CSE 143 Assignment 3

WRITEUP

Ojas Upadhye, Sean Fujiwara, Sriharsha Madala, Tanay Khajanchi

**1.** Text Classification

Model Description:

Goal: Define a sentiment analysis model using Simple RNN, and train a model on the IMDB movie dataset using the compiled model and training set.

* Sentiment analysis dataset: The code loads the IMDB movie reviews dataset using Keras’ `imdb.load\_data()` function.
* Encoding and padding: the dataset and the words contained are encoded using integer IDs and padded to have a fixed length.
* Model creation - details: along with the Simple RNN, the model uses sigmoid activation to calculate the output.
* Model compilation: model is compiled with the binary cross-entropy loss and trains it on the training set.
* Examples of pre trained embeddings from TensorFlow Hub

1.1 With Simple RNN:

Final training accuracy: 0.9453

Training Loss: 0.1447

Accuracy on test set: 0.6834

Testing Loss: 0.9552

With simple recurrent neural network, the accuracy of the model with the training data was 0.9453 with the loss only being 0.1447. However, while the accuracy on the test set was 0.6834, the loss was high as well, getting a 0.9552. This disparity between the training data and testing data might be attributed to overfitting.

1.2 With LSTM:

Final training accuracy: 0.9489

Training loss: 0.1353

Accuracy on test set: 0.7353

Testing Loss: 0.9097

The LSTM model typically provides higher accuracy on longer sequences as they are equipped to retain and utilize information from distant past time steps. The results show a smaller training loss of 0.1353, and a higher final accuracy of 0.9489 compared to the RNN model. This can be attributed to the LTSM layers being able to capture contextual information from longer sequences before feeding it into the transformer layer. The final training accuracy for both models are similar, but the training and testing loss or accuracy numbers are favored for the LTSM model. The ideal model will depend greatly on the specific task and dataset, however the LTSM model is well equipped to handle the vanishing gradient problem.

**2. Viterbi Algorithm**

2.1 Derivation:

: Rewrite as the sum of the max score of jth tag and highest score over tag sequences of length j-1.

: Substitution by definition of Viterbi variable

2.2

The Viterbi algorithm first iterates over the k tags and for each of them sets the local score of the first token and the start token. This takes O(k) time. Then the algorithm iterates over each of the tokens in the sentences and for each token, iterates over the k possible tags to find the score for each combination of a word and tag. This takes O(mk) time where m is the number of words in the sequence and k is the number of tags. Since the score of the combination of the ith word and jth tag depends on the highest score over the sequences up to the i-1th word, the algorithm also needs to find the max over all the tags for sequences up to the i-1th word. This results in an overall runtime of O(mk2 + k) = O(mk2 ).

3. Implementing the Viterbi Algorithm

Model Description:

Goal: Find highest scoring tag sequence based on a scoring function.

Based on the skeleton code, our implementation of the Viterbi algorithm and its logic is pretty much entirely enclosed within the `decode()` function, the very first function defined in the file, `CSE-143\_A3\_Code.ipynb`.

* Using `Dictionaries`to help keep track of sequence tags
* For our scoring function, it takes into account the current tag, the previous tag, and the position of the token in the input sequence
* Iterate over all tags, calculating the maximum score and keeping track of the previous scores. Then dynamically choose the highest score from all previously seen scores.
* The score is the sum of the previous position's score for the tag and the score of transitioning from the tag to "<STOP>".

|  | **Precision** | **recall** | **F1** |
| --- | --- | --- | --- |
| **Dev Set** | **59.80** | **41.25** | **48.82** |
| **Test Set** | **53.28** | **37.41** | **43.96** |

**Dev set output:**

accuracy: 41.38%; (non-O)

accuracy: 89.61%; **precision: 59.80%; recall: 41.25%; FB1: 48.82**

LOC: precision: 87.53%; recall: 58.31%; FB1: 69.99 1219

MISC: precision: 69.94%; recall: 63.89%; FB1: 66.78 835

ORG: precision: 36.04%; recall: 42.65%; FB1: 39.07 1587

PER: precision: 49.43%; recall: 11.90%; FB1: 19.18 441

**Test set output:**

accuracy: 37.73%; (non-O)

accuracy: 88.02%; **precision: 53.28%; recall: 37.41%; FB1: 43.96**

LOC: precision: 86.52%; recall: 55.88%; FB1: 67.91 1076

MISC: precision: 54.45%; recall: 50.64%; FB1: 52.48 652

ORG: precision: 37.26%; recall: 44.93%; FB1: 40.74 1986

PER: precision: 32.75%; recall: 4.68%; FB1: 8.19 229

The results show that the algorithm was implemented successfully.