AN ABSTRACT OF THE DISSERTATION OF

Mingbo Ma for the degree of Doctor of Philosophy in Computer Science presented on September 5, 2018.

Title: Structured Neural Models for Natural Language Processing

Abstract approved: ____
Liang Huang

Most tasks in natural language processing (NLP) involves structured information from both input (e.g., a sentence or a paragraph) and output (e.g., a tag sequence, a parse tree or a translated sentence). While neural models achieve great successes in other domains such as computer vision, applying those frameworks to NLP remains challenging for the following reasons. On the source side, we are dealing with input sentences with very complex structures. A simple local swap between two adjacent words could lead to opposite meanings. In addition, input sentences are often noisy as they are collected from real-world scenarios, e.g. online reviews or tweets. Our models need to be able to deal with syntactic variety and polysemy. On the target side, we are often expected to generate structured outputs like translated sentences or parse trees, by searching over an exponentially large search space. In this case, when exact search is intractable, we resort to inexact search methods such as beam search. In this thesis, we start by introducing several classification algorithms with structured information from the source side but unstructured outputs (sentence level classifications, e.g., sentiment analysis). Then we explore models which generate structured output from the unstructured input signal (e.g., image captioning). Finally, we investigate more complex frameworks that deal with structured information on both input and output sides (e.g., machine translation).

Structured Neural Models for Natural Language Processing

by

Mingbo Ma

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented September 5, 2018
Commencement June 2019

Doctor of Philosophy dissertation of Mingbo Ma presented on September 5, 2018.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Mingbo Ma, Author

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF ALGORITHMS

Chapter 1: Introduction

Structured information is essential for solving various natural language processing (NLP) problems. Some general and fundamental tasks in NLP is structured prediction which expects to generate structured output itself. For the simpler cases, when we expect unstructured information from structure input, for example, sentimental classification, question type classification, the structured information could provide a more in-depth understanding of given sequence compared with linear, surface information. In the task of structured prediction, such as Part-Of-Speech tagging, constituent/dependency parsing, and machine translation, structured information becomes more important.

Understanding the structured information is the crucial guidance for utilizing them with difference algorithm. Before we introduce the proposed algorithm in this paper, we will first review different forms of structured information in this chapter.

1.1 Typical Structured Prediction Problems

Structure prediction usually takes sentence or paragraph as input. For the output side, there are many different kinds of outputs are expected on the problems. In this section, we will first review the sequential prediction problem, e.g., Part-Of-Speech (POS) tagging problem and Named-Entity Recognition (NER). Then we will survey different kinds of parsing problems, e.g., constituency and dependency parsing. In the end, we will talk about machine translation which has different challenges than other structure prediction problems.

1.1.1 Sequential Labeling

The goal of sequential labeling is to labeling or tagging each word's semantic role or function in a sentence for a given task.

A typical sequential labeling is POS tagging. The goal of POS tagging is to give each word in the sentence a appropriate predefined labeled role, e.g., noun, verb, adjective and so on. The decision for choosing a label for one word also depends on the syntactic role of other words in the sentence.

Another instants of sequential labeling is NER. In this task, we only need to label each word's semantic meaning within given context, such as location, organization or people's name entity. The difference between POS and NER tagging is not only about the predefined label set or the meaning the labels, in some cases, one particular label could span over a few continues words or sentence segments.

Sequential labeling provides rich information about the neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and about the syntactic structure around the word (nouns are generally part of noun phrases). In the task of NER, it also provides a lot of name entity information. Tagging is useful technique for information extraction, question answering, and shallow parsing. Within a limited efforts, tagging is considered as much easier task compared with parsing and its accuracy is quite high.

Fig. 1.1 shows one example of POS and NER tagging. In the POS tagging case, we annotated different syntactic role of each word. In the NER case, we only considering the location and organization name entities.

	Oregon	State	University	is	an
POS tagging	NNP	NNP	NNP	VBZ	DT
NER	ORG	ORG	ORG	-	-
	international	public	research	university	in
POS tagging	JJ	JJ	NN	NN	IN
NER	-	-	-	-	-
	the	northwest	United	States	,
POS tagging	DT	NNP	NNP	NNPS	,
NER	-	-	LOC	LOC	-
	located	in	Corvallis	,	Oregon
POS tagging	VBN	IN	NNP	,	NNP
NER	-	-	LOC	-	LOC

Table 1.1: Example sentence from the introduction of Oregon State University in Wikipedia. This example combines two different kinds of sequence labeling problem: POS tagging and Named Entity Recognition (NER).

1.1.2 Parsing

More complicated structure than above sequence labeling problem is classical parsing problems, which the input is still an sentence, but we are expecting a tree-formed output. There are two different kinds of parsings in most cases: constituent parsing, or syntactic parsing, and dependency parsing.

Fig. 1.1 is one example of constituent parse tree with given rule set and desired tree structure. From the rule set point of view, we are given predefined branching rules from a parent node (left side) to the children nodes (right side). For example, S has three different of branching possibilities based on the rule set. The constituent parse tree is usually be formed by maxing over all the possible rules with top-down or button-up fashion. Constituent parse is one of the most important core task in NLP. Not only because of the tree structure but it also provide rich information about each word's functionality in the tree.

- S → NP VP
- S → Aux NP VP
- S → VP
- NP → Det Nominal
- Nominal → Noun
- Nominal → Noun Nominal
- Nominal → Nominal PP
- NP → Proper-Noun
- VP → Verb
- VP → Verb NP

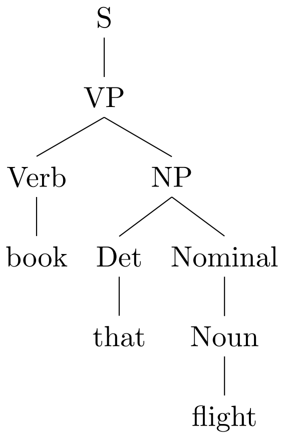


Figure 1.1: Example grammar and desired tree for sentence: Book that flight.

Another type of parsing is dependency parsing, which only analysis the dependent relationships between words. The label of each word is not important in dependency parsing. In dependency parsing, a tree edge start from a tail word to a head word, indicating that the the tail word modifies the head word. One tail word is constrained by only can have one modify on one head word, but on the other hand, one head word can be the modified by many tail words. Fig. 1.2 shows one example of dependency

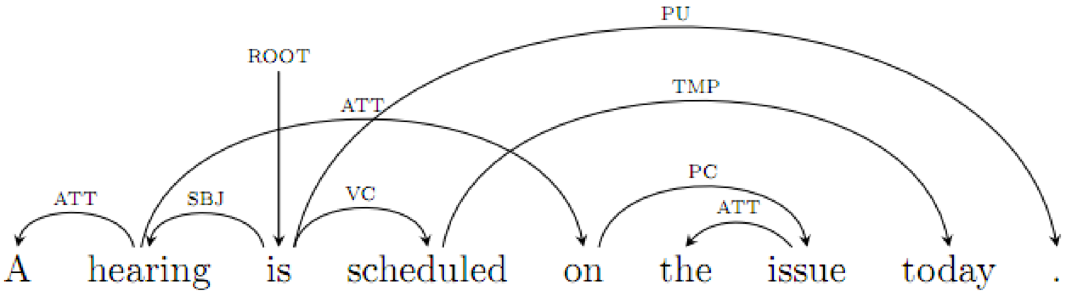


Figure 1.2: An example of dependency parse tree.

parsing.

In this dissertation, we only discuss about how to use parsing information to enrich the sentence representation. Deeper discussion of parsing algorithm is not within the scope of this paper.

In the following chapters, we will discuss utilizing the structure information step by step. In Chapter 2, we will first introduce a model deal with enhancing the sequence representation by implanting structure information into the source side. In this case, we only consider the source side's structures and output classification results over the entire sentence. In Chapter 3, we then introduce a hybrid model between the task of sequential label and sentence level classification. We will demonstrate how these two different tasks could help each other. In the following Chapter 4, we introduce a other model which deals with structured label hierarchical. In this case, we are expecting structured information on the output side. Image captioning is well-known tasks for text generation problem, which also tries to learn the structures on the target side. In Chapter 5, we discuss a simple model which combines image information which are generated from CNNs and source text inputs from on language to generate the structured text output in a different language. At last, we introduce several algorithms that deals with structure information on both sides in Chapter 6. We take machine translation as example, and show how these information from both sides are incorporate each other. We first review several famous frameworks for sequence-to-sequence learning and introduce a task, simultaneous translation. After we discuss the difficulties and challenges for simultaneous translation, we also propose our solutions to this problems.

Chapter 2: Using Structure Information for Sentence Level Classification

2.1 Motivation

In sentence modeling and classification, convolutional neural network approaches have recently achieved state-of-the-art results, but all such efforts process word vectors sequentially and neglect long-distance dependencies. To combine deep learning with linguistic structures, we propose a dependency-based convolution approach, making use of tree-based *n*-grams rather than surface ones, thus utilizing non-local interactions between words. Our model improves sequential baselines on all four sentiment and question classification tasks, and achieves the highest published accuracy on TREC.

The work presented in this chapter has been published in the Proceedings of Association for Computational Linguistics 2015 [40].

2.1.1 Introduction

Convolutional neural networks (CNNs), originally invented in computer vision [36], has recently attracted much attention in natural language processing (NLP) on problems such as sequence labeling [10], semantic parsing [78], and search query retrieval [59]. In particular, recent work on CNN-based sentence modeling [26, 27] has achieved excellent, often state-of-the-art, results on various classification tasks such as sentiment, subjectivity, and question-type classification. However, despite their celebrated success, there remains a major limitation from the linguistics perspective: CNNs, being invented on pixel matrices in image processing, only consider sequential *n*-grams that are consecutive on the surface string and neglect long-distance dependencies, while the latter play an important role in many linguistic phenomena such as negation, subordination, and *wh*-extraction, all of which might dully affect the sentiment, subjectivity, or other categorization of the sentence.

Indeed, in the sentiment analysis literature, researchers have incorporated long-

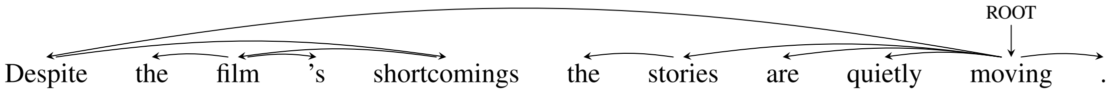


Figure 2.1: Dependency tree of an example sentence from the Movie Reviews dataset.

distance information from syntactic parse trees, but the results are somewhat inconsistent: some reported small improvements [16, 46], while some otherwise [12, 31]. As a result, syntactic features have yet to become popular in the sentiment analysis community. We suspect one of the reasons for this is data sparsity (according to our experiments, tree n-grams are significantly sparser than surface n-grams), but this problem has largely been alleviated by the recent advances in word embedding. For the task of sentence entailment, [81] also show that the sub-tree structure-based entailment could help the overall prediction. Most importantly, these efforts were all before the deep learning revolution, Ideally, one would hope to combine the merits of both worlds: In other words we would like to combine the merits of both worlds: deep learning to control sparsity, and tree structures for long-distance dependencies.

So we propose a very simple dependency-based convolutional neural networks (DCNNs). Our model is similar to [27], but while his sequential CNNs put a word in its sequential context, ours considers a word and its parent, grand-parent, great-grand-parent, and siblings on the dependency tree. This way we incorporate long-distance information that are otherwise unavailable on the surface string.

Experiments on three classification tasks demonstrate the superior performance of our DCNNs over the baseline sequential CNNs. In particular, our accuracy on the TREC dataset outperforms all previously published results in the literature, including those with heavy hand-engineered features.

2.2 Dependency-based Convolution

The original CNN, first proposed by [36], applies convolution kernels on a series of continuous areas of given images, and was adapted to NLP by [10]. Following [27], one dimensional convolution operates the convolution kernel in sequential order in Equa-

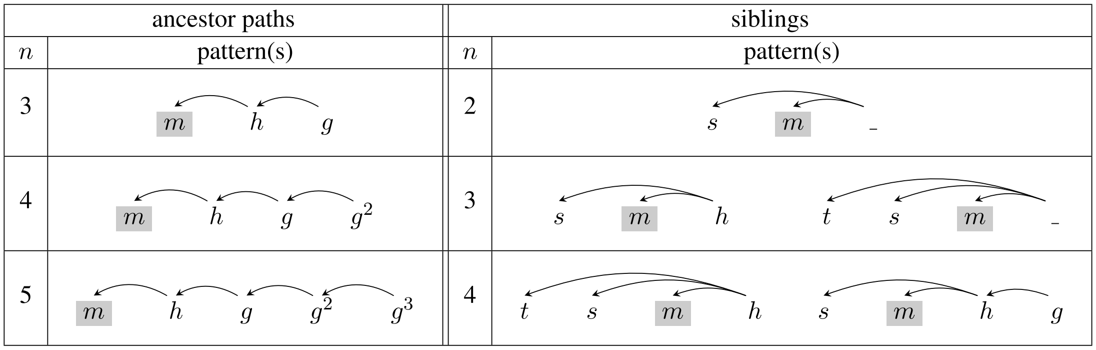


Figure 2.2: Convolution patterns on trees. Word concatenation always starts with m, while h, g, and g^2 denote parent, grand parent, and great-grand parent, etc., and "_" denotes words excluded in convolution.

tion 2.1, where $\mathbf{x}_i \in \mathbb{R}^d$ represents the d dimensional word representation for the i-th word in the sentence, and $\oplus$ is the concatenation operator. Therefore $\widetilde{\mathbf{x}}_{i,j}$ refers to concatenated word vector from the i-th word to the $(i+j)$-th word:

$$\widetilde{\mathbf{x}}_{i,j} = \mathbf{x}_i \oplus \mathbf{x}_{i+1} \oplus \cdots \oplus \mathbf{x}_{i+j} \tag{2.1}$$

Sequential word concatenation $\widetilde{\mathbf{x}}_{i,j}$ works as n-gram models which feeds local information into convolution operations. However, this setting can not capture long-distance relationships unless we enlarge the window indefinitely which would inevitably cause the data sparsity problem.

In order to capture the long-distance dependencies we propose the dependency-based convolution model (DCNN). Figure 2.1 illustrates an example from the Movie Reviews (MR) dataset [51]. The sentiment of this sentence is obviously positive, but this is quite difficult for sequential CNNs because many n-gram windows would include the highly negative word "shortcomings", and the distance between "Despite" and "shortcomings" is quite long. DCNN, however, could capture the tree-based bigram "Despite – shortcomings", thus flipping the sentiment, and the tree-based trigram "ROOT – moving – stories", which is highly positive.

2.2.1 Convolution on Ancestor Paths

We define our concatenation based on the dependency tree for a given modifier $\mathbf{x}_i$:

$$\mathbf{x}_{i,k} = \mathbf{x}_i \oplus \mathbf{x}_{p(i)} \oplus \cdots \oplus \mathbf{x}_{p^{k-1}(i)} \tag{2.2}$$

where function $p^k(i)$ returns the i-th word's k-th ancestor index, which is recursively defined as:

$$p^k(i) = \begin{cases} p(p^{k-1}(i)) & \text{if} \quad k > 0 \\ i & \text{if} \quad k = 0 \end{cases} \tag{2.3}$$

Figure 2.2 (left) illustrates ancestor paths patterns with various orders. We always start the convolution with x_i and concatenate with its ancestors. If the root node is reached, we add "ROOT" as dummy ancestors (vertical padding).

