Assignment 3 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 bidirectional recurrent neural network (RNN) which could be also an LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Units) 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, collections etc. Also from nltk through word_tokenize, the tokenization of a text review into words is made. From the stopwords list, words like couldn't, not etc are removed because it could be useful for

the context of each text review. For every change that is made to each text the review column is renewed 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. Total time for vectorization, creation of vocabulary and preprocessing = 1 minute and 26 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, the first thing is to make easy the choice of the cell that is going to use for the experiments (RNN, LSTM, GRU) through if statements from an dictionary that is created with all the choices of cell types, where we can get out the corresponding outputs for each cell type [2,3,4].

RNN (Recurrent neural network) consists an family of neural networks that take sequential input of any length , apply the same weights on each step and can optionally produce output on each step. LSTM (Long Short-Term Memory) is a way of solving the vanishing gradient problem and one of its preserves information from previous timesteps and GRU (Gated Recurrent Units) , which is an simpler alternative of LSTM, without cell states[5,6,7].

Also the dropout technique is used in the neural network, which is a very effective technique for regularization and preventing the co-adaptation of neurons [8]. This neural network ,for each type of cell that is chosen for experiments, is fed with parameters such as the input and the output size, the type of cells, number of stacked RNNs (number of hidden layers), a boolean value which gives the information if the technique of skip connections is applied, number of hidden units for each hidden layer which is the number of stacked RNNs and the dropout probability [8]. Also forward and backward cells are created as we have bidirectional neural networks for each stacked RNN through the ModuleList by pytorch [9], which is an easy way to have access into the previous elements of a list. In the backward cells for each of RNN/LSTM/GRU the reversed copy of the input is used, which occurs from torch.flip() [10].

Also skip connections is a technique that skip some layers and feeds the output of one layer as the input to the next layers. This is a technique that helps, especially in the ResNets to solve the degradation problem, i.e While training deep neural nets, the performance of the model drops down with the increase in depth of the architecture [11,12]. In this exercise skip connections could be implemented by matrix addition.

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 [13,14,15]. Especially there is a linear layer which is fed by the concatenated outputs of the RNN/LSTM/GRU network, as there is a bidirectional network. Finally the results of the linear network are passed through the sigmoid function in order to make the results probabilities ranging from 0 to 1[16].

5th step Train of our model ,explanations of the results and comparison between the best models of the assignments 1 2 and 3

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 calculated from Binary Cross Entropy Loss function [17]. The dataset is splitted through train_test_split method from sklearn, and with random_state = 250, the results are the same (or almost similar) every time this program is run [18]. 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 [13,14], 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 15 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 (2^{nd} 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 [15]:

- 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

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 [19,20,21]. 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 [22,23,24]. Also in this exercise gradient clipping is applied, which is a way to solve the problem of the exploding gradients, which refers to a large increase in the norm during training [6,7].

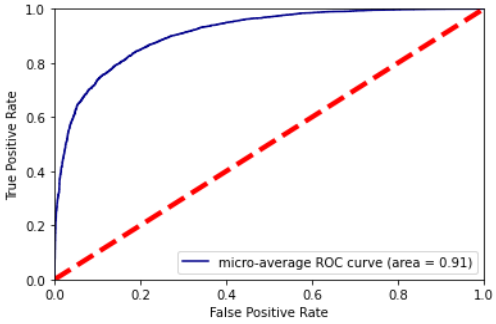
Below some experiments for the neural network that is created are presented with their learning curves about their loss and also their roc curves, which are plotted every 5 epochs[24]. There is presented the results of the last epoch, which is a very representative sample in order to explain better the results that occur:

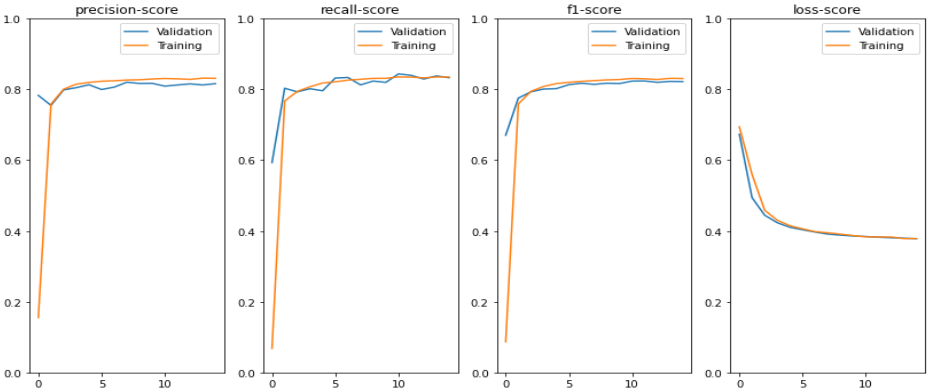
Without skip connections implementation

Cell Type = GRU

- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 5 with size 8, max_norm of gradient clipping by norm = 0.75, total time running (with calcuations, and ROC curves): 1 minute and 54 seconds, the ROC curve of the 11th epoch is represented.

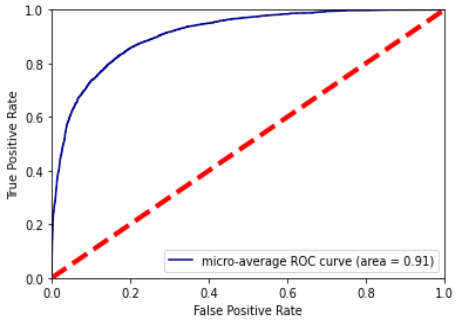
Performance metric	Last epoch value due to early stopping(15th)
Loss function (train)	0.37872
Precision (train)	0.83163
Recall (train)	0.83525
f1-score (train)	0.83113
Acurracy (train)	0.83238
Loss function (validation)	0.37840
Precision (validation)	0.81641
Recall (validation)	0.83327
f1-score (validation)	0.82233
Acurracy (validation)	0.82744

