Implementing Natural Language Processing (Almost) from Scratch

Yunhua Zhao
Vanderbilt University School of Engineering
2301 Vanderbilt Place, PMB 358227
Nashville, Tennessee 37235

Abstract

By directly learning representations of words in context as feature vectors, we hope to train a neural network on said data in order to perform several natural language processing tasks without the aid of *a priori* knowledge and task-specific engineering. We model our architecture on the paper "Natural Language Processing (Almost) from Scratch" (2011) by Collobert *et al.*, while incorporating the work done by Mikolov *et al.* on word2vec. Our pipeline has been used to tag various standard NLP corpuses according to four main tasks: part-of-speech tagging, chunking, named entity recognition, and semantic role labeling.

1. Introduction

Natural language processing is a branch of artificial intelligence that involves the "understanding" of human speech and language by computers. Human languages are extremely tough to generalize. Thus, the initial natural language processing techniques involved very complex sets of rules intending to perform specific tasks

The next major development within NLP came with the growing popularity of machine learning. The introduction of statistical learning techniques removed some of the expert human knowledge required to generalize the very problematic English language. Rather than being purely linguistically designed, human intervention was only needed to provide guiding heuristics for various classification methods, providing much more robust solutions. However, the initial parameter setup and feature selection process still required human ingenuity.

This paper's contributions remove yet another layer of human guidance, by using deep learning techniques to learn a representation for an NLP task, oftentimes succeeding in matching or surpassing the performance of hand-engineered, task-specific features. The learned representations are then fed into another deep learning architecture in order to predict the desired tags. We wish to replicate these techniques, outlined in the paper, and test them on smaller-scale, simpler data sets on personal machines, and compare them to the performance of widely available NLP libraries. The four tasks we will attempt to replicate performance on are: part-of-speech tagging, chunking, named entity recognition, and semantic role labeling.

Our initial tests use the Brown Corpus pre-labeled with part-of-speech tags. We implemented the architecture proposed by Collobert *et al.* [1] for the POS task, and we trained and tested it on various segments of the corpus, using various parameters and setups. Building upon that functional baseline, we modified the pipeline where needed and tested it on the other tasks.

2. Model

The goal of this project is to implement a generalized model that can be applied toward all four main NLP tasks: part-of-speech tagging, chunking, named-entity recognition, and semantic role labelling. Ignoring extra, higher-level features, the model can be divided into two main components.

The first step is feature selection. What typically requires background research of previously successful features or the careful feature engineering of computational linguistics experts is taken over by a neural network. Collobert *et al.* describe a network featuring a lookup table layer that learns vector representations for words from scratch. There were some issues properly implementing the lookup table layer, so we looked to Google's word2vec tool set described by Mikolov *et al.* [2] to produce feature vectors from raw words. Our model also allows convenient augmentation of the feature vectors with additional higher-level features that may be helpful for certain tasks, such as POS tags for the other more difficult prediction models.

We use a window approach network, learning the effects of a word's context by treating vector representations of a window of words as the features to be fed through a neural network of various configurations. Each word window, or n-gram, consists of the word of interest, surrounded on each side by a number of words defined as a hyperparameter. The trained network can then be used to make predictions on the various tags of each word window in a similarly processed test data set.

The model we implemented based on the Collobert paper consisted of an input layer and a single hidden layer consisting of 128 units. The network also used a dropout rate of 0.2 and an L2 regularization parameter of 0.001. We used the Adamax optimizer, which is effective for more sparsely

updated weights, and batches of 128. We ran trials both with the hard tanh activation function proposed by Collobert *et al.* as well as with ReLU, when the hard tanh function had numerical instability issues. Beyond tuning some of the regularization hyper-parameters and learning rate, we also experimented with other network configurations. We ran trials with an additional hidden layer, as well as some with a convolutional layer with maxpooling and dropout prior to the dense hidden layer.

3. Implementation

We developed a generalized pipeline to pre-process the various datasets used for each task into CSV files of n-grams of arbitrary size, one for each word. Our tests use n-grams of sizes 3 and 5. In order to properly pad words at the beginnings and ends of sentences, $\lfloor n/2 \rfloor$ "NULL" words are inserted at the end of each sentence. We do this in order to treat the context of each individual sentence separately from one another. Words not present in the trained dictionary learned through word2vec are replaced with a "UNSEEN" token. This is the baseline pre-processing done for each of the tasks we attempt.