For a given tree-based concatenated word sequence $\mathbf{x}_{i,k}$, the convolution operation applies a filter $\mathbf{w} \in \mathbb{R}^{k \times d}$ to $\mathbf{x}_{i,k}$ with a bias term b described in equation 2.4:

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i,k} + b) \tag{2.4}$$

where f is a non-linear activation function such as rectified linear unit (ReLu) or sigmoid function. The filter $\mathbf{w}$ is applied to each word in the sentence, generating the feature map $\mathbf{c} \in \mathbb{R}^l$:

$$\mathbf{c} = [c_1, c_2, \cdots, c_l] \tag{2.5}$$

where l is the length of the sentence.

2.2.2 Max-Over-Tree Pooling and Dropout

The filters convolve with different word concatenation in Eq. 2.4 can be regarded as pattern detection: only the most similar pattern between the words and the filter could return the maximum activation. In sequential CNNs, max-over-time pooling [10, 27] operates over the feature map to get the maximum activation $\hat{c} = \max \mathbf{c}$ representing the entire feature map. Our DCNNs also pool the maximum activation from feature map to detect the strongest activation over the whole tree (i.e., over the whole sentence). Since the tree no longer defines a sequential "time" direction, we refer to our pooling as "max-over-tree" pooling. Another issue of sentence length variation are solved naturally by max-pooling.

In order to capture enough variations, we randomly initialize the set of filters to detect different structure patterns. Each filter's height is the number of words considered and the width is always equal to the dimensionality d of word representation. Each filter will be represented by only one feature after max-over-tree pooling. After a series of convolution with different filter with different heights, multiple features carry different structural information become the final representation of the input sentence. Then, this sentence representation is passed to a fully connected soft-max layer and outputs a distribution over different labels.

Neural networks often suffer from over-training. Following [27], we employ random dropout on penultimate layer [19]. in order to prevent co-adaptation of hidden units. In our experiments, we set our drop out rate as 0.5 and learning rate as 0.95 by default. Following [27], training is done through stochastic gradient descent over shuffled mini-batches with the Adadelta update rule [80].

2.2.3 Convolution on Siblings

Ancestor paths alone is not enough to capture many linguistic phenomena such as conjunction. Inspired by higher-order dependency parsing [47, 30], we also incorporate siblings for a given word in various ways. See Figure 2.2 (right) for details.

2.2.4 Combined Model

Powerful as it is, structural information still does not fully cover sequential information. Also, parsing errors (which are common especially for informal text such as online reviews) directly affect DCNN performance while sequential n-grams are always correctly observed. To best exploit both information, we want to combine both models. The easiest way of combination is to concatenate these representations together, then feed into fully connected soft-max neural networks. In these cases, combine with different feature from different type of sources could stabilize the performance. The final sentence

Category	Model	MR	SST-1	TREC	TREC-2
This work	DCNNs: ancestor	80.4†	47.7†	95.4†	88.4†
	DCNNs: ancestor+sibling	81.7†	48.3†	**95.6**†	89.0†
	DCNNs: ancestor+sibling+sequential	**81.9**	49.5	95.4†	88.8†
CNNs	CNNs-non-static [27] – baseline	81.5	48.0	93.6	86.4*
	CNNs-multichannel [27]	81.1	47.4	92.2	86.0*
	Deep CNNs [26]	-	48.5	93.0	-
Recursive NNs	Recursive Autoencoder [63]	77.7	43.2	-	-
	Recursive Neural Tensor [64]	-	45.7	-	-
	Deep Recursive NNs [22]	-	**49.8**	-	-
Recurrent NNs	LSTM on tree [84]	**81.9**	48.0	-	-
Other deep learning	Paragraph-Vec [35]	-	48.7	-	-
Hand-coded rules	SVM$_S$ [60]	-		95.0	**90.8**

Table 2.1: Results on Movie Review (MR), Stanford Sentiment Treebank (SST-1), and TREC datasets. TREC-2 is TREC with fine grained labels. †Results generated by GPU (all others generated by CPU). *Results generated from [27]'s implementation.

representation is thus:

$$\hat{\mathbf{c}} = [\underbrace{\hat{c}_a^{(1)}, ..., \hat{c}_a^{(N_a)}}_{\text{ancestors}}; \underbrace{\hat{c}_s^{(1)}, ..., \hat{c}_s^{(N_s)}}_{\text{siblings}}; \underbrace{\hat{c}^{(1)}, ..., \hat{c}^{(N)}}_{\text{sequential}}]$$

where N_a, N_s, and N are the number of ancestor, sibling, and sequential filters. In practice, we use 100 filters for each template in Figure 2.2 . The fully combined representation is 1,100-dimensional by contrast to 300-dimensional for sequential CNN.

2.3 Experiments

Table 2.1 summarizes results in the context of other high-performing efforts in the literature. We use three benchmark datasets in two categories: sentiment analysis on both Movie Review (MR) [51] and Stanford Sentiment Treebank (SST-1) [64] datasets, and question classification on TREC [37].

For all datasets, we first obtain the dependency parse tree from Stanford parser [45].[1] Different window size for different choice of convolution are shown in Figure 2.2. For the dataset without a development set (MR), we randomly choose 10% of the training data to indicate early stopping. In order to have a fare comparison with baseline CNN, we

---

[1] The phrase-structure trees in SST-1 are actually automatically parsed, and thus can not be used as gold-standard trees.

also use 3 to 5 as our window size. Most of our results are generated by GPU due to its efficiency, however CPU could potentially get better results.[2] Our implementation, on top of [27]'s code,[3] will be released.[4]

2.3.1 Sentiment Analysis

Both sentiment analysis datasets (MR and SST-1) are based on movie reviews. The differences between them are mainly in the different numbers of categories and whether the standard split is given. There are 10,662 sentences in the MR dataset. Each instance is labeled positive or negative, and in most cases contains one sentence. Since no standard data split is given, following the literature we use 10 fold cross validation to include every sentence in training and testing at least once. Concatenating with sibling and sequential information obviously improves DCNNs, and the final model outperforms the baseline sequential CNNs by 0.4, and ties with [84].

Different from MR, the Stanford Sentiment Treebank (SST-1) annotates finer-grained labels, very positive, positive, neutral, negative and very negative, on an extension of the MR dataset. There are 11,855 sentences with standard split. Our model achieves an accuracy of 49.5 which is second only to [22].

2.3.2 Question Classification

In the TREC dataset, the entire dataset of 5,952 sentences are classified into the following 6 categories: abbreviation, entity, description, location and numeric. In this experiment, DCNNs easily outperform any other methods even with ancestor convolution only. DCNNs with sibling achieve the best performance in the published literature. DCNNs combined with sibling and sequential information might suffer from overfitting on the training data based on our observation. One thing to note here is that our best result even exceeds SVM_S [60] with 60 hand-coded rules.

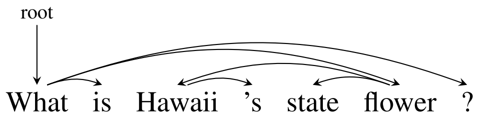
The TREC dataset also provides subcategories such as numeric:temperature, numeric:distance, and entity:vehicle. To make our task more realistic and challenging, we also test the proposed model with respect to the 50 subcategories. There are obvious

---

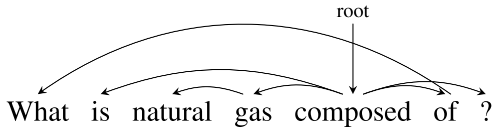
[2] GPU only supports `float32` while CPU supports `float64`.

[3] `https://github.comw/yoonkim/CNN_sentence`

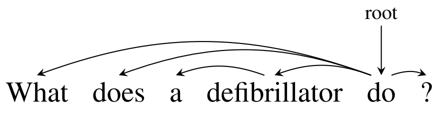
[4] `https://github.com/cosmmb/DCNN`



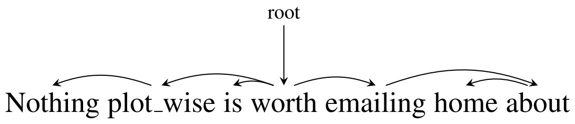
(a) enty ⇒ loc



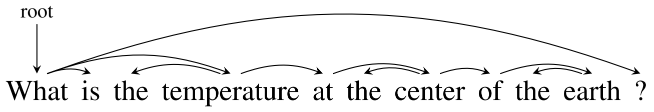
(b) enty ⇒ desc



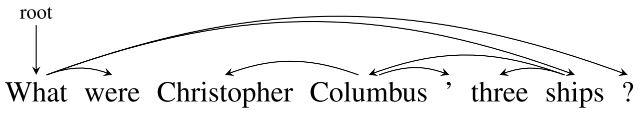
(c) desc ⇒ enty



(d) mild negative ⇒ mild positive

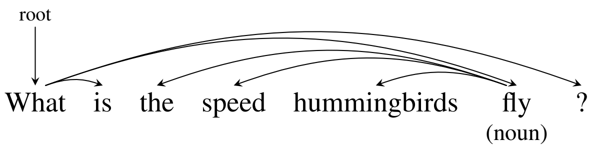


(e) NUM:temp ⇒ NUM:dist

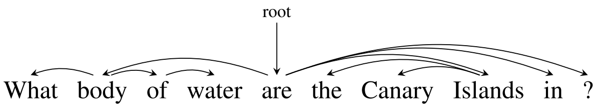


(f) ENTY:veh ⇒ LOC:other

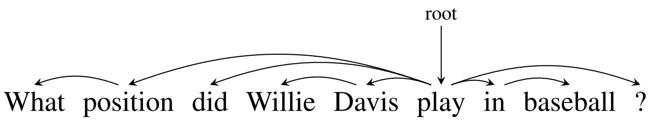
Figure 2.3: Examples from TREC (a–c), SST-1 (d) and TREC with fine-grained label (e–f) that are misclassified by the baseline CNN but correctly labeled by our DCNN. For example, (a) should be *entity* but is labeled *location* by CNN.



(a) num ⇒ enty



(b) loc ⇒ enty



(c) hum ⇒ enty

Figure 2.4: Examples from TREC datasets that are misclassified by DCNN but correctly labeled by baseline CNN. For example, (a) should be *numerical* but is labeled *entity* by DCNN.

improvements over sequential CNNs from the last column of Table 2.1. Like ours, [60] is a tree-based system but it uses constituency trees compared to ours dependency trees. They report a higher fine-grained accuracy of 90.8 but their parser is trained *only* on the QuestionBank [24] while we used the standard Stanford parser trained on both the Penn Treebank and QuestionBank. Moreover, as mentioned above, their approach is rule-based while ours is automatically learned.

2.3.3 Discussions and Examples

Compared with sentiment analysis, the advantage of our proposed model is obviously more substantial on the TREC dataset. Based on our error analysis, we conclude that this is mainly due to the difference of the parse tree quality between the two tasks. In sentiment analysis, the dataset is collected from the *Rotten Tomatoes* website which includes many irregular usage of language. Some of the sentences even come from languages other than English. The errors in parse trees inevitably affect the classification accuracy. However, the parser works substantially better on the TREC dataset since all questions are in formal written English, and the training set for Stanford parser[5] already includes the QuestionBank [24] which includes 2,000 TREC sentences.

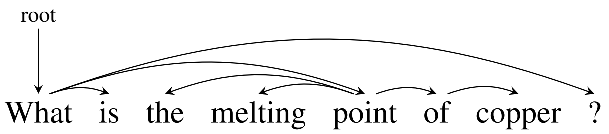
Figure 2.3 visualizes examples where CNN errs while DCNN does not. For example, CNN labels (a) as *location* due to “Hawaii” and “state”, while the long-distance backbone “What – flower” is clearly asking for an *entity*. Similarly, in (d), DCNN captures the obviously negative tree-based trigram “Nothing – worth – emailing”. Note that our model also works with non-projective dependency trees such as the one in (b). The last two examples in Figure 2.3 visualize cases where DCNN outperforms the baseline CNNs in fine-grained TREC. In example (e), the word “temperature” is at second from the top and is root of a 8 word span “the ... earth”. When we use a window of size 5 for tree convolution, every words in that span get convolved with “temperature” and this should be the reason why DCNN get correct.

Figure 2.4 showcases examples where baseline CNNs get better results than DCNNs. Example (a) is misclassified as *entity* by DCNN due to parsing/tagging error (the Stanford parser performs its own part-of-speech tagging). The word “fly” at the end of the sentence should be a verb instead of noun, and “hummingbirds fly” should be a relative clause modifying “speed”.

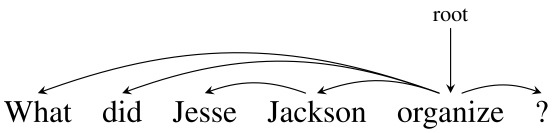
There are some sentences that are misclassified by both the baseline CNN and DCNN. Figure 2.5 shows three such examples. Example (a) is not classified as *numerical* by both methods due to the ambiguous meaning of the word “point” which is difficult to capture by word embedding. This word can mean location, opinion, etc. Apparently, the numerical aspect is not captured by word embedding. Example (c) might be an annotation error.

---

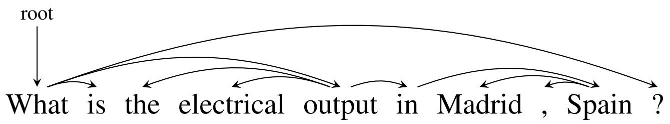
[5] `http://nlp.stanford.edu/software/parser-faq.shtml`



(a) num ⇒ enty and desc



(b) hum ⇒ enty and enty



(c) enty ⇒ num and num

Figure 2.5: Examples from TREC datasets that are misclassified by both DCNN and baseline CNN. For example, (a) should be *numerical* but is labeled *entity* by DCNN and *description* by CNN.

Chapter 3: Jointly Training for Sequential Labeling and Sentence Level Classification

Sentence-level classification and sequential labeling are two fundamental tasks in language understanding. While these two tasks are usually modeled separately, in reality, they are often correlated, for example in intent classification and slot filling, or in topic classification and named-entity recognition. In order to utilize the potential benefits from their correlations, we propose a jointly trained model for learning the two tasks simultaneously via Long Short-Term Memory (LSTM) networks. This model predicts the sentence-level category and the word-level label sequence from the stepwise output hidden representations of LSTM. We also introduce a novel mechanism of "sparse attention" to weigh words differently based on their semantic relevance to sentence-level classification.

The work presented in this chapter has been published in the Proceedings of INTERSPEECH 2017 [43].

3.1 Introduction

We consider the dichotomy between two important tasks in spoken language understanding: the global task of sentence-level classification, such as intention or sentiment, and the local task of sequence labeling or semantic constituent extraction, such as slot filling or named-entity recognitions (NER). Conventionally, these two tasks are modeled separately, with algorithms such as SVM [5] for the former, and Conditional Random Fields (CRF) [33] or structured perceptron [9] for the latter.

