TDT4265 - Computer Vision and Deep Learning
Assignment 2

Jakob Vahlin
Kristian Stensgård

Contents

1 Softmax regression with backpropagation

1.1 Task 1a: Backpropagation

i)

In the neural network considered in this assignment the gradient descent update rule for weights w_{ji} of the **hidden** layer is given by

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}}, \quad \alpha \in \mathbb{R} \tag{1}$$

where $C(w)$ is the cross entropy loss, defined by

$$C(w) = \frac{1}{N} \sum_{n=1}^{N} C^n(w), \quad C^n(w) = -\sum_{k=1}^{K} y_k^n \ln(\hat{y}_k^n) \tag{2}$$

where $\hat{y}$ is the Softmax activation function for the *output* layer, defined by

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}}, \quad \text{where } z_k = w_k^T \cdot a = \sum_{j}^{J} w_{kj} \cdot a_j \tag{3}$$

where w_{kj} are weights in the output layer and $a_j = f(z_j)$ is the is the Sigmoid activation function in the *hidden* layer, defined as

$$a_j = f(z_j) = \frac{1}{1 + e^{-z_j}}, \quad z_j = w_j^T x = \sum_{i}^{I} w_{ji} \cdot x_i \tag{4}$$

By algebraic manipulation it can be shown that (1) can be written as

$$w_{ji} := w_{ji} - \alpha \delta_j x_i \tag{5}$$

by defining $\delta_j = \frac{\partial C}{\partial z_j}$.

To derive the equvialent expression for the gradient descent update rule given by (5) consider the partial derivative of the cross entropy loss function w.r.t the variable w_{ji}. From (2) and (3) it is clear that $C(w)$ has a dependence on z_k. Furthermore, from (3), it is clear that z_k in turn has a dependence on a_j, which in turn has a dependence on z_j, as evident from (4). Finally, from (4) it is clear that z_j has a dependence on w_{ji}. Consequently, the cost functions dependence on the variable w_{ji} is "chained together" through the variables z_k,a_j and z_j. By defining $\boldsymbol{z_k} = [z_1, z_2, \ldots, z_K]$ as the vector containing z values from the output layer, $\boldsymbol{a} = [a_1, a_2, \ldots, a_J]$ as the vector with a values from the hidden layer and $\boldsymbol{z_j} = [z_1, z_2, \ldots, z_J]$ as the vector containing z values from the hidden layer, we can express the cross entropy loss function in terms of its variables in the following way

$$\begin{aligned} C(w) &= C(\boldsymbol{z_k}) \\ &= C\big(\boldsymbol{z_k}(\boldsymbol{a})\big) \\ &= C\Big(\boldsymbol{z_k}\big(\boldsymbol{a}(\boldsymbol{z_j})\big)\Big) \\ &= C\Big(\boldsymbol{z_k}\big(\boldsymbol{a}(\boldsymbol{z_j}(w))\big)\Big) \end{aligned} \tag{6}$$

with K representing the number of nodes in the output layer, and J representing the number of nodes in the hidden layer. In this case, the lowercase w represents the weight matrix from the input layer to the hidden layer, in accordance with the notation in the assignment description. Therefore, the derivative of the cross entropy loss function, w.r.t

to any weight in the *hidden* layer w_{ji}, $\frac{\partial C}{\partial w_{ji}}$, can be expressed by applying the multivariate chain rule.

In the following, to distinguish between z vector from the different layers in the network, entries from the z vector from the hidden layer will be labeled z^j and entries from the z vector from the output layer will be labeled z^k. This is needed to distinguish between i.e z_1 from the hidden layer and z_1 from the output layer.

$$\begin{aligned}\frac{\partial C}{\partial w_{ji}} = &\frac{\partial C}{\partial z_1^k}\left(\frac{\partial z_1^k}{\partial a_1}\frac{\partial a_1}{\partial z_1^j}\frac{\partial z_1^j}{\partial w_{1i}} + \frac{\partial z_1^k}{\partial a_2}\frac{\partial a_2}{\partial z_2^j}\frac{\partial z_2^j}{\partial w_{2i}} + \cdots + \frac{\partial z_1^k}{\partial a_J}\frac{\partial a_J}{\partial z_J^j}\frac{\partial z_J^j}{\partial w_{Ji}}\right) \\ &+ \frac{\partial C}{\partial z_2^k}\left(\frac{\partial z_2^k}{\partial a_1}\frac{\partial a_1}{\partial z_1^j}\frac{\partial z_1^j}{\partial w_{1i}} + \frac{\partial z_2^k}{\partial a_2}\frac{\partial a_2}{\partial z_2^j}\frac{\partial z_2^j}{\partial w_{2i}} + \cdots + \frac{\partial z_2^k}{\partial a_J}\frac{\partial a_J}{\partial z_J^j}\frac{\partial z_J^j}{\partial w_{Ji}}\right) \\ &+ \cdots + \frac{\partial C}{\partial z_K^k}\left(\frac{\partial z_K^k}{\partial a_1}\frac{\partial a_1}{\partial z_1^j}\frac{\partial z_1^j}{\partial w_{1i}} + \frac{\partial z_1^k}{\partial a_2}\frac{\partial a_2}{\partial z_2^j}\frac{\partial z_2^j}{\partial w_{2i}} + \cdots + \frac{\partial z_1^k}{\partial a_J}\frac{\partial a_J}{\partial z_J^j}\frac{\partial z_J^j}{\partial w_{Ji}}\right)\end{aligned} \tag{7}$$

Equation (7) can be simplified to a set of sums (*Notation*: Note that as there no longer exist a need to distinguish between z vectors from the different layers in the network, as this will now be clear from the variable (j or k) used to index the vectors in accordance with the notation in the assignment description).

$$\frac{\partial C}{\partial w_{ji}} = \sum_{k=1}^{K}\frac{\partial C}{\partial z_k}\left(\sum_{j'=1}^{J}\frac{\partial z_k}{\partial a_{j'}}\frac{\partial a_{j'}}{\partial z_{j'}}\frac{\partial z_{j'}}{\partial w_{ji}}\right) \tag{8}$$

To further simplify (8), consider the last partial derviative, $\frac{\partial z_{j'}}{w_{ji}}$. From the definition of z_j given in (4) the derivative w.r.t to w_{ji} can be expressed as

$$\frac{\partial z_{j'}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\sum_{i'=1}^{I} w_{j'i'}x_{i'}, \quad \text{for a given } j' \tag{9}$$

Similar to when deriving the gradients of the cost function in Assignment 1, the resulting differentiation of the sums in (9) w.r.t to a given given weight, $w_{j'i'}$ will only be non-zero for $j' = j$ and $i' = i$. When this is the case the differentiation of $z_{j'} = z_j$ becomes

$$\begin{aligned}\frac{\partial z_j}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}}\sum_{i'=1}^{I} w_{ji'}x_{i'} \\ &= \frac{\partial}{\partial w_{ji}} w_{ji}x_i \\ &= x_i\end{aligned} \tag{10}$$

Thus it is clear that $\frac{\partial z_{j'}}{\partial w_{ji}} = x_i$, for $j' = j$ and thus it is independent of the j' and k indecies. Consequently it can be moved outside of the sums that goes over all j' and over all k.

