Assignment 3 Report
CSE 143: Intro to Natural Language Processing

University of California Santa Cruz
Tommaso Framba, tframba@ucsc.edu, ID: 1815342
Chris Toukmaji, ctoukmaj@ucsc.edu, ID: 1720414
Wednesday June 1, 2022

This assignment was done in Python 3 without use of any NLP libraries (such as NLTK).

Abstract

In this third assignment we implemented two text classification neural networks utilizing SimpleRNN and LSTM layers respectively. We also derived the Viterbi algorithm and implemented the Viterbi algorithm by creating a decode method that returns the max tag sequence for a given sentence, tags, and scores.

1 Programming: Text Classification with Neural Networks

1.1 Programming: Text Classification with RNNs

Model Description In this first part of the first problem we built a text classifier for the task of sentiment analysis in order to determine if a sentence from the IMDb reviews data set has positive or negative sentiment. This model utilizes Keras in order to create the model and apply layers to it. This model is made up of a SimpleRNN layer with 64 hidden dimension size, tanh nonlinearlity, and a dropout of 0.5. This model also utilizes a Dense sigmoid layer in order to classify the sentiment of the text. The model compilation is done with binary cross entropy loss, an adam optimizer that uses 10^{-3} learning rate, and an accuracy metric.

Performance This model can be fully trained in around 10 minutes with each epoch taking 32 seconds.

Experimental Procedure In order to implement this neural network with a SimpleRNN layer we began by replacing the two existing GNN layers and replaced them with the SimpleRNN layer. We then encoded the test set and trained the model to get a baseline accuracy score before tuning the model. Then we tuned the model with nonlinearity of tanh, word embedding dimension size of 16, hidden dimension size of 64, dropout rate of 0.5, optimization method of adam, learning rate of 10^{-3}, training batch size of 32, and number of training epochs of 20. After running the model with these hyperparameters and evaluating on the test set we was satisfied with the result and recorded the deliverables.

Deliverables

1. Training accuracy: 0.9517, Testing accuracy: 0.7146
2. Located in zip as:

   `asg3_1_SimpleRNN.py`

1.2 Programming: Text Classification with LSTMs

Model Description In this second part of the first problem we built a text classifier for the task of sentiment analysis in order to determine if a sentence from the IMDb reviews data set has positive or negative sentiment. This model utilizes Keras in order to create the model and apply layers to it. This model is made up of a LSTM layer with 64 hidden dimension size, tanh nonlinearlity, and a dropout of 0.5. This model also utilizes a Dense sigmoid layer in order to classify the sentiment of the text. The model compilation is done with binary cross entropy loss, an adam optimizer that uses 10^{-3} learning rate, and an accuracy metric.

Performance This model can be fully trained in around 5 minutes with each epoch taking 15 seconds.

Experimental Procedure In order to implement this neural network with a LSTM layer we began by replacing the two existing GNN layers and replaced them with the LSTM layer. We then encoded the test set and trained the model to get a baseline accuracy score before tuning the model. Then we tuned the model with nonlinearity of tanh, word embedding dimension size of 16, hidden dimension size of 64, dropout rate of 0.5, optimization method of adam, learning rate of 10^{-3}, training batch size of 32, and number of training epochs of 20. After running the model with these hyperparameters and evaluating on the test set we was satisfied with the result and recorded the deliverables.

Deliverables

1. Training accuracy: 0.9463, Testing accuracy: 0.7307
2. For our models the LSTM models does in fact have better accuracy than the simple RNN model on longer sequences. In order to test this hypothesis we changed the preprocess method to include one thousand characters from the reviews instead of only three hundred. As a result the simpleRNN took an hour and 25 minutes to train with 4.25 minutes per epoch and produced a training accuracy of 0.8924 and a test accuracy of 0.7011. The LSTM model in comparison trained in 16 minutes total with 45 seconds per epoch and produced a training accuracy of 0.9463 and a test accuracy of 0.7307. This showcases that the long short-term memory model has better accuracy on longer sequences and only resulted in a 300% increase in training time with similar training and test accuracy while the simple RNN model resulted in a 944% increase in training time and a 7% decrease in training accuracy.
3. Located in zip as:

   `asg3_1_LSTM.py`

2 Theory: Deriving the Viterbi Algorithm

Model Description In this section, we derive the Viterbi algorithm using by maximizing a global scoring function:

$$\hat{y} = argmax_y S(x, y) \tag{1}$$

where $S : x, y \mapsto \mathbb{R}$. x represents a sentence, and y represents the tag sequence for that sentence. We want to maximize this function in order to return the most likely sentence. However, the brute force approach of checking every possible tag sequence is very inefficient. The Viterbi algorithm alleviates this issue because it is a dynamic-programming algorithm, meaning that we will store the best current path.

Performance The Viterbi algorithm is more efficient than a brute-force approach because we trade space complexity for time complexity in dynamic programming algorithms.

Experimental Procedure We can rewrite $S(x, y)$ in terms of x and y.

$$S(x, y) = \sum_{i=1}^{n+1} s(x, i, y_{i-1}, y_i) \tag{2}$$

Deliverables

1. Let

$$v_j(y_j) = max_{y_1,\ldots,y_{j-1}} \sum_{i=1}^{j} s(x, i, y_{i-1}, y_i) \forall y_j \tag{3}$$

We want to show that

$$\forall y_j, v_j(y_j) = max_{y_{j-1}}[s(x, j, y_{j-1}, y_j) + v_{j-1}(y_{j-1})] \tag{4}$$

By induction,

Base Case: $j = 2$ holds true.

