Text Representation using Convolutional Networks

by

Xiang Zhang

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January 2019

Professor Yann LeCun

Dedication

To my parents, with affection.

Acknowledgements

All the research projects in this dissertation are not possible without the support from my advisor professor Yann LeCun, who offered the best advisement one could ever hope for, even with the tremendous amount of work he needs to do for other obligations due to the booming of deep learning. I also want to express my gratitude for professor Sam Bowman and professor Kyunghyun Cho, whose suggestions for my dissertation proposal helped me to understand better about the position of these research projects in the natural language processing community.

The readers of this dissertation – Dr. Sumit Chopra and Dr. Arthur Szlam – helped me in a great deal during the 2015 summer internship at Facebook, especially with the understanding of challenges in machine reasoning for dialogue systems. Dr. Antoine Bordes and Dr. Jason Weston from Facebook AI Research (FAIR) also provided many inspiring discussions at that time that formed my view on research in NLP.

I had close collaboration with Jake (Junbo) Zhao on the character-level convolutional network project, who did all the logistic regression experiments in chapter 2. Good discussions were made with Younduck Choi, Ross Goroshin, Mikael Henaff, Yoon Kim, Ilya Kostrikov, Michael Mathieu, Aditya Ramesh, Sainbayar Sukhbaatar, Will Whitney, Jiakai Zhang and many others throughout all of these projects.

A great deal of computational resources were provided by FAIR during 2017 summer, and subsequently Element AI, to speed up the projects in this dissertation. I want to gratefully acknowledge the support of NVIDIA Corporation with the donation of 2 Tesla K40 GPUs used for training some of the models in this dissertation. Amazon.com Inc also provided an AWS in Education Research grant,

which is used for crawling the datasets used in this dissertation.

Finally, I want to express love to my parents, who encouraged their only son on a long but rewarding journey abroad.

Abstract

This dissertation applies convolutional networks for learning representations of text, and it consists of several parts. The first part offers an empirical exploration on the use of character-level convolutional networks (ConvNets) for text classification. We constructed several large-scale datasets to show that character-level convolutional networks could achieve state-of-the-art or competitive results. Comparisons are offered against traditional models such as bag of words, n-grams and their TFIDF variants, and deep learning models such as word-based ConvNets and recurrent neural networks. These results indicate that using low-level inputs – in this case characters – for convolutional networks could be feasible for text representation learning. The second part concerns which text encoding method might work for convolutional networks. We include a comprehensive comparison of different encoding methods for the task of text classification using 14 large-scale datasets in 4 languages including Chinese, English, Japanese and Korean. Different encoding levels are studied, including UTF-8 bytes, characters, words, romanized characters and romanized words. For all encoding levels, whenever applicable, we provide comparisons with linear models, fastText and convolutional networks. For convolutional networks, we compare between encoding mechanisms using character glyph images, one-hot (or one-of-n) encoding, and embedding. From these 473 models, one of the conclusions is that byte-level one-hot encoding works consistently best for convolutional networks. Based on this, in the third part of the dissertation we develop a convolutional network at the level of bytes for learning representations through the task of auto-encoding. The proposed model is a multi-stage deep convolutional encoder-decoder framework using residual connections, containing up to 160 parameterized layers. Each encoder or decoder contains a shared group of

modules that consists of either pooling or up-sampling layers, making the network recursive in terms of abstraction levels in representation. The decoding process is non-sequential. Results for 6 large-scale paragraph datasets are reported, in 3 languages including Arabic, Chinese and English. Analyses are conducted to study several properties of the proposed model. Experiments are presented to verify that the auto-encoder can learn useful representations. In the fourth part of the dissertation, we use the improved design from the previous auto-encoding model to text classification, adding comparisons between residual and dense connections. This further validates the choice of the architecture we made for the auto-encoding model, and the effectiveness of the recursive architecture with residual or dense connections.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

To help the readers put this dissertation in perspective, the introduction chapter first gives a brief overview on text representation learning and convolutional networks, and then introduces the motivation for the particular methodology and model design used in this dissertation.

1.1 Background Overview for Text Representation Learning

This section offers an overview of representation learning for text. Throughout this dissertation, representation learning indicates a set of methods that learn some feature to be used in a machine learning system. For text, it is the process of transforming the raw and variable-length sequence of characters into vectors. Representation learning for text is a non-trivial problem and it faces many challenges even with the recent advancements in deep learning methods. We will discuss some of these challenges throughout this dissertation.

1.1.1 Simple Representation

As one of the most traditional research fields in computer science, natural language processing has evolved a long way in the representation for text. The traditional non-learning representation methods including the likes of one-hot vectors, bag-of-words representation and its TFIDF [Jones, 1972] variants, and n-grams representation which can take into consideration some of the information in word order. These methods are still quite popular, mainly because more advanced methods such as neural networks have limited success in doing better than them in some simple tasks such as text classification. That said, text classification could still be useful as a benchmark for studying different aspects of applying neural networks in NLP, because training and testing errors are relatively straightforward. This is compared to some other measurements which are usually surrogates because many NLP tasks have structured (e.g. sequential) outputs.

1.1.2 Distributed Representation

A slightly more advanced method for text representation is the idea of word embedding or word vector – associating each word with a real-valued vector. Compared to simple representation methods such as one-hot encoding and bag-of-words, these can be thought as "distributed" representation because there is no pre-defined meaning to each dimension of the real-valued vector. Depending on training methodology, these are mostly divided between pre-trained word vectors and word vectors jointly trained with the task at hand.

The earliest work on distributed representation for words can be traced back to [Miikkulainen and Dyer, 1991]. [Bengio et al., 2003] posed it as a method to mit-

igate the curse-of-dimensionality problem facing simple representation methods for sequential data such as causal language modeling. Later, [Collobert and Weston, 2008] and [Collobert et al., 2011b] proposed to learn a convolutional network with randomly initialized word vectors in a multi-task setting, with competitive results for part-of-speech (POS) tagging, chunking, named entity recognition (NER) and semantic role labeling.

In the following years, [Mikolov et al., 2013a] proposed to learn word vectors in an unsupervised fashion with the ideas of continuous bag-of-words (CBOW) and skip-grams with training tricks such as hierarchical softmax and negative sampling. It was followed by many papers showing its usefulness in various NLP tasks. The idea of global vectors for word representation (GloVe) [Pennington et al., 2014] was also proposed as an alternative, which is a form of matrix factorization on the co-occurrence matrix of the vocabulary. [Levy et al., 2015] shows that skip-grams with negative sampling can be thought of as a form of matrix factorization as well, making it possible to understand all these unsupervised word embedding learning methods in the same framework.

Beyond embedding vectors at word-level, it is also possible to embed text at paragraph or even document level. [Le and Mikolov, 2014] proposed to learn document vectors by associating a vector to each paragraph and document, and use it as an additional vector in the word embedding algorithms. It was shown to be useful for sentiment analysis and information retrieval tasks. Although relatively simple to implement, this method has the drawback that the inference process is an optimization algorithm that needs to be run at test time, which limits its scalability.

One way to mitigate the inference problem during test time is to replace the

association between sentences and vectors by a deep learning model such as recurrent neural networks or convolutional networks, which is used by the idea of skip-thought vectors [Kiros et al., 2015]. The core idea of skip-though vectors is to use the neighbouring sentence positions as an objective for sentence representation learning. This method requires an encoder-decoder architecture, where the encoder can be a convolutional or recurrent network. The original paper used a recurrent encoder. The encoder produces a vector that represents a sentence, while the decoder is used to predict the surrounding sentences. This simple approach worked well for learning representations that are useful for tasks such as semantic relatedness, paraphrase detection, image-sentence ranking, text classification and story generation. The idea of predicting surrounding sentences is similar to the idea of skip-gram, but the latter operates on words instead of sentences.

1.1.3 Recurrent Neural Networks

With the advancement of deep learning, neural network models have also become popular for NLP because of their ability to learn multi-level representation for text. Two major model variants are recurrent neural networks (RNN) and convolutional networks (ConvNet or CNN). This section discusses RNNs.

The idea of applying recurrent neural networks to NLP is quite straightforward because text is naturally sequential. In the previous sections we already introduced the neural language modeling work by [Bengio et al., 2003]. Besides proposing the idea of word embedding, it was also one of the first papers to propose using a neural network to predict the next word in text – that is, modeling sequences in a causal fashion. It could either be a time-delay neural network or a recurrent neural network. Similar ideas were also proposed by [Sutskever et al., 2011] in

which the focus was to use recurrent networks as a sequence generation model, and their experiments are conducted at a much larger scale. Since then, many authors have used these ideas for many different tasks taking advantage of their rich representational capacity, especially in a sequence-to-sequence (seq2seq) [Sutskever et al., 2014] or encoder-decoder [Cho et al., 2014b] framework.

There are many variants in the realm of RNNs, and two of the major ones are long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] and gated recurrent unit (GRU) [Cho et al., 2014b]. They are both proposed as alternatives to simple multi-layer recurrent networks to address the gradient vanishing problem [Bengio et al., 1994] [Hochreiter et al., 2001] faced by recurrent models during training. This is done by allowing the network to decide how much information to remember from the past state vector, and how much new information can be taken in with the current new input from the sequence. The difference between LSTM and GRU is that the former has 3 gates instead of 2 in GRU. The extra gate that allows an LSTM unit to mix the outputs from its state and from the current input. The work by [Chung et al., 2014] offers a focused study on the properties of GRU and compares it against other RNN variants including LSTM.

The research in RNNs have opened doors for handling text generation in a sequential fashion. However, despite the ideas of LSTM and GRU, in practice it is still difficult to generate long sequences accurately from only the recurrent structure. As a result, many authors have also proposed the idea of attention [Bahdanau et al., 2015] to help the generation process. The basic idea is to put a weight on some input signal that the model attends to, and this weight changes at each step of the generative process. This idea has been shown to work for tasks such as machine translation [Bahdanau et al., 2015], image captioning [Xu

et al., 2015], reading comprehension [Hermann et al., 2015], and even for internal distributed memory representations [Sukhbaatar et al., 2015]. In a nutshell, the idea of attention models can be applied to perhaps anything that attempts to generate some sequential data by observing some input signal, in which the focus of the model can change from step-to-step. Requiring some signal to attend to is both an advantage and a limitation of these attention models, since not all tasks have reasonable a input signal that spans a natural dimension such as time or space. One example is a free-style dialogue system in which reasoning from some internal common sense memory is required, which may or may not have a natural time or spacial dimension for applying the dynamic attention weights.

Another dimension of research for recurrent networks is to augment them by a memory component. For example, the paper by [Sukhbaatar et al., 2015] proposed a model named end-to-end memory networks that uses an attention mechanism on a pre-defined set of embedding vectors as the memory, following the framework proposed earlier by [Weston et al., 2014]. Meanwhile, [Graves et al., 2014] proposed the neural Turing machine of which a controller network can be trained to utilize an external memory matrix. Later, [Gulcehre et al., 2016] extended the neural Turing machine to include a trainable addressing mechanism. Around the same time, the paper by [Joulin and Mikolov, 2015] also proposes a stack-like memory mechanism that can learn various simple algorithms such as counting, memorization and binary addition. Text representation learning becomes a pre-requisite problem for these models, since they concern more about reasoning rather than turning variable-length text into vectors. The next section introduces the difference between text representation and reasoning as defined in this dissertation..

These results being presented, a recent piece of work by [Vaswani et al., 2017]

shows that using only attention without the recurrent network component can do as well for machine translation, with the text generation processing still being sequential – that is, generating one word or symbol at a time in the reading order of text. This puts the necessity of RNNs in question for sequential text generation when there is some signal that the model can attend to at different generation steps. Beyond sequential text generation, [Gu et al., 2018] also proposed a model that can do non-autoregressive machine translation. All of these also opened a door for another possible dimension for futher research – that is, whether doing away with RNN or sequential process for text generation is possible.

1.1.4 Convolutional Networks

Originated from the field of computer vision, convolutional networks (ConvNets or CNN) [LeCun et al., 1989] [LeCun et al., 1990] [LeCun et al., 1998] have become a standard toolkit for image recognition, localization and detection, especially since the ImageNet 2012 [Russakovsky et al., 2015] breakthrough achieved by [Krizhevsky et al., 2012]. A ConvNet usually consists of a set of convolutional layers in which the parameters are the filters, and perhaps some linear layers in later stages, with non-linearities in between these parameterized layers. Pooling layers can also be added to aggregate information from nearby values to build representations at different levels of abstraction.

One motivation for ConvNets was the idea that when there is a natural dimension (such as width and height in image data, or the time dimension in sequential data), local groups of these values are often highly correlated [LeCun et al., 2015]. Since such correlation are often repeated across these natural dimensions, convolution is a natural process for detecting these local patterns. Using multi-stage

convolutional layers, non-linearity layers and pooling layers, ConvNets are capable of learning hierarchical features at different levels of abstraction. For NLP, text can be thought of as a sequence of symbols in which there is a natural dimension in terms of the flow of reading, therefore the 1-D variant of ConvNets can be applied.

1-D ConvNets are dated as old as the 2-D version used for computer vision. They appear as "time-delay neural networks" for speech recognition [Waibel et al., 1989] and document reading [LeCun et al., 1998], in which the convolutional kernels are usually constrained to be "causal filters", a terminology used in the field of signal processing.

Besides using the idea of word embedding, the work by [Collobert and Weston, 2008] and [Collobert et al., 2011b] was also the first to use convolutional networks for NLP in general. The ConvNet architecture there starts with a look-up table that turns indices of words into a sequence of vectors, and then a few convolutional layers are appended to learn features. Max-over-time layers are appended to aggregate abstract features, and then some optional fully-connected layers can be appended to give the model more capacity for feature transformation. Finally, a few soft-max layers are appended for classification at word-level. This generic architecture can be applied to many different sequence labeling tasks in NLP, such as part-of-speech (POS) tagging, chunking, named entity recognition (NER) and semantic role labeling.

Since then, there have been a few papers attempting to use ConvNets for NLP. The paper by [Kim, 2014] proposed to do text classification using a ConvNet, and [Zhang et al., 2015] extended it to character level. Character-aware language models were also proposed with ConvNets [Kim et al., 2016]. Meanwhile, the idea of using ConvNets to generate texts has also been tried, such as the convolutional

seq2seq model proposed by [Gehring et al., 2017], which is shown to work as well as the seq2seq or encoder-decoder structures using recurrent networks for machine translation with attention.

Originally proposed for speech and audio synthesis, the idea of WaveNet [Oord et al., 2016] is a way of using a 1-D ConvNets to do causal signal synthesis. One important trick for training WaveNet is the re-use of past generated intermediate representations to be fed into the next layer. It is also able to construct representations at different abstraction levels, with the use of dilated convolution (convolution with holes) to mimic pooling layers for dense representation that are equivariant to the input in size. Later, [Kalchbrenner et al., 2016] applied the WaveNet architecture to character-level language modeling and machine translation, and achieved competitive results for machine translation. A later paper from the same group [van den Oord et al., 2017] also shows that conditional audio and speech synthesis without feeding the output signals back to the model is possible, in which the generated signals have a causal dependence on the noise variables. This can be thought of as an implicit form of sequential modeling.

1.1.5 Learning from Character and Byte Levels

Almost all of the representation learning techniques for text assume that the basic constructs of languages are words. For many non-English languages this could be undesirable, either because they have a rich morphology – such as German and Hungarian, or because there is no unambiguous word boundary in the flow of text – such as some texts in Chinese. Therefore, another dimension of representation learning for text is the exploration of methods that are free from the word assumption by going lower-level to characters or bytes. These models are

shown to be more applicable across languages while requiring less pre-processing.

One of the main approaches that has been explored so far is to augment word-level models with lower-level information. The papers by [Luong et al., 2013] and [Botha and Blunsom, 2014] propose to augment recurrent networks by morphemes for better word representation, which produced better results for word similarity and sequential language modeling tasks. Augmenting character-level information for word-level models has been explored by [dos Santos and Zadrozny, 2014], [dos Santos and Gatti, 2014], [dos Santos et al., 2015] and [Shen et al., 2014], which was shown to be useful for part-of-speech (POS) tagging, sentiment classification, named entity recognition (NER) and information retrieval, respectively.

The idea of pure character-level language modeling was explored by [Sutskever et al., 2011] with recurrent networks. The idea of only using character-level information for text classification using convolutional networks was proposed by [Zhang et al., 2015]. At the same time, [Kim et al., 2016] proposed to do language modeling using only character-level inputs also with a convolutional network, but their output is still at the level of words. A large-scale benchmark of language modeling techniques done by [Józefowicz et al., 2016] shows that character-level convolutional network achieved competitive perplexity, especially when scaling with data and computation. Purely character-level machine translation has also been explored by [Lee et al., 2017].

For some languages, the number of characters in its alphabet could be huge, such as the collection of Chinese, Japanese and Korean. In this case, simple one-hot encoding has the same curse-of-dimensionality problem at word-level. Besides the idea of using character-level embedding, another approach is to go even lower for the input – to the level of bytes. This has been explored by [Gillick et al., 2016],

where they apply an LSTM-based [Hochreiter and Schmidhuber, 1997] sequence-to-sequence [Cho et al., 2014b] [Sutskever et al., 2014] model at byte-level for a variety of tasks including part-of-speech tagging and named entity recognition, for four languages including English, German, Spanish and Dutch. The paper by [Zhang and LeCun, 2017] also includes benchmarks comparing byte-level convolutional networks against many other encoding mechanisms for text classification.

It is worth noting that besides character or byte level information, the pictorial representation of text – the glyphs of characters – could also be useful, especially for topologically rich languages such as Chinese, Japanese and Korean (CJK). Recent research has shown that glyphs of CJK characters can help to improve the results of various tasks including text classification [Shimada et al., 2016] [Liu et al., 2017] and translation [Costa-jussà et al., 2017]. The paper by [Zhang and LeCun, 2017] also includes a comparison of glyph encoding against other encoding mechanisms for text classification.

1.2 Motivation

This section offers an overview of the author's approach to research in applying deep learning for natural language processing (NLP), and presents motivations for the particular problems and model design choices made in this dissertation.

1.2.1 End-to-End Learning for Text

One of the promises of deep learning is that it can solve problems in an end-to-end fashion without hand-crafted feature extractors. Note that all deep learning models are neural networks, and neural networks are vector processors which take

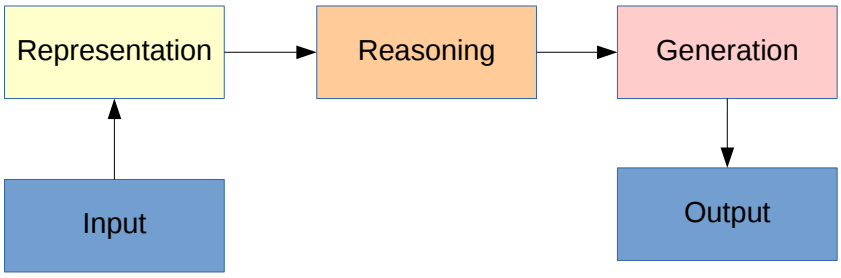


Figure 1.1: A framework for research in deep learning for NLP

vectors in and bring vectors out. Therefore, the gap between text and end-to-end deep learning models is the variable-length nature of text and the fixed-length nature of vectors. As a result, the trichotomy in figure 1.1 naturally follows.

- Representation: the study of models that can transform a piece of free text into a numerical vector to be used for reasoning and generation.
- Reasoning: the study of methods that can infer from input representation what kind of output is desired for the task at hand.
- Generation: the study of models that can transform a numerical vector obtained from reasoning to free-text.

It is necessary to provide good modules for all the 3 parts in order to improve the applicability and performance of deep learning models for NLP. That said, this dissertation is focused on only the first module of the trichotomy – representation, which turns out to be non-trivial for various reasons, such as the difficulty of learning long-term dependencies in neural networks and the inconsistent assumptions of applying deep learning to different languages.

Note that this trichotomy is only one possible framework to put in perspective the research in deep learning for NLP, and we put it here only to assist the understanding of the motivations of this dissertation. The next section takes a closer

look at the one component studied in this dissertation – representation learning for text.

1.2.2 Motivation for Text Representation Learning

Using deep learning models effectively for the representation learning of text is of great interests for the natural language processing (NLP) community. Among the available models that are applicable to text, recurrent networks and convolutional networks are 2 of the major choices. In this dissertation, we focus our effort in convolutional networks, and show that it can learn useful representations for text classification and auto-encoding.

The author's goals of research in applying deep learning for natural language processing consist of the following, and they are also the reasons for the model design choices made in this dissertation:

- Generality across languages: given sufficient data, the model can be applied to all different languages in the same fashion.
- Scalability: the performance of proposed model can scale with the improvement of data size and computation.

Generality of model applicability across languages has become an interest of natural language processing (NLP) community recently, partly represented by the recent use of deep learning models in learning representation from inputs at a lower-level than words such as characters. This expands the generality and applicability of models across languages compared to word-level models, due to the fact that not all languages have clear word boundaries, such as some texts in Chinese and Japanese.

One of the issues in scalability is the difficulty of learning long-term dependency in sequences because of gradient (or output) vanishing (or exploding) [Bengio et al., 1994] [Hochreiter et al., 2001] in the current sequential models such as recurrent neural networks. We refer to sequential models as those that read or generate symbols or words according to the text reading order, and they do not have to be recurrent networks. There are a few approaches to mitigate this, such as the attention models used in machine translation and other tasks [Bahdanau et al., 2015], and the development of hierarchical recurrent networks [Schmidhuber, 1992] [El Hihi and Bengio, 1996] [Koutnik et al., 2014] [Chung et al., 2016]. This dissertation focuses on an alternative, which is to use convolutional networks in a non-sequential fashion – that is, generating all output symbols at once. Being recursive means that the a shared module is used to build up different levels of abstraction repeatedly, each time reducing the length of representation by a multiplicative factor. Therefore, the depth of the network will be on a logarithm order of the sequence length. It could potentially better mitigate the gradient (or output) vanishing (or exploding) problem facing non-hierarchical recurrent networks, since the latter has a depth on the linear order of the sequence length.

The next section details the content of this dissertation based on these 2 goals.

1.3 Dissertation Content

The first part of the dissertation is a study on using character-level convolutional networks for text classification. We constructed several large-scale datasets to show that character-level convolutional networks could achieve state-of-the-art or competitive results. Comparisons are offered against traditional models such as

bag of words, n-grams and their TFIDF variants, and deep learning models such as word-based ConvNets and recurrent neural networks. This shows the feasibility of using low-level inputs – in this case characters – with convolutional networks using text classification as the task.

To further the research on the generality across languages with convolutional networks, we follow with a comprehensive comparison on different encoding levels and mechanisms for text classification in Chinese, English, Japanese and Korean. Different encoding levels are studied, including UTF-8 bytes, characters, words, romanized characters and romanized words. For all encoding levels, whenever applicable, we provide comparisons with linear models, fastText and convolutional networks. For convolutional networks, we compare between encoding mechanisms using character glyph images, one-hot (or one-of-n) encoding, and embedding. There are many conclusions and analysis from these 473 models, and one of them is that byte-level one-hot encoding works consistently best for convolutional networks.

Based on these results, we use byte-level one-hot encoding for the next part of the dissertation. Besides introducing the recursive convolutional network design, we also use it as an auto-encoder for learning representations of text. The proposed model is a multi-stage deep convolutional encoder-decoder framework using residual connections, containing up to 160 parameterized layers. Each encoder or decoder contains a shared group of modules that consists of either pooling or up-sampling layers, making the network recursive in terms of abstraction levels in representation. The decoding process is non-sequential. Results for 6 large-scale paragraph datasets are reported, in 3 languages including Arabic, Chinese and English. Analyses are conducted to study several properties of the proposed model.

We present results on using the representation for both text classification in which improvements were observed.

Finally, we explore the use of residual and dense connections for recursive convolutional network in text classification without pre-training, which shows that byte-level recursive convolutional networks can obtain better results than previous models. This further validates the model design, paving the way for future research in other tasks using recursive convolutional networks for text representation, reasoning and generation.

The models in this dissertation were designed at different times throughout the past 5 years, and they are heavily influenced by the progress of convolutional network design in computer vision. The models in chapter 2 are designed with reference to the OverFeat paper by [Sermanet et al., 2014], which won the localization task for ImageNet 2013 competition. The models in chapter 3 were designed with similarity to the VGG network [Simonyan and Zisserman, 2014], which the the first and second places of the localization and classification tasks respectively, the ImageNet 2014. One of the key component in the auto-encoding models in chapter 4 – residual connections – is inspired by the residual network paper [He et al., 2016], which won the ImageNet 2015 classification task. In addition to residual connections, chapter 5 also includes models designed using dense connections [Huang et al., 2016].

The basic conclusion from this dissertation is that representation learning using byte-level recursive convolutional network can achieve both generality and scalability when applied to different languages and datasets. That said, representation consists of many other different tasks as well, and we have yet to show the effectiveness of byte-level recursive convolutional networks in text reasoning and

generation. These are both valid research problems to follow, and the final chapter on conclusions and outlook will give some discussion.

Chapter 2

Character-level Convolutional Networks for Text Classification

This chapter offers an empirical exploration on the use of character-level convolutional networks (ConvNets) for text classification. We constructed several large-scale datasets to show that character-level convolutional networks could achieve state-of-the-art or competitive results. Comparisons are offered against traditional models such as bag of words, n-grams and their TFIDF variants, and deep learning models such as word-based ConvNets and recurrent neural networks.

2.1 Introduction

Text classification is a classic topic for natural language processing, in which one needs to assign predefined categories to free-text documents. The range of text classification research goes from designing the best features to choosing the best possible machine learning classifiers. To date, almost all techniques of text

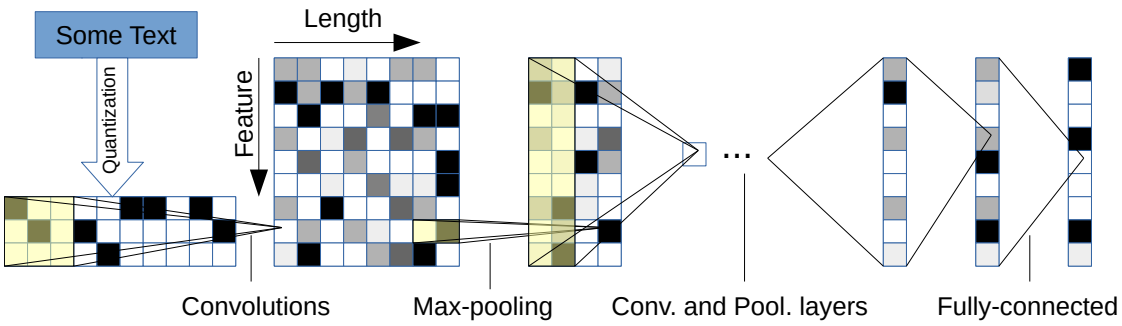


Figure 2.1: Illustration of our model

classification are based on words, in which simple statistics of some ordered word combinations (such as n-grams) usually perform the best [Joachims, 1998].

On the other hand, many researchers have found convolutional networks (ConvNets) [LeCun et al., 1989] [LeCun et al., 1998] are useful in extracting information from raw signals, ranging from computer vision applications to speech recognition and others. In particular, time-delay networks used in the early days of deep learning research are essentially convolutional networks that model sequential data [Bottou et al., 1989] [Waibel et al., 1989].

In this chapter we explore treating text as a kind of raw signal at character level, and applying temporal (one-dimensional) ConvNets to it. For this chapter we only used a classification task as a way to exemplify ConvNets' ability to understand texts. Historically we know that ConvNets usually require large-scale datasets to work, therefore we also build several of them. An extensive set of comparisons is offered with traditional models and other deep learning models.

Applying convolutional networks to text classification or natural language processing at large was explored in literature. It has been shown that ConvNets can be directly applied to distributed [dos Santos and Gatti, 2014] [Kim, 2014] or discrete [Johnson and Zhang, 2014] embedding of words, without any knowledge on

Table 2.1: Convolutional layers used in our experiments. The convolutional layers have stride 1 and pooling layers are all non-overlapping ones, so we omit the description of their strides.

Layer	Large Feature	Small Feature	Kernel	Pool
1	1024	256	7	3
2	1024	256	7	3
3	1024	256	3	N/A
4	1024	256	3	N/A
5	1024	256	3	N/A
6	1024	256	3	3

the syntactic or semantic structures of a language. These approaches have been proven to be competitive to traditional models.

There are also related works that use character-level features for language processing. These include using character-level n-grams with linear classifiers [Kanaris et al., 2007], and incorporating character-level features to ConvNets [dos Santos and Zadrozny, 2014] [Shen et al., 2014]. In particular, these ConvNet approaches use words as a basis, in which character-level features extracted at word [dos Santos and Zadrozny, 2014] or word n-gram [Shen et al., 2014] level form a distributed representation. Improvements for part-of-speech tagging and information retrieval were observed.

