Ph.D. DISSERTATION

Sentence Pair Modeling using Deep Neural Network Sentence Encoders

딥 뉴럴 네트워크 기반의 문장 인코더를 이용한 문장 간 관계 모델링

BY

Jihun Choi

FEBRUARY 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Sentence Pair Modeling using Deep Neural Network Sentence Encoders

딥 뉴럴 네트워크 기반의 문장 인코더를 이용한 문장 간 관계 모델링

BY

Jihun Choi

FEBRUARY 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Abstract

Sentence matching is a task of predicting the degree of match between two sentences. Since high level of understanding natural language text is needed for a model to identify the relationship between two sentences, it is an important component for various natural language processing applications.

In this dissertation, we seek for the improvement of the sentence matching module from the following three ingredients: sentence encoder, matching function, and semi-supervised learning. To enhance a sentence encoder network which takes responsibility of extracting useful features from a sentence, we propose two new sentence encoder architectures: Gumbel Tree-LSTM and Cell-aware Stacked LSTM (CAS-LSTM). Gumbel Tree-LSTM is based on a recursive neural network (RvNN) architecture, however unlike typical RvNN architectures it does not need a structured input. Instead, it learns from data a parsing strategy that is optimized for a specific task. The latter, CAS-LSTM, extends the stacked long short-term memory (LSTM) architecture by introducing an additional forget gate for better handling of vertical information flow.

And then, as a new matching function, we present the element-wise bilinear sentence matching (ElBiS) function. It aims to automatically find an aggregation scheme that fuses two sentence representations into a single one suitable for a specific task. From the fact that a sentence encoder is shared across inputs, we hypothesize and empirically prove that considering only the element-wise bilinear interaction is sufficient for comparing two sentence vectors. By restricting the interaction, we can largely reduce the number of required parameters compared with full bilinear pooling methods without losing the advantage of automatically discovering useful aggregation schemes.

Finally, to facilitate semi-supervised training, i.e. to make use of both labeled and unlabeled data in training, we propose the cross-sentence latent variable model (CS-LVM). Its generative model assumes that a target sentence is generated from the latent representation of a source sentence and the variable indicating the relationship between the source and the target sentence. As it considers the two sentences in a pair together in a single model, the training objectives are defined more naturally than prior approaches based on the variational auto-encoder (VAE). We also define semantic constraints that force the generator to generate semantically more plausible sentences.

We believe that the improvements proposed in this dissertation would advance the effectiveness of various natural language processing applications containing modeling sentence pairs.

Keywords: sentence matching, sentence pair modeling, sentence encoder, matching function, semi-supervised learning, deep neural network, deep generative model
Student Number: 2014-22694

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Sentence Matching

One of the most prominent characteristics of natural language is that the same meaning could be expressed by various forms (Dagan et al., 2005). This means that in most cases a certain meaning can be related to multiple expressions and vice versa, making it hard for a computer to comprehend natural language. At the same time, despite its difficulty, a plethora of commercial applications or research problems require high level of natural language understanding; these include virtual assistant, market analysis, automatic translation, question answering system, etc.

Among various subfields in natural language understanding, research on sentence matching aims to predict the degree of match between two (or more) sentences. Identifying the relationship between two sentences, e.g. semantic similarity or entailment, is deeply related to understanding the meaning of natural language, thus it is an important ingredient for many natural language processing problems, and building a high-performance sentence matching model plays a key role in enhancing quality of systems for those problems.

For example, in natural language inference, a model has to predict whether a hypothesis sentence could be inferred from a given premise sentence by performing semantic inference (Dagan et al., 2005; Bowman et al., 2015). In paraphrase identification a model should compare meaning of two sentences and detect whether one sentence is a paraphrase of the other (Dolan and Brockett, 2005; Xu et al., 2014), i.e. a model needs to capture the variability of language. Answer sentence selection requires measuring the conformance of each candidate answer sentence to a question (Tan et al., 2016; Tymoshenko and Moschitti, 2018), and similarly text retrieval accompanies finding the most relevant text sequence given a query (Mitra and Craswell, 2017; Mitra et al., 2017). Machine comprehension (Rajpurkar et al., 2016) also contains the process of matching a paragraph and a question to extract the best answer span from the paragraph. Table 1.1 contains examples of some tasks that require the sentence matching component.

1.2 Deep Neural Networks for Sentence Matching

Traditionally, since the tasks stated above have distinct characteristics, approaches to those tasks have been based on manually extracting specific features inherent in each task (Dagan et al., 2005; Lan and Xu, 2018). This means that, as other traditional algorithms do, an algorithm built for a specific task requires domain-specific or task-specific knowledge and thus could not be easily reused in other tasks.

Meanwhile, with recent advancements in deep neural networks and emergence of large-scale datasets, methodologies based on deep learning have been permeating almost every field of machine learning. Following the success demonstrated in computer vision (Krizhevsky et al., 2012; Goodfellow et al., 2014; Kingma and Welling, 2014; He et al., 2016, *inter alia)*, research on natural language understanding is also

Task	**Sentence A**	**Sentence B**	**Label**
NLI	Two boys jumping on a trampoline.	There are two males.	entailment
NLI	A woman walking with her umbrella.	A woman standing under a scaffolding.	contradiction
PI	How can I get better in math?	What are some ways to get better at maths?	paraphrase
PI	How are organic compounds digested?	What are organic compounds?	not paraphrase
AS	what bird family is the owl	Owls are a group of birds that belong to the order Strigiformes, ...	answer
AS	what bird family is the owl	They are found in all regions of the Earth except Antarctica, ...	not answer

Table 1.1: Example sentence pairs of some sentence matching tasks. NLI: natural language inference, taken from the SNLI dataset (Bowman et al., 2015). PI: paraphrase identification, taken from the Quora Question Pairs dataset (Wang et al., 2017b). AS: answer sentence selection, taken from the WikiQA dataset (Yang et al., 2015).

experiencing unprecedented progress and achievements thanks to sophisticated deep learning algorithms and abundant data.

Recent deep neural network–based work on sentence matching could roughly be categorized into two subclasses: i) methods that exploit inter-sentence features and ii) methods based on sentence encoders. In the former, interaction between the two sentences is allowed in obtaining the representation of each sentence. This includes applying various types of the cross-sentence attention mechanism (Parikh et al., 2016; Chen et al., 2017a; Kim et al., 2019a), dependent reading (Sha et al., 2016; Ghaeini et al., 2018), and iterative answer refinement (Liu et al., 2018). These methods often outperform others that do not allow the inter-sentence interaction, due to more aggressive and direct use of information between sentences.

By contrast, in the latter, i.e. sentence encoder–based methods, each sentence is separately encoded by not seeing each other until the corresponding sentence representation is computed. Then the two sentence vectors are aggregated by a matching function, and finally fed into a classifier network to obtain the prediction. In other words, it is based on the siamese network, which refers to a set of architectures where an encoder is shared across multiple inputs and the encoded representations are aggregated afterwards. Fig. 1.1 depicts the overall architecture of the sentence encoder–based sentence matching method.

1.3 Scope of the Dissertation

Between the two classes of sentence matching methods described above, we address the latter in this dissertation: the sentence encoder–based approach, for the following reasons.

First of all, it is more general. Regardless of the characteristics of each task, the overall structure is fixed to a siamese architecture (Fig. 1.1), thus an improvement

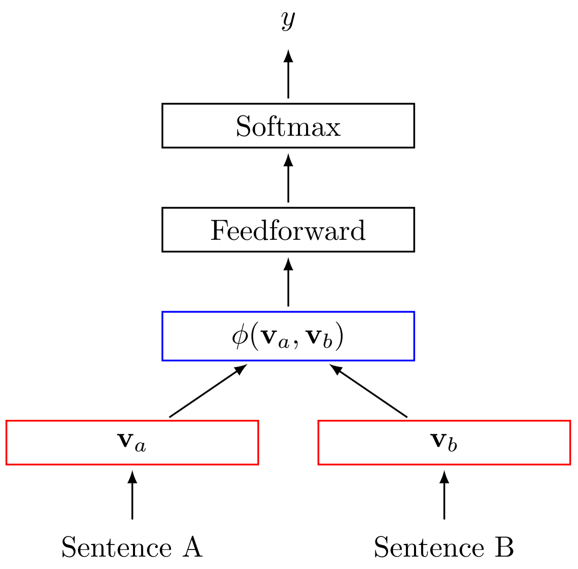


Figure 1.1: Illustration of the sentence encoder–based sentence matching architecture. $\mathbf{v}_a$ and $\mathbf{v}_b$ indicate sentence representations encoded by a sentence encoder for sentence A and B respectively, and $\phi(\mathbf{v}_a, \mathbf{v}_b)$ is a matching function that aggregates $\mathbf{v}_a$ and $\mathbf{v}_b$ into a single vector to be used as input to a classifier MLP network.

proven on one task is likely to work on other tasks and can be adopted in a wide range of work. This generality also conforms to the spirit of the PASCAL recognizing textual entailment (RTE) challenge (Dagan et al., 2005), whose objective was to establish a generic evaluation framework to compare systems for semantic inference.

Moreover, since each sentence is encoded separately, computing sentence vectors, which is often the most time-consuming and resource-intensive stage in sentence matching, could be done priorly, making this type of approach more efficient. This characteristic could be beneficial in cases that involve comparing a query sentence against sentences in a database, e.g. text retrieval. Also due to the property that an encoder only consider a single sentence, it can be used as a general feature extractor and can be used in transfer learning, as demonstrated by Conneau et al. (2017).

Specifically, we find room for improvement of the sentence encoder–based sentence matching in three orthogonal directions: sentence encoder, matching function, and semi-supervised training.

Sentence Encoders

A sentence encoder takes the role of reading and understanding each sentence. As it is the only component that has access to input sentences, it greatly influences the overall performance of sentence matching. In Ch. 3 and 4, we propose two new sentence encoders: Gumbel Tree-LSTM (Choi et al., 2018b) and Cell-aware Stacked LSTM (CAS-LSTM, Choi et al., 2019a).

Gumbel Tree-LSTM is a novel extension of a recursive neural network (RvNN) architecture that removes the need of tree-structured inputs in training and inference. It achieves the property by learning how to parse a sentence in a way that is most effective for a certain task from unstructured (plain) text. At the inference time, it then uses the learned strategy to parse and encode a sequence of words.

CAS-LSTM is a method of stacking multiple long short-term memory (LSTM) layers. At each layer, it accepts as input not only the hidden states but also the memory cell states from the previous layer, by introducing an additional forget gate. Due to the modification, a model could be trained more stably and accomplish improved performance.

Matching Functions

Once the two sentences are encoded using a sentence encoder, the sentence representations should be aggregated into a single vector to be used as input to a classifier network. We refer to a function that fuses two sentence vectors as matching function. In Ch. 5, we propose the element-wise bilinear sentence matching function (ElBiS, Choi et al., 2018a) that automatically finds a suitable matching scheme that maximizes the performance for a task.

The ElBiS algorithm is inspired by bilinear pooling methods suggested in the literature of visual question answering (Fukui et al., 2016; Kim et al., 2017b; Yu et al., 2017). However it exploits the fact that the two sentences are encoded using a shared encoder (i.e. the siamese architecture, as shown in Fig. 1.1) and thus both sentence vectors would lie in the identical or at least very similar semantic spaces, unlike the case of visual question answering where the spaces of text and image representations are clearly distinct. From this motivation, we assume that a certain dimension of one sentence vector would share the same semantical meaning with the corresponding dimension of the others, and consider the bilinear interaction only within values of same dimension.

Semi-Supervised Training

There is another possibility of improvement in utilizing training data efficiently by applying semi-supervised training, where a large amount of unlabeled data along with a handful of labeled data are used in training a model. In Ch. 6, we propose the cross-sentence latent variable model (CS-LVM, Choi et al., 2019b).

CS-LVM is a deep generative model (Kingma and Welling, 2014; Rezende et al., 2014) that is specialized for sentence matching. The optimization objective for the CS-LVM framework is extended to make use of unlabeled data, enabling the semi-supervised training. As it is trained to generate a sentence that has a given relationship with a source sentence, both sentences in a pair are utilized together and thus training objectives are defined more naturally than other models that consider each sentence separately (Zhao et al., 2018; Shen et al., 2018a).

Chapter 2

Background and Related Work

Throughout the dissertation, we try to address the sentence matching problem in three orthogonal point of view: i) constructing a powerful sentence encoder, ii) designing a better matching function, and iii) introducing semi-supervised training. In this chapter, we will briefly see preliminaries related to the three components and review prior work.

2.1 Sentence Encoders

Techniques for mapping natural language into a vector space have received a lot of attention, due to their capability of representing ambiguous semantics of natural language using dense vectors. Methods of learning word representations, e.g. word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014), are relative well-studied empirically and theoretically (Baroni et al., 2014; Levy and Goldberg, 2014), and some of them became typical choices to consider when initializing word representations for better performance at downstream tasks.

Whereas, research on sentence representation is still in active progress, and accordingly various architecture—designed with different intuition and tailored for different tasks—are being proposed.

Convolutional neural networks (CNNs, Kim, 2014; Kalchbrenner et al., 2014) utilize local distribution of words to encode sentences, similar to n-gram models. To handle variable-length inputs, a pooling operation is often applied over time, and thus temporal dependencies longer than the size of a convolution window are ignored, which limits the expressivity in exchange for the lightweight and parallelizable computation.

Recurrent neural networks (RNNs, Graves, 2012; Dai and Le, 2015; Kiros et al., 2015; Hill et al., 2016) encode sentences by reading words in sequential order. Among several variants of the original RNN (Elman, 1990), gated recurrent architectures such as long short-term memory (LSTM, Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRU, Cho et al., 2014b) have been accepted as de-fact standard choices for RNNs due to their capability of addressing the vanishing and exploding gradient problem and considering long-term dependencies. Gated RNNs achieve these properties by introducing additional gating units that learn to control the amount of information to be transferred or forgotten (Goodfellow et al., 2016), and are proven to work well without relying on complex optimization algorithms or careful initialization (Sutskever, 2013).

At the same time, the common practice for further enhancing the expressive power of RNNs is to stack multiple RNN layers, each of which has distinct parameter sets (stacked RNN, Schmidhuber, 1992; El Hihi and Bengio, 1995). Stacked RNNs are shown to work well due to increased depth (Pascanu et al., 2014) or their ability to capture hierarchical time series (Hermans and Schrauwen, 2013) which are inherent to the nature of the problem being modeled.

Recursive neural networks (RvNNs, Socher et al., 2011a; Irsoy and Cardie, 2014;

Bowman et al., 2016a) rely on structured input (e.g. parse tree) to encode sentences, based on the intuition that there is significant semantics in the hierarchical structure of words. It is also notable that RvNNs are generalization of RNNs, as linear chain structures on which RNNs operate are equivalent to left- or right-skewed trees. Similar to the fact that gated RNN like LSTM and GRU is widely used in practice, gated RvNN architectures such as tree-LSTM (Tai et al., 2015; Zhu et al., 2015; Le and Zuidema, 2015) and tree-GRU are also proposed.

Recently, self-attention–based architectures (or Transformer architectures, Vaswani et al., 2017; Shen et al., 2018b), where CNN or RNN components are replaced by the highly parallelizable attention mechanism, are also widely used in encoding sentences (Shen et al., 2018b,c).

2.2 Matching Functions

The classifier network used in our setting of sentence matching takes two vectors from each input sentence as input. This means that the two vectors should be aggregated into a single vector before passed into a feedforward network, and the design of an aggregation function, which we call matching function, is an important factor; if a matching function does not sufficiently reflect the nature of a task then a model could not perform well even with a sophisticated feedforward network since the input can only give limited information to the network.

One might argue that the theoretical fact states that even a single-hidden layer feedforward network can approximate any arbitrary function (Cybenko, 1989; Hornik, 1991), however despite the theory the space of network parameters is too large and it is always helpful to narrow down the search space by directly giving information about interaction between the two sentences to the subsequent network.

Ji and Eisenstein (2013) empirically proved that the use of element-wise multiplication and absolute difference as a matching function substantially improves performance on paraphrase identification, and Tai et al. (2015) applied the same matching scheme to the semantic relatedness prediction task. Mou et al. (2016) showed that using the element-wise multiplication and difference along with the concatenation of sentence vectors yields a gain in performance in natural language inference, despite the fact that some values are redundant and could be induced from another values; for example the element-wise difference can be achieved via a simple linear transformation from the values in the concatenated vector. Yogatama et al. (2017); Chen et al. (2017b) used modified versions of the herustic matching function proposed by Mou et al. (2016) in natural language inference.

There is some prior work on automatically discovering a matching function suitable for a certain task, though not directly related to sentence matching. Recursive neural tensor network (RNTN, Socher et al., 2013) introduces a tensor multiplication to compose two children vectors in tree-structured neural networks. Lin et al. (2015) applied obtained a feature vector by applying outer product between outputs produced by passing an image through two CNNs. Wu et al. (2016); Krause et al. (2016) proposed multiplicative RNN (and LSTM) architectures that exploit bilinear relation between an input and the previous hidden state at each time step.

Also, there have been several works built for matching vectors from different semantic spaces (i.e. matching heterogeneous vectors). Wu et al. (2013) used a bilinear model to match queries and documents from different domains. Approximate bilinear matching techniques such as multimodal compact bilinear pooling (MCB, Fukui et al., 2016), low-rank bilinear pooling (MLB, Kim et al., 2017b), and factorized bilinear pooling (MFB, Yu et al., 2017) are successfully applied in visual question answering (VQA) tasks, outperforming previous heuristic feature functions (Xiong et al., 2016; Agrawal et al., 2017). MCB approximates the full bilinear matching using Count Sketch (Charikar et al., 2002) algorithm, and MLB and MFB decompose a third-order tensor into multiple weight matrices. Multimodal Tucker fusion (MUTAN, Ben-younes et al., 2017) uses Tucker decomposition to parameterize bilinear interactions.

2.3 Semi-Supervised Training

Another important research direction of machine learning is to learn from data efficiently. Mitigating the data scarcity problem is exceptionally important in the era of deep neural networks, as deep learning is believed to require abundant training data to learn appropriate features and outperform classical models.

The most simple and straightforward way of addressing the issue would simply gathering or constructing more labeled training data, however it is a time-consuming and labor-intensive process and not always an available option. Semi-supervised learning aims to handle the problem by taking advantage of unlabeled data which is much easier to collect (Chapelle et al., 2010). For example in sentence matching, possibly related sentence pairs could be retrieved via simple heuristics such as word overlap. These unlabeled data enable a supervised model to learn fairly well even from a small amount of labeled data.

Semi-supervised text classification is an important subject and there exists much previous research (Zhu et al., 2003; Nigam et al., 2006; Zhu, 2008, to name a few). Notably, deep probabilistic generative models (Kingma et al., 2014; Rezende et al., 2014) are capable of learning from unlabeled data by applying techniques of probabilistic inference e.g. marginalization. The work of Xu et al. (2017) applies the semi-supervised variational auto-encoder (VAE, Kingma et al., 2014) to the single-sentence text classification problem. Zhao et al. (2018); Shen et al. (2018a) presented VAE models for the semi-supervised sentence matching.

However, VAE models for sentence matching have some drawbacks that we will try to address in this dissertation. Current VAE-based sentence matching models (Zhao et al., 2018; Shen et al., 2018a) exploit the VAE component (more specifically, approximate inference model of the VAE) as a simple feature extractor, and the training objective is merely a linear combination of the auto-encoding and the classifier objective. Thus the classifier network is updated only by labeled data, and it in turn means that unsupervised training could not fully benefit from carefully designed probabilistic inference (i.e. marginalization). This weakens the coupling between classifier and VAE parameter, which might make the objectives for the two components compete and make a model prone to degenerate.

Outside the research on deep generative models, Dai and Le (2015); Ramachandran et al. (2017) train an encoder-decoder network on large external corpora and fine-tune the learned encoder on a specific task. Also recently there have been remarkable improvements in pre-trained language representations (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018), where language models trained on extremely large data brought a huge performance boost. Since these pre-trained language representations act as a dynamic (or contextualized) weight initialization scheme from which a model starts to learn, they could be used along with other semi-supervised models e.g. VAE-based ones for further improvement.

Chapter 3

Sentence Encoder: Gumbel Tree-LSTM

3.1 Motivation

Recursive neural networks (RvNNs) (Socher et al., 2011a; Irsoy and Cardie, 2014) use structure information of a sentence—constituency-based parse trees or dependency-based parse trees for example—in encoding a sentence, based on the intuition that significant semantics lies in the hierarchical structure of words.

Although there is significant benefit in processing a sentence in a tree-structured recursive manner, data annotated with parse trees could be expensive to prepare. Furthermore, the optimal hierarchical composition of words might differ depending on the properties of a task.

In this chapter, we propose Gumbel Tree-LSTM, which is a novel RvNN architecture that does not require structured data and learns to compose task-specific tree structures without explicit guidance. Our Gumbel Tree-LSTM model is based on tree-structured long short-term memory Tree-LSTM) architecture (Tai et al., 2015;

Zhu et al., 2015; Le and Zuidema, 2015), which is one of the most renowned variants of RvNN.

To learn how to compose task-specific tree structures without depending on structured input, our model introduces composition query vector that measures validity of a composition. Using validity scores computed by the composition query vector, our model recursively selects compositions until only a single representation remains. We use Straight-Through (ST) Gumbel-Softmax estimator (Jang et al., 2017; Maddison et al., 2017) to sample compositions in the training phase. ST Gumbel-Softmax estimator relaxes the discrete sampling operation to be continuous in the backward pass, thus our model can be trained via the standard backpropagation. Also, since the computation is performed layer-wise, our model is easy to implement and naturally supports batched computation.

The contributions of our work are as follows. Firstly, We designed a novel sentence encoder architecture that learns to compose task-specific trees from plain text data. Also, We showed from experiments that the proposed architecture outperforms or is competitive to state-of-the-art models. We also observed that our model converges faster than others. Finally, we saw that our model significantly outperforms previous RvNN works trained on parse trees in all conducted experiments, from which we hypothesize that syntactic parse tree may not be the best structure for every task and the optimal structure could differ per task.

3.2 Preliminaries

3.2.1 Recursive Neural Networks

Recursive neural networks (RvNNs) (Socher et al., 2011a; Irsoy and Cardie, 2014) use structure information of a sentence—constituency-based parse trees or dependency-based parse trees for example—in encoding a sentence, based on the intuition that significant semantics lies in the hierarchical structure of words.

Socher et al. (2011a) proposed the recursive auto-encoder architecture that composes two children nodes in a binary tree using a feedforward neural network. To enhance the compositionality between constituents, Socher et al. (2012) designed the matrix-vector RvNN architecture that maintains both a matrix and a vector for each constituent and defines the composition process as a matrix-vector product. From similar motivation, Socher et al. (2013) modeled the composition as a tensor product and reduced the number of required parameters by a large margin compared to the matrix-vector RvNN architecture.

Analogous to the trend that recurrent neural network (RNN) architectures armed with a gating mechanism became the de-facto standard, e.g. long short-term memory (LSTM, Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRU, Cho et al., 2014b), Tai et al. (2015); Zhu et al. (2015); Le and Zuidema (2015) proposed tree-structured LSTM architectures (tree-LSTMs) that computes the representation of a parent constituent using LSTM-like gating functions. Along with the exemplars described, several variants of RvNN have also been suggested (Hashimoto et al., 2013; Dong et al., 2014; Qian et al., 2015; Liu et al., 2017; Teng and Zhang, 2017; Wang et al., 2017a; Huang et al., 2017; Kim et al., 2019b, to name a few).

3.2.2 Training RvNNs without Tree Information

Although there is significant benefit in processing a sentence in a tree-structured recursive manner, data annotated with parse trees could be expensive to prepare. Furthermore, the optimal hierarchical composition of words might differ depending on the properties of a task. To address the drawbacks, several works that aim to learn hierarchical latent structure of text by recursively composing words into sentence representation, without assuming that tree information is given in the training dataset.

