**Philip Mortimer ML**

Task 1

1a + b)

A graph of a graph

Description automatically generated with medium confidence

Figure - Validation and training loss of neural network.

I trained a neural network to classify activity type. My model achieves a validation accuracy of 84%. The model’s training loss flatlines after about 5 epochs, demonstrating quick convergence. The validation loss follows a similar trend initially. However, the validation loss is much less smooth and only really settles after about 40 epochs. The final validation loss is about 30% larger than the training loss, which indicates that the model is overfitting to the training set slightly. However, given that both training and validation loss follow similar trends and achieve similar loss, I would say the model has trained effectively.

When training the model, I noticed that the neural network will often get stuck at a local minimum of about 60% classification accuracy. Thus, it typically requires multiple training attempts with different weight initialisations to avoid this local optimum and achieve a higher classification accuracy.

On the test dataset, the neural network achieves a loss of 0.319 and an accuracy of 89.9%. This would indicate that the model generalises well as this is marginally higher than both the training and validation classification accuracies.

1c)

There are wide variety of model hyperparameters to tune when training a neural network. I have decided to analyse the learning rate, neural network architecture, activation function and batch size. For each feature, I have selected a small number of values which are commonly used and cover a broad range of options. To compare them, I define a base neural network architecture and then investigate the relative performance as each variable is changed individually. Each model is trained for the same number of epochs to ensure effective comparison. For the number of training epochs, we can see from *Figure 1* that 5 epochs tends to be sufficient for a model to achieve strong classification. To select the best overall parameters, a randomised grid search would be a suitable search strategy.

A graph of a graph showing the impact of learning rate on model training

Description automatically generated

Figure - Impact of learning rate on ANN performance

*Figure 2* depicts the impact learning rate has on performance. From this graph, we can see that the model fails to learn for learning rates of 1e-5 and 0.1, whilst it converges for a learning rate of 0.001. This suggests that that 1e-3 is a stronger choice of learning rate. Indeed, this is conclusion is widely adopted as this is typically the default learning rate for the ADAM optimiser [1] and is viewed to be a value that yields effective performance [2]. A larger learning rate means that step sizes are larger, leading to faster initial convergence but also typically struggling to converge at the exact optimum point. Conversely, smaller step sizes converge more slowly but more accurately, even if they are more prone to getting stuck at local optima. However, in the case of ADAM, an adaptive learning rate algorithm, these problems are typically less apparent. A graph of a model