$$\begin{aligned}\frac{\partial C}{\partial w_{ji}} &= \sum_{k=1}^{K}\frac{\partial C}{\partial z_k}\left(\sum_{j'=1}^{J}\frac{\partial z_k}{\partial a_{j'}}\frac{\partial a_{j'}}{\partial z_{j'}}\frac{\partial z_{j'}}{\partial w_{ji}}\right) \\ &= \sum_{k=1}^{K}\frac{\partial C}{\partial z_k}\left(\sum_{j'=1}^{J}\frac{\partial z_k}{\partial a_{j'}}\frac{\partial a_{j'}}{\partial z_{j'}}x_i\right) \\ &= \left(\sum_{k=1}^{K}\frac{\partial C}{\partial z_k}\left(\sum_{j'=1}^{J}\frac{\partial z_k}{\partial a_{j'}}\frac{\partial a_{j'}}{\partial z_{j'}}\right)\right)x_i\end{aligned} \tag{11}$$

Furthermore, consider the definition of δ_j, given by

$$\delta_j = \frac{\partial C}{\partial z_j} \tag{12}$$

From the chain of dependencies given in (6), the derivative in (12) can be expressed by applying the multivariate chain rule

$$\delta_j = \frac{\partial C}{\partial z_j} = \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \left(\sum_{j=1}^{J} \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \right) \tag{13}$$

By inserting the result in (13) into (11), the derivative of the cost function w.r.t w_{ji} becomes

$$\begin{aligned} \frac{\partial C}{\partial w_{ji}} &= \left(\sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \left(\sum_{j=1}^{J} \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \right) \right) x_i \\ &= \frac{\partial C}{\partial z_j} x_i \\ &= \delta_j x_i \end{aligned} \tag{14}$$

Finally, inserting the result in (14) into (1), the desired result given by (5) is found. □

ii)

With δ_j defined as

$$\delta_j = \frac{\partial C}{\partial z_j} \tag{15}$$

where z_j is defined as in (3), it can be shown that δ_j can be expressed as

$$\delta_j = f'(z_j) \sum_{k=1}^{K} w_{kj} \delta_k \tag{16}$$

where $\delta_k = \frac{\partial C}{\partial z_k} = -(y_k - \hat{y}_k)$ and $f'(z_j) = \frac{\partial f}{\partial z_j} = \frac{\partial a_j}{\partial z_j}$ is the slope of the Sigmoid activation function for the hidden layer, given by (3).

To prove the relationship given by (16), recall from part *i)* and (13) the definition of δ_j with the derivative written out according to the chain rule

$$\delta_j = \frac{\partial C}{\partial z_j} = \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \left(\sum_{j'=1}^{J} \frac{\partial z_k}{\partial a_{j'}} \frac{\partial a_{j'}}{\partial z_j} \right) \tag{17}$$

First, consider the term $\frac{\partial z_k}{a_{j'}}$ which, from the definition of z_k, can be expressed as

$$\frac{\partial z_k}{\partial a_{j'}} = \frac{\partial}{\partial a_{j'}} \sum_{j''=1}^{J} w_{kj''} \cdot a_{j''}, \quad \text{for a given } k \tag{18}$$

Again, the expression in (18) is only non-zero for $j'' = j'$, in which the resulting expression becomes

$$\begin{aligned}\frac{\partial z_k}{\partial a_{j'}} &= \frac{\partial}{\partial a_{j'}} \sum_{j''=1}^{J} w_{kj''} \cdot a_{j''} \\ &= \frac{\partial}{\partial a_{j'}} w_{kj'} \cdot a_{j'} \\ &= w_{kj'}\end{aligned} \tag{19}$$

Inserting the result in (19) into (17) gives

$$\begin{aligned}\delta_j = \frac{\partial C}{\partial z_j} &= \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \Big(\sum_{j'=1}^{J} \frac{\partial z_k}{\partial a_{j'}} \frac{\partial a_{j'}}{\partial z_j} \Big) \\ &= \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \Big(\sum_{j'=1}^{J} w_{kj'} \frac{\partial a_{j'}}{\partial z_j} \Big)\end{aligned} \tag{20}$$

Next, consider the expression $\sum_{j'=1}^{J} w_{kj'} \frac{\partial a_{j'}}{\partial z_j}$. Differentiating $a_{j'}$ w.r.t the variable z_j will only be non-zero when $j' = j$. When this is the case, the resulting derivative becomes

$$\begin{aligned}\sum_{j'=1}^{J} w_{kj'} \frac{\partial a_{j'}}{\partial z_j} &= w_{kj} \frac{\partial a_j}{\partial z_j} \\ &= w_{kj} f'(z_j)\end{aligned} \tag{21}$$

since $a_j = f(z_j)$. Inserting the result in (21) into (20) gives

$$\begin{aligned}\delta_j &= \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \Big(\sum_{j'=1}^{J} w_{kj'} \frac{\partial a_{j'}}{\partial z_j} \Big) \\ &= \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \Big(w_{kj} f'(z_j) \Big)\end{aligned} \tag{22}$$

Observe that $f'(z_j)$ is independent of k and consequently can be moved outside the sum over all k. Finally, by the definition $\delta_k = \frac{\partial C}{\partial z_k}$ the desired expression for δ_j is achieved

$$\delta_j = f'(z_j) \sum_{k=1}^{K} \delta_k w_{kj} \tag{23}$$

□

1.2 Task 1b: Vectorize computation

Notation: To represent matrices, a bold uppercase letter, i.e $\mathbf{W}$, will be used. To represent vectors, a bold and italic lowercase letter, i.e $\boldsymbol{w}$, will be used. To represent a scalar, an italic lowercase letter, i.e w, will be used.

Furthermore, the number of nodes in the input layer, will be labeled I. The number of nodes in the hidden layer will be labeled J. The number of nodes in the output layer will be labeled K.

i) The input layer to hidden layer

Using matrix notation, the update rule for the weight matrix from the input layer to hidden layer, $\mathbf{W_1}$, becomes

$$\mathbf{W_1} := \mathbf{W_1} - \alpha \boldsymbol{\delta_J} \boldsymbol{x}^T, \quad \alpha \in \mathbb{R} \tag{24}$$

The contents and dimensions of the matrices and vectors in (24) are as follows:

For the $\mathbf{W_1}$ matrix:

$$\mathbf{W_1} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1I} \\ w_{21} & w_{22} & \dots & w_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ w_{J1} & w_{J2} & \dots & w_{JI} \end{bmatrix}, \quad \mathbf{W_1} \in \mathcal{M}_{J\times I}(\mathbb{R}) \tag{25}$$

where the matrix entries w_{ji} are updated according to (5). In a network with an input layer of $I = 785$ units and one hidden layer with $J = 64$ units, the dimension of $\mathbf{W_1}$ becomes $[64 \times 785]$.

