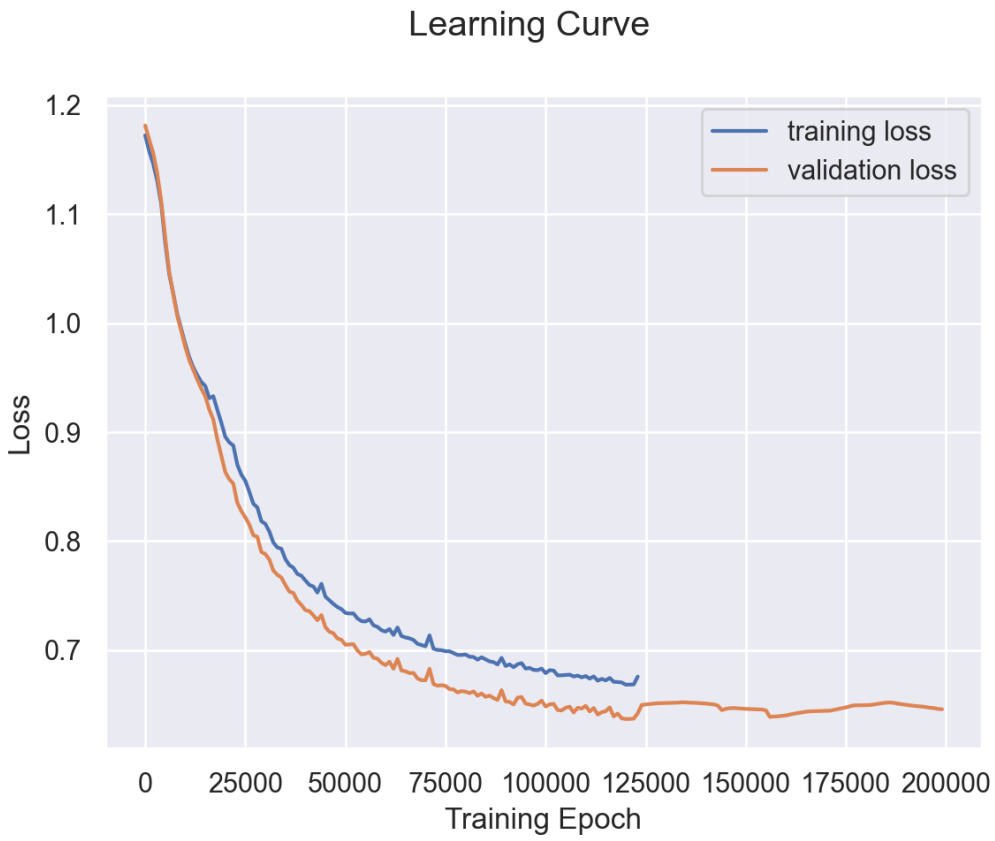
MSAI 349: Machine Learning HW#3

JAMES WILKINSON, DEVYANI GAURI, LILI BARSKY, JOSH CHEEMA, KALEEM AHMAD

Q1. Please find the code for our feed-forward neural network for insurability in our starter3.py file.

i. After running the model for 200,000 epochs, the resulting learning curves for our training and validation datasets are as displayed below. We found it compelling that the training loss was greater than the validation loss. This may have been due to noise in the training data, or the smaller size of the validation dataset, allowing it to be more linearly separable or to more readily conform to the model. We note that experiments viewing the learning curves showed no evidence of the model overfitting (which would have the training loss continuing to improve while the validation loss does not). We found the training asymptotes to a loss of 0.65, which trains in a reasonable amount of time and shows no sign of under-fitting. Going forward, we use 0.65 as a stopping-criteria for the model. For classification we used Cross Entropy as our loss function.



ii. Our confusion matrix and corresponding F1 scores are documented below. These suggest a robust model. The model involved all the optimal hyper parameter settings found in part iv.

