

# UVA CS 4774: Machine Learning

## Lecture 12: Neural Network (NN) and More: BackProp

Dr. Yanjun Qi

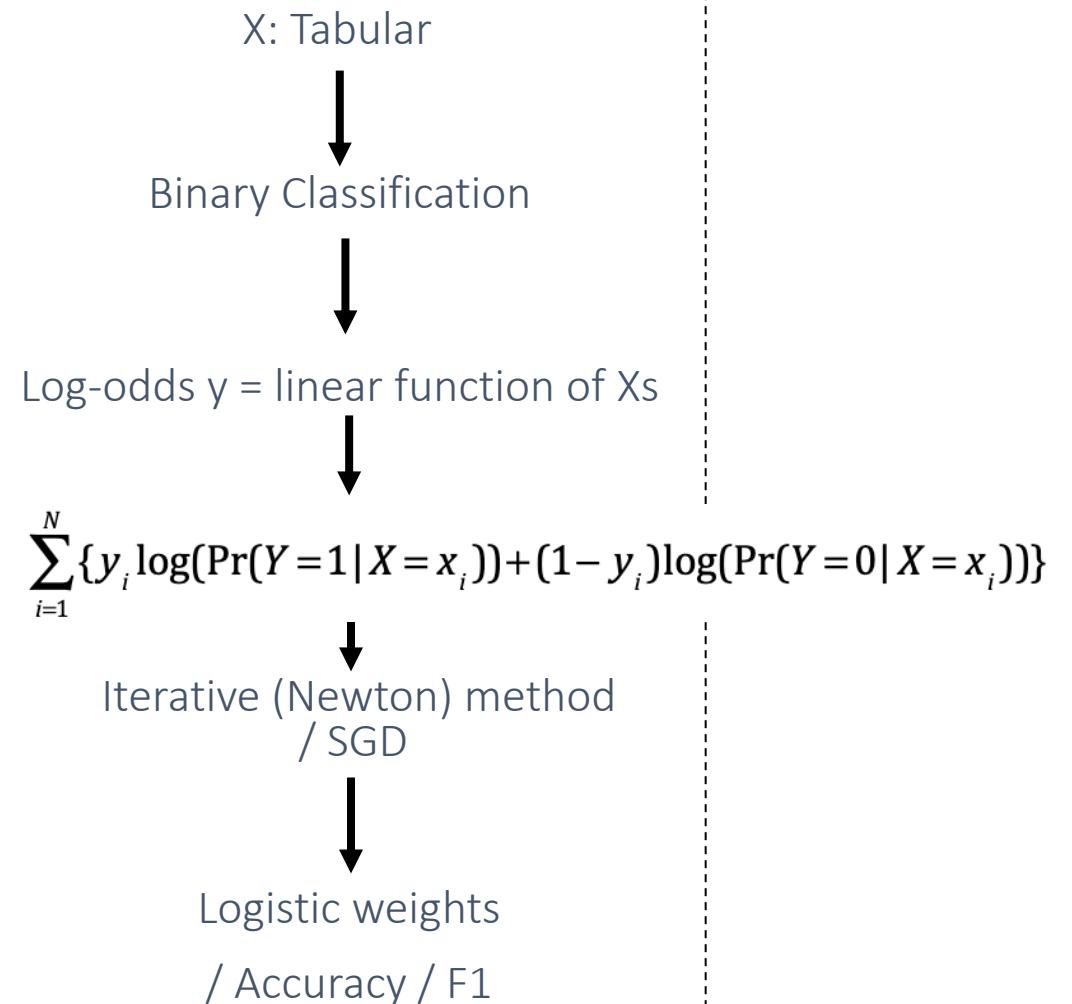
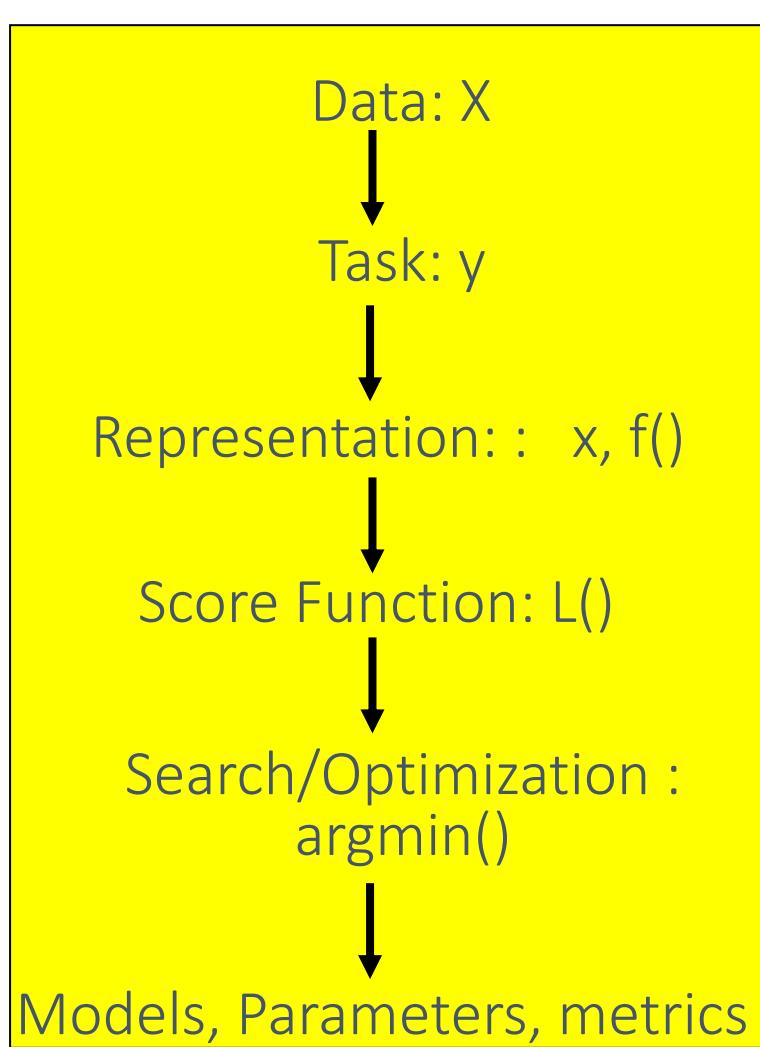
University of Virginia

Department of Computer Science

Last: Logistic Regression **Classifier**

$$P(y=1|x) = \frac{e^{\beta_0 + \beta^T x}}{1 + e^{\beta_0 + \beta^T x}}$$

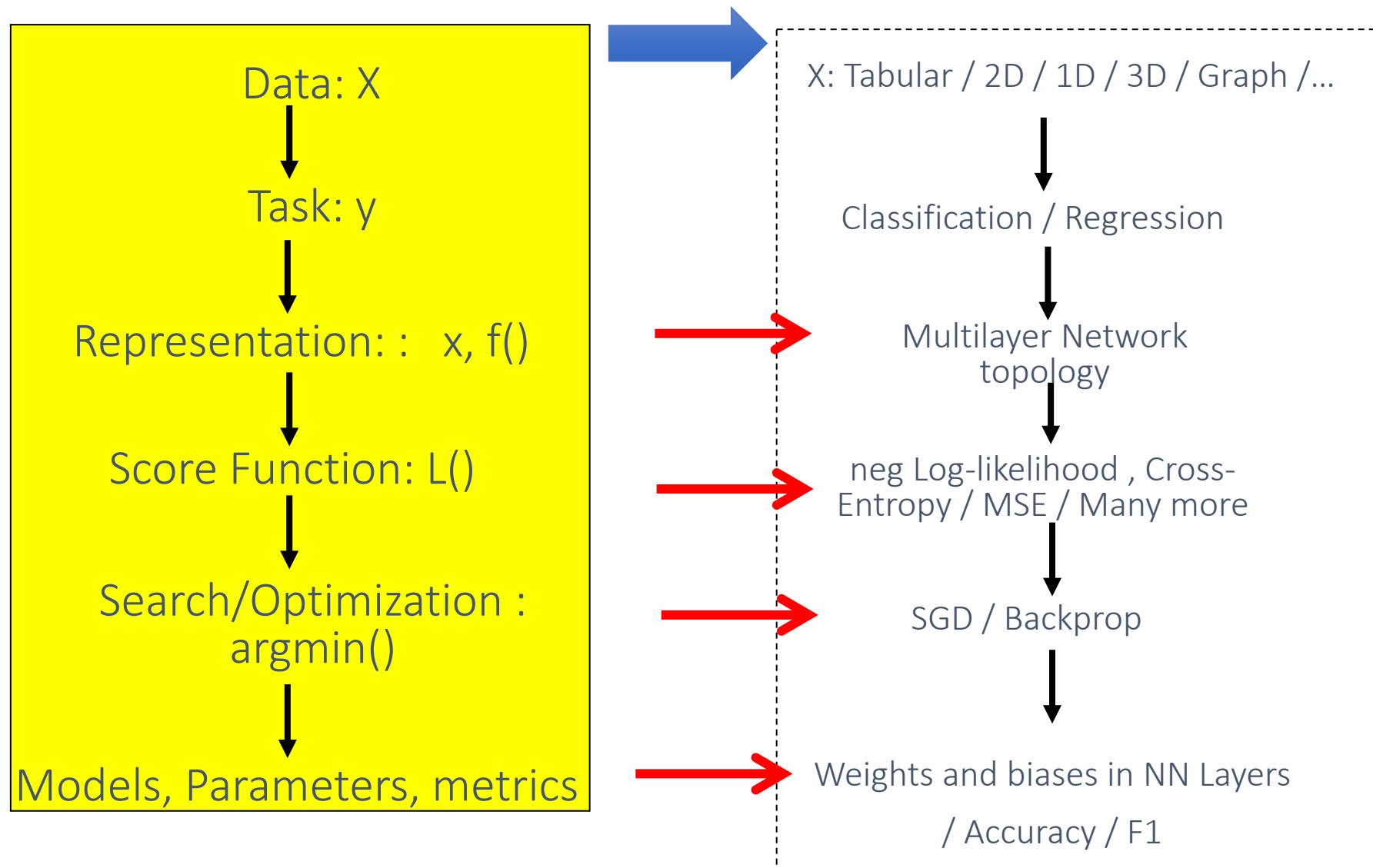
P(Head)



# Last: Logistic Regression Classifier

- View I:  $\text{logit}(y)$  as linear of Xs
- View II: model Y as Bernoulli with  $p(y=1|x)$  as  $p(\text{Head})$
- View III: S" shape function compress to  $[0,1]$
- View IV: models a linear classification boundary!
- View V: Two stages: summation + sigmoid

# Today: Basic Neural Network Models



# Roadmap: DNN Basics

- Basics of Neural Network (NN)
  - • single neuron, e.g. logistic regression unit
  - multilayer perceptron (MLP)
  - various loss function
    - E.g., when for multi-class classification, softmax layer
  - training NN with backprop algorithm
    - A few advanced tricks

# ReWrite Logistic Regression as two stages:

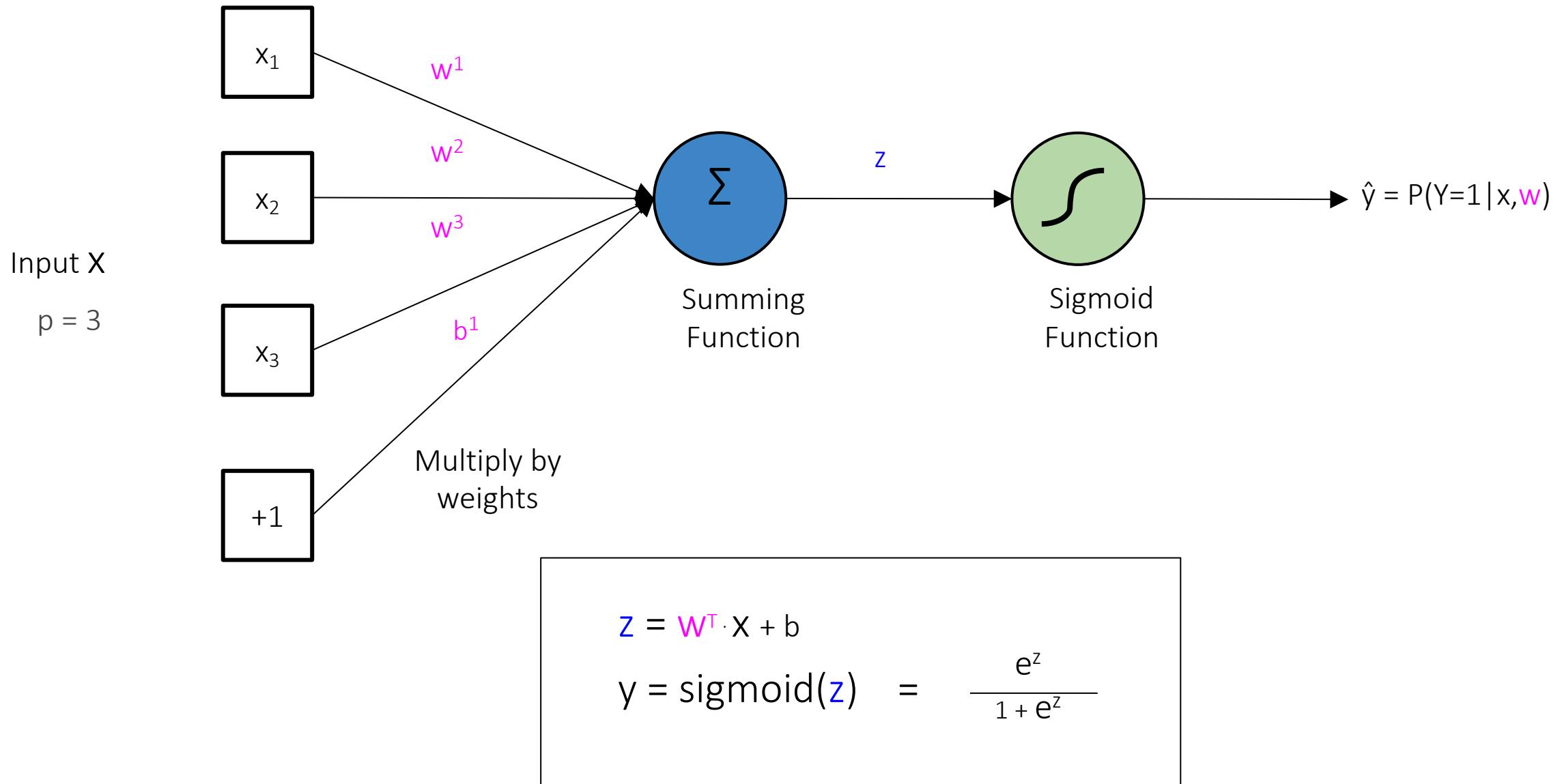
First:

Summing  $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$

Second:

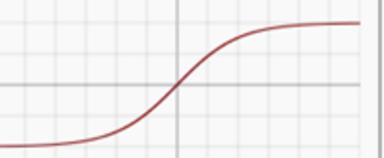
Sigmoid  $\hat{y} = P(y=1|x) = \frac{e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p}} = \frac{e^z}{1 + e^z}$

# One “Neuron”: Expanded Logistic Regression



# E.g., Many Possible Nonlinearity Functions

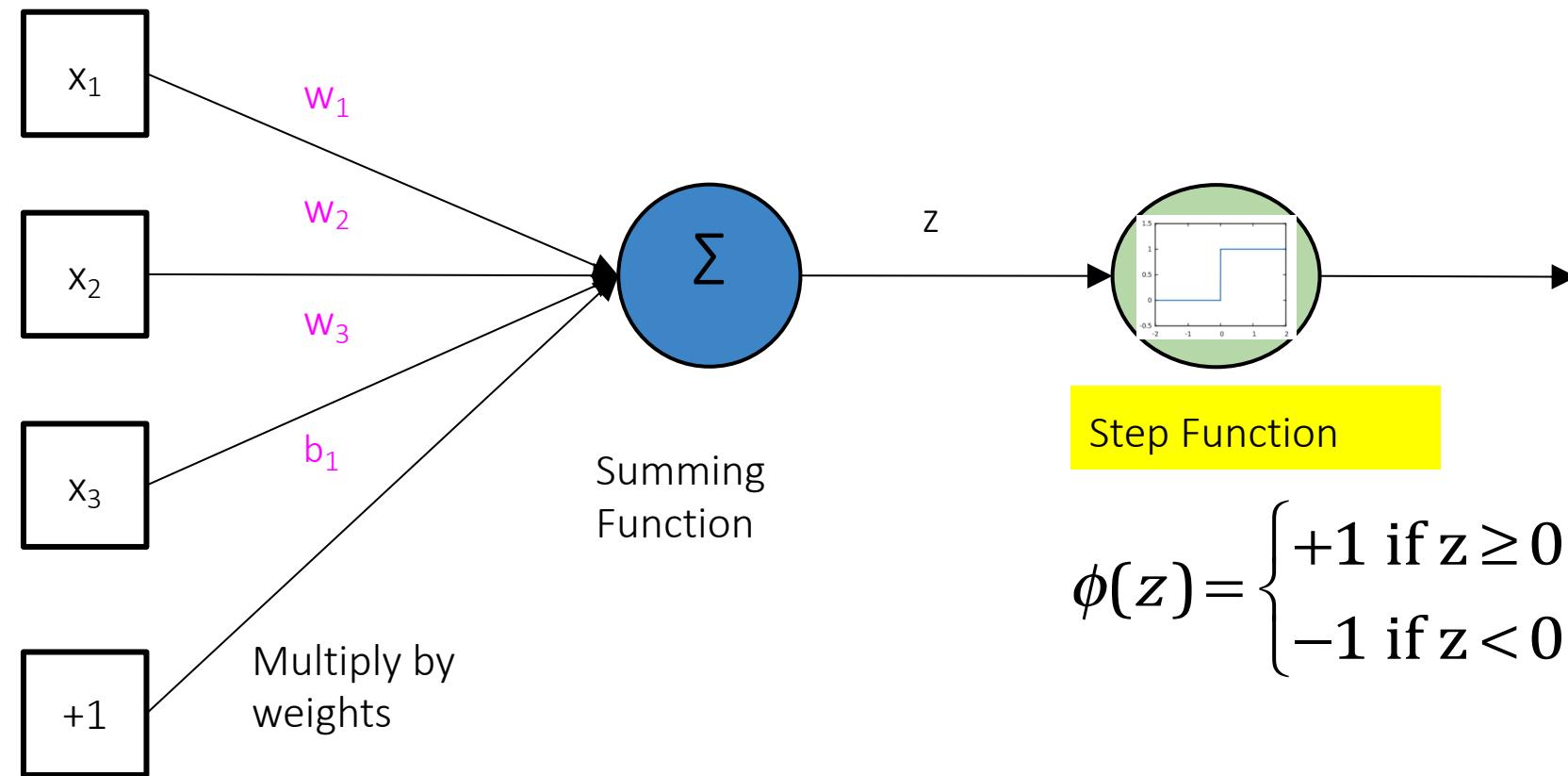
(aka transfer or activation functions)

Name	Plot	Equation	Derivative ( w.r.t x )
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)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Rectifier (ReLU) <sup>[9]</sup>		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

usually works best in practice

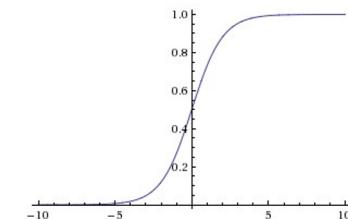
# History → Perceptron: 1-Neuron Unit with Step

- First proposed by Rosenblatt (1958)
- A simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function**