To the best of our knowledge, gated recursive convolutional neural network (grConv, Cho et al., 2014a) is the first model of its kind and was used as an encoder for neural machine translation. The grConv architecture uses gating mechanism to control the information flow from children to parent. grConv and its variants are also applied to sentence classification tasks (Chen et al., 2015; Zhao et al., 2015). Neural tree indexer (NTI, Munkhdalai and Yu, 2017b) utilizes soft hierarchical structures by using tree-LSTM instead of grConv.

Although models that operate with soft structures are naturally capable of being trained via backpropagation, the structures predicted by them are ambiguous and thus it is hard to interpret them. CYK Tree-LSTM (Maillard et al., 2017, 2019) resolves the ambiguity while maintaining the soft property by introducing the concept of the CYK parsing algorithm (Cocke and Schwartz, 1970; Younger, 1967; Kasami, 1965). Though their model reduces the ambiguity by explicitly representing a node as a weighted sum of all candidate compositions, it is memory intensive since the number of candidates linearly increases by depth.

On the other hand, there exists some previous work that maintains the discreteness of tree composition processes, instead of relying on the soft hierarchical structure. The architecture proposed by Socher et al. (2011b) greedily selects two adjacent nodes whose reconstruction error is the smallest and merges them into the parent. In their work, to guide a model to learn a meaningful parsing strategy, reconstruction error is used along with the classification loss and the L-BFGS (Liu and Nocedal, 1989) algorithm over the complete training data is used in training.

Yogatama et al. (2017) introduce reinforcement learning to achieve the desired effect of discretization. They show that REINFORCE (Williams, 1992) algorithm can be used in estimating gradients to learn a tree composition function minimizing classification error. However, slow convergence due to the reinforcement learnign setting is one of its drawbacks, according to the authors.

In research areas outside the RvNN, compositionality in a vector space also has been a longstanding subject (Plate, 1995; Mitchell and Lapata, 2010; Grefenstette and Sadrzadeh, 2011; Zanzotto and Dell'Arciprete, 2012, to list but a few). And more recently, there exists work aiming to learn hierarchical latent structure from unstructured data (Kim et al., 2017c; Chung et al., 2017).

3.3 Model Description

Our proposed architecture is built based on the tree-structured long short-term memory architecture. We introduce several additional components into the tree-LSTM architecture to allow model to dynamically compose tree structures in a bottom-up manner and to effectively encode a sentence into a vector. In the following, we describe the components of our Gumbel Tree-LSTM model in detail.

3.3.1 Tree-LSTM

Tree-structured long short-term memory network (tree-LSTM, Tai et al., 2015; Zhu et al., 2015; Le and Zuidema, 2015) is an elegent variant of RvNN, where it controls information flow from from children to parent using similar mechanism to the LSTM (Hochreiter and Schmidhuber, 1997). Tree-LSTM introduces *cell state* in computing a parent representation, which assists each cell to capture distant vertical dependencies.

The following are formulae that our model uses to compute parent representation from its children:

$$\begin{bmatrix} \mathbf{i} \\ \mathbf{f}_l \\ \mathbf{f}_r \\ \mathbf{o} \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(\mathbf{W}_{comp} \begin{bmatrix} \mathbf{h}_l \\ \mathbf{h}_r \end{bmatrix} + \mathbf{b}_{comp} \right) \tag{3.1}$$

$$\mathbf{c}_p = \mathbf{f}_l \odot \mathbf{c}_l + \mathbf{f}_r \odot \mathbf{c}_r + \mathbf{i} \odot \mathbf{g} \tag{3.2}$$

$$\mathbf{h}_p = \mathbf{o} \odot \tanh(\mathbf{c_p}), \tag{3.3}$$

where $\mathbf{W}_{comp} \in \mathbb{R}^{5D_h \times 2D_h}$ $\mathbf{b}_{comp} \in \mathbb{R}^{2D_h}$, and $\odot$ is the element-wise product. Note that our formulation is akin to that of SPINN (Bowman et al., 2016a), but our version does not include the tracking LSTM. Instead, our model can apply an LSTM to leaf nodes, which we will soon describe.

3.3.2 Gumbel-Softmax

Gumbel-Softmax (Jang et al., 2017) (or Concrete distribution, Maddison et al., 2017) is a method of utilizing discrete random variables in a network. Since it approximates one-hot vectors sampled from a categorical distribution by making them continuous, gradients of model parameters can be calculated using the reparameterization trick and the standard backpropagation. Gumbel-Softmax is known to have an advantage over score-function-based gradient estimators such as REINFORCE (Williams, 1992) which suffer from high variance and slow convergence (Jang et al., 2017).

Gumbel-Softmax distribution is motivated by Gumbel-Max trick (Maddison et al., 2014), an algorithm for sampling from a categorical distribution. Consider a k-dimensional categorical distribution whose class probabilities $p_1, \ldots, p_k$ are defined in terms of unnormalized log probabilities $\pi_1, \ldots, \pi_k$:

$$p_i = \frac{\exp(\log(\pi_i))}{\sum_{j=1}^{k} \exp(\log(\pi_j))}. \tag{3.4}$$

Then a one-hot sample $\mathbf{z} = (z_1, \ldots, z_k) \in \mathbb{R}^k$ from the distribution can be easily drawn by the following equations:

$$z_i = \begin{cases} 1 & i = \operatorname{argmax}_j(\log(\pi_j) + g_j) \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

$$g_i = -\log(-\log(u_i)) \tag{3.6}$$

$$u_i \sim \mathcal{U}(0, 1), \tag{3.7}$$

where $\mathcal{U}(a, b)$ is the uniform distribution whose minimum and maximum value is a and b respectively. Here, $g_i \sim \text{Gumbel}(0, 1)$, namely *Gumbel noise*, perturbs each $\log(\pi_i)$ term so that taking argmax becomes equivalent to drawing a sample weighted on $p_1, \ldots, p_k$.

In Gumbel-Softmax, the discontinuous argmax function of Gumbel-Max trick is replaced by the differentiable softmax function. That is, given unnormalized probabilities $\pi_1, \ldots, \pi_k$, and Gumbel noises $g_1, \ldots, g_k \sim \text{Gumbel}(0, 1)$, a sample $\mathbf{y} = (y_1, \ldots, y_k)$ from the Gumbel-Softmax distribution is drawn by

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)}, \tag{3.8}$$

where τ is a temperature parameter; as τ diminishes to zero, a sample from the Gumbel-Softmax distribution becomes *cold* and resembles the one-hot sample.

Straight-Through (ST) Gumbel-Softmax estimator (Jang et al., 2017), whose name reminds of Straight-Through estimator (STE) (Bengio et al., 2013), is a discrete version of the continuous Gumbel-Softmax estimator. Similar to the STE, it maintains sparsity by taking different paths in the forward and backward propagation. Obviously ST estimators are biased, however they perform well in practice, according to several previous works (Chung et al., 2017; Gu et al., 2018) and our own result.

In the forward pass, it discretizes a continuous probability vector $\mathbf{y}$ sampled from the Gumbel-Softmax distribution into the one-hot vector $\mathbf{y}^{ST} = (y_1^{ST}, \ldots, y_k^{ST})$, where

$$y_i^{ST} = \begin{cases} 1 & i = \operatorname{argmax}_j y_j \\ 0 & \text{otherwise} \end{cases}. \tag{3.9}$$

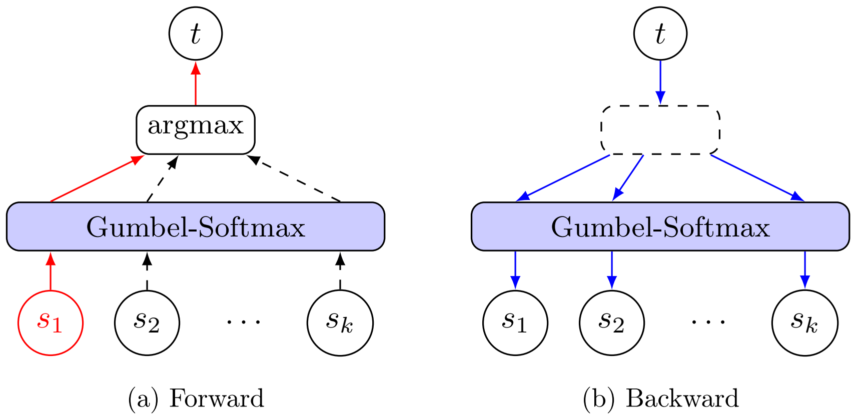


Figure 3.1: Visualization of forward and backward computation path of ST Gumbel-Softmax. In the forward pass, a model can maintain sparseness due to argmax operation. In the backward pass, since there is no discrete operation, the error signal can backpropagate.

And in the backward pass it simply uses the continuous $\mathbf{y}$, thus the error signal is still able to backpropagate. See Fig. 3.1 for the visualization of the forward and backward pass.

ST Gumbel-Softmax estimator is useful when a model needs to utilize discrete values directly, for example in the case that a model alters its computation path based on samples drawn from a categorical distribution.

3.3.3 Gumbel Tree-LSTM

In our Gumbel Tree-LSTM model, an input sentence composed of N words is represented as a sequence of word vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$, where $\mathbf{x}_i \in \mathbb{R}^{D_x}$. Our basic model applies an affine transformation to each $\mathbf{x}_i$ to obtain the initial hidden and cell state:

$$\mathbf{r}_i^1 = \begin{bmatrix} \mathbf{h}_i^1 \\ \mathbf{c}_i^1 \end{bmatrix} = \mathbf{W}_{leaf}\mathbf{x}_i + \mathbf{b}_{leaf}, \tag{3.10}$$

which we call *leaf transformation*. In Eq. 3.10, $\mathbf{W}_{leaf} \in \mathbb{R}^{2D_h \times D_x}$ and $\mathbf{b}_{leaf} \in \mathbb{R}^{2D_h}$. Note that we denote the representation of i-th node at t-th layer as $\mathbf{r}_i^t = \left[\mathbf{h}_i^t; \mathbf{c}_i^t\right]$.

Assume that t-th layer consists of M_t node representations: $(\mathbf{r}_1^t, \ldots, \mathbf{r}_{M_t}^t)$. If two adjacent nodes, say $\mathbf{r}_i^t$ and $\mathbf{r}_{i+1}^t$, are selected to be merged, then Eqs. 3.1–3.3 are applied by assuming $[\mathbf{h}_l; \mathbf{c}_l] = \mathbf{r}_i^t$ and $[\mathbf{h}_r; \mathbf{c}_r] = \mathbf{r}_{i+1}^t$ to obtain the parent representation $[\mathbf{h}_p; \mathbf{c}_p] = \mathbf{r}_i^{t+1}$. Node representations which are not selected are copied to the corresponding positions at layer $t+1$. In other words, the $(t+1)$-th layer is composed of $M_{t+1} = M_t - 1$ representations $(\mathbf{r}_1^{t+1}, \ldots, \mathbf{r}_{M_{t+1}}^t)$, where

$$\mathbf{r}_j^{t+1} = \begin{cases} \mathbf{r}_j^t & j < i \\ \text{Tree-LSTM}\left(\mathbf{r}_j^t, \mathbf{r}_{j+1}^t\right) & j = i \cdot \\ \mathbf{r}_{j+1}^t & j > i \end{cases} \tag{3.11}$$

This procedure is repeated until the model reaches N-th layer and only a single node is left. It is notable that the property of selecting the best node pair at each stage resembles that of the easy-first parsing (Goldberg and Elhadad, 2010).

Parent selection

Since information about the tree structure of an input is not given to the model, a special mechanism is needed for the model to learn to compose task-specific tree structures in an end-to-end manner. We now describe the mechanism for building up the tree structure from an unstructured sentence.

First, our model introduces the trainable *composition query vector* $\mathbf{q} \in \mathbb{R}^{D_h}$. The composition query vector measures how valid a representation is. Specifically, the *validity score* of a representation $\mathbf{r} = [\mathbf{h}; \mathbf{c}]$ is defined by $\mathbf{q} \cdot \mathbf{h}$.

At layer t, the model computes candidates for the parent representations using Eqs. 3.1–3.3: $(\tilde{\mathbf{r}}_1^{t+1}, \ldots, \tilde{\mathbf{r}}_{M_{t+1}}^{t+1})$. Then, it calculates the validity score of each candidate

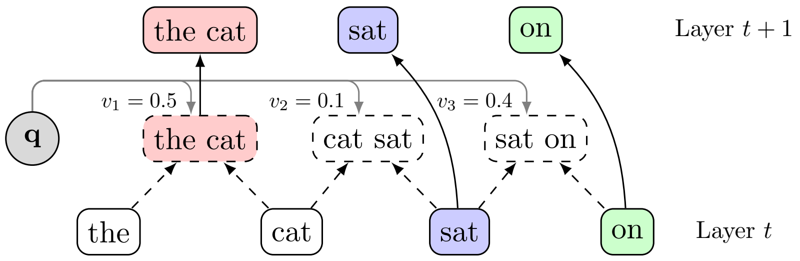


Figure 3.2: An example of the parent selection. At layer t (the bottom layer), the model computes parent candidates (the middle layer). Then the validity score of each candidate is computed using the query vector $\mathbf{q}$ (denoted as v_1, v_2, v_3). In the training time, the model samples a parent node among candidates weighted on v_1, v_2, v_3, using ST Gumbel-Softmax estimator, and in the testing time the model selects the candidate with the highest validity. At layer $t+1$ (the top layer), the representation of the selected candidate ('the cat') is used as a parent, and the rest are copied from those of layer t ('sat', 'on'). Best viewed in color.

and normalize it so that $\sum_{i=1}^{M_{t+1}} v_i = 1$:

$$v_i = \frac{\exp(\mathbf{q} \cdot \tilde{\mathbf{h}}_i^{t+1})}{\sum_{j=1}^{M_{t+1}} \exp(\mathbf{q} \cdot \tilde{\mathbf{h}}_j^{t+1})}. \tag{3.12}$$

In the training phase, the model samples a parent from candidates weighted on v_i, using the ST Gumbel-Softmax estimator described above. Since the continuous Gumbel-Softmax function is used in the backward pass, the error backpropagation signal safely passes through the sampling operation, hence the model is able to learn to construct the task-specific tree structures that minimize the loss by backpropagation.

In the validation (or testing) phase, the model simply selects the parent which maximizes the validity score.

An example of the parent selection is depicted in Fig. 3.2

LSTM-based leaf transformation

The basic leaf transformation using an affine transformation (Eq. 3.10) does not consider information about the entire sentence of an input and thus the parent selection is performed based only on local information.

SPINN (Bowman et al., 2016a) addresses this issue by using the tracking LSTM which sequentially reads input words. The tracking LSTM makes the SPINN model *hybrid*, where the model takes advantage of both tree-structured composition and sequential reading. However, the tracking LSTM is not applicable to our model, since our model does not use shift-reduce parsing or maintain a stack.

In the tracking LSTM's stead, our model applies an LSTM on input representations to give information about previous words to each leaf node:

$$\mathbf{r}_i^1 = \begin{bmatrix} \mathbf{h}_i^1 \\ \mathbf{c}_i^1 \end{bmatrix} = \text{LSTM}(\mathbf{x_i}, \mathbf{h}_{i-1}^1, \mathbf{c}_{i-1}^1), \tag{3.13}$$

where $\mathbf{h}_0^1 = \mathbf{c}_0^1 = \vec{0}$.

From the experimental results, we validate that the LSTM applied to leaf nodes has a substantial gain over the basic leaf transformation function.

3.4 Implementation Details

Implementation-wise, we used multiple *mask* matrices in implementing the proposed Gumbel Tree-LSTM model. Using the mask matrices, Eq. 3.11 can be rewritten as the following single equation:

$$\mathbf{r}_{1:M_{t+1}}^{t+1} = \mathbf{M}_l \odot \mathbf{r}_{1:M_t-1}^t + \mathbf{M}_r \odot \mathbf{r}_{2:M_t}^t + \mathbf{M}_p \odot \tilde{\mathbf{r}}_{1:M_{t+1}}^{t+1}. \tag{3.14}$$

In the above equation, $\mathbf{M}_l, \mathbf{M}_r, \mathbf{M}_p \in \mathbb{R}^{D_h \times M_{t+1}}$, and $\mathbf{r}_{1:L}^t \in \mathbb{R}^{D_h \times L}$ is a matrix whose columns are $\mathbf{r}_1^t, \ldots, \mathbf{r}_L^t \in \mathbb{R}^{D_h}$.

The mask matrices are defined by the following equations.

$$\mathbf{M}_l = \begin{bmatrix} \mathbf{m}_l & \cdots & \mathbf{m}_l \end{bmatrix}^\top \tag{3.15}$$

$$\mathbf{M}_r = \begin{bmatrix} \mathbf{m}_r & \cdots & \mathbf{m}_r \end{bmatrix}^\top \tag{3.16}$$

$$\mathbf{M}_p = \begin{bmatrix} \mathbf{m}_p & \cdots & \mathbf{m}_p \end{bmatrix}^\top \tag{3.17}$$

$$\mathbf{m}_l = \mathbf{1} - \texttt{cumsum}(\bar{\mathbf{y}}_{1:M_{t+1}}) \tag{3.18}$$

$$\mathbf{m}_r = \begin{bmatrix} 0 & \texttt{cumsum}(\bar{\mathbf{y}}_{1:M_{t+1}-1}) \end{bmatrix}^\top \tag{3.19}$$

$$\mathbf{m}_p = \bar{\mathbf{y}}_{1:M_{t+1}} \tag{3.20}$$

Here, $\texttt{cumsum}(\mathbf{c})$ is a function that takes a vector $\mathbf{c} = [c_1, \ldots, c_k]^\top$ and outputs a vector $\mathbf{d} = [d_1, \ldots, d_k]^\top$ s.t. $d_i = \sum_{j=1}^{i} c_j$. $\bar{\mathbf{y}}_{1:M_{t+1}} \in \mathbb{R}^{M_{t+1}}$ is a vector which will be defined below, and $\mathbf{1} \in \mathbb{R}^{M_{t+1}}$ is a vector whose values are all ones.

In the forward pass, $\bar{\mathbf{y}}_{1:M_{t+1}}$ is defined by a one-hot vector $\mathbf{y}^{ST}_{1:M_{t+1}}$, which is sampled from the categorical distribution of validity scores $(v_1, \ldots, v_{M_{t+1}})$ using Gumbel-Max trick.

$$y_i^{ST} = \begin{cases} 1 & i = \operatorname{argmax}_j \left(\mathbf{q} \cdot \tilde{\mathbf{h}}_j^{t+1} + g_j \right) \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

$$g_i = -\log(-\log(u_i + \epsilon) + \epsilon) \tag{3.22}$$

$$u_i \sim \mathcal{U}(0, 1) \tag{3.23}$$

Note that $\epsilon = 10^{-20}$ is added when calculating g_i for numerical stability.

In the backward pass, instead of the one-hot version, the continuous vector $\mathbf{y}_{1:M_{t+1}}$ obtained from Gumbel-Softmax is used as $\bar{\mathbf{y}}_{1:M_{t+1}}$. Note that the Gumbel noise samples $g_1, \ldots, g_{M_{t+1}}$ drawn in the forward pass are reused in the backward pass (i.e. noise values are not re-sampled in the backward pass).

In typical deep learning libraries supporting automatic differentiation such as PyTorch (Paszke et al., 2017) and TensorFlow (Abadi et al., 2016), this discrepancy between forward and backward pass can be implemented as

$$\bar{\mathbf{y}}_{1:M_{t+1}} = \texttt{detach}(\mathbf{y}^{ST}_{1:M_{t+1}} - \mathbf{y}_{1:M_{t+1}}) + \mathbf{y}_{1:M_{t+1}}, \tag{3.24}$$

where $\texttt{detach}(\cdot)$ is the special function that prevents error from backpropagating through its input.

3.5 Experiments

We evaluate performance of the proposed Gumbel Tree-LSTM model on two tasks: natural language inference and sentiment analysis. The implementation is made publicly available.[1]

3.5.1 Natural Language Inference

Natural language inference (NLI) is a task of predicting the relationship between two sentences (hypothesis and premise). In the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015), which we use for NLI experiments, a relationship is either contradiction, entailment, or neutral. For a model to correctly predict the relationship between two sentences, it should encode semantics of sentences accurately, thus the task has been used as one of standard tasks for evaluating the quality of sentence representations.

The SNLI dataset is composed of about 550,000 sentences, each of which is binary-parsed. However, since our model operate on plain text, we do not use the parse tree information in both training and testing. The classifier architecture used in our SNLI experiments is similar to the ones used in Mou et al. (2016); Chen et al. (2017b). Given the premise sentence vector ($\mathbf{h}^{pre}$) and the hypothesis sentence vector ($\mathbf{h}^{hyp}$)

[1] `https://github.com/jihunchoi/unsupervised-treelstm`

which are encoded by the proposed Gumbel Tree-LSTM model, the probability of relationship $r \in \{\text{entailment}, \text{contradiction}, \text{neutral}\}$ is computed by the following equations:

$$p(r|\mathbf{h}^{pre}, \mathbf{h}^{hyp}) = \text{softmax}(\mathbf{W}^r_{clf}\mathbf{a} + \mathbf{b}^r_{clf}) \tag{3.25}$$

$$\mathbf{a} = \Phi(\mathbf{f}) \tag{3.26}$$

$$\mathbf{f} = \begin{bmatrix} \mathbf{h}^{pre} \\ \mathbf{h}^{hyp} \\ \left|\mathbf{h}^{pre} - \mathbf{h}^{hyp}\right| \\ \mathbf{h}^{pre} \odot \mathbf{h}^{hyp} \end{bmatrix}, \tag{3.27}$$

where $\mathbf{W}^r_{clf} \in \mathbb{R}^{1\times D_c}$, $\mathbf{b}^r_{clf} \in \mathbb{R}^1$, and Φ is a multi-layer perceptron (MLP) with the rectified linear unit (ReLU) activation function.

The composition query vector is initialized by sampling from Gaussian distribution $\mathcal{N}(0, 0.01^2)$. The last linear transformation that outputs the unnormalized log probability for each class is initialized by sampling from uniform distribution $\mathcal{U}(-0.005, 0.005)$. All other parameters are initialized following the scheme proposed by He et al. (2015). We used Adam optimizer (Kingma and Ba, 2015) with default hyperparameters and halved learning rate if there is no improvement in accuracy for one epoch. The size of mini-batch is set to 128 in all experiments. The temperature parameter τ of Gumbel-Softmax is set to 1.0, and we did not find that temperature annealing improves performance.

For 100D experiments (where $D_x = D_h = 100$), we use a single-hidden layer MLP with 200 hidden units (i.e. $D_c = 200$. The word vectors are initialized with GloVe (Pennington et al., 2014) 100D pretrained vectors[2] and fine-tuned during training.

For 300D experiments (where $D_x = D_h = 300$), we set the number of hidden units of a single-hidden layer MLP to 1024 ($D_c = 1024$) and added batch normalization

[2]`http://nlp.stanford.edu/data/glove.6B.zip`

layers (Ioffe and Szegedy, 2015) followed by dropout (Srivastava et al., 2014) with probability 0.1 to the input and the output of the MLP. We also apply dropout on the word vectors with probability 0.1. Similar to 100D experiments, we initialize the word embedding matrix with GloVe 300D pretrained vectors[3], however we do not update the word representations during training.

Since our model converges relatively fast, it is possible to train a model of larger size in a reasonable time. In the 600D experiment, we set $D_x = 300$, $D_h = 600$, and an MLP with three hidden layers ($D_c = 1024$) is used. The dropout probability is set to 0.2 and word embeddings are not updated during training.

The results of SNLI experiments are summarized in Table 3.1. First, we can see that LSTM-based leaf transformation has a clear advantage over the affine-transformation-based one. It improves the performance substantially and also leads to faster convergence.

Secondly, comparing ours with other models, we find that our 100D and 300D model outperform all other models of similar numbers of parameters. Our 600D model achieves the accuracy of 86.0%, which is comparable to that of the state-of-the-art model (Nie and Bansal, 2017), while using far less parameters.

