CS 6347

Lecture 22

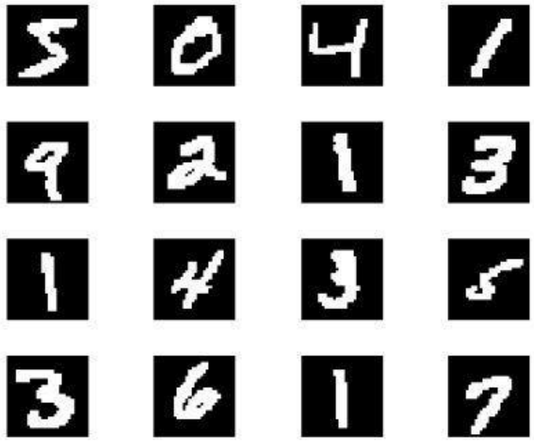
Neural Networks

Classification Problems

- We've been focusing primarily on two different types of learning problems
  - Classification: given a collection of labelled data for training, correctly predict the label of unseen/unlabelled data
  - Structured prediction
- Many natural machine learning tasks can be formulated as classification problems

Handwritten Digit Recognition

- Given a collection of handwritten digits and their corresponding labels, we'd like to be able to correctly classify handwritten digits
  - A simple algorithmic technique can solve this problem with 95% accuracy
    - This seems surprising, in fact, state-of-the-art methods can achieve near 99% accuracy (you've probably seen these in action if you've deposited a check recently)



Digits from the MNIST data set

Neural Networks

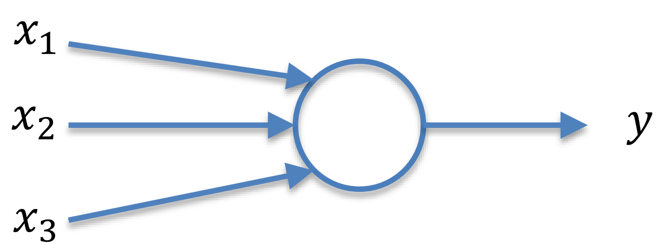
- The basis of neural networks was developed in the 1940s -1960s
  – The idea was to build mathematical models that might “compute” in the same way that neurons in the brain do
  – As a result, neural networks are biologically inspired, though many of the algorithms that are used to work with them are not biologically plausible

Neural Networks

- Neural networks consist of a collection of artificial neurons
- There are different types of neuron models that are commonly studied
  - The perceptron (one of the first studied)
  - The sigmoid neuron (most common)
- A neural network is typically a directed graph consisting of a collection of neurons (the nodes in the graph), directed edges (each with an associated weight), and a collection of fixed binary inputs

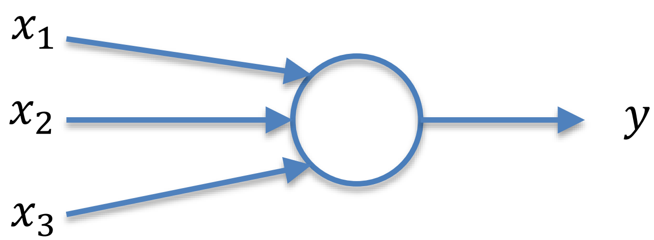
The Perceptron

- A perceptron is an artificial neuron that takes a collection of binary inputs and produces a binary output
  - The output of the perceptron is determined by summing up the weighted inputs and thresholding the result: if the weighted sum is larger than the threshold, the output is one (and zero otherwise)



$$y = \begin{cases} 1 & w_1 x_1 + w_2 x_2 + w_3 x_3 > threshold \\ 0 & otherwise \end{cases}$$

