Neural Networks

CMSC 422

SOHEIL FEIZI

sfeizi@cs.umd.edu

What we have learned so far ...

Classification Problem:
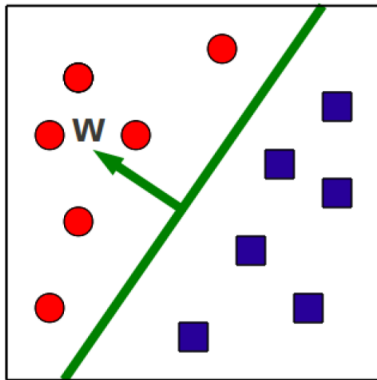

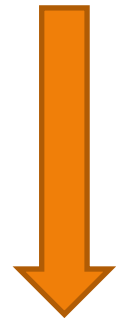Decision trees   KNNs   Perceptron   Empirical Risk Minimization   Maximum Likelihood

Linear model $(w^t x + b)$
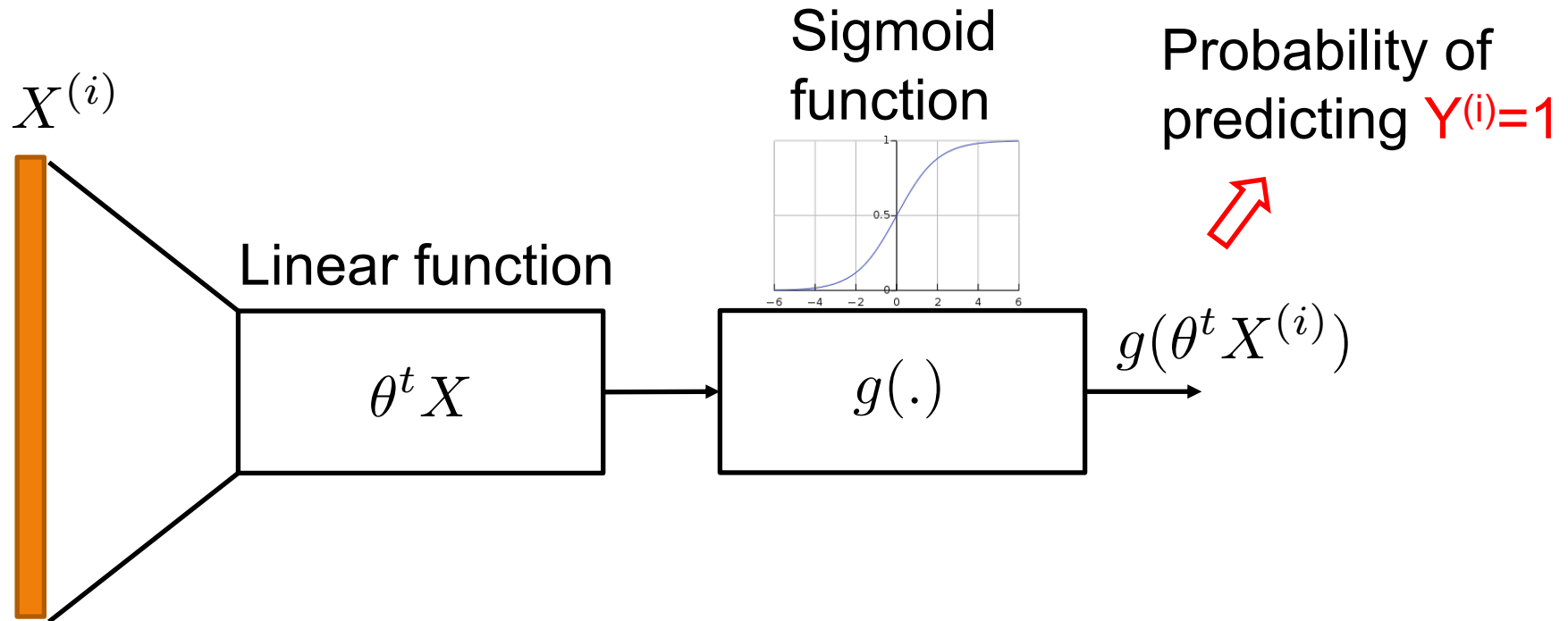
0-1 loss $\rightarrow$ Hinge loss

Cross-entropy loss

Why did we restrict our models to linear (w^t x +b)?

- Optimizations turned out to be convex.

- Why this is important?

