

Neural Networks: The Hard Way

David E. Weirich

March 5, 2018

Talk Objectives

- ▶ Understand the mathematical basis for artificial neural networks.
- ▶ Use that knowledge to design and build a neural network without relying on preexisting ML frameworks.

Talk Objectives

- ▶ Understand the mathematical basis for artificial neural networks.
- ▶ Use that knowledge to design and build a neural network without relying on preexisting ML frameworks.

Or in other words...

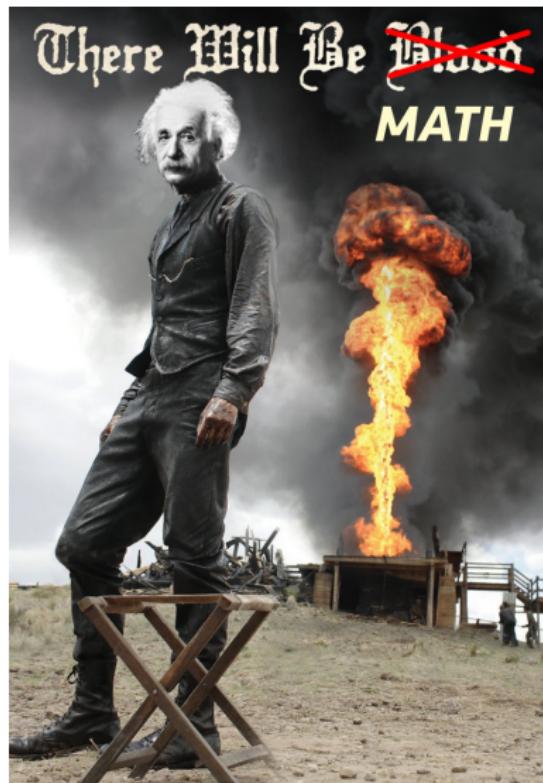
Talk Objectives

- ▶ Understand the mathematical basis for artificial neural networks.
- ▶ Use that knowledge to design and build a neural network without relying on preexisting ML frameworks.

Or in other words...

Learn Neural Networks THE HARD WAY

Disclaimer



Why the “hard way” ??? I prefer easy!

What do I mean by "the hard way"?

Doing things the hard way means teaching you how to teach yourself.

The Problem

We wish to approximate a function given a finite collection of known inputs and outputs.

$$||F - \widehat{F}|| < \varepsilon$$

Here F is the true function, and \widehat{F} is our approximation.

What are Neural Networks?

Q: What is a neural network?

A: *A neural network is this super confusing thing that is kind of like an artificial brain.*

Q: Why do we care?

A: *It can learn anything, do anything, and no one really understands them.*

Neurons

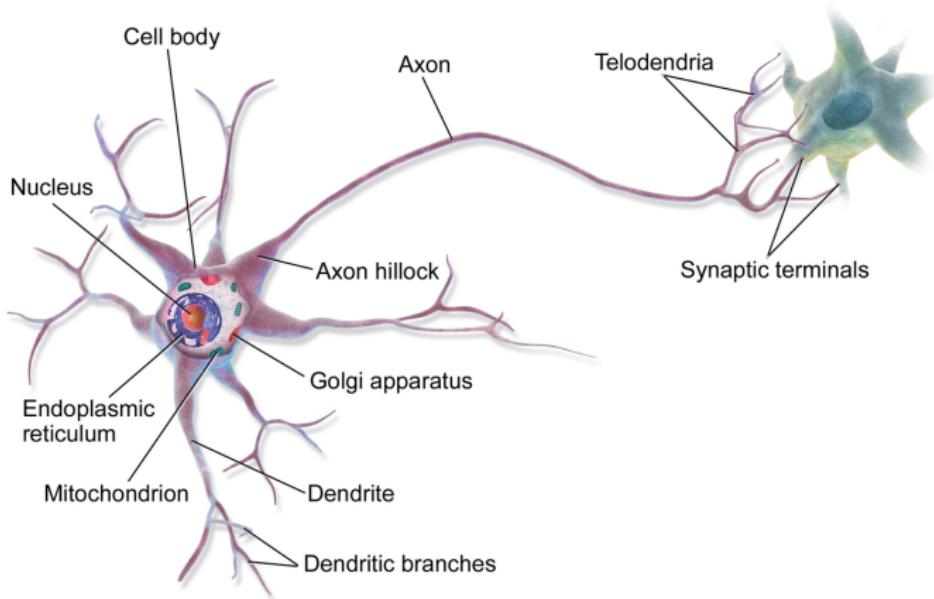


Figure: A neuron. These things are involved somehow?

