CSA 250 : Deep Learning Project III
Report

Magare Aadesh Gajanan (15605)

April 21, 2020

1 Task

The project is to implement a deep network for the task of Natural Language Inference. Given two sentences, we will predict if the sentence pair constitutes entailment/contradiction/neutral. We'll build a simple Logistic regression classifier using TF-IDF features and a deep learning model specific for text like LSTM.

2 Dataset

We would use one of the standard datasets for this task called Stanford Natural Language Inference (SNLI). We would primarily use the files snli_1.0_train.jsonl for training the model and snli_1.0_test.jsonl for evaluating the performance of the model. In each of these files the relevant fields to be considered are gold_label, sentence1 and sentence2.

Where sentence1 is the first sentence of the pair. This is also called premise. sentence2 is the second sentence in the pair. This is also called hypothesis and gold_label represents the target label.

3 Approach

I tried two different architectures for the task i.e. simple logistic regression classifier using TF-IDF features and deep learning based model like LSTM. For both the approaches I'll present how I arrived on final architecture, justify different architectural choices and hyper parameter tuning strategy.

3.1 Logistic Regression Classifier using TF-IDF Features

The training data consists of two sentences and a corresponding label. We first take the TF-IDF features for both the sentences of a training example and concatenate them. The resultant vector is passed through a simple logistic regression classifier to predict the label. The simple baseline model gives an accuracy of **55.09 %** on the validation set. Figure 1 shows the training and validation loss during the training process.

Next we'll explore some architectural changes in the model to achieve better performance. Starting with pre-processing, we'll pre-process the input to better differentiate between the premise and hypothesis. Each word from premise is appended with token 's1_' while each word from hypothesis is appended with token 's2_'. The intuition behind the approach is to have two different vocabularies, separate for premise and hypothesis, allowing to capture different meanings of the word depending on whether it appears in premise or hypothesis. Following the pre-processing the same model gives a validation accuracy of **64.61%** i.e. an increase of around **9.5%**.

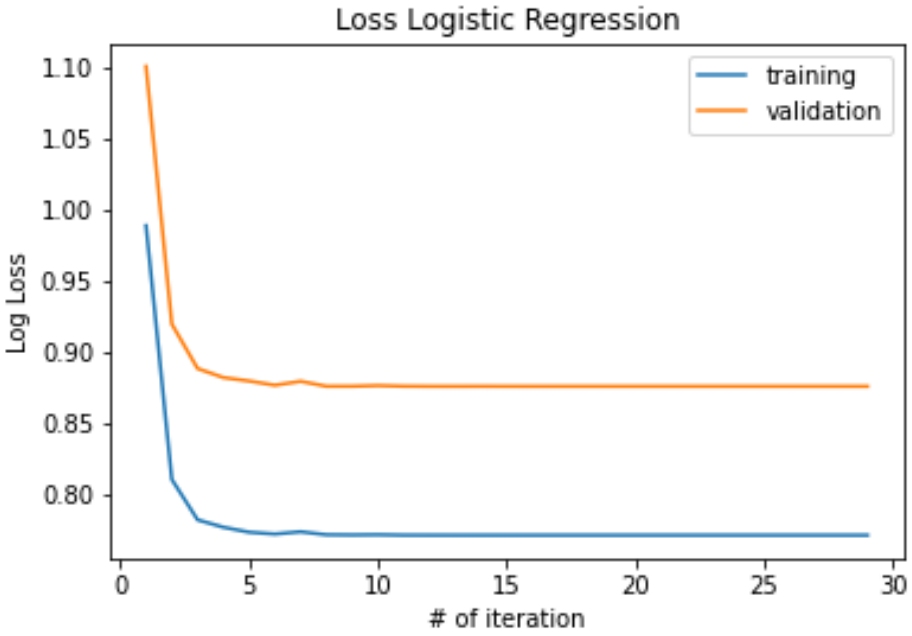

Figure 1: Logistic Regression Loss

Then We'll try removing the stop words which is standard practice in NLP, however it gives no improvement in model performance but further decreases the accuracy by 0.5%.

