Probabilistic Machine Learning and AI

Outline of the lecture

This lecture introduces you to the fascinating subject of classification and regression with artificial neural networks.

In particular, it

- What is Machine Learning ?
- Introduces Multi-Later Perceptron (MLPs)
- Teaches you how to combine probability with neural networks so that nets can be applied to regression, binary classification and multivariate classification.
- Describes the relation between energy functions (cost/loss functions) and probabilistic models.

What is Machine Learning

- Pattern Recognition
- Neural Networks
- Data Mining
- Adaptive Control
- Statistical Modeling
- Data Analytics / Data Science
- Artificial Intelligence
- Machine Learning

The view from different fields

Engineering: signal processing, system identification, adaptive and optimal control, information theory, robotics

Computer science: Artificial Intelligence, computer vision, information retrieval, …

Statistics: learning theory, data mining, learning and inference from data

Cognitive Science and Psychology: perception, movement control, reinforcement learning, mathematical psychology, computational linguistics, …

Computational Neuroscience: neural networks, neural information processing

Economics: decision theory, game theory, operational research, …

Different fields, convergent ideas

- The same set of ideas and mathematical tools have emerged in many of these fields, albeit with different emphases.

- Machine learning is an interdisciplinary field focusing on both the mathematical foundation and practical applications of systems that learn reason and act.

An Information Revolution?

We are in an era of abundant data:

Society: the web, social network, mobile networks, government, digital archives

Science: large-scale scientific experiments, biomedical data, climate data, scientific literature

Business: e-commerce, electronic trading, advertising, personalization

We need tools for modeling, searching, visualizing, and understanding large data sets.

Modeling Tools

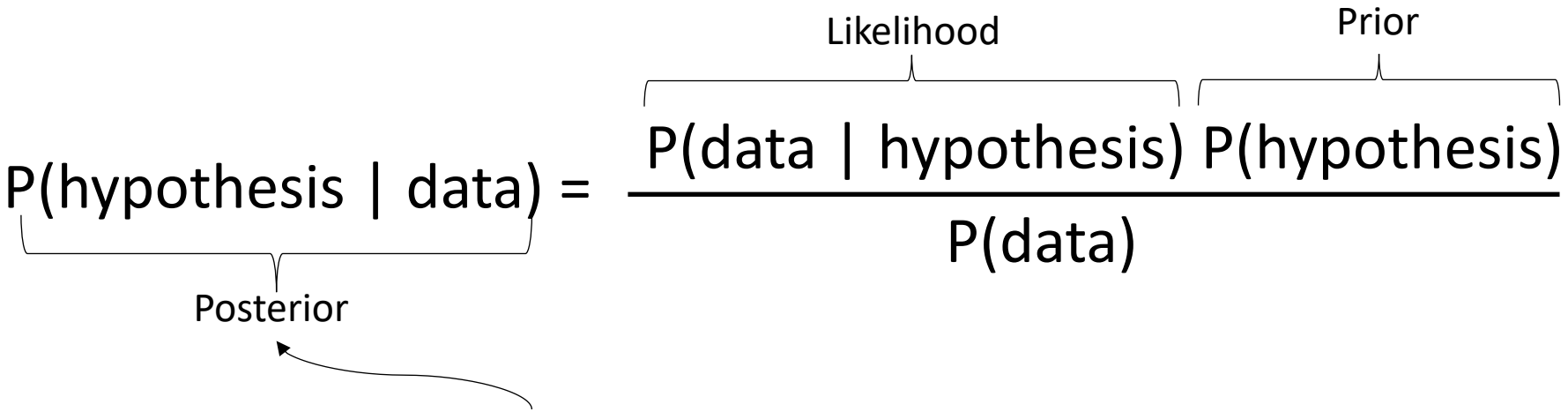
Our modeling tools should:

- Faithfully represent **uncertainty** in our model structure and parameters and noise in our data
- Be automated and **adaptive**
- Exhibit **robustness**
- **Scale well** to large data sets

Probabilistic modeling

- A model describes data that one could observe from a system
- If we use the mathematics of probability theory to express all forms of uncertainty and noise associate with out model
- Then inverse probability (i.e. Bayes rule) allows us to infer unknown quantities, adapt our models, make predictions and learn from data.

Bayes Rule



- Bayes rule tells us how to do inference about hypotheses from data
- Learning and prediction can be seen as forms of inference.

Modeling vs toolbox views of Machine Learning

Machine learning seeks to learn models of data: define a space of possible models: learn the parameters and structure of the models from data; make predictions and decisions

Machine learning is a toolbox of methods for processing data: feed the data into one of many methods; choose methods that have good theoretical or empirical performance; make predictions and decisions.

Plan

- Introduce Foundations
  - Same canonical problems: classification, regression, density estimation
  - Representing beliefs and the Cox axioms
  - The Dutch Book Theorem
  - Asymptotic Certainty and Consensus
- The Intractability Problem
- Approximation Tools
- Advanced Topics
- Limitations and Discussion

Some Canonical Machine Learning Problems

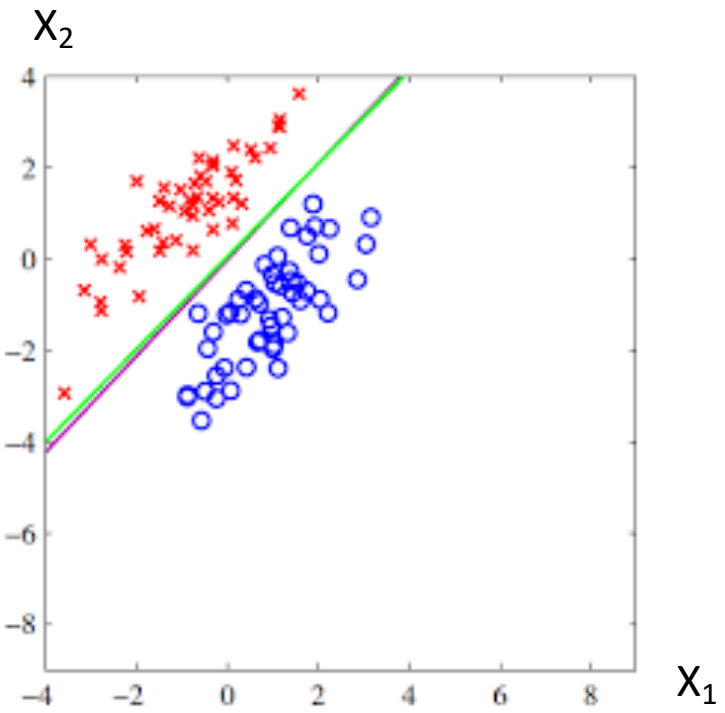
- Linear Classification
- Polynomial Regression
- Clustering with Gaussian Mixtures (Density Estimation)

Linear Classification

Data D = $\{(x^{(n)}, y^{(n)}\}$ for n= 1 .. N data points

$x^{(n)} \in R^d$

$y^{(n)} \in \{ +1, -1)$



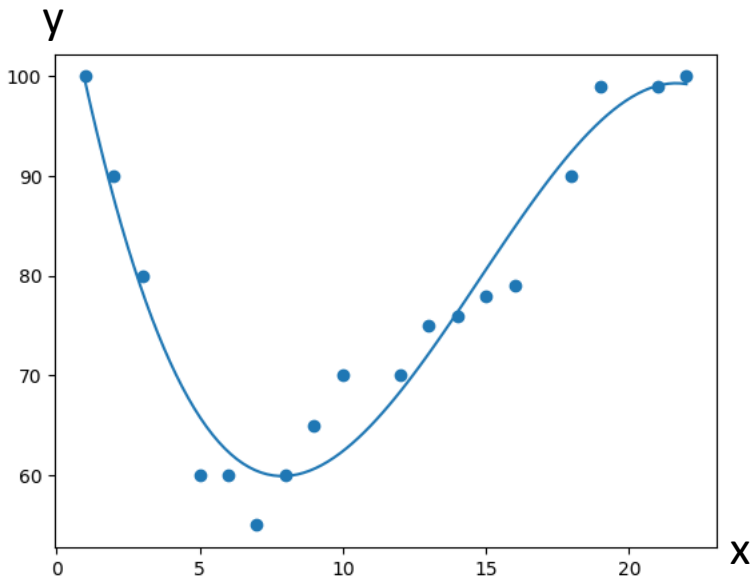
Model:

$$P(y^{(n)} = +1 | \emptyset, x^{(n)}) = \begin{cases} 1 \text{ if } \sum_{d=1}^{D} \emptyset_d x_d^{(n)} + \emptyset_0 \geq 0 \\ 0 \end{cases}$$