We also wrote a tagging script that requests input in plain text form and loads the appropriate trained model depending on the desired NLP task. The raw text is then returned with each word followed by its tag surrounded by parentheses. We use this script to evaluate our model (by eye) on unlabeled data we collect from the internet.

3.1 Part-of-Speech Tagging

Part-of-speech tagging, as the name implies, is the labeling of each individual word in a corpus with the part of speech that is most indicative of that word's functional role in its context. This task is generally considered the easiest of the four mentioned, and thus functioned as our benchmark for the development of our pipeline, to ensure that everything was behaving more or less as expected.

For training the model for the POS task, we chose to use the Brown corpus, which consists of just over one million words labeled with 34 different parts of speech, and is commonly used in natural language processing research. The distribution of the labels is given below:

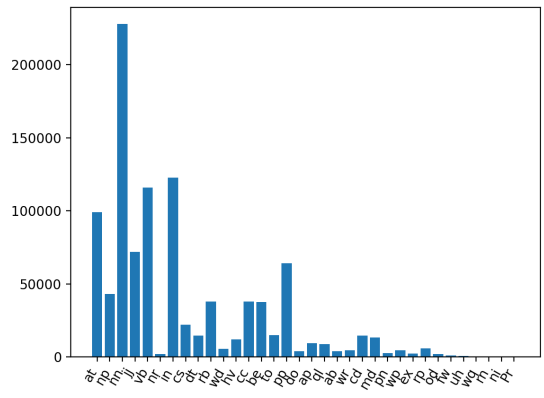

Figure 1: Brown corpus POS label frequencies

We first trained our word2vec model for 100 epochs on the unlabeled Brown corpus. We then saved the trained model in order to access its dictionary mapping known words to feature vectors.

The CSV data files generated by our pre-processing pipeline are then converted into two-dimensional numpy arrays, where each row is a single instance, consisting of the feature vectors for each word in the n-gram concatenated in the same order. "NULL" words are replaced with the zero vector, and "UNSEEN" words are replaced with a vector of ones.

Similarly, the POS labels for each word are extracted from the Brown corpus, converted into a one-hot encoding scheme, and saved as numpy arrays segmented in the same manner as the n-grams.

We then describe various neural network architectures using TensorFlow, for the purpose of finding the most effective one for our dataset, varying the number of hidden layers, the activation function, and the regularization parameters.

During the initial development and debugging phase, we trained our models on 10% of the dataset over 50 epochs with a 70/30 training/validation split, and we saved checkpoints of the weights for each epoch that improves upon the previous best validation loss, allowing our training to make use of early stopping as a regularization technique. Predictions were then made using the last checkpoint on the next 1% of the dataset, reproducing a CSV file of labels, which are then compared using an evaluation script.

3.2 Chunking

Chunking is the labeling of sentence segments with syntactic functions like noun phrases, verb phrases, and adjective phrases. This word-phrase labeling scheme is done at the individual word level by differentiating the beginnings, interiors, and ends of phrases in each word's tag, along with including the type of phrase. We chose to use the begin-interior scheme, where phrases are delimited by "begin" labels, and the other words in the phrase are simply labeled "interior" (along with the phrase's grammatical function, of course).

The dataset we used came from the Conference on Natural Language Learning in 2000. This corpus had words labeled with 21 different tags. The dataset is pre-divided into a training set and a test set. The distributions of labels in the training and test sets are provided below:

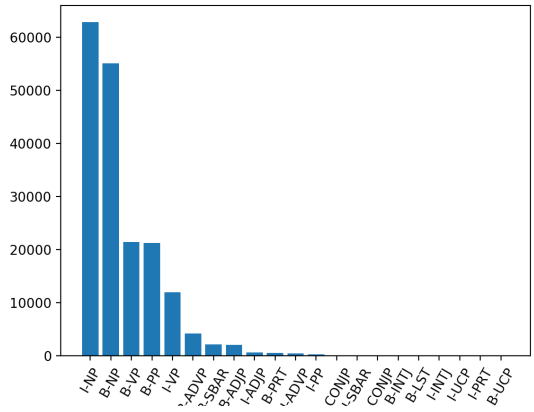

Figure 2: Training set label frequencies

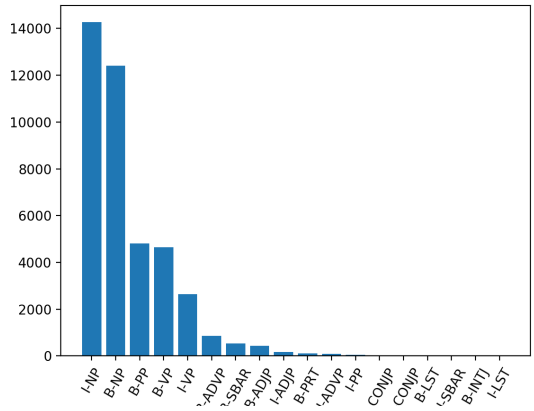

Figure 3: Test set label frequencies

The dataset also provided punctuation marks, generally labeled with the letter "O" to show that they were outside of any phrases. We removed these punctuation marks, but kept track of the ones denoting the ends of sentences, in order to properly separate the context for different sentences.