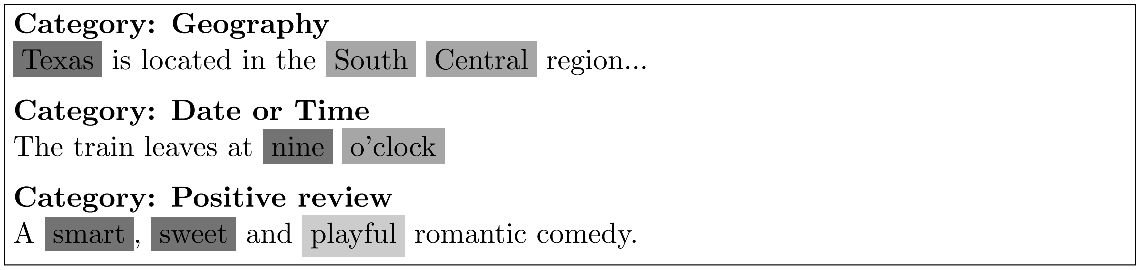
In reality, however, these two tasks are often correlated. Consider the problems of sentence topic classification and NER in Figure 3.1. Different sentence-level classifications provide different priors for each word's label; for example if we know the sentence is about IT news then the word "Apple" is almost certainly about the company. Likewise, different word-level label sequence also influence the sentence-level category distribution; for example if we know the word "Apple" is about fruits then the sentence topic is more

Topic: *IT news*
Apple's new iPad will be out soon ... (Company)

Topic: *Agriculture reports*
Apple harvest in Washington state .... (Fruit)

Topic: *Tourist information*
"Big *Apple*" is a nickname of NYC ... (Location)

Figure 3.1: The same word "Apple" has different meanings (see tags in blue) in different different topics.



Category: Geography
Texas is located in the South Central region...

Category: Date or Time
The train leaves at nine o'clock

Category: Positive review
A smart, sweet and playful romantic comedy.

Figure 3.2: Examples of various magnitudes of attentions for sentence level classification. Darker words are more important.

likely to be agricultural.

Indeed, previous work has explored joint modeling between the two tasks. For example, Jeong and Lee [23] propose a discrete CRF model for joint training of sentence-level classification and sequence labeling. A follow-up work [75] leverages the feature representation power of convolution neural networks (CNNs) to make the CRF model generalize better for unseen data. However, the above CRF-based methods still suffer from the following limitations: firstly, although CRF is trained globally, it still lacks the ability for capturing long-term memory at each time step which is crucial for sentence-level classification and sequential labeling problem; secondly, the CNNs-based CRF [75] only use CNNs for nonlinear local feature extraction. But globally, it is still a linear model which has limited generalization power to unseen data.

In order to overcome the two above issues, we propose a novel LSTM-based model to jointly train sentence-level classification and sequence labeling, which incorporates long-term memory and nonlinearity both locally and globally. We make the following contributions:

- Our Long Short-Term Memory (LSTM) [20] network analyzes the sentence using features extracted by a convolutional layer. Basically, each word-level label is greedily determined by the hidden representation from LSTM in each step, and the global sentence category is determined by an aggregate (e.g., pooling) of hidden representations from all steps (Sec. 3.3.1).
- We also propose a novel **sparse attention model** which promotes important words in a given sentence and demotes semantically vacuous words (Fig. 3.2; Sec. 3.3.2).
- Finally, we develop a **latent variable** version of our jointly trained model which can be trained for the single task of sentence classification by treating word-level labels as latent information (Sec. 3.3.3).

3.2 LSTM for Labeling and Classification

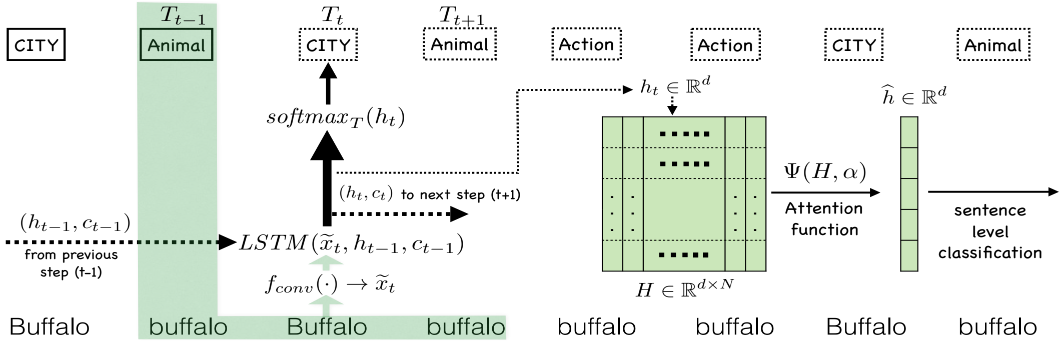


Figure 3.3: Jointly trained sequence labeling and sentence classification. The green mask means that convolution operate between one previous tag and two surrounding words (when window size is 3). Ψ is the attention function in Eq. 3.5. Here the sentence length is N=8.

We use LSTM to model the representation of the sentence at each word step, which is powerful in modeling sentence semantics [66, 7]. Assume the length of the sentence is N. LSTM represents the meaning of the sentence at the t-th word by a pair of vectors $(h_t, c_t) \in (\mathbb{R}^d, \mathbb{R}^d)$, where h_t is the output hidden representation of the word, c_t is the memory of the network, and d is the number of dimensions of the representation space:

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ \hat{c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W_{\text{LSTM}} \cdot [x_t, h_{t-1}] \tag{3.1}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{3.2}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.3}$$

Here x_t represents the t-th word, which is usually the word embedding vector, and vectors i_t, f_t, and o_t are gated activations that control flow of hidden information. The separation of the output representation h_t from its internal memory c_t, in principle, makes the knowledge about the sentence prefix be remembered by the network for longer time to interfere with the current output at word x_t. These carefully designed activation gates alleviate the problem of vanishing gradient problem in vanilla recurrent neural network models.

In the task of sequence labeling, the label for each word is determined by hidden representation h_t. As described in Eq. 3.3, at time step t, we will get an output h_t which represents all the current information. The probability distribution of the t-th word's label is calculated by $\mathit{softmax}_T(W_T \cdot h_t)$, where weight $W_T \in \mathbb{R}^{d \times |T|}$ maps h_t to the space of the labels, and $|T|$ is the number of possible labels. For the sequence labeling problem, the loss ℓ_{seq} is calculates as the sum of the Negative Log-Likelihood (NLL) over this label distribution $\mathit{softmax}_T(h_t)$ at each time step.

In the task of sentence classification, the entire sentence representation is obtained by aggregating all history outputs h_t which are stored in $H \in \mathbb{R}^{d \times N}$, where N is the length of sequence. Similar to CNNs, max pooling [10, 27, 40] operates over the history outputs H to get the average activation $\hat{h} = \mathit{pooling}(H)$ summarizing the entire outputs. Then, this sentence representation is passed to a fully connected soft-max layer which outputs a distribution over sentence categories.

In above two different tasks, the hidden representation h_t functions as a key component in different, separate ways. In many cases, when sequence-level labels and sentence categories are both available we should use both information within the same framework by joint training the two tasks.

3.3 Joint Sequence Classification & Labeling

3.3.1 Joint Training Model

Sentence 1	I	want	to	go	from	Denver	to	Boston	today	Category
Slots	O	O	O	O	O	B-FromCity	O	B-ToCity	B-Date	Flight
Sentence 2	to	come	back	to	Los	Angeles	on	Friday	evening	Category
Slots	O	O	O	O	B-ReturnCity	I-ReturnCity	O	B-RETURN.DAY	B-RETURN.PERIODOFDAY	Return

Table 3.1: Examples of ATIS sentences and annotated slots and categories.

We aim at developing a model which could learn the label sequence and sentence-level category simultaneously. To this end, we modify the standard LSTM structure to generate the word labels on the fly based on output h_t, and predict the sentence category with the sequence of h_t, $t = 1, \ldots, N$.

In LSTM, at time step t, only information of the current word x_t is being fed into the network. This mechanism overlooks the problem that the meanings of the same word in different contexts might vary (*cf.* Fig. 3.1). In particular, words that follow x_t are not represented at step t in LSTM.

This observation motivates us to include more contextual information around the current word and previous tags as part of the input to LSTM. We employ convolutional neural network (CNN) to automatically mine the meaningful knowledge from both the context of word x_t and previous tags T_{t-1}, and use this knowledge as the input for LSTM. We formally define the new input for LSTM as:

$$\widetilde{x}_t = f_{\text{conv}}(W_{\text{conv}} \cdot x_{t,k} + b_{\text{conv}}) \tag{3.4}$$

where f_{conv} is a non-linear activation function like rectified linear unit (ReLU) or sigmoid, and $x_{t,k}$ is a vector representing the context of word x_t and previous tags $T_{t-k}...T_{t-1}$, e.g., the concatenation of the surrounding words and previous tags,
$x_{t,k} = [x_{t-k}, \ldots, x_{t+k}, T_{t-k}, \ldots, T_{t-1}]$ in Eq. 3.4, where the convolution window size is $2k + 1$ and T_{t-1} represents the embedding for tag T_{t-1}. W_{conv} is the collection of filters applied on the context. During the convolution, each row of W_{conv} is a filter that will be fired if it matches some useful pattern in the input context. The convolutional layer functions as a feature extraction tool to learn meaningful representation from both words and tags automatically. Note that the above contextual representation is different from

bi-LSTM which only learns surrounding contextual of a given word. However, our model can learn both contextual and label information by convolution.

In our model, the joint training between sentence-level classification and sequential labeling is done in two directions: in forward pass, word label representation sequence is used for sentence level classification; during backward training, the sentence level prediction errors also fine-tune the label sequence.

Fig. 3.3 illustrates our proposed model with one classical NLP exemplary sentence which only contains the word “buffalo” as the running example. At each time step, we first use the convolutional layer as a feature extractor to get the nonlinear feature combinations from the embeddings of words and tags. In the case of window size equaling to 1, the convolution operates over the $t-1$, t and $t+1$ words and the $t-1$ tag. The contextual representation $x_{t,1} = [x_{t-1}, x_t, x_{t+1}, T_{t-1}]$ is then fed into the convolution layer to find feature representation $\widetilde{x}_t$ following Eq. 3.4. $\widetilde{x}_t$ is used as the input for the following LSTM to generate h_t and c_t, based on history information h_{t-1} and previous cell information c_{t-1}.

The example in Fig 3.3 is a grammatically correct sentence in English [54]. The word “buffalo” has three different meanings: Buffalo, NY (city), bison (animal), or bully (action). It is hard for standard LSTM to differentiate the different meanings of “buffalo” in different time step since the x_t is the same all the time. However, in our case, instead of simply using word representation x_t itself, we also consider the contextual information with their tags through convolution from $x_{t,k}$ (Eq. 3.4).

3.3.2 Sparse Attention

In the previous section, the sentence-level representation $\hat{h}$ is obtained by a simple average pooling on H. This process assumes every words contribute to the sentence equally, which is not the case in many scenarios. Fig. 3.2 shows a few examples of different words with different magnitudes of attention in different sentence categories. In order to incorporate these differences into consideration, we further propose a novel sparse attention constraint for sentence level classification. The sparse constraint assigns bigger weights for important words and lower the weights or even totally ignores the less meaningful words such like “the” or “a”. The attention-based sentence-level representation is formulated as follows:

$$\widehat{h} = \Psi(H, \alpha) = \sum_{h_t \in H} \psi(h_t \cdot \alpha) h_t \tag{3.5}$$

where $\alpha \in \mathbb{R}^d$ is an attention measurement which decides the importances for different inputs based on their semantics h_i. This importance is calculated through a nonlinear function ψ which can be sigmoid or ReLU and we use sigmoid in our case.

Sparse Autoencoders [50, 44] show that getting sparse representations in the hidden layers can further improve the performance. In our model, we apply similar sparse constraints by first calculating the average attention over the training samples in the same batch:

$$\hat{\rho}_t = \frac{1}{m} \sum_{i=1}^{m} \psi(h_t^i \cdot \alpha) \tag{3.6}$$

where m is the size of training batch, and h_t^i is the output of LSTM at step t for example i. In order to keep the above attention within a fix budget, similar to Sparse Autoencoders [50], we have an extra penalty term as follows:

$$KL(\rho || \hat{\rho}_t) = \rho \log \frac{\rho}{\hat{\rho}_t} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_t},$$

where ρ is the desired sparsity of the attention. This penalty term uses KL divergence to measure the difference between two distributions. Then our new objective is defined as follows:

$$\ell_{\text{sparse}}(\cdot) = \ell_{\text{seq}}(\cdot) + \ell_{\text{sent}}(\cdot) + \beta \sum_{t=1}^{N} KL(\rho || \hat{\rho}_t), \tag{3.7}$$

where N is the number of hidden units, ℓ_{seq} is the sequence labeling loss, and ℓ_{sent} is the sentence classification loss. β controls the weight of the sparsity penalty term. Note that the term $\hat{\rho}_t$ is implicitly controlled by optimizing α and output h_t.

3.3.3 Label Sequence as Latent Variable

In practice, it is expensive to annotate the data with both sequential label and sequence category. In many cases, the sequence labels are missing since it requires significantly more efforts to annotate the labels word-by-word. However, even without this sequence labeling information, it is still helpful if we could utilize the possible hidden labels for each words.

In our proposed model, we could consider the sentence-level classification task as the major learning objective and treat the unknown sequence labels as latent information. The only adaptation we need to make is to replace the T_t (tag embedding) with h_t (output at time step t) in $x_{t,k}$. In this case, we exploit the latent meaning representation to further improve the feature extraction of the convolutional layer.

3.4 Experiments

Model	Sent. Acc.
CNN non-static [27]	93.6
CNN multichannel [27]	92.2
Deep CNN [26]	93.0
Independent LSTM (baseline)	92.2
latent LSTM + CNN	92.6
latent LSTM + CNN + attention	93.4
latent LSTM + CNN + sparse att.	**94.0**

Figure 3.4: Sentence-level accuracy of our latent-variable model on TREC, compared with various neural network-based models.

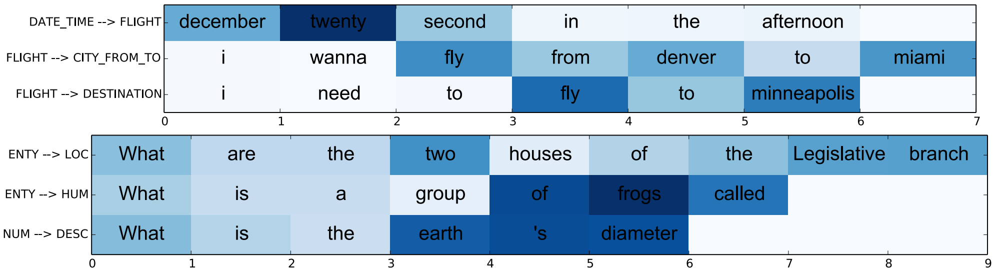


Figure 3.5: Examples that we outperform the model without sparse attention (LSTM+CNN). Higher weights are darker.

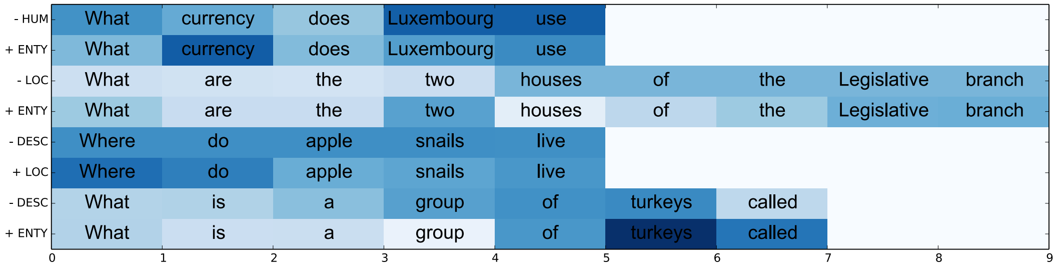


Figure 3.6: Comparison between softmax-based attention (upper) and sparse attention (lower) for for some examples. The sign '-' means mis-classified label, and '+' for the correct label. Darker blue represents higher weights.