Parameters: $\emptyset \in R^{D+1}$

Objective: to infer $\emptyset$ from the data and to predict future labels P(y|D, x)

Polynomial Regression



Data D = $\{(x^{(n)}, y^{(n)}\}$ for n= 1 .. N data points

$x^{(n)} \in R$

$y^{(n)} \in R$

Model:

$$y^{(n)} = a_0 + a_1 x^{(n)} + a_2 x^{(n)^2} + a_m x^{(n)^m} + \epsilon \quad \text{where} \quad \epsilon \sim N(0, \sigma^2)$$

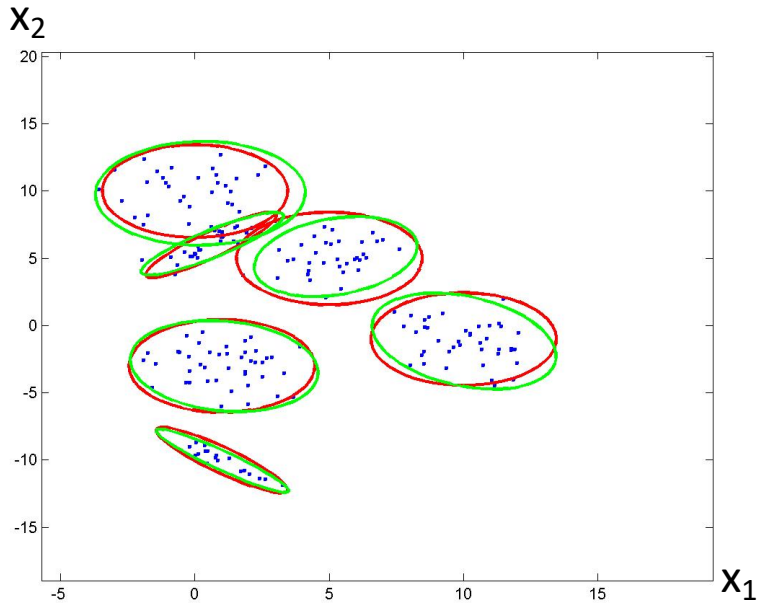
Parameters: $\emptyset \in R^{D+1}$

Objective: to infer $\emptyset$ from the data and to predict future labels $P(y|D,x)$

Clustering with Gaussian Mixtures (Density Estimation)

Data D = {($x^{(n)}$} for n= 1 .. N data points

$x^{(n)} \in R^d$



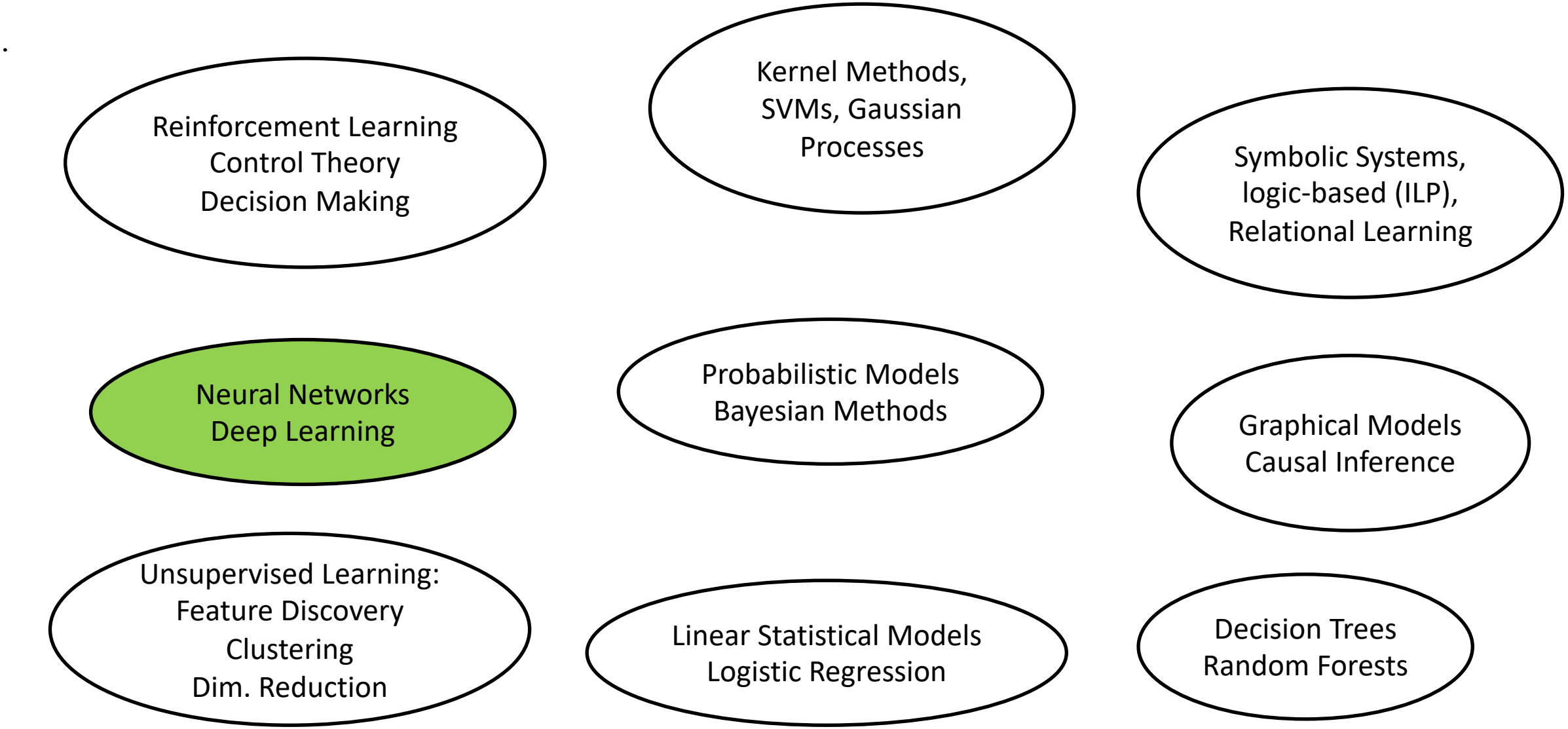
Model:

$$X^{(n)} \sim \sum_{i=1}^{m} \pi\, pi(x^{(n)}) \quad \text{where} \quad p_i(x^{(n)}) = N(\mu^{(i)}, \Sigma^{(i)})$$

Parameters: $\theta = ((\mu^{(1)}, \Sigma^{(1)}) \ldots (\mu^{(m)}, \Sigma^{(m)}), \pi)$

Objective: To infer θ from the data, predict the density p(x | D, m), and infer which points belong to the same cluster.