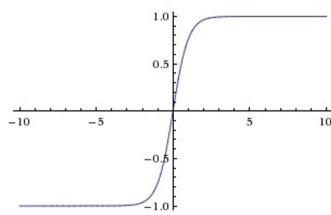


Sigmoid

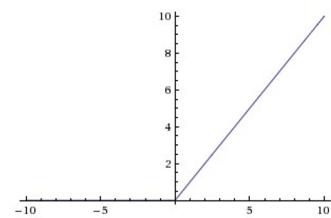
$$\sigma(x) = 1/(1 + e^{-x})$$



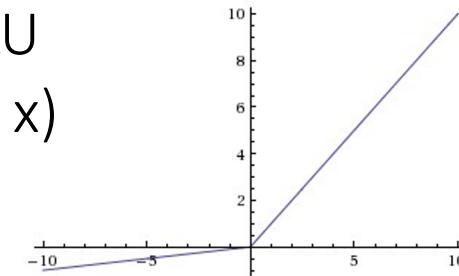
tanh  $\tanh(x)$



ReLU  $\max(0, x)$

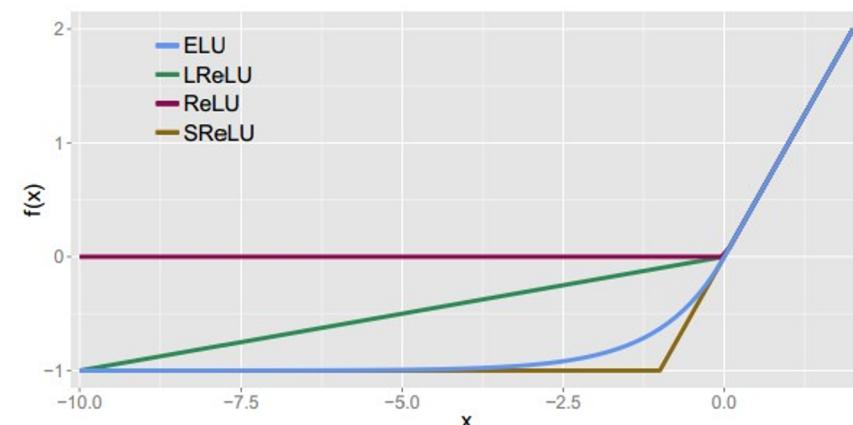


Leaky ReLU  
 $\max(0.1x, x)$

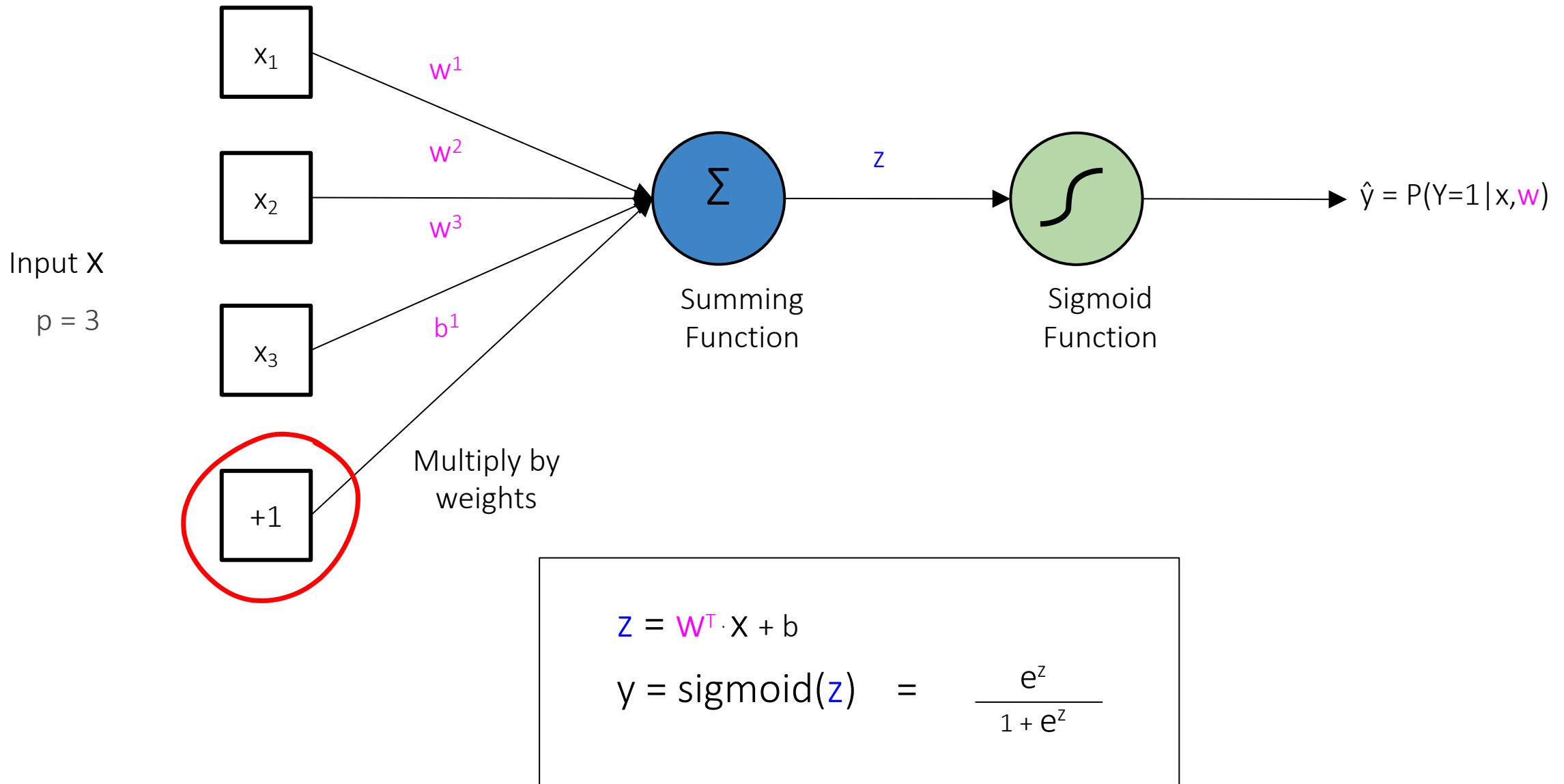


Maxout  $\max(w_1^T x + b_1, w_2^T x + b_2)$

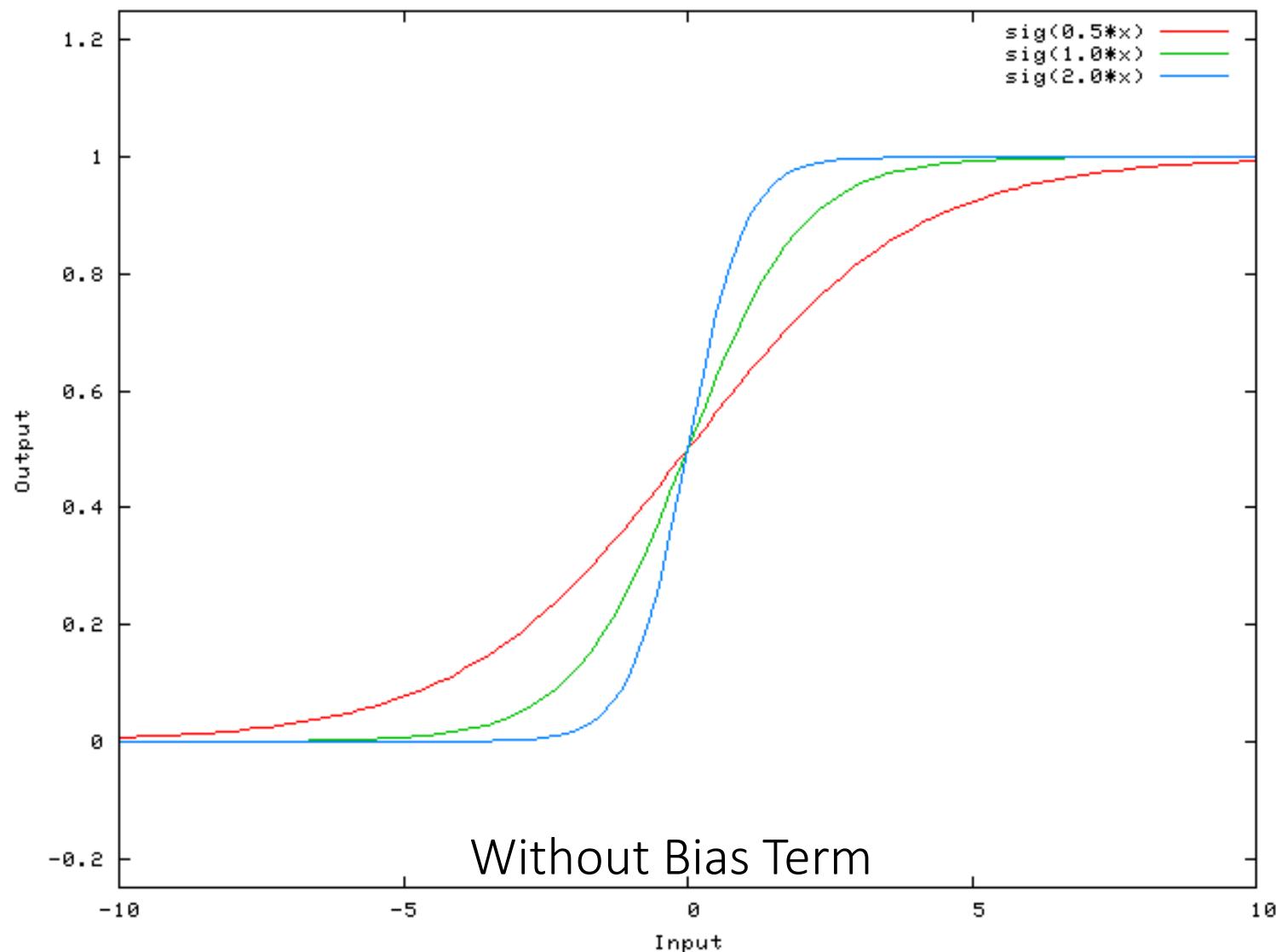
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



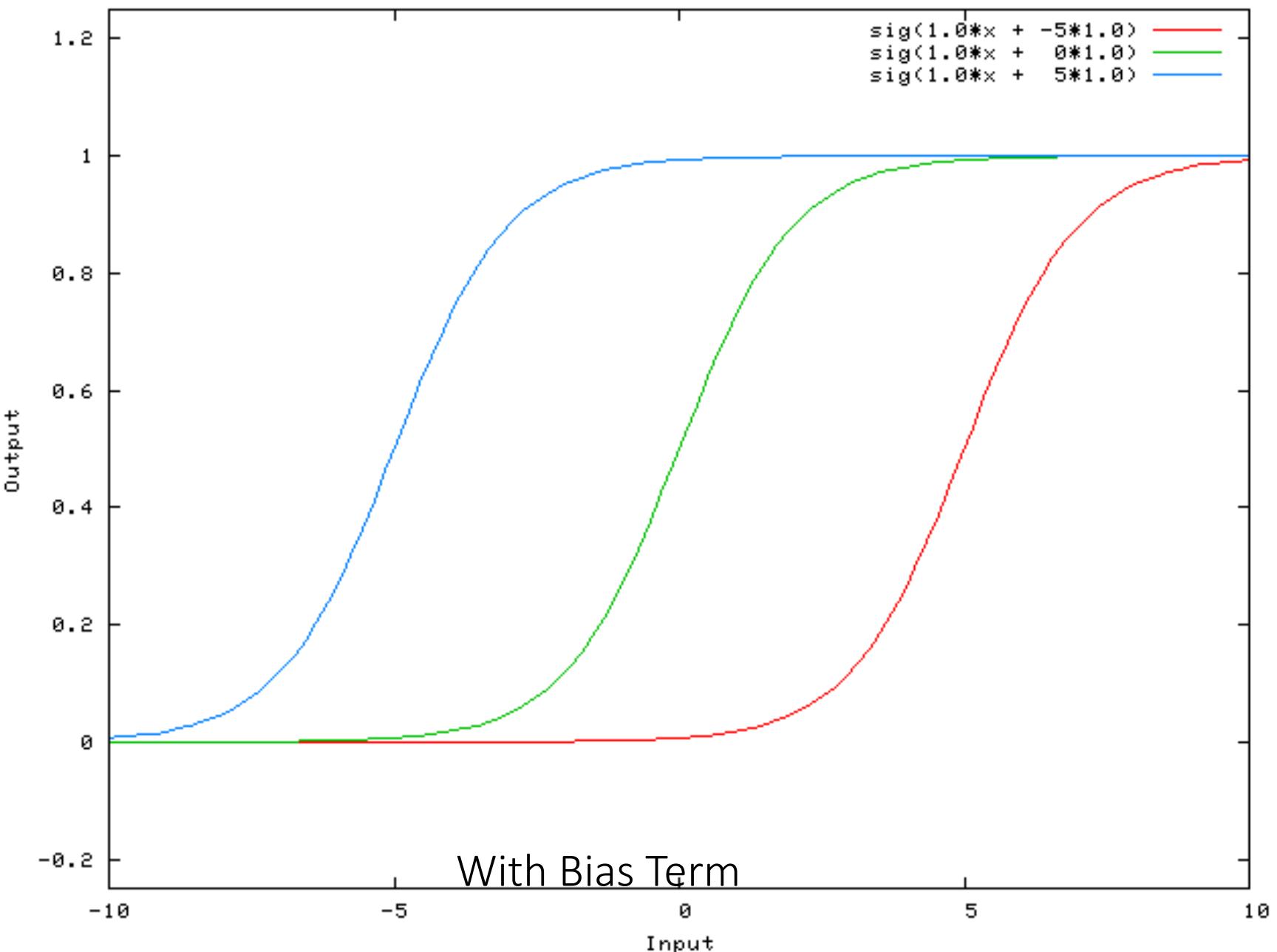
# Bias Term?



# Without Bias Term



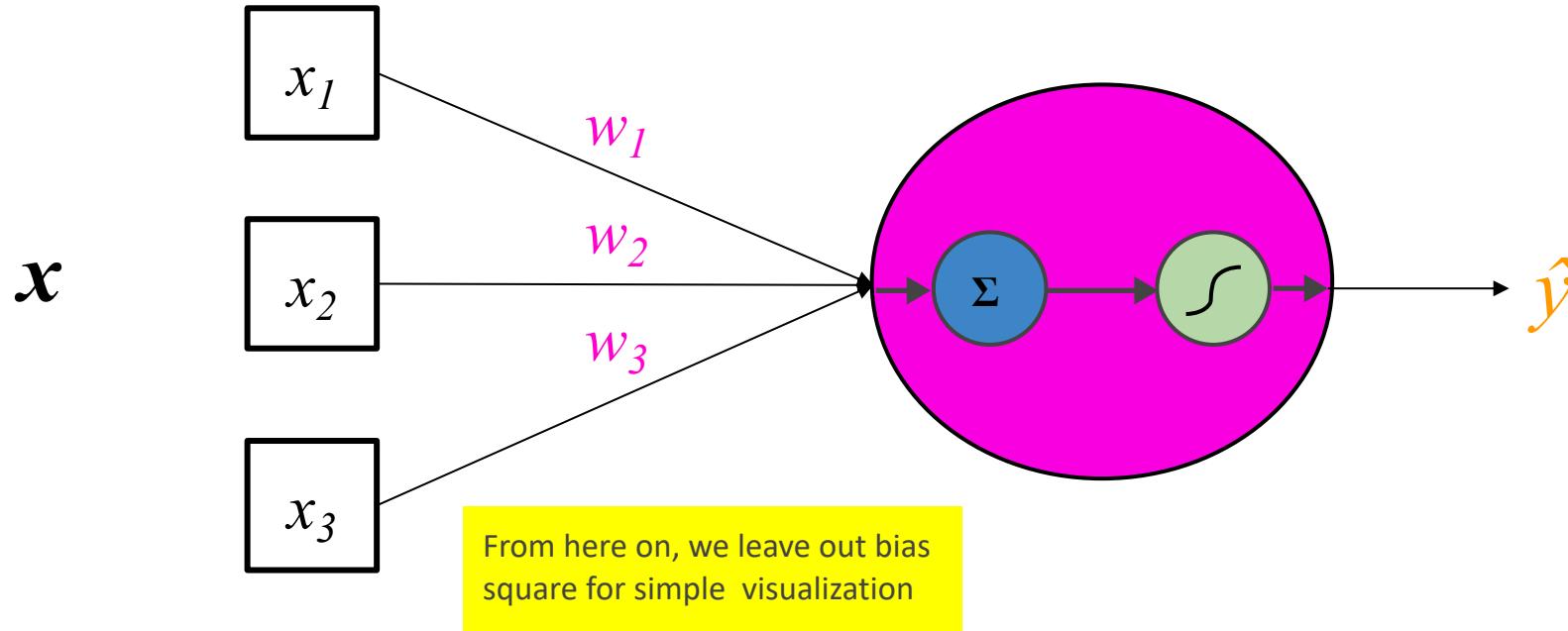
# With Bias Term



# Roadmap: DNN Basics

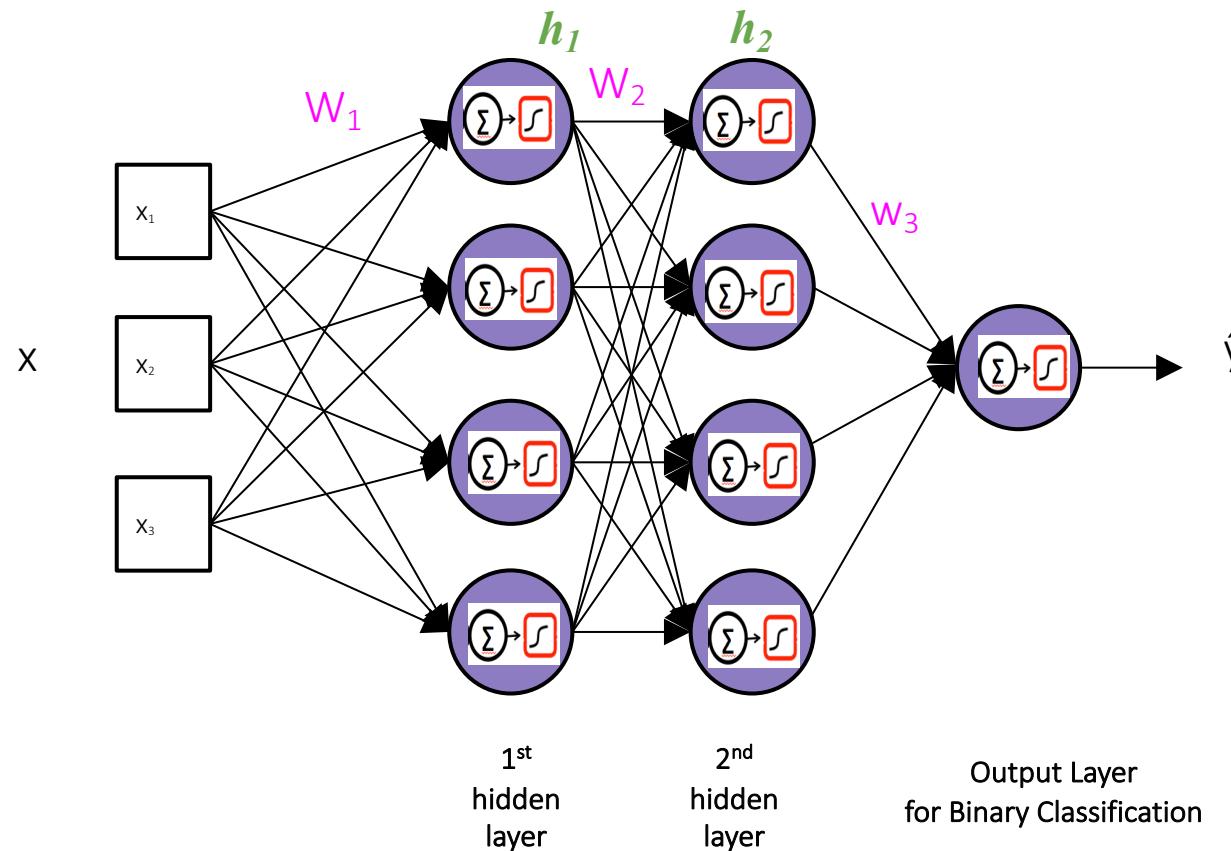
- Basics of Neural Network (NN)
    - single neuron, e.g. logistic regression unit
    - multilayer perceptron (MLP)
    - various loss function
      - E.g., when for multi-class classification, softmax layer
    - training NN with backprop algorithm
- 

# Neuron Representation

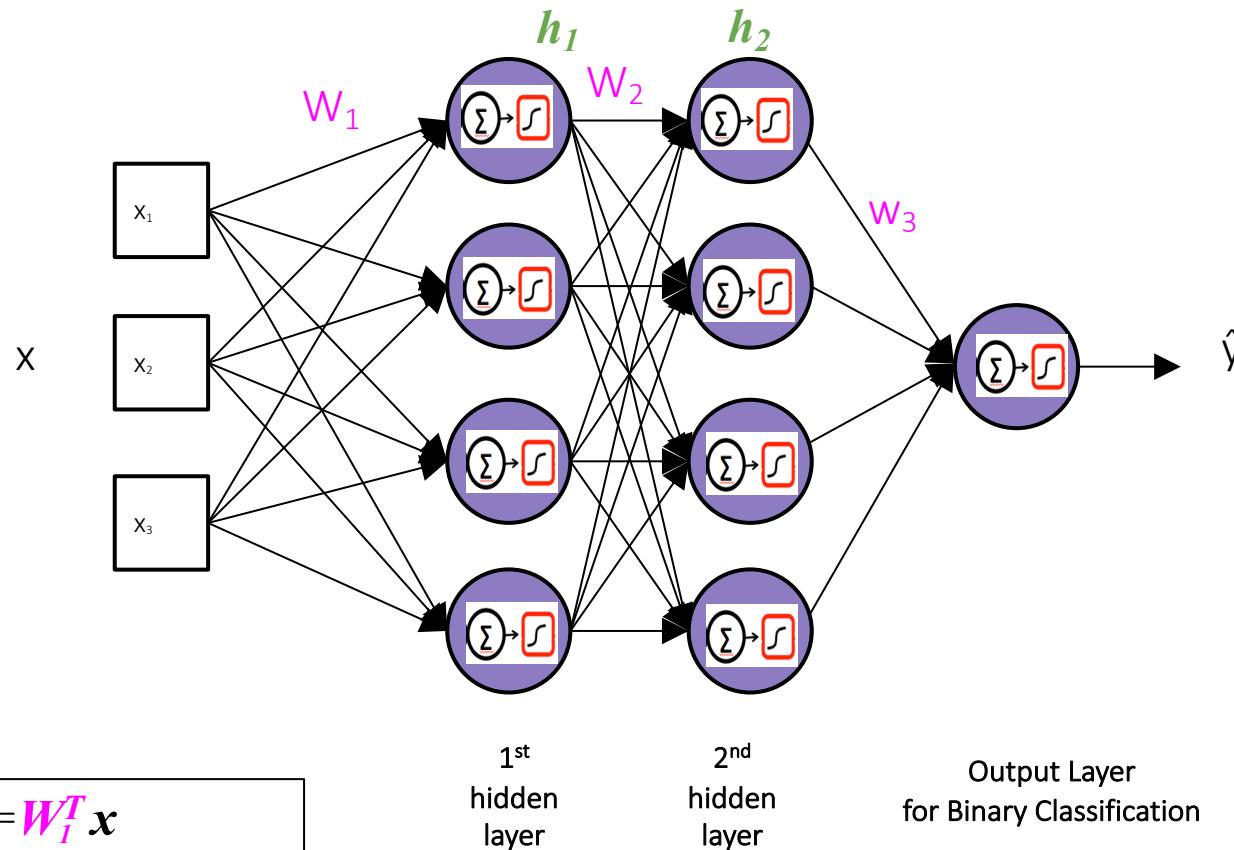


The linear transformation and nonlinearity together is typically considered a single neuron