- An efficient method (Stochastic GD) to find the global optimizer

- What do we lose by restricting ourselves to linear models?
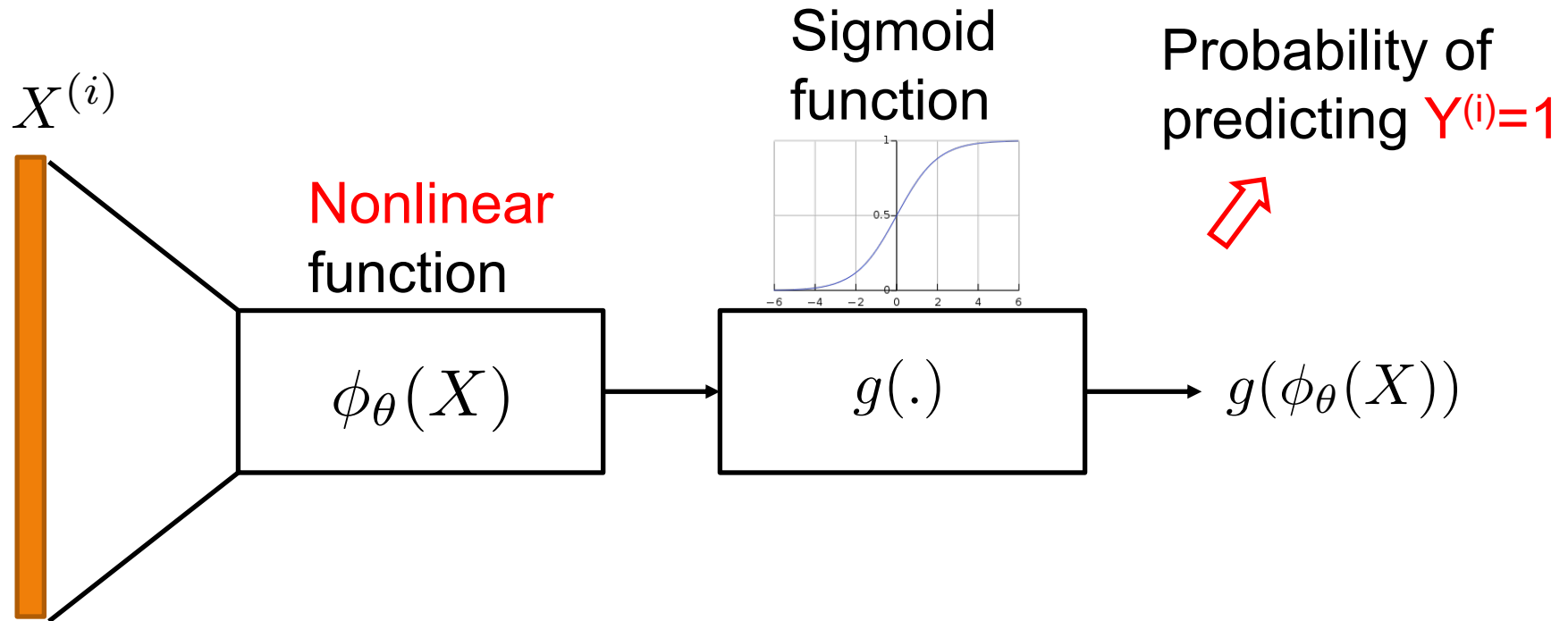
Recall the logistic regression

$X^{(i)}$

Linear function

$\theta^t X$

Sigmoid function

$g(.)$

$g(\theta^t X^{(i)})$

Probability of predicting $Y^{(i)}=1$

Is X->Y model linear? No!

Pick model parameters using cross-entropy loss optimization

$$\max_{\theta} \sum_{i=1}^{N} Y^{(i)} \log g(\theta^t X^{(i)}) + (1 - Y^{(i)}) \log(1 - g(\theta^t X^{(i)}))$$

Linear → Nonlinear

$X^{(i)}$

Nonlinear
function

$\phi_\theta(X)$

Sigmoid
function

$g(.)$

Probability of
predicting Y(i)=1

$g(\phi_\theta(X))$

Is X->Y model linear?   No!

Pick model parameters using cross-entropy loss optimization

$$\max_\theta \quad \sum_{i=1}^{N} Y^{(i)} \log g(\phi_\theta(X^{(i)})) + (1 - Y^{(i)}) \log(1 - g(\phi_\theta(X^{(i)})))$$
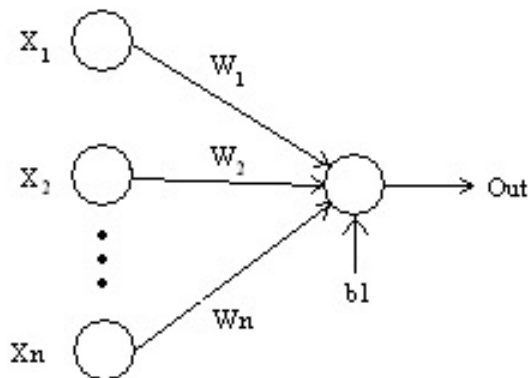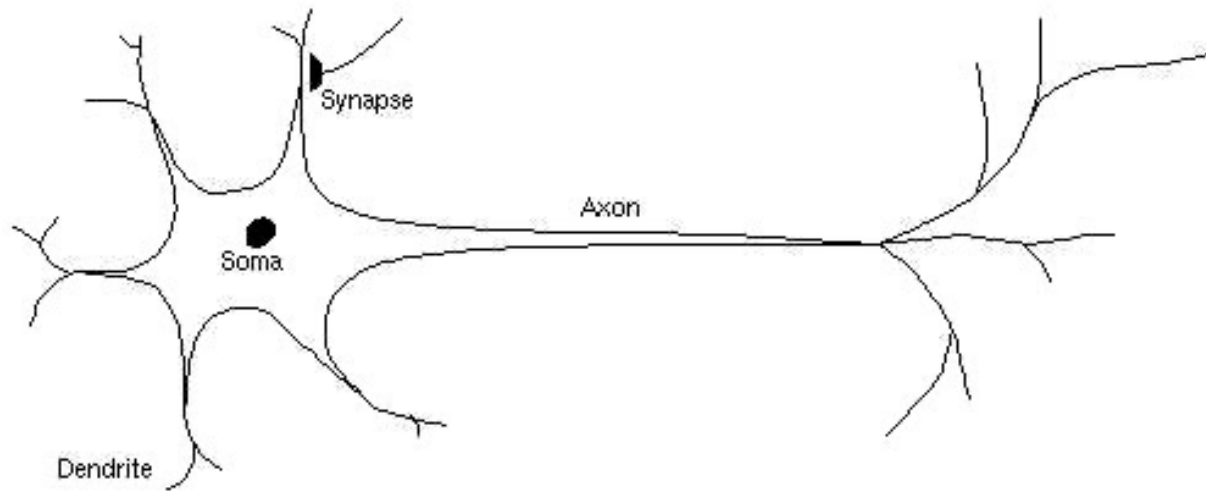
Two Questions

- What is a "good" family of nonlinear functions to consider?

- How to solve the resulting optimization? Can we still use stochastic GD?