We start by evaluating the performance on a conventional joint learning task (Sec. 3.4.1). Then we show the performance when we treat the label sequence as hidden information in Sec. 3.4.2. We also analyze some concrete examples to show the performance of the sparse attention constraint in our model (Sec. 3.4.3).

In the experiments, we set the convolution window sizes as 3, 5, and 7. There are 100 different filters for each window size. Word embeddings are randomly initialized in the ATIS experiments. In the TREC experiments, we use 300 dimension pre-trained word embeddings. We use AdaDelta for optimization [80] with learning rate 0.001 and minibatch 16. The weights in our framework are uniformly randomly initialized between $[-0.05, 0.05]$. We use ReLu at the convolutional layer and also regularize the feature with dropout 0.5 [19].

We evaluate on ATIS[23] and TREC [37] datasets. We follow the TriCRF paper[23] in our evaluation on ATIS. There are 5,138 dialogs with 21 types of intents and 110 types of slots annotated [1]. This dataset is first used for joint learning model with both slot and intent labels in the joint learning experiments (Sec. 3.4.1). We later use it for evaluating the performance when the slot labels are not available (Sec. 3.4.2). The TREC dataset[2] is a factoid question classification dataset, with 5,952 sentences being classified into 6 categories. Since only the sentence-level categories are annotated, we treat the unknown tags as latent information in our experiments (Sec. 3.4.2).

---

[1] Note we do not compare our performance with [17] since their ATIS dataset is not published and is different from our ATIS dataset.

[2] http://cogcomp.cs.illinois.edu/Data/QA/QC/

3.4.1 Joint Training Experiments

We first perform the joint training experiments on the ATIS dataset. Tab. 3.1 shows two examples from the ATIS dataset. As mentioned in Sec. 3.3.2, only a few keywords in the two sentences in Tab. 3.1 are relevant to determining the category, i.e., locations (cities) and date for sentence I, and locations, date, and time for sentence II. Our model should be able to recognize these important keywords and predict the categories mostly based on them. Once the model knows the tags of the words in the sentence, it is straightforward to determine the categories of the sentence.

We show the performance of our model comparing with other individually trained or jointly trained models in Tab. 3.2. We observe that due to its strong generalization power, neural networks based models outperform discrete models with an impressive margin. Our jointly trained neural model achieves the best performance, with an F1 boost of ∼2 points in slot filling. After adding the sparse attention constraint, our model achieves the lowest error rate for the sentence classification on ATIS.

3.4.2 Label Sequence as Latent Variable

We further show the performance of our jointly trained model when the sequence label information is missing. Tab. 3.2 (bottom) shows the results on ATIS dataset. There is a small increase of error rate when sequence labels are unobserved, but our model still outperforms existing models. Similarly, Fig. 3.4 compares our latent-variable model with conventional (non-latent) neural models on TREC dataset (in sentence category accuracy). Our model outperforms others after adding sparse attention.

3.4.3 Sparsity Visualization

In Fig. 3.5, we compare the sparse attention model to the model without sparse constraints. We list a few examples that the sparse attention is better than the one without sparse attention constraint. The labels on the right side are mis-classified by the model without sparse attention constraint. The label on the left side of the arrow is the ground truth.

In Fig. 3.6, we also compare the difference between two attention mechanisms: softmax-based attetion and sparse attention. From the first sentence, we can tell that softmax-based attention puts more emphasis on “Luxembourg” while sparse attention prefers “currency” which leads to the correct prediction of entity instead of human. In some cases, like the third example in Fig. 3.6, softmax-based sometimes gets confused by distributing the probabilities flatly. Compared with the attention model between dual sentences, the phenomenon of flat distribution is more obvious in single sentence attention. Similar results can be found in the first figure in [8] as the word “run” is aligned to many unrelated words. It is possible that in single sentence attention, the softmax-based attention is easier to get confused since there is no obvious alignment or corresponding relationship between words for a given sentence or the words and their corresponding sentence category.

	Model	Slot	Intent
Independent Model	CRF [23]	$90.67^{\dagger}$	$7.91^{\dagger}$
	CNN [75]	$92.43^{\dagger}$	$6.65^{\dagger}$
CRF Joint Model	TriCRF [23]	94.42	6.93
	CNN TriCRF [75]	95.42	5.91
Independent Model	Vanilla LSTM (baseline)	$93.74^{\dagger}$	$7.21^{\dagger}$
Jointly Trained Model (Secs. 3.3.1 & 3.3.2)	+ joint	95.54	6.32
	+ joint + CNN	**97.35**	5.96
	+ joint + CNN + attention	96.73	5.71
	+ joint + CNN + sparse att.	96.98	**5.12**
Independent Model	Vanilla LSTM (baseline)	-	$7.21^{\dagger}$
Seq. Label as Latent Var. (Sec. 3.3.3)	+ joint + CNN	-	6.43
	+ joint + CNN + attention	-	5.61
	+ joint + CNN + sparse att.	-	**5.42**

Table 3.2: **Main results:** Our jointly trained models compared with various independent (marked †) and existing joint models. The "Slot" column shows the F1 score of sequence labeling, and "intent" shows the error rates for sentence classification.

Chapter 4: Group Sparse Convolutional Neural Networks for Learning Label Hierarchy

Question classification is an important task with wide applications. However, traditional techniques treat questions as general sentences, ignoring the corresponding answer data. In order to consider answer information into question modeling, we first introduce novel group sparse autoencoders which refine question representation by utilizing group information in the answer set. We then propose novel group sparse CNNs which naturally learn question representation with respect to their answers by implanting group sparse autoencoders into traditional CNNs.

The work presented in this chapter has been published in the Proceedings of Association for Computational Linguistics 2017 [41].

4.1 Problems in Question Answering

Question classification has applications in many domains ranging from question answering to dialog systems, and has been increasingly popular in recent years. Several recent efforts [27, 26, 40] treat questions as general sentences and employ Convolutional Neural Networks (CNNs) to achieve remarkably strong performance in the TREC question classification task.

Most existing approaches to this problem simply use existing sentence modeling frameworks and treat questions as general sentences, without any special treatment. For example, several recent efforts employ Convolutional Neural Networks (CNNs) to achieve remarkably strong performance in the TREC question classification task as well as other sentence classification tasks such as sentiment analysis [27, 26, 40].

We argue, however, that those general sentence modeling frameworks neglect two unique properties of question classification. First, different from the flat and coarse categories in most sentence classification tasks (i.e. sentimental classification), question classes often have a hierarchical structure such as those from the New York State DMV

1: Driver License/Permit/Non-Driver ID
a: *Apply for original* (49 questions)
b: *Renew or replace* (24 questions)
...
2: Vehicle Registrations and Insurance
a: *Buy, sell, or transfer a vehicle* (22 questions)
b: *Reg. and title requirements* (42 questions)
...
3: Driving Record / Tickets / Points
...

Figure 4.1: Examples from NYDMV FAQs. There are 8 top-level categories, 47 sub-categories, and 537 questions (among them 388 are *unique*; many questions fall into multiple categories).

FAQ[1] Another unique aspect of question classification is the well prepared answers for each question or question category. These answer sets generally cover a larger vocabulary (than the questions themselves) and provide richer information for each class. We believe there is a great potential to enhance question representation with extra information from corresponding answer sets.

To exploit the hierarchical and overlapping structures in question categories and extra information from answer sets, we consider dictionary learning [6, 55] which is a common approach for representing samples from many correlated groups with external information. This learning procedure first builds a dictionary with a series of grouped bases. These bases can be initialized randomly or from external data (from the answer set in our case) and optimized during training through Sparse Group Lasso (SGL) [61].

To apply dictionary learning to CNN, we first develop a neural version of SGL, *Group Sparse Autoencoders* (GSAs), which to the best of our knowledge, is the first full neural model with group sparse constraints. The encoding matrix of GSA (like the dictionary in SGL) is grouped into different categories. The bases in different groups can be either initialized randomly or by the sentences in corresponding answer categories. Each question sentence will be reconstructed by a few bases within a few groups. GSA can use either linear or nonlinear encoding or decoding while SGL is restricted to be

---

[1] Crawled from `http://nysdmv.custhelp.com/app/home`.

linear. Eventually, to model questions with sparsity, we further propose novel *Group Sparse Convolutional Neural Networks* (GSCNNs) by implanting the GSA onto CNNs, essentially enforcing group sparsity between the convolutional and classification layers. This framework is a jointly trained neural model to learn question representation with group sparse constraints from both question and answer sets.

4.2 Group Sparse Autoencoders

4.2.1 Sparse Autoencoders

Autoencoder [4] is an unsupervised neural network which learns the hidden representations from data. When the number of hidden units is large (e.g., bigger than input dimension), we can still discover the underlying structure by imposing sparsity constraints, using sparse autoencoders (SAE) [50]:

$$J_{\text{sparse}}(\rho) = J + \alpha \sum_{j=1}^{s} KL(\rho \| \hat{\rho}_j) \tag{4.1}$$

where J is the autoencoder reconstruction loss, ρ is the desired sparsity level which is small, and thus $J_{\text{sparse}}(\rho)$ is the sparsity-constrained version of loss J. Here α is the weight of the sparsity penalty term defined below:

$$KL(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{4.2}$$

where

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} h_j^i$$

represents the average activation of hidden unit j over m examples (SAE assumes the input features are correlated).

As described above, SAE has a similar objective to traditional sparse coding which tries to find sparse representations for input samples. Besides applying simple sparse constraints to the network, group sparse constraints is also desired when the class categories are structured and overlapped. Inspired by group sparse lasso [79] and sparse group lasso [61], we propose a novel architecture below.

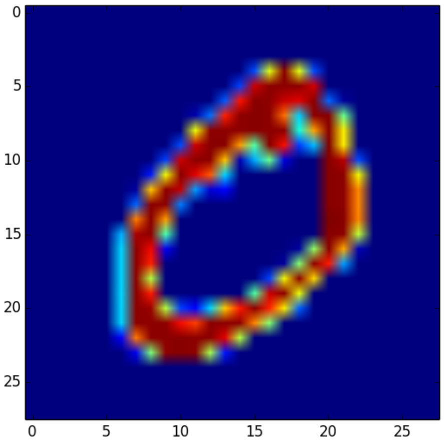


Figure 4.2: The input figure with hand written digit 0.

4.2.2 Group Sparse Autoencoders

Group Sparse Autoencoder (GSA), unlike SAE, categorizes the weight matrix into different groups. For a given input, GSA reconstructs the input signal with the activations from only a few groups. Similar to the average activation $\hat{\rho}_j$ for sparse autoencoders, GSA defines each grouped average activation for the hidden layer as follows:

$$\hat{\eta}_p = \frac{1}{mg} \sum_{i=1}^{m} \sum_{l=1}^{g} \|h_{p,l}^{i}\|_2 \tag{4.3}$$

where g represents the size of each group, and $\hat{\eta}_j$ first sums up all the activations within p^{th} group, then computes the average p^{th} group respond across different samples' hidden activations.

Similar to Eq. 4.2, we also use KL divergence to measure the difference between estimated intra-group activation and global group sparsity:

$$KL(\eta \| \hat{\eta}_p) = \eta \log \frac{\eta}{\hat{\eta}_p} + (1-\eta) \log \frac{1-\eta}{1-\hat{\eta}_p} \tag{4.4}$$

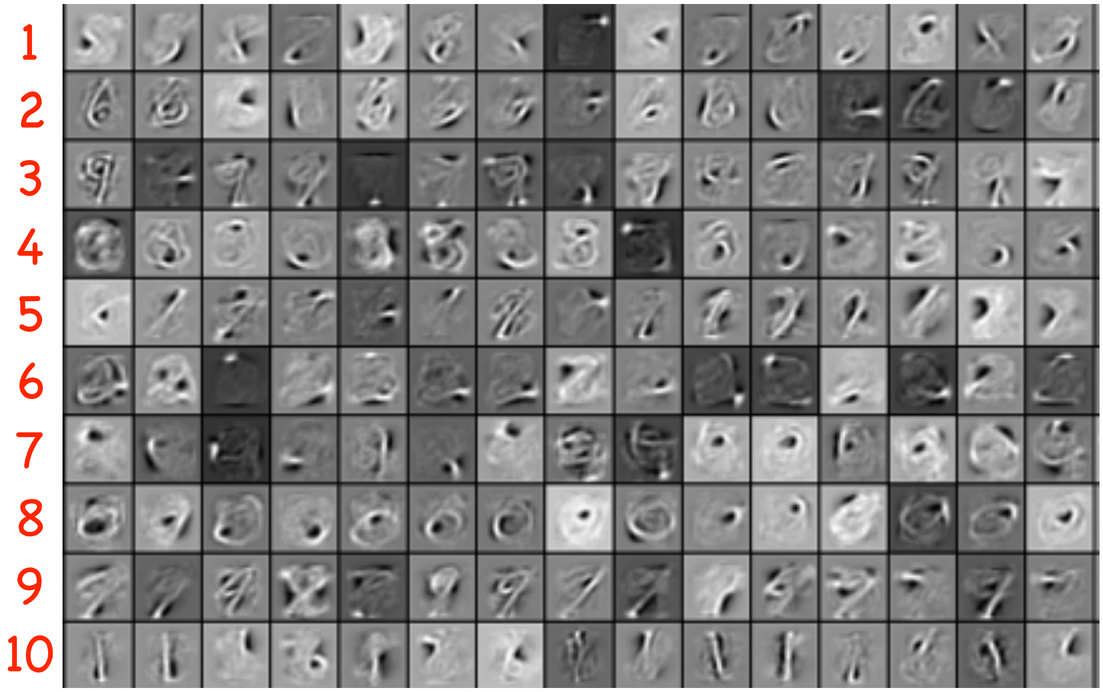


Figure 4.3: The visualization of trained projection matrix **W** on MNIST dataset. Different rows represent different groups of **W** in Eq. 4.5. For each group, we only show the first 15 (out of 50) bases. The red numbers on the left side are the indices of 10 different groups.

where G is the number of groups. Then the objective function of GSA is:

$$\begin{aligned} J_{\text{groupsparse}}(\rho, \eta) = J &+ \alpha \sum_{j=1}^{s} KL(\rho \| \hat{\rho}_j) \\ &+ \beta \sum_{p=1}^{G} KL(\eta \| \hat{\eta}_p) \end{aligned} \tag{4.5}$$

where ρ and η are constant scalars which are our target sparsity and group-sparsity levels, resp. When α is set to zero, GSA only considers the structure between difference groups. When β is set to zero, GSA is reduced to SAE.

4.2.3 Visualizing Group Sparse Autoencoders

In order to have a better understanding of GSA, we use the MNIST dataset to visualize GSA's internal parameters. Fig. 4.3, Fig. 4.4 and Fig. 4.5 illustrate the projection matrix and the corresponding hidden activations. We use 10,000 training samples. We set the size of the hidden layer to 500 with 10 groups. Fig. 4.2 visualizes the input image for hand written digit 0.

In Fig. 4.3, we find similar patterns within each group. For example, group 8 has different forms of digit 0, and group 9 includes different forms of digit 7. However, it is difficult to see any meaningful patterns from the projection matrix of basic autoencoders in Fig. 4.4.

