4-23-2019 2:30 PM

Improving Neural Sequence Labelling Using Additional Linguistic Information

Muhammad Rifayat Samee, *The University of Western Ontario*

Supervisor: Mercer, Robert E., *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Acknowledgement

I would like to express my sincere gratitude to my thesis supervisor Dr. Robert E. Mercer. Without his continuous guidance and significant contributions this work would not have been possible. Special thanks to my research colleague Mahtab Ahmed for his help and contributions to this work.

I would like to thank the Department of Computer Science of The University of Western Ontario for providing me with the Graduate Research Scholarship.

Last but not the least, I would like to thank my parents for believing in me and for providing me with continuous support and encouragement throughout my graduate study.

Abstract

Sequence Labelling is the task of mapping sequential data from one domain to another domain. As we can interpret language as a sequence of words, sequence labelling is very common in the field of Natural Language Processing (NLP). In NLP, some fundamental sequence labelling tasks are Parts-of-Speech Tagging, Named Entity Recognition, Chunking, etc. Moreover, many NLP tasks can be modeled as sequence labelling or sequence to sequence labelling such as machine translation, information retrieval and question answering. An extensive amount of research has already been performed on sequence labelling. Most of the current high performing models are neural network models. These Deep Learning based models are outperforming traditional machine learning techniques by using abstract high dimensional feature representations of the input data. In this thesis, we propose a new neural sequence model which uses several additional types of linguistic information to improve the model performance. The convergence rate of the proposed model is significantly less than similar models. Moreover, our model obtains state of the art results on the benchmark datasets of POS, NER, and chunking.

Keywords: Sequence Labelling, Neural Network, Deep Learning, NLP

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In Natural Language Processing, often we need to find a function to map sequence pairs from different domains. An example of such sequence pairs is a sequence of words(i.e, a sentence) and a sequence of parts-of-speech. This is the well-known parts-of-speech (POS) tagging problem. Formally, given an input sequence $X = x_1, x_2, ..., x_n$ where each x_i is from a domain α and a corresponding tagging/state sequence $Y = y_1, y_2, ..., y_n$ where each y_i is from a domain γ, our goal is to find a functional mapping $f : X \mapsto Y$. For example, in the POS tagging problem one possible input sequence might be *"France won the Match"*. The corresponding POS tagged sequence will be *N V D N*. Here *N, V,* and *D* represent tags for *Noun, Verb,*and *Determiner,* respectively. In this example, our function f maps word sequences to their possible parts-of-speech sequences.

Sequence Labelling can be treated as a supervised machine learning task. Given m training examples $(x^{(i)}, y^{(i)})$ for $i = 1 \ldots m$ where each $x^{(i)}$ is a sentence $x_1^{(i)} \ldots x_{n_i}^{(i)}$ of length n_i, and each $y^{(i)}$ is a tag sequence $y_1^{(i)} \ldots y_{n_i}^{(i)}$ of length n_i, our task is to train a model to correctly predict the tagged sequence Y for an unknown input sentence X. In this chapter we briefly introduce some fundamental sequence labelling problems. Moreover, we give an introduction to our approach and will give the outline of this thesis.

1.1 Some Sequence Labelling Problems in NLP

1.1.1 Parts-of-Speech (POS) Tagging

Parts-of-speech (POS) Tagging is probably the oldest and most famous sequence labelling problem in the field of Natural Language Processing. In this problem the goal is to map each word of a sentence to its correct POS. Extensive research has been performed on POS tagging. Among all the sequence labelling problems, POS tagging is treated as an almost solved problem with state of the art accuracy being between 97% and 98% [24]. In 2011, C. Manning [26] argued that, to go from 97% to 100%, we need to explore more linguistic features. His idea is reflected in the recent deep learning architectures where along with features which are implicitly extracted by Neural Networks, researchers are incorporating carefully designed linguistic features. Figure 1.1 shows an example of POS tagging. The input sequence is "cats can not fly" and the corresponding POS tags are NNS (plural common noun), MD (modal auxiliary), RB (adverb) and VB (verb).

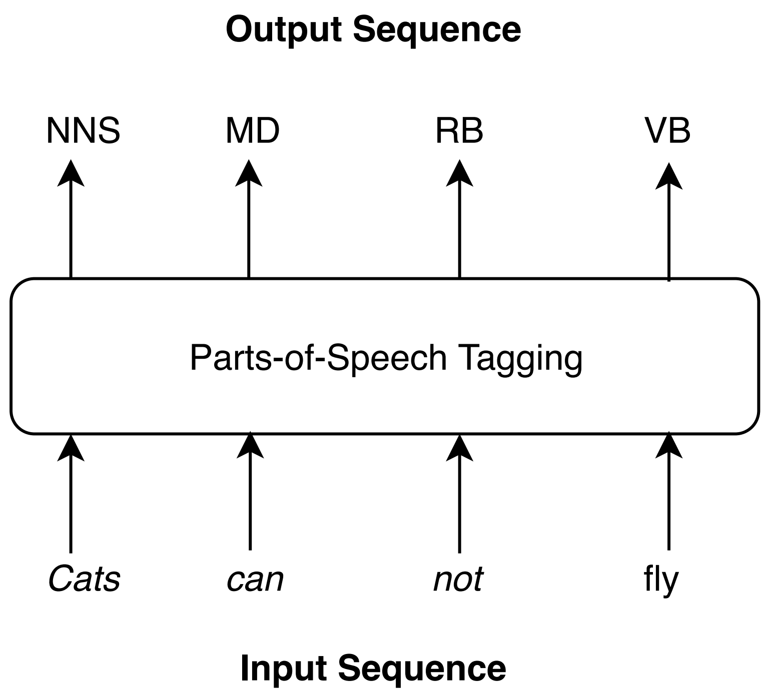


Figure 1.1: Example of POS tagging

1.1.2 Named Entity Recognition (NER)

Named Entity Recognition is another sequence labelling problem: the input sequence is a sentence and the output sequence is the sentence tagged with possible named entities. Some common named entities are persons, organizations, locations and names. Named entity recognition is challenging because a large amount of feature and lexicon knowledge is required to achieve high performance. Most of the high performing models for the NER task are neural network models. Current state of the art F1 score (4.1.3) for English NER task is around 91%. Figure 1.2 shows a simple example of NER task. The input sentence is *"Federer won ITF US open in New York"*. And the NER model identifies Federer as a *"PERSON"*, *ITF* as *"ORGANIZATION"* and "New York" as "LOCATION".

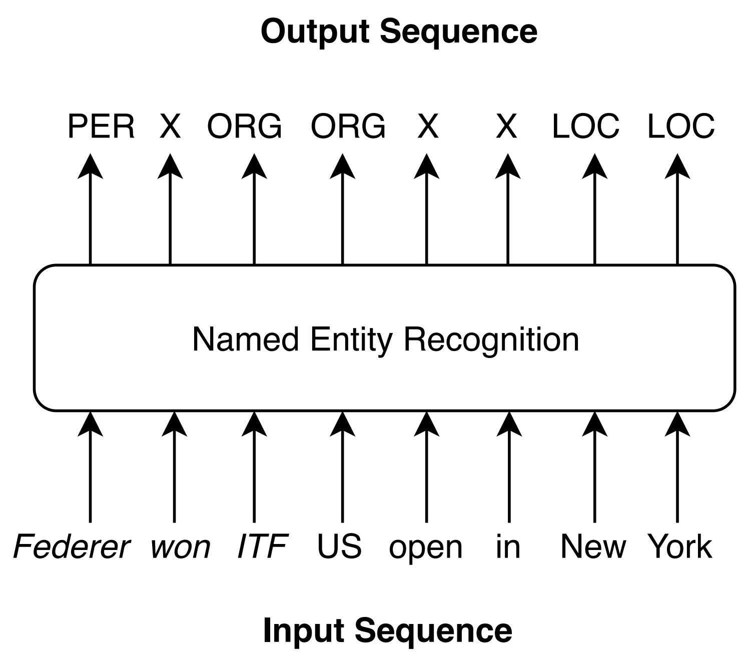


Figure 1.2: Example of NER tagging

1.1.3 Chunking (Shallow Parsing)

Chunking, also known as shallow parsing, is the task of dividing texts to syntactically correlated sequence of words. For example: *"The morning flight from Denver has arrived."* can be divided as *"[NP The morning flight] [PP from] [NP Denver] [VP has arrived]"*. Here the syntactic constructs are noun phrase (NP), verb phrase (VP) and preposition (PP). This shallow parsing can be an intermediate step before parsing a sentence. Chunking can be designed as a sequence labelling task where each word of the same chunk will be tagged with the same label. Unlike POS tagging and NER, for Chunking, most of high performing models are not neural models. The Best F1-score of 95.23% is achieved by Shen et al. [46] using a Hidden Markov Model with a voting scheme.

1.2 Our Approach

In this thesis, our goal is to improve neural sequence labelling by using additional linguistic features. We propose a new neural architecture for the Sequence Labelling problem which is capable of using several linguistic features to improve tagging accuracy. In our approach we used sense embedding along with word level knowledge to capture semantic features. Moreover, to capture morphological features we used character level embeddings for words. Additionally, we used selective pickup from a character level Long Short Term Memory Unit

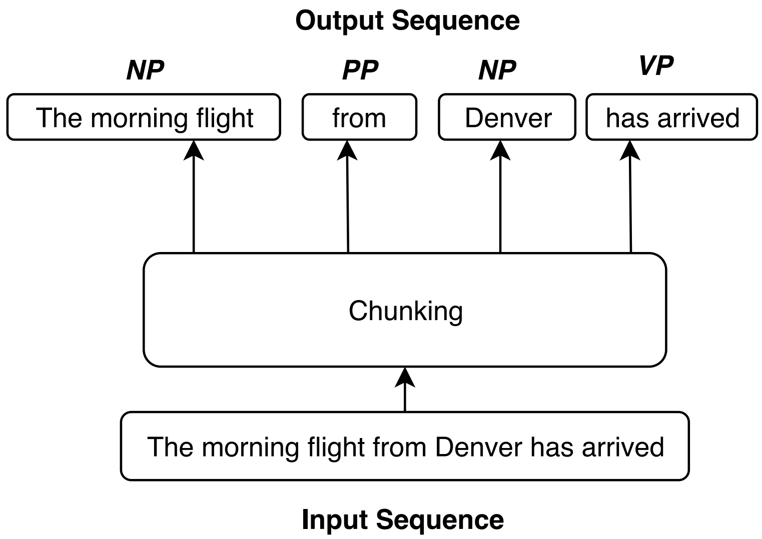


Figure 1.3: Example of Chunking

(LSTM)[16] to capture contextual and morphological features simultaneously. We used a Convolutional Neural Network (CNN) to exploit bigram(two adjacent words) information. Several hand crafted spelling features (prefix, suffix, patterns) were used with the implicit neural feature extraction done by the Deep Neural Learning. To connect all the features appropriately we used three connection methods: Residual connection, Concatenate with word embedding and Concatenate with second last output layer. Our architecture achieved state of the art results on benchmark datasets of POS tagging, NER and Chunking without using any additional data or multi-task learning.