Again, we used the more successful neural network architectures from our POS task to train and test for this task. For our evaluation of the models, we examined two different schemes. As is normal, we simply compared labels for each individual word and computed the percentage that were the same. But this typical scheme does not capture the fact the phrases, despite being labeled on a word-by-word basis, behave as a single unit. Thus, two consecutive noun phrases incorrectly labeled as a single noun phrase would only be marked wrong for the single "I-NP" tag in the middle that should have been a "B-NP" tag. Thus, we developed a specialized evaluation script for the chunking task, comparing a single phrase at a time.

3.3 Named Entity Recognition

Named entity recognition is the locating and labeling of named entities like proper nouns, locations, quantities, etc. The labeling scheme typically used is just single-word labeling of several pre-defined categories, with non-named entities and other words simply labeled with an "O."

The dataset we used for this task is the corpus provided for the 2002 shared task by the Conference on Natural Language Learning, including labels for geographical entities, organizations, people, geopolitical entites, times, artifacts, events, natural phenomena, etc. Additionally, it also includes POS labels for each of the words, which we attempt to make use of in improving our NER performance. The label frequencies for the data are again provided below, with the non-named entity labels removed:

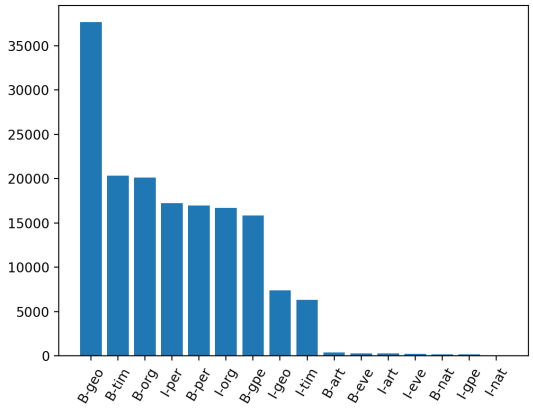

Figure 4: Training and test set combined frequencies

As before, we format the dataset for compatibility with our model, breaking into individual sentences and stripping unrelated punctuation. We also split the dataset into training and test sets at a ratio of 9 to 1.

We trained our models on three versions of the dataset. The first version is analogous to the previous tasks. The second is the same as the first, but augmented with the additional feature of the POS label for the word of interest for each instance. The third is further augmented with the POS labels of all the words in the ngram.

For the evaluation of this task, we realized that the high percentage of non-named entities somewhat artificially boosts the accuracy. So we wrote a script to calculate the following additional metrics to best examine the performance of the network: named entities correctly labeled, named entities incorrectly labeled, named entities missed, and percentage of non-named entities falsely labeled.

3.4 Semantic Role Labeling

Semantic role labeling gives each word or phrase in a sentence a label that defines their role in a function-like parse of the sentence, such as "agent" or "result" or "action." At the high level, words are either arguments or functions. Since these labels often apply to phrases, this task is also known as shallow semantic parsing.

Unfortunately, the datasets provided by the CoNLL 2005 shared task require a license to access, and the other data available is in extremely poorly formatted plain text, delimited by variable numbers of spaces, asterisks, and parentheses, so we left the SRL task out of this project.

4. Results and Evaluation

We began our initial experiments using just 10% of the POS dataset, and training on 3-grams, in order to speed up the developmental and debugging phase. As we perfected our pipeline, we began using the full dataset for the POS tas and all subsequent datasets for their respective tasks.

4.1 POS

We first experimented with training the network on the 3-gram data using the hard tanh activation function as proposed in the Collobert paper (defined in TensorFlow as the "hard sigmoid"). Unfortuantely, our selected hyperparameters resulted in poor performance, and we were unable to find a significantly more suitable set of hyperparameters, so we continued using the same set, resulting in the following loss graphs.

