DEEP LEARNING ASSIGNMENT 3

MANASVI AGGARWAL
MTECH(RES)
manasvia@iisc.ac.in
16223

May 15, 2020

TASK 1: LOGISTIC REGRESSION

In this part of the assignment, I implemented Logistic regression using TF-IDF features in python for the task of Natural Language Inference on SNLI dataset. Basic preprocessing is carried out on the data such as tokenizing or removing stop words. I trained a logistic regression model using TF-IDF features which are formed by using TfidfVectorizer. After fitting the model on the train set I further predict the accuracy on the test set. I tried different parameter settings of logistic regression model and set the values based on the validation set performance. I varied the max_iter parameter in {100, 150, 200, 250}. The performance for all these values is nearly same on both the validation as well as the test set. Also, I tried different values for penalty parameter such as {'l1', 'l2', 'None'} but the performance of my model remains the same on both sets. On test set logistic regression gives 63.30% accuracy. On validation set accuracy is 63.59%.

TASK 2: DEEP MODEL

In this task we have to build a Deep model specific for text like RNN/GRU/LSTM or any of the new approaches like Transformers, BERT etc for NLI (Natural Language Inference) i.e. to predict if the sentence pair constitutes entailment/contradiction/neutral for SNLI dataset. Some pre processing of the data is done as carried out in task 1 too. I implement a Recurrent Neural Network for this task. For better performance of the model I used glove pretrained embeddings to convert words to a d-dimension vectors. I also tested without these pretrained embeddings. The observation is that the model performance is improved by 2% when using the pretrained vectors. These pretrained vectors are available on `https://www.kaggle.com/chenwgen`. RNN is build to encode both the left (hypothesis) and right (premise) parts. Finally, a relu layer is augmented which is followed by a final softmax layer for the prediction.
This model has various hyperparameters such as:

- Number of Epochs
- Activation Function
- Number of Layers
- Learning Rate
- Patience Paramter

For tunning these hyperparameters, I tried various ranges. For learning rate I tried {0.01, 0.001, 0.0001} values. For hidden dimension I tried {64, 128, 256, 300, 500, 512}. Various different activation functions are tried such as relu, sigmoid, elu, tanh etc. Further, the hyperparameter #layers is searched in {1, 2}. Based to the performance on validation set, I have set the learning rate as 0.001, activation function as relu, number of layers as 1 and number of epochs as 30 with patience parameter fixed at 4. The hidden dimension is fixed at 300.

The performance of my model with the values of parameters set as described above is 83% on train set, 82.45% on validation set, and on test set it is able to achieve 81.32%. This is when using Glove pretrained vectors. The performance when no pretained embeddings are used is 81% on the model performance is 84.44% on train set, 81% validation set and 79.% on test set. On my repository I have saved the model which uses Glove pretrained embeddings.

Requirements to run the code To run the code, we need to download these two files from the links given below:

- Link `https://drive.google.com/open?id=127a06iyy2Sm4bXN8GC2aByPal_D2ArwA` has the complete snli dataset. To run the code this folder should be present in the same directory as the running code.
- Link `https://drive.google.com/open?id=1HJGpooMPc3Jj76htcqrJijvnHUUVR6Dd` has the pretrained weight matrix for task 2 (deep learning model). We need to keep this file in the same directory as the code.

Figure 1 depicts the model loss for different learning rates i.e. for learning rate 0.01, 0.001 and 0.001. For this experiment the #epochs is set to 30. Figure 2 shows the loss trend and Figure 2 shows the accuracy trend on training and validation sets during training stage.

Below are some plots corresponding to the hyperparameters of the model. In each plot, I vary one of the hyperparameters and keep others constant and analyse how the performance of the model or the accuracy changes.

