Data Science 210 Final Project

Thomas Kwashnak

Fall 2021

.

Contents

1 Background

1.1 Goals

The main goal of this project is to create an algorithm that will be able to predict whether or not a player will win, from a given position on a Connect-4 board. To create an algorithm that does this, I used a neural network

2 Design

During this section, a variety of variables and variable syntax will be used. The following is a list that explains each variable. Remember that for a given "layer", the output is considered the 0th layer, the hidden layer is considered the 1st layer, and the input is considered the 2nd layer.

y^n	The output vector of the nodes in the n^{th} hidden layer.
$y^n_{i,0}$	The output value of node i in the n^{th} hidden layer.
y	By default, if y is left alone, it represents the output value, known as $y^0_{0,0}$
$\hat{y}$	The expected final output as a scalar value
W^n	The weight matrix for a given layer n, as it applies to the values Y^{n+1} (the outputs of the previous layer)
$w^n_{i,j}$	The weight value for the value passed from node j of layer $n+1$ as it is passed to node i of layer n

I've sectioned off the design of the algorithms into the independant algorithms, ordering them to try and explain the progression of deriving the algorithms.

2.1 Sigmoid Activation Function

In neural networks, an activation function is typically used to to condense the output of a node down to a [0-1] scale. In this lab, I used a sigmoid function. Below is the sigmoid function and it's derivative.

$$\sigma(n) = \frac{1}{1+e^{-n}}$$

$$\sigma'(n) = \sigma(n) \cdot (1 - \sigma(n))$$

I used the following algorithm so that if I passed in a matrix, it would just apply the sigmoid function to each value in the matrix.

Algorithm 1: $\sigma(n)$ function for both constants and matrices

```
Input: $A$ - which can either be a constant, or a $n$ x $m$ matrix
Output: $B$ - which is equivilant to $\sigma(A)$
if $A$ is a Matrix then
    $B \leftarrow$new Matrix($n$ x $m$) of 0s
    for $i = 0, 1, 2...n$ do
        for $j = 0, 1, 2...m$ do
            $B[i][j] \leftarrow \sigma(A[i, j])$
        end
    end
    return $B$
else
    return $1/(1 + e^{-A})$
end
```

2.2 Feed-Forward Algorithm

The Feed-Forward algorithm is the algorithm used to ”run” a neural network. This is one of the fundamental steps in how a neural network works. This step is used whenever you want to find the output that a neural network will calculate from a given set of inputs. Since the neural network is siplit into 3 layers, the transition of the values from one layer to the next can be described as:

$$\vec{y^n} = \sigma(W_n \cdot \vec{y^{n+1}})$$

If we substitute in all of our layers, we can get the following equation as the result of the entire network.

$$y = y^0_{0,0} = \sigma(W_0 \cdot \sigma(W_1 \cdot \vec{y^2}))_{0,0}$$

2.3 Loss Function

In order to use gradient descent, we will need to create a function that measures the correctness of our network. This is known as the ”Loss Function”. The premise of the loss function is that the higher it is, the worse the network is. Therefore, in order to make our network more accurate, we need to minimize this value.

$$L = (y^0_0 - \hat{y})^2 = (y - \hat{y})^2$$

2.4 0th Layer Weight Derivative

In back propagation, we need to find the derivative of each of the weights in terms of the loss function. First, we need to find the derivative of the loss function with respect to the weights that connect the hidden layer with the output layer.
Given the following equations:

$$y^1_{i,0} = \sigma(\sum_h w^1_{i,h} \cdot y^2_{h,0})$$

$$y^0_{0,0} = \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0})$$

We can calculate the derivative as follows:

$$\frac{\delta L}{\delta w^0_{0,i}} = \frac{\delta L}{\delta y^0_{0,0}} \cdot \frac{\delta y^0_{0,0}}{\delta \sum_h w^0_{0,h} \cdot y^1_{h,0}} \cdot \frac{\delta \sum_h w^0_{0,h} \cdot y^1_{h,0}}{\delta w^0_{0,i}}$$

$$\frac{\delta L}{\delta w^0_{0,i}} = 2 \cdot (y^0_{0,0} - \hat{y}) \cdot \sigma'(\sum_h w^0_{0,h} \cdot y^1_{h,0}) \cdot y^1_{i,0}$$

$$\frac{\delta L}{\delta w^0_{0,i}} = 2 \cdot (y^0_{0,0} - \hat{y}) \cdot \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0}) \cdot (1 - \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0})) \cdot y^1_{i,0}$$

$$\frac{\delta L}{\delta w^0_{0,i}} = 2 \cdot (y^0_{0,0} - \hat{y}) \cdot y^0_{0,0} \cdot (1 - y^0_{0,0}) \cdot y^1_{i,0}$$

2.5 1st Layer Weight Derivative

Next, we need to find the derivative of each of the weights that connect the input layer to the hidden layer. Given the following equations:

$$y^1_{i,0} = \sigma(\sum_h w^1_{i,h} \cdot y^2_{h,0})$$

$$y^0_{0,0} = \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0})$$

We can calculate the derivative as the following:

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta y^0_{0,0}} \cdot \frac{\delta y^0_{0,0}}{\delta \sum_h w^0_{0,h} \cdot y^1_{h,0}} \cdot \frac{\delta \sum_h w^0_{0,h} \cdot y^1_{h,0}}{\delta y^1_{i,0}} \cdot \frac{\delta y^1_{i,0}}{\delta \sum_h w^1_{i,h} \cdot y^2_h} \cdot \frac{\delta \sum_h w^1_{i,h} \cdot y^2_{h,0}}{\delta w^1_{i,j}}$$

$$\frac{\delta L}{\delta w^1_{i,j}} = 2 \cdot (y^0_{0,0} - \hat{y}) \cdot \sigma'(\sum_h w^0_{0,h} \cdot y^1_{h,0}) \cdot w^0_{0,i} \cdot \sigma'(\sum_h w^1_{i,h} \cdot y^2_{h,0}) \cdot y^2_{j,0}$$

$$\frac{\delta L}{\delta w^1_{i,j}} = 2 \cdot (y^0_{0,0} - \hat{y}) \cdot \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0}) \cdot (1 - \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0})) \cdot w^0_{0,i} \cdot \sigma(\sum_h w^1_{i,h} \cdot y^2_{h,0}) \cdot (1 - \sigma(\sum_h w^1_{i,h} \cdot y^2_{h,0})) \cdot y^2_{j,0}$$

$$\frac{\delta L}{\delta w^1_{i,j}} = 2 \cdot (y^0_{0,0} - \hat{y}) \cdot y^0_{0,0} \cdot (1 - y^0_{0,0}) \cdot w^0_{0,i} \cdot y^1_{i,0} \cdot (1 - y^1_{i,0}) \cdot y^2_{j,0}$$

We can then reorder it as..

$$\frac{\delta L}{\delta w^1_{i,j}} = 2 \cdot (y^0_{0,0} - \hat{y}) \cdot y^0_{0,0} \cdot (1 - y^0_{0,0}) \cdot y^1_{i,0} \cdot w^0_{0,i} \cdot (1 - y^1_{i,0}) \cdot y^2_{j,0}$$

And substitute in the 0^{th} layer derivative!

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta w^0_{0,i}} \cdot w^0_{0,i} \cdot (1 - y^1_{i,0}) \cdot y^2_{j,0}$$

2.6 Backward Propagation

Backward Propagation is the whole accumilation of the previous algorithms to train a neural network. First, the Backward Propagation algorithm calculates the results of the network. After doing so, it calculates the partial derivatives of the loss function in relation to each of the weights (using the Layer Weight Derivatives above). Using these derivatives, it nudges the weights according to the Step coefficient to bring the network

closer to the proper variable.
Additionally, we will also calculate the percent error of the propagation, which will make more sense in training the network

Algorithm 2: Backward Propagation for a 2-layer neural network

```
Input:  $Y^2$ - The input vector (matrix), dimensions $m$ x 1.
        $W^1_{\text{in}}$ - The weight matrix, dimensions $h$ x $m$, that represents the weights used as variables go from $Y^2 \to Y^0$
        $W^0_{\text{in}}$ - The weight matrix, dimensions 1 x $h$, that represents the weights used as variables go from $Y^1 \to Y^0$
        $S$ - The step coefficient to modify weights by, as a scalar value.
        $\hat{y}$ - The expected result from Forward-Feeding
Output: $W^1$ - The modified weight matrix, describing the same thing as $W^1_{\text{in}}$
        $W^0$ - The modified weight matrix, describing the same thing as $W^0_{\text{in}}$
        Err - The % error of the network
1  $W^1 \leftarrow$ copy of $W^1_{\text{in}}$
2  $W^0 \leftarrow$ copy of $W^0_{\text{in}}$
3  $Y^1 \leftarrow \sigma(W^1 \cdot Y^2)$
4  $Y^0 \leftarrow \sigma(W^0 \cdot Y^1)$
5  Err $\leftarrow |(Y^0_{0,0} - \hat{y})/\hat{y}|$
6  for $i \leftarrow (0, 1, \ldots, h-1)$ do
7      $\Delta W^0 \leftarrow -S \cdot 2 \cdot (Y^0_{0,0} - \hat{y}) \cdot Y^0_{0,0} \cdot (1 - Y^0_{0,0}) \cdot Y^1_{i,0}$
8      $W^0_{0,i} \leftarrow W^0_{0,i} + \Delta W^0$
9      for $j \leftarrow (0, 1, \ldots, m-1)$ do
10         $W^1_{i,j} \leftarrow W^1_{i,j} + \Delta W^0 \cdot W^0_{0,i} \cdot (1 - y^1_{i,0}) \cdot y^2_{j,0}$
11     end
12 end
13 return $W^1, W^0$, *Err*
```

2.7 Training the Network

Now, if you give the same input and expected output into the Backwards Propagation enough times, it's going to give you a network wtih 0% error for that input. However, it will perform very poorly when tested against a different set of inputs.
While we do have an abundance of trials that we are able to draw from (67,557 entries to be exact), we could simply run through each and every entry multiple times to get enough data. The question relies in how we manage our Step value, which should be large at first, and as it progresses, gets smaller and smaller.

3 Implementation

3.1 Data Importing

Since the data was given in a string csv format, an additional python script was created to import the data into two matricies, one contianing the inputs and one contianing the expected outputs.

4 Validation

5 Reflection