



KTH ROYAL INSTITUTE OF TECHNOLOGY

DEEP LEARNING IN DATASCIENCE
DD2424

Assignment 1 (Bonus)

Authors:
Ali Banaei Mobarak Abadi

April 4, 2022

Question 2.1

In this question we try to improve our network's performance. To do so, we used the following methods.

Increasing training set

As the first experiment, the size of the validation set was reduced to only 1K samples so we would have a bigger training set. We then trained the model with four different settings as before. The results of evaluation on test set are presented in [Table 1](#).

Table 1: Evaluation of the model trained on bigger training set.

λ	η	Loss	Cost	Accuracy
0	0.1	6.70	6.70	0.26
0	0.001	2.01	2.01	0.33
0.1	0.001	1.74	1.79	0.40
1	0.001	1.85	1.92	0.37

Now, if we compare these values and ones we got in the last part, we do not see any significant changes (for more confident conclusion, we may need to perform multiple tests and use significance tests, but here based on our inductive expectation and the values we can be sure the difference is not significant). It was expected that this will not lead to a major enhancement since the number of samples was not a major concern for us (we had enough samples to train our simple model).

Augmenting the training data

In this part, the effect of augmentation was investigated. In each step, in order to make a minibatch, we perform flipping by the probability of 0.5 for each sample. As we saw, despite some minor increases in performance in some cases, this method does not have a huge impact. One may again use the same argument that before which since we have a considerable amount of data and a very simple network, these methods, which aim to increase the diversity of the input data fed to the network, may not have a significant impact since the bottleneck here is the low complexity of the model. The test results for this part can be found in [Table 2](#).

Table 2: Performance of trained network using data augmentation on test data

λ	η	Loss	Cost	Accuracy
0	0.1	5.91	5.91	0.30
0	0.001	2.02	2.02	0.33
0.1	0.001	1.74	1.80	0.41
1	0.001	1.85	1.92	0.38

Learning rate decay

As the next improvement method, learning rate decay was used. At the beginning of the learning process, the learning rate was set to 0.01. Then, after 20 epochs, it was changed to 0.001. Then, after every 40 epochs, the learning rate was divided by 10. Increasing the learning rate after we converge to a flat area may help the network to converge faster to the optimal values instead of keeping jumping around it. As we can see in [Figure 1](#), on epochs that we decrease the learning rate, we have a sudden increase in accuracy and decrease in the loss function. Also, the performance of the network on the test data was increased, and in the end, we reached an accuracy of almost 0.43, which we had never seen before.

Question 2.2

We first derive the partial derivative of the loss function with respect to p_k .

$$\begin{aligned}\frac{\partial l}{\partial p_k} &= \frac{\partial \left(-\frac{1}{K} [(1 - y_k) \log(1 - p_k) + y_k \log(p_k)] \right)}{\partial p_k} = -\frac{1}{K} [(1 - y_k) \frac{-1}{1 - p_k} + y_k \frac{1}{p_k}] = \\ &= -\frac{1}{K} \left[\frac{y_k - 1}{1 - p_k} + \frac{y_k}{p_k} \right] = -\frac{1}{K} \frac{y_k - p_k}{p_k(1 - p_k)}\end{aligned}$$

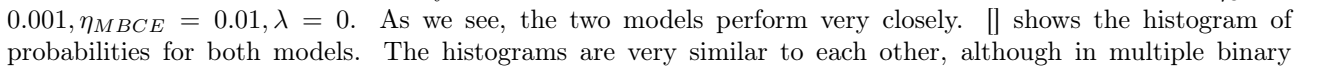
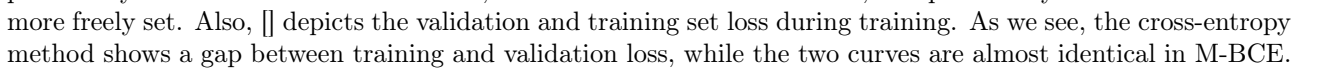
Now, we have

$$\begin{aligned}\frac{\partial p_k}{\partial s_k} &= \frac{\partial}{\partial s} (1 + e^{-s_k})^{-1} = -1(1 + e^{-s_k})^{-2} (-e^{-s_k}) = \frac{1}{1 + e^{-s_k}} \frac{e^{-s_k}}{1 + e^{-s_k}} = \\ &= \frac{1}{1 + e^{-s_k}} \left(1 - \frac{1}{1 + e^{-s_k}} \right) = \sigma(s_k)(1 - \sigma(s_k)) = p_k(1 - p_k)\end{aligned}$$

