Assignment 2 in Artificial Intelligence II 2022-2023

Dimitris Orinos ic121006-7115132100006

In this exercise the main purpose is to create a classifier which can predict if a movie review has a negative or positive review based on ratings from 1.0 to 10.0, i.e for ratings above 7.0 the movie has a positive review and a movie with rating below to 4.0. This classifier is created through a feed forward neural network.

More specifically the steps that are followed in order to achieve the creation of this classifier are described below:

1st step Download of the glove.6B.zip file, determination of reproducibility and creation of vocabulary

Firstly the official site of stanford http://nlp.stanford.edu/data/glove.6B.zip is downloaded through the !wget method and also the glove.6B.zip file is downloaded through !unzip glove.6B.zip. Through the function set_seed(), where we set the seed as seed=1234 and the results are the same (or almost similar) every time this program is run. Also from the glove file with the arrays of 300 embeddings for each word (glove.6B.300d.txt) a dictionary is created after splitting the whole content of the glove file, where the first part of the content i.e the words are the keys of the dictionary and its corresponding arrays with values is the value for each key of the dictionary. This dictionary consists the vocabulary that is going to be used later, in order to create the input of model.

2nd step Data preprocessing and creation of the inputs for the neural network from glove embeddings

The next step refers about the ‘cleaning’ of our text. This means that the text must be emptied from duplicate words, punctuation, stopwords, numbers and whitespaces. All these are removed from every text review of the dataset of this exercise by using nltk library or other python libraries such as re. Also from nltk through word_tokenize, the tokenization of a text review into words is made. For every change that is made to each text the review column is renewaled through the method apply().

The last of all these columns is going to be used in order to create for each text review a vector using glove embeddings. More analytically after the end of data preprocessing and through the vocabulary that it is created in the 1st step, for each review a vector is created from the count of the corresponding arrays of 300 values for each word that exists in the corresponding vocabulary for the glove.6B.300d.txt that is used divided by the number of the arrays that are summed. All the words that there are not in the vocabulary are omitted. Time for vectorization and preprocessing = 38 seconds.

3rd step Splitting of the dataset converting train and validation arrays to tensors and creation of dataloaders

The splitting of the data set into train and validation dataset is made in the begining when the csv file is read, which in this split the numpy arrays made through numpy.asarray() are transformed into tensors and through .float() the train tensors are converted to float. Finally for each one of the train and validation dataset through pytorch TensorDataset and DataLoader, where it combines a dataset and a sampler, and provides an iterable over the given dataset, [1] the final dataset is created.

4th step Creation of the feed forward neural network

Another important step is the creation of the feed forward neural network that can help into the classification of the movie reviews into negative and positive. In order to do that referring to the netural network one thing that is used for many experiments in order to find the best neural network for the movies classification is the number of hidden layers. Also the activation functions ReLU, which guarantees that the output will always be positive and also they are applied after linear transformations to introduce nonlinearity, helping neural networks learn a wide variety of phenomena [2,3,4,5,6].

Also the dropout technique is used, which is a very effective technique for regularization and preventing the co-adaptation of neurons [7]. It is defied also a more general model class of the netural network, so it is more easy to pass different parameters, such as number of hidden layers, number of neurons, activation functions to have more experiments. Always the ouput of the netural network is 1 neuron and

the input is equal to 300 neurons, i.e the dimension of the glove embeddings that are used.

5th step Train of our model and explanations of the results

For the training of this model very important is the Dataloader which is created in the previous step, which as referred in the previous step iterates the 80% of the csv file i.e. the training set which is splitted in batches with size 64. In this iteration over each batch loss from is calcuated from Binary Cross Entropy Loss function [8]. About the model that is created in the 4th step, there is the ability for experimenting with many hyperparameters such as the type of optimizer, the learning rate value, which shows how much to update models parameters at each batch/epoch [4], the number of neurons in each layer or the number of layers, the dropout propability [7]. All these iterations for the dataloader are implemented in 21 epochs, which is an good number in order to see how each model and its metrics progress, in order to avoid overfitting or underfitting.