It is also worth noting that our models converge much faster than other models. All of our models converged within a few hours on a machine with NVIDIA Titan Xp GPU. Note that the models of Yogatama et al. (2017); Maillard et al. (2017), which share the same objective of learning task-specific tree structures as ours, are hard to be evaluated in 300D or 600D settings, due to slow convergence or large memory consumption.

We also plot validation accuracies of various models during first 5 training epochs in Fig. 3.3, and validate that our models converge significantly faster than others, not only in terms of total training time but also in the number of iterations.

[3]`http://nlp.stanford.edu/data/glove.840B.300d.zip`

Model	**Accuracy (%)**	**# Params**	**Time (hours)**
100D Latent Syntax Tree-LSTM (Yogatama et al., 2017)	80.5	500k	72–96*
100D CYK Tree-LSTM (Maillard et al., 2017)	81.6	231k	240*
100D Gumbel Tree-LSTM, without Leaf LSTM (Ours)	81.8	202k	0.7
100D Gumbel Tree-LSTM (Ours)	**82.6**	262k	**0.6**
300D LSTM (Bowman et al., 2016a)	80.6	3.0M	4†
300D SPINN (Bowman et al., 2016a)	83.2	3.7M	67†
300D NSE (Munkhdalai and Yu, 2017a)	84.6	3.0M	26†
300D Gumbel Tree-LSTM, without Leaf LSTM (Ours)	84.4	2.3M	3.1
300D Gumbel Tree-LSTM (Ours)	**85.6**	2.9M	**1.6**
600D (300+300) Gated-Attention BiLSTM (Chen et al., 2017b)	85.5	11.6M	8.5†
512–1024–2048D Shortcut-Stacked BiLSTM (Nie and Bansal, 2017)	**86.1**	140.2M	3.8†‡
600D Gumbel Tree-LSTM (Ours)	86.0	10.3M	**3.4**

Table 3.1: Results of SNLI experiments. The above two sections group models of similar numbers of parameters. The bottom section contains results of state-of-the-art models. Word embedding parameters are not included in the number of parameters. $*$: values reported in the original papers. $\dagger$: values estimated from per-epoch training time on the same machine our models trained on. $\ddagger$: cuDNN library is used in RNN computation.

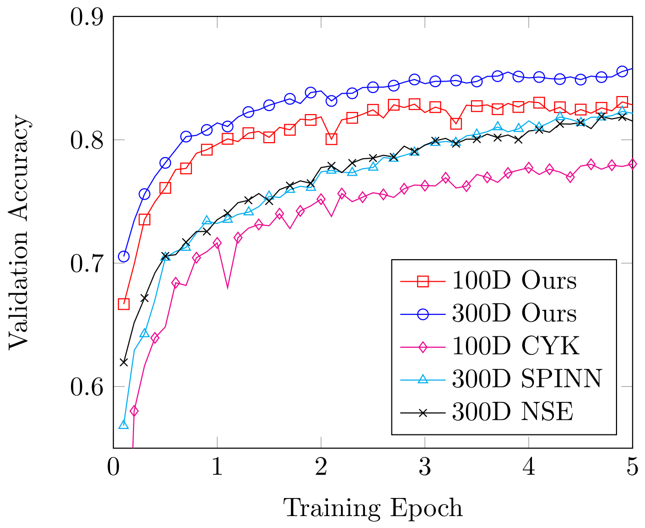


Figure 3.3: Validation accuracies on the SNLI dataset during training. 100D CYK: 100-dimensional unsupervised Tree-LSTM (Maillard et al., 2017). 300D SPINN: 300-dimensional SPINN-PI (Bowman et al., 2016a). 300D NSE: 300-dimensional NSE (Munkhdalai and Yu, 2017a). Our models and 300D NSE are trained with batch size 128. 100D CYK and 300D SPINN are trained with batch size 16 and 32 respectively, as in the original papers. We observed that our models still converge faster than others when a smaller batch size (16 or 32) is used.

3.5.2 Sentiment Analysis

To evaluate the performance of our model in single-sentence classification, we conducted experiments on Stanford Sentiment Treebank (SST) (Socher et al., 2013) dataset. In the SST dataset, each sentence is represented as a binary parse tree, and each subtree of a parse tree is annotated with the corresponding sentiment score. Following the experimental setting of previous works, we use all subtrees and their labels for training, and only the root labels are used for evaluation.

The classifier has a similar architecture to SNLI experiments. Specifically, for a sentence embedding $\mathbf{h}$, the probability for the sentence to be predicted as label $s \in \{0, 1\}$ (in the binary setting, SST-2) or $s \in \{1, 2, 3, 4, 5\}$ (in the fine-grained setting, SST-5) is computed as follows:

$$p(s|\mathbf{h}) = \mathrm{softmax}(\mathbf{W}^{s}_{clf}\mathbf{a} + \mathbf{b}^{s}_{clf}) \tag{3.28}$$

$$\mathbf{a} = \Phi(\mathbf{h}), \tag{3.29}$$

where $\mathbf{W}^{s}_{clf} \in \mathbb{R}^{1 \times D_c}$, $\mathbf{b}^{s}_{clf} \in \mathbb{R}^{1}$, and Φ is a single-hidden layer MLP with the ReLU activation function. Note that subtrees labeled as neutral are ignored in the binary setting in both training and evaluation.

The composition query vector is initialized by sampling from Gaussian distribution $\mathcal{N}(0, 0.01^2)$. The last linear transformation that outputs the unnormalized log probability for each class is initialized by sampling from uniform distribution $\mathcal{U}(-0.002, 0.002)$. All other parameters are initialized following the scheme proposed by He et al. (2015).

We trained our SST-2 model with hyperparameters $D_x = 300$, $D_h = 300$, $D_c = 300$. The word vectors are initialized with GloVe 300D pretrained vectors and fine-tuned during training. We apply dropout ($p = 0.5$) on the output of the word embedding layer and the input and the output of the MLP layer. The size of mini-batches is set to 32, and Adadelta (Zeiler, 2012) optimizer with default hyperparameters is used for optimization. We halved learning rate if there is no improvement in accuracy for two epochs.

For our SST-5 model, hyperparameters are set to $D_x = 300$, $D_h = 300$, $D_c = 1024$. Similar to the SST-2 model, we optimize the model using Adadelta optimizer with batch size 64 and apply dropout with $p = 0.5$.

Table 3.2 summarizes the results of SST experiments. Our SST-2 model outperforms all other models substantially except byte-mLSTM (Radford et al., 2017), where a byte-level language model trained on the large product review dataset is used to obtain sentence representations.

We also see that the performance of our SST-5 model is on par with that of the current state-of-the-art model (McCann et al., 2017), which is pretrained on large parallel datasets and uses character n-gram embeddings alongside word embeddings, even though our model does not utilize external resources other than GloVe vectors and only uses word-level representations. The authors of (McCann et al., 2017) stated that utilizing pretraining and character n-gram embeddings improves validation accuracy by 2.8% (SST-2) or 1.7% (SST-5).

In addition, from the fact that our models substantially outperform all other RvNN-based models, we conjecture that task-specific tree structures built by our model help encode sentences into vectors more efficiently than constituency-based or dependency-based parse trees do.

3.5.3 Qualitative Analysis

We conduct a set of experiments to observe various properties of our trained models. First, to see how well the model encodes sentences with similar meaning or syntax into close vectors, we find nearest neighbors of a query sentence. Second, to validate that the trained composition functions are non-trivial and task-specific, we visualize

Model	**SST-2 (%)**	**SST-5 (%)**
DMN (Kumar et al., 2016)	88.6	52.1
NSE (Munkhdalai and Yu, 2017a)	89.7	52.8
byte-mLSTM (Radford et al., 2017)	**91.8**	52.9
BCN+Char+CoVe (McCann et al., 2017)	90.3	**53.7**
RNTN (Socher et al., 2013)	85.4	45.7
Constituency Tree-LSTM (Tai et al., 2015)	88.0	51.0
NTI-SLSTM-LSTM (Munkhdalai and Yu, 2017b)	89.3	53.1
Latent Syntax Tree-LSTM (Yogatama et al., 2017)	86.5	–
Constituency Tree-LSTM + Recurrent Dropout (Looks et al., 2017)	89.4	52.3
Gumbel Tree-LSTM (Ours)	<u>90.7</u>	<u>**53.7**</u>

Table 3.2: Results of SST experiments. The bottom section contains results of RvNN-based models. Underlined score indicates the best among RvNN-based models.

#	**sunshine is on a man 's face .**	**a girl is staring at a dog .**	**the woman is wearing boots .**
1	a man is walking on sunshine .	the woman is looking at a dog .	the girl is wearing shoes
2	a guy is in a hot , sunny place	a girl takes a photo of a dog .	a person is wearing boots .
3	a man is working in the sun .	a girl is petting her dog .	the woman is wearing jeans .
4	it is sunny .	a man is taking a picture of a dog , while a woman watches .	a woman wearing sunglasses .
5	a man enjoys the sun coming through the window .	a woman is playing with her dog .	the woman is wearing a vest .

Table 3.3: Nearest neighbor sentences of query sentences. Each query sentence is unseen in the dataset.

trees composed by SNLI and SST model given identical sentence.

Nearest neighbors

We encode sentences in the test split of SNLI dataset using the trained 300D model and find nearest neighbors given a query sentence. Table 3.3 presents five nearest neighbors for each selected query sentence. In finding nearest neighbors, cosine distance is used as metric. The result shows that our model effectively maps similar sentences into vectors close to each other; the neighboring sentences are similar to a query sentence not only in terms of word overlap, but also in semantics. For example in the second column, the nearest sentence is 'the woman is looking at a dog', whose meaning is almost same as the query sentence. We can also see that other neighbors partially share semantics with the query sentence.

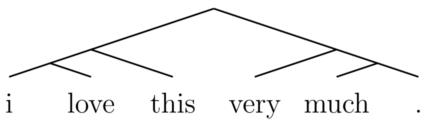
Tree examples

Fig. 3.4 show that two models (300D SNLI and SST-2) generate different tree structures given an identical sentence. In Fig. 3.4a and 3.4b, the SNLI model groups the phrase ‘i love this’ first, while the SST model groups ‘this very much’ first. Fig. 3.4c and 3.4d present how differently the two models process a sentence containing relative pronoun ‘which’. It is intriguing that the models compose visually plausible tree structures, where the sentence is divided into two phrases by relative pronoun, even though they are trained without explicit parse trees. We hypothesize that these examples demonstrate that each model generates a distinct tree structure based on semantic properties of the task and learns non-trivial tree composition scheme.

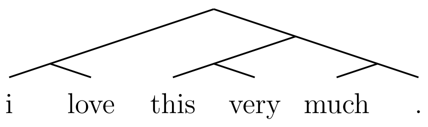
3.6 Summary

We proposed Gumbel Tree-LSTM, a novel Tree-LSTM-based architecture that learns to compose task-specific tree structures. Our model introduces the composition query vector to compute validity of the candidate parents and selects the appropriate parent according to validity scores. In training time, the model samples the parent from candidates using ST Gumbel-Softmax estimator, hence it is able to be trained by standard backpropagation while maintaining its property of discretely determining the computation path in forward propagation.

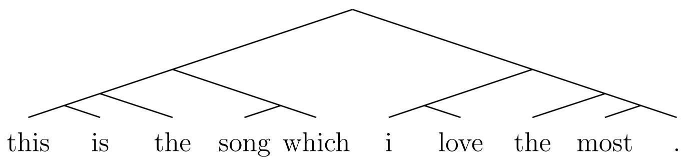
From experiments, we validate that our model outperforms all other RvNN models and is competitive to state-of-the-art models, and also observed that our model converges faster than other complex models. The result poses an important question: *what is the optimal input structure for RvNN?* We empirically showed that the optimal structure might differ per task, and investigating task-specific latent tree structures could be an interesting future research direction.



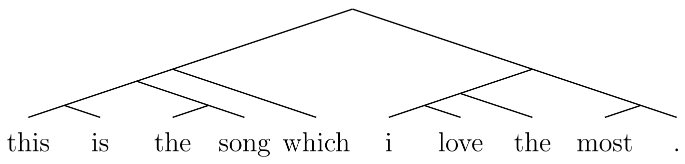
(a) SNLI



(b) SST



(c) SNLI



(d) SST

Figure 3.4: Tree structures built by models trained on SNLI and SST.

Chapter 4

Sentence Encoder: Cell-aware Stacked LSTM

4.1 Motivation

In the field of natural language processing (NLP), one of the most prevalent neural approaches to obtaining sentence representations is to use recurrent neural networks (RNNs), where words in a sentence are processed in a sequential and recurrent manner. Along with their intuitive design, RNNs have shown outstanding performance across various NLP tasks e.g. language modeling (Mikolov et al., 2010; Graves, 2013), machine translation (Cho et al., 2014c; Sutskever et al., 2014; Bahdanau et al., 2015), text classification (Zhou et al., 2015; Tang et al., 2015), and parsing (Kiperwasser and Goldberg, 2016; Dyer et al., 2016).

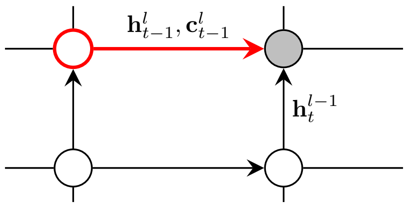
As reviewed in Ch. 2, gated RNNs such as long short-term memory (LSTM, Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRU, Cho et al., 2014b) are currently accepted as standard choices for RNNs due to the ease of training and their expressivity. Also, to further boost performance, the technique of stacking multiple RNN layers (Schmidhuber, 1992; El Hihi and Bengio, 1995), where the hidden states of a layer are fed as input to the next layer, is widely used.

However the typical setting of stacking RNNs might hinder the possibility of more sophisticated structures since the information from lower layers is simply treated as input to the next layer, rather than as another class of state that participates in core RNN computations. Especially for gated RNNs such as LSTMs and GRUs, this means that the vertical layer-to-layer connections cannot fully benefit from the carefully constructed gating mechanism used in temporal transitions.

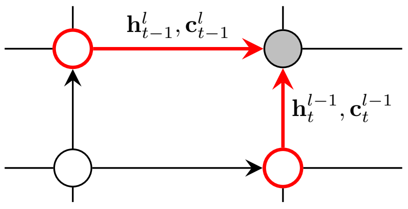
From this motivation, we study a method of constructing multi-layer LSTMs where memory cell states from the previous layer are used in controlling the vertical information flow. This architecture utilizes states from the left and the lower context equally in computation of the new state, thus the information from lower layers is elaborately filtered and reflected through a soft gating mechanism. The proposed architecture is easy-to-implement, effective, and can replace conventional stacked LSTMs without much modification of the overall architecture.

We call the proposed architecture Cell-aware Stacked LSTM, or CAS-LSTM, and evaluate it on multiple benchmark tasks: natural language inference, paraphrase identification, sentiment classification, and machine translation. From experiments we show that the CAS-LSTMs consistently outperform typical stacked LSTMs, opening the possibility of performance improvement of architectures based on stacked LSTMs. See Fig. 4.1 for the comparison between a plain stacked LSTM and the proposed CAS-LSTM architecture.

Our contribution is summarized as follows. Firstly, we bring the idea of utilizing states coming from multiple directions to construction of stacked LSTM and apply the idea to the research of sentence representation learning. There is some prior work addressing the idea of incorporating more than one type of state (Graves et al., 2007; Kalchbrenner et al., 2016; Zhang et al., 2016b), however to the best of our knowledge



(a) Plain stacked LSTM


$\mathbf{h}_{t-1}^{l}, \mathbf{c}_{t-1}^{l}$
$\mathbf{h}_{t}^{l-1}, \mathbf{c}_{t}^{l-1}$

(b) Cell-aware stacked LSTM

Figure 4.1: Visualization of a plain stacked LSTM and a CAS-LSTM architecture. The red nodes indicate the blocks whose cell states directly affect the cell state $\mathbf{c}_t^l$.

there is little work on applying the idea to modeling sentences for better understanding of natural language text.

Secondly, we conduct extensive evaluation of the proposed method and empirically prove its effectiveness. The CAS-LSTM architecture provides consistent performance gains over the stacked LSTM in all benchmark tasks: natural language inference, paraphrase identification, sentiment classification, and machine translation. Especially in SNLI, SST-2, and Quora Question Pairs datasets, our models outperform or at least are on par with the state-of-the-art models. We also conduct thorough qualitative analysis to understand the dynamics of the suggested approach.

4.2 Related Work

In this section, we summarize prior work related to the proposed method. We group the previous work that motivated our work into three classes: i) enhancing interaction between vertical layers, ii) RNN architectures that accepts latticed data, and iii) tree-structured RNNs.

Stacked Recurrent Neural Networks

There is some prior work on methods of stacking RNNs beyond the plain stacked RNNs (Schmidhuber, 1992; El Hihi and Bengio, 1995). Residual LSTMs (Kim et al., 2017a; Tran et al., 2017) add residual connections between the hidden states computed at each LSTM layer, and shortcut-stacked LSTMs (Nie and Bansal, 2017) concatenate hidden states from all previous layers to make the backpropagation path short. In our method, the lower context is aggregated via a gating mechanism, and we believe it modulates the amount of information to be transmitted in a more efficient and effective way than vector addition or concatenation. Also, compared to concatenation, our method does not significantly increase the number of parameters.[1]

Highway LSTMs (Zhang et al., 2016b) and depth-gated LSTMs (Yao et al., 2015) are similar to our proposed models in that they use cell states from the previous layer, and they are successfully applied to the field of automatic speech recognition and language modeling. However in contrast to CAS-LSTM, where the additional forget gate aggregates the previous layer states and thus contexts from the left and below participate in computation equitably, in Highway LSTMs and depth-gated LSTMs the states from the previous time step are not considered in computing vertical gates.

Multidimensional Recurrent Neural Networks

There is another line of research that aims to extend RNNs to operate with multidimensional inputs. Grid LSTMs (Kalchbrenner et al., 2016) are a general n-dimensional LSTM architecture that accepts n sets of hidden and cell states as input and yields n sets of states as output, in contrast to our architecture, which emits a single set of states. In their work, the authors utilize 2D and 3D Grid LSTMs in character-level language modeling and machine translation respectively and achieve performance im-

[1]The l-th layer of a typical stacked LSTM requires $(d_{l-1} + d_l + 1) \times 4d_l$ parameters, and the l-th layer of a shortcut-stacked LSTM requires $(\sum_{k=0}^{l-1} d_k + d_l + 1) \times 4d_l$ parameters. CAS-LSTM uses $(d_{l-1} + d_l + 1) \times 5d_l$ parameters at the l-th $(l > 1)$ layer.

provement. Multidimensional RNNs (Graves et al., 2007; Graves and Schmidhuber, 2008) have similar formulation to ours, except that they reflect cell states via simple summation and weights for all columns (vertical layers in our case) are tied. However they are only employed to model multidimensional data such as images of handwritten text with RNNs, rather than stacking RNN layers for modeling sequential data. From this view, CAS-LSTM could be interpreted as an extension of two-dimensional LSTM architecture that accepts a 2D input $\{\mathbf{h}_t^l\}_{t=1,l=0}^{T,L}$ where $\mathbf{h}_t^l$ represents the hidden state at time t and layer l.

Tree-structured Recurrent Neural Networks

The idea of having multiple states is also related to tree-structured RNNs (Goller and Kuchler, 1996; Socher et al., 2011a). Among them, tree-structured LSTMs (tree-LSTMs) (Tai et al., 2015; Zhu et al., 2015; Le and Zuidema, 2015) are similar to ours in that they use both hidden and cell states of children nodes. In tree-LSTMs, states of children nodes are regarded as input, and they participate in computing the states of a parent node equally through weight-shared or weight-unshared projection. From this perspective, each CAS-LSTM layer can be seen as a binary tree-LSTM where the structures it operates on are fixed to right-branching trees.

Indeed, our work is motivated by the recent analysis (Williams et al., 2018a; Shi et al., 2018) on latent tree learning models (Yogatama et al., 2017; Choi et al., 2018b) which has shown that tree-LSTM models outperform the sequential LSTM models even when the resulting parsing strategy generates strictly left- or right-branching parses, where a tree-LSTM model should read words in the manner identical to a sequential LSTM model. We argue that the active use of cell state in computation could be one reason of these counter-intuitive results and empirically prove the hypothesis in this work.

4.3 Model Description

In this section, we give the detailed formulation of architectures used in experiments.

4.3.1 Stacked LSTMs

While there exist various versions of LSTM formulation, in this work we use the following, the most common variant:

$$\mathbf{i}_t^l = \sigma(\mathbf{W}_i^l \mathbf{h}_t^{l-1} + \mathbf{U}_i^l \mathbf{h}_{t-1}^l + \mathbf{b}_i^l) \tag{4.1}$$

$$\mathbf{f}_t^l = \sigma(\mathbf{W}_f^l \mathbf{h}_t^{l-1} + \mathbf{U}_f^l \mathbf{h}_{t-1}^l + \mathbf{b}_f^l) \tag{4.2}$$

$$\tilde{\mathbf{c}}_t^l = \tanh(\mathbf{W}_c^l \mathbf{h}_t^{l-1} + \mathbf{U}_c^l \mathbf{h}_{t-1}^l + \mathbf{b}_c^l) \tag{4.3}$$

$$\mathbf{o}_t^l = \sigma(\mathbf{W}_o^l \mathbf{h}_t^{l-1} + \mathbf{U}_o^l \mathbf{h}_{t-1}^l + \mathbf{b}_o^l) \tag{4.4}$$

$$\mathbf{c}_t^l = \mathbf{i}_t^l \odot \tilde{\mathbf{c}}_t^l + \mathbf{f}_t^l \odot \mathbf{c}_{t-1}^l \tag{4.5}$$

$$\mathbf{h}_t^l = \mathbf{o}_t^l \odot \tanh(\mathbf{c}_t^l), \tag{4.6}$$

where $t \in \{1, \cdots, T\}$ and $l \in \{1, \cdots, L\}$. $\mathbf{W}_\cdot^l \in \mathbb{R}^{d_l \times d_{l-1}}$, $\mathbf{U}_\cdot^l \in \mathbb{R}^{d_l \times d_l}$, $\mathbf{b}_\cdot^l \in \mathbb{R}^{d_l}$ are trainable parameters, and $\sigma(\cdot)$ and $\tanh(\cdot)$ are the sigmoid and the hyperbolic tangent function respectively. Also we assume that $\mathbf{h}_t^0 = \mathbf{x}_t \in \mathbb{R}^{d_0}$ where $\mathbf{x}_t$ is the t-th element of an input sequence.

The input gate $\mathbf{i}_t^l$ and the forget gate $\mathbf{f}_t^l$ control the amount of information transmitted from $\tilde{\mathbf{c}}_t^l$ and $\mathbf{c}_{t-1}^l$, the candidate cell state and the previous cell state, to the new cell state $\mathbf{c}_t^l$. Similarly the output gate $\mathbf{o}_t^l$ soft-selects which portion of the cell state $\mathbf{c}_t^l$ is to be used in the final hidden state.

We can clearly see that the cell states $\mathbf{c}_{t-1}^l$, $\tilde{\mathbf{c}}_t^l$, $\mathbf{c}_t^l$ play a crucial role in forming horizontal recurrence. However the current formulation does not consider the cell state from $(l-1)$-th layer ($\mathbf{c}_t^{l-1}$) in computation and thus the lower context is reflected only through the rudimentary way, hindering the possibility of controlling vertical information flow.

4.3.2 Cell-aware Stacked LSTMs

Now we extend the stacked LSTM formulation defined above to address the problem noted in the previous subsection. To enhance the interaction between layers in a way similar to how LSTMs keep and forget the information from the previous time step, we introduce the *additional forget gate* $\mathbf{g}_t^l$ that determines whether to accept or ignore the signals coming from the previous layer.