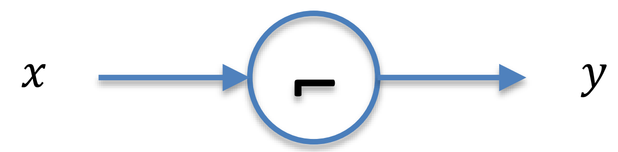
The Perceptron



$$y = \begin{cases} 1 & w_1x_1 + w_2x_2 + w_3x_3 > threshold \\ 0 & otherwise \end{cases}$$

- The weights can be both positive and negative
- Many simple decisions can be modeled using perceptrons
  - Example: AND, OR, NOT

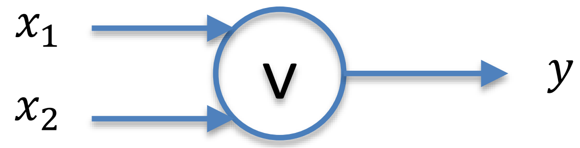
Perceptron for NOT



- Choose $w = -1$, threshold $= -.5$
- $y = \begin{cases} 1 & -x > -.5 \\ 0 & -x \leq -.5 \end{cases}$

Perceptron for OR

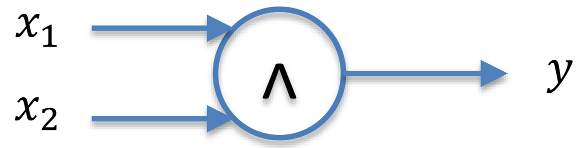
Perceptron for OR



- Choose $w_1 = w_2 = 1$, threshold $= 0$
- $y = \begin{cases} 1 & x_1 + x_2 > 0 \\ 0 & x_1 + x_2 \leq 0 \end{cases}$

Perceptron for AND

Perceptron for AND



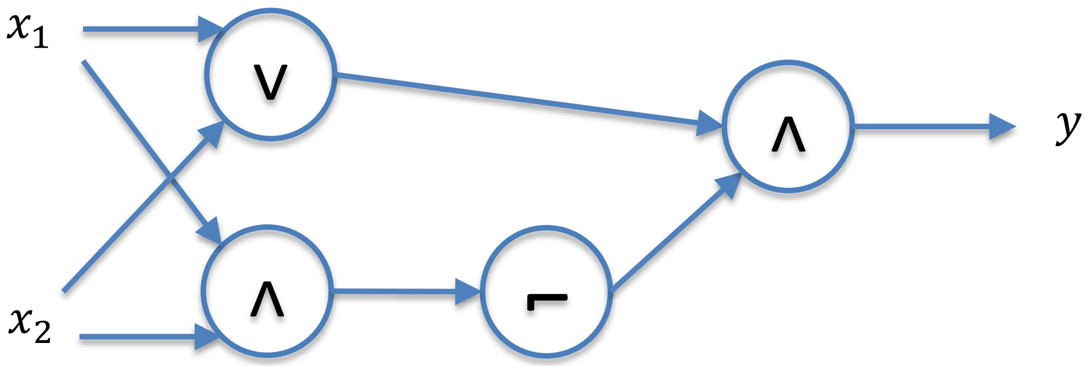
- Choose $w_1 = w_2 = 1$, threshold $= 1.5$

- $y = \begin{cases} 1 & x_1 + x_2 > 1.5 \\ 0 & x_1 + x_2 \leq 1.5 \end{cases}$

Perceptron for XOR

Perceptron for XOR

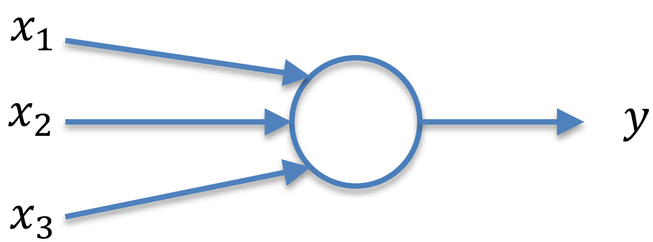
- Need more than one perceptron!



- Weights for incoming edges are chosen as before
  - Networks of perceptrons can encode any circuit!

