpart takes the input as a *series of time step* $x_1, x_2, \ldots, x_n$ and processes them one by one, taking into account the information stored in the previous steps. Concretely, for each input x_i, the network updates the hidden memory h_i based on the previous one h_{i-1}.

$$h_i = \mathcal{F}(x_i, hi - 1) \tag{2.17}$$

We will cover some popular RNN variations in the upcoming sections. In general, the strong point of these models lie in the ability to dynamically model sequences with arbitrary length, which the feed-forward neural networks cannot achieve. The advantage comes with the cost that the recurrent models are generally hard to train, due to the properties of back-propagation. A change in an arbitrary position of the sequence can lead to a change in the objective function, therefore training methods for RNNs typically have to trace back the previous time steps. In other words, the RNNs are equivalent to feed-forward neural networks with many hidden layers that share parameters across each other. Importantly, the model capacity of RNNs do not depend on the length of the sequences, but in the recurrence mechanism - the way the hidden layers are updated.

Recurrent Language Models Language Modeling can be viewed as a sequence modeling problem, in which each time step corresponds to one word. The first recurrent language model (RNNLM) [33] employed the “Vanilla” model of Elman et al [10] as can be seen in Figure 2.2, in which the hidden steps are updated as follows:

$$h_i = f(Wx_i + Uhi - 1 + b) \tag{2.18}$$

The activation function f can be either Tangent Hyperbolic, Sigmoid or ReLU as mentioned before. The starting state h_0 is set to 0 to denote the initial state of the memory. In each time step, the RNNLM can optionally produces the probability distribution for a

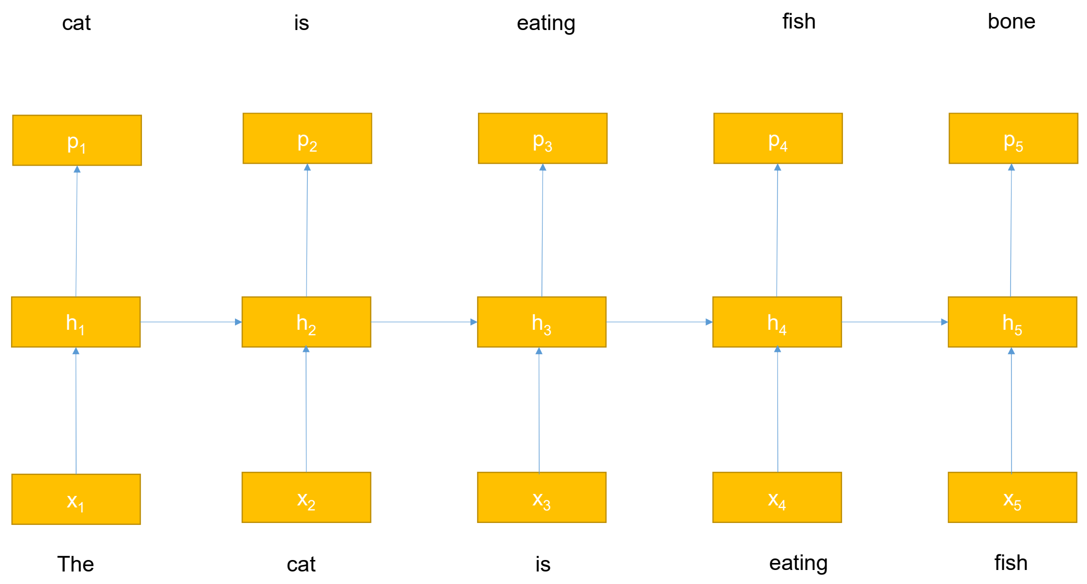

Figure 2.2: A simple RNNLM [33] predicting the sequence "The cat is eating fish bone".

predicted word, given the sequence that network has scanned previously. The probability distribution over the vocabulary is derived similarly to the feed-forward networks:

$$\begin{aligned} o_i &= W^o h_i + b^o \\ p_i &= \text{softmax}(o_i) \end{aligned} \tag{2.19}$$

with the Softmax function explained in Equation 2.8. To be clear, o_i and p_i denote the unnormalised and normalised distribution generated at time step i. For the language modeling scenario, the input and output samples of the network in each training iteration are two sequences X and Y in which the output sequence is the shift-by-1 version of the input sequence. The parameter set of the network including W, b, U, W^o and b^o are shared across time steps.

2.3.2 Training Recurrent Networks

Similarly to the feed-forward models, The recurrent models can be efficiently trained with stochastic gradient descent (SGD). However, since the networks contain shared parameters at arbitrary numbers of time steps, the gradients are computed differently using back-propagation through time (BPTT). The details of this algorithm are divided into two parts: the local gradients computed at each time step, and the global gradients accumulated by un-folding the networks.