1.3 Thesis Outline

This thesis is organized in four chapters. In the second chapter we summarize previous work done in sequence labelling using Deep Neural Networks. We discuss the main components of a typical deep sequence labelling model. Chapter 3 details our proposed model. For each of the modules of our model, we talk about the reason behind using the module and the detail functionality of the module. Chapter 4 is dedicated to experimental results. In this chapter, we show our experimental setup and discuss our experimental findings. We compare our results with several state of the art models on different benchmark data sets for POS, NER and the Chunking task. Moreover, we do an ablation study of our model to see the impact and usefulness of different modules of our architecture. We also discuss feature connection methods used by our architecture for connecting features optimally. Chapter 5 concludes the thesis.

Chapter 2

Related Work

In this chapter we summarize previous work done on the sequence labelling task. First we give an overview of the previous work for Parts-of-speech (POS) tagging using traditional approaches. Then we present a summary of research work using Deep Neural Architectures for sequence labelling tasks. Then we introduce some well-known and commonly used architectures and techniques for Deep Sequence Labelling Networks. Finally, we compare several Deep learning based models on benchmark datasets for the sequence labelling task.

2.1 Previous Work

2.1.1 Statistical Sequence Labelling Models

Before the advancement of Deep Learning, most of the high performing models for sequence labelling were Statistical in nature. Models were based on Hidden Markov Models (HMMs) [12, 59] and Conditional Random Fields (CRFs) [21, 33]. For example, Weischedel et al. [55] introduced a generic n-gram probabilistic model for sequence labelling. They experimented using tri-gram version of their model on the WSJ corpus for POS tagging, achieving 96.3% accuracy. Their experimental result showed that probabilistic models are capable of extracting lexical features which can replace rule based systems. Even though this paper used several features to deal with the sparsity problem, the paper did not talk about clear ways to incorporate features to make the model's feature space more robust. The Maximum-Entropy Model [40] introduced a way to incorporate an arbitrary number of binary features into the probabilistic model. As expected, the Maximum-Entropy model achieved better accuracy than the tri-gram model on unknown words (85.6%) which leads to overall accuracy of 96.6%. The next improvement on overall accuracy came from a Hidden Markov model using second-order approximations. The model proposed by Scott and Harper [52] uses a 3d transition probability matrix which not only depends on the current state but also on the previous state. Moreover, they used a separate parameter for unknown words. The model got 96.9% accuracy on the WSJ corpus, which is an improvement over the previous models. However, the unknown word accuracy (84.9%) is significantly less than the Maximum-Entropy model. The reason behind it is that the Second-Order Markov model did not use any hand-crafted features. It only uses statistical approximations from the data. The experimental results of the three statistical approaches described above clearly indicate that, for supervised sequence labelling problems using only probabilistic parameter estimation is not enough. We need to exploit specialized features to boost model accuracy.

2.1.2 Deep Learning models for Sequence Labelling

In the past few years, Deep Neural Network architectures became very popular in the field of Natural Language Processing (NLP). Recent advancements in distributional word representation and Recurrent Neural Networks (RNN) made neural architectures superior to the traditional statistical machine learning techniques in almost every NLP tasks. All of the architectires and other techniques used in this and the next paragraph will be decribed in the next sections.

Huang et al. [17] propose a few models for the sequence tagging task. Apart from just word embeddings, they use morphological, bigram and trigram information as their input features. Later, they use Long Short Term Memory (LSTM) and Bidirectional LSTM (BLSTM) with Conditional Random Field (CRF) to do the final tagging. Lample et al. [22] extract character embeddings from both the left and right directions, concatenate these with word embeddings and use a stacked LSTM with CRF to do the tagging. Liu et al. [24] propose a model leveraging word and character level features. It includes a language model to represent the character level knowledge along with a highway layer to avoid the feature collision. Finally it is trained jointly as a multitask learning. Yu et al. [57] propose a general purpose tagger using a Convolutional Neural Network (CNN). First, they use CNNs to extract the character level features and then concatenate it with word embeddings, position embeddings and binary features. Finally they use another CNN to get the contextual features as well to do the tagging. Ma and Hovy [25] propose an end-to-end sequence labelling model using a combination of BLSTM, CNN and CRF. They use a CNN to get the character level information, concatenate it with word embeddings and then apply BLSTM to model the contextual information. Finally they generate the tags by using a sequential CRF layer. Rei [41] trains a language model type objective function using BLSTM-CRF to predict the surrounding words for every word in the corpus and utilizes it for sequence labelling.

In the next portion of this chapter, we will study commonly used architectures and techniques for a typical Deep Neural Sequence Labelling model. At the end of the chapter, we will compare different neural models on three sequence labelling tasks: POS tagging, NER and Chunking on benchmark datasets.

2.2 Recurrent Neural Network (RNN)

Traditional feedforward neural networks cannot capture dependencies in sequential data. For this reason, feedforward neural networks are not suitable for natural language where past context is important to predict the future. For example: let's consider predicting the word "*defeated*" in the sentence *"France won the World cup final 2018, so they defeated Croatia 2-1 in the final"*. To predict "*defeated*" we need to remember the inputs from earlier in the sentence. Being a bipartite graph, a feedforward neural network does not have the mechanism to address this issue. Recurrent Neural Networks (RNN) are specially designed to have internal memory to remember some important portion of the previous data to predict the future events. RNNs have loops that allow information to persist. Figure 2.1 shows a typical RNN structure. There, x_t is the input at time step t and the o_t is the corresponding output at time step t. W,V ,U are the parameters of an RNN. All the parameters are shared across time steps. So the output o_{t+1} is influenced by all the previous inputs $x_t, x_{t-1} \ldots x_1$ and the current input data x_{t+1}.

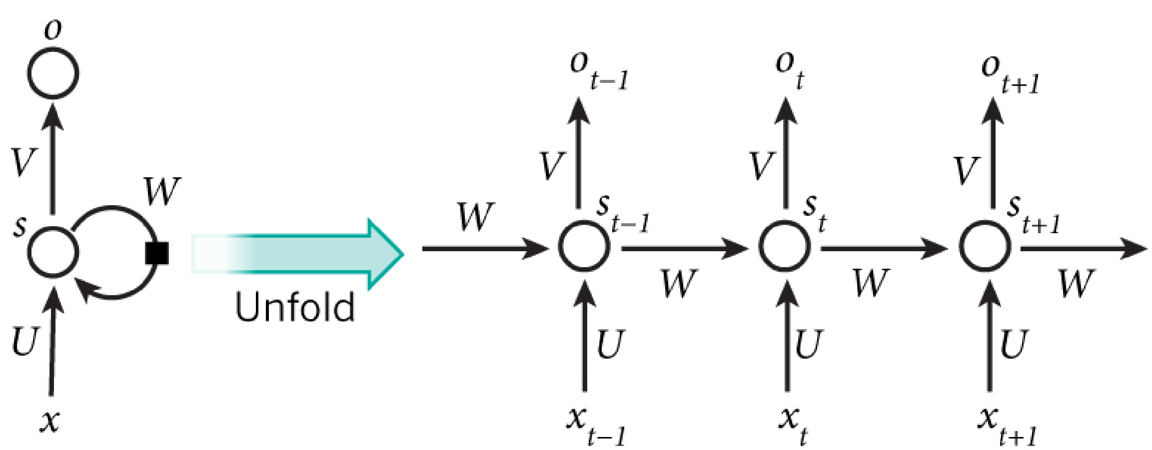


Figure 2.1: Recurrent Neural Network (RNN) [6]

Long Short Term Memory Unit (LSTM)

Theoretically, Recurrent Neural Networks are supposed to capture long distance dependencies in sequential data. However, in practice, RNNs face difficulty capturing dependencies because of the gradient vanishing/exploding problem [5] [35]. Long Short Time Memory Units (LSTM) are designed to capture long distance dependencies without succumbing to the gradient vanishing/exploding problem. LSTMs were first proposed by Hochreiter and Schmidhuber [16] in 1997. LSTMs have three multiplicative gates: input gate, forget gate and output gate. Each gate has a transformation matrix W_i, W_f and W_o respectively. All the gates are sigmoid gates so the output of the gates are between 0 and 1. Using the gates, an LSTM decides what information from the past time step should be remembered. Fig 2.2(a) shows the internal structure of an LSTM cell. C_{t-1} and h_{t-1} are the cell state and hidden state at time step $t-1$. First the forget gate decides what portion of the C_{t-1} we want to forget (eq 2.1) by using a sigmoid activation on the linearly transformed representation of the concatenation of the previous hidden state h_{t-1} and the current input vector x_t. In the equation, $[h_{t-1}, x_t]$ means the concatenation operation of two vectors: h_{t-1} and x_t. Then the input gate decides which new values we need to update by seeing the previous hidden state h_{t-1} and the current input x_t (eq 2.2). A candidate cell state $\bar{C}_t$ is calculated by a $tanh()$ layer (eq 2.3) (here $*$ is the point-wise multiplication function). Finally the new cell state is created using the outputs from the forget gate and the input gate (eq 2.4). Now to find the new hidden representation for the current time step t, LSTM uses the new cell state C_t and the output of the output gate o_t (eq2.5) (eq2.6).

