Assignment 2 Report

This is an outline for your report to ease the amount of work required to create your report. Jupyter notebook supports markdown, and I recommend you to check out this cheat sheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet). If you are not familiar with markdown.

Before delivery, **remember to convert this file to PDF**. You can do it in two ways:

1. Print the webpage (ctrl+P or cmd+P)
2. Export with latex. This is somewhat more difficult, but you'll get somehwat of a "prettier" PDF. Go to File -> Download as -> PDF via LaTeX. You might have to install nbconvert and pandoc through conda; `conda install nbconvert pandoc`.

Task 1

Task 1a)

(a)

$$a_j = f(z_j)$$

$$z_j = \sum_{i=0}^{I} w_{ji} x_i$$

$$\frac{\partial z_j}{\partial w_{ji}} = x_i$$

$$\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial a_j} \times \frac{\partial a_j}{\partial z_j} \times \frac{\partial z_j}{\partial w_{ji}}$$

$$= \frac{\partial C}{\partial z_j} \times \frac{\partial z_j}{\partial w_{ji}}$$

$$= \delta_j \times \frac{\partial z_j}{\partial w_{ji}}$$

$$= \delta_j \times x_i$$

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}}$$

$\therefore$ $w_{ji} = w_{ji} - \alpha \delta_j x_i$ (shown)

$$z_{kj} = \sum_{j=0}^{J} w_{kj} x_j$$

$$C = \sum_k C_k$$

$$\frac{\partial C}{\partial C_k} = 1$$

$$\frac{\partial C}{\partial z_k} = \sum_k \frac{\partial C_k}{\partial z_k}$$

$$\delta_j = \frac{\partial C}{\partial z_j} = \frac{\partial C}{\partial z_k} \times \frac{\partial z_k}{\partial a_j} \times \frac{\partial a_j}{\partial z_j}$$

$$= \sum_k \frac{\partial C_k}{\partial z_k} \times \frac{\partial z_k}{\partial a_j} \times \frac{\partial a_j}{\partial z_j}$$

$$\frac{\partial C}{\partial C_k} \times \frac{\partial C_k}{\partial z_k} = \frac{\partial C}{\partial z_k}$$

$$1 \times \frac{\partial C_k}{\partial z_k} = \frac{\partial C}{\partial z_k}$$

$$\frac{\partial C}{\partial z_k} = \frac{\partial C_k}{\partial z_k}$$

$$\therefore \quad \delta_j = \sum_k \frac{\partial C}{\partial z_k} \times \frac{\partial z_k}{\partial a_j} \times \frac{\partial a_j}{\partial z_j}, \text{ with } \delta_k = \frac{\partial C}{\partial z_k}$$

$$z_k = \sum_j w_{kj} a_j$$

$$\frac{\partial z_k}{\partial a_j} = w_{kj}$$

$$a_j = f(z_j)$$

$$\frac{\partial a_j}{\partial z_j} = f'(z_j)$$

$$\therefore \quad \delta_j = \sum_k \delta_k \times w_{kj} \times f'(z_j)$$

$$= f'(z_j) \sum_k w_{kj} \delta_k \qquad (\text{Shown})$$

Task 1b)

(b)

Hidden layer – Output layer

$[m, n]$ means a $m \times n$ matrix

$$w_{kj} := w_{kj} - \alpha \delta_k a_j^T$$

$$\delta_K = (\hat{y}_K - y_K) = \begin{bmatrix} \hat{y}_1 - y_1 \\ \hat{y}_2 - y_2 \\ \vdots \\ \hat{y}_K - y_K \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_K \end{bmatrix} \rightarrow [k, 1]$$

$$a_J = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_j \end{bmatrix} \rightarrow [j, 1]$$

$$a_j^T = [a_1 \ a_2 \ a_3 \cdots a_j] \rightarrow [1, j]$$

$$\alpha \delta_K a_J^T = \alpha \begin{bmatrix} \delta_1 a_1 & \delta_1 a_2 & \cdots & \delta_1 a_j \\ \delta_2 a_1 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \delta_K a_1 & \cdots & \cdots & \delta_K a_j \end{bmatrix} \rightarrow [k, j]$$

α is a scalar

$$w_{KJ} := \begin{bmatrix} w_{11} - \alpha\delta_1 a_1 & w_{12} - \alpha\delta_1 a_2 & \cdots & w_{1j} - \alpha\delta_1 a_j \\ w_{21} - \alpha\delta_2 a_1 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ w_{K1} - \alpha\delta_K a_1 & \cdots & \cdots & w_{Kj} - \alpha\delta_K a_j \end{bmatrix} \rightarrow [k, j]$$