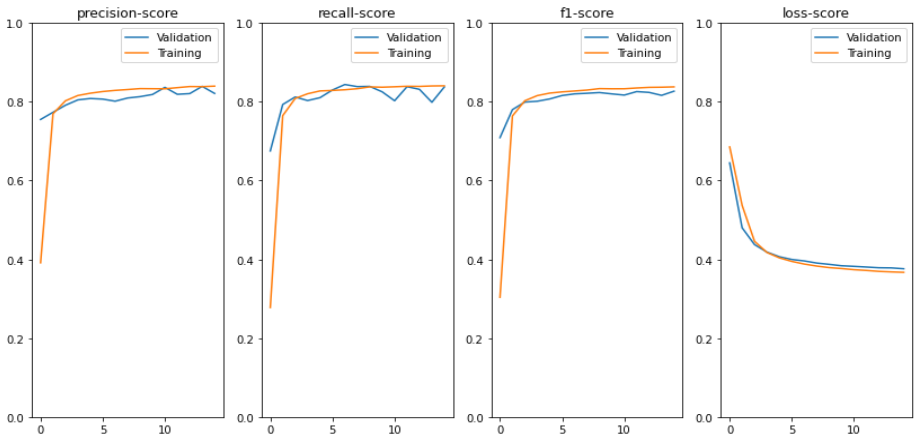




- Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of stacked RNNs = 5 with size 8, max_norm of gradient clipping by norm = 0.70, total time running (with calcuations, and ROC curves): 1 minutes and 54 seconds. The ROC curve of the 11th epoch is represented.

Performance metric	Last epoch value due to early stopping(15th)
Loss function (train)	0.36712
Precision (train)	0.83984
Recall (train)	0.84071
f1-score (train)	0.83798
Acurracy (train)	0.83889
Loss function (validation)	0.37624
Precision (validation)	0.82138
Recall (validation)	0.83768
f1-score (validation)	0.82702
Acurracy (validation)	0.83044

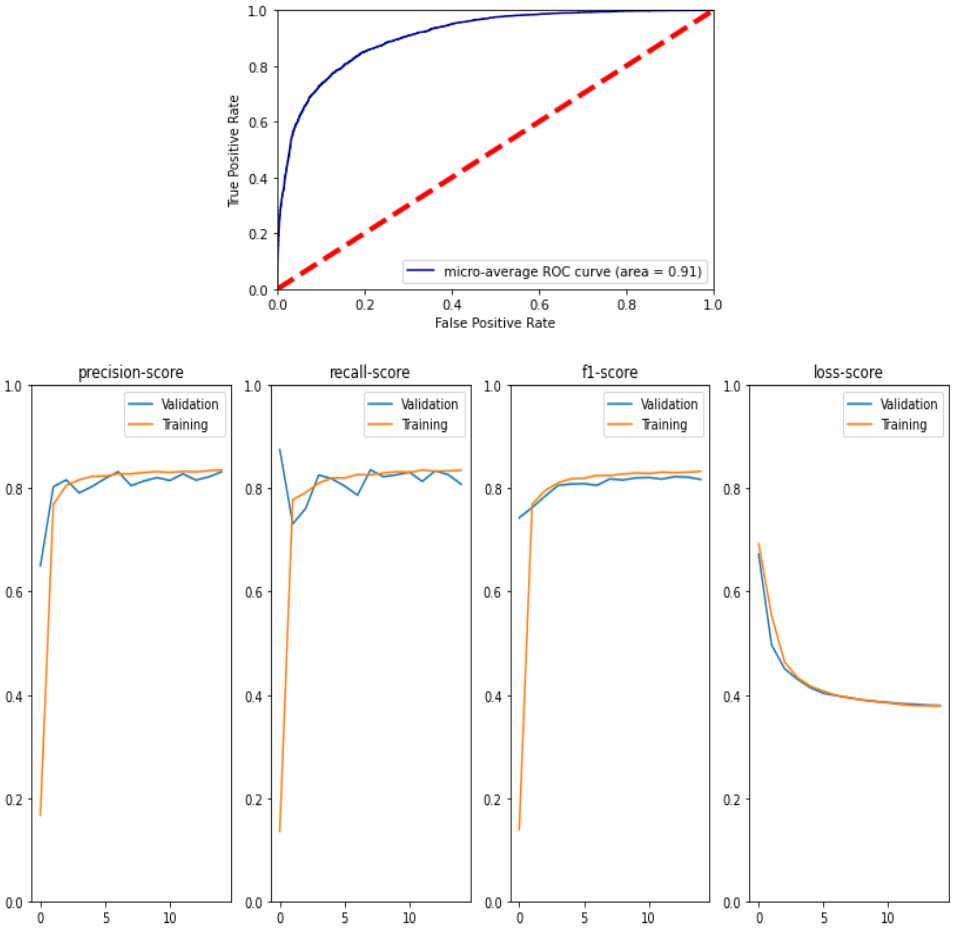




- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 8 with size 8, max_norm of gradient clipping by norm = 0.75, total time running (with calcuations, and ROC curves): 2 minutes and 37 seconds. The ROC curve of the 11th epoch is represented.

Performance metric	Last epoch value due to early stopping(14th)
Loss function (train)	0.37852
Precision (train)	0.83591
Recall (train)	0.83499
f1-score (train)	0.83316
Acurracy (train)	0.83407
Loss function (validation)	0.37914
Precision (validation)	0.83246

Recall (validation)	0.80816
f1-score (validation)	0.81747
Acurracy (validation)	0.82677