Pictures that look like this

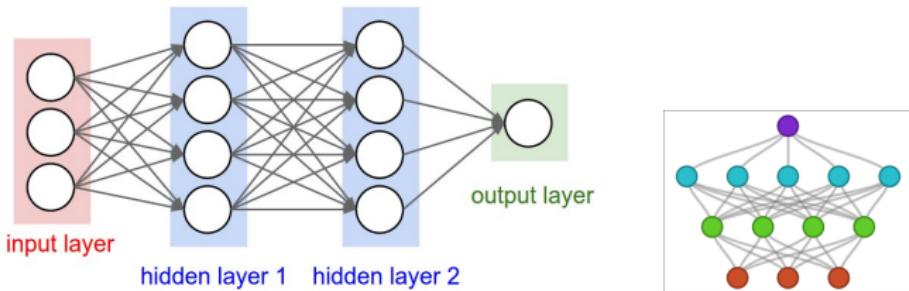


Figure: Some diagrams with circles and arrows.

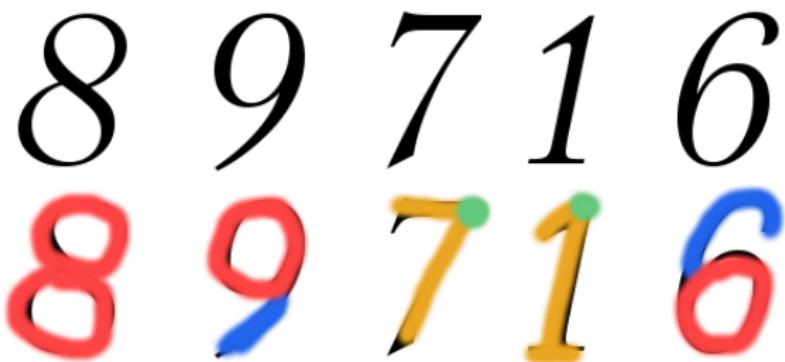
A Motivating Example: Classifying Digits

Suppose you were the kind of person who wanted to identify some pictures of numerical digits.

8 9 7 1 6

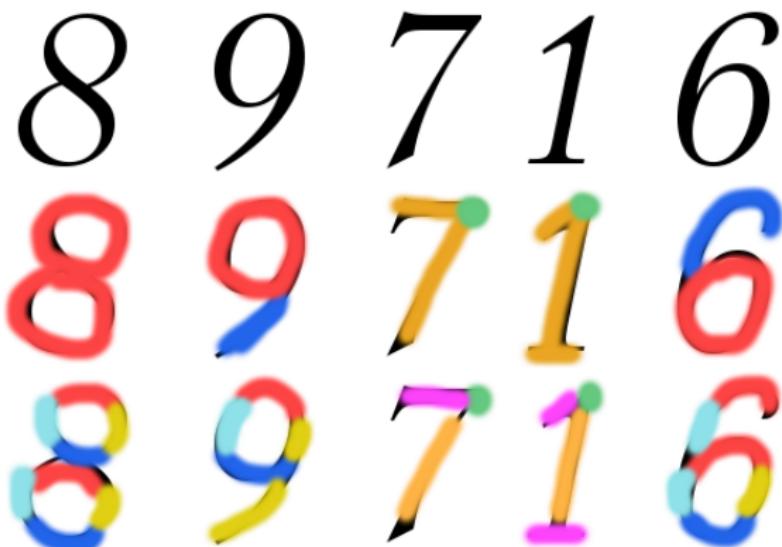
A Motivating Example: Classifying Digits

Suppose you were the kind of person who wanted to identify some pictures of numerical digits.



A Motivating Example: Classifying Digits

Suppose you were the kind of person who wanted to identify some pictures of numerical digits.



The Σexy Part of the Talk

Let's take a look at the math under the hood:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\hat{y} = \hat{F}(x) = W_2\sigma(W_1\sigma(W_0x + b_0) + b_1) + b_2$$

where W_i is a weight matrix, b_i is a bias vector, and σ is an activation function.

Breaking that down

Let's look at that again, step by step:

$$z_0 := W_0x + b_0$$

$$a_0 := \sigma(z_0)$$

$$z_1 := W_1a_0 + b_1$$

$$a_1 := \sigma(z_1)$$

⋮

$$\hat{y} := W_n a_{n-1} + b_n$$

Universal Approximation Theorem

Theorem 1 (Cybenko (1989), Hornik (1991))

Let σ be a nonconstant, bounded, and monotonically increasing function. Given any $\varepsilon > 0$ and any continuous function F defined on a compact subdomain Ω of \mathbb{R}^n there exists a constant N , real constants v_i and b_i , and real vectors w_i such that we may define

$$\widehat{F}(x) := \sum_{i=1}^N v_i \sigma(w_i^T \cdot x + b_i)$$

with

$$|F(x) - \widehat{F}(x)| < \varepsilon$$

for all x in Ω .

