Using neural nets to recognize hand-written digits

Srikumar Ramalingam

School of Computing

University of Utah

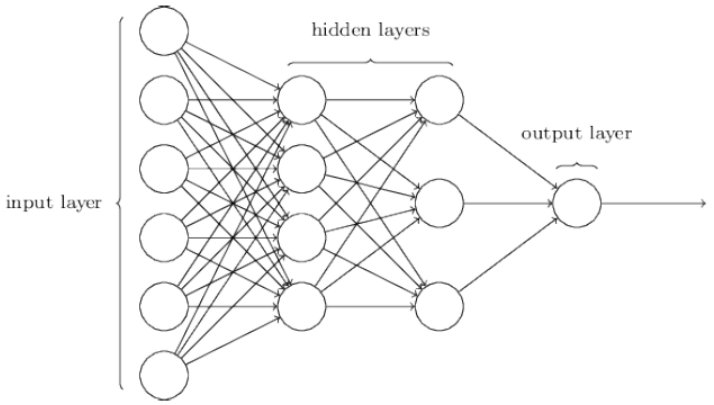
Reference

Most of the slides are taken from the first chapter of the online book by Michael Nielson:

- neuralnetworksanddeeplearning.com

Introduction

- Deep learning allows computational models that are composed of multiple layers to learn representations of data.
- Significantly improved state-of-the-art results in speech recognition, visual object recognition, object detection, drug discovery and genomics.



“deep” comes from having multiple layers of non-linearity

[Source: Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, Deep Learning, Nature 2015]

Introduction

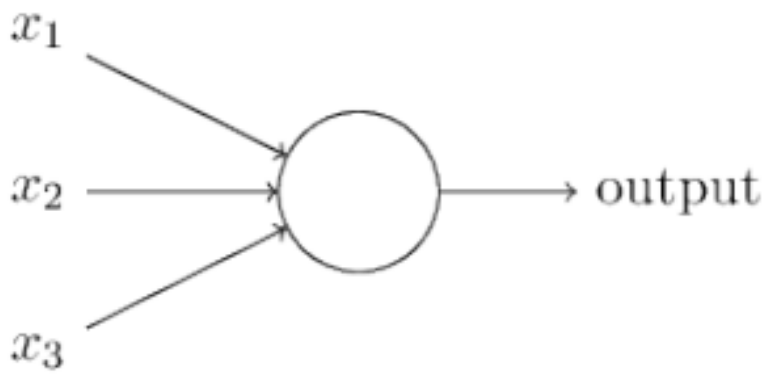
- "neural" is used because it is loosely inspired by neuroscience.

- The goal is generally to approximate some function f^*, e.g., consider a classifier $y = f^*(x)$:

  We define a mapping $y = f(\theta, x)$ and learn the value of the parameters θ that result in the best function approximation.

- Feedforward network is a specific type of deep neural network where information flows through the function being evaluated from input x through the intermediate computations used to define f, and finally to the output y.

Perceptron



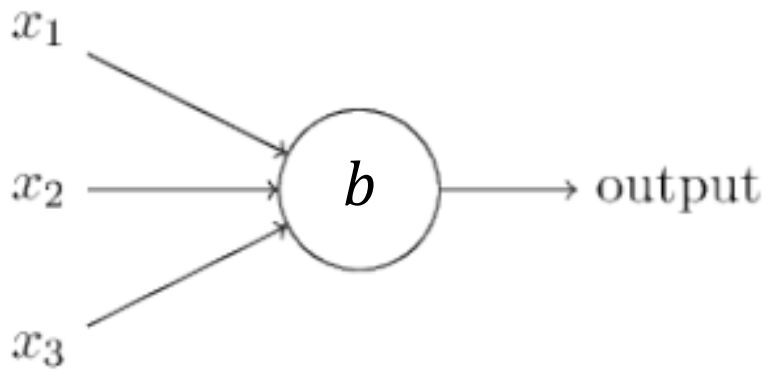
- A perceptron takes several Boolean inputs (x_1, x_2, x_3) and returns a Boolean output.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

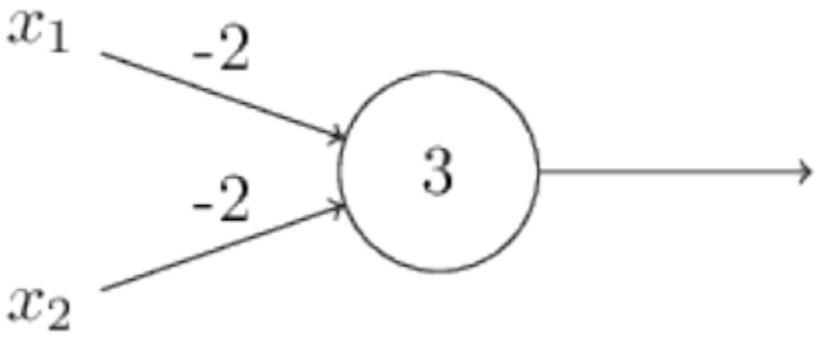
- The weights (w_1, w_2, w_3) and the threshold are real numbers.