Via Equation 1,
$v_2(y_2) = max_{y_1} \sum_{i=1}^{2} s(x, i, y_{i-1}, y_i) = s(x, 1, y_0, y_1) + s(x, 2, y_1, y_2)$

Via Equation 2,
$v_2(y_2) = max_{y_1}[s(x, 2, y_1, y_2) + v_1(y_1)]$
$= max_{y_1}[s(x, 2, y_1, y_2) + max_{y_0} \sum_{i=1}^{1} s(x, i, y_{i-1}, y_i)]$
$= max_{y_1}[s(x, 2, y_1, y_2) + s(x, 1, y_0, y_1)]$
$= s(x, 2, y_1, y_2) + s(x, 1, y_0, y_1)$

Inductive Hypothesis:
Assume $v_k(y_k) = max_{y_{k-1}}[s(x, k, y_{k-1}, y_k) + v_{k-1}(y_{k-1})]$ is true for $j = k$. We want to show it is true for $j = k + 1$

Inductive Step:
$v_{k+1}(y_{k+1}) = max_{y_k}[s(x, k + 1, y_k, y_{k+1}) + v_k(y_k)]$
By our Inductive Hypothesis,
$v_{k+1}(y_{k+1}) = max_{y_k}[s(x, k + 1, y_k, y_{k+1}) + max_{y_{k-1}}[s(x, k, y_{k-1}, y_k) + v_{k-1}(y_{k-1})]]$
$v_{k+1}(y_{k+1}) = max_{y_k}[s(x, k+1, y_k, y_{k+1}) + max_{y_{k-1}}[s(x, k, y_{k-1}, y_k) + max_{y_{k-2}}[s(x, k-1, y_{k-2}, y_{k-1}) + v_{k-2}(y_{k-2})]]$
$v_{k+1}(y_{k+1}) = max_{y_k}[s(x, k+1, y_k, y_{k+1}) + max_{y_{k-2},y_{k-1}}[s(x, k, y_{k-1}, y_k) + s(x, k-1, y_{k-2}, y_{k-1}) + v_{k-2}(y_{k-2})]]$
$v_{k+1}(y_{k+1}) = max_{y_k}[s(x, k + 1, y_k, y_{k+1}) + max_{y_1,\ldots,y_{k-1}}[\sum_{i=1}^{k} s(x, i, y_{i-1}, y_i)]]$

By Equation 1,
$v_{k+1}(y_{k+1}) = max_{y_k}[s(x, k+1, y_k, y_{k+1}) + v_k(y_k)]$

Thus proved for $j = k + 1$.

2. The viterbi algorithm runs in a time of $O(Tn^2)$ given a sequence of T observations from a HMM that has n states. The viterbi algorithm considers all paths to each node and takes the maximum probability path and discards all of the other options. Since for each layer there are n options or states and in total there are n layers for each word there are n^2 operations the algorithm needs to do. Given that there are T words or observations in total the viterbi algorithm runtime is $O(Tn^2)$.

3 Programming: Implementing the Viterbi algorithm

3.1 Coding the Viterbi Algorithm

Model Description In this third problem we implemented the viterbi algorithm by completing the decode method in the assigned starter code. The decode algorithm takes in the input length, set of tags minus start and stop, and the scoring function that returns a score given the previous tag, current tag, and index of the tag. In order to implement this algorithm we created a matrix for the viterbi variables that score the best score at the location of the tag, and a backmarker matrix to store the path leading to the best score at the location of the tag. Our implementation first computes the viterbi variables, and backmarkers that lead to the first set of tags from the start tag. Then the algorithm runs through the rest of the length of input and calculates the max of all the possible options to reach a tag and utilizes dynamic programming to provide the correct viterbi variables and backmarkers. Finally the path is computed by iterating through the tags and computing the best way to reach the stop tag. Finally the decode method returns the correct path.

Performance The model can be fully trained in around 23 minutes.

Experimental Procedure We first began by uncommenting the test decoder method and implenting the algorithm step by step in order to provide the correct result. Once the max score of 14 was obtained we ran the algorithm on the main train function and recorded the results.

Deliverables

1. Report the precision, recall, and F1:

processed 51578 tokens with 5917 phrases; found: 4891 phrases; correct: 3769.
accuracy: 69.56%; (non-O)
accuracy: 94.23%; precision: 77.06%; recall: 63.70%; FB1: 69.74 LOC: precision: 96.76%; recall: 65.19%; FB1: 77.90 1233 MISC: precision: 88.18%; recall: 72.65%; FB1: 79.66 753 ORG: precision: 62.77%; recall: 69.65%; FB1: 66.03 1488 PER: precision: 69.02%; recall: 53.38%; FB1: 60.20 1417

2. Test and Dev

accuracy: 69.56%; (non-O)
accuracy: 94.23%; precision: 77.06%; recall: 63.70%; FB1: 69.74

LOC: precision: 96.76%; recall: 65.19%; FB1: 77.90 1233
MISC: precision: 88.18%; recall: 72.65%; FB1: 79.66 753
ORG: precision: 62.77%; recall: 69.65%; FB1: 66.03 1488
PER: precision: 69.02%; recall: 53.38%; FB1: 60.20 1417
processed 46666 tokens with 5616 phrases; found: 4181 phrases; correct: 2858.
accuracy: 57.83%; (non-O)
accuracy: 91.36%; precision: 68.36%; recall: 50.89%; FB1: 58.34
LOC: precision: 92.28%; recall: 63.15%; FB1: 74.98 1140
MISC: precision: 73.57%; recall: 62.34%; FB1: 67.49 594
ORG: precision: 62.33%; recall: 57.56%; FB1: 59.85 1521
PER: precision: 45.46%; recall: 26.28%; FB1: 33.31 926

3. Located in zip as:

   `asg3_Viterbi.py`