**Neural Networks and Deep Learning**

**Laboratory 1**

**Recurrent Neural Networks; Tunning and Regularizing**

The dataset contains information about the number of passengers on international flights starting with January 1949 until December 1960.

As you probably have guessed you will be trying to predict the number of passengers on international flights with a one-step forecast.

1. For comparable results set the random seed in both numpy and tensorflow to (123). You can use this code:

*np.random.seed(123)*

*tf.random.set\_seed(123)*

1. If you don’t work in google colab, import the appropriate metric that will be used for optimization (remember you are predicting a continuous, not categorical variable).
2. To download the csv file from github use !wget -nc https://lazyprogrammer.me/course\_files/airline\_passengers.csv
3. Create your data frame. To ensure that all works fine index the ‘Month’ column and parse the dates.
4. Plot the data and check for stationarity. You can use the Dickey Fuller test to do this. Or if you would rather rely just on visual inspection, argument your conclusion on whether data are stationary or not. Remember, for stationarity you will want a constant variance (i.e. variance is not increasing/decreasing with time) and no trend in your data. If data proved to be non-stationary correct for this problem. (to correct for variance issues you can log transform the data, for trend, you can take the first difference).
5. Split the data into training and test sets. Take the data from the very last year (1960) for the test set (i.e. the last 12 entries).
6. Create the index arrays for both train and test datasets. If you don’t remember how to do this from our ML class here is one example: *train\_idx = df.index <= train.index[-1],* so just do the same for the test set.
7. Create the appropriate data structure for a time series analysis where the past 10 datapoints will be used to make predictions of the next 1 datapoint.
8. Reshape your created dataframe into the 3D format that RNN expects.
9. Create the X\_train, y\_train, X\_test and y\_test.
10. Create a one-layer RNN with 25 units (neurons). Train it for 100 epochs with the default mini batch size of 32.
11. Compile and fit the model. When compiling, specify the ‘sgd’ optimizer, and choose the ‘accuracy’ metric to judge the goodness of your predictions. To compare further on the prediction results with the true values in the test set, specify inside the fit function your validation data: *validation\_data = (X\_test, y\_test).* Save the history of the loss and the accuracy results in each epoch as the model is being trained, so that you can later on plot them and compare.
12. Plot the loss and accuracy in each epoch for the training set as well as the validation set. Comment on your results. Do you see signs of overfitting? If yes, explain. If no, explain.
13. Do you see signs of vanishing or exploding gradient? Explain how the two would manifest in your graph.
14. Compute the model predictions, undo the differencing and plot the results.
15. Now try other options and see if you can beat your initial predictions:
16. Train the model for 500 epochs. Are your predictions better? Should you have stopped before the end of the 500 epochs judging by your loss and accuracy plots? If yes at what epoch would you stop and why?
17. Try two, three, and four layers for your RNN with 50 and 100 neurons in each of the hidden layers (you should get altogether 6 models – 2 layers & 50 neurons, 2 layers & 100 neurons, 3 layers & 50 neurons, etc.). Train each of them for 100 epochs and save and plot the accuracies and the losses for each of the 6 combinations. Do you see signs of overfitting? Which model overfits most? Explain.
18. For the most severely overfitting model from the 6 options above use the Dropuot in combination with the L2 method, to regularize your overfitting model. Be creative, try different options to correct for the overfit.
19. Could you create a model that beats the first one (the one with 1 layer and 25 units) and is not overfitting after regularization was applied?