For the $\boldsymbol{\delta_J}$ vector:

$$\boldsymbol{\delta_J} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_J \end{bmatrix} = \begin{bmatrix} f'(z_1) \sum_{k=1}^{K} w_{k1}\delta_k \\ f'(z_2) \sum_{k=1}^{K} w_{k2}\delta_k \\ \vdots \\ f'(z_J) \sum_{k=1}^{K} w_{kJ}\delta_k \end{bmatrix}, \quad \boldsymbol{\delta_J} \in \mathbb{R}^J \tag{26}$$

where $f'(z_j)$ is the derivative of the activation function and δ_k is the error vector found in 31. Using matrix notation the expression in (26) becomes:

$$\boldsymbol{\delta_J} = f'(\boldsymbol{z_J}) \odot \mathbf{W_2}^T \boldsymbol{\delta_K}, \quad \boldsymbol{\delta_J} \in \mathbb{R}^J \tag{27}$$

In a network with $J = 64$ hidden units, the dimension of the $\boldsymbol{\delta_J}$ vector is $[64\times1]$

For the $\boldsymbol{x}$ vector:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_I \end{bmatrix}, \quad \boldsymbol{x} \in \mathbb{R}^I \tag{28}$$

where the x_i values are the pixel values in the image and one bias. In a network with $I = 784 + 1 = 785$ input unist, the dimension of the $\boldsymbol{x}$ vector is $[785 \times 1]$

Note that $\mathbf{A} = \boldsymbol{\delta_J} \cdot \boldsymbol{x}^T$ will yield a matrix $\mathbf{A} \in \mathcal{M}_{J\times I}(\mathbb{R})$ due to the fact that $\boldsymbol{x}$ vector is transposed in the vector product. Thus the matrix subtraction $\mathbf{W_1} - \mathbf{A}$ is valid, as both matrices are have dimension $[J \times I]$.

ii) The hidden layer to output layer

Using matrix notation, the update rule for the weight matrix from the hidden layer to output layer, $\mathbf{W_2}$, becomes

$$\mathbf{W_2} := \mathbf{W_2} - \alpha \boldsymbol{\delta_K} \boldsymbol{a}^T, \quad \alpha \in \mathbb{R} \tag{29}$$

The contents and dimensions of the matrices and vectors in (29) are as follows:

For the $\mathbf{W_2}$ matrix:

$$\mathbf{W_2} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1I} \\ w_{21} & w_{22} & \dots & w_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ w_{J1} & w_{J2} & \dots & w_{JI} \end{bmatrix}, \quad \mathbf{W_2} \in \mathcal{M}_{K\times J}(\mathbb{R}) \tag{30}$$

where the matrix entries w_{ji} are updated according to $w_{kj} := w_{kj} - \alpha\delta_k a_j$.

In a network layer with $J = 64$ hidden units and $K = 10$ output units, the dimension of $\mathbf{W_2}$ becomes $[10 \times 64]$.

For the $\boldsymbol{\delta_K}$ vector:

$$\boldsymbol{\delta_K} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_K \end{bmatrix} = \begin{bmatrix} -(y_1 - \hat{y}_1) \\ -(y_2 - \hat{y}_2) \\ \vdots \\ -(y_K - \hat{y}_K) \end{bmatrix}, \quad \boldsymbol{\delta_K} \in \mathbb{R}^K \tag{31}$$

where $\delta_k = \frac{\partial C}{\partial z_k} = -(y_k - \hat{y}_k)$ is the result found in Assignment 1. In a network with $K = 10$ output units, the dimension of the $\boldsymbol{\delta_K}$ vector becomes $[10 \times 1]$.

For the $\boldsymbol{a}$ vector:

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_J \end{bmatrix} = \begin{bmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_J) \end{bmatrix}, \quad \boldsymbol{a} \in \mathbb{R}^J \tag{32}$$

where $f(z_j)$ is the sigmoid activation in the hidden layer, as defined in (3). In a network with $J = 64$ hidden units, the dimension of the $\boldsymbol{a}$ vector becomes $[64 \times 1]$

Note that $\mathbf{A} = \boldsymbol{\delta_K} \cdot \boldsymbol{a}^T$ will yield a matrix $\mathbf{A} \in \mathcal{M}_{K \times J}(\mathbb{R})$ due to the fact that $\boldsymbol{a}$ vector is transposed in the vector product. Thus the matrix addition $\mathbf{W_2} - \mathbf{A}$ is valid, as both matrices are have dimension $[K \times J]$.

Note in matrix dimensionality when implementing network: In the code used to implement these equations, a few changes has been made as opposed to the notation used in the assignment text and in the derivation of the backpropagation. Firstly the weight matrices $\mathbf{W_1}$ and $\mathbf{W_2}$ are transposed, i.e the weight at w_{ji} is placed at w_{ij} and the weight at w_{kj} is placed at w_{jk}. Secondly we have implemented mini-batch gradient descent. To implement this we have a batch of input vectors, $\boldsymbol{x}$, as rows in a matrix where the number of rows is the number of samples in the batch. In our case the number of samples per batch is 32. This results in the following equations for $\boldsymbol{\delta_K}$ and $\boldsymbol{\delta_J}$ that are implemented in our code.

$$\boldsymbol{\delta_K} = -(\boldsymbol{y_K} - \hat{\boldsymbol{y}}_{\boldsymbol{K}}) \tag{33}$$

$$\boldsymbol{\delta_J} = f'(\boldsymbol{z_J}) \odot \boldsymbol{\delta_K} \mathbf{W_2^T} \tag{34}$$

The entries in the gradients becomes as shown in Equation 35 and Equation 36, where $\mathbf{X}$ is the batch of input vectors and $\mathbf{A}$ is the batch output from the hidden layer.

$$\frac{\partial C}{\partial w_{ij}} = \frac{\mathbf{X}^T \cdot \boldsymbol{\delta_j}}{\text{batch size}} \tag{35}$$

$$\frac{\partial C}{\partial w_{jk}} = \frac{\mathbf{A}^T \cdot \boldsymbol{\delta_j}}{\text{batch size}} \tag{36}$$

2 Task 2: Softmax Regression with Backpropagation

2.1 Task 2a: Mean and standard deviation of training set

Before the pixel values of the image are fed into the network they are normalized in the following way

$$X_{norm} = \frac{X - \mu}{\sigma} \tag{37}$$

where μ and σ are the mean and standard deviation of the *training set*. The values for mean and standard deviation of the training dataset is shown in Table 1.

Table 1: Characteristics of the training dataset

Distribution parameter	Value
Mean	33.55
Standard deviation	78.88

2.2 Task 2c: Softmax regression with mini-batch gradient descent

A multi layer nerual network using Softmax regression with mini-batch gradient descent has been implemented. The network consists of a single hidden layer with 64 hidden units and an output layer with 10 output units. Before training the weights have been initialized to values uniformly distributed in the interval $[-1, 1]$.

Plots of the resulting loss and accuracy when training the network is shown in Figure 1 and Figure 2 respectively. As evident from the plots, the training loss and accuracy greatly outperforms the validation loss and accuracy.

