**IDS 703: Final Project**

**Authors: Godwin Anguzu, John Owusu Duah and Victor Nomwesigwa**

**Date: 12/10/2021**

**Inline Response to Tasks**

**1. Choose an NLP problem**

We selected Part-of-Speech (PoS) tagging problem

**2. Identify or construct a solution based on a generative probabilistic (language) model. Describe the model in detail and develop a solution using parameter inference (and/or decoding)**

We used Theano to carry out logistic regression because Scikit-learn took too long to run as a result of the size of our real-world training data. We used Scikit-learn’s Decision Tree Classifier as a sanity check.

Before running Logistic Regression, we abstracted the encoding of words (tokens) and PoS tags as feature and target vectors. In the fit method of our Logistic Regression class, we specified a learning rate of 0.1, a momentum of 0.99, batch size of 100 and 6 epochs. In the fit method, we convert our feature and target vectors to theano’s integer vectors. To compute the probability of a PoS tag, given a word, we get the word vector of the input word, multiply it by its weight, add the bias term and apply the softmax function of the result. This gives each word its own tag. We defined stochastic gradient descent as our cost and updates which we used with batch size and epochs to minimize the cost.

Since F1 score is the harmonic mean of the precision and recall, we decided to use the f1 score on the test data to assess how accurately our model was correctly classifying part of speech tags.

To generate synthetic data, we used a trigram generative probabilistic model based on the brown corpus.

**3. Identify or construct a solution based on a (discriminative) neural network. Describe the network structure in detail and develop a solution using gradient descent.**

We constructed a recurrent neural network with Theano to solve PoS tagging. We chose a recurrent neural network because it has a feedback loop where the hidden layer goes back into itself. This allows the recurrent neural network to take account of data which occurred in the past. In the context of sentences, any prediction will depend on all the previous words in the sentence. Taking past words into account will help us correctly classify words which have multiple tags in different contexts. We chose Theano after running into dependency issues with Tensorflow because our workstation is an M1 MacBook. Theano afforded us the opportunity to build a recurrent neural network, thereby meeting the requirement of the project.

The structure comprises of the input layer which encodes our data set to integer vectors. After the input layer, the vectors are multiplied by weights and are fed into ReLU activation function into the hidden layer of recurrent units. After which, we apply a softmax function to map the result of the activation from the recurrent units in the hidden layer to the output layer. The softmax returns probabilities so to find the predictions, we select the PoS tag that maps onto the highest probability. The cost is softmax cross entropy.

**4. Apply both approaches to synthetic data that you generate according to the generative model from Step 2. Evaluate the results qualitatively and quantitatively. Highlight situations where each approach performs well and poorly.**

As mentioned earlier, the synthetic data was generated with a n-gram language (i.e. trigram) model. We set the word (token) in maximum batches of 25 tokens. With probability distribution of tokens from the brown corpus, the generated tokens were saved and manually labeled in a .txt file titled “test\_synthetic.txt”. We evaluated both models with the synthetic data as the test data because it was impractical to generate and manually label a large data set with a trigram model to be used for training both models. Also, because generated tokens do no follow grammatically rules of the English language, it would not be possible to properly evaluate both models trained on grammatically wrong language. It is worthy to note this limitation of using a generative probabilistic model to generate tokens. Generated tokens will not follow grammatical rules of the English language. The logistic regression model had a training f1 score of 0.82 and test f1 score of 0.65. This shows that the trained model does not generalize well to new data.

We evaluated the recurrent neural network and had a training f1 score of 0.92 and a test f1 score of 0.86. Even though the synthetic data makes little sense, grammatically, qualitatively, the neural network does a better job at modeling or classifying the contextual nuances of parts of speech.

**5. Apply both approaches to “real” data acquired legally. Evaluate the results qualitatively and quantitatively. Highlight situations where each approach performs well and poorly. Any unusual/unexpected results require explanation (and frankly, probably debugging)**

The real-world data set consists of the same partitions of the Wall Street Journal corpus as the widely used data for noun phrase chunking. It can be found at this link https://www.clips.uantwerpen.be/conll2000/chunking/. It consists of 211,727 tokens and tags and it has each token or word with its corresponding tag on one line. The test data has 47,377 tokens or words. The annotation of the data has been derived from the WSJ corpus by a program written by Sabine Buchholz from Tilburg University, Netherlands. In the train and test data sets, each token is on a line and the part of speech tag is beside it. For instance, we have “expected VBN I-VP” in our data set. “expected” is the token and “VBN I-VP” is the part of speech tag.