Fig. 4.5(a) shows the hidden activations with respect to the input image of digit 0. The patterns of the 10^{th} row in Fig. 4.3 are very similar to digit 1 which is very different from digit 0 in shape. Therefore, there is no activation in group 10 in Fig. 4.5(a). The majority of hidden layer activations are in groups 1, 2, 6 and 8, with group 8 being the most significant. When compared to the projection matrix visualization in Fig. 4.3, these results are reasonable since the 8^{th} row has the most similar patterns of digit 0. However, we could not find any meaningful pattern from the hidden activations of basic autoencoder as shown in Fig. 4.5(b).

GSA could be directly applied to small image data (e.g. MINIST dataset) for pre-training. However, in tasks which prefer dense semantic representations (e.g. sentence classification), we still need CNNs to learn the sentence representation automatically. In order to combine advantages from GSA and CNNs, we propose Group Sparse Convolutional Neural Networks below.

4.3 Group Sparse CNNs

CNNs were first proposed by [36] in computer vision and adapted to NLP by [10]. Recently, many CNN-based techniques have achieved great successes in sentence modeling and classification [27, 26].

Following sequential CNNs, one dimensional convolutions operate the convolution kernel in sequential order $\mathbf{x}_{i,j} = \mathbf{x}_i \oplus \mathbf{x}_{i+1} \oplus \cdots \oplus \mathbf{x}_{i+j}$, where $\mathbf{x}_i \in \mathbb{R}^e$ represents the e dimensional word representation for the i-th word in the sentence, and $\oplus$ is the concatenation operator. Therefore $\mathbf{x}_{i,j}$ refers to concatenated word vector from the i-th word to the $(i+j)$-th word in sentence.

A convolution operates a filter $\mathbf{w} \in \mathbb{R}^{n \times e}$ to a window of n words $\mathbf{x}_{i,i+n}$ with bias term b' by $a_i = \sigma(\mathbf{w} \cdot \mathbf{x}_{i,i+n} + b')$ with non-linear activation function σ to produce a new feature. The filter $\mathbf{w}$ is applied to each word in the sentence, generating the feature map $\mathbf{a} = [a_1, a_2, \cdots, a_L]$ where L is the sentence length. We then use $\hat{a} = \max\{\mathbf{a}\}$ to represent the entire feature map after max-pooling.

In order to capture different aspects of patterns, CNNs usually randomly initialize a set of filters with different sizes and values. Each filter will generate a feature as described above. To take all the features generated by N different filters into count, we use $\mathbf{z} = [\hat{a_1}, \cdots, \hat{a_N}]$ as the final representation. In conventional CNNs, this $\mathbf{z}$ will be directly fed into classifiers after the sentence representation is obtained, e.g. fully connected neural networks [27]. There is no easy way for CNNs to explore the possible hidden representations with underlaying structures.

In order to exploit these structures, we propose Group Sparse Convolutional Neural Networks (GSCNNs) by placing one extra layer between the convolutional and the classification layers. This extra layer mimics the functionality of GSA from Section 4.2. Shown in Fig. 4.6, after the conventional convolutional layer, we get the feature map $\mathbf{z}$ for each sentence. In stead of directly feeding it into a fully connected neural network for classification, we enforce the group sparse constraint on $\mathbf{z}$ in a way similar to the group sparse constraints on hidden layer in GSA from Sec. 4.2. Then, we use the sparse hidden representation $\mathbf{h}$ in Eq. 4.5 as the new sentence representation, which is then fed into a fully connected neural network for classification. The parameters $\mathbf{W}$ in Eq. 4.5 will also be fine tunned during the last step.

Different ways of initializing the projection matrix in Eq. 4.5 can be summarized below:

- **Random Initialization**: When there is no answer corpus available, we first randomly initialize N vectors to represent the group information from the answer set. Then we cluster these N vectors into G categories with g centroids for each category. These centroids from different categories will be the initialized bases for projection matrix $\mathbf{W}$ which will be learned during training.
- **Initialization from Questions**: Instead of using random initialized vectors, we

can also use question sentences for initializing the projection matrix when the answer set is not available. We need to pre-train the sentences with CNNs to get the sentence representation. We then select G largest categories in terms of number of question sentences. Then we get g centroids from each category by k-means. We concatenate these $G \times g$ vectors to form the projection matrix.

- **Initialization from Answers**: This is the most ideal case. We follow the same procedure as above, with the only difference being using the answer sentences in place of question sentences to pre-train the CNNs.

4.4 Experiments

Since there is little effort to use answer sets in question classification, we did not find any suitable datasets which are publicly available. We collected two datasets ourselves and also used two other well-known ones. These datasets are summarized in Table 4.1. INSURANCE is a private dataset we collected from a car insurance company's website. Each question is classified into 319 classes with corresponding answer data. All questions which belong to the same category share the same answers. The DMV dataset is collected from New York State the DMV's FAQ website. The YAHOO Ans dataset is only a subset of the original publicly available YAHOO Answers dataset [15, 58]. Though not very suitable for our framework, we still included the frequently used TREC dataset (factoid question type classification) for comparison.

We only compare our model's performance with CNNs for two following reasons: we consider our "group sparsity" as a modification to the general CNNs for grouped feature selection. This idea is orthogonal to any other CNN-based models and can be easily applied to them; in addition, as discussed in Sec. 4, we did not find any other model in comparison with solving question classification tasks with answer sets.

There is crucial difference between the INSURANCE and DMV datasets on one hand and the YAHOO set on the other. In INSURANCE and DMV, all questions in the same (sub)category share the same answers, whereas YAHOO provides individual answers to each question.

For multi-label classification (INSURANCE and DMV), we replace the softmax layer in CNNs with a sigmoid layer which predicts each category independently while softmax

Datasets	C_t	C_s	N_{data}	N_{test}	N_{ans}	Multi-label
TREC	6	50	5952	500	-	No
INSURANCE	-	319	1580	303	2176	Yes
DMV	8	47	388	50	2859	Yes
YAHOO Ans	27	678	8871	3027	10365	No

Table 4.1: Summary of datasets. C_t and C_s are the numbers of top-level and subcategories, resp. N_{data}, N_{test}, N_{ans} are the sizes of data set, test set, and answer set, resp. Multilabel means each question can belong to multiple categories.

	TREC	INSUR.	DMV	YAHOO dataset		
				sub	top	unseen
CNN†	93.6	51.2	60	20.8	53.9	47
+sparsity‡	93.2	51.4	62	20.2	54.2	46
$\mathbf{W}_R$	93.8	53.5	62	21.8	54.5	48
$\mathbf{W}_Q$	**94.2**	53.8	64	22.1	54.1	48
$\mathbf{W}_A$	-	**55.4**	**66**	**22.2**	**55.8**	**53**

Table 4.2: Experimental results. Baselines: †sequential CNNs ($\alpha = \beta = 0$ in Eq. 4.5), ‡CNNs with global sparsity ($\beta = 0$). $\mathbf{W}_R$: randomly initialized projection matrix. $\mathbf{W}_Q$: question-initialized projection matrix. $\mathbf{W}_A$: answer set-initialized projection matrix. There are three different classification settings for YAHOO: subcategory, top-level category, and top-level accuracies on unseen sub-labels.

is not.

All experimental results are summarized in Table 4.2. The improvements are substantial for INSURANCE and DMV, but not as significant for YAHOO and TREC. One reason for this is the questions in YAHOO/TREC are shorter, which makes the group information harder to encode. Another reason is that each question in YAHOO/TREC has a single label, and thus can not fully benefit from group sparse properties.

Besides the conventional classification tasks, we also test our proposed model on an unseen-label case. In these experiments, there are a few sub-category labels that are not included in the training data. However, we still hope that our model could still return the correct parent category for these unseen subcategories at test time. In the testing set of YAHOO dataset, we randomly add 100 questions whose subcategory labels are unseen in training set. The classification results of YAHOO-unseen in Table 4.2 are obtained by mapping the predicted subcategories back to top-level categories. The improvements are substantial due to the group information encoding.

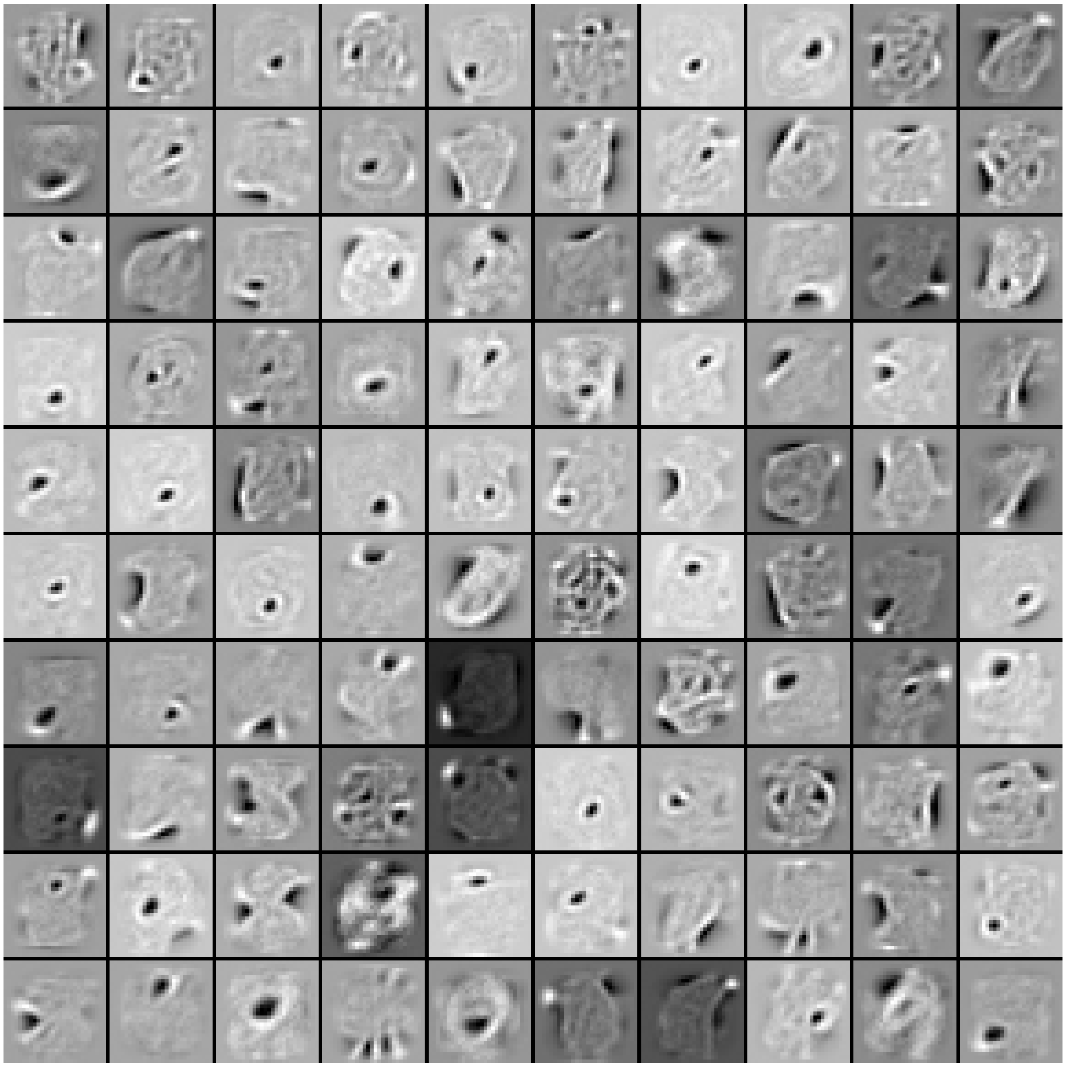
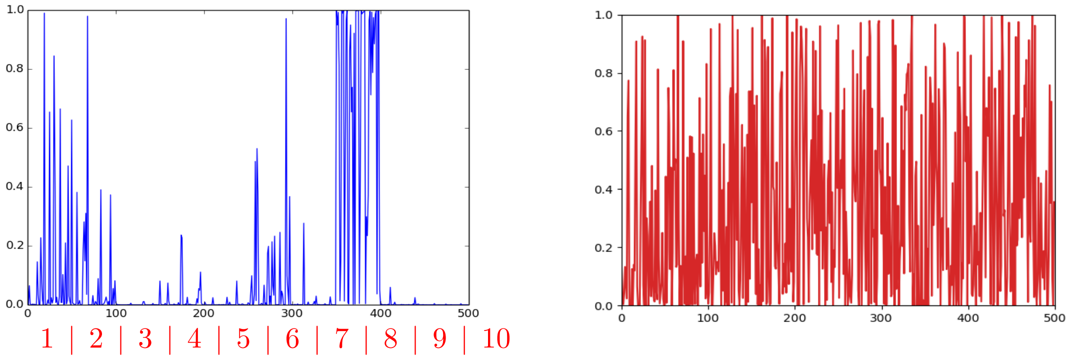


Figure 4.4: The projection matrix from basic autoencoders.The differences are easier to see in pdf.



(a) (b)

Figure 4.5: (a): the hidden activations **h** for the input image in Fig. 4.2. The red numbers corresponds to the index in Fig. 4.3. (b): the hidden activations **h** for the same input image from basic autoencoders.

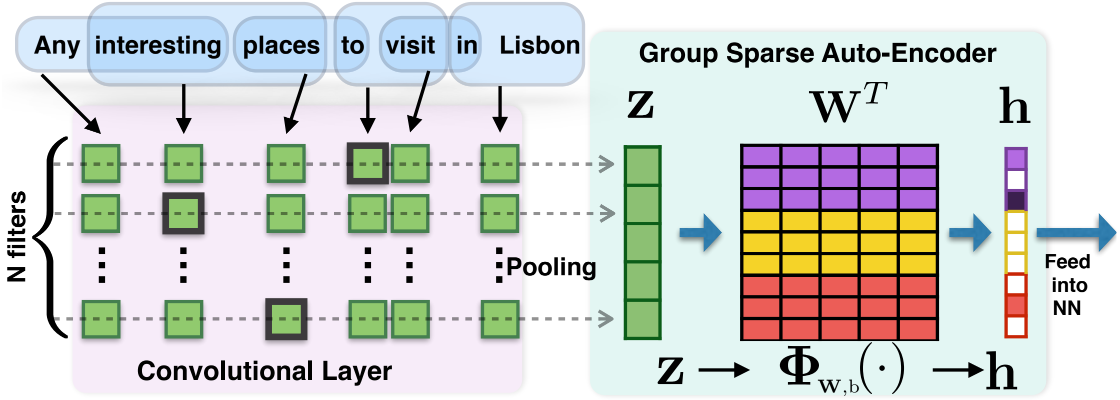

Any
interesting
places
to
visit
in
Lisbon
N filters
Pooling
Convolutional Layer
Group Sparse Auto-Encoder
z
$\mathbf{W}^T$
h
Feed into NN
$\mathbf{z} \longrightarrow \Phi_{\mathbf{W},\mathrm{b}}(\cdot) \longrightarrow \mathbf{h}$

Figure 4.6: Group Sparse CNN. We add an extra dictionary learning layer between sentence representation **z** and the final classification layer. **W** is the projection matrix (functions as a dictionary) that converts **z** to the group sparse representation **h** (Eq. 4.5). Different colors in the projection matrix represent different groups. We show $\mathbf{W}^\top$ instead of **W** for presentation purposes. Darker colors in **h** mean larger values and white means zero.

