TDT4265: Computer Vision and Deep Learning

Assignment 2

Espen Bragerhaug, Alfred Lieth Årøe

espentb@stud.ntnu.no, alfredla@stud.ntnu.no

Task 1: Softmax regression with backpropagation, theory

Task 1a: Backpropagation

Our task is to rewrite the equation to update the weights

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

such that it can be written as a recursive update rule that can be applied without computing $\frac{\partial C}{\partial w_{ji}}$ directly. We start by rewriting this term and use the fact that $\delta_j = \frac{\partial C}{\partial z_j}$

$$\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}} = \delta_j \cdot \frac{\partial z_j}{\partial w_{ji}}$$

We now calculate the term $\frac{\partial z_j}{\partial w_{ji}}$

$$\frac{\partial z_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left(\sum_{\hat{\imath}=0}^{d} w_{j\hat{\imath}} \cdot x_{\hat{\imath}}\right) = \frac{\partial}{\partial w_{ji}}\left(w_{ji} \cdot x_i + \sum_{\hat{\imath}\neq i}^{d} w_{j\hat{\imath}} \cdot x_{\hat{\imath}}\right) = x_i$$

Combining this we get that

$$w_{ji} = w_{ji} - \alpha \delta_j x_i$$

Now we have to find out what δ_j is and start by again using that $\delta_j = \frac{\partial C}{\partial z_j}$

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

From the assignment text we have that $\delta_k = \frac{\partial C}{\partial z_k}$ and that $\frac{\partial a_j}{\partial z_j} = f'(z_j)$

$$\delta_j = \sum_k \delta_k \cdot \frac{\partial z_k}{\partial a_j} \cdot f'(z_j)$$

The calculation of $\frac{\partial z_k}{\partial a_j}$ is almost the same as $\frac{\partial z_j}{\partial w_{ji}}$

$$\frac{\partial z_k}{\partial a_j} = \frac{\partial}{\partial a_j}\left(\sum_{\hat{\jmath}=0}^{d} w_{k\hat{\jmath}} \cdot a_{\hat{\jmath}}\right) = \frac{\partial}{\partial a_j}\left(w_{kj} \cdot a_j + \sum_{\hat{\jmath}\neq j}^{d} w_{k\hat{\jmath}} \cdot a_{\hat{\jmath}}\right) = w_{kj}$$

Combining this we get that

$$\delta_j = f'(z_j) \cdot \sum_k \delta_k \cdot w_{kj}$$

Task 1b: Vectorize computation

In this task we are going to show the update rule for (1) the weight matrix from the hidden layer to output layer and for (2) the weight matrix from input layer to hidden layer, using matrix/vector notation. In my notation I will use · as a matrix dot multiplication and ⊙ as the the Hadamard product as it is called in Nielsen's book (i.e. elementwise matrix multiplication).

Starting with (1):

$$w_{kj} := w_{kj} - \alpha\delta_k a_j = w_{kj} - \alpha\left(-(y_k - \hat{y}_k)^T \cdot a_j\right)^T$$

Here y_k have the shape [batch size, num_classes] and $\hat{y}_k$ have the shape [batch size, num_outputs]. These should always be the same shapes. a_j has the shape [batch size, neurons in layer j]. This means that we and up with ([num_classes , batch size] · [batch size, neurons in layer j])^T = [num_classes, neurons in layer j]^T = [neurons in layer j, num_classes]. This is also the shape of w_{kj}.

Now we look at (2):

$$w_{ji} := w_{ji} - \alpha\delta_j x_i = w_{ji} - \alpha \cdot \left(\left(\left(w_{kj} \cdot {\delta_k}^T\right)^T \odot f'(z_j)\right)^T \cdot x_i\right)^T$$

where $z_j = \sum_{i=0}^{d} x_i \cdot w_{ji}$.
x_i has the shape [batch size, I] and w_{ji} has the shape [I , neurons in layer j], meaning that z_j (and therefor also $f'(z_j)$) has the shape [batch size, neurons in layer j]. We know that the shape of w_{kj} it [neurons in layer j, num_classes], and that δ_k has the shape [batch size, num_classes]. Performing the calculations and transformations between the two we get a shape of [batch size, neurons in layer j], which is the same shape that $f'(z_j)$ has, making the elementwise matrix multiplication possible. The final shape of $\delta_j x_i$ becomes [I, neurons in layer j].