Perceptrons

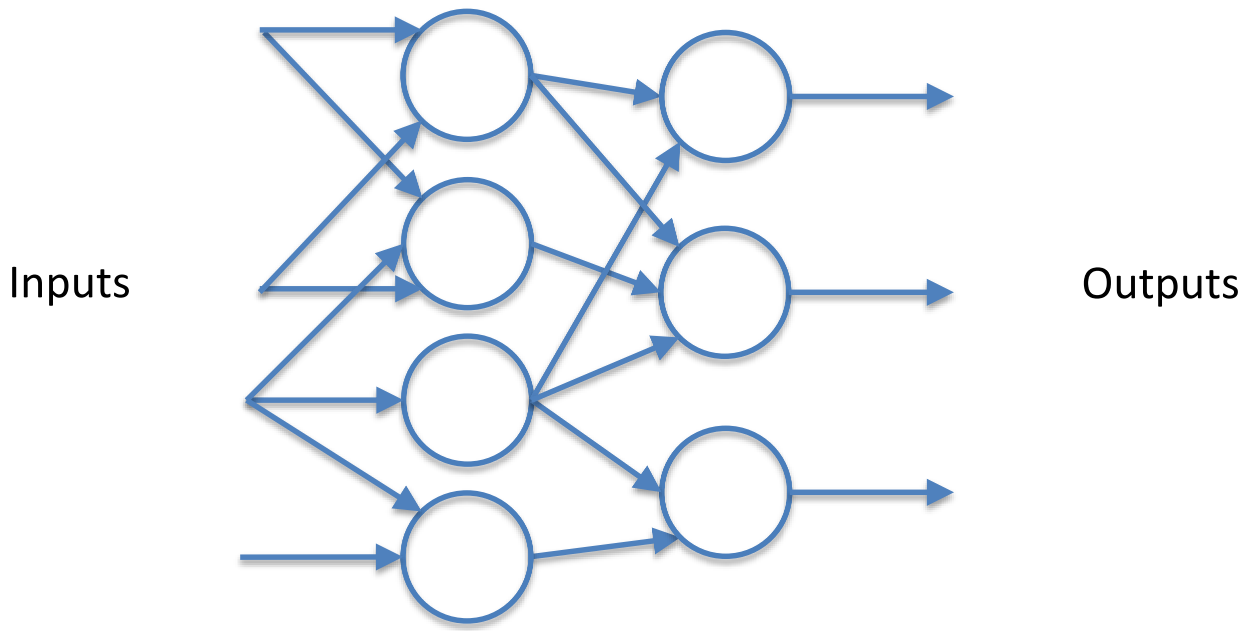
- Perceptrons are usually expressed in terms of a collection of input weights and a bias b (which is the negative threshold)



$$y = \begin{cases} 1 & w_1x_1 + w_2x_2 + w_3x_3 + b > 0 \\ 0 & otherwise \end{cases}$$

Neural Networks

- Gluing a bunch of perceptrons together gives us a neural network
- In general, neural nets have a collection of binary inputs and a collection of binary outputs

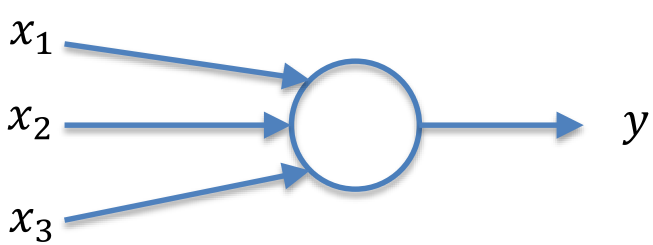


Beyond Perceptrons

- Given a collection of input-output pairs, we'd like to learn the weights of the neural network so that we can correctly predict the ouput of an unseen input
  - We could try learning via gradient descent (e.g., by minimizing the error)
    - This approach doesn't work so well: small changes in the weights can cause dramatic changes in the output
    - This is a consequence of the discontinuity of the sharp thresholding