Input to hidden

$w_{ji} := w_{ji} - \alpha \delta_j x_i^T$ where α is a scalar.

$$w_{ji} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ w_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ w_{j1} & \cdots & \cdots & w_{ji} \end{bmatrix} \rightarrow [j, i]$$

$$x_i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \end{bmatrix} \rightarrow [i, 1]$$

$$x_i^T = [x_1 \ x_2 \ldots x_i] \rightarrow [1, i]$$

$$\delta_j = \begin{bmatrix} f'(z_1) \Sigma_k \delta_k w_{kj} \\ f'(z_2) \Sigma_k \delta_k w_{kj} \\ \vdots \\ f'(z_j) \Sigma_k \delta_k w_{kj} \end{bmatrix} \rightarrow [j, 1]$$

δ_k defined earlier.

$$w_{kj} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ w_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ w_{k1} & \cdots & \cdots & w_{kj} \end{bmatrix} \rightarrow [k, j]$$

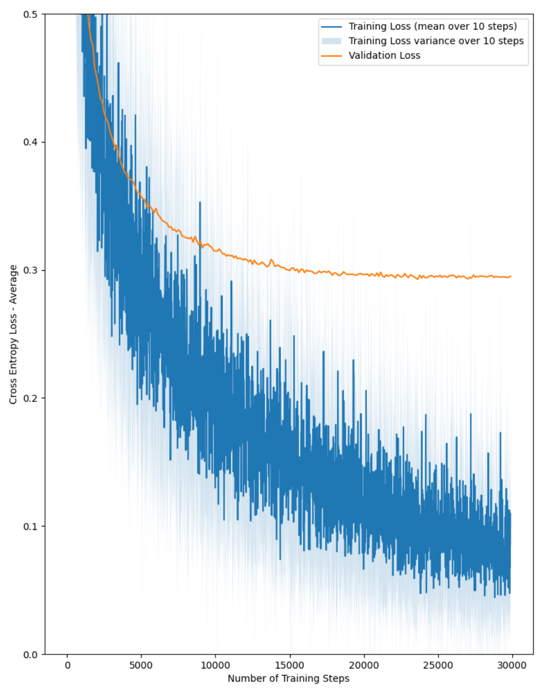
$$\Sigma_k \delta_k w_{kj} = \begin{bmatrix} \Sigma_k \delta_k w_{kj} \\ \Sigma_k \delta_k w_{kj} \\ \vdots \\ \Sigma_k \delta_k w_{kj} \end{bmatrix} \rightarrow [j, 1]$$

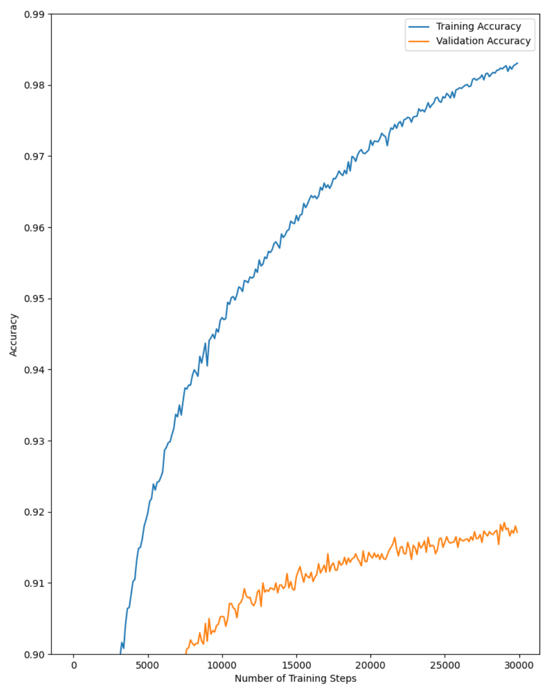
$$f'(z_j) = \begin{bmatrix} f'(z_1) \\ f'(z_2) \\ \vdots \\ f'(z_j \end{bmatrix} \rightarrow [j, 1]$$

Task 2

The mean is 33.553 and standard deviation is 78.876 (5.s.f).