This chapter is the first to apply ConvNets only on characters. We show that when trained on large-scale datasets, deep ConvNets do not require the knowledge of words, in addition to the conclusion from previous research that ConvNets do not require the knowledge about the syntactic or semantic structure of a language. This simplification of engineering could be crucial for a single system that can work for different languages, since characters always constitute a necessary construct regardless of whether segmentation into words is possible. Working on only

Table 2.2: Fully-connected layers used in our experiments. The number of output units for the last layer is determined by the problem. For example, for a 10-class classification problem it will be 10.

Layer	Output Units Large	Output Units Small
7	2048	1024
8	2048	1024
9	Depends on the problem	

characters also has the advantage that abnormal character combinations such as misspellings and emoticons may be naturally learnt.

2.2 Character-level Convolutional Networks

In this section, we introduce the design of character-level ConvNets for text classification. The design is modular, where the gradients are obtained by back-propagation [Rumelhart et al., 1986] to perform optimization.

2.2.1 Key Modules

The main component is the temporal convolutional module, which simply computes a 1-D convolution. Suppose we have a discrete input function $g(x) \in [1, l] \to \mathbb{R}$ and a discrete kernel function $f(x) \in [1, k] \to \mathbb{R}$. The convolution $h(y) \in [1, \lfloor (l-k)/d \rfloor + 1] \to \mathbb{R}$ between $f(x)$ and $g(x)$ with stride d is defined as

$$h(y) = \sum_{x=1}^{k} f(x) \cdot g(y \cdot d - x + c),$$

where $c = k - d + 1$ is an offset constant. Just as in traditional convolutional networks in vision, the module is parameterized by a set of such kernel functions

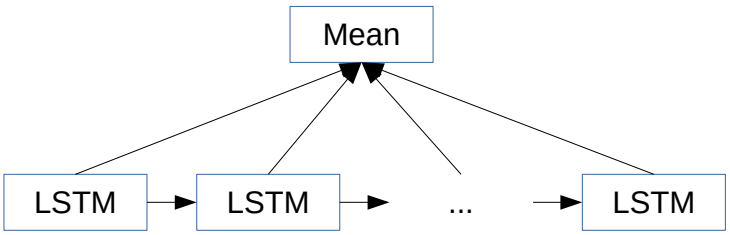


Figure 2.2: long-short term memory

$f_{ij}(x)$ ($i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$) which we call *weights*, on a set of inputs $g_i(x)$ and outputs $h_j(y)$. We call each g_i (or h_j) input (or output) *features*, and m (or n) input (or output) feature size. The outputs $h_j(y)$ is obtained by a sum over i of the convolutions between $g_i(x)$ and $f_{ij}(x)$.

One key module that helped us to train deeper models is temporal max-pooling. It is the 1-D version of the max-pooling module used in computer vision [Boureau et al., 2010a]. Given a discrete input function $g(x) \in [1, l] \to \mathbb{R}$, the max-pooling function $h(y) \in [1, \lfloor (l-k)/d \rfloor + 1] \to \mathbb{R}$ of $g(x)$ is defined as

$$h(y) = \max_{x=1}^{k} g(y \cdot d - x + c),$$

where $c = k - d + 1$ is an offset constant. This very pooling module enabled us to train ConvNets deeper than 6 layers, where all others fail. The analysis by [Boureau et al., 2010b] might shed some light on this.

The non-linearity used in our model is the rectifier or thresholding function $h(x) = \max\{0, x\}$, which makes our convolutional layers similar to rectified linear units (ReLUs) [Nair and Hinton, 2010]. The algorithm used is stochastic gradient descent (SGD) with a minibatch of size 128, using momentum [Polyak, 1964] [Sutskever et al., 2013] 0.9 and initial step size 0.01 which is halved every 3

Table 2.3: Statistics of our large-scale datasets. Epoch size is the number of minibatches in one epoch

Dataset	Classes	Train Samples	Test Samples	Epoch Size
AG's News	4	120,000	7,600	5,000
Sogou News	5	450,000	60,000	5,000
DBPedia	14	560,000	70,000	5,000
Yelp Review Polarity	2	560,000	38,000	5,000
Yelp Review Full	5	650,000	50,000	5,000
Yahoo! Answers	10	1,400,000	60,000	10,000
Amazon Review Full	5	3,000,000	650,000	30,000
Amazon Review Polarity	2	3,600,000	400,000	30,000

epoches for 10 times. Each epoch takes a fixed number of random training samples uniformly sampled across classes. This number will later be detailed for each dataset separately. The implementation is done using Torch 7 [Collobert et al., 2011a].

2.2.2 Character quantization

Our models accept a sequence of encoded characters as input. The encoding is done by prescribing an alphabet of size m for the input language, and then quantize each character using 1-of-m encoding (or "one-hot" encoding). Then, the sequence of characters is transformed to a sequence of such m sized vectors with fixed length l_0. Any character exceeding length l_0 is ignored, and any characters that are not in the alphabet including blank characters are quantized as all-zero vectors. The character quantization order is backward so that the latest reading on characters is always placed near the begin of the output, making it easy for fully connected layers to associate weights with the latest reading.

The alphabet used in all of our models consists of 70 characters, including 26

English letters, 10 digits, 33 other characters and the new line character. The non-space characters are:

```
abcdefghijklmnopqrstuvwxyz0123456789

-,;.!?:’’’/\|_@#$%ˆ&*˜‘+=<>()[]{}
```

Later we also compare with models that use a different alphabet in which we distinguish between upper-case and lower-case letters.

2.2.3 Model Design

We designed 2 ConvNets – one large and one small. They are both 9 layers deep with 6 convolutional layers and 3 fully-connected layers. Figure 2.1 gives an illustration.

The input have number of features equal to 70 due to our character quantization method, and the input feature length is 1014. It seems that 1014 characters could already capture most of the texts of interest. We also insert 2 dropout [Hinton et al., 2012] modules in between the 3 fully-connected layers to regularize. They have dropout probability of 0.5. Table 2.1 lists the configurations for convolutional layers, and table 2.2 lists the configurations for fully-connected (linear) layers.

We initialize the weights using a Gaussian distribution. The mean and standard deviation used for initializing the large model is $(0, 0.02)$ and small model $(0, 0.05)$.

For different problems the input lengths may be different (for example in our case $l_0 = 1014$), and so are the frame lengths. From our model design, it is easy to know that given input length l_0, the output frame length after the last convolutional layer (but before any of the fully-connected layers) is $l_6 = (l_0 - 96)/27$. This number multiplied with the frame size at layer 6 will give the input dimension the first fully-connected layer accepts.

Table 2.4: Testing errors of all the models. Numbers are in percentage. "Lg" stands for "large" and "Sm" stands for "small". "w2v" is an abbreviation for "word2vec", and "Lk" for "lookup table". "Th" stands for thesaurus. ConvNets labeled "Full" are those that distinguish between lower and upper letters

Model	AG	Sog.	DBP.	Y.P.	Y.F.	Yah.	A.F.	A.P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-of-means	**16.91**	**10.79**	**9.55**	**12.67**	**47.46**	**39.45**	**55.87**	**18.39**
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg w2v Conv	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
Sm w2v Conv	11.35	4.54	1.71	5.56	42.13	31.50	42.59	6.00
Lg w2v Conv Th	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
Sm w2v Conv Th	10.88	-	1.53	5.36	41.09	29.86	42.50	5.63
Lg Lk Conv	8.55	4.95	1.72	4.89	40.52	29.06	45.95	5.84
Sm Lk Conv	10.87	4.93	1.85	5.54	41.41	30.02	43.66	5.85
Lg Lk Conv Th	8.93	-	1.58	5.03	40.52	28.84	42.39	5.52
Sm Lk. Conv Th	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg Full Conv	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm Full Conv	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg Full Conv Th	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm Full Conv Th	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg Conv	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm Conv	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg Conv Th	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm Conv Th	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

2.2.4 Data Augmentation using Thesaurus

Many researchers have found that appropriate data augmentation techniques are useful for controlling generalization error for deep learning models. These techniques usually work well when we could find appropriate invariance properties that the model should possess. In terms of texts, it is not reasonable to augment the data using signal transformations as done in image or speech recognition, because the exact order of characters may form rigorous syntactic and semantic meaning.

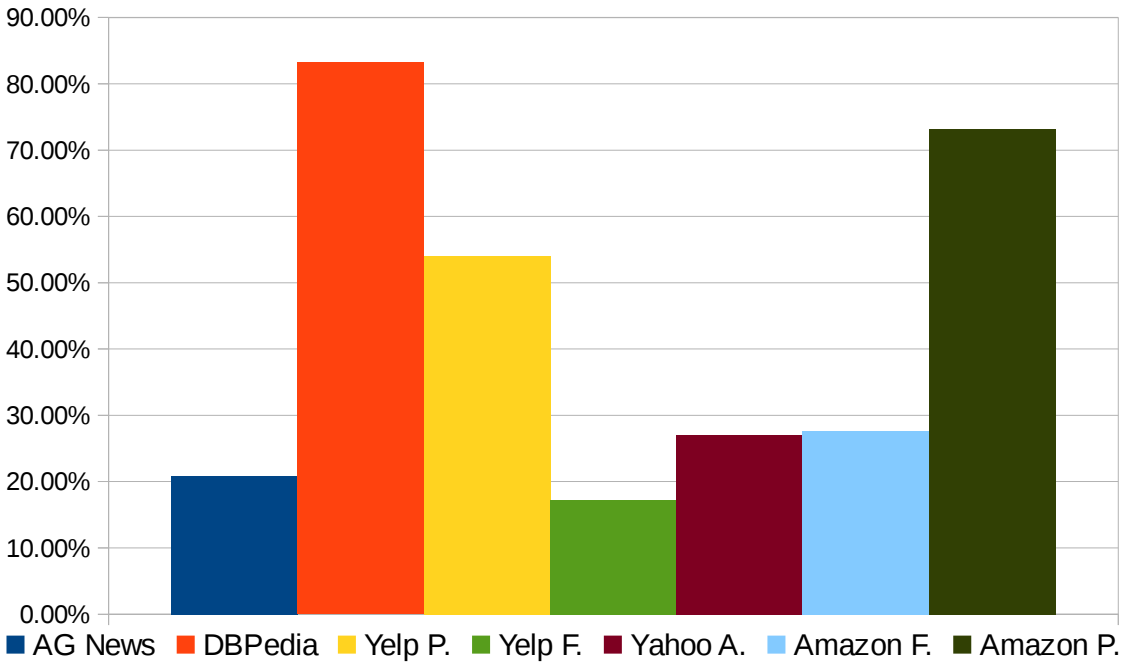


Figure 2.3: Relative errors with bag-of-means

Therefore, the best way to do data augmentation would have been using human rephrases of sentences, but this is unrealistic and expensive due the large volume of samples in our datasets. As a result, the most natural choice in data augmentation for us is to replace words or phrases with their synonyms.

We experimented data augmentation by using an English thesaurus, which is obtained from the `mytheas` component used in LibreOffice[1] project. That thesaurus in turn was obtained from WordNet [Fellbaum, 2005], where every synonym to a word or phrase is ranked by the semantic closeness to the most frequently seen meaning. To decide on how many words to replace, we extract all replaceable words from the given text and randomly choose r of them to be replaced. The probability of number r is determined by a geometric distribution with parameter p in which $P[r] \sim p^r$. The index s of the synonym chosen given a word is also determined

[1] `http://www.libreoffice.org/`

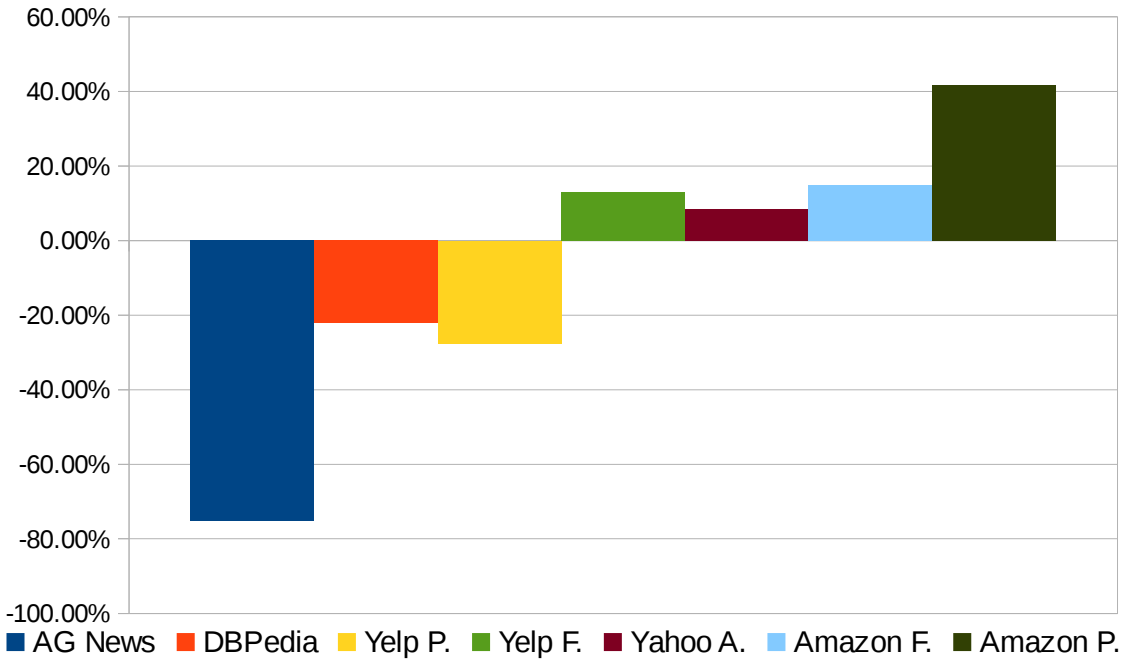


Figure 2.4: Relative errors with n-grams TFIDF

by a another geometric distribution in which $P[s] \sim q^s$. This way, the probability of a synonym chosen becomes smaller when it moves distant from the most frequently seen meaning. We will report the results using this new data augmentation technique with $p = 0.5$ and $q = 0.5$.

2.3 Comparison Models

To offer fair comparisons to competitive models, we conducted a series of experiments with both traditional and deep learning methods. We tried our best to choose models that can provide comparable and competitive results, and the results are reported faithfully without any model selection.

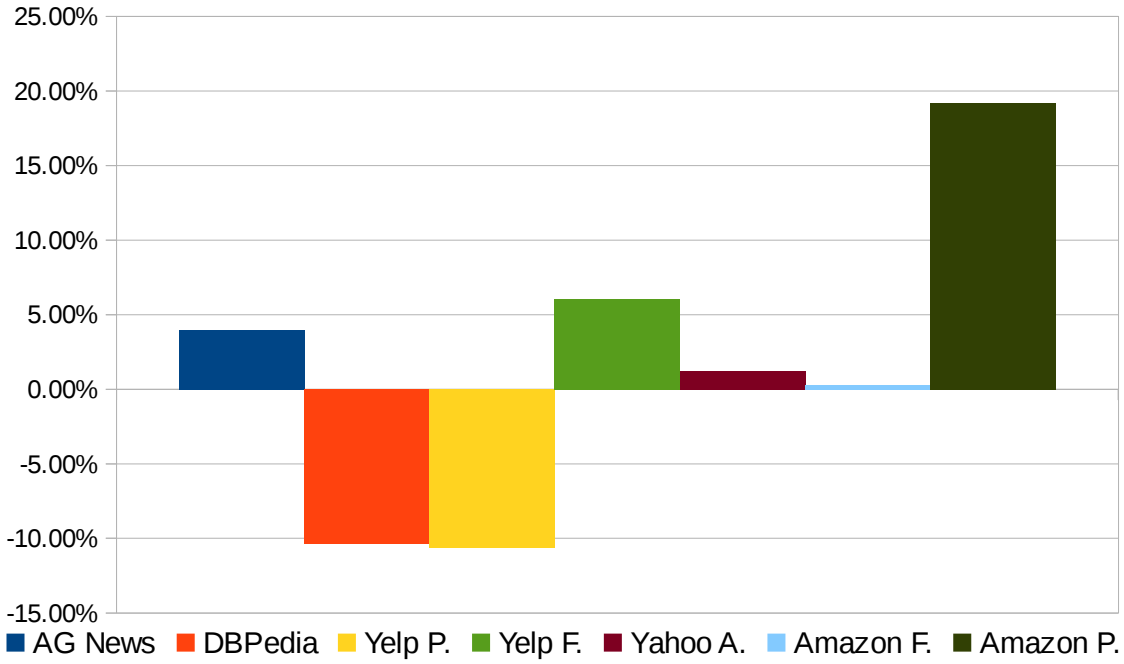


Figure 2.5: Relative errors with LSTM

2.3.1 Traditional Methods

We refer to traditional methods as those that using a hand-crafted feature extractor and a linear classifier. The classifier used is a multinomial logistic regression in all these models.

Bag-of-words and its TFIDF. For each dataset, the bag-of-words model is constructed by selecting 50,000 most frequent words from the training subset. For the normal bag-of-words, we use the counts of each word as the features. For the TFIDF (term-frequency inverse-document-frequency) [Jones, 1972] version, we use the counts as the term-frequency. The inverse document frequency is the logarithm of the division between total number of samples and number of samples with the word in the training subset. The features are normalized by dividing the largest feature value.

Bag-of-ngrams and its TFIDF. The bag-of-ngrams models are constructed

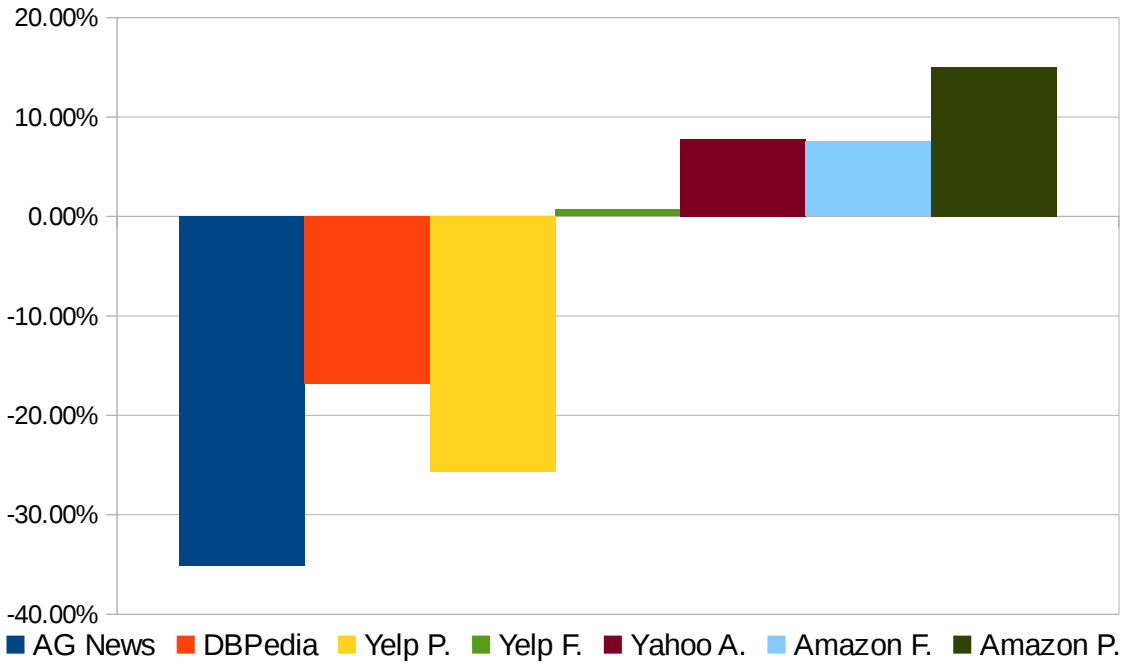


Figure 2.6: Relative errors with word2vec ConvNet

by selecting the 500,000 most frequent n-grams (up to 5-grams) from the training subset for each dataset. The feature values are computed the same way as in the bag-of-words model.

Bag-of-means on word embedding. We also have an experimental model that uses k-means on word2vec [Mikolov et al., 2013b] learnt from the training subset of each dataset, and then use these learnt means as representatives of the clustered words. We take into consideration all the words that appeared more than 5 times in the training subset. The dimension of the embedding is 300. The bag-of-means features are computed the same way as in the bag-of-words model. The number of means is 5000.

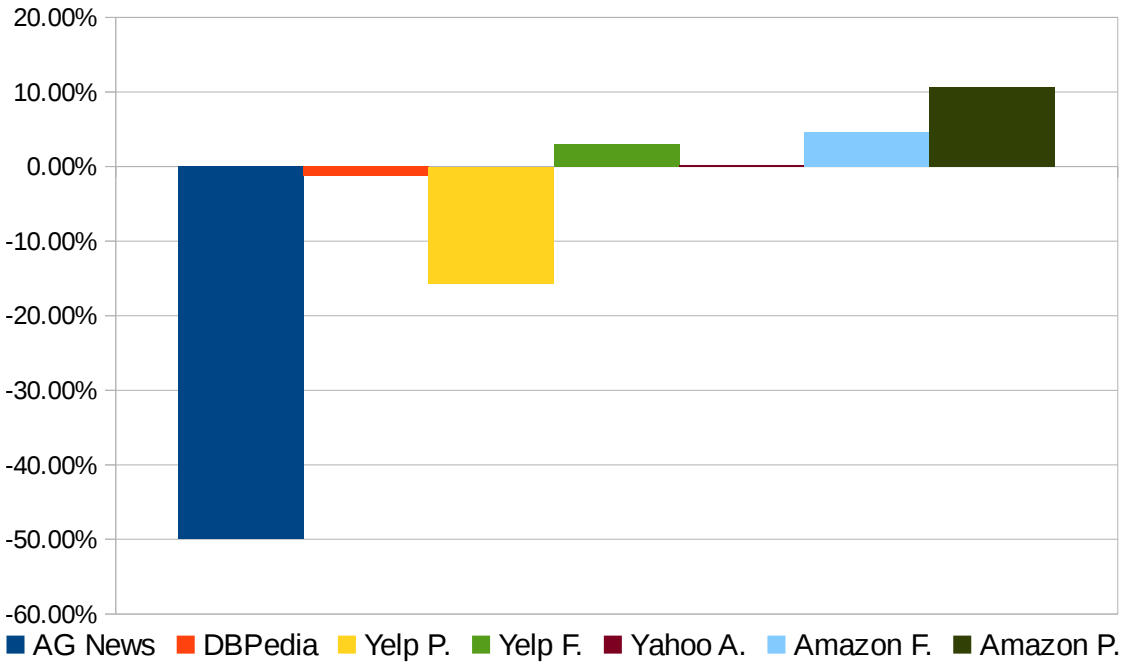


Figure 2.7: Relative errors with lookup table ConvNet

2.3.2 Deep Learning Methods

Recently deep learning methods have started to be applied to text classification. We choose two simple and representative models for comparison, in which one is word-based ConvNet and the other a simple long-short term memory (LSTM) [Hochreiter and Schmidhuber, 1997] recurrent neural network model.

Word-based ConvNets. Among the large number of recent works on word-based ConvNets for text classification, one of the differences is the choice of using pretrained or end-to-end learned word representations. We offer comparisons with both using the pre-trained word2vec [Mikolov et al., 2013b] embedding [Kim, 2014] and using lookup tables [Collobert et al., 2011b]. The embedding size is 300 in both cases, in the same way as our bag-of-means model. To ensure fair comparison, the models for each case are of the same size as our character-level ConvNets, in terms of both the number of layers and each layer's output size. Experiments using

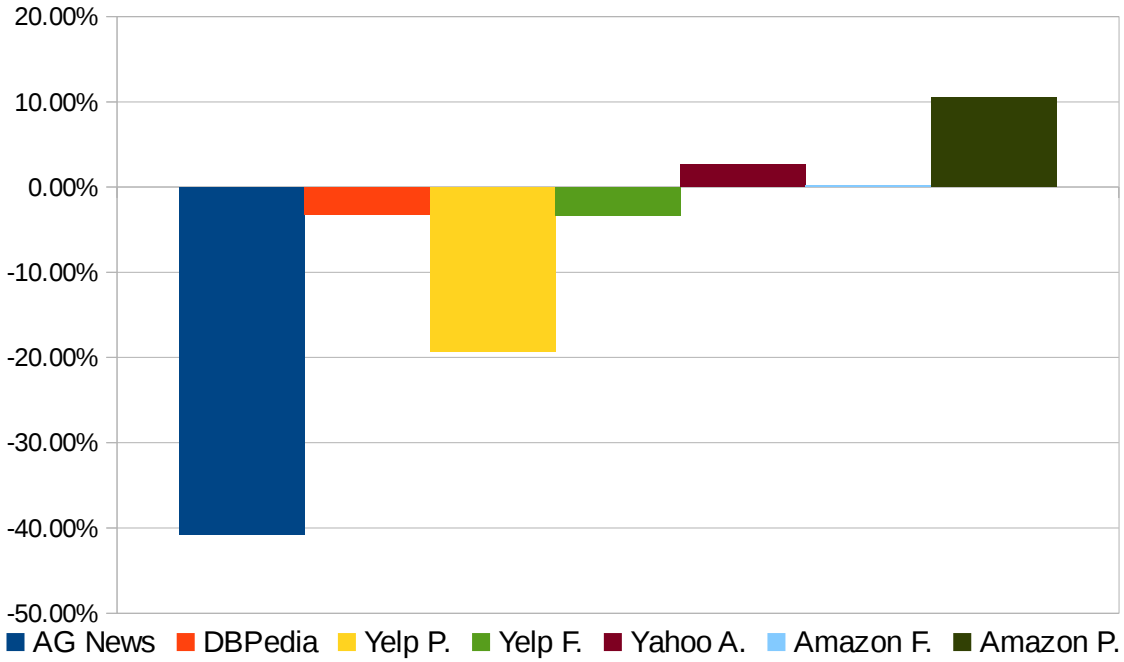


Figure 2.8: Relative errors with full alphabet ConvNet

a thesaurus for data augmentation are also conducted.

Long-short term memory. We also offer a comparison with a recurrent neural network model, namely long-short term memory (LSTM) [Hochreiter and Schmidhuber, 1997]. The LSTM model used in our case is word-based, using pre-trained word2vec embedding of size 300 as in previous models. The model is formed by taking mean of the outputs of all LSTM cells to form a feature vector, and then using multinomial logistic regression on this feature vector. The output dimension is 512. The variant of LSTM we used is the common "vanilla" architecture [Graves and Schmidhuber, 2005] [Greff et al., 2015]. We also used gradient clipping [Pascanu et al., 2013] in which the gradient norm is limited to 5. Figure 2.2 gives an illustration.

2.3.3 Choice of Alphabet

For the alphabet of English, one apparent choice is whether to distinguish between upper-case and lower-case letters. We report experiments on this choice and observed that it usually (but not always) gives worse results when such distinction is made. One possible explanation might be that semantics do not change with different letter cases, therefore there is a benefit of regularization.

2.4 Large-scale Datasets and Results

Previous research on ConvNets in different areas has shown that they usually work well with large-scale datasets, especially when the model takes in low-level raw features like characters in our case. However, most open datasets for text classification are quite small, and large-scale datasets are split with a significantly smaller training set than testing [Lewis et al., 2004]. Therefore, instead of confusing our community more by using them, we built several large-scale datasets for our experiments, ranging from hundreds of thousands to several millions of samples. Table 2.3 is a summary.

AG's news corpus. We obtained the AG's corpus of news article on the web[2]. It contains 496,835 categorized news articles from more than 2000 news sources. We choose the 4 largest classes from this corpus to construct our dataset, using only the title and description fields. The number of training samples for each class is 30,000 and testing 1900.

Sogou news corpus. This dataset is a combination of the SogouCA and SogouCS news corpora [Wang et al., 2008], containing in total 2,909,551 news articles

[2] `http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html`

in various topic channels. We then labeled each piece of news using its URL, by manually classifying the their domain names. This gives us a large corpus of news articles labeled with their categories. There are a large number categories but most of them contain only few articles. We choose 5 categories – "sports", "finance", "entertainment", "automobile" and "technology". The number of training samples selected for each class is 90,000 and testing 12,000. Although this is a dataset in Chinese, we used `pypinyin` package combined with `jieba` Chinese segmentation system to produce Pinyin – a phonetic romanization of Chinese. The models for English can then be applied to this dataset without change. The fields used are title and content.

DBPedia ontology dataset. DBpedia is a crowd-sourced community effort to extract structured information from Wikipedia [Lehmann et al., 2014]. The DBpedia ontology dataset is constructed by picking 14 non-overlapping classes from DBpedia 2014. From each of these 14 ontology classes, we randomly choose 40,000 training samples and 5,000 testing samples. The fields we used for this dataset contain title and abstract of each Wikipedia article.

Yelp reviews. The Yelp reviews dataset is obtained from the Yelp Dataset Challenge in 2015. This dataset contains 1,569,264 samples that have review texts. Two classification tasks are constructed from this dataset – one predicting full number of stars the user has given, and the other predicting a polarity label by considering stars 1 and 2 negative, and 3 and 4 positive. The full dataset has 130,000 training samples and 10,000 testing samples in each star, and the polarity dataset has 280,000 training samples and 19,000 test samples in each polarity.

