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 structure, which I trained specifically to do this calculation. To train the neural network, I used Stochastic Steepest Descent, and Back-Propagation as algorithms in order to "train" the network into accurately predicting if player 1 will win.

1.2 Mathematics Overview

The process for modifying the weights goes as follows. First, we use the neural network to take a set of inputs and get the output. We then compare that to the expected value using a loss function. Using that loss function, we find the derivative of the loss function in relation to each individual weight. Since we want to minimize the loss function, we take the derivative of each weight, which points towards the direction of greatest ascent, and go in the opposite direction by some step size. We repeat this whole process with different data entries, gradually reducing the step size, approaching a point where the network can generally predict the correct value.
In order to make the forward propagation, or calculating the output of a neural network from a given input, we placed the weights in matricies. The inputs are put in a vector, and the weights in a matrix. Each row of the weight matrix represents a node in that network layer, and each column represent the weight associated with one of the input variables. For example, an example of a 2-layer network, that only contains an input and an output layer, is represented as follows:

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{pmatrix} 1 \\ 4 \end{pmatrix} = \begin{pmatrix} 9 \\ 6 \end{pmatrix}$$

Where the first value is the weights, the second value is the input vector, and the last value is the output vector. When there are multiple layers, these are just recursively chained onto eachother. Each step of the way they are also normalized using a sigma function, which is described later in the design process.

1.3 Dataset

In order to train the network to evaluate a state of a connect-4 game, I used a dataset of connect-4 games, which provided a list of connect-4 game states, and the resulting value of the game.

> John Tromp. (1995). Connect-4 `https://archive.ics.uci.edu/ml/datasets/Connect-4`. Irvine, CA: University of California, School of Information and Computer Science.

The dataset contains 67,557 entries, each one representing a state in a game of Connect-4. As decribed on the database page:

> This database contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced.
> x is the first player; o the second.
> The outcome class is the game theoretical value for the first player.

The data is stored in a .csv file. Each row of the dataset represents a legal position. Each row is composed of 43 comma separated variables. The first 42 variables represent the board state, and teh last one represents the expected value of whether player 1 wins, loses or draws.
For the board position, there are three possible states. 'x' means that the first player has placed a piece in that square. 'o' indicates that the second player has placed their piece in that square. 'b' indicates that the square is vacant.
There are three possible "results" of the board state. "win" means that player 1 is expected to win from this game state. "lose" means that player 2 is expected to win from this game state. "draw" means that neither player is expected to win, or that the board will be completely filled before any player can get 4 in a row.

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. This will take some experimentation, but the first idea is to

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 ←new Matrix(n x m) of 0s
    for i = 0, 1, 2...n do
        for j = 0, 1, 2...m do
            B[i][j] ← σ(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 0^{th} 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 absolute 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_{in} - The weight matrix, dimensions h x m, that represents the weights used as variables go from $Y^2 \to Y^0$
W^0_{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_{in}
W^0 - The modified weight matrix, describing the same thing as W^0_{in}
Err - The absolute error of the network

$W^1 \leftarrow$ copy of W^1_{in}
$W^0 \leftarrow$ copy of W^0_{in}
$Y^1 \leftarrow \sigma(W^1 \cdot Y^2)$
$Y^0 \leftarrow \sigma(W^0 \cdot Y^1)$
Err $\leftarrow |(Y^0_{0,0} - \hat{y})|$
for $i \leftarrow (0, 1, \ldots, h-1)$ **do**
  $\Delta W^0 \leftarrow -S \cdot 2 \cdot (Y^0_{0,0} - \hat{y}) \cdot Y^0_{0,0} \cdot (1 - Y^0_{0,0}) \cdot Y^1_{i,0}$
  $W^0_{0,i} \leftarrow W^0_{0,i} + \Delta W^0$
  for $j \leftarrow (0, 1, \ldots, m-1)$ **do**
    $W^1_{i,j} \leftarrow W^1_{i,j} + \Delta W^0 \cdot W^0_{0,i} \cdot (1 - y^1_{i,0}) \cdot y^2_{j,0}$
  end
end
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. The way that the network is trained will be up to a bit of experimentation, which I will be experimenting with.

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