DD2424 Deep Learning - Assignment 2

Hugo Brunlid

16 April 2024

1 Introduction

In the following assignment, we expand upon the network built previously by adding a hidden layer, making is a 2-layer neural network. For the forward pass, linear scoring $W\mathbf{x} + \mathbf{b}$ is used in combination with ReLU to propagate through the network. Softmax is used in the final step to normalize the values into probabilities for each class. We have

$$\begin{aligned} \mathbf{s}_1 &= W_1\mathbf{x} + \mathbf{b}_1 \\ \mathbf{h} &= \max(0, \mathbf{s}_1) \\ \mathbf{s} &= W_2\mathbf{h} + \mathbf{b}_2 \\ \mathbf{p} &= \text{Softmax}(\mathbf{s}) \end{aligned}$$

where the matrix W_1 and W_2 have size $m \times d$ and $K \times m$ respectively, and the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ have sizes $m \times 1$ and $K \times 1$. We use cross-entropy loss in combination with a L_2-regularization term to get the cost function.

$$J(\mathcal{D}, \lambda, \mathbf{\Theta}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} l_{\text{cross}}\ (\mathbf{x}_i, y_i, \mathbf{\Theta}) + \lambda \sum_{l=1}^{2} \sum_{i,j} W_{l,ij}^2$$

where the cross-entropy loss can be computed using the probabilities from the forward pass $\mathbf{p}$ and a one-hot representation of the labels $\mathbf{y}$

$$l_{\text{cross}}\ (\mathbf{x}, y, \mathbf{\Theta}) = -\mathbf{y}^T \log(\mathbf{p})$$

Mini-batch gradient descent is used to train the network, updating the weights and biases using analytically computed gradients. Different partitions of the CIFAR-10 data set are used for training, validation and testing. A computational graph for computing the cost function of the network is presented below.

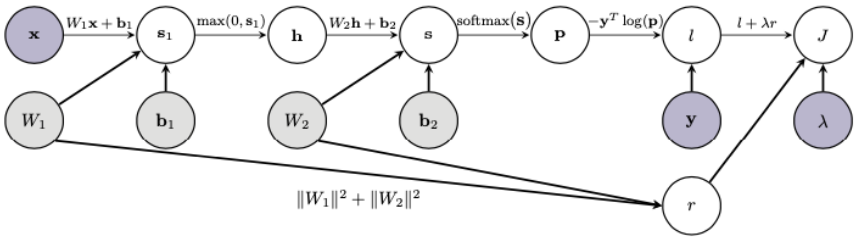


Figure 1: Cost Function

2 Gradient Computations

In order to train the network, we need to compute gradients for the weights and biases in each of the 2 layers. Here, we use mini-batch efficient gradient computations.

Let $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{n_b}, \mathbf{y}_{n_b})\}$ be the data in the mini-batch $\mathcal{D}^{(t)}$.

1. **Organize the batch data**
   Gather all $\mathbf{x}_i$:s from the batch into a matrix, and similarly for $\mathbf{y}_i$:s

$$X_{\text{batch}} = \begin{pmatrix} \uparrow & & \uparrow \\ \mathbf{x}_1 & \cdots & \mathbf{x}_{n_b} \\ \downarrow & & \downarrow \end{pmatrix}, \quad Y_{\text{batch}} = \begin{pmatrix} \uparrow & & \uparrow \\ \mathbf{y}_1 & \cdots & \mathbf{y}_{n_b} \\ \downarrow & & \downarrow \end{pmatrix}$$

2. **Forward pass** (using ReLU)

$$\begin{aligned} H_{\text{batch}} &= \max\left(W_1 X_{\text{batch}} + \mathbf{b}_1 \mathbf{1}_{n_b}^T, 0\right) \\ \mathbf{P}_{\text{batch}} &= \text{SoftMax}\left(W_2 H_{\text{batch}} + \mathbf{b}_2 \mathbf{1}_{n_b}^T\right) \end{aligned}$$

3. **Backward pass**
   Set $\mathbf{G}_{\text{batch}} = -\left(\mathbf{Y}_{\text{batch}} - \mathbf{P}_{\text{batch}}\right)$, then

$$\frac{\partial L}{\partial W_2} = \frac{1}{n_b} G_{\text{batch}} \, H_{\text{batch}}^T \qquad \frac{\partial L}{\partial \mathbf{b}_2} = \frac{1}{n_b} G_{\text{batch}} \, \mathbf{1}_{n_b}$$

   Propagate the gradient through the second layer

$$\begin{aligned} G_{\text{batch}} &= W_2^T G_{\text{batch}} \\ G_{\text{batch}} &= G_{\text{batch}} \odot \text{lnd}\left(H_{\text{batch}} > 0\right) \end{aligned}$$

