**Deep Learning Assignment 1: MNIST Classification Using a Neural Network from Scratch**

**By: Kfir Nissim (206559585) and Yonatan Baruch(\_\_\_\_\_\_\_\_\_\_)**

**Introduction**

This report summarizes the implementation and evaluation of a fully connected neural network trained on the MNIST dataset. The network was built from scratch using NumPy, following all assignment requirements, including forward and backward propagation, batch normalization, mini-batch training, categorical cross-entropy loss, and optional L2 regularization. The experiments were conducted in four configurations:

1. Basic network (no batch normalization)
2. Network with batch normalization
3. Network with L2 regularization and no batch normalization
4. Network with batch normalization and L2 regularization

Each experiment was evaluated based on training/validation/test accuracy, cost over training steps, and convergence behavior.

**Implementation Overview**

All required functions were implemented using vectorized NumPy operations. The neural network consists of multiple layers with ReLU activation, followed by a softmax output layer. Below is a summary of the key functions:

* initialize\_parameters(layer\_dims) – Initializes the weights with small random values and biases with zeros for each layer.
* linear\_forward(A, W, b) – Computes the linear transformation , storing intermediate values in a cache.
* relu(Z) – Applies the ReLU activation function, returning the output and a cache of for backpropagation.
* softmax(Z) – Computes the softmax function along each column of , returning the activations and caching the raw input .
* linear\_activation\_forward(A\_prev, W, b, activation) – Performs the linear transformation followed by the selected activation function ('relu' or 'softmax'), and returns the result with cache.
* l\_model\_forward(X, parameters, use\_batchnorm) – Iteratively applies linear\_activation\_forward across all layers. Batch normalization is optionally applied to the activations after each ReLU.
* apply\_batchnorm(A) – Performs batch normalization on activations , using mean and variance along the batch axis.
* compute\_cost(AL, Y, parameters=None, lambd=0.0) – Calculates the categorical cross-entropy cost. When lambd is greater than zero, adds an L2 regularization penalty based on all weight matrices.
* linear\_backward(dZ, cache) – Computes gradients , , and for the linear part of the layer during backpropagation.
* relu\_backward(dA, activation\_cache) – Computes for ReLU using the cached .
* softmax\_backward(dA, activation\_cache) – Computes for softmax, using the true labels stored in the cache.
* linear\_activation\_backward(dA, cache, activation) – Performs backpropagation through a full layer, computing gradients based on the chosen activation function.
* l\_model\_backward(AL, Y, caches) – Performs full backpropagation across all layers using stored caches.
* update\_parameters(parameters, grads, learning\_rate, lambd=0.0) – Updates parameters using gradient descent. Adds L2 regularization gradient component if lambd > 0.
* predict(X, Y, parameters, use\_batchnorm=False) – Computes forward pass on input X, returns accuracy against Y. Includes control over batch normalization via the use\_batchnorm flag.

*Note:* Throughout training, each iteration refers to a **single mini-batch**, not a full epoch. This detail is consistent with how training steps were counted in practice.

**Difference between Training Implementations:** Two versions of the l\_layer\_model function were used:

**Basic version (per assignment instructions, Section 3):**

l\_layer\_model(X, Y, layers\_dims, learning\_rate, num\_iterations, batch\_size, use\_batchnorm=False)

This implementation trains the model for a fixed number of iterations. It computes cost every 100 iterations averaged over all mini-batches in an epoch, without any validation logic or early stopping.

**Extended version (used in experiments):**

l\_layer\_model(X, Y, X\_val, Y\_val, layers\_dims, learning\_rate, max\_training\_steps, batch\_size, max\_no\_improve\_steps=100, use\_batchnorm=False)

* Early stopping based on validation cost improvement
* Separate validation set evaluation
* Cost collection every 100 steps (not per epoch)
* Step-wise control rather than epoch-based training

This approach allowed for finer-grained control and reproducibility in experiments. Although not explicitly requested in the assignment, it enabled better evaluation and monitoring.