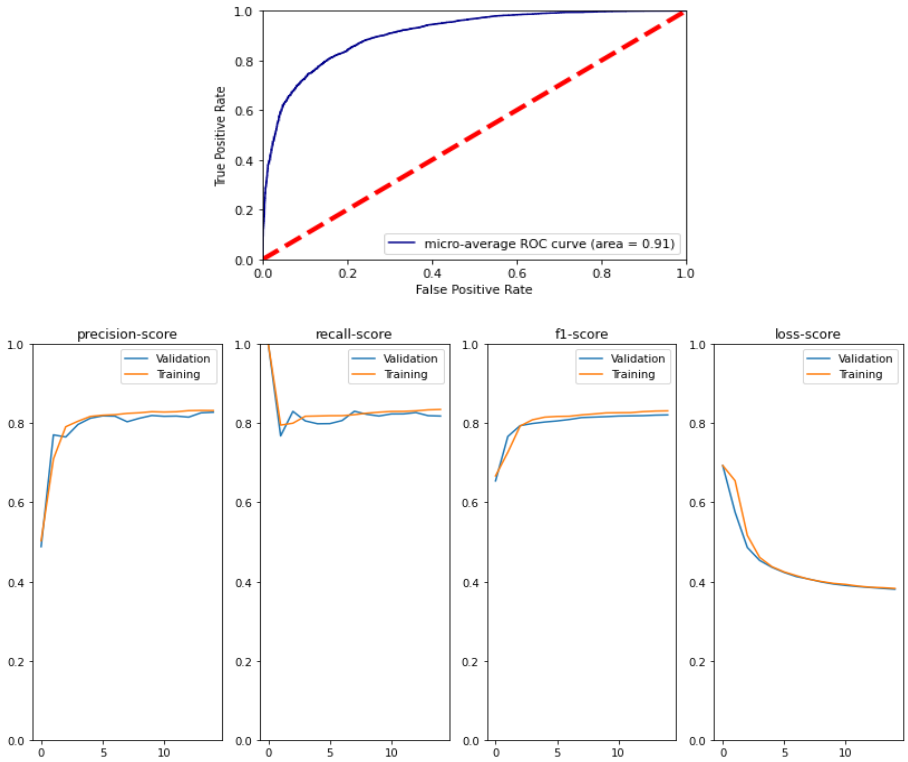


Cell Type = LSTM

- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 5 with size 8, max_norm of gradient clipping by norm = 0.75, total time running (with calcuations, and ROC curves): 1 minute and 59 seconds. The ROC curve of the 11th epoch is represented.

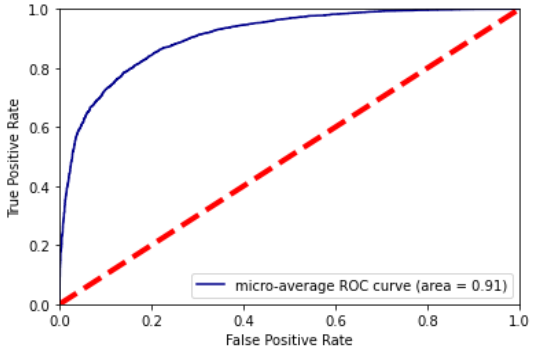
Performance metric	**Last epoch value due to early stopping(15th)**
Loss function (train)	0.38275
Precision (train)	0.83230
Recall (train)	0.83499
f1-score (train)	0.83147
Acurracy (train)	0.83228
Loss function (validation)	0.38060
Precision (validation)	0.82797
Recall (validation)	0.81835
f1-score (validation)	0.82122

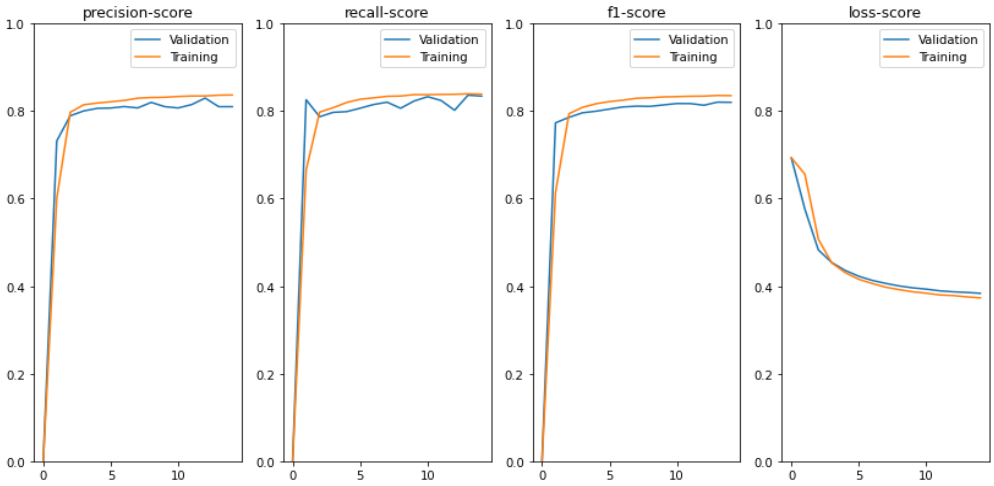
Acurracy (validation)	0.82783



- Optimizer = Adam, dropout probability = 0.01 , learning_rate = 1e-4, number of stacked RNNs = 5 with size 8, max_norm of gradient clipping by norm = 0.70 with no skip connections, total time running (with calcuations, and ROC curves): 1 minute and 58 seconds. The ROC curve of the 11th epoch is represented.