ML: Many Methods with Many Links to other Fields



Neural Networks and Deep Learning

- Parameters Ø are weights of neural net. Neural nets model $p(y^{(n)} \mid x^{(n)}, Ø)$ as a nonlinear function of Ø and x, e.g.:
- $p(y^{(n)} = +1 \mid x^{(n)}, Ø) = \sigma\left(\sum_i Ø_i x_i^{(n)}\right)$

- Multilayer neural networks model the overall functions as a composition of functions (layers), e.g.:

- $y^{(n)} = \sum_j Ø_j^{(2)} \sigma\left(\sum_i Ø_{ji}^{(1)} x_i^{(n)}\right) + \in^{(n)}$

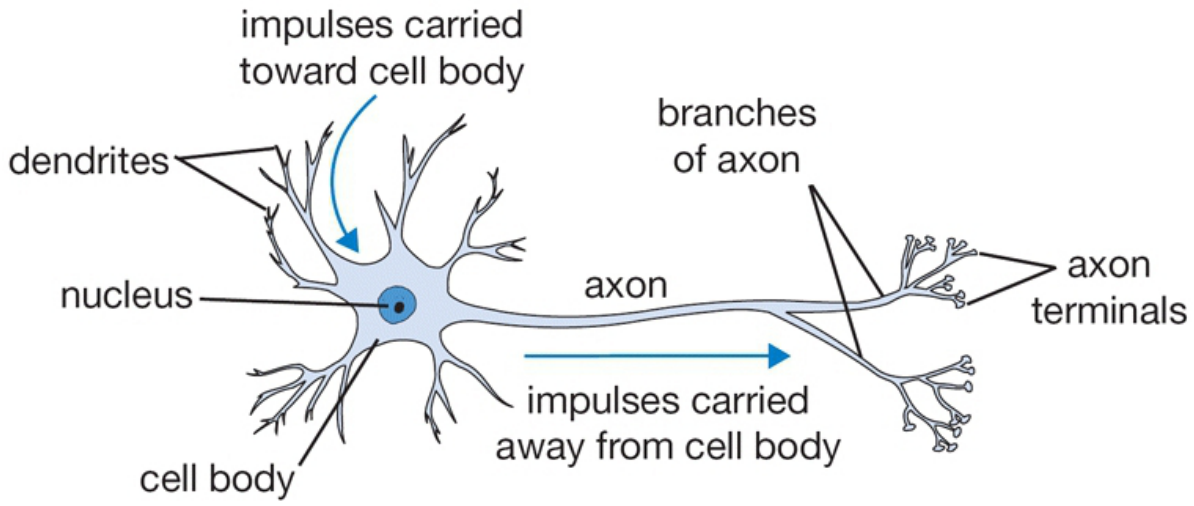
- Usually trained to maximum likelihood using variants of stochastic gradient descent (SGD) optimization.

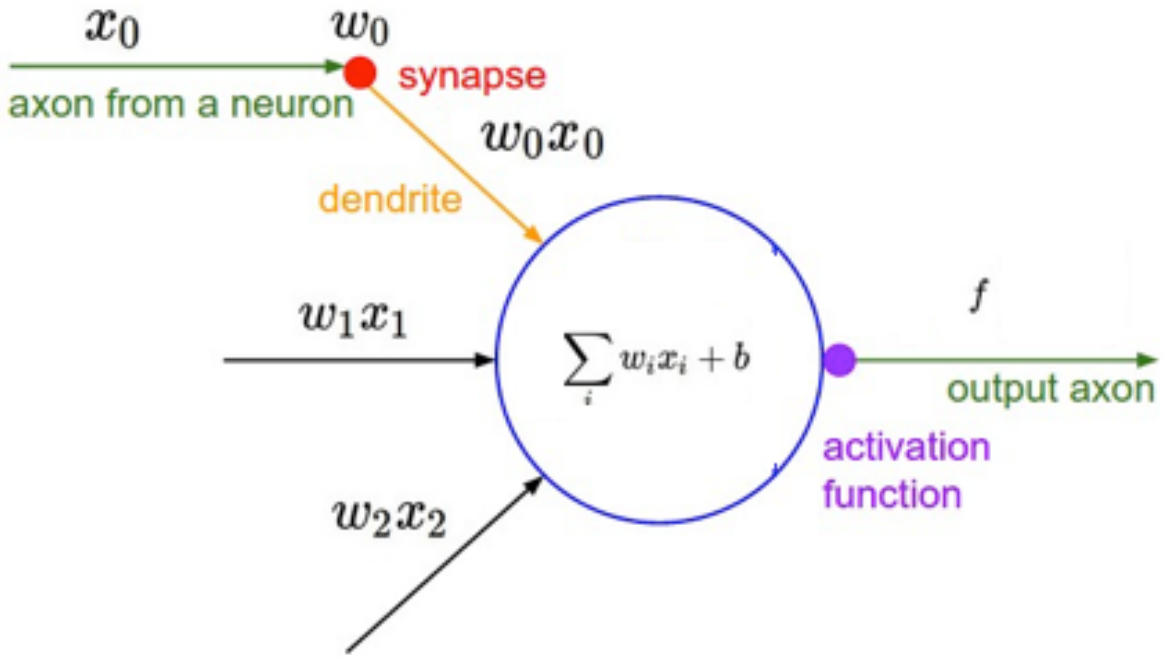
- **NN = nonlinear function + basic stats + basic optimization**

Limitations of Deep Learning

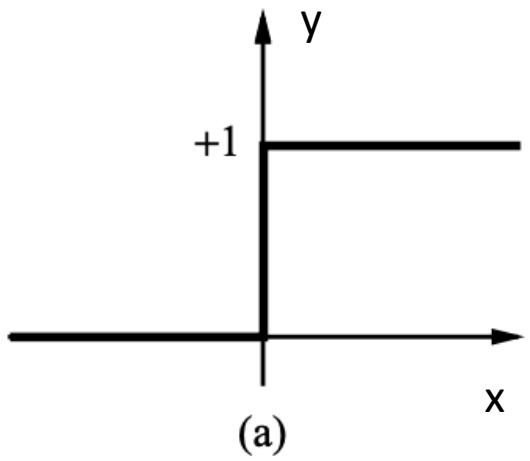
- Neural networks and deep learning systems give amazing performance on many benchmark tasks but they are generally?
  - Very data hungry (e.g. often millions of examples)
  - Very compute intensive to train and deploy (GPU resources)
  - Poor at representing uncertainty
  - Easily fooled by adversarial examples
  - Finicky to optimize : no—convex + choice of architecture, learning procedure, initialization, require expert knowledge and experimentation
  - Uninterruptable black-boxes, lacking in transparency, difficult to trust

Biological neuron and mathematical model



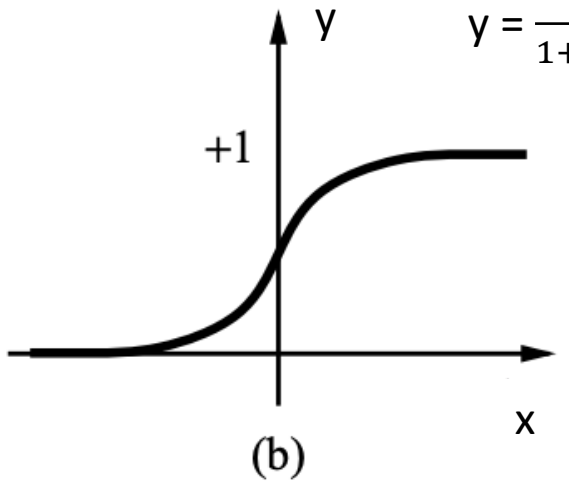


Activation function



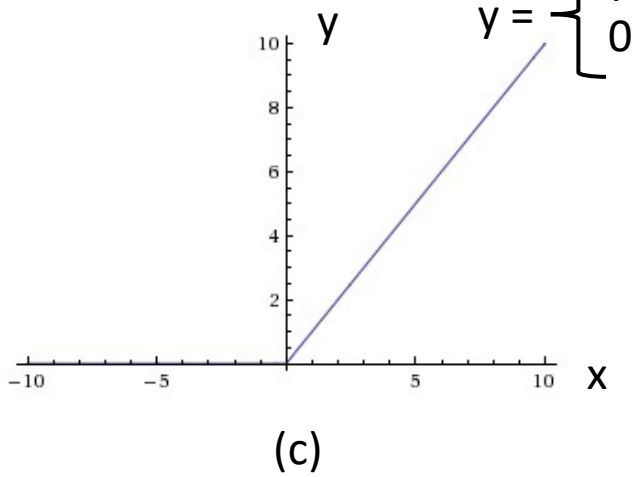
Sigmoid function:

$y = \frac{1}{1+e^{-y}}$



ReLU function:

$y = \begin{cases} y & y \Rightarrow 0 \\ 0 & y < 0 \end{cases}$



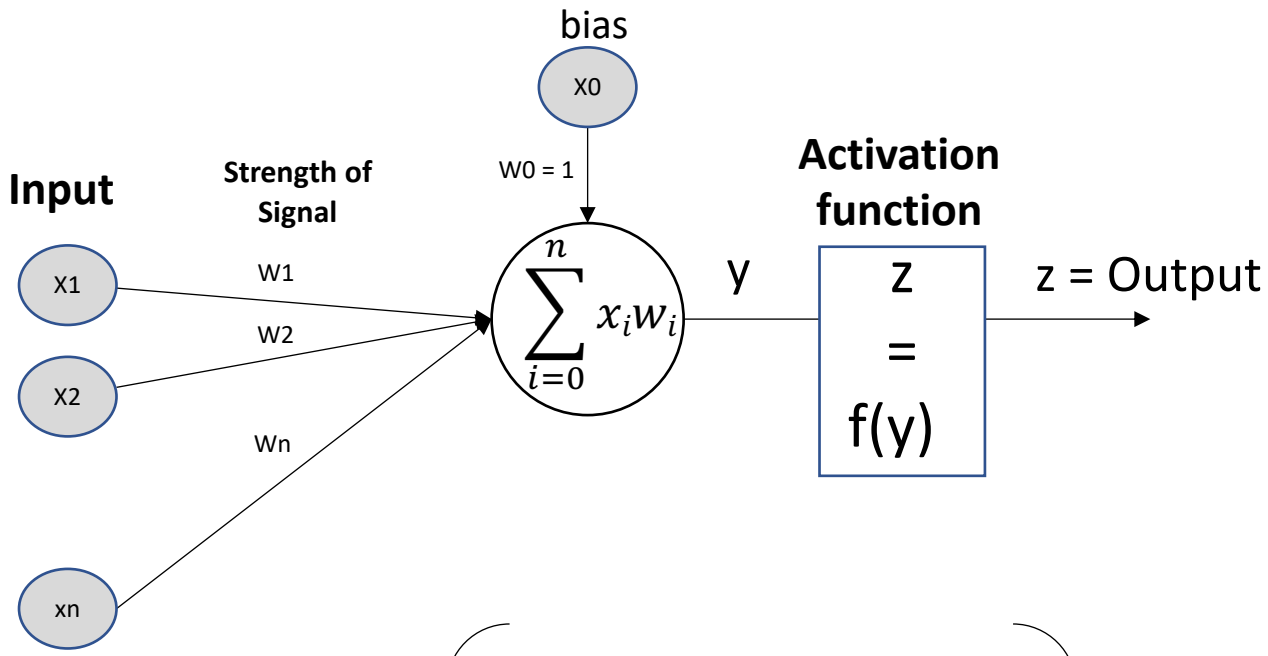
(a) Is a step function or threshold function

(b) is a sigmoid function $y = 1/(1+e^{-x})$

(c) ReLu function

Sigmoid takes a real-valued input and squashes it to range between 0 and 1

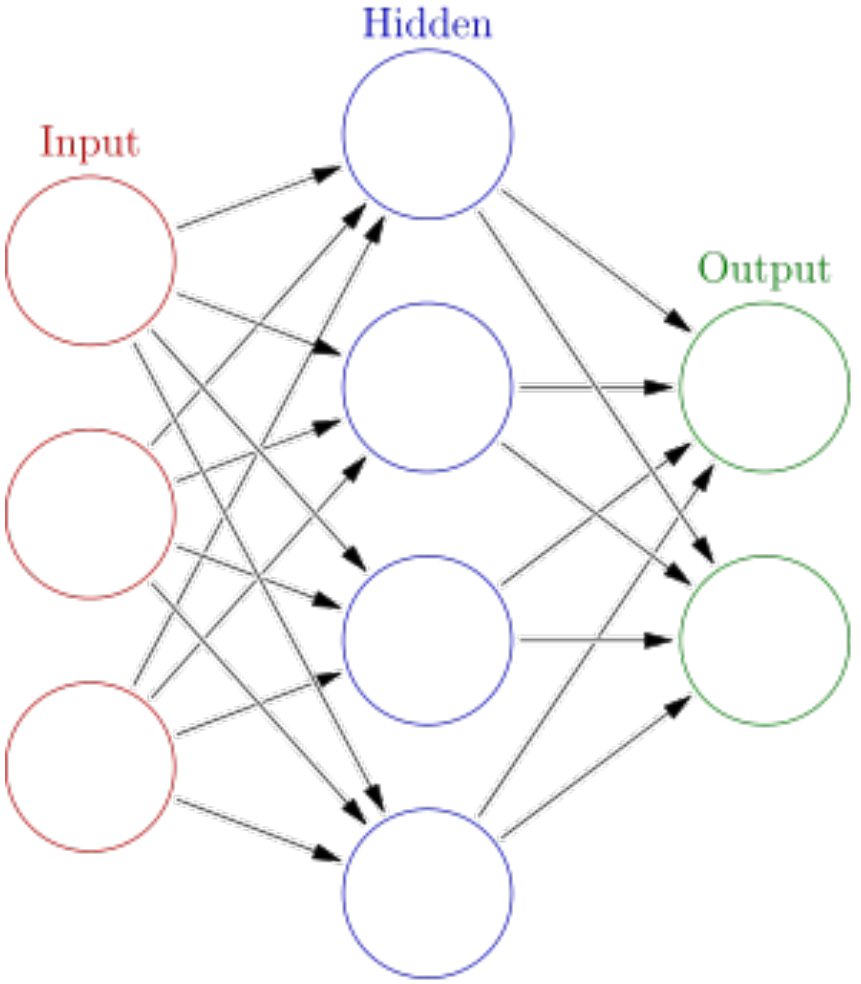
Single Layer Neural Networks



Output = $f\left(\begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_n \end{bmatrix} X \begin{bmatrix} 1 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} \right)$

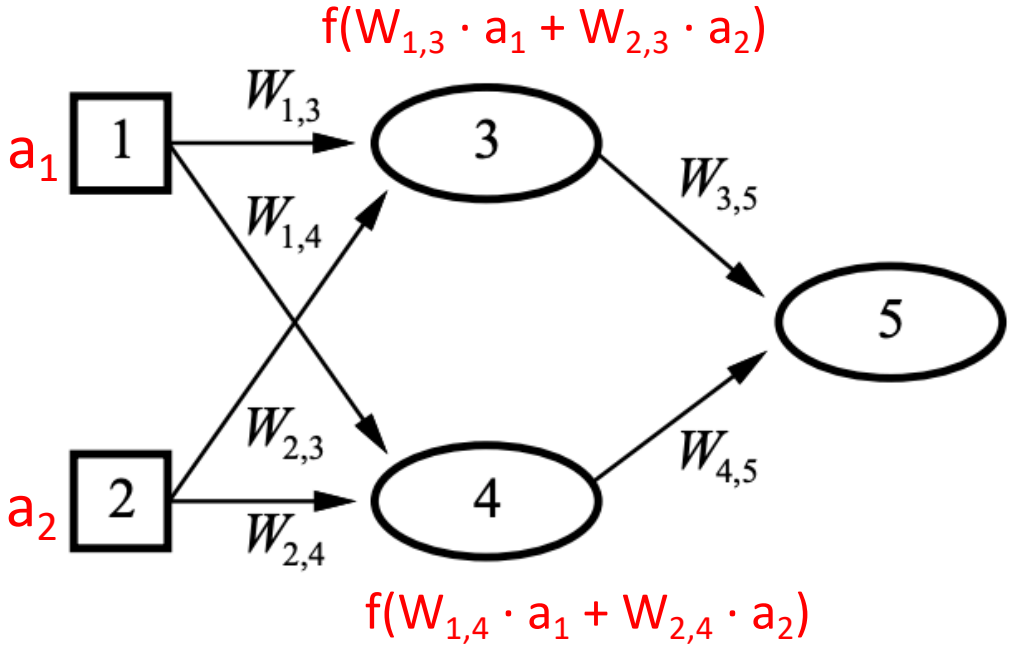
What Is A Neural Network?

