Department of Electrical Engineering
Machine learning for signal processing
Q3, 2020-2021

5LSL0 - Machine learning for signal processing

CBL Assignment 1

C. Schmitt 1619020
S. Maulod 1457284

Coordinators:
Dr. ir. R. Vullings
Dr. ir. R.J.G. van Slouns

Eindhoven, April 26, 2021

Exercise 1

(a)

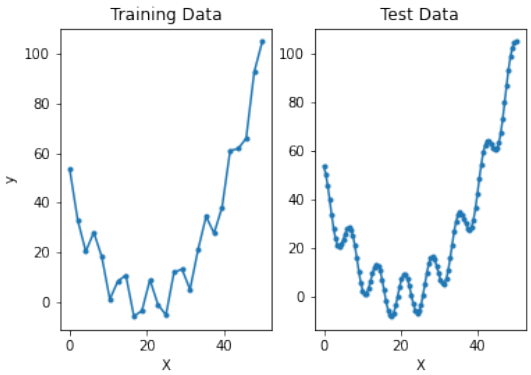


Figure 1: Plot of Training and Test Data

(b) For our network, we chose a network with three layers, by using the `Dense` layer from Keras and set the number of neurons in each layer as 2, 5 and 3 corresponding to the first, second and third layer respectively.

Too complex a neural network architecture can lead to overfitting where the resulting model is well trained on the training data but performs poorly in predicting from a validation set. Also, a more elaborate architecture may need more training data in order to perform comparably to a less complex architecture should one exist.

(c) The aim of the optimizer is to tune the parameters/kernels used in the network with the goal of minimizing the losses during training.
Similarities
Both approaches use the same approach to calculate the gradient with input training data.
Differences

1. Stochastic gradient descent (SGD) samples a random subsample of data for the gradient calculation while steepest descent samples entire dataset to calculate gradient which is computationally intensive for large data sets.
2. SGD takes more iterations than plain gradient descent, but the overall computational cost still tends to be lower.

(d) Refer to Figure 2. The untrained model will use the default weights from the `Dense` layer and without any activation function would thus be a direct linear model with parameters not dependant on the training data. The prediction is expectedly bad for this "pure guess" scenario.

(e) Refer to Figure 3a. The validation loss shows how good the model performs on new data, it is therefore the best indication of the overall performance. The gap between training- and validation loss can give an indication of whether overfitting (if training loss is low but validation loss is high) or underfitting (if both training- and validation loss are high) occurs.
Notice that for this assignment, there is significantly less training data than test data (opposite to most typical machine learning applications) so the gap is not necessarily due to overfitting, but likely also due to the lack of training data.

(f) See Figure 3b. The lack of nonlinearities means that the concatenation of linear operations reduces to a single linear operation. This leads to a model that's linear (with an offset), which naturally can't capture the complexity of the desired function.

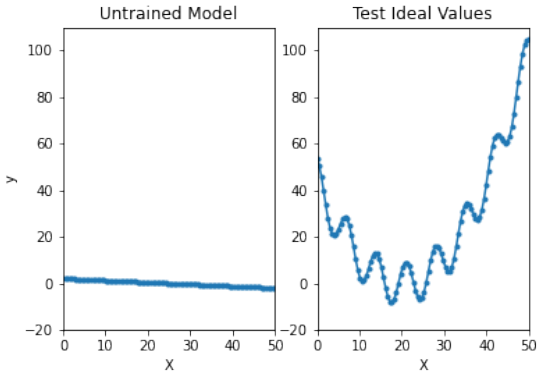
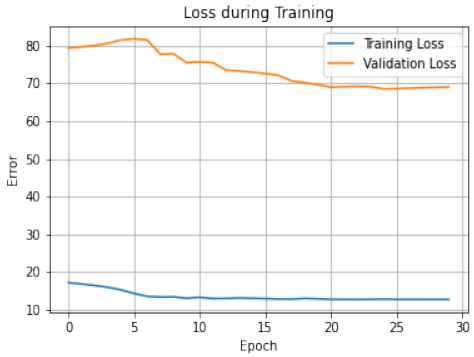
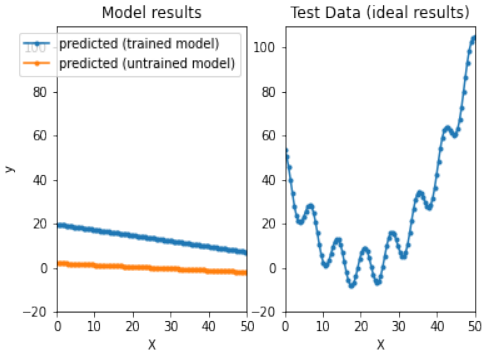