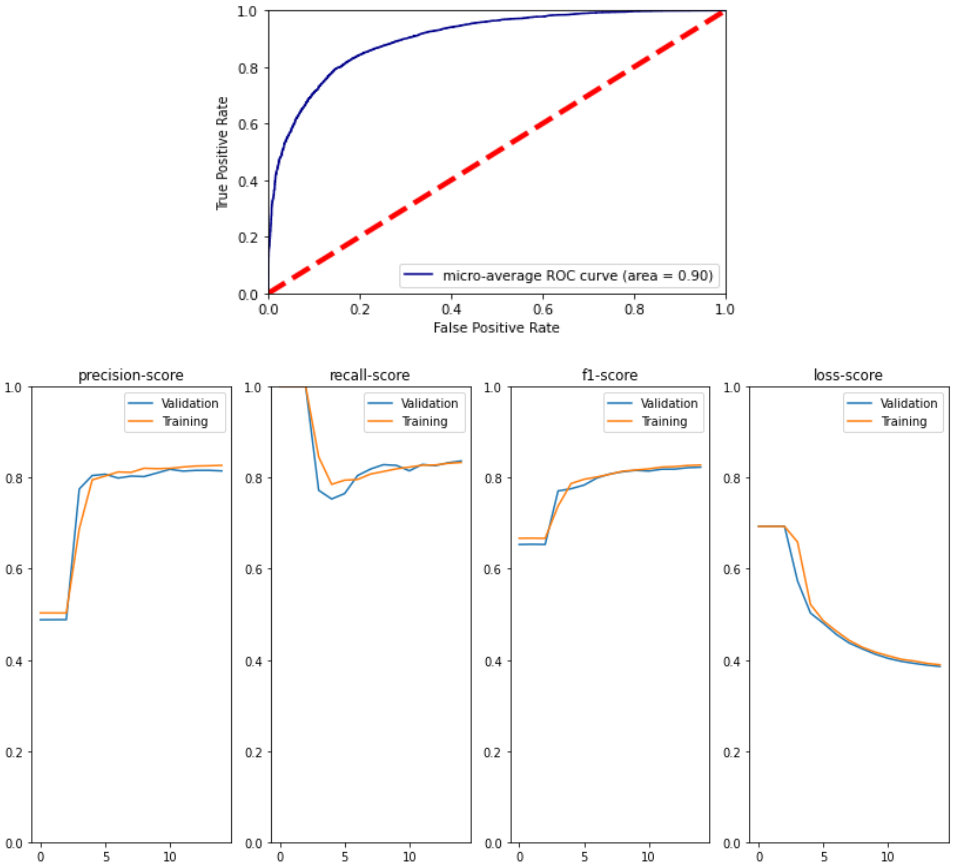
Performance metric	**Last epoch value due to early stopping(15th)**
Loss function (train)	0.37317
Precision (train)	0.83684
Recall (train)	0.83870
f1-score (train)	0.83545
Acurracy (train)	0.83623
Loss function (validation)	0.38365
Precision (validation)	0.81022
Recall (validation)	0.83429
f1-score (validation)	0.82017
Acurracy (validation)	0.82472





- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 8 with size 8, max_norm of gradient clipping by norm = 0.75 with no skip connections, total time running (with calcuations, and ROC curves): 2 minutes and 29 seconds. The ROC curve of the 11th epoch is represented.

Performance metric	**Last epoch value due to early stopping(15th)**
Loss function (train)	0.38927
Precision (train)	0.82717
Recall (train)	0.83350
f1-score (train)	0.82794
Acurracy (train)	0.82862
Loss function (validation)	0.38553
Precision (validation)	0.81504
Recall (validation)	0.83685
f1-score (validation)	0.82324
Acurracy (validation)	0.82733

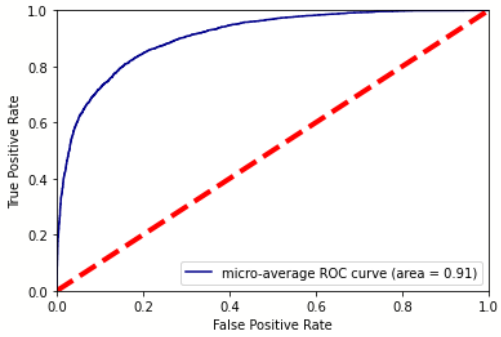


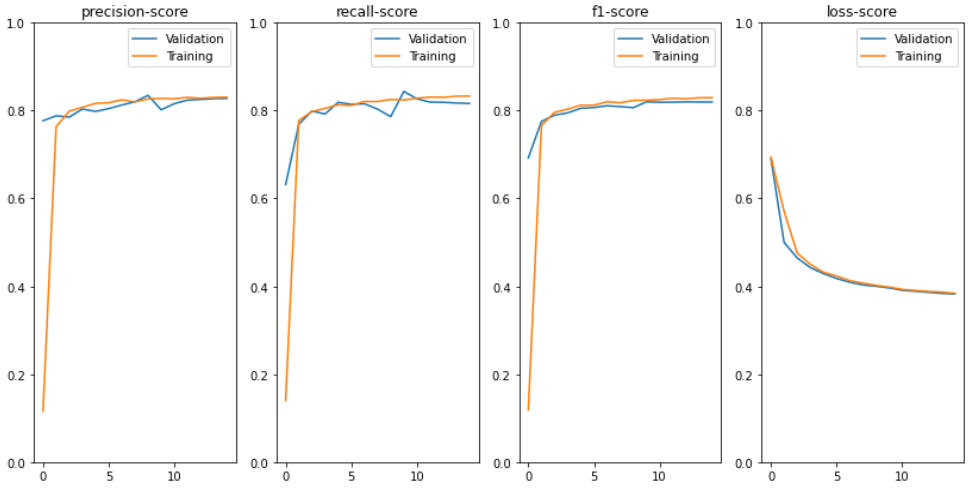
With skip connections implementation

Cell Type = GRU

- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 14 with size 8, max_norm of gradient clipping by norm = 0.75 with skip connections = false (boolean value in the code of the network), total time running (with calcuations, and ROC curves): 4 minutes and 1 second. The ROC curve of the 11th epoch is represented.

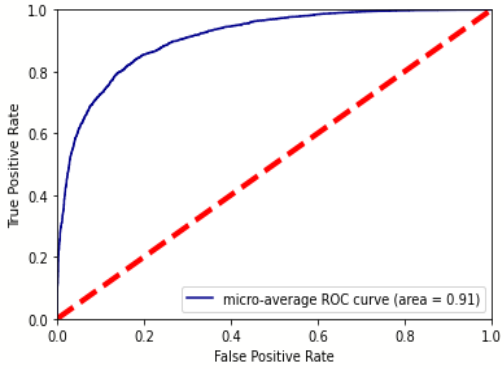
Performance metric	Last epoch value due to early stopping(15th)
Loss function (train)	0.38412
Precision (train)	0.83077
Recall (train)	0.83314
f1-score (train)	0.82960
Acurracy (train)	0.83019
Loss function (validation)	0.38245
Precision (validation)	0.82754
Recall (validation)	0.81613
f1-score (validation)	0.81940
Acurracy (validation)	0.82734