- The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:
- ***"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.***
- In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989



- Every node in one layer is connected to every node in the next layer.
- Signals get transmitted from the input, to the hidden layer, to the output

Feed-forward example



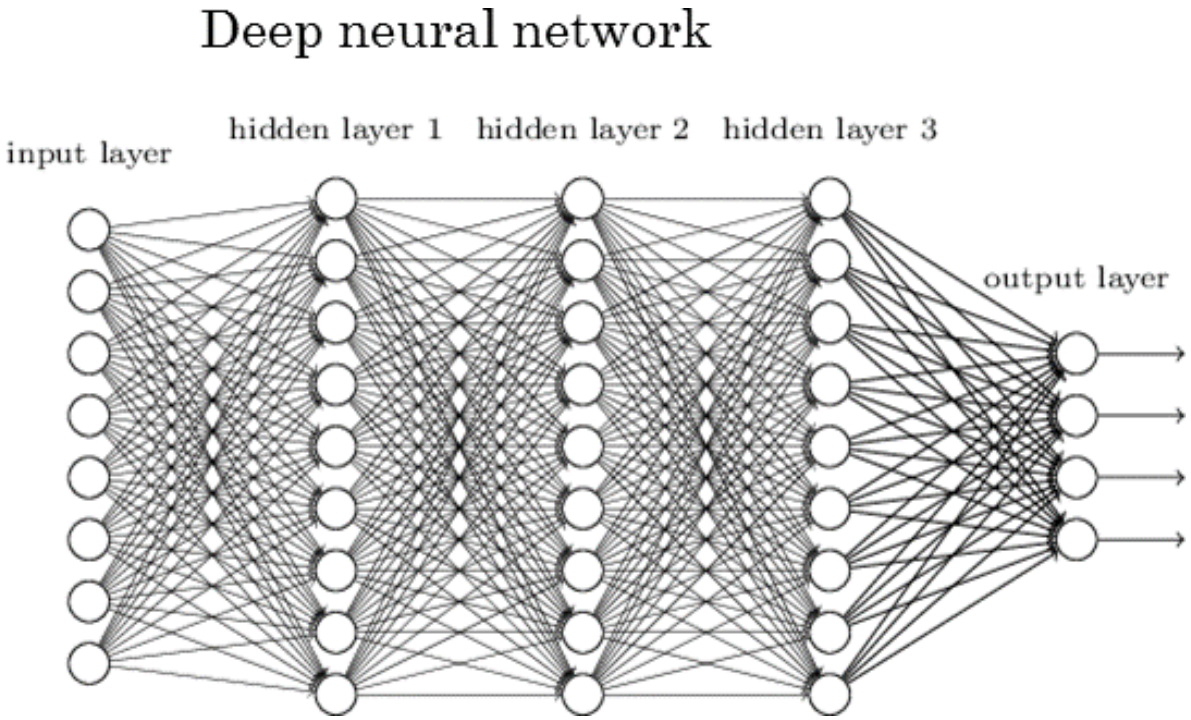
Feed-forward network = a parameterized family of nonlinear functions:

$$a5 = f(W_{3,5} \cdot f(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot f(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))$$

Adjusting weights changes the function: do learning this way! Learn by adjusting weights to reduce error on training set

Deep Learning Architecture

An Artificial Neural Network with one or more hidden layers = Deep Learning



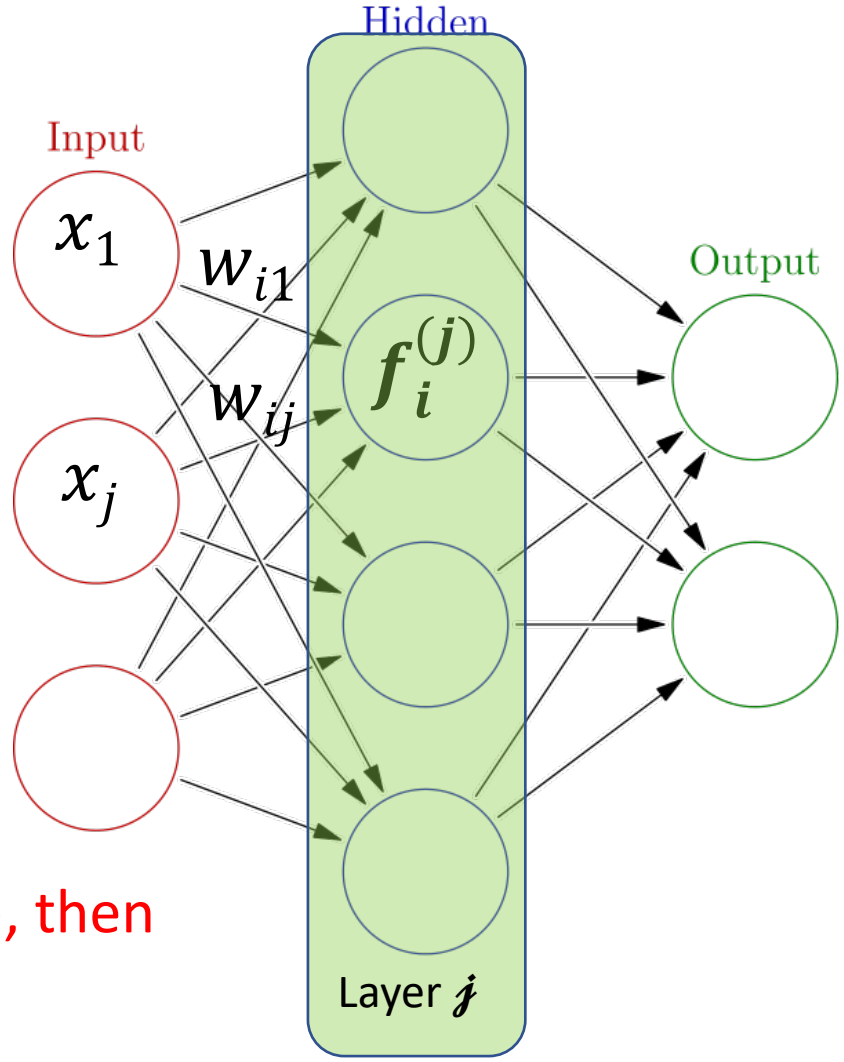
- They typically consist of many hundreds of simple processing units which are wired together in a complex communication network.
- Each unit or node is a simplified model of a real neuron which fires (sends off a new signal) if it receives a sufficiently strong input signal from the other nodes to which it is connected.
- The output is aiming for a target.

