An Examination of Backpropagation

Zachary Johnston
North Carolina State University
ztjohnst@ncsu.edu

Michael Patel
North Carolina State University
mrpatel5@ncsu.edu

John Ravi
North Carolina State University
jjravi@ncsu.edu

***Abstract*—In the era of Big Data, deep learning has found new life. With new and exciting applications in digital imaging and natural language processing, the field of deep learning has revolutionized data visualization capabilities. Beneath the hood of all this excitement in hidden layers and multi-layer perceptrons is backpropagation. Backpropagation is used to find gradients necessary for model training. In fact, backpropagation can be viewed as the workhorse behind deep learning. This report outlines context of forward and backwards propagation, the derivation of backpropagation to our specific neural network architecture, and some discussion on hyperparameters to improve model generalization.**

I. Introduction

The goal of deep learning is to discover hierarchical models that accurately represent data. The models seek to internalize the essence of data, which could be images, audio, etc. With the success of backpropagation, deep learning models have flourished in the realm of classification.

Image classification involves the task of determining what class any given image belongs to. People, even young children, are able to perform this task quite easily, but computer models may not fair as much. For example, suppose an image classifier was presented with an image of an animal (a dog), and asked to determine whether it was a dog or cat. Humans could be quite certain that the image was a dog, but the classifier may only predict dog with 78% confidence.

II. Background

A. Derivation of Backpropagation

Backpropagation is used to find gradients in the network. Once the gradients have been computed, the learning aspect of deep learning can begin with Stochastic Gradient Descent (SGD). Even a modestly-sized network such as ours will perform backpropagation many times; thus, it is imperative that backpropagation be contextualized and derived so as to communicate effectively the performance of our model.

The backpropagation algorithm is iterative and utilizes properties of the chain rule of calculus. Therefore, to back-trace the gradients, we take a first-order derivative of the cost function, which may involve other first-order derivatives because of the chain rule. For this project, we used a sigmoid activation function and the cross entropy loss function. Our predictions $\hat{y}$ were used in the cross entropy loss function $L_y\hat{y}$.

$$\hat{y} = \sigma(W^T x + B)$$
$$L_y\hat{y} = y \ln \hat{y} - (1-y)\ln(1-\hat{y})$$
$$L_y\hat{y} = \sum_{k=1}^{K}((-y_k \ln \hat{y_k}) - (1-y_k)\ln(1-\hat{y_k}))$$

$$\text{Let } \tilde{L_y}(u) = L_y(\sigma(u)) = L_y(\hat{y})$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = (\nabla L_y(\sigma(w)))^T \frac{\partial \sigma(w)}{\partial u_j}$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = \frac{\partial L_y(\sigma(w))}{\partial \hat{y_j}}(\sigma(u_j)(1-\sigma(u_j)))$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = (\frac{-y_j}{\sigma(u_j)} + \frac{1-y_j}{1-\sigma(u_j)})(\sigma(u_j)(1-\sigma(u_j)))$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = -y_j(1-\sigma(u_j)) + (1-y_j)\sigma(u_j)$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = -y_j + y_j\sigma(u_j) + \sigma(u_j) - y_j\sigma(u_j)$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = -y_j + \sigma(u_j)$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = \sigma(u_j) - y_j$$
$$\frac{\partial \tilde{L_y}(u)}{\partial u_j} = \hat{y_j} - y_j = delta$$

For the backpropagation algorithm, the value delta is used to signify the derivative of the cross entropy loss function. This simplification to a difference equation helps to save excessive computations on exponentials (from the sigmoid activation function) and logarithmics (from the first-order derivative of cross entropy) that may drive the gradient values to zero - which is highly undesirable since we want model learning to take place.

III. Methodology

Our project is based on the provided project repository [1]. The dataset used for training, validation and testing is the MNIST dataset. Our implementation for backpropagation followed the algorithm outlined in Figure 1.

	Algorithm – Back-Propagation for MLP
1	Run Forward Propagation for MLP
2	$G \leftarrow \nabla_{\hat{y}} L(\hat{y}, y)$
3	**for** $k = K$ down to 1 **do**
4	$G \leftarrow \nabla_{a^{(k)}} L = G \odot g'(a^{(k)})$
5	$\nabla_{b^{(k)}} L = G + \lambda \cdot \nabla_{b^{(k)}} \Omega(\theta)$
6	$\nabla_{W^{(k)}} L = G \cdot h^{(k-1)^\top} + \lambda \cdot \nabla_{W^{(k)}} \Omega(\theta)$
7	$G \leftarrow \nabla_{h^{(k-1)}} L = W^{(k)^\top} \cdot G$
8	**end for**

Fig. 1. Backpropagation algorithm

IV. Implementation

A. Project Environment

Language: Python 3.6
Libraries: numpy, matplotlib
Datasets: MNIST

B. Network Architecture

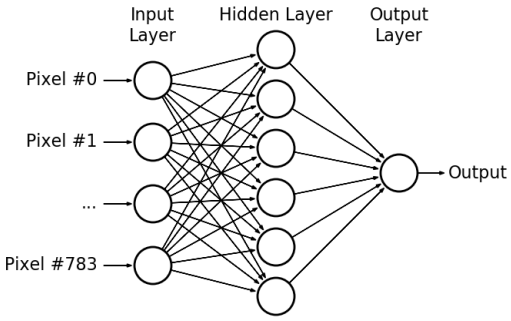

Fig. 2. Default Network Architecture

The network architecture was originally composed of three layers. The three layers are as follows: one input layer of all the pixels in one image (28x28), a hidden layer composed of 20 neurons, and an output layer of 10 neurons (one for each number). A simplified diagram of the default network architecture is shown in Figure 2 which shows that the network is fully connected; the diagram only shows one output neuron, but there are 10 output neurons (one for each digit 0 to 9) in the final layer.