Chapter 5: Generating Structured Output from Non-Structured Input

Multimodal is a special task which takes two types of inputs, image and text, and predicts the source text in a different language. In this task, all the sentence pairs are image captions in different languages. The key difference between this task and conventional machine translation is that we have corresponding images as additional information for each sentence pair. In this paper, we introduce a simple but effective system which takes an image shared between different languages, feeding it into the both encoding and decoding side.

The work presented in this chapter has been published in the Proceedings of the second conference on Machine Translation [42].

5.1 Problems in Multimodal

Natural language generation (NLG) is one of the most important tasks in natural language processing (NLP). It can be applied to a lot of interesting applications such like machine translation, image captioning, question answering. In recent years, Recurrent Neural Networks (RNNs) based approaches have shown promising performance in generating more fluent and meaningful sentences compared with conventional models such as rule-based model [48], corpus-based n-gram models [71] and trainable generators [65].

More recently, attention-based encoder-decoder models [1] have been proposed to provide the decoder more accurate alignments to generate more relevant words. The remarkable ability of attention mechanisms quickly update the state-of-the-art performance on variety of NLG tasks, such as machine translation [39], image captioning [73, 77], and text summarization [56, 49].

However, for multimodal translation [14, 83], where we translate a caption from one language into another given a corresponding image, we need to design a new model since the decoder needs to consider both language and images at the same time.

This paper describes our participation in the WMT 2017 multimodal task 1. Our model feeds the image information to both the encoder and decoder, to ground their hidden representation within the same context of image during training. In this way, during testing time, the decoder would generate more relevant words given the context of both source sentence and image.

5.2 Model Description

For the neural-based machine translation model, the encoder needs to map sequence of word embeddings from the source side into another representation of the entire sequence using recurrent networks. Then, in the second stage, decoder generates one word at a time with considering global (sentence representation) and local information (weighted context) from source side. For simplicity, our proposed model is based on the attention-based encoder-decoder framework in [39], refereed as “Global attention”.

On the other hand, for the early work of neural-basic caption generation models [69], the convolutional neural networks (CNN) generate the image features which feed into the decoder directly for generating the description.

The first stage of the above two tasks both map the temporal and spatial information into a fixed dimensional vector which makes it feasible to utilize both information at the same time.

Fig. 5.1 shows the basic idea of our proposed model (OSU1). The red character **I** represents the image feature that is generated from CNN. In our case, we directly use the image features that are provided by WMT, and these features are generated by residual networks [18].

The encoder (blue boxes) in Fig. 5.1 takes the image feature as initialization for generating each hidden representation. This process is very similar to neural-basic caption generation [69] which grounds each word’s hidden representation to the context given by the image. On the decoder side (green boxes in Fig. 5.1), we not only let each decoded word align to source words by global attention but also feed the image feature as initialization to the decoder.

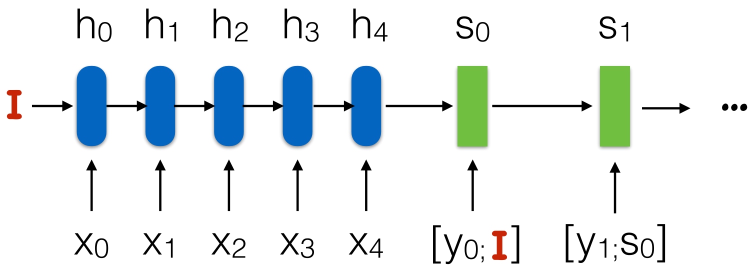


Figure 5.1: The image information is feed to both encoder and decoder for initialization. I (in red) represents the image feature that are generated by CNN.

5.3 Experiments

5.3.1 Datasets

In our experiments, we use two datasets Flickr30K [13] and MSCOCO [38] which are provided by the WMT organization. For both datasets, there are triples that contains English as source sentence, its German and French human translations and corresponding image. The system is only trained on Flickr30K datasets but are also tested on MSCOCO besides Flickr30K. MSCOCO datasets are considered out-of-domain (OOD) testing while Flickr30K dataset are considered in-domain testing. The datasets' statics is shown in Table 5.1

Datasets	Train	Dev	Test	OOD ?
Flickr30K	29,000	1,014	1,000	No
MSCOCO	-	-	461	Yes

Table 5.1: Summary of datasets statistics.

5.3.2 Training details

For preprocessing, we convert all of the sentences to lower case, normalize the punctuation, and do the tokenization. For simplicity, our vocabulary keeps all the words that show in training set. For image representation, we use ResNet [18] generated image features which are provided by the WMT organization. In our experiments, we only use average pooled features.

Our implementation is adapted from on Pytorch-based OpenNMT [28]. We use two layered bi-LSTM [67] on the source side as encoder. Our batch size is 64, with SGD optimization and a learning rate at 1. For English to German, the dropout rate is 0.6, and for English to French, the dropout rate is 0.4. These two parameters are selected by observing the performance on development set. Our word embeddings are randomly initialized with 500 dimensions. The source side vocabulary is 10,214 and the target side vocabulary is 18,726 for German and 11,222 for French.

5.3.3 Beam search with length reward

During test time, beam search is widely used to improve the output text quality by giving the decoder more options to generate the next possible word. However, different from traditional beam search in phrase-based MT where all hypotheses know the number of steps to finish the generation, while in neural-based generation, there is no information about what is the most ideal number of steps to finish the decoding. The above issue also leads to another problem that the beam search in neural-based MT prefers shorter sequences due to probability-based scores for evaluating different candidates. In this paper, we use Optimal Beam Search [21, 76] (OBS) during decoding time. OBS uses bounded length reward mechanism which allows a modified version of our beam search algorithm to remain optimal.

Figure 5.2 and Figure 5.3 show the BLEU score and length ratio with different rewards for different beam size. We choose beam size equals to 5 and reward equals to 0.1 during decoding.

5.3.4 Results

WMT organization provides three different evaluating metrics: BLEU [52], METEOR [34] and TER [62].

Table 5.2 to Table 5.5 summarize the performance with their corresponding rank among all other systems. We only show a few top performing systems in the tables to make a comparison. OSU1 is our proposed model and OSU2 is our baseline system without any image information. For MSCOCO dataset, the translation from English to German (Table 5.3), which is the hardest tasks compared with others since it is from

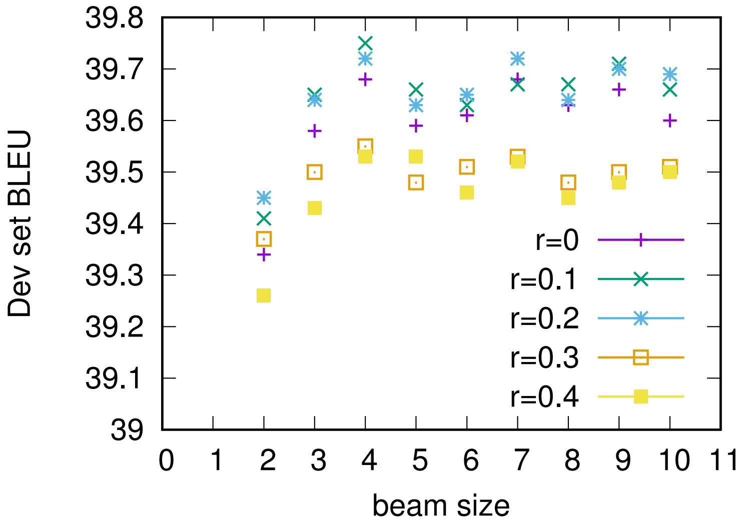


Figure 5.2: BLEU vs. beam size

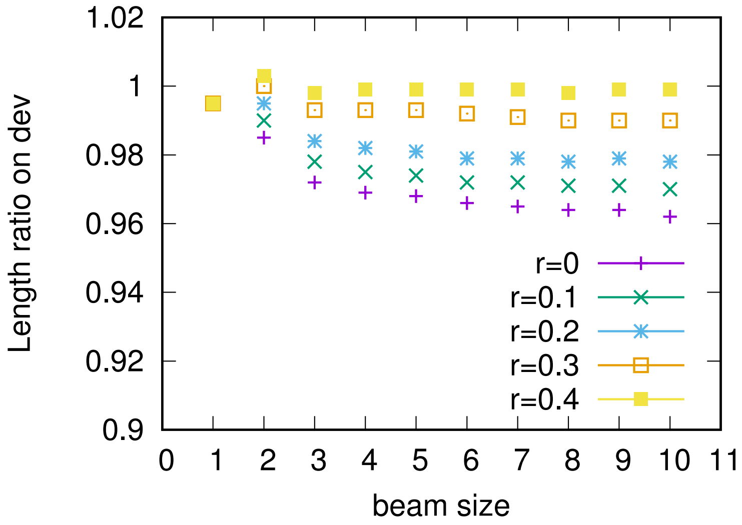


Figure 5.3: length ratio vs. beam size

English to German on OOD dataset, we achieve best TER score across all other systems.

System	Rank	TER	METEOR	BLEU
UvA-TiCC	1	**47.5**	53.5	**33.3**
NICT	2	48.1	**53.9**	31.9
LIUMCVC	3 & 4	48.2	53.8	33.2
CUNI	5	50.7	51	31.1
OSU2†	6	50.7	50.6	31
OSU1†	8	51.6	48.9	29.7

Table 5.2: Experiments on Flickr30K dataset for translation from English to German. 16 systems in total. † represents our system.

System	Rank	TER	METEOR	BLEU
OSU1†	1	**52.3**	46.5	27.4
UvA-TiCC	2	52.4	48.1	28
LIUMCVC	3	52.5	**48.9**	**28.7**
OSU2†	8	55.9	45.7	26.1

Table 5.3: Experiments on MSCOCO dataset for translation from English to German. 15 systems in total. † represents our system.

System	Rank	TER	METEOR	BLEU
LIUMCVC	1	**28.4**	**72.1**	**55.9**
NICT	2	28.4	72	55.3
DCU	3	30	70.1	54.1
OSU2†	5	32.7	68.3	51.9
OSU1†	6	33.6	67.2	51

Table 5.4: Experiments on Flickr30K dataset for translation from English to French. 11 systems in total. † represents our system.

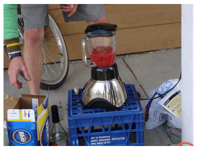
As describe in section 3.3.1, OSU1 is the model with image information for both encoder and decoder, and OSU2 is only the neural machine translation baseline without any image information. From the above results table we found that image information would hurt the performance in some cases. In order to have more detailed analysis, we show some test examples for the translation from English to German on MSCOCO dataset.

Fig 5.4 shows two examples that NMT baseline model performances better than

System	Rank	TER	METEOR	BLEU
LIUMCVC	1	**34.2**	**65.9**	**45.9**
NICT	2	34.7	65.6	45.1
DCU	3	35.2	64.1	44.5
OSU2†	4	36.7	63.8	44.1
OSU1†	6	37.8	61.6	41.2

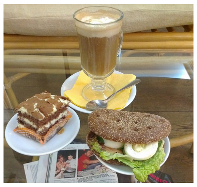
Table 5.5: Experiments on MSCOCO dataset for translation from English to French. 11 systems in total.

input	a finger pointing at a hotdog with cheese , sauerkraut and ketchup .
OSU1	ein finger zeigt auf einen hot dog mit einem messer , wischmobs und napa .
OSU2	ein finger zeigt auf einen hotdog mit hammer und italien .
Reference	ein finger zeigt auf einen hotdog mit kse , sauerkraut und ketchup .

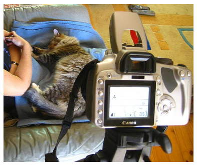


input	a man reaching down for something in a box
OSU1	ein mann greift nach unten , um etwas zu irgendeinem .
OSU2	ein mann greift nach etwas in einer kiste .
Reference	ein mann bckt sich nach etwas in einer schachtel .

Figure 5.4: Two testing examples that image information confuses the NMT model.



input	there are two foods and one drink set on the clear table .
OSU1	da sind zwei speisen und ein getrnk am klaren tisch .
OSU2	zwei erwachsene und ein erwachsener befinden sich auf dem rechteckigen tisch .
Reference	auf dem transparenten tisch stehen zwei speisen und ein getrnk .



input	a camera set up in front of a sleeping cat .
OSU1	eine kameracrew vor einer schlafenden katze .
OSU2	eine kamera vor einer blonden katze .
Reference	eine kamera , die vor einer schlafenden katze aufgebaut ist

Figure 5.5: Two testing examples that image information helps the NMT model.

OSU1 model. In the first example, OSU1 generates several unseen objects from given image, such like knife. The image feature might not represent the image accurately. For the second example, OSU1 model ignores the object “box” in the image.

Fig 5.5 shows two examples that image feature helps the OSU1 to generate better results. In the first example, image feature successfully detects the object “drink” while the baseline completely neglects this. In the second example, the image feature even help the model figure out the action of the cat is “sleeping”.

Chapter 6: Structured Prediction Problems from Structured Input

In this chapter, we talk about the most widely used sequence generation framework, RNN-based sequence-to-sequence model. Different from all the models that are from previous chapter, in this type of framework, we only expect structure inputs and generate another form of structure outputs on the target side. This kind of framework is very popular in the application of parsing, machine translation and summarization. In this chapter, we only discuss the application of machine translation.

In recent years, neural text generation with RNN-based sequence-to-sequence model has attracted many attentions and we witnessed a rapid growth and progress in many different variants of RNN-based models in order to suit different usages in different scenarios. This RNN-based model quickly setting up the state-of-the-art paradigms in machine translation [25, 68, 2], summarization [57, 53] and image captioning [70, 74].

Thought the great successfulness in RNN-based framework, the current NMT still has its own shortcoming in many different ways. In this chapter, we will first discuss some penitential problems, for example, label bias, training and testing metric mismatch and so on. Then we would propose our own solution.

6.1 Problems in Structured Prediction for MT

6.1.1 Difference Between Training and Testing Phase

The most popular approach for training NMT system is as a conditional language model, with training maximizing the likelihood of each successive target word conditioned on the input sequence and the gold history of target words. In this way, the training signal is guided word-level loss, e.g., cross-entropy loss over target vocabulary. This approach has demonstrated its effectiveness in both efficiency and accuracy for training NMT models.

Fig. 6.1 show one example for typical encoder-decoder model with attention mechanism. In this model, the encoder will encoder all the source side words into a sequence of hidden state with recurrent fashion. On the other side, the decoder takes t^{th} word from gold sequence to generate the $(t+1)^{th}$ word representation. In this step, the decoder side's word will also be attended to source side to get the alignment. Based on the hidden state and attended context representation, the decoder will generate the most possible word based on word probabilities. Minimizing the difference between this predicted word distribution and the gold word probability is the training goal for this system. We refer this per-word based training style as local training based on the fact that gold reference will be feed into the decoder at each time step.

However, this kind of per-word or local training mechanism must be different from real testing time decoding due to the face that the gold sequence is unknown to the system. In this way, we have a training and testing time mis-match problem since the model never generate fully-formed word sequences.