Neural Networks

- What are Neural Networks?

- Why are neural networks powerful?

Aside: biological inspiration


Analogy: the perceptron as a neuron

History of Neural Networks

- 1943: McCulloch and Pitts proposed a model of a neuron
- 1960s: Widrow and Hoff explored Perceptron networks (which they called "Adelines") and the delta rule.
- 1957: Frank Rosenblatt invents the **Perceptron**
  1962: Rosenblatt proved convergence of the perceptron training rule.
- 1969: Minsky and Papert showed that the Perceptron cannot deal with nonlinearly-separable data sets---even those that represent simple function (e.g., X-OR)
- 1970-1985: Very little research on Neural Nets
- 1986: Invention of Back Propagation [Rumelhart & McClelland; Parker; Werbos] which can learn nonlinearly-separable data sets.
- Since 1985: A lot of research in Neural Nets!
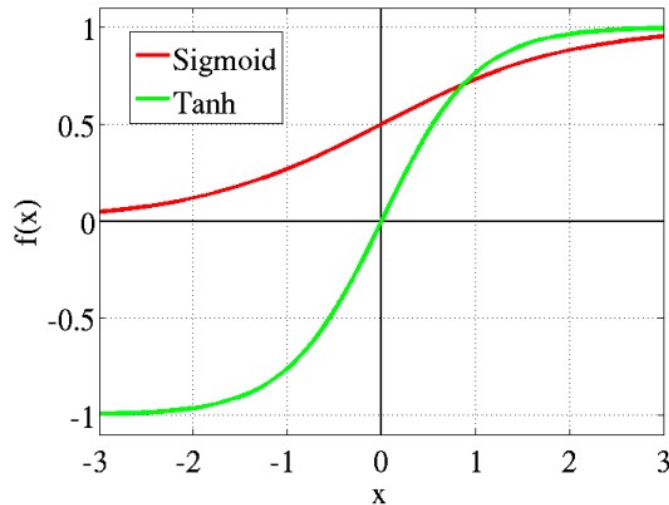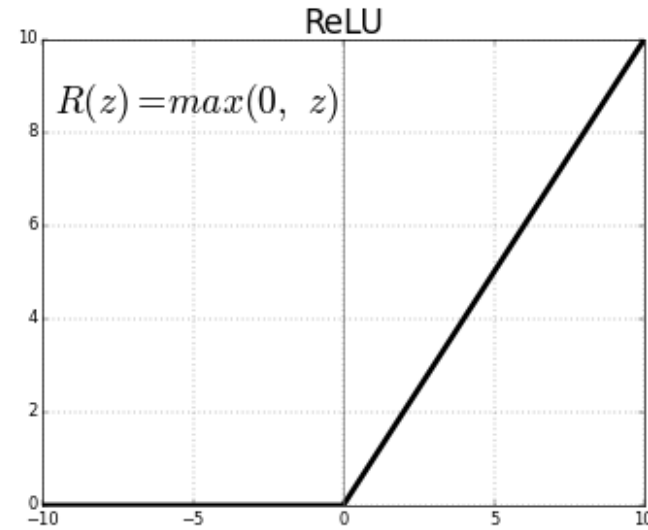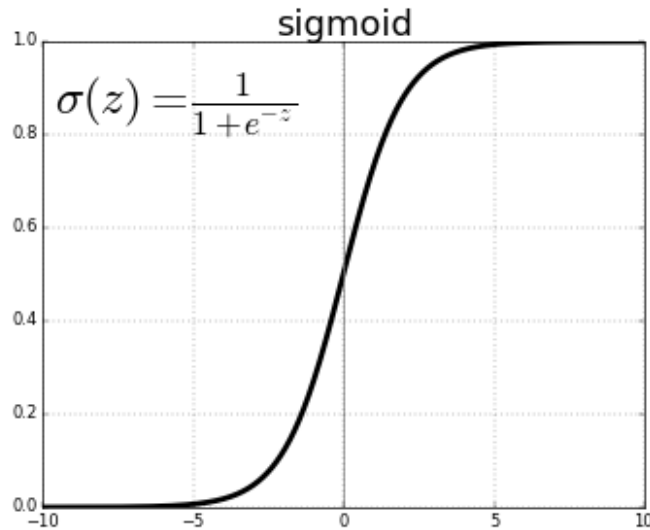- Geoff Hinton

