Harshvardhan
Student Id: 609162
COSC 522: Machine Learning

Task 1

Difference between batch processing and online processing

We have to update our gradient ∇C separately for each training point and then average over all the values for gradient descent. This is easy for a small dataset, but as the number of observations increase, this update can be costly and very slow.

Stochastic gradient descent can be used for faster updates: only a selected few points can be used to calculate ∇C. Thus, the weights and biases can be updated by using a few points for calculating gradients. This is known as batch processing, as the slopes are calculated in batches.

Consider the case that the mini-batch size is just one. That is, we update our weights and biases after every iteration. This procedure is known as online or incremental learning. Online system updates parameters instantly after every iteration; batch processing system updates parameters after every epoch.

Difference between gradient descent and stochastic gradient descent

The difference comes from the calculation of gradient ∇C. In gradient descent, we use all the data points to update the slope ∇C. In stochastic gradient descent, we use selected observations of the overall data to calculate the gradient ∇C.

When the data size is large, calculating gradient descent will take a long time because we will update the weights and biases for every iteration. Stochastic gradient descent will converge much faster as we will only use a set of observations.

The cost function is not entirely minimised since we use fewer observations to calculate the gradient in stochastic gradient descent. However, the approximation is valuable enough in most cases.

The Minibatch is a mixture of both approaches. We don't utilise all points or one single point. We use an arbitrarily chosen set of data from the complete dataset. Along these lines, we reduce the cost function and get a lower variance than the stochastic gradient.

Difference between perceptron and sigmoid neurons

Perceptrons are functions that take several binary inputs and produce a single binary output. Sigmoid neurons are modified perceptrons such that a slight change in the model's weights and biases will cause only a tiny change in their output.

Unlike perceptron, which gives binary output, sigmoid neurons use a sigmoid function or logistic function to rescale binary results to a value between 0 and 1. Perceptrons are step functions, and sigmoid neurons are smoothed versions of a step function.

The smooth nature of sigmoid neurons implies that a slight change in weights or biases will cause only a tiny change in the output.

Difference between feedforward and backpropagation

In feedforward neural networks, the information flows only in one direction — from the input to the output layer. Once the weights are decided at a layer, they are usually not changed.

In backpropagation, the information flows from the input layer to the output layer. The error is calculation is communicated back to the previous layers, and the nodes' readjust their weights and biases.

Need to introduce bias

A neuron's output is determined because the weighted sum is more significant than a specific threshold value. Because different decisions can have an extra level of decision making, they can have different threshold values.

Bias (equal to the negative threshold) measures the threshold for decision making and change. Also, since perceptrons are linear boundaries, the line would have to pass through the origin if there is no intercept term. This severely limits the hyperplanes we have for dividing the observations.