The Sigmoid Neuron

- A sigmoid neuron is an artificial neuron that takes a collection of inputs in the interval $[0,1]$ and produces an output in the interval $[0,1]$
  - The output is determined by summing up the weighted inputs plus the bias and applying the sigmoid function to the result



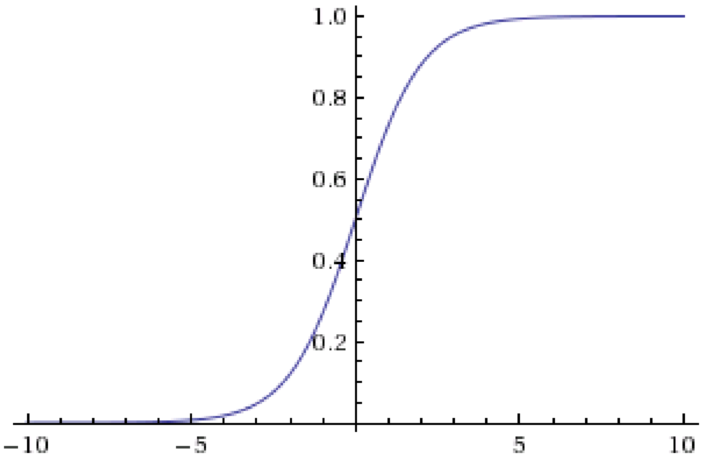
$$y = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$$

where σ is the sigmoid function

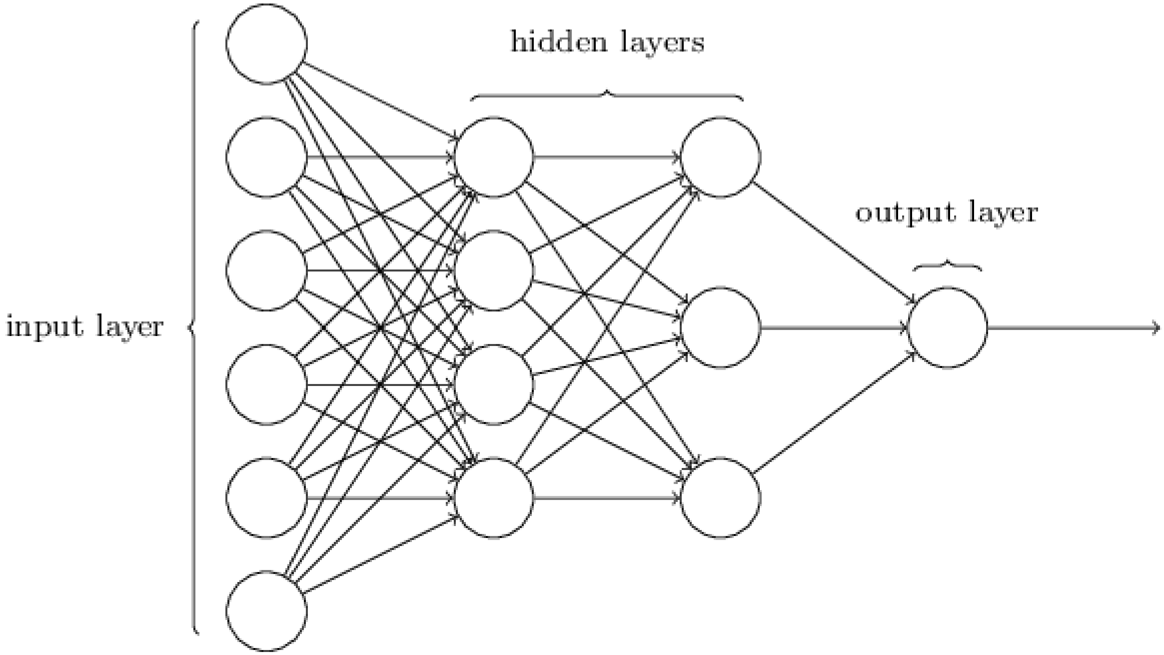
The Sigmoid Function

- The sigmoid function is a continuous function that approximates a step function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