Neural networks

- Neural networks are made up of **nodes** or **units**, connected by **links**

- Each link has an associated **weight** and **activation level**

- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Neural Unit

X_1

X_2

X_3

X_d

w_1

w_2

w_3

w_4

$$\sum X_i w_i \quad \phi(.)$$

$$\phi\left(\sum_i X_i w_i\right)$$

Activation function
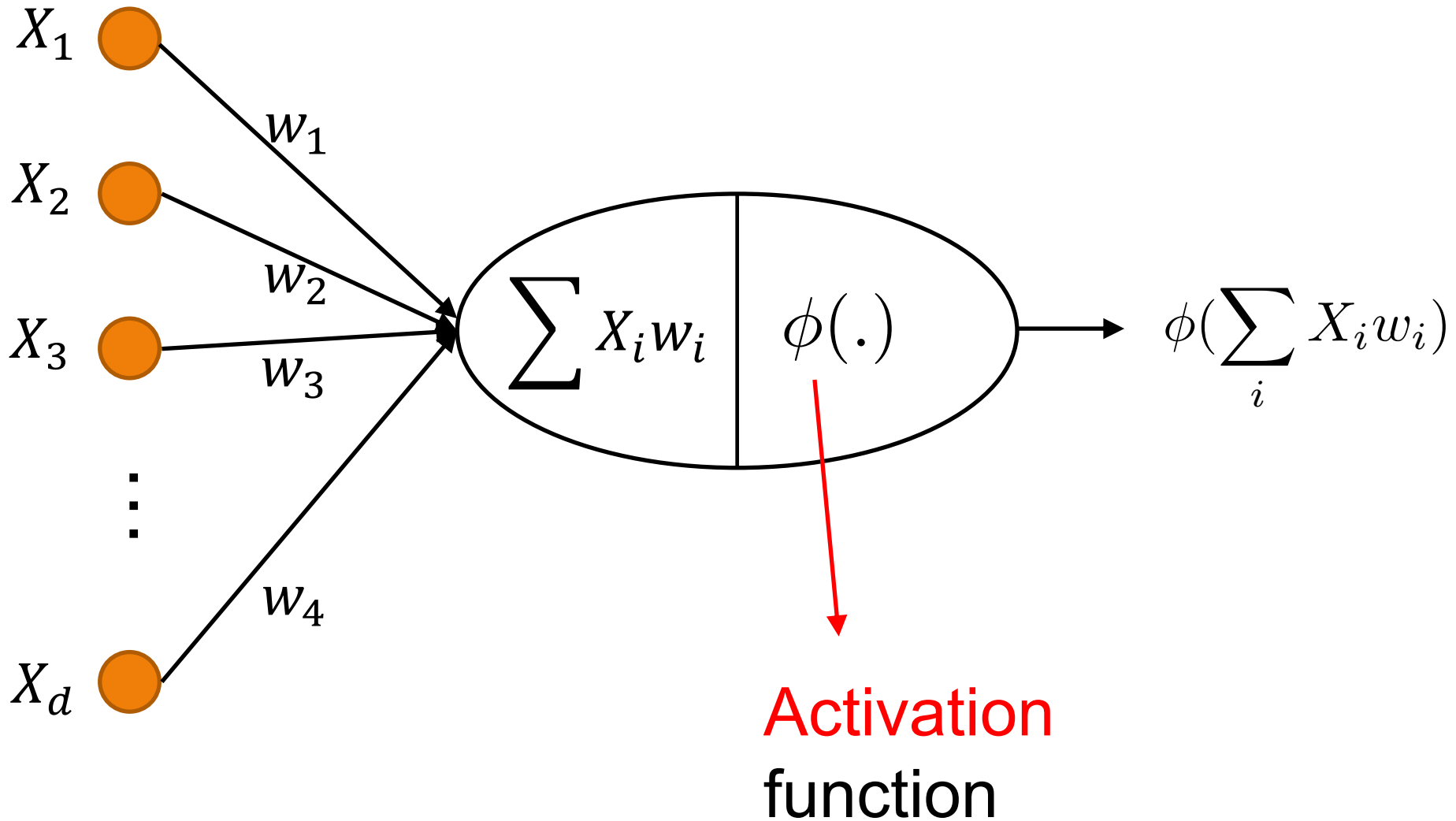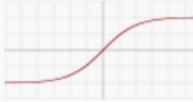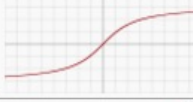
Popular Activation Functions

sigmoid

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

ReLU

$$R(z) = max(0, \ z)$$

Sigmoid
Tanh

Q: how to choose a proper activation function?

So many more activation functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \dfrac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \dfrac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \dfrac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \dfrac{1}{1 + e^{-x}}$

