MLPs and Back-Propagation

Tre' R. Jeter

September 24, 2023

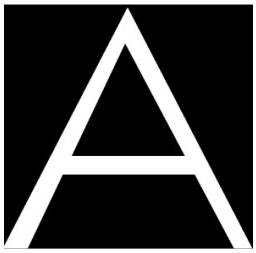
Abstract

Multi-Layer Perceptrons (MLPs) and the backpropagation algorithm represent a foundational framework in the field of neural networks and deep learning. MLPs are versatile feedforward neural networks characterized by multiple layers of interconnected neurons, each layer contributing to hierarchical feature extraction. These networks have found extensive applications in various domains, from image and speech recognition to natural language processing. The backpropagation algorithm serves as the cornerstone for training MLPs by efficiently adjusting the network's weights to minimize prediction errors. It accomplishes this through a process of error propagation, iteratively updating weights based on gradients computed during forward and backward passes. Together, MLPs and backpropagation have revolutionized machine learning, enabling the development of deep neural networks capable of modeling complex, high-dimensional data and solving a wide range of tasks. Their significance lies not only in their practical utility but also in their role as a fundamental building block for modern deep learning architectures.

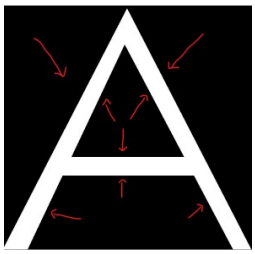
1 Question 1

In a 16x16 grid, select by hand the parameters of a two hidden layer MLP with signum units (threshold nonlinearity) as shown in Figure 1a (assume that the center of the figure is (0,0)). State the smallest number of hidden units the network needs in each layer and explain their role in creating the mask. Assume that black is 0 and white is 1. Can you achieve the same goal with a single hidden layer network? Justify your answer.

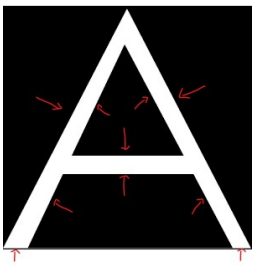
Answer: The input layer will have two units, for x and y coordinates, respectively. For the first hidden layer, my initial estimate of how many units to use is eight to represent the linear boundaries that make up each region of the mask (see Figure 1b). However, depending on if the bottom of the 'A' is also taken into account, this estimate will become ten (see Figure 1c).



(a) Original image.



(b) Initial estimate for first hidden layer.



(c) Second estimate for first hidden layer.

Figure 1: Estimating the number of units in the first hidden layer of the two-hidden layer MLP.

The second hidden layer will have four units representing the four main regions of the image (see Figure 2). The output layer will only have one unit because this is a two-class classification problem meaning the output will be 0 or 1. So, the smallest number of hidden units for this network will be either 12 (initial estimate) or 14 (second estimate). Based on the ***Universal Approximation Theorem***, classifying this mask with one hidden layer is possible. Specifically, the theorem states that a single hidden layer network with a non-linear activation function can approximate any function. Using one hidden layer would take away our exact knowledge of how many units to use and we'd need much more data to properly represent the function, but it should still be possible. Because my initial estimates of 12 and 14 were the smallest units making up my two-hidden layer

MLP, for a single hidden layer MLP, I could simply have one hidden layer with 12 or 14 hidden units instead of splitting them into two hidden layers. However, mathematically, knowing that I have a 16x16 pixel image, with a single hidden layer I could also use 256 hidden units (16x16) to represent each pixel of the image.

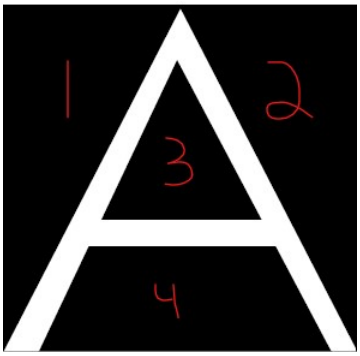


Figure 2: Number of units in the second hidden layer are depicted by numbers in red.

2 Question 2

Code the backpropagation algorithm and test it in the following 2 class problem. Use a single hidden layer MLP and specify the size of the hidden layer and tell why you select that number of hidden PEs.

x_1	x_2	d
1	0	1
0	1	1
-1	0	1
0	-1	1
0.5	0.5	0
-0.5	0.5	0
0.5	-0.5	0
-0.5	-0.5	0

It is expected that the system learns this pattern exactly. Is there just one possible solution to exactly separate the two classes of given points? How can you make the solution unique? Show experimentally how your suggestion works.

