NTNU
Norwegian University of
Science and Technology

TDT4265 - Computer Vision and Deep Learning

Assignment 1

Jakob Løver
Einar Henriksen

January 31, 2019

Contents

1 Derivations of gradients for Logistic and Softmax Regression

1.1 Task 1.1 Derive the gradient for Logistic Regression

The loss function is given as

$$E(w) = -\frac{1}{N}\sum_{n=1}^{N} t^n \ln(y^n) + (1-t^n)\ln(1-y^n) \tag{1}$$

To find the gradient of the loss function for logistic regression, the goal is to find $\frac{\partial E^n(w)}{\partial w_j}$. This can be done through the chain rule as follows

$$-\frac{\partial E^n(w)}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{N}\sum_{n=1}^{N} t^n \ln(y^n) + (1-t^n)\ln(1-y^n) \tag{2}$$

$$= \frac{1}{N}\sum_{n=1}^{N} t^n \frac{\partial}{\partial w_j}\ln(y^n) + (1-t^n)\frac{\partial}{\partial w_j}\ln(1-y^n) \tag{3}$$

$$= \frac{1}{N}\sum_{n=1}^{N} t^n \frac{\frac{\partial}{\partial w_j}y^n}{y^n} + (1-t^n)\frac{\frac{\partial}{\partial w_j}(1-y^n)}{1-y^n} \tag{4}$$

$$= \frac{1}{N}\sum_{n=1}^{N} t^n \frac{\frac{\partial}{\partial w_j}\sigma(w_k^\intercal x^n)}{y^n} + (1-t^n)\frac{\frac{\partial}{\partial w_j}(1-\sigma(w_k^\intercal x^n))}{1-y^n} \tag{5}$$

$$= \frac{1}{N}\sum_{n=1}^{N} t^n \frac{y^n(1-y^n)\frac{\partial}{\partial w_j}(w_k^\intercal x^n)}{y^n} - (1-t^n)\frac{y^n(1-y^n)\frac{\partial}{\partial w_j}(w_k^\intercal x^n)}{1-y^n} \tag{6}$$

$$= \frac{1}{N}\sum_{n=1}^{N} t^n(1-y^n)x_j^n - (1-t^n)y^n x_j^n \tag{7}$$

$$= \frac{1}{N}\sum_{n=1}^{N} (t^n - y^n)x_j^n \tag{8}$$

$$= (t^n - y^n)x_j^n \tag{9}$$

1.2 Task 2.1 Derive the gradient for Softmax Regression

The cost function is given as

$$E = -\sum_{n=1}^{N}\sum_{k=1}^{C} t_k^n \ln(y_k^n) \tag{10}$$

where each element in the output vector y^n is given by

$$y_k^n = \frac{e^{a_k^n}}{\sum_{k'} e^{a_{k'}^n}}, \text{ where } a_k^n = w_k^\mathsf{T} x^n. \tag{11}$$

The gradient of the cost function is found in the following way:

$$-\frac{\partial E^n(w)}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{C} t_k^n \ln(\frac{e^{a_k^n}}{\sum_{k'} e^{a_{k'}^n}}) \tag{12}$$

$$= \frac{\partial}{\partial w_{kj}} \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{C} t_k^n (\ln(e^{a_k^n}) - \ln(\sum_{k'} e^{a_{k'}^n})) \tag{13}$$

$$\tag{14}$$

Separates into the cases where $y^n = k$ and where it isn't:

$$= \frac{1}{N} \sum_{n=1}^{N} t_k^n (x_j^n - \frac{1}{\sum_{k'} e^{a_{k'}^n}} e^{a_k^n} x_j^n) - t_k^n \frac{1}{\sum_{k'} e^{a_{k'}^n}} e^{a_k^n} x_j^n \tag{15}$$

$$= \frac{1}{N} \sum_{n=1}^{N} x_j^n (t_k^n - y_k^n) \tag{16}$$

$$= x_j^n (t_k^n - y_k^n) \tag{17}$$

2 Logistic Regression through gradient descent

2.1 Task 2.1 Logistic Regression through gradient descent

2.1.1 Task a and b

First, we removed all images that did not contain 2 or 3. The training set was set to 20000 images, where 10% of them were used as a validation set. We used the first 2000 images for the validation set. An early stopping algorithm was implemented to stop the training when the validation set cost increased over three epochs. The entire logistic regression algorithm was implemented as a class in Python to abstract as much as possible away from the run files. This mimics the behavior of frameworks like TensorFlow. We normalized the data by dividing all the pixel values by 255, the maximum grayscale value in the data set.

