DD2424 - Assignment 1

Silpa Soni Nallacheruvu

April 4, 2025

The goal of this project was to implement a simple image classifier for the CIFAR-10 dataset using softmax regression.

The project includes custom implementations for data loading, preprocessing, and one-hot encoding from the CIFAR-10 dataset. Core functions such as `ApplyNetwork`, `ComputeLoss`, and `BackwardPass` handle forward propagation, cross-entropy loss calculation, and gradient computation respectively. Gradient correctness is verified against PyTorch using relative error checks. The `MiniBatchGD` function performs model training with mini-batch gradient descent while tracking loss and cost across epochs. Visualizations of the learned weights and performance metrics help interpret the learning progress of the classifier.

I was able to successfully implement the analytical computation of gradients for the cross-entropy loss with L2 regularization in the `BackwardPass` function. To verify the implementation, I compared my gradients with numerically computed gradients from PyTorch on a small subset of the data (d = 10, n = 3).

The comparison was made using a relative error metric:

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

where g_a is the analytical gradient, g_n is the numerical gradient and ϵ is a small constant to avoid division by zero.

I tested the gradients of both the weights and biases, and the max relative error in W was $\approx$ 1.09e-15 and in b it was $\approx$ 9.78e-17. I was able to conclude that the analytical gradients are correct, as the relative errors are very small and within the acceptable range for numerical precision.

Q1. Loss and Cost Function plots

The loss function is defined as the average negative log-likelihood of the true labels given the predicted probabilities. The cost function includes L2 regularization to prevent overfitting. The loss and cost functions are computed during training and plotted to visualize the model's performance.

Q1.1: Loss function

The plot shows the training and validation loss over 40 epochs of mini-batch gradient descent.

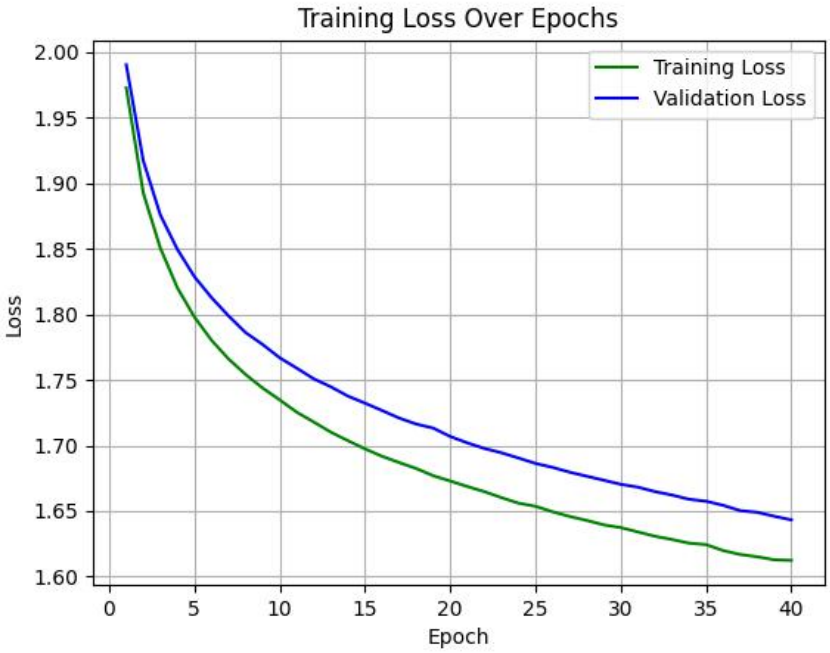

Figure 1: Training and Validation Loss Plot

Both losses decrease steadily, indicating that the model is learning effectively. The training loss remains slightly lower than the validation loss, suggesting mild overfitting but overall good generalization.

Q1.2: Cost function

The plot shows the training and validation cost over 40 epochs of mini-batch gradient descent, including both loss and L2 regularization.

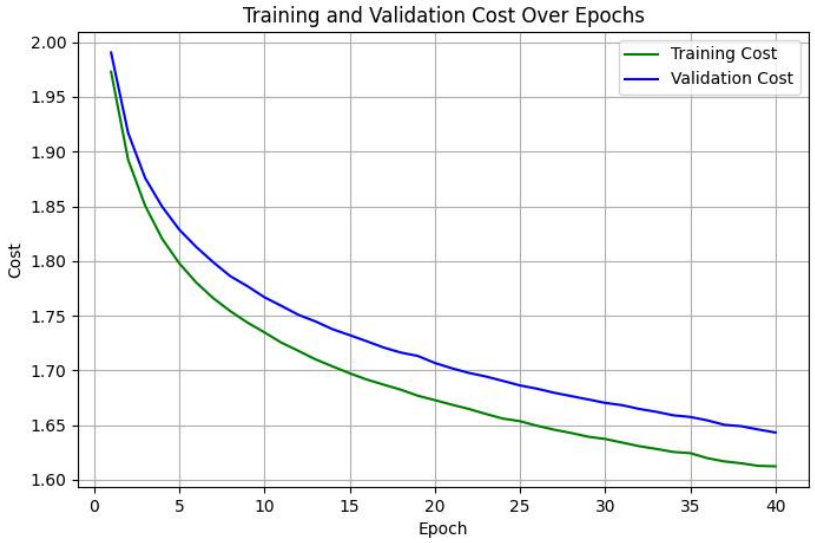

Figure 2: Training and Validation Cost Plot

Both curves decrease consistently, indicating a successful optimization. The validation cost remains slightly higher, reflecting regularization's role in promoting generalization.