Next we'll employ stemming by using NLTK package, stemming helps by only a slight amount with model performance and the resultant model accuracy is slightly higher (0.07%). However lemmetization does not help in this case and further reduces model accuracy.

Hyper-Parameters Tuning After choosing the best model so far i.e. we'll tune the remaining hyper-parameters.

1. **ngram_range:** We'll try out different commonly used values of ngram for CountVectorizer i.e. (1, 1), (1, 2), (2, 2).
2. **stop_words** whether to remove stop words or not
3. **TF_IDF** Several hyper parameters associated with TF-IDF such as whether to use IDF or not, smooth IDF or not and the penalty to be used l1, l2 or None.
4. **Solver** What solver to be used in logistic regression classifier, we'll try several values such as 'newton-cg', 'sag', 'saga', 'lbfgs' along with the choice of regularization for each one of them.

Grid Search is used for searching optimal hyper parameters. The resultant model gives validation accuracy of **67.42%**. The optimal hyper parameters for the model are ngram_range of (1, 2), removing English stop words, using IDF and the solver of type 'saga'.

Finally we take the best performing model with tuned hyper parameters and test the performance on test set. Here it gives **accuracy of 66.41%**. Figure 2 shows the confusion matrix for test data. The dataset has several examples without label i.e. the corresponding label is '-'. Removing such examples from test set gives slightly better performance of **67.59%**. The confusion matrix shows the results after removing such examples.

Model	Validation Accuracy (%)
Baseline	55.09
preprocessing with s_1 and s_2 tokens	64.61
stemming	64.68
Best model + tuned Hyper parameters	67.42

Table 1: Summary of different architectural choices (Logistic Regresion)

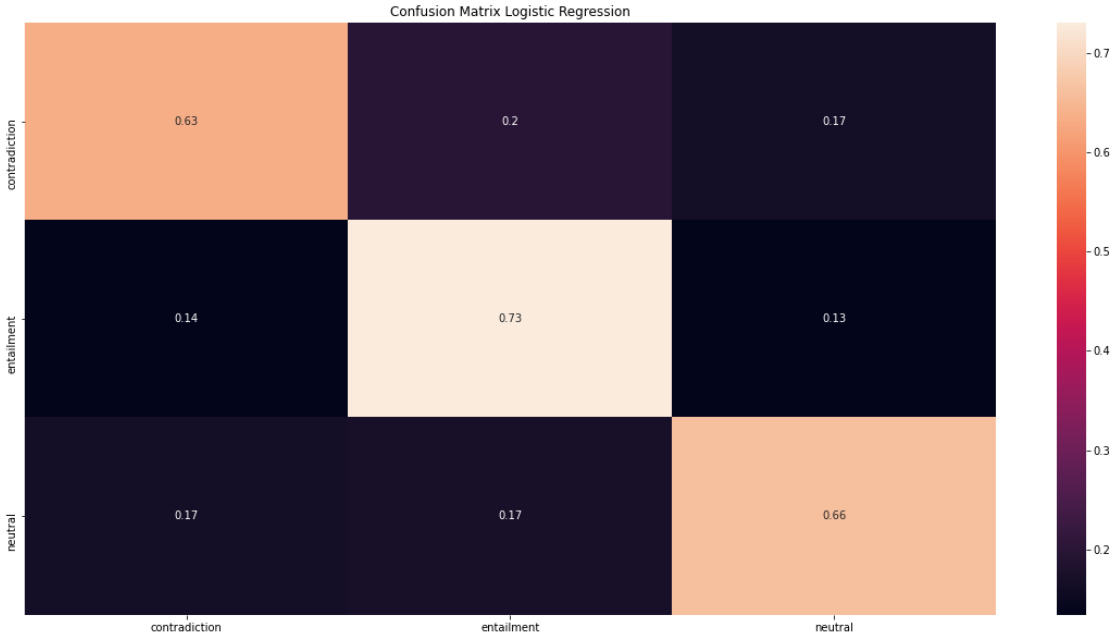

Figure 2: Logistic Regression Confusion Matrix

3.2 Deep Neural Network based approaches

Next we'll experiment with NNs specialised for text or sequential data and will follow the same approach as logistic regression based model i.e. We'll start with simple baseline model and gradually improve it.