Multi Layer NN Nonlinear Math Model

$f_i^{(j)}$ = "activation function" of unit i in layer j

W $^{(j)}$ = matrix of weights controlling function mapping from layer j to layer j+1

$$y_i = fi(\sum_{j=0}^{n} w_{ij}\, x_j\,)$$



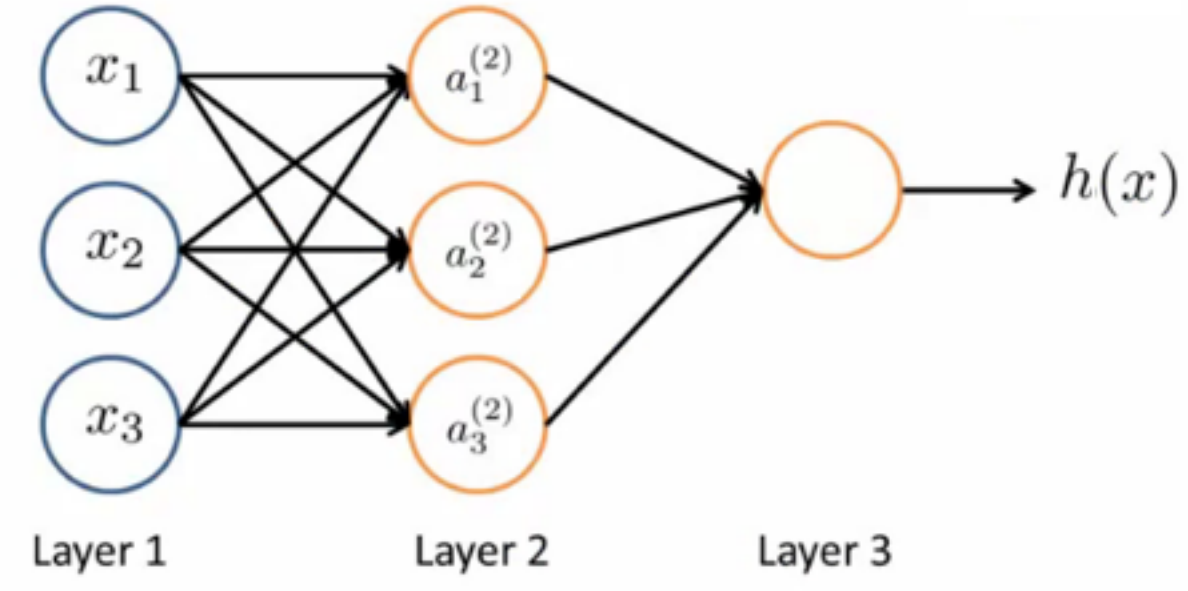
If network has N units in layer j, and M units in layer (j-1), then W $^{(j)}$ will be of dimension of (M+1) x N

Feed Forward Propagation to Calculate h(x)

$$a_1^{(2)} = f\left(\sum_{j=0}^{n} w_{1j}\, x_j\right) =$$

$$a_2^{(2)} = f\left(\sum_{j=0}^{n} w_{2j}\, x_j\right) =$$

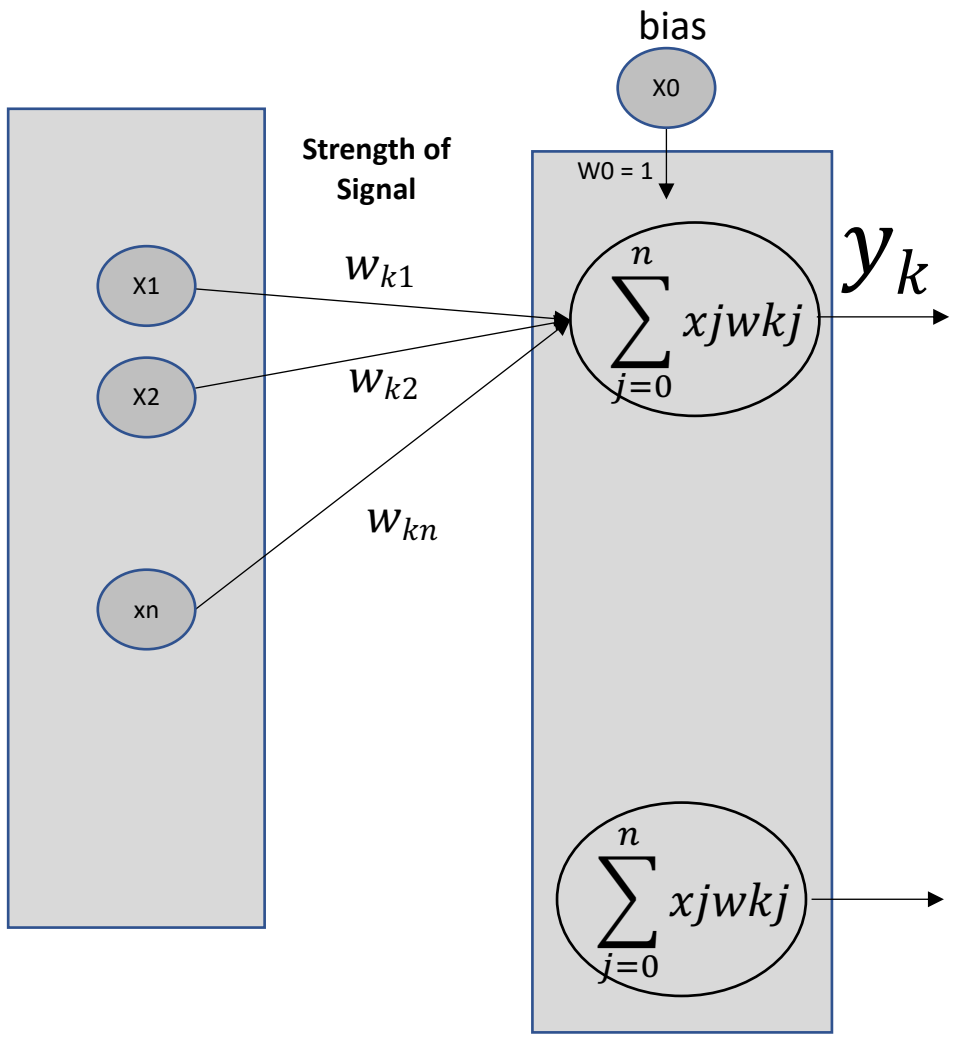
$$a_3^{(2)} = f\left(\sum_{j=0}^{n} w_{3j}\, x_j\right) =$$



$$a_i^{(2)} = f\left(\sum_{j=0}^{n} w_{ij}\, x_j\right)$$

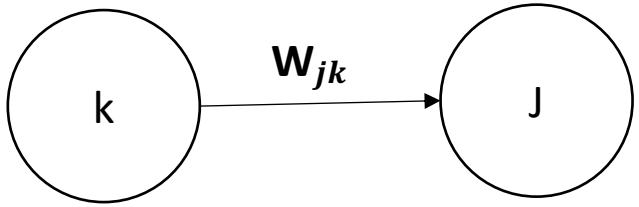
$$\boldsymbol{h(x) = f(w_3 a_3^{(2)} + w_2 a_2^{(2)} + w_1 a_1^{(2)} + a_0)}$$

Multi Layer NN Nonlinear Math Model



$$y_k = fk\left(\sum_{j=0}^{n} x_j w_{kj}\right)$$

Connection weight from neuron in layer k to neuron in layer j = $\mathbf{W_{jk}}$



$$\begin{bmatrix} y_1 \\ y_2 \\ \\ \\ y_k \end{bmatrix} = f \begin{bmatrix} w_{k1} & w_{k2} & & w_{kj} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ \\ xj \end{bmatrix}$$

Backpropagation

The final step in a forward pass is to evaluate the **predicted output *s*** against an **expected output *y***.

Evaluation between *s* and *y* happens through a **cost function**. This can be as simple as MSE (mean squared error), more complex like cross-entropy or any other cost function.

$$C = cost(s, y)$$

backpropagation aims to minimize the cost function by adjusting network's weights and biases.