# Multi-Layer Perceptron (MLP)- (Feed-Forward NN)



# Multi-Layer Perceptron (MLP)- (Feed-Forward NN)



hidden layer 1 output

$$z_1 = \mathbf{W}_1^T \mathbf{x}$$

$$\mathbf{h}_1 = \text{sigmoid}(\mathbf{z}_1)$$

$$z_2 = \mathbf{W}_2^T \mathbf{h}_1$$

$$\mathbf{h}_2 = \text{sigmoid}(\mathbf{z}_2)$$

$$z_3 = \mathbf{W}_3^T \mathbf{h}_2$$

$$\hat{y} = \text{sigmoid}(\mathbf{z}_3)$$

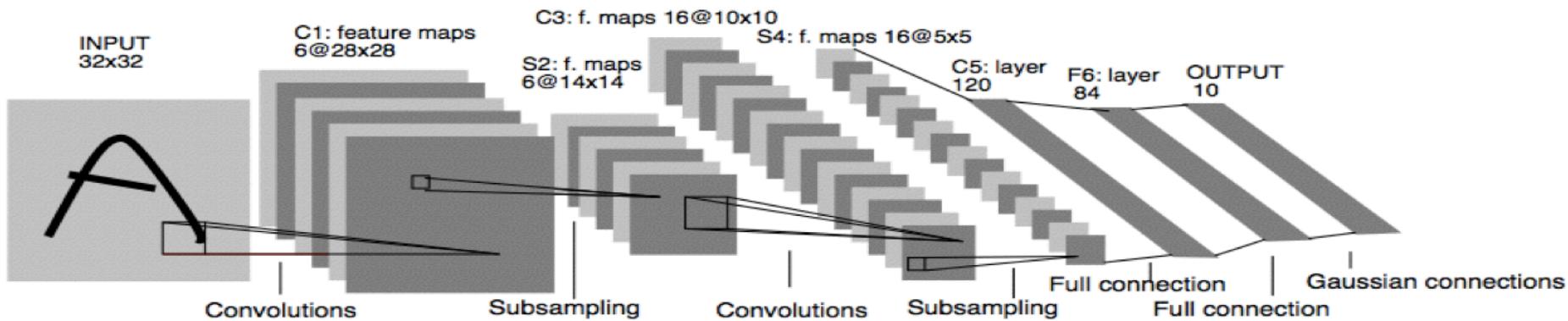
1<sup>st</sup>  
hidden  
layer

2<sup>nd</sup>  
hidden  
layer

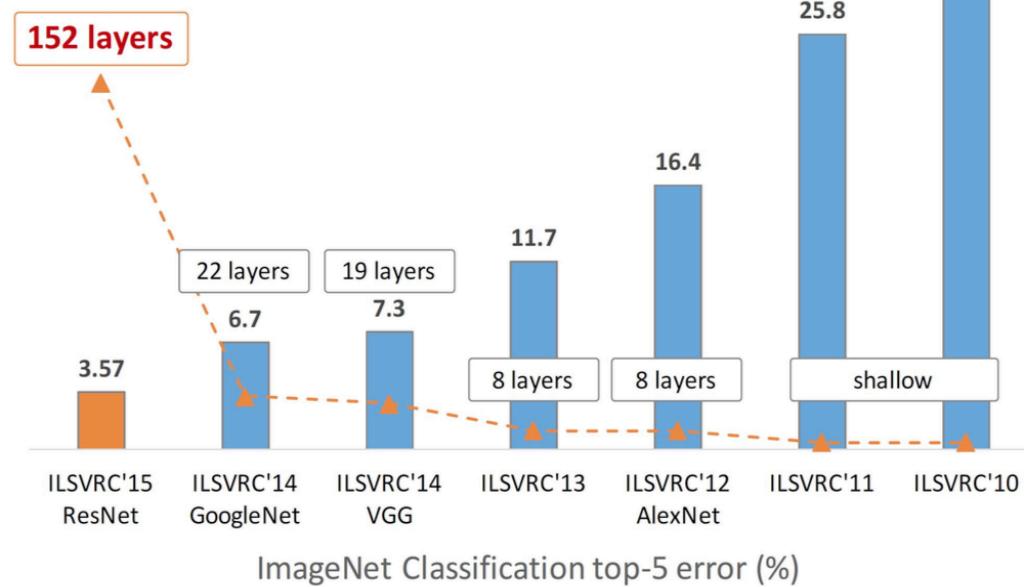
Output Layer  
for Binary Classification

hidden layer 2 output

# “Deep” Neural Networks (i.e. many hidden layers)



## Revolution of Depth



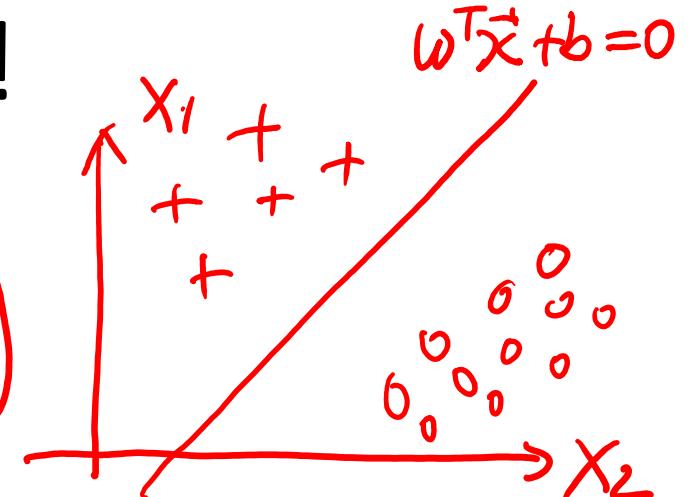
# View IV: Logistic Regression models a linear classification boundary!

$$y \in \{0,1\}$$

$$\ln \left[ \frac{P(y|x)}{1-P(y|x)} \right] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$$

$P(y=1|x)$   
 $= P(y=0|x)$

Boundary  
no decision

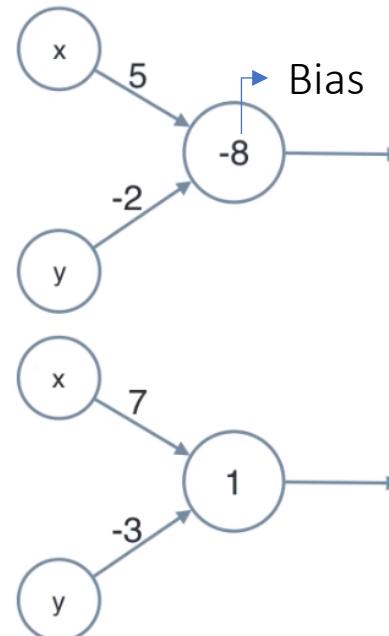
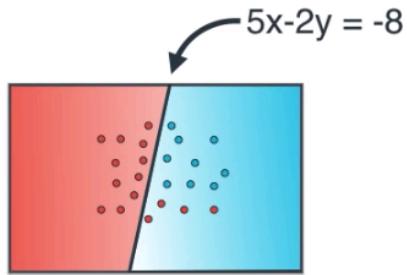


Decision Boundary → equals to zero

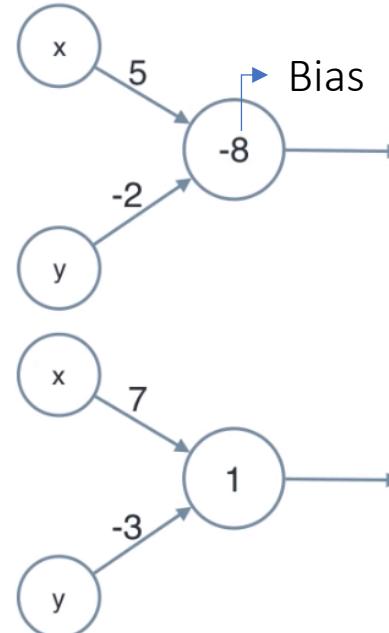
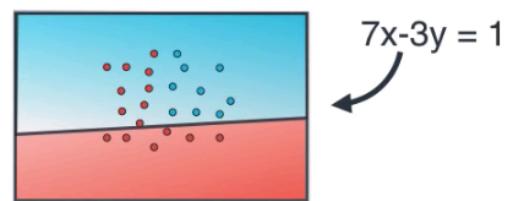
$$\ln \left[ \frac{P(y=1|x)}{P(y=0|x)} \right] = \ln \left[ \frac{P(y=1|x)}{1-P(y=1|x)} \right] = \alpha + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$$

$$= w^T \vec{x} + b = 0$$

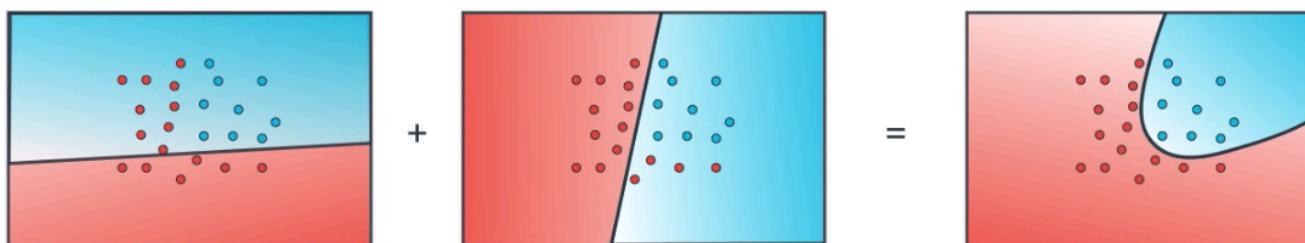
One neuron

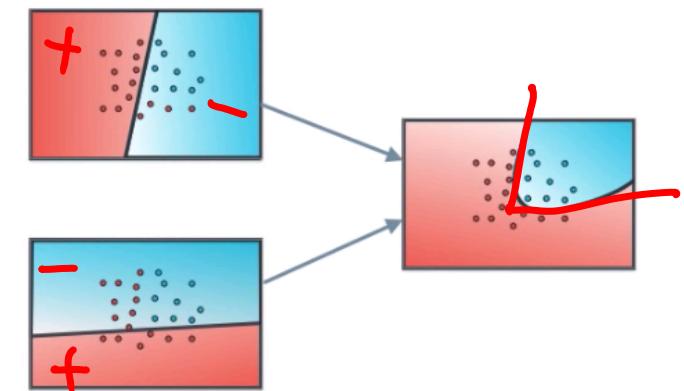
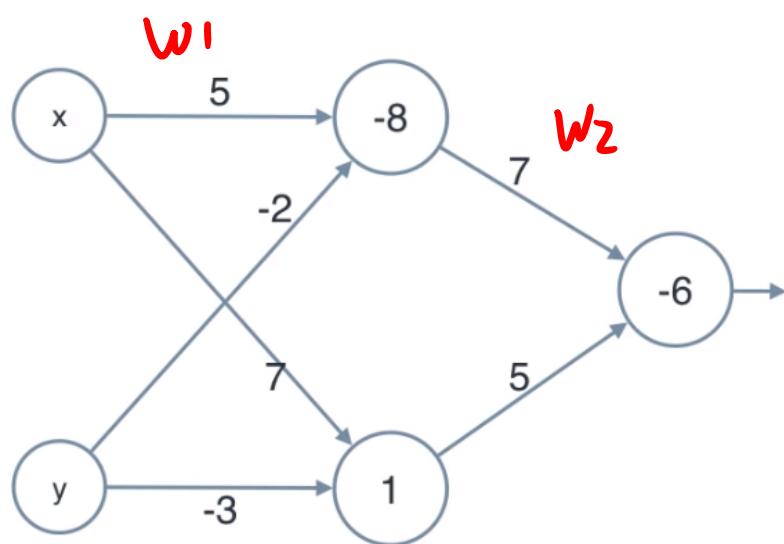
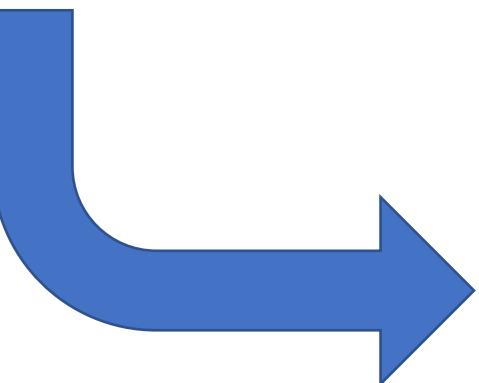
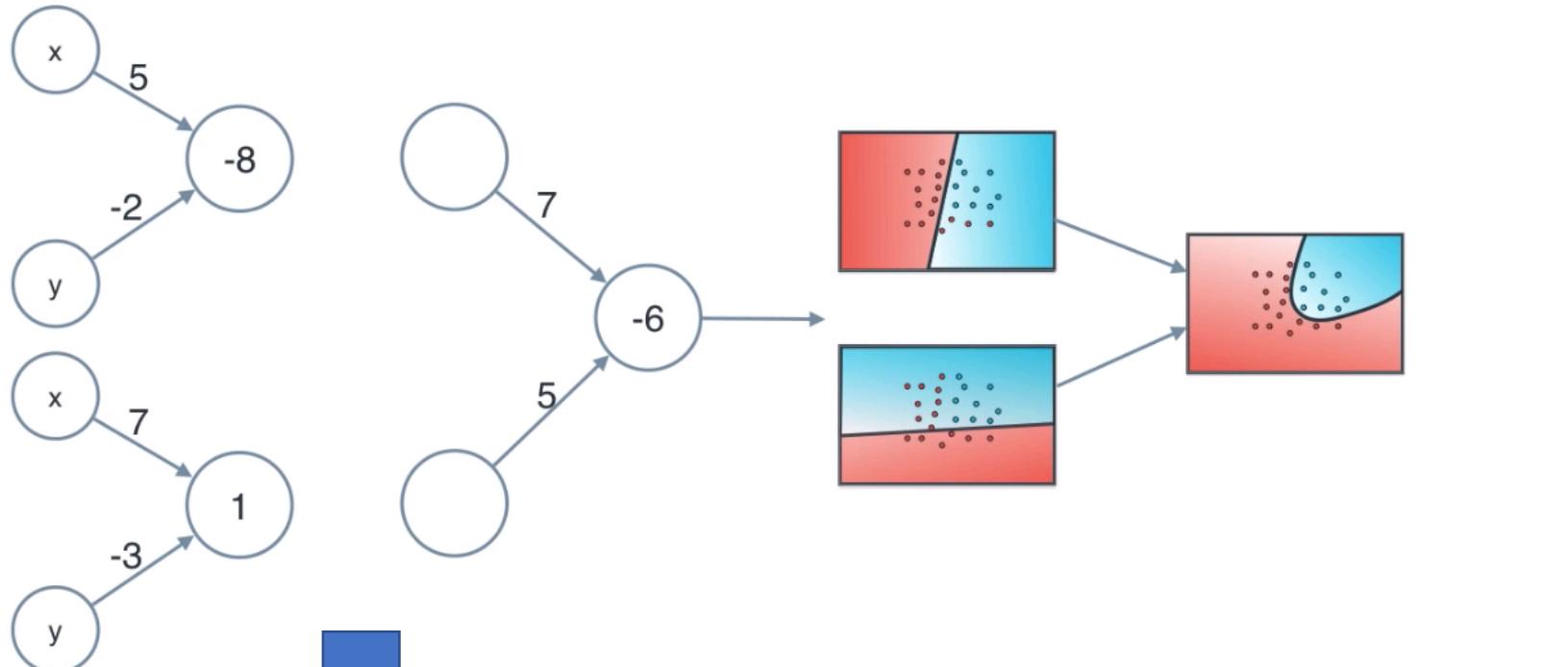


Another neuron

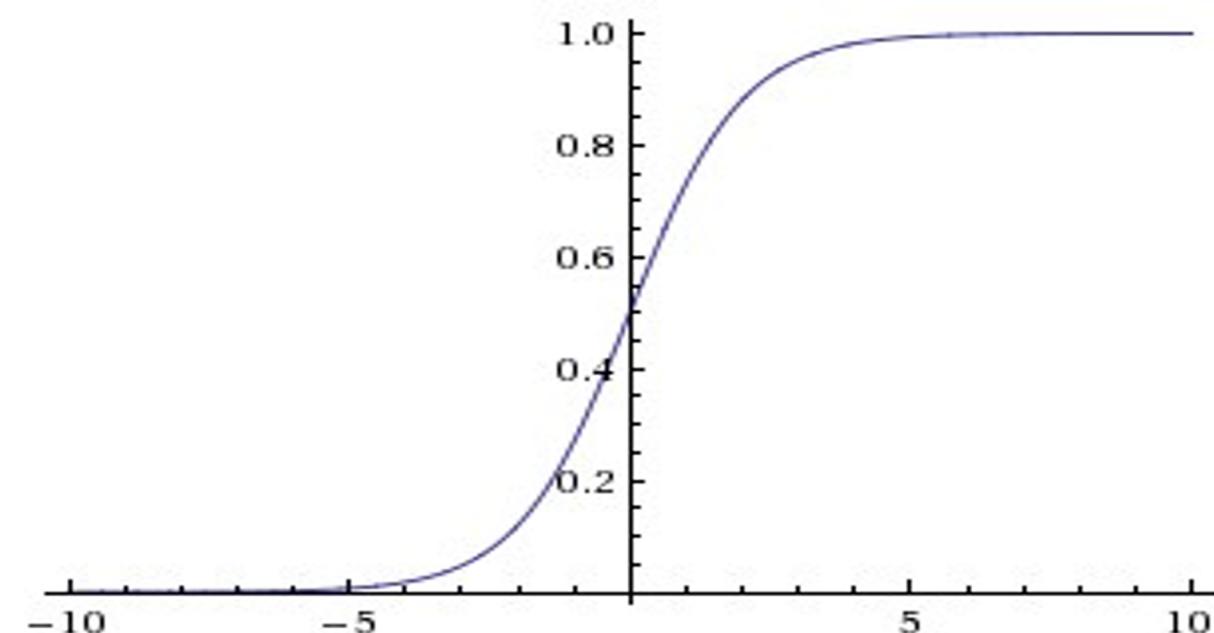
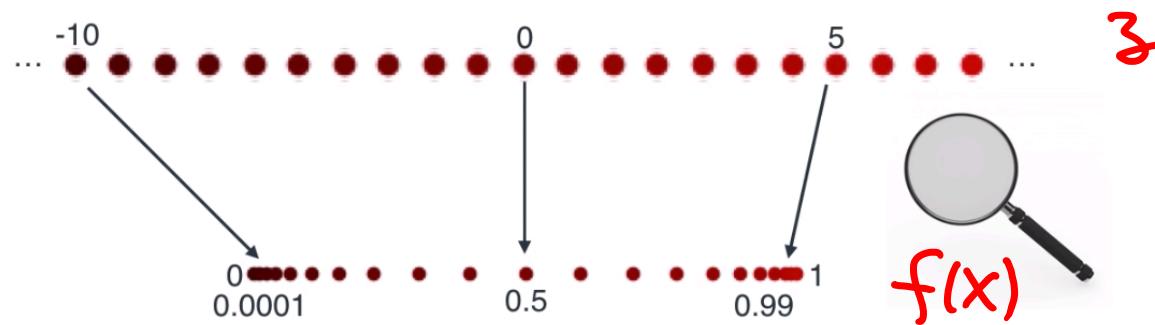


How to combine  
to get nonlinear  
decision  
boundary?