Task 2

Task 2.1

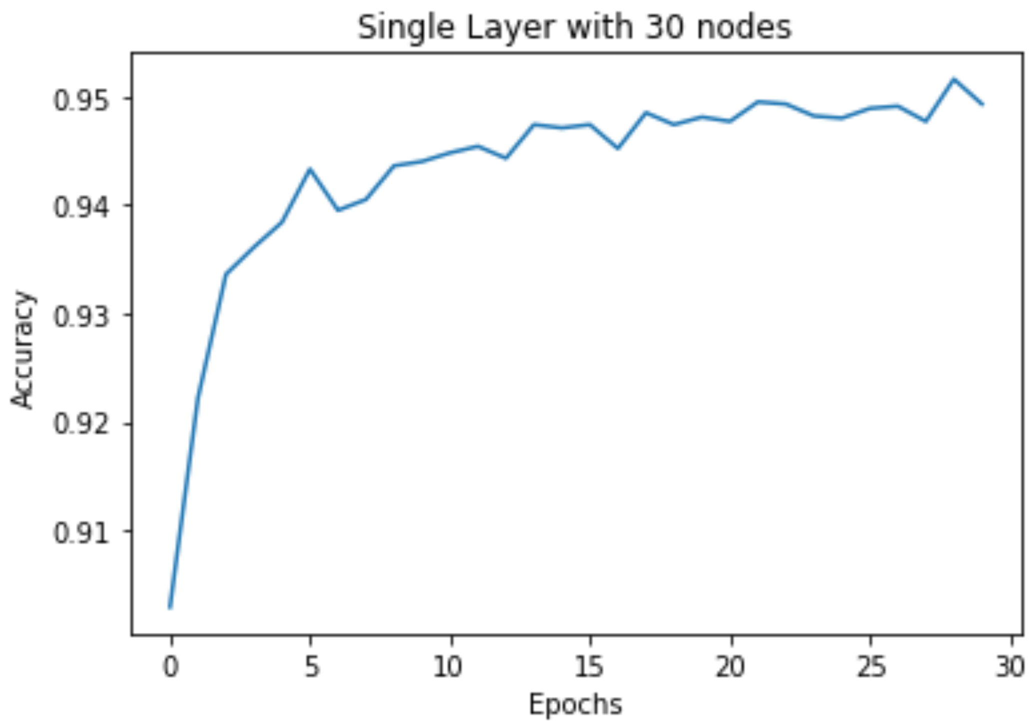


Figure 1 Accuracy vs Epoch Size, Nielsen's Example

As expected, with the increase in the number of epochs, the accuracy increases.

Task 2.2

A.

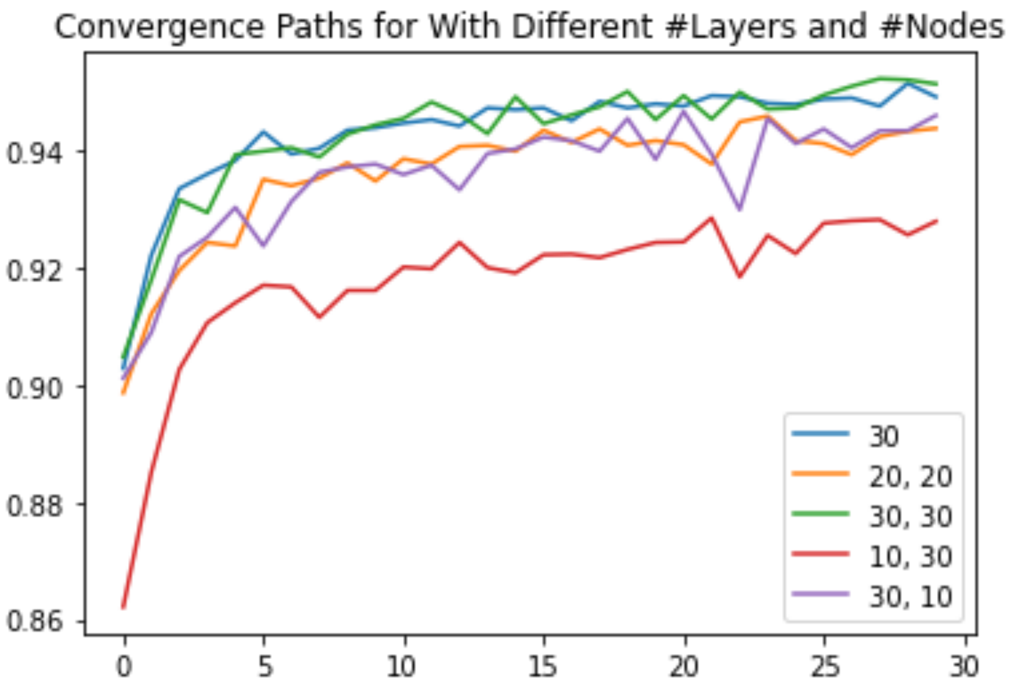


Figure 2 Effect of Network Structure

A single layer neural network performs surprisingly better than most double-layer networks. The best performances are offered by (30,30), i.e. two hidden layers with thirty nodes each and (30) a single hidden layer with thirty nodes.

Because the latter model is more straightforward, I will continue to use this for future use cases.

