DD2424 - Deep Learning in Data Science

Assignment 1

Eloi Dieme

March 2024

Analytical Computation of the Gradient

We computed the gradients using the formulas:

$$\mathbf{G} = -(\mathbf{Y} - \mathbf{P})$$
$$\Rightarrow \begin{cases} \frac{\partial L}{\partial W} = \frac{1}{n_b}\mathbf{G}.\mathbf{X}^T + 2\lambda W \\ \frac{\partial L}{\partial \mathbf{b}} = \frac{1}{n_b}\mathbf{G}.\mathbf{1} \end{cases}$$

We used Numpy arrays and vectorized operations for this. We tested the correctness of the gradients with a function that compares each coefficient of the numerical and analytical gradient and outputs the maximum relative difference, computed according to the formula:

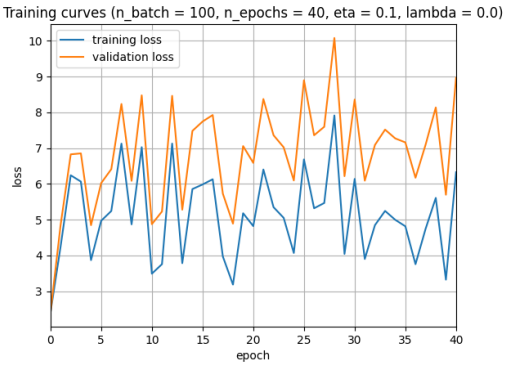
$$\frac{|g_a - g_n|}{\max(\epsilon, |g_a| + |g_n|)}$$

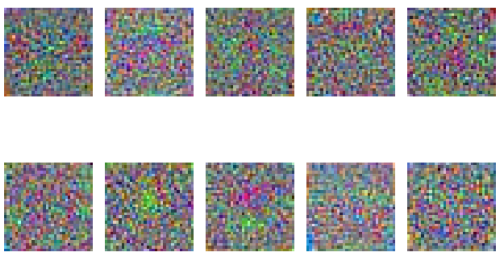
This difference doesn't go above `1e-6` for all the different cases, indicating that the analytical gradient is correct.

Results

- lambda=0, n_epochs=40, n_batch=100, eta=.1

Training curves and learned weights:

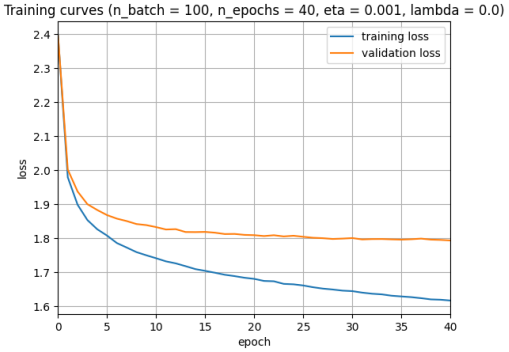


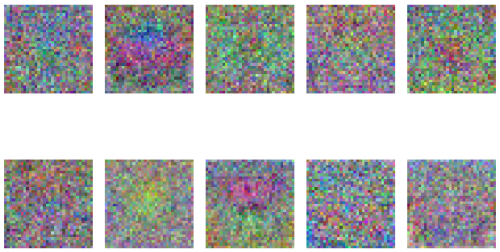


Accuracy on test data: **0.2571**

- lambda=0, n_epochs=40, n_batch=100, eta=.001

Training curves and learned weights:

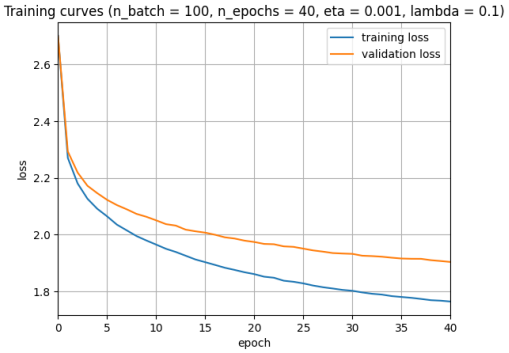


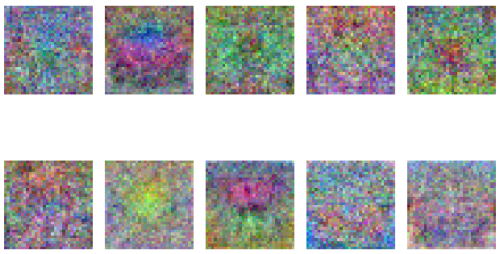


Accuracy on test data: **0.3906**

- lambda=.1, n_epochs=40, n_batch=100, eta=.001

Training curves and learned weights:

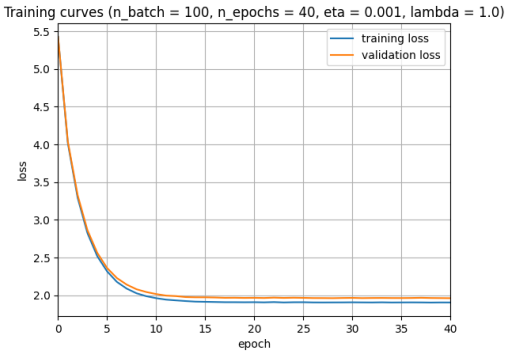


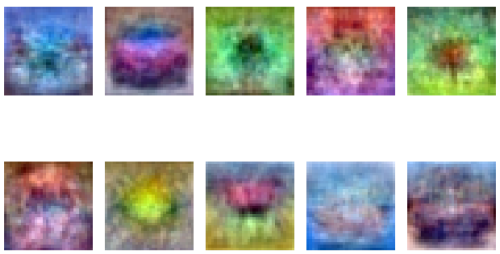


Accuracy on test data: **0.3915**

- lambda=1, n_epochs=40, n_batch=100, eta=.001

Training curves and learned weights:





Accuracy on test data: **0.3717**

We notice that when regularization increases the learned weights are "smoother", as if anti-aliasing was applied to the pictures. As expected, generalization is better overall but when λ is too high (1.0 for example), there is a slight underfitting, thus reducing the accuracy on test data. We also notice that when the learning rate is too high ($\eta = 0.1$), the gradient descent doesn't converge and the accuracy is much worse than usual.

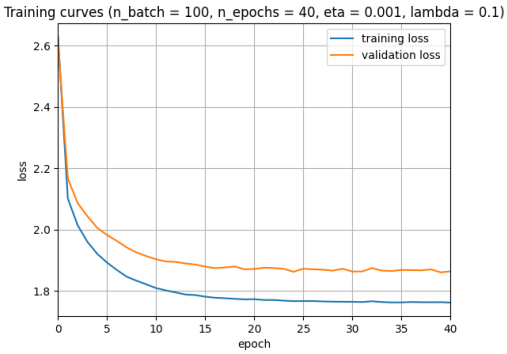
Bonus Points

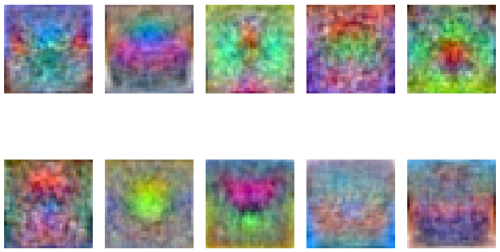
Improve performance of the network

To improve the performance of the network, we implemented three methods, namely using all the available training data, grid search and step decay.

Using all the available training data

The training set now contains 49000 samples, the validation set has 1000 and the test set hasn't been changed. We kept the best parameters found in the first part, namely $\lambda = 0.1$, `n_epochs` $= 40$, `n_batch` $= 100$, $\eta = 0.001$. We get the following results:





Accuracy on test data: **0.4058**

The accuracy has increased by approximately 1%, but the training took way longer than before. So this is not really worth it by itself. However, we notice that we could have stop around the twentieth epoch and we would have got approximately the same losses, so we could save some time here.

Grid Search

For the grid search, we specify a list of dictionaries containing various parameters and we train a model for each combination of parameters. The problem is that it can take a lot of time if we want the losses to converge for each model. So we can either search through a very small parameter space and train using 20 to 40 epochs, or search through a large parameter space but with a small amount of epochs (under 10). We chose the first solution. For the first one, we get these accuracies:

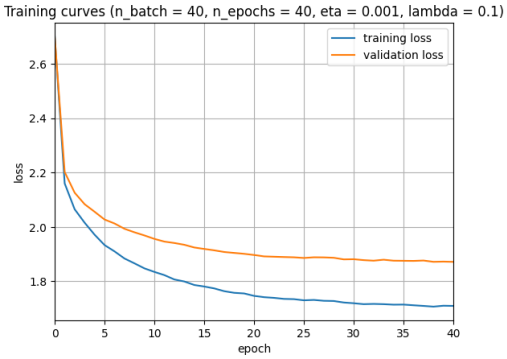
```
{
    "10_0.001_0.0": 0.359,
    "10_0.001_0.1": 0.3613,
    "10_0.001_0.5": 0.3432,
    "10_0.01_0.0": 0.2978,
    "10_0.01_0.1": 0.27,
    "10_0.01_0.5": 0.2212,
    "50_0.001_0.0": 0.3838,
    "50_0.001_0.1": 0.3846,
    "50_0.001_0.5": 0.3696,
    "50_0.01_0.0": 0.3389,
    "50_0.01_0.1": 0.3414,
    "50_0.01_0.5": 0.3228,
    "500_0.001_0.0": 0.3448,
    "500_0.001_0.1": 0.3452,
    "500_0.001_0.5": 0.3526,
    "500_0.01_0.0": 0.3796,
    "500_0.01_0.1": 0.3784,
    "500_0.01_0.5": 0.3728
}
```