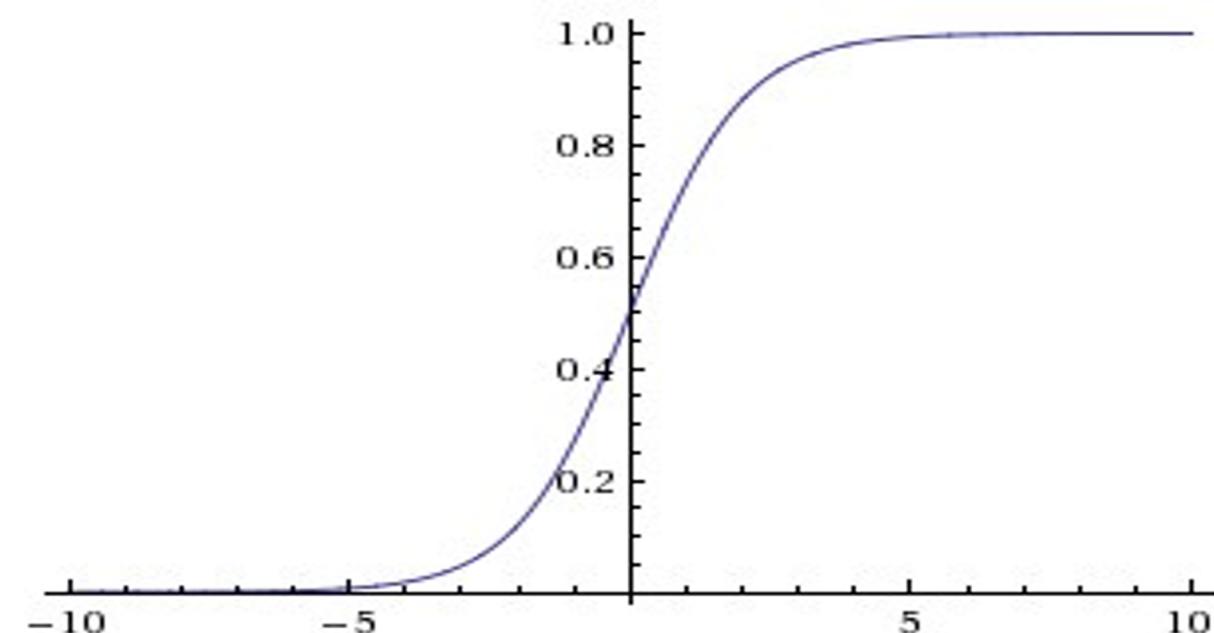
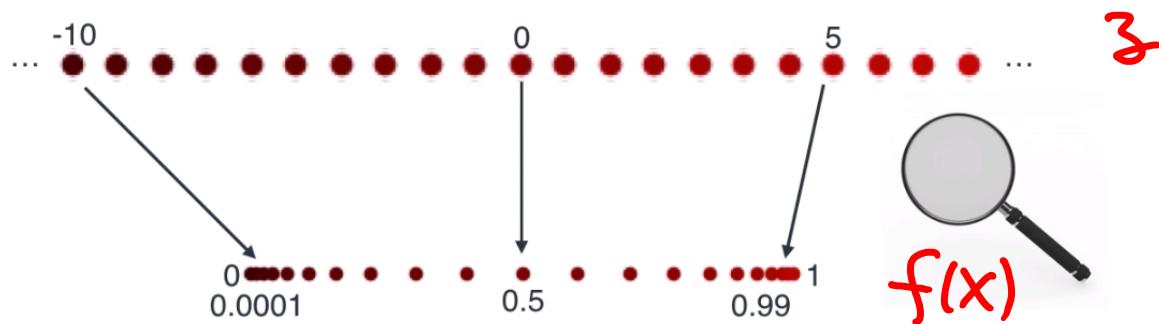
## Activation function



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

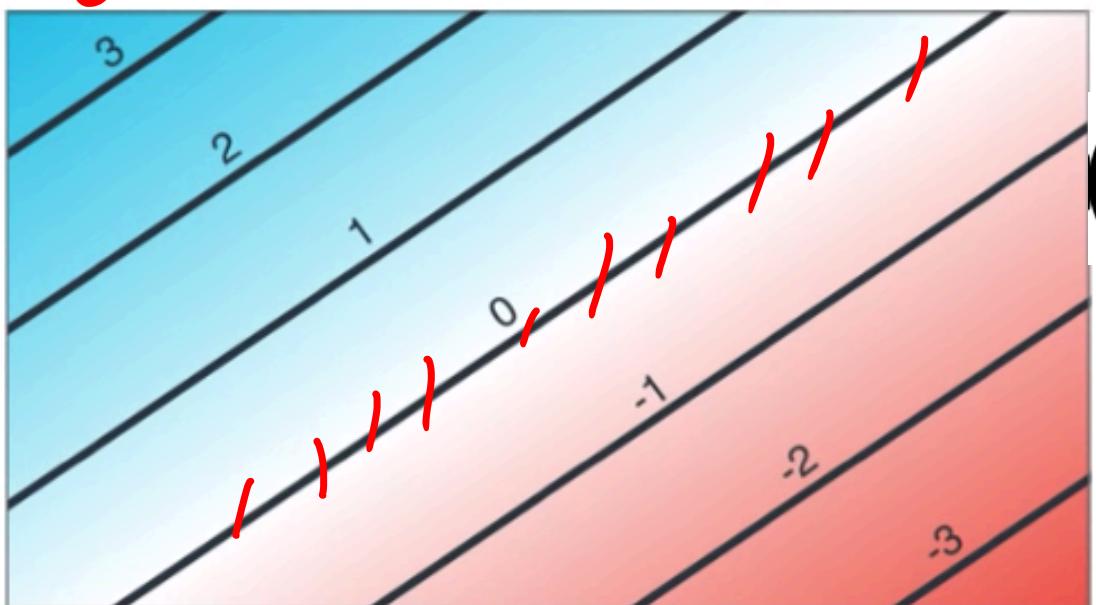


## Activation function



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

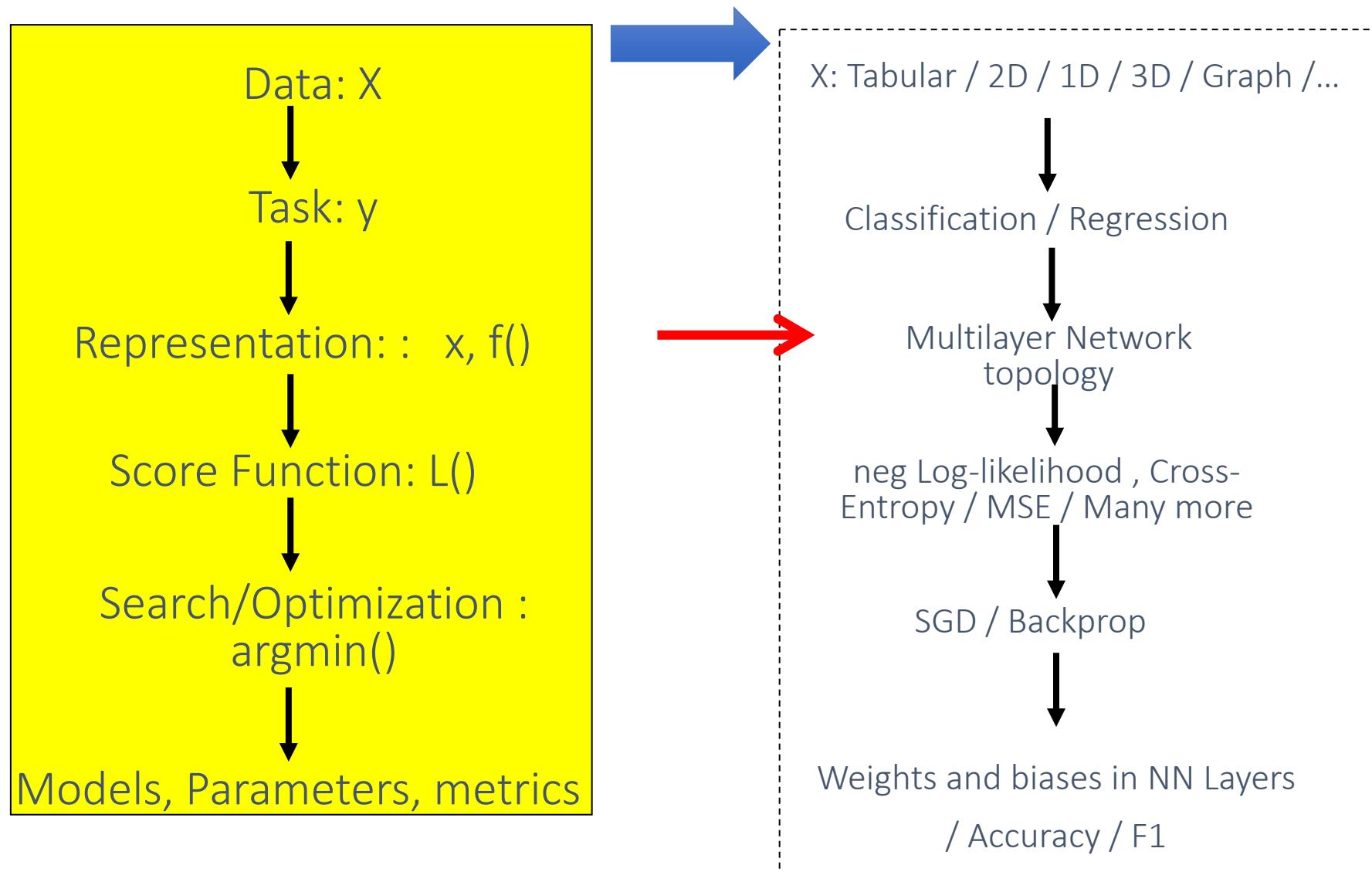
$x \rightarrow \sum \rightarrow \text{sig} \rightarrow f(x)$



$$\Sigma \rightarrow \text{sig}$$



# Today: Basic Neural Network Models



# Thank You



# UVA CS 4774: Machine Learning

## Lecture 12: Neural Network (NN) and More: BackProp

Dr. Yanjun Qi

University of Virginia

Department of Computer Science

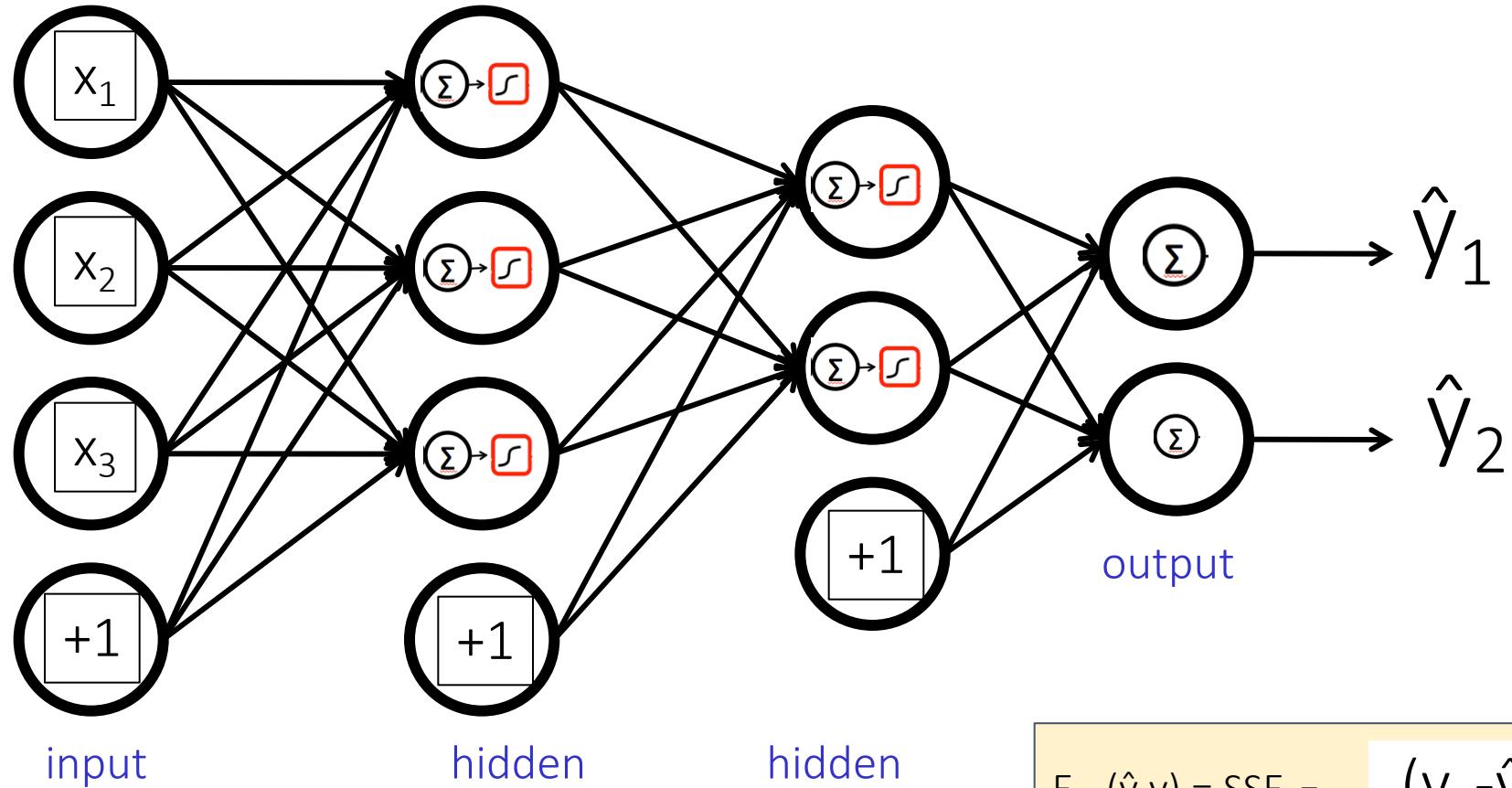
Module II

# Roadmap: DNN Basics

- Basics of Neural Network (NN)
    - single neuron, e.g. logistic regression unit
    - multilayer perceptron (MLP)
    - various loss function
      - E.g., when for multi-class classification, softmax layer
    - training NN with backprop algorithm
- 

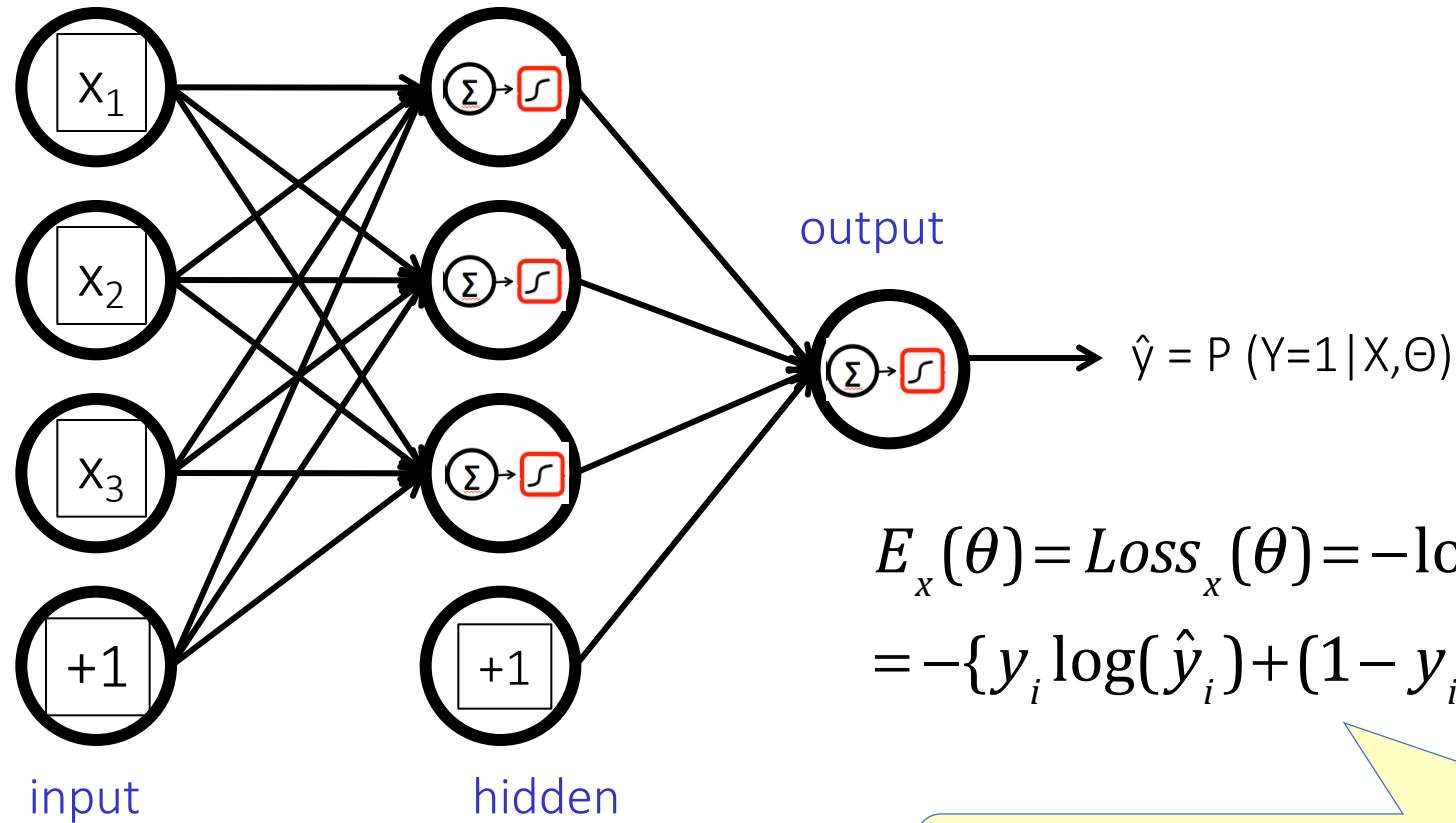
# E.g., SSE loss on Multi-Layer Perceptron (MLP) for Regression

Example: 2 Hidden Layer MLP network with 2 output units:



$$E_w(\hat{y}, y) = \text{SSE} = (y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2$$

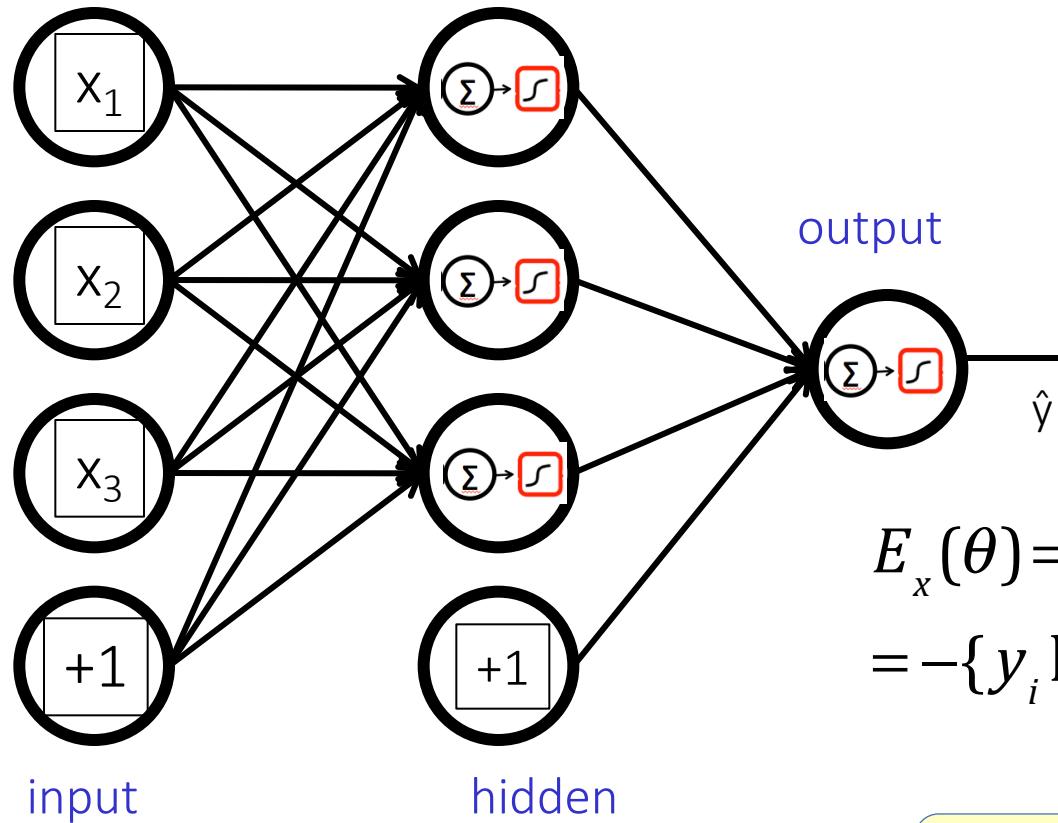
e.g., Cross-Entropy loss for Multi-Layer Perceptron (MLP) for Binary Classification



$$E_x(\theta) = \text{Loss}_x(\theta) = -\log \Pr(Y = y | X = x)$$
$$= -\{y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\}$$

Cross-entropy loss function, OR named as  
“deviance”, OR negative log-likelihood

e.g., Cross-Entropy loss for Multi-Layer Perceptron (MLP) for Binary Classification



For Bernoulli distribution,

$$p(y=1|x)^y(1-p)^{1-y}$$

$$\hat{y} = P(Y=1|X,\Theta)$$

$$\begin{aligned} E_x(\theta) &= Loss_x(\theta) = -\log Pr(Y=y | X=x) \\ &= -\{y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i)\} \end{aligned}$$