Yahoo! Answers dataset. We obtained Yahoo! Answers Comprehensive Questions and Answers version 1.0 dataset through the Yahoo! Webscope program.

The corpus contains 4,483,032 questions and their answers. We constructed a topic classification dataset from this corpus using 10 largest main categories. Each class contains 140,000 training samples and 5,000 testing samples. The fields we used include question title, question content and best answer.

Amazon reviews. We obtained an Amazon review dataset from the Stanford Network Analysis Project (SNAP), which spans 18 years with 34,686,770 reviews from 6,643,669 users on 2,441,053 products [McAuley and Leskovec, 2013]. Similarly to the Yelp review dataset, we also constructed 2 datasets – one full score prediction and another polarity prediction. The full dataset contains 600,000 training samples and 130,000 testing samples in each class, whereas the polarity dataset contains 1,800,000 training samples and 200,000 testing samples in each polarity sentiment. The fields used are review title and review content.

Table 2.4 lists all the testing errors we obtained from these datasets for all the applicable models. Note that since we do not have a Chinese thesaurus, the Sogou News dataset does not have any results using thesaurus augmentation. We labeled the best result in blue and worse result in red.

2.5 Discussion

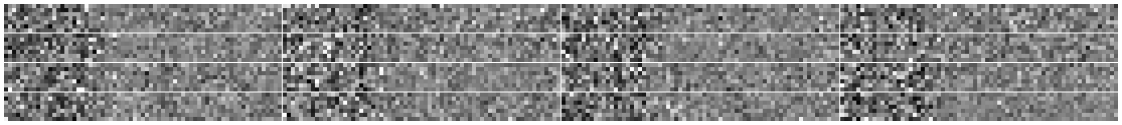


Figure 2.9: First layer weights. For each patch, height is the kernel size and width the alphabet size

To understand the results in table 2.4 further, we offer some empirical analysis

in this section. To facilitate our analysis, we present the relative errors in figures 2.3 - 2.8 with respect to comparison models. Each of these plots is computed by taking the difference between errors on comparison model and our character-level ConvNet model, then divided by the comparison model error. All ConvNets in the figure are the large models with thesaurus augmentation respectively.

Character-level ConvNet is an effective method. The most important conclusion from our experiments is that character-level ConvNets could work for text classification without the need for words. This is a strong indication that language could also be thought of as a signal no different from any other kind. Figure 2.9 shows 12 random first-layer patches learnt by one of our character-level ConvNets for DBPedia dataset.

Dataset size forms a dichotomy between traditional and ConvNets models. The most obvious trend coming from all the plots in figures 2.3 - 2.8 is that the larger datasets tend to perform better. Traditional methods like n-grams TFIDF remain strong candidates for dataset of size up to several hundreds of thousands, and only until the dataset goes to the scale of several millions do we observe that character-level ConvNets start to do better.

ConvNets may work well for user-generated data. User-generated data vary in the degree of how well the texts are curated. For example, in our million scale datasets, Amazon reviews tend to be raw user-inputs, whereas users might be extra careful in their writings on Yahoo! Answers. Plots comparing word-based deep models (figures 2.5, 2.6 and 2.7) show that character-level ConvNets work better for less curated user-generated texts. This property suggests that ConvNets may have better applicability to real-world scenarios. However, further analysis is needed to validate the hypothesis that ConvNets are truly good at

identifying exotic character combinations such as misspellings and emoticons, as our experiments alone do not show any explicit evidence.

Choice of alphabet makes a difference. Figure 2.8 shows that changing the alphabet by distinguishing between uppercase and lowercase letters could make a difference. For million-scale datasets, it seems that not making such distinction usually works better. One possible explanation is that there is a regularization effect, but this is to be validated.

Semantics of tasks may not matter. Our datasets consist of two kinds of tasks: sentiment analysis (Yelp and Amazon reviews) and topic classification (all others). This dichotomy in task semantics does not seem to play a role in deciding which method is better.

Bag-of-means is a misuse of word2vec [Lev et al., 2015]. One of the most obvious facts one could observe from table 2.4 and figure 2.3 is that the bag-of-means model performs worse in every case. Comparing with traditional models, this suggests such a simple use of a distributed word representation may not give us an advantage to text classification. However, our experiments does not speak for any other language processing tasks or use of word2vec in any other way.

There is no free lunch. Our experiments once again verifies that there is not a single machine learning model that can work for all kinds of datasets. The factors discussed in this section could all play a role in deciding which method is the best for some specific application.

2.6 Conclusion and Outlook

This chapter offers an empirical study on character-level convolutional networks for text classification. We compared with a large number of traditional and deep learning models using several large-scale datasets. On one hand, analysis shows that character-level ConvNet is an effective method. On the other hand, how well our model performs in comparisons depends on many factors, such as dataset size, whether the texts are curated and choice of alphabet.

In the future, we hope to apply character-level ConvNets for a broader range of language processing tasks especially when structured outputs are needed.

Chapter 3

Which Encoding is the Best for Text Classification in Chinese, English, Japanese and Korean?

This chapter offers an empirical study on the different ways of encoding Chinese, Japanese, Korean (CJK) and English languages for text classification. Different encoding levels are studied, including UTF-8 bytes, characters, words, romanized characters and romanized words. For all encoding levels, whenever applicable, we provide comparisons with linear models, fastText [Joulin et al., 2016] and convolutional networks. For convolutional networks, we compare between encoding mechanisms using character glyph images, one-hot (or one-of-n) encoding, and embedding. In total there are 473 models, using 14 large-scale text classification datasets in 4 languages including Chinese, English, Japanese and Korean. Some conclusions from these results include that byte-level one-hot encoding based on UTF-8 consistently produces competitive results for convolutional networks, that

word-level n-grams linear models are competitive even without perfect word segmentation, and that fastText provides the best result using character-level n-gram encoding but can overfit when the features are overly rich.

3.1 Introduction

Processing different kinds of languages in a fashion that ensures model generality acrosss languages is of great interest to the natural language processing (NLP) community, especially with the recent advancements in deep learning methods. Among these languages, Chinese, Japanese and Korean (CJK) pose unique challenges due to reasons in both linguistics and computation. Unlike some alphabetic languages such as English, there is no clear word boundary for some of the Chinese and Japanese texts. This makes it difficult to apply many language processing methods that assume word as the basic construct.

Layers	Description
1-2	Conv 256x3
3	Pool 2
4-5	Conv 256x3
6	Pool 2
7-8	Conv 256x3
9	Pool 2
10-11	Conv 256x3
12	Pool 2
13-14	Conv 256x3
15	Pool 2
16-17	Full 1024

Table 3.1: The large classifier

Recently, many authors have proposed to use character-level encoding for language processing with convolutional networks (ConvNets) [Kim et al., 2016] [Zhang

et al., 2015], casting away the word segmentation problem. Unfortunately, working with characters for CJK languages is not direct, because the amount of characters can be huge. For example, one-hot (or one-of-n) encoding used by [Zhang et al., 2015] is not practical because each one-hot vector would be prohibitively large.

This drives us to search for alternative ways of encoding CJK texts. The encoding mechanisms considered in this chapter include character glyph images, one-hot encoding and embedding. For one-hot encoding, we considered feasible encoding levels including UTF-8 bytes and characters after romanization. For embedding, we performed experiments on encoding levels including character, UTF-8 bytes, romanized characters, segmented word with a prebuilt word segmenter, and romanized word. A brief search in the literature seems to confirm that this chapter is the first to study all of these encoding mechanisms in a systematic fashion.

Layers	Description
1-2	Conv 256x3
3	Pool 3
4-5	Conv 256x3
6	Pool 3
7-8	Conv 256x3
9	Pool 3
10-11	Full 1024

Table 3.2: The small classifier

Historically, linear models such as (multinomial) logistic regression [Cox, 1958] and support vector machines [Cortes and Vapnik, 1995] have been the default choice for text classification, with bag-of-words features and variants such as n-grams and TF-IDF [Sparck Jones, 1972]. Therefore, in this chapter we provide extensive comparisons using multinomial logistic regression, with bag-of-characters, bag-of-words and their n-gram and TF-IDF [Sparck Jones, 1972] variants. Fur-

thermore, experiments using the recently proposed fastText [Joulin et al., 2016] are also presented with all these different feature variants.

Large-scale multi-lingual datasets are required to make sure that our comparisons are meaningful. Therefore, we set out to crawl the Internet for several large-scale text classification datasets. Eventually, we were able to obtain 14 large-scale datasets in 4 languages including Chinese, English, Japanese and Korean, for 2 different tasks including sentiment analysis and topic categorization. We released all the code used in this chapter under an open source license, including crawling, preprocessing, and training on all datasets [1].

The conclusions of this chapter include that the one-hot encoding model at UTF-8 byte level consistently offers competitive results for convolutional networks, that linear models remain strong for the text classification task, and that fastText provides the best results with character n-grams but tends to overfit when the features are overly rich. We hope that these results can offer useful guidance for the community to select appropriate encoding mechanisms that can handle different languages in a fashion that generalizes across languages.

3.2 Encoding Mechanisms for Convolutional Networks

For the purpose of fair comparisons, all of our convolutional networks share the same design except for the first few layers. We call this common part the classifier, and the different first several layers the encoder. In the benchmarks we have 2 classifier designs - one large and the other small. The large classifier

[1] `https://github.com/zhangxiangxiao/glyph`

consists of 12 layers, and the small one 8. Table 3.1 and 3.2 details the designs. All parameterized layers use ReLU [Nair and Hinton, 2010] as the non-linearity.

3.2.1 Character Glyph

Glyph is a typography term indicating a readable character for the purposes of writing. CJK languages consist of characters that are rich in their topological forms, where strokes and parts could represent semantic meaning. This makes glyph a potentially feasible encoding solution.

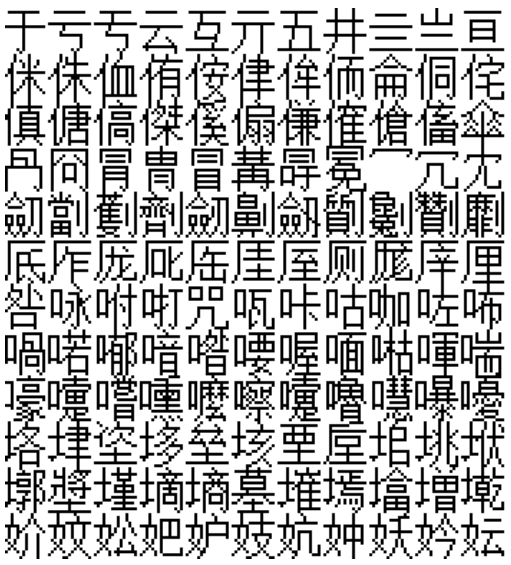


Figure 3.1: GNU Unifont

In the context of this chapter, we refer to glyphs as images of characters rendered by some font. In the experiments we use the freely available GNU Unifont [2] (version 8.0.01), where each character is converted to a 16-by-16 pixel image. We consider all characters that belong to the Unicode basic multi-lingual plane (BMP), which have code points less than or equal to the hex value FFFF. Figure 3.1 shows some glyph examples in this font.

For the large classifier, the glyph encoder contains 8 parameterized layers with 6 spatial convolutional layers and 2 linear layers. The small model consists of a 6-layer glyph encoder with 4 spatial convolutional layers and 2 linear layers. Table

[2] `http://unifoundry.com/unifont.html`

3.3 and 3.5 present the design choices.

In the benchmarks we will refer to these 2 models as large GlyphNet and small GlyphNet respectively. During training, each sample consists of at most 512 characters for the large GlyphNet and 486 for the small one. Zero is padded if the length of the sample string is shorter, and characters beyond these limits are ignored. Note that each character must pass through the spatial glyph encoder and each sample could contain hundreds of characters. As a result, the training time of GlyphNet is significantly longer than any other model considered in this chapter.

Layers	Description
1-2	Conv 64x3x3
3	Pool 2
4-5	Conv 128x3x3
6	Pool 2
7-8	Conv 256x3x3
9	Pool 2
10	Full 1024
11	Full 256

Table 3.3: Large GlyphNet encoder

It is worth noting that recent research has shown that CJK characters can help to improve the results of various tasks including text classification [Shimada et al., 2016] [Liu et al., 2017] and translation [Costa-jussà et al., 2017], further justifying the potential of encoding CJK characters via glyphs.

3.2.2 One-hot Encoding

In the simplest version of one-hot (or one-of-n) encoding, each entity must be converted into a vector whose size equals to the cardinality of the set of all possible entities, and all values in this vector are zero except for the position

Dataset	Language	#	Train	Dev	Val	Batch
Dianping	Chinese	2	2,000,000	250,000	250,000	100,000
JD full	Chinese	5	3,000,000	125,000	125,000	100,000
JD binary	Chinese	2	4,000,000	180,000	180,000	100,000
Rakuten full	Japanese	5	4,000,000	250,000	250,000	100,000
Rakuten binary	Japanese	2	3,400,000	200,000	200,000	100,000
11st full	Korean	5	750,000	50,000	50,000	100,000
11st binary	Korean	2	4,000,000	200,000	200,000	100,000
Amazon full	English	5	3,000,000	325,000	325,000	100,000
Amazon binary	English	2	3,600,000	200,000	200,000	100,000
Ifeng	Chinese	5	800,000	25,000	25,000	100,000
Chinanews	Chinese	7	1,400,000	56,000	56,000	100,000
NYTimes	English	7	1,400,000	52,500	52,500	100,000
Joint full	Multilingual	5	10,750,000	750,000	750,000	400,000
Joint binary	Multilingual	2	15,000,000	780,000	780,000	400,000

Table 3.4: Datasets. The 3rd column is the number of classes.

that corresponds to the index of the entity in the set. For example, in the paper by [Zhang et al., 2015], each entity is a character and the size of the vector equals to the size of the alphabet containing all characters. Unfortunately, this naive way of using one-hot encoding is only computationally feasible if the entity set is relatively small. Texts in CJK languages can easily span tens of thousands of characters.

In this chapter, we consider 2 simple solutions to this problem. The first one is to treat the text (in UTF-8) as a sequence of bytes and encode at byte-level. The second one, already presented in [Zhang et al., 2015], is to romanize the text so that encoding using the English alphabet is feasible. Note that the second solution is equivalent of encoding at byte-level with romanized text, because the English alphabet is contained in UTF-8 and they will not go beyond the limit of one byte.

In the following we will call these 2 models byte-level OnehotNet and roman-

ization OnehotNet. Similar to GlyphNet, each OnehotNet also has a large variant and a small variant depending on the classifier used. Both variants use the same encoder design that consists of 4 convolutional layers, in which the large variant admits input length 2048 and the small 1944. Table 3.6 provides the configuration. Compared to GlyphNet, OnehotNet is significantly faster because the encoder handles all symbols in the input at once.

The idea of language processing at byte level has been explored by [Gillick et al., 2016], where they apply an LSTM-based [Hochreiter and Schmidhuber, 1997] sequence-to-sequence [Cho et al., 2014b] [Sutskever et al., 2014] model at byte-level for a variety of tasks including part-of-speech tagging and named entity recognition, for 4 languages including English, German, Spanish and Dutch. The advantage of byte-level processing is that they can be immediately applied to any language regardless of whether there are too many entities at character or word levels. The same advantage applies to CJK, and perhaps any language that can be digitized as well.

Layers	Description
1-2	Conv 64x3x3
3	Pool 3
4-5	Conv 128x3x3
6	Pool 3
7-8	Full 256

Table 3.5: Small GlyphNet encoder

3.2.3 Embedding

We use the terminology "embedding" to refer to the idea of associating each entity a fixed size vector, same as most papers in the machine learning literature.

These vectors are randomly initialized, and then learnt either with an unsupervised criterion or jointly with the task at hand. The advantage of embedding models is there there is no need to explicitly construct one-hot vectors, therefore the memory footprint of embedding models is significantly smaller than that of OnehotNet. As a result, embedding can be applied to almost any encoding level.

In this chapter, we use embedding at a variety of different levels, including byte, character, word, romanization character, and romanization word. All of of our embedding vectors are of size 256, and they are learnt jointly with the text classification task at hand. The size of vocabulary is 257 for byte-level and romanized-level encoding, 65537 for character-level encoding, and 200,002 for word level and romanized word-level encoding.

Layers	Description
1-2	Conv 256x3
3	Pool 2
4-5	Conv 256x3
6	Pool 2

Table 3.6: OnehotNet encoder

The character-level encoding considers all code points in the basic multilingual plane (BMP) of Unicode. The word and romanized-word vocabularies are built by selecting the 200,000 most frequent entities appeared in the training data for each dataset, plus one additional entry to represent an out-of-vocabulary symbol. One additional entry is also added to each vocabulary to include a padding symbol for shorter texts. There are 2 embedding models, since we have designed the classifier with 2 different sizes. We will refer to them as large EmbedNet and small EmbedNet respectively. The large Embednet admits input length of 512, and the small one 486.

When the input text is represented by explicit one-hot vectors, embedding is equivalent of using a linear first layer. Therefore, the difference between OnehotNet and EmbedNet in this chapter is whether the first layer is linear or convolutional.

The idea of embedding has been applied to ConvNet-based text processing pretty early on, with representative work for tasks like named entity recognition, part-of-speech tagging [Collobert et al., 2011c], text classification at word level [Kim, 2014] and language modeling at character level [Kim et al., 2016].

Dataset	Large	Small
Dianping	24.26	24.45
JD f.	48.94	49.25
JD b.	9.82	10.07
Rakuten f.	46.82	47.06
Rakuten b.	6.64	6.81
11st f.	32.70	32.98
11st b.	13.87	14.29
Amazon f.	46.63	47.95
Amazon b.	8.56	9.11
Ifeng	18.04	18.57
Chinanews	12.28	12.91
NYTimes	18.20	18.58
Joint f.	45.20	45.82
Joint b.	10.00	10.42

Table 3.7: GlyphNet results. The numbers are development errors in percentage.

3.3 Linear Models and fastText

Besides ConvNets, we also offer benchmarks in linear models using multinomial logistic regression, and the fastText program by [Joulin et al., 2016].

3.3.1 Linear Models

The linear multinomial logistic regression models are all bag-of-entity models, where the entity is character, word, romanized word. The 1-gram bag-of-entity model admits a feature of size 200,000 by selecting the most frequent ones from the training dataset. The 5-gram model admits grams of length up to 5, using the 1,000,000 most frequent features in the training dataset.

Dataset	Byte		Romanized	
	large	small	large	small
Dianping	**23.08**	23.21	23.46	**23.46**
JD f.	**48.06**	48.24	48.40	**48.49**
JD b.	9.32	**9.29**	9.47	**9.49**
Rakuten f.	**45.13**	45.39	45.15	**45.41**
Rakuten b.	**5.92**	**6.09**	6.01	6.06
11st f.	32.53	**32.39**	**32.69**	32.66
11st b.	**13.29**	13.31	13.40	**13.43**
Amazon f.	**42.22**	**42.34**	–	–
Amazon b.	**6.53**	**6.60**	–	–
Ifeng	16.72	**16.50**	**18.94**	18.93
Chinanews	**10.64**	10.75	11.74	**11.79**
NYTimes	**14.28**	**14.24**	–	–
Joint f.	**42.95**	43.11	**43.31**	43.29
Joint b.	**8.80**	8.80	9.02	**9.04**

Table 3.8: OnehotNet results. The numbers are development errors in percentage. The best result for each dataset is marked blue and the worst red.

Note that word segmentation is not a simple problem for some of Chinese and Japanese texts, because they sometimes do not contain clear word boundaries like the space character in most alphabetic languages. Section 3.4.2 introduces how word segmentation is done for each language.

The idea of bag-of-character and its n-gram version has been explored by [Peng et al., 2003] for text classification in Asian languages, where they observed comparable results with word-level models. This is probably because of the large

character vocabularies in these languages, in which each character has a similar sparsity in representing meaning compared to each word in an alphabetic language.

3.3.2 fastText

fastText [Joulin et al., 2016] is a recent tool for fast text classification by incorporating several tricks such as hierarchical softmax [Goodman, 2001] [Mikolov et al., 2013a] and feature hashing [Weinberger et al., 2009]. Combined with an efficient implementation and a highly optimized learning rate schedule, fastText is able to process input text at a speed of several orders of magnitude of that of ConvNets. This gives it a particular advantage and we hope to include the its results as a reference for our community.

The fastText model is essentially a 2-layer fully connected neural network without non-linearity. The number of hidden units is 10 across all of our experiments. During training, we use an initial learning rate of 0.1 and a hashing bucket size of 10,000,000. We used 10% of the training dataset as validation and remaining as training to choose the best number of epoches, from the choices 2, 5 and 10. This validation process necessary because fastText does not have weight decay [Joulin et al., 2016] and it relies on early stopping to prevent overfitting. It is also the only model fast enough for such hyper-parameter tuning in this chapter. For each dataset, we explored features at character, word and romanized word levels, with variants of 1-gram, 2-gram and 5-gram features.

3.4 Datasets and Preprocessing

To ensure that our results are significant enough to demonstrate the differences between encoding methods, we need to acquire large-scale datasets. To do that, we set out to crawl the Internet for text classification datasets in 4 language including Chinese, English, Japanese and Korean. Eventually, we were able to obtain 14 datasets, most of which are at the scale of millions of samples. We performed experiments using all aforementioned models on all of these datasets.

3.4.1 Datasets

In total, we have obtained 8 sentiment classification datasets from online shopping reviews in Chinese, English, Japanese and Korean, 1 sentiment classification dataset from online restaurant reviews in Chinese, and 3 news topic classification dataset in English and Chinese. Additionally, we were able to combine the online shopping review datasets in different languages to construct 2 joint datasets, which can be used to test each model's ability to handle different languages in a unified fashion. Table 3.4 summarizes the statistics of all these datasets.

Dianping. The Dianping dataset consists of user reviews crawled from Chinese online restaurant review website `dianping.com`. This dataset was developed and used by Zhang et al. for research in collaborative filtering [Zhang et al., 2013a] [Zhang et al., 2013b] and sentiment analysis [Zhang et al., 2014a] [Zhang et al., 2014b]. After removing duplicated texts, we pre-processed the dataset such that stars 1, 2 and 3 belong to the negative class, and stars 4 and 5 belong to the positive class. Then we randomly selected 2,000,000 samples for training and 500,000 samples for testing with equal number of samples in each sentiment. The

testing dataset is then split into a development dataset and a validation dataset in equal proportions.

JD. The JD dataset consists of user reviews crawled from the Chinese online shopping website `jd.com`. After duplication removal, we were able to obtain 2 sentiment classification datasets in which one is to predict the full 5 stars and the other is binary. The binary dataset was built such that stars 1 and 2 belong to the negative sentiment, and stars 4 and 5 belong to the positive sentiment. Star 3 is ignored in the JD binary dataset. There are 3,000,000 training samples and 250,000 testing samples in the JD full dataset, and 4,000,000 training samples and 360,000 testing samples in the JD binary dataset. In each case, the samples are evenly distributed across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

Rakuten. The Rakuten dataset consists of user reviews crawled from the Japanese online shopping website `rakuten.co.jp`. After duplication removal, we were able to obtain 2 sentiment classification datasets in which one is to predict the full 5 stars and the other is binary. The binary dataset was built such that stars 1 and 2 belong to the negative sentiment, and stars 4 and 5 belong to the positive sentiment. Star 3 is ignored in the Rakuten binary dataset. There are 4,000,000 training samples and 500,000 testing samples in the Rakuten full dataset, and 3,400,000 training samples and 400,000 testing samples in the Rakuten binary dataset. In each case, the samples are evenly distributed across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

11st. The 11st dataset consists of user reviews crawled from the Korean online shopping website `11st.co.kr`. After duplication removal, we were able to obtain

Dataset	Character		Byte		Romanized		Word		Rom. word	
	large	small	large	small	large	small	large	small	large	small
Dianping	**23.52**	23.63	24.00	24.15	25.35	**25.99**	24.50	24.60	23.64	23.75
JD f.	**48.24**	48.40	48.54	48.61	48.73	49.26	50.01	**50.11**	49.12	49.21
JD b.	9.41	9.39	**9.17**	9.19	9.43	9.69	10.35	**10.43**	9.56	9.67
Rakuten f.	**45.21**	45.70	45.97	**46.92**	46.17	46.80	46.35	46.57	45.97	46.37
Rakuten b.	**6.05**	6.11	6.47	6.84	6.54	**6.93**	6.80	6.84	6.54	6.65
11st f.	**32.26**	32.31	34.80	34.99	35.38	35.67	**42.86**	42.76	42.57	42.50
11st b.	13.42	13.48	**13.23**	13.46	13.46	13.68	16.39	15.71	17.63	**17.66**
Amazon f.	**43.72**	44.23	–	–	–	–	44.29	**44.84**	–	–
Amazon b.	**7.19**	7.50	–	–	–	–	7.93	**8.04**	–	–
Ifeng	**17.03**	17.11	17.14	17.57	19.23	20.02	**20.83**	20.75	19.49	19.50
Chinanews	11.04	11.12	**10.55**	10.84	11.84	12.77	14.75	**14.95**	11.92	12.06
NYTimes	**14.10**	14.59	–	–	–	–	17.62	**17.77**	–	–
Joint f.	**43.67**	44.13	44.22	44.82	44.76	45.49	45.50	**45.88**	45.04	45.35
Joint b.	**9.03**	9.20	9.11	9.29	9.35	9.67	10.67	**10.79**	9.96	10.03

Table 3.9: EmbedNet results. The numbers are development errors in percentage. The best result for each dataset is marked blue and the worst red.

2 sentiment classification datasets in which one is to predict the full 5 stars and the other is binary. The binary dataset was built such that stars 1, 2 and 3 belong to the negative sentiment, and stars 4 and 5 belong to the positive sentiment. There are 750,000 training samples and 100,000 testing samples in the 11st full dataset, and 4,000,000 training samples and 400,000 testing samples in the 11st binary dataset. In each case, the samples are evenly distributed across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

Amazon. The Amazon dataset consists of users reviews crawled from the English online shopping website `amazon.com`. We use the same datasets constructed by [Zhang et al., 2015], which came from the Stanford Network Analysis Project

(SNAP) [3] and developed by [McAuley and Leskovec, 2013] for sentiment analysis. There are 2 sentiment classification datasets in which one is to predict the full 5 stars and the other is binary. The binary dataset was built such that stars 1 and 2 belong to the negative sentiment, and stars 4 and 5 belong to the positive sentiment. Star 3 is ignored in the Amazon binary dataset. There are 3,000,000 training samples and 650,000 testing samples in the Amazon full dataset, and 3,600,000 training samples and 400,000 testing samples in the Amazon binary dataset. In each case, the samples are evenly distributed across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

Ifeng. The Ifeng dataset consists of first paragraphs of news articles from the Chinese news website `ifeng.com`. We crawled all news from the year 2006 to the year 2016 and selected 5 different news channels as 5 topic classes. These classes are mainland China politics, International news, Taiwan - Hong Kong- Macau politics, military news, and society news. After duplication removal, the dataset consists of 800,000 training samples and 50,000 testing samples. These samples are evenly distributed across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

Chinanews. The Chinanews dataset consists of first paragraphs of news articles from the Chinese news website `chinanews.com`. We crawled all news from the year 2008 to the year 2016 and selected 7 different news channels as 7 topic classes. These classes are mainland China politics, Hong Kong - Macau politics, Taiwan politics, International news, financial news, culture, entertainment, sports, and health. After duplication removal, the dataset consists of 1,400,000 training samples and 112,000 testing samples. These samples are evenly distributed

[3] `http://snap.stanford.edu/`

Dataset	Character				Word				Romanized Word			
	1-gram		5-gram		1-gram		5-gram		1-gram		5-gram	
	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf
Dianping	25.99	26.78	24.20	23.53	23.96	24.17	23.51	**22.99**	27.26	**27.93**	24.48	23.28
JD f.	51.43	51.60	48.39	**48.14**	49.35	50.03	48.26	48.27	52.10	**52.75**	48.95	48.44
JD b.	11.79	12.11	9.09	8.89	9.83	9.96	9.05	**8.81**	13.10	**13.51**	9.10	8.97
Rakuten f.	52.18	**52.84**	47.63	46.50	47.60	47.75	45.82	**45.29**	47.93	48.34	46.98	45.68
Rakuten b.	12.50	**12.98**	8.13	7.28	8.36	8.39	7.33	**6.62**	8.80	8.94	7.45	6.70
11st f.	43.83	48.32	43.55	**43.38**	49.75	48.23	**54.80**	51.88	45.33	44.72	45.53	44.21
11st b.	17.65	18.00	14.44	**14.33**	15.33	15.57	**18.92**	17.24	14.94	15.15	14.61	14.41
Amazon f.	**69.49**	68.61	56.89	51.36	45.42	44.92	44.68	**42.73**	–	–	–	–
Amazon b.	**34.48**	33.70	15.02	12.17	9.35	8.82	8.56	**8.25**	–	–	–	–
Ifeng	22.13	22.45	21.55	22.00	19.14	**18.32**	20.20	19.75	26.61	**27.34**	23.13	22.39
Chinanews	15.38	15.09	14.95	13.39	11.66	**10.78**	13.44	12.94	20.07	**20.50**	15.59	14.00
NYTimes	**57.54**	53.85	40.89	26.39	18.22	**15.29**	20.04	18.28	–	–	–	–
Joint f.	**60.29**	59.71	49.20	48.23	46.84	46.56	45.28	**45.08**	47.55	47.18	46.91	46.42
Joint b.	**20.21**	19.74	12.13	10.93	10.85	10.68	9.48	**9.01**	11.74	11.45	11.33	11.02

Table 3.10: Linear model results. The numbers are development errors in percentage. The best result for each dataset is marked blue and the worst red.

across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

NYTimes. The NYTimes dataset consists of first paragraphs of news articles from the English news website `nytimes.com`. We crawled all news from the year 1981 to the year 2015 and combined several channels to construct 7 topic classes. These classes are business news, New York regional news, sports, U.S. politics, world news and opinions, arts and fashion, and entertainment and science. After duplication removal, the dataset consists of 1,400,000 training samples and 105,000 testing samples. These samples are evenly distributed across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

Joint. The four dataset sources JD, Rakuten, 11st and Amazon are all sen-

Dataset	Character			Word			Romanized Word		
	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram
Dianping	25.78	22.80	**22.28**	23.64	22.55	22.57	**26.89**	22.86	22.37
JD f.	51.21	48.27	**47.96**	49.20	48.07	48.55	**52.18**	48.32	48.09
JD b.	11.79	8.89	**8.70**	9.82	9.08	9.91	**13.03**	9.04	8.70
Rakuten f.	**52.01**	44.90	**43.30**	46.58	43.83	46.33	47.08	43.82	43.30
Rakuten b.	**12.22**	6.91	**5.43**	8.01	5.89	5.43	8.45	5.93	5.43
11st f.	**43.17**	38.67	**38.53**	41.58	41.12	42.14	40.67	41.68	43.12
11st b.	**17.64**	13.63	**13.09**	14.51	14.25	14.87	14.85	14.49	14.96
Amazon f.	**67.08**	54.00	41.07	43.82	40.21	**40.05**	–	–	–
Amazon b.	**32.81**	18.59	6.30	8.38	5.60	**5.42**	–	–	–
Ifeng	**21.62**	16.60	**16.34**	17.85	16.67	16.97	26.06	18.22	17.88
Chinanews	13.94	9.33	**9.13**	9.88	9.28	9.25	**18.71**	9.69	9.40
NYTimes	**51.39**	24.65	12.69	13.59	**11.81**	13.21	–	–	–
Joint f.	**59.50**	49.38	44.05	46.07	43.56	43.66	46.74	**43.15**	43.28
Joint b.	**19.86**	12.37	**9.02**	10.06	9.05	9.28	11.40	8.82	8.85

Table 3.11: fastText results. The numbers are development errors in percentage. The best result for each dataset is marked blue and the worst red.

timent classification tasks from online shopping websites, with both full 5 stars prediction or binary prediction. Therefore, we could combine them in each case to form two new joint datasets of 5 classes or 2 classes. This dataset is particularly useful since it spans 4 languages and can be used to test a model's ability to handle different languages in a unified fashion. In total, there are 10,750,000 training samples and 1,500,000 testing samples in the joint full dataset, and 15,000,000 training samples and 1,560,000 testing samples in the joint binary dataset. All samples are evenly distributed across classes. The testing dataset is then split into a development dataset and a validation dataset in equal proportions.