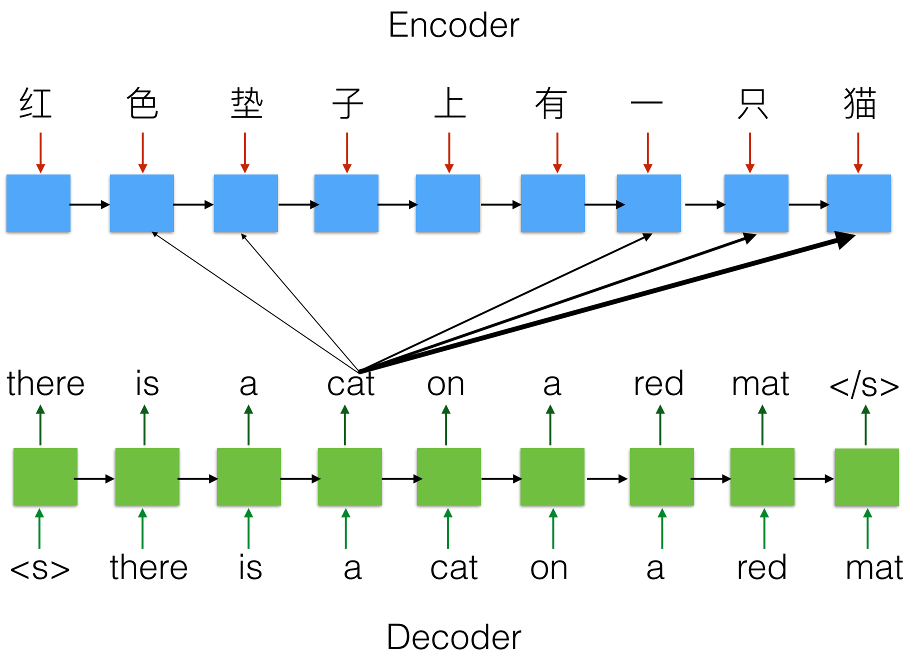


Figure 6.1: LSTM-based encoder-decoder model with attention mechanism.

In practice, beam search is used for exploring better candidates which are sorted by the model score. However, the mis-match between training and testing phase are still problematical due to the following issues.

The first problem is Exposure Bias. Since the model is never exposed to its own errors during training time, in this case, the inferred histories at test time do not resemble the gold training histories. Once the decode meet a unseen path from the history during testing time. There is no way that the gold reference could come back to the beam again. Multi-reference training [82] is one approach to introduce more possible search path during training which potentially release the exposure bias problem.

The second problem is Loss-Evaluation Mismatch. As we mentioned above, during training time, we do local training. In this way, we only could use word level loss. However, during test time, we evaluate our model's performance with BLEU score which is global evaluation. BLEU score is hard to incorporate into training phase due to the face that BLEU score is undecomposable and undifferentiable.

The third problem is Label Bias [32]. Since the word-level probability are normalized by softmax, guaranteeing that successors of incorrect histories receive the same mass as do the successors of the true history. As it is shown in Fig.6.2, State 1 almost always prefers to go to state 2 and state 2 almost prefers to go to state 2. Based the cumulative calculation with per-step probability, the path 1-1-1-1 has the probability of 0.09 and path 2-2-2-2 has the probability of 0.018. Since the local choice is always normalized by the overall number of choice, sometimes, the probability numbers do not represent the real preferred choice of model.

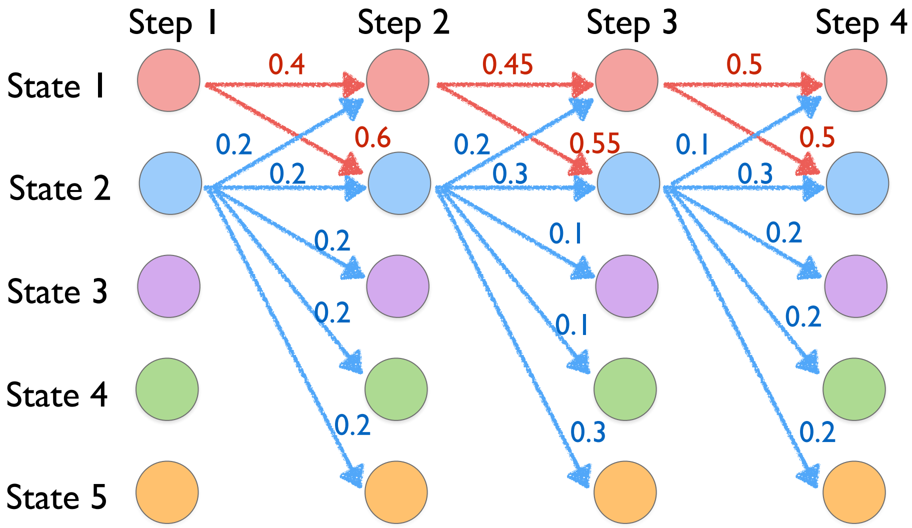


Figure 6.2: Local normalization introduces label bias problem.

6.1.2 Problems of Beam Search

The last problem is about beam search for beam search is used in practice for exploring better candidates. However, there is key difference between the beam search of phrase-based MT or shift-reduce parsing and the one in neural generation. In phrase-based MT or shift-reduce parsing, all the hypotheses finish at the same number of steps, while in NMT, different hypotheses do not have to finish at the same number of steps. In NMT, once a hypotheses completed generation by decoding a `</s>` symbol, all other active hypotheses which survive in the beam still can continue to grow, and some of them potentially could get better scores. Therefore, this problem leads to another type of question: when can you end the beam search?

There are many different kind of approaches for solving this problem, for example, the widely influential RNNsearch [2] propose to use “shrinking beam” method: alway reduce the beam size by 1 when you get a finished hypothesis. In OpenNMT [29], they terminate the beam search whenever the highest-ranking hypothesis in the current step is completed.

Another thread of beam search problems is about candidates ranking methods. As it is describe in [21, 76], since all the hypotheses are evaluated by probabilistic measurements, in this case, beam search always return the highest hypotheses which is often the shortest candidates from beam search 6.3. RNNsearch propose to use “length normalization” methods, which normalize the score by the length of the hypothesis since the beam search favors shorter sentences.

6.2 Training with Beam Search

In this chapter, in order to get rid of the above mentioned problems, we propose a non-probabilistic framework based on existing RNN-based model. In our model, instead of locally normalized on each step, we directly use raw score for evaluating the model preference. Inspired by the work named Learning as Search Optimization [11, 72], we directly defines the loss function during beam search in stead of word-level cross-entropy loss. Furthermore, our algorithm also measures the loss on sequence level, which matches the training and testing time decoding strategy.

We first introduce the background and set up the notation for later discussion. Then

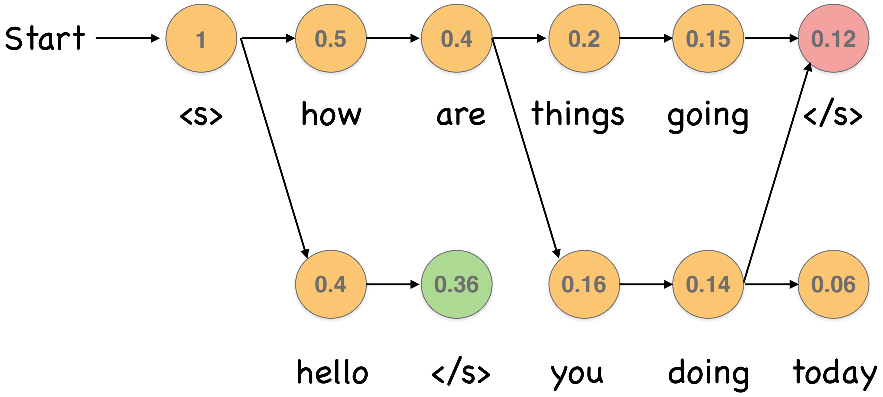


Figure 6.3: Beam search favors shorter sentences.

we introduce our approach for solving the previously mentioned issue.

6.2.1 Preliminaries

Our model is adapted based the most popular seq2seq framework. Assume we have a collection of source and target sequence pairs. Our goal is to learn the predicted based on given training instance. The task of MT is to first encode the source side sentence and compress the input information into a collection of hidden representations $\mathbf{x}$. In this chapter, we use $\mathbf{x}$ to represent the encoded source sentence representation.

After we obtain the collection of source-side representation, another part of seq2seq model, decoder, will start generating the predicted sequence word-by-word based on the $\mathbf{x}$ as input signal. The generated words from decoder are constraint with a given vocabulary $\mathcal{V}$. More specifically, during testing time decoding, words are generated one-by-one based on the condition on the input representation $\mathbf{x}$ and the history which is previously generated words. We use $\mathbf{w}_{1..T}$ to refer any word sequence with length of T, and $y_{1..T}$ to represent the gold sequence for target side for a given input $\mathbf{x}$.

The following equation is one simple representation of hidden state's representation on target side:

$$\mathbf{h}_t \leftarrow \mathrm{RNN}(m_t, h_{t-1}; \theta) \tag{6.1}$$

where the θ represents the model parameters, which are kept the same over time in order to reduce the model complexity. Based on the above definition, the RNN-based framework recursively generate the hidden state based on the previously generated state and new coming words m_t. In this case, m_t represents the word embeddings.

For the decoder side, the model functions more like a conditional language model, which is conditional on the given context $\mathbf{x}$ and previously generated words:

$$p(w_t|w_{1:t-1}, x) = g(w_t, h_{t-1}, x) \tag{6.2}$$

where g is a linear or nonlinear mapping from each decoder's hidden state to the word space followed by a softmax layer. Softmax layer attempts to normalize the probability distribution given the raw-score value which is generated from projection layer. During training time, the model is to to minimize the cross-entropy loss at each time-step while conditioning on the gold history when decoder follows the gold path. This is what we call local training. And the loss function is defined as follows:

$$-\sum_{1}^{T} logp(y_t|y_{t-1}, \mathbf{x}) \tag{6.3}$$

Once our model is trained based on above definition, during testing time decoding, we could generate the predicted sentence with maximizing the following conditional distribution:

$$\hat{\mathbf{y}}_{1..T} = \mathbf{argmax} \sum_{1}^{T} logp(y_t|y_{t-1}, \mathbf{x}) \tag{6.4}$$

The above searching criterion is so-called greedy search. However, in real application, in order to improve the searching quality, we usually use a small beam to explore more possibilities.

6.2.2 Two Types of Violations

We adopt the seq2seq model by making small changes over different steps. The first change is, instead of predicting the probability of the next word, we instead learn to produce (non-probabilistic) scores for ranking sequences. We define $f(w_t, h_{t-1}, x)$ as the function of model score over a sequence consisting of history $w_{1:t-1}$ followed by a single word w_t. f is a parameterized function examining the current hidden-state of the relevant RNN at time $t-1$ as well as the input representation x. In our model, f is the same with the previous RNN-based generation model but without the final softmax transformation layer, which converts unnormalized raw scores into probabilities, in this way, allowing the model to avoid issues associated with the label bias problem. Please note that we do not use probability model in our framework, and this is the first key difference between previous work.

In our proposed method, we have two different kinds of violations, the internal violations and last step violations. Fig. 6.4 shows one running example of these violations.

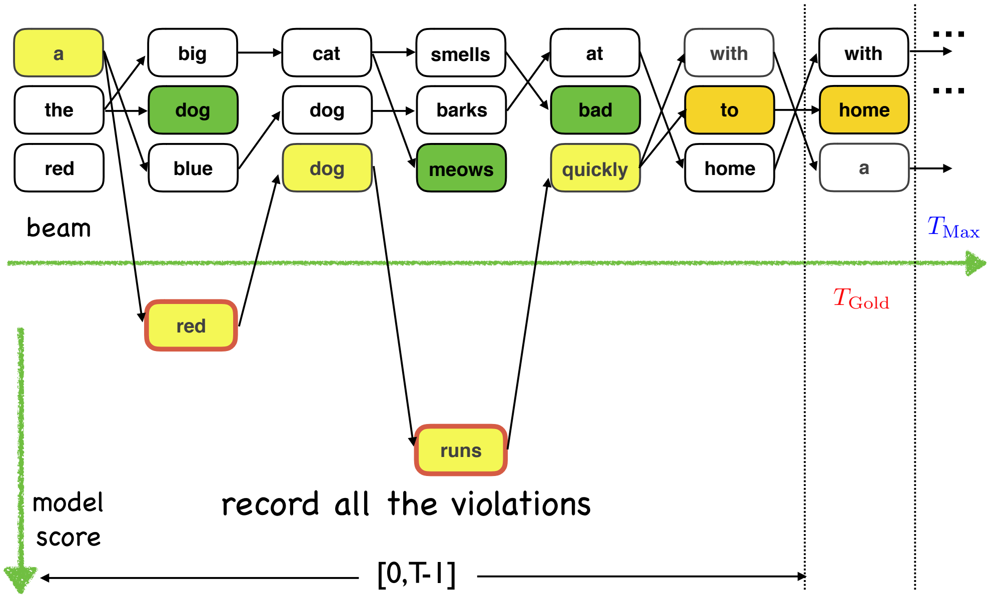


Figure 6.4: Our proposed model, training with beam search.

Similar with BSO [72], for internal violation, the violation is always defined on the same time step. For example, at a arbitrary time step t, when beam size is K, we collect all the sentences with length t to form the candidates in beam. We will compare the last candidate in beam, d_t^K, with the gold reference d_t^*. If the model score of candidate d_t^K is greater than the gold reference at t, d_t^*, by a margin 1, we define this is a violation. The goal of internal violation is try to keep the gold reference inside of the beam. In this way, during testing time decoding, the gold reference will be likely to be reach in practice. The violation is defined as follows:

$$\text{score}_w(x, d_t^*) < \text{score}_w(x, d_t^K) + 1 \tag{6.5}$$

When we have a internal violation, we will restart the beam like LaSO. This force action will guarantee that the gold reference is always alive in the beam.

During checking the internal violation, we also keep track of finished sentences. We put all the finished candidate sentences into a ordered set. This ordered set is sorted by the model score with a testing metric loss, e.g. BLEU score, which is defined in Eq. 6.6.

For last step violation, when the gold reference reaches the last symbol, we compare the loss-augmented models score of the gold sequence against to the finished candidates. The violation is defined as follows:

$$\begin{aligned}
&d_T^* < d_{T'} \\
&d_T^* = \text{score}_w(x, d_t^K) \\
&d_{T'} = \arg\max \text{score}_w(x, d_f) - \Delta(y(d_f), y), \\
&\Delta(y(d_f), y) = 1 - \text{BLEU}(y(d_f), y)
\end{aligned} \tag{6.6}$$

The most important difference between BSO and our framework, the above equation basically compare the gold reference against to **all the finished candidates**. In BSO, they only compare the candidates who are in the same time step with gold. This kind of comparison neglects the nature between finished candidates and unfinished candidates. As it is shown in Fig. 6.5, all the red dots represent the finished candidates during searching. Among these candidates, some of the sentences have a very low BLEU score, and some of them have high models score. The motivation of above equation is trying to find the candidate with the highest score sum over the model and ΔBELU score. In this way, we always can find the worst candidate, who has a better model score but it's BLEU is very low.