Confusion Matrix - Insurability

	Good (actual)	Neutral (actual)	Bad (actual)
Good (predicted)	48	4	0
Neutral (predicted)	0	97	5
Bad (predicted)	0	3	43

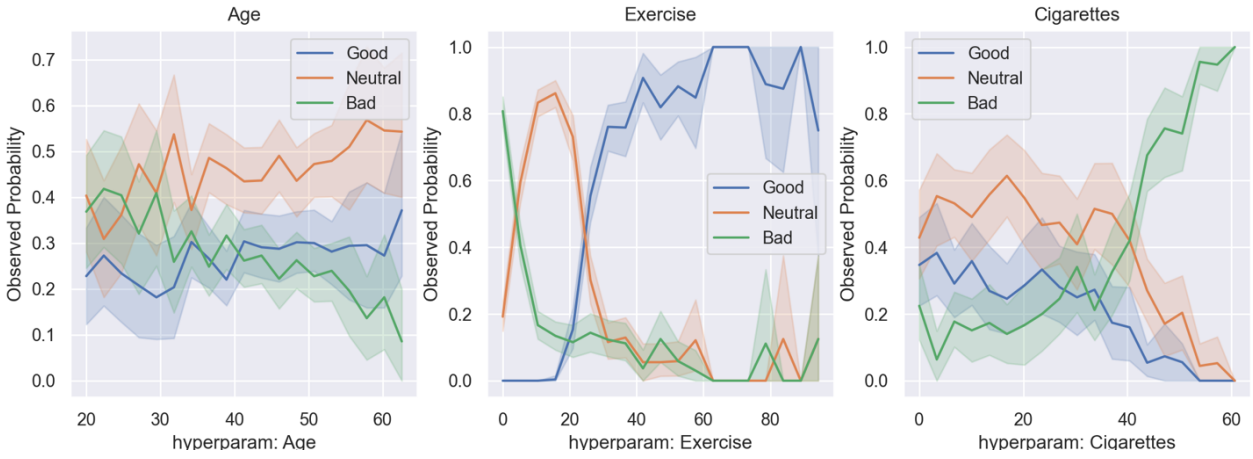
F1 Values:

Class Good: 0.96
Class Neutral: 0.94
Class Bad: 0.92

iii. A neural network is not the most ideal type of multiple-class classification algorithm for this type of analysis. This is because it is difficult to find justification for why a neural network produced a given output. In the case of insurance, it is important to be able to explain *why* the model has denied an individual insurance. As such, more transparent classifiers such as naive bayes could be useful.

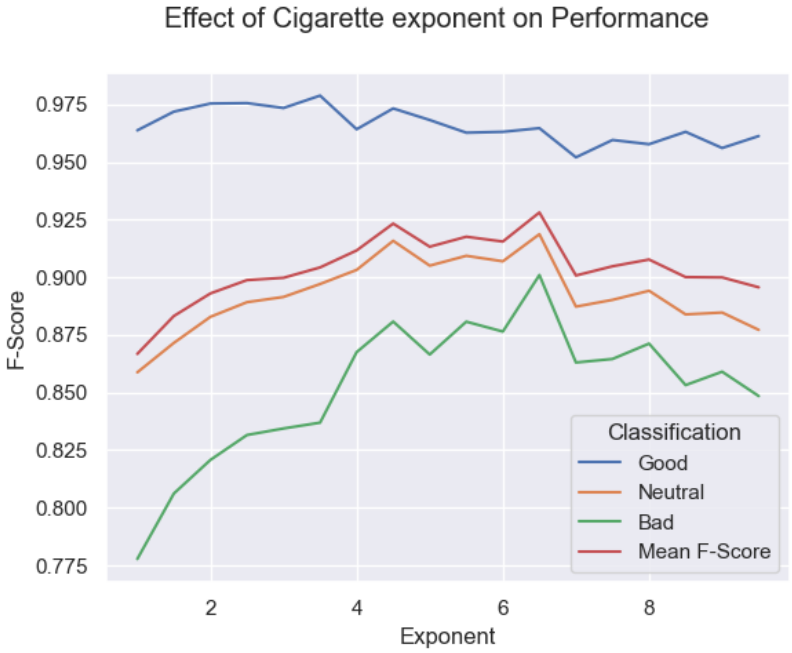
iv. For all analyses, hyper-parameters were first normalized to lie approximately between 0 and 1, utilizing the minimum and maximum values in the training dataset. We then independently evaluated the effects of varying the "cigarettes" parameter exponent, the presence of bias and momentum and target loss values, as described below:

a. **Cigarettes Parameter Exponent:** We first examined how the age, exercise and cigarettes parameters relate directly to the probability that the output for any class should be one.