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.1}$$

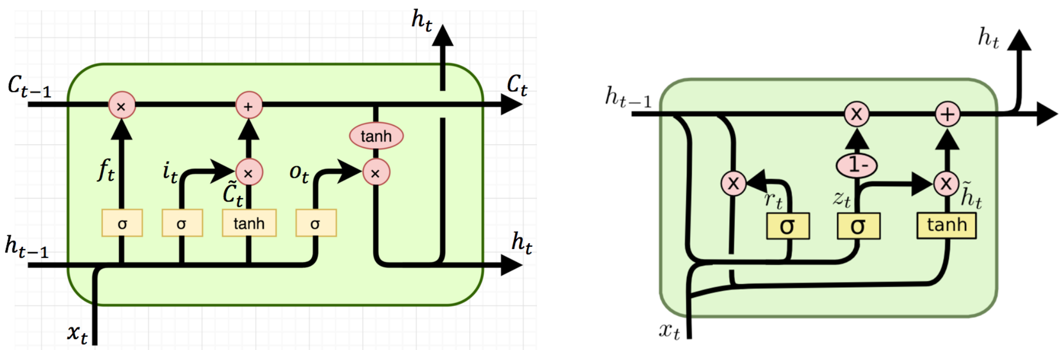
$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{2.2}$$

$$\bar{C}_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \tag{2.3}$$

$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t \tag{2.4}$$

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{2.5}$$

$$h_t = o_t * tanh(C_t) \tag{2.6}$$



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Figure 2.2: Long Short Term Memory Unit (LSTM) and Gated Recurrent Unit (GRU) [34]

Gated Recurrent Units (GRU)

A widely used and very popular variation of LSTM is Gated Recurrent Units (GRU). GRU was proposed by Cho, et al. [9] in 2014. In GRUs the forget gate and input gate are combined into a single update gate. So, GRUs have fewer parameters compared to LSTMs. For this reason, unlike LSTMs, GRUs are less prone to over-fitting. Fig 2.2 (b) shows the structure of a single GRU unit. Another major difference between LSTMs and GRUs is that the GRU structure does not include any cell state. Updating the hidden state of a GRU unit is done as following:

$$z_t = \sigma(W_z.[h_{t-1}, x_t]) \tag{2.7}$$

$$r_t = \sigma(W_r.[h_{t-1}, x_t]) \tag{2.8}$$

$$\bar{h}_t = tanh(W.[r_t * h_{t-1}, x_t]) \tag{2.9}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \bar{h}_t \tag{2.10}$$

2.3 Distributional Representation of Words

Representing words as continuous vectors in a vector space (word embeddings) has been extensively researched. Bengio et al. [4] first proposed a neural architecture to find vector representation of words. This model is mainly designed for estimating a neural network language model (NNLM). The model is a feedfoward neural network with a linear projection layer and a non-linear hidden layer. This idea of learning word representations using a neural network has been followed by many others.

2.3.1 Word2Vec

In 2013, Mikolov et al. [31] proposed a method to estimate high quality word vectors from huge datasets. They proposed two different models: Continuous Bag of Words Model (CBOW) and Continuous Skip-Gram model. Both models use a shallow neural network with one hidden layer. These models use a fixed length context window for each target word. In CBOW, the model predicts the target word from the context words. CBOW is similar to the feedforward NNLM model where the projection layer is shared for all the words. On the other hand, the Skip-Gram model predicts the context words from the target word. Figure 2.3 shows the model architectures. The result of the Skip-Gram model is very surprising. When the model is trained on a huge amount of data, the resulting vectors can capture subtle semantic relationships between words. Moreover, it is possible to do meaningful algebraic operations on the word vectors. For example: the resulting vector of the expression: $vector(\text{“}biggest\text{''}) - vector(\text{“}big\text{''}) + vector(\text{“}small\text{''})$ is very close to the $vector(\text{“}smallest\text{''})$. Here, distance is measured by the cosine similarity of the vectors. Using pretrained word2vec vectors, most of the neural NLP models get significant improvements in performance. Nowadays, it is common practice to use word vectors trained on extremely large amount of data for initializing embedding layers of neural models.

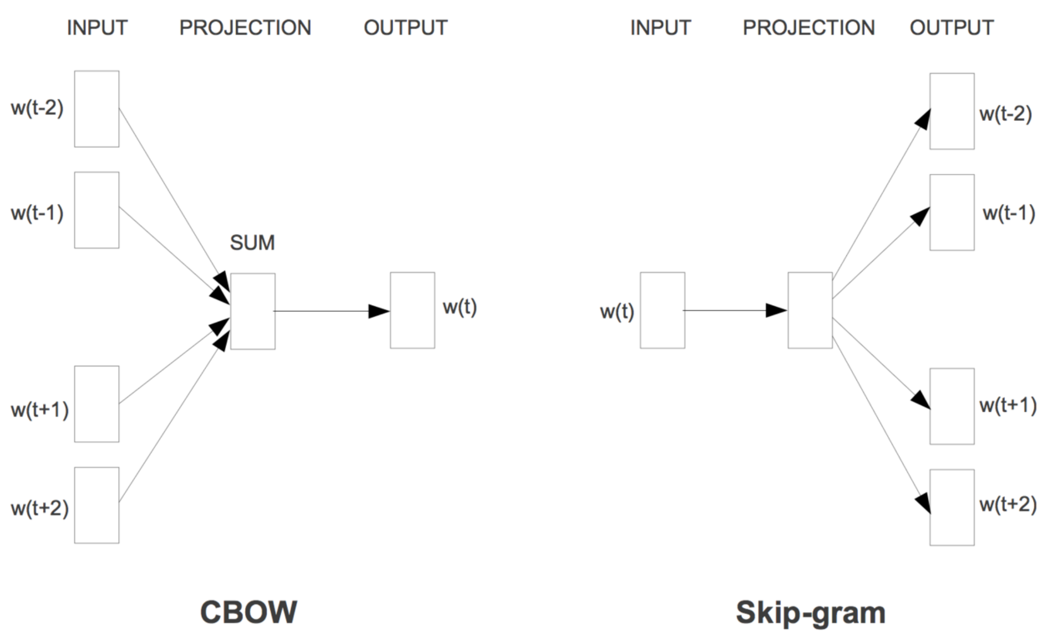


Figure 2.3: Word2Vec Models [31]

2.3.2 GloVe: Global Vectors for Word Representation

Two main ideas for learning word vectors are the global matrix factorization method, such as latent semantic analysis (LSA) [11] and the fixed local context window method, such as the Skip-Gram model [31]. The limitation of LSA and other global matrix factorization methods is that the similarity measures of the vectors are dominated by the frequent words. On the other hand, the Word2Vec Skip-gram model concentrate only on local context information and fails to leverage global statistical information. In 2014 Pennington et al. [37] proposed a new regression model which combines the advantages of the global matrix factorization method and the local context features. Global Vectors (GloVe) first builds a co-occurrence matrix X of a fixed window size. X_{ij} represents the number of times word j appears in the context of word i. Then, GloVe tries to predict the co-occurrence ratio from the word vectors. For example, if $w \in R^d$ are word vectors and $\bar{w} \in R^d$ are separate context word vectors then the basic equation is the following:

$$F(w_i, w_j, \bar{w}_k) \approx \frac{P_{ik}}{P_{jk}} \tag{2.11}$$

Here $\frac{P_{ik}}{P_{jk}}$ is the ratio of co-occurrence probabilities. As one of the goals of GloVe is to find a linear relation between the word vectors, it is possible to restrict the equation to the difference between the word vectors:

$$F(w_i - w_j, \bar{w}_k) \approx \frac{P_{ik}}{P_{jk}} \tag{2.12}$$

The final loss function used by GloVe is the following:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + \bar{b}_j - log(X_{ij}))^2 \tag{2.13}$$

(here V is the vocabulary size. The full mathematical simplification and optimization is not included here.)
GloVe vectors outperform Word2Vec vectors in several NLP tasks including word analogy, word similarity and Name Entity Recognition task. There is no significant different between Word2Vec and GloVe in terms of performance, but with GloVe vectors it is easy to parallelize the model. Thus GloVe can be trained a lot faster than Word2Vec.

2.4 Character Level Representation in Neural Network

In Neural Network architectures, words are represented using word embeddings. The Word embedding layer is either initialized with random vectors or pretrained word vectors like

Word2Vec or Glove. To capture character level features, neural network models often concatenate character representations of words with their corresponding word embeddings. Other research [10][17][49] has used feature look up tables to represent several character level features like prefix, suffix, capitalization, etc. However, they have used predefined character level features. To extract features implicitly, a separate Neural Network model can be used.

2.4.1 Convolutional Neural Network for Character Level Representation

Convolutional Neural Networks (CNN) were first introduced by [43] for character embeddings. They used a convolution kernel to produce local features for each character of a word and then used a max-pooling operation to produce a fixed length character embedding for that word. Later, this idea of using CNN for character representation was used by [8][25][44]. Figure 2.4 shows the network used by [25] on the word *Playing*. First they used a convolution kernel of size 3 over the padded character sequence of the word. Then the model used max-pooling to create a vector representation of size 4 for the word *Playing*. In addition to that, a drop-out layer[48] was used after the max-pooling layer.

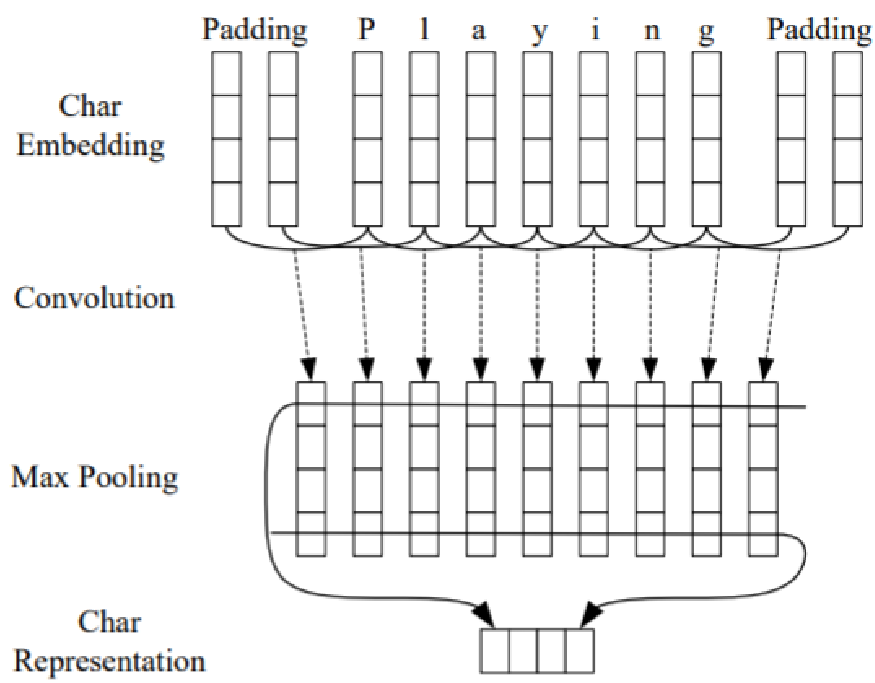


Figure 2.4: Convolutional Neural Network (CNN) for Character level representation [25]

2.4.2 Long Short Term Memory Units for Character Level Representation

Bi-directional Long Short Term Memory Units (Bi-LSTMs) can also be used for finding character embeddings for words [22]. Figure 2.5 shows the Bi-LSTM model used by [22]. The model uses two layers of LSTM named forward LSTM and backward LSTM. Forward LSTM captures features by going over the character sequence from left to right. The backward LSTM captures features in the right to left direction. Then a single character representation vector is created by concatenating the forward LSTM output and the backward LSTM output.

