A Mathematicians Guide to Deep Learning Part I: Neural Networks

Table of Contents

Introduction

This will be the first in a series of articles that I have written for two reasons. Firstly to solidify my knowledge of the mathematical foundations of the subject. Secondly, I am posting the series online to help those, like myself, who are interested in the mathematics of deep learning but find it hard to know where to start or where to look.

I have a friend with a similar mathematical background to myself, studying for a PhD in Mathematical Physics. I would say his research interests lie in geometrically understanding physical theories (some might say he is interested in a version of Hilbert's sixth problem). He is also interested in the mathematical aspects of deep learning. It is to him that I aim this work and thus the level of mathematics needed is high. One does not need to have a PhD to approach the topic (I do not), but an undergraduate mathematical education will suffice.

The purpose is to provide an insight into the overlap of physics/geometry with machine learning theory. Specifically how the paper on Gauge-CNN's uses gauge theory to solve some problems in modern deep learning (DL) theory.

In this first article, we will be concerned with a rigorous mathematical definition of a neural network.

We start by giving a very brief & simple introduction to Hebbian Theory. This is a model of how brain cells react during the learning process. Continuing on to define the mathematical model for the constituent elements of neural networks, the neuron. Also looking at a simple case we define the loss function which enables 'learning' for this neuron model.

Finally, we formally define neural networks in two ways, one as a function over feature space and a set of parameters. The second definition uses a collection of sequences to inherently subsume these notions and makes clearer the process of iterative abstraction that allows a neural network to learn.

Simplistically Imitating Biological Neural Networks

An artificial neural network (NN) is a mathematical model commonly used in machine learning. They are based on biological counterpart the neural networks of the brain. Over the past decade or so NNs have massively improved our ability to do things like beat world champion board game players and even teach cars to drive themselves.

Biological neural networks are massively connected structures of cells called neurons. The human brain contains on the order of 10^{11} neurons, each of which connects to approximately 6000 others![1]

Hebbian Theory

This theory describes the role of individual neurons contribution to the brains adaption during learning.

[1] Images taken from Calin, O., 2020. Deep Learning Architectures. Cham: Springer.

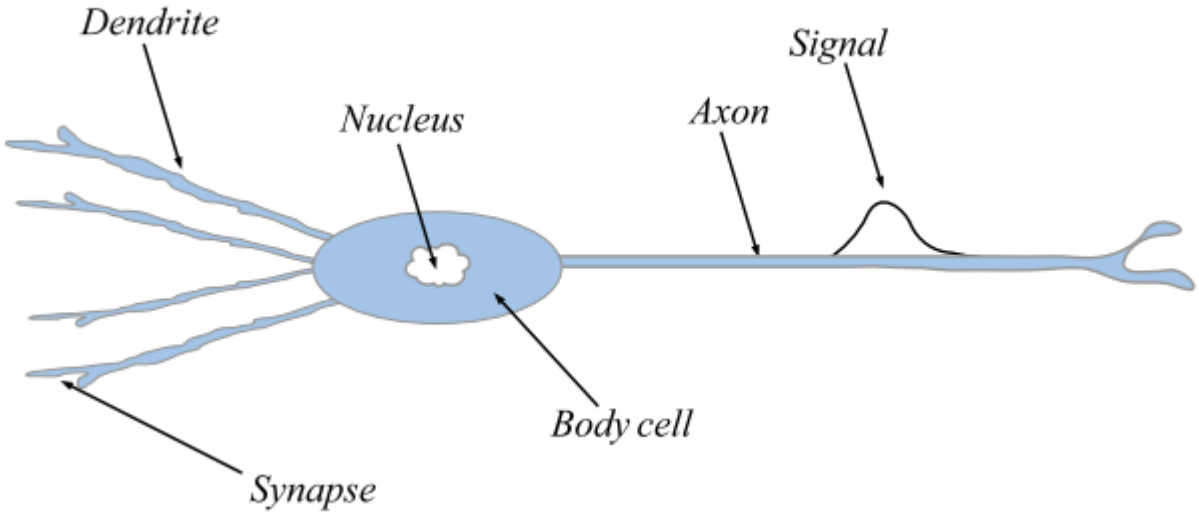

Figure 1: A neuron cell

The synapses of individual neuron cells pass chemicals to the axons of others, which either react strongly or weakly. The information of the interaction is transmitted along all the dendrites to the body cell. Where the signals are then combined, causing the neuron to fire provided a threshold is exceeded.

The formation of memories occurs under repeated signalling between neurons, i.e. when both neurons are firing, the synapses are actually strengthened. Meaning that the weights are updated each time this occurs. This more finely tunes the neuron whereas there is degradation of the synapse strength when the neurons do not fire together.