3.4.2 Word Segmentation and Romanization

Since there is no clear word boundary in some of the Chinese and Japanese texts, word segmentation is necessary before applying any of the word-level models. Romanization for some of the CJK texts also depends on word segmentation to produce the correct transliteration in the English alphabet. In this section, we present both word segmentation and romanization processes used for producing the results, for each languages Chinese, Japanese and Korean. All the tools we used are relatively popular and standard for CJK language processing.

Chinese. For Chinese, we use the freely available word segmentation package called jieba [4] (version 0.38). The romanization standard we used is Pinyin, using the pypinyin [5] (version 0.12) package which in turn calls jieba for disambiguate between characters with multiple pronunciations.

Japanese. For Japanese, we use the freely available word segmentation and tagging package MeCab [6] (version 0.996) with the default model for Japanese. The romanization form used is Hepburn, which is done by converting the segmented words using python-romkan [7] (version 0.2.1).

Korean. Word segmentation is done for Korean using the python package hangul-utils [8], facilitated by a MeCab version in the Korean language [9]. The romanization standard used is the Revised Romanization of Korean (RR), which is done in 2 steps. The first step is to convert any Hanja in the text to Hangul via the python package hanja (version 0.11) [10], and the second step is to transliterate

[4] https://github.com/fxsjy/jieba
[5] https://github.com/mozillazg/python-pinyin
[6] http://taku910.github.io/mecab
[7] https://www.soimort.org/python-romkan
[8] https://github.com/kaniblu/hangul-utils
[9] https://bitbucket.org/eunjeon/mecab-ko-dic
[10] https://github.com/suminb/hanja

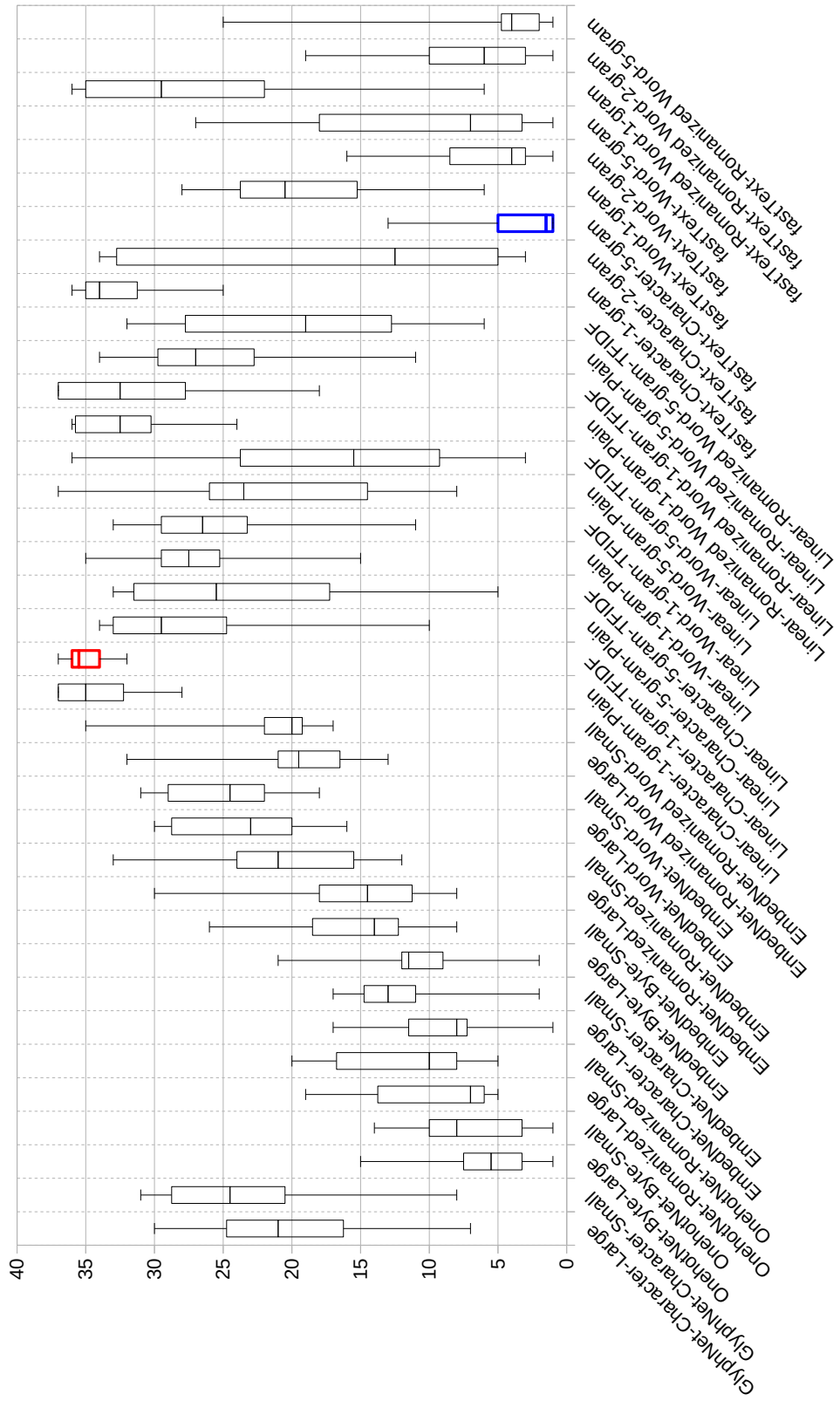


Figure 3.2: Rank box plot of development errors for different models

the generate Hangul using the python package hangul-romanize [11].

3.5 Experiments

After introducing the optimization parameters used for all of our models, this section then presents the results for these models. Most of our experiments are implemented using Torch 7 [Collobert et al., 2011a], with NVIDIA CUDNN [12] as the GPU backend.

3.5.1 Optimization

The optimization process used for all convolutional network models is stochastic gradient descent (SGD) with momentum [Polyak, 1964] [Sutskever et al., 2013]. The training process operates on random minimabatches of size 16, with different numbers of minibatches per epoch. The sixth column in Table 3.4 shows the number of minibatches for one epoch for each dataset. The model parameters are initialized in the same way as in [He et al., 2015] – for each layer the bias is set to 0, and weights are randomly sampled from a Gaussian distribution of mean 0 and standard deviation $\sqrt{2/n}$, where n is the number of output units each input unit connects to. All the models have an initial learning rate of 0.00001, which is halved every 8 epoches. The training stops at the 100th epoch. A small weight decay of 0.00001 is applied to the model to stabilize training. Each model is trained using one NVIDIA Tesla K40 GPU.

The optimizaiton algorithm used for all linear models is parallelized SGD. Each model is trained with a sparse representation via HOGWILD! [Niu et al., 2011]

[11] `https://github.com/youknowone/hangul-romanize`
[12] `https://developer.nvidia.com/cudnn`

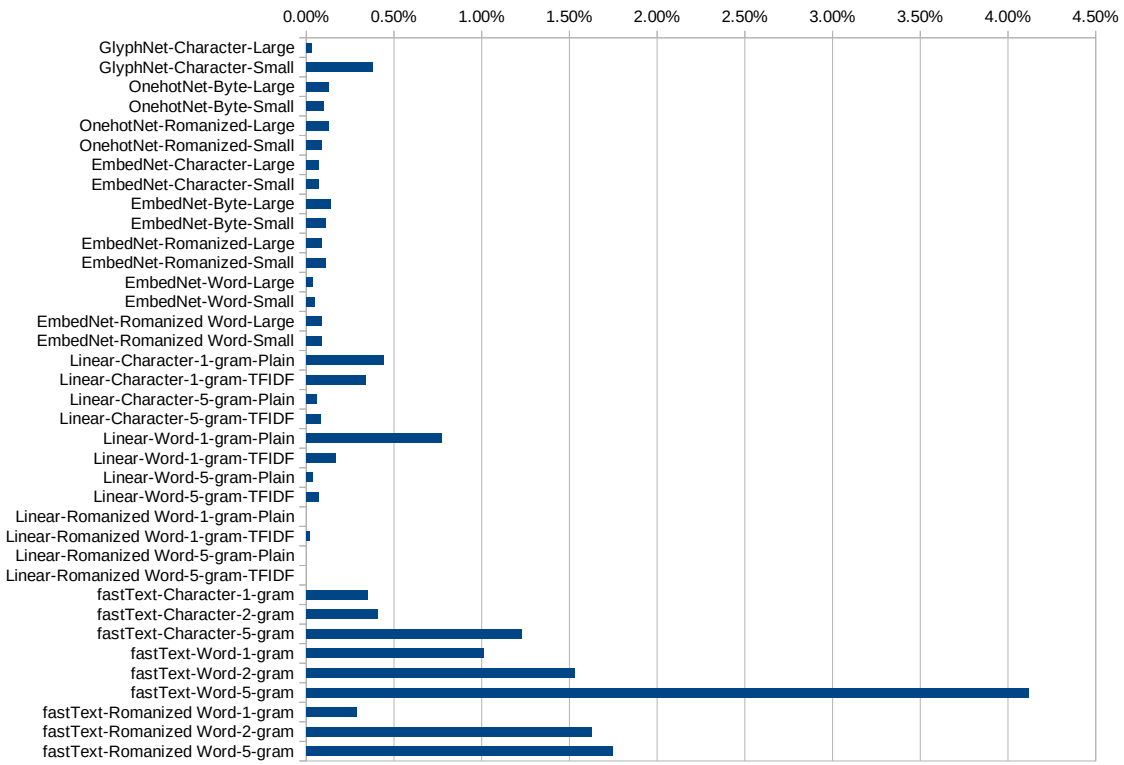


Figure 3.3: Generalization gap of Joint binary dataset

parallelization using 10 CPU cores. An extra core is used for continuously testing on both training and development datasets. The learning rate used for the algorithm is 0.001. A small weight decay of 0.00001 is applied to each model to stabilize the training process. The training stops after 1000 continuous testing steps are done. All of our models are run with a batch of INTEL XEON E5-2630 v2 CPUs.

The optimization parameters for fastText [Joulin et al., 2016] are controlled by the original authors' program [13]. We set the embedding dimension to be 10 with a bucket size of 10,000,000, and going through each dataset for 2, 5 or 10 epoches depending on the validation result from 10% of the training dataset. The opti-

[13] `https://github.com/facebookresearch/fastText`

mization algorithm is SGD with decaying learning rate, where the initial learning rate is set to 0.1 and the decay change rate set to 100. The number of CPU cores used is 10, with a batch of INTEL XEON E5-2630 v2 CPUs. All other parameters used are the program's defaults.

3.5.2 Results

The results for all the models are split into several tables. Table 3.7 lists the results for GlyphNet, where the numbers are development errors in percentages. Similarly, Tables 3.8, 3.9, 3.10 and 3.11 list the development errors for OnehotNet, EmbedNet, linear models and fastText. As long as it is applicable in each table, the best result for each dataset is marked blue and the worst red. The epoch numbers for fastText models are presented in appendix B

For each Chinese, Japanese and Korean dataset, we have 37 models each, and for English we have 22. In total, there are 473 models benchmarked in this chapter. Due to space limitations, the training and validation errors are not present in the main text of this chapter, but readers can refer to appendix A for them.

3.6 Analysis

In this section, we provide some analysis on the development results presented in the previous section. These analyses include average ranks between models, generalization ability of each model under different encoding mechanisms, and estimations of training time.

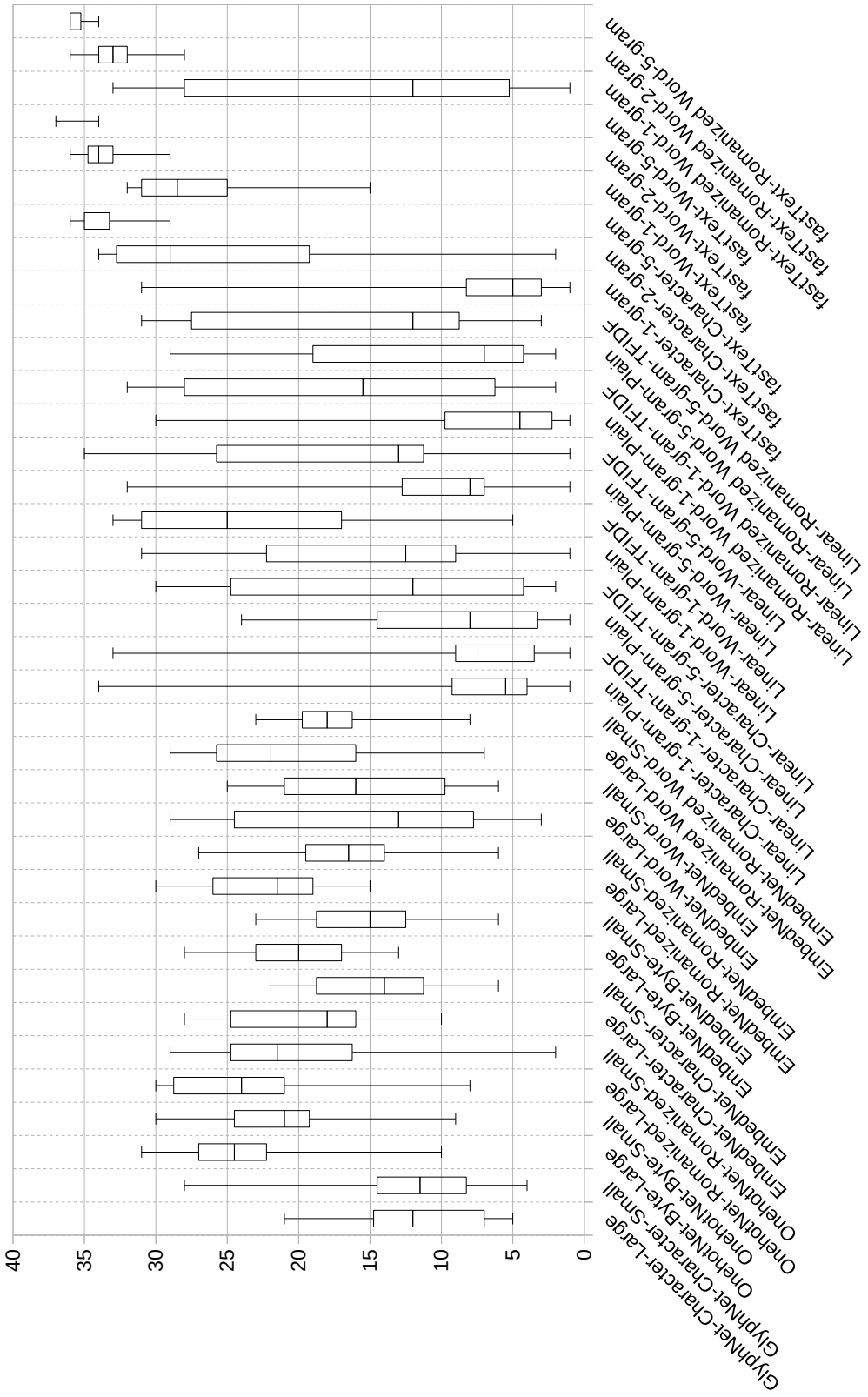


Figure 3.4: Rank box plot of generalization gap for different models

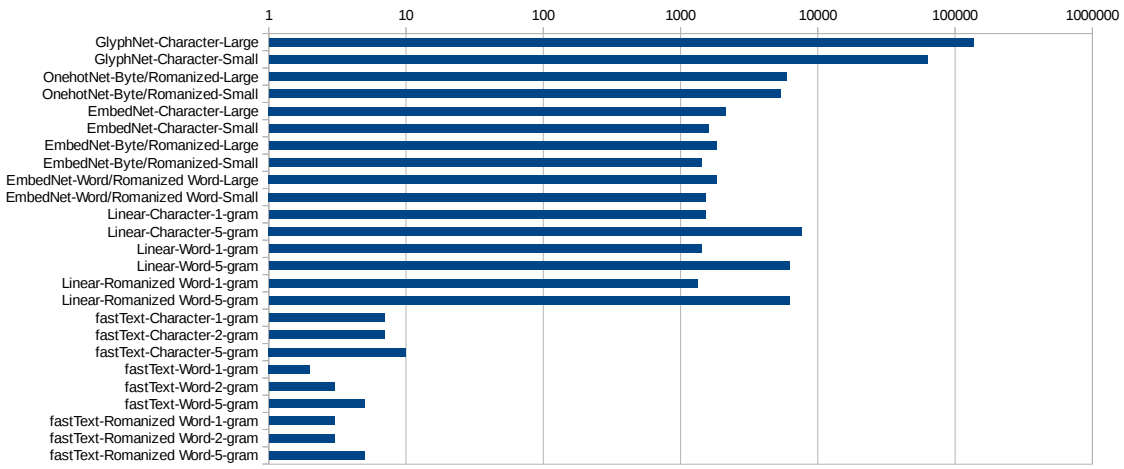


Figure 3.5: Time for different models to go over 1,000,000 samples. The time axis is in logarithmic scale.

3.6.1 Rank the Models

To compare between different encoding mechanisms, this section presents the ranking of development errors of all models. For English datasets, there are some missing values in various models in Tables 3.8, 3.9, 3.10 and 3.11. These values are missing because the corresponding models operate on romanized texts, and there is no romanization because the texts are already in the English alphabet. However, in order to make the model rank between different datasets comparable, we need to make sure that every dataset has the same number of models. To do this, we simply fill the missing values in romanized models with their corresponding ones for English. As a result, all datasets have 37 models to rank.

For each dataset, we rank all of the models in ascending order of their development errors. The rank is the index of the model in this ordering. As a result, the smaller the rank, the better the model performs. Then, we compute the minimum, first quartile, median, third quartile, and maximum rank across different datasets for each model and put these numbers as a box plot in Figure 3.2. The numbers

in Figure 3.2 indicate both how each models perform on average, and how stable these models are across different datasets and languages.

From the results, the model achieved the best consistent performance is the character-level 5-gram fastText [Joulin et al., 2016] model. The result is more apparent in table 3.11, where for almost all Chinese, Japanese and Korean datasets the best encoding is character-level 5-gram for fastText. For English, the best encoding is often word n-grams, although character-level 5-gram models are quite competitive as well. Character-level encoding with number of grams less than 5 are significantly worse, with the worst being bag-of-character linear model with TFIDF features. Word-level n-grams feature for both linear models and fastText are competitive, although our data processing pipeline did not guarantee perfect word segmentation for CJK languages because of the segmenters used.

Convolutional networks consistently have the best stability across different datasets and languages, with the best being byte-level large OnehotNet. This suggests that handling different language at byte-level regardless of whether characters could span multiple bytes is quite a feasible solution for handling different languages in a unified fashion. What is better is that byte-level language processing requires the least amount of pre-processing – just present UTF-8 encoded strings to the model. Therefore, we believe byte-level model is a promising approach towards applying deep learning to natural language processing.

Finally, many different models have hit rank 1 as their minimum, suggesting that there is no single best models across different datasets and languages. However, this is limited to the model hyperparameters we chose. It is worth noting that hyperparameters are more thoroughly explored for fastText than other models in this chapter.

3.6.2 Generalization

In this section, we look at the generalization gap – the expected difference between training and testing errors – of different models. The generalization gap in this chapter is approximated by the subtraction of the training error from the development error. The approximation to the underlying sample distribution should be pretty accurate because all our datasets are very large.

As an example, Figure 3.3 visualizes the generalization gap for the Joint binary dataset. This figure exemplifies typical generalization properties of different models for all of our datasets. Additionally, Figure 3.4 offers a box plot for the rankings on generalization error, computed in the same way as Figure 3.2 for development errors.

From these figures, one could easily observe that fastText [Joulin et al., 2016] tends to overfit much more aggressively than either convolutional networks or our own implementation of linear models, in spite of our effort in hyper-parameter tuning. Also, it overfits more using richer features as the number of grams goes from 1 to 5. Given the theoretical fact that fastText could not have more representation capacity than a linear model, this could be a result of the lack of regularization and the aggressive optimization strategy in fastText.

3.6.3 Training Time

The training times of different models vary greatly in our experiments. Table 3.12 offers an estimation of time it took for each model to go over 1,000,000 samples with the hardware mentioned in the previous section. In general, fastText [Joulin et al., 2016] offers the best training time and only requires CPUs,

whereas convolutional networks take the longest time and require GPUs. Depending on the methods of encoding, the performance between convolutional networks also differ drastically, with EmbedNet tens of times faster than GlyphNet. Figure 3.5 visualizes the estimations as a bar chart.

These results show that fastText [Joulin et al., 2016] offers the fastest training and evaluation while achieving competitive results. On the other hand, models using convolutional networks consume the most amount of computation time. As a result, in this chapter we could afford to do hyper-parameter tuning for fastText but not on convolutional networks.

The convolutional network models in this chapter are designed not for achieving the best performance, but for the fairness of comparing between different encoding mechanisms within the computational budget we possess. Given the fact that different designs of convolutional networks could offer drastically different performance, we believe there is a great deal of potential for improvement from different design choices on convolutional networks.

It is also worth noting that the task in question – text classification – is quite simple. Convolutional networks may not show an advantage in this specific task, but may become more useful for more complicated reasoning tasks concerning text inputs and outputs. The comparison between different encoding mechanisms presented this chapter indicate that byte-level encoding is an advantageous choice for convolutional networks due to its generality across languages, simplicity in pre-processing and high performance in results.

3.6.4 Influence from Representation

The representation mechanisms in this chapter affect the results in significant ways, although differently depending on how comparisons are made. Comparing between different representation mechanisms for convolutional networks, one conclusion from figure 3.2 is that GlyphNet – which uses images of characters – gives moderate performance while require tremendous amount of computation. In reality, such models might not be practical for use in real problems.

Comparing between OnehotNet and EmbedNet that use the same level of encoding (byte and romanized character levels), OnehotNet consistently produces better results. The reason here is because the encoder used for OnehotNet is a 4-layer convolutional network, while in EmbedNet it is simply a look-up table which is equivalent of using one-hot encoding with a linear layer. As a result, OnehotNet is able to start learning the contextual patterns and associations of byte or character sequences from earlier stages, resulting in an advantage of its representational capacity.

The advantage of fastText over our own implementation linear models is quite intriguing, because in theory fastText models do not have more representational capacity than its linear counterparts, despite the fact that it is a 2-layer model. To exclude the possibility that optimization is an issue, figure 3.6 on the error and loss values shows that our optimization process as detailed in previous sections achieved acceptable optimality. This is by the fact that further reducing the learning rate and training do not improve the loss values in any significant way in later stages of optimization. As a result, the only possible reason is due to limiting the number of entities (character, words, or n-grams) to a maximum of 1,000,000 in the linear models. For fastText, the model can consider all of the possible entities, and they

are hashed into 10,000,000 bins. That essentially results in more representational capacity than our linear models.

Finally, when comparing between plain and TFIDF features for linear models in table 3.10, TFIDF features seem to always produce results that are a little bit better than plain ones.

3.6.5 Linguistic Properties

Besides the machine learning and computational factors discussed in previous analysis, influence from linguistic properties of these languages is also present in the development error rank plot in figure 3.2. It is known that for some texts in Chinese and Japanese, word segmentation is an inaccurate process because these texts do not have clear word boundaries. In this case, segmenting the text into words has ambiguity in both linguistic and modeling factors. One example of linguistic factor is the granularity of segmented words, which is represented by different people's opinion on what is considered as a word in a word-segmentation dataset. On the other hand, modeling ambiguity comes from the intrinsic assumptions of different word segmentation models, such as the probabilistic sequential dependencies considered in Markov models or conditional random fields (CRFs) – the most popular word or morpheme segmentation models used today for CJK.

As a result of these factors, word segmentation becomes a process that introduces ambiguity. This could be the reason behind the difference in the results for EmbedNet and fastTest models in table 3.9 and table 3.11 respectively. Note however, for word-level EmbedNet this problem is also compounded by information loss due to limiting the words to 200,000 most frequent ones. If we look at the difference between low level (character or byte levels) and word-level mod-

els, when they are given enough representational capacity, low-level models tend to work better because they do not lose information through word segmentation. By "given enough representational capacity", we refer to the fact that EmbedNet models are large and deep, and that the fastText character-level models worked well with enough length for the grams.

Besides the word-segmentation process, another source that introduces ambiguity and loses information is the romanization process. For example, in Chinese many characters have multiple possible pronunciations, which is usually discerned by the surrounding contexts via word segmentation. Furthermore, because the decision between multiple pronunciations relies on word segmentation, the loss of information is actually compounded by segmentation ambiguity. This is why compared among the different results in one series of models, romanized ones usually have the worst performance, in both low-level and word-level models. Readers can refer to the results of byte-level and romanized character-level models in OnehotNet and EmbedNet in table 3.8 and table 3.9 for these effects, and word-level and romanized word-level models in EmbedNet and fastText in table 3.9 and table 3.11.

3.7 Other Models

In spite of the 473 models we have benchmarked, this chapter is in no way a complete essay on every possible model for text classification. Some of the interesting models we did not benchmark include recurrent networks, the use of sparse convolutions for text, and different variations of convolutional architectures.

By focusing on different encoding mechanisms for deep learning models, this

chapter performs experiments only on one kind – convolutional networks. Another often-used kind for processing texts is recurrent networks, constructed using different types of cells like long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] and gated recurrent units (GRU) [Cho et al., 2014a]. Some authors have found that recurrent networks applied to different levels of encoding can offer good results for text classification as well (for example, [Dai and Le, 2015] and [Liu et al., 2016]). Combinations of convolutional networks and recurrent networks are also explored for text classification (for example, [Xiao and Cho, 2016]).