Simplification (Threshold -> Bias)

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



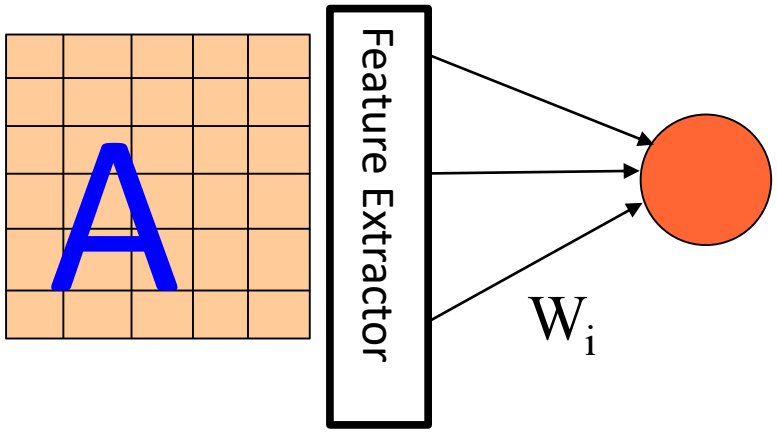
NAND gate using a perceptron



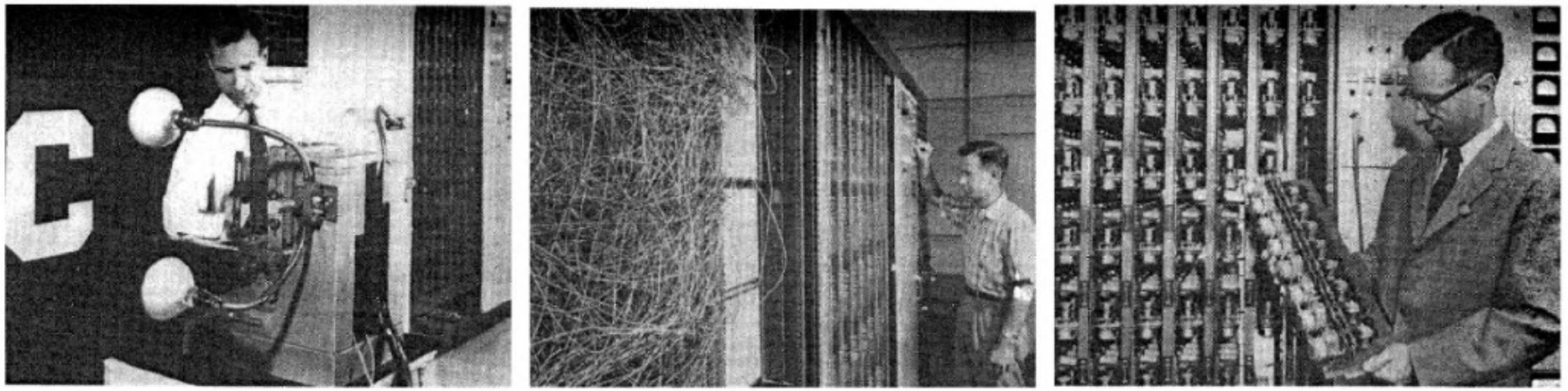
- NAND is equivalent to NOT AND

It's an old paradigm

- The first learning machine: the Perceptron
  - Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.



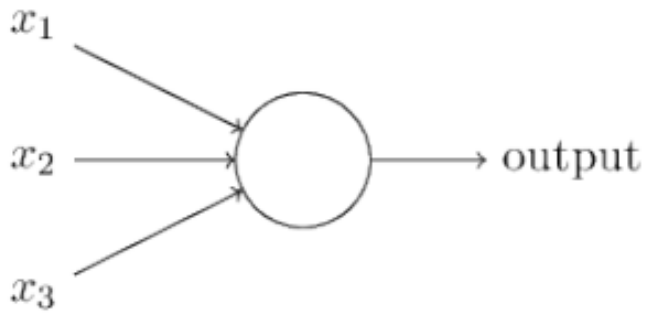
$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Design the weights and thresholds for the following truth table

When all the three Boolean variables are 1s, we output 1, otherwise we output 0.

