Devon Callanan
Machine Learning Final Project
Fall 2020
University of Pittsburgh

Algorithm

The neural network consists of layers or neurons. Each neuron activates based on the sum of inputs to the layer, biased, and passed through some activation function such as a sigmoid. The activations for a layer are then passed to the next set of neurons and weighted by a set of weights. A neural network learns by changing the set of weights and biases to minimize the error, or cost, of each classification. Improper classifications have a higher “cost” than proper classifications. Back-propagation is the process of using the derivative of the cost, in a method called gradient descent, to modify the weights and biases layer by layer.

Implementation

The neural network was implemented with three layers of neurons, the first layer of 10800 neurons for each pixel in the re-sized 120x90 image. Prior to learning, images were inverted (black to white/white to block) and normalized to 1 or zero values for each pixel. The hidden layer consists of 100 neurons, and the final layer has 62 output neurons, one for each class of character. All weights and biases were initialized with a random Gausian distribution centered at zero. The activation function used was the sigmoid seen in Figure 1. Saturation was initially an issue so z, the sum of inputs plus the bias term, was scaled to 1/10. Tests showed this was more effective that initializing weights with a smaller scaling factor.

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

Figure 1: Sigmoid function

In Figure 2, the blue represents the original sigmoid, while the red is the modified sigmoid.

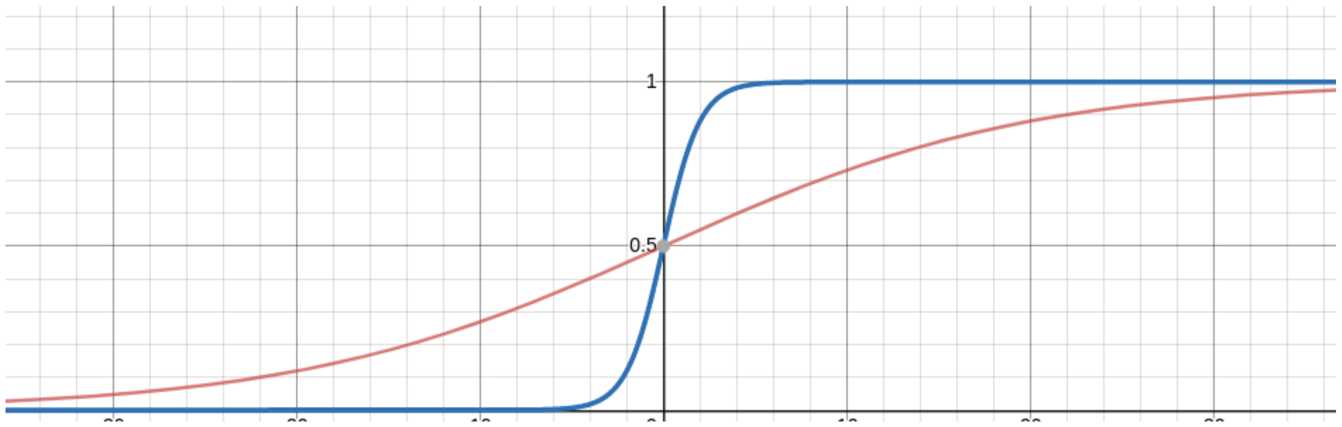

Figure 2: Modified sigmoid to avoid saturation

Activations can be written efficiently as a vector resulting from the product of the weights and prior activations plus the bias plugged into the sigmoid (Figure 3).

$$a^l = \sigma(w^l a^{l-1} + b^l).$$

Figure 3: Activation function in matrix form

With the feedforward completed, the cost (a cross entropy loss function) can be calculated. Given the cost, the gradient for each weight is calculated by first calculating the final layer (figure 4) and then applying the chain rule to find all previous layers' gradients (figure 5).

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} \sigma'(z^L_j).$$

Figure 4: Final layer error derivative

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l),$$

Figure 5: Prior layer error derivative

After much tuning, a high learning rate of 1 was found to be most effective. The results are shown bellow in Figures 6 and 7.