   Finally, the gradients for the first layer are then computed as

$$\frac{\partial L}{\partial W_1} = \frac{1}{n_b} G_{\text{batch}} \, X_{\text{batch}}^T \qquad \frac{\partial L}{\partial \mathbf{b}_1} = \frac{1}{n_b} G_{\text{batch}} \, \mathbf{1}_{n_b}$$

These analytically computed gradients are compared with numerically computed gradients through `compute_grads_num_slow` implementing the *centered difference method* introduced in Assignment 1. This function is adapted to multi-layer networks and translated from MATLAB to Python. Analytical and numerical gradients are computed and compared for the first 10 training examples. Both visual comparison of and computing of the average absolute error yields promising results. *This indicates that analytical gradient computations are correct.*

Table 1: Average Absolute Error

	First Layer	**Hidden Layer**
Weights	1.51e-11	1.52e-11
Bias	3.3e-10	7.7e-11

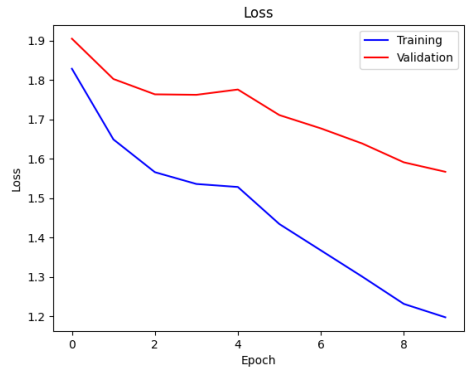
3 Cyclical Learning Rates

By varying the learning rate η in a cyclical fashion, model performance can be improved. Learning rate is varied in cycles using the step size n_s, and batch size to dictate the number of epochs. Here, the model is trained in two configurations to evaluate the impact of implementing cyclical learning rates. Note that both the training and validation data is of size $(3072, 10000)$.

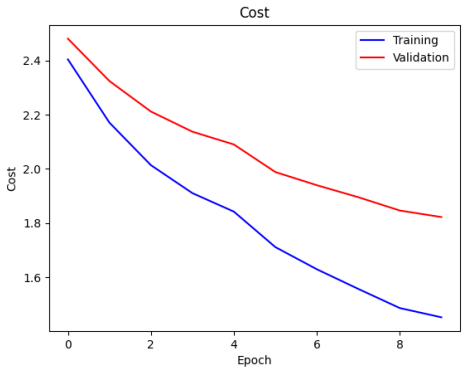
3.1 Configuration 1

$$\eta_{\min} = 10^{-5},\ \eta_{\max} = 10^{-1},\ n_s = 500,\ \lambda = 0.01,\ \texttt{batch\_size} = 100,\ \texttt{cycles} = 1$$

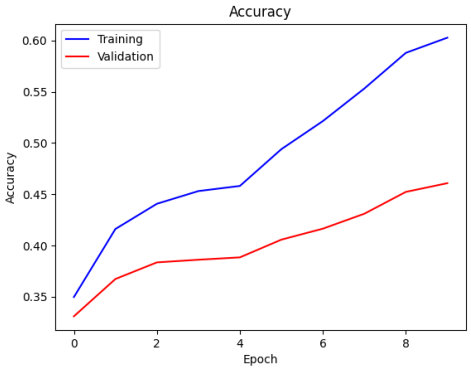
This configuration corresponds to the one in *Figure 3* in the assignment. One cycle of training is performed, corresponding to 10 epochs with the current parameter settings. The training progresses nicely, decreasing cost and loss over each epoch whilst increasing accuracy on the validation set. Although, model performance can likely be improved as we are only using a small subset of the available data and a single training cycle. At the end of the first cycle, the accuracy on the test set is 45.780%.