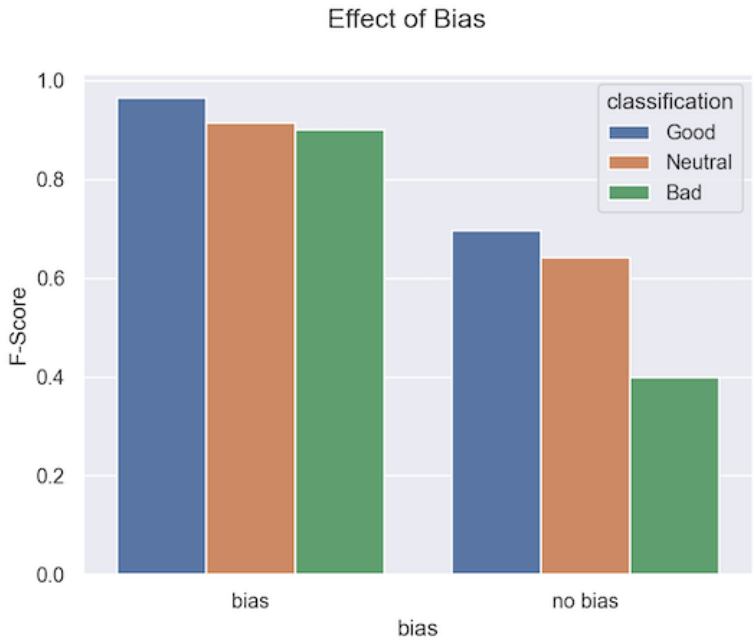
Based on this analysis, we observed that cigarettes has a distinct non-linear effect on the probability of an individual being classified as bad. We hypothesised from this that our model could benefit by replacing the hyper parameter “Cigarettes” with itself raised to a power, so that these relationships become more linear, and thus more sparsely distributed in hyperparameter space when compared to the target prediction*. Using the validation set, we determined that an exponent of 6.5 improved the model maximally, yielding the highest F-scores. Notice that the classification F-score for “Good” decreases marginally as the exponent of cigarettes is increased. However, the F-score for “Neutral” improves, and the F-score for “Bad” dramatically improves, along with the model’s overall average F-score. We decided 6.5 was optimal based on the compromise between these observations, which overwhelmingly point towards a net-benefit from replacing [Cigarettes] with [Cigarettes^6.5].

**Without raising cigarettes to a power, we can see from the chart that having 0-40 cigarettes (interval of 40 cigarettes) has little effect on the target output. However the smaller increase from 40-60 (interval of 20) has a significant effect. Raising cigarettes to a power will even out the distribution of target variables across the hyper parameter space.*