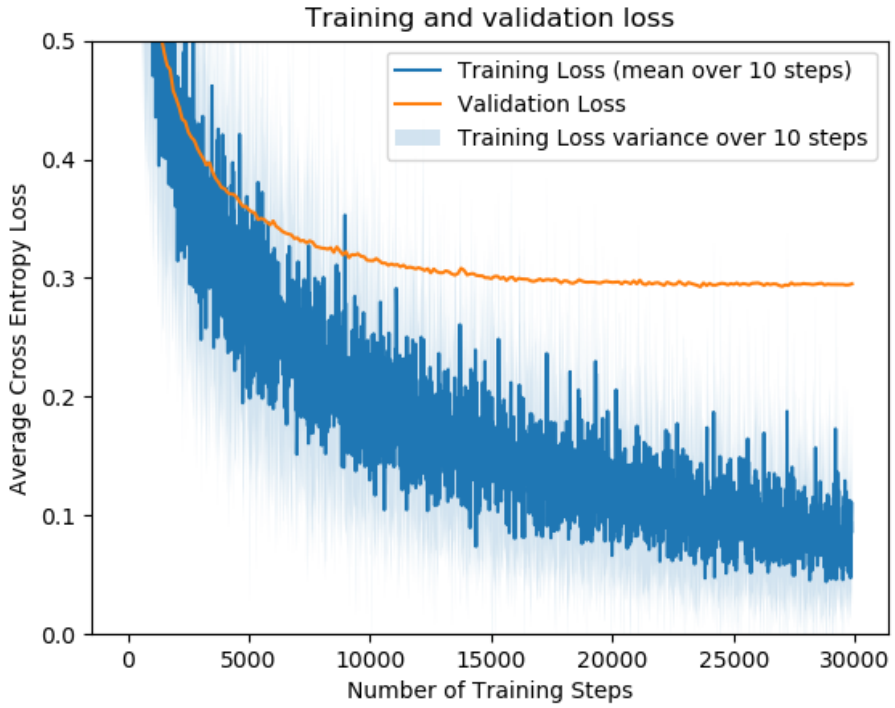

Figure 1: Training and validation loss of a neural network with two layers.

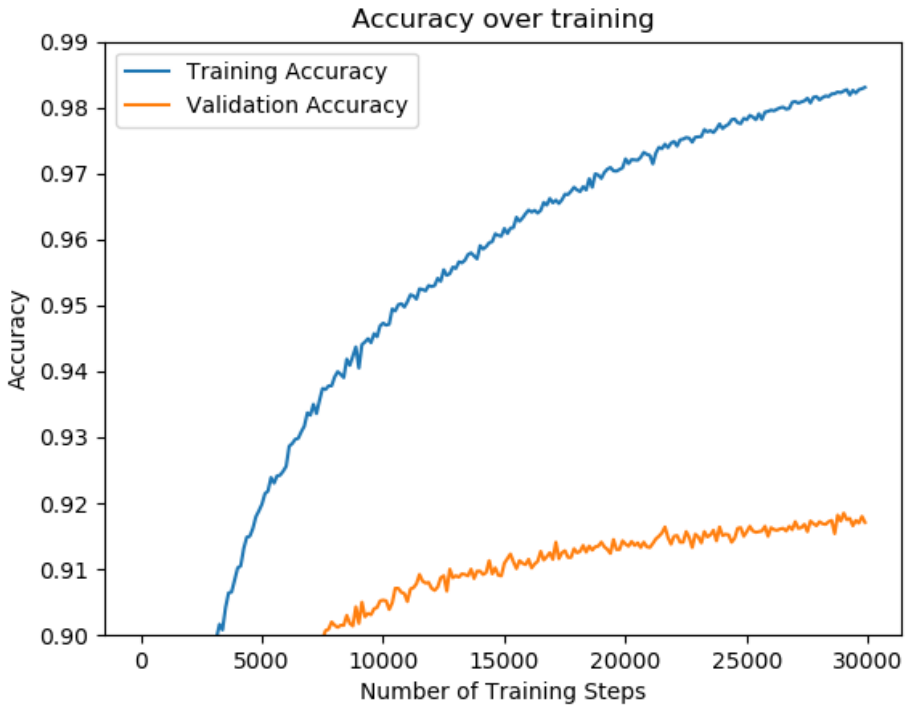

Figure 2: Training accuracy of a neural network with two layers.

2.3 Task 2d: Number of parameters

The number of parameters in the network is defined as the number of weights plus the number of biases. In this case the network has two layers. 64 nodes in the hidden layer and 10 nodes in the output layer. The input vector $\boldsymbol{x}$ consists of 785 elements, 784 data points and 1 bias. The weights include biases, because the bias trick has been implemented. The resulting number of parameters P is thus:

$$P = 785 \cdot 64 + 64 \cdot 10 = 50,880 \tag{38}$$

3 Adding the "Tricks of the Trade"

In order to improve the performance of the network a few "tricks" have been added to the model. To clearly document the effect of a given "trick", performance variations will be discussed with each addition. A complete overview of the final metrics in the network for each addition is outlined in Table 2.

3.1 Task 3a: Improved weight initialization

The first addition to the model that will be discussed is improved initialization of the input weights. The initial weight matrix, $\mathbf{W}_{t=0}$, is drawn from a normal distribution, $\mathbf{W}_{t=0} \sim \mathcal{N}(\mu, \sigma)$, with mean $\mu = 0$ and standard deviation $\sigma = 1/\sqrt{\text{fan-in}}$. The number "fan-in" is the number of inputs to the given neuron, i.e for the hidden layer fan-in = 785 and for the output layer fan-in = 64.

The loss and accuracy on the training and validation data sets for the network with and without improved weight initialization is shown in Figure 3 and Figure 4.

From both plots it is clear that introducing the improved Sigmoid lowers both the final training loss and accuracy, aswell as the final validation loss and accuracy. Furthermore, by studying Figure 3 one can observe that the "gap" between the training and validation loss is larger without the improved weights, compared to the gap between the training

and validation loss *with* improved weights. Table 2 confirms this gap. This suggests that the model has been more generalized, and is consquently less overfitted after adding the improved weights. Finally one can also observe that the learning speed and convergence rate is improved since there is increased performance compared to the network without additions, with the same number of training steps.

Figure 3: Training and validation loss with and without improved weight initialization.

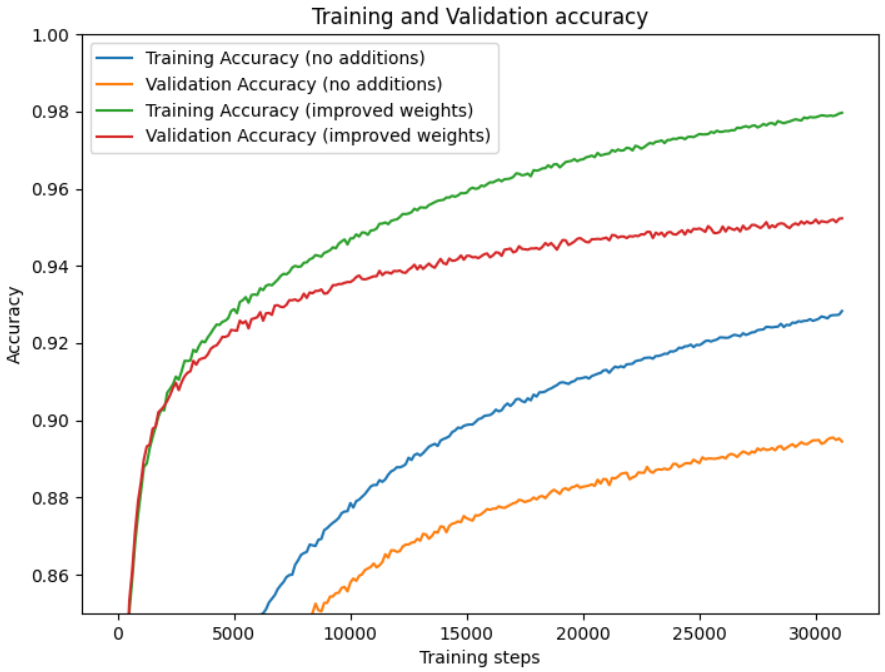

Figure 4: Training and validation accuracy with and without improved weight initialization.

