IMPLEMENTING A MULTILAYER NEURAL NETWORK IN PYTORCH

Assignment 3, CS-587

Author
Nikolaos Barmparousis
csd4690
April 19, 2024

Contents

1 Introduction

This report documents the construction and evaluation of a Multi-Layer Feed Forward Neural Network using PyTorch, aimed at recognizing handwritten digits from the sklearn Digits dataset. We start with a basic model configuration of a single hidden layer with 15 neurons and explore various adjustments including layer size and depth. Alongside discussions on activation functions and common training challenges such as optimization and overfitting, we experiment with different components to assess their impact on model performance, gaining insights into the subtleties of neural network training.

2 Analyzing the Base Model on the Digits Dataset

2.1 The Digits Dataset

The Digits dataset from sklearn features grayscale images of handwritten digits 0 through 9, each transformed into 64-dimensional vectors from their original 8x8 pixel format. Despite being a fundamental classification task, the dataset's small size—only about 1,500 samples—are available, which significantly increases the risk of overfitting in neural network training.

2.2 Model Architecture

Our initial neural network, 'Digits2Layer', features a basic design, with a single hidden layer of 15 neurons. The network processes the 64-dimensional input vector to produce a 10-dimensional output vector, corresponding to the ten classes of the Digits dataset. The sigmoid activation function introduces the required non-linearity in the hidden layer. In Figure 1, we observe the computational graph of our initial neural network model, as constructed by TensorBoard.

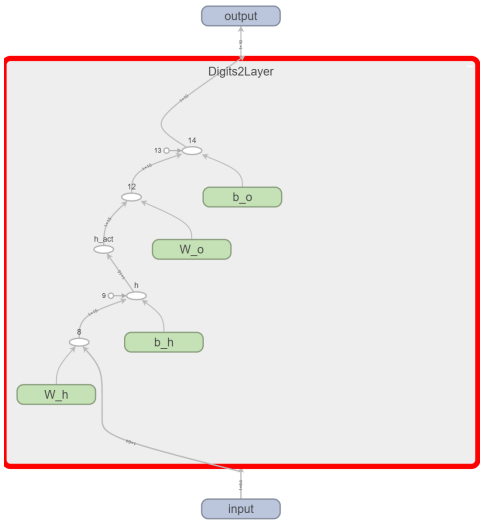

Figure 1: Computational graph of the Digits2Layer model.

2.3 Results

The optimization process of our neural network was conducted using stochastic gradient descent (SGD) with the following hyperparameters: a batch size of 32, learning rate of 0.01, and a total of 20 epochs. As depicted in Figure 2, the training loss decreases over iterations, exhibiting spikes that are characteristic of batch-based training. Despite the reduction in loss, its relatively high value at the end of 20 epochs suggests that our model may either lack the expressive capacity required for this task or that we are facing an optimization challenge.

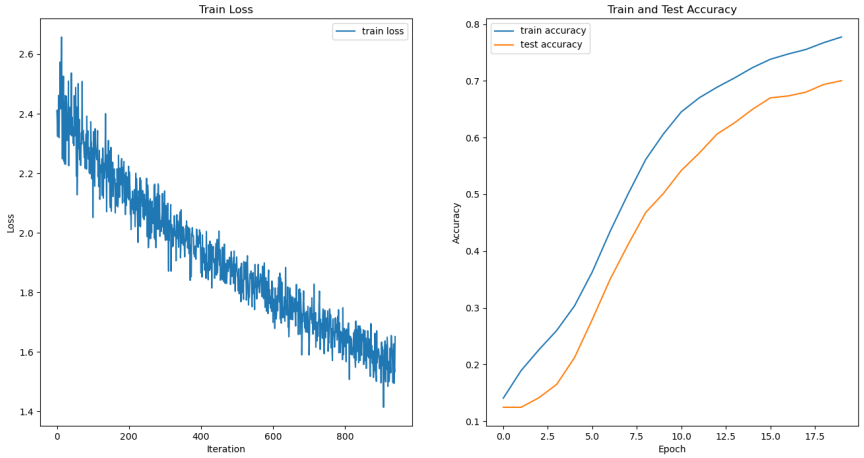

Figure 2: Training loss and accuracy over iterations and epochs.