The proposed Cell-aware Stacked LSTM (CAS-LSTM) architecture is defined as follows:

$$\mathbf{i}_t^l = \sigma(\mathbf{W}_i^l \mathbf{h}_t^{l-1} + \mathbf{U}_i^l \mathbf{h}_{t-1}^l + \mathbf{b}_i^l) \tag{4.7}$$

$$\mathbf{f}_t^l = \sigma(\mathbf{W}_f^l \mathbf{h}_t^{l-1} + \mathbf{U}_f^l \mathbf{h}_{t-1}^l + \mathbf{b}_f^l) \tag{4.8}$$

$$\mathbf{g}_t^l = \sigma(\mathbf{W}_g^l \mathbf{h}_t^{l-1} + \mathbf{U}_g^l \mathbf{h}_{t-1}^l + \mathbf{b}_g^l) \tag{4.9}$$

$$\tilde{\mathbf{c}}_t^l = \tanh(\mathbf{W}_c^l \mathbf{h}_t^{l-1} + \mathbf{U}_c^l \mathbf{h}_{t-1}^l + \mathbf{b}_c^l) \tag{4.10}$$

$$\mathbf{o}_t^l = \sigma(\mathbf{W}_o^l \mathbf{h}_t^{l-1} + \mathbf{U}_o^l \mathbf{h}_{t-1}^l + \mathbf{b}_o^l) \tag{4.11}$$

$$\mathbf{c}_t^l = \mathbf{i}_t^l \odot \tilde{\mathbf{c}}_t^l + (\mathbf{1} - \boldsymbol{\lambda}) \odot \mathbf{f}_t^l \odot \mathbf{c}_{t-1}^l + \boldsymbol{\lambda} \odot \mathbf{g}_t^l \odot \mathbf{c}_t^{l-1} \tag{4.12}$$

$$\mathbf{h}_t^l = \mathbf{o}_t^l \odot \tanh(\mathbf{c}_t^l), \tag{4.13}$$

where $l > 1$ and $d_l = d_{l-1}$. $\boldsymbol{\lambda}$ can either be a vector of constants or parameters. When $l = 1$, the equations defined in the previous subsection are used. Therefore, it can be said that each non-bottom layer of CAS-LSTM accepts two sets of hidden and cell states—one from the left context and the other from the below context. The left and the below context participate in computation with the equivalent procedure so that the information from lower layers can be efficiently propagated. Fig. 4.1 compares CAS-LSTM to the conventional stacked LSTM architecture, and Fig. 4.2 depicts the computation flow of the CAS-LSTM.

We argue that considering $\mathbf{c}_t^{l-1}$ in computation is beneficial for the following reasons. First, contrary to $\mathbf{h}_t^{l-1}$, $\mathbf{c}_t^{l-1}$ contains information which is not filtered by

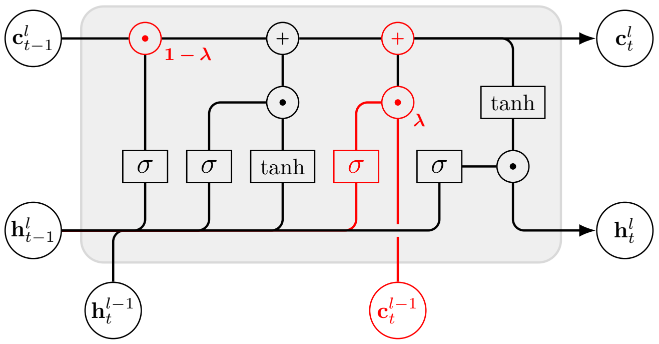


Figure 4.2: Schematic diagram of a CAS-LSTM block.

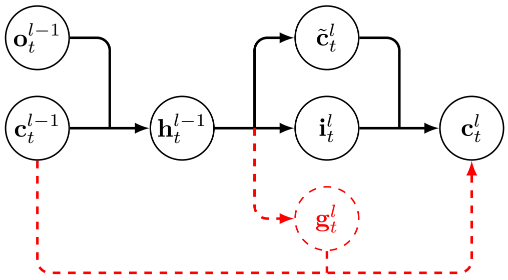


Figure 4.3: Visualization of paths between $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$. In CAS-LSTM, the direct connection between $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$ exists (denoted as red dashed lines).

$\mathbf{o}_t^{l-1}$. Thus a model that directly uses $\mathbf{c}_t^{l-1}$ does not rely solely on $\mathbf{o}_t^{l-1}$ for extracting information, due to the fact that it has access to the raw information $\mathbf{c}_t^{l-1}$, as in temporal connections. In other words, $\mathbf{o}_t^{l-1}$ no longer has to take all responsibility for selecting useful features for both horizontal and vertical transitions, and the burden of selecting information is shared with $\mathbf{g}_t^l$.

Another advantage of using the $\mathbf{c}_t^{l-1}$ lies in the fact that it directly connects $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$. This direct connection could help and stabilize training, since the terminal error signals can be easily backpropagated to the model parameters by the shortened propagation path. Fig. 4.3 illustrates paths between the two cell states.

Regarding $\boldsymbol{\lambda}$, we find experimentally that there is little difference between having it be a constant and a trainable vector bounded in $(0, 1)$, and we practically find that setting $\lambda_i = 0.5$ works well across multiple experiments. We also experimented with the architecture without $\boldsymbol{\lambda}$ i.e. two cell states are combined by unweighted summation similar to multidimensional RNNs (Graves and Schmidhuber, 2008), and found that it leads to performance degradation and unstable convergence, likely due to mismatch in the range of cell state values between layers ($(-2, 2)$ for the first layer and $(-3, 3)$ for the others). Experimental results on various $\boldsymbol{\lambda}$ are presented in §4.4.6.

4.3.3 Sentence Encoders

For text classification tasks, a variable-length sentence should be represented as a fixed-length vector. We describe the sentence encoder architectures used in experiments in this subsection.

First, we assume that a sequence of T one-hot word vectors is given as input: $(\mathbf{w}_1, \cdots, \mathbf{w}_T)$, $\mathbf{w}_t \in \mathbb{R}^{|V|}$ where V is the vocabulary set. The words are projected to corresponding word representations: $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ where $\mathbf{x}_t = \mathbf{E}^\top \mathbf{w}_t \in \mathbb{R}^{d_0}$, $\mathbf{E} \in \mathbf{R}^{|V| \times d_0}$. Then $\mathbf{X}$ is fed to a L-layer CAS-LSTM model, resulting in the representations $\mathbf{H} = (\mathbf{h}_1^L, \cdots, \mathbf{h}_T^L) \in \mathbb{R}^{T \times d_L}$. The encoded sentence representation $\mathbf{s} \in \mathbb{R}^{d_L}$ is computed by max-pooling $\mathbf{H}$ over time as in the work of Conneau et al. (2017). Similar to their results, from preliminary experiments we found that the max-pooling performs consistently better than the mean-pooling and the last-pooling.

For better modeling of semantics, a bidirectional CAS-LSTM network may also be used. In the bidirectional case, the representations obtained by left-to-right reading $\mathbf{H} = (\mathbf{h}_1^L, \cdots, \mathbf{h}_T^L) \in \mathbb{R}^{T \times d_L}$ and those by right-to-left reading $\widehat{\mathbf{H}} = (\widehat{\mathbf{h}}_1^L, \cdots, \widehat{\mathbf{h}}_T^L) \in \mathbb{R}^{T \times d_L}$ are concatenated and max-pooled to yield the sentence representation $\mathbf{s} \in \mathbb{R}^{2d_L}$. We call this bidirectional architecture Bi-CAS-LSTM in experiments.

To predict the final task-specific label, we apply a task-specific feature extraction function ϕ to the sentence representation(s) and feed the extracted features to a classifier network. For the classifier network, a multi-layer perceptron (MLP) with the ReLU activation followed by the linear projection and the softmax function is used:

$$P(\mathbf{y}|\mathbf{X}) = \mathrm{softmax}(\mathbf{W}_c\mathrm{MLP}(\phi(\cdot))), \tag{4.14}$$

where $\mathbf{W}_c \in \mathbb{R}^{|L|\times d_h}$, $|L|$ is the number of label classes, and d_h the dimension of the MLP output.

4.4 Experiments

We evaluate our method on three benchmark tasks on sentence encoding: natural language inference (NLI), paraphrase identification (PI), and sentiment classification. To further demonstrate the general applicability of our method on text generation, we also evaluate the proposed method on machine translation. In addition, we conduct analysis on gate values model variations for the understanding of the architecture.

For the NLI and PI tasks, there exists architectures specializing in sentence pair classification. However in this work we confine our model to the architecture that encodes each sentence using a shared encoder without any inter-sentence interaction, in order to focus on the effectiveness of the architectures in extracting semantics. But note that the applicability of CAS-LSTM is not limited to sentence encoder–based approaches.

For all experiments, weight matrices for recurrent connections are initialized according to the orthogonal initialization scheme (Saxe et al., 2014). All other weight matrices are initialized using the scheme proposed by He et al. (2015), except the weights for the last fully-connect layer which are initialized by sampling from the uniform distribution $\mathcal{U}(-0.005, 0.005)$. Bias vectors are initialized to zero.

4.4.1 Natural Language Inference

For the evaluation of performance of the proposed method on the NLI task, SNLI (Bowman et al., 2015) and MultiNLI (Williams et al., 2018b) datasets are used.

The objective of both datasets is to predict the relationship between a premise and a hypothesis sentence: *entailment*, *contradiction*, and *neutral*. SNLI and MultiNLI datasets are composed of about 570k and 430k premise-hypothesis pairs respectively.

GloVe pretrained word embeddings[2] (Pennington et al., 2014) are used and remain fixed during training. The dimension of encoder states (d_l) is set to 300 and a 1024D MLP with one or two hidden layers is used. Adam optimizer (Kingma and Ba, 2015) is used for training, and the learning rate is annealed according to cosine schedule (Loshchilov and Hutter, 2017) with the initial learning rate of 0.001. For all models, we added the L_2 norm of the parameters to the classification loss with the factor of 0.002. The dimensions of encoder states and MLP hidden layers are set to 300 and 1024 respectively. Batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) is applied to the word embeddings and the MLP layers. Dropout is also applied to word embeddings, and we denote the drop probability of word embeddings by p_w and that of MLP input and layer outputs by p_c. The maximum length of each sentence is 35 for SNLI and 55 for MultiNLI experiments, and words beyond the sentence boundary are discarded. Each minibatch is composed of 128 data samples. Table 4.1 and 4.2 list hyperparameters used in the SNLI and MultiNLI experiments.

The features used as input to the MLP classifier are extracted by the following equation:

$$\phi(\mathbf{s}_1, \mathbf{s}_2) = \mathbf{s}_1 \oplus \mathbf{s}_2 \oplus |\mathbf{s}_1 - \mathbf{s}_2| \oplus (\mathbf{s}_1 \odot \mathbf{s}_2), \tag{4.15}$$

where $\oplus$ is the vector concatenation operator.

Table 4.3 and 4.4 contain results of the models on SNLI and MultiNLI datasets. Along with other state-of-the-art models, the tables include several stacked LSTM–based models to facilitate comparison of our work with prior related work. Liu et al. (2016); Chen et al. (2017b, 2018a) adopt advanced pooling algorithms motivated by

[2] `https://nlp.stanford.edu/projects/glove/`

Model	**# Encoder Layers**	**Bidirectional**	**# MLP Layers**	$\boldsymbol{p_w}$	$\boldsymbol{p_c}$
2-layer CAS-LSTM	2		1	0.10	0.10
2-layer Bi-CAS-LSTM	2	✓	2	0.15	0.15
3-layer CAS-LSTM	3		2	0.15	0.20
3-layer Bi-CAS-LSTM	3	✓	2	0.15	0.15

Table 4.1: Hyperparameters for SNLI models.

Model	**# Encoder Layers**	**Bidirectional**	**# MLP Layers**	$\boldsymbol{p_w}$	$\boldsymbol{p_c}$	$\boldsymbol{L_2}$ **weight**
2-layer CAS-LSTM	2		1	0.10	0.10	0.0025
2-layer Bi-CAS-LSTM	2	✓	2	0.15	0.20	0.0020
3-layer CAS-LSTM	3		2	0.10	0.15	0.0025
3-layer Bi-CAS-LSTM	3	✓	2	0.15	0.20	0.0020

Table 4.2: Hyperparameters for MultiNLI models.

Model	Acc. (%)	# Params
300D LSTM (Bowman et al., 2016a)	80.6	3.0M
300D TBCNN (Mou et al., 2016)	82.1	3.5M
300D SPINN-PI (Bowman et al., 2016a)	83.2	3.7M
600D BiLSTM + intra-attention (Liu et al., 2016)	84.2	2.8M
4096D BiLSTM + max-pooling (Conneau et al., 2017)	84.5	40M
300D BiLSTM + gated pooling (Chen et al., 2017b)	85.5	12M
300D Gumbel Tree-LSTM (Choi et al., 2018b)	85.6	2.9M
600D Shortcut stacked BiLSTM (Nie and Bansal, 2017)	86.1	140M
300D Reinforced self-attention network (Shen et al., 2018c)	86.3	3.1M
600D BiLSTM + generalized pooling (Chen et al., 2018a)	86.6	65M
300D 2-layer CAS-LSTM (ours)	86.4	2.9M
300D 2-layer Bi-CAS-LSTM (ours)	86.8	6.8M
300D 3-layer CAS-LSTM (ours)	86.4	4.8M
300D 3-layer Bi-CAS-LSTM (ours)	**87.0**	8.6M

Table 4.3: Results of the models on the SNLI dataset.

the attention mechanism to obtain a fixed-length sentence vector. Nie and Bansal (2017) use the concatenation of all outputs from previous layers as input to the next layer.

In SNLI, our best model achieves the accuracy of 87.0%, which is the new state-of-the-art among the sentence encoder–based models, with relatively fewer parameters. Similarly in MultiNLI, our models match the accuracy of state-of-the-art models in both in-domain (matched) and cross-domain (mismatched) test sets. Note that only the GloVe word vectors are used as word representations, as opposed to some models that introduce character-level features. It is also notable that our proposed architecture does not restrict the selection of pooling method; the performance could further be improved by replacing max-pooling with other advanced algorithms e.g. intra-sentence attention (Liu et al., 2016) and generalized pooling (Chen et al., 2018a).

4.4.2 Paraphrase Identification

We use Quora Question Pairs dataset (Wang et al., 2017b) in evaluating the performance of our method on the PI task. The dataset consists of over 400k question pairs, and each pair is annotated with whether the two sentences are paraphrase of each

Model	**In (%)**	**Cross (%)**	**# Params**
CBOW (Williams et al., 2018b)	64.8	64.5	-
BiLSTM (Williams et al., 2018b)	66.9	66.9	-
Shortcut stacked BiLSTM (Nie and Bansal, 2017)*	**74.6**	73.6	140M
BiLSTM + gated pooling (Chen et al., 2017b)	73.5	73.6	12M
BiLSTM + generalized pooling (Chen et al., 2018a)	73.8	**74.0**	18M**
2-layer CAS-LSTM (ours)	74.0	73.3	2.9M
2-layer Bi-CAS-LSTM (ours)	**74.6**	73.7	6.8M
3-layer CAS-LSTM (ours)	73.8	73.1	4.8M
3-layer Bi-CAS-LSTM (ours)	74.2	73.4	8.6M

Table 4.4: Results of the models on the MultiNLI dataset. 'In' and 'Cross' represent accuracy calculated from the matched and mismatched test set respectively. *: SNLI dataset is used as additional training data. **: computed from hyperparameters provided by the authors.

Model	**# Encoder Layers**	**Bidirectional**	**# MLP Layers**	p_w	p_c
CAS-LSTM	2		1	0.10	0.10
Bi-CAS-LSTM	2	✓	1	0.15	0.20

Table 4.5: Hyperparameters for Quora Question Pairs models.

other or not.

Similarly to the NLI experiments, GloVe pretrained vectors, 300D encoders, and 1024D MLP are used. The number of CAS-LSTM layers is fixed to 2 in PI experiments, and all models use the L_2 weight of 0.002. Two sentence vectors are aggregated using the following equation and fed as input to the classifier.

$$\phi(\mathbf{s}_1, \mathbf{s}_2) = |\mathbf{s}_1 - \mathbf{s}_2| \oplus (\mathbf{s}_1 \odot \mathbf{s}_2) \tag{4.16}$$

The hyperparameters used are lised in Table 4.5.

The results on the Quora Question Pairs dataset are summarized in Table 4.6. Again we can see that our models outperform other models, especially compared to conventional LSTM–based models. Also note that Multi-Perspective LSTM (Wang et al., 2017b), LSTM + ElBiS (Choi et al., 2018a), and REGMAPR (BASE+REG) (Brahma, 2018) in Table 4.6 are approaches that focus on designing a more sophisticated function for aggregating two sentence vectors, and their aggregation functions

Model	**Acc. (%)**
CNN (Wang et al., 2017b)	79.6
LSTM (Wang et al., 2017b)	82.6
Multi-Perspective LSTM (Wang et al., 2017b)	83.2
LSTM + ElBiS (Choi et al., 2018a)	87.3
REGMAPR (BASE+REG) (Brahma, 2018)	88.0
CAS-LSTM (ours)	88.4
Bi-CAS-LSTM (ours)	**88.6**

Table 4.6: Results of the models on the Quora Question Pairs dataset.

could be also applied to our work for further improvement.

4.4.3 Sentiment Classification

In evaluating sentiment classification performance, the Stanford Sentiment Treebank (SST) (Socher et al., 2013) is used. It consists of about 12,000 binary-parsed sentences where constituents (phrases) of each parse tree are annotated with a sentiment label (*very positive*, *positive*, *neutral*, *negative*, *very negative*). Following the convention of prior work, all phrases and their labels are used in training but only the sentence-level data are used in evaluation.

In evaluation we consider two settings, namely SST-2 and SST-5, the two differing only in their level of granularity with regard to labels. In SST-2, data samples annotated with 'neutral' are ignored from training and evaluation. The two positive labels (very positive, positive) are considered as the same label, and similarly for the two negative labels. As a result 98,794/872/1,821 data samples are used in training/validation/test, and the task is considered as a binary classification problem. In SST-5, all 318,582/1,101/2,210 data samples are used and the task is a 5-class classification problem.

Since the task is a single-sentence classification problem, we use the sentence representation itself as input to the classifier. We use 300D GloVe vectors, 2-layer 150D or 300D encoders, and a 300D MLP classifier for the models, however unlike previous experiments we tune the word embeddings during training. Models are trained using ADADELTA algorithm (Zeiler, 2012) instead of Adam. Table 4.7 and 4.8 contain the hyperparameter configurations used for SST-2 and SST-5 experiments.

The results on SST are listed in Table 4.9. Our models clearly outperform plain LSTM- and BiLSTM-based models, and are competitive to other state-of-the-art models, without utilizing parse tree information.

4.4.4 Machine Translation

We use the IWSLT 2014 machine evaluation campaign dataset (Cettolo et al., 2014) in machine translation experiments. We used the fairseq library (Gehring et al., 2017) for experiments. Moses tokenizer[3] is used for word tokenization and the byte pair encoding (Sennrich et al., 2016) is applied to confine the size of the vocabulary set up to 10,000. The `lstm_wiseman_iwslt_de_en` configuration is used as the base architecture, and we implemented the CAS-LSTM counterpart. We set the number of LSTM layers to 2 and selected the dropout probability p from $\{0.1, 0.2, 0.3\}$, and set $p = 0.1$ for the base architecture and $p = 0.2$ for the CAS-LSTM architecture.

Similar to Wiseman and Rush (2016), a 2-layer 256D sequence-to-sequence LSTM model with the attentional decoder is used as baseline, and we replace the encoder and the decoder network with the proposed architecture for the evaluation of performance improvement. For decoding, beam search with $B = 10$ is used. For fair comparison, we tune hyperparameters for all models based on the performance on the validation dataset and train the same model for five times with different random seeds. Also, to cancel out the increased number of parameters, we experiment with the 247D CAS-LSTM model which has the roughly same number of parameters as the baseline model (8.2M).

[3]`https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl`

Model	**# Encoder Layers**	**Bidir.**	**Encoder Dim.**	**MLP Hidden Dim.**	p_w	p_c	L_2 **weight**
CAS-LSTM	2		300	300	0.5	0.5	0.010
Bi-CAS-LSTM	2	✓	150	300	0.5	0.5	0.005

Table 4.7: Hyperparameters for SST-2 models.

Model	**# Encoder Layers**	**Bidir.**	**Encoder Dim.**	**MLP Hidden Dim.**	p_w	p_c	L_2 **weight**
CAS-LSTM	2		300	300	0.4	0.4	0.005
Bi-CAS-LSTM	2	✓	300	300	0.5	0.5	0.005

Table 4.8: Hyperparameters for SST-5 models.

Model	SST-2 (%)	SST-5 (%)
Recursive Neural Tensor Network (Socher et al., 2013)	85.4	45.7
2-layer LSTM (Tai et al., 2015)	86.3	46.0
2-layer BiLSTM (Tai et al., 2015)	87.2	48.5
Constituency Tree-LSTM (Tai et al., 2015)	88.0	51.0
Constituency Tree-LSTM + Recurrent Dropout (Looks et al., 2017)	89.4	52.3
byte mLSTM (Radford et al., 2017)*	<u>91.8</u>	52.9
Gumbel Tree-LSTM (Choi et al., 2018b)	90.7	**53.7**
BCN + Char + ELMo (Peters et al., 2018)*	-	<u>54.7</u>
2-layer CAS-LSTM (ours)	91.1	53.0
2-layer Bi-CAS-LSTM (ours)	**91.3**	53.6

Table 4.9: Results of the models on the SST dataset. *: models pretrained on large external corpora are used.

Model	BLEU
256D LSTM	28.1 ± 0.22
256D CAS-LSTM	$28.8 \pm 0.04^*$
247D CAS-LSTM	$28.7 \pm 0.07^*$

Table 4.10: Results of the models on the IWSLT 2014 de-en dataset. *: $p < 0.0005$ (one-tailed paired t-test).

From Table 4.10, we can see that the CAS-LSTM models bring significant performance gains over the baseline model.

4.4.5 Forget Gate Analysis

To inspect the effect of the additional forget gate, we investigate how the values of vertical forget gates are distributed. We sample 1,000 random sentences from the development set of the SNLI dataset, and use the 3-layer CAS-LSTM model trained on the SNLI dataset to compute gate values.

If all values from a vertical forget gate $\mathbf{g}_t^l$ were to be 0, this would mean that the introduction of the additional forget gate is meaningless and the model would reduce to a plain stacked LSTM. On the contrary if all values were 1, meaning that the vertical forget gates were always *open*, it would be impossible to say that the information is modulated effectively.

Fig. 4.4a and 4.4b represent histograms of the vertical forget gate values from the second and the third layer. From the figures we can validate that the trained model does not fall into the degenerate case where vertical forget gates are ignored. Also the figures show that the values are right-skewed, which we conjecture to be a result of focusing more on a strong interaction between adjacent layers.

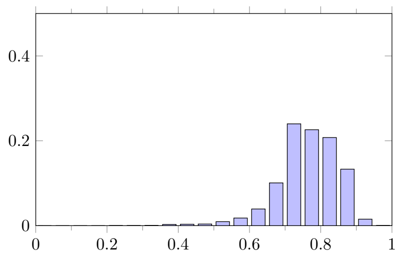
To further verify that the gate values are diverse enough within each time step, we compute the distribution of the range of values per time step, $R(\mathbf{g}_t^l) = \max_i g_{t,i}^l - \min_i g_{t,i}^l$, where $\mathbf{g}_t^l = [g_{t,1}^l, \cdots, g_{t,d_l}^l]^\top$. We plot the histograms in Fig. 4.4c and 4.4d. From the figures we see that the vertical forget gate controls the amount of information flow effectively, making diverse decisions of retaining or discarding signals across dimensions.