Task 2: Softmax regression with backpropagation, programming

Task 2c:

Here is a plot of the results from task 2c).

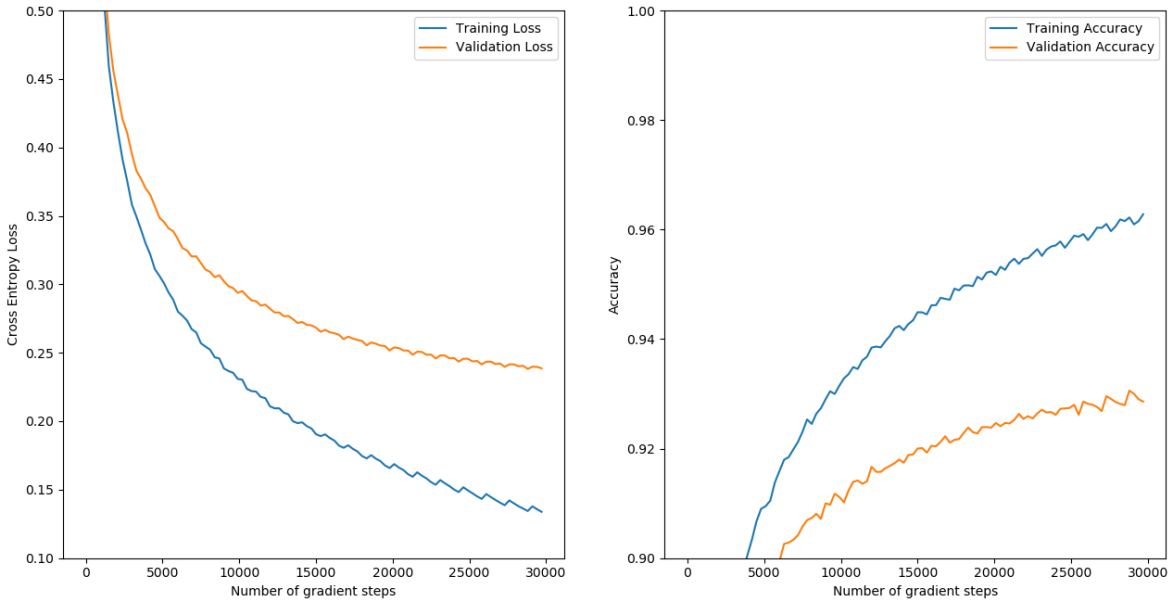

Figure 1: Entropy loss and acuracy graphs in task 2c)

Task 2d:

Number of parameters: $(784 + 1) * 64 + 64 * 10 = 50880$
We only use a bias for the first hidden layer

Task 3: Adding the ”Tricks of the Trade”

In this task we have collected all of the results in one graph, where we separated training loss, validation loss, training accuracy and validation accuracy. The tricks are added one after another in the order of the tasks, meaning the trick that is using the improved sigmoid also uses the shuffle trick and so on. One thing to notice here is that we had to lower the y-limit on the training loss graph, as the last two tricks became lower than 0.1.