Starting with simple LSTM based model. We'll have an embedding layer followed by projection and LSTM layers. Final output layer is an sequential model with couple of fully connected layers. It's a multi-class classification problem thus I'll be using CrossEntropy loss. ReLu is good default choice for activation function to begin experiment, we'll see how other activation functions compare. I have used Adam optimizer with 1e-3 learning rate, which will be tuned during hyper parameter tuning phase.

This baseline model gives accuracy of **77.16%** on the validation set. Figure 3 shows the training loss and validation loss decreasing with no of epocs. Training is stopped at 10 epocs due to limitations of training resources. We'll stick to 10 epocs of training for further experiments for fair comparison.

Next we'll increase the capacity of network by adding additional layers and model complexity. Fully connected layers before output layer of the model can be increased for more model capacity. We'll vary the fully connected layers from 1 to 3 before the output layer.

Further the LSTM layer used can be enhanced with stacked LSTM making it 2 or 3 layer LSTM. Apart

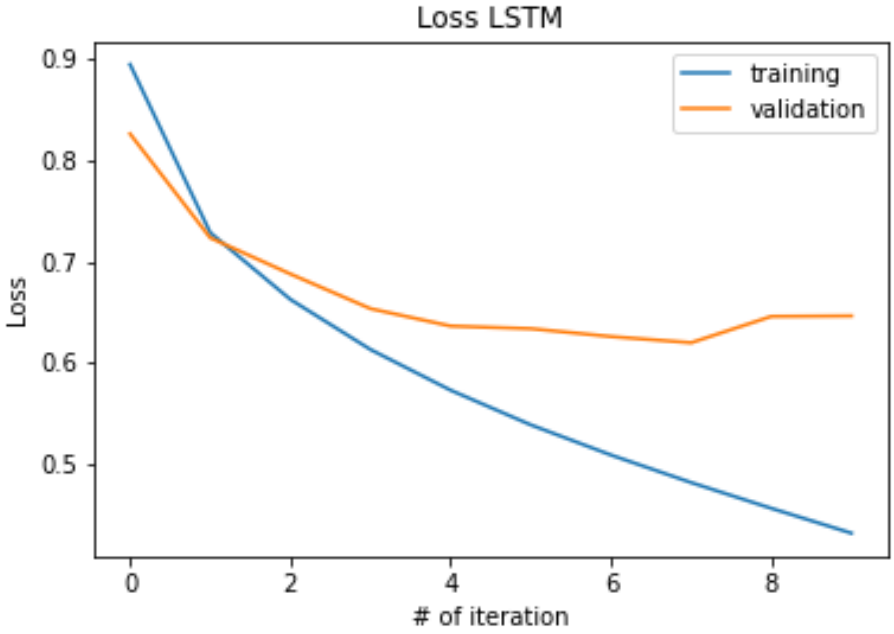

Figure 3: LSTM Loss

from this we can use BiLSTM in place of simple LSTM layer. Apart from this we'll experiment with the use of pre-trained word embeddings like Glove. Apart from LSTM, the core model can be changed to GRU or a TransformerEncoder. Due to lack of training resources we're restricting the training to 10 epocs thus a very large model like transformer is not able to show expected performance and the performance is less than that of LSTM.

We'll choose the best performing architecture giving validation accuracy of **77.47%** out of this for further experiments involving hyper parameter tuning.

Regularization: Following forms of regularization were tried, it was also observed that combining multiple forms of regularization does not help to improve performance.

1. **L2 penalty** L2 regularization is a widely used regularization method for linear models. It's rarely used in DNN models and here the resultant model shows no performance gain with it but further reduces accuracy by about 1%.
2. **Dropout** It's the de-facto regularization method for many tasks and shown to prevent over-fitting in wide variety of network architectures. Here the resultant model shows an increase of 0.6% to 0.8% of increase in accuracy.

Next we try different activation functions i.e. Sigmoid, Tanh, ReLU. as expected ReLU performs the best from the alternatives. Sigmoid gives 2.6% reduction in accuracy where as using tanh drastically hinders the model performance.