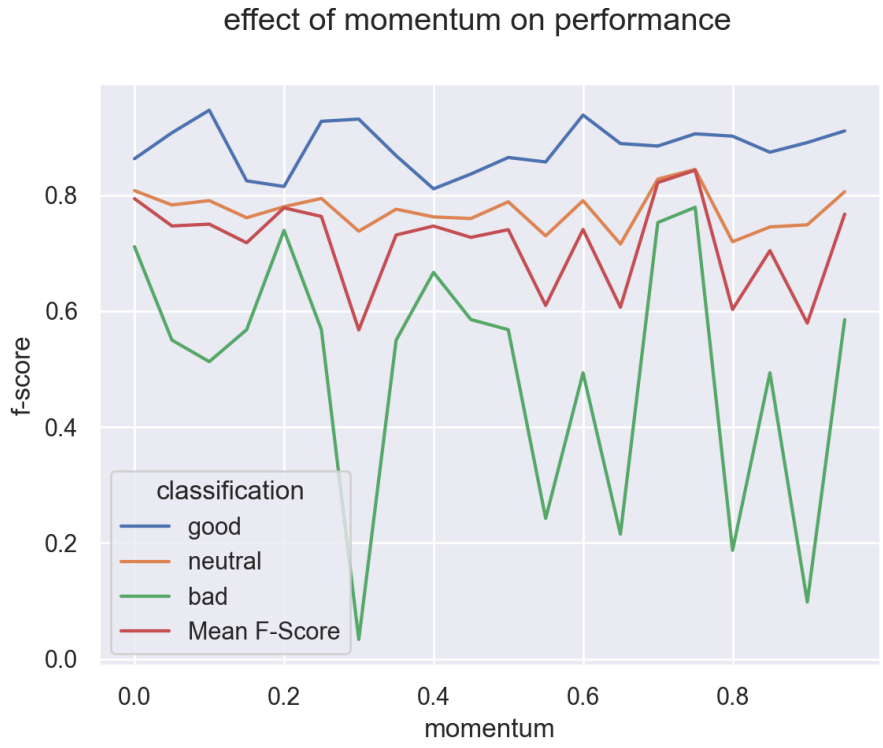


b. **Bias:** We determined that adding a bias term to the model improves the F-scores, as shown below. This is likely due to the flexibility afforded by the presence of bias, particularly the ability to shift the activation function to the left or right. It is

also effectively adding another hyper parameter to the model, and so improved accuracy (while not guaranteed) is somewhat natural to expect here. Note that the worst case scenario, assuming our gradient descent always converges to a global optimum, is that bias performs equally with no bias, as the weights corresponding to the bias terms would fall to 0.

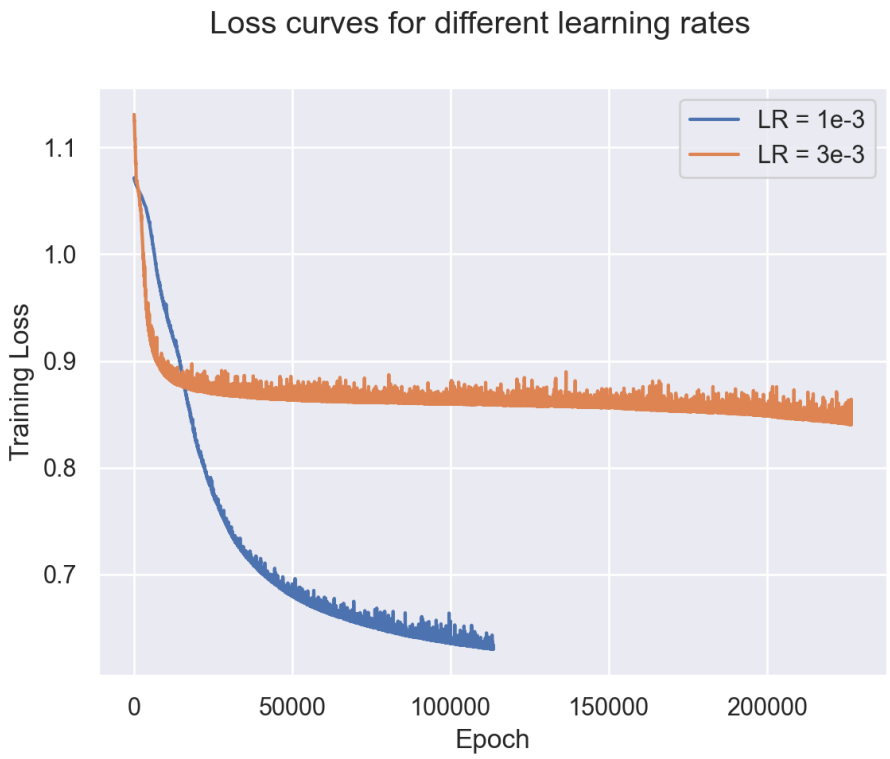


c. **Momentum:** While the value of momentum did not affect performance as shown here higher momentum was associated with faster learning, as evidenced by the training loss curves below. Thus, we opted for a momentum value of 0.9 which produced good accuracy, and fast learning.



d. **Target Loss:** As expected, a lower, stricter target loss value improved the F-scores for the model. We trained our models to a loss of 0.65 which produced optimal accuracies in a reasonable run-time.