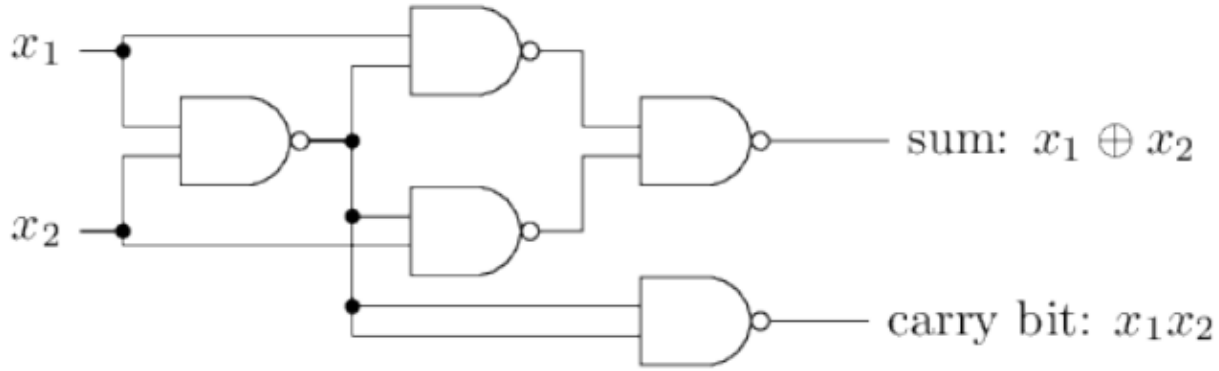


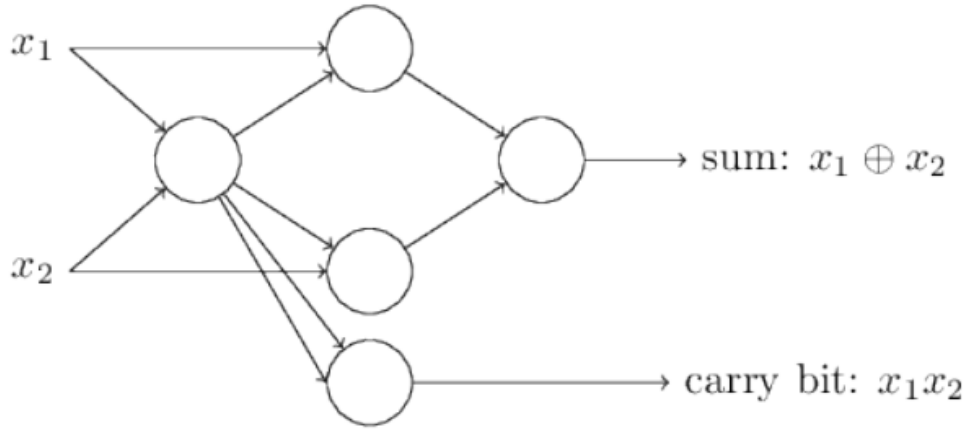
When all the three Boolean variables are 0s, we output 1, otherwise we output 0.

When two of the three Boolean variables are 1s, we output 1, otherwise we output 0.

NAND is universal for computation

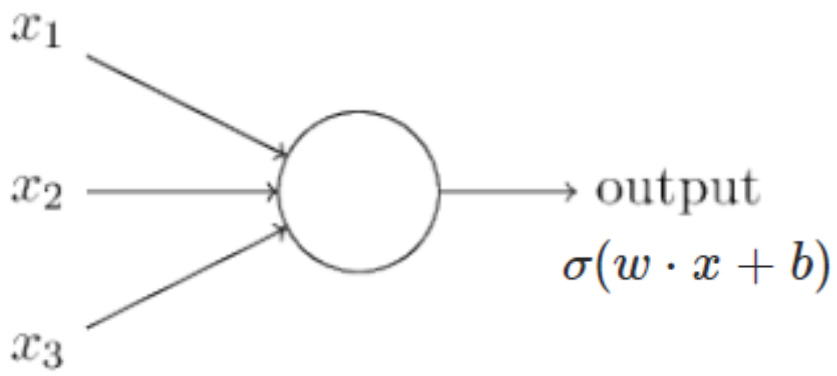
- XOR gate and AND gate





OR gate using perceptrons?

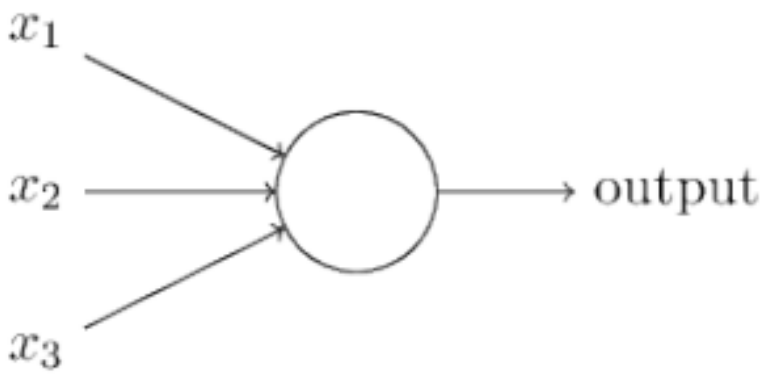
Sigmoid neuron



- A sigmoid neuron can take real numbers (x_1, x_2, x_3) within 0 to 1 and returns a number within 0 to 1. The weights (w_1, w_2, w_3) and the bias term b are real numbers.