Accuracy vs Epoch

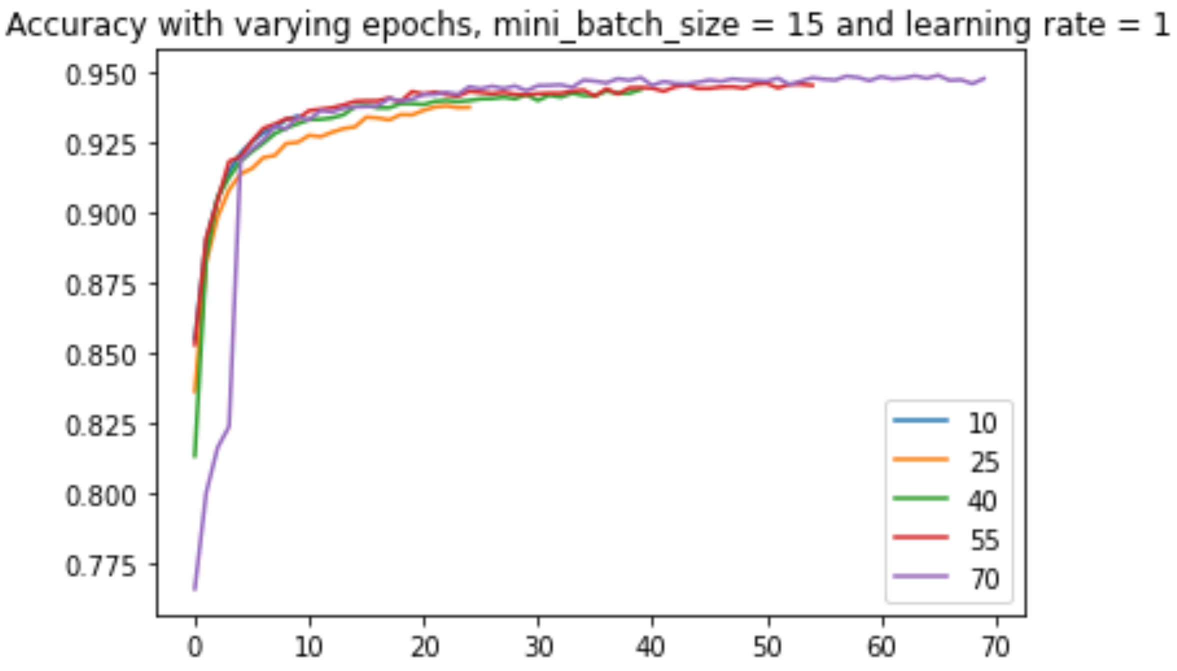


Figure 3 Effect of Epoch Size

The accuracy increases with the increase in epochs. However, beyond a point, there is no noticeable increase in accuracy. It probably makes sense to stick with 20 as the number of epochs. There is minimal gain in the accuracy but a massive increase in runtime.

For further calculations, I will use 20 as the number of epochs.