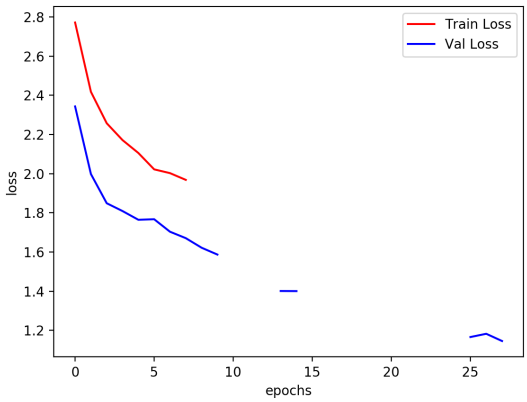

Figure 5

The gaps in the graph lines represent points where the calculated loss is "not a number." The testing accuracy is just 72.99%, which is significantly worse than the results from the ReLU networks that we will examine next.

Our first successful trials were still trained on the first 10% of the corpus. Upon switching the activation function to ReLU, and appropriately decreasing the learning rate, we were able to get the following loss graphs.

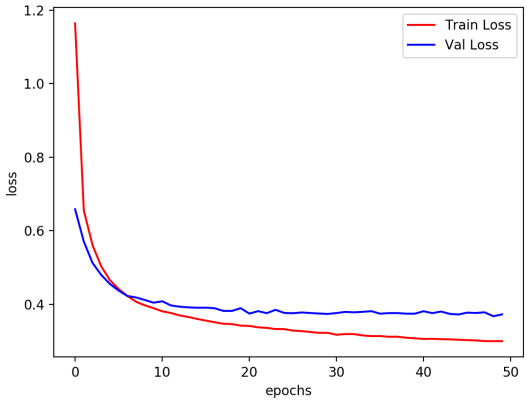

Figure 6

This model achieved a training accuracy of 90.25% and a testing accuracy of 89.96%, when tested on the next 10% of the corpus.

With this very reasonable accuracy, we doubled the amount of contextual information each training word of interest had access to, but increasing the window size from 3 to 5. After generating the new (for now just 1% of the total dataset) training data, we trained and tested our model for two more trials. The training and validation loss graphs are included below.

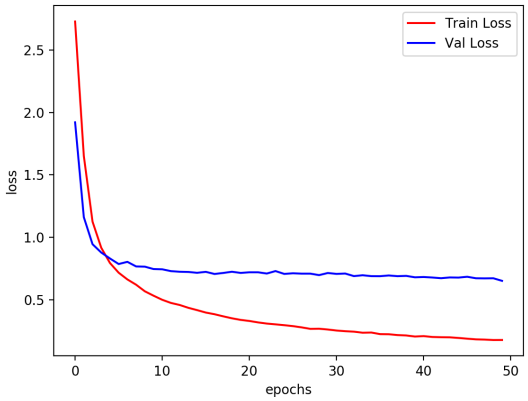

Figure 7

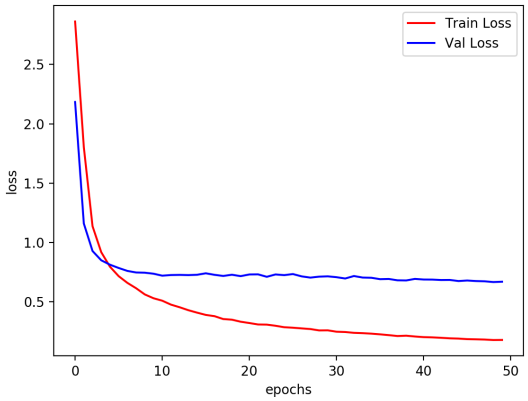

Figure 8

When tested and evaluated on the original training data, the second trial was marginally better, producing an accuracy of 96.37%. When tested and evaluated on the next segment of data, it produced an accuracy of 88.34%.

Perhaps unsurprisingly, decreasing the training data size allows greater opportunity for overfitting, so we increased the training data amount back to the first 10% of the data for another two trials, whose loss graphs are given below.

