DD2424 Deep Learning - Bonus Assignment 2

Hugo Brunlid

17 April 2024

1 Introduction

In this report, we improve on the 2-layer feed forward neural network developed in the main part of the assignment. The first section uses cyclical learning rates, changing hyperparameters and augmenting the data to boost performance. In the second section, we instead use ADAM optimizer to update the weights and biases in the mini-batch gradient descent.

2 Optimize Network Performance

Three of the suggested improvements are implemented, with the results presented below. Baseline final model from the main assignment achieved a testing accuracy of 52.780% with the following parameter settings.

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

2.1 Increasing Number of Hidden Nodes

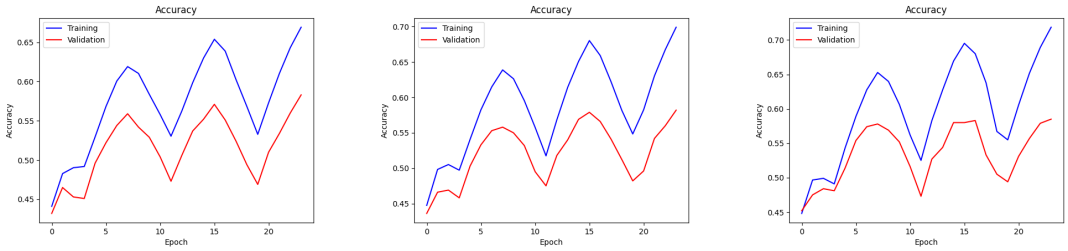
> Explore whether having significantly more hidden nodes improves the final classification rate. One would expect that with more hidden nodes then the amount of regularization would have to increase.

Setting the number of hidden nodes in a neural network is important for determining the model architecture. It is a crucial hyperparameter that directly impacts the network's performance, generalization ability and susceptibility to overfitting or underfitting. Too few hidden nodes may lead to underfitting, where the network fails to capture the underlying structure of the data. Too many hidden nodes may lead to overfitting, where the network adapts to the training data but generalizes poorly to unseen data.

Previously, the network has been composed of 50 hidden nodes. In this section, we increase the number of hidden nodes to evaluate whether a *wide* network has greater generalization ability. Results for a subset of number of hidden nodes are presented in the table. Due to computational limitations, the number of hidden nodes are capped at 200.

Table 1: Test Accuracy for Number of Hidden Nodes

Hidden Nodes	**Test Accuracy**
50	52.50%
100	54.74%
150	55.46%
200	55.61%



(a) 100 Hidden Nodes (b) 150 Hidden Nodes (c) 200 Hidden Nodes

Figure 1: Validation Accuracy when Changing Hidden Nodes

2.2 Applying Dropout

Apply dropout to your training if you have a high number of hidden nodes and you feel you need more regularization.

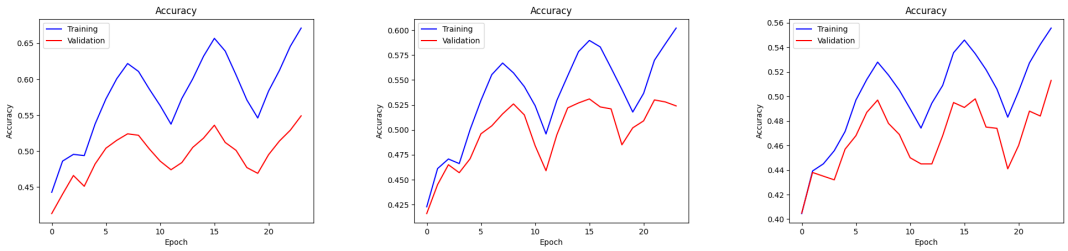
As the number of hidden nodes increases, we see a trend of the validation accuracy deviating from training accuracy, signifying overfitting. Therefore, it might be a good option to increase the amount of regularization. This can be done both through increasing the value of lambda, and utilizing dropout. After testing a few values of lambda, a decision is made to stick with the *best lambda* of ~ 0.025 used previously. Moving forward, we will use 100 hidden nodes as it provides a balance between computational efficiency and marginal gains in test accuracy.