Cross-entropy loss function, OR named as  
“deviance”, OR negative log-likelihood

LIKELIHOOD:

$$L(p) = \prod_{i=1}^n p^{z_i} (1-p)^{1-z_i}$$

↑  
function of  $p = \Pr(\text{head})$

Basile Bernoulli      {      Logistic / Bernoulli

$$\begin{aligned} L(\beta) &= \prod_{i=1}^n p(y_i=1|x_i)^{y_i} (1-p(y_i=1|x_i))^{\hat{y}_i} \\ &= \prod_{i=1}^n \hat{y}_i^{y_i} (1-\hat{y}_i)^{1-y_i} \end{aligned}$$

$$\begin{aligned} \log(L(p)) &= \log \left[ \prod_{i=1}^n p^{z_i} (1-p)^{1-z_i} \right] \\ &= \sum_{i=1}^n (z_i \log p + (1-z_i) \log(1-p)) \end{aligned}$$

Log likelihood

$$\ell(\beta) = \sum_{i=1}^n \left( y_i \log \hat{y}_i + (1-y_i) \log(1-\hat{y}_i) \right)$$

# Binary Classification → Multi-Class Classification

models the target binary random variable with Bernoulli whose parameter  $p=p(y=1|x)$  predefined as function on  $x$



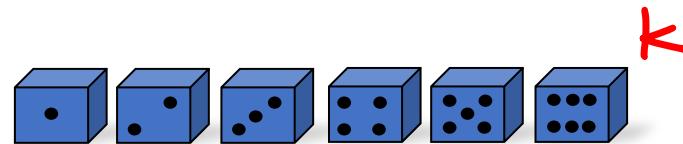
$$1-p(y=1|x)$$

$$P(y=1|x)$$

$$= \frac{e^z}{1 + e^z}$$



Multinoulli Distribution, e.g.,



$$p(y=1|x)$$

$$p(y=2|x)$$

$$p(y=3|x)$$

:

$$p(y=k|x)$$

## Multi-class target variable representation

K=4

Total Class  $\downarrow$

$\vec{y}_K$

$g$	$y_1$	$y_2$	$y_3$	$y_4$
3	0	0	1	0
1	1	0	0	0
2	0	1	0	0
4	0	0	0	1
1	1	0	0	0

↓



- Multi-class output variable → An indicator basis vector representation
  - If output variable  $G$  has  $K$  classes, there will be  $K$  indicator variable  $y_i$

## Review: Multi-class variable representation

Class  $\gamma_K$

$g$	$y_1$	$y_2$	$y_3$	$y_4$
3	0	0	1	0
1	1	0	0	0
2	0	1	0	0
4	0	0	0	1
1	1	0	0	0

N ↓



- Multi-class output variable → An indicator basis vector representation
  - If output variable  $G$  has  $K$  classes, there will be  $K$  indicator variable  $y_i$
- How to classify to multi-class ?
  - First: learn  $K$  different regression
  - Then: Softmax using all  $K$  outputs as input

## Review: Multi-class variable representation

Class	$y_1$	$y_2$	$y_3$	$y_4$
$g$				
3	0	0	1	0
1	1	0	0	0
2	0	1	0	0
4	0	0	0	1
1	1	0	0	0

- Multi-class output variable → An indicator basis vector representation
  - If output variable  $G$  has  $K$  classes, there will be  $K$  indicator variable  $y_i$
- How to classify to multi-class ?
  - First: learn  $K$  different regression
  - Then: Softmax using all  $K$  outputs as input
  - Then:  
 $\rightarrow \hat{G}(x) = \operatorname{argmax}_{k \in g} \hat{f}_k(x)$   
Identify the largest component of  $\hat{f}(x)$   
And Classify according to MAP Rule

## Review: Multi-class variable representation

Class  $y_k$

$g$	$y_1$	$y_2$	$y_3$	$y_4$
3	0	0	1	0
1	1	0	0	0
2	0	1	0	0
4	0	0	0	1
1	1	0	0	0

N

discriminative classifier

- Multi-class output variable → An indicator basis vector representation

- If output variable  $G$  has  $K$  classes, there will be  $K$  indicator variable  $y_i$

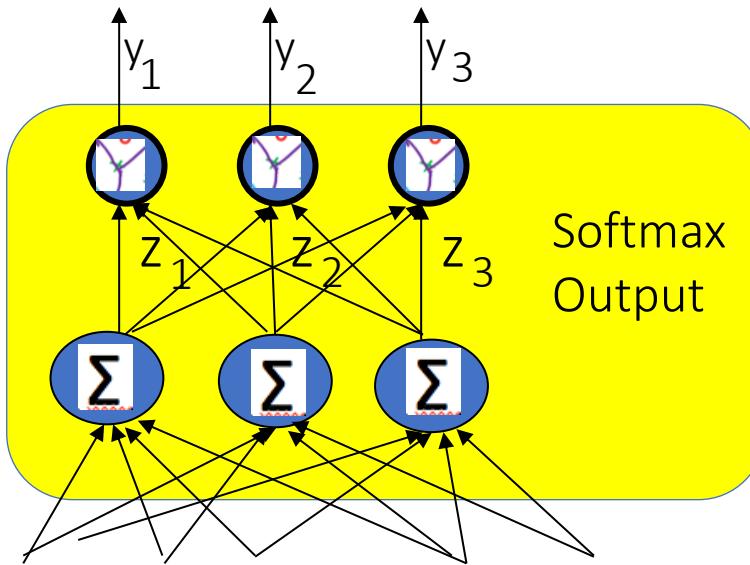
- How to classify to multi-class ?

- First: learn  $K$  different regression
- Then: Softmax using all  $K$  outputs as input
- Then:

$$\rightarrow \hat{G}(x) = \operatorname{argmax}_{k \in g} \hat{f}_k(x) p_k(x)$$

Identify the largest component of  $\hat{f}(x)$   
And Classify according to MAP Rule

## Strategy : Use “softmax” layer function for multi-class classification



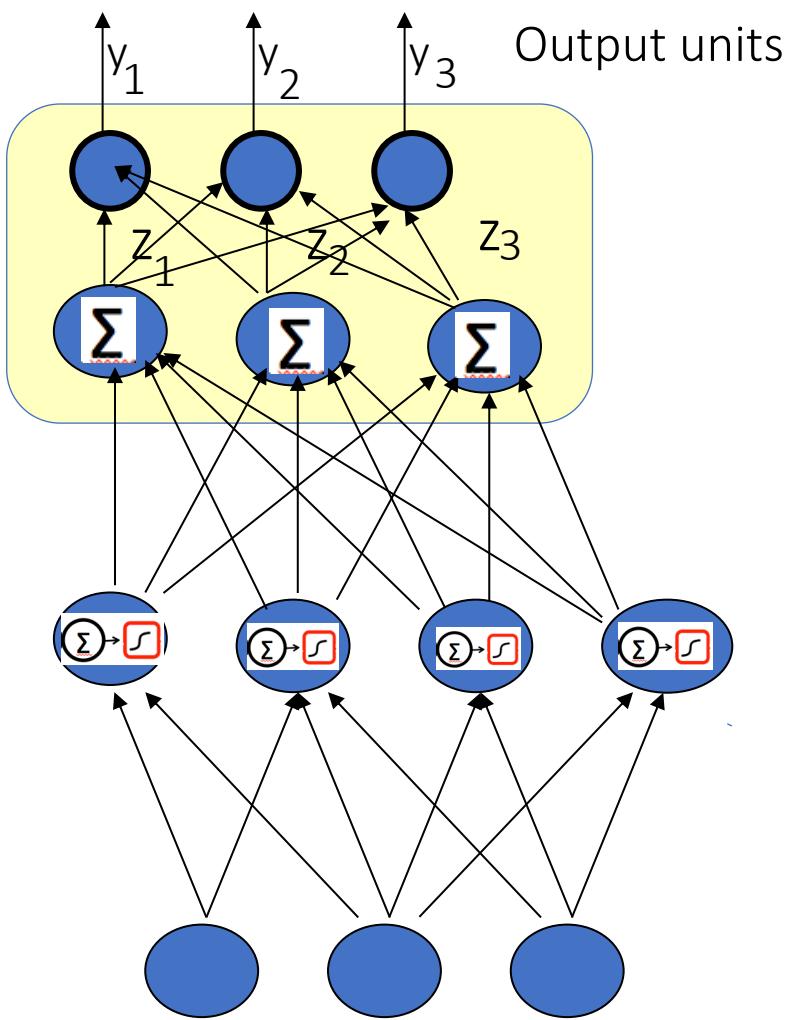
$$\begin{aligned} f_1(x) + f_2(x) \\ + \dots + f_k(x) = 1 \\ [0 \leq f_i(x) \leq 1] \end{aligned}$$

$$Pr(G = k | X = x) = Pr(Y_k = 1 | X = x)$$

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

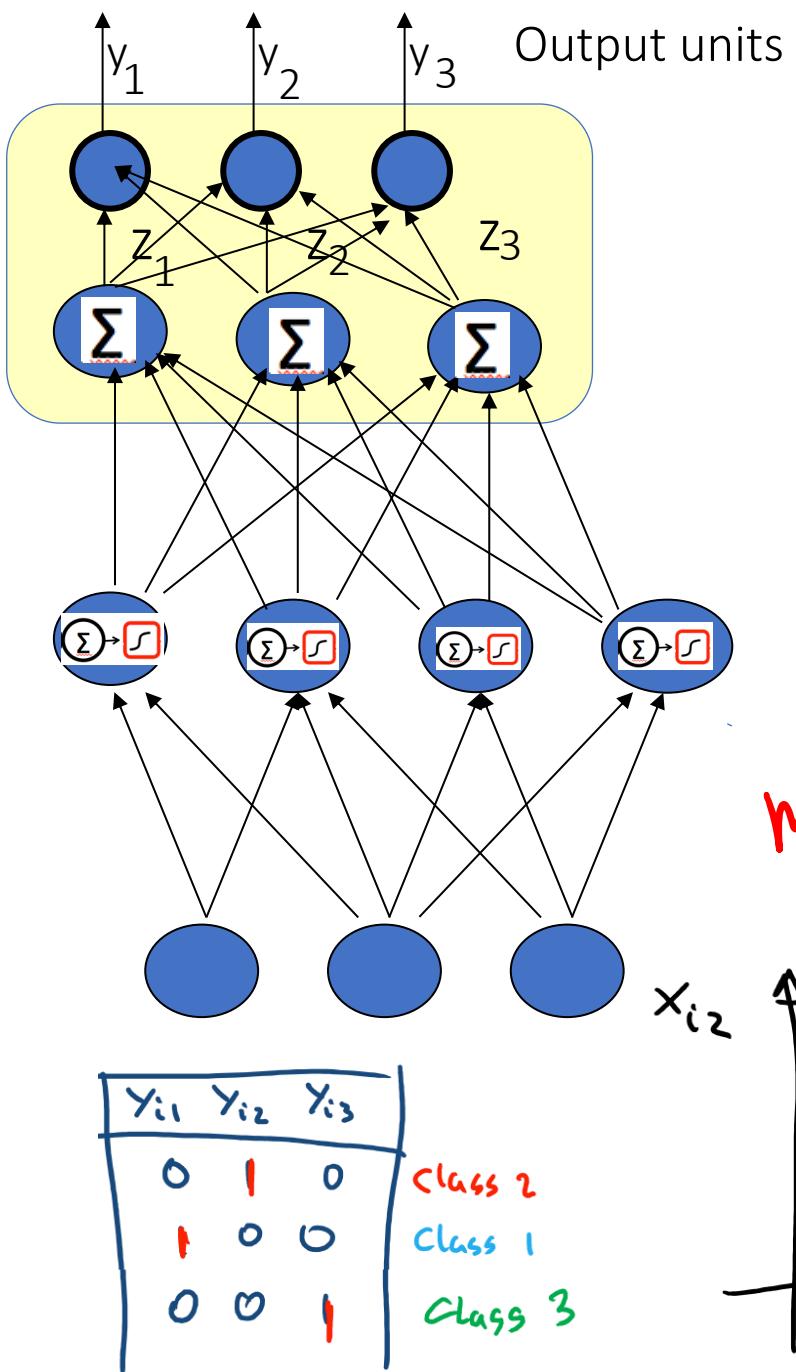
“Softmax” function: Normalizing function which converts each class output to a probability.

$$\frac{\partial y_i}{\partial z_i} = y_i (1 - y_i)$$



When for multi-class classification  
(last output layer: softmax layer)

last layer is softmax output layer →  
a Multinoulli logistic regression unit

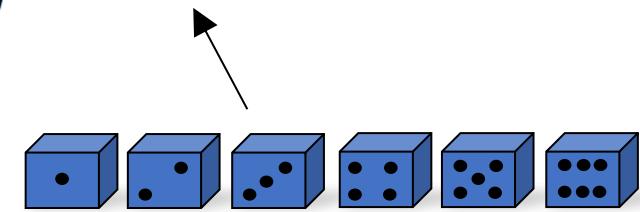
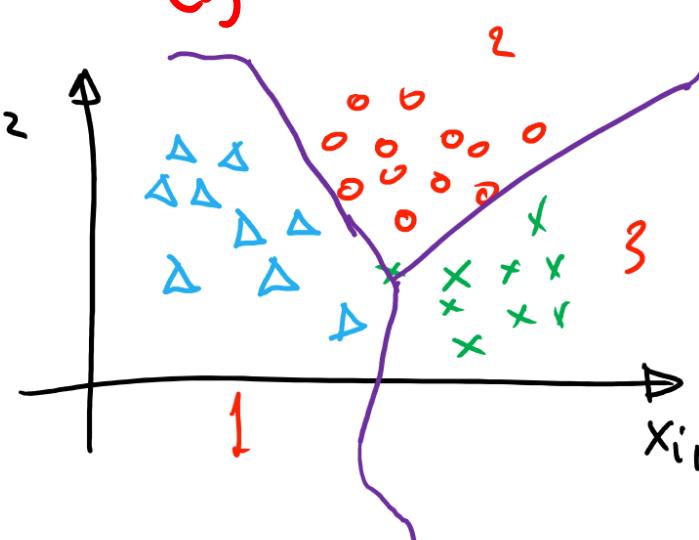


When for multi-class classification  
(last output layer: softmax layer)

last layer is softmax output layer →  
a Multinoulli logistic regression unit

$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}} = P(y_i = 1 | x)$$

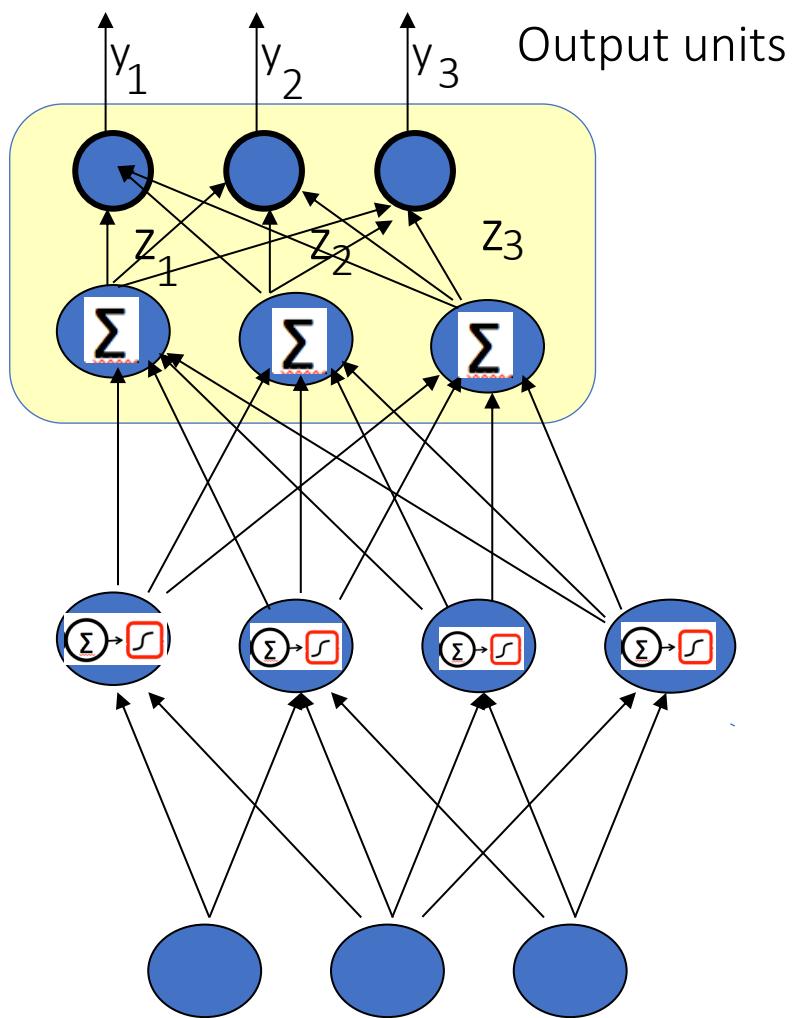
multiclass logistic regression



$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$y_{\text{true}} \quad \hat{y}$$

"0" for all  
except  
true class



When for multi-class classification  
(last output layer: softmax layer)

last layer is softmax output layer →  
a Multinoulli logistic regression unit

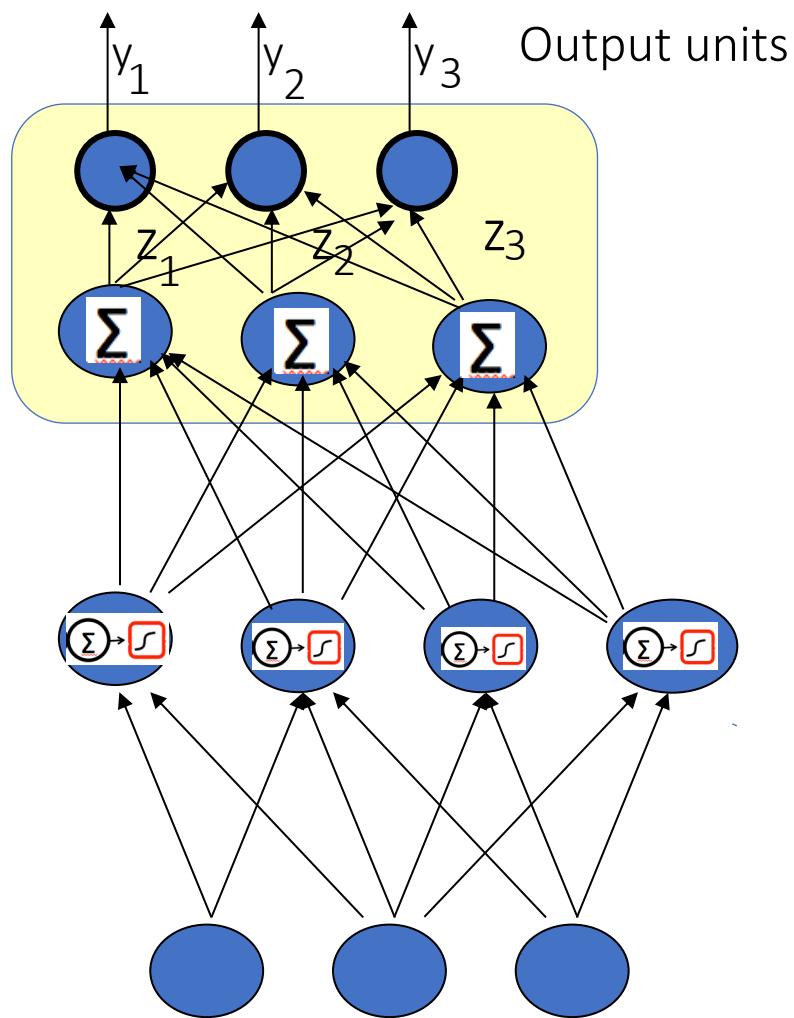
$$E_w(\hat{y}, y) = \text{cross-}E = - \sum_{j=1 \dots K} y_j \ln \hat{y}_j$$

MLE / the negative log probability of the right answer / Cross entropy loss function :

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

"0" for all  
except  
true class



Output units

When for multi-class classification  
(last output layer: softmax layer)

last layer is softmax output layer →  
a Multinoulli logistic regression unit

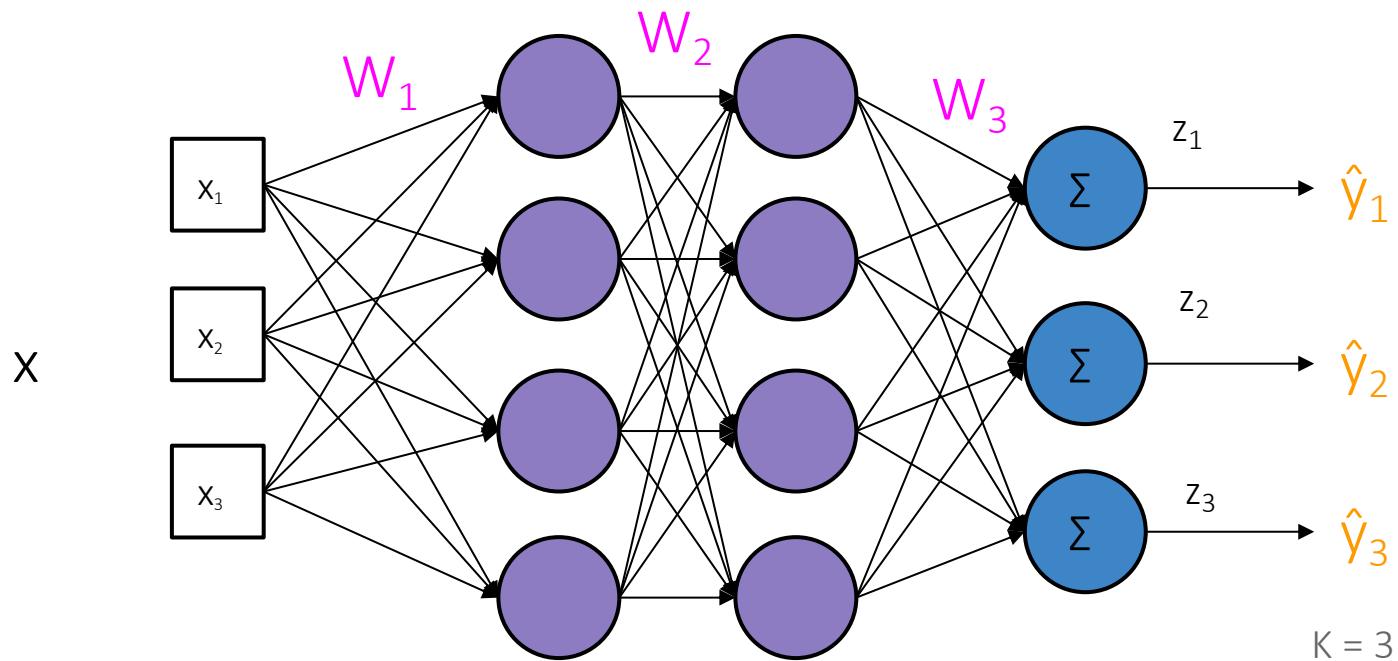
$$E_w(\hat{y}, y) = \text{cross-}E = - \sum_{j=1 \dots K} y_j \ln \hat{y}_j$$

MLE / the negative log probability of the right answer / Cross entropy loss function :

$$\frac{\partial E}{\partial z_i} = \sum_{j=1 \dots K} \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_i} = \hat{y}_i - y_{true,i}$$

# Summary Recap: Multi-Class Classification Loss

## Cross Entropy Loss



$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0.1 \\ 0.7 \\ 0.2 \end{pmatrix}$$

$y$                      $\hat{y}$

$$E = \text{loss} = - \sum_{j=1 \dots K} y_j \log \hat{y}_j$$

“0” for all except true class

$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}} = P(y_i = 1 | x)$$

“Softmax” function.  
Normalizing function which  
converts each class output to  
a probability.