Figure 2: Plot of untrained model and test values



(a) Training and validation losses during training



(b) Plot of trained and untrained model compared to actual test data.

Figure 3: Response to Question 1.e and 1.f.

Exercise 2

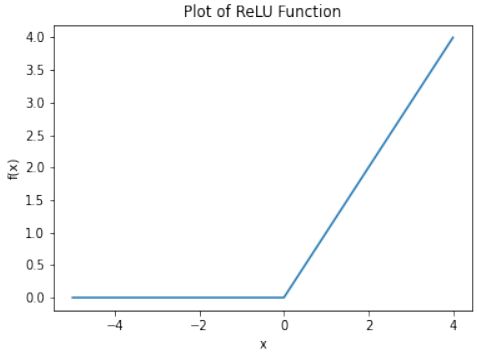
(a)

```
def relu_activation(x):
    return tf.math.maximum(0.0,x)

xynonlin_model = tf.keras.Sequential([
    xnormalizer,
    layers.Dense(units=5,  activation=relu_activation),
    layers.Dense(units=20, activation=relu_activation),
    layers.Dense(units=5,  activation=relu_activation),
    layers.Dense(units=1)])

```

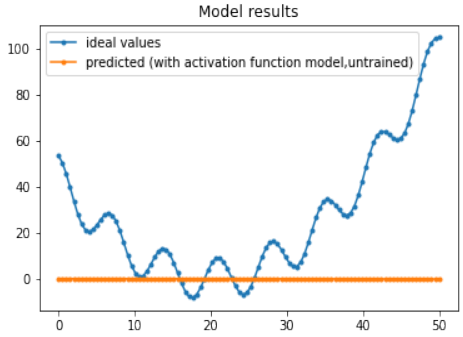
Refer to Figure 4a for ReLU function visualisation. This matches what we expected : 0 for $x \leq 0$, $\max(x)$ for $x > 0$

(b) SGD is computationally fast, but requires a high number of iterations and might fluctuate strongly.It simply follows the estimated steepest decent, and might easily get stuck in local optima. Adam (Adaptive Moment Estimation) behaves similar to a heavy ball with friction, making use of past iterations to calculate the equivalent of momentum and velocity to avoid fluctuations and tend towards flat minima in the error surface. Tesla AI Director Andrej Karpathy estimates Adam is used in 23% or more of deep learning papers, showing it is a popular and widely suited choice.

(c) Refer to Figure 4b. As expected, the output is still very bad: no training has taken place, so the model is not based on the data. For a "pure guess" scenario, the nonlinearities can not help.



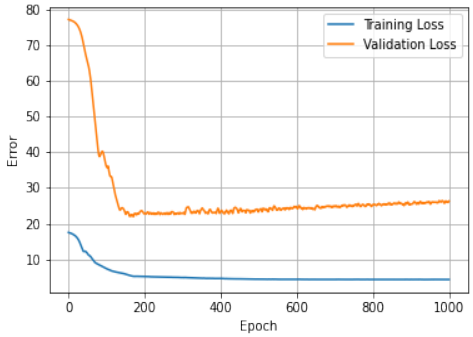
(a) Plot of ReLU function



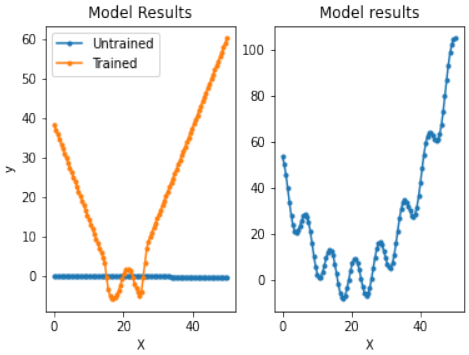
(b) Plot of predicted values from untrained ReLU model

Figure 4: Plot of ReLU function and untrained ReLU model.