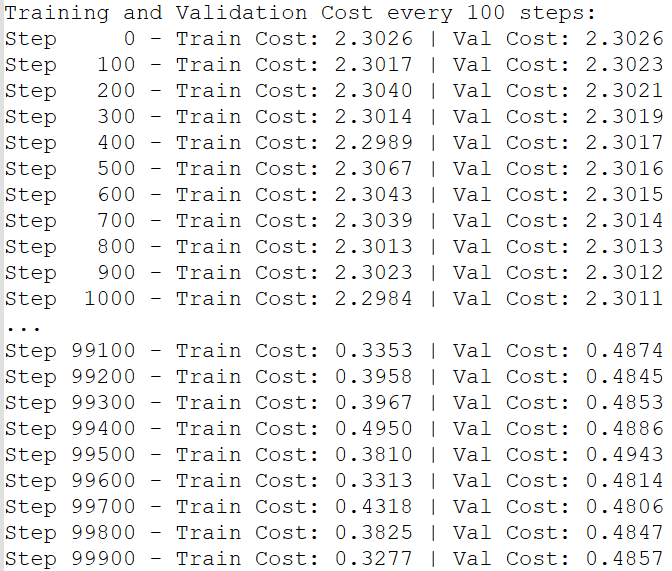
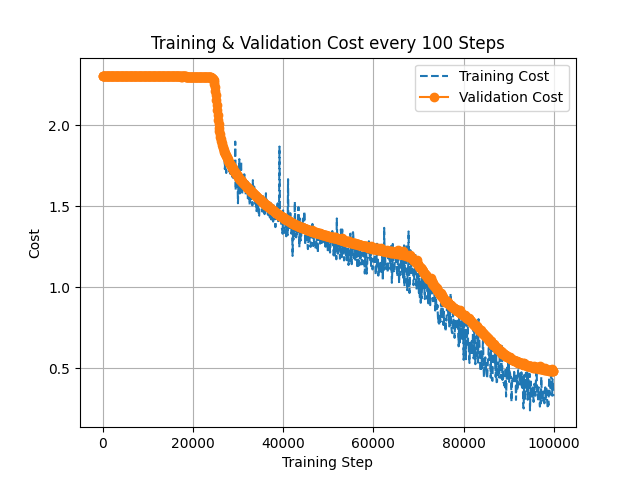
Additionally, both l\_layer\_model and predict accept a use\_batchnorm flag, allowing toggling batch normalization externally. This design choice, while not specified in the instructions, was added for flexibility and ease of experimentation.

**Experiment Settings**

* Dataset: MNIST (60,000 training, 10,000 test samples)
* Input shape: 784 (28x28 flattened grayscale images)
* Network architecture: [784, 20, 7, 5, 10]
* Activation: ReLU in hidden layers, softmax in output layer
* Loss: Categorical cross-entropy
* Optimizer: Gradient Descent
* Batch size: 128
* Learning rate: 0.009
* Early stopping: Stop after 100 consecutive steps without validation improvement

**Experiment 1 – Without Batch Normalization**

* **Total training steps:** 100,000
* **Best validation cost:** 0.4806
* **Final accuracy:**
  + Train: 86.17%
  + Validation: 83.71%
  + Test: 84.13%