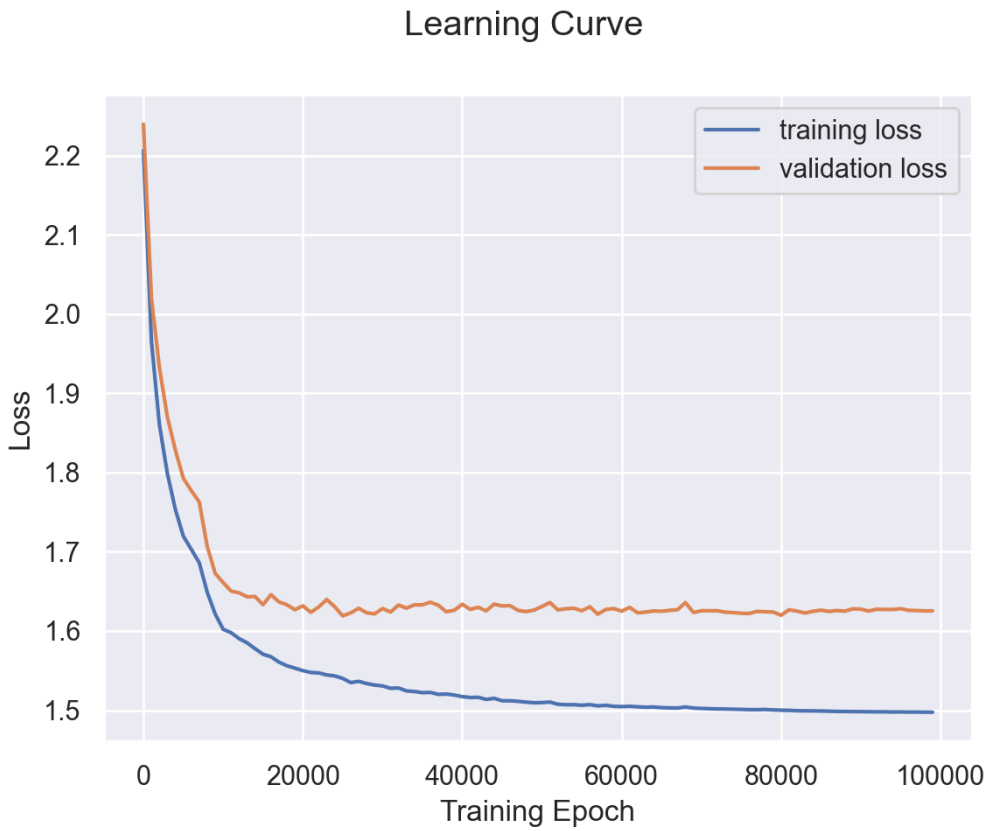
e. We also experimented with **Learning Rate:** We found that learning rate had an effect on both model performance (because training loss asymptotes at higher levels for higher learning rates), and speed (because higher learning rates increase the speed of learning). We found a learning rate of 1e-3 to be appropriate here. Note in the chart how the higher learning rate initially trains faster, but its inability to train past training losses of 0.9 quickly slows it down below the speed of the lower learning rate.



Q2. For our MNIST 10-class classification model, we implemented a deep feed-forward neural network. Prior to model training, as per assignment instructions, we performed the same data transformations as we implemented in HW#2. We converted the values to binary, as we found that the highest accuracy of our models in HW#2 were associated with this transformation. Additionally, we applied a similar variance threshold filter of 50%, removing pixels below this threshold. The 50% percentile value was chosen, as this yielded the best model performance when applied to the validation dataset, as shown below:

In terms of network architecture, the number of input nodes corresponded to the number of pixels, instead of the 3 input nodes utilized in Part 1, due to the vastly different sizes of the training datasets. The hidden layer in our MNIST model was designed to have 10 nodes, corresponding to each of the classes. We implemented the cross-entropy loss function here, as it permitted adaptation to a more scalable measurement, and is mathematically most beneficial in an inference framework of maximum likelihood. We selected a target training loss value of 1.6, as we found that our MNIST model stopped improving on the validation data after this point, and actually may have been prone to overfitting at values below this threshold. This is evidenced by our training and validation learning curves below.

Please also note that we opted to use the “Mean F-Score” across all classes as an accuracy metric. While in Q1, we could afford to individually optimise the scores across all output classes, we found this challenging with the MNIST data because of the high number of outputs (10).



Our confusion matrix and corresponding F-values are as follows, reflecting a relatively robust model, with the exception of Class 5 which was the weakest of all the classes.