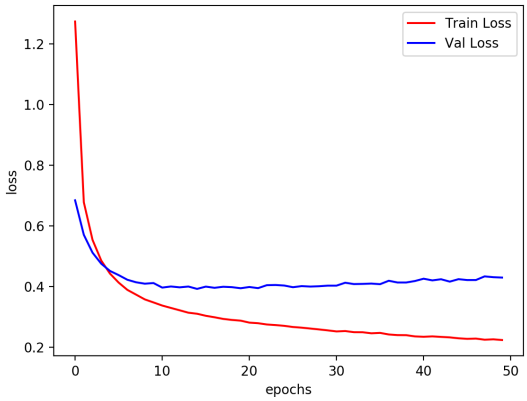

Figure 9

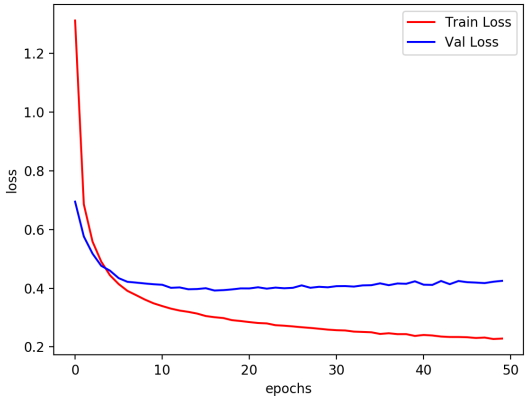

Figure 10

The training accuracies were 90.76% and 90.83%, and the testing accuracies were 90.60% and 90.82%, suggesting the increased data size allowed for a smaller generalization error compared to training with one-tenth of the data, and also suggesting that the increased n-gram size gave a smaller generalization error compared to just having one contextual word before and after the word of interest.

Interestingly, when we compared the two predictions to each other, we saw that they shared just 94.82% of the same tags. This shows that, despite extremely similar performance, the local minima found by each network are not at all the same, suggesting that perhaps training several networks and applying a voting scheme could result in further improved performance.

To test this theory, we created an ensemble using a simple majority voting scheme, between the two 5-gram networks presented above as well as our best 3-gram network from before. We achieved a slightly better test accuracy of 91.34%, which is not terribly significant, but still shows that, perhaps with more diversely trained models, we could further improve upon that accuracy.

Finally training on the entire dataset, we achieved 94.54% training accuracies using the same configuration, with the following loss graphs:

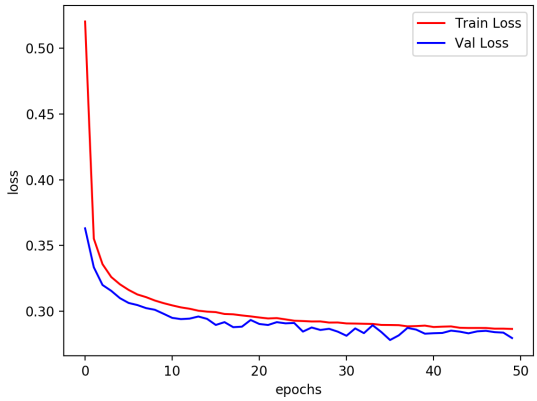

Figure 11

Using this model, we compared the label frequencies of the predicted labels to the actual frequencies (plotted above). The graphs appeared to be extremely similar, with the slight overall pattern of the model favoring the most common labels at the expense of the least common ones.

We also experimented with inserting an additional 1 dimensional convolutional layer with a kernel size of 100 and a stride length of 10 before the dense layers, followed by a maxpooling layer and another dropout layer. We picked the kernel size in order to match the length of the vector representing each individual word. The model achieved 90.75% training accuracy, with the following loss graphs:

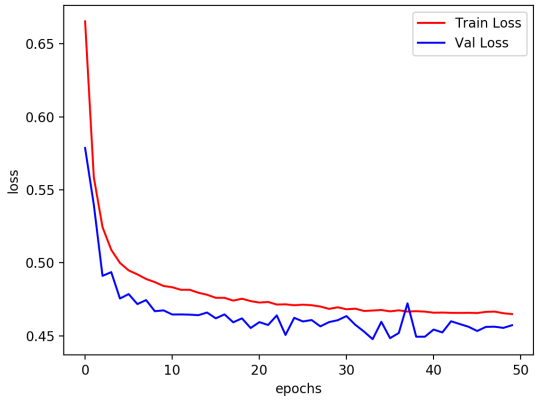

Figure 12

With decent performance on these tasks, we sought to ensure the model was doing more than learning the usual part of speech tag for a word, or overly relying on that single words word2vec features, so we developed a few basic sanity checks. First, we developed a program that takes raw text input and labels each word with a POS tag using the model trained on the standard configuration on the full Brown corpus. Then, we used it to take simple English grammar quizzes online that tested the ability to differentiate the same word used as both nouns and verbs in different sentences. The quiz we used had 10 questions with 4 sentences each, 3 using the word as a noun and one as a verb. Our

model correctly classified 37 out of the 40 sentences, only missing 1 question. While by no means comprehensive, it does lend credence to the claim that it is learning some underlying structure to English grammar.

