Machine Learning for Signal Processing

*[5LSL0]*

Rik Vullings
Ruud van Sloun
Nishith Chennakeshava
Hans van Gorp

# Assignment: Familiarisation with Keras and TensorFlow

April 2021

# Familiarisation with Keras and TensorFlow

---

In this assignment, you will implement a basic fully connected neural network. This is for the purpose of performing a regression on given data. This data is produced using a non-linear function. The aim would then be to interpolate the intermediary data.

You should be able to complete this assignment using what is given here, the tutorials, and tensorflow's web-pages. Please use Tensorflow keras to complete this assignment.

---

## Neural Networks

There are many forms of Neural Networks (NN), and are used for varying applications. In this assignment, a Feed Forward Neural Network (FFNN) is used, for the purpose of regression. These networks are comprised of "neurons", which can be described as,

$$y = f\left(\sum_{i=0}^{n} w_i x_i + b\right) \tag{.1}$$

where $w_i$ is a weight multiplied to the input $x_i$, and $b$ is a bias added to the product. Finally, an activation function $f$ is applied, giving the output, $y$.

Neural networks of this fashion can be used as universal function approximators. As such, you will now be asked to do just that. Familiarise yourself with Keras and Python, all while testing the abilities of neural networks to operate as function approximators. In effect, adding to your knowledge and experience from the Keras tutorial document.

It might be good to explain at this point that you do not necessarily have all of the information or knowledge to fully understand the technical aspects of what you are expected to do in this assignment. But the goal is for you to acquire this knowledge in the process of completing this assignment. All components of this assignment will also be covered in future classes of this course, in more detail.

As a first step you will need to install TensorFlow systems so that you can exploit this powerful library for designing, implementing, and training your neural network. To install TensorFlow, go to `https://www.tensorflow.org/install/pip` and follow the instructions.

It is recommended to use TensorFlow-Keras (built into TensorFlow), over Keras (the independent library), so that everything is consistent within one library. However, the following will still work fine with Keras, so feel free to use either. Both offer similar functionality with similar commands. But you should be aware that loading in both libraries at the same time can sometimes throw errors.

Note: None of the questions below require an explanation of more than 2/3 paragraphs, in addition to any graphs that may be required. You will be graded out of 10 points.

## Exercise 1

(a) Import the training data given as 'train.txt', and the test data 'test.txt'. After importing it, you can now plot the first row vs the second row (x vs y), to visualise what the data actually looks like. You will now have to assign train (xtrain, ytrain) and test (xtest, ytest) arrays. Ensure that you also change the format to float32. Plot two graphs, 1) (xtrain, ytrain), and 2) (xtest, ytest), side by side, using the `plt.subplot()` command. **(0.5 points)**

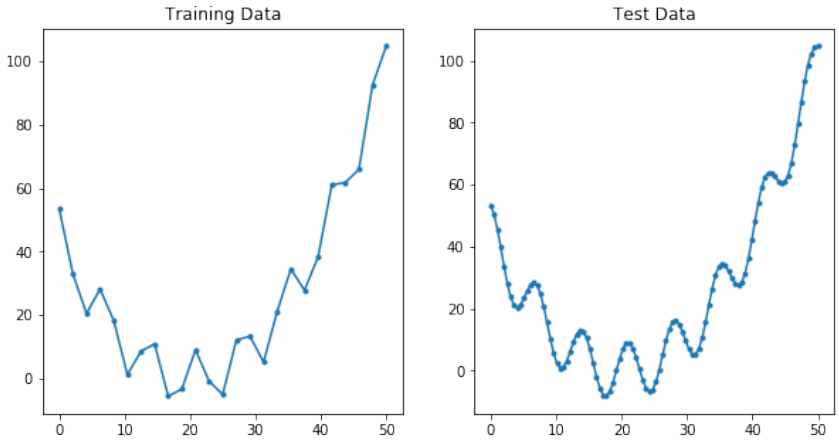


Figure .1: An illustration of the train and test data.

(b) We can now start building the neural network!

Build a fully connected network to approximate the function that was used to produce the train data. Here, you are simply building something that can map the inputs (xtrain) to the targets (ytrain), effectively performing a regression over the training data. Start with "off the shelf" models available on TensorFlow's website, for regression. Set all activation functions to None.

(Hint: A multi layered network with multiple neurons per layer might work better. While a single neuron network is too small, a network with more than 10 layers might be too much.)

A network that is highly complex can introduce its own set of problems. What problems, other than computational and memory load, can occur due to a too complex network architecture? **(0.5 points)**

(c) For the network to start training, we also need an optimiser.

What is the goal of an optimiser within a Neural Network? **(0.2 points)**

For the implementation, start with using the SGD (Stochastic Gradient Descent) given by TensorFlow. This has similarities to the Steepest Gradient Descent algorithm.