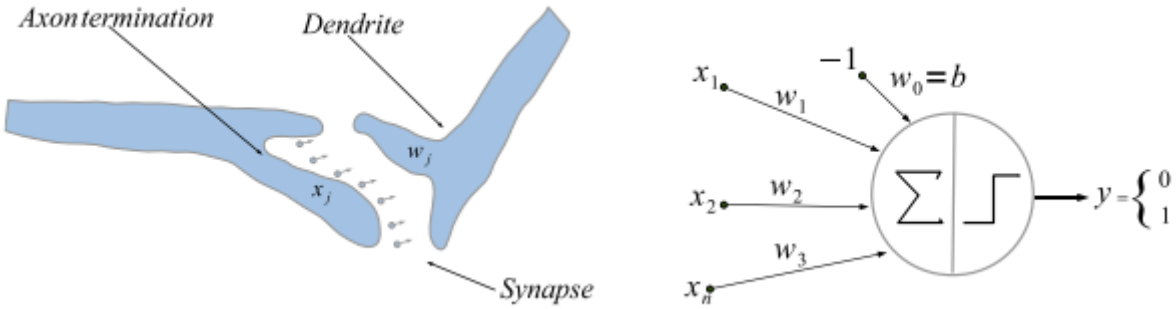

Figure 2: Synaptic Gap

A schematic representing the mathematical action of the neurons:

$w_0 = b$:	This bias term represents the threshold for the neuron to fire.
w_i :	These weights represent the strength of the interaction.
x_i :	The input representing information from other neurons.
y :	The output representing whether or not the neuron has fired.

The circle (which represents the body-cell) is split into two parts, the first $\sum$ representing the linear combination of then dendrite information, i.e. $\sum_{i=1}^{n} x_i w_i$. Then the second half represents the application of what is called the **activation function** which we will denote by φ. This allows us to write the output of the neuron as

$$y = \varphi_b\left(\sum_{i=1}^{n} x_i w_i\right) = \varphi_0\left(\sum_{i=1}^{n} x_i w_i - b\right)$$

Where the activation functions are the unit step function centered at b φ_b and the Heaviside function.

The point of the activation function is to map the outpoint into the set $\{0, 1\}$ based on where or not $\sum_{i=1}^{n} x_i w_i > b$.

This model of a neuron is referred to as a ***linear perceptron*** or just a ***perceptron***. However, the neuronal model, where we do not have such a requirement on the image of the activation function, is a strictly more general notion encompassing that of the perception.

For the neuronal model, we employ a trick to simplify notation. By considering the bias as an additional weight with input -1 (already done in the picture above) we can write the output as

$$y = \varphi_0\left(\sum_{i=0}^{n} x_i w_i\right)$$

This trick will appear multiple times, so for brevity, I will refer to this simplification as 'zero indexing the bias'.

Learning

Although this model is often considered oversimplified as to the true nature of neurons in the brain, we are mathematicians & physicists so oversimplified is a good place to start *insert spherical cow reference here*. This results in the learning of the output function. The question is, what class of output functions can be learned in this model?

Well as we seen above, the model is simply determining whether the inequality $\sum_{i=1}^{n} x_i w_i > b$ is satisfied or not. Thus the output function is a piece-wise function determining whether or not some point $(x_1, ..., x_n)$ belongs to some $n-$dimensional hyperplane.

However, the function cannot determine whether or not such a point lies within the interior of say, an n-sphere, since this would require an inequality quadratic in the inputs.

Although it is possible to learn non-linear functions provided one also uses a non-linear activation. Thus it is commonly said that activation functions are responsible for introducing non-linearities to the NNs.

Networks of Artificial Neurons

We can actually layer the above mathematical model of neurons, by simply composing the learned affine functions. Such a network is called a ***multi-layer perceptron (MLP)*** or a ***feed-forward neural network***.

Multi-Layer Perceptrons as a function Suppose we have N vector spaces $\mathcal{X}^{(\ell)}$ and let $\Theta^{(\ell)}$ be the set of possible parameters for each layer (denoted by ℓ), both labelled by $\ell \in \{0, ..., N-1\}$.

These spaces are often called ***feature spaces***, a neural network then layers these spaces connecting them with maps $\Psi^{(\ell)}$,

$$\Psi^{(\ell)} : \mathcal{X}^{(\ell)} \times \Theta^{(\ell)} \to \mathcal{X}^{(\ell+1)}$$

Then combining these forms the ***neural network***

$$\Psi : \mathcal{X} \times \Theta \to \mathcal{Y}$$

$$x \mapsto \underbrace{\Psi^0(z^0, \theta^0)}_{\text{first hidden layer}} \mapsto ... \mapsto \underbrace{\Psi^j(z^j, \theta^j)}_{(j+1)\text{-st layer}} \mapsto ... \mapsto \underbrace{\Psi^{N-1}(z^{N-1}, \theta^{N-1})}_{\text{output layer}}$$

The initial layer $\ell = 0$ is called the ***input layer***, then intermediate steps $\ell \in \{1, ..., N-2\}$ are referred to as ***hidden layers*** and the final layer $\ell = N-1$ is called the ***output layer***.

Bear in mind that the set notation may be abstracting way the meaning of the domain.

That is, $\mathcal{X}$ actually the first vector space $\mathcal{X}^0$, whereas the Θ is not Θ^0 but instead $\Theta = \Theta^0 \times ... \times \Theta^{N-1}$. That is, Θ represents the **entirety of the networks parameters** and $\mathcal{X}$ is the ***input space***.