This chapter explores one-hot encoding for convolutional networks using byte-level encoding and romanization. Another alternative is to implement a convolutional module that can take sequences of indices instead of explicit vectors to represent one-hot encoding. This would avoid the memory overflow problem when applying one-hot encoding to large vocabularies. However, so far there has been no deep learning toolkit that has implementation of such a sparse convolutional module. Furthermore, it may require special numerical optimization that would merit its own essay. Therefore, it is not included for presentation in this chapter.

Finally, the results on convolutional networks in this chapter are limited to the purpose of offering fair comparisons between different encoding mechanisms. Another dimension of exploration is the design variants of convolutional networks for text processing, such as very deep networks [Conneau et al., 2017], residual [He et al., 2016] and dense [Huang et al., 2016] connections, and advanced pooling schemes for handling the variable length problem [Kalchbrenner et al., 2014] [Johnson and Zhang, 2017]. We are optimistic that exploration of all these different architecture designs could improve the results further for convolutional networks.

3.8 Conclusion

This chapter explores the use of different encoding mechanisms for both deep learning and linear models for text classification in Chinese, English, Japanese and Korean. These encoding mechanisms include one-hot encoding, embedding and images of character glyphs. Different levels of encoding are applied to each mechanism whenever application, including UTF-8 encoded bytes, characters, words, romanized characters and romanized words. There are in total 473 models benchmarked in this chapter, including convolutional networks, linear models and fastText [Joulin et al., 2016].

A total of 14 large-scale datasets were built in this chapter for benchmarking these models in 4 languages including Chinese, English, Japanese and Korean. Most of these datasets have millions of samples for training, and 2 of these datasets contains samples mixed in all these 4 languages to testing different model's ability to handle different languages in a fashion that generalizes over languages.

Some conclusions from these results are:

1. fastText [Joulin et al., 2016] has the best result with character-level n-gram encoding for Chinese, Japanese and Korean texts. For English, the best encoding for fastText is word-level n-grams.

2. Word-level encoding for CJK languages are competitive even without perfect segmentation, for both fastText and linear models.

3. The best encoding mechanism for convolutional networks is byte-level one-hot encoding. This indicates that convolutional networks have the ability to understand text from a low-level representation, and offers great simplicity for handling multiple languages in a consistent and unified fashion.

4. fastText tends to overfit more than convolutional networks, in spite of the fact that it does not have more representation capacity than a linear model.

In the future, we hope to extend the results to recurrent networks, and explore how different designs of convolutional networks would affect the results. We plan to release all the source code used for all the benchmarks, and hope that these results are useful for the community to choose which encoding mechanism to use when facing with multi-lingual text processing.

Model	Levels	Variant	Time	Day-hh:mm:ss
GlyphNet	Character	Large	136,250	1-13:50:50
		Small	63,125	17:32:05
OnehotNet	Byte Romanized	Large	5,906	1:38:26
		Small	5,331	1:28:51
EmbedNet	Character	Large	2,143	35:43
		Small	1,599	26:39
	Byte Romanized	Large	1,829	30:29
		Small	1,417	23:37
	Word Romanized Word	Large	1,844	30:44
		Small	1,536	25:36
Linear	Character	1-gram	1,518	25:18
		5-gram	7,647	2:07:27
	Word	1-gram	1,417	23:37
		5-gram	6,250	1:44:10
	Romanized Word	1-gram	1,333	22:13
		5-gram	6,253	1:44:13
fastText	Character	1-gram	7	7
		2-gram	7	7
		5-gram	10	10
	Word	1-gram	2	2
		2-gram	3	3
		5-gram	5	5
	Romanized Word	1-gram	3	3
		2-gram	3	3
		5-gram	5	5

Table 3.12: Estimated training time for going over 1,000,000 samples using joint binary dataset. The time estimation in the fourth column is in seconds. Encoding levels that will give the identical models are grouped together because the time estimation would be the same. These estimations are only for reference and may vary depending on actual computing environment.

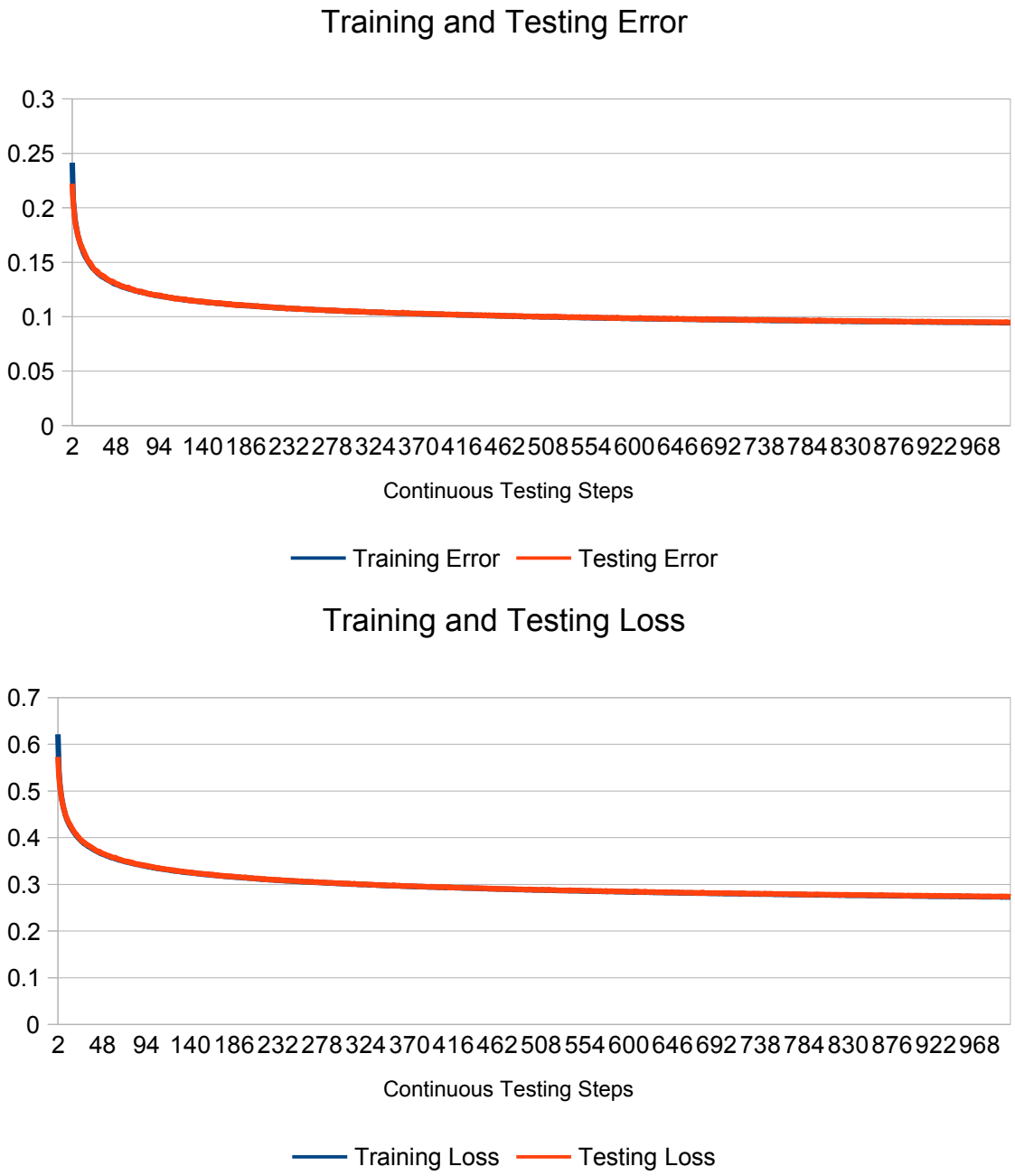


Figure 3.6: Error and loss values for training linear model on jdbinary dataset using word-level 5-gram plain features

Chapter 4

Byte-level Recursive Convolutional Auto-Encoder

This chapter proposes a byte-level auto-encoder architecture for text using convolutional networks with a recursive component. The proposed model is a multi-stage deep convolutional encoder-decoder framework using residual connections [He et al., 2016], containing up to 160 parameterized layers. Each encoder or decoder contains a shared group of modules that consists of either pooling or up-sampling layers, making the network recursive in terms of abstraction levels in representation. Results for 6 large-scale paragraph datasets are reported, in 3 languages including Arabic, Chinese and English. Analyses are conducted to study several properties of the proposed model. Experiments are presented to verify that the auto-encoder can learn useful representations.

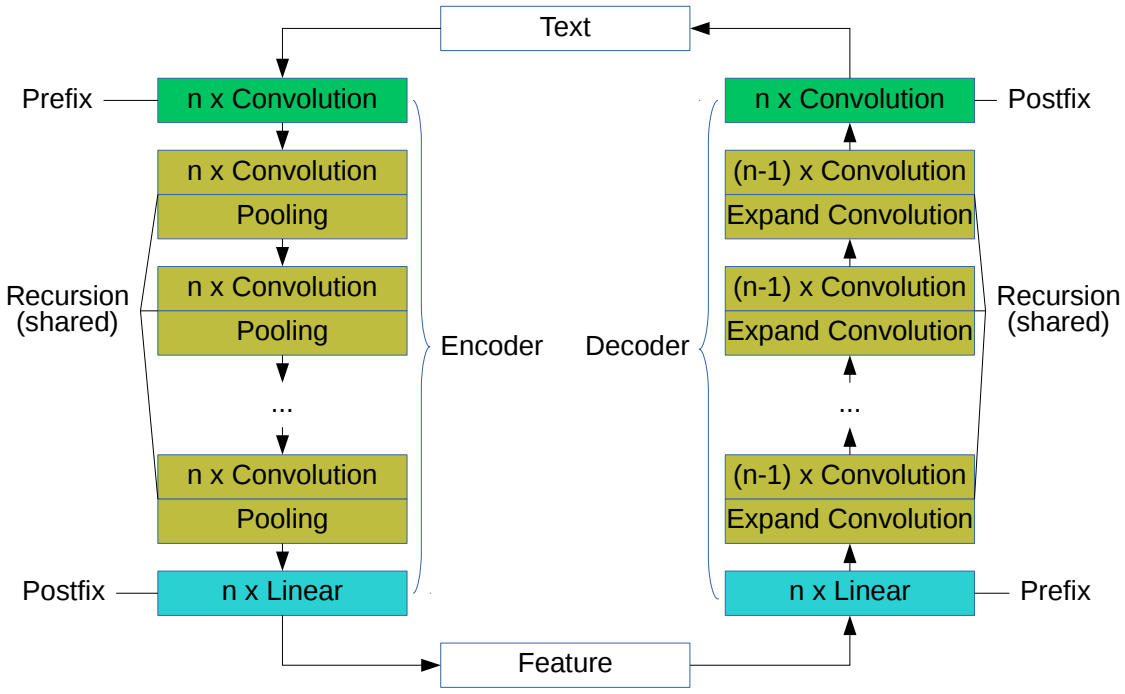


Figure 4.1: The autoencoder model

4.1 Introduction

Recently, generating text using convolutional networks (ConvNets) starts to become an alternative to recurrent networks for sequence-to-sequence learning [Gehring et al., 2017]. The dominant assumption for both these approaches is that texts are generated one word at a time. Such sequential generation process bears the risk of output or gradient vanishing or exploding problem [Bengio et al., 1994] [Hochreiter et al., 2001], which limits the length of its generated results. Such limitation in scalability prompts us to explore whether non-sequential text generation is possible.

Meanwhile, text processing from lower levels than words – such as characters [Zhang et al., 2015] [Kim et al., 2016] and bytes [Gillick et al., 2016] [Zhang and

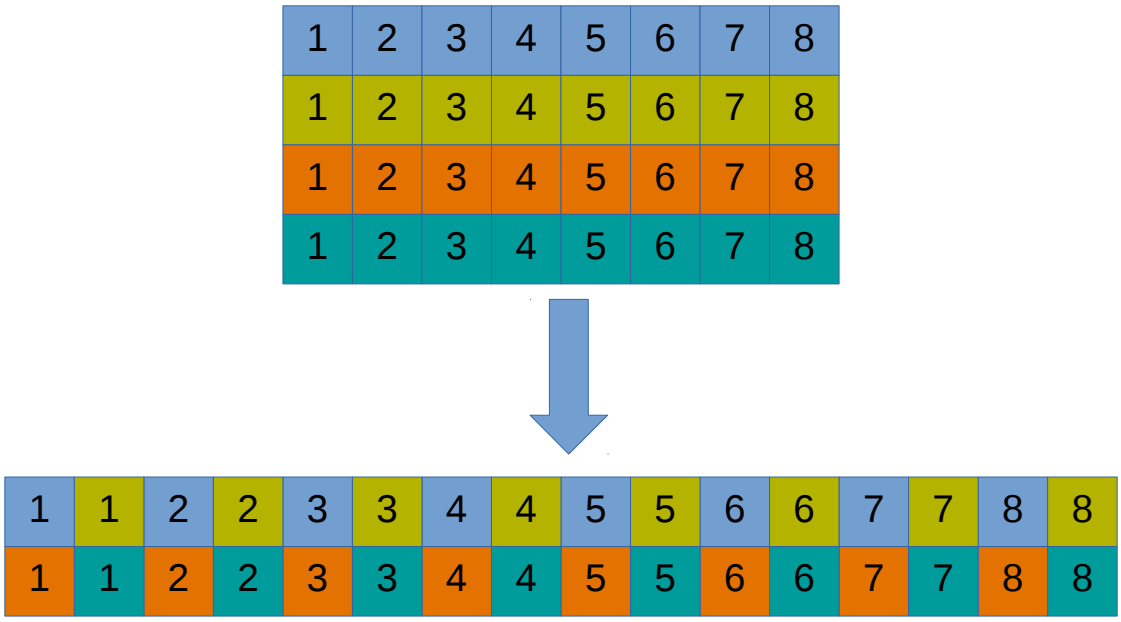


Figure 4.2: The reshaping process. This demonstrates the reshaping process for transforming a representation of feature size 4 and length 8 to feature size 2 and length 16. Different colors represent different source features, and the numbers are indices in length dimension.

LeCun, 2017] – is also being explored due to its promise in handling distinct languages in the same fashion. In particular, the work by Zhang and LeCun [Zhang and LeCun, 2017] shows that simple one-hot encoding on bytes could give the best results for text classification in a variety of languages, as far as convolutional networks are concerned. This is because byte-level one-hot encoding achieved the best balance between computation and classification accuracy.

Inspired by these results, this chapter explores auto-encoding of text using byte-level convolutional networks that has a recursive structure. Our main contribution is the network design, which paves the way towards low-level and non-sequential text generation. Several properties of the model are also presented. Furthermore, using the task of text classification, we show that the learnt representation from

Table 4.1: Datasets. All the numbers are paragraphs.

NAME	TRAIN	DEV	VAL	LANGUAGE
enwiki	41,256,261	2,291,946	2,291,947	English
hudong	53,675,117	2,999,960	2,999,960	Chinese
argiga	27,989,646	1,558,359	1,558,360	Arabic
engiga	116,456,520	6,484,485	6,484,485	English
zhgiga	38,094,390	2,118,821	2,118,822	Chinese
allgiga	182,540,556	10,161,766	10,161,766	Multi-lingual

auto-encoding is useful for improving supervised learning as well, especially when using a small dataset with the models of this size from the level of bytes.

The properties studies in this chapter include the following: 1) comparison with recurrent networks, 2) end-of-sequence symbol generation, 3) avoiding degenerating into the identity function, 4) influence of sample lengths on errors, 5) pooling layer variations, 6) the advantage of the recursive structure, and 7) model depth variations.

For the task of text auto-encoding, we should avoid the use of common attention mechanisms like those used in machine translation [Bahdanau et al., 2015], because they always provide a direct information path that enables the auto-encoder to directly copy from the input. This diminishes the purpose of studying the representational ability of different models. Therefore, all models considered in this chapter encode to and decode from a fixed-length vector representation.

The paper by Zhang et al. [Zhang et al., 2017b] was an anterior result on using word-level convolutional networks for text auto-encoding. This chapter differs in several ways of using convolutional networks, such as encoding level (bytes instead of words) and the recursive structure, which by design could prevent trivial

Table 4.2: Training, development and validation byte-level errors

DATASET	LANGUAGE	TRAIN	DEV	VAL
enwiki	English	3.34%	3.35%	3.34%
hudong	Chinese	3.21%	3.15%	3.17%
argiga	Arabic	3.08%	3.10%	3.08%
engiga	English	2.09%	2.09%	2.08%
zhgiga	Chinese	5.11%	5.24%	5.25%
allgiga	Multi-lingual	2.48%	2.50%	2.50%

solutions such as the identity function. We also use one of the latest network design heuristics – residual connections [He et al., 2016] – so that our network can scale up to several hundred of layers deep.

Recently, the concept of "non-autoregressive" text generation has been proposed for machine translation [Gu et al., 2018]. It is the same as the concept of "non-sequential" text generation in this chapter, meaning to generate items – be it bytes, characters or words – all at once, rather than one after another.

4.2 Recursive Convolutional Auto-Encoder

In this section, we introduce the design of the convolutional auto-encoder model with a recursive structure. The model consists of 6 groups of modules, with 3 for the encoder and 3 for the decoder. The model first encodes a variable-length input into a fixed-length vector of size 1024, then decodes back to the same input length. The decoder architecture is a reverse mirror of the encoder. All convolutional layers in this chapter have zero-padding added to ensure that each convolutional layer outputs the same length as the input. They also all have feature size 256 and

Table 4.3: Byte-level errors for long short-term memory (LSTM) recurrent network

DATASET	LANGUAGE	TRAIN	DEV	VAL
enwiki	English	67.71%	67.78%	67.82%
hudong	Chinese	64.47%	64.55%	67.58%
argiga	Arabic	61.23%	61.30%	61.29%
engiga	English	70.47%	70.45%	70.45%
zhgiga	Chinese	75.91%	75.88%	75.94%
allgiga	Multi-lingual	72.39%	72.44%	72.55%

kernel size 3. All parameterized layers in our model use ReLU [Nair and Hinton, 2010] as the non-linearity.

In the encoder, the first group of modules consist of n temporal (1-D) convolutional layers. It accepts an one-hot encoded sequence of bytes as input, where each byte is encoded as a 256-dimension vector. This first group of modules transforms the input into an internal representation. We call this group of modules the prefix group. The second group of modules consists of n temporal convolutional layers plus one max-pooling layer of size 2. This group reduces the length of input by a factor of 2, and it can be applied again and again to recursively reduce the representation length. Therefore, we name this second group the recursion group. The recursion group is applied until the size of representation becomes 1024, which is actually a feature of dimension 256 and length 4. Then, following the final recursion group is a postfix group of n linear layers for feature transformation.

The decoder is a symmetric reverse mirror of the encoder. The decoder prefix group consists of n linear layers, followed by a decoder recursion group that expand the length of representation by a factor of 2. This expansion is done at the first convolutional layer of this group, where it outputs 512 features that will be reshaped

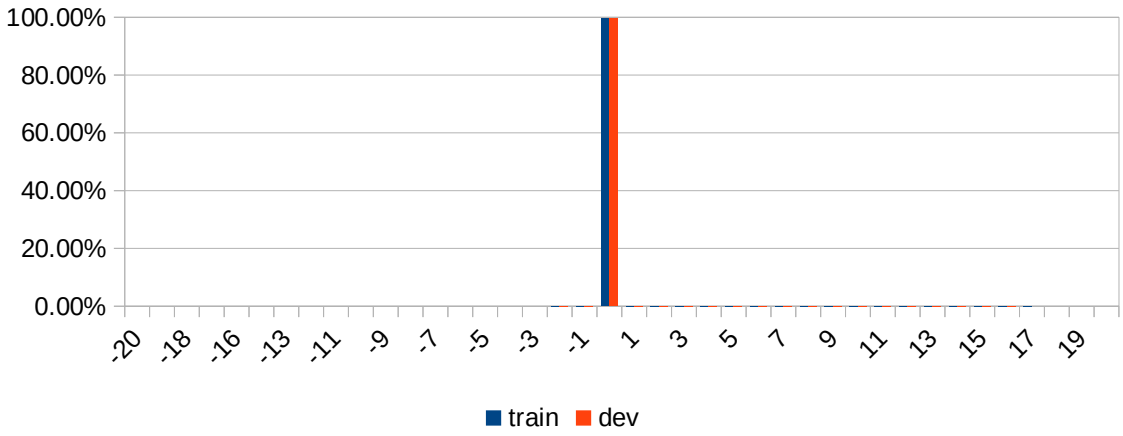


Figure 4.3: The histogram of length difference

into 256 features. The reshaping process we use ensures that feature values correspond to nearby field of view in the input, which is similar to the idea of sub-pixel convolution (or pixel shuffling) [Shi et al., 2016]. Figure 4.2 depicts this reshaping process for transforming representation of feature size 4 and length 8 to feature size 2 and length 16. After applying this recursion group several times (same as that of the encoder recursion group), a decoder postfix group of n convolutional layers is applied to decode the recursive features into a byte sequence.

The final output of the decoder is interpreted as probabilities of bytes after passing through a softmax function. Therefore, the loss we use is simply negative-log likelihood on the individual softmax outputs. It is worth noting that this does not imply that the output bytes are unconditionally independent of each other. For our non-sequential text decoder, the independence between output bytes is conditioned on the representation from the encoder, meaning that their mutual dependence is modeled by the decoder itself.

Depending on the length of input and size of the encoded representation, our model can be extremely deep. For example, with $n = 8$ and encoding dimension

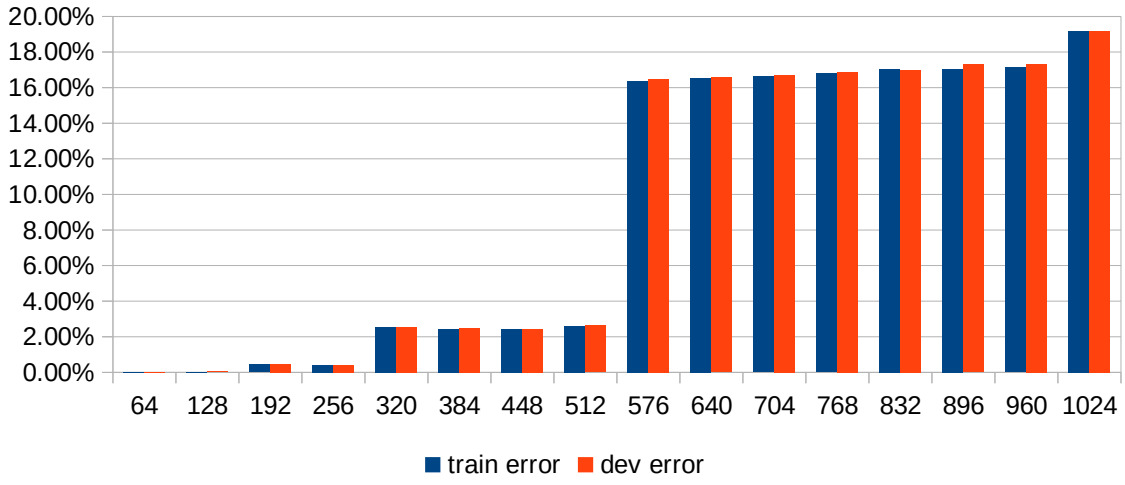
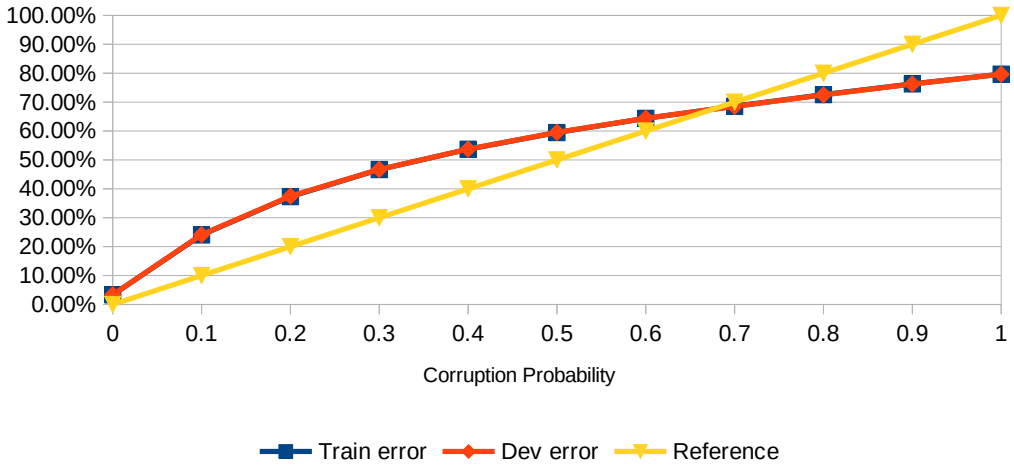


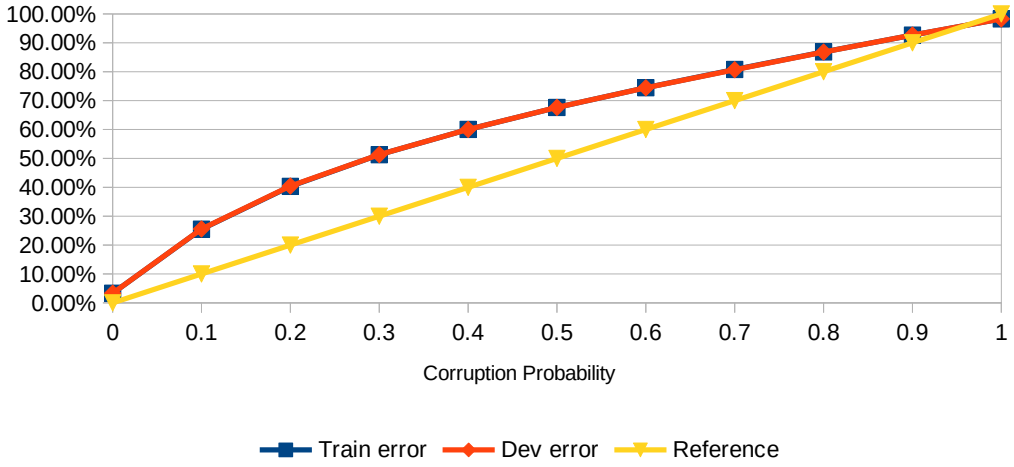
Figure 4.4: Byte-level error by length

1024 (reduced to a length 4 with 256 features), for a sample length of 1024 bytes, the entire model has 160 parameterized layers. Training such a deep dynamic model can be very challenging using stochastic gradient descent (SGD) due to the gradient vanishing problem [Bengio et al., 1994] [Hochreiter et al., 2001]. Therefore, we use the recently proposed idea of residual connections [He et al., 2016] to make optimization easier. For every pair of adjacent parameterized layers, the input feature representation is passed through to the output by addition. We were unable to train a model designed in this fashion without such residual connections.

For all of our models, we use an encoded representation of dimension 1024 (recursed to length of 4 with 256 features). For an input sample of arbitrary length l, we first append the end-of-sequence "null" byte to it, and then pad it to length $2^{\lceil \log_2(l+1) \rceil}$ with all zero vectors. This makes the input length a base-2 exponential of some integer, since the recursion groups in both encoder and decoder either reduce or expand the length of representation by a factor of 2. If $l < 4$, it is padded to size of 4 and does not pass through the recursion groups. It is easy to



(a) Groundtruth



(b) Mutated

Figure 4.5: Byte-level errors with respect to randomly mutated samples

see that the depth of this dynamic network for a sample of length l is on the order of $\log_2 l$, potentially making the hidden representations more efficient and easier to learn than recurrent networks which has a linear order in depth.

4.3 Result for Multi-lingual Auto-Encoding

In this section, we show the datasets and the results.

