Ben-Gurion University of the Negev
Faculty of Engineering Sciences
Department of Software and Information systems Engineering

Deep Learning
Assignment 1

Sol Amara
Ido Gurevich

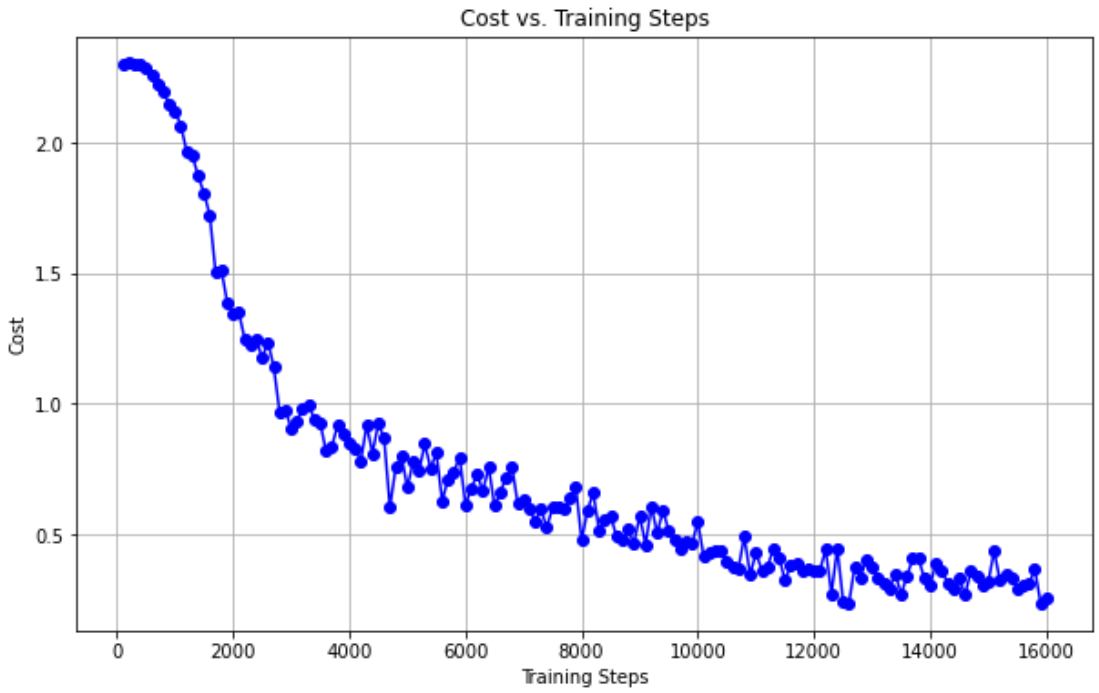
Section 4: Classify the MNIST dataset

The data was divided into 80% training and 20% test sets, with 20% of the training data further used for validation. The model was trained until the improvement in validation cost was less than 0.0001 for 100 training steps. Training required 15900 training steps with 188 epochs, batch size of 256 (meaning $\frac{15900}{188}$ iteration). The training stops when cost reached to 0.3303.

The final accuracy values for the train are: 0.9136.
The final accuracy values for the validation are: 0.9098.
The final accuracy values for the test are: 0.9092.

These results indicating that the model generalized well with minimal overfitting, as evidenced by the close performance metrics across training, validation, and test sets. Additionally, the high accuracy demonstrates the model's success in accurately classifying the different types of images in the MNIST dataset.

The costs values:



This graph shows how the model's performance changes over training. Notably, the cost value decreases consistently in increments of 100 training steps, ultimately stabilizing around the 0.3 mark in the final iterations.

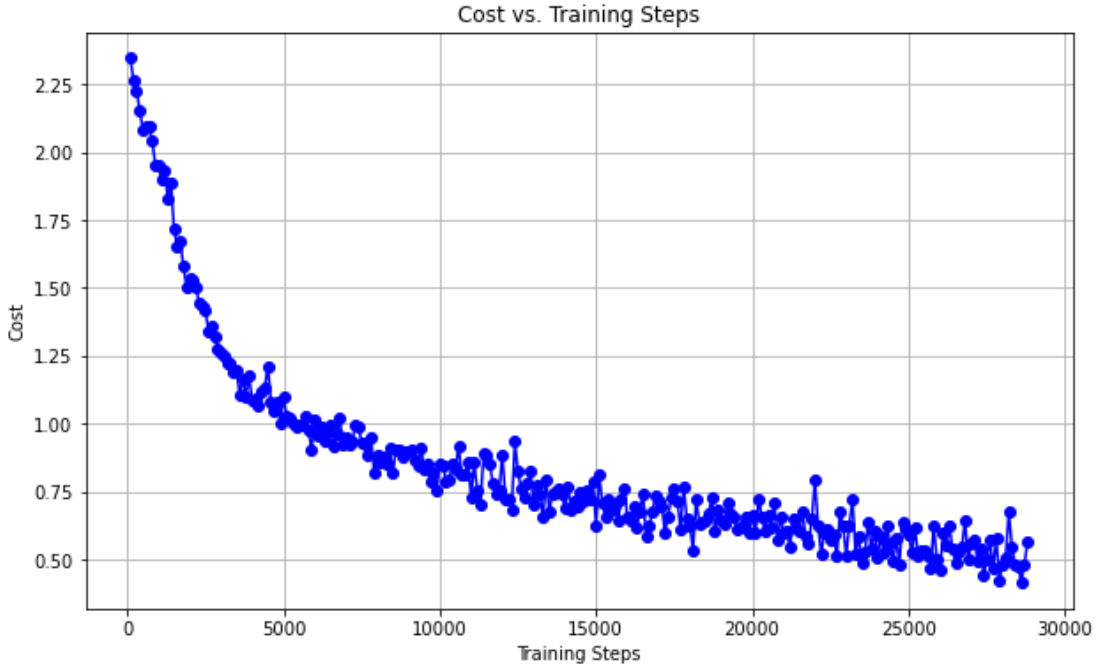
Section 5: Classify the MNIST dataset with batchnorm