Sigmoid function $\sigma(z) \equiv \frac{1}{1+e^{-z}}$

Sigmoid neuron

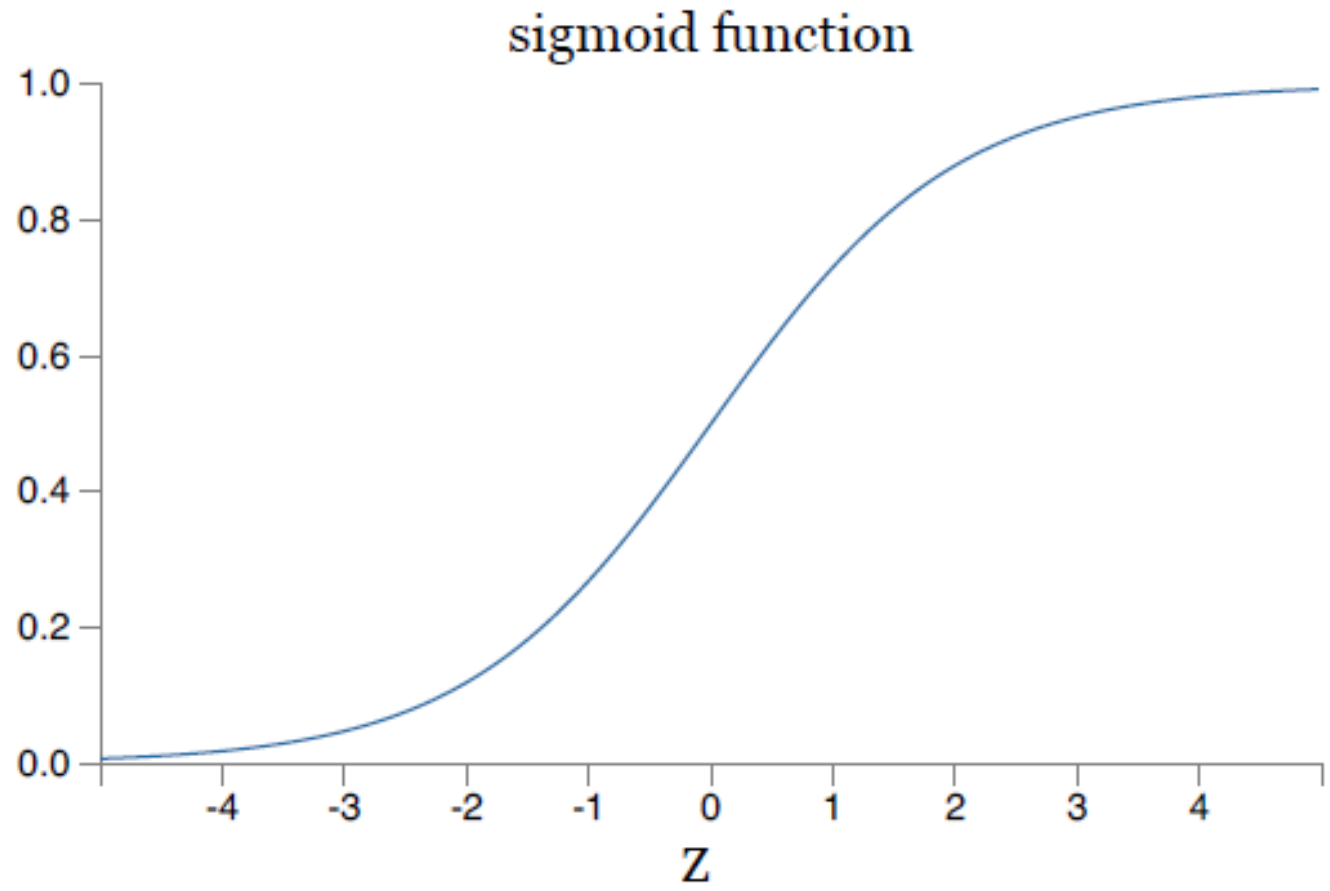


$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$

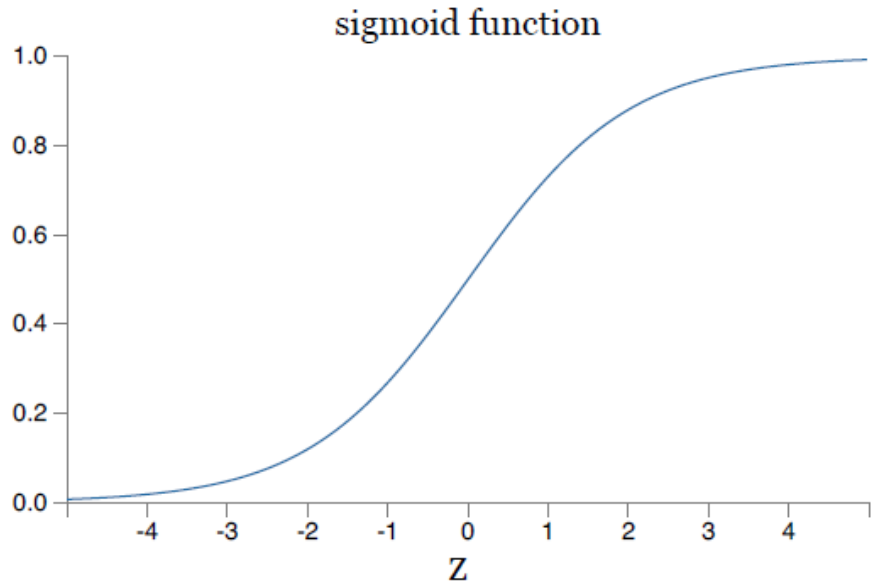
- A sigmoid neuron can take real numbers (x_1, x_2, x_3) within 0 to 1 and returns a number within 0 to 1. The weights (w_1, w_2, w_3) and the bias term b are real numbers.