As a more objective check, we also attempted to train a network on the Brown corpus using scrambled labels. This way, the label frequencies are distributed in the same way, but there is no longer any rhyme or reason behind the grammar, a quality which people often accuse the English language of having anyway.

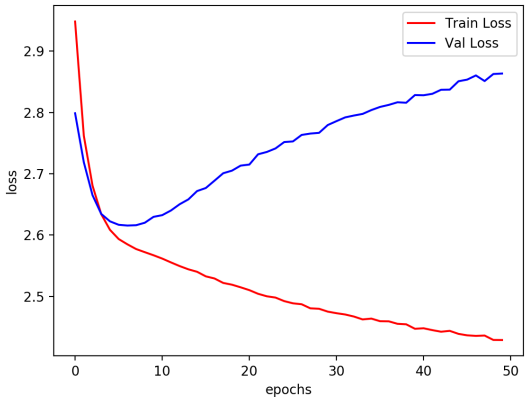

Figure 13

The network achieved a 2.518% testing accuracy, and likely could have been worse if we had not utilized early stopping.

4.2 CHUNK

Using the same architecture as our most successful POS trials, we ran several trials with our chunking dataset.

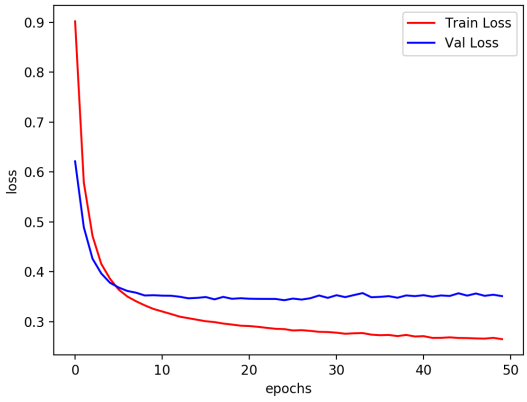

Figure 14

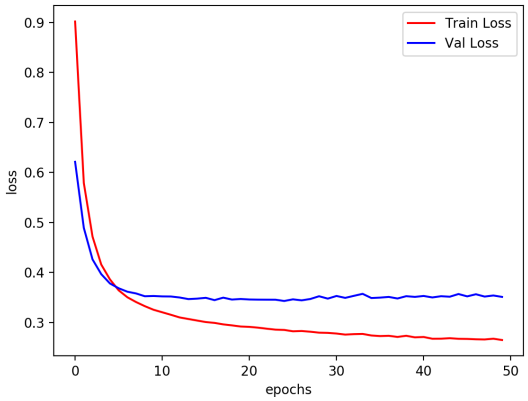

Figure 15

Interestingly, our model performed at nearly the same level as the POS task, with training accuracies of 94.75% and 94.67% and testing accuracies of 89.52% and 91.21%, despite the POS task generally being considered the easier task. This could be the result of the chunking dataset we use simply having fewer labels.

However, once we realized that it does not necessarily make sense to evaluate the chunking performance in the same word-by-word way that we evaluated the POS task, we evaluated the predictions one phrase at a time, as described in section 3.2. The results are, understandbly, slightly worse, since a single mislabeled word invalidates the entire phrase. Nevertheless, we believe this is a more accurate representation of the model's performance.

	Correct chunks	Total chunks	Accuracy
Training	97797	106932	0.9146
Test	20510	23838	0.8604

NER

As mentioned in the implementation section, our NER trials have very high accuracy due to the abundance of "O" labels. The results of the trials using the standard pipeline are presented below:

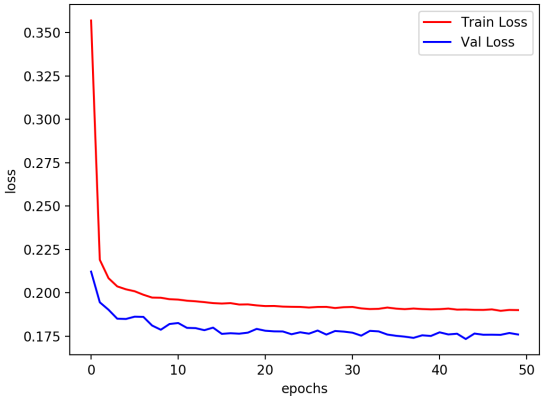

Figure 16

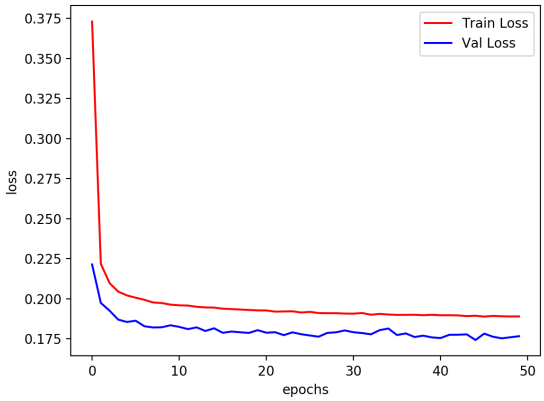

Figure 17

***

Training Accuracy: 95.99%
Named entities correctly labeled: 78.00%
Named entities incorrectly labeled: 11.46%
Named entities missed: 10.55%
Percentage of non-named entities falsely labeled: 0.64%
Testing Accuracy: 95.72%
Named entities correctly labeled: 76.71%
Named entities incorrectly labeled: 11.66%
Named entities missed: 11.63%
Percentage of non-named entities falsely labeled: 0.66%
***

The model correctly labels about 77% of the named entities, and incorrectly labels about 12%. overall, about 89% of named entities are at least correctly identified as named entities. In terms of false positives, less than 1% of the regular words are labeled as named entities of some kind. However, we wanted to investigate the effects of augmenting our dataset with the POS labels for each word of interest.

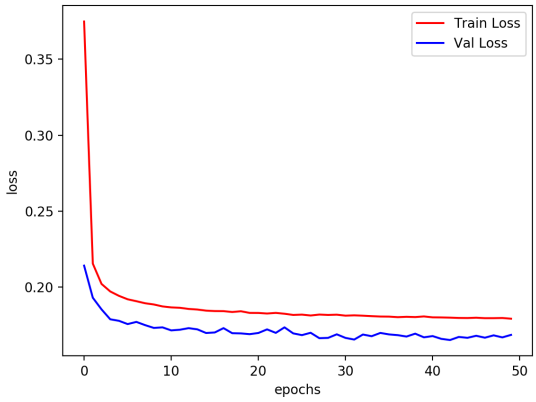

Figure 18

***

Train Accuracy: 96.26%
Named entities correctly labeled: 79.37%
Named entities incorrectly labeled: 11.85%
Named entities missed: 8.78%
Percentage of non-named entities falsely labeled: 0.58%
Test Accuracy: 95.98%
Named entities correctly labeled: 78.25%
Named entities incorrectly labeled: 12.35%
Named entities missed: 9.40%
Percentage of non-named entities falsely labeled: 0.64%
***

While there is a definite improvement, it is very slight, so we ran another trial by further augmenting the features with the POS labels for each word in the ngram.

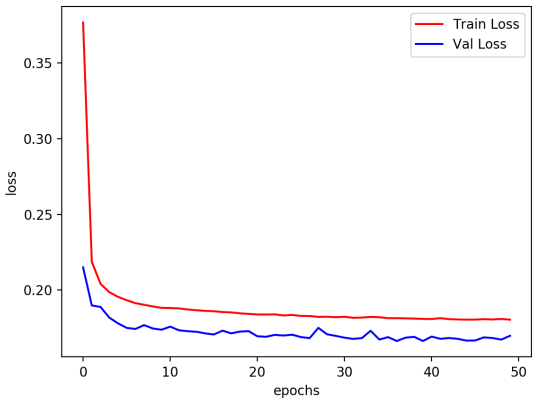

Figure 19

***

Train Accuracy: 96.16%
Named entities correctly labeled: 79.06%
Named entities incorrectly labeled: 12.53%
Named entities missed: 8.41%
Percentage false positives: 0.6376%
Test Accuracy: 95.08%
Named entities correctly labeled: 76.46%
Named entities incorrectly labeled: 10.20%
Named entities missed: 13.34%
Percentage false positives: 1.098%
***

We see that further augmenting seems to slightly decrease the performance, moreso on the test accuracy than training accuracy, suggesting that overfitting is the culprit. It is rather surprising that adding the additional POS features does not improve performance, thought it could be a result of the single feature for each word in the ngram being considered equally as the other 100 features associated with it, when it likely should have more predictive power, thus clashing with the regularization parameters that are otherwise necessary.