Predicted \ Actual	0	1	2	3	4	5	6	7	8	9
0	17	0	1	1	0	0	0	0	1	2
1	0	25	1	0	0	0	0	0	1	1
2	0	1	15	0	0	1	0	1	0	0
3	0	0	0	16	0	1	0	0	1	0
4	0	0	0	0	22	0	0	1	0	2
5	1	0	0	0	0	7	0	1	2	0
6	0	1	1	0	1	0	13	0	0	0
7	0	0	1	0	0	0	0	21	0	1
8	0	0	0	1	0	4	0	0	16	0
9	0	0	0	0	2	0	0	0	0	16

F1 Values:

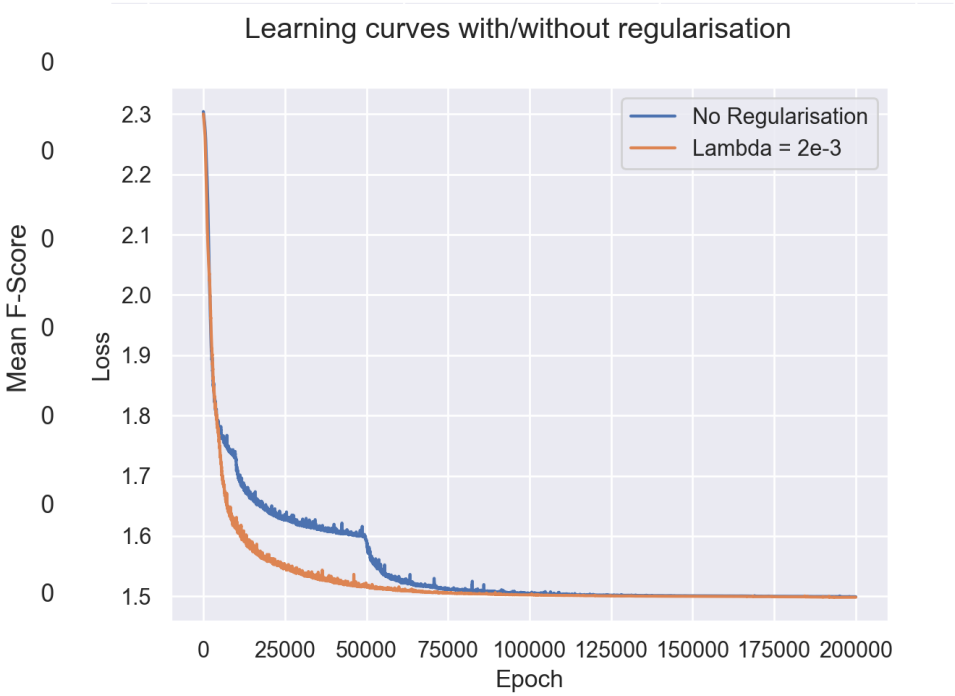
Class 0: 0.85
Class 1: 0.91
Class 2: 0.81
Class 3: 0.89
Class 4: 0.88
Class 5: 0.58
Class 6: 0.90
Class 7: 0.89
Class 8: 0.76
Class 9: 0.80

Ave F-Score: 0.83

When compared with our HW#2 models, not surprisingly, the performance of our MNIST neural network model exceeds that of the k-means function. This is due to the increased complexity of the neural network model as compared with the unsupervised looser structure of k-means. Additionally, a neural network implementation affords the ability for optimization by adjustment of hyper-parameters. Perhaps more unexpectedly, our k-nearest neighbors function appears to perform (marginally) better than our neural network. This may be the result of performing binary conversion. While dimensionality reduction was important for k-nearest neighbors performance, it may have negatively affected our neural network, which relies on a larger dataset for improved learning.