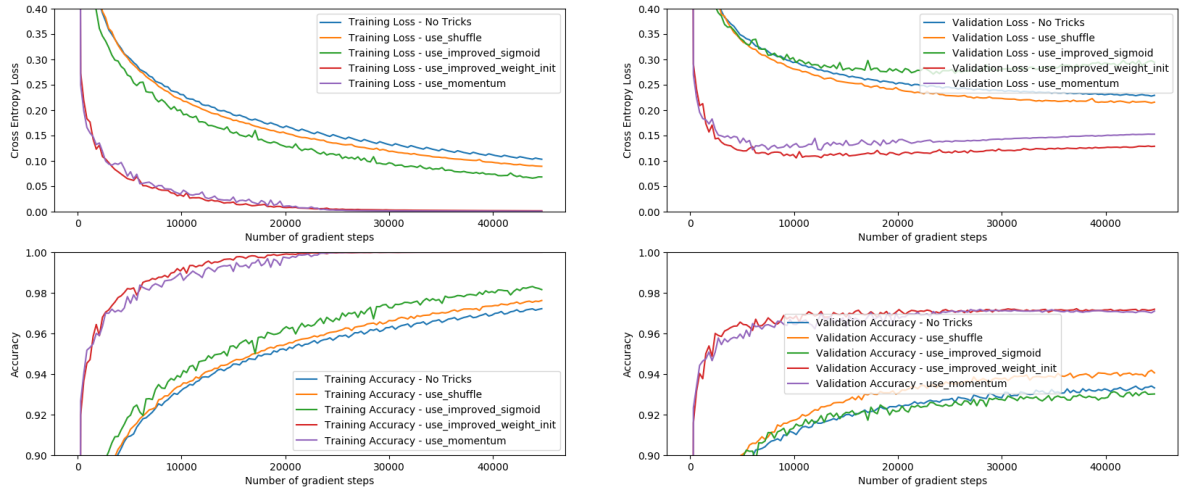

Figure 2: Results from every trick in task 3

The first thing we notice is that adding the trick in general improves the network. The one exception is the improved sigmoid, which is slightly worse than using no tricks. The trick that improved the results the most is the improved weight initialization. The other thing to note here is that the two best network converges early on and are starting to overfit around 10 000 steps.

Task 4: Experiment with network topology

Task 4a:

Here is the plot form task 4a where number of hidden units is set to 16.

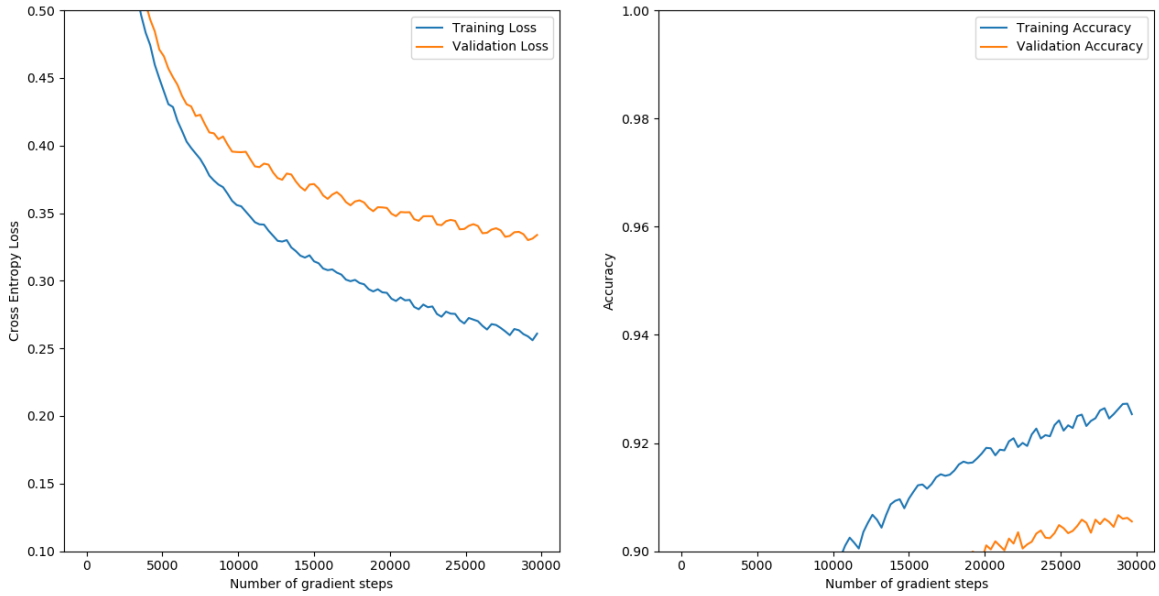

Figure 3: Resulting graph from setting number of hidden units to 16

We observe that this network does not achieve as good accuracy as the network with 64 hidden units.