What are the similarities and differences between the two optimisers? **(0.3 points)**

Start with a small learning rate to begin with. Once the model has been set up, make sure you compile it.

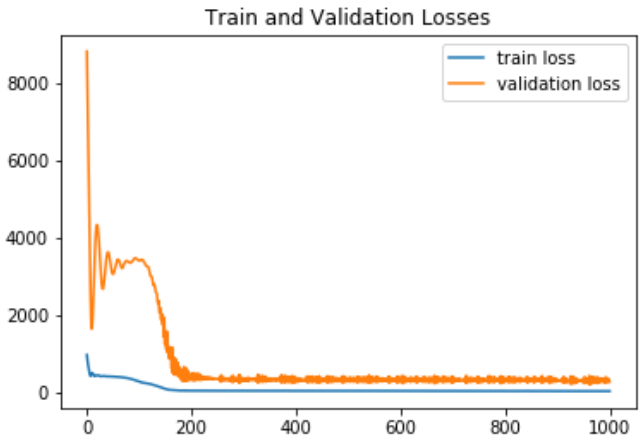


Figure .2: An example of what the training and validation losses look like after 1000 epochs.

(d) During training, make sure that you save the weights from your model to a folder, you may look up the process here (`https://www.tensorflow.org/tutorials/keras/save_and_load`). To ensure that your model can actually run, make a prediction with the untrained network using the test set.

Plot the prediction alongside the test data (using subplot). Comment on the prediction of the untrained model. Why does it look the way it does? **(0.5 points)**

(e) Now, all that's left to do is to actually train your model! Use the `history = model.fit()` function to do this. At this point, you're ready to plot the loss from the training process. Both training losses and the validation losses. You can do this by extracting train and validation losses from the 'history' object.

Plot the training loss and the validation loss in one graph, with the proper labels (and a legend).

How could you utilise this information to assess the quality of your model? **(1.0 point)**

(f) Reload the latest weights that you've saved during training, and make a prediction on the test set. Compare this to the prediction made on the untrained model.

Plot two graphs side-by-side, 1) Overlay the prediction from the untrained model and the trained model, 2) The test set data. As usual, make sure to have the proper labels and legends.

Comment on your results. Why did the this network fail to approximate even the rough shape of the function? **(1.0 point)**

## Exercise 2

Now, we would like you to build upon what you did in the previous exercise. One of the reasons your network might not have learnt to approximate the given function might be due to the architecture of the network, and it requires some degree of non-linearity. We are going to introduce this non-linearity to the network in the form of an activation function.

(a) Build an activation function (ReLU), and include it with the model. The definition for a ReLU is given as,

$$f(x) = max(0, x) \tag{.2}$$

(Hint: Build this using tensorflow's internal functions, rather than numpy)

You may use the same network as before, but now use your own activation function as the activation function (Do not use the ReLU activation function given by TensorFlow or Keras)!

Plot the output from this activation function. Does it match what you expect? Where in the network does the activation function have to be implemented? Include your custom built activation in the network. Also include the code that you have written. **(1.0 point)**

(b) A second major factor can be the optimiser that you chose to perform this regression. While SGD does what is advertised, there are more sophisticated optimisers out there.

For the next round of training, pick a more sophisticated optimiser, and justify your choice. What are the problem that can occur with SGD and how are these resolved conceptually by more sophisticated optimizers? **(1.0 point)**

(c) As with exercise 1d, make a prediction using the untrained network using the test data.

Comment on the prediction of the untrained network. **(0.5 points)**

(d) Train this network, and ensure to save the weights from the training, so that you can load it in later to make prediction.

Plot train and val losses using 'history', like you did in exercise 1e.

At this point, it is good to reevaluate if the network architecture you have chosen is appropriate for the problem you are trying to solve. **(1.0 point)**

(e) Load the latest weights from the training process, and make a prediction.

Plot two graphs side-by-side, 1) Overlay the prediction (using the test data) from the untrained model and the trained model, 2) The test set data. As usual, make sure to have the proper labels and legends.

Compare the prediction made by the first trained model (from Exercise one), to the prediction made by the second trained model (from Exercise two). Also comment on the differences between the two models. **(1.5 points)**

## Exercise 3

While training complex networks, which can take tens of hours to train, we would ideally not want to wait until it is done training to evaluate the quality of the network. Therefore, we would want to visualise the losses live, while the training process is ongoing.

(a) We can build this visualisation in many ways. But a quick way would be to build a callback function that plots the training and validation losses and saves it to a folder (as .png), after every 100 epochs.

Show a representative set of examples of the png files that your network has saved throughout the training (must include the png from the last epoch). Comment on the training and validation losses between these png files. How could you assess the quality of the network using callbacks such as this, during training? Why would it be useful to do this? **(1.0 point)**