At each time step

For global derivatives

2.3.3 Variations

Problems with BPTT Although truncated back-propagation through time provides a practical training method for RNNs, the nonlinear iterative nature of the simple RNN architecture still makes capturing long-term dependencies difficult. The two common problems encountered while training RNNs are the *exploding* and the *vanishing* gradients [1, 42]. On the one hand, the gradients can be exponentially large as in the back-propagation through time process which is detrimental for learning. One the other hand, we can also experience the phenomenon that gradients go quickly towards zero after being propagated through time steps. Consequently, the model is not able to track the signal and complete loses the memory trace in the past. For example, the original RNNLM normally has to truncate the BPTT at about $5 - 10$ steps, which has the same modeling capacity and performance with 10-gram feed-forward models [15, 34]. The gradient exploding problem can be tackled adequately using gradient clipping. Pascanu et al [42] suggested to clip the norm of the gradients (for all parameters): given a gradient vector dx that is computed with BPTT, if the norm $||dx||$ is greater than a threshold value δ, then dx would be softly scaled:

$$dx \leftarrow dx \frac{||dx||}{\delta} \tag{2.20}$$

Dealing with gradient vanishing A number of solutions were proposed to solve the gradient vanishing problem. We will make a brief review of the most prominent approaches. First, Mikolov et al [35] proposed to integrate another memory layer which is formed by the bag-of-word addition of the input words over time, decays slower than the main hidden memory and is initialised as an identity matrix. The same initialisation trick is applied together with using Rectified Linear Units (ReLU) as the activation function in [28]. While both works mentioned are fairly simple, the RNNs can also be trained efficiently with second order derivatives using Hessian-Free optimization [32]. Even though the method was prove to allow the network to acquire a reliable memory which can remain stable after hundreds of time steps, it is not easy to implement efficiently compared to traditional back-propagation. The most successful method that is applied to sequential modeling in general and language modeling in particular is the Long-Short Term Memory LSTM networks [19] that use an explicit memory cell combined with a gating mechanism to intensively deal with the gradient vanishing problem.

LSTM Structure

Training LSTM

2.4 Analysis

2.4.1 Computational Complexity

Bottleneck in the Softmax layer

Approximating the Softmax

2.4.2 Improvement Room

Chapter 3

Convolutional Neural Language Models

3.1 Related works

Convolutional Neural Networks (CNNs) were originally designed to deal with hierarchical representation in Computer Vision [30]. Deep convolutional networks have been successfully applied in image classification and understanding [16, 48]. In such systems the convolutional kernels learn to detect visual features at both local and more abstract levels.

In NLP, CNNs have been mainly applied to static classification task for discovering latent structures in text. Kim [23] use a CNN to tackle sentence classification, with competitive results. This work also introduces kernels with varying window sizes to learn complementary features at different aggregation levels. Kalchbrenner et al. [22] propose a convolutional architecture for sentence representation that vertically stacks multiple convolution layers, each of which can learn independent convolution kernels. CNNs with similar structures have also been applied to other classification tasks, such as semantic matching [20], relation extraction [38] and information retrieval [47]. In contrast, Collobert et al. [8] explore a CNN architecture to solve various sequential and non-sequential NLP tasks such as part-of-speech tagging, named entity recognition and also language modeling. This is perhaps the work that is closest to ours in the existing literature. However, their model differs from ours in that it uses a max-pooling layer that picks the most activated feature across time, thus ignoring temporal information, whereas we explicitly avoid doing so. More importantly, the language models trained in this work are only evaluated through downstream tasks and through the quality of the learned word embeddings, but not on the sequence prediction task itself.

In this work we focus on statistical language modeling, which differs from most of the tasks where CNNs have been applied before in multiple ways. First, the input normally consists of incomplete sequences of words rather than complete sentences. Second, as a classification problem, it features an extremely large number of classes (the words in a large vocabulary). Finally, temporal information, which can be safely discarded in many settings with little impact in performance, is critical here: An n-gram appearing close to the predicted word may be more informative, or yield different information, than the same n-gram appearing several tokens earlier.

3.2 Base FFLM

Our CNN is constructed by extending a feed-forward language model (FFLM) with convolutional layers. In what follows, we first explain the implementation of the base FFLM and next, we describe the CNN that we study.

Our baseline feed-forward language model (FFLM) is almost identical to the original model proposed by Bengio et al. [3], with only slight changes to push its performance as high as we can, producing a very strong baseline. In particular, we extend it with highway layers and use Dropout as regularization. The model is illustrated in Figure 3.1 and works as follows. First, each word in the input n-gram is mapped to a low-dimensional vector (viz. embedding) though a shared lookup table. Next, these word vectors are concatenated and fed to a highway layer [49]. Succinctly, highway layers improve the gradient flow of the network by computing as output a convex combination between its input (called the *carry*) and a traditional non-linear transformation of it (called the *transform*). As a result, if there is a neuron whose gradient cannot flow through the transform component (e.g., because the activation is zero), it can still receive the back-propagation update signal through the carry gate. We empirically observed the usage of a single highway layer to improve importantly the performance of the model. Even though a systematic evaluation for this model is beyond the scope of the current paper, our empirical results demonstrate that it is a very competitive one (see Section **??**).