In the training iteration the vector of reviews and reviews ratings (2nd step) a for every batch requires grad. It is important to refer that in the iteration of the training set the optimization consists of the steps below [4]:

- Call optimizer.zero_grad() to reset the gradients of model parameters. Gradients by default add up; to prevent double-counting, we explicitly zero them at each iteration.
- Backpropagate the prediction loss with a call to loss.backward(). PyTorch deposits the gradients of the loss w.r.t. each parameter.
- Once we have our gradients, we call optimizer.step() to adjust the parameters by the gradients collected in the backward pass

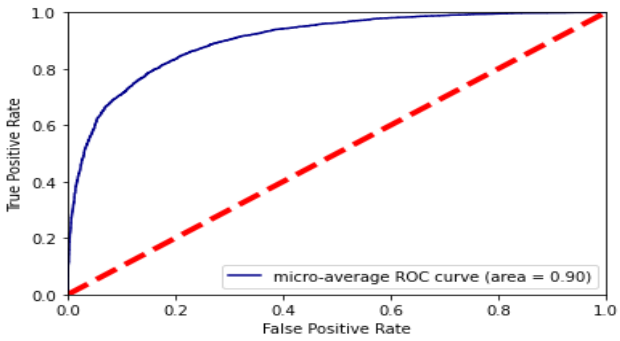
Also in the neural network the basic module that is used is the linear which applies a linear transformation on the input using its stored weights and biases [5]. In this exercise the early stopping is used which helps to achieve a better convergence time for each model , because after a specific number of epochs(in this exercise the limit of patience is two regardless if they are consecutive), if the validation loss is increased i.e is not improved then the model training is stopped to avoid overfitting [3,9,10,11]. After the training of the model, in the rest 20% of the csv dataset which does not require grad, i.e the validation set ,evaluations of the model are made with the help of sigmoid function

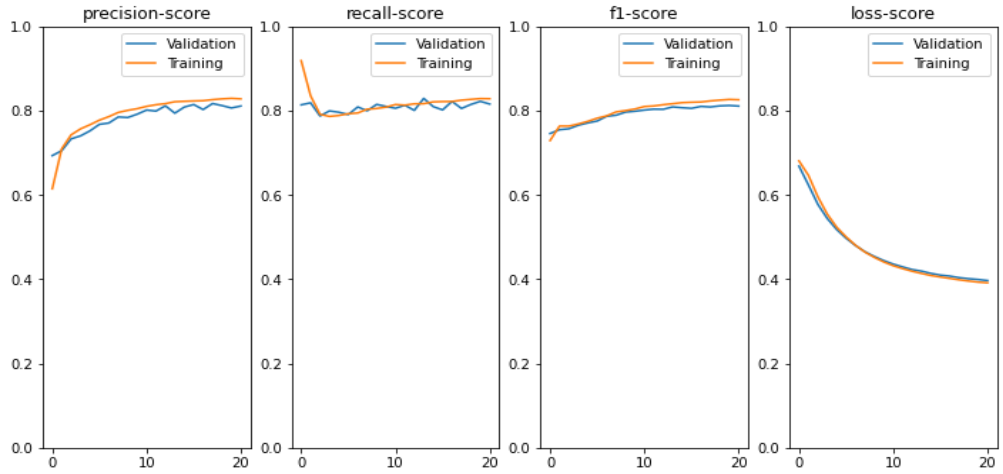
in order to make the prediction numbers probabilities and also calculate metrics from sklearn such as precision, F1-score, recall, accuracy and roc curve [3,12,13].

Below the best experiments for the neural network that is created are presented with their learning curves about their loss and also their roc curves[13]. There is presented the results of the last epoch, which is a very representative sample in order to explain better the results that occur:

1. Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of hidden layers = 1 with size 8, total time running (with calcuations, ROC and learning curves): 58 seconds

