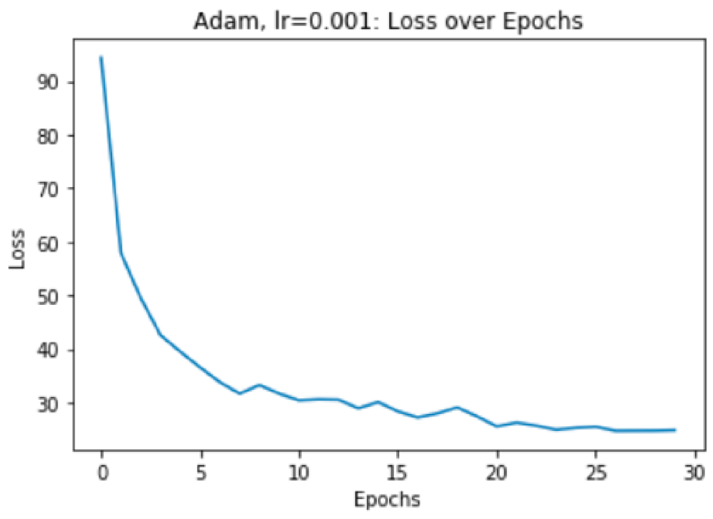
CS 577– Homework 2 Solutions

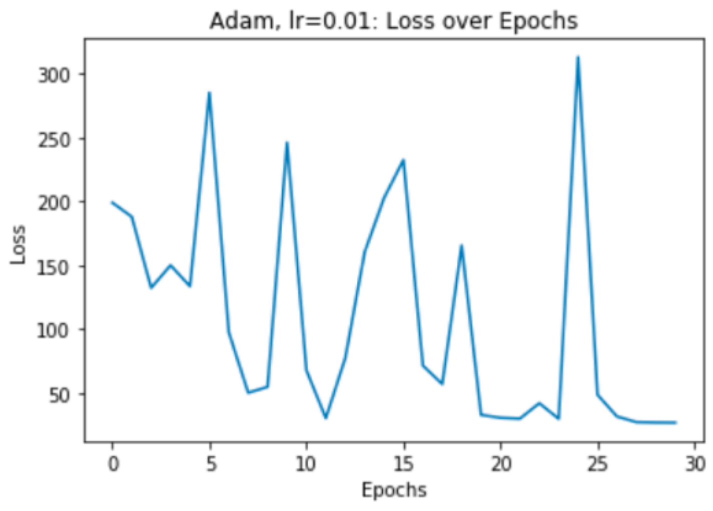
Joshua Yeung

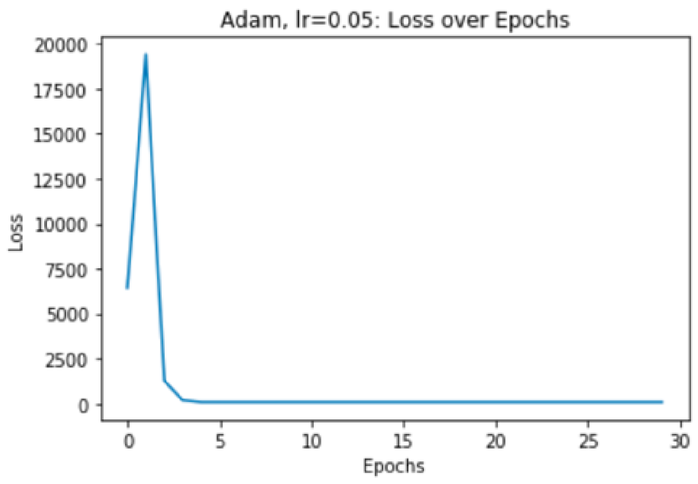
1. Logistic Regression

(a) State one hyper-parameter you tuned in case of the Neural Network. How did tuning these hyper-parameters change the result? Explain with learning curves. You may have tuned more than one hyper-parameter. In this section explain only one. For this question, you are not allowed to use embedding type as a hyper-parameter. (4 Points)

For my Option 2 model I tuned the learning rate for my optimizer. I chose the Adam optimizer, and experimented with different learning rates ranging from 0.001 to 0.05 over 30 epochs on the entire training set, and plotted some charts. It seems that 0.001 was the best learning rate. As the graphs tell, when learning rate was increased, the loss became more erratic and didn't converge as nicely. We can see for the learning rate 0.05 the loss exploded and eventually dropped to a lower level, but not as low as the plots with smaller learning rates.

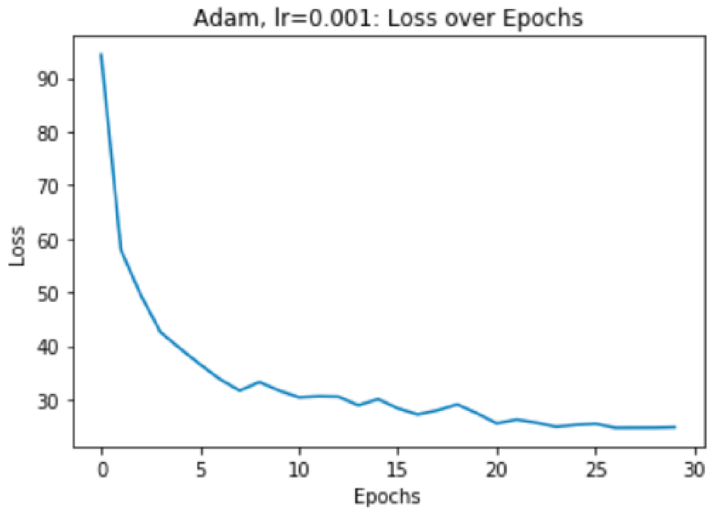



Adam, lr=0.01: Loss over Epochs
Loss
Epochs
300
250
200
150
100
50
0
5
10
15
20
25
30


Adam, lr=0.05: Loss over Epochs
Loss
Epochs
20000
17500
15000
12500
10000
7500
5000
2500
0
5
10
15
20
25
30

2.

(a) My Option 2 performed the best out of my models. Performance suggests that pretrained embeddings can be helpful for tasks where we work with small datasets.



3.

(a) In MEMM, we provide an input context [current word embedding,prev tag embedding] and try to predict the target tag for the current word. For options 1 and 2, I used a bigram implementation, which helped add some context for tagging. The bigram input context [prev word embedding,current word embedding, prev label embedding]. These word embeddings were either randomly initialized and then trained along with the neural network, or loaded from Word2Vec embedding binaries. I concatenated the bigram context with tag embedding (one hot encoded) and then fed that into MLP, with a NLL Loss function. For Option 3, I used a Bi-LSTM instead of bigrams to capture context from before and after the word.

Option 1 might be good if you have a vocabulary that pretrained embeddings like word2vec don't capture (for example, twitter text). Maybe this vocabulary is unique to your context (medical terminology, etc.) and you want to learn representations using your own data.

Using pretrained embeddings like in option 2 might be a better choice in a small dataset where it may be hard to train a representative embedding of words, like in this homework.

Option 3 uses a BiLSTM which looks at the whole sentence instead of two words. This can better capture context than n-grams, especially from words in front of the current word.

Due to technical issues I wasn't able to tune my models to predict true positives, thus the F1 score was 0.

4.

(a) Our model performance would be worse because we lost the features that capture the context of the current word. Then our model would just predict P(tag — word) and return the NER tag that occurs most often with the current word in the training set. The performance would be just as if we shuffled around all the words in the sentences into a list and randomly picked words from that list to train our model.