Algorithm 1 shows the detailed algorithms for our proposal. Different from BSO, instead of stop the beam search at time step T, we encourage the beam search proceed even further, which is $1.5 \times T$, where T is the source side length. During beam search

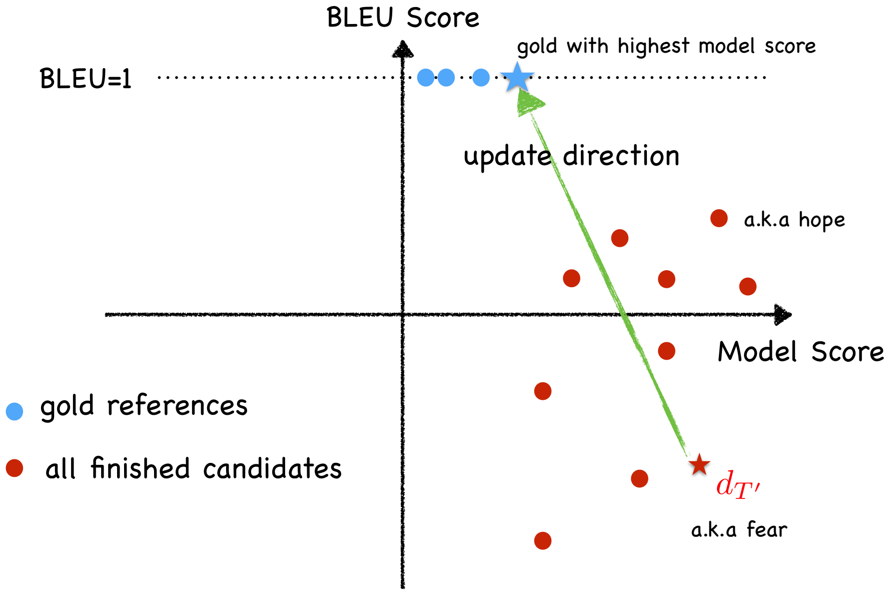


Figure 6.5: Geometric understanding of our proposed model.

training, we have two different violation pool: internal violation and last step violation. The internal violation is defined based on 6.5 which is similar with BSO. Our proposed framework is mainly about the difference of last step violation. In BSO, the last step violation only compares with the candidates which are in the same time step. However, in our last step violation, we compare with all the finished candidates during beam search. This major difference will encourage the model to learn the right length ratio for the candidates. For example, in probabilistic model, the predicted sentence are tend to shorter, then our model will compare the gold with the shorter sentence, in this way, we encourage the model to generate longer sentence.

Algorithm 1 Finished Candidates Beam-Search Optimization

```
procedure FINISHED BSO($\mathbf{x}, K, \textsc{succ}$)
    Initialization: gold sequence $\hat{y}_{1..T}$; previously generated sequence $y_{1..t}$ ;
    and corresponding hidden state $\hat{h}_{1..t}$; internal violations, interv $\leftarrow \{0\}$;
    last step violations, lastv $\leftarrow \{0\}$; r $\leftarrow$ 0.
    for $t = 1, ..., 1.5 \times T$ do
        if score($y_t, h_{t-1}$) $<$ score($\hat{y}_t^K, h_{t-1}^K$) $+ 1$ then
            $\hat{h}_{r:t-1} \leftarrow \hat{h}_{r:t-1}^K$
            $\hat{y}_{r+1:t} \leftarrow \hat{y}_{r+1:t}^K$
            Add $t$ to interv
            $r \leftarrow t$
            $S_{t+1} \leftarrow$ TopK(succ($y_{1:t}$));
        else
            $S_{t+1} \leftarrow$ TopK($\bigcup_{k=1}^{K}$ succ($\hat{y}_{1:t}^k$));
        for $k = 1, ..., K$ do
            if $y_t^k$ is </s> then
                Add $y_t^k$ to lastv
```

6.3 Experiments

6.3.1 Datasets

For our machine translation task, we evaluate our model on a small dataset, which allow our model to train with larger beam size more efficiently. We use the TED talk dataset from the work [53]. This dataset is a small portion from IWSLT 2014 machine translation evaluation campaign. There are 153K training sentence pairs, 7K development dataset and 7K test sentences. In order to make fair and accurate comparison, we use the same split and preprocessing as [53].

To keep different model with roughly within the same complexity, we also use single-layer LSTM decoder with 256 units. We set the dropout rate as 0.2 between each LSTM layer. We choose to use global attention as our attention mechanism.

Our code base is developed based on OpenNMT code base with PyTorch while the original BSO is implemented with Lua-based packages. In order to keep both work on the same level of starting point, we first re-implement the Lua-base BSO with PyTorch.

6.3.2 Results

	K=1	K=5	K=10
Seq2Seq (Lua)	22.53	24.03	23.87
BSO (Lua) [72]	23.83	26.36	25.48
DAD [3]	20.12	22.25	22.40
MIXER [53]	20.73	21.81	21.83
Seq2Seq (Ours Implementation)	25.7	27.21	26.91
BSO (Ours Implementation)	26.32	27.97	27.51
Our proposed model	27.28	**28.44**	28.03

Table 6.1: Machine translation performance comparison with different SOTA models on test set.

Table 6.1 showcases the performance comparison between our proposed model and other state-of-the-art models. We achieve the best performance on test set across different models. Both BOS and our model are first got pre-trained on conventional Seq2seq model. BSO is originally developed on Lua. Their pre-trained achieves 24.03 in BLEU score and BSO further improve the performance to 26.36.

In order to make a fair comparison, we first use our pre-trained model which has the performance 27.21. Since our pre-trained model even achieves much better performance compare with BSO (around 1 point in BLEU), we re-implement the BSO framework on our code base. Our version of BSO achieve 27.97 in BLEU. Based on our BSO implementation, we make further adaptation to get our model, and our model sets up a higher SOTA performance on this dataset.

	K=2	K=4	K=6	K=8
BSO (Ours Implementation)	29.3 (0.95)	31.25 (1.007)	30.71 (1.03)	30.42 (1.05)
Our proposed model	29.4 (0.94)	31.67 (1.005)	31.13 (1.02)	30.61 (1.03)

Table 6.2: Compare our results with baseline on development set. We compare both BLEU and length ratio.

Table 6.1 illustrates the performance between BSO and our model with different beam size. Since BSO and our model use raw score instead of probability-based score, the generated sentence tend to be longer instead of shorter. In probabilistic model, shorter sentences usually have larger model score since in every step, we only add a negative per-step score to that sequence. However, in raw score-based model, usually the raw score are most positive, therefore, the longer sentences always have better score.

In our proposed model, we propose to define the last step violation across all finished candidates. In this way, all the candidates which are not in the right length will be penalized by the model. From the Table 6.1, we also conclude that we learn better length ratio compare with BSO.

Chapter 7: Summary

In this dissertation, we reviewed several popular neural-based framework for several well-known NLP tasks. We grouped these NLP tasks into different categories by the natural of their input and output requirements.

We started from the most simple task, sentence classification Chapter 2, which is been recognized as popular unstructured problem. But in our analysis, we found that including tree-based structure could improve the framework's ability to handle long-distance structure information. Figure 2.3 illustrates the intuition of the reasons why our model works better than the baseline models. Furthermore, Figure 2.4 and Figure 2.5 also point out the limitations of our proposed model and general problems for neural-based model. These phenomenons also appoint us the new directions for improving the existing models.

Following the above discussion, in Figure 2.5 we found that words' syntactic variety and polysemy sometime confuses the framework easily. Therefore, in Chapter 3, we propose a joint model for eliminating this kind of ambiguities. As shown in Figure 3.1, given a sentence-level category, the word ambiguities will be highly eliminated. In the other way around, given a correct understand of words, the model will be also very certain about the sentence-level category.

Starting from Chapter 4, we introduce another types of frameworks, which expects the output also to be structured. We start from introducing the most simple case, the input is still non-structured, but we need to predict the hierarchical labels on the target side. The above mentioned sentence classification is one type of sentence modeling techniques whose labels are mutually exclusive to each other. However, this is not necessary in other kind of sentence classification task, question answering. In the task of question answering, question classes often have a hierarchical structure and those classes are often related to each other. Our proposed model successfully handles these information with a dictionary which learn by CNNs.

In Chapter 5, we introduce another framework which takes the unstructured image input feature and structured source text in one language to a different language. We demonstrate that incorporating the image information as one extra input sometimes improves the target side predictions.

In Chapter 6, we briefly introduce several existing common problems for neural generation problems. Based on BSO, we did some simple adaptation based on existing models and propose our global training framework. Our proposed model could learn better length ratio which improves the overall performance.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, 2014.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1171–1179, Cambridge, MA, USA, 2015. MIT Press.

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J.C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

[5] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 1992.

[6] Emmanuel J. Candè and Michael B. Wakin. An Introduction To Compressive Sampling. In *Signal Processing Magazine, IEEE*, volume 25, 2008.

[7] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. 2016.

[8] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[9] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, 2002.

[10] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. volume 12, pages 2493–2537, 2011.

[11] Hal Daumé, III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of ICML*, 2005.

[12] Kushal Dave, Steve Lawrence, and David M Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM, 2003.

[13] D. Elliott, S. Frank, K. Sima'an, and L. Specia. Multi30k: Multilingual english-german image descriptions. *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74, 2016.

[14] Desmond Elliott, Stella Frank, and Eva Hasler. Multi-language image description with neural sequence models. *CoRR*, 2015.

[15] Simon Fleming, Dan Chalmers, and Ian Wakeman. A deniable and efficient question and answer service over ad hoc social networks. volume 15, pages 296–331. Springer Netherlands, 2012.

[16] Michael Gamon. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. In *Proceedings of the 20th international conference on Computational Linguistics*, page 841. Association for Computational Linguistics, 2004.

[17] Daniel Guo, Gokhan Tur, Wen-Tau Yih, and Geoffrey Zweig. Joint semantic utterance classification and slot filling with recursive neural networks. In *IEEE Workshop on Spoken Language Technology (SLT)*, 2014.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition CVPR*, 2016.

[19] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *Journal of Machine Learning Research*, 15, 2014.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Cambridge, MA, USA, 1997.

[21] Liang Huang, Kai Zhao, and Mingbo Ma. Optimal beam search for neural text generation (modulo beam size). In *EMNLP 2017*, 2017.

[22] Ozan Irsoy and Claire Cardie. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems*, pages 2096–2104, 2014.

[23] Minwoo Jeong and Geunbae G. Lee. Triangular-Chain Conditional Random Fields. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16, 2008.

[24] John Judge, Aoife Cahill, and Josef van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of COLING*, 2006.

[25] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, volume 3, page 413, 2013.

[26] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[27] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[28] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. *ArXiv e-prints*, 2017.

[29] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*, 2017.

[30] Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics, 2010.

[31] Taku Kudo and Yuji Matsumoto. Proceedings of the 2004 conference on empirical methods in natural language processing. 2004.

[32] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, 2001.

[33] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[34] Alon Lavie and Michael J. Denkowski. The meteor metric for automatic evaluation of machine translation. *Machine Translation*, 2009.

[35] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. 2014.

[36] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Mller, E. Sckinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In *INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS*, pages 53–60, 1995.

[37] Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[38] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. 2014.

[39] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, 2015.

[40] Mingbo Ma, Liang Huang, Bing Xiang, and Bowen Zhou. Dependency-based convolutional neural networks for sentence embedding. In *Proceedings of ACL 2015*, 2015.

[41] Mingbo Ma, Liang Huang, Bing Xiang, and Bowen Zhou. Group sparse cnns for question classification with answer sets. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 335–340, 2017.

[42] Mingbo Ma, Dapeng Li, Kai Zhao, and Liang Huang. Osu multimodal machine translation system report. In *Proceedings of the Second Conference on Machine Translation*, pages 465–469. Association for Computational Linguistics, 2017.

[43] Mingbo Ma, Kai Zhao, Liang Huang, Bing Xiang, and Bowen Zhou. Jointly trained sequential labeling and classification by sparse attention neural networks. In *Interspeech*, 2017.

[44] Alireza Makhzani and Brendan Frey. K-sparse autoencoders. In *International Conference on Learning Representations*. 2014.

[45] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[46] Shotaro Matsumoto, Hiroya Takamura, and Manabu Okumura. Sentiment classification using word sub-sequences and dependency sub-trees. In *Proceedings of the*

9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, 2005.

[47] Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, 2006.

[48] Danilo Mirkovic, Lawrence Cavedon, Matthew Purver, Florin Ratiu, Tobias Scheideck, Fuliang Weng, Qi Zhang, and Kui Xu. Dialogue management using scripts and combined confidence scores. *US Patent*, pages 7,904,297, 2011.

[49] Ramesh Nallapati, Bowen Zhou, and Mingbo Ma. Classify or select: Neural architectures for extractive document summarization. *CoRR*, 2016.

[50] Andrew Ng. Sparse autoencoder. 2011.

[51] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL*, pages 115–124, 2005.

[52] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002.

[53] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *ICLR*, 2016.

[54] William J. Rapaport. A history of the sentence "buffalo buffalo buffalo buffalo buffalo.". 2012.

[55] R. Rubinstein, A. M. Bruckstein, and M. Elad. Dictionaries for sparse representation modeling. 2010.

[56] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. 2015.

[57] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[58] Chirag Shah and Jefferey Pomerantz. Evaluating and predicting answer quality in community qa. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 411–418, New York, NY, USA, 2010. ACM.

[59] Yelong Shen, Xiaodong he, Jianfeng Gao, Li Deng, and Gregoire Mesnil. Learning semantic representations using convolutional neural networks for web search. WWW 2014, April 2014.

[60] J. Silva, L. Coheur, A. C. Mendes, and Andreas Wichert. From symbolic to subsymbolic information in question classification. *Artificial Intelligence Review*, 35, 2011.

[61] Noah Simon, Jerome Friedman, Trevor Hastie, and Rob Tibshirani. A sparse-group lasso. 2013.

[62] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. *In Proceedings of Association for Machine Translation in the Americas*, 2006.

[63] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[64] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Stroudsburg, PA, October 2013. Association for Computational Linguistics.

[65] Amanda Stent, Rashmi Prasad, and Marilyn Walker. Trainable sentence planning for complex information presentation in spoken dialog systems. *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, 2004.

[66] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[67] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2014.

[68] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[69] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[70] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[71] Tsung-Hsien Wen, Milica Gasic, Dongho Kim, Nikola Mrksic, Pei-hao Su, David Vandyke, and Steve J. Young. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. *CoRR*, 2015.

[72] Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. volume abs/1606.02960, 2016.

[73] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015.

[74] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 14, pages 77–81, 2015.

[75] Puyang Xu and Ruhi Sarikaya. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *ASRU*, pages 78–83. IEEE, 2013.

[76] Yilin Yang, Liang Huang, and Mingbo Ma. Breaking the beam search curse: A study of (re-)scoring methods and stopping criteria for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[77] Zhilin Yang, Ye Yuan, Yuexin Wu, William W. Cohen, and Ruslan Salakhutdinov. Review networks for caption generation. *Advances in Neural Information Processing Systems*, 2016.

[78] Wen-tau Yih, Xiaodong He, and Christopher Meek. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 643–648. Association for Computational Linguistics, 2014.

[79] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. volume 68, pages 49–67, 2006.

[80] Mattgew Zeiler. Adadelta: An adaptive learning rate method. *Unpublished manuscript*: `http://arxiv.org/abs/1212.5701`, 2012.

[81] Kai Zhao, Liang Huang, and Mingbo Ma. Textual entailment with structured attentions and composition. volume abs/1701.01126, 2017.

[82] Renjie Zheng, Mingbo Ma, and Huang Liang. Multi-reference training with pseudo-references for neural translation and text generation. 2018.

[83] Renjie Zheng, Yilin Yang, Mingbo Ma, and Huang Liang. Ensemble sequence level training for multimodal mt: Osu-baidu wmt18 multimodal machine translation system report. 2018.

[84] Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. Long short-term memory over tree structures. 2015.