Hyper-Parameters tuning After choosing the best model so far i.e. Bidirectional LSTM model with just 1 layer and dropout regularization along with ReLU activation function, we'll tune the remaining hyper-parameters.

1. **Learning rate** is probably the most important hyper parameter, I've started with good default of 1e-3 and used "ReduceLROnPlateau" learning rate scheduler in PyTorch. It gradually decreases

the learning rate upon reaching a plateau. Here for the scheduler to work better, the model is trained for 15 epocs. Resultant model gives accuracy of 78.30% on validation set.

2. **Batch Size** affects both the training speed and the accuracy of resultant model, I have tried several values for batch size and 512 gives the best results.

After all hyper-parameter tuning and using learning rate scheduler we get a validation accuracy of **78.30%.**. Table 2 gives the summary of overall experiments.

Model	Validation Accuracy (%)
Baseline (1 conv layer)	91.03
LSTM Based	77.47
GRU Based	75.74
TransformerEncoder Based	33.80
Stacked LSTM Based	76.86
LSTM Based with tuned HP	**78.30**

Table 2: Summary of different architectural choices (NN)

Finally we take the best performing model and test the performance on actual test set. Here it gives **accuracy of 77.63%**. Figure 4 shows the confusion matrix for test data.

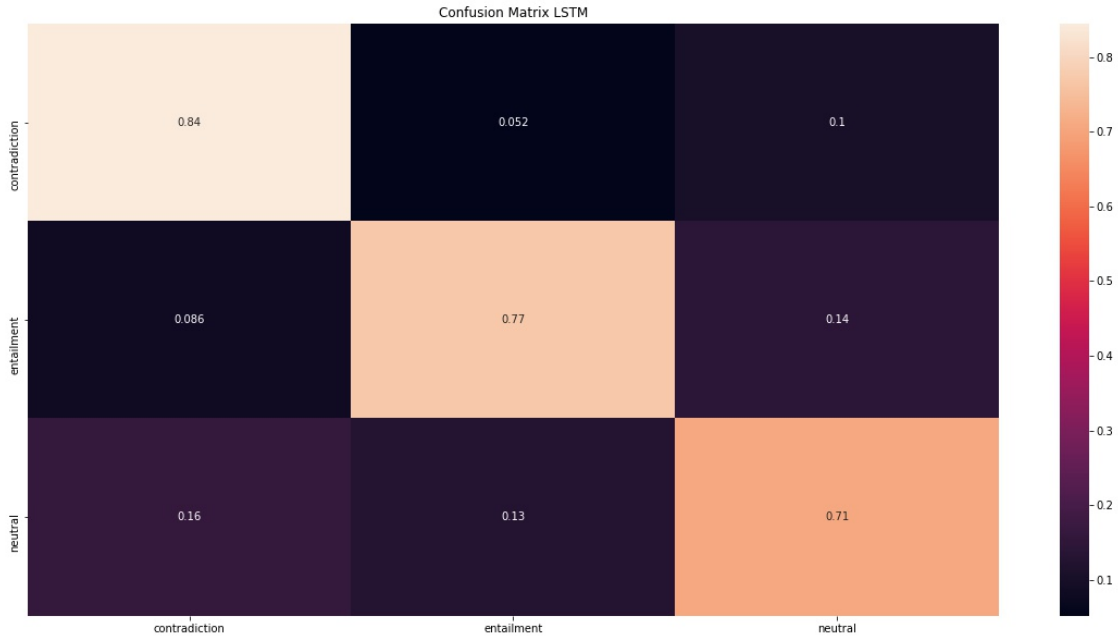

Figure 4: LSTM Confusion Matrix

Table 3 shows the performance of both logistic regression based model and LSTM based model on the actual test dataset. LSTM model clearly performs better than logistic regression model with **11%** better accuracy, which clearly shows it's superiority for textual data and preserving long term dependency.

Model	Test Set Accuracy (%)
Logistic Regression Based	67.59
LSTM Based	77.63

Table 3: Result on test dataset