5. Related Work

The Collobert *et al.* paper proposes its own model for feature extraction from raw words, and it also extracts extra higher-level features for use alongside the learned features. Since it was written in 2011, before the Mikolov *et al.* paper describing word2vec was published and established as a standard in 2013, it understandably does not make use of it. Collobert *et al.* also propose a sentence approach in addition to their window approach, which better takes context into account.

There are many other approaches toward each of the four tasks. Focusing on part-of-speech tagging, many methods proposed prior to 2011 rely on approaches other than deep learning.

In the early 1990s, hidden Markov models (HMMs) were commonly applied toward the POS task. These HMM-based models use a table of the probabilities of certain sequences in order to determine the likeliest tag based on previous tags. Collectively, they treat texts as being produced by a Markovian process with hidden state and the tagging problem as deciding which states the Markov process went through during text generation. Toutanova *et al.* [3] made use of maximum entropy classifiers and inference in a bidirectional dependency network, directly taking into account effects from both preceding and following word and tag contexts.

In 1992, Eric Brill introduced a simple rule-based tagger that iteratively identifies and remedies its own weaknesses [4]. During training, rules are automatically learned from a pre-tagged corpus. Rules can be something like [article] + [noun] $-$ $>$ [verb], and a sufficient set typically has over 80 rules. The trained model initializes by tagging each word with the most commonly-seen part of speech associated with it, or, if its a word unknown to the classifier, naively initializing to be nouns. The rules are then repeatedly applied until convergence is reached (*i.e.* no more rules can be applied) or a certain threshold has been reached.

Widely used libraries such as the Natural Language Tool Kit (NLTK) provide POS taggers as well. NLTK uses a pretrained greedy averaged perceptron tagger, which guesses tags based on the tags of words before and after as features. However, it does not train the model "from scratch" using extremely limited prior knowledge as Collobert *et al.* describe and our model attempts to emulate.

Shifting to the chunking task, Kudo and Matsumoto developed an approach using support vector machines in 2001, winning the CoNLL 2000 shared task. Their approach achieves high generalization even with high-dimensional data using SVMs, and they boost their accuracy by applying weighted voting of 8 SVMs, both of which were fairly recently developed techniques. The use of SVMs allowed for feature selection to be easier than with HMMs. The features consisted of a window around the word of interest containing POS tags and words as features, as well as surrounding tags [5].

The NER CoNLL shared task in 2003 was won by an ensemble model by Florian et al. [6]. It combines 4 very different classifiers: a transformation-based learning model from Charniak, an HMM, a robust risk minimization model, and a maximum entropy model.

The performance of these other approaches on all these tasks is obviously very good, often with much lighter-weight models. However, the novelty of the method proposed by Collobert *et al.* is the independence from outside knowledge and task-specific engineering.

6. Conclusion, Limitations, and Future Work

By making use of word2vec's feature extraction via self-supervised learning, we have been able to label our processed Brown corpus with mostly correct part-of-speech tags. We were able to identify and label the grammatical function phrases in text. And we fairly reliably located and classified named entities among a sea of other less relevant words. Additionally, we have explored the effects of several changes to our model.

Our experiments were limited by the specifications of the machine they were run on. However, we were table to train on entire datasets by making use of swap space in addition to RAM and a little bit of patience. Although we were successfully able to complete the first three tasks, the lack of useable data for the SRL task was disappointing, though our initial project scope did not include completing the SRL task, which is widely considered to be the most difficult of the four.

Additionally, the two CoNLL datasets that we used had unescaped commas in some words even though they are in CSV files, which caused many hidden issues in our preprocessing pipeline. This design flaw led to us making the decision of stripping the intra-sentence punctuation from the training process.

7. References

[1] Ronan Collobert, Jason Weston, Lon Bottou, Michael Karlen, Koray Kavukcuoglu, Pavel Kuksa. Natural Language Processing (Almost) from Scratch. The Journal of Machine Learning Research, 12, p.2493-2537, 2/1/2011

[2] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient estimation of word representations in vector space. In ICLR.

[3] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technologies (NAACL-HLT), 2003.

[4] Brill, Eric (1992). A simple rule-based part of speech tagger. In Proceedings, Third Conference on Applied Natural Language Processing, ACL, Trento, Italy.

[5] T. Kudo and Y. Matsumoto. Chunking with support vector machines. In Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT), pages 18, 2001.

[6] R. Florian, A. Ittycheriah, H. Jing, and T. Zhang. Named entity recognition through classifier combination. In Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL-HLT), pages 168171, 2003.