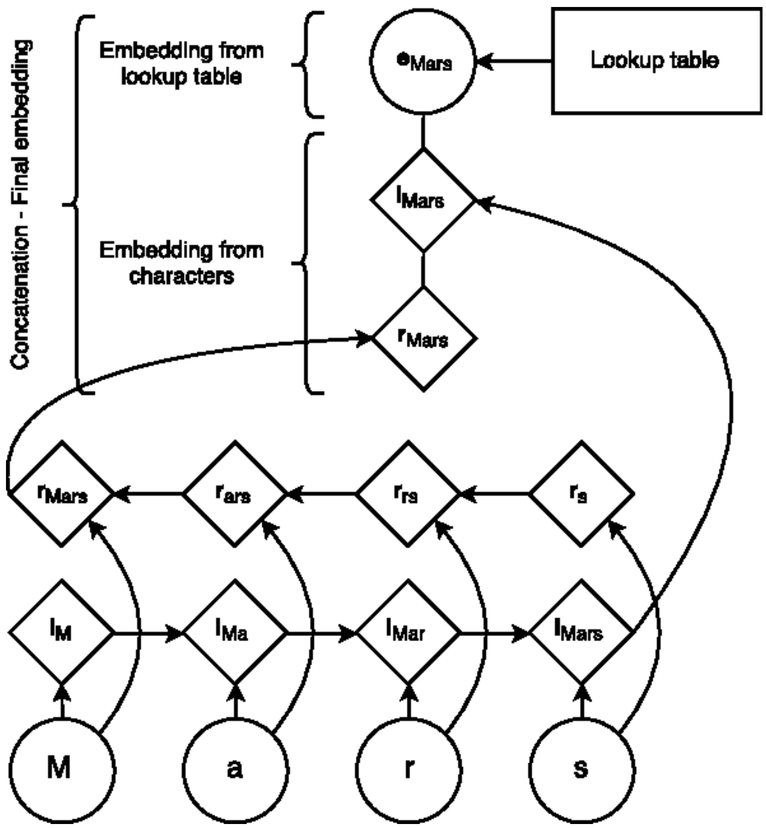
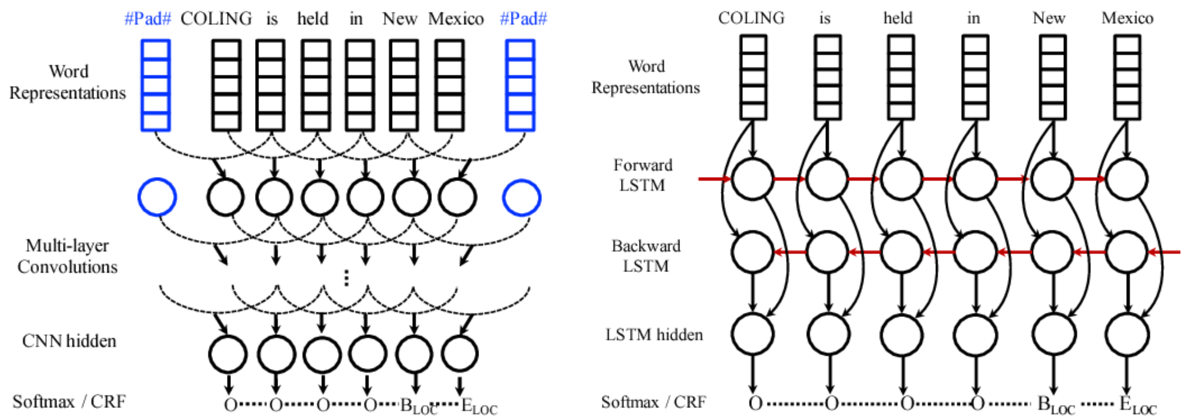


Figure 2.5: Long Short Term Memory Unit (LSTM) for Character level representation [22]

2.5 Word Sequence Representation in Neural Network

After extracting a feature vector for each word of a sentence from its corresponding character sequence, each word is represented with a vector which is the result of a simple concatenation of the word embedding vector and the character level feature vector for that word. Now using this word level representation, we can further calculate word sequence/sentence representation.



(a) Word CNN. (b) Word LSTM.

Figure 2.6: Word Sequence Representation [56]

Like character representation, CNN and LSTM can be used to model word sequence information. LSTM is the more popular choice for word sequence representation because we can extract features from both forward and backward direction of a sentence.[8][17][22][25][38][43] used LSTM for word sequence representation. [10] used CNN for word sequence representation. Figure 2.6 (a) and (b) shows the Word CNN and Word LSTM architecture respectively. The figure shows the way to represent the sentence *"COLING is held in New Mexico"*. As the figure indicates, the idea is similar to character level representation. The Word CNN model uses multiple layers of convolution over the word vector representation of the sentence and finally uses a linear layer to map the representation to the inference layer. Word LSTM model uses one or more layers of Bi-LSTM over the word vectors to capture forward and backward representation of the sentence. Finally the sentence vector for each time step is calculated by concatenating the forward and backward vector of that time step. The final layer (the inference layer) in both architectures uses either Softmax or CRF. These inference layers are discussed in the next section.

2.6 Inference Layer in Neural Network

In sequence labelling models, after finding the word sequence representation of the sentence, we need to assign desired labels to the word sequence. The Inference layer takes the word sequence level representation as input and assigns labels to the word sequence. Softmax and Conditional Random Field (CRF) are two commonly used inference layers for Neural Sequence Labelling Models. These inference layers are described in the next two sections.

2.6.1 Softmax Inference

Softmax is a normalized exponential function. The function can be written as:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Here x_i is the ith component of the vector x. We can first map the word sequence representation to the label vocabulary by using a linear fully connected layer. After that, the softmax function can be applied on the mapping to generate a probability distribution over the label vocabulary. However, softmax does not generate conditional probabilities so dependencies between neighbouring labels cannot be considered. So for tasks with strong output label dependencies Conditional Random Fields (CRF) is a better choice than the Softmax inference.

2.6.2 Conditional Random Fields (CRF)

In sequence Labelling problems it is often better to consider correlations between neighbouring labels to find the best possible tagging sequence. For example, for the POS task, it is more likely that an adjective is followed by a noun rather than a verb. To capture this kind of label dependency while finding the most probable tagging sequence, we can use Conditional Random Fields (CRF). CRF is a Markov network over the labels, which specifies a conditional probability distribution. Given an input sequence $Z = \{z_1, z_2, ..., z_n\}$ and the corresponding label sequence $Y = \{y_1, y_2, ..., y_n\}$, CRF defines a family of conditional probabilities over all possible label sequences y given z as follows:

$$p(y|z; W, b) = \frac{\prod_{i=1}^{n} \chi_i(y_{i-1}, y_i, z)}{\sum_{\bar{y} \in \gamma(z)} \prod_{i=1}^{n} \chi_i(y_{\bar{i-1}}, \bar{y}_i, z)}$$

where W, b are the parameters of the model and $\chi_i(y_{\bar{i-1}}, \bar{y}_i, z) = e^{W_{\bar{y},y}^T z_i + b_{\bar{y},y}}$ is the potential function. We will describe χ_i further in Section 3.6. CRFs are trained by maximizing conditional likelihood of $p(y|z; W, b)$. And the Viterbi algorithm [39] is used to find the label sequence $y*$ with the highest conditional probability $y* = argmax_{y \in \gamma(z)}(y|z; W, b)$.

2.7 Using Hand Crafted Features in Neural Networks

One of the main advantages of using Deep Neural Networks is that neural models can find rich feature representations of input data automatically. However, specialized hand crafted features along with the implicit features found by the Neural Networks can improve the overall performance. Most of the best performing models [10],[17],[49] use handcrafted features in their model. Two kinds of handcrafted features were used by [17]: spelling features and contextual features. There was a total of 12 spelling features for a given word. Some examples of spelling features are: *whether the word starts with capital letter*, *whether the word only consist of capital letters* and *is the word made with letter and digits*. As contextual feature they used *Uni-gram,Bi-gram and Tri-Gram* information. There are different ways to introduce handcrafted features in a neural model. Handcrafted features can be treated as word features and can be concatenated with the word vectors. Another way is to connect the features directly to the output layer just before doing the inference. The experimental results of [17] showed that handcrafted features as word features have the same tagging accuracy as the direct connection. However, direct connection is much faster and accelerated training of the model significantly.

2.8 Performance Comparison Between Neural Network Models

In the previous sections, we introduced the main components of a neural architecture for sequence labelling. Now we compare different models for three sequence labelling problems: POS tagging, NER and Chunking. Comparing Neural Networks is difficult. Neural Networks are very sensitive to hyperparameters. Moreover, datasets used are different from model to model. Models use different training-development-test splits of the same dataset. For example, different development sets for Chunking were used by [24] and [15]. Also for different models the ways to use the development set are different. For instance, the development set has been used only for hyper parameter tuning in [22][25] while the development set has been included into the training in [8][38]. We try to compare the models as fairly as possible. For each of the sequence labelling tasks we present a performance table on a specific benchmark dataset. All tables have 6 columns:

- **Model** : Name of the model.
- **Word Seq**: Technique used for word sequence representation.
- **Char Seq**: Technique used for Character representation.
- **Inference**: Inference layer used.
- **HF**: Whether the model used handcrafted features.

- **F1-score/Accuracy**: Performance metric for the task.

Table 2.1 shows the performance of several models on the CoNLL 2003 dataset [54] for the NER task. The best performing model uses LSTM to find the word sequence representation, CNN for character embeddings and CRF for inference [25]. However the model does not use any handcrafted features. Table 2.2 shows performance comparison for the chunking task on CoNLL 2000 dataset [53]. For chunking, the best F1-score of 95.00% with LSTM for word sequence representation, CNN for character representation and CRF inference layer [38]. Table 2.3 shows the performance comparison among several models on POS tagging task. The data set used for the comparison is the WSJ corpus [28]. For POS tagging two different models get the same best accuracy of 97.55% [17][25]. Both of the models use LSTM for word sequence and CRF for inference. CNN for character embedding was used by [25]. However, instead of using any character level LSTM/CNN, handcrafted features to capture character level features was used by [17]. Careful analysis of the performance tables yield insights about neural sequence labelling models.