Task 2c)





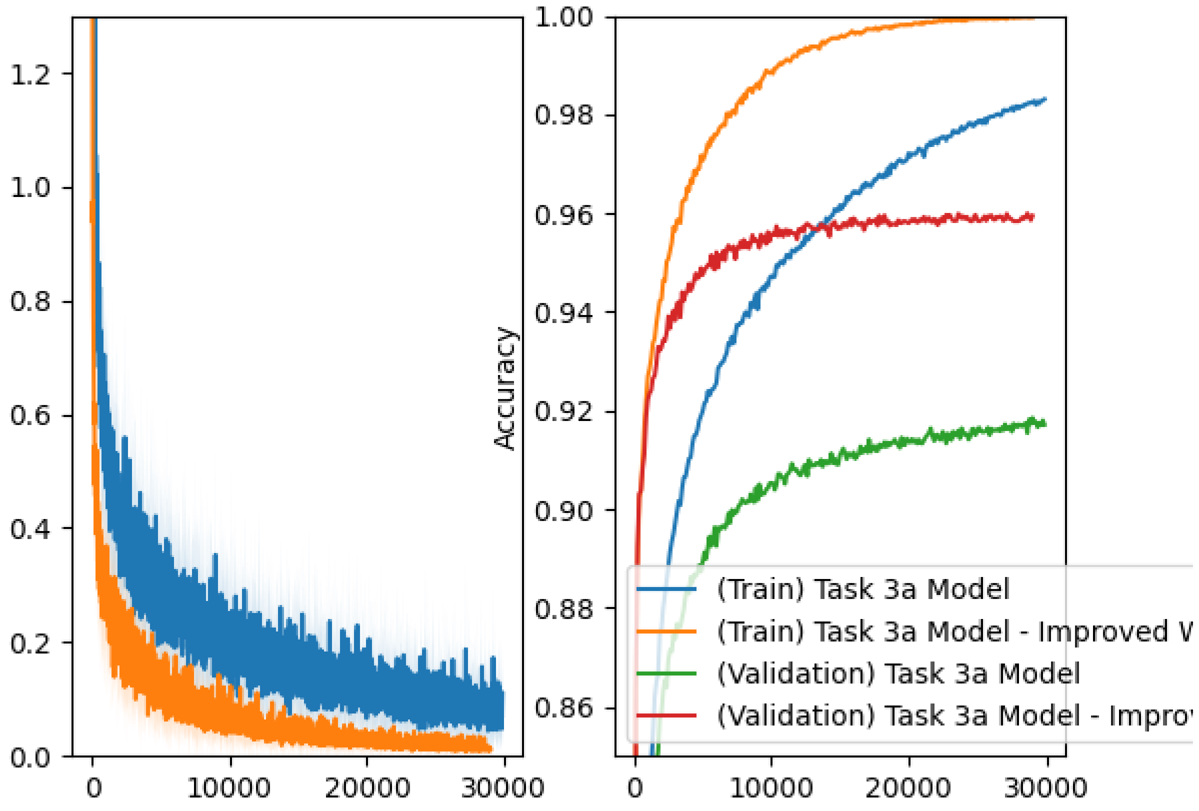
Task 2d)

As we have 1 input layer with 785 nodes, 1 hidden layer with 64 nodes and 1 output layer with 10 nodes, we can calulate the number of parameters as follows.

Number of parameters = 785 * 64 + 64 * 10 = 50,880.