The gradient shows how much the parameter $w_{ij}^{(l)}$ *needs to change (in positive or negative direction) to minimize C.*

Error - The final step in a forward pass is to evaluate the **predicted output** against an **expected output** *y*.

$$x_j^{(l)} = \theta\left(\sum_{i=0}^{d} w_{ij}^{(l-1,l)} x_i^{(l-1)}\right) \qquad h(x) = x_j^{(l)} \text{ when } l = \mathcal{L}$$

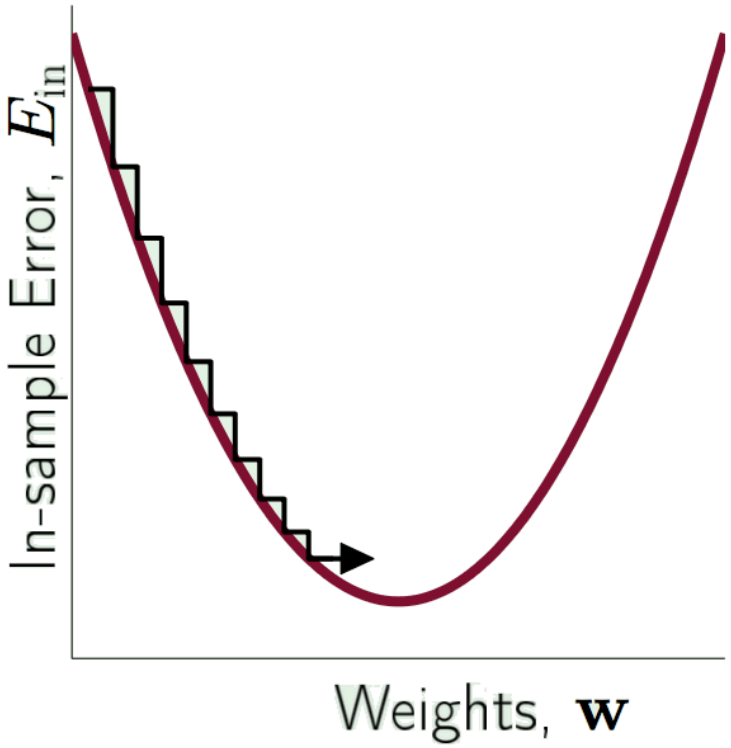
If we have all the weights $w=\left\{w_{ij}^{(l-1,l)}\right\}$ *for each layer then we can determine* $h(x)$.

Error on example (Xn, Yn) *is* $e(h(Xn), Yn) = e(w)$

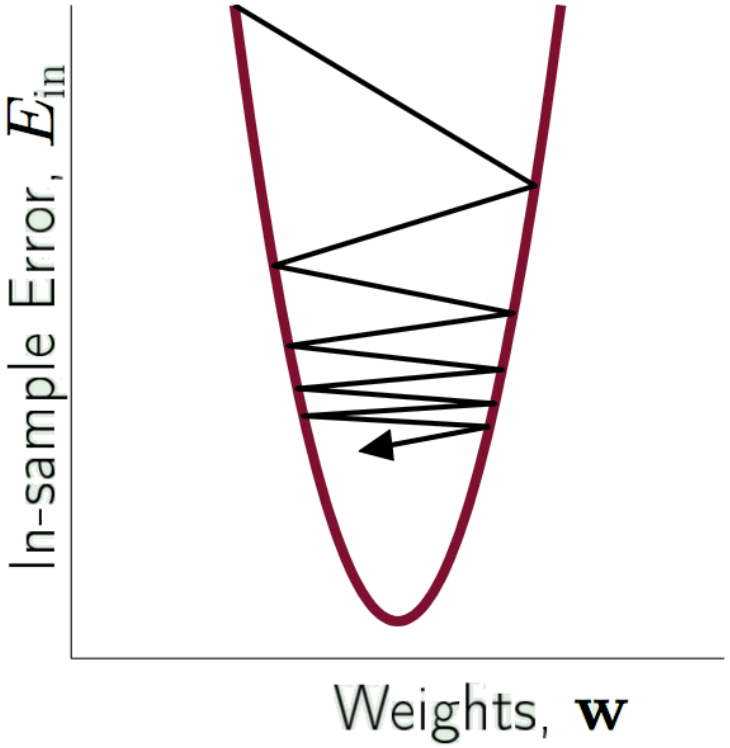
We need to calculate Gradient:

$$\nabla \mathbf{e}(\mathbf{w}): \quad \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}} \quad \text{for all} \quad i, j, l$$

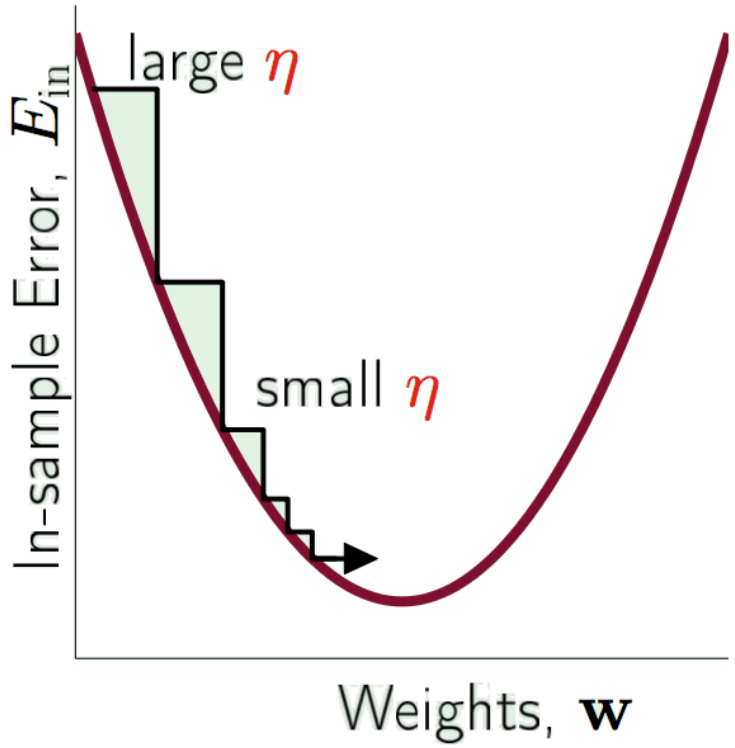
Gradient



η too small



η too large



variable η – just right

Derivative of Sigmoid Function

$$\frac{ds(x)}{dx} = \frac{1}{1+e^{-x}}$$

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

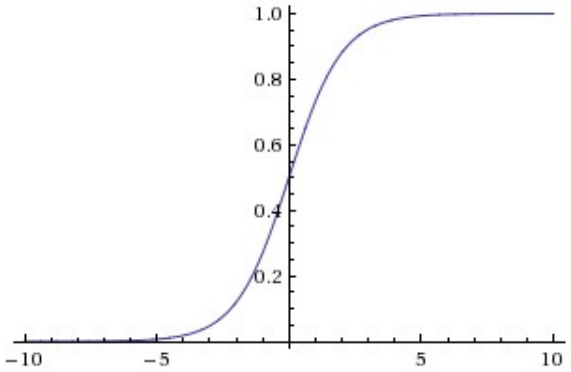
$$= \left(\frac{1}{1+e^{-x}}\right)^2 e^{-x}(-1)$$

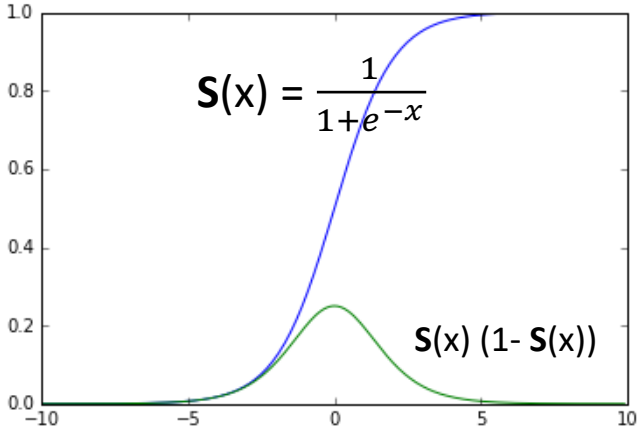
$$= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)(-e^{-x})$$

$$= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{-e^{-x}}{1+e^{-x}}\right)$$