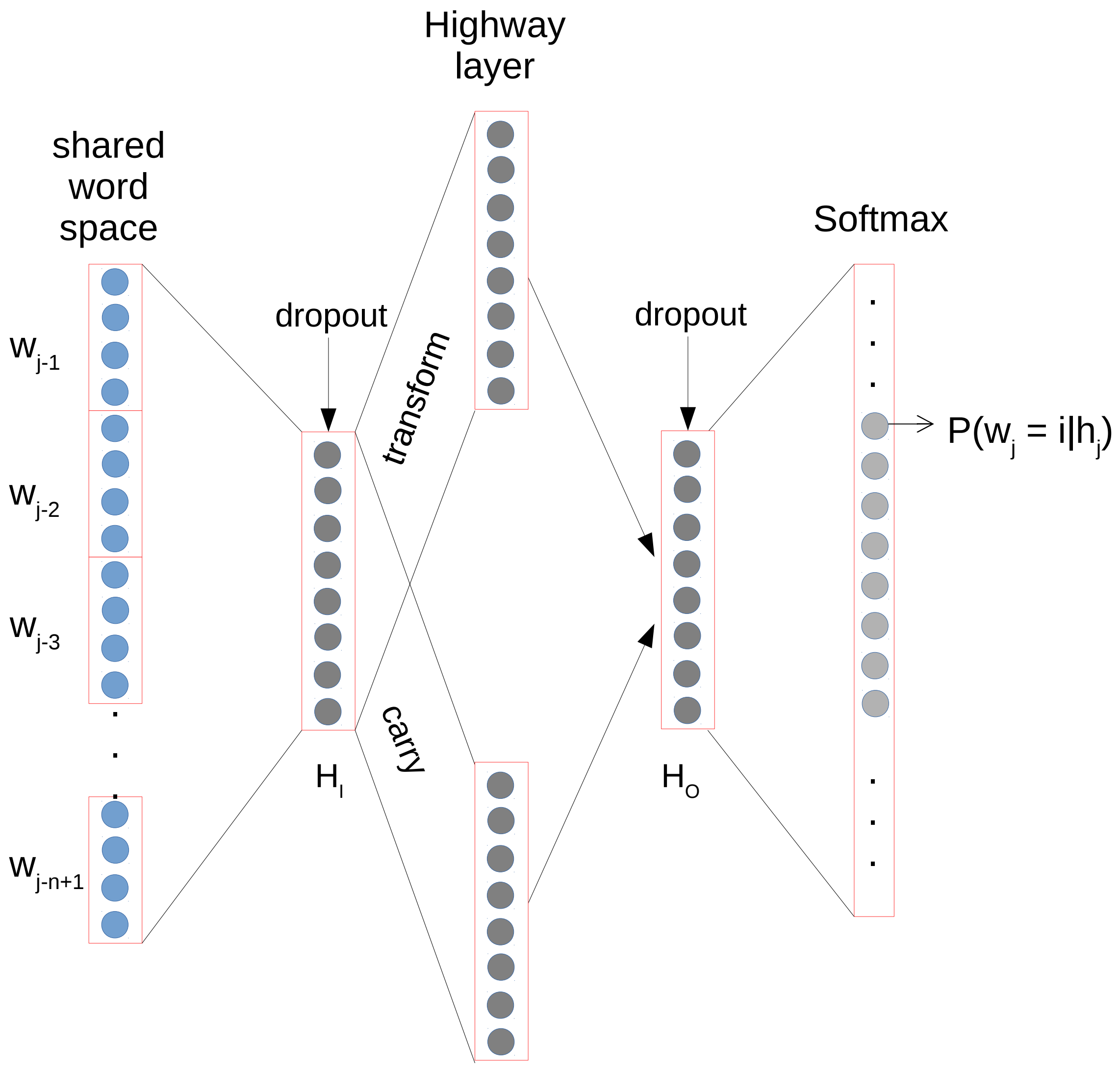

Figure 3.1: Overview of baseline FFLM.

Finally, a softmax layer computes the model prediction for the upcoming word. We use ReLU for all non-linear activations and Dropout [18] is applied between each hidden layer.

3.3 CNN and variants

The proposed CNN network is produced by injecting a convolutional layer right after the words in the input are projected to their embeddings (Figure 3.2). Rather than being concatenated into a long vector, the embeddings $x_i \in \mathbb{R}^k$ are concatenated transversally producing a matrix $x_{1:n} \in \mathbb{R}^{n \times k}$, where n is the size of the input and k is the embedding size. This matrix is fed to a time-delayed layer, which convolves a sliding window of w input vectors centered on each word vector using a parameter matrix $W \in \mathbb{R}^{w \times k}$. Convolution is performed by taking the dot-product between the kernel matrix W and each sub-matrix $x_{i-w/2:i+w/2}$ resulting in a scalar value for each position i in input context. This value represents how much the words encompassed by the window match the feature represented by the filter W. A ReLU activation function is applied subsequently so negative activations are discarded. This operation is repeated multiple times using various kernel matrices W, learning different features independently. Here we tie the number of learned kernels to be the same as the embedding dimensionality k. Thus, the output of this stage will be another matrix of dimensions $n \times k$ containing the activations for each kernel at each time step.

