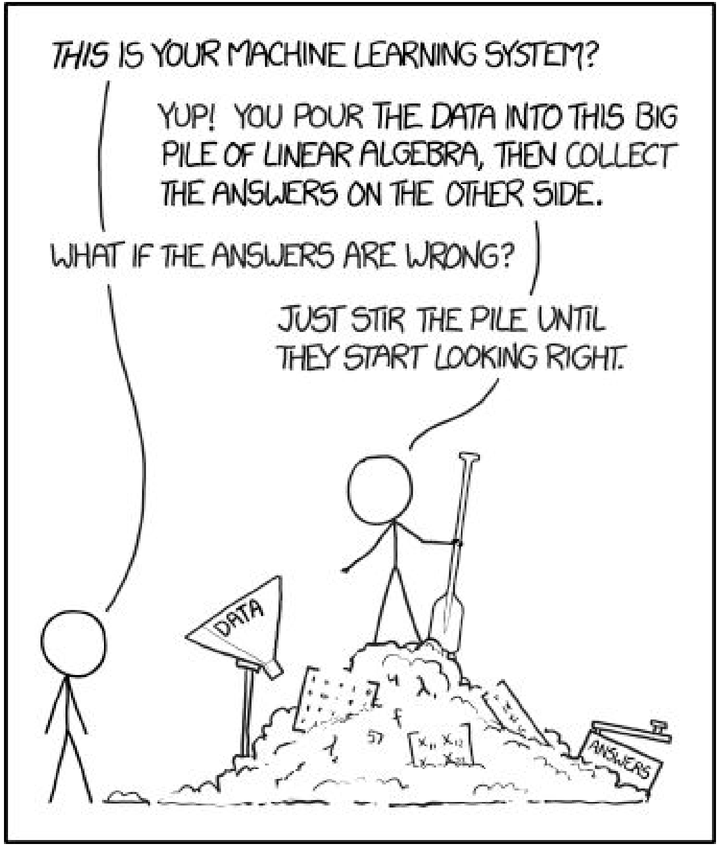
The Mathematics Behind Neural Networks

Pattern Recognition and Machine Learning
by Christopher M. Bishop

Student: Shivam Agrawal

Mentor: Nathaniel Monson



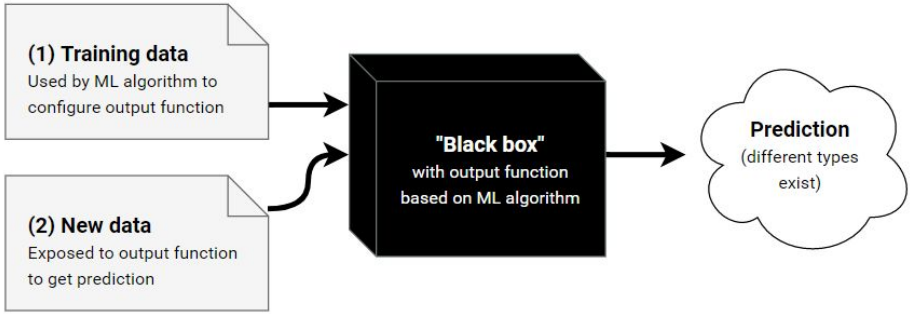
Courtesy of xkcd.com

The Black Box

“Training” the network tunes a network function.

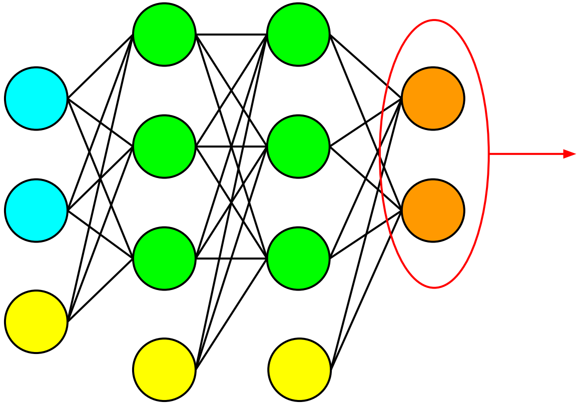
This **network function** is used to approximate functions that we believe model some data.

For example, we believe that whether a picture has a dog or a cat is modeled by some function, and we train NN’s to approximate this function.

