DD2424/FDD3424 Deep Learning in Data Science, Assignment 1 (Bonus), 2021 Wai-Hong Anton Fu, whafu@kth.se

Bonus 1

We were asked to improve the performance of our network, I used three different optimization strategies, which are the following:

- Optimization a) Use all the available dataset to train the network.
- Optimization c) Perform a grid search to find the best configuration.
- Optimization d) Perform *step decay*.

Optimization a)

The results from the optimization can be seen on table 1. The optimization was done on the 4 cases from the baseline assignment which are the following cases:

- Case 1. lambda=0, n_epochs=40, n_batch=100, eta=0.1
- Case 2. lambda=0, n_epochs=40, n_batch=100, eta=0.001
- Case 3. lambda=0.1, n_epochs=40, n_batch=100, eta=0.001
- Case 4. lambda=1, n_epochs=40, n_batch=100, eta=0.001

We see that increasing the train size certainly increased the performance for almost all cases, except for case 4 (high lambda) which makes sense, since high lambda results in a very simple model, where more training samples wouldn't make much difference.

Case	Test Accuracy, without optimization (%)	Test Accuracy, with optimization (%)	Increase after optimization (%)
1	27.88	32.30	+4.43
2	38.80	40.20	+1.4
3	39.29	40.30	+1.01
4	37.57	37.50	-0.07

Tabell 1: Table of how much improvement each optimization a) got on each case.

Optimization c)

Grid search was used for optimization c), which is done easily by creating three loops, one for lambda, one for number of batches, and one for eta, then trying all the different configurations and saving the best ones with the highest test accuracy. The top five results can be seen on 2. Gridsearch is good, but very slow way to find appropriate configurations.

Test Accuracy (%)	Lambda	eta	Number Batches
39.29	0.1	0.001	100
39.17	0.1	0.01	1000
39.06	0.1	0.001	50
38.97	0.1	0.0001	10
38.90	0.01	0.01	1000

Tabell 2: Table that show the top 5 parameter configuration obtained from grid search.

Optimization d)

A simple weight decay was implemented, with other words, the eta (learning rate) was declined every epoch. The result is very dependent of the hyperparameters, for example when and how much to decrease the eta. In my implementation, I divided eta with 5 every 10th iteration. The results can be seen on 3 where most of the results are bad, except for case 1 with started with a really high eta. The other cases got worse results, likely due

to the eta being to low after a certain number of epochs, which cause the learning to be to slow.

A grids search may have been appropriate for weight decay to find the best hyperparameters that would otherwise be difficult to deduce on one's own. In the end for this experiment, the best accuracy was obtained on case 3 (which the gridsearch also suggested was the best configuration) whilst using all available dataset for training. The obtained test accuracy was 40.30%.

Case	Test Accuracy, without optimization (%)	Test Accuracy, with optimization (%)	Increase after optimization (%)
1	27.88	34.37	+6.49
2	38.80	36.85	-1.95
3	39.29	36.84	-2.45
4	37.57	36.64	-0.93

Tabell 3: Table of how much improvement each optimization d) got on each case.

Bonus 2

First and foremost, we write down the expression for $\frac{\partial l}{\partial \mathbf{s}}$ which per the rule of derivative is $\frac{\partial l}{\partial \mathbf{s}} = \frac{\partial l}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{s}}$. Beginning with the left term where the loss function l (multiple binary cross-entropy losses) is defined as:

$$l(\mathbf{x}, \mathbf{y}) = -\frac{1}{K} \Sigma_{K=1}^{K} [(1 - y_k) \log(1 - p_k) + y_k \log(p_k)] \tag{1}$$

And its derivative is:

$$\frac{\partial l}{\partial \mathbf{p}}(\mathbf{x}, \mathbf{y}) = \{\text{linearity}\} = \frac{1}{K} \Sigma_{K=1}^{K} \left[\frac{p_k - y_k}{(1 - p_k) p_k} \right] \tag{2}$$