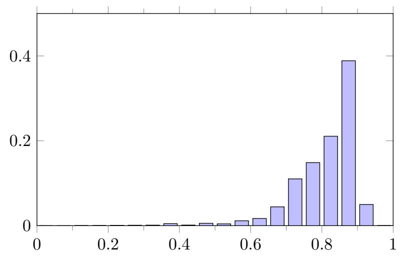
Finally, to investigate the argument presented in §4.3 that the additional forget gate helps the previous output gate with reducing the burden of extracting all needed information, we inspect the distribution of the values from $|\mathbf{g}_t^l - \mathbf{o}_t^{l-1}|$. This distribution indicates how differently the vertical forget gate and the previous output gate select information from $\mathbf{c}_t^{l-1}$. From Fig. 4.4e and 4.4f we can see that the two gates make fairly different decisions, from which we demonstrate that the direct path between $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$ enables a model to utilize signals overlooked by $\mathbf{o}_t^{l-1}$.

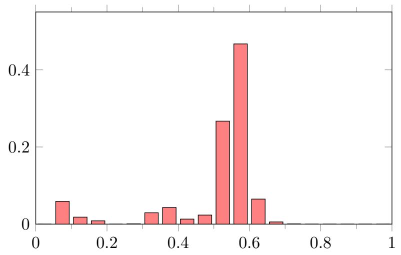
4.4.6 Model Variations

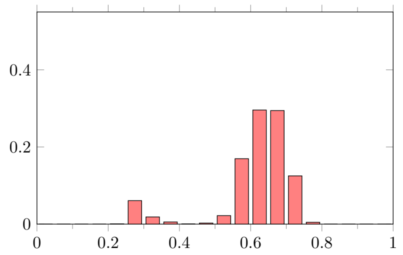
In this subsection, we see the influence of each component of a model on performance by removing or replacing its components. the SNLI dataset is used for experiments, and the best performing configuration is used as a baseline for modifications. We consider the following variants: (*i*) models with different $\boldsymbol{\lambda}$, (*ii*) models without $\boldsymbol{\lambda}$, and (*iii*) models that integrate lower contexts via peephole connections.

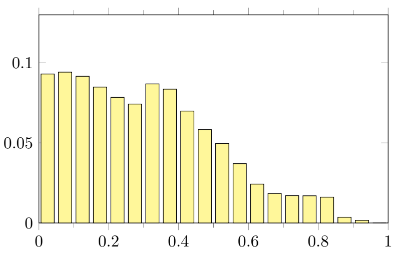
Variant (*iii*) calculates and applies the forget gate $\mathbf{g}_t^l$ which takes charge of integrating lower contexts via the equations below, following the work of Zhang et al.


(a) g_i^2


(b) g_i^3


(c) $R(\mathbf{g}_\cdot^2)$


(d) $R(\mathbf{g}_\cdot^3)$


(e) $|g_i^2 - o_i^1|$

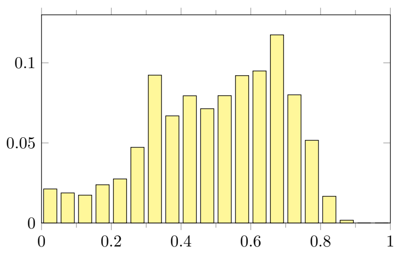

(f) $|g_i^3 - o_i^2|$

Figure 4.4: Results of forget gate analysis. (a), (b): Histograms of vertical forget gate values. (c), (d): Histograms of the ranges of vertical forget gate per time step. (e), (f): Histograms of the absolute difference between the previous output gate and the current vertical forget gate values.

Model	**Acc. (%)**	Δ
Bi-CAS-LSTM (*baseline*)	87.0	
(i) Diverse $\boldsymbol{\lambda}$		
(a) $\lambda_i = 0.25$	86.8	-0.2
(b) $\lambda_i = 0.75$	86.8	-0.2
(c) Trainable $\boldsymbol{\lambda}$	86.9	-0.1
(ii) No $\boldsymbol{\lambda}$	86.6	-0.4
(iii) Integration through peepholes	86.5	-0.5

Table 4.11: Results of model variants.

(2016b):

$$\mathbf{g}_t^l = \sigma(\mathbf{W}_g^l \mathbf{h}_t^{l-1} + \mathbf{p}_{g_1}^l \odot \mathbf{c}_{t-1}^l + \mathbf{p}_{g_2}^l \odot \mathbf{c}_t^{l-1} + \mathbf{b}_g^l) \tag{4.17}$$

$$\mathbf{c}_t^l = \mathbf{i}_t^l \odot \tilde{\mathbf{c}}_t^l + \mathbf{f}_t^l \odot \mathbf{c}_{t-1}^l + \mathbf{g}_t^l \odot \mathbf{c}_t^{l-1}, \tag{4.18}$$

where $\mathbf{p}_{\cdot}^l \in \mathbb{R}^{d_l}$ represent peephole weight vectors that take cell states into account. We can see that the computation formulae of $\mathbf{f}_t^l$ and $\mathbf{g}_t^l$ are not consistent, in that $\mathbf{h}_{t-1}^l$ does not participate in computing $\mathbf{g}_{t-1}^l$, and that the left and the below context are reflected in $\mathbf{g}_{t-1}^l$ only via element-wise multiplications which do not consider the interaction among dimensions. By contrast, ours uses the analogous formulae in calculating $\mathbf{f}_t^l$ and $\mathbf{g}_t^l$, considers $\mathbf{h}_{t-1}^l$ in calculating $\mathbf{g}_t^l$, and introduces the scaling factor $\boldsymbol{\lambda}$.

Table 4.11 summarizes the results of model variants. From the results of *baseline* and *(i)*, we validate that the selection of $\boldsymbol{\lambda}$ does not significantly affect performance but introducing $\boldsymbol{\lambda}$ is beneficial (*baseline vs. (ii)*) possibly due to its effect on normalizing information from multiple sources, as mentioned in §4.3. Also, from the comparison between *baseline* and *(iii)*, we show that the proposed way of combining the left and the lower contexts leads to better modeling of sentence representations than that of Zhang et al. (2016b).

4.5 Summary

We proposed a method of stacking multiple LSTM layers for modeling sentences, dubbed CAS-LSTM. It uses not only hidden states but also cell states from the previous layer, for the purpose of controlling the vertical information flow in a more elaborate way. We evaluated the proposed method on various benchmark tasks: natural language inference, paraphrase identification, and sentiment classification. Our models outperformed plain LSTM-based models in all experiments and were competitive other state-of-the-art models. The proposed architecture can replace any stacked LSTM only under one weak restriction—the size of states should be identical across all layers.

Chapter 5

Matching Function: Element-wise Bilinear Sentence Matching

5.1 Motivation

When we build a neural network model predicting the relationship between two sentences, the most general and intuitive approach is to use a siamese architecture, where sentence vectors obtained from a shared encoder is given as input to a classifier network. For a model to predict the relationship correctly, along with obtaining appropriate sentence vectors, forming an input that contains information useful for predicting the relationship by comparing the two sentence vectors is also of great importance, since the classifier should infer the relationship from the given aggregated input.

The most naïve method is to simply concatenate the two vectors and delegate the role of extracting features to subsequent network components. However, despite the theoretical fact that even a single-hidden layer feedforward network can approximate any arbitrary functions (Cybenko, 1989; Hornik, 1991), the space of network parameters is too broad, and it is always helpful to narrow down the search space by directly giving information about interaction to the classifier network, as empirically proven in a plethora of previous works (Ji and Eisenstein, 2013; Mou et al., 2016).

As an answer to this problem, we propose a matching function which learns from data to fuse two sentence vectors and extract suitable features. Unlike bilinear pooling methods designed for matching vectors from heterogeneous domain (e.g. image and text), our proposed method focuses on element-wise bilinear interaction between vectors rather than inter-dimensional interaction.

In Ch. 2, we reviewed some prior work on fusing multiple vectors using a tensor multiplication or bilinear pooling, however to the best of our knowledge there exists little work on a method that adaptively learns to extract features from two sentence vectors encoded by a shared encoder.

5.2 Proposed Method: *ElBiS*

As pointed out by previous works on sentence matching (Ji and Eisenstein, 2013; Mou et al., 2016), heuristic matching functions bring substantial gain in performance over the simple concatenation of sentence vectors. However, we believe that there could be other important interaction that simple heuristics miss, and the optimal heuristic could differ from task to task. In this section, we propose a general matching function that learns to extract compact and effective element-wise features from data.

Let $\mathbf{a} = (a_1, \cdots, a_d) \in \mathbb{R}^d$ and $\mathbf{b} = (b_1, \cdots, b_d) \in \mathbb{R}^d$ be sentence vectors obtained from a encoder network.[1] And let us define $\mathbf{G} \in \mathbb{R}^{d \times 3}$ as a matrix constructed by stacking three vectors $\mathbf{a}, \mathbf{b}, \vec{\mathbf{1}} \in \mathbb{R}^d$ where $\vec{\mathbf{1}}$ is the vector of all ones, and denote the i-th row of $\mathbf{G}$ by $\mathbf{g}_i$.

[1]Throughout this chapter, we assume a d-dimensional vector is equivalent to the corresponding $d \times 1$ matrix.

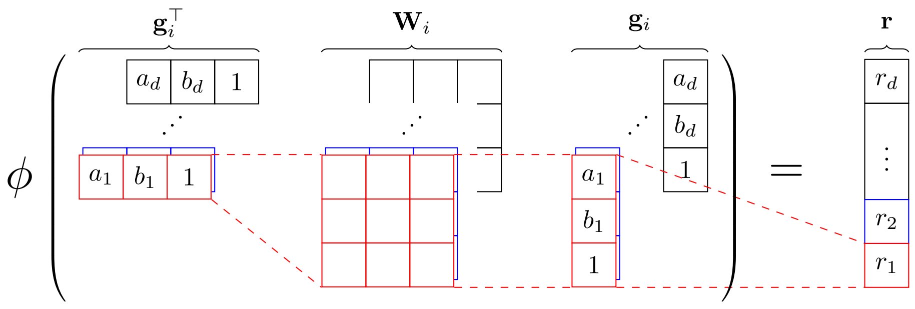


Figure 5.1: Illustration of the ElBiS matching function.

Then the result of applying our proposed matching function, $\mathbf{r} = (r_1, \cdots, r_d) \in \mathbb{R}^d$, is defined by

$$r_i = \phi\left(\mathbf{g}_i^\top \mathbf{W}_i \mathbf{g}_i\right), \tag{5.1}$$

where $\mathbf{W}_i \in \mathbb{R}^{3\times3}, i \in \{1, \cdots, d\}$ is a matrix of trainable parameters and $\phi(\cdot)$ an activation function (tanh in our experiments).

Due to the use of bilinear form, it can model every quadratic relation between a_i and b_i, i.e. can represent every linear combination of $\{a_i^2, b_i^2, a_ib_i, a_i, b_i, 1\}$. This means that the proposed method is able to express frequently used element-wise heuristics such as element-wise sum, multiplication, subtraction, etc., in addition to other possible relations.[2] Fig. 5.1 depicts the computation of the ElBiS matching function.

The current formulation can only represent a single quadratic relation per dimension, so to consider multiple types of element-wise interaction, we can repeat the same process for M times. That is, for each $\mathbf{g}_i$, we get M scalar outputs $(r_i^1, \cdots, r_i^M)$ by applying Eq. 5.1 using a set of separate weight matrices $(\mathbf{W}_i^1, \cdots, \mathbf{W}_i^M)$:

$$r_i^m = \phi\left(\mathbf{g}_i^\top \mathbf{W}_i^m \mathbf{g}_i\right). \tag{5.2}$$

[2]Though a bilinear form cannot represent the absolute difference between inputs, note that $(a_i - b_i)^2 = a_i^2 - 2a_ib_i + b_i^2$ can alternatively used as a commutative difference function. Yogatama et al. (2017) use this quadratic form instead of the absolute difference.

Implementation-wise, we vertically stack G for M times to construct $\tilde{\mathbf{G}} \in \mathbb{R}^{Md \times 3}$, and use each row $\tilde{\mathbf{g}}_i$ as input to Eq. 5.1. Consequently, the resulting output $\mathbf{r}$ becomes a Md-dimensional vector:

$$r_i = \phi \left(\tilde{\mathbf{g}}_i^\top \mathbf{W}_i \tilde{\mathbf{g}}_i \right), \tag{5.3}$$

where $\mathbf{W}_i \in \mathbb{R}^{3 \times 3}, i \in \{1, \cdots, Md\}$. Eq. 5.1 is the special case of Eq. 5.2 and 5.3 where $M = 1$. We call our proposed element-wise bilinear matching function *ElBiS* (element-wise bilinear sentence matching function).

Note that our element-wise matching requires only $M \times 3 \times 3 \times d$ parameters, the number of which is substantially less than that of full bilinear matching, Md^3. For example, in the case of $d = 300$ and $Md = 1200$ (the frequently used set of hyperparameters in NLI), the full bilinear matching needs 108 million parameters, while the element-wise matching needs only 10,800 parameters.

Why element-wise? In the scenario we are focusing on, sentence vectors are computed from a siamese network, and thus it can be said that the vectors are in the same (or very similar) semantic space. Therefore, the effect of considering interdimensional interaction is less significant than that of multimodal pooling (e.g. matching a text and a image vector obtained from different encoders), so we decided to model more powerful interaction within the same dimension instead. We also would like to remark that our preliminary experiments, where MFB (Yu et al., 2017) or MLB (Kim et al., 2017b) was adopted as matching function, were not successful and did not improve performance.

5.3 Experiments

We evalute our proposed ElBiS model on the natural language inference and paraphrase identification task.

5.3.1 Natural language inference

Natural language inference (NLI, Bowman et al., 2015), also called recognizing textual entailment (RTE, Dagan et al., 2005), is a task whose objective is to predict the relationship between a premise and a hypothesis sentence. We conduct experiments using Stanford Natural Language Inference Corpus (SNLI, Bowman et al., 2015), one of the most famous dataset for the NLI task. The SNLI dataset consists of roughly 570k premise-hypothesis pairs, each of which is annotated with a label (entailment, contradiction, or neutral).

For sentence encoder, we choose the encoder based on long short-term memory (LSTM, Hochreiter and Schmidhuber, 1997) architecture as a baseline model, which is similar to that of Bowman et al. (2015) and Bowman et al. (2016a). It consists of a single layer unidirectional LSTM network that reads a sentence from left to right, and the last hidden state is used as the sentence vector. We also conduct experiments using a more elaborated encoder model, Gumbel Tree-LSTM (Choi et al., 2018b). As a classifier network, we use an MLP with a single hidden layer. In baseline experiments with heuristic matching, we use the heuristic features proposed by Mou et al. (2016) and adopted in many works on the NLI task:

$$\mathbf{r} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{a} - \mathbf{b} \\ \mathbf{a} \odot \mathbf{b} \end{bmatrix}, \tag{5.4}$$

where **a** and **b** are encoded sentence vectors.

For all experiments, we used the Adam (Kingma and Ba, 2015) optimizer with a learning rate 0.001 and halved the learning rate when there is no improvement in accuracy for one epoch. Each model is trained for 10 epochs, and the checkpoint with the highest validation accuracy is chosen as the final model. Sentences longer than 25 words are trimmed to have the maximum length of 25 words, and batch size of 64 is used for training.

We set the dimensionality of sentence vectors to 300. 300-dimensional GloVe (Pennington et al., 2014) vectors trained on 840 billion tokens[3] were used as word embeddings and not updated during training. The number of hidden units of the single-hidden layer MLP is set to 1024.

Dropout (Srivastava et al., 2014) is applied to word embeddings and the input and the output of the MLP. The dropout probability is selected from $\{0.10, 0.15, 0.20\}$. Batch normalization (Ioffe and Szegedy, 2015) is applied to the input and the output of the MLP.

Recurrent weight matrices are orthogonally initialized (Saxe et al., 2014), and the final linear projection matrix is initialized by sampling from the uniform distribution $\mathcal{U}(-0.005, 0.005)$. All other weights are initialized following the scheme of He et al. (2015).

Table 5.1 and 5.2 contain results on the SNLI task. We can see that models that adopt the proposed ElBiS matching function extract powerful features leading to a performance gain, while keeping similar or less number of parameters. Also, though not directly related to our main contribution, we found that, with elaborated initialization and regularization, simple LSTM models (even the one with the heuristic matching function) achieve competitive performance with those of state-of-the-art models.[4]

It is also notable that increasing M, the number of repetition of ElBiS algorithm, does not always improve performance. We conjecture that this occurs due to over-parameterization, by which a model learns characteristics or patterns that do not related to the nature of the NLI problem but only appear in a training data. We

[3]`http://nlp.stanford.edu/data/glove.840B.300d.zip`
[4]`https://nlp.stanford.edu/projects/snli`

Matching Fn.	**# Params.**	**Acc. (%)**
Concat	1.34M	81.6
Heuristic	1.96M	83.9
ElBiS ($M = 1$)	1.04M	84.4
ElBiS ($M = 2$)	1.35M	84.5
ElBiS ($M = 3$)	1.66M	**85.0**
ElBiS ($M = 4$)	1.97M	84.6

Table 5.1: Results on the SNLI dataset using LSTM-based sentence encoders.

Matching Fn.	**# Params.**	**Acc. (%)**
Concat	2.25M	82.4
Heuristic	2.86M	84.6
ElBiS ($M = 1$)	1.94M	84.8
ElBiS ($M = 2$)	2.25M	85.6
ElBiS ($M = 3$)	2.56M	**85.9**
ElBiS ($M = 4$)	2.87M	85.6

Table 5.2: Results on the SNLI dataset using Gumbel Tree-LSTM–based sentence encoders.

applied several regularization techniques such as dropout and adding a L_2 penalty term, however we still experienced a similar result.

5.3.2 Paraphrase Identification

Another popular task on identifying relationship between a sentence pair is paraphrase identification (PI). The objective of the PI task is to predict whether a given sentence pair has the same meaning or not. In other words, a PI model should capture the variability of natural language, where the same meaning could have multiple expressions. Thus to correctly identify the paraphrase relationship, an input to a classifier should contain the semantic similarity and difference between sentences.

For evaluation of paraphrase identification, we use Quora Question Pairs dataset[5]. The dataset contains 400k question pairs, each of which is annotated with a label indicating whether questions of a pair have the same meaning. To our knowledge, the

[5]`https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`

Quora dataset is the largest available dataset of paraphrase identification. We used the same training, development, test splits as the ones used in Wang et al. (2017b).

For baseline experiments with heuristic matching, we used the function proposed by Ji and Eisenstein (2013), which is shown by the authors to be effective in matching vectors in latent space compared to simple concatenation. It is composed of the element-wise product and absolute difference between two vectors:

$$\mathbf{r} = \begin{bmatrix} \mathbf{a} \odot \mathbf{b} \\ |\mathbf{a} - \mathbf{b}| \end{bmatrix}, \tag{5.5}$$

where $\mathbf{a}$ and $\mathbf{b}$ are encoded sentence vectors.

We used the same architecture and training procedures as NLI experiments, except the final projection matrix and the heuristic matching function. Also, we found that the PI task is more sensitive to hyperparameters than NLI, so we apply different dropout probabilities to the encoder network and to the classifier network. Both values are selected from $\{0.10, 0.15, 0.20\}$. Each model is trained for 15 epochs, and the checkpoint with the highest validation accuracy is chosen as the final model.

The results on the PI task is listed in Table 5.3. Again we can see that the models armed with the ElBiS matching function discover parsimonious and effective interaction between vectors.

Though the proposed ElBiS matching function brought performance gain compared to the concatenation and the heuristic matching function, the differences were not as large as SNLI experiments. We speculate that it is due to the additional inherent trait of the PI task: a matching function should have the commutative property, i.e. if sentence $\mathbf{a}$ is a paraphrase of $\mathbf{b}$, then $\mathbf{b}$ must also be a paraphrase of $\mathbf{a}$. The heuristic function used is designed to reflect this property, while the ElBiS function has to find out the property automatically from data. Imposing a restriction of commutativeness on the proposed function might help discovering a suitable aggregation

Matching Fn.	**# Params.**	**Acc. (%)**
Concat	1.34M	85.0
Heuristic	1.34M	87.0
ElBiS ($M = 1$)	1.04M	86.7
ElBiS ($M = 2$)	1.35M	**87.3**
ElBiS ($M = 3$)	1.66M	87.1

Table 5.3: Results on the Quora Question Pairs dataset using LSTM-based sentence encoders.

scheme, however from the results we observed that it can still learn to extract appropriate features without any prior knowledge about the task.

5.4 Summary and Discussion

In this chapter, we proposed ElBiS, a general method of fusing information from two sentence vectors. Our method does not rely on heuristic knowledge constructed for a specific task, and adaptively learns from data the element-wise connections between vectors from data. From experiments, we demonstrated that the proposed method outperforms or matches the performance of commonly used concatenation-based or heuristic-based feature functions, while maintaining the fused representation compact.

Although the main focus of this work is about sentence matching, the notion of element-wise bilinear interaction could be applied beyond sentence matching. For example, many models that specialize in NLI have components where the heuristic matching function is used, e.g. in computing intra-sentence or inter-sentence attention weights. It could be interesting future work to replace these components with our proposed matching function.

One of the main drawback of our proposed method is that, due to its improved expressiveness, it makes a model overfit easily. When evaluated on small datasets such as Sentences Involving Compositional Knowledge dataset (SICK, Marelli et al., 2014) and Microsoft Research Paraphrase Corpus (MSRP, Dolan and Brockett, 2005), we observed performance degradation, partly due to overfitting. Similarly, we observed that increasing the number of interaction types M does not guarantee consistent performance gain. In preliminary experiments we attempted to use typical regularization techniques such as dropout and L_2 penalty, however it does not help mitigate the overfitting. Considering our objective—obtaining a compact and parsimonious aggregation scheme, we conjecture that the problem could be alleviated by applying regularization techniques that control the sparsity of interaction e.g. L_1 penalty. For better understanding of learned weights, block-sparse regularization techniques e.g. $L_{2,1}$ norm could also be adopted, and we think it would be an intriguing future work direction.

Chapter 6

Semi-Supervised Training: Cross-Sentence Latent Variable Model

6.1 Motivation

With the emergence of large-scale corpora, end-to-end deep learning models are achieving remarkable results on text sequence matching; these include architectures that are linguistically motivated (Bowman et al., 2016a; Chen et al., 2017b), that introduce external knowledge (Chen et al., 2018b), and that use attention mechanisms (Parikh et al., 2016; Shen et al., 2018b).

However, despite the success of deep neural networks in natural language processing, the fact that they require abundant training data might be problematic, as constructing labeled data is a time-consuming and labor-intensive process. To mitigate the data scarcity problem, several semi-supervised learning paradigms, that take advantage of unlabeled data when only some of the data examples are labeled (Chapelle et al., 2010), are proposed. These unlabeled data are much easier to collect, thus utilizing them could be a good option; for example in sentence matching possibly related sentences pairs could be retrieved from a database of text via simple heuristics such as word overlap.

In this chapter, we propose a cross-sentence latent variable model for semi-supervised sentence matching. The proposed framework is based on deep probabilistic generative models (Kingma and Welling, 2014; Rezende et al., 2014) and is extended to make use of unlabeled data. As it is trained to generate a sentence that has a given relationship with a source sentence, both sentences in a pair are utilized together, and thus training objectives are defined more naturally than other models that consider each sentence separately (Zhao et al., 2018; Shen et al., 2018a). To further regularize the model to generate more plausible and diverse sentences, we define semantic constraints and use them for fine-tuning.