C. Hyperparameters

1) Network Width: We explored the width of the network by altering the number of neurons in the middle layer. The default configuration was 20 neurons, but we performed model training with both 40 and 80 neurons in the middle layer. Modifying the network width had the greatest effect on training accuracy. As in Figure 3, the training accuracy exceeded the default 90% accuracy when the middle layer was composed of 80 neurons fully connected with both input and output layers.

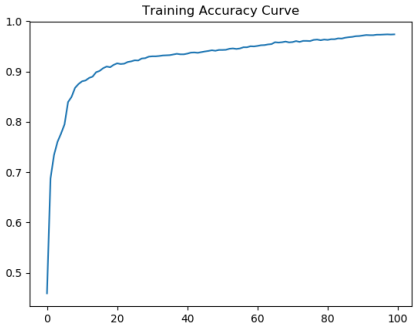

Fig. 3. 784-80-10 structure, Accuracy curve: Training data

2) Network Depth: Another area of the architecture we examined was network depth (the number of layers in the network). We found no significant improvement in model training when increasing the number of layers to 4 or 5 from the initial 3-layer structure. However, one intriguing observation can be seen in Figure 4. At about 70 epochs into the training, our accuracy quickly lost consistency and jumped dramatically to produce 'spikes' in the accuracy curve.

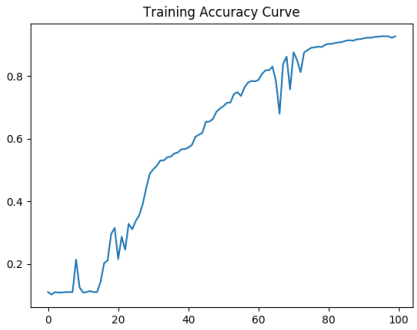

Fig. 4. 784-80-40-20-10 structure, Accuracy curve: Training data

3) Learning Rate: The last hyperparameter we surveyed was model learning rate. The original learning rate was 0.001. We performed training using learning rates of both 0.0001 and 0.00001. Unfortunately, the slower learning rates meant that after 100 epochs, we did not achieve as high an accuracy. Figure 5 shows that with a learning rate of 0.0001, the training accuracy was marginally below 90%.

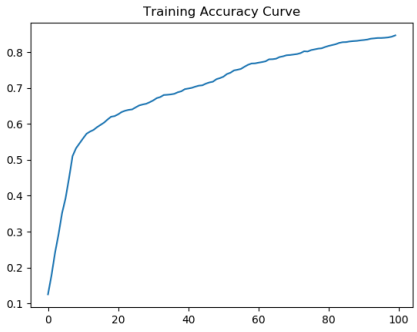

Fig. 5. Learning rate = 0.0001, Accuracy curve: Training data

V. Results

The loss and accuracy curves of our model over training, validation and testing sets were plotted using the following hyperparameters: 100 epochs, batch size equal to 128, and learning rate equal to 0.001. Our final evaluation accuracy on the testing set was 90.25%.

VI. Conclusion

Relatively successful image classification on the MNIST dataset can be achieved even with a 3-layer multi-layer percep-

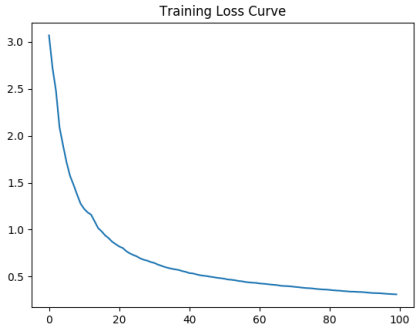

Fig. 6. Loss curve: Training data

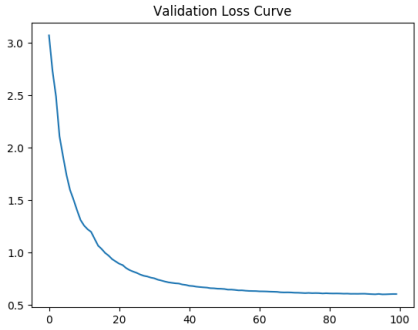

Fig. 7. Loss curve: Validation data

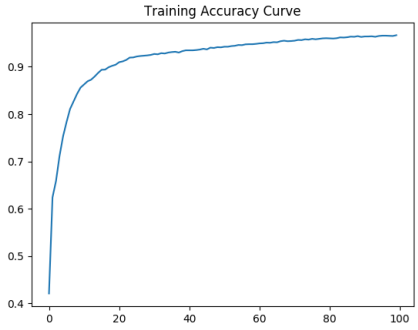

Fig. 8. Accuracy: Training data

tron network. The backpropagation algorithm and SGD process allowed us to develop a model that could internalize what a '1', '2', or '3' should look like in a relatively small amount of time and computing resources. Perhaps by optimizing the structure of the backpropagation algorithm and using advanced gradient descent heuristics, a more robust model could be developed to better leverage the power of deep learning moving forward. Another interesting research avenue would be exploring the depth and width of the network structure to find the right inflection point where a deeper network's benefits would outweigh the associated computational costs.

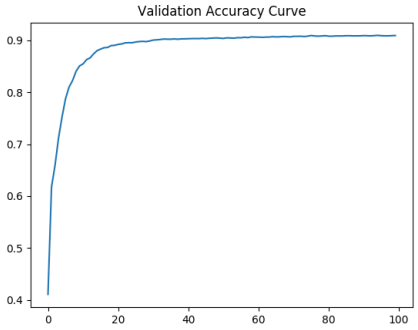

Fig. 9. Accuracy: Validation data

REFERENCES

[1] https://github.ncsu.edu/qge2/ece542-2018fall/tree/master/project/01