We'll do the same with with the activation function $\mathbf{p} = \sigma(\mathbf{s})$ that is defined as:

$$\sigma(s) = \frac{\exp(s)}{\exp(s) + 1} \tag{3}$$

Due to the independencies assumed between each s, we can easily derivate $\sigma(s)$ as:

$$\frac{\partial \mathbf{p}}{\partial \mathbf{s}} = \frac{\exp(s)}{[\exp(s) + 1]^2} \tag{4}$$

After some calculus, we finally arrive at:

$$\frac{\partial l}{\partial \mathbf{s}} = \frac{\partial l}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{s}} = \frac{1}{K} \Sigma_{K=1}^{K} [\sigma(s) - y_k] \tag{5}$$

So a few minor adjustments are required for this implementation. The cost function has to be replaced by equation 1 and the activation function was replaced with the sigmoid function on equation 3. Ultimately, the gradient was calculated by multiplying X with the result from equation 5.

On the same four cases as before, the new implementation (BCE) and the one used previously in the assignment (baseline) were used. Table 4 shows the results, and while the differences aren't significant, the BCE appears to perform better by a small margin. Note that, due to the $1/K$ factor in the BCE loss function, table 4 may not be a fair 1-to-1 comparison.

Case	Test Accuracy, for BCE (%)	Test Accuracy, for baseline (%)	Increase with BCE (%)
1	35.95	35.65	+0.3
2	39.01	39.60	-0.59
3	39.49	39.49	+0
4	38.30	37.74	+0.56

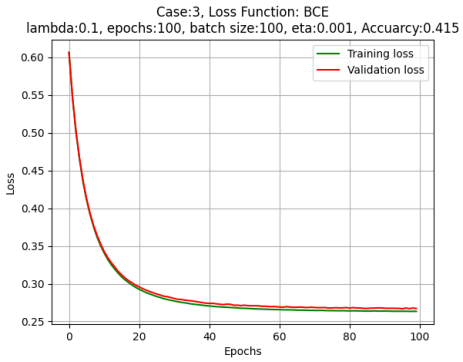
Tabell 4: Table test accuracy implementing with BCE vs baseline loss function.

There seems to be no apparent qualitative differences between the two implementations when looking at the percentage of correct- and incorrect classification for each label, which can be seen on the histograms on figure 3 and 4.

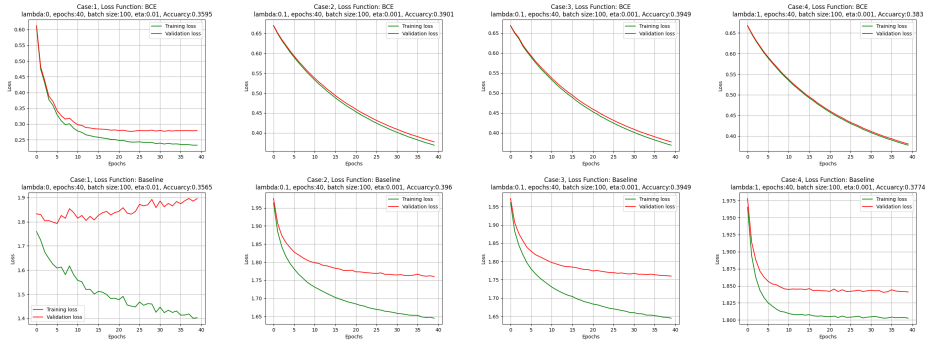
When looking at the loss functions on 2, there appears to be a clear trend that the BCE implementation generalizes much better. This is evident because, when compared to the baseline implementation, the gap between training and validation loss is much smaller for BCE.

Finally, because BCE appears to converge slowly (as shown in figure 2), I tried to increase the number of epochs to 100. In addition, similar to optimization a), I used all of the training samples and obtained a test accuracy of 41.50%, which is the best result so far. The loss function of this run is shown on figure 1.