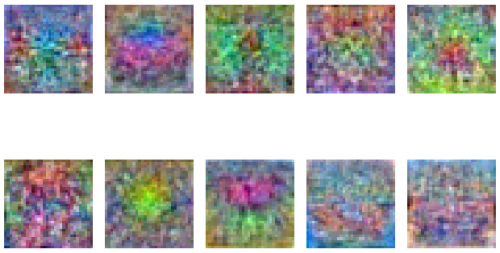
So the best accuracy is **0.3846** for 20 epochs and `n_batches` $= 50$, $\eta = 0.001$, $\lambda = 0.1$. We notice that increasing `n_batches` or decreasing it too much doesn't really improve performance and that the best values for η and λ

are the ones we had already found, namely 0.001 and 0.1. Now we can focus on `n_batches` and vary between 20 and 50 to find the best value:

```
{
    "20_0.001_0.1": 0.3759,
    "30_0.001_0.1": 0.3829,
    "40_0.001_0.1": 0.3875,
    "50_0.001_0.1": 0.3849
}
```

So we can select 40 and train with these parameters on 40 epochs. We finally get these results:

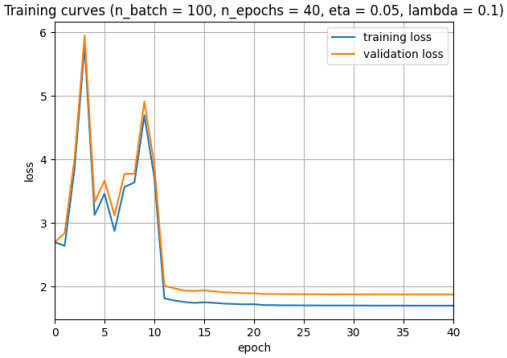


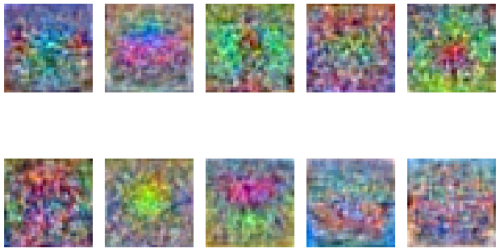


Accuracy on test data: **0.3884**
Unfortunately, we can't do better. It's probably the case that the grid search need to be performed with 40 epochs to find the best parameters for this configuration. But this would take too much time for not a big improvement. It's better to stick with the parameters $\{100, 0.001, 0.1\}$ for 40 epochs.

Step decay

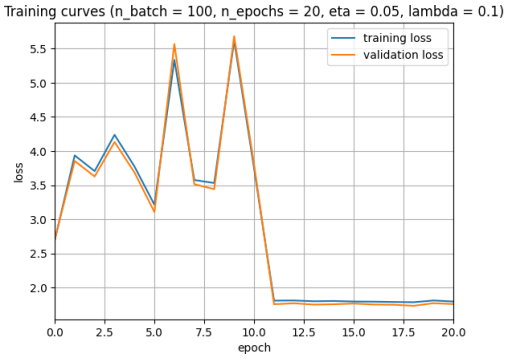
For step decay, we divide the learning rate by 10 every 10th epoch. We start with a learning rate of 0.05 and we get the following results:

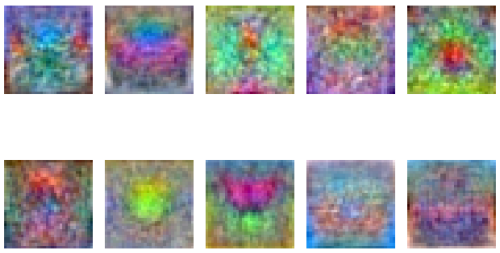




Accuracy on test data: **0.3951**
It's slightly better than the original performance. We could play around with different configurations for this step decay to further increase performance.