Three configurations of dropout are tested, with the implementation as follows.

1. Compute the first set of activation values $\mathbf{h} = \max(0, W_1\mathbf{x} + \mathbf{b}_1)$
2. Randomly choose which entries to keep using a probability $p \in (0, 1]$

$$\mathbf{u} = (\text{random}(\text{size}(\mathbf{h})) < p$$
$$\mathbf{h}_{\text{drop}} = \mathbf{h}.*\mathbf{u}$$

3. Probabilities after the forward pass, $\mathbf{p} = \text{SoftMax}(W_2\mathbf{h}_{\text{drop}} + \mathbf{b}_2)$



(a) No Dropout, Acc 55.02% (b) 10% Dropout, Acc 52.09% (c) 20% Dropout, Acc 50.70%

Figure 2: Validation Accuracy for Dropout

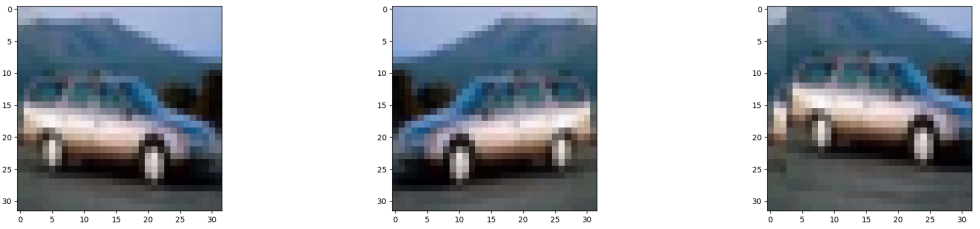
In this case, we note that increasing dropout actually decreases the accuracy on the validation data. This might be due to the regularization becoming too large, or that 100 hidden nodes in a 2-layer network is not sufficiently complex to benefit from regularization by dropout.

2.3 Data Augmentation

Apply data augmentation during training - random mirroring as described in the bonus part of assignment and also random translations.

Another improvement is data augmentation by flipping and translating the images in the data set. By augmenting the training data set, the neural network should become more robust and capable of generalizing better to unseen data. To augment the data set, we combine flipping and translating images.

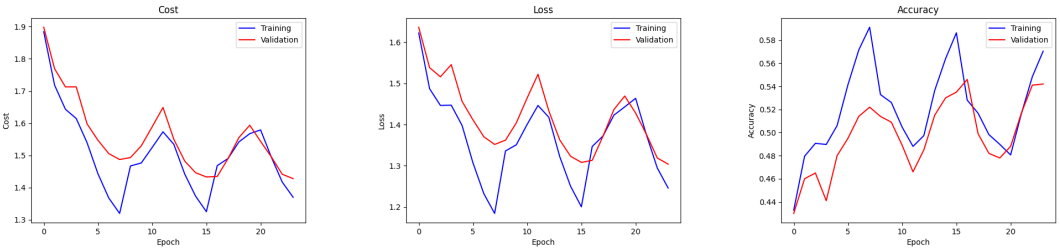
- **Flipping**: An image is flipped horizontally with probability 50% when augmenting.
- **Translating**: Pixels of the image are shifted by (`tx, ty`) in the x and y-axis where (`tx, ty`) are randomly selected from $\{-3, -2, -1, 0, 1, 2, 3\}$. To preserve the original size of the image, the pixels from where the image has shifted are replaced with corresponding pixels from the original image. This might create some distortion along the edges of certain images, but is a reasonable approximation.



(a) Original Image (b) Flipped Image (c) Translated Image

Figure 3: Data Augmentation

$\eta = 10^{-3}$, $\lambda = 0.0025$, `batch_size` $= 100$, `num_epochs` $= 24$, `epoch` $\% \ 4 = 0 \rightarrow$ `augment`



(a) Cost Function (b) Loss Function (c) Accuracy

Figure 4: Results when training using augmented data

We immediately see that the accuracy and cost/loss for the training and validation data is much closer, signifying that overfitting is less of a problem. Due to some distortion appearing when translating the images, we often get worse accuracy on the training data set than the un-augmented validation/test set. Creating a more sophisticated augmentation approach might yield better model generalization results. Test accuracy after training the model using these settings is 53.74%.