- For all the three tasks, models with character representation performed much better than models which do not use any character LSTM/CNN.

- Observing POS tagging, two models attain the best performance [17][25], handcrafted character features are used by only one of the methods [17]. This is a possible indication that, if we can avoid possible feature collision, we can use character embedding and handcrafted features together to boost model performance.

- For all three tasks, word-LSTM is better than word-CNN. Sometimes significantly [10][25].

- Char-CNN is superior to Char-LSTM for all the three tasks. However, in some cases the performance difference of Char-LSTM and Char-CNN might not be significant for POS tagging [22] and [25].

- As expected, CRF is better than Softmax for NER task. We expect CRF to be better because CRF considers dependencies between labels and the NER task has strong label dependency. The difference can be observed by comparing [17] and [49].

Model	Word Seq	Char	Inference	HF	F1-score
Collobert et al.(2011) [10]	CNN	–	CRF	Yes	89.59%
Huang et al.(2015) [17]	LSTM	–	CRF	Yes	90.10%
Lample et al. (2016) [22]	LSTM	–	CRF	No	90.20%
Strubell et al. (2017) [49]	LSTM	–	CRF	Yes	90.43%
Strubell et al. (2017) [49]	LSTM	–	Softmax	Yes	89.34%
Ma and Hovy (2016) [25]	LSTM	–	Softmax	No	87.00%
Chiu & Nichols (2016) [8]	LSTM	CNN	CRF	No	90.91%
Peters et al. (2017) [38]	LSTM	CNN	CRF	No	90.87%
Ma and Hovy (2016) [25]	LSTM	CNN	CRF	No	91.21%
Lample et al. (2016) [22]	LSTM	LSTM	CRF	No	90.94%

Table 2.1: Performance on CoNLL 2003 (NER)

Paper	Word Seq	Char	Inference	HF	F1-score
Collobert et al.(2011) [10]	CNN	–	CRF	Yes	94.32%
Huang et al.(2015) [17]	LSTM	–	CRF	Yes	94.46%
Zhai et al. (2017) [58]	LSTM	–	Softmax	No	94.13%
Hashimoto et al. (2017) [15]	LSTM	–	Softmax	Yes	95.02%
Rei (2017) [42]	CNN	LSTM	CRF	No	93.15%
Peters et al. (2017) [38]	LSTM	CNN	CRF	No	95.00%

Table 2.2: Performance on CoNLL 2000 (Chunking)

Paper	Word Seq	Char	Inference	HF	Accuracy
Collobert et al.(2011) [10]	CNN	–	CRF	Yes	97.29%
Huang et al.(2015) [17]	LSTM	–	CRF	Yes	97.55%
Santos&Zadrozny (2014) [43]	CNN	–	Softmax	No	96.13%
Hashimoto et al. (2017) [15]	LSTM	–	Softmax	Yes	97.45%
Ma and Hovy (2016) [25]	LSTM	–	Softmax	No	96.93%
Lample et al. (2016) [22]	LSTM	LSTM	CRF	No	97.35%
Santos & Zadrozny (2014) [43]	LSTM	CNN	Softmax	No	97.32%
Ma and Hovy (2016) [25]	LSTM	CNN	CRF	No	97.55%
Ma and Hovy (2016) [25]	LSTM	CNN	Softmax	No	97.33%

Table 2.3: Performance on WSJ (POS tagging)

Chapter 3

Proposed Model

In this chapter we will talk about our proposed model for Sequence Labelling tasks. The main contribution of the proposed model is it integrates word sense embeddings with word level features — word embeddings, and morphological features — character embeddings. And then our model uses novel ways to connect these embeddings with the ouput of a Convolutional Neural Network (CNN), a Bidirectional Long Short Term Memory (BLSTM) and a Conditional Random Field (CRF) module. We will describe each of the modules in detail. We will also explain all the morphological and semantic features that we have used to improve our model. Figure 3.4 in Section 3.8 shows our proposed model structure which is a reference for all the module that we are going to describe in detail.

3.1 Word Embedding Module

For any neural network learning, we need to project the lexicons in some vector space. The main motivation of using higher dimensional word vectors is to capture word level features. One obvious way of extracting word level features is to randomly initialize the embeddings and tune them with gradient optimization. However, we can also initialize the vectors with pretrained word vectors like Word2Vec and Glove 2.3 and tune them with the model during training phase. Pre-trained vectors are trained with huge datasets and these vectors are capable of capturing semantic and syntactical relationships between the words. For this reason most of the time, using pre-trained vectors for initialization of the word vectors gives better results than relaying on random vectors. We initialized our word embedding module with random embeddings and pretrained embeddings (Glove/Word2Vec). We represent each sentence as a column vector I_{mx1} where each element in the vector is a unique index of the corresponding word. Here m is the length of the sentence. The word embeddings module transforms the column vector I_{mx1} into a matrix W_{mxd}. Each of the row of the matrix is d dimensional which represent the corresponding word vector.

3.2 Character Embedding Module

Character Embedding Module is used to capture morphological features of each word. For example, for the word *hourly*, the suffix *-ly* gives us a good indication that, *hourly* is an adverb. To learn this kind of low level morphological features, we first represent a word as a $C_{k\times1}$ column matrix where k is the number of characters for that word. Then we used a embedding layer to initialize random embeddings of dimention n for each of the characters. So now the word is represented by a matrix $C_{k\times n}$. After that, we run an LSTM on the matrix and used the last hidden state $C_{1\times n}$ as the character embedding of the word. So if a sentence has m words then the character level representation of the sentence will be a matrix $C_{m\times n}$ where each row $C_{1\times n}$ is the character embedding of the corresponding word found by the LSTM. Figure 3.1 shows how the embedding for the word *"cats"* is calculated using 4 unfoldings of an LSTM cell.

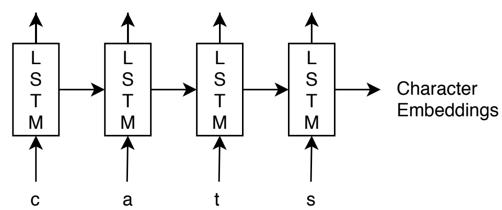


Figure 3.1: Character Embedding for the Word cats

3.3 Selective Pickup from Char-LSTM

Word embeddings are used to capture contextual features. On the other hand Character Embedding Module described in Section 3.2 can extract morphological features of a word. To capture both morphological and contextual features of a sentence together, we introduce a new way of using a BLSTM network. First each sentence is represented in character level by a vector $I_{k\times m}$ where k is the number of characters in the sentence and m is the total words. Then we used an embedding layer to represent each of the k characters in d dimensional vectors. So now we have a matrix of size $k\times m\times d$. Now we run a BLSTM over the character sequence representing the whole sentence. While running the BLSTM we picked up the hidden state from the time step when any word is ended. So each selective pickup contains information about the previous words as well as character sequence up to that time step in both direction.

$$\tilde{C}_{m\times d} = \texttt{SELECT}(\texttt{BLSTM}(I_{(k\times m)\times d})) \tag{3.1}$$

Figure 3.2 shows the Selective Pickup module in action for the sentence *"cats can not fly"*.

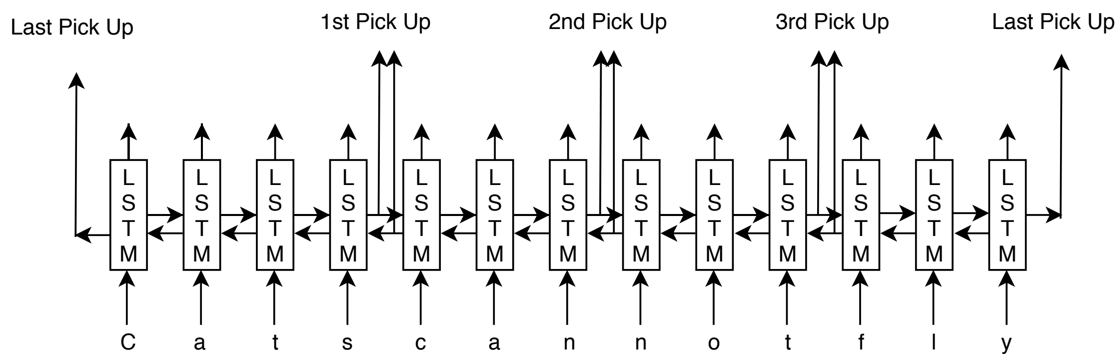


Figure 3.2: Pick Up Character Embedding

3.4 Word Sense Embeddings using Adaptive Skip-Gram

Knowing the sense of a word prior to tagging makes the tagging task more straightforward. Generally, polysemy is captured in standard word vectors, but the senses are not represented as multiple vectors. So we have trained an adaptive skip gram model, AdaGram, [3] which gives a vector for each sense of a word. It is a non-parametric Bayesian extension of the skip-gram model and is based on the constructive definition of Dirichlet process (DP) [13]. It can learn the required number of representations of a word automatically.

In our model, we denote a set of input words as $X = \{x_i\}_{i=1}^{N}$ and their context as $Y = \{y_i\}_{i=1}^{N}$. The ith training pair (x_i, y_i) consists of words $x_i = o_i$ with context $y_i = (o_t)_{t\in c(i)}$, where C is the context window size and $c(i)$ is the index of the context words. Then, instead of maximizing the probability of generating a word given its contexts [32], we maximize the probability of generating the context given its corresponding input words [3]. The skipgram (Section 2.3.1) final objective function becomes,

$$p(Y|X,\theta) = \prod_{i=1}^{N} p(y_i|x_i,\theta) = \prod_{i=1}^{N}\prod_{j=1}^{C} p(y_{ij}|x_i,\theta) \tag{3.2}$$

where, θ is the set of model parameters. The drawbacks of this objective function is that it captures just one representation of a word which goes against a word having different senses depending on the context [2]. To counter this, AdaGram introduces a new latent variable z

which captures the required number of senses even though the number of structure components of the data is unknown a priori. In AdaGram, if the similarities of a word vector with all its existing sense vectors are below a certain threshold, a new sense is assigned to that word with a prior probability p. The prior probability of the kth meaning of word w is

$$\begin{aligned} p(z = k|w, \beta) &= \beta_{wk} \prod_{r=1}^{k-1} (1 - \beta_{wr}), \\ p(\beta_{wk}|\alpha) &= Beta(\beta_{wk} 1, \alpha), k = 1 \ldots \end{aligned} \tag{3.3}$$

where β is a latent variable and α controls the number of senses. Theoretically, it is possible to have an infinite number of senses for each word w. However, as long as we have a finite amount of data, the number of senses can not be more than the number of occurrences of that word. With more data, it can increase the complexity of the latent variables thereby allowing more distinctive meanings to be captured. Taking all the facts into account, our final objective function becomes,

$$\begin{aligned} p(Y, Z, \beta|X, \alpha, \theta) = \prod_{w=1}^{V} \prod_{k=1}^{\infty} p(\beta_{wk}|\alpha) \\ \prod_{i=1}^{N} [p(z_i|x_i, \beta) \prod_{j=1}^{C} [p(y_{ij}|z_i, x_i, \theta)] \end{aligned} \tag{3.4}$$

where $Z = \{z_i\}_{i=1}^{N}$ is a set of senses for all the words.