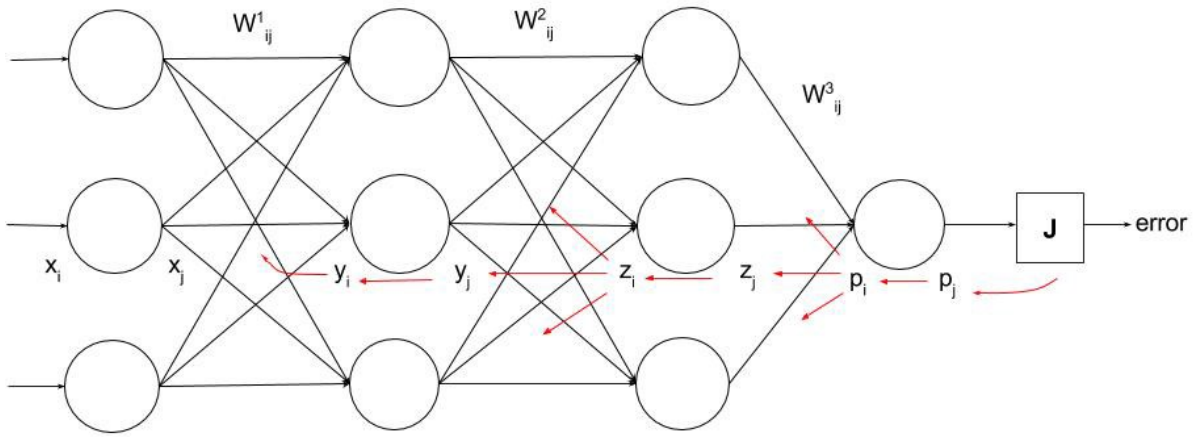
$$= s(x)(1-s(x))$$

$$\mathbf{S}(x) = \frac{1}{1+e^{-x}}$$





Gradient Descend

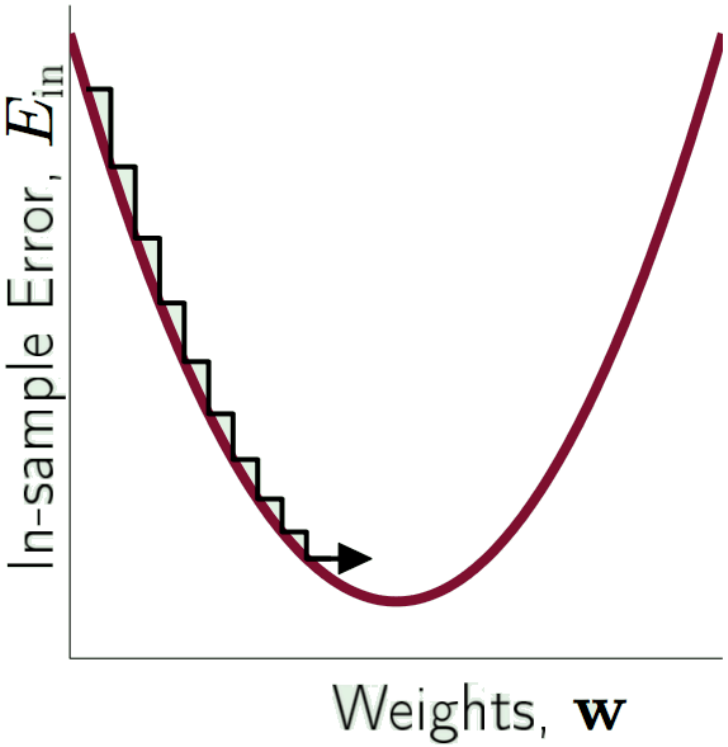


$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

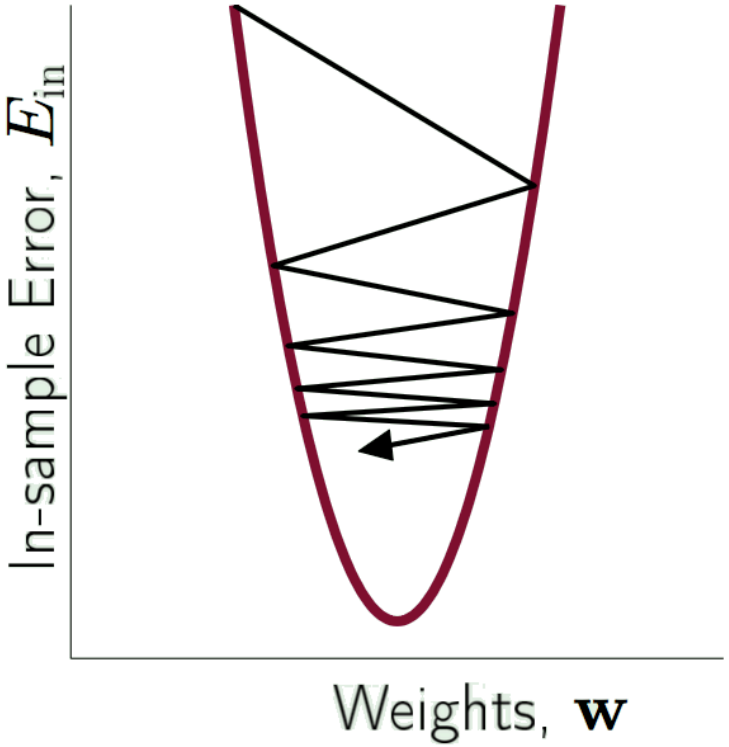
$$w := w - \epsilon \frac{\partial C}{\partial w}$$

$$b := b - \epsilon \frac{\partial C}{\partial b}$$

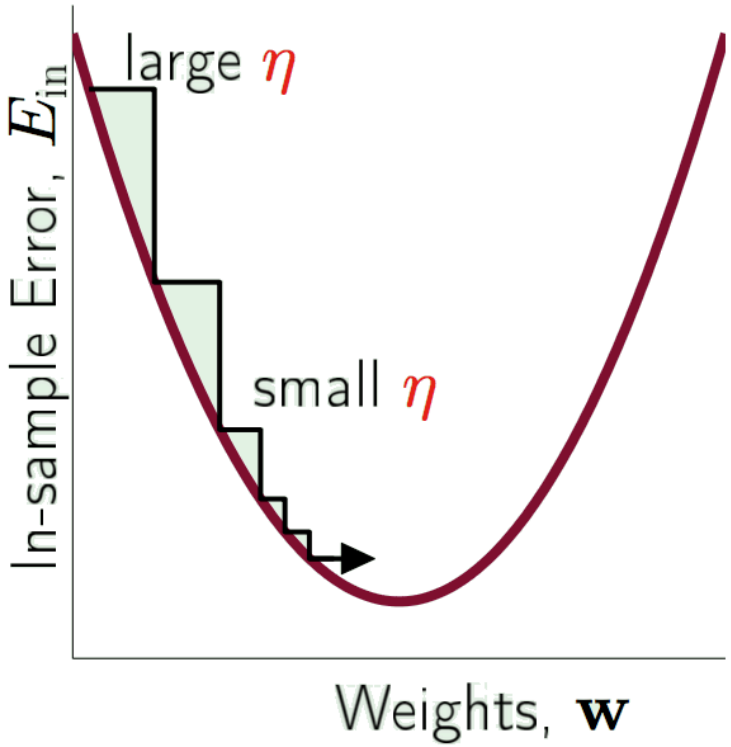
Learning Rate: Steps



η too small



η too large



variable η – just right

Learning Rate should increase with the slope