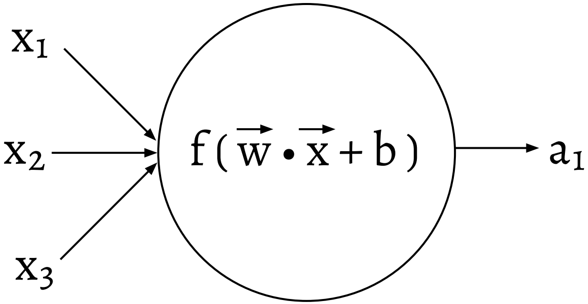


Structure of a NN

- Input Layer
- Hidden Layers
- Output Layers

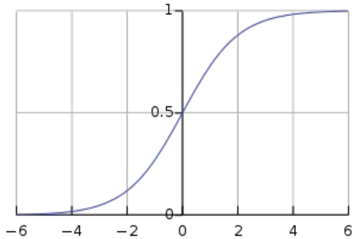


What the output layer spits out is considered the value of the network function for the given input

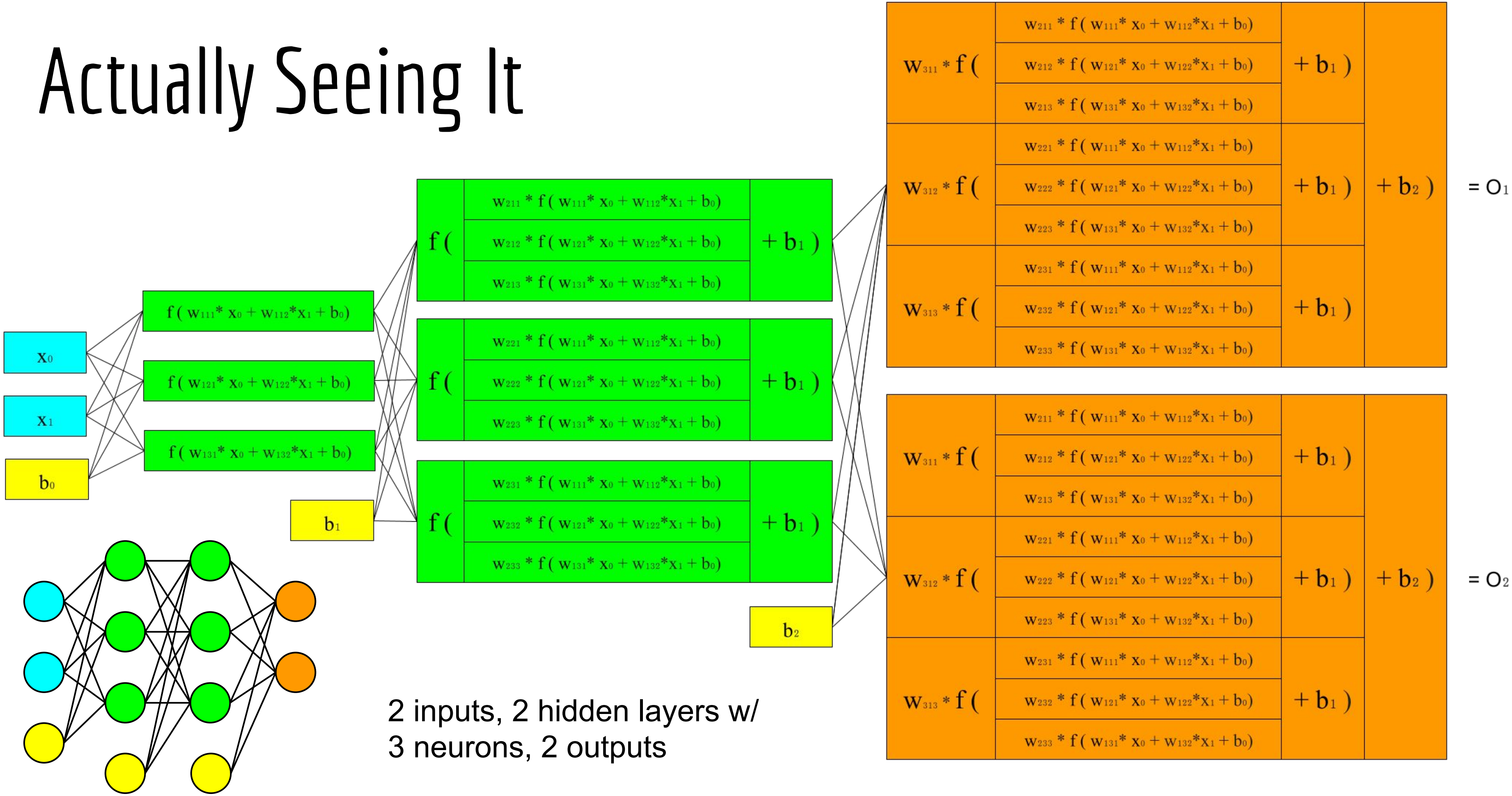


Individual neurons make up layers

- Input - *Vector of activations of previous layer*
- Weights - *Vector for linear transformation of input*
- Bias - *To shift function*
- Activation Function - *Applies nonlinear transformation*
- Activation - *Output of neuron*