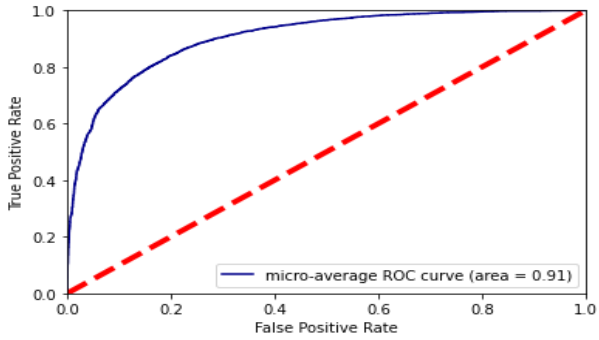
Performance metric	**Last epoch value due to early stopping(21st)**
Loss function (train)	0.39135
Precision (train)	0.82773
Recall (train)	0.82804
f1-score (train)	0.82565
Acurracy (train)	0.82674
Precision (validation)	0.39663
Recall (validation)	0.81073
f1-score (validation)	0.81542
Loss function (validation)	0.81057
Acurracy (validation)	0.81825

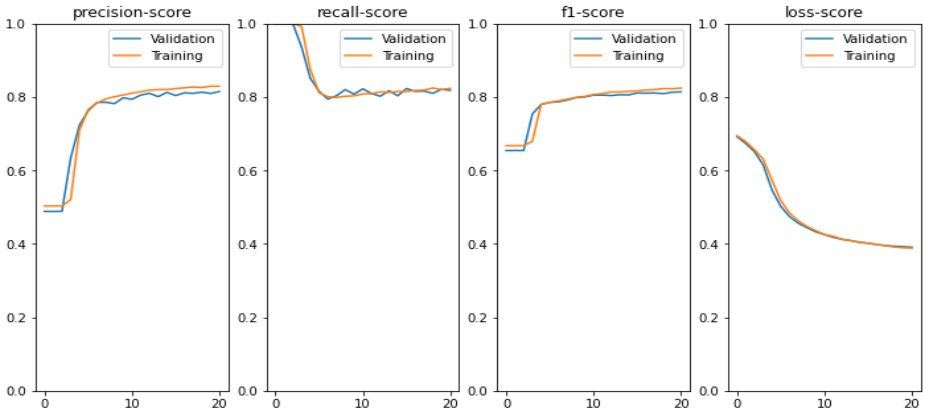




2. Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of hidden layers = 2 with sizes 8 both of them total time running (with calcuations, ROC and learning curves): 58 seconds

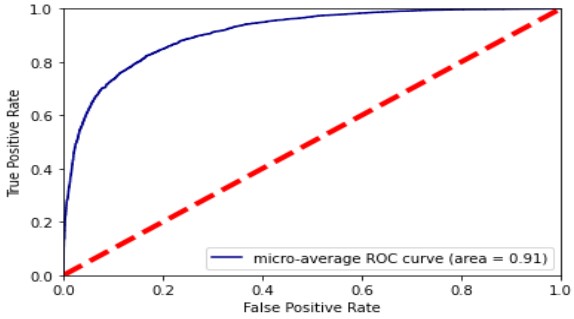
Performance metric	**Last epoch value due to early stopping(21st)**
Loss function (train)	0.38848
Precision (train)	0.82904
Recall (train)	0.82338
f1-score (train)	0.82406
Acurracy (train)	0.82562
Loss function (validation)	0.39088
Precision (validation)	0.81467
Recall (validation)	0.81805
f1-score (validation)	0.81375
Acurracy (validation)	0.81984

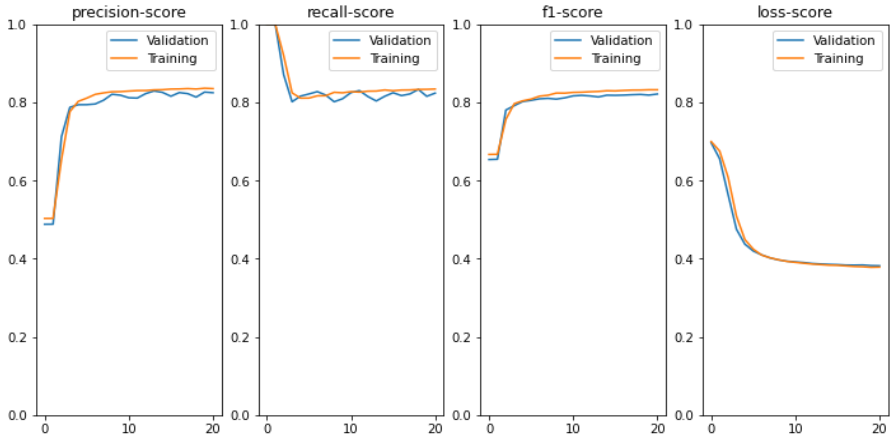




3. Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of hidden layers = 3 with sizes 8 all of them, total time running (with calcuations, ROC and learning curves): 1 minute (ROC curve is plotted in the 11th epoch)