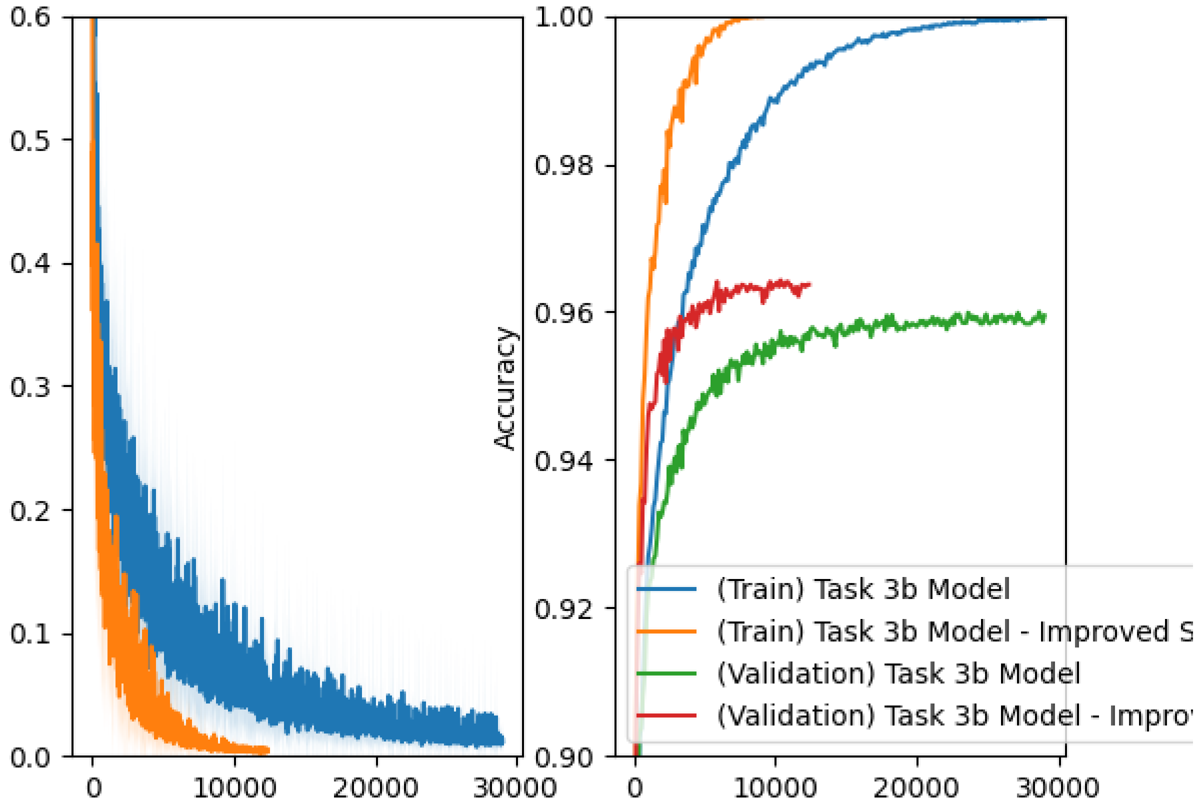
Task 3

Task 3a)



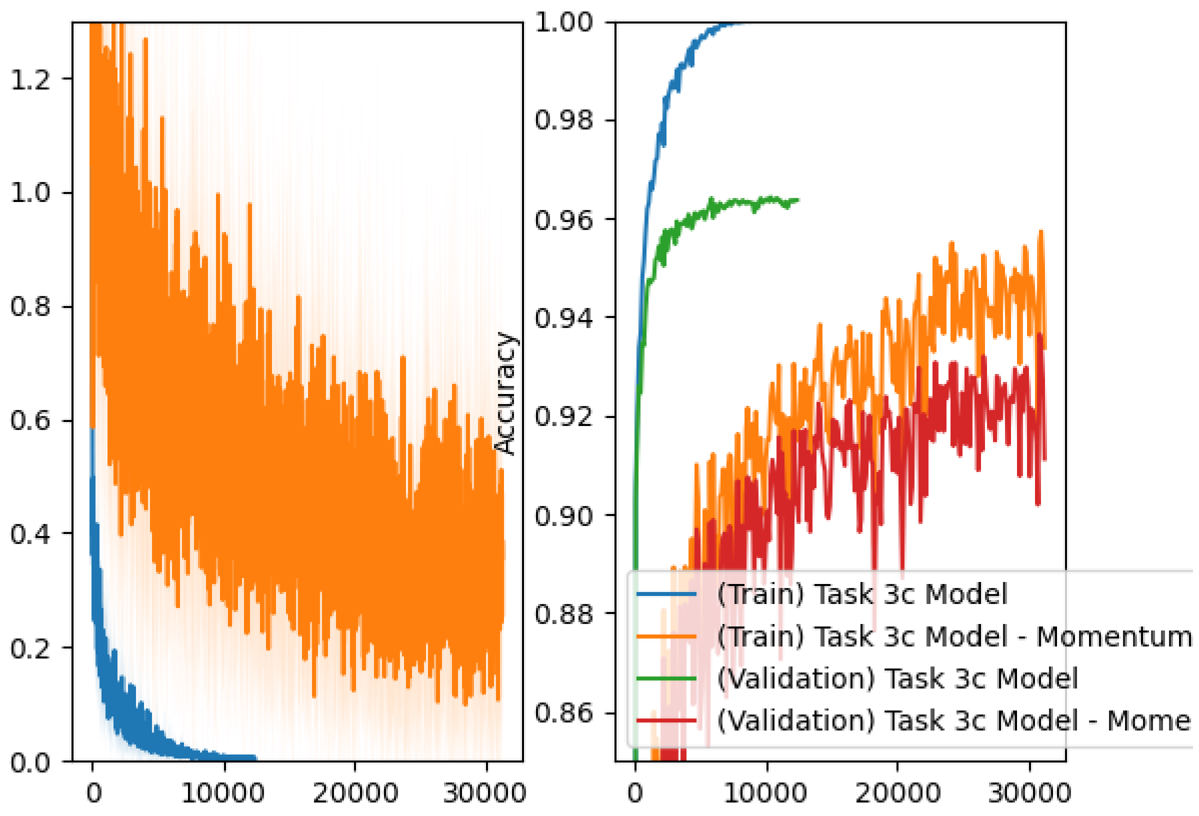
With the improved weight initialization, we observe that the model actually performs better and converges at a higher validation accuracy. We can also observe that the model with the improved weight initialization also ended training slightly earlier at 46 epoch instead of 47 for the one without improved weight initialization. The deviation between train-validation accuracy is significantly lower with an improved weight initialization, indicating that overfitting is reduced quite significantly.