Table 4.4: Byte-level errors for different pooling layers

POOL	TRAIN	DEV	VAL
max	3.34%	3.35%	3.34%
average	7.91%	7.98%	7.97%
L2	6.85%	6.77%	6.76%

4.3.1 Dataset

All 6 of our large-scale datasets in 3 languages – Arabic, Chinese and English – are at the level of paragraphs. No pre-processing other than raw text extraction is applied to them since our model can be applied to all languages in the same fashion – an advantage of byte-level text processing. We also constructed a dataset with samples mixed in all three languages to test the model's ability to handle multilingual data. Table 4.1 is a summary of these datasets.

enwiki. This dataset contains paragraphs from the English Wikipedia [1], constructed from the dump on June 1st, 2016. We were able to obtain 8,484,895 articles, and then split our 7,634,438 for training and 850,457 for testing. The number of paragraphs for training and testing are therefore 41,256,261 and 4,583,893 respectively. The testing dataset is then split into 2 equal sized development and validation datasets.

hudong. This dataset contains pragraphs from the Chinese encyclopedia website `baike.com` [2]. We crawled 1,799,095 article entries from it and used 1,618,817 for training and 180,278 for testing. The number of paragraphs for training and testing are 53,675,117 and 5,999,920. The testing dataset is then split into 2 equal

[1] `https://en.wikipedia.org`
[2] `http://www.baike.com/`

Table 4.5: Byte-level errors for recursive and static models

MODEL	TRAIN	DEV	VAL
recursive	3.34%	3.35%	3.34%
static	8.01%	8.05%	8.06%

sized development and validation datasets.

argiga. This dataset contains paragraphs from the Arabic Gigaword Fifth Edition release [Parker et al., 2011a], which is a collection of Arabic newswire articles. In total there are 3,346,167 articles, and we use 3,011,403 for training and 334,764 for testing. As a result, we have 27,989,646 paragraphs for training and 3,116,719 for testing. The testing dataset is then split into 2 equal sized development and validation datasets.

engiga. This dataset contains paragraphs from the English Gigaword Fifth Edition release [Parker et al., 2011c], which is a collection of English newswire articles. In total there are 9,876,096 articles, and we use 8,887,583 for training and 988,513 for testing. As a result, we have 116,456,520 paragraphs for training and 12,969,170 for testing. The testing dataset is then split into 2 equal sized development and validation datasets.

zhgiga. This dataset contains paragraphs from the Chinese Gigaword Fifth Edition release [Parker et al., 2011b], which is a collection of Chinese newswire articles. In total there are 5,664,377 articles, and we use 5,097,198 for training and 567,179 for testing. As a result, we have 38,094,390 paragraphs for training and 4,237,643 for testing.

allgiga. Since the three Gigaword datasets are very similar to each other, we combined them to form a multi-lingual dataset of newswire article paragraphs. In

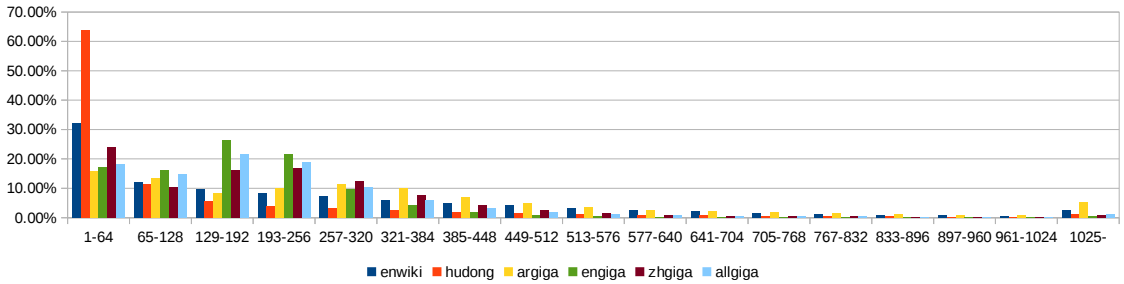


Figure 4.6: Histogram of sample frequencies in different lengths

this dataset, there are 18,886,640 articles with 16,996,184 for training and 1,890,456 for testing. The number of paragraphs for training and testing are 182,540,556 and 20,323,532 respectively. The testing dataset is then split into 2 equal sized development and validation datasets.

4.3.2 Result

For our large-scale datasets, testing time could be prohibitively long. Therefore, we report all the results based on 1,000,000 samples randomly sampled from either training or development subsets depending on the scenario. Very little overfitting was observed even for our largest model.

We set the initial learning rate to 0.001, half it every 10 epoches, and stop training at the 100th epoch. Each training epoch contains 1,000,000 randomly selected samples. A momentum of 0.9 is applied to speed up training. A small weight decay of 0.00001 is used to stabilize training. We find that dividing the gradients of the recursion groups by the number of times they were applied can speed up training. Because engiga and allgiga datasets have more than 100,000,000 training samples, when training stops the model has not seen the entirety of training data. However, further training does not achieve any observable improvement.

Table 4.2 details the byte-level errors for our model on all of the aforemen-

Table 4.6: Byte-level errors depending on model depth

n	DEPTH	TRAIN	DEV	VAL
2	40	9.05%	9.07%	9.07%
4	80	5.07%	5.11%	5.11%
8	160	3.34%	3.35%	3.34%
16	320	2.91%	2.93%	2.92%

tioned datasets. These results indicate that our models can achieve good results on different languages. The result for allgiga dataset also indicates that the model has no trouble in learning from multi-lingual datasets.

4.4 Analysis

This section offers comparisons with recurrent networks, and studies different properties of our proposed auto-encoding model. Most of these results are performed using the enwiki dataset.

4.4.1 Comparison with Recurrent Networks

We constructed a simple baseline recurrent network using the "vanilla" long short-term memory units [Hochreiter and Schmidhuber, 1997]. In this model, both input and output bytes are embedded into vectors of dimension 1024. The hidden representation is also of dimension 1024. The encoder reads the text in reverse order, which is observed to improve quality of outputs [Sutskever et al., 2014]. The 1024-dimension hidden output of the last cell is used as the input for the decoder.

Figure 4.7: Errors during training for recursive and static models.

During decoding, the most recently generated byte is fed to the next time step. This was observed to improve generation compared to only relying on transforming the hidden representation. The decoding process uses beam search of size 2.

Table 4.3 details the result for LSTM. The results of our models in table 4.2 are better by at least one order of magnitude. The limitation of recurrent networks is the gradient or output vanishing problem, as a result of which it can easily fail on auto-encoding sequences beyond a few tens of items. The recursive non-sequential decoding process in this chapter could be a better alternative.

4.4.2 End of Sequence

For sequential generative process such as recurrent decoders, we could stop the generation when some end-of-sequence symbol is generated. However, our model generates in a non-sequential fashion therefore this does not apply. One simple solution is to regard the first encountered end-of-sequence symbol as the end, but it will inevitably generate some extra symbols.

To show whether this naive approach is effective, we computed the difference of end-of-sequence symbols between generated text and its groundtruth for 1,000,000 samples, for both the training and development subsets of the enwiki dataset. We

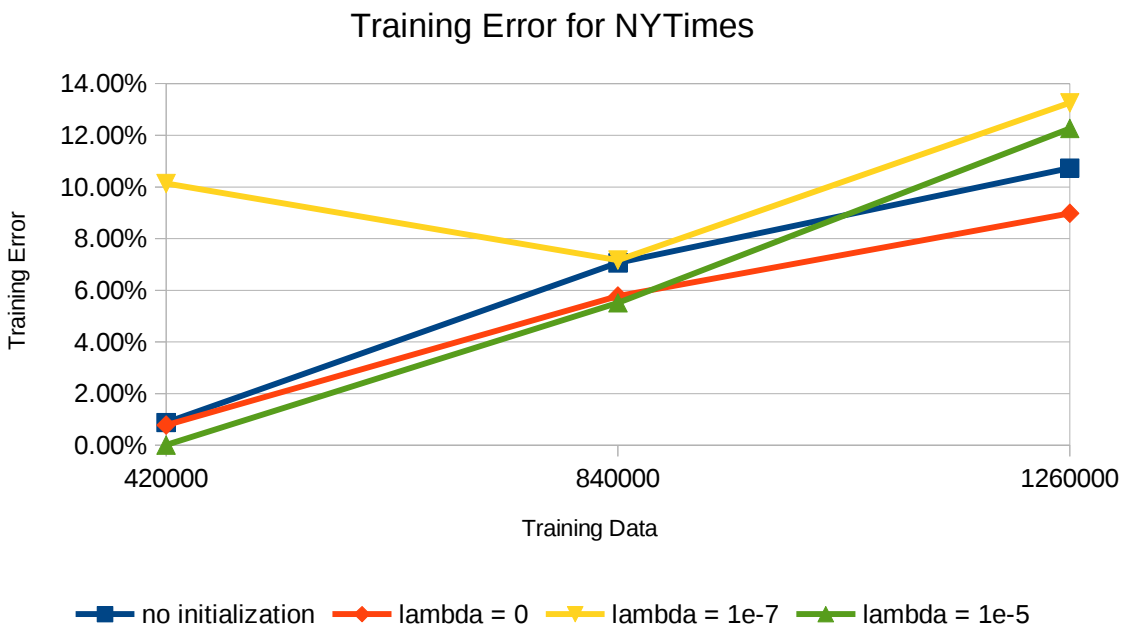


Figure 4.8: Training Error for NYTimes dataset

discovered that the distribution concentrated at 0, at 99.63% for both training and development datasets. Figure 4.3 shows the full histogram, in which length differences other than 0 is barely visible. This suggests that our non-sequential generation process can model the end-of-sequence symbol accurately.

4.4.3 Random Permutation of Samples

One problem specific to the task of auto-encoding is the risk of learning the identity function as the degenerated solution. One way to test this is to mutate the input bytes randomly and see whether the error rates match with the mutation probability. We experimented with mutation probability from 0 to 1 with an interval of 0.1, and for each case we tested the byte-level errors for 100,000 samples in both training and development subsets of the enwiki dataset.

We can compute the byte-level errors in 2 ways – with respect to the groundtruth

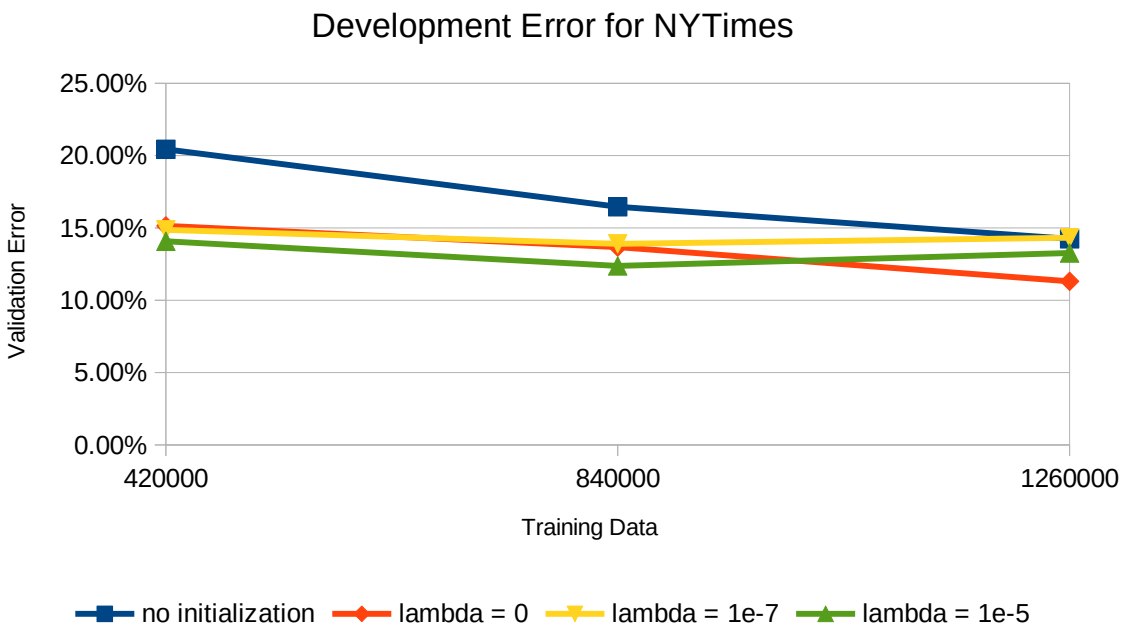


Figure 4.9: Development Error for NYTimes dataset

samples, or with respect to the mutated samples. Figure 4.5 shows the results. If the solution is degenerated to the identity function, then the byte-level errors with respect to the groundtruth should correlate with the probability of mutation, and the byte-level errors with respect to the mutated samples should be near 0 regardless of the mutation probability. None of these happen in figure 4.5.

It is worth noting that the errors with respect to the groundtruth samples in figure 4.5 also demonstrate that our model lacks the ability to denoise mutated samples. This can be seen from the phenomenon that the errors for each mutation probability is higher than the reference diagonal value, instead of lower. This is due to the lack of a denoising criterion in our training process.

4.4.4 Sample Length

We also conducted experiments to show how the byte-level errors vary with respect to the sample length. Figure 4.6 shows the histogram of sample lengths for all datasets. It indicates that a majority of paragraph samples can be well modeled under 1024 bytes. Figure 4.4 shows the byte-level error of our models with respect to the length of samples. This figure is produced by testing 1,000,000 samples from each of training and development subsets of enwiki dataset. Each bin in the histogram represent a range of 64 with the indicated upper limit.

In figure 4.4, the errors are highly correlated with the number of recursion groups applied to the sample. In the plot, bins 64, 128, 192-256, 320-512, 576-1024 represent recursion levels of 4, 5, 6, 7, 8 respectively. The errors for the same recursion level are almost the same to each other, despite huge length differences when the recursion levels get deep. This result suggests it is advantageous to reduce the functional order of network depth with respect to the sample length. This shows an advantage of recursion – the depth is on a logarithmic order of sample length.

4.4.5 Pooling Layers

This section details an experiment in studying how do the training and development errors vary with the choice of pooling layers in the encoder network. The experiments are conducted on the aforementioned model with $n = 8$, and replacing the max-pooling layer in the encoder with average-pooling or L2-pooling layers. Table 4.4 details the result. The numbers strongly indicate that max-pooling is the best choice. Max-pooling selects the largest values in its field of view, helping

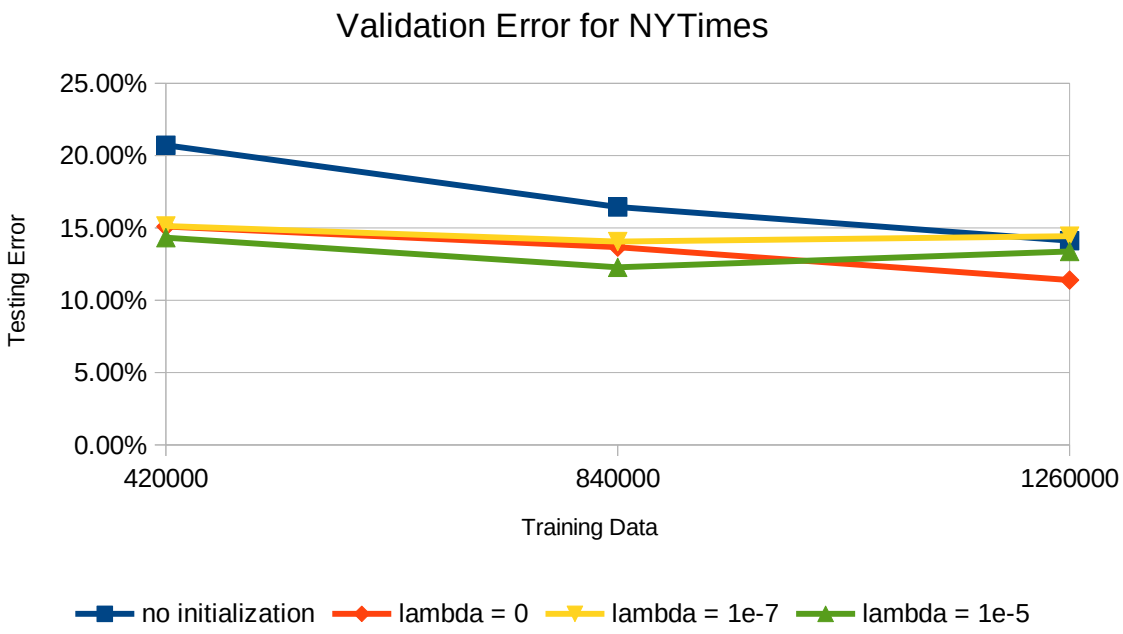


Figure 4.10: Validation Error for NYTimes dataset

the network to achieve a better optimum [Boureau et al., 2010b].

4.4.6 Recursion

The use of recursion in the proposed model is from a linguistic intuition that the structure may help the model to learn better representations. However, there is no guarantee that such intuition could be helpful for the model, unless comparison is done between a recursive model and a static model. Note that the recursive model is essentially a dynamic differentiable program that scales with the length of input.

Figure 4.7 shows the training and development errors when training a static model with the same hyper-parameters. The static model takes 1024 bytes, and zero vectors are padded if the sample length is smaller. The recursion group is therefore applied for 8 times in both the encoder and decoder, albeit their weights

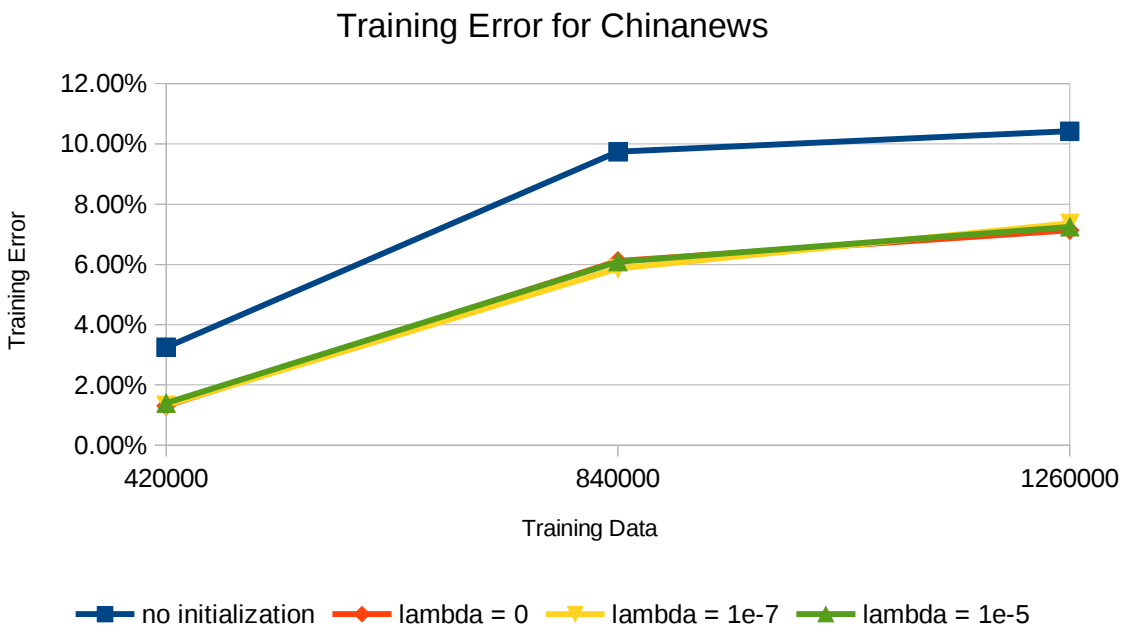


Figure 4.11: Training Error for Chinanews dataset

are not shared. The result indicates that a recursive model not only learns faster, but can also achieve better results. Table 4.5 lists the byte-level errors.

4.4.7 Model Depth

This section explores whether varying the model size can make a different on the result. Table 4.6 lists the training and development errors of different model depths with $n \in \{2, 4, 8, 16\}$. The result indicates that best error rates are achieved with the largest model, with very little overfitting. This is partly due to the fact that our datasets are quite large for the models in question.

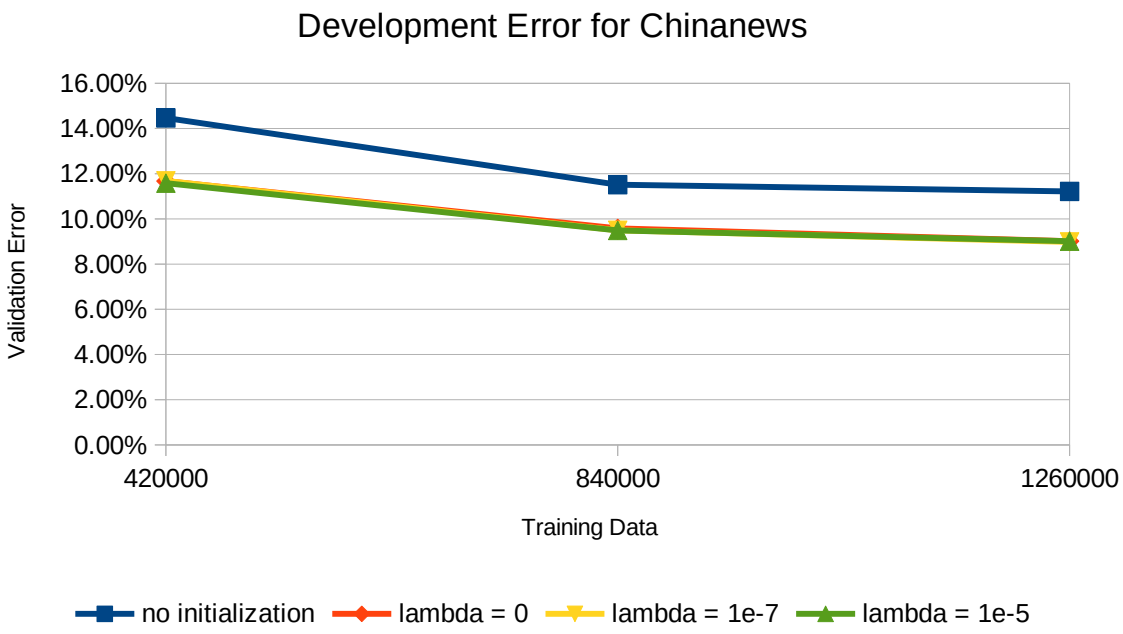


Figure 4.12: Development Error for Chinanews dataset

4.5 Representation Learning for Text Classification

Besides being able to satisfactorily auto-encode text, another potential of the proposed model is the usefulness of its learnt features. To demonstrate this, we present results on a supervised task – text classification. We compare between random initialization and initialization from a pre-trained auto-encoder. Additionally, we also show the effectiveness of regularizing towards a pre-trained auto-encoder.

2 text classification datasets were used – one is in English, and the other Chinese. They are the NYTimes and Chinanews datasets from Zhang and LeCun [Zhang and LeCun, 2017]. Both of these datasets have 1,400,000 training samples equally distributed in 7 classes. We run experiments using 420,000, 840,000 and 1,260,000 training samples respectively, which shows how the model performs

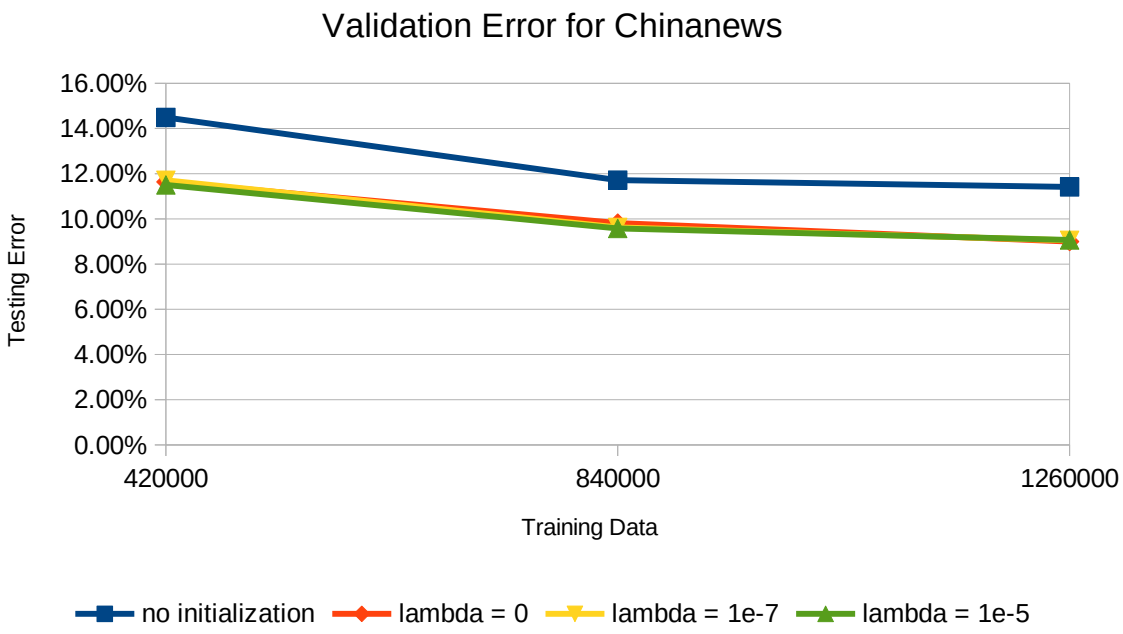


Figure 4.13: Validation Error for Chinanews dataset

under smaller dataset size.

These datasets contain news articles, similar to our engiga and zhgiga datasets. As a result, we choose to use the trained engiga and zhgiga auto-encoders as reference models to initialize our models. The model used for this task is simply the encoder of the auto-encoder plus a 1024-by-7 linear layer for classification. This linear layer is always randomly initialized.

Besides initializing the classifier network with a learnt auto-encoder, we also test whether regularizing towards the pre-trained encoder could help. The regularization objective is

$$\lambda \|w - w_c\|^2, \tag{4.1}$$

in which w represents the parameters of the classifier and w_c represents the parameters of the pre-trained encoder. It has been shown theoretically that such regularization can reduce the generalization gap in a probably approximately cor-

Train Data	420,000			840,000			1,260,000		
Error Type	Train	Dev	Val	Train	Dev	Val	Train	Dev	Val
No Init.	0.88%	20.44%	20.71%	7.06%	16.44%	16.46%	10.73%	14.24%	14.16%
$\lambda = 0$	0.78%	15.15%	15.09%	5.78%	13.65%	13.67%	8.98%	11.32%	11.39%
$\lambda = 10^{-7}$	10.13%	14.86%	15.11%	7.17%	13.89%	14.07%	13.26%	14.30%	14.45%
$\lambda = 10^{-5}$	0.01%	14.07%	14.31%	5.51%	12.36%	12.30%	12.27%	13.25%	13.39%

Table 4.7: Training, development and validation errors for NYTimes dataset.

rect (PAC) sense, if w_c is close to the ideal model [Zhang, 2013]. We tested with $\lambda = 1e-7$ and $\lambda = 1e-5$ respectively. The results are shown in figures 4.8 - 4.10 and 4.11 - 4.13. More detailed numbers are collected in a table in the supplemental material.

From the comparison on whether the auto-encoder is used to initialize the model – the blue line versus others in figure figures 4.8 - 4.10 and 4.11 - 4.13, a significant improvement can be observed. The improvement is more obvious for development and validation errors when there are less training data. Additionally, a small central regularization with $\lambda = 1e-7$ or $\lambda = 1e-5$ can improve the results even further for smaller training data size, but not so much for larger training data size.

Moreover, for both NYTimes and Chinanews datasets we achieved new state-of-the-art classification performance on the validation errors. The previous best model – fastText [Joulin et al., 2016] – was reported to have a validation error of 11.80% for NYTimes and 9.07% for Chinanews [Zhang and LeCun, 2017]. Our model initialized from the parameters of a pre-trained auto-encoder without central regularization ($\lambda = 0$) achieved 11.40% and 9.00% respectively, using the largest training data size 1,260,000.

Train Data	420,000			840,000			1,260,000		
Error Type	Train	Dev	Val	Train	Dev	Val	Train	Dev	Val
No Init.	3.25%	14.45%	14.49%	9.74%	11.48%	11.76%	10.42%	11.18%	11.45%
$\lambda = 0$	1.30%	11.65%	11.65%	6.12%	9.59%	9.81%	7.14%	9.00%	8.99%
$\lambda = 10^{-7}$	1.33%	11.69%	11.70%	5.86%	9.51%	9.52%	7.37%	8.98%	9.05%
$\lambda = 10^{-5}$	1.40%	11.55%	11.51%	6.09%	9.48%	9.58%	7.25%	9.02%	9.07%

Table 4.8: Training, development and validation errors for Chinanews dataset.

4.6 Conclusion

In this chapter, we propose to auto-encode text using a recursive convolutional network. The model contains 6 parts – 3 for the encoder and 3 for the decoder. The encoder and decoder both contain a prefix module group and a postfix module group for feature transformation. A recursion module group is included in between the prefix and postfix for each of the encoder and decoder, which recursively shrink or expand the length of representation. As a result, the decoder essentially generate text in a non-sequential fashion.

Experiments using this model are done on 6 large scale datasets in Arabic, Chinese and English. Comparison with recurrent networks is offered to show that our model achieved great results in text auto-encoding. Properties of the proposed model are studied, and experiments are conducted to show the usefulness of the learnt representation.

In the future, we hope to use the network architecture designed in this chapter for some actual tasks concerning text generation.

Chapter 5

Model Improvement for Text Classification

This chapter shows that the recent design advancements for byte-level convolutional networks can achieve new state-of-the-art results for text classification. These improvements include residual [Huang et al., 2016] and dense [He et al., 2016] connections, and the use of recursive structure [Zhang and LeCun, 2018b] to build very deep networks. Experiments are conducted using 14 large-scale datasets in Chinese, English, Japanese and Korean [Zhang and LeCun, 2017].

5.1 Introduction

Recently, many authors have proposed to process text from lower levels than words – such as characters [Zhang et al., 2015] [Kim et al., 2016] and bytes [Gillick et al., 2016] [Zhang and LeCun, 2017] [Zhang and LeCun, 2018b], due to its promise in handling distinct languages in the same fashion. In particular, the work by

[Zhang and LeCun, 2017] shows that simple one-hot encoding on bytes could give near state-of-the-art results for text classification in a variety of languages.

Meanwhile, the design of convolutional networks have kept evolving. On one hand, the ideas of residual [He et al., 2016] and dense [Huang et al., 2016] have been dominating the leaderboards for computer vision tasks such as the ImageNet challenge [Russakovsky et al., 2015]. On the other hand, the recursive structure proposed by [Zhang and LeCun, 2018b] has been proven useful for auto-encoding text at byte-level. The model is essentially a dynamic differentiable program that scales with the byte length of each sample.