And finally we have

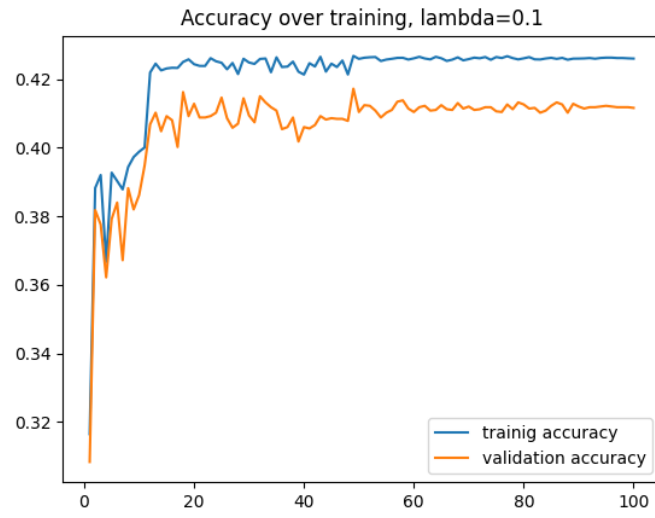
$$\frac{\partial l}{\partial s_k} = \frac{\partial l}{\partial p_k} \frac{\partial p_k}{\partial s_k} = \frac{1}{K} \frac{p_k - y_k}{p_k(1 - p_k)} p_k(1 - p_k) = \frac{1}{K} (p_k - y_k)$$

So, as we see, this is the same as cross-entropy with softmax activation except the deviation by the number of classes. So, in the code we only need to change the forward pass (replacing the softmax with sigmoid) and in gradient descent we can simply divide the gradients by the number of classes. Since all the trainable parameters are before s then this method will result in the same learning.

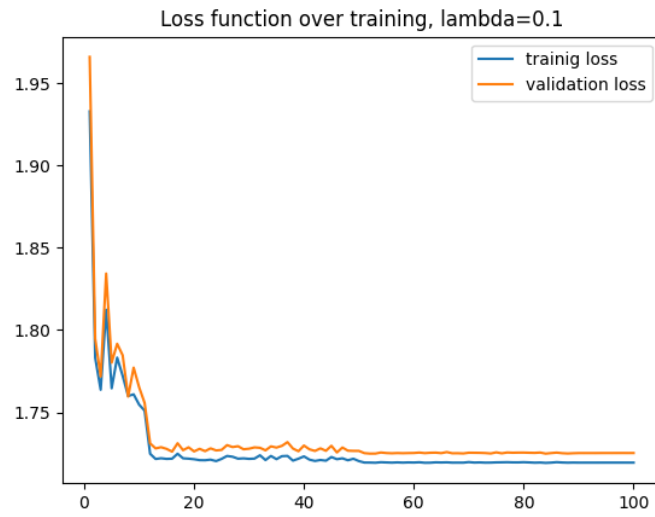
After training the network with the new objective function, the model was evaluated on the test results. The new model reached the accuracy of 0.33 while the former model could reach 0.32 when we used $\eta_{CE} = 0.001, \eta_{MBCE} = 0.01, \lambda = 0$. As we see, the two models perform very closely.  shows the histogram of probabilities for both models. The histograms are very similar to each other, although in multiple binary classification methods, we have some winner classes that have a probability below 0.2. In cross-entropy, since we use softmax and all probabilities sum to one, then we do not see a lot of winner classes with low assigned probability. But because in the M-BCE, we do not have this constraint, the probability of winner classes is more freely set. Also,  depicts the validation and training set loss during training. As we see, the cross-entropy method shows a gap between training and validation loss, while the two curves are almost identical in M-BCE. So, the M-BCE is less prone to overfitting.

Code documentation

Please go to the report for the mandatory part of the assignment for a full explanation of the implementation.