Next, we add a batch normalization stage immediately after the convolutional output, which facilitates learning by addressing the internal covariate shift problem and regularizing the learned representations [21].

Finally, this feature matrix is directly fed into a fully connected layer that can project the extracted features into a lower-dimensional representation. This is different from previous work, where a max-over-time pooling operation was used to find the most activated feature in the time series. Our choice is motivated by the fact that the max pooling operator loses the specific position where the feature was detected, which is important for word prediction.

After this initial convolutional layer, the network proceeds identically to the FFNN by feeding the produced features into a highway layer, and then, to a softmax output.

This is our basic CNN architecture. We also experiment with three expansions to the basic model, as follows. First, we generalize the CNN by extending the shallow linear kernels with deeper multi-layer perceptrons, in what is called a Network-in-Network (**NIN**) structure [31]. This allows the network to produce non-linear filters, and it has achieved state-of-the-art performance in object recognition while reducing the number of total layers compared to other mainstream networks. Concretely, we implement NIN networks by using another convolutional layer with a 1×1 kernel on top of the convolutional layer output. Notably, we do not include the global pooling layer in the original work.

Second, we explore stacking convolutional layers on top of each other (Multi-layer CNN or **ML-CNN**) to connect the local features into broader regional representations, as commonly done in computer vision. While this proved to be useful for sentence representation [22], here we have found it to be rather harmful for language modeling, as shown in Section **??**.

Finally, we consider combining features learned through different kernel sizes (**COM**), as depicted in Figure 3.3. For example, we can have a combination of kernels that learn filters over 3-grams with others that learn over 5-grams. This is achieved simply by applying in parallel two or more sets of kernels to the input and concatenating their respective outputs [23].

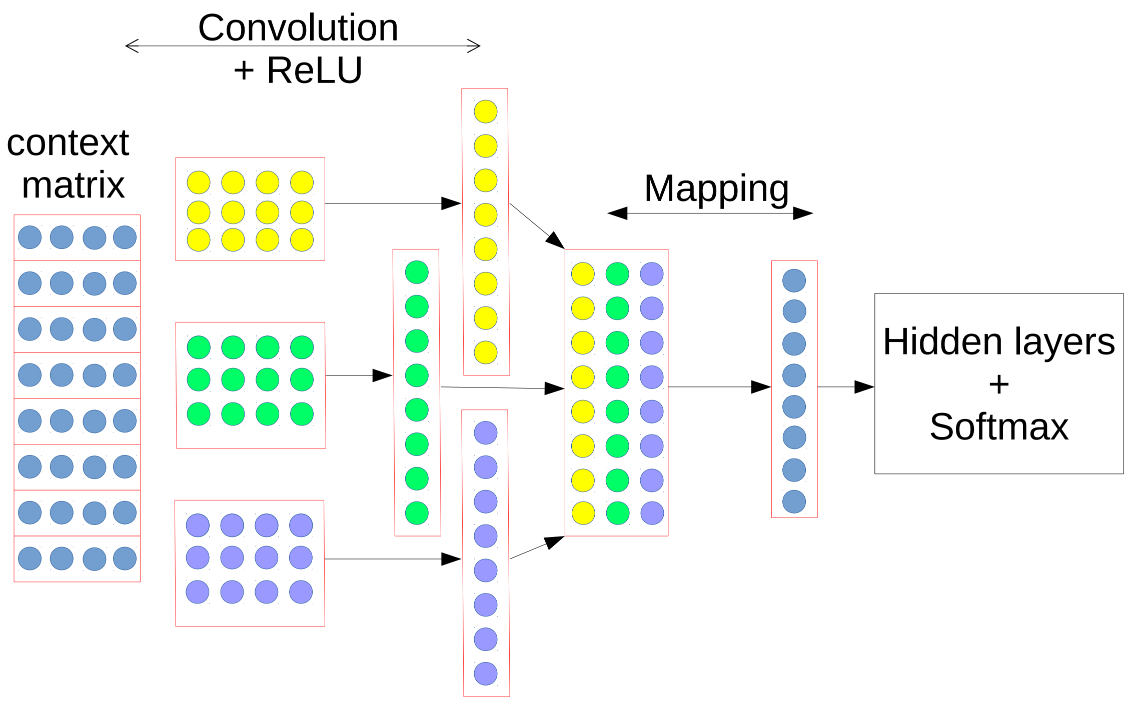

Figure 3.2: Convolutional layer on top of the context matrix.