Inspired by these results, this chapter explores whether these recent model design improvements can help with text classification for byte-level convolutional networks. We designed several deep convolutional networks by combining residual or dense connections, and recursive structure via sub-module sharing. Results on 14 large-scale text classification datasets [Zhang and LeCun, 2017] show that they achieve state-of-the-art classification errors, surpassing the previous best models such as fastText [Joulin et al., 2016] for most of these datasets.

5.2 Recursive Convolutional Networks using Residual or Dense Connections

The convolutional networks used in this chapter are derived from the encoder design in [Zhang and LeCun, 2018b], which achieved high accuracy for auto-encoding text at byte-level.

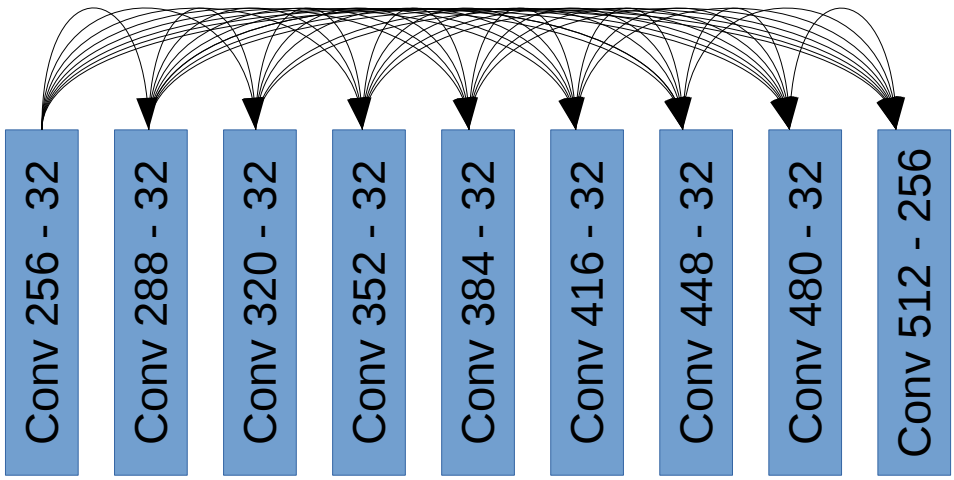


Figure 5.1: The dense block. The arrows represent concatenation operation.

5.2.1 Residual Connections

The recursive convolutional network model is inspired by the encoder part of the auto-encoder architecture in [Zhang and LeCun, 2018b]. The encoder model consists of 3 parts – a prefix, a recursion group, and a postfix. The prefix consists of 8 convolutional layers with input and output feature size 256 and filter size 3. The recursion group consists of 8 convolutional layers of the same configuration, plus a pooling layer of size 2. The recursion group layer is applied again and again until the feature size can be reduced to 1024 dimensions for each sample. This forms a recursive hierarchy of abstraction levels in terms of internal representation. The postfix consists of 8 linear layers with input and output feature size 1024 that performs feature transformation as needed.

In all of the convolutional and linear layers, a threshold function is appended to make them rectified linear units (ReLUs) [Nair and Hinton, 2010]. For every 2 convolutional or linear layers, a residual connection [He et al., 2016] is added to help learning when the recursion levels get deep. In this chapter we also append a

Dataset	Language	#	Train	Dev	Val	Batch
Dianping	Chinese	2	2,000,000	250,000	250,000	100,000
JD full	Chinese	5	3,000,000	125,000	125,000	100,000
JD binary	Chinese	2	4,000,000	180,000	180,000	100,000
Rakuten full	Japanese	5	4,000,000	250,000	250,000	100,000
Rakuten binary	Japanese	2	3,400,000	200,000	200,000	100,000
11st full	Korean	5	750,000	50,000	50,000	100,000
11st binary	Korean	2	4,000,000	200,000	200,000	100,000
Amazon full	English	5	3,000,000	325,000	325,000	100,000
Amazon binary	English	2	3,600,000	200,000	200,000	100,000
Ifeng	Chinese	5	800,000	25,000	25,000	100,000
Chinanews	Chinese	7	1,400,000	56,000	56,000	100,000
NYTimes	English	7	1,400,000	52,500	52,500	100,000
Joint full	Multilingual	5	10,750,000	750,000	750,000	400,000
Joint binary	Multilingual	2	15,000,000	780,000	780,000	400,000

Table 5.1: Datasets. The 3rd column is the number of classes.

linear layer after the postfix to transform the final 1024-dimension representation to the correct number of classes depending on the datasets. We call this final linear layer the classification layer.

5.2.2 Dense Connections

Besides using residual connections, in this chapter we also modify the aforementioned model to use dense connections [Huang et al., 2016]. This is done at the level of layer groups, where each prefix, recursion group and postfix is constructed as a dense block. All the convolutional layers in the dense blocks use kernel size 3, and all parameterized layers use ReLU [Nair and Hinton, 2010] as the non-linearity.

For the prefix and the recursion group, it is done by replacing all the convolutional layers with a 1-D (temporal) dense block with 8 convolutional layers. The first convolutional layer has input feature size 256, and the later convolutional lay-

ers have their features grew 32 each time. Eventually, the final convolutional layer output a feature of size 512. After the dense block, we also add another convolutional layer to turn input feature size 512 to output feature size 256. Figure 5.1 is an illustration of the convolutional dense block in the prefix and the recursion group.

Dataset	Residual		Dense	
	Dynamic	Static	Dynamic	Static
Dianping	22.26%	23.41%	**22.17%**	23.49%
JD f.	46.98%	47.50%	46.85%	**46.77%**
JD b.	8.32%	8.48%	8.33%	**8.29%**
Rakutenf f.	42.28%	42.80%	42.64%	**41.92%**
Rakuten b.	**4.60%**	4.84%	4.83%	4.75%
11st f.	36.65%	37.54%	**32.73%**	36.03%
11st b.	**12.27%**	12.48%	12.50%	12.47%
Amazon f.	38.35%	38.31%	38.91%	**38.16%**
Amazon b.	**4.31%**	4.41%	4.62%	4.47%
Ifeng	17.49%	17.32%	**15.60%**	16.01%
Chinanews	**9.31%**	10.34%	10.15%	9.32%
NYTimes	11.75%	13.10%	13.24%	**11.38%**
Joint f.	**40.24%**	40.32%	41.45%	40.25%
Joint b.	**7.48%**	7.49%	7.78%	7.49%

Table 5.2: Development errors

All of the layers in the postfix are linear layers, but similarly we can construct a linear dense block like that for convolutional layers. The linear dense block is constructed by growing the feature size by 128 for 8 times, beginning with a feature size of 1024. Then, the 2048 feature is turned into a 1024 feature using an additional linear layer.

Dataset	Our Best		Previous Best		Change
	Error	Model	Error	Model	
Dianping	**22.17%**	**Dense - Dynamic**	22.34%	fastText - Character	-0.76%
JD f.	**46.77%**	**Dense - Static**	47.99%	fastText - Character	-2.54%
JD b.	**8.29%**	**Dense - Static**	8.72%	fastText - Character	-4.93%
Rakuten f.	**41.93%**	**Dense - Static**	43.27%	fastText - Character	-3.10%
Rakuten b.	**4.60%**	**Residual - Dynamic**	5.44%	fastText - Roman. Word	-15.44%
11st f.	32.73%	Dense - Dynamic	**32.29%**	**EmbedNet - Character**	1.36%
11st b.	**12.27%**	**Residual - Dynamic**	13.11%	fastText - Character	-6.41%
Amazon f.	38.16%	Dense - Static	**37.00%**	**Very Deep ConvNet**	3.14%
Amazon b.	4.31%	Residual - Dynamic	**4.28%**	**Very Deep ConvNet**	0.70%
Ifeng	**15.60%**	**Dense - Static**	16.31%	fastText - Character	-4.35%
Chinanews	9.31%	Residual - Dynamic	**9.10%**	**fastText - Character**	2.31%
NYTimes	**11.38%**	**Dense - Static**	11.84%	fastText - Word	-3.89%
Joint f.	**40.24%**	**Residual - Dynamic**	42.93%	OnehotNet - Byte	-6.27%
Joint b.	**7.48%**	**Residual - Dynamic**	8.65%	fastText - Word	-13.53%

Table 5.3: Comparison with previous state-of-the-art models

5.2.3 Static (Non-Recursive) Variants

All the aforementioned models in this section build a recursive hierarchy of representation by dynamically applying the recursion groups. This essentially constructs a dynamic differentiable program that scales with the byte length of each sample.

Due to computation limitations, we limit the maximum length that the model accepts to 1024 bytes, which means each recursion group is applied at most 8 times. Note also that the model must accept samples with length equal to $2^i, i \geq 2$. This is done by padding zero vectors

For the comparison purposes, we also experimented on a static differentiable program in which the recursion group is cloned 8 times and not shared, and the input byte-length is fixed to be 1024. We present results for both the dynamic (recursive) and static (non-recursive) models.

5.3 Datasets and Results

This section introduces the datasets we used to benchmark the models, and presents the results.

5.3.1 Datasets

The datasets used in this chapter are the same as in [Zhang and LeCun, 2017]. In total, there are 14 large-scale datasets in 4 languages including Chinese, English, Japanese and Korean. There are 2 kinds of tasks in these datasets, including sentiment analysis and topic classification. Furthermore, 2 of the datasets are constructed by combining samples in all 4 languages to test the model's ability to handle different languages in the same fashion. Table 5.1 is a summarization.

In table 5.1, the first 9 datasets are sentiment analysis datasets in either Chinese, English, Japanese or Korean, from either restaurant review or online shopping websites. The following 3 datasets are topic classification datasets in either Chinese or English, from online news websites. The last two are multi-lingual datasets by combining online shopping reviews from 4 languages.

5.3.2 Training Parameters and Results

The parameters of the model are initialized using a normal distribution using mean 0 and standard deviation equal to $2/\sqrt{n}/1000$, where n is the number of output units each input unit connects to. This was inspired by [He et al., 2016], but due to the use of residual and dense connections the standard deviation is 1000 times smaller to prevent output explosion. The last feature-transforming convolutional layer in the dense block is still initialized with a standard deviation

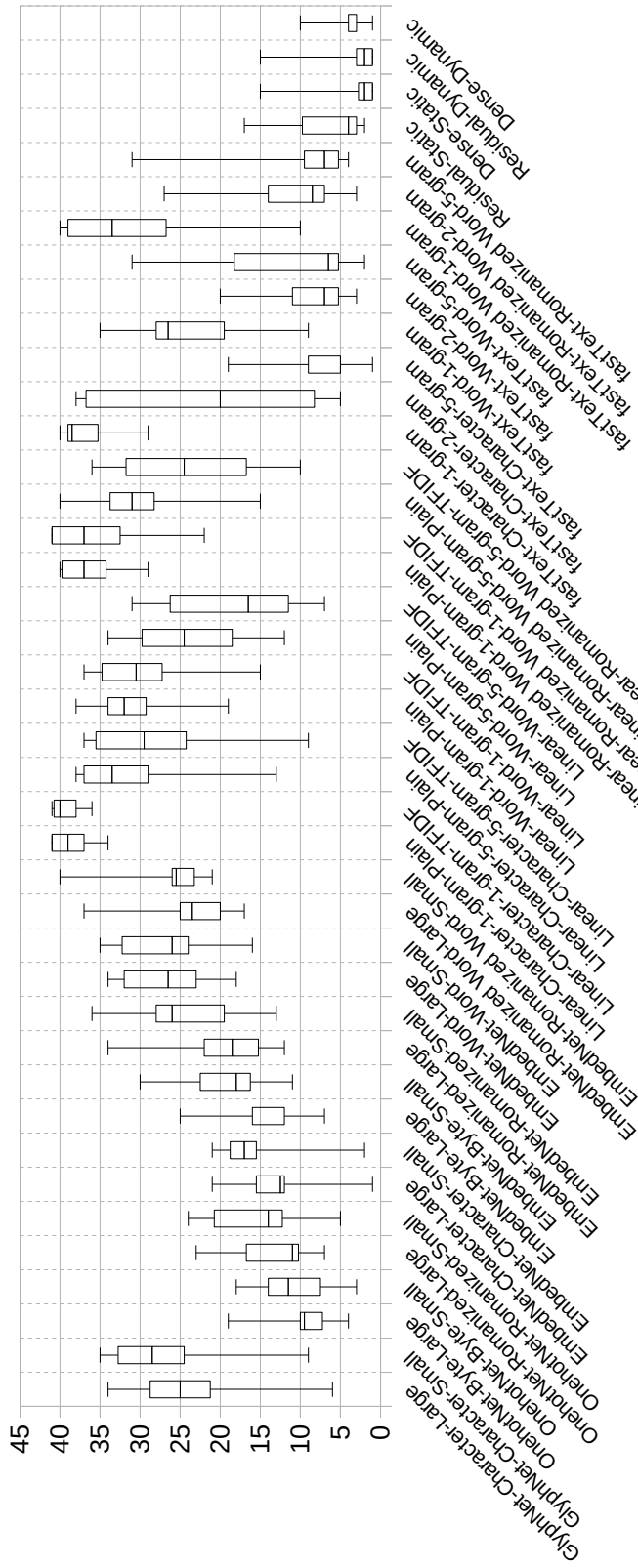


Figure 5.2: The rank box plot on development errors

of $2/\sqrt{n}$.

All of our models are trained using stochastic gradient descent (SGD) with momentum [Polyak, 1964] [Sutskever et al., 2013]. The initial learning rate is 0.001, which is halves every 1,000,000 steps. Each step runs through an example sampled with replacement from the training dataset. For most datasets except for joint full and joint binary, the training stops at the 25,000,000-th step. For joint full and joint binary, the training stops at the 100,000,000-th step because it has a lot more training data than others. For dynamic (recursive) models, we find that divide the gradients of the recursion groups by the number of times it was applied to the sample can speed up training.

The development errors of all the model considered in this chapter is summarized in table 5.2. The best results for each dataset are presented in bold font. We also list the training errors in the supplemental material. Analysis in section 5.4 shows that these models achieved new state-of-the-art results.

5.4 Discussion

This section demonstrates that the residual and dense convolutional networks in this chapter achieved state-of-the-art results, sometimes by a large margin. Comparisons are offered with previous convolutional networks, linear classifiers, and fastText [Joulin et al., 2016] models.

5.4.1 State-of-the-Art Models

For most of the datasets, we query the best model from the paper by [Zhang and LeCun, 2017]. Their work is an unbiased benchmark on the same 14 large-scale

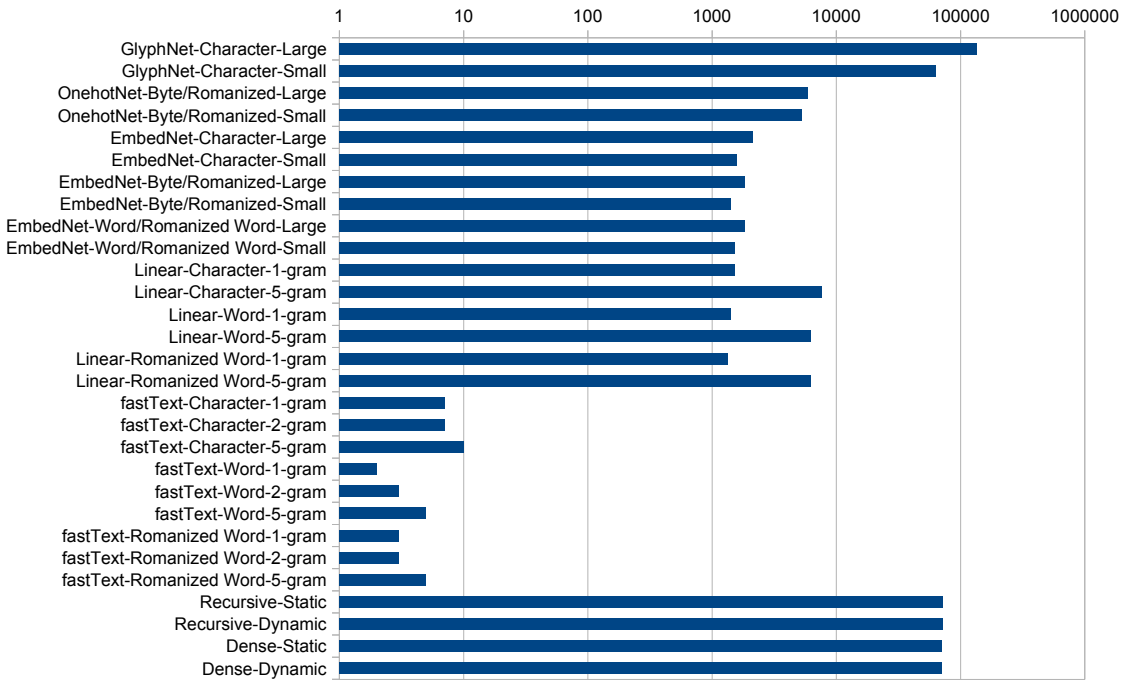


Figure 5.3: Time of going over 1,000,000 samples. Using the same data and hardware from [Zhang and LeCun, 2017]. The time axis is in logarithm scale

text classification datasets used in this chapter, on many different models. The first series of models are convolutional networks, operating on glyph, one-hot or embedding encoding mechanisms. For each of these encoding mechanisms, results were presented at byte, character, word or romanized word levels. Their work also includes comparisons with linear classifier using logistic regression and fastText [Joulin et al., 2016], using 1-gram, 2-gram and 5-gram variants, at character, word or romanized word levels.

Additionally, for the Amazon full and binary datasets we queried the best models from the paper by [Conneau et al., 2017], in which very deep convolutional networks are tried with variants on depth and pooling layer choices. The fastText paper [Joulin et al., 2016] also includes results on these 2 datasets, and the results

Model	Convolution	Linear	Total
Residual-Static	14,179,333	8,396,800	22,576,133
Residual-Dynamic	3,154,949	8,396,800	11,551,749
Dense-Static	6,092,203	3,606,528	9,698,821
Dense-Dynamic	1,357,829	3,606,528	4,964,357

Table 5.4: Number of parameters in different models

were confirmed to be the same within statistical tolerance in [Zhang and LeCun, 2017], because the same hyper-parameters were used for 2-gram models.

Table 5.3 lists our best results and the previous state-of-the-art. The last column also records the relative differences, in which negative values indicating our models are better and positive values otherwise. Our models achieved new state-of-the-art results for 10 out of 14 datasets. For the other 4 datasets, our models are not far away from the previous best models.

5.4.2 Aggregated Comparison

Following the analysis from [Zhang and LeCun, 2017], we put a box plot on the rank of development errors aggregated across different datasets in figure 5.2. For each dataset, we rank all of the models in ascending order of their development errors. As a result, the smaller the rank, the better the model performs. The box plot in figure 5.2 shows the minimum, first quartile, median, third quartile, and maximum rank across different datasets. In total there are 42 models in comparison.

From this box plot, we could easily see that both residual and dense connections improved the results consistently. In particular, either of the models in this chapter can do better than the previous best model – fastText [Joulin et al., 2016]

with 5-gram feature – in both result and consistency. That said, the amount of computation our models require is far more than that of fastText, as evidenced from 5.3. This figure is produced by running our models using the same hardware as [Zhang and LeCun, 2017] – one NVIDIA Tesla K40 GPU, but the experiments in this article are actually conducted on more recent GPUs such as NVIDIA Tesla V100 and NVIDIA GTX 1080Ti, which are 2 to 3 times faster due to both software and hardware improvements.

Given that these models all perform roughly the same, what sets these models apart is their number of parameters. The number of parameters for each of the models in this chapter is detailed in table 5.4 and figure 5.4. The use of dense connections in general can reduce the number of parameters to less than 40% of its counterparts using residual connections, whereas the use of recursion can reduce the parameters in the convolutional layers to less than 25% of its static counterparts. As a result, the dense-dynamic variant of our models has only less than 25% of the number of parameters compared to residual-static variant, while for convolutional layers the comparison is less than 10%. This aggressive reduction in parameters while keeping roughly the same performance is the reason why both dense connections and the recursive structure could pose an advantage in these model designs.

5.5 Conclusion

This chapter benchmarked byte-level convolutional networks using recent design improvements including residual [Huang et al., 2016] and dense [He et al., 2016] connections, and the use of recursive structure [Zhang and LeCun, 2018b].

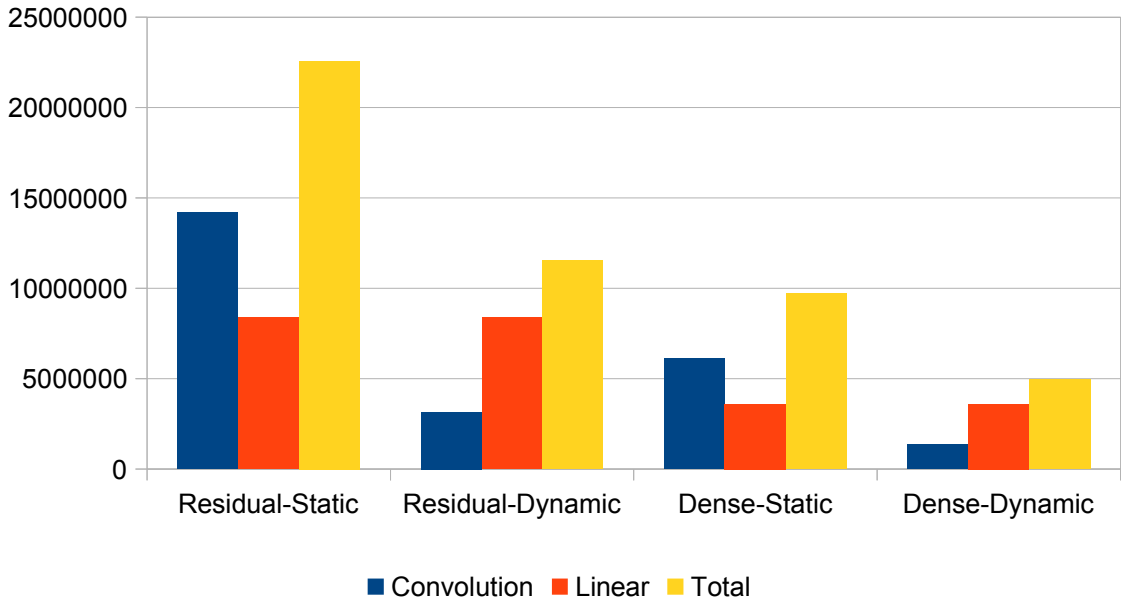


Figure 5.4: Number of parameters in different models

Results indicate that our models can achieve new state-of-the-art results for text classification in 4 languages including Chinese, English, Japanese and Korean. Analysis shows that our models are better than the previous best model in both result and consistency.

Chapter 6

Conclusion and Outlook

This chapter provides some conclusions for this dissertation, and discusses some possible research directions in the future.

6.1 Conclusion

This dissertation was split into 4 parts. The first part offers an empirical exploration on the use of character-level convolutional networks (ConvNets) for text classification. The second part provides an exhaustive study on which encoding for text could work well for convolutional networks. The third part proposes an auto-encoder architecture using byte-level recursive convolutional networks. The fourth part improves the previous text classification results using the auto-encoder design proposed in previous parts.

We obtained many large-scale datasets in many different languages for these research problems, including Arabic, Chinese, English, Japanese and Korean. These multi-lingual large-scale datasets are very helpful for us to show the advantage of character-level and byte-level encoding for convolutional networks, in which the

applicability of these models is greatly expanded since they are free from the need to pre-process any language to the level of words. This is done mainly in the first 2 parts of the article in which the task being studied is text classification. Hundreds of models are included in the comparisons, and many useful conclusions were drawn such as byte-level text processing is both feasible and competitive.

Because of these results, the next parts of the article focus only on byte-level convolutional networks. We proposed an auto-encoding architecture that is able to reduce variable-length texts into fixed-length vectors, and then expand these fixed-length vectors into variable-length texts. The entire process operates at the level of bytes, and the maximum length admitted by the model is 1024. This is actually quite long, and comparison with a simple recurrent network baseline shows some advantage for the task of auto-encoding. Instead of processing the sequence of symbols in a sequential way, the proposed architecture has recursion groups that reduce or expand the length by a factor of 2. As a result, the depth of the network is logarithm of that of recurrent networks, and the gradient vanishing problem was mitigated due to this reduced order of depth.

With these positive results in the auto-encoding architecture, the fourth part of the dissertation turns back to the text classification problem and uses the architecture design principles in the previous part to achieve new state-of-the-art results in text classification.

6.2 Short-term Outlook

The research projects shown in this dissertation are only the beginning of the story in tackling challenges in natural language processing using byte-level recursive

convolutional networks. There are a few short-term research directions that the author hopes to take in the future as continuation of the research projects presented in this dissertation.

6.2.1 Text Generation with Uncertainty

The first step for text generation beyond auto-encoding using recursive ConvNets should be focused on the uncertainty problem. By uncertainty, we refer to the case when there could be multiple acceptable outputs for the same input. One example of uncertainty is in translation, in which there could be multiple acceptable target translations for the same source text. Without an explicit way of handling such uncertainty, traditional supervised training alone for recursive ConvNets would fail because it does not model dependencies at the outputs. In other words, compared to a sequential text generation process, recursive ConvNets must be able to disentangle the mutual relationship of the outputs by itself, since no generated outputs are fed back to the model.

Due to the dominance of sequential text generation models, the problem of uncertainty is not broadly discussed for natural language processing. In these models, one entity (a word, character or byte) is generated at a time, and it is fed back to the next step for further generation. This allows the network to model the causal dependency of the output texts explicitly while it can still be trained in a supervised fashion, therefore avoiding the uncertainty problem. Unfortunately this is not the case for non-sequential generation, because it assumes conditional independence between output entities and does not capture the sequence dependency explicitly.

In the extreme case when there is no input but only multiple outputs, the uncer-

tainty framework collapses into purely generative models. Therefore, a simplified project that can focus on handling uncertainty is to learn a generative model to map random numbers into real samples. Then, the recently successful generative models such as variational auto-encoders (VAE) [Kingma and Welling, 2013] and generative adversarial networks (GAN) [Goodfellow et al., 2014] can be applied to it.

There have been many attempts for applying GAN for text generation. The idea of SeqGAN proposed by [Yu et al., 2017] uses policy gradient [Sutton et al., 2000] to provide gradients to the generator, by casting the problem as a sequential decision making process. On the other hand, MaskGAN [Fedus et al., 2018] uses a discriminator that accepts a discrete word with its surrounding context, using the same policy gradient method in an actor-critic framework [Sutton and Barto, 1998] [Degris et al., 2012]. Beyond reinforcement learning approaches, MaliGAN [Che et al., 2017] uses the maximum likelihood principle by assuming the discriminator has achieved optimum with respect to the current generator. Professor forcing [Goyal et al., 2016] was proposed to use GAN on the hidden units to ensure generator stability, which improves the quality of long samples. Adversarial feature matching [Zhang et al., 2017a] was an idea to improve RNN generators using a convolutional discriminator on the hidden units. Adversarially regularized auto-encoder (ARAE) [Zhao et al., 2018] makes the generator match the feature from the encoder. [Kusner and Hernández-Lobato, 2016] proposed to use a Gumbel-softmax distribution on the output of an RNN while the samples are provided as one-hot vectors.

The author of this dissertation also proposed one approach – Adversarially-Trained Normalized Noisy-Feature Auto-Encoder (ATNNFAE) [Zhang and LeCun,

2018a]. An ATNNFAE consists of an auto-encoder where the internal code is normalized on the unit sphere and corrupted by additive noise. Simultaneously, a replica of the decoder (sharing the same parameters as the AE decoder) is used as the generator and fed with random latent vectors. An adversarial discriminator is trained to distinguish training samples reconstructed from the AE from samples produced through the random-input generator, making the entire generator-discriminator path differentiable for discrete data like text. The combined effect of noise injection in the code and shared weights between the decoder and the generator can prevent the mode collapsing phenomenon commonly observed in GANs. The auto-encoder used here is the same as the one used in chapter 4 of this dissertation due to its advantageous performance on text auto-encoding.

6.2.2 Controllable Machine Translation

The previous research projects for text generation using convolutional networks focus on 2 basic questions – what kind of model design can work, and how to handle uncertainty at large scale and low levels. If these two problems can be successfully solved, applying byte-level text generation to differrent language processing tasks would be feasible. By ensuring that the 2 goals of homogeneity and scalability can be achieved, these models would have the advantage of better applicability to different languages with a minimal requirement for design adaptation, and the ability to scale with large-scale linguistic corpora.

Currently, most neural machine translation systems operate at the mixed levels of words, sub-words and characters. Although better than word-level only systems, they still require a significant amount of preprocessing to adapt to different languages. Meanwhile, most of them use some form of sequence-to-sequence recurrent

network model [Cho et al., 2014a] [Sutskever et al., 2014], and the scalability issue with long text generation is circumvented by the use of attention [Bahdanau et al., 2015] models. Our projects would offer a potential alternative to these models and add to the already growing set of sequence-to-sequence toolkits for different language tasks.

