Hyperparameters Explored

Batch size

A batch size of 40 was used as the default setting in the data_iter. However to fulfil the following criteria:
'the total number of context points Nc should be different (and randomly chosen) for every batch so that the model learns to handle different number of context points at test time.'

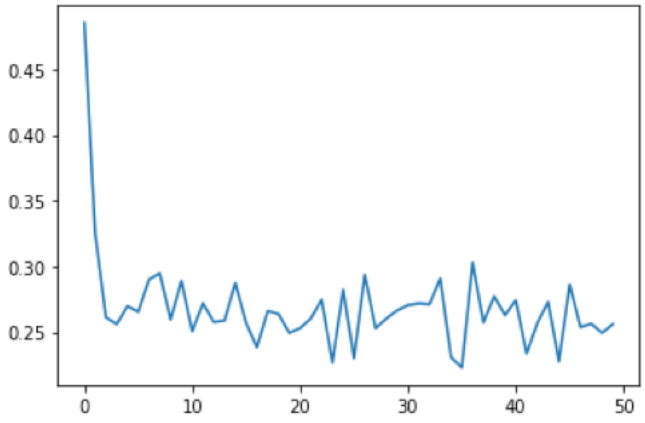
```
for X, y in train_iter:
      define number of context points - random number between 3 and 35
    n_c = random.randint(3, 35)
    x_tensor = X
      only take first n_c values of x to get model used to handling different number of context points
    x_split_tensor = torch.split(x_tensor, n_c)
    y_tensor = y
    y_split_tensor = torch.split(y_tensor, n_c)
```

This piece of code was written. This produces a random number for the number of context points (n_c). n_c is passed into the split function creating a tensor of dimensions (n_c, 40, 1). This means that a randomly sized batch of context data is being passed into the model for every X and y value in the data_iter.

Optimisers

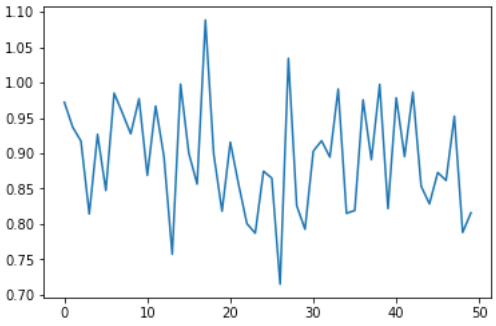
Adam

When using the Adam Optimiser it was found that it was the faster of the two Optimisers that were used. During training the goal was to get the loss of the model down to both an acceptable level and to a level in which the gradient of loss curve was almost flat. The number of epochs that were required to do this was a lot less than the SGD Optimiser. The Optimiser appeared to also have an impact on the plot of the loss of the model. When using a learning rate of 0.01 and 50 epochs the loss of the model started at around 0.5 and worked its way down to plateu around 0.25.



SGD

The SGD Optimiser was the slower of the two Optimisers that were tried. The Optimiser appeared to also have an impact on the plot of the loss of the model. When using a learning rate of 0.01 and 50 epochs the loss of the model started at around 0.97 and ended around 0.8. Here the impact of having a slower Optimiser is visible.



Given the results of the tests outlined Adam was chosen as the optimiser.

Loss Function

The MSELoss function was the only loss function tried. This loss function 'Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and the target y' (PyTorch, 2019). The model performs a regression after function selection to learn the relationship of the function. This, I believe, makes the MSE Loss function the best for this task.

Learning rate

Learning rates were tested at intervals of 1, 0.1 and 0.01 over the course of 50 epochs.

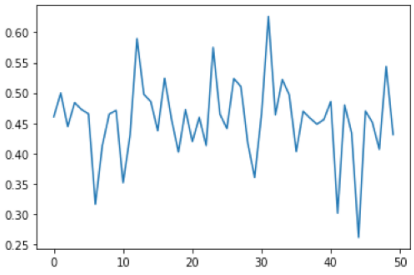


Figure 1 - Adam at Learning Rate 1

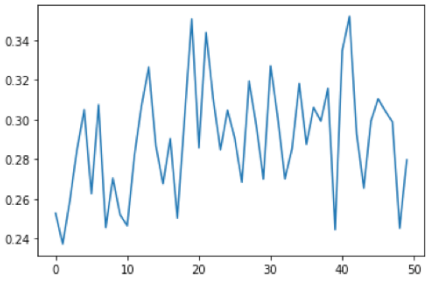


Figure 2 - Adam at Learning Rate 0.1

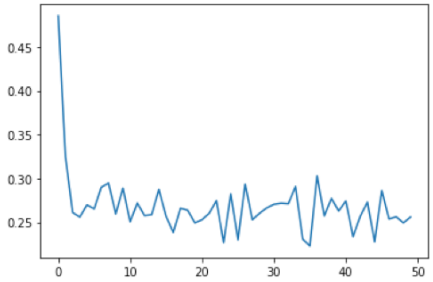


Figure 3 - Adam at Learning Rate 0.01

Ultimately a learning rate of 0.01 was chosen.

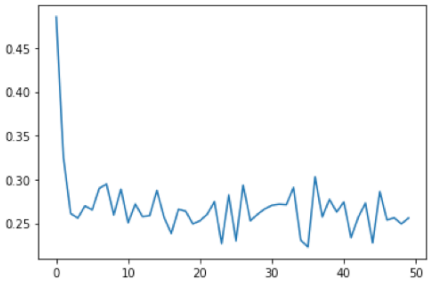
References

PyTorch, 2019. *MSELoss.* [Online]
Available at: https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html
[Accessed 13 April 2021].

Activation Functions

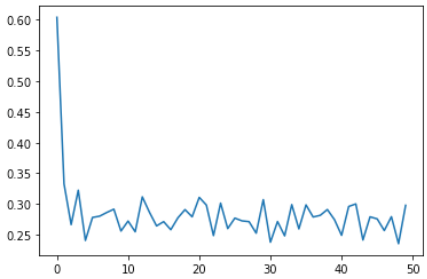
Sigmoid

When using the Sigmoid Activation function with MSELoss, Adam Optimiser and 50 epochs this was the resulting loss graph:



ReLU

When using the Relu Activation function with MSELoss, Adam Optimiser and 50 epochs this was the resulting loss graph:

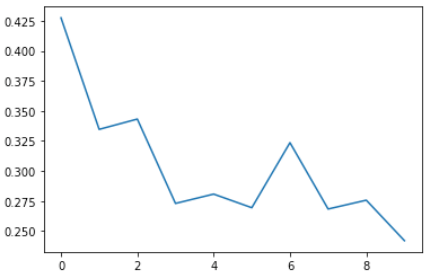


ReLU produced slightly more accurate results and therefore was chosen as the activation function.

Epochs

As visible in the previous tests using 50 epochs is too many. The loss of the model plateus after about 10 epochs. Therefore 10 was the value chosen for the number of epochs

Plot of Training Loss



6 Plots Showing Results of Applying Model