(d) Refer to Figure 5a. We stuck with the general structure we had in Exercise 1 as this seemed to work well, but added a few more neurons to be able to capture this rather complex function.



(a) Train and validation loss for ReLU model



(b) Comparison between trained and untrained ReLU model with test data

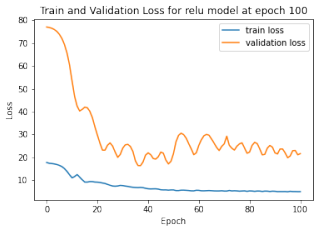
Figure 5: Plot of ReLU model performance

(e) Refer to Figure 5b. The nonlinearity introduced by the ReLU function means that the dynamics of the network can no longer be expressed in one linear function. That allows more complex functions to be built, resulting in a closer approximation of the data than in Q1 (where the result was a straight line).
While the function is still not perfectly matching the test data, it approximates the shape reasonably over the whole range and, and very well in the region around x=20.

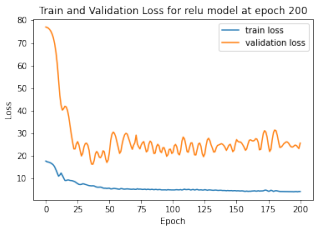
Exercise 3

(a) In the event that each epoch takes a considerable length of time to complete (large dataset or complex architecture), having the training progress visualised during training would be useful as we can see how the losses evolve over time. Should the gap between the training and validation loss remain wide continually for a number of epochs, we can decide to end the entire training prematurely to save time and try another architecture. Equally, we can determine if we reach a plateau and further training does not increase the performance - because of the rather small set of training data, this seems to already be the case after 100 to 200 epochs here.

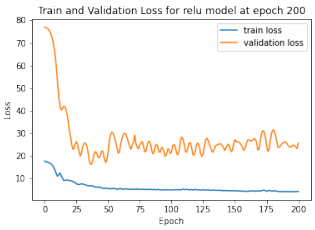
Refer to Table 1 in the next page for the output of the epoch visualisation.



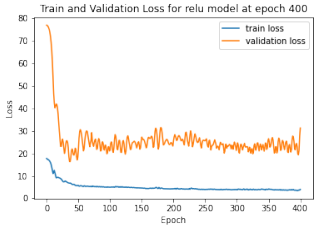
(a) Epoch 100



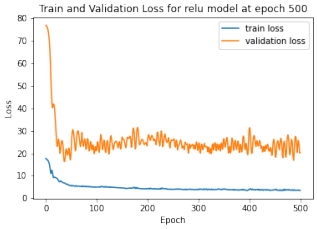
(b) Epoch 200



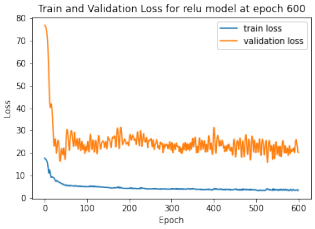
(c) Epoch 300



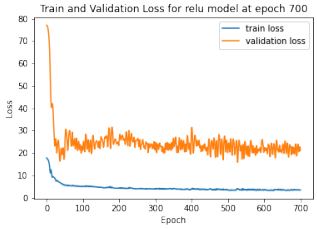
(d) Epoch 400



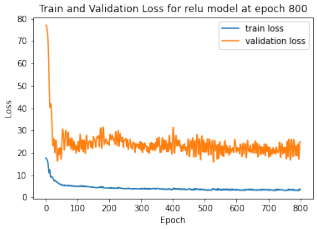
(e) Epoch 500



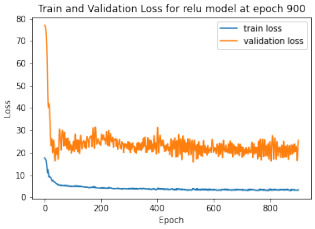
(f) Epoch 600



(g) Epoch 700



(h) Epoch 800



(i) Epoch 900

Table 1: Plots of increasing epoch length at intervals of 100 epochs, N=900 epochs