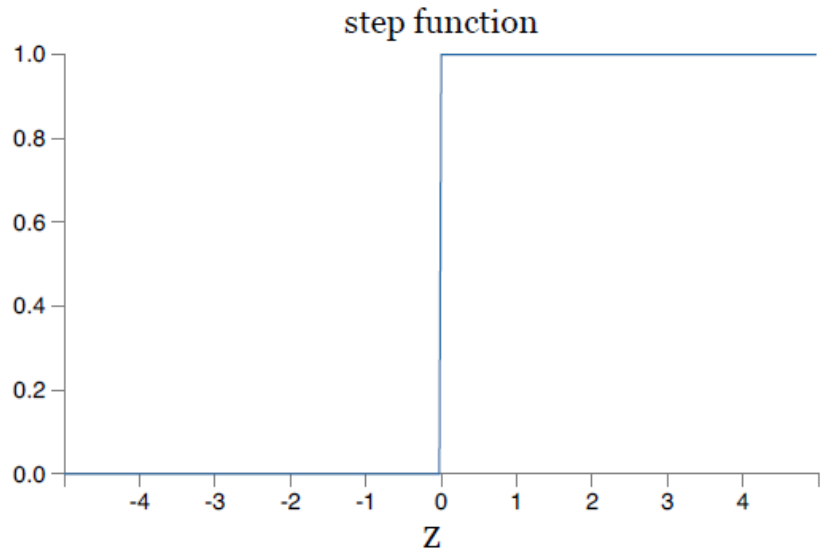
Sigmoid function

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

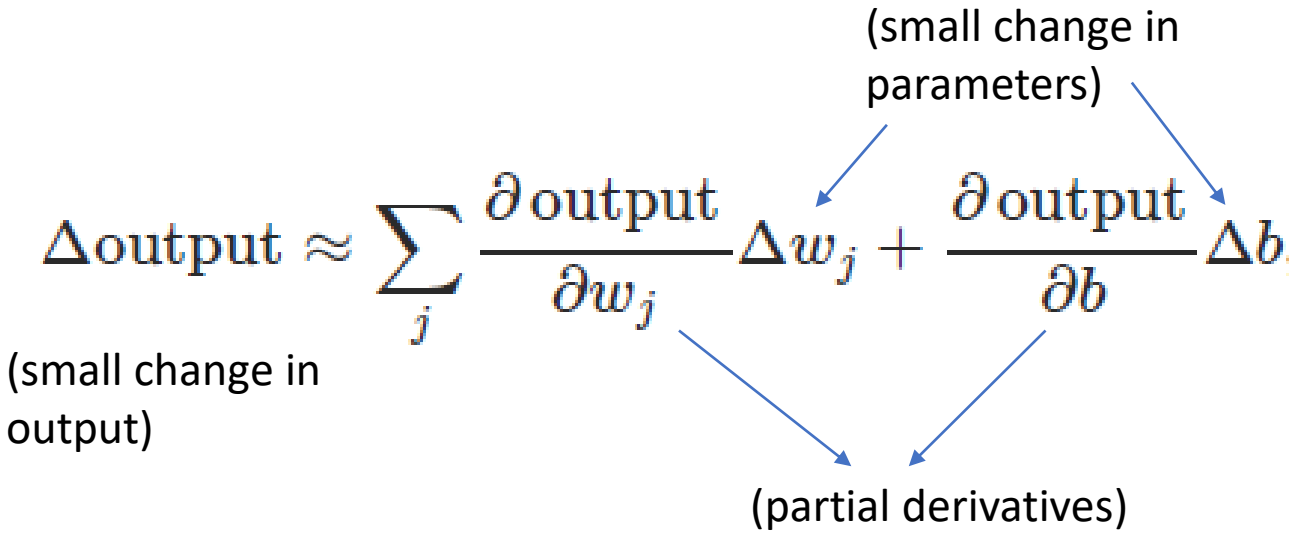


Sigmoid function can be seen as smoothed step function



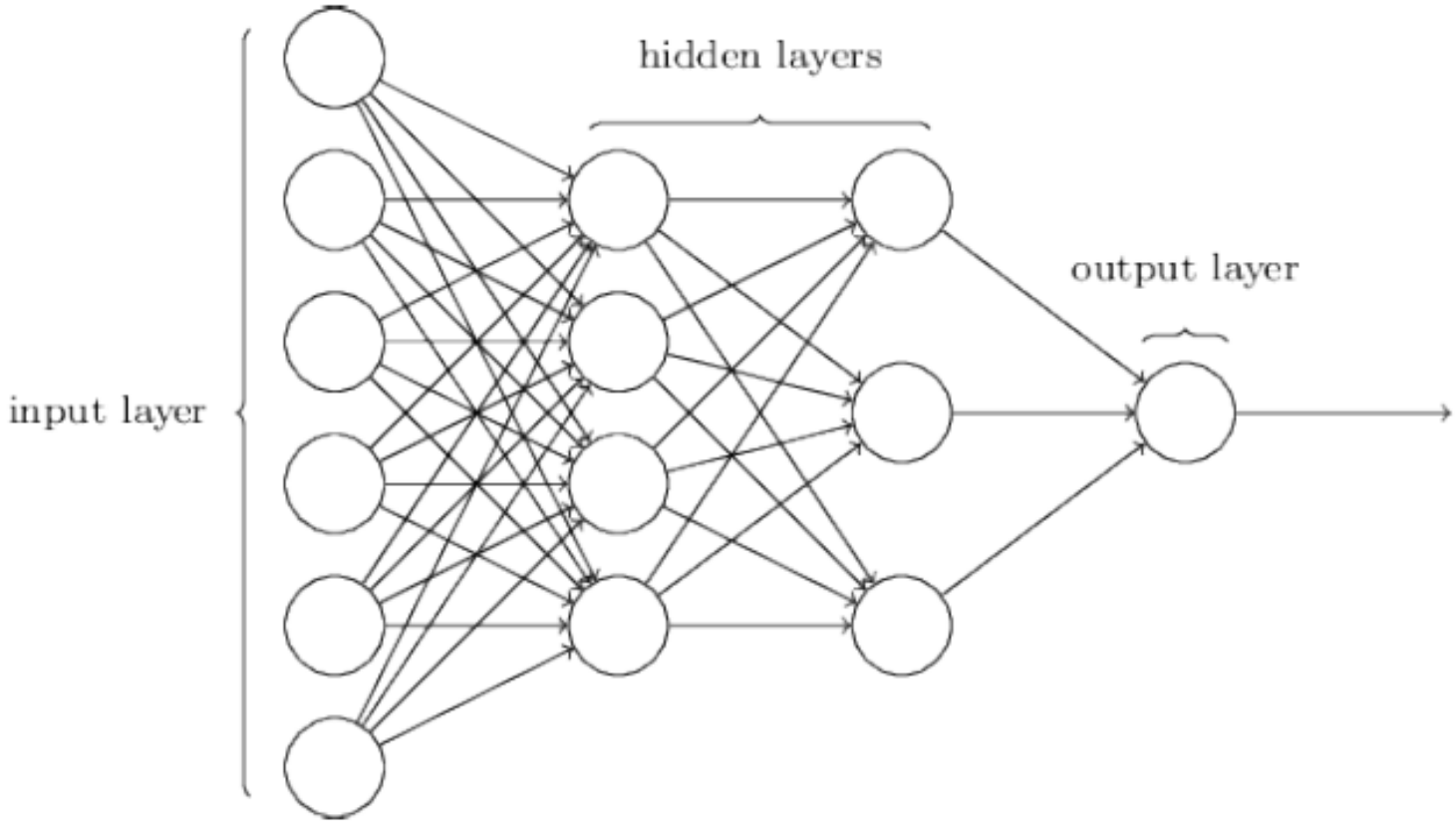


Small changes in parameters produce small changes in output for sigmoid neurons



- $\Delta output$ is approx. a linear function in small changes in weights and bias terms.
- Not for perceptrons!
    - The outputs flip from 0 to 1 or vice versa for small change in inputs.