Task 4b:

Here is the plot form task 4a where number of hidden units is doubled from the original, a total of 128.

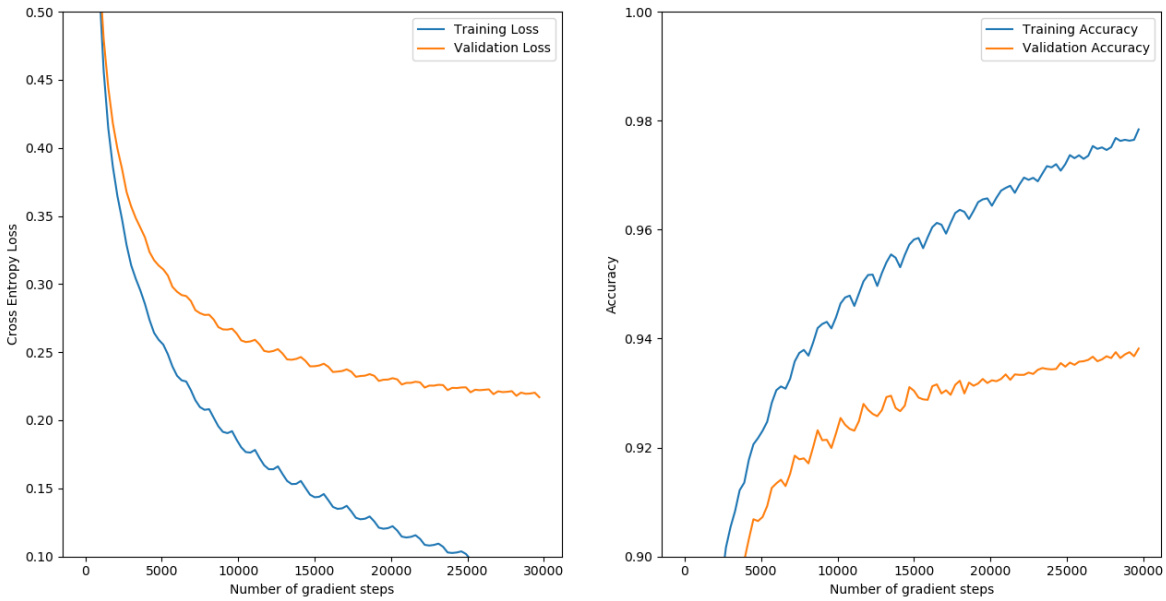

Figure 3: Resulting graph from setting number of hidden units to 128

We observe that this network does not achieve as good accuracy as the network with 64 hidden units, and we start to see a large difference between the training and validation. Is seems like the network does not generalize.

Task 4 c:

After some work, all the tests finally passed!

Task 4 d:

The number of parameters in task 3 is the same as what we calculated in task 2, namely 50890. Now we are given the task to make a new model with two hidden layers of equal size that have approximately the same number of parameters as the network from task 3. For the two hidden layers we chose to have 60 hidden units in each. This yields a network with a total of

$$(784 + 1) * 60 \; + \; 60 * 60 \; + \; 60 * 10 \; = \; 51300$$

parameters.

While training we chose not to use any of the tricks implemented in task 3. Here is the results from the training.