6.2 Preliminaries

6.2.1 Variational Auto-Encoders

Variational auto-encoder (VAE, Kingma and Welling, 2014) is a deep generative model for modeling the data distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$. It assumes that a data point $\mathbf{x}$ is generated by the following random process: (1) $\mathbf{z}$ is sampled from $p(\mathbf{z})$ and (2) $\mathbf{x}$ is generated from $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

Thus the natural training objective would be to directly maximize the marginal log-likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log \int_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z}$. However it is intractable to compute the marginal log-likelihood without using simplifying assumption such as mean-field approximation (Blei et al., 2017). Therefore the following variational lower bound $-\mathcal{L}$ is used as a surrogate objective:

$$-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \right], \tag{6.1}$$

where $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is a variational approximation to the unknown $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, and $D_{KL}(q \| p)$

is the Kullback-Leibler (KL) divergence between q and p. Maximizing the surrogate objective $-\mathcal{L}$ is proven to minimize $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$, and it can also be seen as maximizing the expected data log-likelihood with respect to $q_{\boldsymbol{\phi}}$ while using $D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}))$ as a regularization term.

VAEs are successfully applied in modeling various data: including image (Pu et al., 2016; Gulrajani et al., 2017), music (Roberts et al., 2018), and text (Miao et al., 2016; Bowman et al., 2016b). The VAE framework can also be extended to constructing conditional generative models (Sohn et al., 2015) or learning from semi-supervised data (Kingma et al., 2014; Xu et al., 2017).

VAEs for Text Pair Modeling

The most simple approach to modeling text pairs using the VAE framework is to consider two text sequences separately (Zhao et al., 2018; Shen et al., 2018a). That is, a generator is trained to reconstruct a single input sequence rather than integrating both sequences, and the two latent representations encoded from a variational posterior are given to a classifier network. When label information is not available, only the reconstruction objective is used for training. This means that the classifier parameters are not updated in the unsupervised setting, and thus the interaction between the variational posterior (or encoder) and the classifier could be restricted.

Though not directly related to the problem we tackle, there exists some prior work on cross-sentence generating latent variable models (LVMs). Shen et al. (2017) introduce a similar data generation assumption to ours and apply the idea to unaligned style transfer and natural language generation. Zhang et al. (2016a); Serban et al. (2017) use latent variable models for machine translation and dialogue generation. Deudon (2018) build a sentence-reformulating deep generative model whose objective is to measure the semantic similarity between a sentence pair. However their work cannot be applied to a multi-class classification problem, and the generative objective is only used in pre-training, not considering the joint optimization of the generative and the discriminative objective. To the best of our knowledge, our work is the first work on introducing the concept of cross-sentence generating LVM to the semi-supervised text matching problem.

6.2.2 von Mises–Fisher Distribution

Since the advent of deep generative models with variational inference, the typical choice for prior and variational posterior distribution has been the Gaussian, likely due to its well-studied properties and easiness of reparameterization. However it often leads a model to face the posterior collapse problem where a model ignores latent variables by pushing the KL divergence term to zero (Chen et al., 2017c; van den Oord et al., 2017), especially in text generation models where powerful decoders are used (Bowman et al., 2016b; Yang et al., 2017).

Various techniques are proposed to mitigate this problem: including KL cost annealing (Bowman et al., 2016b), weakening decoders (Yang et al., 2017), skip connection (Dieng et al., 2019), using different objectives (Alemi et al., 2018), and using alternative distributions (Guu et al., 2018). In this work, we take the last approach by utilizing a von Mises–Fisher (vMF) distribution.

A vMF distribution is a probability distribution on the $(d-1)$-sphere, therefore samples are compared according to their directions, reminiscent of the cosine similarity. It has two parameters—mean direction $\boldsymbol{\mu} \in \mathbb{R}^d$ and concentration $\kappa \in \mathbb{R}$. The probability density function (pdf) of vMF$(\boldsymbol{\mu}, \kappa)$ is defined by

$$f(\mathbf{x}; \boldsymbol{\mu}, \kappa) = C_m(\kappa) \exp(\kappa \boldsymbol{\mu}^\top \mathbf{x}), \tag{6.2}$$

where

$$C_m(\kappa) = \frac{\kappa^{m/2-1}}{(2\pi)^{m/2} I_{m/2-1}(\kappa)}$$

and $I_v(\kappa)$ is the modified Bessel function of the first kind at order v.

The KL divergence between a vMF distribution $\mathrm{vMF}(\boldsymbol{\mu}, \kappa)$ and the hyperspherical uniform distribution $\mathcal{U}(S^{m-1}) = \mathrm{vMF}(\cdot, 0)$ can be derived analytically:

$$D_{KL}(\mathrm{vMF}(\boldsymbol{\mu}, \kappa) \| \mathrm{vMF}(\cdot, 0)) = \log C_m(\kappa) - \log \frac{\Gamma(m/2)}{2\pi^{m/2}} + \kappa \frac{I_{m/2}(\kappa)}{I_{m/2-1}(\kappa)}. \tag{6.3}$$

Note that the KL divergence does not depend on $\boldsymbol{\mu}$, thus the KL divergence is a constant if κ is fixed. Intuitively, this is because the hyperspherical uniform distribution has equal probability density at every point on the unit hypersphere, and $D_{KL}(\mathrm{vMF}(\boldsymbol{\mu}, \kappa) \| \mathrm{vMF}(\cdot, 0))$ should not be changed under rotations. Therefore when $\mathrm{vMF}(\boldsymbol{\mu}, \kappa)$ with fixed κ and $\mathrm{vMF}(\cdot, 0)$ are used as posterior and prior, the posterior collapse does not occur inherently.

A sample from a vMF distribution is drawn from the acceptance-rejection scheme presented in Algorithm 1 of Davidson et al. (2018). In their algorithm, a stochastic variable obtained from the acceptance-rejection sampling does not depend on μ, thus the sampling process can be rewritten as a deterministic function that accepts the stochastic variable as input (i.e. reparameterization trick).

To the best of our knowledge, Guu et al. (2018) were the first to use vMF as posterior and prior for VAEs, and Xu and Durrett (2018) empirically proved the effectiveness of vMF-VAE in natural language generation. Davidson et al. (2018) generalized the vMF-VAE and proposed the reparameterization trick for vMF.

6.3 Proposed Framework: CS-LVM

In this section, we describe the proposed framework in detail. We formally define the cross-sentence latent variable model (CS-LVM) and describe the optimization objectives. We also introduce semantic constraints to keep learned representations in a semantically plausible region. Fig. 6.1 illustrates the entire framework.

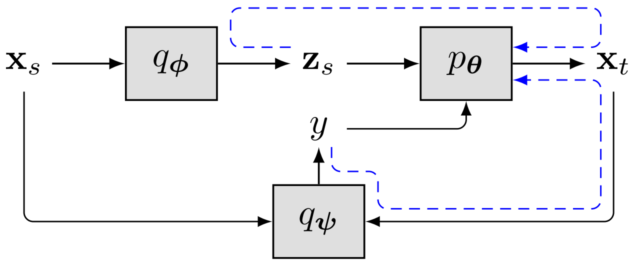


Figure 6.1: The overview of the entire CS-LVM framework. Blue dashed lines indicate semantic constraints.

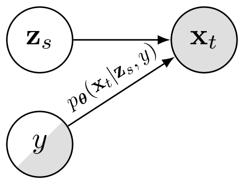
6.3.1 Cross-Sentence Latent Variable Model

Though the auto-encoding frameworks described in §6.2.1 have intriguing properties, it may hinder the possibility of training an encoder to extract rich features for text pair modeling, due to the fact that the generative modeling process is confined within a single sequence. Therefore the interaction between a generative model and a discriminative classifier is restricted, since the two sequences are separately modeled and the pair-wise information is only considered through the classifier network.

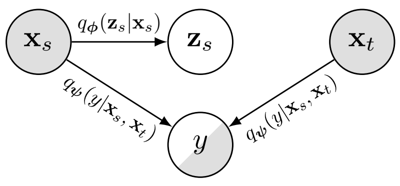
Our proposed CS-LVM addresses this problem by cross-sentence generation of text given a text pair and its label. As the sentences in a pair are directly related within a generative model, the training objectives are defined in a more principled way than VAE-based semi-supervised text matching frameworks. Notably it also mimics the dataset construction process of some corpora: *a worker generates a target text given a label and a source text* (e.g. Bowman et al., 2015; Williams et al., 2018b).

Given a pair $(\mathbf{x}_1, \mathbf{x}_2)$, let $\mathbf{x}_s$, $\mathbf{x}_t \in \{\mathbf{x}_1, \mathbf{x}_2\}$ be a source and a target sequence respectively. Then we assume $\mathbf{x}_t$ is generated according to the following process (see Fig. 6.2a):

1. a latent variable $\mathbf{z}_s$ that contains the content of a source sequence is sampled from $p(\mathbf{z}_s)$,



(a) Generation model


$\mathbf{x}_s$
$q_{\phi}(\mathbf{z}_s|\mathbf{x}_s)$
$\mathbf{z}_s$
$\mathbf{x}_t$
$q_{\psi}(y|\mathbf{x}_s, \mathbf{x}_t)$
y
$q_{\psi}(y|\mathbf{x}_s, \mathbf{x}_t)$

(b) Recognition model

Figure 6.2: Illustration of the graphical models of CS-LVM. (a) the generative process of the output $\mathbf{x}_t$; (b) the approximate inference of $\mathbf{z}_s$ and the discriminative classifier for y.

2. a variable y that determines the relationship between a target and the source sequence is sampled from $p(y)$,

3. $\mathbf{x}_t$ is generated from a conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y)$.

In the above process, the class label y is treated as a hidden variable in the unsupervised case and an observed variable in the supervised case.

Accordingly, when the label information is available, the optimization objective for a generative model is the marginal log-likelihood of the observed variables $\mathbf{x}_t$ and y:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}_t, y) = \log \int p_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{z}_s, y) d\mathbf{z}_s = \log \int p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y) p(\mathbf{z}_s) p(y) d\mathbf{z}_s. \tag{6.4}$$

To address the intractability of computing the true objective, we derive the lower bound of Eq. 6.4.

Let $q_{\boldsymbol{\theta}}(\mathbf{z}_s|\cdot)$ be a distribution that has the same support with $p(\mathbf{z}_s)$. Then the KL divergence between $q_{\boldsymbol{\theta}}(\mathbf{z}_s|\cdot)$ and $p_{\boldsymbol{\theta}}(\mathbf{z}_s|\mathbf{x}_t, y)$ can be written as

$$\begin{aligned}
&D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}_s|\cdot)\|p_{\boldsymbol{\theta}}(\mathbf{z}_s|\mathbf{x}_t, y)) \\
&\qquad = \int q_{\boldsymbol{\phi}}(\mathbf{z}_s|\cdot)\log\frac{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\cdot)}{p_{\boldsymbol{\theta}}(\mathbf{z}_s|\mathbf{x}_t, y)}d\mathbf{z}_s \\
&\qquad = \int q_{\boldsymbol{\phi}}(\mathbf{z}_s|\cdot)\log\frac{p_{\boldsymbol{\theta}}(\mathbf{x}_t, y)q_{\boldsymbol{\phi}}(\mathbf{z}_s|\cdot)}{p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y)p(\mathbf{z}_s)p(y)}d\mathbf{z}_s \\
&\qquad = \log p_{\boldsymbol{\theta}}(\mathbf{x}_t, y) + D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}_s|\cdot)\|p(\mathbf{z}_s)) \\
&\qquad\qquad - \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\cdot)}[\log p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y)] - \log p(y) \\
&\qquad \geq 0.
\end{aligned} \tag{6.5}$$

From the inequality above we obtain the lower bound of $\log p_{\boldsymbol{\theta}}(\mathbf{x}_t, y)$ as follows:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}_t, y) \geq -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)\|p(\mathbf{z}_s)) + \log p(y) \\
+ \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)}[\log p_{\boldsymbol{\theta}}(\mathbf{x}_t|y, \mathbf{z}_s)],
\end{aligned} \tag{6.6}$$

where $q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)$ is a variational approximation of the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}_s|\mathbf{x}_t, y)$.

Though Eq. 6.6 holds for any $q_{\boldsymbol{\phi}}$ having the same support with $p(\mathbf{z}_s)$, we choose this form of variational posterior from the following motivation: *since* $\mathbf{x}_s$ *is related to* $\mathbf{x}_t$ *by the label information* y*,* $\mathbf{x}_s$ *would have an influence on the space of* $\mathbf{z}_s$ *in a similar way to* $(\mathbf{x}_t, y)$. Due to this particular choice of $q_{\boldsymbol{\phi}}$, $\mathbf{z}_s$ depends only on $\mathbf{x}_s$ and is independent of the label information possibly permeated in $\mathbf{x}_t$. In other words, this design induces $q_{\boldsymbol{\phi}}$ to extract the features needed for controlling the semantics only from $\mathbf{x}_s$, while preventing $q_{\boldsymbol{\phi}}$ from encoding other biases.

To extend the objective to the unsupervised setup, we marginalize out y from Eq. 6.6 using a classifier distribution. We will provide more detailed explanation of the optimization objectives in §6.3.3.

6.3.2 Architecture

Now we describe the architectures we used for constructing CS-LVM. We first encode a source sequence into a fixed-length representation using a recurrent neural network (RNN): $g^{enc}(\mathbf{x}_s) = \mathbf{m}_s$. From $\mathbf{m}_s$ we obtain a variational approximate distribution $q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s) = g^{code}(\mathbf{m}_s)$ and sample a latent representation $\mathbf{z}_s \sim q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)$. In our experiments, a long short-term memory (LSTM) recurrent network and a feed-forward network are used as g^{enc} and g^{code} respectively. From the fact that the mean direction parameter $\boldsymbol{\mu}_s$ of $\mathrm{vMF}(\boldsymbol{\mu}_s, \kappa)$ should be a unit vector, g^{code} additionally normalizes the output of the feed-forward network to be $\|g^{code}(\mathbf{m}_s)\|_2 = 1$.

Then we generate the target sequence $\mathbf{x}_t$ from $\mathbf{z}_s$ and y. Similarly to the encoder network, we use an LSTM for a decoder, thus the distribution is factorized as follows:

$$p_{\boldsymbol{\theta}}(\mathbf{x}_t|y, \mathbf{z}_s) = \prod_{i=1}^{N_{\mathbf{x}_t}+1} p_{\boldsymbol{\theta}}(w_{t,i}|w_{t,<i}, y, \mathbf{z}_s), \tag{6.7}$$

where $\mathbf{x}_t = (x_{t,1}, \ldots, x_{t,N_{\mathbf{x}_t}})$ and $w_{t,0} = \texttt{<s>}$, $w_{t,N_{\mathbf{x}_t}+1} = \texttt{</s>}$ are special tokens indicating the start and the end of a sequence.

We project the word index $w_{t,i}$ and label index y into embedding spaces to obtain the word embedding $\mathbf{w}_{t,i}$ and label embedding $\mathbf{y}$. Then to construct an input for i-th time step, $\mathbf{v}_t$, we concatenate the i-th target word embedding $\mathbf{w}_{t,i}$, the label embedding $\mathbf{y}$, and the latent representation $\mathbf{z}_s$ altogether:

$$\mathbf{v}_i = [\mathbf{w}_{t,i}; \mathbf{y}; \mathbf{z}_s]. \tag{6.8}$$

Thus $p_{\boldsymbol{\theta}}(w_{t,i}|w_{t,<i}, \mathbf{z}_s, y)$ is computed from i-th state $\mathbf{s}_i$ of the decoder RNN:

$$p_{\boldsymbol{\theta}}(w_{t,i}|w_{t,<i}, y, \mathbf{z}_s) = \mathrm{softmax}(g^{out}(\mathbf{s}_i)) \tag{6.9}$$

$$\mathbf{s}_i = g_i^{dec}(\mathbf{v}_i, \mathbf{s}_{i-1}), \tag{6.10}$$

where g^{out} is a feed-forward network and g_i^{dec} is the state transition function of the decoder LSTM at i-th time step.

For a discriminative classifier network we follow the siamese architecture. $\mathbf{x}_s$ and $\mathbf{x}_t$ are fed to a shared LSTM network f^{enc} to obtain sentence vectors $\mathbf{h}_1 = f^{enc}(\mathbf{x}_s)$ and $\mathbf{h}_2 = f^{enc}(\mathbf{x}_t)$. Then $\mathbf{h}_1$ and $\mathbf{h}_2$ are combined by the function f^{fuse} to form a single fused vector, and the fused representation is given to a feed-forward network f^{disc} to infer the relationship:

$$q_\psi(y|\mathbf{x}_1, \mathbf{x}_2) = \text{softmax}(f^{disc}(f^{fuse}(\mathbf{h}_1, \mathbf{h}_2))). \tag{6.11}$$

To learn from data more efficiently and to reduce the number of trainable parameters, we tie the weights for two encoders—for the generative model and the discriminative classifier; i.e. $g^{enc} = f^{enc}$. This mitigates the problem that only source sequences are used for training g^{enc} and enhances the interaction between the generative model and the classifier. We will see from experiments that tying encoder weights improves performance and stabilizes optimization (§6.4.3).

6.3.3 Optimization

In this subsection we describe how the entire model is optimized. We first define optimization objectives for supervised and unsupervised training, and then introduce constraints to regularize the model to generate sequences with intended semantic characteristics. The entire optimization procedure is summarized in Algorithm 1.

Supervised Objective

In the supervised setting, a data sample is assumed to contain label information: $(\mathbf{x}_1, \mathbf{x}_2, y) \in \mathcal{X}_l$. Without loss of generality let us assume $(\mathbf{x}_s, \mathbf{x}_t) = (\mathbf{x}_1, \mathbf{x}_2)$.[1] Since y is an observed variable in this case, we can directly use Eq. 6.6 in training. From

[1] The relationship between a source and a target may either be unidirectional, bidirectional, or reflexive, depending on the characteristics of a task. For some experiments we additionally used swapped data examples, $(\mathbf{x}_s, \mathbf{x}_t) = (\mathbf{x}_2, \mathbf{x}_1)$, for training. We explain more on this in §6.4.

Algorithm 1 Training procedure of CS-LVM.

```
Input: Labeled dataset $\mathcal{X}_l$,
       Unlabeled dataset $\mathcal{X}_u$,
       Model parameters $\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}$
 1: procedure TRAIN($\mathcal{X}_l, \mathcal{X}_u, \boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}$)
 2:     repeat
 3:         Sample $(\mathbf{x}_{l,s}, \mathbf{x}_{l,t}, y_l) \sim \mathcal{X}_l$
 4:         Sample $(\mathbf{x}_{u,s}, \mathbf{x}_{u,t}) \sim \mathcal{X}_u$
 5:         Compute $\mathcal{L}_l(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}; \mathbf{x}_{l,s}, \mathbf{x}_{l,t}, y_l)$ by (6.14)
 6:         Compute $\mathcal{L}_u(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}; \mathbf{x}_{u,s}, \mathbf{x}_{u,t})$ by (6.18)
 7:         Update $\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}$ by gradient descent on $\mathcal{L}_l + \mathcal{L}_u$
 8:     until stop criterion is met
 9: procedure FINETUNE($\mathcal{X}_l, \mathcal{X}_u, \boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}$)
10:     repeat
11:         Update $\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}$ following line 3–7
12:         Update $\boldsymbol{\theta}$ by gradient descent on (6.20–6.23)
13:     until stop criterion is met
```

Eqs. 6.6 and 6.7, the objective for the generative model is defined by:[2]

$$-\mathcal{L}_l^{gen}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}_s, \mathbf{x}_t, y) = \log p_{\boldsymbol{\theta}}(\mathbf{x}_t | y, \mathbf{z}_s) + \log p(y) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s) \| p(\mathbf{z}_s)), \quad (6.12)$$

where $\mathbf{z}_s \sim q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)$ and $p(y)$, $p(\mathbf{z}_s)$ are prior distributions of y, $\mathbf{z}_s$. Considering that we assume $p(y)$ to be a fixed uniform distribution of labels, the $\log p(y)$ term can be ignored in training: $\|\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \log p(y)\|_2 = 0$.

For training, the typical teacher forcing method is used; i.e. ground-truth words are used as input words. We use vMF$(g^{code}(\mathbf{m}_s), \kappa)$ (κ: hyperparameter) for the variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)$ and vMF$(\cdot, 0)$ for the prior $p(\mathbf{z}_s)$.

[2]Note that we define all objectives $\mathcal{L}$, $\mathcal{R}$ as *minimization objectives* to avoid confusion.

The discriminator objective is defined as a conventional maximum likelihood:

$$-\mathcal{L}_l^{disc}(\boldsymbol{\psi}; \mathbf{x}_s, \mathbf{x}_t, y) = \log q_{\boldsymbol{\psi}}(y|\mathbf{x}_s, \mathbf{x}_t). \tag{6.13}$$

Finally, the two objectives are combined to construct the objective for supervised training:

$$\mathcal{L}_l(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}; \mathbf{x}_s, \mathbf{x}_t, y) = \mathcal{L}_l^{gen} + \lambda \mathcal{L}_l^{disc}, \tag{6.14}$$

where λ is a hyperparameter.

Unsupervised Objective

In this case, the model does not have an access to label information; a data point is represented by $(\mathbf{x}_s, \mathbf{x}_t) \in \mathcal{X}_u$ and thus y is a hidden variable. To facilitate the unsupervised training, we marginalize y out as below and derive the lower bound:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}_t) &= \log \sum_y \int p_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{z}_s, y) d\mathbf{z}_s \\
&= \log \mathbb{E}_{q_{\boldsymbol{\phi},\boldsymbol{\psi}}(y, \mathbf{z}_s|\mathbf{x}_s, \mathbf{x}_t)} \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y) p(\mathbf{z}_s) p(y)}{q_{\boldsymbol{\phi},\boldsymbol{\psi}}(y, \mathbf{z}_s|\mathbf{x}_s, \mathbf{x}_t)} \right] \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi},\boldsymbol{\psi}}(y, \mathbf{z}_s|\mathbf{x}_s, \mathbf{x}_t)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y) p(\mathbf{z}_s) p(y)}{q_{\boldsymbol{\phi},\boldsymbol{\psi}}(y, \mathbf{z}_s|\mathbf{x}_s, \mathbf{x}_t)} \right].
\end{aligned} \tag{6.15}$$

And from the assumption presented in the graphical model (Fig. 6.2b),

$$q_{\boldsymbol{\phi},\boldsymbol{\psi}}(y, \mathbf{z}_s|\mathbf{x}_s, \mathbf{x}_t) = q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s) q_{\boldsymbol{\psi}}(y|\mathbf{x}_s, \mathbf{x}_t). \tag{6.16}$$

From Eq. 6.16,

$$\begin{aligned}
&\mathbb{E}_{q_{\boldsymbol{\phi},\boldsymbol{\psi}}(y, \mathbf{z}_s|\mathbf{x}_s, \mathbf{x}_t)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y) p(\mathbf{z}_s) p(y)}{q_{\boldsymbol{\phi},\boldsymbol{\psi}}(y, \mathbf{z}_s|\mathbf{x}_s, \mathbf{x}_t)} \right] \\
&= \mathbb{E}_{q_{\boldsymbol{\psi}}} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}_s, y) p(\mathbf{z}_s) p(y)}{q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)} \right] \right] - \mathbb{E}_{q_{\boldsymbol{\psi}}}[\log q_{\boldsymbol{\psi}}(y|\mathbf{x}_s, \mathbf{x}_t)] \\
&= \mathbb{E}_{q_{\boldsymbol{\psi}}} \left[-\mathcal{L}_l^{gen}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}_s, \mathbf{x}_t, y) \right] + \mathcal{H}(q_{\boldsymbol{\psi}}(y|\mathbf{x}_s, \mathbf{x}_t)).
\end{aligned} \tag{6.17}$$

Finally we obtain the following lower bound for $\log p_{\boldsymbol{\theta}}(\mathbf{x}_t)$:

$$\mathcal{L}_u(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}; \mathbf{x}_s, \mathbf{x}_t) = -\mathcal{H}(q_{\boldsymbol{\psi}}(y|\mathbf{x}_s, \mathbf{x}_t)) + \mathbb{E}_{q_{\boldsymbol{\psi}}(y|\mathbf{x}_s, \mathbf{x}_t)} \left[\mathcal{L}_l^{gen}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}_s, \mathbf{x}_t, y) \right]. \tag{6.18}$$

Here the second expectation term can be computed either by enumeration or sampling, and we used the former as the datasets we used have relatively small label sets (2 or 3) and it is known to yield better results than sampling (Xu et al., 2017). We will compare the two methods in §6.4.3.