Furthermore, facing the uncertainty problem, the current neural machine translation models requires searching the probablistic space of outputs. With a text generation model that can handle uncertainty explicitly, we hope that it is possible to enable a translation system to output multiple acceptable translated results in a controllable fashion via internal representation or input hints. One way to implement such a model is to augment the generative models with a conditional component. In this case, the generated target translation is conditioned on a representation from the source text. The uncertainty problem – multiple acceptable target translations for the same source text – could be solved in the same way as the generative models in the previous section. The scalability issue for generating long texts is handled by the use of the non-sequential convolutional network text generator.

6.2.3 From Representation to Reasoning

When we can sufficiently solve text representation and generation problems, we would then be able to focus on reasoning models that can use memory, can infer from a knowledge base, or can combine feature learned from visual recognition or other senses. This is essentially the problem of how to learn, store and use common-sense knowledge, and it could be a goal that can span multiple years with multiple projects. In these projects, the byte-level recursive convolutional network

architecture proposed in this dissertation could become a useful component in both representation and generation. Due to its conceptual simplicity and applicability towards different languages (though not computationally simple or data efficient), these models could hopefully help researchers focus on more important challenges in machine reasoning.

In a summary, by thinking of natural language processing research as 3 components – representation, reasoning and generation, the author focuses on research projects that solve challenges and problems that are identified all across these components. With the 2 goals for research – generality and scalability, the author is focusing on models that can handle all different languages in the same fashion and can scale with computation and data. The projects presented in this dissertation are only the first steps towards applying convolutional networks to natural language processing, and they are only focused on the representation problem. There are many further challenges in all of representation, reasoning and generation that convolutional networks could become part of the solution.

Appendix A

Training and Validation Errors for Text Classification

The training errors of all models in chapter 3 are detailed in tables A.1, A.2, A.3 and A.4. The validation errors of all the models are detailed in tables A.5, A.6, A.7 and A.8.

Dataset	GlyphNet		OnehotNet			
	Character		Byte		Romanized	
	large	small	large	small	large	small
Dianping	23.97	24.33	22.42	22.68	22.78	22.97
JD f.	48.63	49.03	47.58	47.62	47.79	47.95
JD b.	9.73	9.97	9.03	9.06	9.16	9.17
Rakuten f.	46.59	46.94	44.62	44.95	44.74	44.96
Rakuten b.	6.45	6.63	5.54	5.71	5.64	5.75
11st f.	31.73	32.09	28.84	29.54	29.35	29.99
11st b.	13.78	13.88	13.01	13.07	13.15	13.20
Amazon f.	46.29	47.70	41.51	41.65	–	–
Amazon b.	8.26	8.90	6.06	6.11	–	–
Ifeng	16.35	17.35	12.43	13.62	14.99	15.90
Chinanews	11.58	12.44	9.32	9.61	10.12	10.52
NYTimes	17.24	17.99	12.36	12.57	–	–
Joint f.	45.04	45.62	43.18	43.34	43.56	43.69
Joint b.	9.97	10.04	8.67	8.70	8.89	8.95

Table A.1: GlyphNet and OnehotNet training errors

Dataset	Character		Byte		Romanized		Word		Rom. word	
	large	small	large	small	large	small	large	small	large	small
Dianping	22.97	23.17	23.33	23.60	24.66	25.45	24.03	24.25	23.07	23.31
JD f.	47.80	47.90	47.99	48.22	48.15	48.74	49.50	49.68	48.58	48.65
JD b.	9.23	9.24	8.96	9.04	9.17	9.48	10.19	10.23	9.34	9.47
Rakutenf f.	44.85	45.40	45.60	46.55	45.69	45.59	46.03	46.24	45.71	46.10
Rakuten b.	5.71	5.84	6.12	6.55	6.19	6.67	6.45	6.55	6.22	6.37
11st f.	29.80	30.39	31.65	32.90	32.50	33.74	33.50	34.70	38.79	39.74
11st b.	13.18	13.29	13.02	13.27	13.21	13.50	13.92	14.01	17.38	17.47
Amazon f.	43.05	43.76	–	–	–	–	43.78	44.32	–	–
Amazon b.	6.68	7.20	–	–	–	–	7.56	7.68	–	–
Ifeng	13.71	14.44	14.44	15.36	16.82	18.17	16.85	17.72	15.66	16.65
Chinanews	9.72	9.92	9.57	9.90	10.53	11.93	14.75	13.68	10.56	10.86
NYTimes	12.75	13.51	–	–	–	–	15.83	16.15	–	–
Joint f.	43.74	44.18	44.35	44.94	44.88	45.70	45.33	45.62	45.15	45.45
Joint b.	8.96	9.13	8.97	9.18	9.26	9.56	10.75	9.94	9.87	9.94

Table A.2: EmbedNet training errors

Dataset	Character				Word				Romanized Word			
	1-gram		5-gram		1-gram		5-gram		1-gram		5-gram	
	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf
Dianping	26.03	26.72	24.26	23.30	23.94	23.39	23.47	22.59	27.35	28.03	24.37	23.14
JD f.	50.84	51.30	47.27	46.08	47.74	46.55	45.86	43.62	52.19	52.61	47.74	46.15
JD b.	11.84	12.12	8.99	8.68	9.68	9.49	8.89	8.28	13.06	13.35	9.09	8.75
Rakutenf f.	52.21	52.82	47.18	45.57	46.96	45.90	45.55	43.61	47.66	46.71	46.85	44.17
Rakuten b.	12.43	12.94	8.10	7.25	8.35	8.13	7.19	6.47	8.78	8.71	7.31	6.57
11st f.	43.51	47.63	43.14	43.16	44.14	42.20	42.16	40.62	40.17	35.30	40.52	35.29
11st b.	17.73	18.01	14.33	14.24	15.09	15.03	13.34	12.91	14.32	13.87	13.83	12.79
Amazon f.	69.28	68.13	56.28	50.01	44.84	42.85	43.97	41.78	–	–	–	–
Amazon b.	34.36	33.91	14.92	12.08	9.15	8.37	8.43	8.03	–	–	–	–
Ifeng	22.00	21.78	21.42	21.35	18.49	16.51	19.60	18.34	26.59	26.81	23.08	22.06
Chinanews	15.16	14.93	15.14	13.31	11.28	10.00	13.32	12.68	20.10	20.39	15.62	13.76
NYTimes	57.21	53.77	39.78	26.29	17.81	14.30	19.63	17.88	–	–	–	–
Joint f.	57.16	56.72	49.68	47.43	46.53	45.57	44.97	44.43	47.93	47.19	47.22	46.61
Joint b.	19.77	19.40	12.07	10.85	10.08	10.51	9.44	8.94	11.86	11.43	11.41	11.02

Table A.3: Linear model training errors

Dataset	Character			Word			Romanized Word		
	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram
Dianping	25.82	21.03	19.34	23.12	21.45	16.21	27.07	22.07	19.43
JD f.	50.98	47.36	44.34	48.04	46.14	39.51	51.97	46.58	43.32
JD b.	11.83	8.32	7.49	9.60	6.19	6.14	13.01	8.64	7.48
Rakuten f.	51.92	43.38	40.54	45.83	42.30	36.13	46.57	41.15	37.66
Rakuten b.	12.15	6.51	3.91	7.80	4.88	3.26	8.30	5.18	3.68
11st f.	42.94	35.91	25.96	39.08	35.49	30.93	39.06	31.94	25.74
11st b.	17.68	13.17	12.12	15.12	12.75	10.63	11.80	11.29	8.11
Amazon f.	67.00	53.88	38.40	42.30	37.60	31.15	–	–	–
Amazon b.	32.72	18.60	5.46	7.78	3.47	0.22	–	–	–
Ifeng	21.00	11.21	3.90	13.84	10.81	0.62	25.84	13.11	4.40
Chinanews	13.55	5.93	0.17	7.03	1.61	0.02	18.58	6.53	2.36
NYTimes	51.07	24.13	8.75	10.73	3.53	6.47	–	–	–
Joint f.	56.79	46.61	40.52	45.47	41.98	33.90	47.01	41.08	38.17
Joint b.	19.51	11.96	7.79	10.31	7.36	6.09	11.11	7.19	7.10

Table A.4: fastText training errors

Dataset	GlyphNet		OnehotNet			
	Character		Byte		Romanized	
	large	small	large	small	large	small
Dianping	24.36	24.60	23.27	23.40	23.59	23.81
JD f.	49.00	49.33	48.14	48.30	48.44	48.58
JD b.	9.89	10.10	9.33	9.32	9.51	9.52
Rakuten f.	46.75	47.03	45.06	45.37	45.12	45.35
Rakuten b.	6.67	6.85	5.94	6.12	6.05	6.10
11st f.	32.74	33.04	32.59	32.47	32.76	32.72
11st b.	13.90	14.31	13.31	13.35	13.42	13.47
Amazon f.	46.61	47.89	42.20	42.28	–	–
Amazon b.	8.52	9.09	6.51	6.58	–	–
Ifeng	18.00	18.54	16.67	16.48	18.90	18.87
Chinanews	12.23	12.87	10.59	10.71	11.68	11.73
NYTimes	18.24	18.62	14.32	14.28	–	–
Joint f.	45.17	45.78	42.91	43.07	43.27	43.23
Joint b.	9.97	10.38	8.78	8.76	8.98	9.00

Table A.5: GlyphNet and OnehotNet validation errors

Dataset	Character		Byte		Romanized		Word		Rom. word	
	large	small	large	small	large	small	large	small	large	small
Dianping	23.68	23.70	24.18	24.30	25.48	26.09	24.60	24.77	23.77	23.88
JD f.	48.33	48.45	48.59	48.71	48.78	49.30	50.09	50.16	49.17	49.29
JD b.	9.45	9.43	9.22	9.23	9.49	9.74	10.39	10.46	9.60	9.71
Rakuten f.	45.18	45.66	45.94	46.86	46.14	46.76	46.33	46.53	45.94	46.31
Rakuten b.	6.09	6.14	6.50	6.88	6.57	6.97	6.82	6.88	6.56	6.69
11st f.	32.32	32.37	34.88	35.05	35.48	35.72	42.92	42.86	42.63	42.55
11st b.	13.44	13.51	13.27	13.48	13.49	13.72	16.43	15.75	17.67	17.70
Amazon f.	43.68	44.21	–	–	–	–	44.24	44.80	–	–
Amazon b.	7.15	7.46	–	–	–	–	7.91	8.00	–	–
Ifeng	16.99	17.05	17.10	17.53	19.19	19.98	20.81	20.71	19.44	19.46
Chinanews	11.01	11.10	10.52	10.82	11.83	12.75	14.72	14.94	11.90	12.04
NYTimes	14.14	14.61	–	–	–	–	17.64	17.81	–	–
Joint f.	43.61	44.09	44.16	44.76	44.72	45.44	45.48	45.84	45.00	45.32
Joint b.	9.01	9.17	9.07	9.26	9.32	9.63	10.65	10.75	9.92	10.01

Table A.6: EmbedNet validation errors

Dataset	Character				Word				Romanized Word			
	1-gram		5-gram		1-gram		5-gram		1-gram		5-gram	
	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf	plain	tfidf
Dianping	26.15	26.87	24.39	23.64	24.14	24.35	23.60	23.08	27.37	28.12	24.61	23.42
JD f.	51.52	51.67	48.48	48.23	49.43	50.09	48.33	48.33	52.18	52.80	49.00	48.50
JD b.	11.83	12.16	9.11	8.95	9.89	10.00	9.08	8.84	13.13	13.54	9.13	8.98
Rakuten f.	52.11	52.79	47.61	46.44	47.54	47.71	45.76	45.23	47.89	48.28	46.95	45.63
Rakuten b.	12.54	13.02	8.17	7.31	8.40	8.41	7.35	6.65	8.83	8.96	7.47	6.73
11st f.	43.92	48.38	43.63	43.45	49.81	48.31	54.86	51.95	45.42	44.82	45.59	44.25
11st b.	17.67	18.02	14.46	14.36	15.35	15.61	18.94	17.26	14.98	15.17	14.63	14.45
Amazon f.	69.45	68.59	56.83	51.30	45.36	44.88	44.66	42.67	–	–	–	–
Amazon b.	34.46	33.68	14.98	12.13	9.32	8.78	8.54	8.21	–	–	–	–
Ifeng	22.11	22.41	21.50	21.95	19.10	18.28	20.14	19.70	26.58	27.32	23.11	22.36
Chinanews	15.33	15.06	14.89	13.35	11.60	10.74	13.40	12.90	20.01	20.46	15.53	13.95
NYTimes	57.56	53.87	40.93	26.43	18.27	15.34	20.06	18.31	–	–	–	–
Joint f.	60.27	59.67	49.18	48.19	46.80	46.52	45.24	45.06	47.50	47.13	46.87	46.37
Joint b.	20.19	19.72	12.11	10.89	10.81	10.64	9.44	8.98	11.72	11.43	11.30	11.00

Table A.7: Linear model validation errors

Dataset	Character			Word			Romanized Word		
	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram
Dianping	25.87	22.90	22.39	23.80	22.69	22.68	27.09	22.95	22.47
JD f.	51.29	48.31	48.02	49.27	48.16	48.64	52.24	48.41	48.17
JD b.	11.83	8.94	8.74	9.86	9.14	9.95	13.05	9.08	8.76
Rakuten f.	51.96	44.87	43.25	46.55	43.79	46.29	47.04	43.77	43.25
Rakuten b.	12.26	6.95	5.47	8.05	5.91	5.46	8.49	5.95	5.45
11st f.	43.23	38.73	38.62	41.66	41.20	42.20	40.75	41.78	43.20
11st b.	17.67	13.67	13.13	14.55	14.29	14.91	14.89	14.53	14.99
Amazon f.	67.06	53.97	41.03	43.80	40.19	39.99	–	–	–
Amazon b.	32.78	18.55	6.27	8.35	5.58	5.40	–	–	–
Ifeng	21.58	16.56	16.29	17.79	16.63	16.93	26.02	18.20	17.84
Chinanews	13.89	9.30	9.07	9.84	9.22	9.23	18.66	9.63	9.38
NYTimes	51.45	24.69	12.74	13.61	11.86	13.25	–	–	–
Joint f.	59.47	49.36	44.01	46.05	43.53	43.64	46.70	43.13	43.24
Joint b.	19.84	12.34	8.98	10.02	9.01	9.25	11.37	8.79	8.83

Table A.8: fastText validation errors

Appendix B

Validated Epoches for fastText

The validated epoches for running all fastText models in chapter 3 are detailed in Table B.1.

Dataset	Character			Word			Romanized Word		
	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram	1-gram	2-gram	5-gram
Dianping	10	10	2	10	2	2	5	5	2
JD f.	2	2	2	5	2	2	10	5	2
JD b.	10	10	2	10	10	2	10	5	2
Rakutenf f.	10	10	2	10	2	2	10	5	2
Rakuten b.	10	10	5	10	5	2	10	5	2
11st f.	5	5	5	5	2	2	2	2	2
11st b.	10	10	2	2	2	2	5	2	2
Amazon f.	10	10	10	10	2	2	–	–	–
Amazon b.	10	10	10	10	5	5	–	–	–
Ifeng	10	5	5	5	2	5	10	5	5
Chinanews	10	5	10	5	5	10	10	5	5
NYTimes	10	10	10	5	5	2	–	–	–
Joint f.	10	10	2	10	2	2	10	5	2
Joint b.	10	10	2	10	5	2	10	10	2

Table B.1: fastText epoches

Appendix C

Training and Validation Errors for Improved Model

Training and validation errors are listed in the table C.1 and table C.2.

Dataset	Residual		Dense	
	Dynamic	Static	Dynamic	Static
Dianping	20.57%	17.41%	20.03%	17.40%
JD f.	45.11%	43.96%	45.80%	44.57%
JD b.	7.62%	7.48%	7.29%	7.19%
Rakuten f.	41.16%	40.35%	41.76%	39.43%
Rakuten b.	3.80%	3.48%	4.08%	3.09%
11st f.	13.89%	7.10%	26.67%	12.33%
11st b.	11.57%	11.16%	11.80%	10.98%
Amazon f.	36.99%	35.83%	37.42%	33.56%
Amazon b.	3.59%	3.50%	3.72%	2.56%
Ifeng	2.96%	0.86%	9.67%	2.27%
Chinanews	6.61%	1.65%	4.51%	2.28%
NYTimes	9.36%	3.33%	6.07%	4.08%
Joint f.	39.85%	39.45%	41.55%	39.42%
Joint b.	7.13%	7.07%	7.54%	6.89%

Table C.1: Training errors

Dataset	Residual		Dense	
	Dynamic	Static	Dynamic	Static
Dianping	22.40%	23.56%	22.32%	23.58%
JD f.	47.03%	47.54%	46.94%	46.81%
JD b.	8.35%	8.50%	8.34%	8.35%
Rakuten f.	42.25%	42.77%	42.61%	41.90%
Rakuten b.	4.64%	4.88%	4.86%	4.77%
11st f.	36.71%	37.62%	32.79%	36.09%
11st b.	12.29%	12.51%	12.52%	12.49%
Amazon f.	38.29%	38.29%	38.87%	38.11%
Amazon b.	4.28%	4.39%	4.59%	4.44%
Ifeng	17.45%	17.27%	15.54%	15.97%
Chinanews	9.28%	10.31%	10.12%	9.27%
NYTimes	11.79%	13.16%	13.27%	11.42%
Joint f.	40.20%	40.30%	41.41%	40.20%
Joint b.	7.44%	7.46%	7.76%	7.45%

Table C.2: Validation errors

Bibliography

[Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166.

[Botha and Blunsom, 2014] Botha, J. and Blunsom, P. (2014). Compositional morphology for word representations and language modelling. In *International Conference on Machine Learning*, pages 1899–1907.

[Bottou et al., 1989] Bottou, L., Fogelman Soulié, F., Blanchet, P., and Lienard, J. (1989). Experiments with time delay networks and dynamic time warping for speaker independent isolated digit recognition. In *Proceedings of EuroSpeech 89*, volume 2, pages 537–540, Paris, France.

[Boureau et al., 2010a] Boureau, Y.-L., Bach, F., LeCun, Y., and Ponce, J. (2010a). Learning mid-level features for recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2559–2566. IEEE.

[Boureau et al., 2010b] Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010b). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 111–118.

[Che et al., 2017] Che, T., Li, Y., Zhang, R., Hjelm, R. D., Li, W., Song, Y., and Bengio, Y. (2017). Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*.

[Cho et al., 2014a] Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103.

[Cho et al., 2014b] Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

[Chung et al., 2016] Chung, J., Ahn, S., and Bengio, Y. (2016). Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[Collobert et al., 2011a] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011a). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

[Collobert et al., 2011b] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011b). Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537.

[Collobert et al., 2011c] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011c). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

[Conneau et al., 2017] Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2017). Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1107–1116, Valencia, Spain. Association for Computational Linguistics.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Costa-jussà et al., 2017] Costa-jussà, M. R., Aldón, D., and Fonollosa, J. A. R. (2017). Chinese–spanish neural machine translation enhanced with character and word bitmap fonts. *Machine Translation*, pages 1–13.

[Cox, 1958] Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242.

[Dai and Le, 2015] Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems, NIPS*.

[Degris et al., 2012] Degris, T., Pilarski, P. M., and Sutton, R. S. (2012). Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC), 2012*, pages 2177–2182. IEEE.

[dos Santos and Gatti, 2014] dos Santos, C. and Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

[dos Santos et al., 2015] dos Santos, C., Guimaraes, V., Niterói, R., and de Janeiro, R. (2015). Boosting named entity recognition with neural character embeddings. In *Proceedings of NEWS 2015 The Fifth Named Entities Workshop*, page 25.

[dos Santos and Zadrozny, 2014] dos Santos, C. and Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

[El Hihi and Bengio, 1996] El Hihi, S. and Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499.

[Fedus et al., 2018] Fedus, W., Goodfellow, I., and Dai, A. M. (2018). MaskGAN: Better text generation via filling in the _____. In *International Conference on Learning Representations*.

[Fellbaum, 2005] Fellbaum, C. (2005). Wordnet and wordnets. In Brown, K., editor, *Encyclopedia of Language and Linguistics*, pages 665–670, Oxford. Elsevier.

[Gehring et al., 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional Sequence to Sequence Learning. In *Proc. of ICML*.

[Gillick et al., 2016] Gillick, D., Brunk, C., Vinyals, O., and Subramanya, A. (2016). Multilingual language processing from bytes. In *Proceedings of NAACL-HLT*, pages 1296–1306.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Goodman, 2001] Goodman, J. (2001). Classes for fast maximum entropy training. In *ICASSP*, pages 561–564. IEEE.

[Goyal et al., 2016] Goyal, A., Lamb, A. M., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for train-

ing recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609.

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610.

[Graves et al., 2014] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[Greff et al., 2015] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015). LSTM: A search space odyssey. *CoRR*, abs/1503.04069.

[Gu et al., 2018] Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. (2018). Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

[Gulcehre et al., 2016] Gulcehre, C., Chandar, S., Cho, K., and Bengio, Y. (2016). Dynamic neural turing machine with soft and hard addressing schemes. *arXiv preprint arXiv:1607.00036*.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hermann et al., 2015] Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv: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*, 9(8):1735–1780.

[Huang et al., 2016] Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. (2016). Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*.

[Joachims, 1998] Joachims, T. (1998). Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, pages 137–142. Springer-Verlag.

[Johnson and Zhang, 2014] Johnson, R. and Zhang, T. (2014). Effective use of word order for text categorization with convolutional neural networks. *CoRR*, abs/1412.1058.

[Johnson and Zhang, 2017] Johnson, R. and Zhang, T. (2017). Deep pyramid convolutional neural network for text classification. In *Proceedings of the 55nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

[Jones, 1972] Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

[Joulin et al., 2016] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

[Joulin and Mikolov, 2015] Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pages 190–198.

[Józefowicz et al., 2016] Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

[Kalchbrenner et al., 2016] Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

[Kalchbrenner et al., 2014] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 212–217. Association for Computational Linguistics.

[Kanaris et al., 2007] Kanaris, I., Kanaris, K., Houvardas, I., and Stamatatos, E. (2007). Words versus character n-grams for anti-spam filtering. *International Journal on Artificial Intelligence Tools*, 16(06):1047–1067.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *EMNLP 2014*.

[Kim et al., 2016] Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

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

[Kiros et al., 2015] Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

[Koutnik et al., 2014] Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A clockwork rnn. In *International Conference on Machine Learning*, pages 1863–1871.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Kusner and Hernández-Lobato, 2016] Kusner, M. J. and Hernández-Lobato, J. M. (2016). Gans for sequences of discrete elements with the gumbel-softmax distribution. *CoRR*, abs/1611.04051.

[Le and Mikolov, 2014] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

[LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan-Kaufmann.

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

[Lee et al., 2017] Lee, J., Cho, K., and Hofmann, T. (2017). Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378.

[Lehmann et al., 2014] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2014). DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*.

[Lev et al., 2015] Lev, G., Klein, B., and Wolf, L. (2015). In defense of word embedding for generic text representation. In Biemann, C., Handschuh, S., Freitas, A., Meziane, F., and Métais, E., editors, *Natural Language Processing and Information Systems*, volume 9103 of *Lecture Notes in Computer Science*, pages 35–50. Springer International Publishing.

[Levy et al., 2015] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

[Lewis et al., 2004] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397.

[Liu et al., 2017] Liu, F., Lu, H., Lo, C., and Neubig, G. (2017). Learning character-level compositionality with visual features. In *The 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, Vancouver, Canada.

[Liu et al., 2016] Liu, P., Qiu, X., and Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2873–2879. AAAI Press.

[Luong et al., 2013] Luong, T., Socher, R., and Manning, C. (2013). Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113.

[McAuley and Leskovec, 2013] McAuley, J. and Leskovec, J. (2013). Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM.

[Miikkulainen and Dyer, 1991] Miikkulainen, R. and Dyer, M. G. (1991). Natural language processing with modular pdp networks and distributed lexicon. *Cognitive Science*, 15(3):343–399.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Niu et al., 2011] Niu, F., Recht, B., Re, C., and Wright, S. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc.

[Oord et al., 2016] Oord, A. v. d., 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. *arXiv preprint arXiv:1609.03499*.

[Parker et al., 2011a] Parker, R., Graff, D., Chen, K., Kong, J., and Maeda, K. (2011a). Arabic gigaword fifth edition ldc2011t11. Web Download.

[Parker et al., 2011b] Parker, R., Graff, D., Chen, K., Kong, J., and Maeda, K. (2011b). Chinese gigaword fifth edition ldc2011t13. Web Download.

[Parker et al., 2011c] Parker, R., Graff, D., Kong, J., Chen, K., and Maeda, K. (2011c). English gigaword fifth edition ldc2011t07. Web Download.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *ICML 2013*, volume 28 of *JMLR Proceedings*, pages 1310–1318. JMLR.org.

[Peng et al., 2003] Peng, F., Huang, X., Schuurmans, D., and Wang, S. (2003). Text classification in asian languages without word segmentation. In *Proceedings of the sixth international workshop on Information retrieval with Asian languages-Volume 11*, pages 41–48. Association for Computational Linguistics.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Polyak, 1964] Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17.

[Rumelhart et al., 1986] Rumelhart, D., Hintont, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[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)*, 115(3):211–252.

[Schmidhuber, 1992] Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

[Sermanet et al., 2014] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and Lecun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*.

[Shen et al., 2014] Shen, Y., He, X., Gao, J., Deng, L., and Mesnil, G. (2014). 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.

[Shi et al., 2016] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883.

[Shimada et al., 2016] Shimada, D., Kotani, R., and Iyatomi, H. (2016). Document classification through image-based character embedding and wildcard training. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3922–3927.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Sparck Jones, 1972] Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21.

[Sukhbaatar et al., 2015] Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. (2015). End-to-end memory networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc.

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

[Sutskever et al., 2011] Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

[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.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*.

[Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[van den Oord et al., 2017] van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., van den Driessche, G., Lockhart, E., Cobo, L. C., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. (2017). Parallel wavenet: Fast high-fidelity speech synthesis. *CoRR*, abs/1711.10433.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

[Waibel et al., 1989] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(3):328–339.

[Wang et al., 2008] Wang, C., Zhang, M., Ma, S., and Ru, L. (2008). Automatic online news issue construction in web environment. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 457–466, New York, NY, USA. ACM.

[Weinberger et al., 2009] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. (2009). Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM.

[Weston et al., 2014] Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *CoRR*, abs/1410.3916.

[Xiao and Cho, 2016] Xiao, Y. and Cho, K. (2016). Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057.

[Yu et al., 2017] Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858.

[Zhang, 2013] Zhang, X. (2013). Pac-learning for energy-based models. Master's thesis, Computer Science Department, Courant Institute of Mathematical Sciences, New York University.

[Zhang and LeCun, 2017] Zhang, X. and LeCun, Y. (2017). Which encoding is the best for text classification in chinese, english, japanese and korean? *CoRR*, abs/1708.02657.

[Zhang and LeCun, 2018a] Zhang, X. and LeCun, Y. (2018a). Adversarially-trained normalized noisy-feature auto-encoder for text generation. *CoRR*, abs/1811.04201.

[Zhang and LeCun, 2018b] Zhang, X. and LeCun, Y. (2018b). Byte-level recursive convolutional auto-encoder for text. *International Conference on Learning Representations*. rejected.

[Zhang et al., 2015] Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

[Zhang et al., 2017a] Zhang, Y., Gan, Z., Fan, K., Chen, Z., Henao, R., Shen, D., and Carin, L. (2017a). Adversarial feature matching for text generation. In *International Conference on Machine Learning*, pages 4006–4015.

[Zhang et al., 2014a] Zhang, Y., Lai, G., Zhang, M., Zhang, Y., Liu, Y., and Ma, S. (2014a). Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 83–92. ACM.

[Zhang et al., 2017b] Zhang, Y., Shen, D., Wang, G., Gan, Z., Henao, R., and Carin, L. (2017b). Deconvolutional paragraph representation learning. In *Advances in Neural Information Processing Systems*, pages 4169–4179.

[Zhang et al., 2014b] Zhang, Y., Zhang, H., Zhang, M., Liu, Y., and Ma, S. (2014b). Do users rate or review?: Boost phrase-level sentiment labeling with review-level sentiment classification. In *Proceedings of the 37th international*

ACM SIGIR conference on Research & development in information retrieval, pages 1027–1030. ACM.

[Zhang et al., 2013a] Zhang, Y., Zhang, M., Liu, Y., and Ma, S. (2013a). Improve collaborative filtering through bordered block diagonal form matrices. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 313–322. ACM.

[Zhang et al., 2013b] Zhang, Y., Zhang, M., Liu, Y., Ma, S., and Feng, S. (2013b). Localized matrix factorization for recommendation based on matrix block diagonal forms. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1511–1520. ACM.

[Zhao et al., 2018] Zhao, J. J., Kim, Y., Zhang, K., Rush, A. M., and LeCun, Y. (2018). Adversarially regularized autoencoders. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 5897–5906. JMLR.org.