Lets make the picture more concise

Activation function

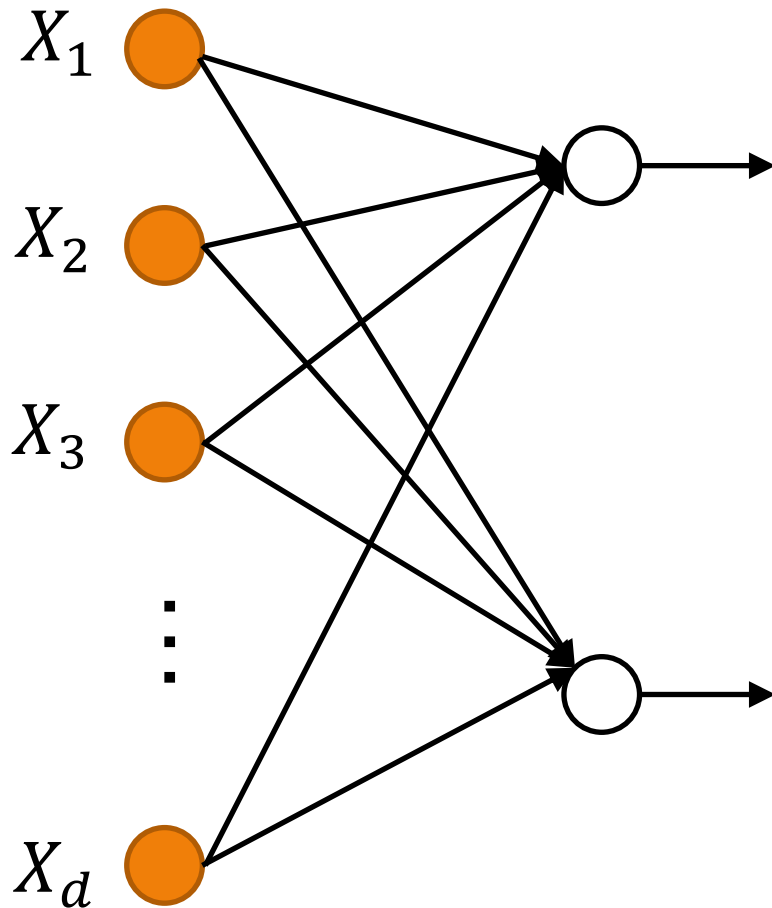# Lets make the picture more concise

X_1
X_2
X_3
$\vdots$
X_d
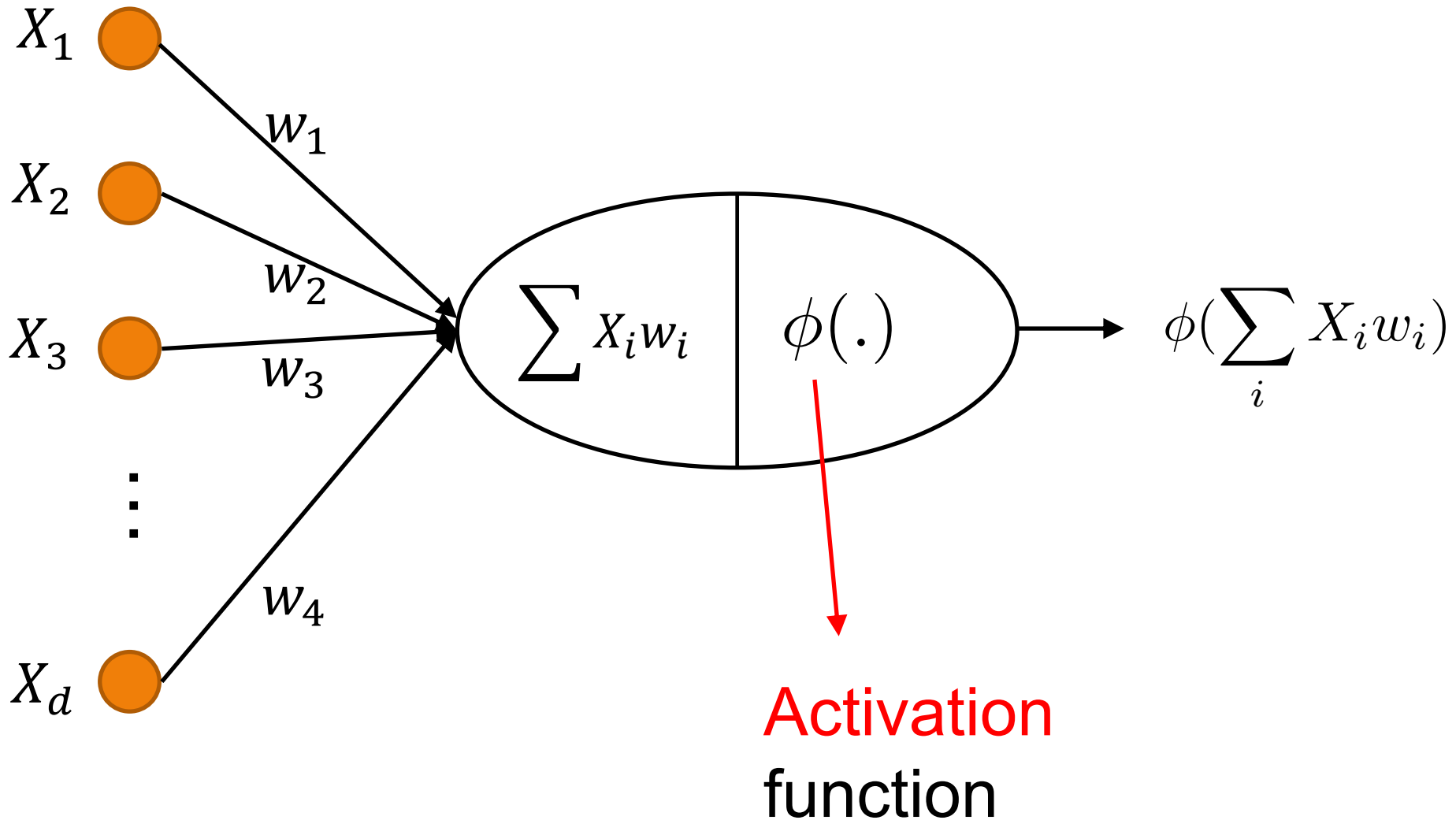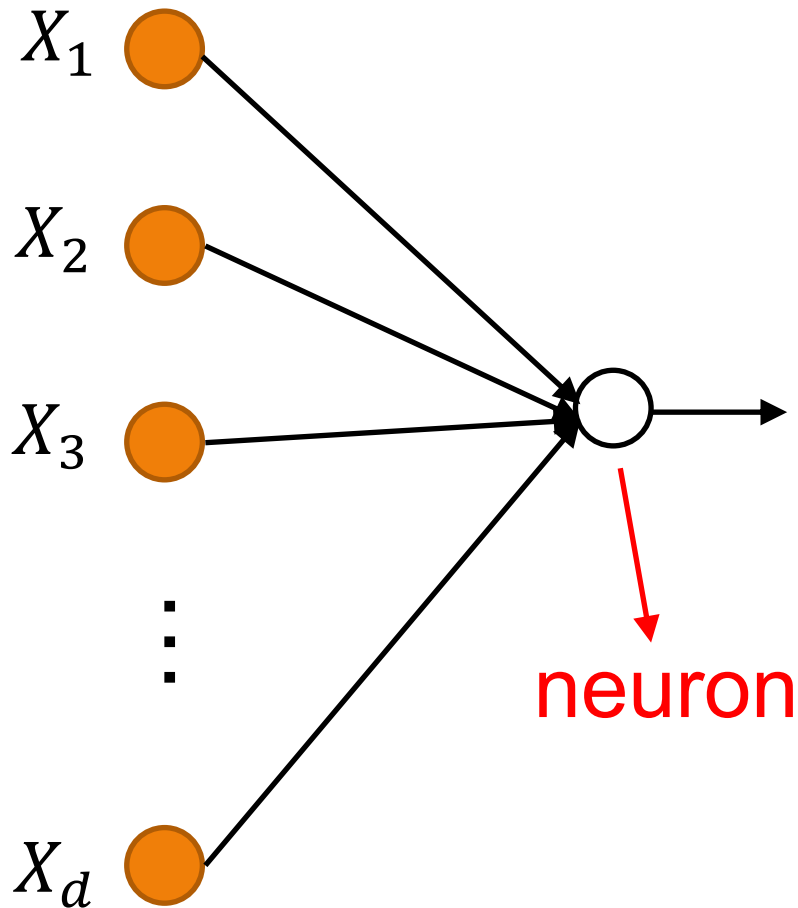
neuron

Implicitly we always assume

- Edges have weights
- Neurons have activations

Q: can we make the function more complex (i.e. higher representation power)?