Q2. Visualization of Class-Specific Filters (Weight Matrix W)

W is the weight matrix of the softmax classifier, where each column corresponds to a class, and each row corresponds to a pixel in the image. The images are reshaped from the weight matrix and displayed to visualize the features learned by the model.

The images below show the visualization of the learned weight matrix W after training the network for 40 epochs using mini-batch gradient descent. Each square represents the weight filter for one of the 10 CIFAR-10 classes, reshaped into a 32×32×3 RGB image. These templates highlight the patterns the model associates with each class based on the training data, using the following respective parameter settings.

Q2.1: High Learning Rate, No Regularization

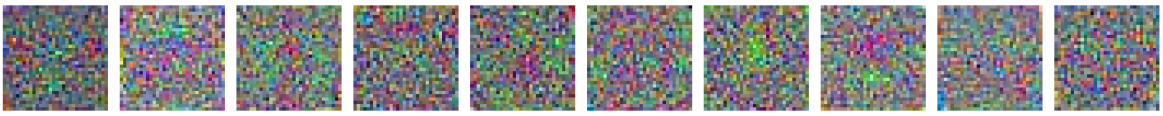

Figure 3: The learnt W matrix visualized with the parameter settings: n_batch=100, eta=.1, n_epochs=40 and lam=0.

After training the network, the training accuracy after the first epoch was 26.72% and the test accuracy was 29.77%. The training accuracy after 40 epochs was 39.48% and the test accuracy was 43.31%. The filters appear highly noisy and lack structure, likely due to the high learning rate causing unstable updates. Despite this, the model learns to separate classes to a reasonable extent, though without clear feature representations.

Q2.2: Lower Learning Rate, No Regularization

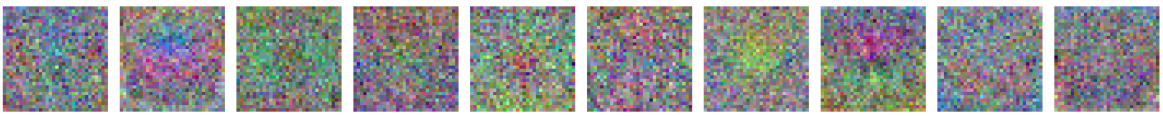

Figure 4: The learnt W matrix visualized with the settings: n_batch=100, eta=.001, n_epochs=40 and lam=0.

After training the network, the training accuracy after the first epoch was 30% and the test accuracy was 30.72%. The final training and test accuracy reached 45.17% and 45.59% respectively. With a smaller learning rate, the filters begin to form smoother and more defined patterns, suggesting more stable and effective convergence without regularization.

Q2.3: Lower Learning Rate with Moderate Regularization

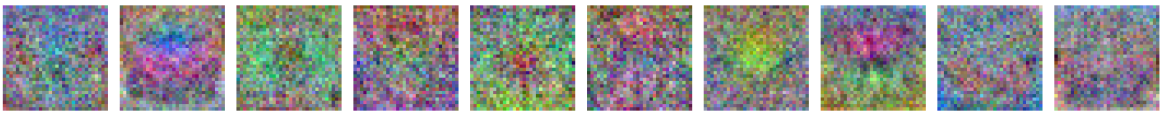

Figure 5: The learnt W matrix visualized with the parameter settings: n_batch=100, eta=.001, n_epochs=40 and lam=0.1

After training the network, the training accuracy after the first epoch was 30.07% and the test accuracy was 30.76%. The final training and test accuracy were 44.87% and 44.72%, respectively. The regularization helps suppress noisy weights, leading to slightly cleaner filters compared to no regularization, which helps generalization without significantly compromising accuracy.

Q2.4: Lower Learning Rate with Strong Regularization

Figure 6: The learnt W matrix visualized with the parameter settings: n_batch=100, eta=.001, n_epochs=40 and lam=1

After training the network, the training accuracy after the first epoch was 30.3% and the test accuracy was 31.01%. The final training accuracy was 40% and test accuracy was 38.99%. The filters become significantly smoother and more structured, suggesting that strong regularization enforces simplicity in the learned weights, though it slightly reduces performance by limiting model capacity.

Observations

Across the four configurations, we observe the impact of varying both the learning rate η and regularization strength λ on the learned weight filters and model performance.

With a high learning rate and no regularization (Q2.1), the filters are highly noisy and unstable, reflecting divergence in the learning process. Reducing the learning rate to $\eta = 0.001$ (Q2.2) leads to more stable training and smoother filters, resulting in the highest test accuracy of 45.59%. Introducing moderate regularization ($\lambda = 0.1$, Q2.3) further de-noises the filters while maintaining similar accuracy, offering a balance between expressiveness and generalization. In contrast, strong regularization ($\lambda = 1$, Q2.4) heavily smooths the filters but slightly reduces the model's performance due to underfitting.

From these results, we see that increasing the regularization parameter λ helps reduce overfitting and improves generalization, as it penalizes overly complex models. However, applying too much regularization can limit the model's capacity to learn, leading to underfitting. In this case, the best performance was observed with $\lambda = 0$, but $\lambda = 0.1$ still achieved nearly equivalent results with cleaner

filters. This highlights the importance of tuning regularization strength to strike the right balance between model complexity and generalization performance.

The learning rate η also plays a crucial role in the training process. It determines the size of parameter updates and thus affects both the speed and stability of learning. A learning rate that is too large $\eta = 0.1$ may cause the model to diverge or oscillate, while a rate that is too small can lead to excessively slow convergence or getting stuck in suboptimal regions. In this experiment, $\eta = 0.001$ proved to be effective in allowing the model to learn steadily across epochs. These results reinforce the importance of carefully selecting both the learning rate and regularization parameter to achieve optimal training outcomes.