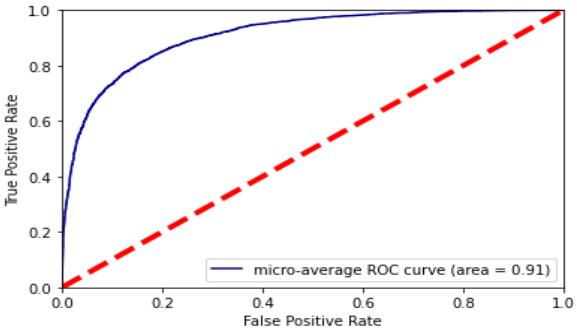
Performance metric	Last epoch value due to early stopping(21st)
Loss function (train)	0.37829
Precision (train)	0.83507
Recall (train)	0.83391
f1-score (train)	0.83207
Acurracy (train)	0.83354
Loss function (validation)	0.38178
Precision (validation)	0.82453
Recall (validation)	0.82353
f1-score (validation)	0.82117
Acurracy (validation)	0.82750

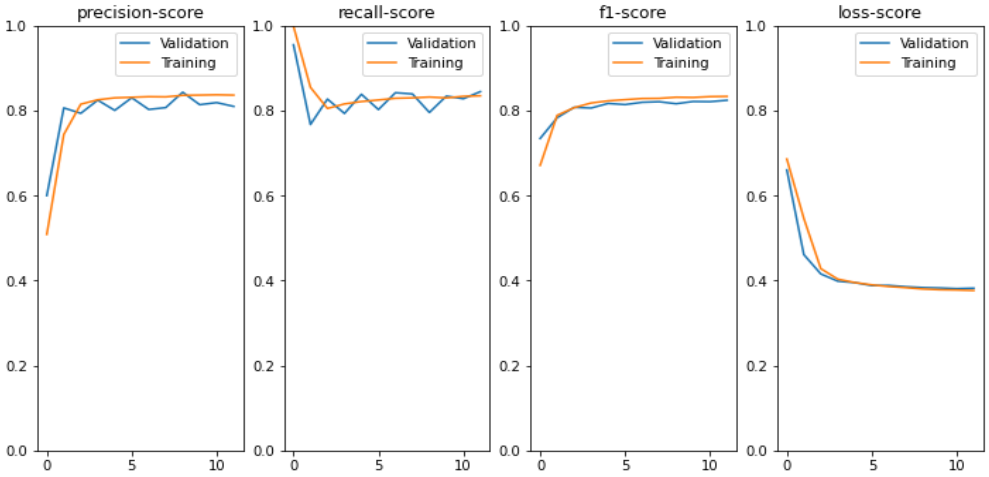




4. Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of hidden layers = 4 with sizes 32,16,8 and 8 , total time running (with calcuations, ROC and learning curves) : 37 seconds (ROC curve is plotted in the 11th epoch)

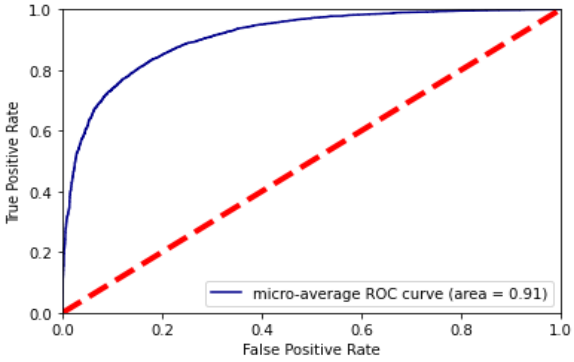
Performance metric	**Last epoch value due to early stopping(11th)**
Loss function (train)	0.37640
Precision (train)	0.83617
Recall (train)	0.83458
f1-score (validation)	0.83316
Acurracy (validation)	0.83448
Loss function (validation)	0.38186
Precision (validation)	0.80965
Recall (validation)	0.84438
f1-score (validation)	0.82407
Acurracy (validation)	0.82595