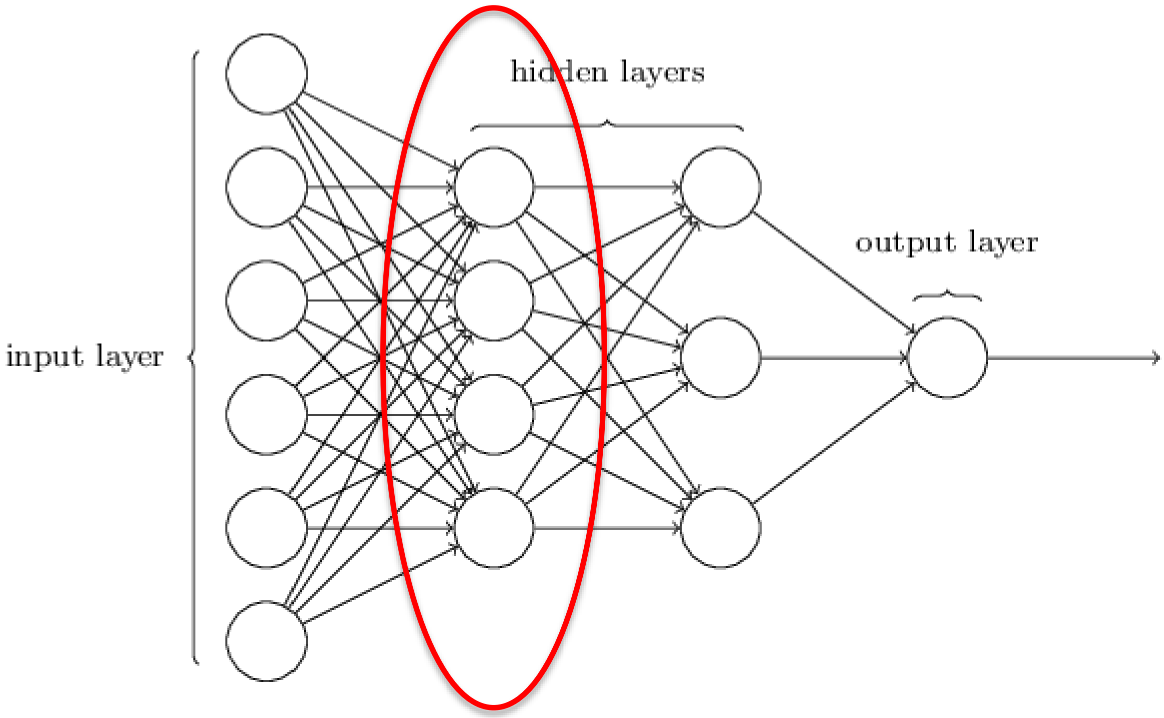
Multilayer Neural Networks



from Neural Networks and Deep Learning by Michael Nielson

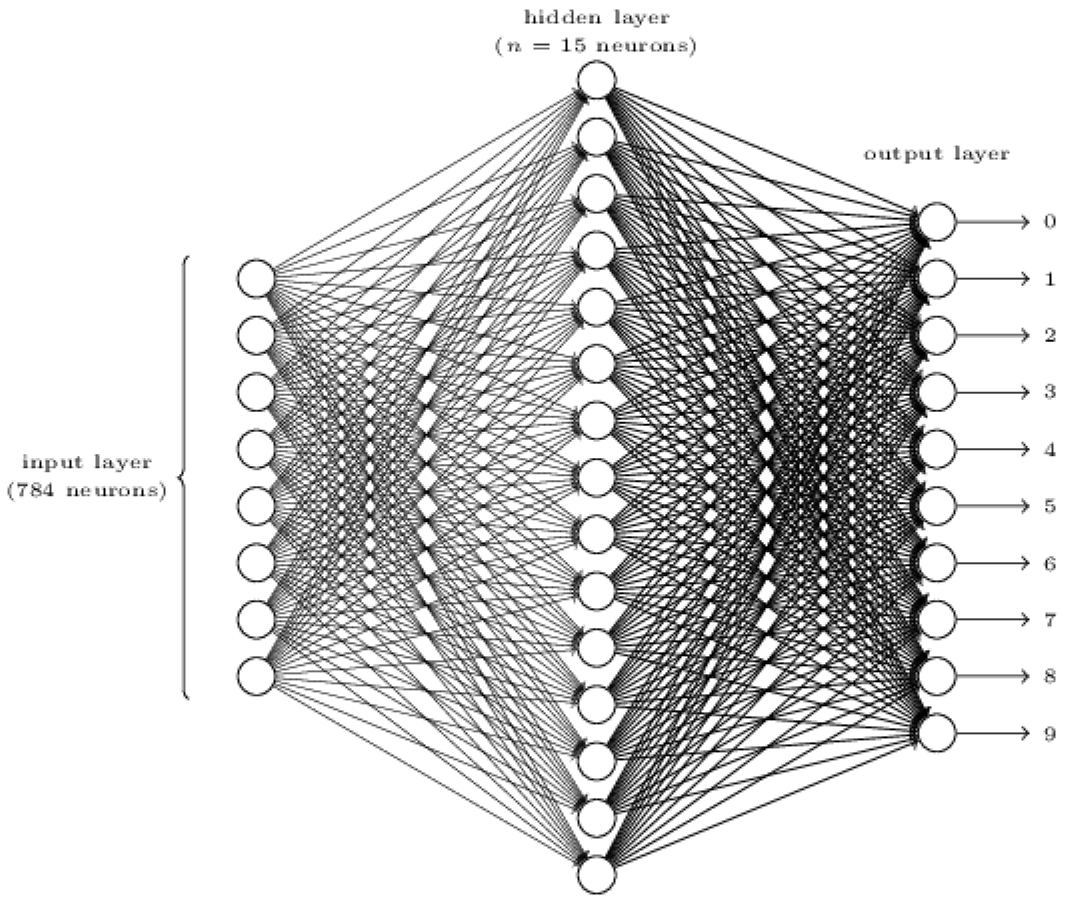
Multilayer Neural Networks

NO intralayer connections



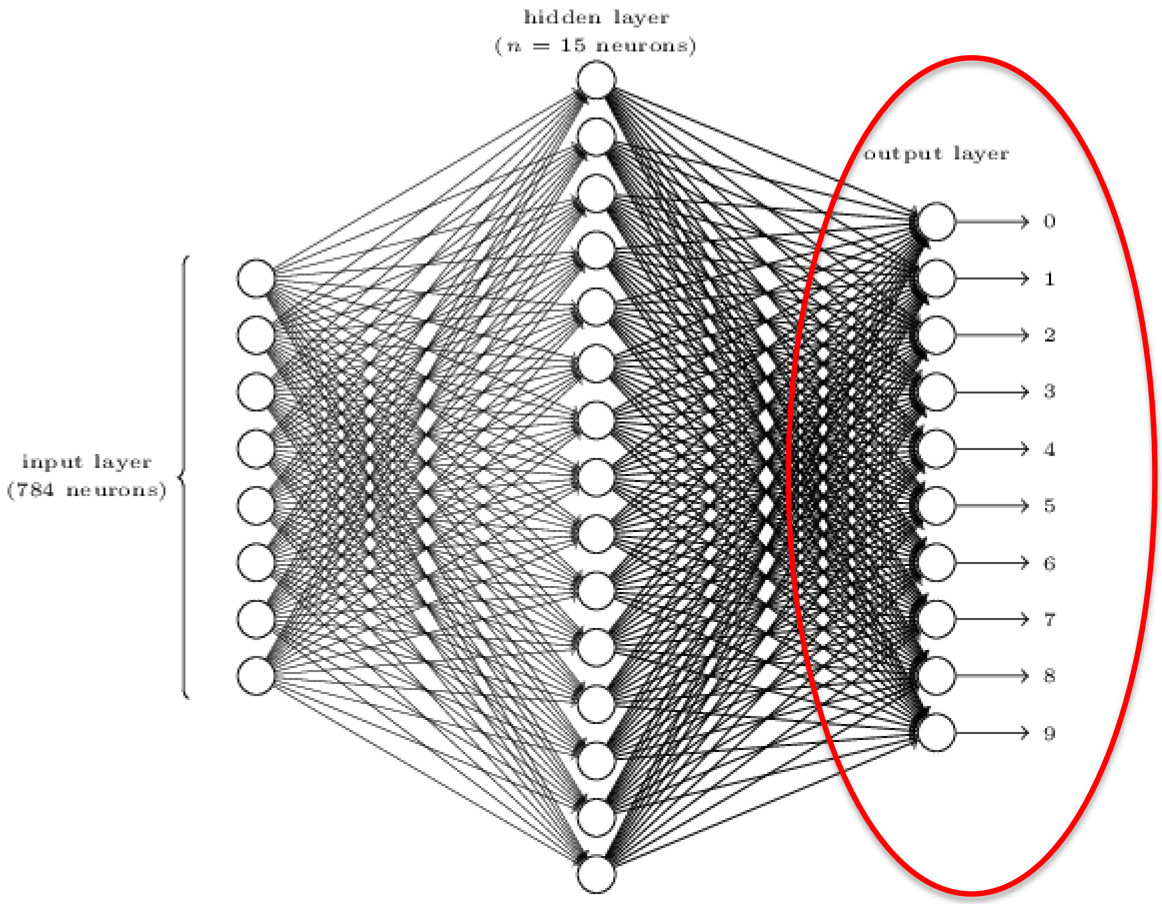
from Neural Networks and Deep Learning by Michael Nielson

Neural Network for Digit Classification



from Neural Networks and Deep Learning by Michael Nielson

Neural Network for Digit Classification



Why 10
instead of 4?

from Neural Networks and Deep Learning by Michael Nielson

Training Neural Networks