To sum up, at every training iteration, given a labeled and unlabeled data sample $(\mathbf{x}_{l,s}, \mathbf{x}_{l,t}, y_l)$, $(\mathbf{x}_{u,s}, \mathbf{x}_{u,t})$, we optimize the following objective.

$$\mathcal{L} = \mathcal{L}_l(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}; \mathbf{x}_{l,s}, \mathbf{x}_{l,t}, y_l) + \mathcal{L}_u(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}; \mathbf{x}_{u,s}, \mathbf{x}_{u,t}) \tag{6.19}$$

Fine-Tuning with Semantic Constraints

Since the generator is trained via maximum likelihood training which considers all words in a sentence equivalently, the label information may not be reflected enough in generation owing to high-frequency words. For example in natural language inference, the word occurrences of the following three hypothesis sentences highly overlap, but they should have different relation with the premise.[3]

- **P**: *A man is cutting metal with a tool .*
- **H1**: *A man is cutting metal .*
- **H2**: *A man is cutting metal with the wrong tool .*
- **H3**: *A man is cutting metal with his mind .*

Thus for some data points, the strategy that only predicts words that overlap across hypotheses could receive a fairly high score, which might weaken the integration of y into the generator. To mitigate this, we fine-tune the trained generator using the following semantic constraint:

$$-\mathcal{R}^y(\boldsymbol{\theta}; \mathbf{x}_s, \mathbf{x}_t) = \log q_{\boldsymbol{\psi}}(\tilde{y}|\mathbf{x}_s, \widetilde{\mathbf{x}}_t), \tag{6.20}$$

[3]Examples are taken from the development split of the SNLI dataset, pair ID `4904199439.jpg#2r1e`, `4904199439.jpg#2r1n`, `4904199439.jpg#2r1c`.

where $\tilde{y} \sim p(y)$, $\mathbf{z}_s \sim q_{\boldsymbol{\phi}}(\mathbf{z}_s|\mathbf{x}_s)$, and $\widetilde{\mathbf{x}}_t = \text{argmax}_{\mathbf{x}_t} p_{\boldsymbol{\theta}}(\mathbf{x}_t|\tilde{y}, \mathbf{z}_s)$. This constraint enforces the sequence $\widetilde{\mathbf{x}}_t$ generated by conditioning on $\tilde{y}$ and $\mathbf{z}_s$ to actually have the relationship $\tilde{y}$ with $\mathbf{x}_s$.

We also introduce a constraint on $\mathbf{z}$ that keeps the distributions of $\widetilde{\mathbf{z}}_t$ (the latent content variable obtained by encoding the generated sequence $\widetilde{\mathbf{x}}_t$) and $\mathbf{z}_s$ close:

$$-\mathcal{R}^{\mathbf{z}}(\boldsymbol{\theta}; \mathbf{x}_s, \mathbf{x}_t) = \log q_{\boldsymbol{\phi}}(\mathbf{z}_t = \widetilde{\mathbf{z}}_t|\mathbf{x}_t), \tag{6.21}$$

where $\widetilde{\mathbf{z}}_t \sim q_{\boldsymbol{\phi}}(\widetilde{\mathbf{z}}_t|\widetilde{\mathbf{x}}_t)$. In other words, it pushes the generated sequence $\widetilde{\mathbf{x}}_t$ to be in a similar semantic space with the ground-truth target sequence $\mathbf{x}_t$. Consequently, it can help alleviate the generator collapse problem where a generator produces only a handful of simple neutral patterns independent of the input sequence, by relating $\widetilde{\mathbf{z}}_t$ to $\mathbf{z}_t$.[4]

From similar motivation, we also add an additional constraint that encourages the generated sentences originating from different source sentences to be dissimilar. To reflect this, we define the following minibatch-level constraint that penalizes the mean direction vectors encoded from the generated sentences for being too close:

$$-\mathcal{R}^{\boldsymbol{\mu}}(\boldsymbol{\theta}; \mathcal{B}) = \mathbb{E}_{\mathcal{B}}[d(\boldsymbol{\mu}_t{}^{(i)}, \bar{\boldsymbol{\mu}}_t)], \tag{6.22}$$

where we denote values related to i-th sample of a minibatch $\mathcal{B}$ using superscript: $\square^{(i)}$. In the above, $\boldsymbol{\mu}_t^{(i)} = g^{code}(g^{enc}(\widetilde{\mathbf{x}}_t^{(i)}))$, $\bar{\boldsymbol{\mu}}_t = \sum_{i=1}^{|\mathcal{B}|} \boldsymbol{\mu}_t^{(i)}/|\mathcal{B}|$, and $d(\cdot,\cdot)$ is a distance measure between vectors. The mean direction vector $\boldsymbol{\mu}$ of vMF$(\boldsymbol{\mu}, \kappa)$ is on a unit hypersphere, so we use the cosine distance: $d(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2) = 1 - \langle \boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \rangle$.

As the sequence generation process is not differentiable, the gradients from the semantic constraints cannot propagate to the generator parameters. To relax the discreteness, we use the Gumbel-Softmax reparameterization (Jang et al., 2017; Maddison et al., 2017). Using the Gumbel-Softmax trick, we obtain a continuous probability

[4] The basic assumption behind this constraint is that a source and a target sequence are associated in a certain aspect, and it generally holds in most of the available pair classification datasets e.g. SNLI, SICK, SciTail, QQP, MRPC.

vector that approximates a sample from the categorical distribution of words at each step, and use the probability vector to compute the expected word embedding for the subsequent step.

When multiple constraints are used, they are combined using the homoscedastic uncertainty weighting (Kendall et al., 2018):[5]

$$\mathcal{R} = \frac{1}{\sigma_1^2}\mathcal{R}^{y} + \frac{1}{\sigma_2^2}\mathcal{R}^{\mathbf{z}} + \frac{1}{\sigma_3^2}\mathcal{R}^{\boldsymbol{\mu}} + \log\sigma_1 + \log\sigma_2 + \log\sigma_3, \tag{6.23}$$

where $\sigma_1, \sigma_2, \sigma_3$ are trainable scalar parameters. Also note that all constraints are *unsupervised*, where label information is not required.

6.4 Experiments

We evaluate the proposed model on two semi-supervised tasks: *natural language inference* and *paraphrase identification*. We also implement a strong baseline that has a similar architecture to LSTM-VAE (Shen et al., 2018a) but uses vMF distribution for prior and posterior, named LSTM-vMF-VAE. To further explore the proposed model, we conduct extensive qualitative analyses.

6.4.1 Natural Language Inference

Natural language inference (NLI) is a task of predicting the relationship given a premise and a hypothesis sentence. We use Stanford Natural Language Inference (SNLI, Bowman et al., 2015) dataset for experiments. It consists of roughly 570k premise-hypothesis pairs, and each pair has one of the following labels: *entailment*, *neutral*, and *contradiction*. Considering the asymmetry in some label classes and for conformance with the dataset generation process, we use premise and hypothesis sentence as source and target respectively: $(\mathbf{x}_s, \mathbf{x}_t) = (\mathbf{x}_{pre}, \mathbf{x}_{hyp})$.

[5]Though the weighting scheme is originally derived from the case of a Gaussian likelihood, Kendall et al. (2018); Xiong et al. (2018); Hu et al. (2018) successfully applied it in weighting various losses e.g. cross-entropy loss, L_1 loss, and reinforcement learning objectives.

Following the work of Zhao et al. (2018); Shen et al. (2018a), we consider scenarios where 28k, 59k, and 120k labeled data samples are available. Also, for fair comparison with the prior work, we set the size of a word vocabulary set to 20,000 and do not utilize pre-trained word embeddings such as GloVe (Pennington et al., 2014).

To combine the representations of a premise and a hypothesis and to construct an input to f^{disc}, we use the following heuristic-based fusion proposed by Mou et al. (2016):

$$f^{fuse}(\mathbf{h}_{pre}, \mathbf{h}_{hyp}) = [\mathbf{h}_{pre}; \mathbf{h}_{hyp}; |\mathbf{h}_{pre} - \mathbf{h}_{hyp}|; \mathbf{h}_{pre} \odot \mathbf{h}_{hyp}], \tag{6.24}$$

where $[\mathbf{a}; \mathbf{b}]$ indicates concatenation of vectors $\mathbf{a}$, $\mathbf{b}$ and $\odot$ is the element-wise product.

Table 6.1 summarizes the result of experiments. We can clearly see that the proposed CS-LVM architecture substantially outperforms other models based on auto-encoding. Also, the semantic constraints brought additional boost in performance, achieving the new state of the art in semi-supervised classification of the SNLI dataset. When all training data are used as labeled data ($\approx$ 550k), CS-LVM also improves performance by achieving accuracy of 82.8%, compared to the supervised LSTM (81.5%), LSTM-AE (81.6%), LSTM-VAE (80.8%), DeConv-VAE (80.9%).

6.4.2 Paraphrase Identification

Paraphrase identification (PI) is a task whose objective is to infer whether two sentences have the same semantics. We use the Quora Question Pairs dataset (QQP, Wang et al., 2017b) for experiments. QQP consists of over 400k sentence pairs each of which has label information indicating whether the sentences in a pair paraphrase each other or not. We experiment for the cases where the number of labeled data is 1k, 5k, 10k, and 25k, and set the vocabulary size to 10,000, following Shen et al. (2018a). Unlike auto-encoding–based models that treat sentences in a pair equivalently, the CS-LVM processes them asymmetrically for its cross-sentence generating property. This property is useful when some relationships are asymmetric (e.g. NLI), however

Model	**28k**	**59k**	**120k**
LSTM(a)	57.9	62.5	65.9
CNN(b)	58.7	62.7	65.6
LSTM-AE(a)	59.9	64.6	68.5
LSTM-ADAE(a)	62.5	66.8	70.9
DeConv-AE(b)	62.1	65.5	68.7
LSTM-VAE(b)	64.7	67.5	71.1
DeConv-VAE(b)	67.2	69.3	72.2
LSTM-vMF-VAE (ours)	65.6	68.7	71.1
CS-LVM (ours)	68.4	73.5	76.9
$+\mathcal{R}^{y}$	**70.0**	**74.5**	77.4
$+\mathcal{R}^{\mathbf{z}}$	69.2	73.9	77.6
$+\mathcal{R}^{\boldsymbol{\mu}}$	69.1	74.0	**77.6**
$+\mathcal{R}^{y}, \mathcal{R}^{\mathbf{z}}, \mathcal{R}^{\boldsymbol{\mu}}$	69.6	74.1	77.4

Table 6.1: Semi-supervised classification results on the SNLI dataset. (a) Zhao et al. (2018); (b) Shen et al. (2018a).

the paraphrase relationship is bidirectional, so that we also use swapped text pairs in training. To fuse sentence representations, the following symmetric function is used, as in Ji and Eisenstein (2013):

$$f^{fuse}(\mathbf{h}_1, \mathbf{h}_2) = [\mathbf{h}_1 + \mathbf{h}_2; |\mathbf{h}_1 - \mathbf{h}_2|]. \tag{6.25}$$

The result of experiments on QQP is summarized in Table 6.2. Again, the proposed CS-LVM consistently outperforms other supervised and semi-supervised models by a large margin, setting the new state-of-the-art result on the QQP dataset with the semi-supervised setting.

6.4.3 Ablation Study

To assess the effect of each element, we experiment with model variants where some of the components are removed. Specifically, we conduct an ablation study for the following variants: (i) without cross-sentence generation (i.e. auto-encoding setup), (ii) replacing the vMF distribution with Gaussian, (iii) computing the expectation term of Eq. 6.18 by sampling, and (iv) without encoder weight sharing (i.e. $f^{enc} \neq g^{enc}$).

Model	**1k**	**5k**	**10k**	**25k**
CNN[(a)]	56.3	59.2	63.8	68.9
LSTM-AE[(a)]	59.3	63.8	67.2	70.9
DeConv-AE[(a)]	60.2	65.1	67.7	71.6
LSTM-VAE[(a)]	62.9	67.6	69.0	72.4
DeConv-VAE[(a)]	65.1	69.4	70.5	73.7
LSTM-vMF-VAE (ours)	65.0	69.9	72.1	74.9
CS-LVM (ours)	**66.5**	71.1	74.6	76.9
$+\mathcal{R}^y$	66.4	70.8	74.5	77.5
$+\mathcal{R}^{\mathbf{z}}$	**66.5**	**71.3**	74.8	77.1
$+\mathcal{R}^{\boldsymbol{\mu}}$	66.4	71.2	**74.9**	77.4
$+\mathcal{R}^y, \mathcal{R}^{\mathbf{z}}, \mathcal{R}^{\boldsymbol{\mu}}$	66.3	**71.3**	74.7	**77.6**

Table 6.2: Semi-supervised classification results on the Quora Question Pairs dataset. (a) Shen et al. (2018a).

Model	**28k**	**59k**	**120k**
CS-LVM	68.4	73.5	76.9
(i) without CS	65.6	68.7	71.1
(ii) Gaussian	66.9	72.0	74.9
(iii) sampling	68.0	72.9	76.5
(iv) $f^{enc} \neq g^{enc}$	63.3	69.1	74.7

Table 6.3: Ablation study results.

SNLI dataset is used for the model ablation experiments, and trained models are not fine-tuned in order to focus only on the efficacy of each model component.

Results of ablation study are presented in Table 6.3. As expected, the cross-sentence generation is the most critical factor for the performance, except for the 28k setting where the encoder weight tying brought the biggest gain. In 59k and 120k settings, all other variants that maintain the cross-generating property outperform the VAE-based models (see *(ii)*, *(iii)*, *(iv)*).

Replacing a vMF with a Gaussian does not severely harm the accuracy, however it requires the additional process of finding a KL cost annealing rate. When sampling is used instead of enumeration for computing Eq. 6.18, about 1.2x speedup is observed in exchange for slight performance degradation, and thus sampling could be a good

Input	**Entailment**	**Neutral**	**Contradiction**
two girls play with bubbles near a boat dock .	two girls are outside .	the girls are friends .	two girls are swimming in the ocean .
a classroom full of men, with the teacher up front .	a group of boys are indoors .	the teacher is teaching the students .	the students are at home sleeping .
a dune buggy traveling on sand .	the vehicle is moving .	the vehicle is red .	a man is riding a bike .

Table 6.4: Selected samples generated from the model trained on the SNLI dataset.

option in the case that the number of label classes is large.

Finally, as mentioned in §6.3.2, variants whose encoder weights are untied do not work well. We conjecture this is because g^{enc} receives the error signal only from a source sentence and could not fully benefit from both sentences. The fact that the performance degradation is larger when the number of labeled data is small also agrees with our hypothesis, since unlabeled data affect the classifier encoder only by the entropy term when encoder weights are not shared.

6.4.4 Generated Sentences

We give examples of generated sentences, to validate that the proposed model learns to generate text having desired properties. From Table 6.4, we can see that sentences generated from the identical input sentence properly reflect the label information given. More generated examples are presented in §A.1.

Further, to quantitatively measure the quality of generated sentences, we construct artificial datasets, where each premise and label in the SNLI development set is used as input to our trained generator and generated hypotheses are collected. Then we prepare a LSTM classifier that is trained on the original SNLI dataset as a surrogate for the ideal classifier, and use it for measuring the quality of generated datasets.[6] We also compute the diversity of the generated hypotheses using the metrics proposed

[6]The accuracy of the trained classifier on the original development set is 81.7%.

Dataset	**Acc.**	***distinct-1***	***distinct-2***
CS-LVM	76.5	.0128	.0441
$+\mathcal{R}^y$	81.9	.0135	.0479
$+\mathcal{R}^{\mathbf{z}}$	79.0	.0140	.0492
$+\mathcal{R}^{\boldsymbol{\mu}}$	77.5	.0141	.0488

Table 6.5: Results of evaluation of generated artificial datasets. *distinct-1* and *distinct-2* compute the ratio of the number of unique unigrams or bigrams to that of the total generated tokens (Li et al., 2016).

by Li et al. (2016), to verify the effect of diversity-promoting semantic constraints.

Results of the evaluation on the artificial datasets are presented in Table 6.5. The classifier trained on the original dataset predicts the generated data fairly well, from which we verify that the generated sentences contain desired semantics. Also, as expected, fine-tuning with $\mathcal{R}^y$ increases the classification accuracy by a large margin, while $\mathcal{R}^{\mathbf{z}}$ and $\mathcal{R}^{\boldsymbol{\mu}}$ enhance diversity.

6.4.5 Implementation Details

We used PyTorch[7] and AllenNLP[8] libraries for implementation. The default weight initialization scheme of the AllenNLP library is used unless explicitly stated.

For all CS-LVM experiments, the size of word embeddings and hidden dimensions of LSTMs are set to 300, and the size of label embeddings is 50. g^{code} is implemented as a linear projection of the last hidden state of the encoder LSTM followed by normalization. g^{out} is a linear projection followed by the softmax function, and we reuse the word embeddings as its weight matrix (Press and Wolf, 2017; Inan et al., 2017). The discriminative classifier is a feedforward network with single hidden layer and the ReLU activation function, and the hidden dimension is set to 1200. We apply dropout on word embeddings and the classifier with probabilities p_w and p_c respectively.

[7]`https://pytorch.org/`
[8]`https://allennlp.org/`

Model	κ	λ	p_w	p_c
28k	150	0.8	0.75	0.1
59k	100	1.0	0.75	0.1
120k	120	0.8	0.50	0.1

Table 6.6: Hyperparameters for the SNLI models.

Model	κ	λ	p_w	p_c
1k	100	0.8	0.50	0.2
5k	120	0.5	0.75	0.2
10k	150	0.5	0.75	0.1
25k	100	0.5	0.75	0.1

Table 6.7: Hyperparameters for the QQP models.

When multiple semantic constraints are used, to make uncertainty weights be always positive and be optimized stably, we instead use $\log \sigma_i^2$ as model parameter, as in Kendall et al. (2018). Each $\log \sigma_i^2$ is initialized with zero. The temperature parameter of the Gumbel-Softmax is linearly annealed using the following schedule:

$$\tau(t) = \max(0.1, 1.0 - rt), \tag{6.26}$$

where $r = 10^{-4}$ is the annealing rate and t is the training step.

Adam optimizer (Kingma and Ba, 2015) with learning rate $\gamma = 10^{-3}$ is used for all experiments, except for 1k QQP experiments where stochastic gradient descent optimizer is used. When fine-tuning the model, we set γ to 10^{-4}. For other hyperparameters, we follow the configuration suggested by the authors. Best hyperparameter configurations found for SNLI and QQP datasets are presented in Tables 6.6 and 6.7.

For generating sentences, beam search with the beam size $B = 10$ is used, and length normalization (Wu et al., 2016) is applied with $\alpha = 0.7$.

6.5 Summary and Discussion

In this chapter, we proposed a cross-sentence latent variable model (CS-LVM) for semi-supervised text sequence matching. Given a pair of text sequences and the corresponding label, it uses one of the sequences and the label as input and generates the other sequence. Due to the use of cross-sentence generation, the generative model and the discriminative classifier interacts more strongly, and from experiments we empirically proved that the CS-LVM outperforms other models by a large margin. We also defined multiple semantic constraints to further regularize the model, and observed that fine-tuning with them gives additional increase in performance.

For future work, generating more realistic text and use the generated text in other tasks e.g. data augmentation and addressing adversarial attack, would be intriguing directions of research. Although the current model makes fairly plausible sentences, it tends to prefer relatively short and *safe* sentences, as the main goal of the training is to accurately predict the relationship between sentences. We expect the model could perform more natural generation via applying recent advancements on deep generative models.

Chapter 7

Conclusion

In this dissertation, we explored methods of improving the performance of sentence matching based on deep neural network sentence encoders. We have sought for room for improvement in three orthogonal directions: i) constructing a sentence encoder architecture capable of better extracting semantics from natural language text, ii) designing a matching function that automatically learns a suitable aggregation scheme from data, and iii) utilizing text pairs without label information in semi-supervised training.

Ch. 3 and 4 are on improving sentence encoders. In Ch. 3, we proposed a Gumbel Tree-LSTM architecture that learns the parsing strategy maximizing the task performance from data composed of sentences without structure information. As inducing a tree structure is composed of a series of discrete sampling operations, we used the straight-through version of the Gumbel-Softmax estimator to facilitate training. From experiments, we examined that the proposed architecture outperforms other recursive neural network models, without relying on a predefined strategy e.g. constituency-based or dependency-based parsing. We also saw that models optimized on different tasks have distinct parsing strategies, proving our hypothesis that an optimal strategy would differ from task to task.

Though the work on finding an optimal parsing strategy for a specific task is originally started from constructing a powerful sentence encoder, it has contributed to some work on unsupervised grammar induction. Williams et al. (2018a); Htut et al. (2018) examined Gumbel Tree-LSTM models trained on various dataset and compared the resulting parsing strategies. Li et al. (2019) introduced an imitation learning approach into training the Gumbel Tree-LSTM model and achieved improved performance on unsupervised grammar induction.

In Ch. 4, we suggested a method of stacking multiple long short-term memory (LSTM) layers. We observed that the typical setting of stacking LSTM layers where only the hidden states from a previous layer is fed as input to the next layer might not fully benefit from the carefully designed gating mechanism of the LSTM architecture, and proposed the Cell-aware Stacked LSTM (CAS-LSTM) architecture that utilizes both hidden and memory cell states from a previous layer. We saw from experiments that the CAS-LSTM architecture brings performance gains on various tasks. We also conducted qualitative analyses to validate arguments on advantages of using the additional vertical forget gate.

Since the applicability of sentence encoders is not confined to the case of sentence matching, we expect that our architectures—Gumbel Tree-LSTM and CAS-LSTM—would generally help understanding a natural language sentence and extracting useful features for a given task.

In Ch. 5, the element-wise bilinear sentence matching (ElBiS) algorithm is presented. Inspired by prior work on heuristic element-wise matching functions and the hypothesis that sentence representations obtained by a shared encoder would lie in similar semantic spaces, we proposed considering bilinear interaction element-wise, which greatly reduces the number of required parameters compared against full bilinear pooling methods. By automatically learning an optimal element-wise feature extraction scheme for a given task, it lessens the need of expert knowledge in making a sentence matching model. We showed that the ElBiS algorithm is capable of finding a suitable matching function automatically from data, outperforming matching functions based on heuristics.

Ch. 6 is about utilizing unsupervised data along with supervised data (i.e. semi-supervised training) for sentence matching. We proposed a cross-sentence latent variable model (CS-LVM), which is a deep generative model that considers both sentences in a pair within a single model. It assumes that a target sentence is generated from a latent representation of a source sentence. We also designed semantic constraints to make the model to generate more semantically plausible sentences. Unlike previous approaches based on variational auto-encoders (VAEs), the optimization objectives for our model are defined in a more natural and probabilistic way. We empirically proved that the proposed CS-LVM outperforms strong baselines and previous VAE-based models given the same amount of labeled data, and that it also produces sentences that have meaningful relationship with a source sentence given. Since the proposed CS-LVM framework is a general and integrated method of modeling sentence pairs, we expect that it could be adopted in various cases for better modeling of sentence pairs.

As comparing two sentences is an important ingredient for building various natural language understanding systems, we believe that the methods discussed in this dissertation would improve the effectiveness of numerous applications. We also want to note again that the three types of improvements do not overlap with each other. That is, advancements in one direction does not prevent the entire model from applying methods in other directions; for example we can use one or more methods proposed in this dissertation at the same time for better performance. We can also integrate recently proposed sophisticated sentence encoders e.g. Transformer or pretrained language models like ELMo and BERT together with our matching function or semi-supervised training framework.