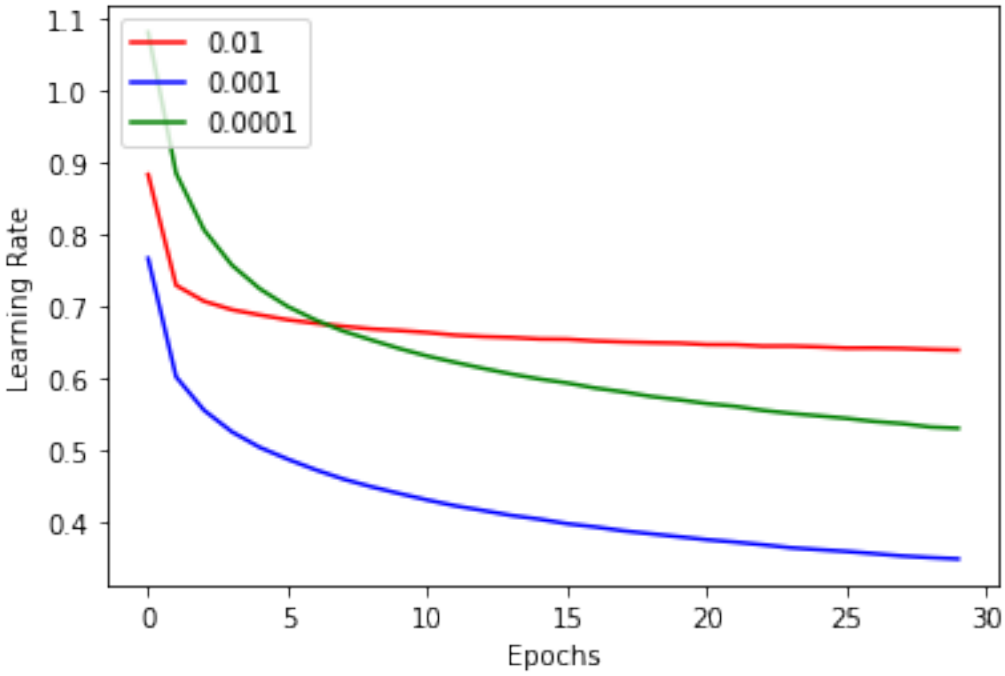

Figure 1: Epochs vs Learning Rate

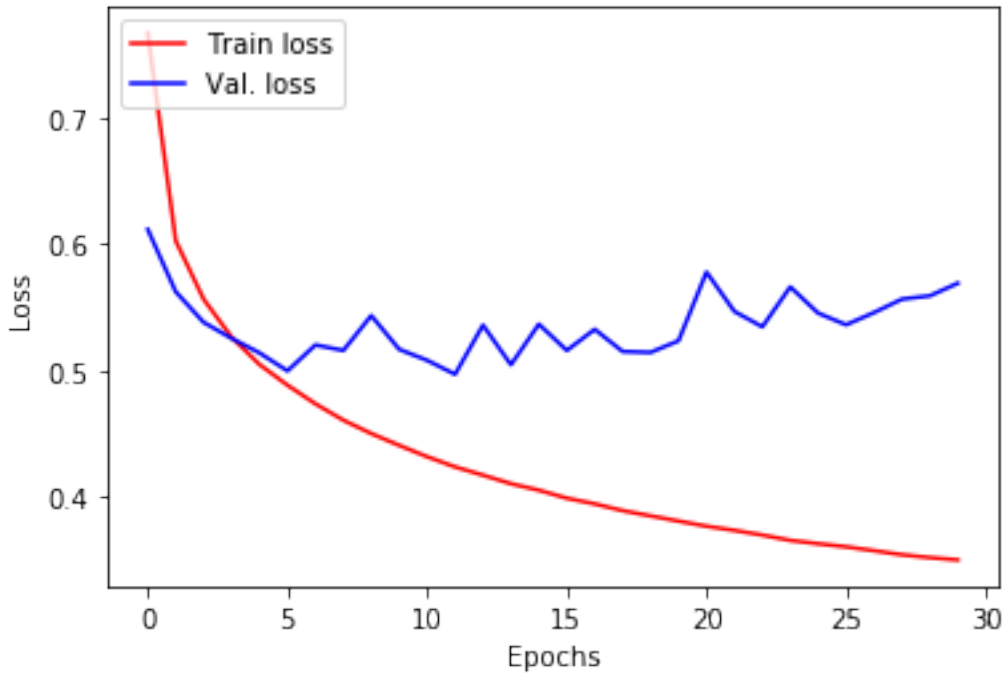

Figure 2: Epochs vs Loss for training and validation sets.