3.2 Task 3b: Improved Sigmoid

The second addition to the model, is improving the Sigmoid activation function for the hidden layer. Previously, the Sigmoid function used was given by $f(x) = \frac{1}{1+e^{-x}}$. The issue with such an activation function, is that its outputs are not centered around zero and alway positive. As discussed in *LeCun 1998* [1], it is desirable to have outputs centered around zero, as this usually results in faster convergence. As a new activation function, the following hyperbolic function has been chosen:

$$f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right) \tag{39}$$

When performing the gradient descent, the derivative of the activation function is needed. Differentiating (39) gives:

$$f'(x) = 1.7159 \frac{2}{3\cosh^2\left(\frac{2}{3}x\right)} \tag{40}$$

The loss and accuracy on the training and validation data sets for the network with and without improved weight initialization and with and without improved weight intitialization and improved Sigmoid is shown in Figure 5 and Figure 6 respectively.

From the plots it is clear that introducing the improved Sigmoid in addition to the improved weights lowers both the final training and validation loss, and increases the training and validation accuracy even further. However, by studying Figure 5 it is clear that the validation loss with both improved weights and improved Sigmoid remains unchanged for around 10 000 to 25 000 timesteps before early stopping kicks in. This results in a large gap between the validation loss and training loss, indicating that the model begins to overfit on the training data. Again there is an improvement in learning speed and convergence rate. One can observe that the model with improved Sigmoid and improved weights requires less training steps to achieve the same performance compared to only using improved weights.

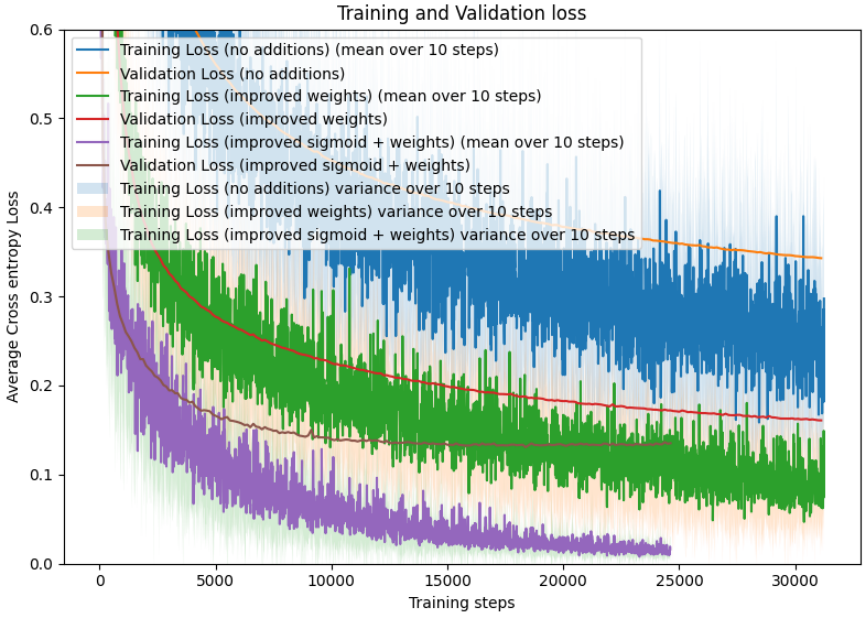

Figure 5: Training and validation loss with and without improved weight initialization and improved Sigmoid.

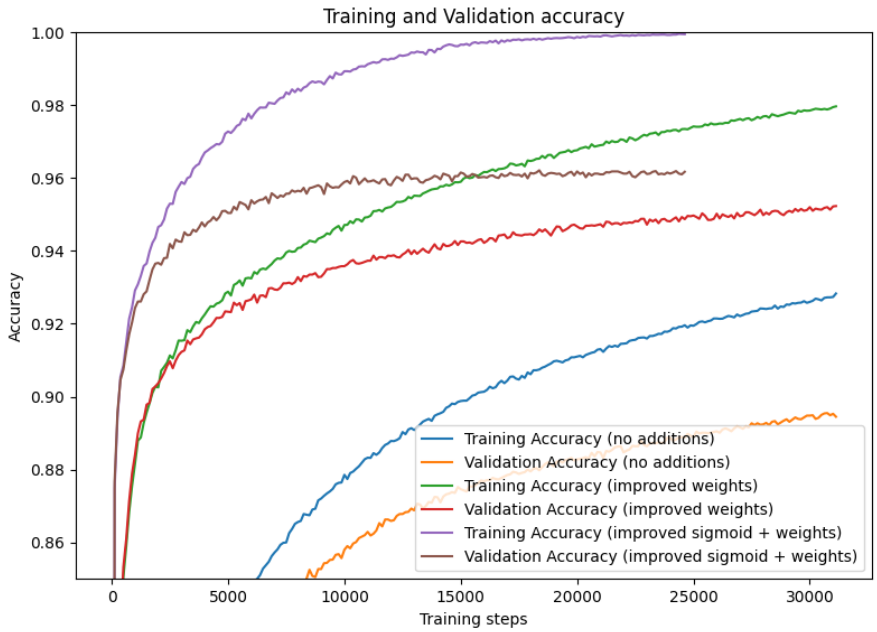

Figure 6: Training and validation accuracy with and without improved weight initialization and improved Sigmoid.

3.3 Task 3c: Momentum

The third and final addition to the model that will be discussed is momentum for the gradient descent. Momentum helps in utilizing a small portion of the previous gradients. This will make the gradient converge faster and may also prevent the network from getting stuck in a local minima as opposed to the desired global minima. The momentum is given by

$$\Delta w(t+1) = \frac{\partial C}{\partial w} + \gamma \Delta w(t) \tag{41}$$

Furthermore, the update step for a given weight, w_{ji}, with momentum added becomes

$$\Delta w_{ji} := w_{ji} - \alpha \cdot \Delta w(t) \tag{42}$$

The resulting training and validation loss for the network with two additions (weights and Sigmoid) and all tree additions(weights, Sigmoid and momentum) are shown in Figure 7. The resulting training and validation accuracy for the network with two additions (weights and sigmoid) and all tree additions(weights, Sigmoid and momentum) are shown in Figure 8. To avoid overfilled plots, the network configuration without any additions and the network configuration with only improved weights have been omitted.