To use the sense embeddings from Adagram, first we initialized an embedding layer for all the words with the sense vectors generated from AdaGram. Then we tag each word in the input sentence using the module `disambiguate` from AdaGram (the word '*apple*' with sense 2 is tagged as '*apple*_2'). This modified input sentence is then passed to the embedding layer initialized before and finally the resultant output is passed to a BLSTM layer. The output of this BLSTM layer gives the sense level contextual feature $S_{m \times d}$.

3.5 Bi-gram Module using Convolutional Neural Network

Convolutional Neural Networks are often used to extract n-Gram features from a given sentence [23]. Using kernel of different size and length of stride we can extract high level features from a given sentence. In our model, we used only bi-gram features. To extract bi-gram features, the input sentence is padded with a start token <*start*> and passed to an embedding layer. This layer represents the input sentence as a matrix of dimension $(m+1) \times d$ where m is the non-padded sentence length and d the dimension of the embedding. Then we run a convolution kernel of size $2 \times d$ with stride length 1. The result is the bigram embedding matrix B of size $m \times d$.

$$B_{i,:} = \sum_{j=1}^{2} I_{i+j,:} * K_{j,:} \tag{3.5}$$

where I is the input sentence matrix, K is the convolution kernel and n is the maximum sequence length for the current batch. Later, this bigram embedding is passed to a BLSTM layer to extract more abstract features.

3.6 Conditional Random Field Module

Each of the tasks that we are modelling requires a tag to be assigned to each word. In addition to using the current word to predict its tag, it is also possible to use the information about the neighboring words' tags. There are two main ways to do this. One way is to calculate the distribution of tags over each time step and then use a beam search-like algorithm, such as Maximum Entropy Markov models [29] and maximum entropy classifiers [40], to find the optimal sequence. Another way is to focus on the entire sentence rather than just the specific positions which leads to Conditional Random Fields (CRFs) [21]. CRFs have proven to give a higher tagging accuracy in cases where there are dependencies between the labels. Like the bidirectionality of BLSTM networks a CRF can provide tagging information by looking at its input features bidirectionally.

In our model we denote a generic input sequence as $x = \{x_i\}_{i=1}^{N}$, generic tag sequence as $y = \{y_i\}_{i=1}^{N}$, and set of possible tag sequences of x as $F(x)$. Then we use CRF to calculate the conditional probability over all possible tag sequences y given x as

$$p(y|x; W, b) = \frac{\prod_{i=1}^{n} \phi_i(y_{i-1}, y_i; x)}{\sum_{y' \in F(x)} \prod_{i=1}^{n} \phi_i(y'_{i-1}, y'_i; x)} \tag{3.6}$$

where $\phi(.)$ is the score function for the transition between the tag pair (y', y) given x. We train this CRF model using maximum likelihood estimation (MLE) [18]. For a training pair (x_i, y_i)

we maximize

$$L(W, b) = \sum_i \log p(y|x; W, b) \tag{3.7}$$

where W is the weight matrix and b is the bias term. While decoding, we search for the best tag $\hat{y}$ with the highest conditional probability using the Viterbi algorithm [45].

$$\hat{y} =_{y \in F(x)} p(y|x; W, b) \tag{3.8}$$

3.7 Morphology: Spelling and suffix features

For the morphological features, we have focused on spelling and suffix features. We extract 14 spelling features for a given word and store it as a binary vector $SV_{1\times14}$:

- Composed only of alphabetics or not
- Contains non-alphabetic characters except '.' or not
- Starts with a capital letter or not
- Composed only of upper case letters or not
- Composed only of lower case letters or not
- Composed only of digits or not
- Composed of alphabetics and numbers or not
- The starting word in the sentence or not
- The last word in the sentence or not
- In the middle of the sentence or not
- Ends with an apostrophe s ('s) or not
- Has punctuation or not
- The sentence starts with a capital letter or not
- Composed mostly of digits or not

Apart from extracting these features, we also replace all the numbers in the corpus with the <*number*>tag.

We have assembled a list of 137 suffixes from `https://www.learnthat.org/pages/view/suffix.html` and have used the ten that occur most often in our corpus for this study. Then for each of these suffixes, we have collected the words that end with that suffix and have recorded their POSs as well as the frequency. Next, we made an assumption that if a word w with POS x ends with a specific suffix s exceeds a frequency threshold in the training set, then s is the true suffix of word w. We record the pair as (w, s). Finally, we create a one hot (binary vector with exactly 1 one) vector $SUV_{1\times 10}$ for each word where a 1 at index k means the word has the kth suffix.

3.8 Connecting different Modules together

In this module, we combine all the features and the modules using some novel connection techniques and build our final BLSTM-CRF model as shown in Fig. 3.4. First we concatenate the word embedding from module 3.1 with the character embedding from module 3.2 and the suffix vector from Subsection 3.7 as $[W_{m\times d}, C_{m\times n}, SUV_{m\times 10}]$. Following this, we apply a BLSTM on this new embedding matrix, calling this output $O^1_{m\times d}$. The outputs of modules 3.3, 3.4 and 3.5 are called $O^2_{m\times d}$, $O^3_{m\times d}$, and $O^4_{m\times d}$, respectively. Then we initialize four scalar weights w_1, w_2, w_3 and w_4 with initial value 1.0 and add them as model parameters. We form a linear combination of the w_i weighted O^is to form the final output.

$$O = \sum_{i=1}^{4} O^i w_i \tag{3.9}$$

The final output ($O_{m\times d}$) have pieces of information from all the features that we calculated above. We choose linear addition rather than concatenation of these output features, because concatenation will result in a very large feature matrix and the network have to tune each of the cell of this matrix during back-propagation. Following this, we initialize an LSTM layer where we pass the final output from Eqn. 3.9 at each time step $\tilde{O}^i_{1\times z} = \mathtt{LSTM}(O^i_{1\times d}, h^{i-1}_{1\times d})$ and store the outputs separately $\tilde{O}_{m\times z} = [\tilde{O}^1, \tilde{O}^2, \ldots, \tilde{O}^m]$. This LSTM layer unfolds at each time step taking the hidden state of the previous time step to initialize the hidden state of the current time step. The previous hidden state has the information about the previous tag and initializing the current hidden state with the previous one explicitly gives this information. Next we pass the output from each time step to a `tanh` layer $T_{1\times d} = \mathtt{tanh}(\tilde{O}^i)$, which squeezes the values between $[-1, 1]$. Then we concatenate this `tanh` output $T_{1\times d}$ with the spelling features $SF_{1\times 14}$

calculated in subsection 3.7 and pass this to a fully connected (FC) layer. This FC layer maps the output to the number of tag classes $Y_{1\times c} = \mathtt{FC}([T_{1\times d}, SUV_{1\times 14}])$, where c represents number of classes. We do this for each time step and concatenate the results to make a final tensor $Y_{m\times c}$. Finally, we pass this tensor to the CRF layer and calculate the possible tag sequence for the given input sequence. While using Suffix, Spelling and Character Embeddings we tried three different feature connection technique : Residual connection (R), Concatenate with word embedding (CW), Concatenate with second last output layer (CO). The motivation of using these connection techniques is to detect and overcome possible feature collision. We will talk about this three connection ideas later in the experimental result chapter. Figure 3.3 shows the CW, R, CO connections.