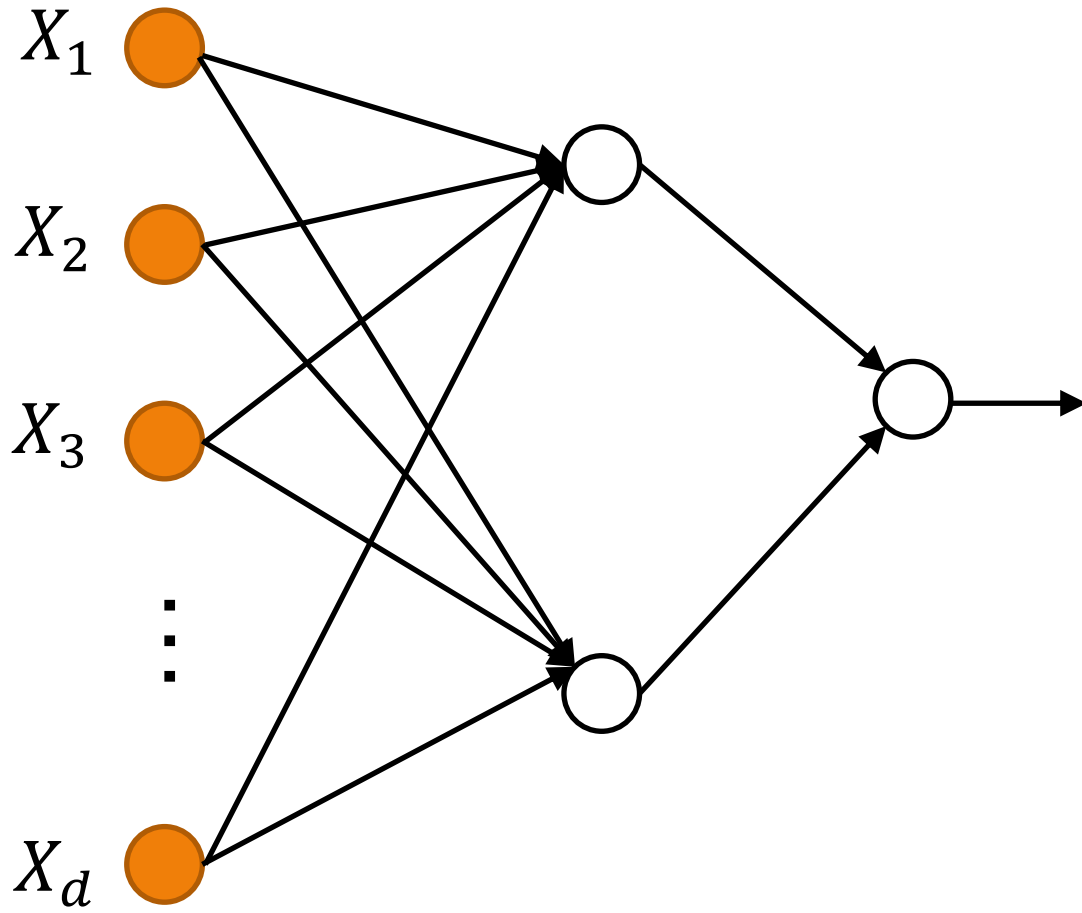# Lets add one more neuron

But my desired function is from d-dimension to one dimension.

How can I resolve this?

Lets add one more neuron

X_1

X_2

X_3

$\vdots$

X_d

Can we add more neurons?

Multi-Layer Neural Network

Multi-layer
perceptron (MLP)

Output neuron
(often with no
activation)