From Figure 7 it is clear that the loss for the training set converges towards zero and the validation loss is starting to increase after about 5000 training steps. This is a clear indication of overfitting. The validation loss is even higher than for the network without the added momentum. The cause of this can be that the training set is too small and/or the training set is not generalized enough. The number of epochs required to train the network is the lowest among the four networks. This proves that the addition of momentum works as intended.

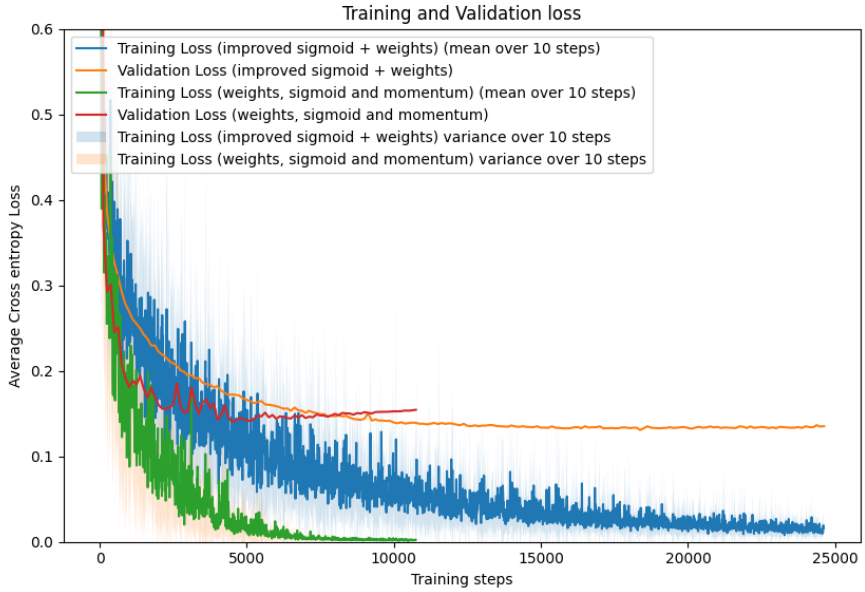

Figure 7: Training and validation loss with and without all three additions.

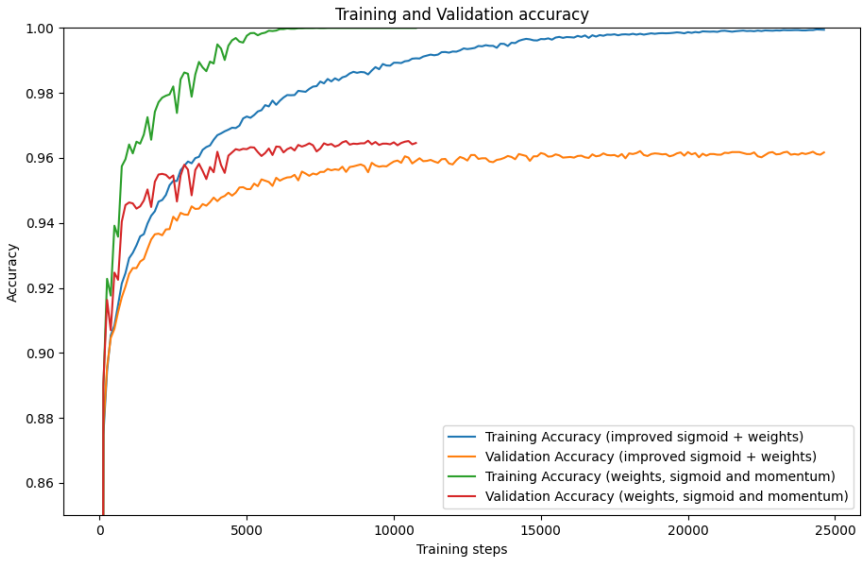

Figure 8: Training and validation accuracy with and without all three additions.

Table 2: Network metrics for different network configurations

Addition	**Validation Loss**	**Validation Accuarcy**	**Training Loss**	**Training Accuracy**	**Epochs**
No additions	0.3427	89.47%	0.2456	92.81%	50 of 50
Improved weights	0.1612	95.2%	0.0874	97.96%	50 of 50
Improved Sigmoid	0.1352	96.17%	0.0137	99.95%	39 of 50
Momentum	0.1544	96.46%	0.0018	100%	17 of 50

4 Experiment with network topology

4.1 Task 4a & 4b: Comparison of number of hidden nodes

To compare a network with a low number of nodes in the hidden layer to a network with a high number of nodes in the hidden layer, a network with $J_{low} = 32$ nodes in the hidden layer has been compared to a network with $J_{high} = 128$. As a baseline for the comparison, $J_{baseline} = 64$ nodes in the hidden layer has been used, as this is the size of the hidden layer used previously in this assignment.

The loss and accuracy for the training and validation sets for different networks with a varying number of hidden nodes is shown in Figure 9 and Figure 10. Final performance metrics are outlined in Table 3.

From Figure 9, Figure 10 and Table 3 it is evident that for a network with $J_{low} = 32$ units the performance, when compared to the baseline network of 64 hidden units. This is clear as neither the training loss nor the validation accuracy for the small hidden layer network reaches the levels of the baseline example. Additionally, the smaller network has a slower convergence speed (?). Thus it can be concluded that if the number of hidden units is too low, the network will perform poorly.

From Figure 9, Figure 10 and Table 3 it is evident that for a network with $J_{high} = 128$ units in the hidden layer, there is an increase in performance so small that it can be considered negligible. There is however, an increase in training time as the plots show that the graph of the 128-unit network requires an extra amount of timesteps compared to the baseline network. As evident in Table 3 the larger network uses an extra epoch before training terminates. Thus it can be concluded that if the number of hidden units is too large, the increase in performance is negligible while and the training time for the network increases.

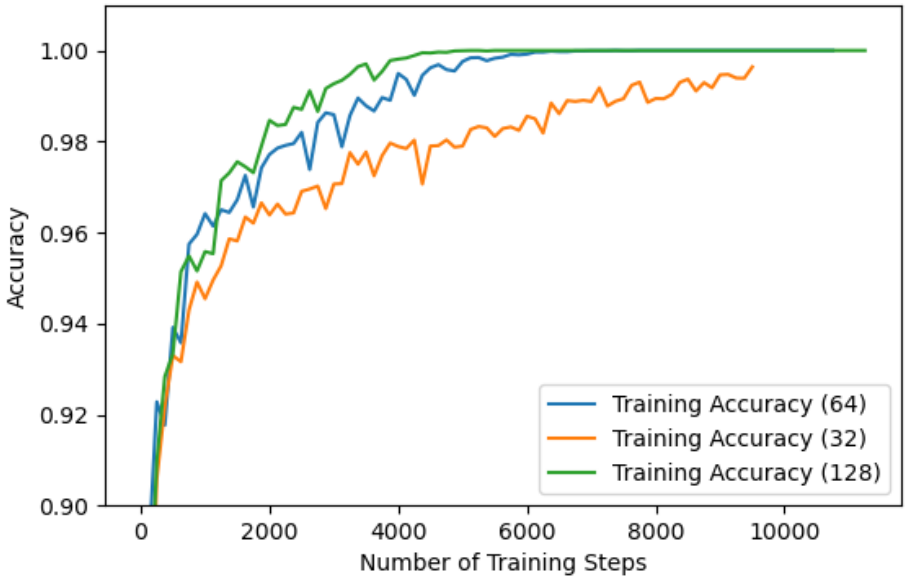

Figure 9: Comparison of performance impact on the training set accuracy with different number of hidden nodes.