In this section we turn the batchnorm flag into 'True'. The `apply_batchnorm` function normalizes the activation values of a given layer using batch normalization. It takes as input the activation values A, calculates their mean and variance, and then normalizes them using these statistics and a small epsilon value to prevent division by zero. The function returns the normalized activation values.

The model was trained until the improvement in validation cost was less than 0.0001 for 100 training steps. Training required 28700 training steps with 188 epochs, batch size of 256 (meaning $\frac{28700}{188}$ iteration).

The final accuracy values for the train are: 0.8581.
The final accuracy values for the validation are: 0.8568.
The final accuracy values for the test are: 0.8578.

The costs values:



Despite incorporating batch normalization into the model architecture, there was no noticeable improvement in performance. Even with larger number of training steps, the accuracy percentages were lower than previous results. This could be because normalization forces the parameters to be adjusted by the mean and standard deviation, potentially leading to a loss of valuable data. Additionally, the dataset might be inherently noisy or complex, making it difficult for the initial layers of the model to learn meaningful representations. When parameters vary significantly, normalization can exacerbate this issue, resulting in worse performance.

Section 6: Classify the MNIST dataset using L2 norm

The L2 norm is added to the loss function to penalize large weights, which helps prevent overfitting. For any weight that is not zero there will be a penalty, meaning the model will have an interest in using small weights.

The modifications were in the L_layer_model function and include:

1. Adding the penalty to the costs according to the formula:

λ is the regularization strength, m is the number of examples in the batch

$$L2_{cost} = \frac{\lambda}{2m} \cdot \sum_{l=1}^{L} \|W^l\|^2$$

In the code: (we choose $\lambda = 1$)

```
# Adding the L2 regularization term to the cost
L2_cost = (1 / (2 * X_batch.shape[1])) * sum([np.sum(np.square(parameters['W' + str(l)])) for l in range(1, len(layers_dims))])
cost = cost + L2_cost
```

2. Changing the gradient:

   The gradient calculation is performed based on the loss function with respect to each weight in the network. Consequently, changes in the cost will result in corresponding adjustments to the gradients that must be computed. Since the changes of the L2 normalization are added to the cost function, the derivative of the function needs to be included in the gradients.

   In the code:

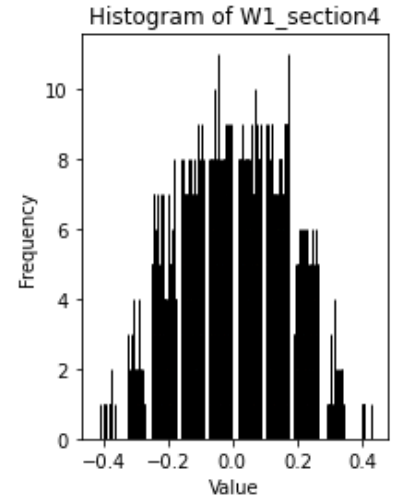
```
# Adding L2 cost to gradients
for l in range(1, len(layers_dims)):
    grads['dW' + str(l)] += (1 / X_batch.shape[1]) * parameters['W' + str(l)]
```

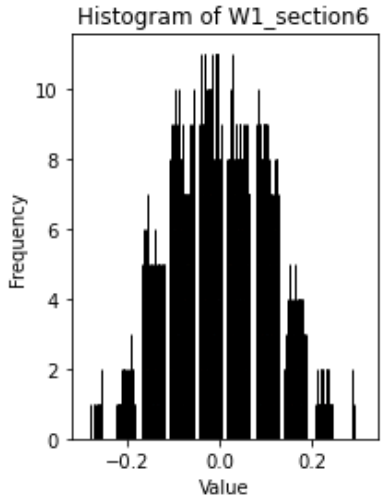
The model was trained until the improvement in validation cost was less than 0.0001 for 100 training steps. Training required 13400 training steps with 188 epochs, batch size of 256 (meaning $\frac{13400}{188}$ iteration). The training stops when cost reached to 0.4211.