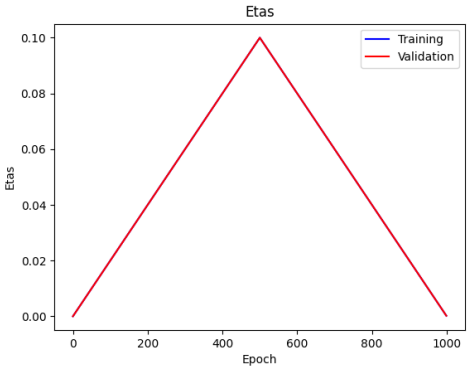
(a) Loss per epoch



(b) Cost per epoch



(c) Accuracy



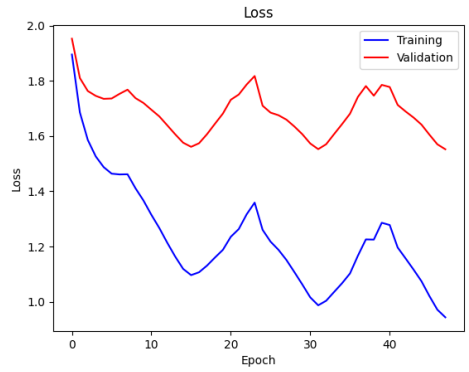
(d) Eta over 3 cycles

Figure 2: Training curves for one cycle of training

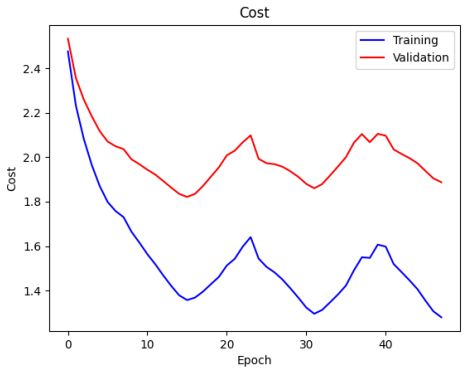
3.2 Configuration 2

$$\eta_{\min} = 10^{-5},\ \eta_{\max} = 10^{-1},\ n_s = 800,\ \lambda = 0.01,\ \texttt{batch\_size} = 100,\ \texttt{cycles} = 3$$

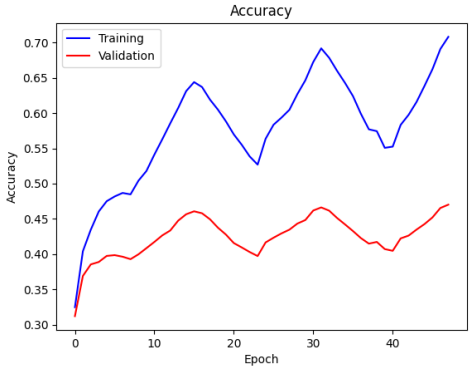
This configuration corresponds to the one in *Figure 4* in the assignment. These parameter settings result in 48 epochs, over 3 cycles of the learning rate. As evident by the loss and cost function plots, we can see how the model performance varies with η_t. As the learning rate increases, the gradient descent algorithm moves away from the minima, and returns once the learning rate is decreased again. This behavior might promote finding nearby local minima while at the same time converging nicely in the end. After all 3 cycles have elapsed, we get a test accuracy of 46.740%.



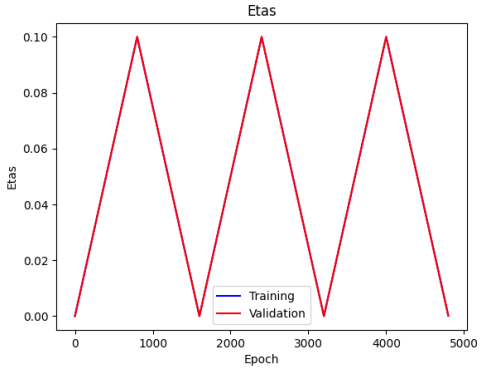
(a) Loss per epoch



(b) Cost per epoch



(c) Accuracy



(d) Eta over 1 cycle

Figure 3: Training curves for three cycles of training

4 Optimizing Lambda

During the lambda search, we used the following hyperparameters.

$\eta_{\min} = 10^{-5}$, $\eta_{\max} = 10^{-1}$, $n_s = 200$, `batch_size` $= 450$, `num_batches` $= 100$, `cycles` $= 3$

4.1 Coarse Search

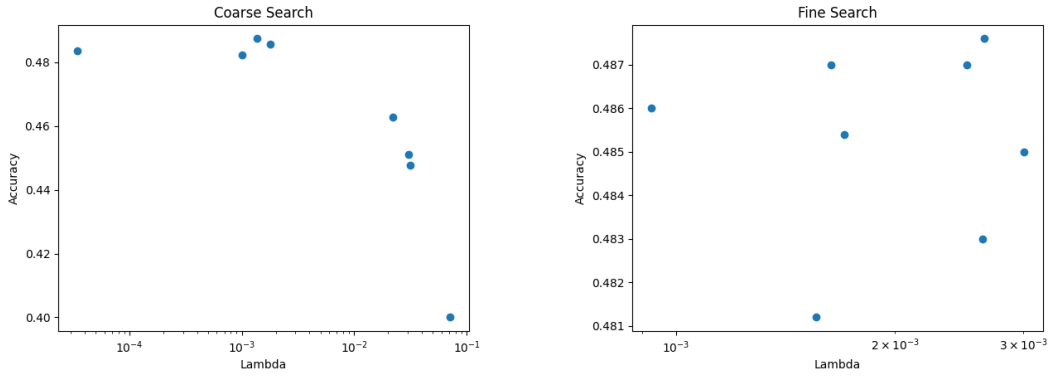
To find adequate values for the regularization parameter λ, a coarse search is performed. For this search, all data contained in CIFAR-10 is used, i.e. using 50000 images, where 45000 are training data and the remaining 5000 are used as validation data. The network is trained for 8 uniformly randomly sampled versions of λ in the range $[10^{-5}, 10^{-1}]$. In the figure, we plot accuracy on the validation set against the values of lambda.