We can actually abstract away this notion of the set of parameters altogether. Since our layers of the network are made up of individual neurons as defined above, we have a more abstract definition of a MLP, which actually clarifies the essence of a neural network.

Multi-Layer Perceptrons as a Sequence Consider the $\mathcal{X}^{(\ell)}-$valued function spaces for the $\ell-$th layer

$$\mathcal{F}^{(\ell)} = \{f \mid f : \underbrace{\mathcal{X}^{(\ell)} \times ... \times \mathcal{X}^{(\ell)}}_{d^{(\ell)}-\text{times}} \to \mathcal{X}^{(\ell)}\}$$

Define some affine function on each neuron in this layer $\alpha_i^{(\ell)} : \mathcal{X}^{(\ell)} \to \mathcal{X}^{(\ell+1)}$ which decomposes as $\alpha_i^{(\ell)}(x^{(\ell)}) = (x^{(\ell)})^T w_i^{(\ell)}$ with the usual convention of zero indexing the bias term.

In general, each neuron in a layer can have distinct activation functions; however, it is common to choose a single activation function per layer[2]. For the sake of generality, let's assume we do indeed have distinct activations since the latter is a simple restriction.

Let the activation functions of each neuron in the $\ell-$th layer be $\{\varphi_i^{(\ell)} \mid i \in D^{(\ell)}\}$, where $D^{(\ell)}$ is an indexing set with $|D^{(\ell)}| = d^{(\ell)}$ the number of neurons in the $\ell-$th layer.

Now let $\alpha^{(\ell)} : \mathcal{F}^{(\ell)} \to \mathcal{F}^{(\ell+1)}$, which can be thought of as a vector of all the affine functions in a layer. Then also define $\varphi^{(\ell)}$ a layer-wide combination of the activation function. Then these compose to act as a map between layers, that is,

$$x^{(\ell+1)} = \varphi^{(\ell)} \circ \alpha^{(\ell)}(x^{(\ell)})$$

The action of this neuron-wise is

$$x_j^{(\ell+1)} = \varphi_j^{(\ell)} \circ \alpha_j^{(\ell)}(x^{(\ell)})$$

Finally a ***feed-forward neural network with L layers*** can then be defined as follows:

> Let $D = \{D^{(1)}, ..., D^{(L)}\}$ and then a sequence of maps $(f^{(i)})_{i \in D}$, together with two other sequences, one of activation functions $(\varphi^{(i)})_{i \in D}$ and one of affine functions $(\alpha^{(i)})_{i \in D}$, is a neural network when maps are defined iteratively:
>
> $$f^{(\ell)} = \varphi^{(\ell)} \circ \alpha^{(\ell)} \circ f^{(\ell-1)}$$
>
> where $f^{(0)}$ is the input.

The network is said to be L layers ***deep*** and $\max_{\ell \in D}(d^{(\ell)})$ ***wide***. The iterative nature may be viewed as forcing information forward throughout the network, hence ***feed-forward***.

What is clearer from this representation is the fact that a deep feed-forward neural network progressively 'collects' a set of ***increasingly abstract reparameterisations***, $f^{(\ell)}$. This is due to the action of all the parameterised functions in both the affine and activation sequences.

Concluding Remarks

This may seem like a painstaking amount of abstraction for something that is essentially repeatedly summing and applying some function. But the advantage is the abstraction immediately distils the essence of the process. Successive reparameterisation.

Without the non-linearities provided by the activation functions, MLPs meet serious limitations. In fact, it can be shown that a network with only linear neurons can be recast as a single layer network. In that case, we lose the reparameterisation property of the network, which allows us to learn important features of the input.

[2] I believe this choice is due to improved computational performance.

The next article on convolutional neural networks will piggyback off this idea and add some more geometry into the mix. These are the two essential ingredients for understanding how gauge theory can help in deep learning. So hopefully in the third article, we will confront the work of Taco Cohen et al.

This article, of course, leaves many questions unanswered about neural networks. We checked what functions we can learn with a single neuron. What about a neural network? What about the process of learning these functions? Must we restrict ourselves to an L^2-sense of norm as we did for the linear neuron? What do we pick up by extending our feature spaces from vector spaces to, say Banach spaces for example?

I hope to answer some of these questions for myself and also to share them when I do.

References

This article is my personal take on some sections from a combination of the following books/papers:

- Calin, O., 2020. Deep Learning Architectures. Cham: Springer.
- Anthony, M. and Bartlett, P., 2010. Neural Network Learning. Cambridge: Cambridge University Press.
- Celledoni, E., Ehrhardt, M.J., Etmann, C., McLachlan, R.I., Owren, B., Schönlieb, C.B. and Sherry, F., 2020. Structure preserving deep learning. arXiv preprint arXiv:2006.03364.

I hope you found the article useful

I am still learning the subject myself, so if you had problems with the material please reach out! Or indeed if you any have any other interesting paper/book/articles suggestions please pass them along.

To connect with me or find more content similar to this article, do the following:

- Check out my personal website www.peterferguson.co.uk
- Follow my Medium
- Leave a comment!