- To do the learning, we first need to define a cost function to minimize

$$C(w,b) = \frac{1}{2M} \sum_m \|y^m - a(x^m, w, b)\|^2$$

- The data consists of input output pairs $(x^1, y^1), \ldots, (x^M, y^M)$
- $a(x, w, b)$ is the output of the neural network for the m^{th} sample
- w and b are the weights an biases

UTD

Gradient of the Cost Function

- The derivative of the cost function is relatively straightforward to calculate

$$\frac{\partial C(w,b)}{\partial w_k} = \frac{1}{M}\sum_m \left[y^m - \frac{\partial a(x^m,w,b)}{\partial w_k}\right]$$

  – To compute the derivative of a, use the chain rule and the derivative of the sigmoid function

$$\frac{d\sigma(z)}{dz} = \sigma(z)\cdot(1-\sigma(z))$$

  – This gets complicated quickly with lots of layers of neurons

Stochastic Gradient Descent

- To make the training more practical, stochastic gradient descent is used instead of standard gradient descent
- The idea of stochastic gradient descent is to approximate the gradient of a sum by sampling a few indices uniformly at random and averaging

$$\nabla_x \sum_{i=1}^{n} f_i(x) \approx \frac{1}{K} \sum_{k=1}^{K} \nabla_x f_{i^k}(x)$$

here, each i^k is sampled uniformly at random from $\{1, \ldots, n\}$