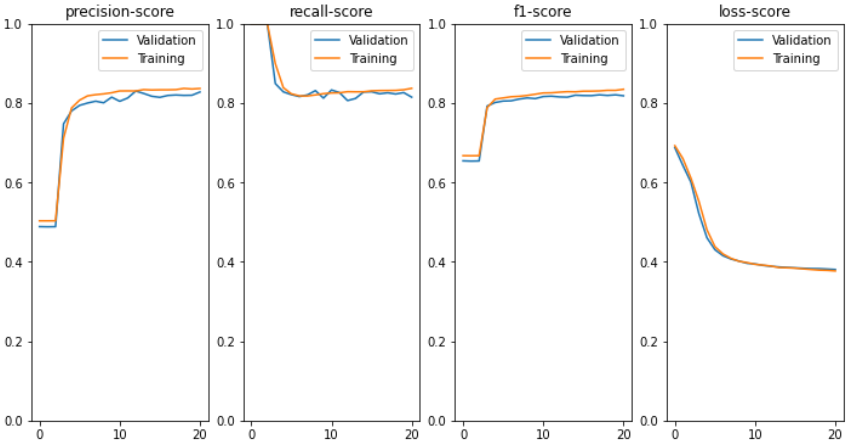




5. Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of hidden layers = 2 with sizes 16 and 8, total time running (with calcuations, ROC and learning curves) : 58 seconds

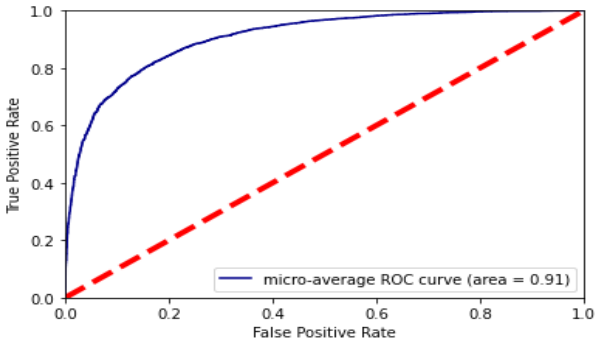
Performance metric	**Last epoch value due to early stopping(21st)**
Loss function (train)	0.37682
Precision (train)	0.83644
Recall (train)	0.83683
f1-score (validation)	0.83440
Acurracy (validation)	0.83526
Loss function (validation)	0.38044
Precision (validation)	0.82760
Recall (validation)	0.81456
f1-score (validation)	0.81807
Acurracy (validation)	0.82678

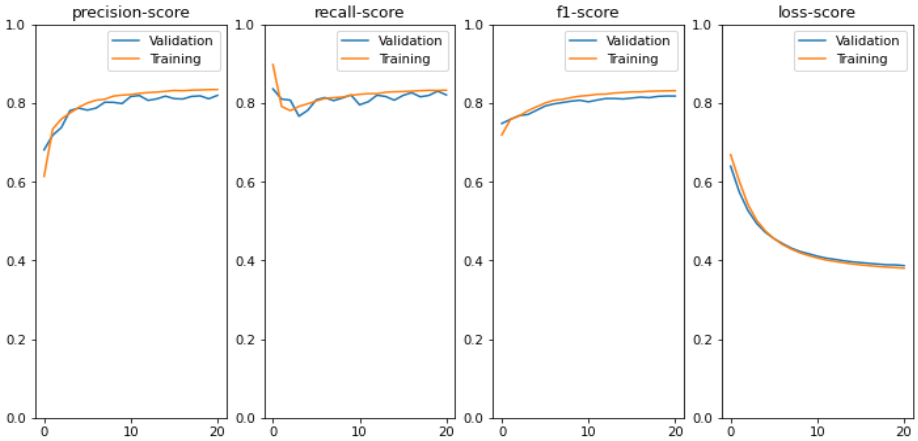




6. Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of hidden layers = 1 with size 16, total time running (with calcuations, ROC and learning curves) : 1 minute and 18 seconds