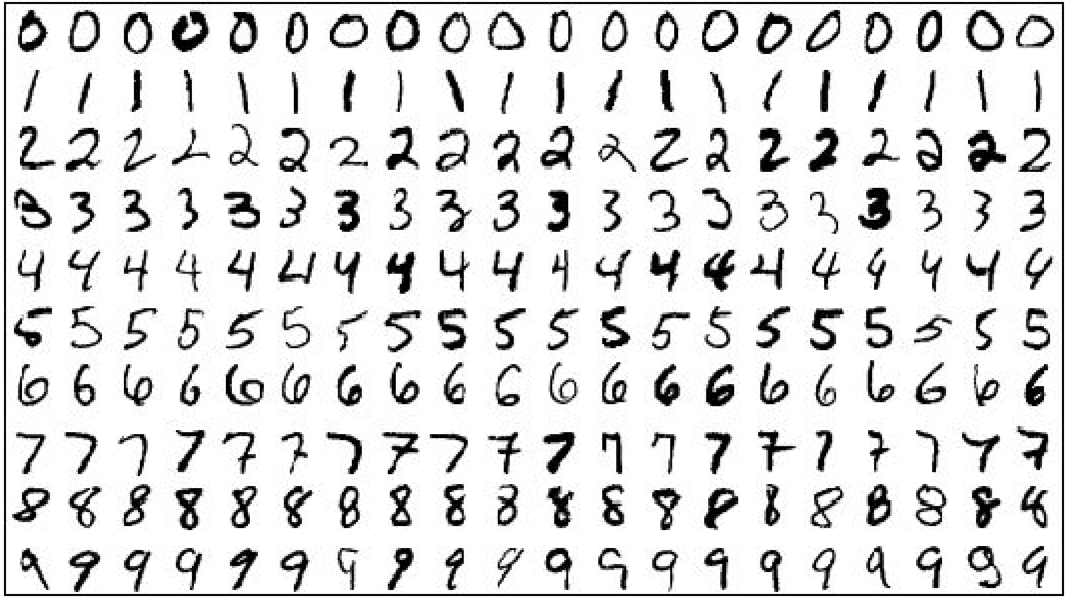
Actually Seeing It



2 inputs, 2 hidden layers w/
3 neurons, 2 outputs

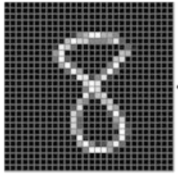
Application to Digit Recognition

We believe there is some function out there, that if we give it a picture of a digit represented as a vector, that it can **classify** the picture as either 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9



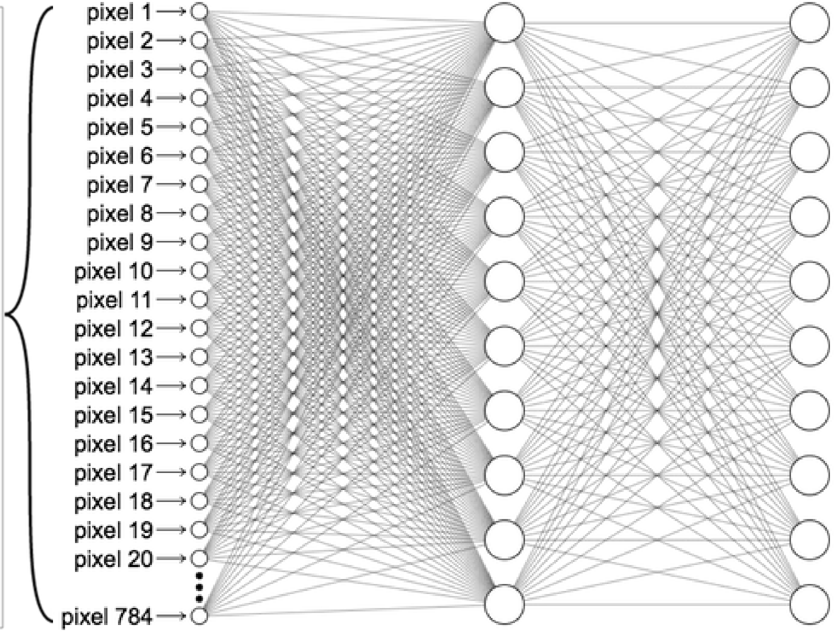
The Network Structure

784 inputs, 1 hidden layer with 10 neurons, output layer with 10 neurons corresponding to probability that the image is of the (n-1)th digit for the nth neuron



