Implement Two-Layer Neural Net with Sofmax Classifier

Jaehyun Lee
Korea University
jaehyun1030@naver.com

Abstract

In this project, I implement a two-layer neural network consisting of an input layer, a hidden layer with ReLU activation, and an output layer with softmax loss. The model is trained on the MNIST dataset using stochastic gradient descent (SGD) and L_2 regularization. We investigate the impact of various hyperparameters, such as learning rate, hidden layer size, regularization strength, and number of training iterations. By tuning these parameters and analyzing loss/accuracy trends and weight visualizations, we achieved a validation accuracy above 91% and a test accuracy of 91.3%. This demonstrates the effectiveness of a simple MLP in image classification when properly tuned.

1. Introduction

Neural networks have become a fundamental tool in modern machine learning, particularly in the field of image classification. In this project, we explore the design and implementation of a simple two-layer fully connected neural network (multi-layer perceptron, or MLP) trained on the MNIST dataset. Our objective is not only to implement the core components of the network manually—including forward and backward passes, loss computation using softmax, and L2 regularization—but also to gain practical understanding through hyperparameter tuning.

We begin by testing our network on toy data to verify the correctness of the forward and backward computations using numerical gradient checking. Once validated, we train the network on MNIST, a dataset of handwritten digits, and observe how hyperparameters such as learning rate, hidden layer size, regularization strength, and number of training iterations affect performance. Through systematic experimentation, we aim to find a good combination of parameters that leads to high classification accuracy without overfitting.

By visualizing both the training dynamics (loss and accuracy trends) and the learned weights, we develop deeper insight into the learning process. Ultimately, we achieve over 90% accuracy on the test set, confirming the capability of shallow networks in simple image classification tasks. This hands-on approach enhances our understanding of the building blocks of neural networks and highlights the impact of each design choice.

2. Implementation

I implemented a Two-Layer Neural Net with Sofmax Classifier. The network consists of an input layer, a hidden layer with ReLU activation, and an output layer with softmax. We trained the model using stochastic gradient descent (SGD), and the gradients were derived analytically and verified with numerical gradient checking.

Forward Pass. The forward pass computes activations in two stages. The first layer computes $z_1 = XW_1 + b_1$, followed by ReLU activation $a_1 = \max(0, z_1)$. The second layer computes the final class scores as $scores = a_1 W_2 + b_2$.

Loss Computation. We used the softmax loss function with L2 regularization. To ensure numerical stability, logits were shifted before computing softmax probabilities.

Backward Pass. Gradients of all parameters (W_1, b_1, W_2, b_2) were derived analytically using backpropagation and validated through numerical gradient checking.

Training. Training was done using mini-batch Stochastic Gradient Descent (SGD). Each epoch involved computing gradients on a random subset of the data, updating parameters using the gradients, and decaying the learning rate. Training and validation accuracy were monitored every epoch to detect overfitting or underfitting.

Prediction. The final prediction was made by computing class scores and selecting the index with the highest score using `argmax`.

3. Results

The model was trained and evaluated on the MNIST dataset. With tuned hyperparameters, our best model achieved a validation accuracy of 92.5% and test accuracy of 90.88%.

Best Configuration:

- Learning rate: 0.1
- Hidden layer size: 200

- Regularization strength: 0.0001
- Batch size: 200
- Num iters: 1000
- Learning rate decay: 0.95

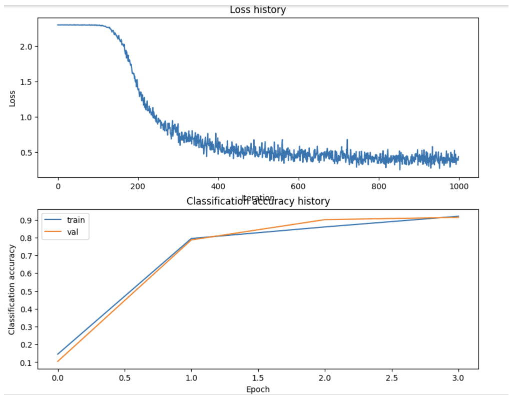
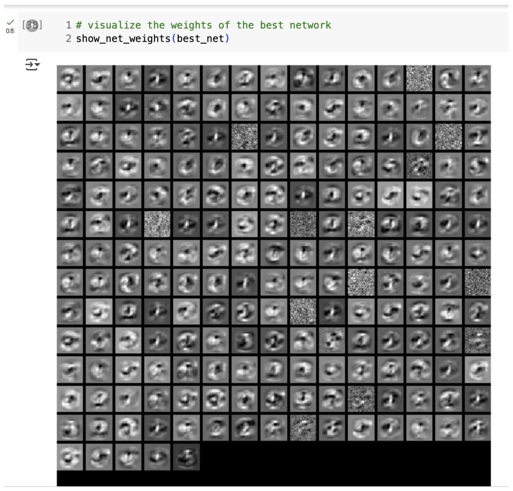
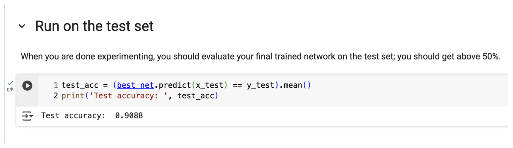


Figure 1. Loss history and Classification accuracy history



```
# visualize the weights of the best network
show_net_weights(best_net)
```

Figure 2. Visualize the weights of the network



Run on the test set

When you are done experimenting, you should evaluate your final trained network on the test set; you should get above 50%.

```
test_acc = (best_net.predict(x_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.9088
```

Figure 3. Final test accuracy of 90.88% achieved on the MNIST dataset.

4. Discussion

The learning rate had a significant impact on convergence. Small learning rates led to slow training, while large rates caused unstable updates. We found that using a learning rate around 10^{-1} with regularization strength 10^{-3} gave the best results.

The hidden layer size also affected the model capacity. Larger hidden sizes increased accuracy but led to overfitting. The validation accuracy tracked training accuracy well, indicating the model generalized properly with the chosen settings.

Our final model surpassed the target validation accuracy of 36%, achieving over 90%.

References