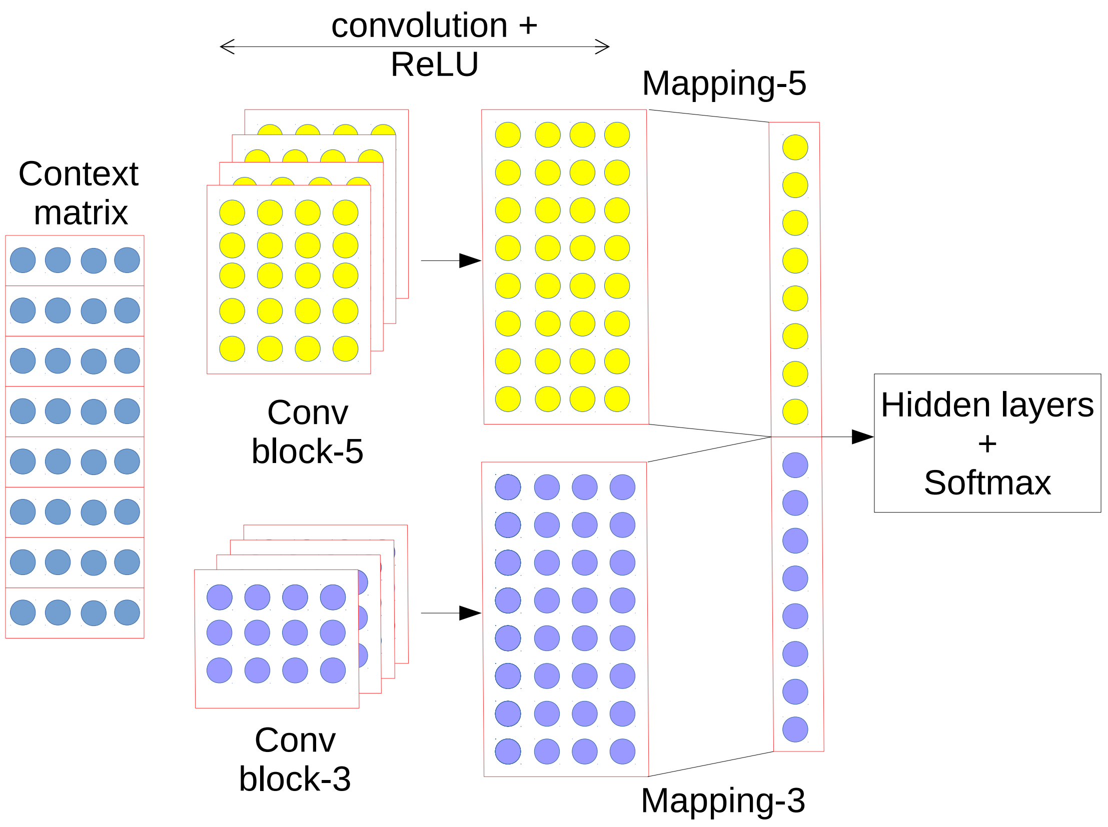

Figure 3.3: Combining kernels with different sizes. We concatenate the outputs of 2 convolutional blocks with kernel size of 5 and 3 respectively.

Chapter 4

Experiments

We evaluate our model on two English corpora of different sizes and genres that have been used for language modeling evaluation before. The **Penn Treebank** contains one million words of newspaper text with 10K words in the vocabulary. We reuse the preprocessing and train/test/valid division from [35]. **Europarl-NC** is a a 64-million word corpus that was developed for a Machine Translation shared task [4], combining Europarl data (from parliamentary debates in the European Union) and News Commentary data. We preprocessed the corpus with tokenization and true-casing tools from the Moses toolkit [26]. The vocabulary is composed of words that occur at least 3 times in the training set and contains approximately 60K words.

We train our models using Stochastic Gradient Descent (SGD) and we reduce the learning rate by a fixed proportion every time the validation perplexity increases after one epoch. The values for learning rate, learning rate shrinking and mini-batch sizes as well as context size are fixed once and for all based on insights drawn from previous work [9, 14, 50] and through experimentation with the Penn Treebank validation set. Experimentally we found SGD to be efficient and adequately fast, while other learning algorithms involve additional hyper parameters (such as alpha in RMSprop [51])

Specifically, the learning rate is set to 0.05, with mini-batch size of 128 (we do not take the average of loss over the batch, and the training set is shuffled). We multiply the learning rate by 0.5 every time we shrink it and clip the gradients if their norm is larger than 12. The network parameters are initialized randomly on a range from -0.01 to 0.01 and the context size is set to 16. In Section **??** we show that this large context window is fully exploited.

For the base FFNN and CNN we study embedding sizes (and thus, number of kernels) $k = 128, 256$. For $k = 128$ we explore the simple CNN, incrementally adding NIN and COM variations (in that order) and, alternatively, using a ML-CNN. For $k = 256$, we only explore the former three alternatives. For the kernel size, we set it to $w = 3$ words for the simple CNN (out of options $3, 5, 7, 9$), whereas for the COM variant we use $w = 3$ and 5. However, we observed the models to be generally robust to this parameter. Dropout rates are tuned specifically for each combination of model and dataset based on the validation perplexity. We also add small dropout $(0.05 - 0.15)$ when we train the networks on the small corpus (Penn Treebank).

The experimental results for recurrent neural network language models, such as Recurrent Neural Networks (RNN) and Long-Short Term Memory (LSTM), on the Penn Treebank are quoted from previous work; for Europarl-NC, we train our own models (we also report the performance of these in-house trained RNN and LSTM models on the Penn Treebank for reference). Specifically, we train LSTMs with embedding size $k = 256$ and number of layers $L = 2$ as well as $k = 512$ with $L = 1, 2$. We train one RNN with $k = 512$ and $L = 2$. To train these models, we use the published source code from [53]. Our own models are also implemented in Torch7[1] for easier comparison.