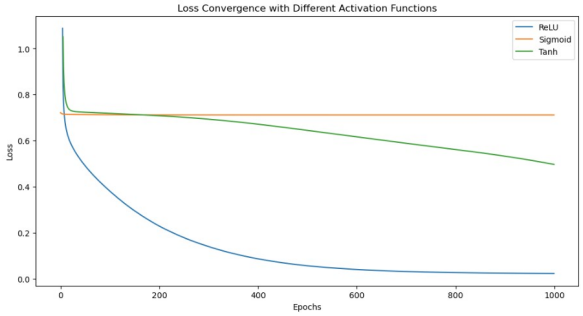
Answer: To learn the pattern exactly, I tested with a number of combinations of learning rates, weight decay, batch size, epochs, and hidden layer size. As this trial of hyperparameters was not exhaustive, I came to the conclusion that the minimum hyperparameters (that I tested) that resulted in a perfect classification scenario was a combination consisting of a learning rate of 0.01, 100 epochs, a hidden layer of size 15, batch size of 8, a weight decay of 0.001, and a ReLU activation function (Figure 3). I chose these hyperparameters, the size of the hidden layer specifically, because it fully allows the MLP to learn the data given without underfitting or overfitting. The main goal of a model is to make it as least complex as possible while still yielding exceptional results. With that said, there is not just one solution for this problem. In my experiments, this just so happened to be the simplest solution I found. However, increasing the epochs to 1,000 and hidden layer size to 32 yields the same results. Reducing the hidden layer size to 15 with the same hyperparameters as before also yields the same results, but converges faster for some activation functions. To test further, I even used multiple activation functions including Leaky ReLU, Sigmoid, Tanh, SeLU, eLU, PreLU, and Swish to see how activation functions also played a role in correctly classifying this two-class problem (see Figures 4 - 6).

```
Expected Outputs: [1 1 1 1 0 0 0 0]
Predicted Outputs: [1 1 1 1 0 0 0 0]
```

Figure 3: Perfect classification with ReLU.

```
Activation Function: ReLU
Final Accuracy: 1.0000

Activation Function: Sigmoid
Final Accuracy: 0.5000

Activation Function: Tanh
Final Accuracy: 0.7500
```

(a) Output of ReLU vs. Sigmoid vs. Tanh.

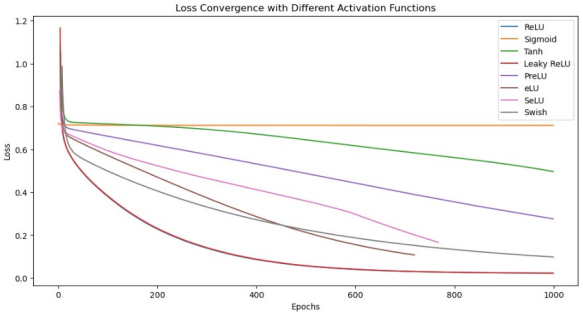


(b) ReLU vs. Sigmoid vs. Tanh

Figure 4: Results of ReLU compared to Sigmoid and Tanh.

```
Activation Function: ReLU
Final Accuracy: 1.0000

Activation Function: Sigmoid
Final Accuracy: 0.5000

Activation Function: Tanh
Final Accuracy: 0.7500

Activation Function: Leaky ReLU
Final Accuracy: 1.0000

Activation Function: PreLU
Final Accuracy: 1.0000

Activation Function: eLU
Final Accuracy: 1.0000

Activation Function: SeLU
Final Accuracy: 1.0000

Activation Function: Swish
Final Accuracy: 1.0000
```

(a) Output of multiple activation functions.

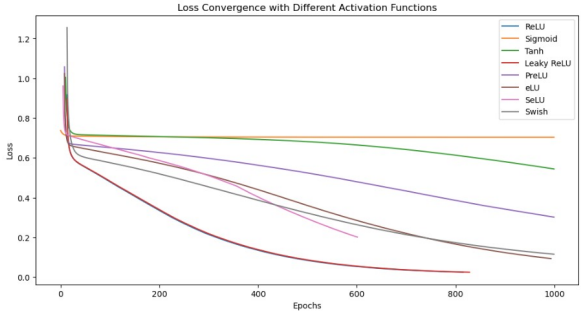


(b) Graphical representation of multiple activation functions.

Figure 5: Results of multiple activation functions in use with hidden layer size = 32.

```
Activation Function: ReLU
Final Accuracy: 1.0000

Activation Function: Sigmoid
Final Accuracy: 0.5000

Activation Function: Tanh
Final Accuracy: 0.8750

Activation Function: Leaky ReLU
Final Accuracy: 1.0000

Activation Function: PreLU
Final Accuracy: 1.0000

Activation Function: eLU
Final Accuracy: 1.0000

Activation Function: SeLU
Final Accuracy: 1.0000

Activation Function: Swish
Final Accuracy: 1.0000
```

(a) Output of multiple activation functions.



(b) Graphical representation of multiple activation functions.

Figure 6: Results of multiple activation functions in use with hidden layer size = 15.

From the results, we can see that the ReLU and Leaky ReLU outperform the rest of the activation functions as they converge quicker and classify our problem with 100% accuracy.

3 References

Below is a list of references used to assist with this assignment:

https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
https://pyimagesearch.com/2021/05/06/backpropagation-from-scratch-with-python/
https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide
https://github.com/ahmedfgad/IntroDLPython/tree/master
https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
https://medium.com/swlh/how-to-build-a-neural-network-from-scratch-b712d59ae641
https://medium.com/towards-data-science/how-to-program-a-neural-network-f28e3f38e811
https://github.com/c-bruce/artificial_neural_network