3 ADAM Optimizer

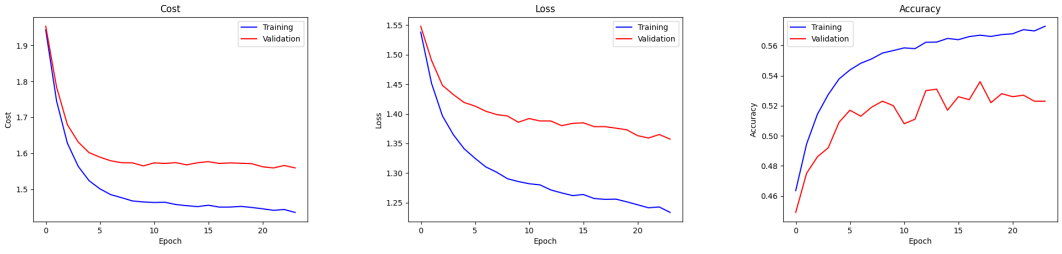
Cyclical learning rates are good for fast training. Implement an Adam optimizer and evaluate whether a 2-layer network could perform just as well or better. Then do some testing (amount of l2 regularization and data-augmentations, have a long training run etc...) to see what level of performance you can achieve by training a wide 2-layer network with data-augmentation and an Adam optimizer.

Adaptive Moment Estimation (ADAM) computes adaptive learning rates for each parameter by storing an *exponential decaying average* of past gradients $\mathbf{m}^{(t)}$, and past squared gradients $\mathbf{v}^{(t)}$. These are used to estimate the mean and variance of the sequence of computed gradients in each dimensions. Variance is then used to dampen/augment the update. Suggested default values of $\beta_1 = .9, \beta_2 = .999, \epsilon = 10^{-8}$ are used.

1. Let $\mathbf{g}_t = \nabla_{\mathbf{x}} f\left(\mathbf{x}^{(t)}\right)$
$$\mathbf{m}^{(t+1)} = \beta_1 \mathbf{m}^{(t)} + (1-\beta_1)\,\mathbf{g}_t$$
$$\mathbf{v}^{(t+1)} = \beta_2 \mathbf{v}^{(t)} + (1-\beta_2)\,\mathbf{g}_t . * \mathbf{g}_t$$
2. Set $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = \mathbf{0}$
3. Both $\mathbf{m}^{(t)}$ and $\mathbf{v}^{(t)}$ are biased towards zero, especially during the initial time-steps. Counter these biases by setting
$$\hat{\mathbf{m}}^{(t+1)} = \frac{\mathbf{m}^{(t+1)}}{1-\beta_1^t}, \quad \hat{\mathbf{v}}^{(t+1)} = \frac{\mathbf{v}^{(t+1)}}{1-\beta_2^t}$$
4. Adam update rule
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(t+1)}} + \epsilon}\hat{\mathbf{m}}^{(t+1)}$$

This method serves as a replacement for using the cyclical learning rates, instead we use a fixed eta of $\eta = 0.001$. Intuitively, instead of getting triangular peaks such as in for cyclical η_t, we get a more smooth convergence of the cost and loss functions. ADAM optimizer is ran for the following configuration, note that λ is doubled to increase regularization.

$$\eta = 10^{-3},\ \lambda = 0.005,\ \texttt{batch\_size} = 700,\ \texttt{num\_epochs} = 24,\ \texttt{adam\_optimizer = True}$$



(a) Cost Function (b) Loss Function (c) Accuracy

Figure 5: Results for ADAM optimizer

In the end, we get an accuracy on the testing data set of 52.06% which provides similar performance to using cyclical learning rates. Varying parameters might yield an additional 1-2% in accuracy for predicting the test data set. But after testing 5-10 configurations, this was the one that performed the best. Notably, the ADAM optimizer was slower to train than when using cyclical learning rates.