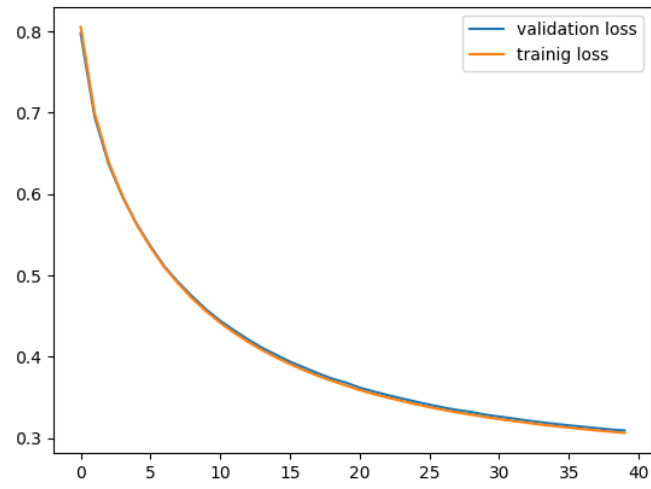


(a) Accuracy

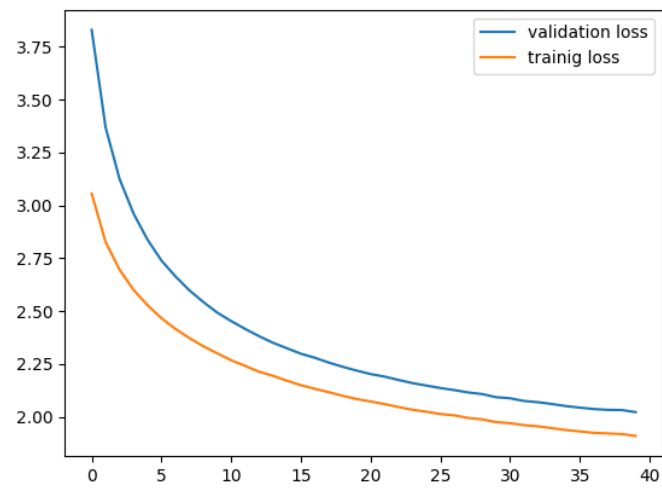


(b) Loss value

Figure 1: Loss and accuracy on training and validation data during trainig using a learning rate scheduler.

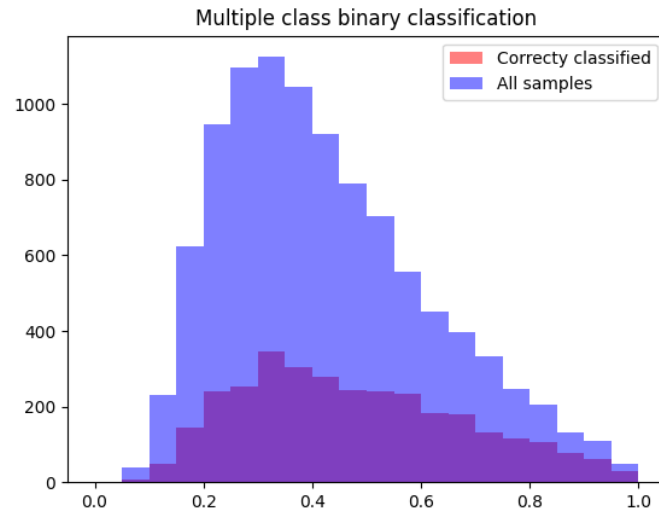


(a) Multiple-BCE

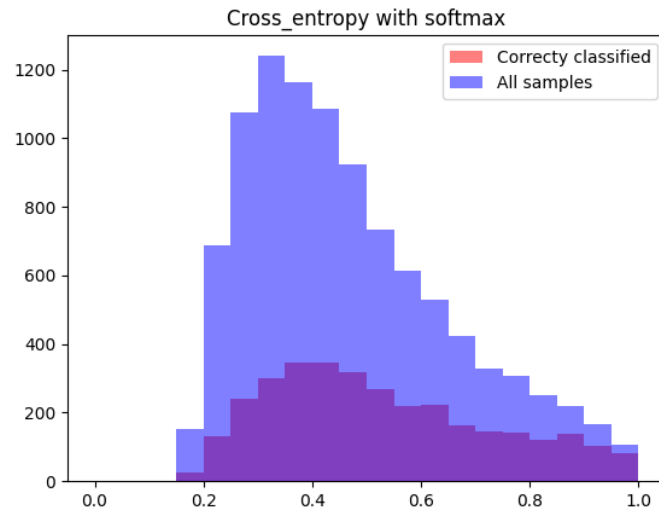


(b) Cross-validation

Figure 2: Loss on training and validation data during trainig.



(a) Multiple-BCE



(b) Cross-entropy

Figure 3: Histogram of the true class probabilities.