For all models trained on Europarl-NC, we used hierarchical softmax to speed up training, clustering the words in the vocabulary into 245 classes based on word frequency.

[1]We'll link the source code here if the paper is accepted

no matter how	as little as	a merc spokesman	amr chairman robert
are afraid how	of more than	a company spokesman	chief economist john
question is how	as high as	a boeing spokesman	chicago investor william
remaining are how	as much as	a fidelity spokesman	exchange chairman john
to say how	as low as	a quotron spokeswoman	texas billionaire robert
would allow the	more evident among	facilities will substantially	have until nov.
does allow the	a dispute among	which would substantially	operation since aug.
still expect ford	bargain-hunting among	dean witter actually	quarter ended sept.
warrant allows the	growing fear among	we 'll probably	terrible tuesday oct.
funds allow investors	paintings listed among	you should really	even before june

Figure 5.1: Some example phrases that have highest activations for 8 example kernels (each box), extracted from the validation set of the Penn Treebank. Model trained with 256 kernels for 256-dimension word vectors.

Chapter 5

Model Analysis

In what follows, we obtain insights into the inner workings of the CNN by looking into the linguistic patterns that the kernels learn to extract and also studying the temporal information extracted by the network in relation to its prediction capacity.

Learned patterns To get some insight into the kind of patterns that each kernel is learning to detect, we fed trigrams from the validation set of the Penn Treebank to each of the kernels, and extracted the ones that most highly activated the kernel. Some examples are shown in Figure 5.1. Since the word windows are made of embeddings, we can expect

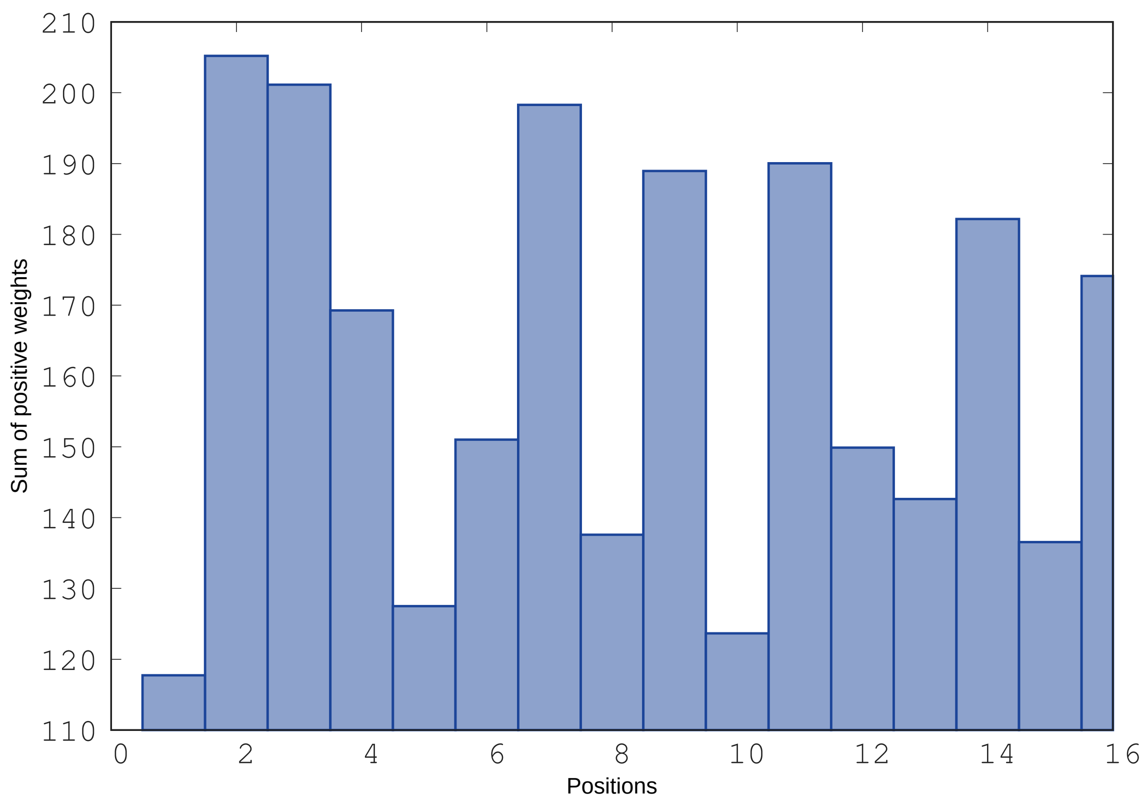

Figure 5.2: The distribution of positive weights over context positions, where 1 is the position closest to the predicted word.

patterns with similar embeddings to have close activation outputs. This is borne out in the analysis: The kernels specialize in distinct features of the data, including more syntactic-semantic constructions (cf. the “comparative kernel” including *as* . . . *as* patterns, but also *of more than*) and more lexical or topical features (cf. the “ending-in-month-name” kernel). Even in the more lexicalized features, however, we see linguistic regularities at different levels being condensed in a single kernel: For instance, the “spokesman” kernel detects phrases consisting of an indefinite determiner, a company name (or the word *company* itself) and the word “spokesman”. We hypothesize that the convolutional layer adds an “I identify one specific feature, but at a high level of abstraction” dimension to a feed-forward neural network, similarly to what has been observed in image classification [27].