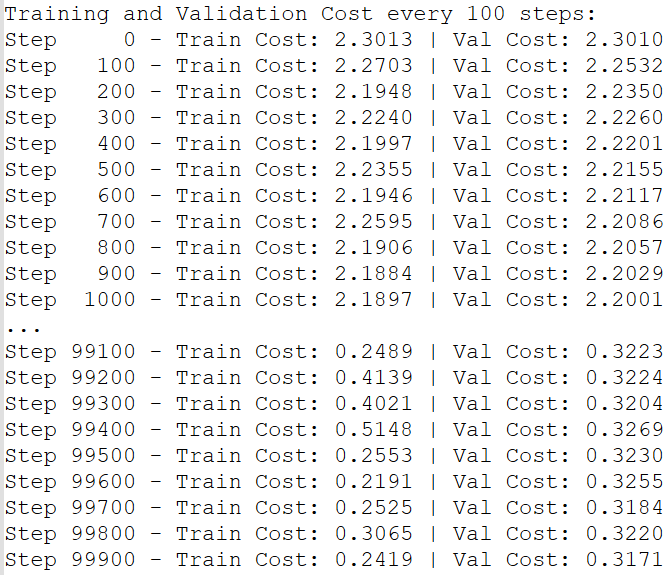
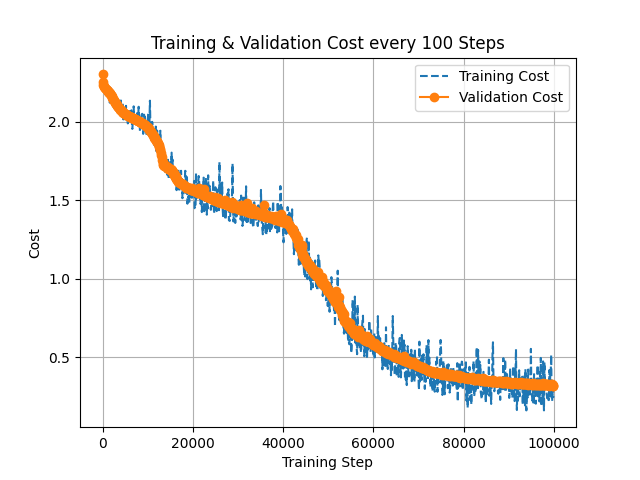
****

Observations:

* Slow convergence during the first 20,000 steps
* Moderate overfitting avoided
* Validation cost remains higher and decays slowly

**Experiment 2 – With Batch Normalization**

* **Total training steps:** 100,000
* **Best validation cost:** 0.3171
* **Final accuracy:**
  + Train: 93.47%
  + Validation: 92.02%
  + Test: 92.03%

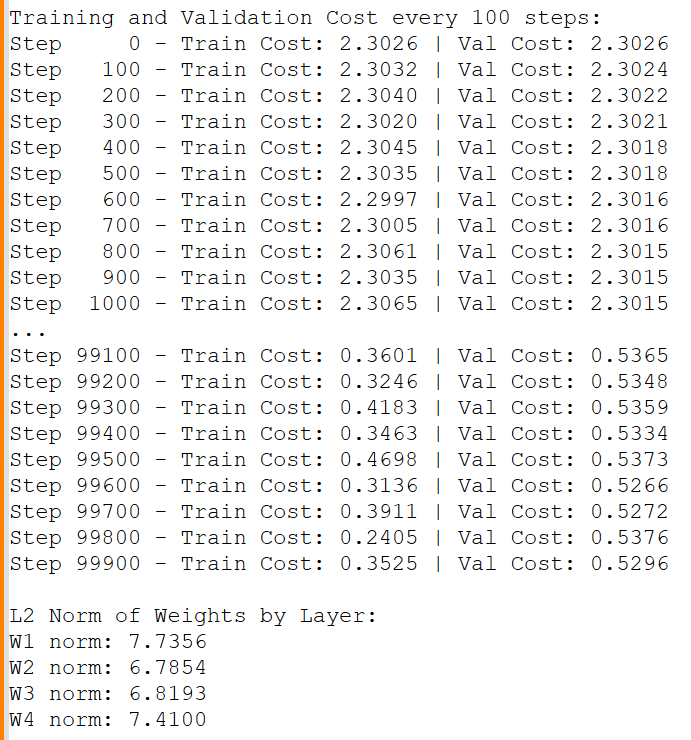
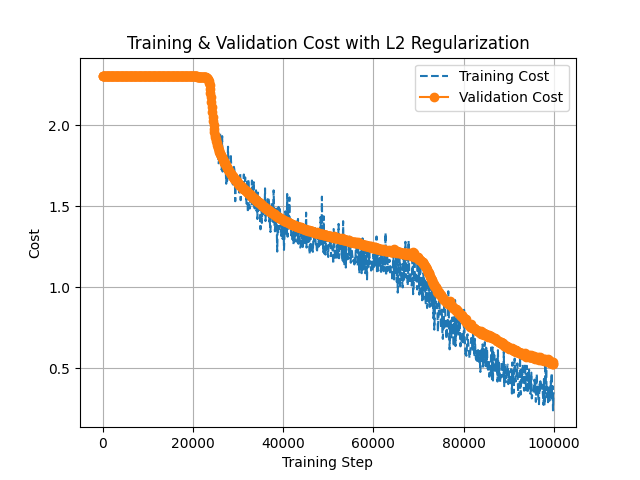
 