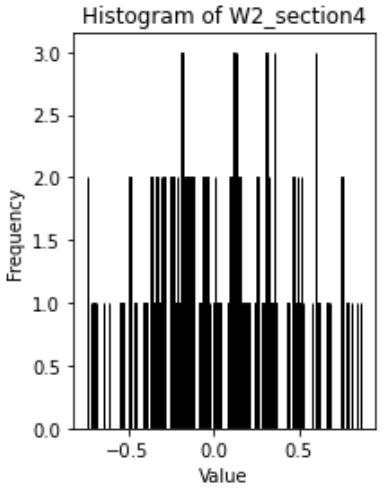
The final accuracy values for the train are: 0.8854.
The final accuracy values for the validation are: 0.8854.
The final accuracy values for the test are: 0.8877.

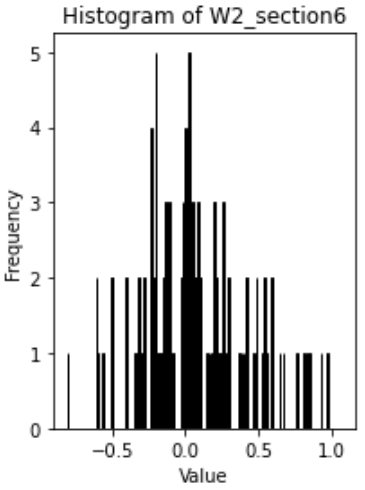
The cost values:

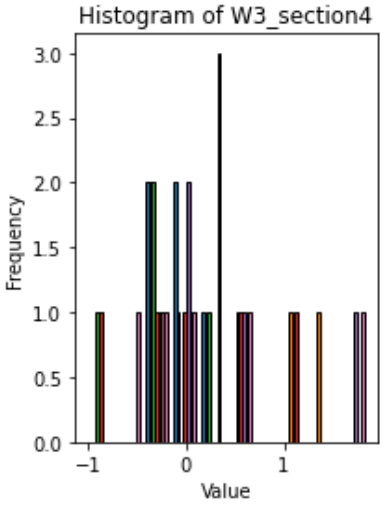
It can be seen that the results we achieved using L2 are good. Although the accuracy is lower than what we reached in the previous sections, the number of training steps is also lower. As we have learned, the use of L2 is intended to prevent overfitting and to produce a general model, even if there is a slight impairment in performance.

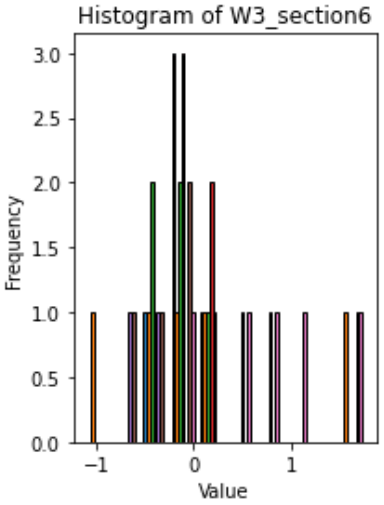


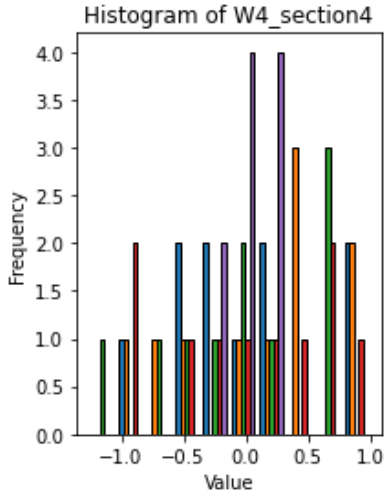


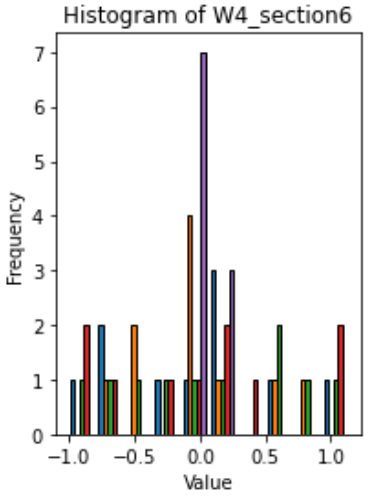












When comparing the weight values in our architecture with normalization (section 6) to those without normalization (section 4), we observe that the values in section 4 are more dispersed and lack a clear distribution pattern. Conversely, in section 6, the weight values follow a normal distribution, indicating that the normalization was executed correctly.