Temporal information To the best of our knowledge, the longest context used in feed-forward language models is 10 [15], where no significant change in terms of perplexity was observed for bigger context sizes, even though in that work only same-sentence contexts were considered. In our experiments, we use a larger context size of 16 while removing the sentence boundary limit (as commonly done in n-gram language models) such that the network can take into account the words in the previous sentences.

To analyze whether all this information was effectively used, we took our best model, the CNN-NIN-COM model with embedding size of 256 (fourth line, second block in Table **??**), and we identified the weights in the model that map the convolutional output (of size $n \times k$) to a lower dimensional vector (the "mapping" layer in Figure 3.2). Recall that the output of the convolutional layer is a matrix indexed by time step and kernel index representing the activation of the kernel when convolved with a window of text centered around the given time step. Thus, output units of the above mentioned mapping predicate over an ensemble of kernel activations for each time-step . We can identify the patterns that they learn to detect by extracting the time-kernel combinations for which they have positive weights (since we have ReLU activations, negative weights are equivalent to ignoring a feature). First, we asked ourselves whether these units tend to be more focused on the time-steps closer to the target or not. To test this, we calculated the sum of the positive weights for each position in time using an average of the mappings that correspond to each output unit. The results are shown in Figure 5.2. As could be expected, positions that are close to the token to be predicted have many active units (local context is very informative; see positions 2-4). However, surprisingly, positions that are actually far from the target are also quite active (see positions 10-14, with a spike at 11). It seems like the CNN is putting quite a lot of effort on characterizing long-range dependencies.

Next, we checked that the information extracted from the positions that are far in the past are actually used for prediction. To measure this, we artificially lesioned the network so it would only read the features from a given range of timesteps (words in the context). To lesion the network we manually masked the weights of the mapping that focus on times outside of the target range by setting them to zero. We started using only the word closest to the final position and sequentially unmasked earlier positions until the full context was

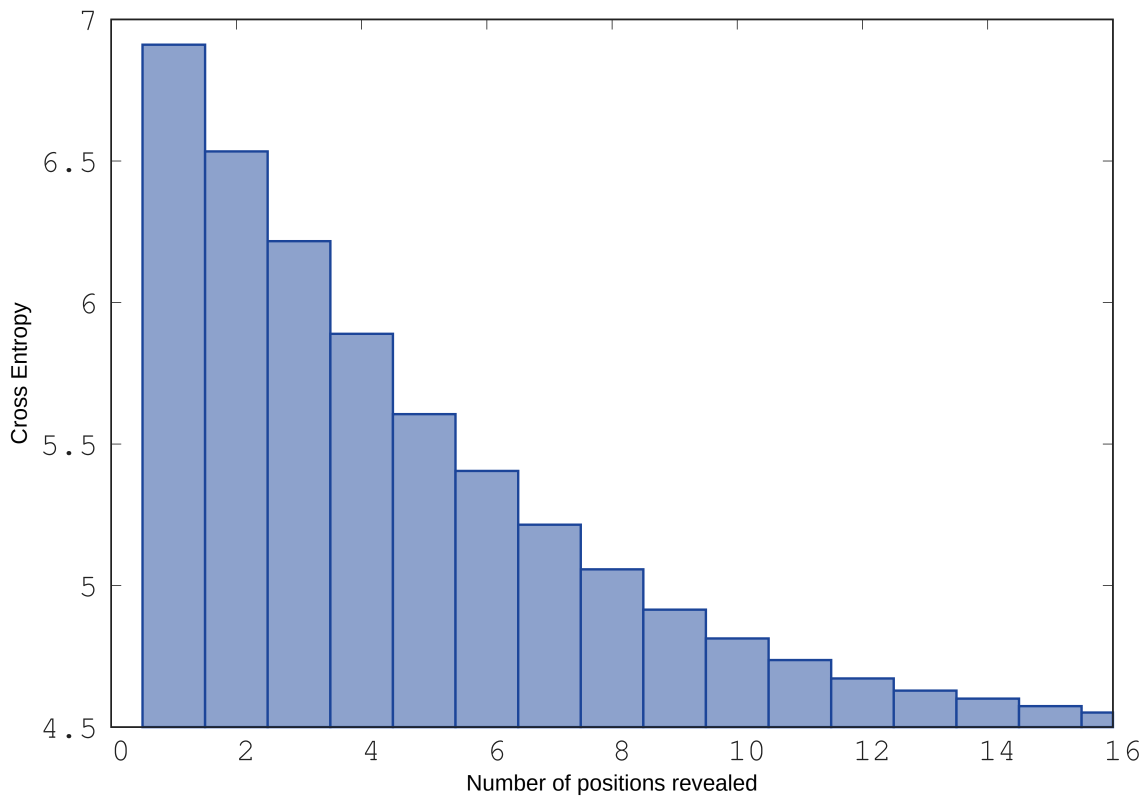

Figure 5.3: Perplexity change over position, by incrementally revealing the Mapping's weights corresponding to each position.