We decided on a learning rate of $\alpha = 0.01$. This learning rate was the largest we were able to use without causing overflows in the log functions.

In figure 2.1, three different learning rates are shown. An annealing learning rate of the form $\alpha(t) = \frac{\alpha_0}{1+kt}$ was implemented, but we decided to just keep using our initial learning rate as we saw little to no improvement for this task.

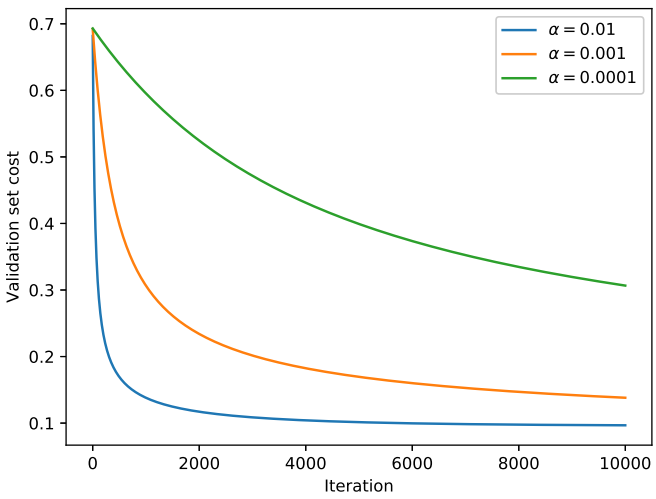

Figure 2.1: Cross entropy loss function over 10000 iterations on validation set

Figure 2.2 shows the cost and accuracy for all sets. On the test set, we

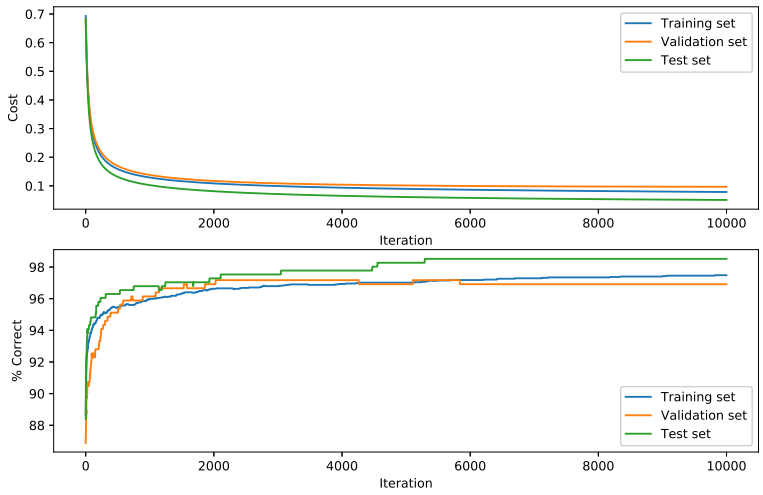

Figure 2.2: Cross entropy loss function of all data sets over 10000 iterations with $\alpha = 0.01$

achieved an accuracy of 98.77%. We observe from figure that the cost is approximately the same. The percentage of correctly classified images flattens out relatively quickly and stays the same for the validation set, so we conclude we achieved a higher score on the test set due to sampling bias, and not due to other circumstances such as underfitting.

In many real world scenarios, the validation set is a good stand-in for the test set. We don't have a test set available in most cases, so testing will often be done by seeing how performance is in a real scenario. It would of course be preferable to test on a solid test set, but the validation sets can at least help prevent overfitting etc. It's not a replacement for the test set, but it's better than nothing.