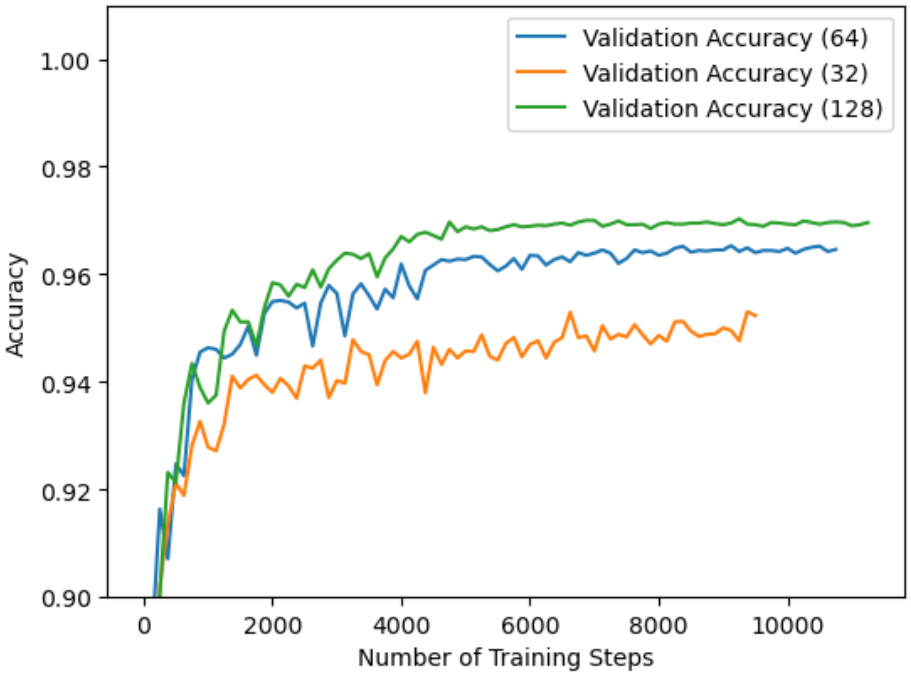

Figure 10: Comparison of performance impact on the validation set accuracy with different number of hidden nodes.

Table 3: Final performance metrics for different number of hidden units.

# Hidden units	Validation Loss	Validation Accuracy	Training Loss	Training Accuracy	Epochs
32	0.1991	95.23%	0.0186	99.64%	15
64	0.1544	96.46%	0.0018	100%	17
128	0.1314	96.96%	0.0008	100%	18

4.2 Task 4d: Multi layer network

A new network with two hidden layers have been implemented. It is desirable to have approximately the same number of parameters in the new network as the number of parameteres in the old, single-hidden-layer network, which from (38) is 50,880. By letting each of the two hidden layers in the new network have a total of 59 units, the total number of parameters becomes

$$P = 785 \cdot 59 + 59 \cdot 59 + 59 \cdot 10 = 50,386 \tag{43}$$

which is almost the same number of parameters as in the old network.

A comparison between the two networks is shown in Figure 11 and Figure 12. Additionally, the final performance metrics are outlined in Table 4.

From Figure 11, Figure 12 and Table 4 it is clear that the difference in the final accuracy of the networks is negligible. However, there is a difference in the final loss. The loss for the double hidden layered network is higher than the single hidden layered network. This could indicate, that while they both predict the same number of digits correctly, the single layer network is more "confident" in its predictions.

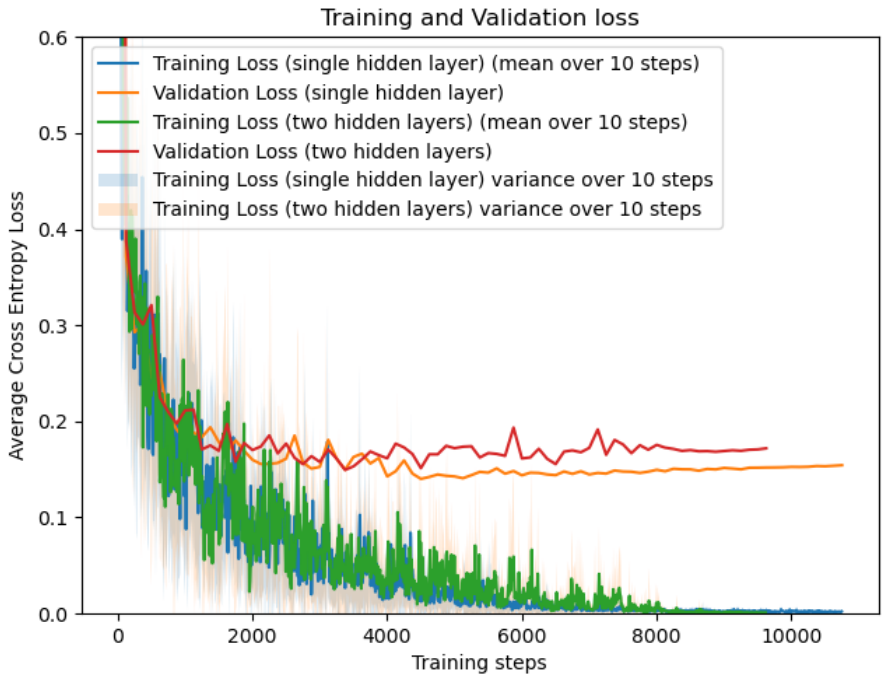

Figure 11: Validation and training loss for networks with approximately the same number of parameters but different topology.