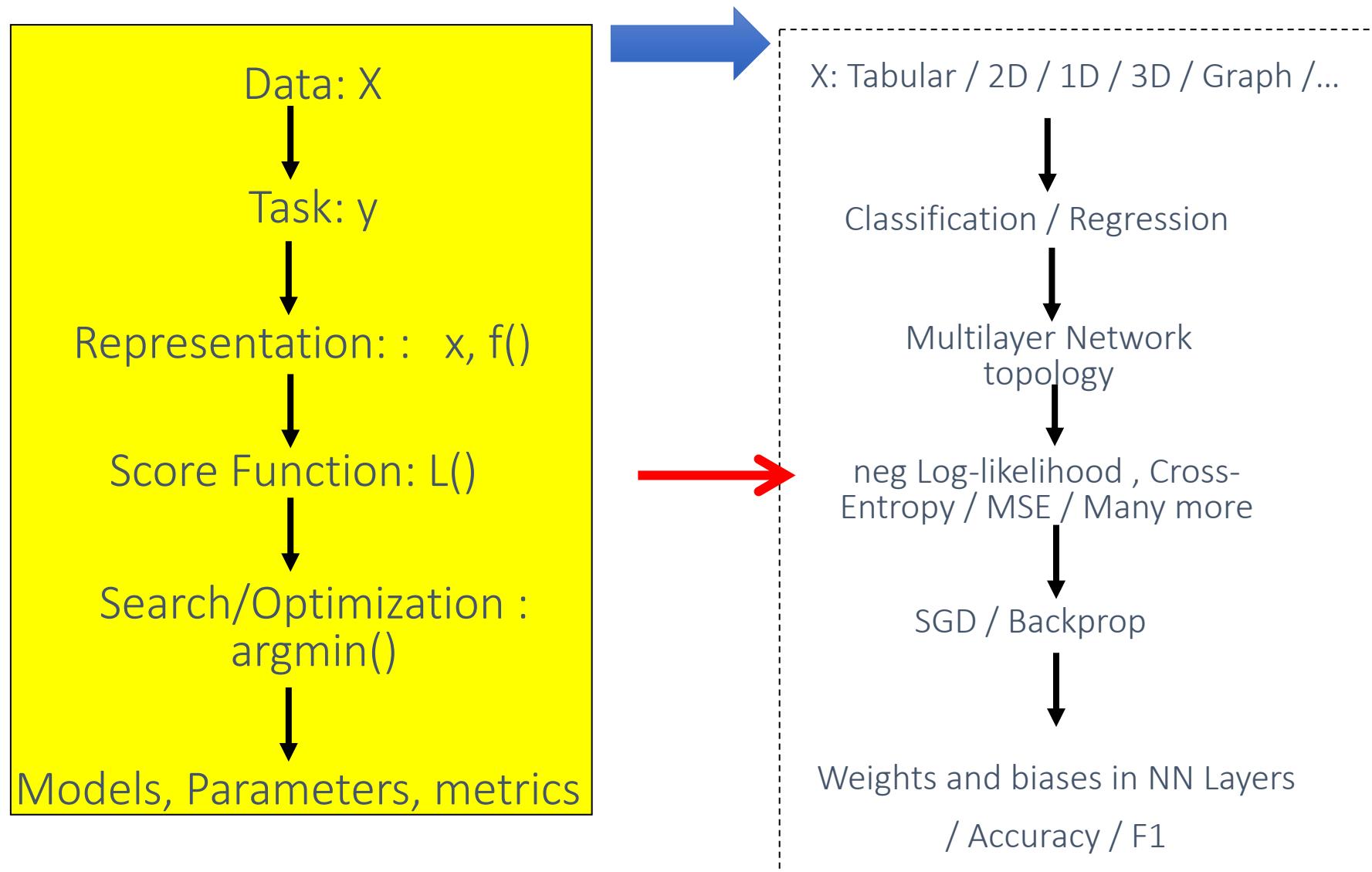
Logistic: a special case of softmax for two classes

{H, T}

$$y_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_0}} = \frac{1}{1 + e^{-(z_1 - z_0)}}$$

- So the logistic binary case is just a special case that avoids using redundant parameters:
  - Adding the same constant to both  $z_1$  and  $z_0$  has no effect.
  - The over-parameterization of the softmax is because the probabilities must add to 1.

# Today: Basic Neural Network Models



# Thank You



# UVA CS 4774: Machine Learning

## Lecture 12: Neural Network (NN) and More: BackProp

Dr. Yanjun Qi

University of Virginia

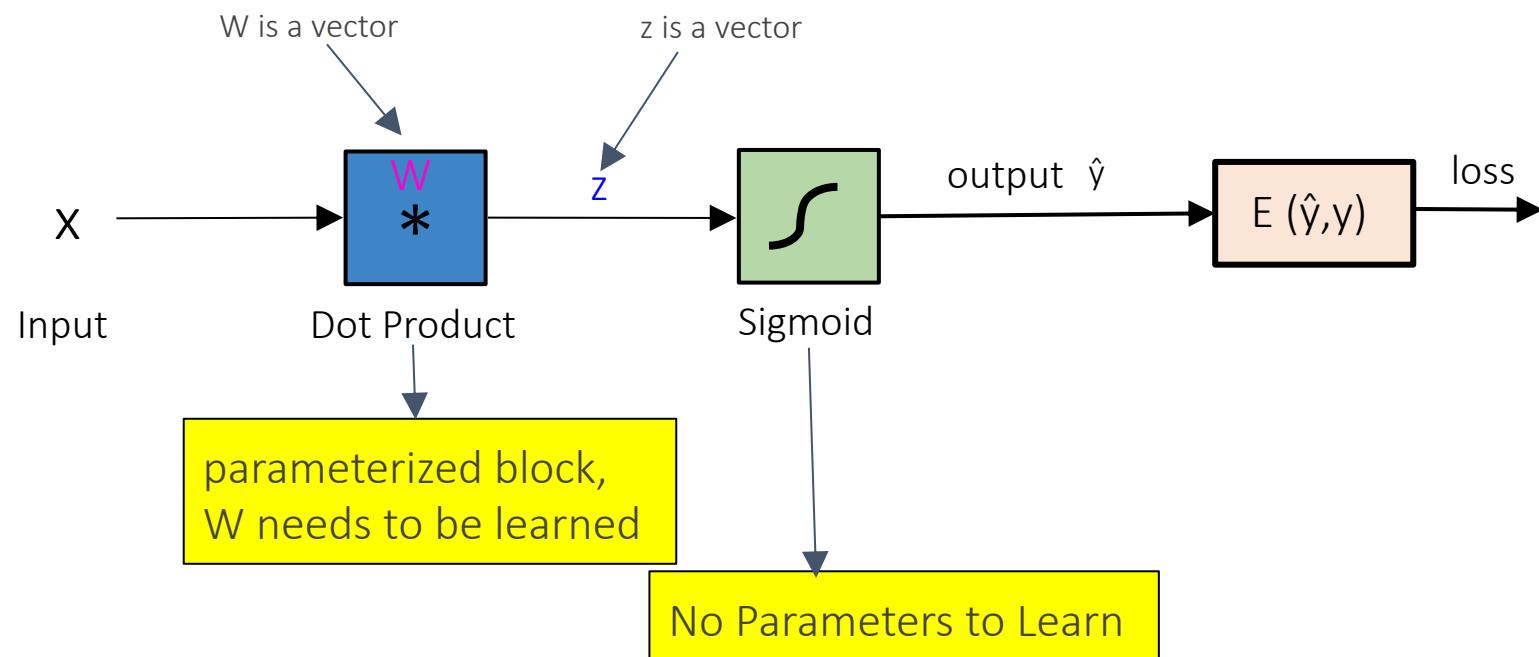
Department of Computer Science

Module III

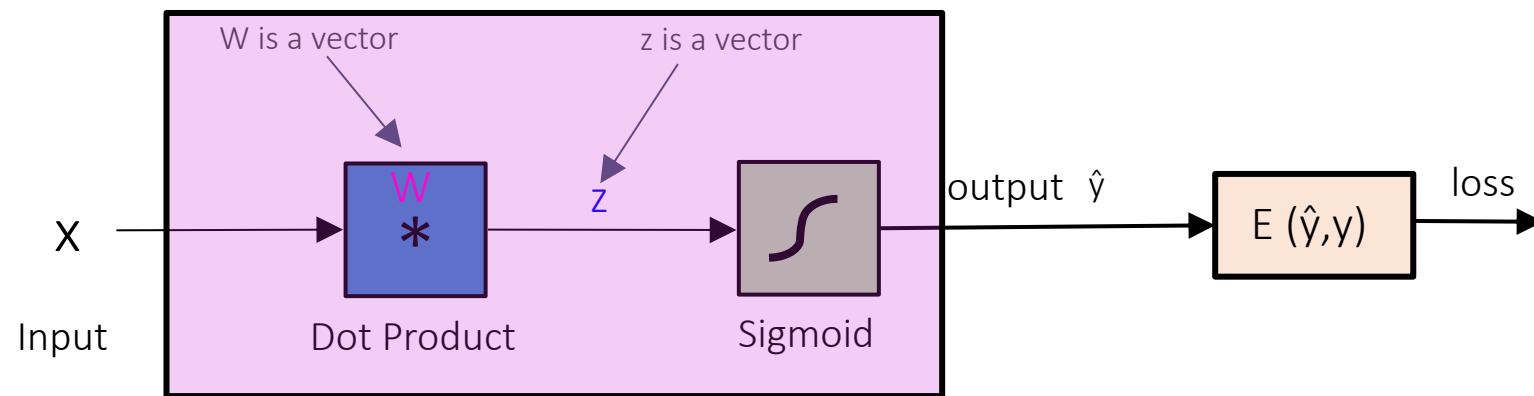
# Roadmap: DNN Basics

- Basics of Neural Network (NN)
  - single neuron, e.g. logistic regression unit
  - multilayer perceptron (MLP)
  - various loss function
    - E.g., when for multi-class classification, softmax layer
- • training NN with backprop algorithm
  - A few advanced tricks

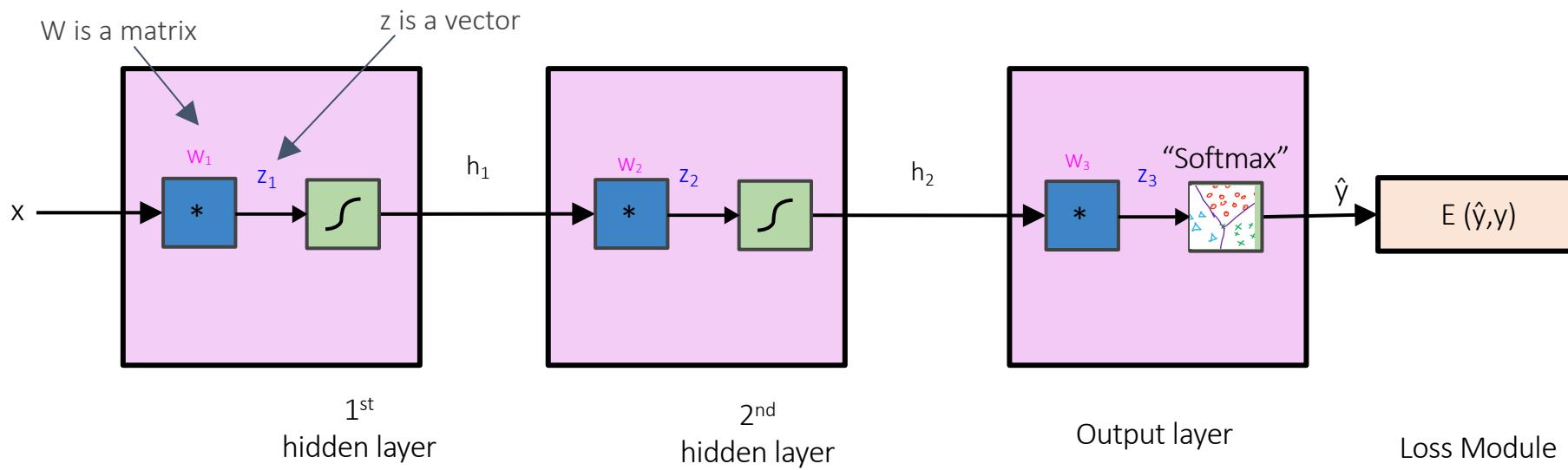
# e.g., “Block View” of Logistic Regression

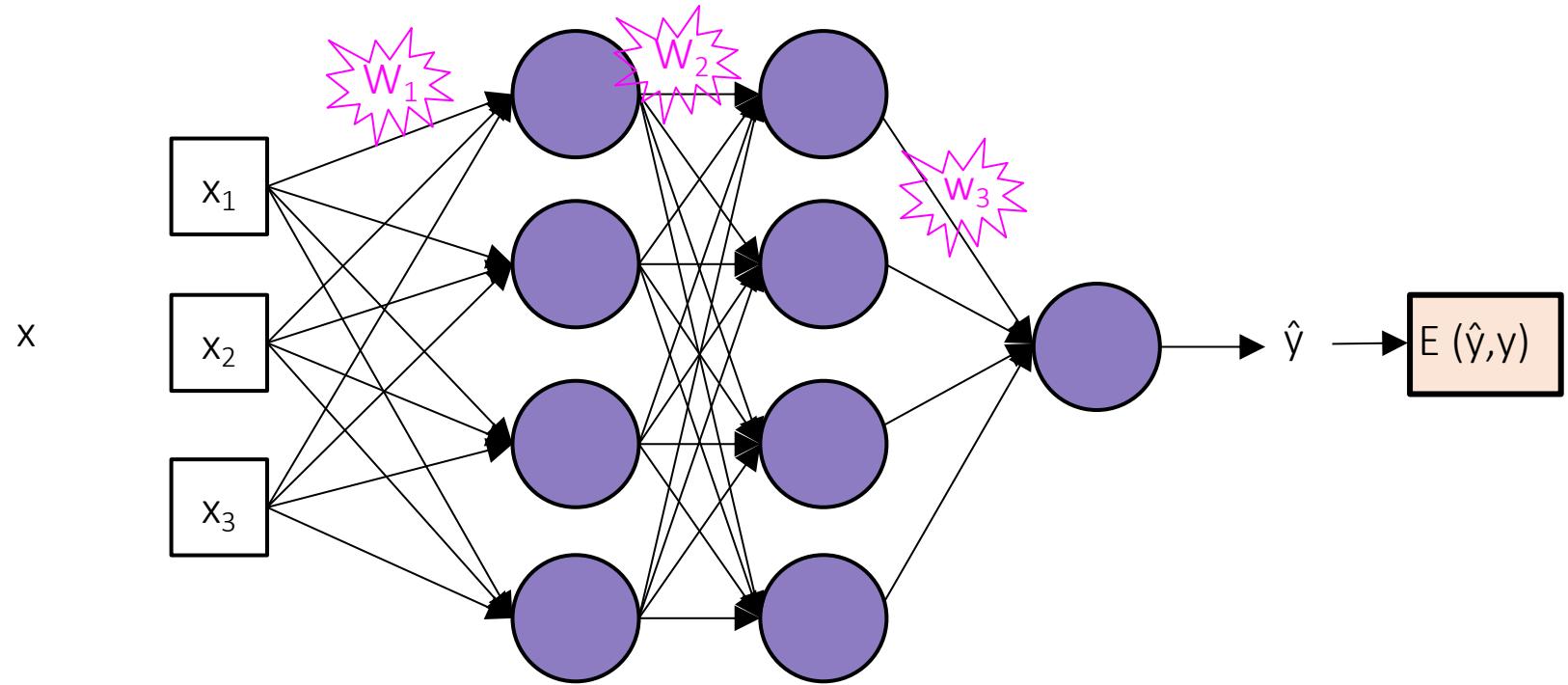


# e.g., “Block View” of Logistic Regression



# e.g., “Block View” of multi-class NN





## Review: Stochastic GD →

- For LR: linear regression, We have the following descent rule:

$$\theta_j^{t+1} = \theta_j^t - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \Big|_t$$

- → For neural network, we have the delta rule

$$\Delta w = -\eta \frac{\partial E}{\partial W^t} \quad w = \{w^1, w^2, \dots, w^T\}$$

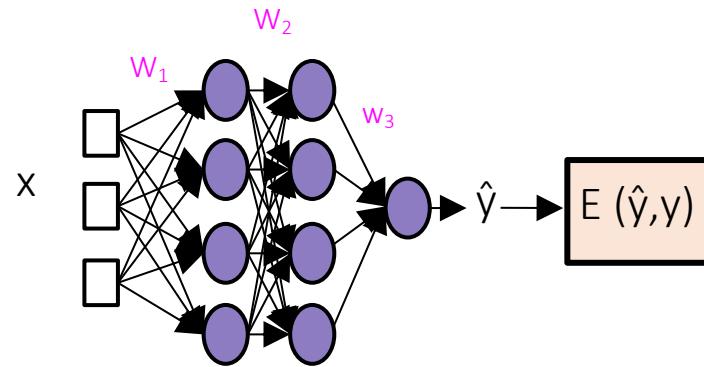
$$W^{t+1} = W^t - \eta \frac{\partial E}{\partial W^t} = W^t + \Delta w$$

# Training Neural Networks by Backpropagation - to jointly optimize all parameters

How do we learn the optimal weights  $W_L$  for our task??

- Stochastic Gradient descent:

$$W_L^{t+1} = W_L^t - \eta \frac{\partial E_x}{\partial W_L}$$



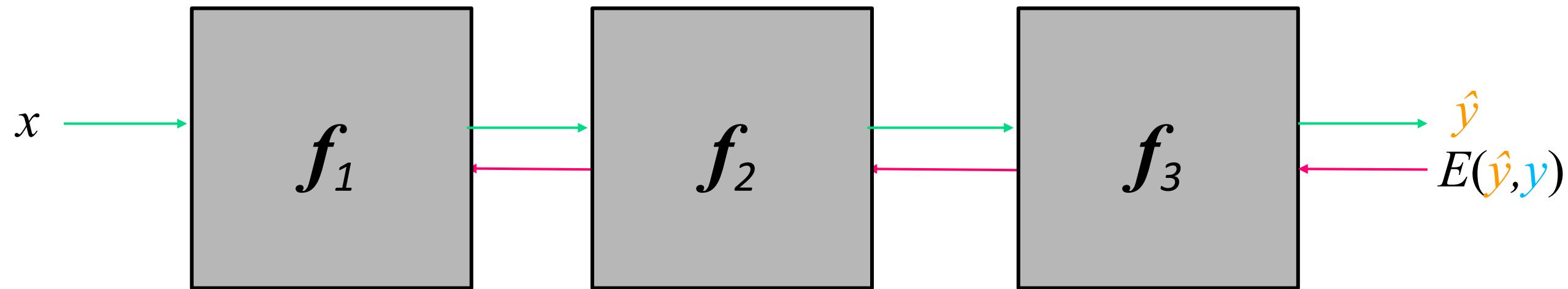
But how do we get gradients of lower layers?