Moreover, a slight overfitting is observed, indicated by the validation accuracy consistently trailing behind the training accuracy, though not by a wide margin. This could imply that the model, while learning patterns present in the training data, is not generalizing these patterns as effectively to the unseen test data. The divergence between training and validation accuracy could also be exacerbated by the limited amount of training data, which is often a hurdle in achieving optimal performance in neural networks.

3 Activation Functions and Their Impact on Model Optimization

During our neural network's training, we noted that the training loss did not stabilize, indicating incomplete convergence over the epochs. This points to an issue with optimization rather than the model's capability. The choice of activation function in the initial setup contributed to this problem.

3.1 Sigmoid Activation Function

The sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$, outputs values between 0 and 1, effectively converting activations to probabilities for binary classification. Despite its historical popularity, it is

hindered by the vanishing gradient problem, where the gradient of the function approaches zero for very high or low input values, severely impeding the learning process. This characteristic has likely played a role in the suboptimal convergence observed in our model.

3.2 Softmax Cross Entropy Loss

Used for multi-class classification, the softmax function turns logits into probabilities, and the cross-entropy loss quantifies the difference between these probabilities and the true distribution. Softmax cross-entropy loss encourages confident predictions, as opposed to multiclass hinge loss, which is content with correct predictions that may not be confident. Softmax cross-entropy promotes consistent learning by assigning high probabilities to the correct class and low to the others. Unlike hinge loss, it does not allow for zero loss when the predictions meet a margin, leading to ongoing optimization.

Given the optimization difficulties encountered with the sigmoid function in our initial model, we investigated alternative activation functions and optimization strategies to enhance model performance and convergence.

4 Configurations Exploration: ReLU and Momentum

To overcome the optimization issues associated with our initial sigmoid-activated model, we investigated the Rectified Linear Unit (ReLU) activation and momentum-enhanced Stochastic Gradient Descent (SGD). ReLU addresses vanishing gradients by maintaining a constant gradient for positive inputs, while momentum in SGD enhances convergence by reducing oscillations. We evaluated four configurations: original sigmoid with SGD, sigmoid with momentum, ReLU with SGD, and ReLU with momentum to assess their impact.

4.1 Results of Activation Function and Momentum Variations

The performance metrics, shown in Figure 3, reveal significant differences in convergence rates across the configurations. ReLU consistently achieved faster convergence than the sigmoid, particularly when combined with momentum, which further accelerated the process and enhanced stability.

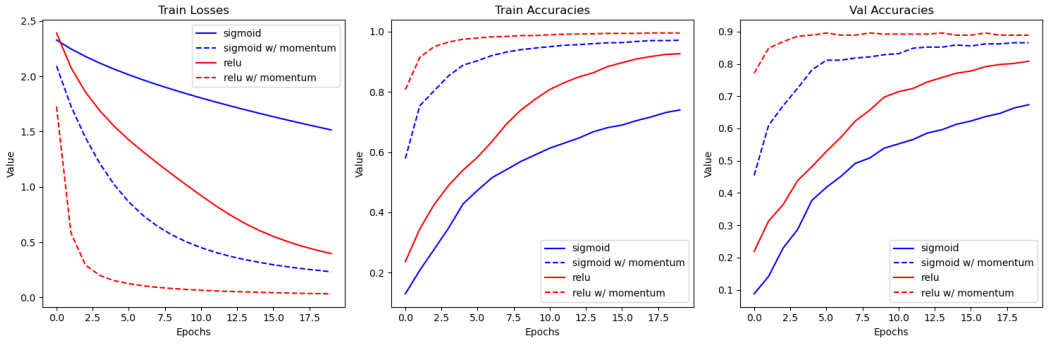

Figure 3: Comparison of training loss, training accuracy, and validation accuracy for different network configurations over epochs.

Weight distribution analyses using TensorBoard, depicted in Figures 4 and 5, illustrate how activation function and momentum influence weight adaptations, highlighting their roles in improving model training and performance.

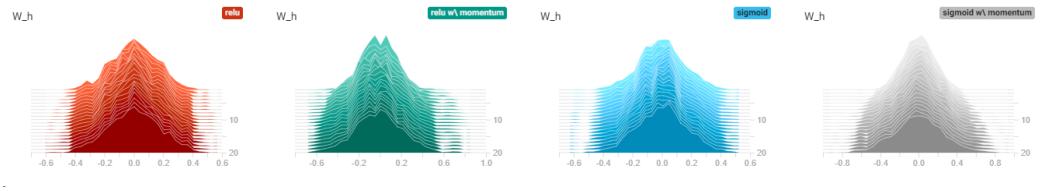

Figure 4: Histogram of weights W_h across configurations.