Task 3b)



With the improved sigmoid, it seems that there is a slight improvement in the validation accuracy. This also indicates a fall in the deviation between the train and validation set. Hence there is improvement in generalization and fall in overfitting of the model with an improved sigmoid. The model also converges significantly faster at 19 epochs compared to the 46 epochs without the improvement.

Task 3c)

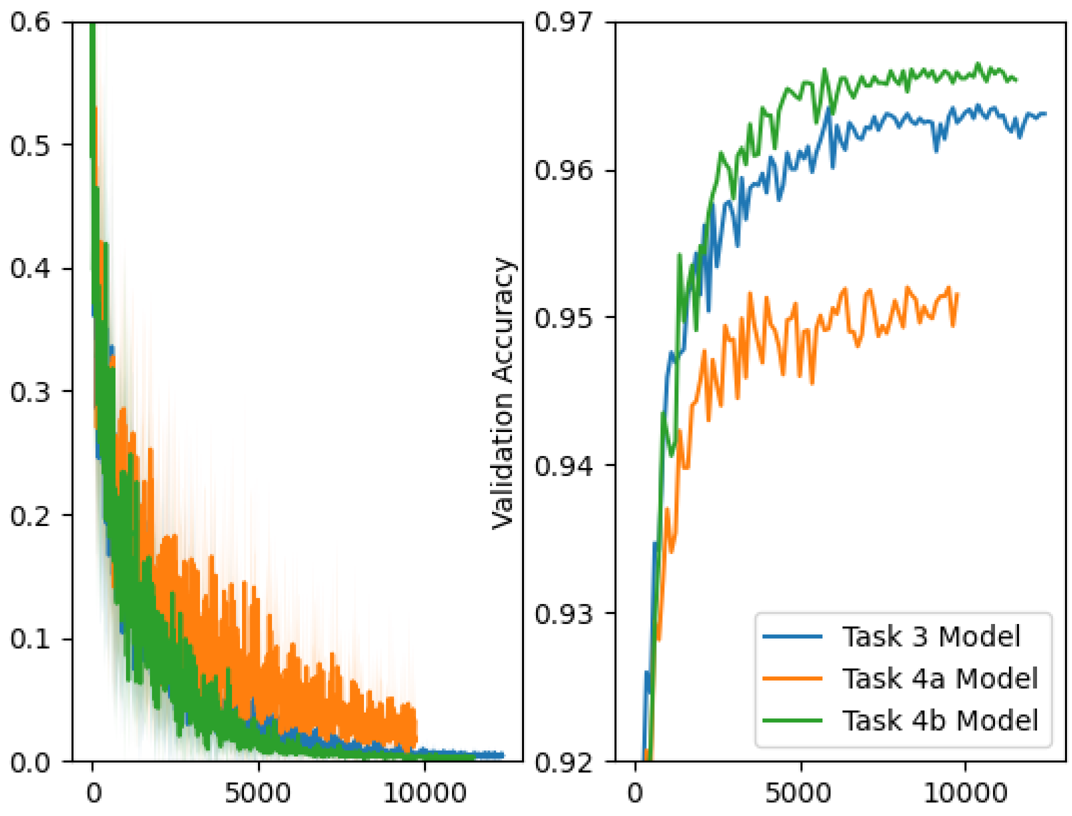


With momentum, we observe that performance of the model actually drops. The validation accuracy of the model with momentum drops by about 4%, causing the model to be worse at generalization. The model converges much slower, taking the full 50 epoch without stabilizing as compared to 19 epoch without momentum. Overfitting however, seems to be lower with momentum as the deviation between train-validation accuracy falls.