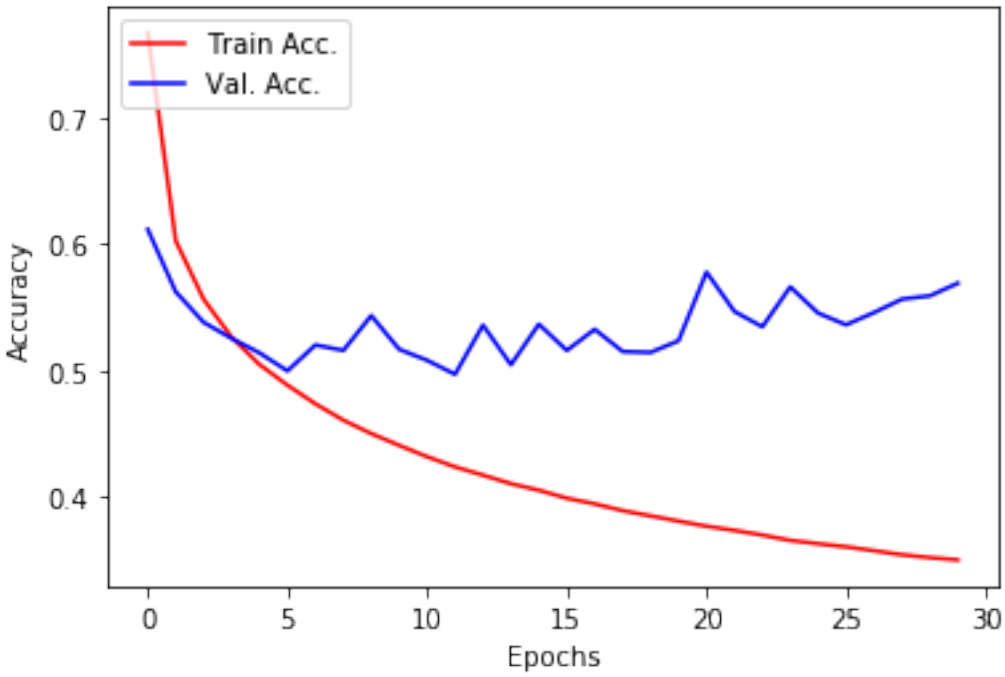

Figure 3: Epochs vs Accuracy for both training and validation sets.

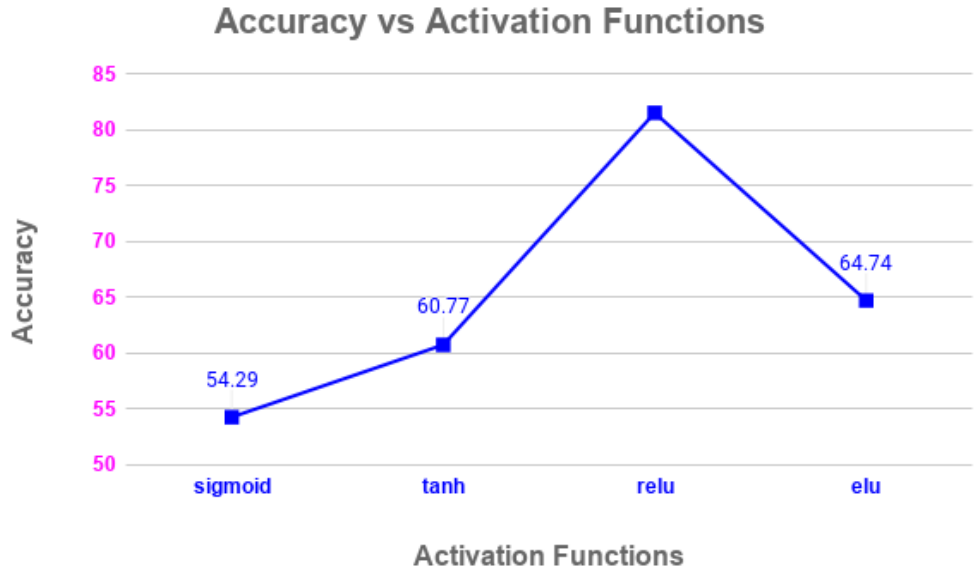

Figure 4: This plot shows the model accuracy for various activation functions.