28 x 28
784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Optimizing the Network

But how do we set the weights and biases and the number of layers and the number of neurons and…(the list goes on and on) → We train the network

Learnable Parameters
Parameters that the network learns over the training period

- Weights
- Biases

Hyper Parameters
Parameters that humans must set before the network training begins

- Structure of the network (number of layers, number of neurons per layer, activation function)
- Learning Rate
- Error Function

Step 1 of Training: How Wrong is It?

To optimize the network, we need to quantify how wrong the network is:

Cost Function *(AKA Error Function)* → A function of the input, weights, and bias

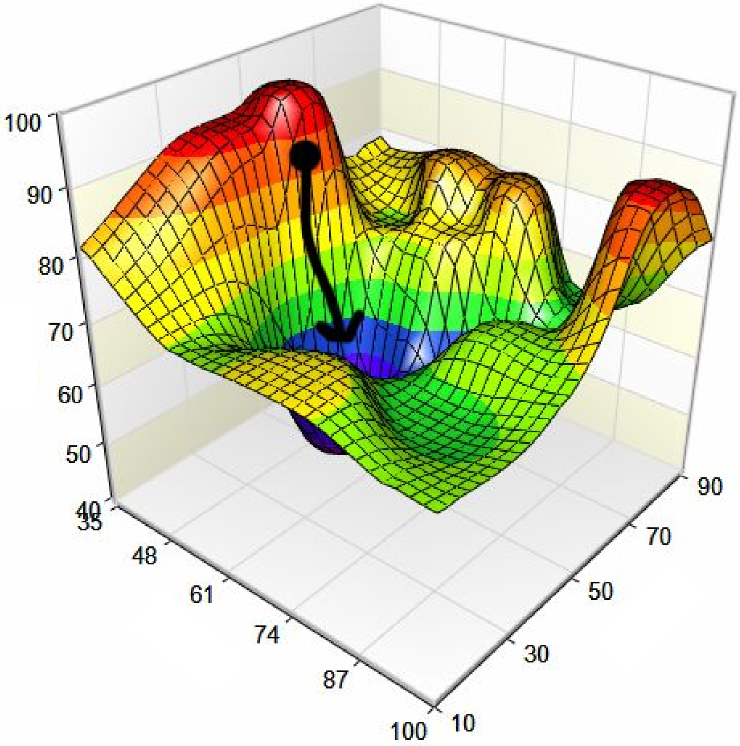
$$E(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} ||y_{pred} - y_{actual}||^2$$

Step 2 of Training: Minimize How Wrong It is!

Error function is a function of input, weights, and biases. Inputs are constant, but weights and biases can be changed.

Say we want to move from point A on the function to point B such that the value of the error function is lower at B than at A.

How? → Move in the direction **OPPOSITE** the gradient. (gradient with respect to weights of a layer and gradient with respect to biases of a layer)



Descending The Curve → Gradient Descent

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \nabla C$$

Where ***w*** is the weight matrix for a layer

t is the index of the iteration

n is the learning rate and **∇C** is the cost with respect to the weights of this layer

Opposite direction → negative sign

Same idea for biases

Types of Gradient Descent

Different Ways - Batch works best in practice considering training time vs accuracy

- Over entire data set - batch gradient descent
- Over portion - mini-batch gradient descent
- Over single input - stochastic gradient descent

$$\nabla C = \frac{1}{N}\sum_{i=1}^{N} \nabla C_i$$

$$\nabla C = \frac{1}{n}\sum_{i=1}^{n} \nabla C_i$$

$$\nabla C = \nabla C_i$$

Calculating the Gradient

- Using Limits
- Backpropagation

Calculate Using Limits

$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \frac{C_2 - C_1}{2\epsilon}$$