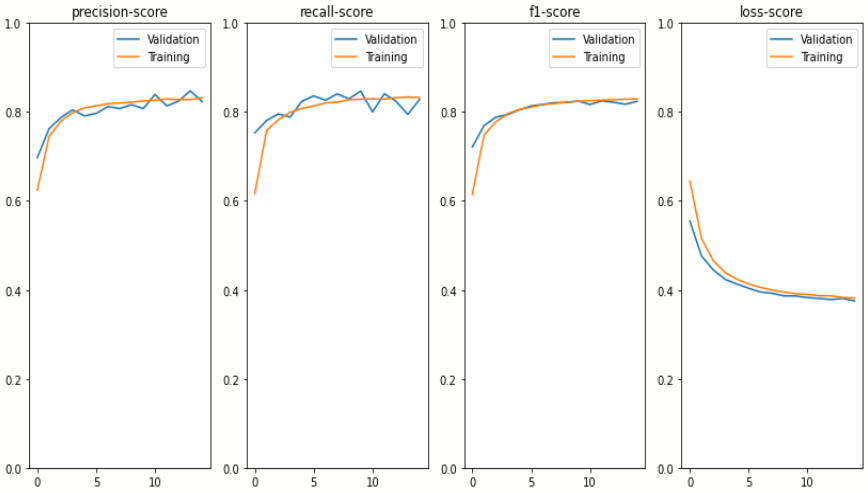




- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 14 with size 8, max_norm of gradient clipping by norm = 0.75 with skip connections = true (boolean value in the code of the network), total time running (with calcuations, and ROC curves): 4 minutes and 9 seconds. The ROC curve of the 11th epoch is represented.

Performance metric	**Last epoch value due to early stopping(15th)**
Loss function (train)	0.38148
Precision (train)	0.83174
Recall (train)	0.83234
f1-score (train)	0.82954
Acurracy (train)	0.83077
Loss function (validation)	0.37505
Precision (validation)	0.82339
Recall (validation)	0.82834
f1-score (validation)	0.82395
Acurracy (validation)	0.82934

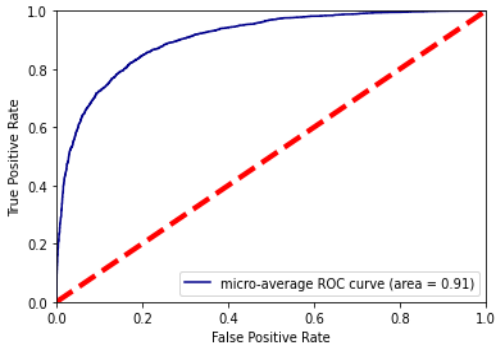


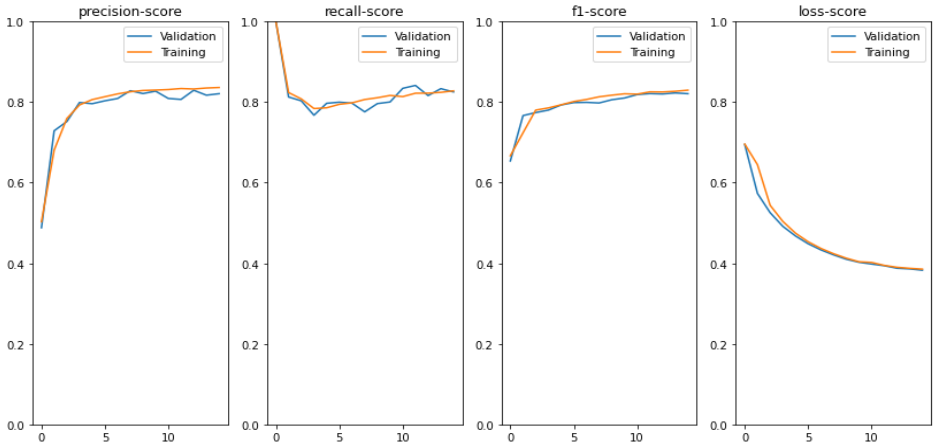


Cell Type = LSTM

- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 14 with size 8, max_norm of gradient clipping by norm = 0.75 with skip connections = false (boolean value in the code of the network), total time running (with calcuations, and ROC curves): 4 minutes and 10 seconds. The ROC curve of the 11th epoch is represented.

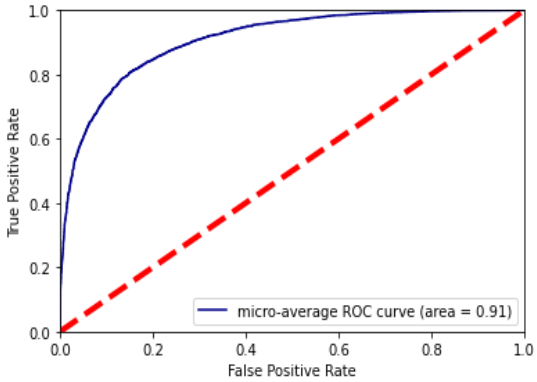
Performance metric	Last epoch value due to early stopping(15th)
Loss function (train)	0.38550
Precision (train)	0.83631
Recall (train)	0.82808
f1-score (train)	0.82979
Acurracy (train)	0.83134
Loss function (validation)	0.38268
Precision (validation)	0.82099
Recall (validation)	0.82535
f1-score (validation)	0.82101
Acurracy (validation)	0.82739

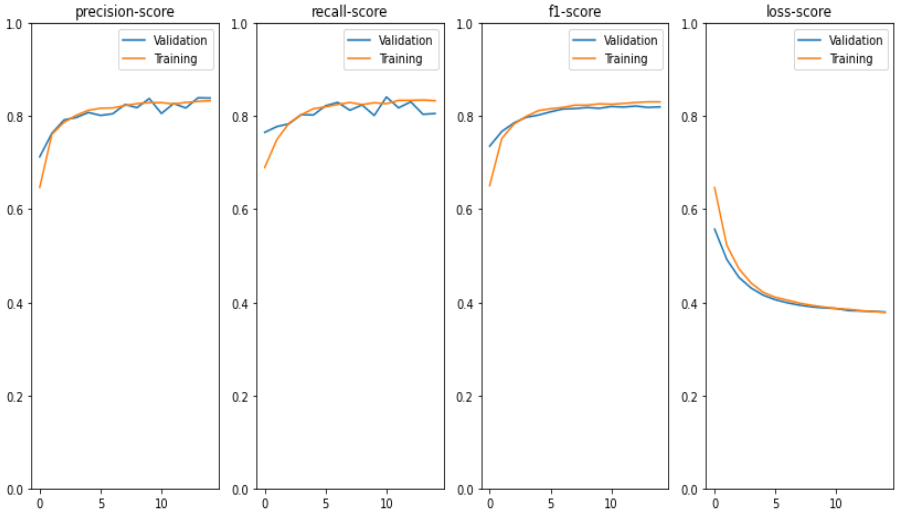




- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 5 with size 8, max_norm of gradient clipping by norm = 0.75 with skip connections = true (boolean value in the code of the network), total time running (with calcuations, and ROC curves): 4 minutes and 20 seconds. The ROC curve of the 11th epoch is represented.