used again. The result of this experiment is presented in Figure 5.3, and it confirms our previous observation that positions that are the farthest away contribute to the predictions of the model. The perplexity drops dramatically as the first positions are unmasked, and then decreases more slowly. However, the last few positions still contribute to the final perplexity. Indeed, the decrease almost follows a power-law distribution, in which the perplexity decreases whenever we unmask a new position according to the following equation.

$$f(x) = 10e3 \times x^{-0.9} \tag{5.1}$$

Chapter 6

Conclusion

In this work, we have investigated the use of Convolutional Neural Networks for language modeling, a sequential prediction task. We incorporate a CNN layer on top of a strong feed-forward model enhanced with modern techniques like Highway Layers and Dropout. Our results show a solid 13% improvement in perplexity with respect to the feed-forward model across two corpora of different sizes and genres when the model uses Network-in-Network and combine kernels of different window sizes. However, even without these additions we show CNNs to effectively learn language patterns to significantly decrease the model perplexity.

In our view, this improvement responds to two key properties of CNNs, highlighted in the analysis. First, as we have shown, they are able to integrate information from larger context windows, using information from words that are as far as 16 positions away from the predicted word. Second, as we have qualitatively shown, the kernels learn to detect specific patterns at a high level of abstraction. This is analogous to the role of convolutions in Computer Vision. The analogy, however, has limits; for instance, a deeper model stacking convolution layers harms performance in language modeling, while it greatly helps in Computer Vision. We conjecture that this is due to the differences in the nature of visual vs. linguistic data. The convolution creates sort of abstract images that still retain significant properties of images. When applied to language, it detects important textual features but distorts the input, such that it is not text anymore.

As for recurrent models, even if our model outperforms RNNs, it is well below state-of-the-art LSTMs. Since CNNs are quite different in nature, we believe that a fruitful line of future research could focus on integrating the convolutional layer into a recurrent structure for language modeling, as well as other sequential problems, perhaps capturing the best of both worlds.

Bibliography

[1] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166, 1994.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

[3] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer, 2006.

[4] Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 1–46, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL `http://aclweb.org/anthology/W15-3001`.

[5] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[6] Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech & Language*, 14(4):283–332, 2000.

[7] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[9] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*, pages 1370–1380. Citeseer, 2014.

[10] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[11] Marcello Federico, Nicola Bertoldi, and Mauro Cettolo. Irstlm: an open source toolkit for handling large scale language models. In *Interspeech*, pages 1618–1621, 2008.

[12] Denis Filimonov and Mary Harper. A joint language model with fine-grain syntactic tags. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1114–1123. Association for Computational Linguistics, 2009.

[13] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[14] Le Hai Son, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain, and François Yvon. Structured output layer neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5524–5527. IEEE, 2011.

[15] Le Hai Son, Alexandre Allauzen, and François Yvon. Measuring the influence of long range dependencies with neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 1–10. Association for Computational Linguistics, 2012.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[17] Kenneth Heafield. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197. Association for Computational Linguistics, 2011.

[18] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `http://arxiv.org/abs/1207.0580`.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*, pages 2042–2050, 2014.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[22] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 655–665, 2014. URL `http://aclweb.org/anthology/P/P14/P14-1062.pdf`.

[23] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[24] Zipf George Kingsley. Selective studies and the principle of relative frequency in language, 1932.

[25] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

[26] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[28] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[29] B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[30] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[31] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[32] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040, 2011.

[33] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[34] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[35] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014.

[36] George A Miller and Walter G Charles. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28, 1991.

[37] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

[38] Thien Huu Nguyen and Ralph Grishman. Relation extraction: Perspective from convolutional neural networks. In *Proceedings of NAACL-HLT*, pages 39–48, 2015.

[39] Thomas R Niesler and Philip C Woodland. A variable-length category-based n-gram language model. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 164–167. IEEE, 1996.

[40] Alberto Paccanaro and Geoffrey E. Hinton. Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):232–244, 2001.

[41] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[42] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[43] Ronald Rosenfeld. Two decades of statistical language modeling: Where do we go from here. In *Proceedings of the IEEE*, page 2000, 2000.

[44] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[45] Jürgen Schmidhuber and Stefan Heil. Sequential neural text compression. *IEEE Transactions on Neural Networks*, 7(1):142–146, 1996.

[46] Holger Schwenk. Continuous space language models. *Computer Speech & Language*, 21(3):492–518, 2007.

[47] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 101–110. ACM, 2014.

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[49] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`.

[50] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439, 2015.

[51] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[52] Ian H Witten and Timothy C Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *Ieee transactions on information theory*, 37(4):1085–1094, 1991.

[53] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[54] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.