2.2 Task 2.2 Regularization

2.2.1 Task a

We can improve our learning algorithm by introducing regularization. This improves generalization when training our model, and prevents overfitting. In our case, we will introduce L2 regularization, which is defined as

$$C(w) = \frac{1}{2}||w||^2 = \frac{1}{2}\sum_{i,j} w_{i,j}^2 \tag{18}$$

where $w_{i,j}$ are all the weights in the weight vector.
Through partial differentiation,

$$\frac{\partial C(w)}{\partial w} = w \tag{19}$$

and our new cost function can now be written with the added term as

$$J(w) = E(w) + \lambda C(w) \tag{20}$$

and our new update term can be written as

$$\Delta w = -\alpha((t^n - y^n)x + \lambda w) \tag{21}$$

Note that we introduced the term $\frac{1}{2}$ in $C(w)$ to make differentiation easier. This term can be compensated for by selecting an appropriate λ.

2.2.2 Task b and c

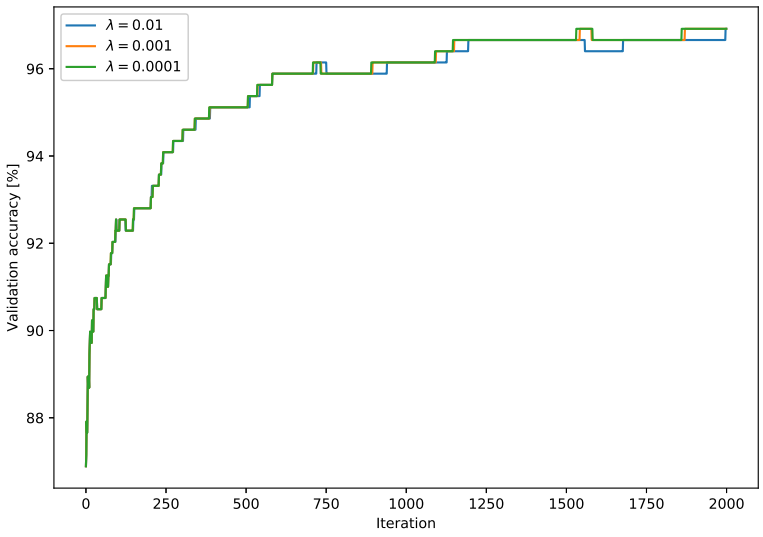

Figure 2.3: Validation accuracy for different values of λ

Figure 2.3 shows the validation accuracy for different values of λ with L2 regularization. The length of the weight vector is visualized in figure 2.4. There is a slight difference, but we decided to use $\lambda = 0.01$ because of it's shorter length, which may prevent overfitting

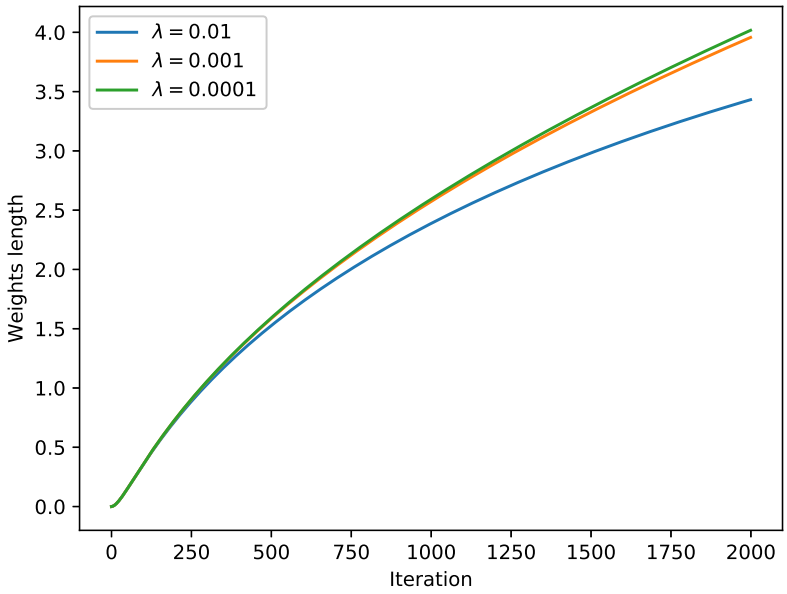

Figure 2.4: Length of weight vector for different values of λ

2.2.3 Task d

The weights are visualized in 2.5, but we see no difference in these.