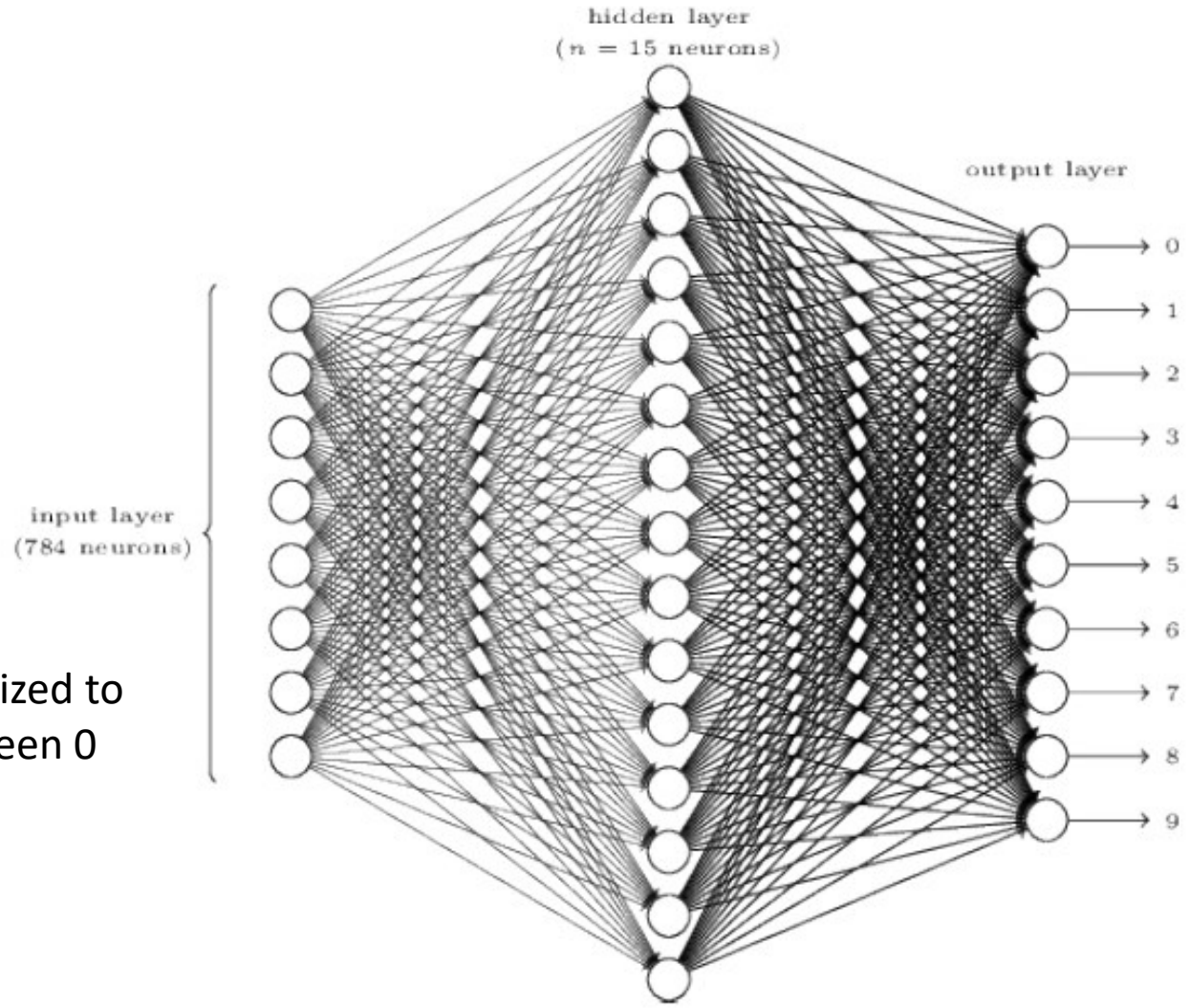
The architecture of neural networks



MNIST data

- Each grayscale image is of size 28x28.
- 60,000 training images and 10,000 test images
- 10 possible labels (0,1,2,3,4,5,6,7,8,9)