Appendix A

Appendix

A.1 Sentences Generated from CS-LVM

Generated sentences are presented in Tables A.1–A.4. Though almost all generated hypotheses are realistic, we see that they lack diversity and fail to encode label information in some cases. For example, the phrase 'is/are sleeping' appears in generated sentences frequently when conditioned on the 'contradiction' label, likely because generating a set of simple patterns could be a shortcut to the objective. In Table 6.5, we verified from experiments that adding constraints helps enhancing accuracy and diversity, however a model is still relatively in favor of generating 'easy' sentences. We conjecture that the problem has its root in the fact that the primary objective of our model is to correctly classify the input, not to generate diverse outputs.

Input	**Entailment**	**Neutral**	**Contradiction**
little kids enjoy sprinklers by running through them outdoors .	kids are running .	the children are siblings .	the children are playing video games .
blurry people walking in the city at night .	people are walking outside .	the people are going to work .	the people are inside .
a woman sits in a chair under a tree and plays an acoustic guitar .	a woman is playing an instrument .	the woman is a musician .	a woman is playing the flute .
three men converse in a crowd .	three men are talking .	~~the men are talking .~~	the men are sleeping .
a woman in a yellow shirt seated at a table .	a woman is sitting .	~~a woman is sitting at a table .~~	the woman is standing .
a woman hugs a fluffy white dog .	a woman is holding a dog .	~~a woman is with her dog .~~	a woman is sleeping .
a crowd of people in colorful dresses .	people in costumes	the people are in a parade .	~~the people are sitting in a circle .~~
a clown making a balloon animal for a pretty lady .	~~a clown is entertaining a crowd .~~	the clown is entertaining a crowd .	the clown is sleeping .

Table A.1: Sentences generated from the CS-LVM model trained on the SNLI dataset. Failure cases are denoted by ~~strikethrough~~ text.

Input	**Entailment**	**Neutral**	**Contradiction**
little kids enjoy sprinklers by running through them outdoors .	kids are playing outside .	the kids are playing in the water .	the kids are sleeping .
blurry people walking in the city at night .	people are walking .	the people are walking to work .	the people are inside .
a woman sits in a chair under a tree and plays an acoustic guitar .	a woman is playing music .	the woman is a musician .	a woman is sleeping .
three men converse in a crowd .	three men are talking .	three men are talking about politics .	the men are sleeping .
a woman in a yellow shirt seated at a table .	a woman is sitting .	a tall human sitting .	the woman is standing .
a woman hugs a fluffy white dog .	a woman is holding a dog .	the dog belongs to the woman .	the dog is black .
a crowd of people in colorful dresses .	people in costumes	the people are in a parade .	the people are sleeping .
a clown making a balloon animal for a pretty lady .	a clown is performing .	the clown is entertaining a crowd .	the clown is sleeping .

Table A.2: Sentences generated from the CS-LVM + $\mathcal{R}^y$ model trained on the SNLI dataset. Note that failed examples in Table A.1 are corrected due to the use of $\mathcal{R}^y$.

Input	**Entailment**	**Neutral**	**Contradiction**
little kids enjoy sprinklers by running through them outdoors .	~~kids are playing in water .~~	the kids are having fun .	the kids are sleeping .
blurry people walking in the city at night .	people are walking .	the people are walking to work .	the people are inside .
a woman sits in a chair under a tree and plays an acoustic guitar .	a woman is playing an instrument .	the woman is a musician .	a woman is playing the drums .
three men converse in a crowd .	three men are talking .	three men are talking about politics .	the men are sleeping .
a woman in a yellow shirt seated at a table .	a woman is sitting .	~~a woman is sitting at a table .~~	the woman is standing
a woman hugs a fluffy white dog .	a woman is holding a dog .	a woman is playing with her dog .	a woman is sleeping .
a crowd of people in colorful dresses .	people are wearing costumes .	the people are in a parade .	~~the people are sitting down .~~
a clown making a balloon animal for a pretty lady .	a clown performs .	~~the clown is a clown .~~	the clown is sleeping .

Table A.3: Sentences generated from the CS-LVM + $\mathcal{R}^{\mathbf{z}}$ model trained on the SNLI dataset. Failure cases are denoted by ~~strikethrough~~ text.

Input	**Entailment**	**Neutral**	**Contradiction**
little kids enjoy sprinklers by running through them outdoors .	kids are playing outside .	the kids are having fun .	the kids are sleeping .
blurry people walking in the city at night .	people are walking .	the people are walking to work .	the people are inside .
a woman sits in a chair under a tree and plays an acoustic guitar .	a woman is playing an instrument .	the woman is a musician .	a woman is playing the piano .
three men converse in a crowd .	three men are talking .	three men are talking about politics .	the men are sleeping .
a woman in a yellow shirt seated at a table .	a woman is sitting .	~~a woman is sitting at a table .~~	the woman is standing
a woman hugs a fluffy white dog .	a woman is holding a dog .	the dog belongs to the woman .	a woman is petting a cat .
a crowd of people in colorful dresses .	people are dressed up .	the people are in a parade .	~~the people are sitting down .~~
a clown making a balloon animal for a pretty lady .	~~a clown is blowing bubbles .~~	~~the clown is a clown .~~	the clown is sleeping .

Table A.4: Sentences generated from the CS-LVM + $\mathcal{R}^{\mu}$ model trained on the SNLI dataset. Failure cases are denoted by ~~strikethrough~~ text.

Bibliography

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283.

Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Devi Parikh, and Dhruv Batra. 2017. VQA: Visual question answering. *International Journal of Computer Vision*, 123(1):4–31.

Alexander Alemi, Ben Poole, Ian Fischer, Joshua Dillon, Rif A. Saurous, and Kevin Murphy. 2018. Fixing a broken ELBO. In *Proceedings of the 35th International Conference on Machine Learning*, pages 159–168.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations.*

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247.

Hedi Ben-younes, Remi Cadene, Matthieu Cord, and Nicolas Thome. 2017. MUTAN: Multimodal tucker fusion for visual question answering. In *Proceedings of 2017 IEEE International Conference on Computer Vision*, pages 2612–2620.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *Computing Research Repository*, arXiv:1308.3432. Version 1.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016a. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1466–1477.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. 2016b. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21.

Siddhartha Brahma. 2018. REGMAPR - a recipe for textual matching. *Computing Research Repository*, arXiv:1808.04343. Verxion 1.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th IWSLT evaluation campaign. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 2–17.

Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. 2010. *Semi-Supervised Learning*, 1st edition. The MIT Press.

Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer.

Qian Chen, Zhen-Hua Ling, and Xiaodan Zhu. 2018a. Enhancing sentence embedding with generalized pooling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1815–1826.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Diana Inkpen, and Si Wei. 2018b. Neural natural language inference models enhanced with external knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2406–2417.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017a. Enhanced LSTM for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1657–1668.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017b. Recurrent neural network-based sentence encoder with gated attention for natural language inference. In *Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP*, pages 36–40.

Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2017c. Variational lossy autoencoder. In *International Conference on Learning Representations*.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Shiyu Wu, and Xuanjing Huang. 2015. Sentence modeling with gated recursive neural network. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 793–798.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014c. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Jihun Choi, Taeuk Kim, and Sang-goo Lee. 2018a. Element-wise bilinear interaction for sentence matching. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 107–112.

Jihun Choi, Taeuk Kim, and Sang-goo Lee. 2019a. Cell-aware stacked LSTMs for modeling sentences. In *Proceedings of The Eleventh Asian Conference on Machine Learning*, pages 1172–1187.

Jihun Choi, Taeuk Kim, and Sang-goo Lee. 2019b. A cross-sentence latent variable model for semi-supervised text sequence matching. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4747–4761.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. 2018b. Learning to compose task-specific tree structures. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5094–5101.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In *International Conference on Learning Representations*.

John Cocke and Jacob T. Schwartz. 1970. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Science.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680.

George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems 28*, pages 3079–3087.

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. 2018. Hyperspherical variational auto-encoders. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 856–865.

Michel Deudon. 2018. Learning semantic similarity in a continuous space. In *Advances in Neural Information Processing Systems 31*, pages 986–997.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *Computing Research Repository*, arXiv:1810.04805. Version 1.

Adji B. Dieng, Yoon Kim, Alexander M. Rush, and David M. Blei. 2019. Avoiding latent variable collapse with generative skip models. In *Proceedings of Machine Learning Research*, pages 2397–2405.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing*, pages 9–16.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2014. Adaptive multi-compositionality for recursive neural models with applications to sentiment analysis. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1537–1543.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209.

Salah El Hihi and Yoshua Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems 8*, pages 493–499.

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

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. 2016. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 457–468.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1243–1252.

Reza Ghaeini, Sadid A. Hasan, Vivek Datla, Joey Liu, Kathy Lee, Ashequl Qadir, Yuan Ling, Aaditya Prakash, Xiaoli Fern, and Oladimeji Farri. 2018. DR-BiLSTM: Dependent reading bidirectional LSTM for natural language inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1460–1469.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks*, pages 347–352.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680.

Alex Graves. 2012. Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 5–13.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *Computing Research Repository*, arXiv:1308.0850. Version 5.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2007. Multi-dimensional recurrent neural networks. In *Proceedings of the 17th International Conference on Artificial Neural Networks*, pages 549–558.

Alex Graves and Jürgen Schmidhuber. 2008. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems 21*, pages 545–552.

Edward Grefenstette and Mehrnoosh Sadrzadeh. 2011. Experimental support for a categorical compositional distributional model of meaning. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1394–1404.

Jiatao Gu, Daniel Jiwoong Im, and Victor O. K. Li. 2018. Neural machine translation with Gumbel-Greedy decoding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5125–5132.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. 2017. PixelVAE: A latent variable model for natural images. In *International Conference on Learning Representations*.

Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. 2018. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450.

Kazuma Hashimoto, Makoto Miwa, Yoshimasa Tsuruoka, and Takashi Chikayama. 2013. Simple customization of recursive neural networks for semantic relation classification. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1372–1376.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of 2015 IEEE International Conference on Computer Vision*, pages 1026–1034.

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

Michiel Hermans and Benjamin Schrauwen. 2013. Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems 26*, pages 190–198.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377.

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

Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

Phu Mon Htut, Kyunghyun Cho, and Samuel Bowman. 2018. Grammar induction with neural language models: An unusual replication. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4998–5003.

Minghao Hu, Yuxing Peng, Zhen Huang, Xipeng Qiu, Furu Wei, and Ming Zhou. 2018. Reinforced mnemonic reader for machine reading comprehension. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 4099–4106.

Minlie Huang, Qiao Qian, and Xiaoyan Zhu. 2017. Encoding syntactic knowledge in neural networks for sentiment classification. *ACM Transactions on Information Systems*, 35(3):26:1–26:27.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *International Conference on Learning Representations*.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456.

Ozan Irsoy and Claire Cardie. 2014. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems 27*, pages 2096–2104.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

Yangfeng Ji and Jacob Eisenstein. 2013. Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 891–896.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. 2016. Grid long short-term memory. In *International Conference on Learning Representations*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. 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.

Tadao Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report 65-758, Air Force Cambridge Research Laboratory.

Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491.

Jaeyoung Kim, Mostafa El-Khamy, and Jungwon Lee. 2017a. Residual LSTM: Design of a deep recurrent architecture for distant speech recognition. In *18th Annual Conference of the International Speech Communication Association*, pages 1591–1595.

Jin-Hwa Kim, Kyoung-Woon On, Woosang Lim, Jeonghee Kim, Jung-Woo Ha, and Byoung-Tak Zhang. 2017b. Hadamard product for low-rank bilinear pooling. In *International Conference on Learning Representations*.

Seonhoon Kim, Inho Kang, and Nojun Kwak. 2019a. Semantic sentence matching with densely-connected recurrent and co-attentive information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6586–6593.

Taeuk Kim, Jihun Choi, Daniel Edmiston, Sanghwan Bae, and Sang-goo Lee. 2019b. Dynamic compositionality in recursive neural networks with structure-aware tag representations. In *Thirty-Third AAAI Conference on Artificial Intelligence*, pages 6594–6601.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017c. Structured attention networks. In *International Conference on Learning Representations*.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *International Conference on Learning Representations*.

Durk P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27*, pages 3581–3589.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in Neural Information Processing Systems 28*, pages 3294–3302.

Ben Krause, Liang Lu, Iain Murray, and Steve Renals. 2016. Multiplicative LSTM for sequence modelling. *Computing Research Repository*, arXiv:1609.07959. Version 3.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1378–1387.

Wuwei Lan and Wei Xu. 2018. Neural network models for paraphrase identification, semantic textual similarity, natural language inference, and question answering. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3890–3902.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 10–19.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems 27*, pages 2177–2185.

Bowen Li, Lili Mou, and Frank Keller. 2019. An imitation learning approach to unsupervised parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3485–3492.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119.

Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. 2015. Bilinear CNN models for fine-grained visual recognition. In *Proceedings of 2015 IEEE International Conference on Computer Vision*.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Dynamic compositional neural networks over tree structure. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4054–4060.

Xiaodong Liu, Kevin Duh, and Jianfeng Gao. 2018. Stochastic answer networks for natural language inference. *Computing Research Repository*, arXiv:1804.07888. Version 2.

Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. 2016. Learning natural language inference using bidirectional LSTM model and inner-attention. *Computing Research Repository*, arXiv:1605.09090. Version 1.

Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. 2017. Deep learning with dynamic computation graphs. In *International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*.

Chris J Maddison, Daniel Tarlow, and Tom Minka. 2014. A* sampling. In *Advances in Neural Information Processing Systems 27*, pages 3086–3094.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2017. Jointly learning sentence embeddings and syntax with unsupervised Tree-LSTMs. *Computing Research Repository*, arXiv:1705.09189. Version 1.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2019. Jointly learning sentence embeddings and syntax with unsupervised Tree-LSTMs. *Natural Language Engineering*, 25(4):433–449.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. 2014. SemEval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, pages 1–8.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems 30*, pages 6294–6305.

Yishu Miao, Lei Yu, and Phil Blunsom. 2016. Neural variational inference for text processing. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1727–1736.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *11th Annual*

Conference of the International Speech Communication Association, pages 1045–1048.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429.

Bhaskar Mitra and Nick Craswell. 2017. Neural models for information retrieval. *Computing Research Repository*, arXiv:1705.01509. Version 1.

Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1291–1299.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. 2016. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 130–136.

Tsendsuren Munkhdalai and Hong Yu. 2017a. Neural semantic encoders. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 397–407.

Tsendsuren Munkhdalai and Hong Yu. 2017b. Neural tree indexers for text understanding. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 11–21.

Yixin Nie and Mohit Bansal. 2017. Shortcut-stacked sentence encoders for multidomain inference. In *Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP*, pages 41–45.

Kamal Nigam, Andrew McCallum, and Tom Mitchell. 2006. Semi-supervised text classification using EM. *Semi-Supervised Learning*, pages 33–56.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems 30*, pages 6306–6315.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. How to construct deep recurrent neural networks. In *International Conference on Learning Representations*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the*

Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237.

Tony A. Plate. 1995. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3):623–641.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163.

Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. 2016. Variational autoencoder for deep learning of images, labels and captions. In *Advances in Neural Information Processing Systems 29*, pages 2352–2360.

Qiao Qian, Bo Tian, Minlie Huang, Yang Liu, Xuan Zhu, and Xiaoyan Zhu. 2015. Learning tag embeddings and tag-specific composition functions in recursive neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1365–1374.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to generate reviews and discovering sentiment. *Computing Research Repository*, arXiv:1704.01444. Version 2.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. Technical report, OpenAI.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016*

Conference on Empirical Methods in Natural Language Processing, pages 2383–2392.

Prajit Ramachandran, Peter Liu, and Quoc Le. 2017. Unsupervised pretraining for sequence to sequence learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 383–391.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1278–1286.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. 2018. A hierarchical latent vector model for learning long-term structure in music. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4364–4373.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. 2014. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations.*

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2017. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3295–3301.

Lei Sha, Baobao Chang, Zhifang Sui, and Sujian Li. 2016. Reading and thinking: Re-read LSTM unit for textual entailment recognition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2870–2879.

Dinghan Shen, Yizhe Zhang, Ricardo Henao, Qinliang Su, and Lawrence Carin. 2018a. Deconvolutional latent-variable model for text sequence matching. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5438–5445.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2018b. DiSAN: Directional self-attention network for RNN/CNN-free language understanding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5446–5455.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Sen Wang, and Chengqi Zhang. 2018c. Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 4345–4352.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems 30*, pages 6830–6841.

Haoyue Shi, Hao Zhou, Jiaze Chen, and Lei Li. 2018. On tree-based neural sentence modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4631–4641.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012.

Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. 2011a. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 129–136.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011b. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 151–161.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. 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.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems 28*, pages 3483–3491.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Ilya Sutskever. 2013. *Training recurrent neural networks*. Ph.D. thesis, University of Toronto.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566.

Ming Tan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2016. Improved representation learning for question answer matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 464–473.

Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432.

Zhiyang Teng and Yue Zhang. 2017. Head-lexicalized bidirectional tree LSTMs. *Transactions of the Association for Computational Linguistics*, 5:163–177.

Quan Tran, Andrew MacKinlay, and Antonio Jimeno Yepes. 2017. Named entity recognition with stack residual LSTM and trainable bias decoding. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 566–575.

Kateryna Tymoshenko and Alessandro Moschitti. 2018. Cross-pair text representations for answer sentence selection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2162–2173.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N

Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

Yizhong Wang, Sujian Li, Jingfeng Yang, Xu Sun, and Houfeng Wang. 2017a. Tag-enhanced tree-structured neural networks for implicit discourse relation classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 496–505.

Zhiguo Wang, Wael Hamza, and Radu Florian. 2017b. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4144–4150.

Adina Williams, Andrew Drozdov, and Samuel R. Bowman. 2018a. Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association for Computational Linguistics*, 6:253–267.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018b. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306.

Wei Wu, Zhengdong Lu, and Hang Li. 2013. Learning bilinear model for matching queries and documents. *Journal of Machine Learning Research*, 14:2519–2548.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *Computing Research Repository*, arXiv:1609.08144. Version 2.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan R Salakhutdinov. 2016. On multiplicative integration with recurrent neural networks. In *Advances in Neural Information Processing Systems 29*, pages 2856–2864.

Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic memory networks for visual and textual question answering. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 2397–2406.

Caiming Xiong, Victor Zhong, and Richard Socher. 2018. DCN+: Mixed objective and deep residual coattention for question answering. In *International Conference on Learning Representations*, pages 770–778.

Jiacheng Xu and Greg Durrett. 2018. Spherical latent spaces for stable variational autoencoders. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4503–4513.

Wei Xu, Alan Ritter, Chris Callison-Burch, William B. Dolan, and Yangfeng Ji. 2014. Extracting lexically divergent paraphrases from Twitter. *Transactions of the Association for Computational Linguistics*, 2:435–448.

Weidi Xu, Haoze Sun, Chao Deng, and Ying Tan. 2017. Variational autoencoder for semi-supervised text classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3358–3364.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018.

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved variational autoencoders for text modeling using dilated convolutions. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3881–3890.

Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. 2015. Depth-gated LSTM. *Computing Research Repository*, arXiv:1508.03790. Version 4.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2017. Learning to compose words into sentences with reinforcement learning. In *International Conference on Learning Representations*.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Zhou Yu, Jun Yu, Jianping Fan, and Dacheng Tao. 2017. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *Proceedings of 2017 IEEE International Conference on Computer Vision*, pages 1821–1830.

Fabio Massimo Zanzotto and Lorenzo Dell'Arciprete. 2012. Distributed tree kernels. In *Proceedings of the 29th International Conference on Machine Learning*.

Matthew D. Zeiler. 2012. ADADELTA: An adaptive learning rate method. *Computing Research Repository*, arXiv:1212.5701. Version 1.

Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. 2016a. Variational neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 521–530.

Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. 2016b. Highway long short-term memory RNNs for distant speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5755–5759.

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence model. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 4069–4076.

Junbo Zhao, Yoon Kim, Kelly Zhang, Alexander Rush, and Yann LeCun. 2018. Adversarially regularized autoencoders. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5902–5911.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. 2015. A C-LSTM neural network for text classification. *Computing Research Repository*, arXiv:1511.08630. Version 2.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1604–1612.

Xiaojin Zhu. 2008. Semi-supervised learning literature survey. Technical report, Department of Computer Sciences, University of Wisconsin-Madison.

Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, pages 912–919.

초록

문장 매칭이란 두 문장 간 의미적으로 일치하는 정도를 예측하는 문제이다. 어떤 모델이 두 문장 사이의 관계를 효과적으로 밝혀내기 위해서는 높은 수준의 자연어 텍스트 이해 능력이 필요하기 때문에, 문장 매칭은 다양한 자연어 처리 응용의 성능에 중요한 영향을 미친다.

본 학위 논문에서는 문장 인코더, 매칭 함수, 준지도 학습이라는 세 가지 측면에서 문장 매칭의 성능 개선을 모색한다. 문장 인코더란 문장으로부터 유용한 특질들을 추출하는 역할을 하는 구성 요소로, 본 논문에서는 문장 인코더의 성능 향상을 위하여 Gumbel Tree-LSTM과 Cell-aware Stacked LSTM이라는 두 개의 새로운 아키텍처를 제안한다. Gumbel Tree-LSTM은 재귀적 뉴럴 네트워크(recursive neural network) 구조에 기반한 아키텍처이다. 구조 정보가 포함된 데이터를 입력으로 사용하던 기존의 재귀적 뉴럴 네트워크 모델과 달리, Gumbel Tree-LSTM은 구조가 없는 데이터로부터 특정 문제에 대한 성능을 최대화하는 파싱 전략을 학습한다. Cell-aware Stacked LSTM은 LSTM 구조를 개선한 아키텍처로, 여러 LSTM 레이어를 중첩하여 사용할 때 망각 게이트(forget gate)를 추가적으로 도입하여 수직 방향의 정보 흐름을 더 효율적으로 제어할 수 있도록 한다.

한편, 새로운 매칭 함수로서 우리는 요소별 쌍선형 문장 매칭(element-wise bilinear sentence matching, ElBiS) 함수를 제안한다. ElBiS 알고리즘은 특정 문제를 해결하는 데에 적합한 방식으로 두 문장 표현을 하나의 벡터로 합치는 방법을 자동으로 찾는 것을 목적으로 한다. 문장 표현을 얻을 때에 서로 같은 문장 인코더를 사용한다는 사실로부터 우리는 벡터의 각 요소 간 쌍선형(bilinear) 상호 작용만을 고려하여도 두 문장 벡터 간 비교를 충분히 잘 수행할 수 있다는 가설을 수립하고 이를 실험적으로 검증한다. 상호 작용의 범위를 제한함으로써, 자동으로 유용한 병합 방법을 찾는다는 이점을 유지하면서 모든 상호 작용을 고려하는 쌍선형 풀링 방법에 비해 필요한 파라미터의 수를 크게 줄일

수 있다.

마지막으로, 학습 시 레이블이 있는 데이터와 레이블이 없는 데이터를 함께 사용하는 준지도 학습을 위해 우리는 교차 문장 잠재 변수 모델(cross-sentence latent variable model, CS-LVM)을 제안한다. CS-LVM의 생성 모델은 출처 문장(source sentence)의 잠재 표현 및 출처 문장과 목표 문장(target sentence) 간의 관계를 나타내는 변수로부터 목표 문장이 생성된다고 가정한다. CS-LVM에서는 두 문장이 하나의 모델 안에서 모두 고려되기 때문에, 학습에 사용되는 목적 함수가 더 자연스럽게 정의된다. 또한, 우리는 생성 모델의 파라미터가 더 의미적으로 적합한 문장을 생성하도록 유도하기 위하여 일련의 의미 제약들을 정의한다.

본 학위 논문에서 제안된 개선 방안들은 문장 매칭 과정을 포함하는 다양한 자연어 처리 응용의 효용성을 높일 것으로 기대된다.

주요어: 문장 매칭, 문장 간 관계 모델링, 문장 인코더, 매칭 함수, 준지도 학습, 심층 신경망, 심층 생성 모델

학번: 2014-22694