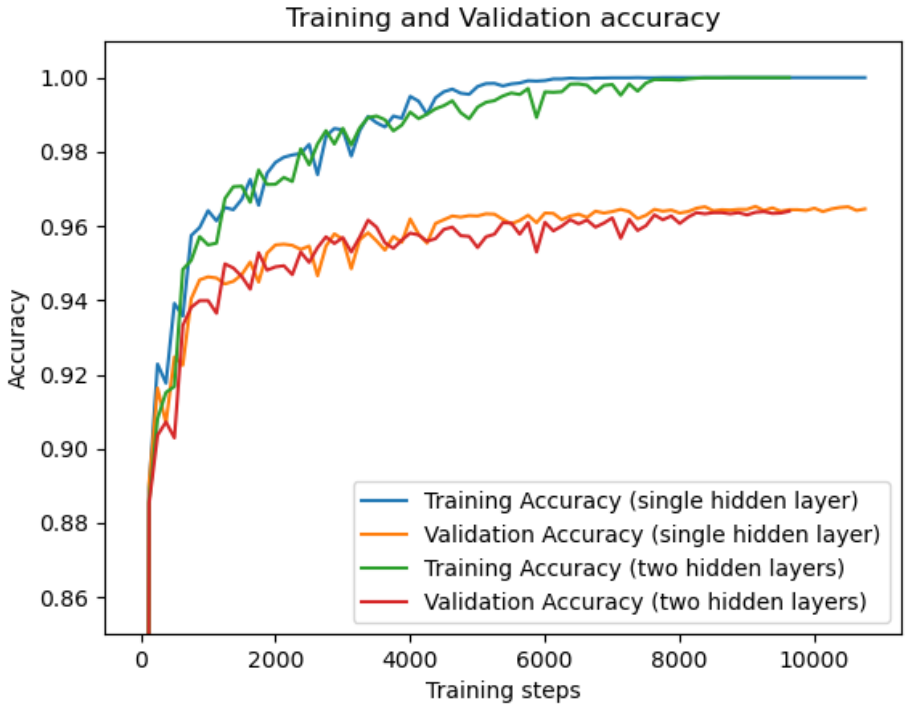

Figure 12: Validation and training accuracy for networks with approximately the same number of parameters but different topology.

Table 4: Final performance metrics for single hidden layer network and double hidden layer network. Both networks have the same number of parameters.

# Hidden units	Validation Loss	Validation Accuracy	Training Loss	Training Accuracy	Epochs
64	0.1543	96.46%	0.0009	100%	17
59, 59	0.1719	96.40%	0.0018	100%	15

4.3 Task 4e: 10 layer network

A network with 10 hidden layers with 64 hidden units for each layer has been implemented. A comparsion of performance between the 10 layer network and the baseline one layer network from task 3 is shown in Figure 13 and Figure 14. Additionally, in Table 5 the final performance metrics for the model is shown.

The results show that the 10 layer network uses significantly more training steps before the training ends. This is a consequence of the large number of parameters in the model. Even though there are more parameters, the network has worse performance, both in terms of loss and accuracy. The model is likely too complex for the given problem, and the dataset maybe too small and not generalized enough. From Table 5 one can see that in the 10 layer model, there are a total of 87,744 parameters. This is a 72% increase from the baseline model, supporting the claim that the model is too complex. The fact that the number of epochs for training is up with 88%, but with no improvement of results, indicates that the number of weights is simply too great.

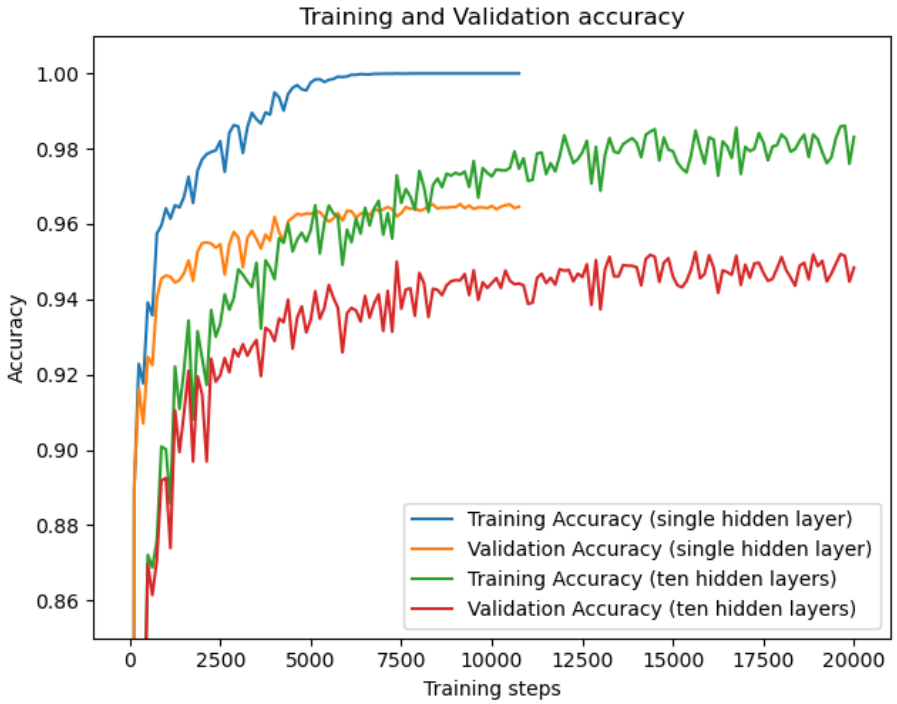

Figure 13: Training and validation accuracy for a single hidden layer network and a 10 hidden layer network.

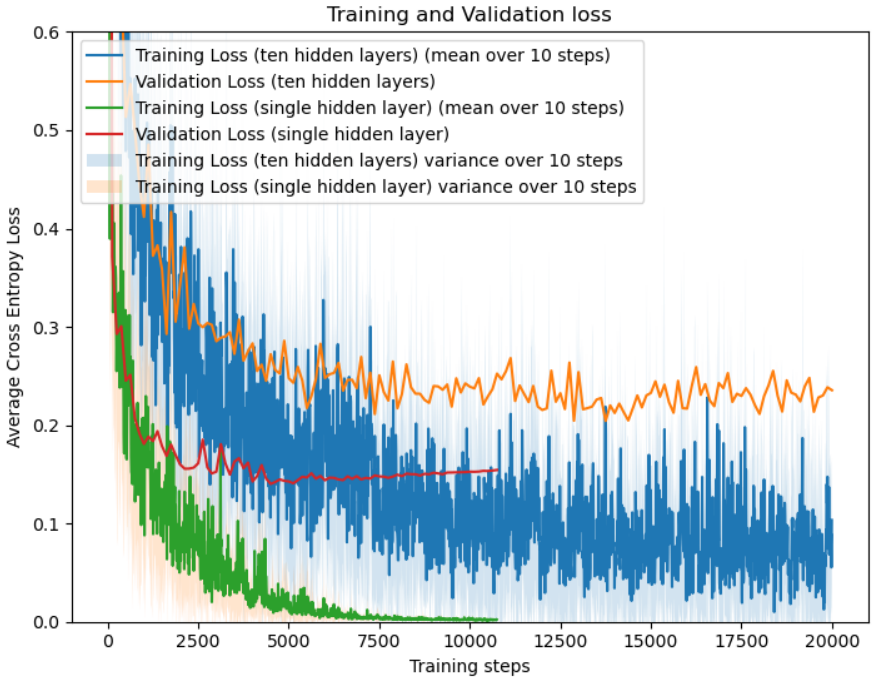

Figure 14: Training and validation loss for a single hidden layer network and a 10 hidden layer network.

Table 5: Final performance metrics for the 10 hidden layer neural network and a single layer neural network.

# Hidden units	**Validation Loss**	**Validation Accuracy**	**Training Loss**	**Training Accuracy**	**Epochs**
64	0.1543	96.46%	0.0018	100%	17
64 ·10	0.2355	94.84%	0.0591	98.32%	32

References

[1] [LeCun et al., 2012] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. *In Neural networks: Tricks of the trade*, pages 9–48. Springer.