Calculate cost after altering a single weight

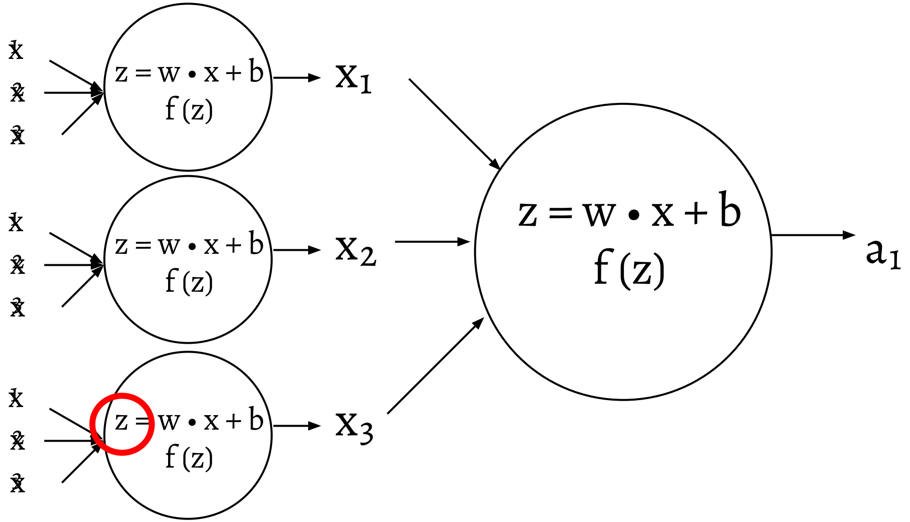
Calculate C_2 with $w_{ij}^{(l)} \rightarrow w_{ij}^{(l)} + \epsilon$

Calculate C_1 with $w_{ij}^{(l)} \rightarrow w_{ij}^{(l)} - \epsilon$

Same idea for biases

Calculate Using Backprop

Intuition: Any change in a neuron will impact the output of the entire layer, which then impacts the output of the next layer, and keeps going till the end of the network → the error propagates forward.



$$\delta^L = \nabla C \odot a^L$$

$$\delta^{(l)} = ((w^{(l+1)})^T \delta^{(l+1)}) \odot f'(z^{(l)})$$

The weights are a scaling factor for the previous layer's output

A value to represent the error in the rest of the network, caused by the change in z

How the change in z affected output of neuron, i.e. the change in activation

Nuances

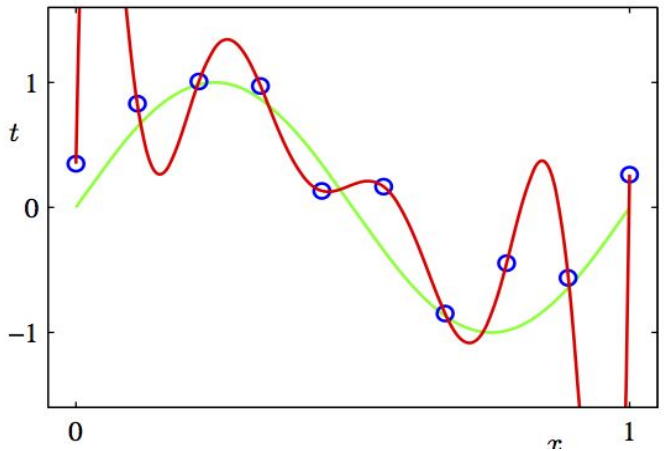
Neural network requires a significant amount of human input:

- Structure of network (number of layers, number of neurons, etc.)
- Choice of Cost function
- Choice of Activation function
- Optimization technique
- Learning rate

Hyperparameters → parameters set before learning and remain constant

Problem: How to pick the right hyperparameters → Good news

Overfitting → Risk of creating a network function that predicts extremely well for training data, but extremely poor on test data.



Network Training Results

Network Parameters:

- Number of Epochs: 30000
- Batch Size: 100
- Learning Rate: 0.5

Results: Around 75% accuracy → this is really bad! (for MNIST)

Future

75% accuracy may be good for certain situations, but it is really bad when the data is as “easy” as MNIST - all the digits are centered, there is little deviation between labels (only so many ways to write a digit), and there is a lot of data.

To improve this accuracy:

- Optimize hyperparameters - there are algorithms that can change the learning rate over time, deactivate/activate neurons to prevent overfitting, etc.
- Use a Convolutional Neural Network - Specifically for images, “3D” connectivity between neurons.

Questions?

P.S. If you need someone to get you coffee everyday, or possibly translate some math into Matlab, Python, Java (soon to include C and R) code, let’s talk! Or drop me a line at ***sagrawa2@terpmail.umd.edu***