- Backpropagation!
  - Repeated application of chain rule of calculus
  - Locally minimize the objective
  - Requires all “blocks” of the network to be differentiable

# Backpropagation

- 1. Initialize network with random weights
- 2. For all training examples:
  - **Forward:** Feed feed inputs to network layer by layer, and calculate output of each layer (**from input layer to until the final layer** (error function))
  - **Backward:** For all layers (starting with the output layer, back to input layer):
    - Propagate local gradients layer by layer from final layer, until back to input layer to calculate each layer's gradient  $\frac{\partial E}{\partial W_l^t}$  → Need to calculate these!
    - Adapt weights in current layer  $W_l^{t+1} = W_l^t - \eta \frac{\partial E}{\partial W_l^t}$

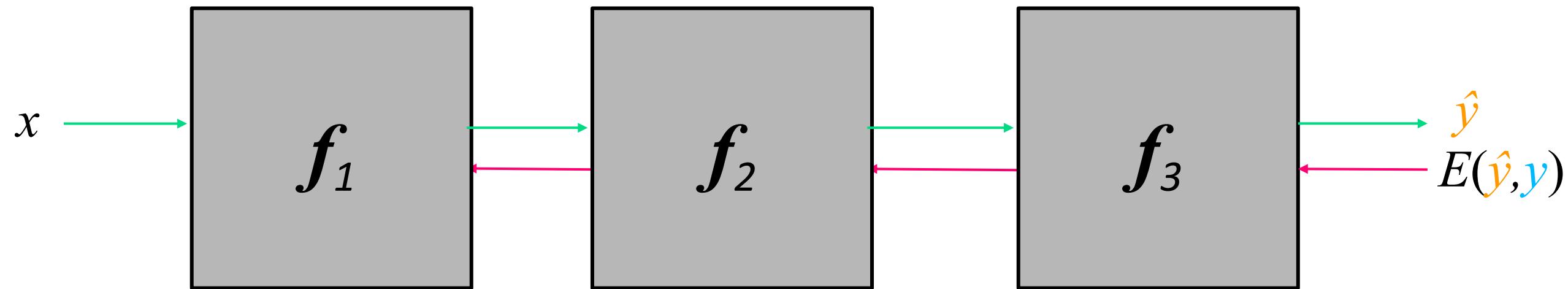
# Layers of Differentiable Parameterized Functions (with nonlinearities)



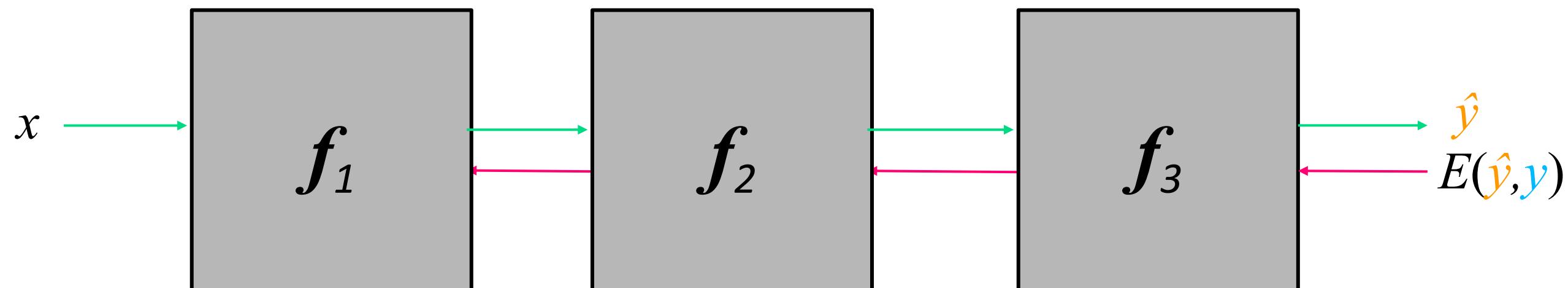
# Layers of Differentiable Parameterized Functions (with nonlinearities)

**Forward:** Feed feed inputs to network layer by layer, and calculate output of each layer (from input layer to until the final layer (error function))

Need to calculate these!



# Layers of Differentiable Parameterized Functions (with nonlinearities)



**Backward:** For all layers (starting with the output layer, back to input layer):

Propagate local gradients layer by layer from final layer, until back to input layer to calculate each layer's gradient

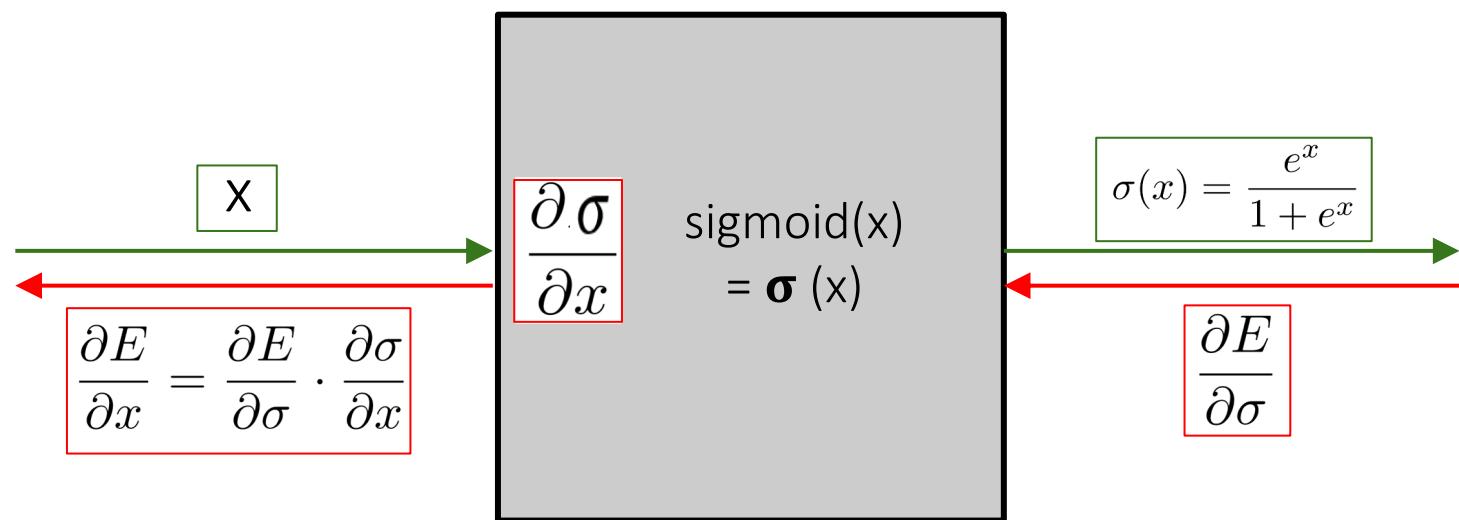
$$\frac{\partial E}{\partial W_l^t}$$

Need to calculate these!

Adapt weights in current layer

$$W_l^{t+1} = W_l^t - \eta \frac{\partial E}{\partial W_l^t}$$

# Example: Sigmoid Block



$$= \frac{\partial E}{\partial \sigma} \cdot \sigma(1 - \sigma)$$

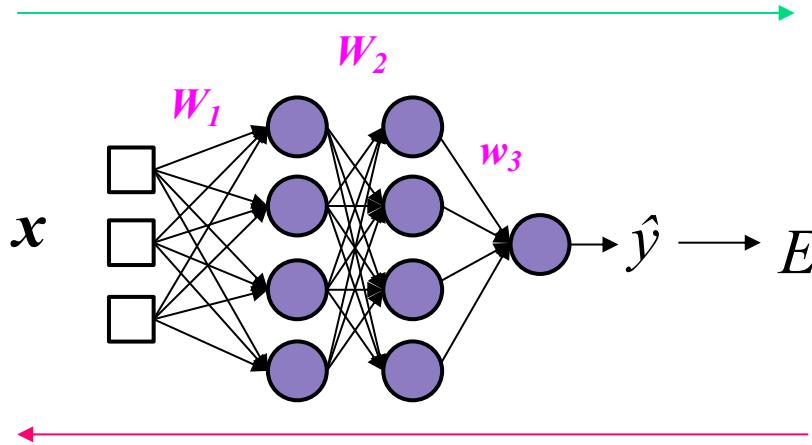
$$\frac{\partial \sigma}{\partial x} = \sigma(1 - \sigma)$$

$\text{argmin}_w \{ f_4(f_3(f_2(f_1(\cdot)))) \}$ 

	Input	Output	Local Gradients = $\frac{\partial \text{Output}}{\partial \text{Input}}$
$f_1$	$x_1 w_1 + x_2 w_3 + b_1$ $z_2 = x_1 w_2 + x_2 w_4 + b_2$	$z_1, z_2$	$\frac{\partial z_1}{\partial x_1} = w_1$
$f_2$	$h_1 = \frac{\exp(z_1)}{1 + \exp(z_1)}$ $h_2 = \frac{\exp(z_2)}{1 + \exp(z_2)}$	$h_1, h_2$	$\frac{\partial h_1}{\partial z_1} = h_1(1-h_1)$
$f_3$	$\hat{y} = h_1 w_5 + h_2 w_6 + b_3$	$\hat{y}$	$\frac{\partial \hat{y}}{\partial h_1} = w_5$
$f_4$	$E = (y - \hat{y})^2$	loss $E$	$\frac{\partial E}{\partial \hat{y}} = -2(y - \hat{y})$

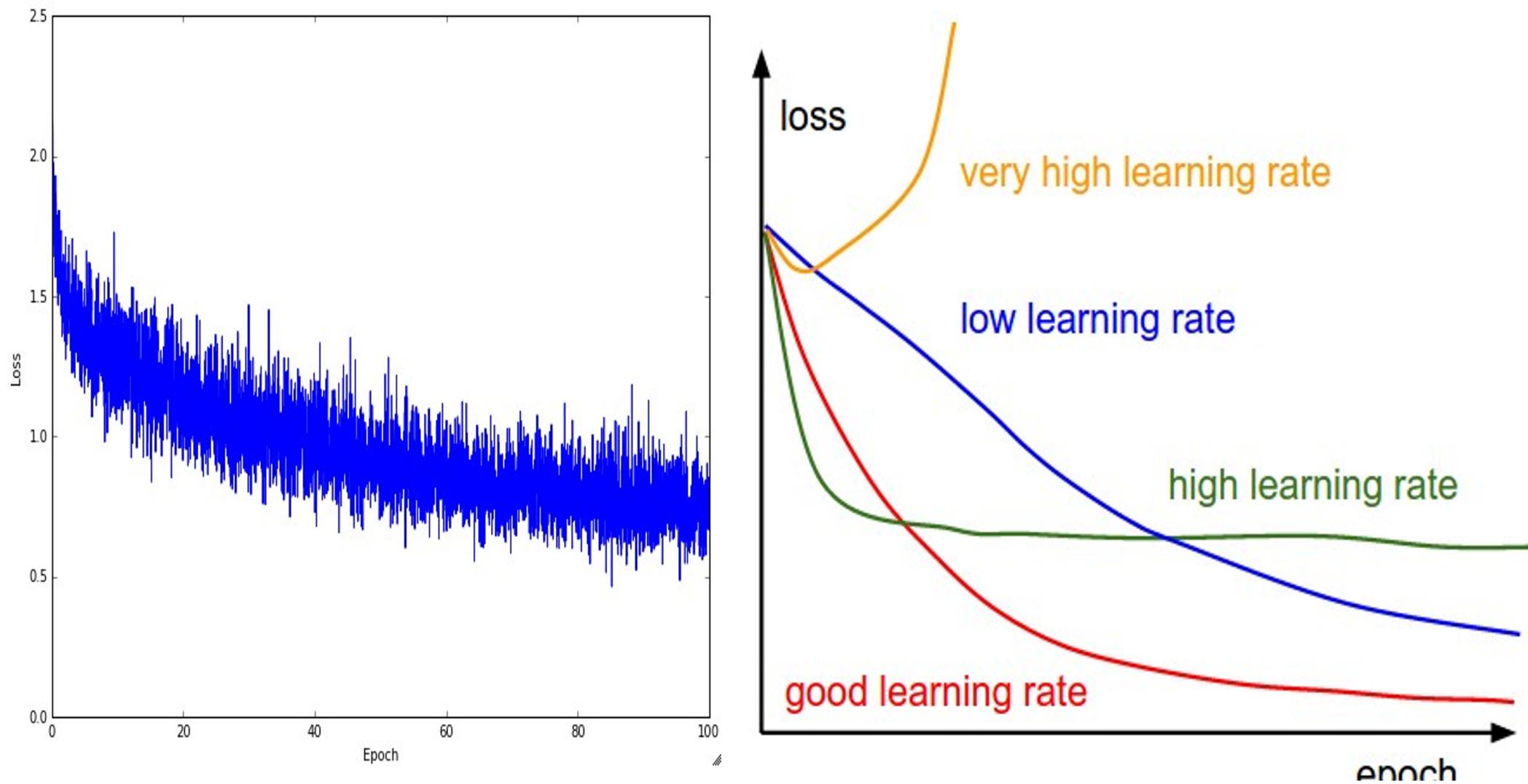
$$\begin{aligned}
 \frac{\partial E}{\partial w_1} &= \frac{\partial E}{\partial h_1} = \frac{\partial f_4}{\partial f_3} \frac{\partial f_3}{\partial f_2} \frac{\partial f_2}{\partial f_1} \frac{\partial f_1}{\partial w_1} \\
 &= -2(y - \hat{y}) \frac{\partial (h_1 w_5 + h_2 w_6 + b_3)}{\partial w_1} \\
 &= -2(y - \hat{y}) (w_5 \frac{\partial h_1}{\partial w_1} + w_6 \frac{\partial h_2}{\partial w_1}) \\
 &= -2(y - \hat{y}) w_5 \frac{\partial h_1}{\partial w_1} = -2(y - \hat{y}) w_5 \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_1} \\
 &= \frac{-2(y - \hat{y})}{f_4 \text{ local}} \frac{w_5}{f_3 \text{ local}} \frac{h_1(1-h_1)}{f_2 \text{ local}} \frac{x_1}{f_1 \text{ local}} \frac{\frac{\partial f_1}{\partial w_1}}{\frac{\partial z_1}{\partial w_1}}
 \end{aligned}$$

# BackProp in Practice: Mini-batch SGD

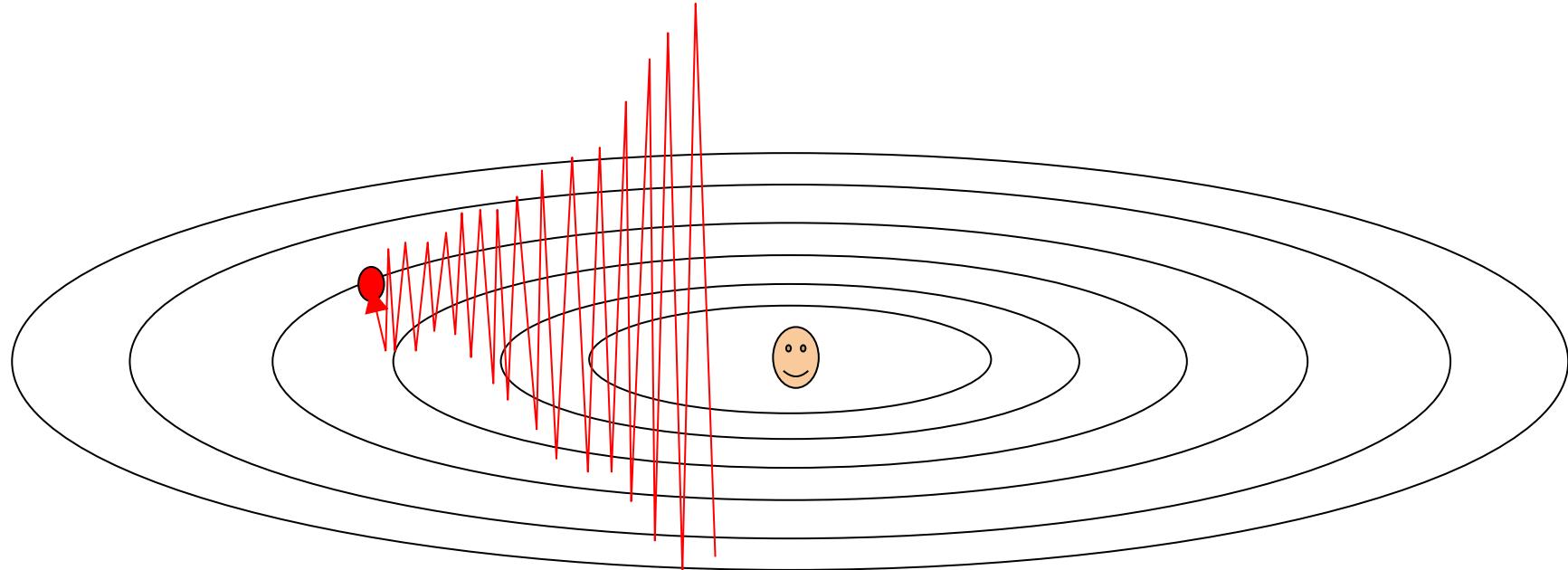


1. Initialize weights
2. For each batch of input samples  $Sx$  :
  - a. Run the network “Forward” on  $S$  to compute outputs and loss
  - b. Run the network “Backward” using outputs and loss to compute gradients
  - c. Update weights using SGD (or a similar method)
2. Repeat step 2 until loss convergence

# Monitor and visualize the loss curve



# Gradient Magnitudes:



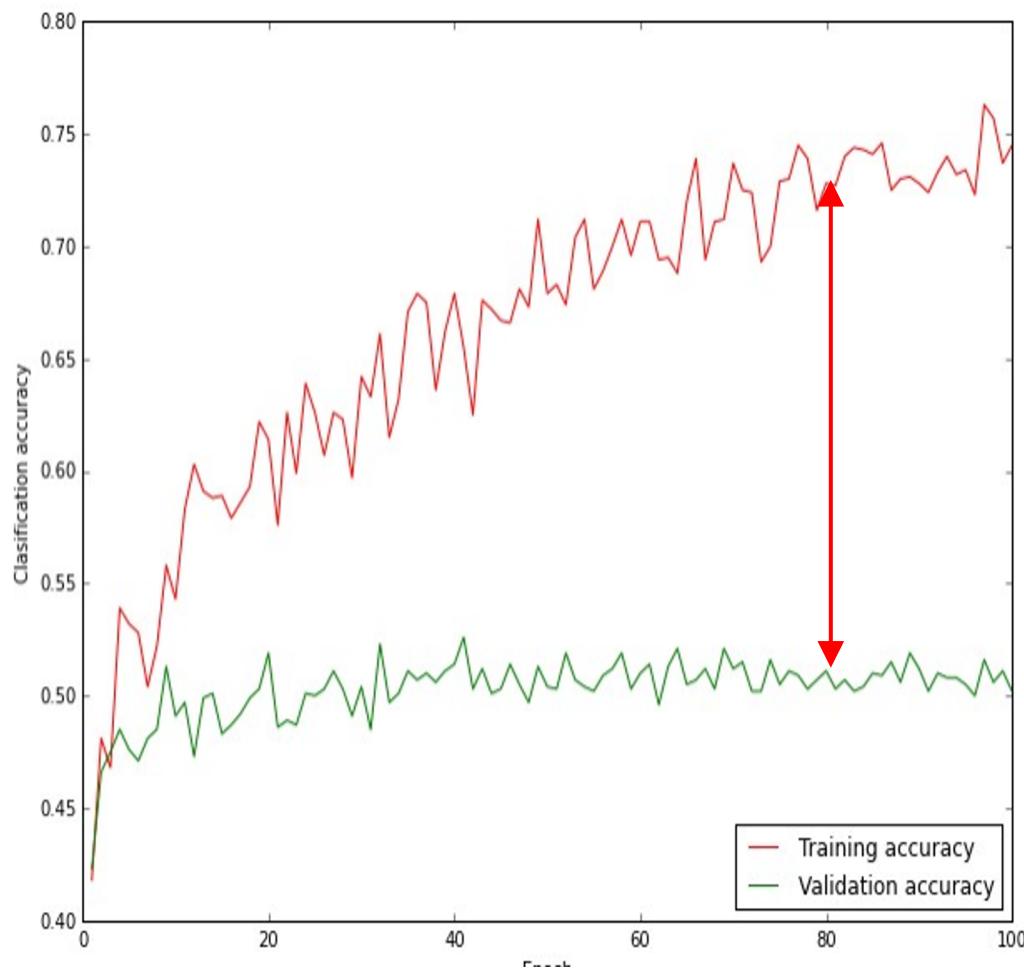
Gradients too big → divergence

Gradients too small → slow convergence

Divergence is much worse!