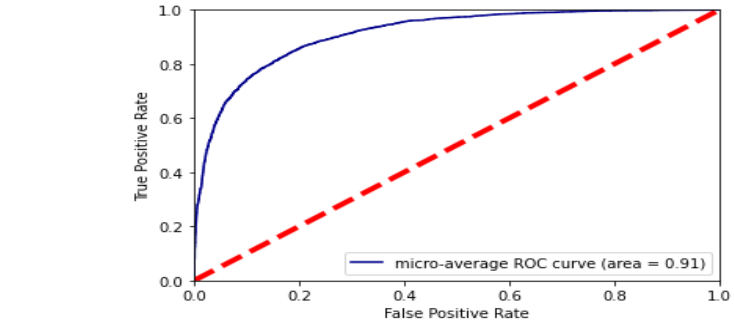
Performance metric	**Last epoch value due to early stopping(21st)**
Loss function (train)	0.38012
Precision (train)	0.83431
Recall (train)	0.83275
f1-score (validation)	0.83116
Acurracy (validation)	0.83300
Loss function (validation)	0.38679
Precision (validation)	0.81938
Recall (validation)	0.82056
f1-score (validation)	0.81769
Acurracy (validation)	0.82350

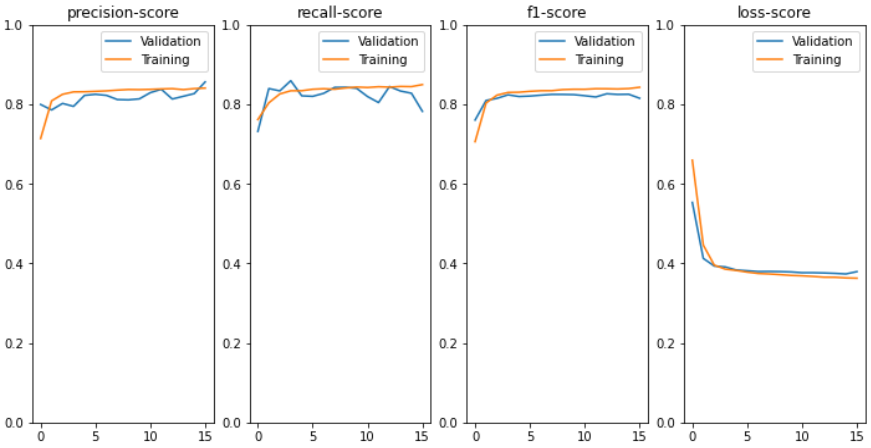




7. Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of hidden layers = 4 with sizes 64, 32, 16 and 8, total time running (with calcuations, ROC and learning curves) : 50 seconds (ROC curve is plotted in the 11th epoch)

Performance metric	**Last epoch value due to early stopping(16th)**
Loss function (train)	0.36262
Precision (train)	0.84052
Recall (train)	0.84927
f1-score (validation)	0.84253
Acurracy (validation)	0.84270
Loss function (validation)	0.37913
Precision (validation)	0.85609
Recall (validation)	0.78194
f1-score (validation)	0.81492
Acurracy (validation)	0.82932





The learning rate is not minimized or maximized a lot , basically it is keeping constant, in order to have a convergence into the global minimum with a normal speed referring to the gradient descent optimization[14]. Also it is observed that in each of the models that if we increase it a little, there is a trying to reach the global minimum more quickly. The Adam optimizer is preferred through other optimizer like sgd(stochastic gradient descent optimizer), because it is a faster optimizer about training of the model and also need smaller tuning of their learning rate hyperparameter (adaptive learning rate algorithm), something that makes more easier this optimizer into its use rather than sgd for example[15].

For the feed forward neural networks [13] with 1 linear layers there is generally no overfitting as it occurs from the learning curves for the binary cross entropy loss function [8]. One reason for that is that in the loss learning curve between train loss and validation loss between these two values observed a very small difference. About the loss all these are applied and in cases of having a network with 2,3 or 4 hidden layers, but it is occurred that in case that the number of neurons or the number of hidden layers is increased, then the risk of overfitting is getting bigger [3,16]. Also lager number of epochs could probably cause an overfit in the learning curves [3].