Digit recognition using 3 layers

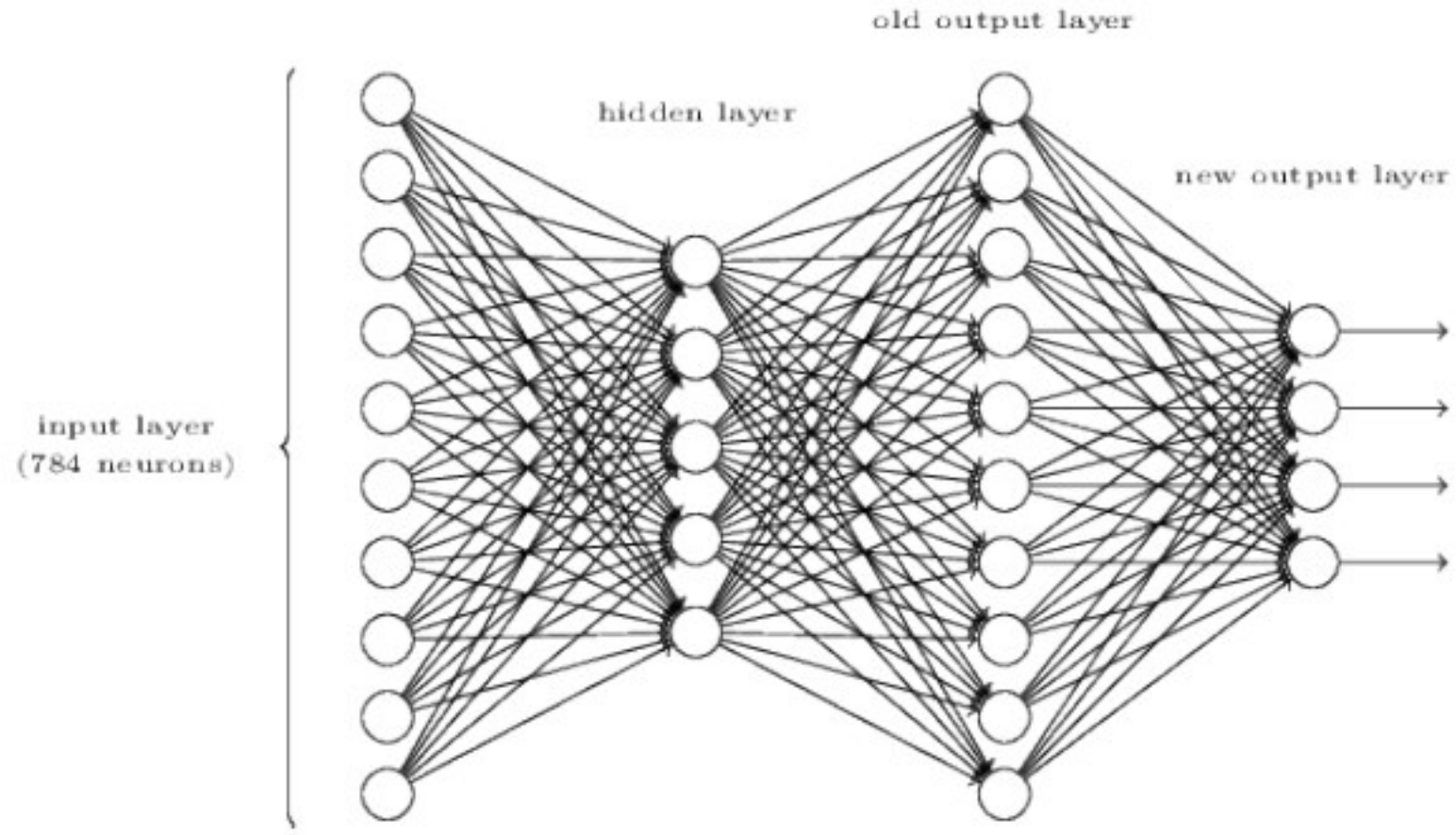


Input normalized to a value between 0 and 1.

Example outputs:

6 ->
[0 0 0 0 0 0 1 0 0 0]’