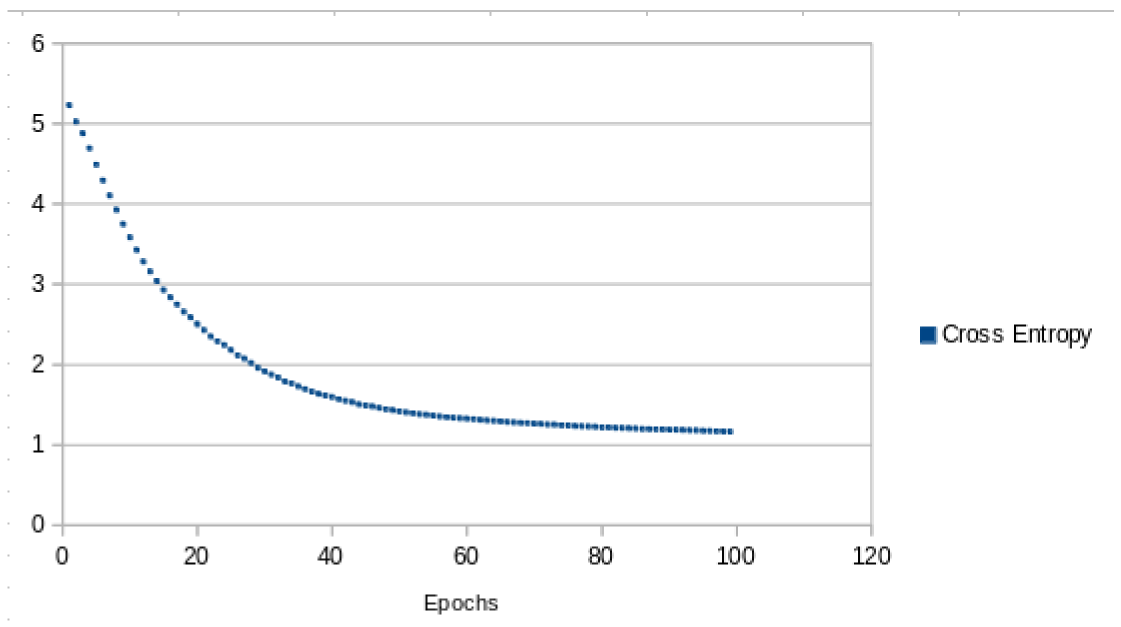

Figure 6: Cross entropy loss

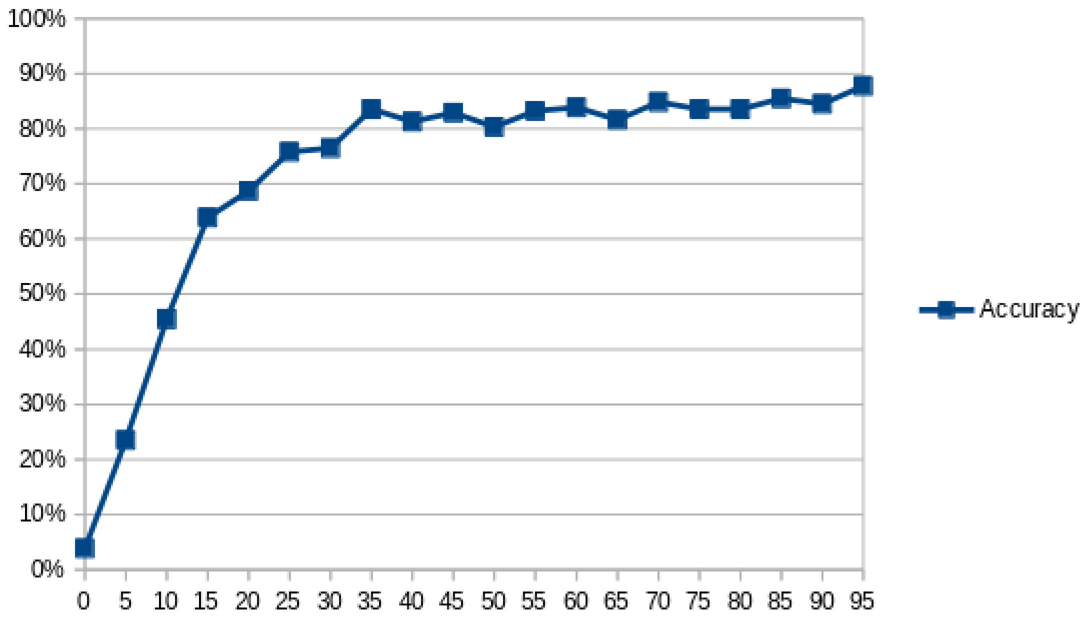

Figure 7: Accuracy

The accuracy was gathered by testing with a test set of five images per class. These images were randomly split from the training set on initialization. The network was not trained with testing data.

Running

dependenies: Python 3. numpy, PIL

To run, install the requirements (in a virtual environment)

python -m pip install -r requirements.txt

Then execute with

python ann.py

The code will run for 40 epochs, about 9 minutes (8 core i7).