- Top 3 Accuracy: $[0.4874, 0.4856, 0.4836]$
- Top 3 Lambdas: $[0.0014, 0.0018, 0.00003]$

4.2 Fine Search

The two best values for lambda from the coarse search are used as the boundaries for the fine search. Since they are very close to each other, the interval is augmented by ± 0.25 in the log10-scale to capture any beneficial parameter values outside this interval. Values are uniformly sampled from this narrower interval, training the model for 8 lambdas and using the same hyperparameters as above. *This method yields a best lambda of* 0.0027.

- Top 3 Accuracy: $[0.4876, 0.4870, 0.4870]$
- Top 3 Lambdas: $[0.0027, 0.0016, 0.0025]$



(a) Coarse Search (b) Fine Search

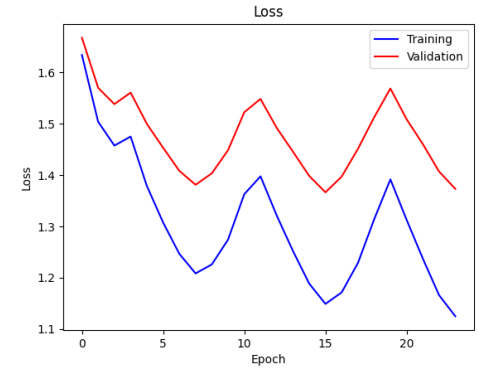
Figure 4: Validation Accuracy for Different Lambdas

5 Final Model

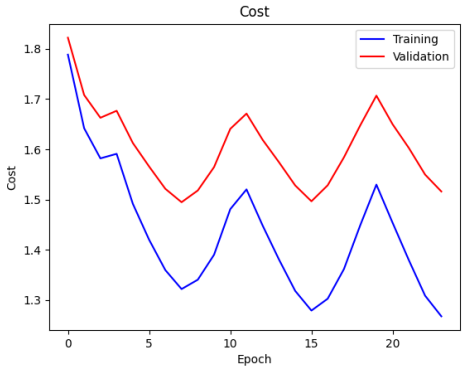
In the final model, we utilize the conclusions made about parameters in the previous models. Here, we use the entire data set of 50000 images with a validation size of 1000, resulting in a training set of size 49000. Using the *best lambda* derived in the previous section, in combination with increasing the step size n_s to get more epochs, we can improve model performance.

$$\eta_{\min} = 10^{-5},\ \eta_{\max} = 10^{-1},\ n_s = 4\lfloor n/n_{\text{batches}} \rfloor = 1960,\ \texttt{batch\_size} = 100,\ \texttt{cycles} = 3$$

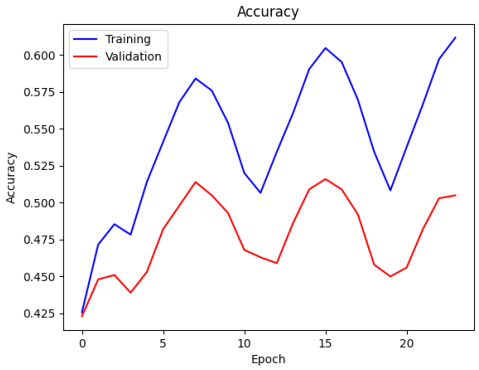
This parameter selection results in 24 epochs, each divided into 490 batches, over a total of 3 learning rate cycles. Loss and cost functions decrease in accordance with the learning rate, creating triangular patterns in proportion to the increasing and decreasing η_t. After training is finished, the model yields an accuracy on the test data set of 52.780%.



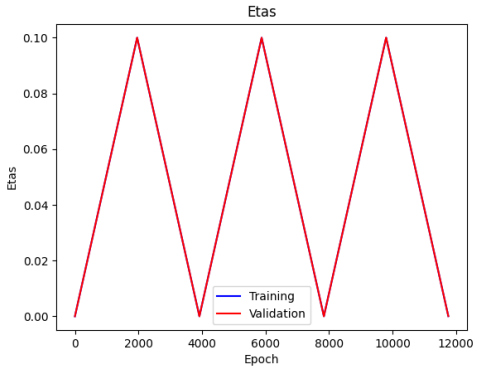
(a) Loss per epoch



(b) Cost per epoch



(c) Accuracy



(d) Eta over 3 cycles

Figure 5: Final model performance over the epochs