If we look all the examples the least good curve is about the recall but the f1-score which is the harmonic mean of precision and recall has a very good behavior as a curve

in all the examples and the values in the 20th epoch or in the epoch that the training stopped due to early stopping, are very close to each other that in some cases are converging. Generally the difference in the learning curves between precision and recall, is a consequence of the precision recall trade/off, i.e the f1 score is a good way to compare recall and precision [3], it is more important than recall and precision metric, but due to the way that the metrics are defined is logical that both of these metrics can not have the same good behavior, because if the recall is more increased, the precision is more reduced and vice versa [3,12]. For example in early epochs the recall learning curve in some cases it has a very small overfiiting, which means that precision has generally a slightly better performance for all the above examples, but all these analysis about these three metrics it is helpful in order to have a more complete opinion for the models that are training.

About early stopping generally is a very good way to prevent the overfitting. In all these experiments [3,9,10,11] if the validation loss for 2 epochs (not strictly consecutive) is getting increased, something not expected, then the training is stopped in the second epoch that the increase of the validation loss is observed. This technique is proven that in this exercise is helping a lot to have better learning curves for recall, precision and f1-score, but especially for the loss ,where after the first 8-12 epochs that the model has be trained on the data a lot and also in the last epochs is prevented a possible divergence and in a large extent a possible overfitting or underfitting.

After some experiments it is occurred that the best learning rate and the best value for dropout probability ,in order to achieve no (or at least to prevent it in a large extent) overfitting, all the metrics (accuracy, f1-score, recall, precision) to have very similar values and τo achieve the biggest convergence, are 10^(-4) and 0.01 correspondingly. The values in all metrics in train must be bigger, equal or to have generally small differences each other. That is something that all the above experiments for the feed forward neural network have achieved.

Generally the roc curves [13] and the accuracy, precision, f1-score and recall values [12] are good in each experiment. The ROC curve is a very good way to check the variation of true positive rate in relation to the false positive rate as a a comparison of these two operating characteristics. The ROC curve is generally constant in 0.90-0.91 values, which is expected as the loss function is the same in all the above experiments.

Also the glove mebeddings of 300d, which is a type of word embeddings (word vectors) [17] for vectorization give a good approach to the creation of the classifier for movies with positive or negative reviews.

6th step: Predicting the test set

After training the model and preprocessing of the test set as the steps 1,2 and 3,4 and 5 explains, the next step is to predict the test set. Validation set is set as none through the variable test_data, in order to evaluate each test (validation) set, only by passing the path of whichever test set, for example test_data = pd.read_csv(path_file) (or for example if we say test_data = into the variable test_data.

The evaluation of the test set if this is existent (if test_data is not None) begins with the creation of vocabulary for the training dataset, the preprocessing and the vectorization as described in the first two steps and afterwards, Dataloader is used to create the data iterator for the training set after the splitting of train our model in the 3rd step. Finally in the two final steps of this process the model of the neural network is created and finally giving values to many parameters such as learning rate, dropout probability or to experiment with different optimizers, we can have the evaluation for the test dataset.

References

1. torch.utils.data — PyTorch 1.13 documentation
2. ReLU — PyTorch 1.13 documentation
3. Artificial Neural Networks - Backpropagation (uoa.gr)
4. Optimizing Model Parameters — PyTorch Tutorials 1.13.0+cu117 documentation
5. Build the Neural Network — PyTorch Tutorials 1.13.0+cu117 documentation
6. Automatic Differentiation with torch.autograd — PyTorch Tutorials 1.13.0+cu117 documentation
7. Dropout — PyTorch 1.13 documentation
8. BCELoss — PyTorch 1.13 documentation
9. PyTorch Early Stopping + Examples - Python Guides

10. Using Learning Rate Scheduler and Early Stopping with PyTorch - DebuggerCafe
11. [PyTorch] Use Early Stopping To Stop Model Training At A Better Convergence Time - Clay-Technology World (clay-atlas.com)
12. API Reference — scikit-learn 1.2.0 documentation
13. Receiver operating characteristic – Wikipedia
14. Regression (uoa.gr)
15. Training Deep Neural Networks (uoa.gr)
16. Artificial Neural Networks - The Perceptron (uoa.gr)
17. Word Vectors (Word Embeddings) (uoa.gr)