Task 4

As the model with momentum performs worse than the one without, for this task, we will be comparing all of them against the model without momentum but with improved sigmoid and improved weight initialization.

Plot for Task 4a and Task 4b



Task 4a)

I observe that although the model with 32 nodes stops training at 15 epochs compared to 19 epochs, the validation accuracy is significantly lower than the one with 64 nodes.

If there are too little nodes, the model may not be able to generalize sufficiently to the data.

Task 4b)

I observe that with 128 nodes, the model stops training earlier than the baseline which stops at 19 epochs. Furthermore, the performance of the model is higher than the baseline.

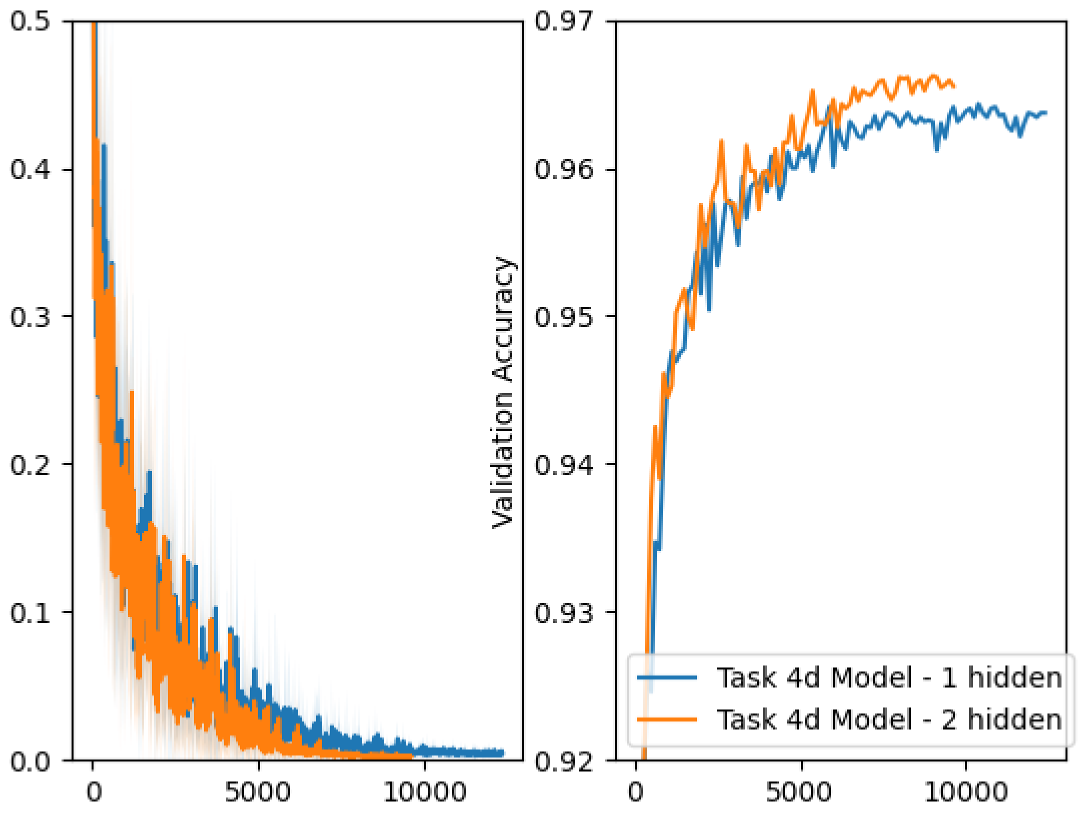
With more nodes, we could expect higher performance with diminishing returns. In turn, although the number of epochs needed to converge falls, each epoch might take a longer time and hence an overall longer training time.

Task 4d)

By trying out different number of nodes, I found that I could have chosen 59 or 60 nodes per layer to obtain a network with similar number of parameters as a [64,10] configuration.