Accuracy vs Mini-batch Size

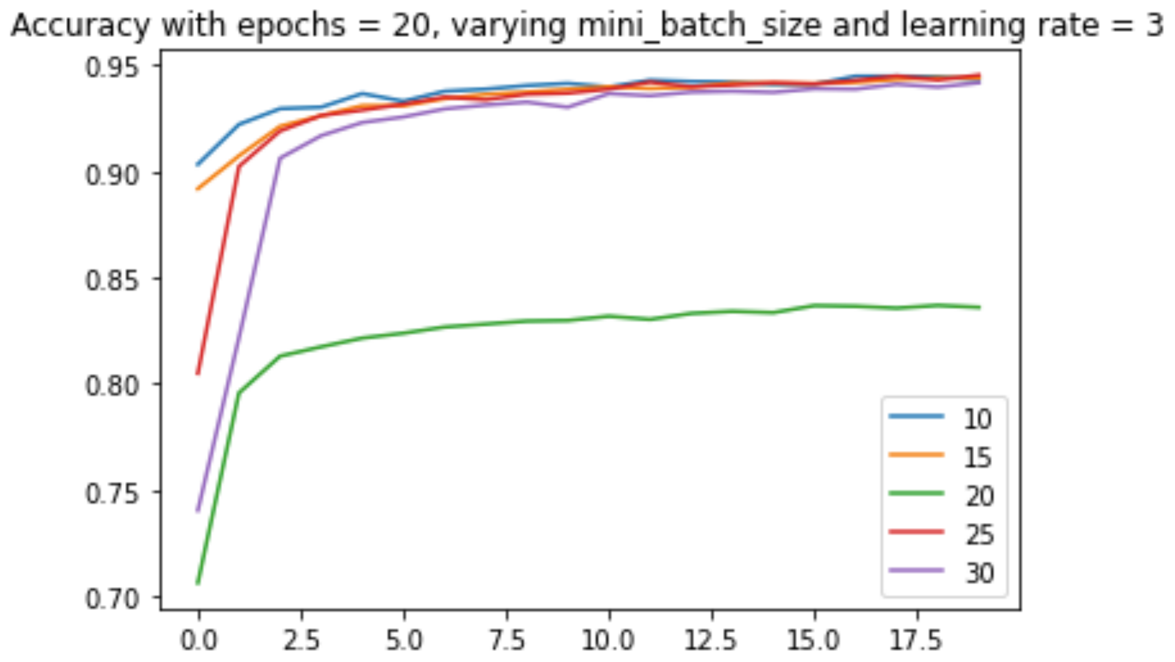


Figure 4 Effect of mini-batch size

There is no linear relationship between the batch size and accuracy. 20 as the batch size gives the least accuracy, and 10 provides one of the highest accuracies. 30 as batch size is also good. Since larger batch size implies costly execution, I will use batch size 10 for future analyses.

Accuracy vs Learning Rate

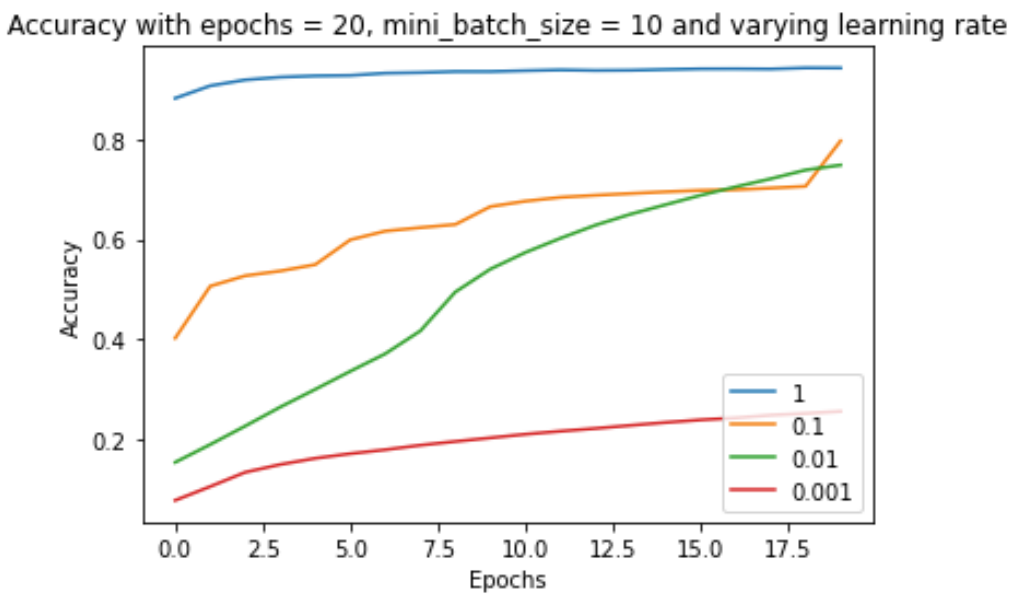


Figure 5 Effect of Learning Rate

Different learning rates give different accuracy increases. The best learning rate out of all these is 1. It is clear that if we choose a minimal learning rate (in this case), we will have no significant accuracy. Using a 0.001 learning rate, we get the highest accuracy with the 20th epoch, which is less than 0.3. Thus, I recommend using one as the learning rate.