Performance metric	**Last epoch value due to early stopping(15th)**
Loss function (train)	0.37845
Precision (train)	0.83337
Recall (train)	0.83344
f1-score (train)	0.83099
Acurracy (train)	0.83219
Loss function (validation)	0.37927
Precision (validation)	0.83924
Recall (validation)	0.80586
f1-score (validation)	0.82020
Acurracy (validation)	0.82983





After some experiments the values that are used for these experiments about the dropout probability and the gradient clipping max norm are the appropriate in each experiment in order to have a good result. About the ROC curves [23] the results are very good because for all of the above experiments the value is between 0.90 and 0.91, which means that true positive rate and false negative rate illustrates (almost) the same result. In fact the predicting power of the classifier is big because the curve is getting closer to the top left corner i.e point (0.0, 1.0) and also because it is farer from the random classifier (dotted line in each roc curve).

It is observed that if the number of stacked RNNs is increased, the time running of the model is getting bigger, due to the bigger number of multiplications that must be done in order to achieve similar results with the case of fewest hidden layers. Generally is observed a good behaviour in all the metrics.

Generally precision and recall show an acceptable behaviour in all cases. This observation, although the probable very small overfitting especially for recall or precision learning curve, ensures that between precision and recall can not have both big values, the increase of one means the decrease of the other[16]. Although the f1-

score ,which is the harmonic mean and has a good behaviour for this classifier, gives us the sense that precision and recall has similar values and behaviour, it is vital to see the progress of recall and precision in order to know the weaknesses and the strongest points of the model.

It is chosen the LSTM and GRU cells to use instead of GRU, because it's too difficult for the RNN to learn to preserve information over many timesteps. LSTM is a good default choice especially for the model to learn long-distance dependencies. GRU is an alternative of LSTM, that can store unboundedly large values in memory cell dimensions, but GPU is generally (not always) more faster and more efficient [6], something that is shown by comparing the corresponding results (metrics values, differences between train and validation losses) of GRU with LSTM results and it is occurred that GRU results are better, especially with skip connections. Without the implementation of skip connections the deeper LSTM/GRU networks are not shown a very good performance and can not generalize well comparing to the LSTM/GRU networks with lower number of layers, something that is proven from the learning curves of the loss, recall, f1-score and precision, where especially on recall curve there is some overfitting.

Skip connections in all these experiments helps to improve the performance of the deepest neural networks. That is proven from the reducing of the train, validation losses and also the reducing of their differences comparing with the case that skip connections is not applied. Basically in deepest networks it is observed generally an little at least increasing in the values of all the metrics (precision, f1-score, recall) and finally skip connections is needed only in deepest neural networks. Overfitting is being prevented in an important extent or almost completely [10,11]. In this a tuning of gradient clipping norm and dropout probability helps importantly.

For the implementation of the skip connections the main idea is to skip the layers of LSTM/GRU networks with even number. So after the Dropout layer that we put the Input in order to regularize a little, this input entries the first layer of the LSTM/GRU network, an output occurs. The next layer(second) because is going to be skipped it has no input and as an output the previous output from the first layer. At the third layer the input is the first layer's output and the output of the third layer is summed with the first layer's output and this is the input of the next (fourth) layer etc [10,11].

This procedure is followed for all the rest layers of LSTM/GRU network , where in each layer with odd number in its output the output of the first layer is summed[10,11].

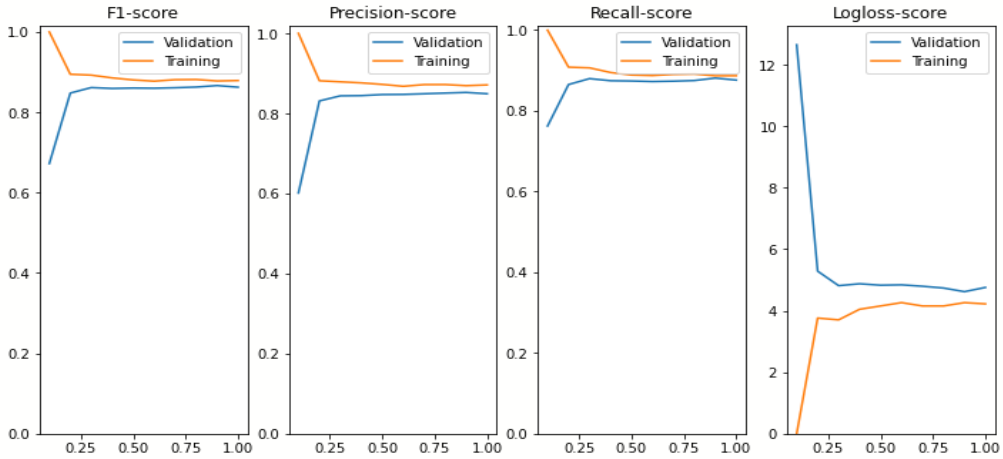
It is important to keep in mind that each output of each layer of the LSTM/GRU network is the input of the next layer of the LSTM/GRU network.

Early stopping in this exercise ,as it referred before, helps a lot the loss to have a normal behaviour, i.e not increased as the number of epochs increased. Especially if in two not consecutive essentially epochs the loss get increased the model training and calcuations stopped. That helps the loss learning curve to achieve a very good convergence between train loss and validation loss values.

