Machine Learning in C

Luka Flores

Contents

Chapter 1

Examples

1.1 Double Input Example

This section will show a simple example of machine learning in which we will train a mode to guess an output that is double the input

It will be written in the `c` language:

The following is the training set which will be used as data help the model with its predictions

Code 1: Training Set

```c
float train[][2] = {
    {0,0},
    {1,2},
    {2,4},
    {3,6},
    {4,8},
}
```

To start a model is a mathematical formula that consists of variables "weights", when the model is trained on data, the algorithm adjusts the variables to give a prediction cloer to the output.

For this basic example we will start with a model that only consists of one variable

$$y = x \times w$$

Where:
$y =$ Input
$x =$ Output
$w =$ Variable / Weight

As we have all taken middle school algebra we could conclude that after casting the model $y = xw$ onto the training model the weight would be 2.

When writing our program the goal is for it to "Learn" the weight.

As a starting point for our model we will generate random values with the following code:

Code 2: Random Float Generator Function

```c
float rand_float(void){
    return (float) rand() / (float) RAND_MAX;
}
```

If we were to call the function without it would produce a random number but if we were to rerun the same function the number return will be the same as the first run.

To produces different random numbers, We have to change the seed every time we run the code. An easy way of doing this is to attach the seed to the current time, and hence will change with every new second.

Code 3: description

```
srand(time(0)); // Generates the seed based on the current time
float w = rand_float();
printf("%f\n", w);
```

This will be useful in the future but to start with we will fix it to one seed with `srand(12)`. Note: 12 is an arbitrary number

We can iterate over the training set and show the expected output from the training set and the predicted output from the model

Code 4: description

```
for (size_t i = 0; i < train_count; ++i) {
    float x  = train[i][0]; // Trained Output
    float y  = x*w;  // Model Output

    printf("Actual:␣%f,␣Expected:␣%f␣\n", y , train[i][1]);
}
```

You will notice that our model currently is not doing any "learning" it is just guessing the same weight every iteration.

Currently there is no method to determine how well the random number (as the weight) satisfies the data set, so we need to build something called a cost or loss function. This function evaluates the current weight and provides insight to it.

To create a cost function we will get the distance between the Actual and Expected values then average the square across the training set.

Code 5: Averaging Distances (Within Iteration Loop)

```
    distance = y - train[i][1];
    result += d*d;
```

Note: The squaring the averages not only finds the absolute distance it also amplifies the errors, making them more noticeable

Now we have to average the distances to produce a meaningful result, the following is the full cost function

Code 6: Cost Function

```
float cost(float w){

    float result = 0;

    for (size_t i = 0; i < train_count; ++i) {
        float x  = train[i][0]; // Trained Output
        float y  = x*w;  // Model Output
        float distance = y - train[i][1];
        result += distance*distance;
        printf("Actual:␣%f,␣Expected:␣%f␣\n", y , train[i][1]);
    }

    return result /= train_count;

}
```

After running the code we can see that the cost function produces a positive number, the goal is to have the number approach 0 (meaning that the model is predicting the expected result exactly)

We can change the cost result by adding very small number (epsilon) to the weight incrementally trying to find a cost result closer to 0

The next question is: do we subtract or add to reduce the cost function

For the sake of illustration we can think about the cost as a quadratic graph

[INSERT GRAPH OF QUARDATIC]

The lowest point on the graph will produce the best result

Given a random point on the graph, how do we find direction of the lowest point?

We use derivatives, a reminder of derivatives

$$L = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

We can implement an estimate of this in code, the following is not recommended in real machine learning problems this is purely in an attempt to understand the concept

Code 7: Derivative Estimation

```
float epsilon = 1e-3;
float dcost = (cost(w + epsilon) - cost(w))/epsilon;

w -= dcost;
```