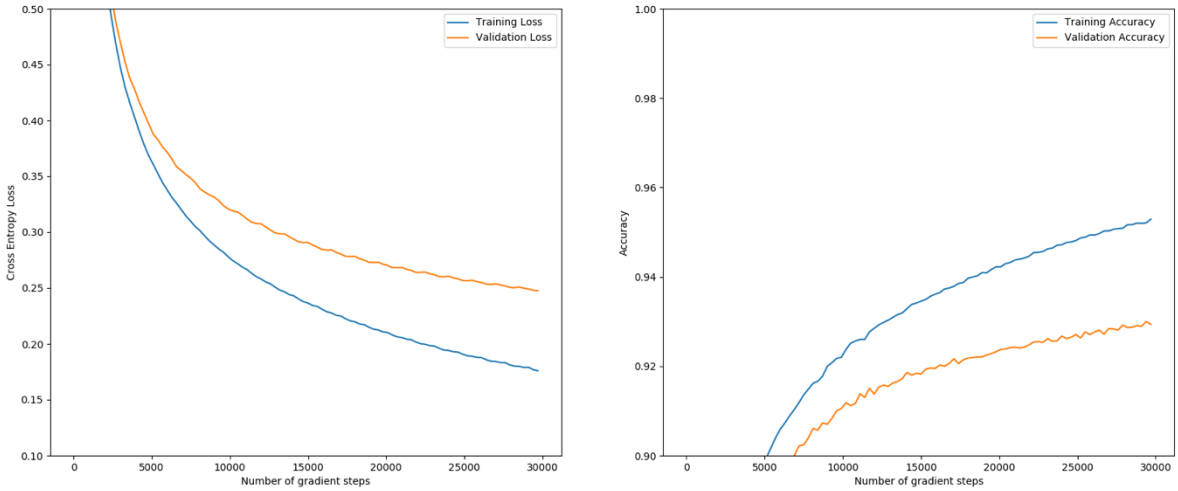

Figure 4: Resulting graph from the new network with two hidden layers

This network hat approximately the same validation accuracy, but in every other aspect it performed slightly worse. One thing to notice however is that the graphs are a lot smoother compared to what they previously were.

Bonus:

We experimented around with several different types of topologies for the network, ranging from one to seven hidden layers. Most of the network performed poorly so there is no point in showing the results here. The most promising network was built up with three hidden layers, all consisting of 64 units. We observes that the model needed more epochs to train before converging, and that a higher learning rate gave better results. Here is the plot from the final trained network.

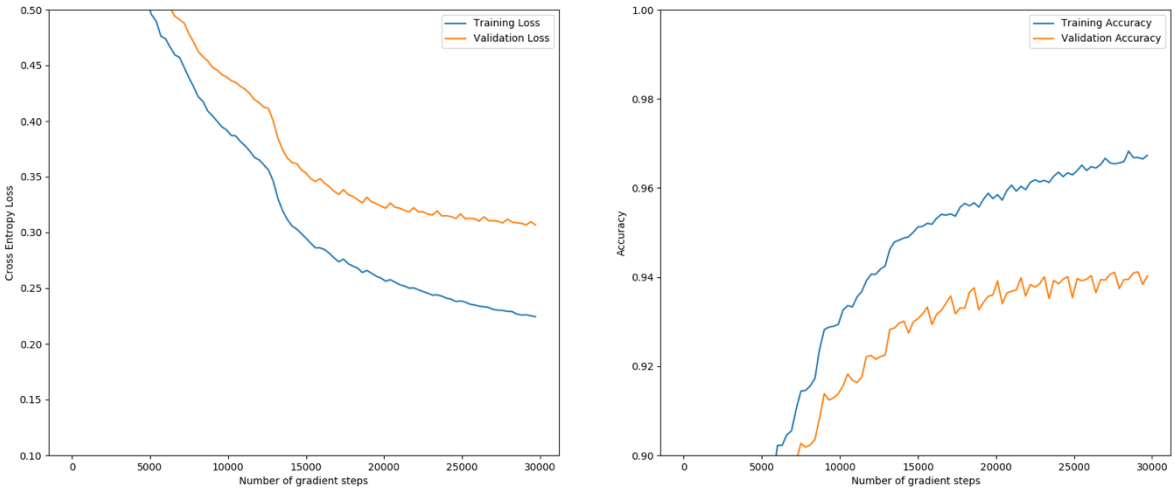

Figure 5: Resulting graph from the custom network