The best models from assignments 1 and 2 are represented below:

- max_iter = 500, penalty = 'l2', class_weight = 'balanced', tol = 1e-4, C = 0.8 , solver = 'saga', class_weight = 'balanced', time running: 54 seconds , time plotting of learning curves: 3 minutes and 55 seconds

Performance metric	**1st fold value**
f1-score(train)	0.8784830997526794
f1-score(validation)	0.8664101154480484
Precision (train)	0.8686956521739131
Precision (validation)	0.8608258684727987
Recall (train)	0.8884936075597554
Recall (validation)	0.8720672864099159
Loss function (train)	4.241859719093663
Loss function (validation)	4.661755461384083

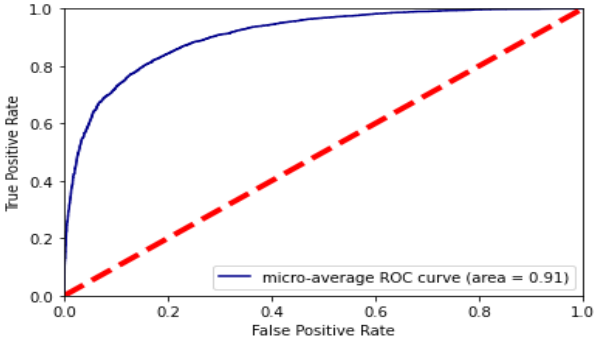


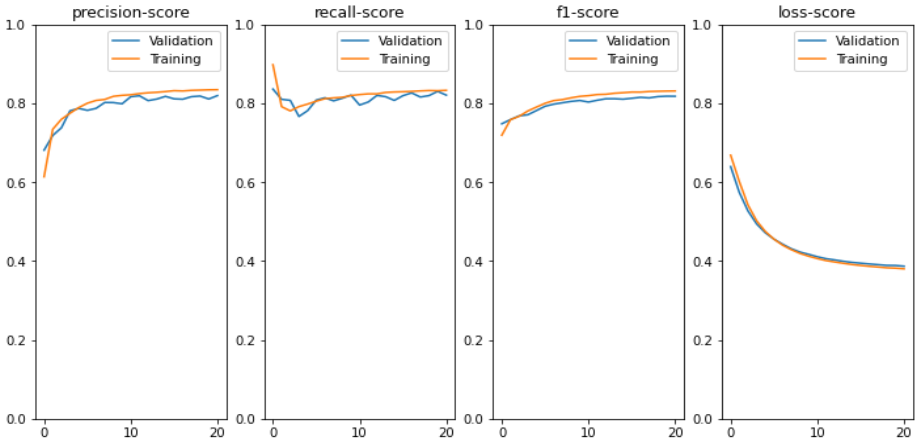
The results of the first fold are represented, because are very similar with the results of all the other folds.

- 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 (Feed Forward Neural Network)

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



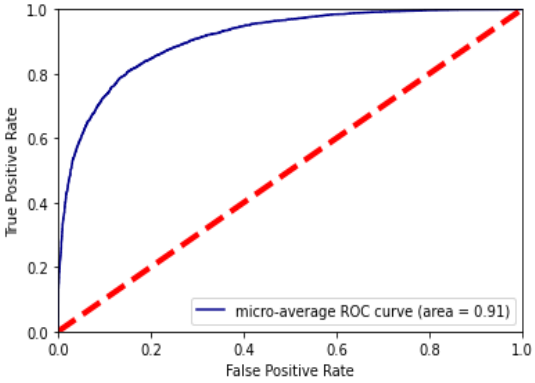


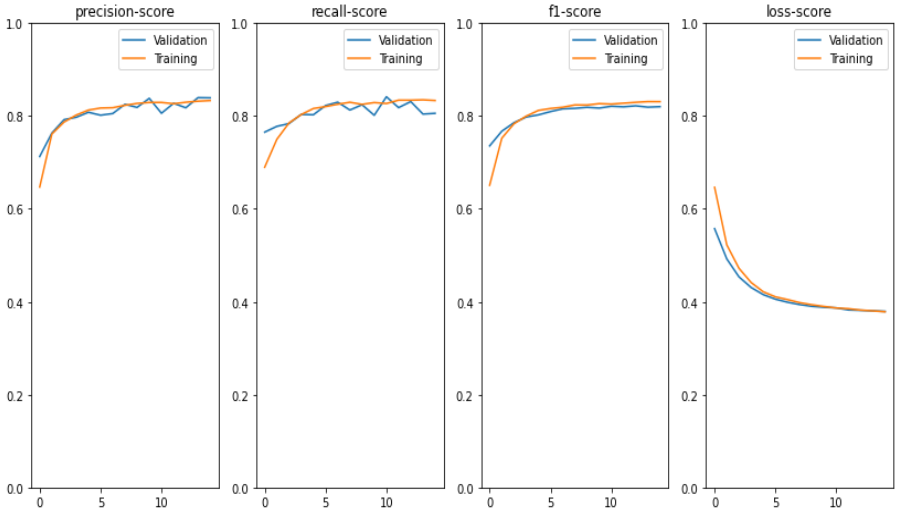
The best model for this exercise is:

- Optimizer = Adam, dropout probability = 0.05 , learning_rate = 1e-4, number of stacked RNNs = 5 with size 8, max_norm of gradient clipping by norm = 0.75 with skip connections = true (boolean value in the code of the network), total time running (with calcuations, and ROC curves): 4 minutes and 20 seconds. The ROC curve of the 11th epoch is represented.

Performance metric	**Last epoch value due to early stopping(15th)**
Loss function (train)	0.37845
Precision (train)	0.83337
Recall (train)	0.83344
f1-score (train)	0.83099
Acurracy (train)	0.83219
Loss function (validation)	0.37927

Precision (validation)	0.83924
Recall (validation)	0.80586
f1-score (validation)	0.82020
Acurracy (validation)	0.82983





Finally for the assignment 3 LSTM is proven that for deeper networks,if skip connections technique is implemented, is a good default option.

Comparing the three models generally the best from these three is the first model where the vectorization was implemented through countVectorizer and TfidfTransformer from sklearn[25,26] and the model that used is the Logistic Regression model from sklearn with its parameters[27]. This issues because of the very good number for all the metrics (recall, f1-score, precision) which is between 85-88% The corresponding scores for the metrics of the other two models including accuracy reach by average 82-84%. Also comparing with the other two models there is no risk of overfitting (or at least very small risk), but in the other two models the risk of overfiiting is a little bigger, although the convergence in all the learning curves of the metrics is similar and very good in all the models.