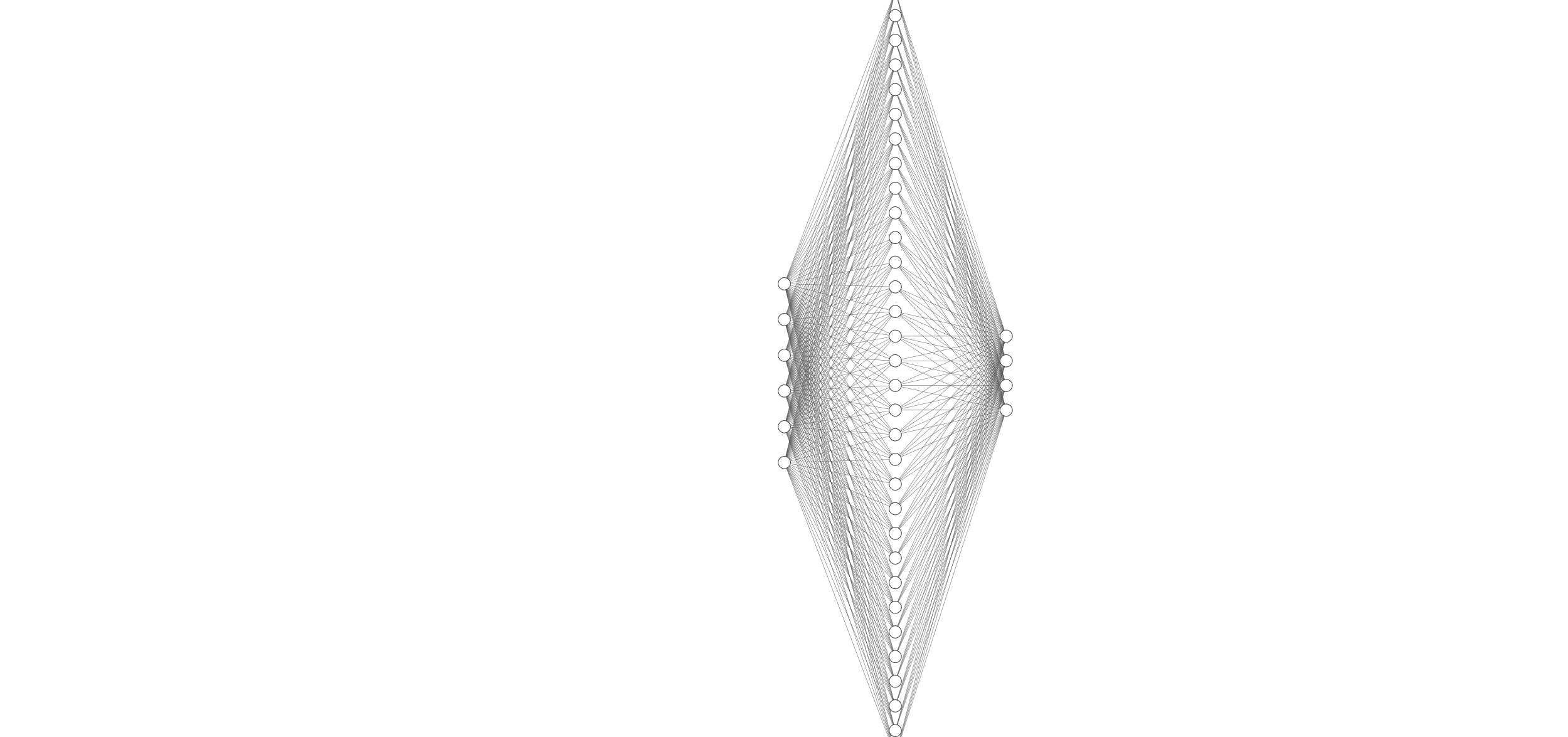
Description automatically generated

Figure 3 – Impact of model architecture on training performance.

Figure 4 – Neural network architecture with one hidden layer.

In order to assess the impact of the number of hidden layers, I created three networks with differing numbers of hidden layers. Each hidden layer has 32 neurons. It appears that larger networks converge more slowly initially but end up achieving a lower loss. This makes sense as larger networks have more capacity to model complex data relationships. However, they also have more tuneable parameters which means that they will likely learn more slowly. In the case of this instance, the difference in performance between 2 and 3 hidden layers was negligible, suggesting that both have learned the same function. Thus in this case, I would suggest that 2 hidden layers is the optimal configuration. Particularly as larger networks tend to have the capacity to overfit to training data as they can almost memorise specific training examples with their surplus power.

A graph of a function

Description automatically generated

Figure 5 - Impact of activation function on ANN training.

I experimented with three commonly used activation functions for neural networks: ReLu, hyperbolic tangent (tanh) and the sigmoid function. These functions have shown to perform well on a number of tasks. Both the sigmoid and hyperbolic tangent functions perform similarly and better than ReLu. This makes sense as the sigmoid function and tanh are fundamentally similar functions, involving exponential functions, and are commonly used for classification tasks.

A graph of a graph showing the impact of optimiser on model training

Description automatically generated

Figure 6 - Impact of model optimiser on training.

I experimented with 3 commonly used optimisers for training artificial neural networks: ADAM, stochastic gradient descent (SGD) and RMSprop. Both ADAM and RMSprop are adaptive learning functions where each weights value is altered by a different rate that decays over time. These approaches are considered state-of-the-art and are widely used. Stochastic gradient descent simply updates each weight by its error gradient multiplied by the learning rate. This approach performs noticeably worse than the adaptive optimisers.

A graph of a graph

Description automatically generated with medium confidence

Figure 7 - Impact of batch size on training.

It is clear that the model struggles to train when the batch size is too small. Batch sizes of 1 and 2 prove too small for the model to converge, whilst batch sizes greater than 16 do appear to work. This is likely because smaller batch sizes make the overall step direction too inaccurate.

Whilst the hyperparameters are interdependent (e.g., the optimal activation function may differ depending on the network architecture), it is clear that some parameters are more important than others. For instance, for this task, the network architecture seems to yield similar results regardless of size. However, using an adaptive optimiser and using either the sigmoid function or tanh as an activation function do seem to impact network performance significantly.

2a + b)

A single decision tree achieves a test accuracy of 83%. Training an ensemble of 200 trees using Adaboost achieves an accuracy of 92%, which is a significant improvement. The ensemble improves performance as each model is trained on more data samples that previous models got wrong. This means that models address the flaws in prior models and means that each model error is related less closely. The models are also weighted according to their classification strength. Thus, the ensemble improves performance over a single tree.

2c)

A graph with a blue line

Description automatically generatedA graph of different colored bars

Description automatically generated

Figure 9 – Individual model errors.

Figure 8 - Error rate of ensemble with different number of models.

The ensemble error rate decreases with the number of models until it plateaus at a final error rate. Twenty individual models appear to be the optimum amount with a low error rate and reduced inference/training time. The individual model error rate appears to be randomly distributed. The ensemble does outperform the average error of individual models. This is expected as we know from the application of Jensen’s inequality that ensemble error will always be at most the average error of individual base models. The ensemble error is bound as follows: where , and .

2c)

A graph of a tree

Description automatically generatedA graph of a tree depth

Description automatically generated

Figure 11 – Training runtime with as max depth varies.

Figure 10 - Impact of tree depth on performance.

A diagram of a number

Description automatically generated

Figure 12 - Example of a depth 2 tree.

The ensemble accuracy increases with maximum tree depth. This is likely because each tree is capable of making more complex splits on the data, leading to better classification. This holds till about a depth of 10. This is because whilst the trees could become deeper, it doesn’t as the data is already sufficiently classified with respect to the training set. As the maximum tree depth increases, so does the runtime. Similar to *Figure 10*, *Figure 11* plateaus at a depth of around 10, indicating that model complexity stagnates at this depth. As decision trees become larger, they are capable of making more advanced (and more accurate classifications). However, they risk overfitting to the training set. Additionally, such trees quickly become hard to interpret as depth grows – a key advantage over other classification techniques.Altering the splitter and criterion hyperparameters seems to have very little impact, suggesting that the model is robust and not overly sensitive to these parameters.

References

[1] <https://keras.io/api/optimizers/adam/>

[2] <https://arxiv.org/pdf/1412.6980.pdf>