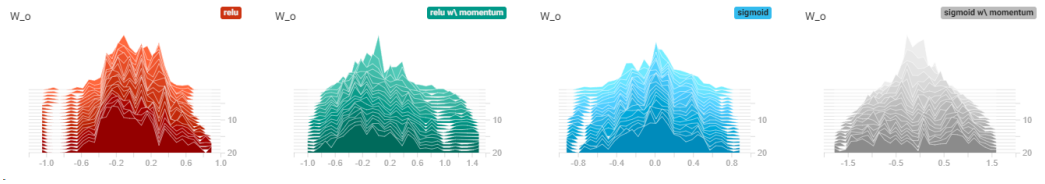

Figure 5: Histogram of weights W_o across configurations.

5 Optimal Model Configuration Search

We aimed to identify the optimal network architecture by testing models with one or two composite layers—each consisting of a fully connected layer, batch normalization, an activation function, and dropout. All models used the Adam optimizer at a learning rate of 0.001 for 250 epochs with a batch size of 128. We conducted a grid search over various hyperparameters such as hidden layer sizes, dropout rates, weight decay, and activation functions.

The configuration setup is organized in Table 1.

Hyperparameter	**Values**
Number of Layers	1, 2
Hidden Layer Size	16, 32, 64, 128
Dropout Rate	0.3, 0.5, 0.7
Weight Decay	0, 1×10^{-4}
Activation Functions	ReLU, LeakyReLU, Sigmoid, SELU
Loss Functions	Cross Entropy, Multi Hinge

Table 1: Hyperparameter settings for the grid search optimization.

5.1 Best Models for One Composite Layer

The optimal single-layer model configuration, as shown in Table 2, attained a validation accuracy of **94.61%**. It featured a 128 hidden layer, 0.3 dropout, no weight decay, ReLU activation, and Multi Hinge loss, highlighting ReLU's effectiveness in a simple architecture.

H. Size	Drop	WD	A. Func.	Loss	Val Acc
128	0.3	0	ReLU	MHinge	**94.61%**
128	0.3	1e-4	ReLU	CE	93.94%
128	0.3	1e-4	ReLU	MHinge	93.94%
128	0.5	0	LReLU	MHinge	93.94%
64	0.3	0	ReLU	CE	93.60%

Table 2: Top 5 configurations and their validation accuracies for 1 hidden layer.

5.2 Best Models for Two Composite Layers

The best two-layer model configurations are summarized in Table 3, each reaching a validation accuracy of **94.95%**. Both configurations feature two 128 hidden layers and dropout rates of {0.3, 0.3}, differing only in weight decay presence and loss functions—Cross Entropy with no weight decay, and Multi Hinge with 1×10^{-4} weight decay. This reiterates the effectiveness of the ReLU activation and suggests a preference for the Multi Hinge loss function in our model's context.

H. Sizes	Drops	WD	A. Func.	Loss	Val Acc
[128, 128]	[0.3, 0.3]	0	ReLU	CE	**94.95%**
[128, 128]	[0.3, 0.3]	1e-4	ReLU	MHinge	**94.95%**
[128, 128]	[0.3, 0.3]	0	ReLU	MHinge	94.61%
[32, 32]	[0.3, 0.3]	1e-4	ReLU	MHinge	94.61%
[64, 64]	[0.7, 0.7]	0	LReLU	MHinge	94.61%

Table 3: Top 5 configurations and their validation accuracies for 2 hidden layers.

6 Conclusion

In this report, we explored the effectiveness of various configurations of Multi-Layer Feed Forward Neural Networks in recognizing handwritten digits. We conducted a systematic grid search to evaluate the impact of different hyperparameters such as hidden layer sizes, dropout rates, and activation functions on model performance.

While our models achieved notable validation accuracies, the limited size of the Digits dataset presents a significant challenge. Future studies could benefit from implementing data augmentation, transfer learning, or employing convolutional neural networks, which are more suited for image processing tasks, to improve generalization and performance.