About the loss in these three model the cross-entropy loss function is calcuated either from sklearn either from pytorch for binary classification[17,28]. The best model about the loss is the third, because it has lower from 1 values and the convergence of train loss and validation loss values is almost perfect, with very small difference between these values, the second model with the feed forward neural network, although it has a little bigger values on train and validation loss with the third model, the difference between losses is a little bigger and finally the model of Logistic Regression has bigger values on train and validation losses and although validation losses are bigger than train losses, due to the model and the vectorization of the 1st assignment perhaps, the convergence is not so good as the other two models.

The model with RNN/LSTM/GRU cells compared to the model with a simple feed forward network, could be assumed as preferred because LSTM and GRU networks could at least provide a way to solve the problem of vanishing gradients , when we have small norms of gradients due to model's complexity [5,6,7], problem that is not occur in the feed forward networks. This makes model of 3rd assignment more completed than the model of the second assignment, although the results referring to numbers and learning curves are very similar. Essentially the model with the LSTM/GRU network to feed a simple neural network of the 3rd assignment is a more evolved model of the feed forward network of the 2nd assignment.

6th step: Predicting the test set and saving the model

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. Test set is set as none through the variable test_data, in order to evaluate each test 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 number of stacked RNNs, dropout probability, or the gradient clipping value, also if the technique of skip connections is

applied and which type of cell (RNN, LSTM, GRU) to use. From all these we can have the evaluation for the test dataset.

About the saving of our model it is implemented through saving by interference, where it is only necessary to save the trained model's learned parameters. Saving the model's state_dict with the torch.save() function will give you the most flexibility for restoring the model later, which is why it is the recommended method for saving models [29].

README

In the zip file about assignment 3 for the post graduate lesson Artificial Intelligence II, there are two .ipynb files. The first with the name "Assignment 3_Artificial_Intelligence_2(without skip connections implementation)", there is the implementation of the classifier that is the main goal of the assignment without the implementation of skip connections technique.

More specifically from all the experiments that are shown, from this .ipynb file there are running the cases that we do not have a very deep LSTM/GRU network, i.e networks with number of layers greater than 10 (experiments of neural networks with 5 or 8 stacked LSTM/GRU layers). The reason for that is that skip connections technique is needed for cases of deeper networks, i.e networks with number of layers greater than 10.

In the second .ipynb file "Artificial_Intelligence_II_Assignment3(with skip connections implementation)" the skip connections technique is implemented and it is running only for netorks with greater than 10 LSTM/GRU stacked layers, in our case 12 stacked layers, either if skip connections technique is implemented either not, i.e either if the value of relative boolean value skip_connections of this code is false or true. I would strongly recommend you to run the cases of 5 and 8 LSTM/GRU stacked layers from Assignment 3_Artificial_Intelligence_2(without skip connections implementation).ipynb file and the case of 12 LSTM/GRU stacked layers from Artificial_Intelligence_II_Assignment3(with skip connections implementation).ipynb file Essentialy these two .ipynb files have the same lines of code with the only difference the code of the neural network which has some differences between these two files.

Finally about the saving of the models, for both of .ipynb files the same way is using (Saving and Loading Models — PyTorch Tutorials 1.12.1+cu102 documentation). The

.pt file of the model that is saving from Artificial_Intelligence_II_Assignment3(with skip connections implementation).ipynb file is RNNSkip.pt, and the name of the corresponding .pt file of the model that is saving from Assignment 3_Artificial_Intelligence_2(without skip connections implementation).ipynb file RNN.pt. If this way of both or one of the files does not work, you have to do the training on your own. In each of these files the time of the training of the LSTM/GRU network generally is not greater than 5-6 minutes.

References

1. torch.utils.data — PyTorch 1.13 documentation
2. RNN — PyTorch 1.13 documentation
3. LSTM — PyTorch 1.13 documentation
4. GRU — PyTorch 1.13 documentation
5. Language Modeling and Recurrent Neural Networks (RNNs) (uoa.gr)Artificial Neural Networks - The Perceptron (uoa.gr)
6. Vanishing Gradients and Fancy RNNs (uoa.gr)
7. Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem) - neptune.ai
8. Dropout — PyTorch 1.13 documentation
9. torch.flip — PyTorch 1.13 documentation
10. ModuleList — PyTorch 1.13 documentation
11. Skip Connections | All You Need to Know About Skip Connections (analyticsvidhya.com)
12. ResNet | Understanding ResNet and Analyzing various Models (analyticsvidhya.com)torch.nn.utils.clip_grad_norm_ — PyTorch 1.13 documentation
13. Optimizing Model Parameters — PyTorch Tutorials 1.13.0+cu117 documentation
14. Build the Neural Network — PyTorch Tutorials 1.13.0+cu117 documentation
15. Automatic Differentiation with torch.autograd — PyTorch Tutorials 1.13.0+cu117 documentation
16. Artificial Neural Networks - Backpropagation (uoa.gr)

17. BCELoss — PyTorch 1.13 documentation

18. sklearn.model_selection.train_test_split — scikit-learn 1.2.0 documentation
19. PyTorch Early Stopping + Examples - Python Guides
20. Using Learning Rate Scheduler and Early Stopping with PyTorch - DebuggerCafe
21. [PyTorch] Use Early Stopping To Stop Model Training At A Better Convergence Time - Clay-Technology World (clay-atlas.com)
22. Regression (uoa.gr)
23. API Reference — scikit-learn 1.2.0 documentation
24. Receiver operating characteristic – Wikipedia
25. sklearn.feature_extraction.text.CountVectorizer — scikit-learn 1.1.3 documentation
26. sklearn.feature_extraction.text.TfidfTransformer — scikit-learn 1.1.3 documentation
27. 1.1. Linear Models — scikit-learn 1.1.3 documentation
28. 3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.2.0 documentation
29. Saving and Loading Models — PyTorch Tutorials 1.12.1+cu102 documentation