Okay, fine, I get it. But how do I set those weights?

For M observations (x_i, y_i) , define a loss function

$$\begin{aligned} J(y_i, \hat{y}) &:= \frac{1}{2M} \sum_{i=1}^N (y - \hat{y})^2 \\ &= \frac{1}{2M} \sum_{i=1}^N (y - \hat{F}(x_i))^2 \\ &= \frac{1}{2M} \sum_{i=1}^N (y - \hat{F}(x_i; W, b))^2 \end{aligned}$$

Now we just have to minimize the loss!

The Gradient

Recall from multivariable calculus that the *gradient* is a vector which points in the direction of steepest ascent on a surface.

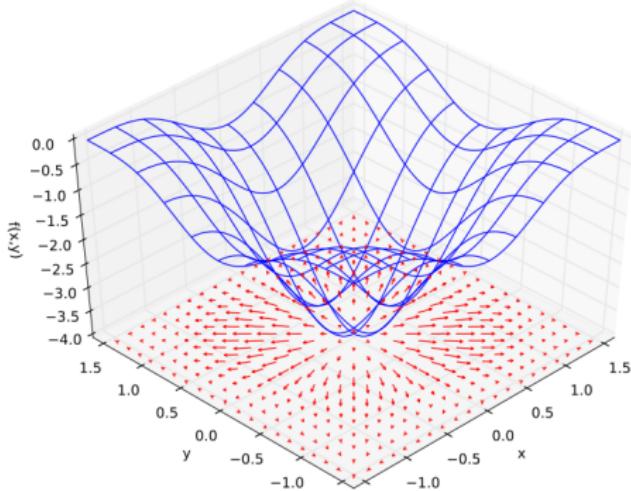


Figure: A function of two variables, with its gradient.

Calculating the Gradient

Normally we would use an algorithm called “Back Propagation¹” to calculate the gradients.

But I just did it this way:

The whiteboard contains several handwritten equations and calculations related to backpropagation:

- A large equation at the top right: $J = \frac{1}{N} \sum_{i=1}^N \left(y_i - \hat{y}_i \right)^2$
- An activation function definition: $\hat{y}_i(x, w_h, b_h, w_o, b_o) = w_o \sigma(w_h x + b_h) + b_o$
- Derivatives of the activation function: $\frac{\partial \hat{y}_i}{\partial w_h} = \sigma'(w_h x + b_h) \cdot x$, $\frac{\partial \hat{y}_i}{\partial b_h} = \sigma'(w_h x + b_h)$, $\frac{\partial \hat{y}_i}{\partial w_o} = \hat{y}_i$, $\frac{\partial \hat{y}_i}{\partial b_o} = 1$
- Derivatives of the error function: $\frac{\partial J}{\partial w_h} = \frac{2}{N} \sum_{i=1}^N (y_i - \hat{y}_i) \cdot \hat{y}_i \cdot x$, $\frac{\partial J}{\partial b_h} = \frac{2}{N} \sum_{i=1}^N (y_i - \hat{y}_i) \cdot \hat{y}_i$, $\frac{\partial J}{\partial w_o} = \frac{2}{N} \sum_{i=1}^N (y_i - \hat{y}_i) \cdot \hat{y}_i \cdot \sigma'(w_h x + b_h) \cdot x$, $\frac{\partial J}{\partial b_o} = \frac{2}{N} \sum_{i=1}^N (y_i - \hat{y}_i) \cdot \hat{y}_i$
- Final gradient calculation: $\frac{\partial J}{\partial w_h} = \frac{2}{N} \sum_{i=1}^N (y_i - \hat{y}_i) \cdot \hat{y}_i \cdot \sigma'(w_h x + b_h) \cdot x = \frac{2}{N} \sum_{i=1}^N (y_i - \hat{y}_i) \cdot \hat{y}_i \cdot [w_o \sigma(w_h x + b_h)] \cdot x$

¹Or as I like to call it, the chain rule with clever bookkeeping.

The Game Plan

- ▶ We start by setting the weights and biases of the NN randomly.
- ▶ Calculate the gradient of the loss function J with respect to these parameters.
- ▶ Update the parameters a tiny bit by taking a step in the direction of the negative gradient.
- ▶ Repeat until the error is acceptable.

WE'LL DO IT LIVE



WE'LL DO IT LIVE!

Concluding Remarks

- ▶ Neural networks are powerful because they combine the ability to approximate any function with a systematic method for improving that approximation.
- ▶ The loss function measures how well a given NN matches the desired function.
- ▶ Gradient descent is the process by which we slowly nudge the parameters of the network in the proper direction in an attempt to minimize the loss function.

Thank you! :)