Thus,

- Best epoch size: 20
- Best mini-batch size: 10
- Best learning rate: 1

Task 2.3

Overall Convergence Path

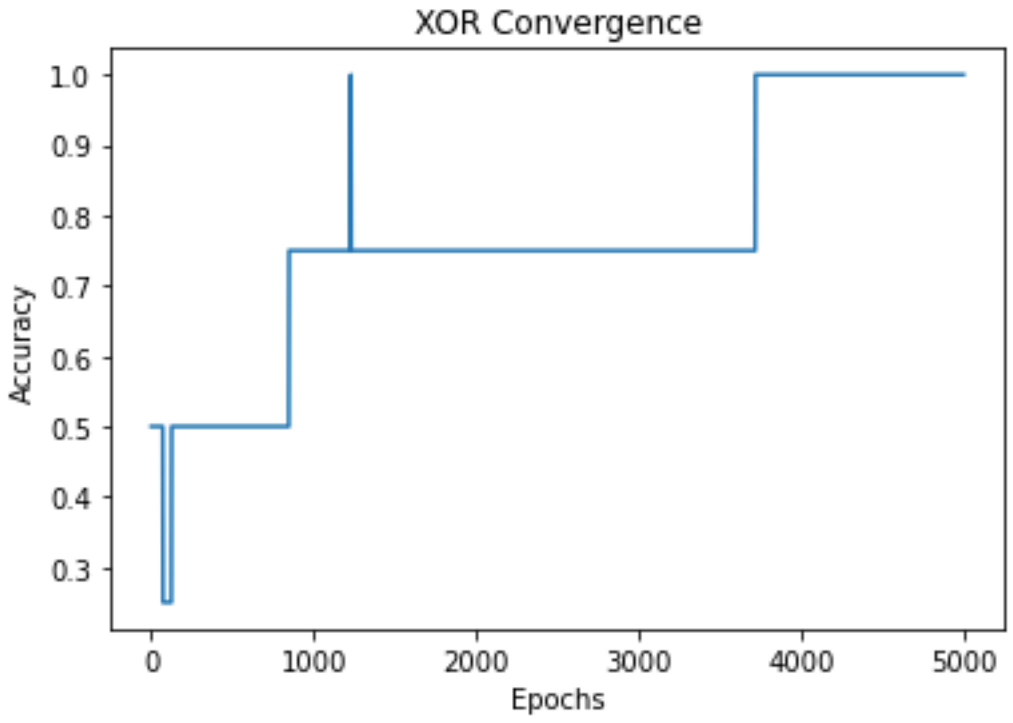


Figure 6 XOR Convergence with three hidden layers (50,10,50), 5000 epochs, mini-batch size as one and learning rate as 0.01.

XOR gate takes a lot of epochs to converge because the boundary of separation is not linear in 2-d space. But slowly, with more complicated networks, the situation improves, and we can find NN that converges the XOR gate.