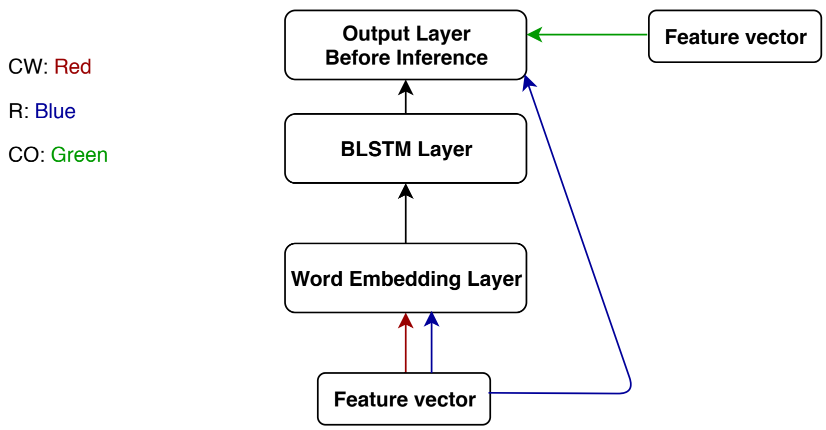


Figure 3.3: CW, R, CO connection

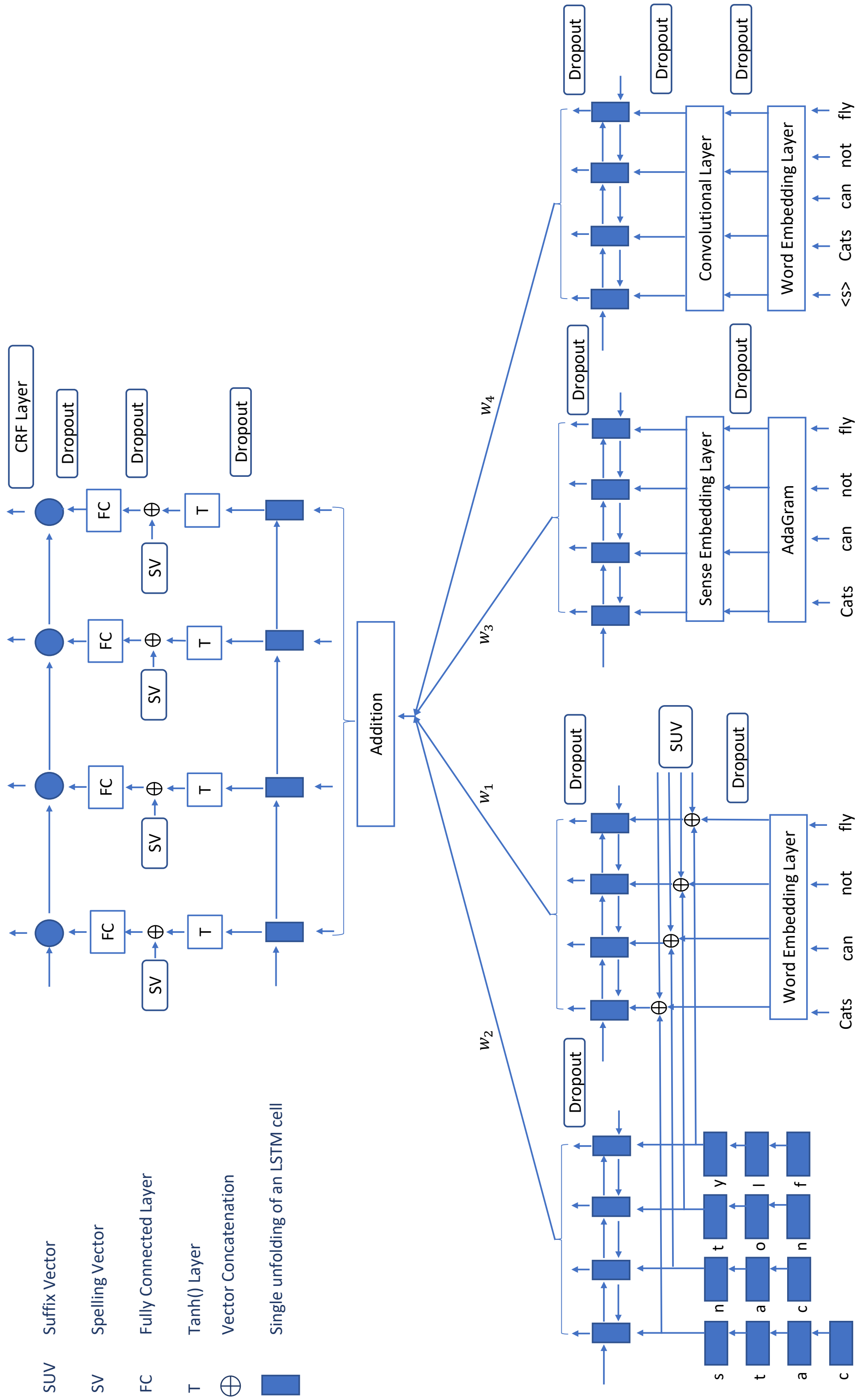


Figure 3.4: BLSTM-CRF model architecture

Chapter 4

Experimental Results

In this chapter we first talk about the experimental setup for evaluating our sequence labelling model on three different sequence labelling tasks: Parts of Speech (POS) tagging, Named entity recognition (NER) and Chunking. Then we compare our model with other state of the art models on benchmark datasets for all the tasks. All the model hyper-parameters are given so that our results can be easily reproduced for further study. We will also discuss the rate of convergence of our model compared to the state of the art one. Moreover, to better understand our model, we present an ablation study of our model on POS tagging by removing certain modules in different combinations.

4.1 Experimental Setup

4.1.1 Datasets Description

We used three different datasets while testing our BLSTM-CRF model on three NLP tasks: Penn TreeBank (PTB) POS tagging [27], CoNLL 2000 chunking [53], and CoNLL 2003 NER [54]. Table 4.1 shows the number of sentences in the training, validation and test sets respectively for each corpus. POS allocates each word with an unique label that shows its syntactic part. Chunking represents each word in terms of its phrase type. NER labels each word with one of four substance: Person, Location, Organization, or Miscellaneous. For NER and Chunking there are several tagging standards like : IO, BIO, BIO2, BIOES etc. We utilize the BIO2 tagging standard for the chunking and NER tasks.

	WSJ	CoNLL00	CoNLL03
Train	39831	8936	14987
Valid	1699	N/A	3466
Test	2415	2012	3684

Table 4.1: Dataset Description

4.1.2 Hardware and Software

For training our model we used GPU accelerated training. We used a machine with NVIDIA GeForce 1080 GPU and 32GB main memory. We implemented our model and several other baseline models using Pytorch 0.3.1 deep learning library. For getting the word sense embedding from Adagram, we used Julia(0.4.5) implementation of AdaGram.

4.1.3 Performance Metrics

We used two different performance metrics while evaluating our model. For NER and Chunking task, we used standard F1-score and for POS tagging we used accuracy metric. F1-score is simply the harmonic mean of precision and recall whereas accuracy is the ratio of total corrected predictions and total number of examples. If TP, TN, FP, FN are true positive, true negative, false positive and false negative respectively then we can define F1-score and accuracy as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$F1 = \frac{2 * Precision \times Recall}{Precision + Recall} \tag{4.3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.4}$$

4.1.4 Hyper-parameters of the Model

Like any other neural architecture, our BLSTM-CRF model has lots of hyper-parameters. Table 4.2 shows detailed hyper-parameter settings of our model. The table also shows the parameters used for AdaGram model for finding the sense vector for all the words in the corpus. Default values are used for the parameters those are not mentioned in the table. We used stochastic gradient descent(SGD) optimizer with learning rate decay and momentum for training. For avoiding over-fitting, we used several regularization techniques like Weight and learning rate decay, drop-out regularization [48]. We used batch learning such that, we update parameter weights of the model after seeing a batch of training examples. For CNN we used only kernel size of 2 because we extract bi-gram features using CNN.

BLSTM-CRF	
Hyper-parameter	**Range Selected**
Learning rate	0.001 / 0.015 / 0.01
Batch size	10 / 50 / 100
No. of LSTM layers	1 / 2 / 3
Momentum	0.9
Dropout	0.5 / 0.2 / 0.1
Word embedding size	300 / 200 / 100
Character embedding size	50 / 30
Initial scalar weight value	1.0
Gradient clipping	5 / 20 / 50
Weight decay	10^{-5}
Learning rate decay	0.05
CNN kernel size	$2 \times (300/200/100)$
AdaGram	
Epoch	1000
Window size	5 / 7 / 10
No. of prototypes	5
Sense embedding size	300
Prior prob. of new sense	0.1
Initial weight on first sense	-1
Word embedding size	300 / 200 / 100

Table 4.2: Ranges of different hyper-parameters.

4.2 Performance on Chunking

Table 4.3 shows the preformance comparison between our model and other state of the art models on Chunking task. The CoNLL 2000 chunking dataset was proposed in a competition and the competition was won by an SVM based model [20] with an F1 score of 93.48%. Later SVM models were outperformed by the neural architectures with Conditional Random Field (CRF) inference. However, an Hidden Markov Model with voting scheme [46] outperformed all the models significantly. Our sequence model which uses CRF inference as well as additional linguistic features out performs all the models with F1 score 96.76% on CoNLL 2000 dataset.

Model	F1-score
SVM classifier [20]	93.48
SVM classifier [19]	93.91
BI-LSTM-CRF [17]	94.13
Second order CRF [30]	94.29
Second order CRF [45]	94.30
Conv. network tagger [10]	94.32
Second order CRF [51]	94.34
BLSTM-CRF (Senna) [17]	94.46
HMM + voting [46]	95.23
BLSTM-CRF (Ours)	**96.76**

Table 4.3: Comparison of F1 scores of different models for chunking

Model	F1-score
Conv-CRF [10]	81.47
BLSTM-CRF [17]	84.26
MaxEnt classifier [7]	88.31
HMM + Maxent [14]	88.76
Semi-supervised [1]	89.31
Conv-CRF + Senna [10]	89.59
BLSTM-CRF [17]	90.10
CRF + LIE [36]	90.90
BLSTM-CRF (Ours)	**91.63**

Table 4.4: Comparison of F1 scores of different models for NER

4.3 Performance on NER Task

Table 4.4 is the performance comparison between models for Named Entity Recognition (NER) task. [10] uses simple Convolutional Neural Network with CRF inference and gets 81.47% F1 score on CoNLL 2003 NER dataset.[17] did many experiments using random and pre-trained embeddings on their model. For random embeddings, they achieved a very low score of 84.26%. However, when they use pre-trained SENNA embeddings [10] along with a gazetteer feature, their F1-score jumped up to 90.10%. Our BLSTM-CRF model gets 91.63% outperforming all the models including [17] which also uses BLSTM-CRF architecture.

4.4 Performance on POS tagging

The performance of several models on POS tagging is shown in table 4.5. As can be seen, a number of models use Convolution or LSTM or BLSTM to get the contextual features and CRF to do the tagging. They achieve very good accuracies of 97.29% [10], 97.51% [22] and 97.55% [25]. Some of the models use multitask learning, doing two or more tasks at the same time. They also achieve very good accuracies: 97.43% [41] and 97.59% [24]. Our model achieves an accuracy of 97.58% which is higher than all of the existing models except LM-LSTM-CRF [24] which leverages a language model for the tagging tasks. LM-LSTM-CRF, however, has a mean accuracy of 97.53% (reported accuracy: 97.53 ± 0.03) which is lower than the our model's mean accuracy (97.57 ± 0.01). Also, as shown in Table 4.6, our model's training time is one quarter that of LM-LSTM-CRF with on par performance.