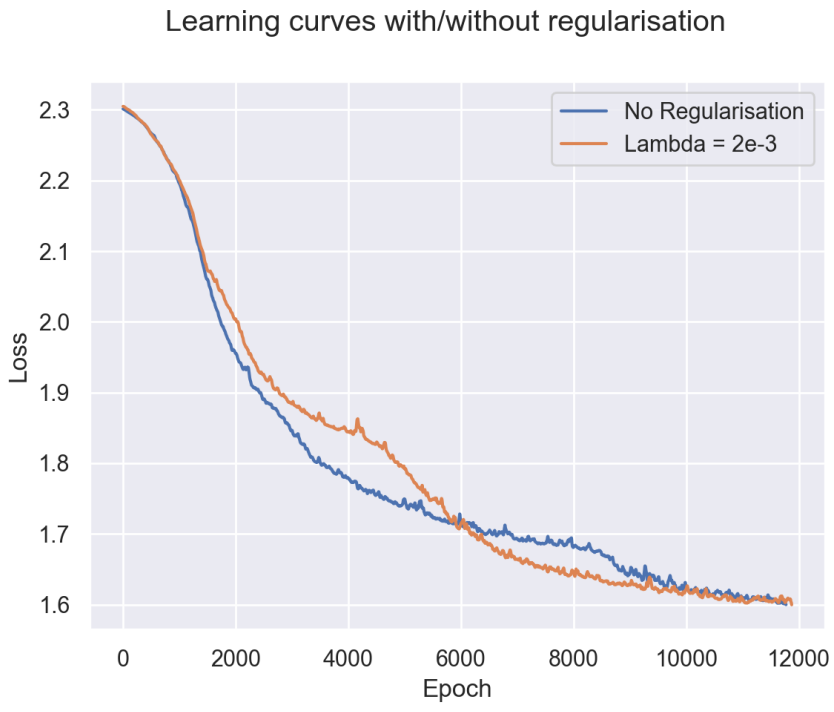
Q3. We implemented a L2 regulariser into the trainNN function within our code (NN.py). This was used to investigate the effect of regularisation on the MNIST data. We began by varying the strength of the regularisation from 0 upwards. We found that for high strengths of regularisation, the training process was unable to converge. This is because the regularisation term in the model is overpowering the underlying loss function. Hence, we tested regularisation strengths up to 0.01 (from 0). We tested many values of different orders of magnitude. The results are shown on logarithmic axes below, highlighting that a value of lambda = 2e-3 is optimal when tested on the validation data.
We note that the introduction of regularisation changes the learning curves. However, different experiments showed that it doesn't necessarily speed up or slow down the training process, but can have either of these effects depending on the starting conditions and the nature of the regularisation. This is because the regularisation can alter the loss-surface in different ways depending on its strength, thus effecting the learning process differently. Below we show two separate experiments for this showing the complexity of how regularisation can alter the learning process. In the top chart, it appears that regularisation speeds up the learning process.

Effect of regularisation constant on performance



In the bottom chart which is more “zoomed in” at a different scale, we can see this is not always the case, and the differences between the loss-surfaces becomes clearer.

The results of implementing regularisation with Lambda = 2e-3 can be found below:



Predicted \ Actual	0	1	2	3	4	5	6	7	8	9
0	16	0	1	1	0	0	0	0	2	2
1	0	25	0	0	0	0	0	0	0	1
2	0	1	17	0	0	0	0	1	0	0
3	0	0	0	15	0	1	0	0	1	0
4	0	0	0	0	21	1	0	1	0	1
5	2	0	0	0	0	7	0	1	0	0
6	0	1	0	0	1	0	13	0	0	0
7	0	0	1	1	0	0	0	21	0	1
8	0	0	0	1	0	4	0	0	18	0
9	0	0	0	0	3	0	0	0	0	17

F Values:

Class 0: 0.80
Class 1: 0.94
Class 2: 0.89
Class 3: 0.86
Class 4: 0.86
Class 5: 0.61
Class 6: 0.93
Class 7: 0.875
Class 8: 0.82
Class 9: 0.81

Ave F-Score: 0.84

We can see that regularisation has increased the accuracy of our model slightly. While some classes were predicted slightly poorer, the majority were improved, and the model's average F-score has increased from the case where no regularisation was used.

Q4. Please see the code for our 3-class feed-forward neural network, with gradients calculated and applied without an optimizer and with no bias terms, in our starter3.py file. Upon testing this model, we found that it was associated with a significant amount of loss. For example, in one iteration, it did not decrease below a threshold of 1.08, following approximately 6500

epochs. This may be the result of not adding any bias term to the network, and it could also have been improved by changing the learning rate for this implementation.