The f1 score on the training job is 0.83 and the f1 score on the test data is 0.82. Since the f1 score of the train and test data sets are approximately the same, it means the model generalizes well to new data. These scores are confirmed by the Decision Tree Classifier. There is a very strong caveat to this high score. Logistic regression does not adequately model or explain the contextual use of words or tokens which recurrent neural networks are good at. For instance, the training data is littered with sentences where the word pressure was used and labeled as a noun. You can observe its use in the sixth sentence in the training data: “If there is another bad trade number, there could be an awful lot of pressure - Briscoe, U.K. economist for Midland Montagu, a unit of Midland Bank PLC.” However, when the model encounters pressure in the test data as used in the following sentence: “Traders said hedging related to the TVA and pricing also pressure Treasury bonds”. It classifies it as a noun instead of a verb because logistic regression can only carry out one-to-one mapping of words to tags. Logistic regression cannot capture the contextual nuances when the same tokens are used as different parts of speech.

On the other hand, as mentioned earlier, recurrent neural network (RNN) takes into account how tokens have been used in the past to tag tokens more accurately than a logistic regression. We assessed the accuracy of the recurrent neural network using f1 score. The f1 score on the training data is 0.985 and the f1 score on the test data is 0.898. It is interesting to note that the RNN outperformed the logistic regression model in classifying the word “pressure” as used in the following sentence in the test data as a verb. “Traders said hedging related to the TVA and pricing also pressure Treasury bonds”.

The script with generative probabilistic (language) model for the real-world data set is saved as “pos\_baseline.py” and the script with the deterministic neural network solution for the synthetic data set is saved as “pos\_rnn\_synthetic.py”.

**6. Discuss the pros and cons of the two approaches. Consider quality/correctness, data, time, and computational requirements and interpretability and any other distinguishing critera.**

Pros of using Generative Probabilistic Language Model for PoS tagging

1. Logistic regression model required less time compared to the recurrent neural network. Averagely it takes about 2 minutes on our workstation. Obviously, this time will vary depending on the computing power available. This time is also dependent on the size of the train and test data. The larger the data set, the longer it will take.
2. The generative probabilistic model used is lightweight and does not require as much computing power as the recurrent neural network. So, in situations where there is lack of computing power, it would be prudent to use a logistic regression model to carry out PoS tagging even though there are tradeoffs and demerits to consider
3. The generative probabilistic model used is more interpretable than recurrent neural network because predictions are purely based on the probability of occurrence of tokens and tags.
4. There are a plethora of available libraries and modules that can carry out PoS tagging with generative probabilistic language models.

Cons of using Generative Probabilistic Language Model for PoS tagging

1. As mentioned earlier, the generative probabilistic language model cannot capture the different grammatical contextual use cases of PoS tags . For instance, in the real-world training data set, the token “pressure” occurred more times as noun than a verb. The probabilistic language rightly assigns a high probability of occurrence as a noun than a verb. Hence, it will always predict “pressure” to be a noun even in situations where it is used as verbs. Unlike recurrent neural network, generative probabilistic models cannot keep a memory of the different contextual use cases of tokens depending on how they have been used in sentences. Hence, they are not as accurate as recurrent neural networks in PoS tagging.

Pros of using Recurrent Neural Network Model (RNN) for PoS tagging.

1. RNNs keep a memory of how tokens have been used as different parts of speech and predicts according to this memory. This makes RNN’s more accurate when carrying a task like PoS tagging.

Cons of using Recurrent Neural Network Model (RNN) for PoS tagging.

1. The RNN model, using Theano library, takes approximately twice as long to execute as compared to the probabilistic model. Averagely it takes about 15 minutes on our workstation. Obviously, this time will vary depending on the computing power available. Initially, we implemented the solution with Tensorflow on Google Colab and it took about 7 hours to carry out PoS tagging. We tried to download and implement Tensorflow on our M1 Macbook work station but we had dependency issues.
2. The RNN requires much more computing power than generative probabilistic models although Theano allowed us to run a relatively light weight implementation.
3. The RNN was not as interpretable and it is difficult to explain **how** the model “understands” the contextual nuances of PoS tagging.
4. There are not a lot of available libraries and modules to implement Recurrent Neural Network for PoS tagging with.