Model	Accuracy
Conv-CRF [10]	97.29
5wShapesDS [26]	97.32
Structure regularization [50]	97.36
Multitask learning [41]	97.43
Nearest neighbor [47]	97.50
LSTM-CRF [22]	97.51
LSTM-CNN-CRF [25]	97.55
LM-LSTM-CRF [24]	97.59
BLSTM-CRF (Ours)	97.51
BLSTM-CRF (Ours) without CNN	**97.58**

Table 4.5: Comparison of Accuracy of different models for POS tagging

Model		Acc.	Time
	LSTM-CRF	97.35	37
	LSTM-CNN-CRF	97.42	21
	LM-LSTM-CRF	97.53	16
	LSTM-CRF	97.44	8
	LSTM-CNN-CRF	96.98	7
Ours	BLSTM-CRF	97.51	4
	BLSTM-CRF without Bigram	97.58	3.5

Table 4.6: Training time (hours) of our BLSTM-CRF model on the WSJ corpus compared with all models of [24] using the same hardware configuration (GPU: Nvidia GTX 1080)

Ablation Study for POS tagging

Table 4.8 gives the ablation study of our model where we show how we apply different combinations of features in different parts of our BLSTM-CRF architecture to get an optimal con-

Module	**W**	100th epoch	200th epoch	300th epoch	400th epoch	500th epoch
Word emb	w_1	0.91	0.84	0.80	0.77	0.78
Sense	w_3	0.85	0.76	0.69	0.64	0.65
SP-CLSTM	w_2	0.81	0.66	0.48	0.35	0.34
Bigram	w_4	0.75	0.49	0.27	0.01	0.01

Table 4.7: Change in w_i's for each module with epochs.

figuration. With so many features and parameters, these sequence models are very much prone to overfit. But with careful tuning as well as with proper feature connections, it is possible to leverage those features. We extract a set of morphological as well as semantic features from our dataset such as spelling, suffix and char-level features. We experiment on applying various combinations of these features in different segments of our model. Our extensive experimentation shows that optimal results are achieved when these features are added in the model through residual connection (R), concatenation with word embeddings (CW) and concatenation with the second last output layer (CO). Focusing on which segment to connect each feature, our experiments found that the spelling feature works best when concatenated with the second last output layer, and the suffix feature as well as the character embeddings work well when concatenated with the word embeddings. This configuration is what is kept in our final model. We further continue our experiments by turning on / off different modules such as word embedding, sense embedding, selective pickup from LSTM and bi-gram embedding. We found significant contribution of word embeddings, sense embeddings and selective pickup from LSTM compared to the bigram modules as shown by the weights at the 500th epoch in Table 4.7. The bigram module gives better performance without considering previously generated POS and vice versa. However, linguistically, the information about the previous tag has a huge influence in generating the current one. So we kept the first three modules along with the previously generated POS and discarded the bigram module from our final model. Our best model as shown in the last row of Table 4.8 gives state of the art results.

Model no.	Word emb	Sense	SP-CLSTM	Bigram	Suffix		Spelling			Char Embed		Prev. POS	Acc.
					CW	CO	R	CW	CO	CW	CO		
1	Rand.	×	×	×	-	-	-	-	-	-	-	×	95.42
2	Glove	×	×	×	-	-	-	-	-	-	-	×	96.13
3	Glove	✓	×	×	-	-	-	✓	-	-	✓	×	97.08
4	Glove	✓	✓	×	✓	-	✓	-	-	✓	-	×	97.15
5	Glove	✓	✓	×	✓	-	-	✓	-	✓	-	×	97.22
6	Glove	✓	×	×	-	✓	-	-	✓	✓	-	×	97.32
7	Glove	✓	✓	×	-	-	-	-	✓	✓	-	×	97.45
8	Glove	✓	✓	×	✓	-	-	-	✓	✓	-	×	97.48
9	Glove	✓	✓	✓	✓	-	-	-	✓	✓	-	×	97.50
10	Glove	✓	✓	×	✓	-	-	-	✓	✓	-	✓	**97.58**

Table 4.8: Ablation study of our BLSTM-CRF model for POS tagging. (R - Residual connection, CW - Concatenate with word embedding, CO - Concatenate with second last output layer)

Chapter 5

Conclusion and Future Work

Sequence Labelling is one of the fundamental problems in Natural Language Processing. In this thesis, a new neural architecture is proposed which can be applied to any sequence labelling tasks. Our model improved neural sequence labelling architecture by leveraging from additional linguistic information such as polysemy, bigrams, character level knowledge and morphological features. The model uses pre-trained word vectors and character embeddings generated from LSTM network. We introduced a clever way of capturing morphological and contextual features using selective pickup from a BLSTM network. We used word sense embedding from Adagram module for each of the word to capture polysemy. Moreover, our model uses spelling and suffix features which are defined for every words. Total 14 carefully designed spelling features are used. For suffixes, our model only focuses on 10 suffixes that occur most often in our training set. Our experimental finding shows that benefitting from such adequately captured linguistic information, we can assemble a considerably more compact model, hence yielding much better training time without loss of effectiveness. Designing a model with such rich feature space needs careful consideration on how to connect the features appropriately. We carefully connected the features so that feature collision can be minimized. To avoid feature collision we performed an extensive ablation study where we produced an optimal model structure along with an optimal set of features. Finally we used a linear combination of different feature modules while predicting the output sequence via a Conditional Random Field inference. Our best model achieved state of the art results on the POS tagging, NER and chunking benchmark datasets and at the same time remains four times faster to train than the best performing model currently available. Our experimental results show that multiple linguistic features and their proper inclusion significantly boosted our model performance.

5.1 Future Work

Using a trained Language Model

We can use a separate trained Language Model like Liu et al. [24] to generalize better. A Language Model can help us to extract features from raw texts. However, extracting knowledge by pretraining a Language Model needs a large external corpus and a significant amount of training time.

More Hand-crafted Features

In this thesis, we used 14 spelling features and 10 most frequent suffixes from a total of 137 available suffixes. In the future, we can introduce more carefully hand-crafted morphological features to improve model performance.

Multitask Learning and Extra Data

We can use multitask learning, optimizing the model with separate loss functions for more than one sequence labelling task simultaneously. Moreover, we can exploit extra training data from another corpora.

Bibliography

[1] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853, 2005.

[2] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy. *arXiv preprint arXiv:1601.03764*, 2016.

[3] Sergey Bartunov, Dmitry Kondrashkin, Anton Osokin, and Dmitry Vetrov. Breaking sticks and ambiguities with adaptive skip-gram. In *Artificial Intelligence and Statistics*, pages 130–138, 2016.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003.

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

[6] Denny Britz. Introduction to RNNs kernel description, 2015.

[7] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition: A maximum entropy approach using global information. In *Proceedings of the 19th International Conference on Computational Linguistics-Volume 1*, pages 1–7, 2002.

[8] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional LSTM-CNNs. *arXiv preprint arXiv:1511.08308*, 2015.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations

using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

[11] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[12] Asif Ekbal, S Mondal, and Sivaji Bandyopadhyay. POS tagging using HMM and rule-based chunking. *The Proceedings of SPSAL*, 8(1):25–28, 2007.

[13] Thomas S Ferguson. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, pages 209–230, 1973.

[14] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003-Volume 4*, pages 168–171, 2003.

[15] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

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

[17] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[18] Søren Johansen and Katarina Juselius. Maximum likelihood estimation and inference on cointegration with applications to the demand for money. *Oxford Bulletin of Economics and Statistics*, 52(2):169–210, 1990.

[19] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies*, pages 1–8, 2001.

[20] Taku Kudoh and Yuji Matsumoto. Use of support vector learning for chunk identification. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language learning-Volume 7*, pages 142–144, 2000.

[21] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 282–289, 2001.

[22] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[23] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Molding CNNs for text: Non-linear, non-consecutive convolutions. *arXiv preprint arXiv:1508.04112*, 2015.

[24] Liyuan Liu, Jingbo Shang, Frank Xu, Xiang Ren, Huan Gui, Jian Peng, and Jiawei Han. Empower sequence labeling with task-aware neural language model. *arXiv preprint arXiv:1709.04109*, 2017.

[25] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. *arXiv preprint arXiv:1603.01354*, 2016.

[26] Christopher D Manning. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 171–189, 2011.

[27] Mitchell Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. Treebank-3 ldc99t42. *CD-ROM. Philadelphia, Penn.: Linguistic Data Consortium*, 1999.

[28] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[29] Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. Maximum Entropy Markov Models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, volume 17, pages 591–598, 2000.

[30] Ryan McDonald, Koby Crammer, and Fernando Pereira. Flexible text segmentation with structured multilabel classification. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 987–994, 2005.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

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

[33] Nam Nguyen and Yunsong Guo. Comparisons of sequence labeling algorithms and extensions. In *Proceedings of the 24th International Conference on Machine Learning*, pages 681–688, 2007.

[34] Christopher Olah. Understanding LSTM Networks kernel description, 2015.

[35] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[36] Alexandre Passos, Vineet Kumar, and Andrew McCallum. Lexicon infused phrase embeddings for named entity resolution. *arXiv preprint arXiv:1404.5367*, 2014.

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

[38] Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*, 2017.

[39] Lawrence R Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[40] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*, 1996.

[41] Marek Rei. Semi-supervised multitask learning for sequence labeling. *arXiv preprint arXiv:1704.07156*, 2017.

[42] Marek Rei, Gamal KO Crichton, and Sampo Pyysalo. Attending to characters in neural sequence labeling models. *arXiv preprint arXiv:1611.04361*, 2016.

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

[44] Cicero Nogueira dos Santos and Victor Guimaraes. Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*, 2015.

[45] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141, 2003.

[46] Hong Shen and Anoop Sarkar. Voting between multiple data representations for text chunking. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 389–400, 2005.

[47] Anders Søgaard. Semisupervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 48–52, 2011.

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[49] Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. Fast and accurate entity recognition with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098*, 2017.

[50] Xu Sun. Structure regularization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 2402–2410, 2014.

[51] Xu Sun, Louis-Philippe Morency, Daisuke Okanohara, and Jun'ichi Tsujii. Modeling latent-dynamic in shallow parsing: A latent conditional model with improved inference. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 841–848, 2008.

[52] Scott M Thede and Mary P Harper. A second-order Hidden Markov Model for part-of-speech tagging. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 175–182, 1999.

[53] Erik F Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning-Volume 7*, pages 127–132, 2000.

[54] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003-Volume 4*, pages 142–147, 2003.

[55] Ralph Weischedel, Richard Schwartz, Jeff Palmucci, Marie Meteer, and Lance Ramshaw. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):361–382, 1993.

[56] Jie Yang, Shuailong Liang, and Yue Zhang. Design challenges and misconceptions in neural sequence labeling. *arXiv preprint arXiv:1806.04470*, 2018.

[57] Xiang Yu, Agnieszka Faleńska, and Ngoc Thang Vu. A general-purpose tagger with convolutional neural networks. *arXiv preprint arXiv:1706.01723*, 2017.

[58] Feifei Zhai, Saloni Potdar, Bing Xiang, and Bowen Zhou. Neural models for sequence chunking. In *AAAI*, pages 3365–3371, 2017.

[59] GuoDong Zhou and Jian Su. Named entity recognition using an HMM-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 473–480, 2002.

Curriculum Vitae

Name:	Muhammad Rifayat Samee
Post-Secondary Education and Degrees:	The University of Western Ontario London, ON 2017 - 2019 M.Sc.
	American International University-Bangladesh Dhaka, Bangladesh 2009 - 2014 B.Sc.
Honours and Awards:	Western Graduate Research Scholarship 2017-2018
Related Work Experience:	Teaching Assistant The University of Western Ontario 2017 - 2018
	Research Assistant The University of Western Ontario 2017-2019