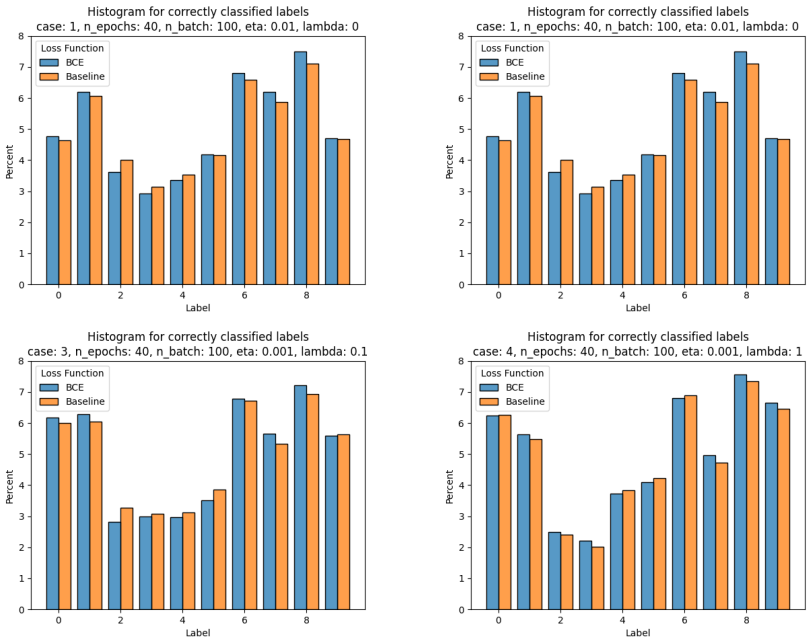
In conclusion, when using BCE as the loss function, there seems to be less overfitting compare to using vanilla cross entropy loss. Furthermore, BCE seems to yield better results, although the convergence seems to be slower, and therefore more epochs should be used.



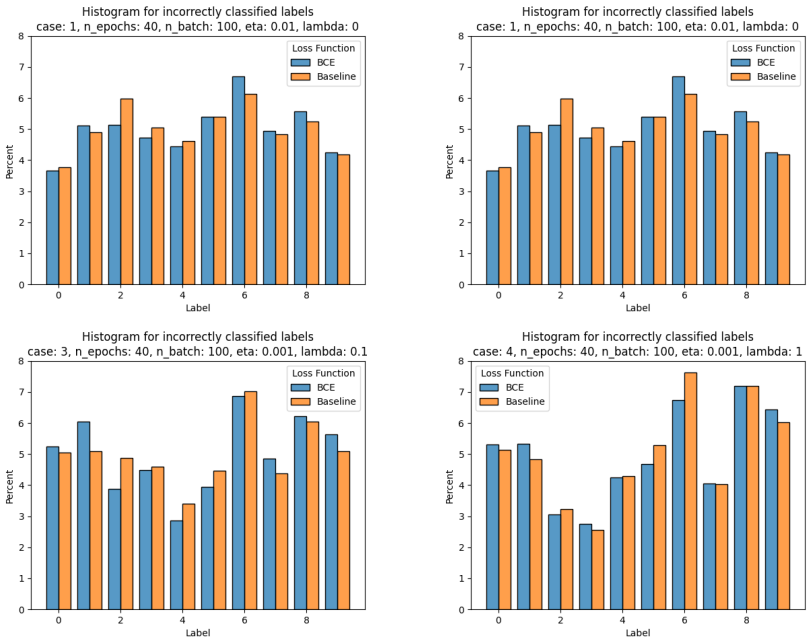
Figur 1: Best configuration with BCE, same configuration as case 3, but with 100 epochs



Figur 2: Loss function across each epoch for each case. The top row is with BCE, and the bottom row with baseline implementation for comparison.



Figur 3: Percentage of correct classifications for each label (0-9).



Figur 4: Percentage of incorrect classification for each label (0-9).