Effect of Network Structure

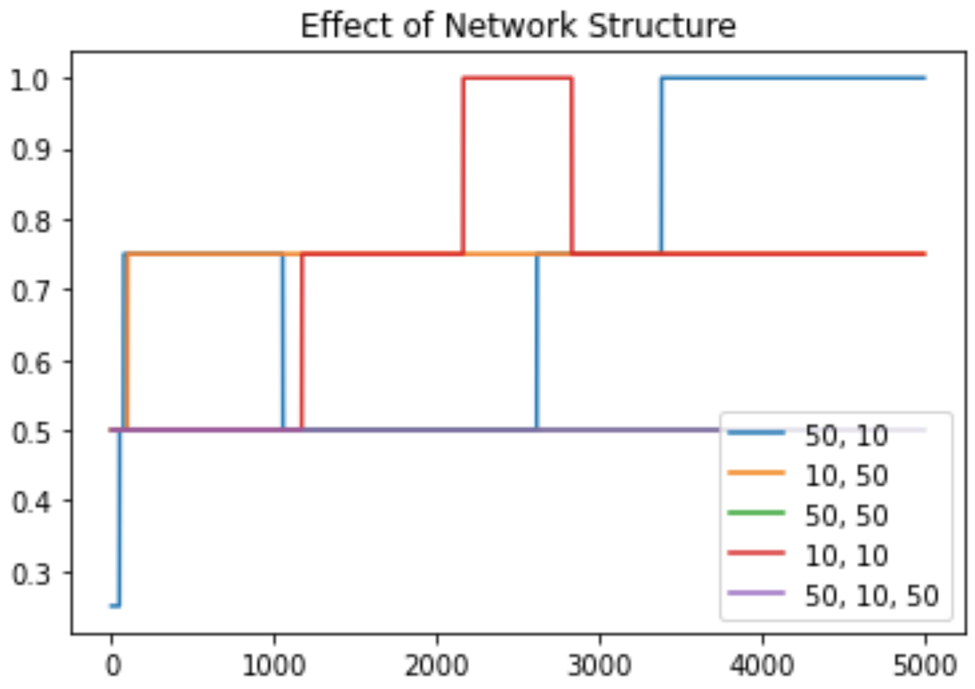
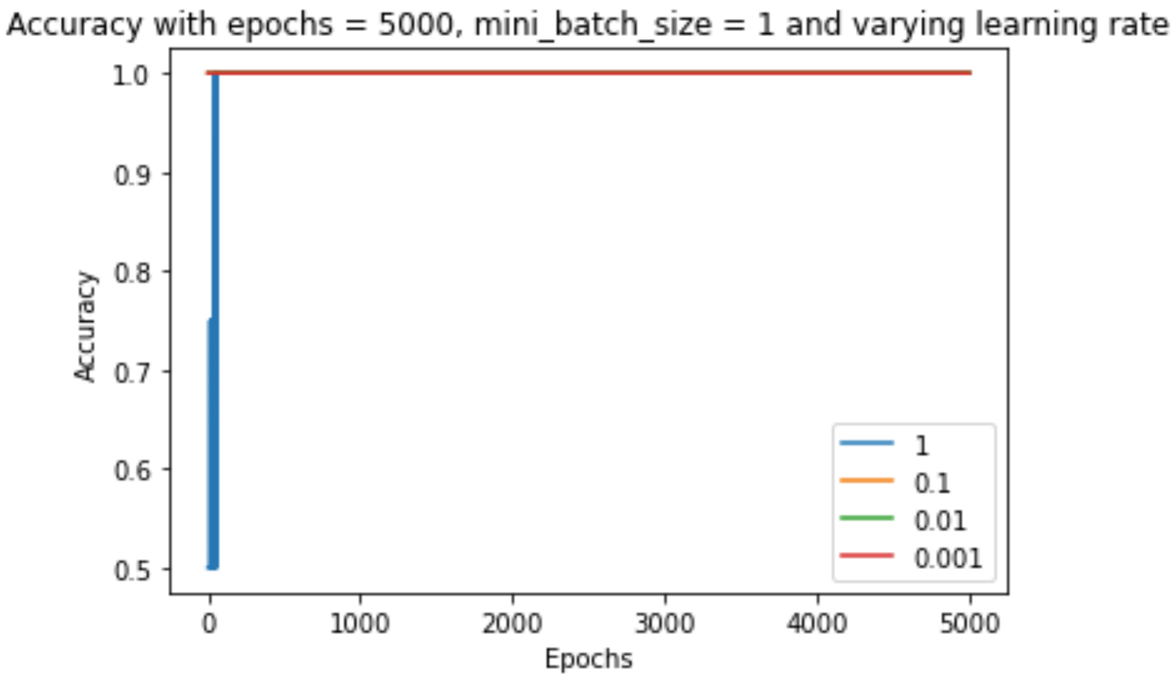


Figure 7 Effect of Network Structure

We need a complex network for XOR. That's why it is so complicated and requires so many hidden layers and nodes.