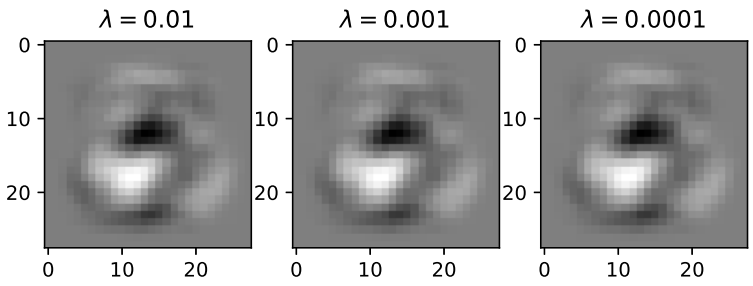

Figure 2.5: Visualization of weights for different values of λ

3 Softmax Regression through gradient descent

3.1 Task a and b

For values around $\lambda = 0.01$ the algorithm would only run for a couple of iterations and call it a day. We would also end up with a fairly low accuracy. This trend was not present when using a lower value, for example $\lambda = 0.0000001$, where we achieved an accuracy on the validation set of about 88%. We decided to use the latter value for the rest of this assignment.

3.2 Task c

Each digit has been visualized in figure 3.2. The images with dark backgrounds represent the average of that digit in the training set. The corresponding image is the weight vector for that digit. We can clearly see how the weights are shaped like the digits. We achieved an accuracy of about 88% on the validation set, and about 92% on the test set after 1000 iterations. This is a bit lower than expected, and we could probably have achieved a higher accuracy by implementing an annealing learning rate or something similar for this task.

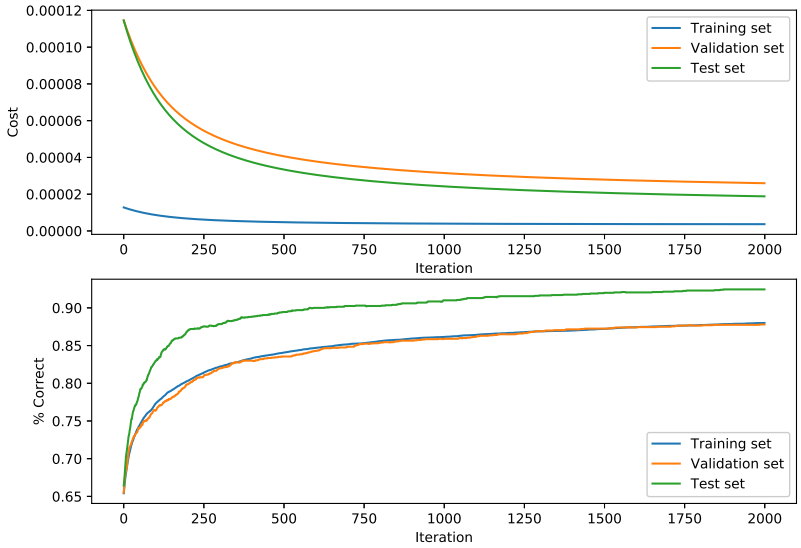

Figure 3.1: Cost function values and accuracy for softmax regression

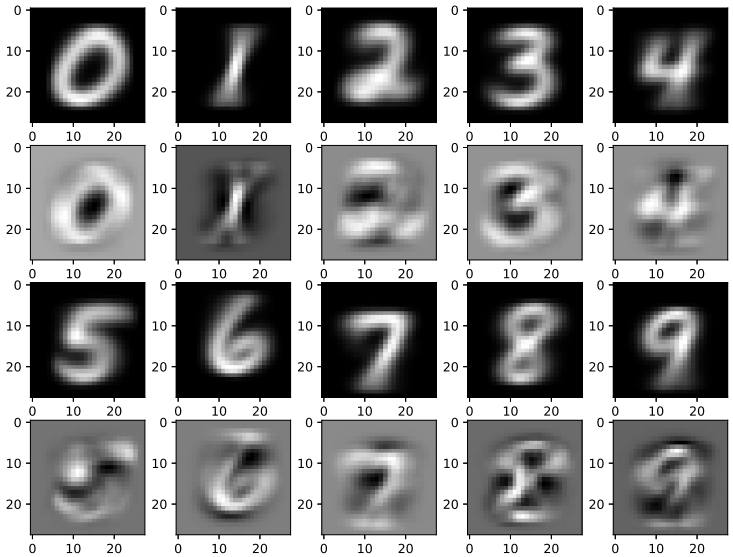

Figure 3.2: Average of all digits in training set together with the weights of each corresponding digit