Neural Networks

Adithya Nair

November 14, 2024

Contents

1 An Introduction

Hello everyone, I'm Adithya and we're from the Journal Society. This is a group of people coming together to discuss mathematical and scientific ideas in a rigourous fashion. We have sessions on Real Analysis guided by Prof. Vineet Nair.

This session is largely an introduction. It's not representative. On January, we'll dig even deeper into these things, slowly building intuitions until the state of the art in deep learning work today.

2 Why The Focus On Rigour?

It's a fundamental belief that drives the society that understanding these concepts with depth is paramount to doing good work. We strive to stand by this notion with every session, that math is the best language we have for expressing ideas. Understanding these things as deeply as possible also allows us to understand where things are going, what on earth is even happening.

If this appeals to you as well, do participate in these discussions as much as possible.

3 Neural Networks

3.1 Definitions

We're going to be using an example of an old classic deep learning problem. Let's say we had M greyscale images of the digits $0 \cdots 9$ drawn on a square. These digits are handwritten. How do you make a computer recognize each digit?

Each image is a set of p pixels, or a vector $\vec{v} = (v_1, v_2, \cdots, v_p)$. Each element v_i tells us how 'dark' or 'light' that pixel is. 0 is black, and 1 is white.

We can say that we have M vectors $\vec{v}$ in p -dimensional space or $\mathbb{R}^p$

We do have a function f that can map these specific vectors v to an output in the discrete set $\{0, \cdots, 9\}$.

3.2 The Problem

The problem is, it **cannot generalize**. There's no *rule* or *expression* at play here that we can use to take a new image and map it to that same discrete set. This rule is the function we're trying to learn. But what on earth could possibly be the best fit?

3.3 The Big Picture

Neural networks are compositions of simple functions. Why do neural networks work so well? How do we even get a function that can even see handwritten digits and recognize them.

This part will be covered in much better depth in January, but there's an idea that neural networks tap into called **universality**, to truly understand it, a session might need to be dedicated to just this property... for now, suffice it to say that neural networks are capable of **universal approximation** and there are theorems that you can go ahead and look into that lay out all the conditions for this capability.

Back to neural networks, they're compositions of functions,

$$F(v) = F_L(F_{L-1} \cdots F_2 F_1(\vec{v})))$$

The reason they work so well, is that we are finding or computing the right parameters such that the output of this function matches the expected output as much as possible.

Now the question arises? What kind of simple functions?

3.4 Linear Functions

The first answer could be a linear function $F(v)$ which maps from $\mathbb{R}^p \to \mathbb{R}^{10}$. The output vector would simply be probabilities from 0 to 9.

Definition 3.1: Linear Functions Or Linear Maps

A linear function, in linear algebra terms, means a mapping between two vector spaces $V \to W$ which preserves the operations of vector addition and scalar multiplication. Or, The function f is such that,

1. $f(u + v) = f(u) + f(v)$
2. $f(cu) = cf(u)$

Linear functions are unfortunately severely limited, they allow for simplicity. Even though intuitively you can say that a 1 and a 0 can be combined to form a 6 or a 9, there's no well-defined rule for such combinations that will generalize. The input-output mapping is not exactly linear.

This was such an issue that this resulted in an AI 'winter'.

3.5 Linear Piecewise Continuous Functions

The rule that turned out best are **Linear Piecewise Continuous Functions**. Now the reason as to *why* this works is somewhat unclear.The best answer I found was given by Gilbert Strang here, at least to develop an intuition for this.

> "Linear for simplicity, continuous to model an unknown but reasonable rule and piecewise to achieve the nonlinearity that is an absolute requirement for real images and data." — Gilbert Strang, Linear Algebra And Learning From Data (2019)

Another note, such functions were also used back in the 1940s to 'model' biological neural networks by none other than the mathematician Householder. Perhaps there is some intuition behind these functions that we're missing which we'll ideally cover when we dive deeper in January.

We're trying to fit a curve, and we in many ways need to 'break' out of simply preserving the operations as they limit what the output function can look like.

Here's a definition of such functions,

Definition 3.2: Piecewise Continuous Functions

A function $f(x)$ is said to be piecewise continuous on an interval $[a, b]$ if it is defined and continuous except possibly at a finite number of points $a \leq x_1 \leq x_2 \leq \cdots \leq x_n \leq b$ Furthermore, at each point of discontinuity, we require that the left and right hand limits exists.

$$f(x_k^-) = \lim_{x \to x_k^-} f(x); f(x_k^+) = \lim_{x \to x_k^+} f(x)$$

At the ends of the domain, the left hand is ignored at a and the right hand limit is ignored at b.

You will see as this progresses how such functions are used.

We use composition to create complex functions from simple ones.

In the form,

$$F(v) = F_L(F_{L-1} \cdots F_2 F_1(\vec{v})))$$

Each function is either *linear* or *affine*, with our activation function applied on top.

Definition 3.3: Affine Functions

An affine function is a function with a linear transformation as well as a translation.

This means that the functions are of the form $F_i(\vec{v}) = A_i\vec{v} + b_i$ where A_i is the linear transformation and b_i is the translation.

These functions are then composed like mentioned above $F(\vec{v}) = F_L(F_{L-1} \cdots F_2 F_1(\vec{v})))$.

3.6 The Goal Of Training A Neural Network

The goal of neural networks becomes making it so that the A_i and b_i that we compute for the entire function is such that the loss or error between the function's output and the true expected output is **as minimal as possible**

3.7 A Primer On Multivariable Calculus

Definition 3.4: Partial Derivative

The derivative of a function dependent on multiple variables with respect to one of the variables, while the others are kept constant. For a function $f(x, y, \cdots)$,

$$\frac{\partial f}{\partial x} = f'_x$$

Definition 3.5: Gradient Operator

The gradient operator is a linear operator that helps calculate the total derivative.

$$\text{where}, \nabla = \frac{\partial()}{\partial x_1}\hat{e_1} + \frac{\partial()}{\partial x_2}\hat{e_2} \cdots \frac{\partial()}{\partial x_n}\hat{e_n}$$
$$\nabla f = \frac{\partial f}{\partial x_1}\hat{e_1} + \frac{\partial f}{\partial x_2}\hat{e_2} + \cdots + \frac{\partial f}{\partial x_n}\hat{e_n}$$

This gradient operator is very useful to us. If we can somehow compute the 'gradient' of a loss function, i.e. a function that calcululates how far off our network is from predicting the right output and the partial derivatives with respect to **each** weight and bias, we could use that information to update our A_i and b_i to better match the expected output.

If we move our weights and biases in the opposite direction of the gradient then we would get closer and closer to a minima.

We'll be defining this with a bit more depth and rigour in January, but this intuition is all that you'll need so that you can feel like you learned something.

This, in some sense is an intuition for gradient descent.

4 The Entire Flow From Beginning To End.