Compute the weights and biases for the last layer

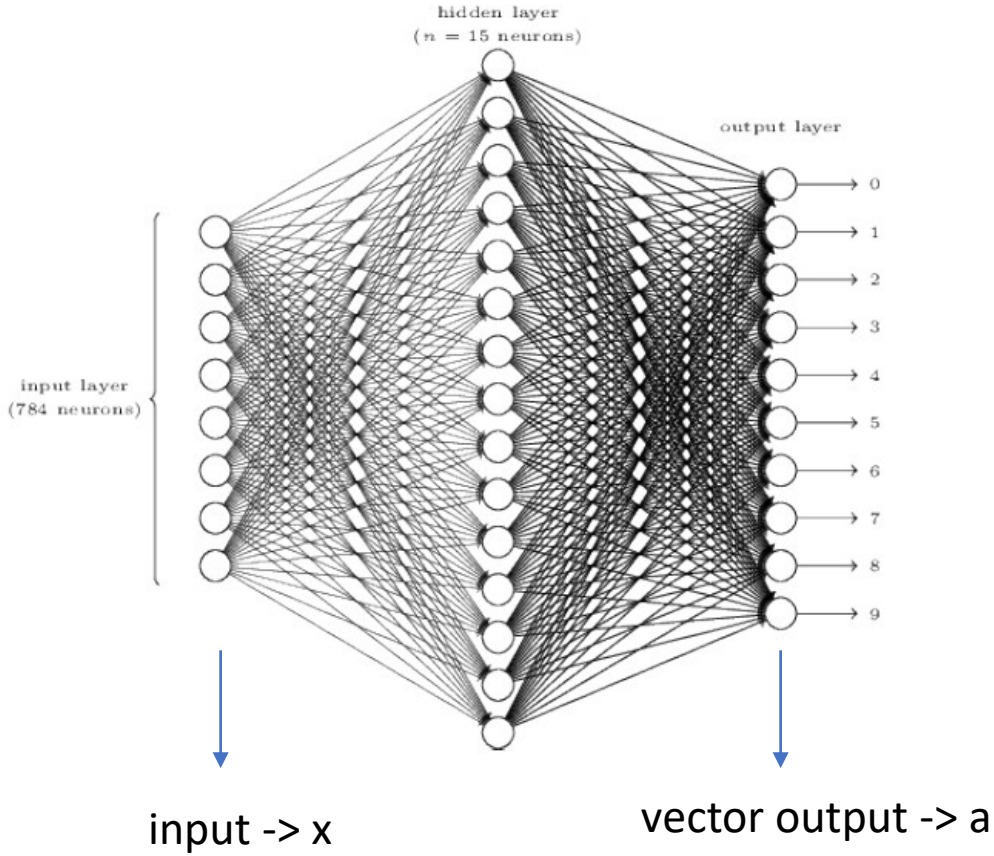


Cost function

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

parameters to compute

# of input samples



input -> x

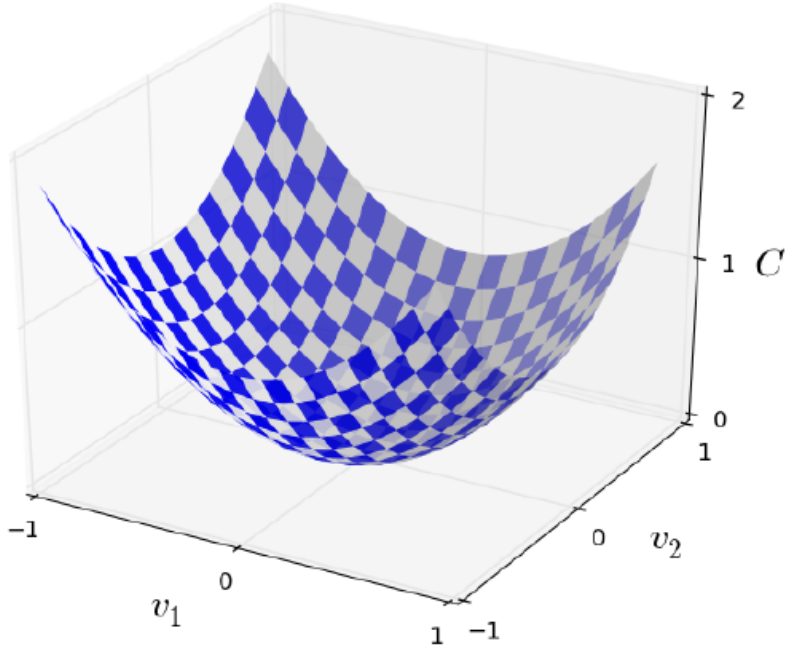
vector output -> a

- We assume that the network approximates a function $y(x)$ and outputs a.
- We use a quadratic cost function, i.e., mean squared error or MSE.

Cost function

- Can the cost function be negative in the above example?
- What does it mean when the cost is approximately equal to zero?

Gradient Descent



$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

Small changes in parameters to leads to small changes in output

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$$

Gradient vector!

$$\Delta v = -\eta \nabla C$$

Change the parameter using learning rate (positive) and gradient vector!

$$v \rightarrow v' = v - \eta \nabla C$$

Update rule!

- Let us consider a cost function $C(v_1, v_2)$ that depends on two variables.
- The goal is to change the two variables to minimize the cost function.

Cost function from the network

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

parameters to compute

# of input samples

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

What are the challenges in gradient descent when you have a large number of training samples?

$\nabla C = \frac{1}{n} \sum_x \nabla C_x$ Gradient from a set of training samples.

Stochastic gradient descent

- The idea is to compute the gradient using a small set of randomly chosen training data.
- We assume that the average gradient obtained from the small set is close to the gradient obtained from the entire set.

Stochastic gradient descent

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

- Let us consider a mini-batch with m randomly chosen samples.

- Provided that the sample size is large enough, we expect the average gradient from the m samples is approximately equal to the average gradient from all the n samples.

Thank You

DERIVATIVE RULES

$$\frac{d}{dx}\left(x^n\right) = nx^{n-1}$$

$$\frac{d}{dx}\left(a^x\right) = \ln a \cdot a^x$$

$$\frac{d}{dx}\left(f(x)\cdot g(x)\right) = f(x)\cdot g'(x) + g(x)\cdot f'(x)$$

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{g(x)\cdot f'(x) - f(x)\cdot g'(x)}{\left(g(x)\right)^2}$$

$$\frac{d}{dx}\left(f(g(x))\right) = f'(g(x))\cdot g'(x)$$

$$\frac{d}{dx}\left(\ln x\right) = \frac{1}{x}$$

$$\frac{d}{dx}\left(\sin x\right) = \cos x$$

$$\frac{d}{dx}\left(\tan x\right) = \sec^2 x$$

$$\frac{d}{dx}\left(\sec x\right) = \sec x \tan x$$

$$\frac{d}{dx}\left(\arcsin x\right) = \frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\left(\operatorname{arc\,sec} x\right) = \frac{1}{x\sqrt{x^2-1}}$$

$$\frac{d}{dx}\left(\sinh x\right) = \cosh x$$

$$\frac{d}{dx}\left(\cos x\right) = -\sin x$$

$$\frac{d}{dx}\left(\cot x\right) = -\csc^2 x$$

$$\frac{d}{dx}\left(\csc x\right) = -\csc x \cot x$$

$$\frac{d}{dx}\left(\arctan x\right) = \frac{1}{1+x^2}$$

$$\frac{d}{dx}\left(\cosh x\right) = \sinh x$$

Source: http://math.arizona.edu/~calc/Rules.pdf

INTEGRAL RULES

$$\int x^n dx = \frac{1}{n+1} x^{n+1} + c, \quad n \neq -1$$

$$\int a^x dx = \frac{1}{\ln a} a^x + c$$

$$\int \frac{1}{x} dx = \ln|x| + c$$

$$\int \frac{dx}{\sqrt{1-x^2}} = \arcsin x + c$$

$$\int \frac{dx}{1+x^2} = \arctan x + c$$

$$\int \frac{dx}{x\sqrt{x^2-1}} = \operatorname{arc\,sec} x + c$$

$$\int \sin x dx = -\cos x + c$$

$$\int \cos x dx = \sin x + c$$

$$\int \sec^2 x dx = \tan x + c$$

$$\int \sinh x dx = \cosh x + c$$

$$\int \csc^2 x dx = -\cot x + c$$

$$\int \sec x \tan x dx = \sec x + c$$

$$\int \csc x \cot x dx = -\csc x + c$$

$$\int \cosh x dx = \sinh x + c$$

Source: http://math.arizona.edu/~calc/Rules.pdf