Observations*:*

* Significant improvement in convergence speed and final accuracy
* Stable learning curve with reduced variance in training cost

**Experiment 3 – With L2 Regularization Only (No BatchNorm)**

* **Total training steps:** 100,000
* **Best validation cost:** 0.5266
* **Final accuracy:**
  + Train: 86.55%
  + Validation: 83.33%
  + Test: 83.88%

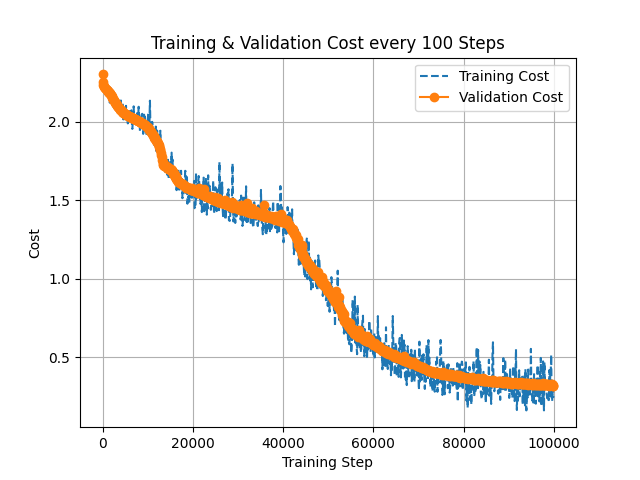
 

Observations*:*

* Performance was similar to the baseline (no BN, no L2)
* L2 regularization alone did not provide sufficient benefit in the absence of batch normalization

**Experiment 4 – With BatchNorm and L2 Regularization**

* **Total training steps:** 100,000
* **Best validation cost:** 0.2955
* **Final accuracy:**
  + Train: 94.47%
  + Validation: 92.65%
  + Test: 92.53%

Observations:

* Achieved the best results across all metrics
* L2 penalty contributed to weight regularization and generalization

**Comparative Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Configuration** | **Train Acc** | **Val Acc** | **Test Acc** | **Best Val Cost** |
| No BatchNorm | 86.17% | 83.71% | 84.13% | 0.4806 |
| BatchNorm | 93.47% | 92.02% | 92.03% | 0.3171 |
| L2 Only (No BatchNorm) | 86.55% | 83.33% | 83.88% | 0.5266 |
| BatchNorm + L2 | 94.47% | 92.65% | 92.53% | 0.2955 |

Batch normalization significantly improved convergence speed and final accuracy. Adding L2 regularization further refined the model, reducing overfitting and yielding superior validation and test performance. L2 regularization alone, however, was not sufficient to meaningfully improve results in the absence of normalization.

**Conclusion**

The experiments demonstrate that incorporating batch normalization and L2 regularization significantly enhances training dynamics and generalization of deep neural networks. The final architecture with both techniques achieved the best results and converged more efficiently than the baseline model.