1st hidden
layer

2nd hidden
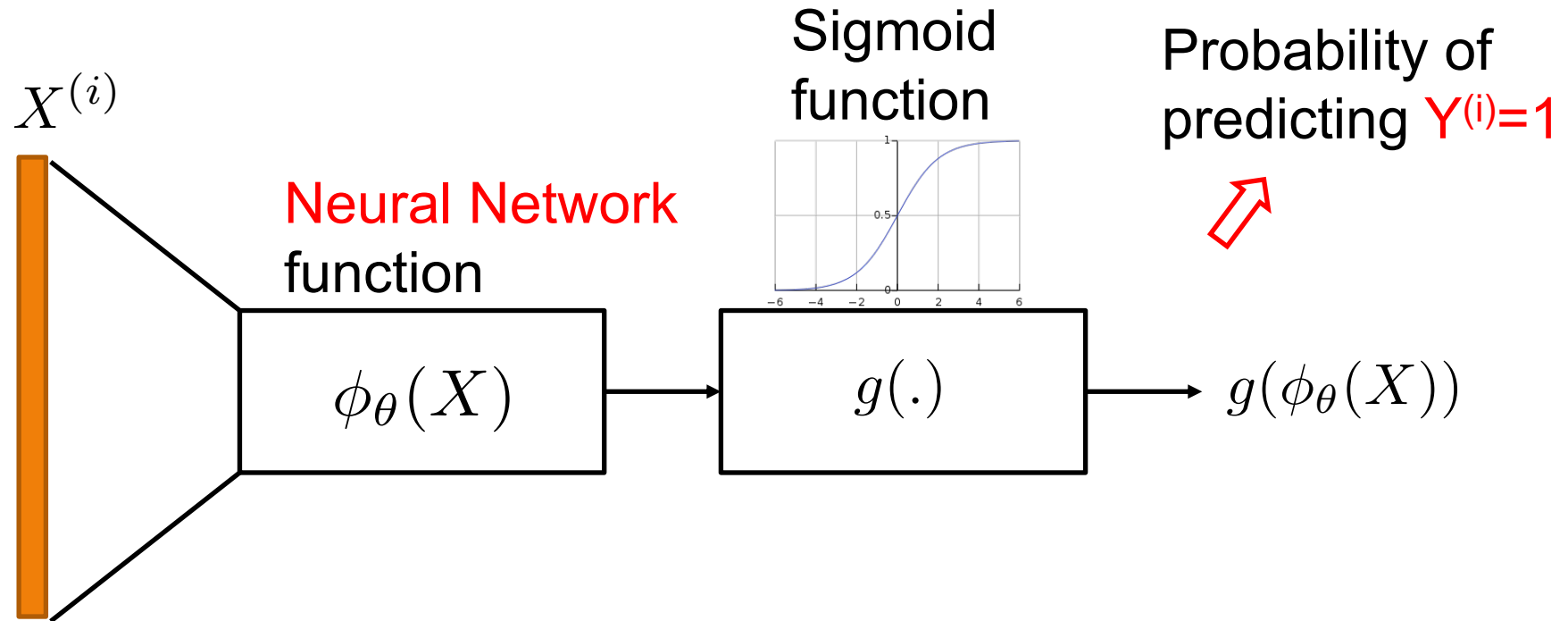layer

X_1

X_2

X_3

X_d
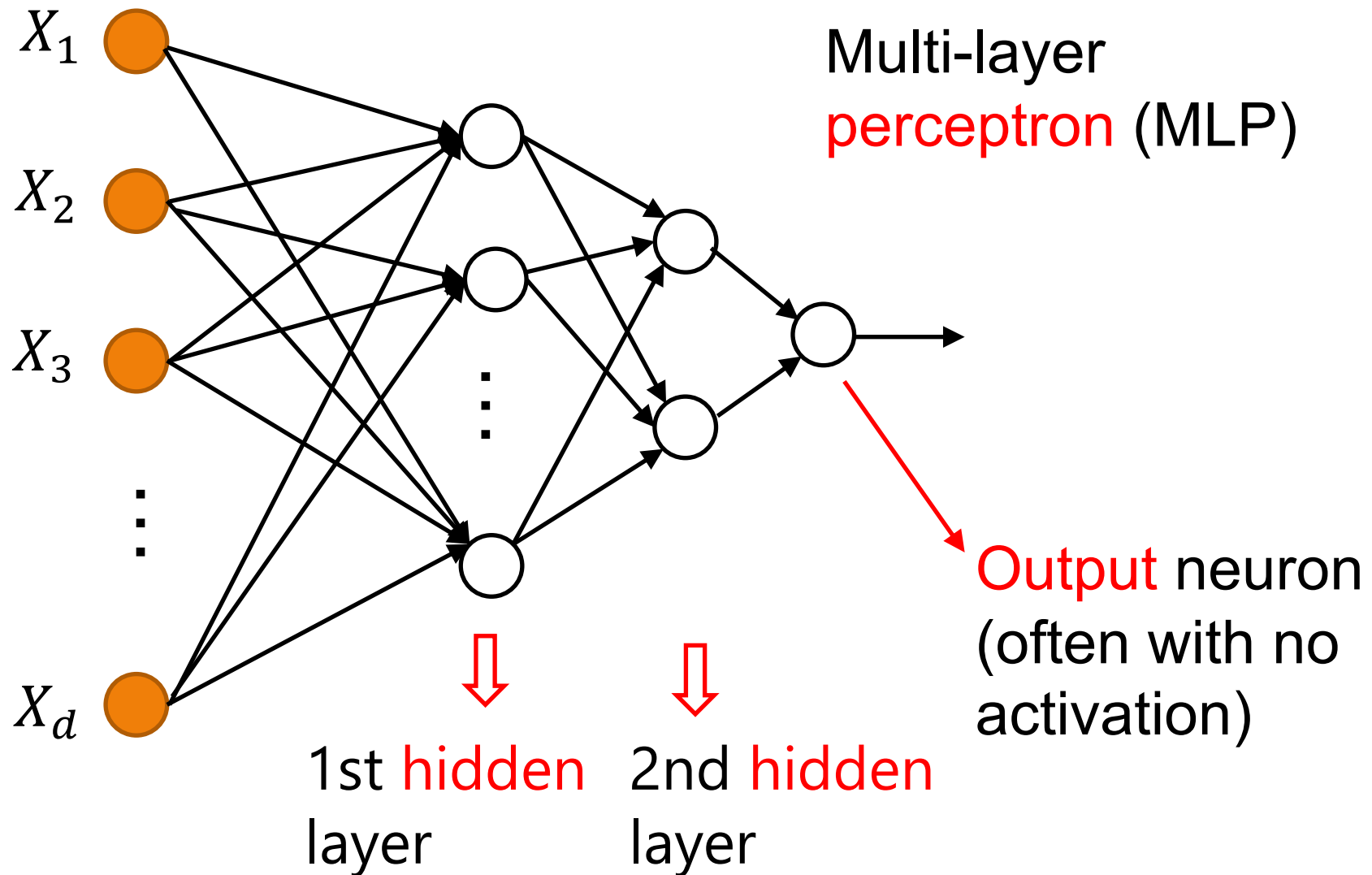
Two-Layer Networks are Universal Function Approximators

- ## Theorem (Th 9 in CIML):

Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer neural network $\hat{F}$ with a finite number of hidden units that approximates F arbitrarily well. Namely, for all x in the domain of F, $\left| F(\boldsymbol{x}) - \hat{F}(\boldsymbol{x}) \right| < \epsilon$

Classification using Neural Network

$X^{(i)}$

Sigmoid function

Probability of predicting Y^(i)=1

Neural Network function

$\phi_\theta(X)$

$g(.)$

$g(\phi_\theta(X))$

What is θ ?

