Notes on Math and Architectures of Deep Learning

Benjamin Santana

April 2023

1 introduction

Before jumping into deep learning, let us talk about machine learning a bit. What is machine learning? Living creatures are taking decisions at every moment. Say for a example a dog sees an object and has to take a decision, does it get away from it, does it runs to towards it, or does it simply ignore it.

This will depend on certain factors about the object, is it sharp? does it looks cozy? Believe it or not the dog has a machine inside its head, taking this decisions. Its brain.

Machine learning is trying to emulate this machine, taking different parameters and output a decision.

Computers already do this, but the new thing is that now we do not give the computer step by step instructions on how to take the decision. Instead we develop a mathematical model for the problem.

Let us continue with the dog example I was taking about (which I completely stole from the book).

We have two *features*, hardness and sharpness.

We could define a *parameterized function* with a *weighted sum of inputs*:

$$y(hardness, sharpness) = w_0 \cdot hardness + w_1 \cdot sharpness + b \quad (1)$$

The weights *w0* , *w1* and the bias b are the parameters of the function. The output y can be interpreted as a threat score.

The weights are not know at first, we need to estimate them, this is what we call *model training*.

The steps to solve any machine learning problem are the following:

1. We first design a parameterized model function (e.g. weighted sum) with unknown parameters (weights). This makes the *model architecture*.

2. Estimate the weights via model training.

3. Now we have a complete model, we can send inputs not seen before and it should work, this is called *inference*.

In supervised learning, us humans will create the training data, this means creating example inputs, each corresponding with the desired ouput.

The training the model part is iterating through different weight values and checking with the training data to see if we are close to the desire output.

We optimize the way we look for those weights, we will see that in detail in the future.

At first, our model will be dumb, it will not be able to perform well, like a baby. But once we have run some iterations and we correct it, it will start performing better.

2 simple model

The book used a simple example on how to apply machine learning to the dog problem we stated above, in the book it was a cat but I am more of a dog person so...

We have a hypothetical dog which only needs to take one decision in life, run away from an object, ignore it, or approach it. It takes this decision based only on two quantitative inputs.

2.1 input features

$x_0 = hardness$, $x_1 = sharpness$

we can normalize this input

$$v_{norm} = \frac{(v - v_{min})}{(v_{max} - v_{min})} \tag{2}$$

where v_{min} is the min value possible and v_{max} is the max value possible.

This will generate values between 1 and 0, thus we can define the input as

$v \in [v_{min}, v_{max}] \rightarrow v_{norm} \in [0, 1]$

A single input becomes

$$\vec{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \in [0, 1]^2 \tag{3}$$

2.2 output decisions

We will have our model estimate a threat score (y) and based on a threshold (δ) we will assing a class to it. If $y > \delta$ high threat, run away. If y is around 0, do nothing. If $y < \delta$ negative threat, approach.

2.3 model estimation

We need to estimate the function which transforms the input vetor to the output. This consists on two parts

- model architecture selection
- model training

2.3.1 model architecture selection

Our model has three parameters w_0, w_1, b they can be represented as

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \tag{4}$$

and b as a scalar. All three being real numbers. We will define a super simple function such as

$$y(x_0, x_1) = w_0 x_0 + w_1 x_1 + b = \vec{w}^T \vec{x} + b \tag{5}$$

Where b is the bias.

2.3.2 model training

We defined $y(\vec{x})$, now we have to find the best weights for the function. We will choose the parameters so that the outputs on the training data inputs, match the corresponding outputs as close as possible.

How do we calculate the error?

On the i^{th} training data pair $(\vec{x}(i), y_{gt}(i))$ (gt ground truth) we can say $y_{predicted}(i) = \vec{w}^T \vec{x}(i) + b$, then we compare $y_{predicted}(i)$ with y_{gt}

$$e(i)^2 = (y_{predicted}(i) - y_{gt})^2 \tag{6}$$

It is squared to take into account negative values

This way we calculate the error of one iteration, the overal error (aka loss) is the sum of all $e(i)^2$

$$E^2 = \sum_{i=0}^{i=N} e(i)^2 \tag{7}$$

our goal is to find the $\vec{w}$ that minimizes E^2