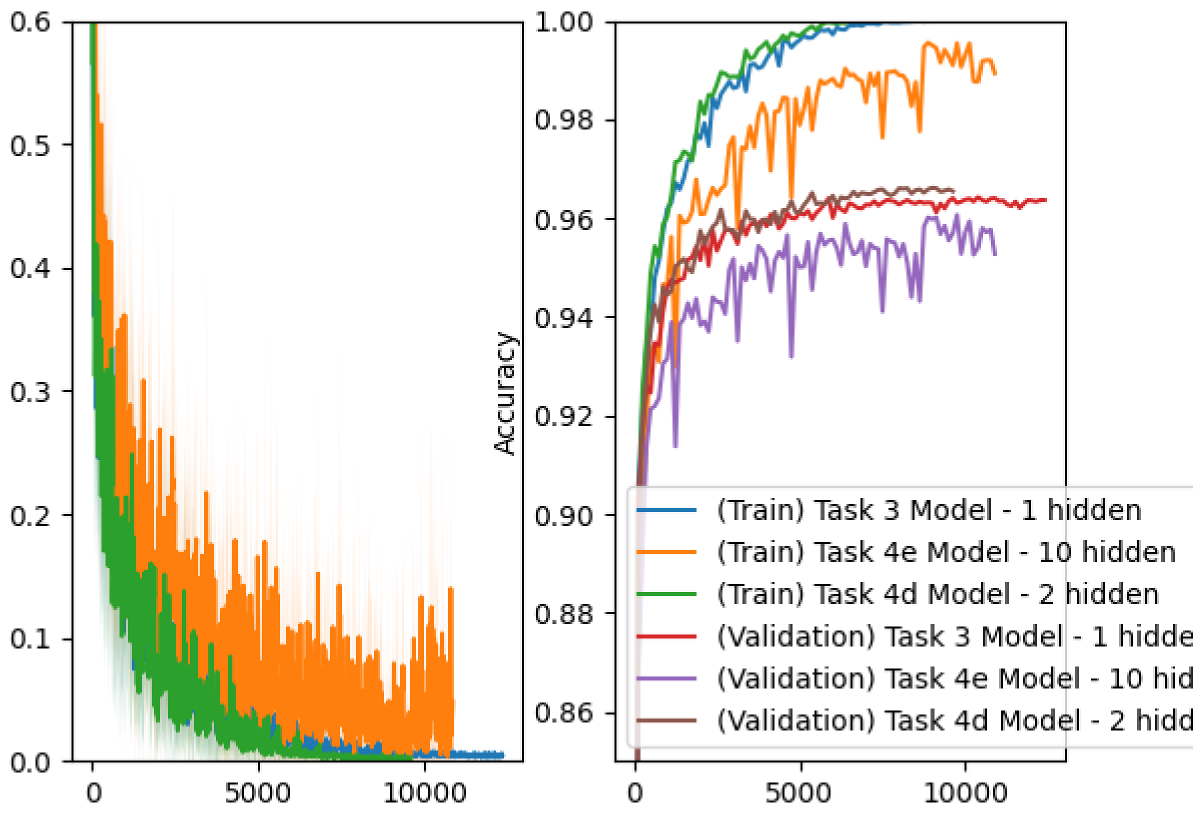
I chose to use 59 hidden units per hidden layer setup, which would give me the following number of paramaters.

Number of parameters: 785 * 59 + 59 * 59 + 59 * 10 = 50,386. Which is similar to the network from task 3 with 50,880 parameters.



The model converges faster at 15 epoch as compared to 19 epoch of the baseline. The network also seems to perform slightly better than having only 1 hidden layer, despite having the same number of parameters. This may be due to the network being able to model more complex relationship due to a second hidden layer.

Task 4e)



The model converges faster at 17 epoch as compared to the 1 hidden layer model. However, both train and validation accuracy of the 10 hidden layer model is significantly lower than both models with 1 and 2 hidden layers.

Initially, my thought is that the model with 10 hidden layers (model X), overfitted due to the insanely high complexity of the model, hence it basically memorized the answer. However, the training accuracy is still lower than that of the the model with 1 hidden layer (model Y) and the model with 2 hidden layers (model Z). After some thought, this might still be true due to dataset shuffling. Model X in this case memorized each batch through backpropagation to a very high degree. To the point that the model may not have been looking at general shapes of the number, but instead the actual pixel itself. This could possibly explain the high variance in loss and accuracy for model X due to the distribution of images across each batch.