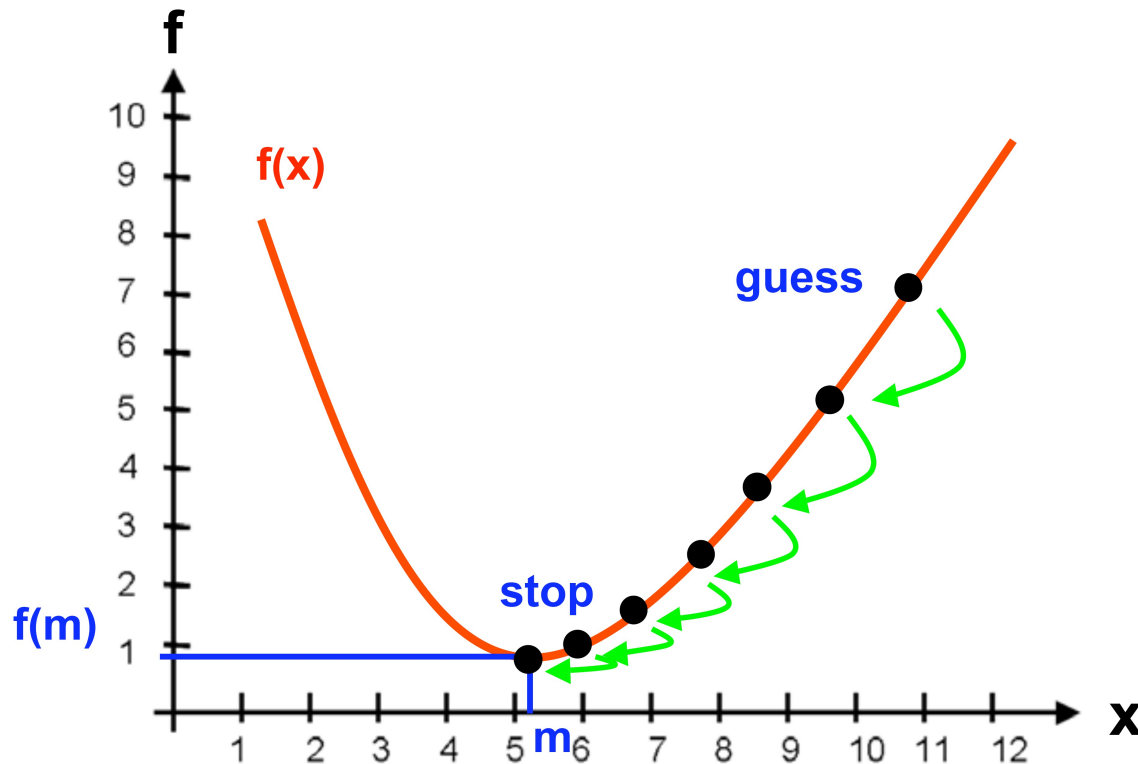Pick model parameters using cross-entropy loss optimization

$$\max_\theta \quad \sum_{i=1}^{N} Y^{(i)} \log g(\phi_\theta(X^{(i)})) + (1 - Y^{(i)}) \log(1 - g(\phi_\theta(X^{(i)})))$$

Two Questions

- What is a "good" family of nonlinear functions to consider? Neural Networks

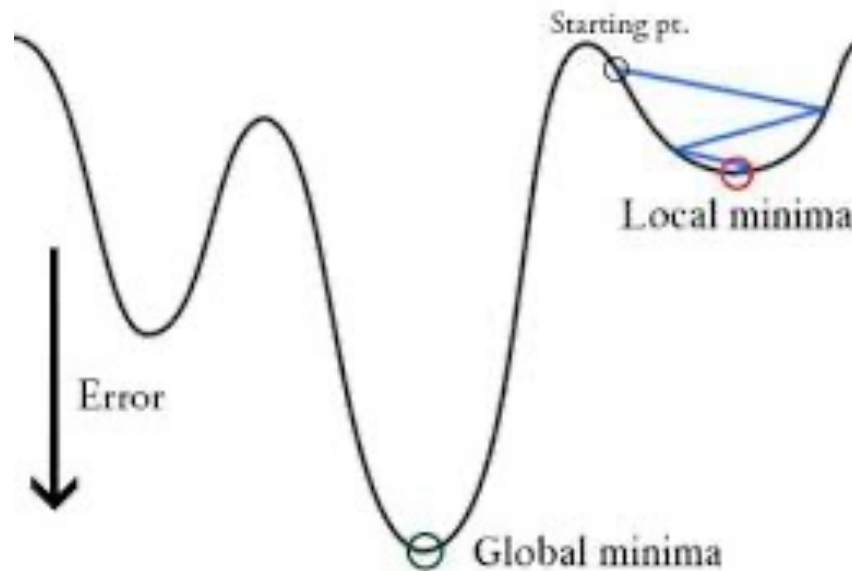- How to solve the resulting optimization? Can we still use stochastic GD?

Stochastic Gradient Descent

If the objective of optimization is convex

Stochastic Gradient Descent

If the objective of optimization is non-convex

In practice, SGD even in a non-convex deep learning optimization performs well. Why?

Stochastic Gradient Descent

What do we need to be able to use SGD in deep learning?

Computation of the gradient of the loss function with respect to model parameters

Next lecture, an efficient algorithm for this task!