Effect of Learning Rate



We need a puny learning rate like 0.001 to converge.

Effect of Minibatch Size

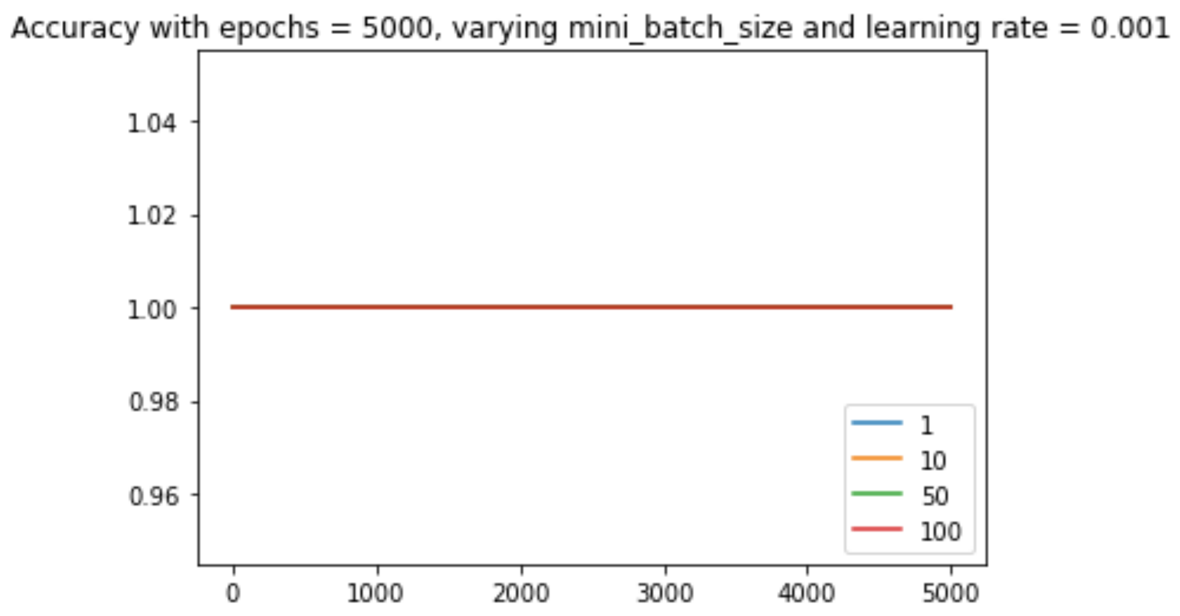


Figure 8 Effect of mini-batch size.

It is clear from the plot that the mini-batch size doesn't matter. Therefore it makes more sense to use a moderate mini-batch size and not 1.

Why is it more challenging to train a smaller network?

It is more challenging to train the XOR network because it is more complicatedly designed. XOR gate and the boundary is not linear in two-dimensional space. To separate them, we need to project them in the higher dimension and then split them. This requires multiple hidden layers with many nodes, rendering the network complicated to interpret but valuable.

By training my model for these two projects, I realised that the number of hidden layers is the first network to optimise for. We should start with many hidden layers and nodes and then selectively prune the ones that are not required.

The weights are stochastically determined using stochastic gradient descent, and thus we can optimise for it appropriately. For the hyperparameters, the primary goal is to find the ones that maximise the accuracy while not blowing up computational overload.

So, my step-by-step recommendations are the follows.

1. Choose the number of weights to be roughly one-tenth of the training dataset sample size.
2. Start with a large number of hidden layers and nodes. Prune or eliminate weights selectively.
3. Try several options for network structure, mini-batch size, epochs and learning rate. Choose the ones that give the best accuracy without significant cost overheads.
4. Reconstruct the final neural network with the tuned hyperparameters and network structure.