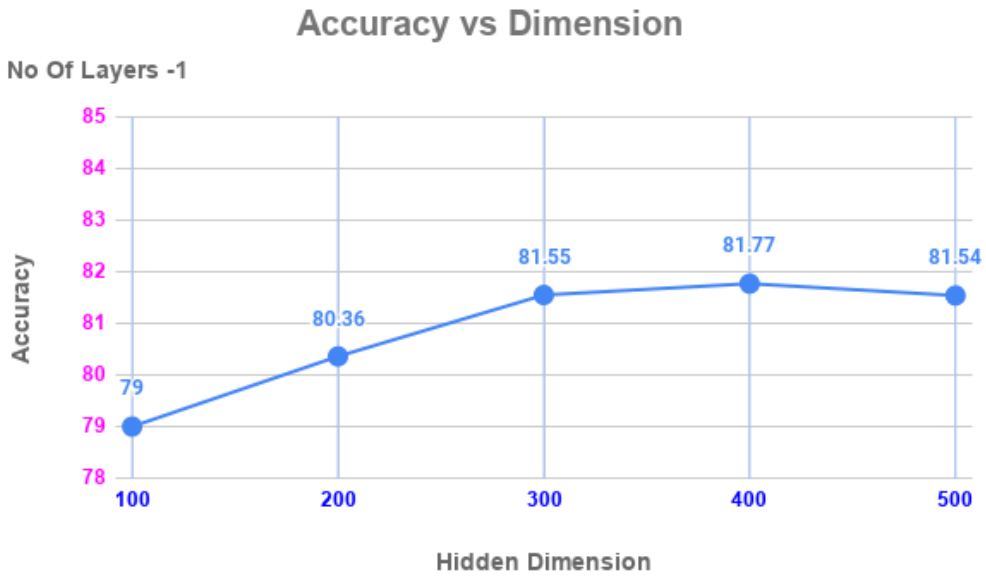

Figure 5: This plot shows the model accuracy for different dimensions.

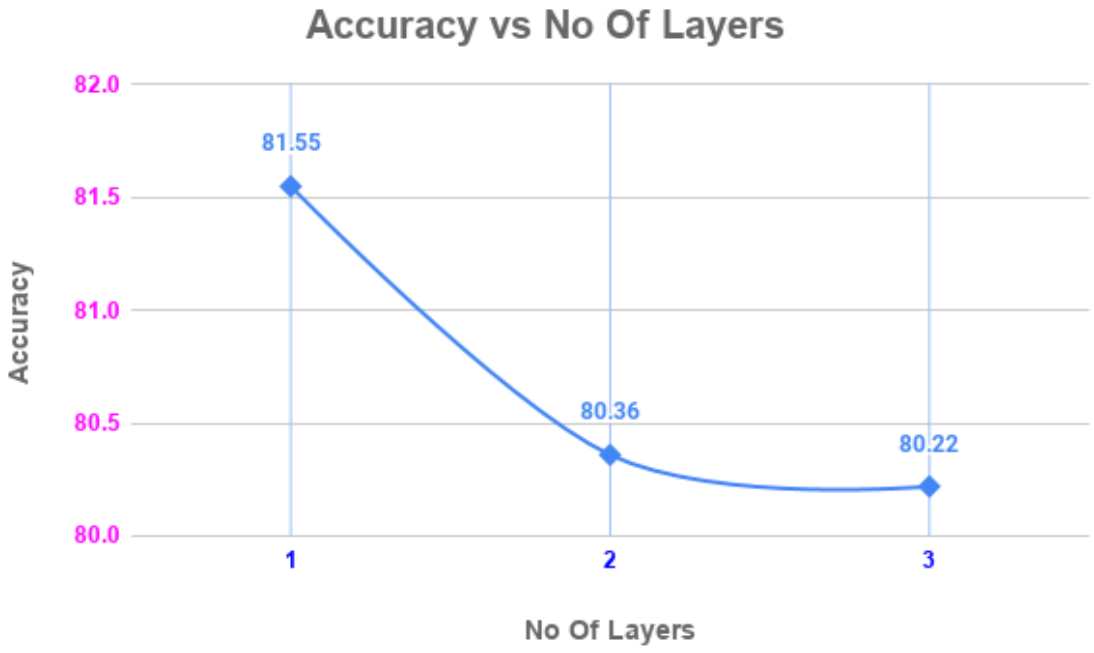

Figure 6: This plot depicts the model accuracy by varying the number of layers.