Finally, the method with the most performance improvement is training with all the available data, but it is very long. Most likely, the linear boundary decision is limiting the performance of our model in a way that makes it essentially impossible to break 40%+ accuracy. We can still try our best parameters with step decay on the full available data to see if we get something better:





Accuracy on test data: **0.3882**

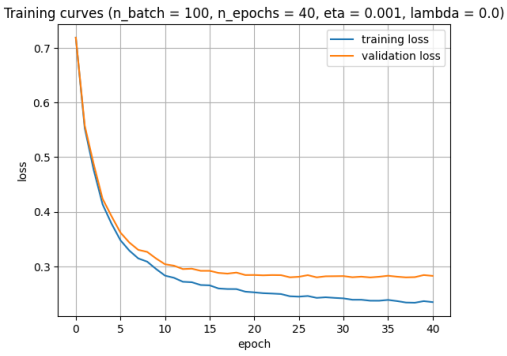
It seems like the individual improvements do not stack up to give a significant improvement. This was expected.

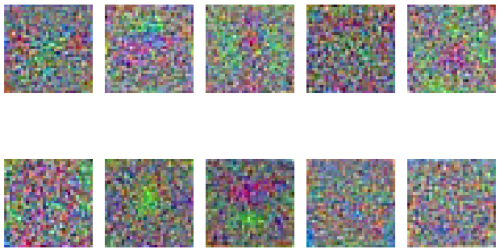
Multiple binary cross-entropy losses

Each coefficient of $\frac{\partial l_{\text{multiple\_bce}}}{\partial \mathbf{s}}$ has the form

$$\frac{\partial l_{\text{multiple\_bce}}}{\partial s_k} = -\frac{1}{K}\left[(1-y_k)\left(\frac{1}{1-\sigma(s_k)}-1\right)+y_k(1-\sigma(s_k))\right]$$

We get the following curves and weights after training:

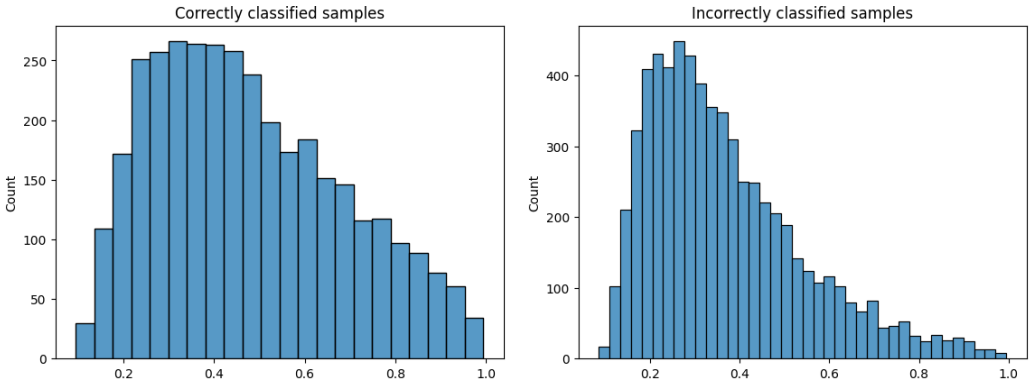




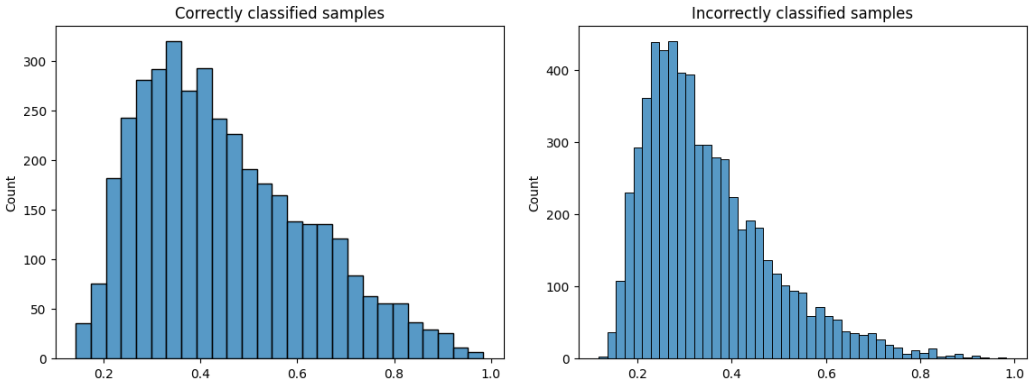
Accuracy on test data: **0.3546**

The accuracy is a bit lower than for the cross-entropy + softmax case but there seems to be less overfitting even without regularization. However, we would have to find the best gradient descent parameters to really compare the two methods fairly.

When plotting the histogram of the probability for the ground truth class for examples correctly and incorrectly classified we get:



And for the softmax + cross-entropy training:



We notice that the histograms for the softmax + cross-entropy training are slightly more skewed towards lower probabilities with noticeably thinnier tails, which is logical since with the softmax function the probabilities must sum up to one, meaning that the individual probabilities are usually smaller than for the sigmoid + multiple binary cross-entropy training.