Many great tools, e.g., Adam  
<https://arxiv.org/abs/1609.04747>

# Monitor and visualize the train / validation loss / accuracy: Bias Variance Tradeoff



big gap = overfitting  
=> increase regularization strength?

no gap, e.g. underfitting / both bad  
=> increase model capacity?

## Other things to plot and check:

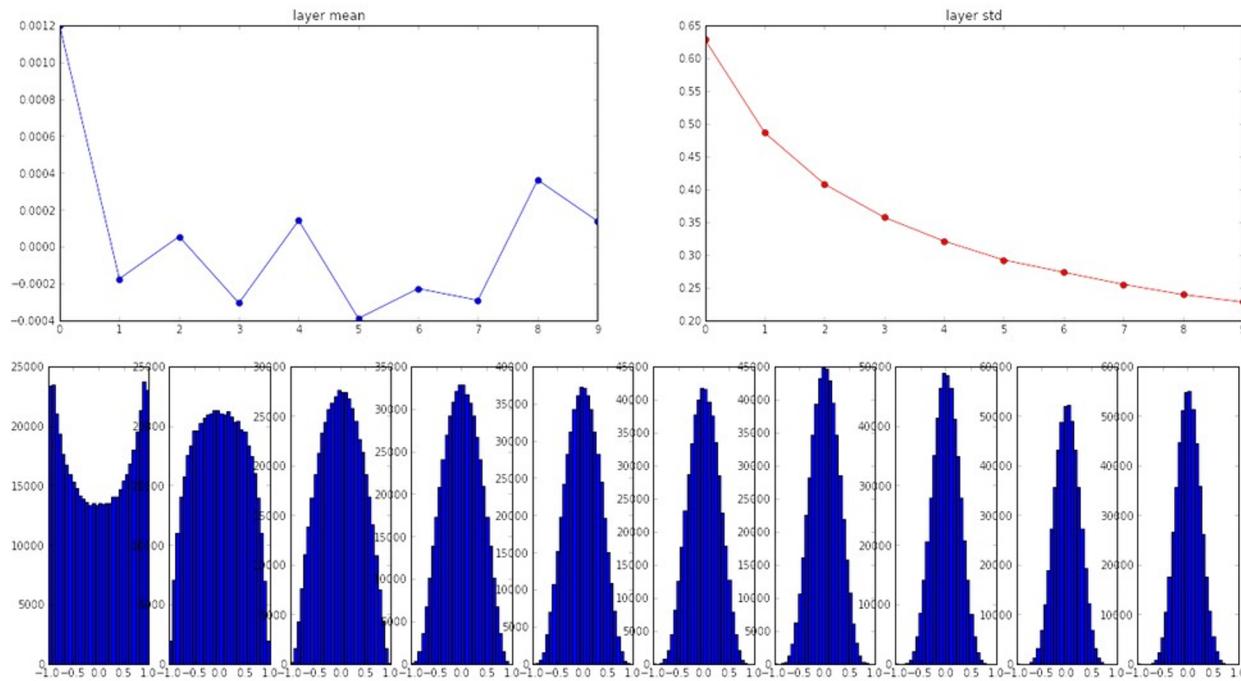
- Per-layer activations:
  - Magnitude, center (mean or median), breadth (sdev or quartiles)
  - Spatial/feature-rank variations
- Gradients
  - Magnitude, center (mean or median), breadth (sdev or quartiles)
  - Spatial/feature-rank variations
- Learning trajectories
  - Plot parameter values in a low-dimensional space

# Weight Initialization

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

“Xavier initialization”  
[Glorot et al., 2010]



Reasonable initialization.  
(Mathematical derivation  
assumes linear activations)

# Batch Normalization: implicit regularization

[Ioffe and Szegedy, 2015]

Normalize:

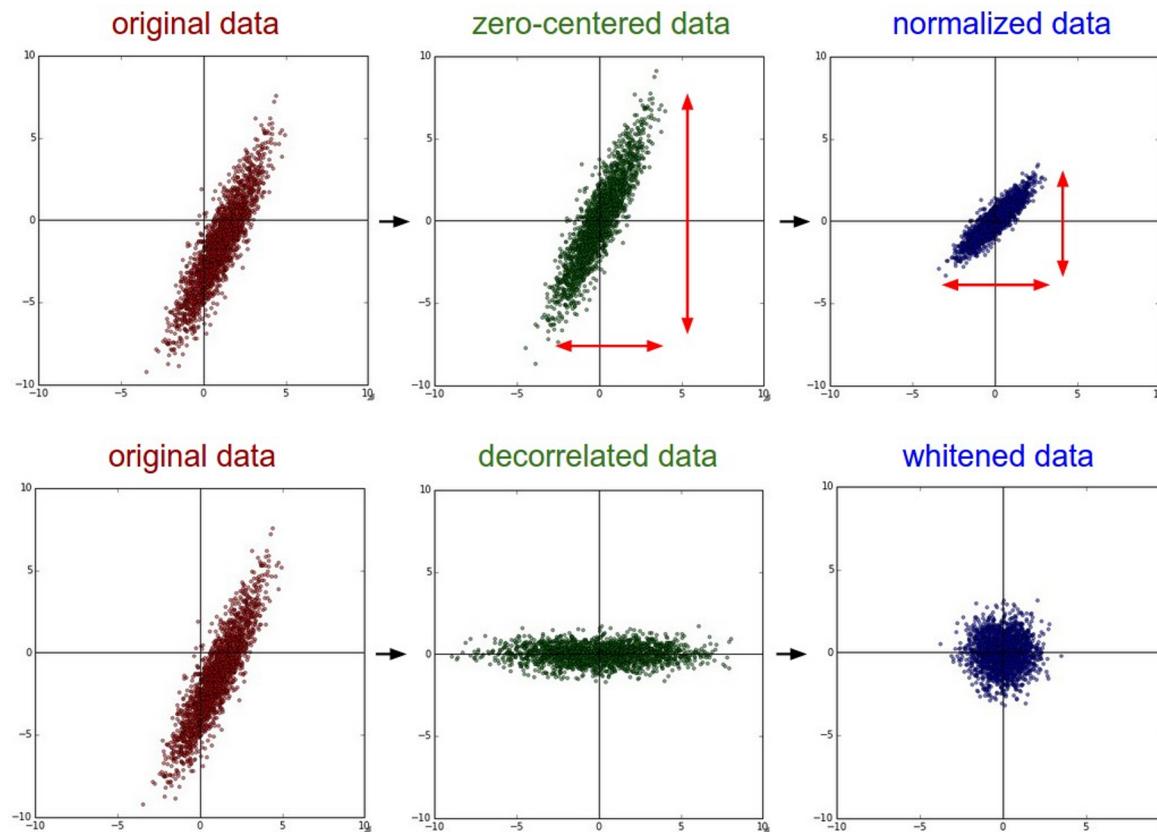
$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

# Data Preprocessing



# Hyperparameters to play with:

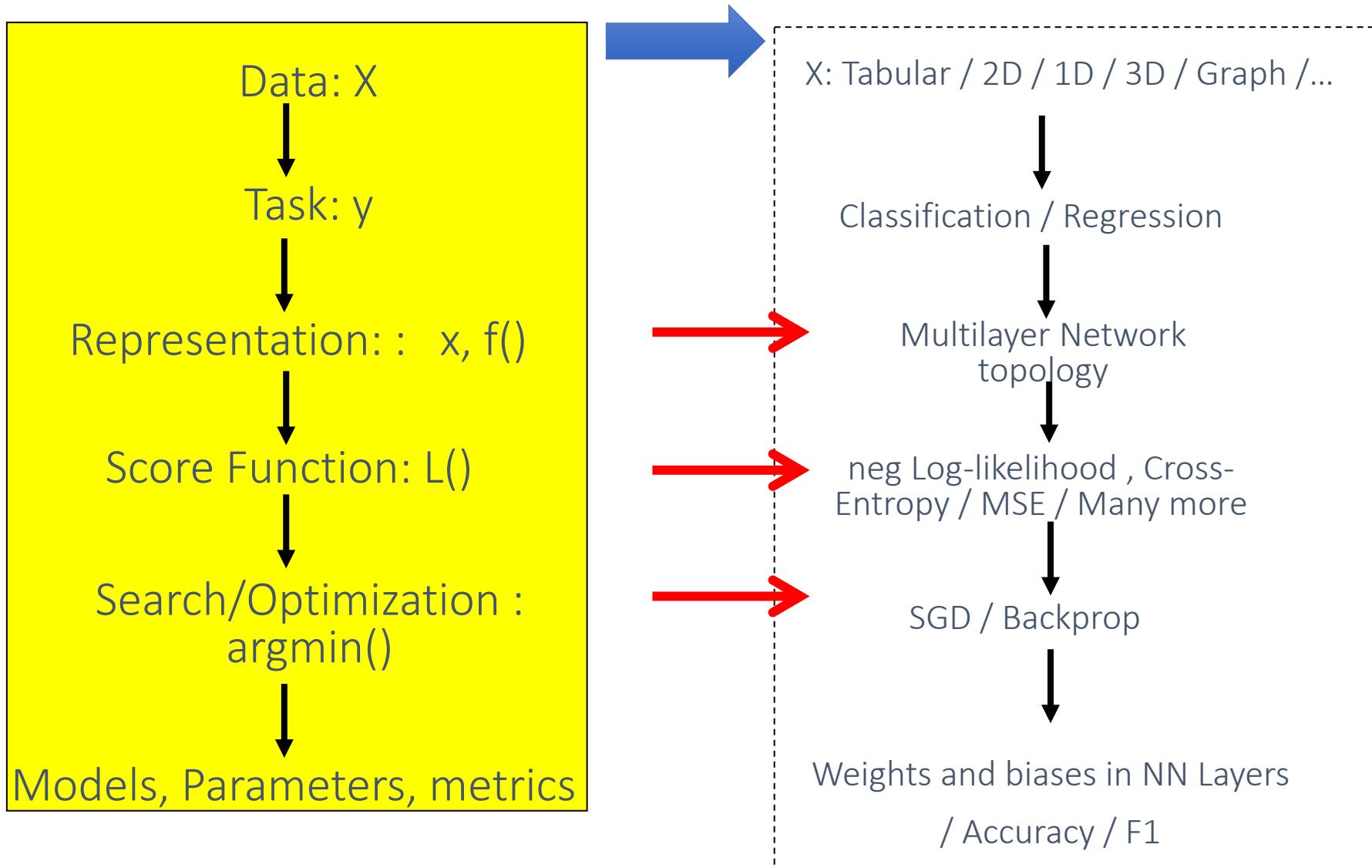
- network architecture
- learning rate, decay schedule, update type
- regularization (L2/Dropout strength)

How to become a great neural  
networks practitioner  
→ Craft? / Talent? / Experience?

Your Friend: loss function



# Today: Basics of Neural Network Models



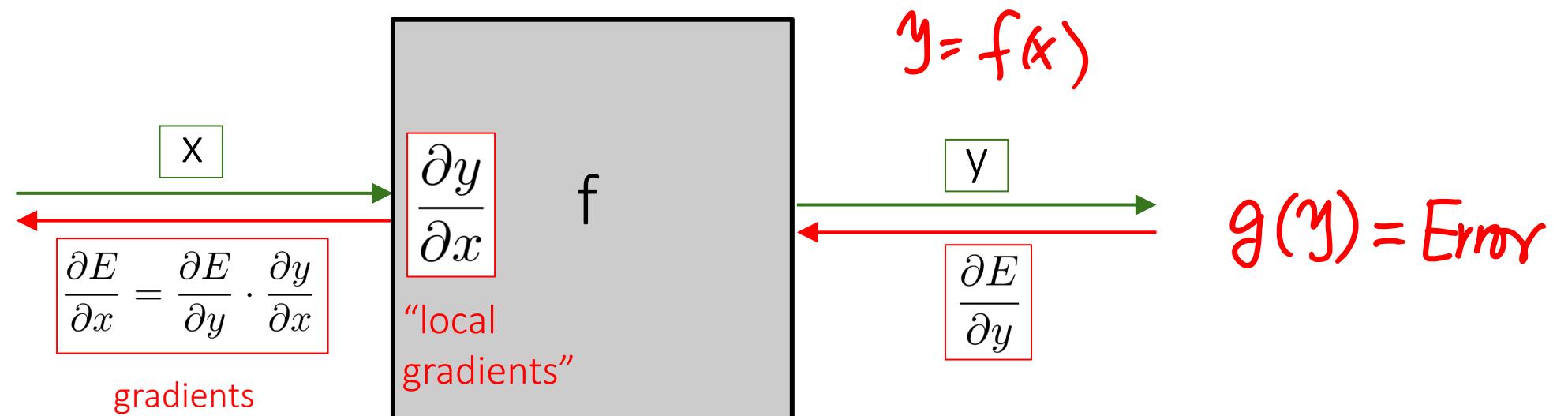
# Thank You



# References

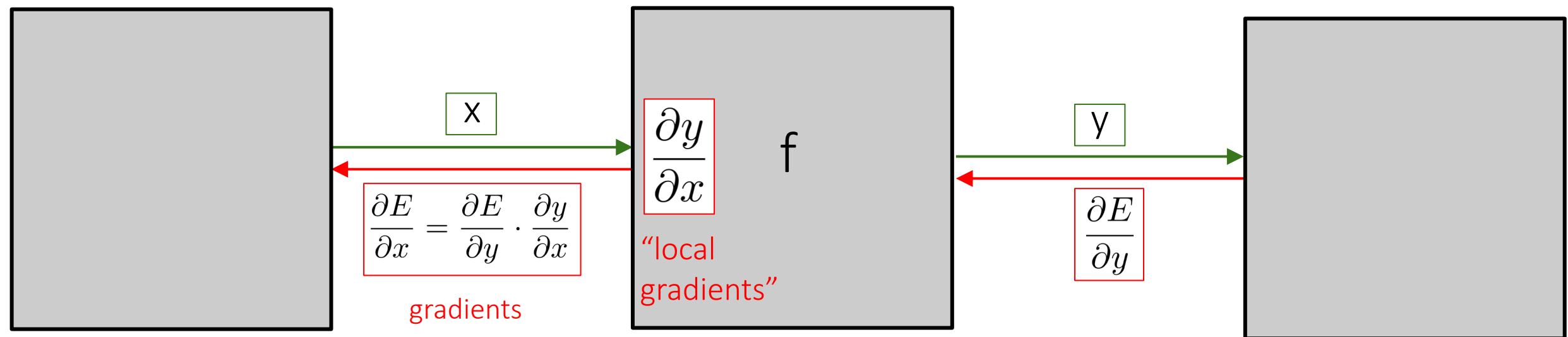
- ❑ Dr. Yann Lecun's deep learning tutorials
- ❑ Dr. Li Deng's ICML 2014 Deep Learning Tutorial
- ❑ Dr. Kai Yu's deep learning tutorial
- ❑ Dr. Rob Fergus' deep learning tutorial
- ❑ Prof. Nando de Freitas' slides
- ❑ Olivier Grisel's talk at Paris Data Geeks / Open World Forum
- ❑ Hastie, Trevor, et al. *The elements of statistical learning*. Vol. 2. No. 1. New York: Springer, 2009.
- ❑ Dr. Hung-yi Lee's CNN slides

# “Local-ness” of Backpropagation

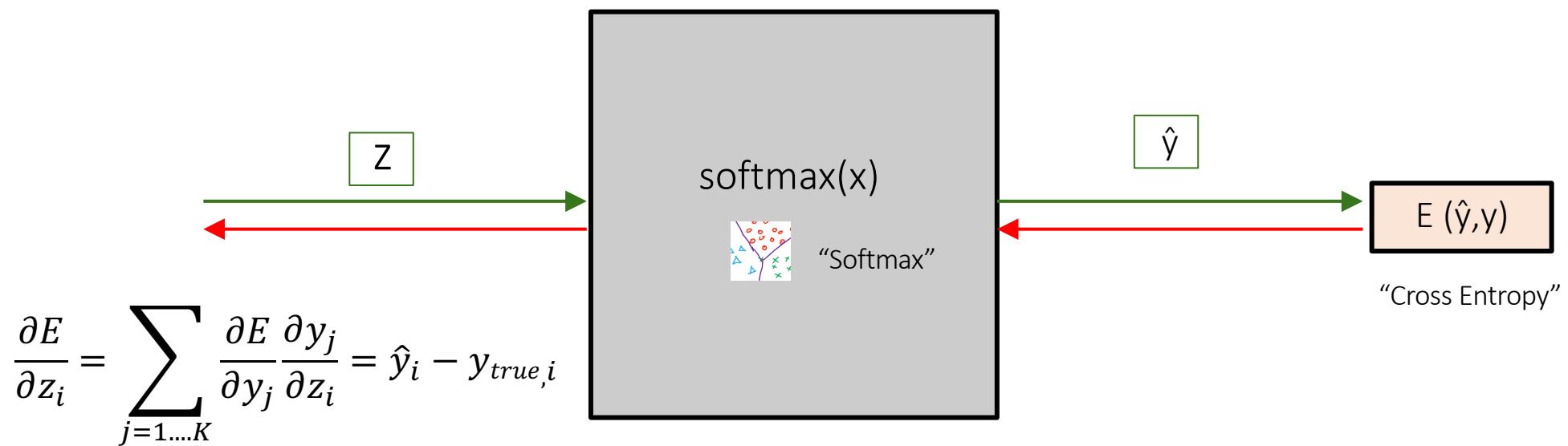


$$E = g(y) = g(f(x)) \Rightarrow \frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial x}$$

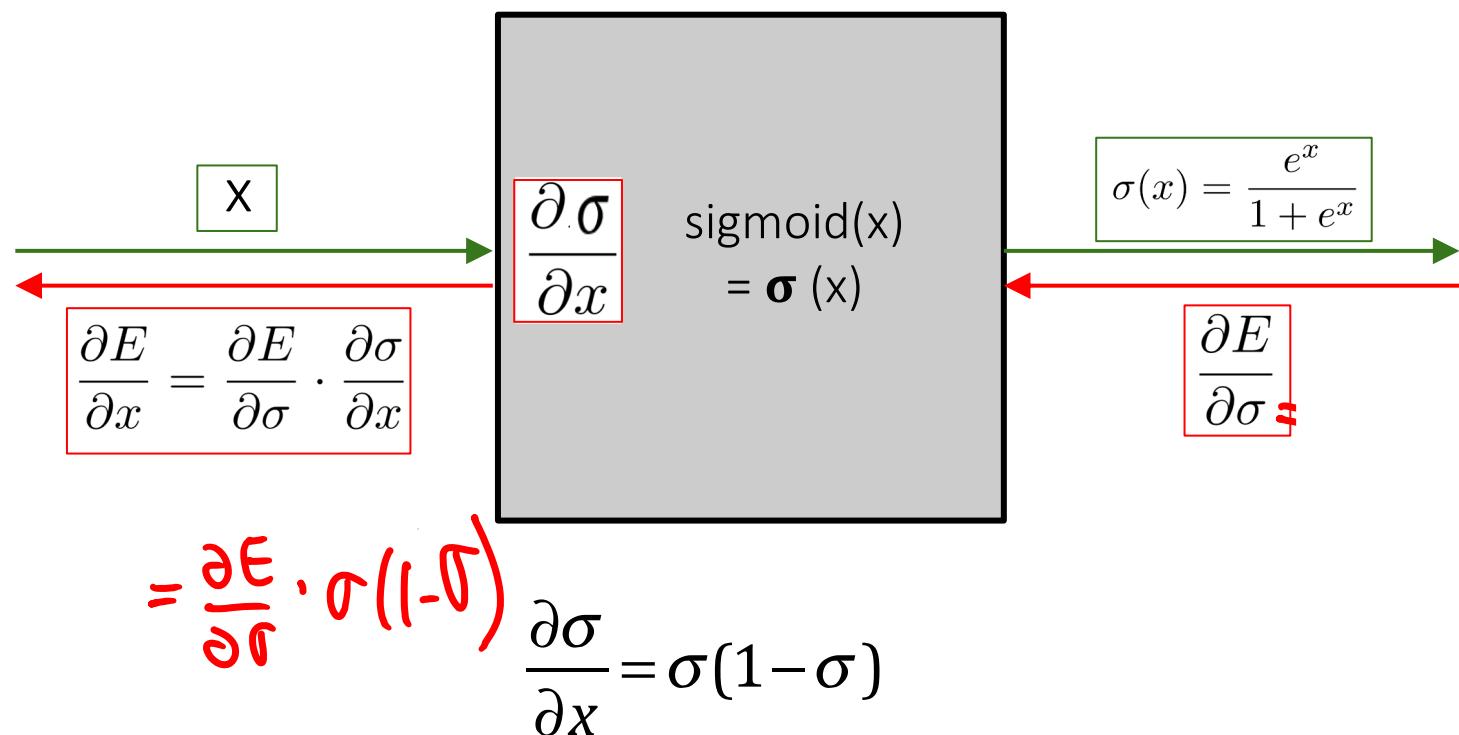
# “Local-ness” of Backpropagation



# Example: Softmax Block (right before loss layer)



# Example: Sigmoid Block



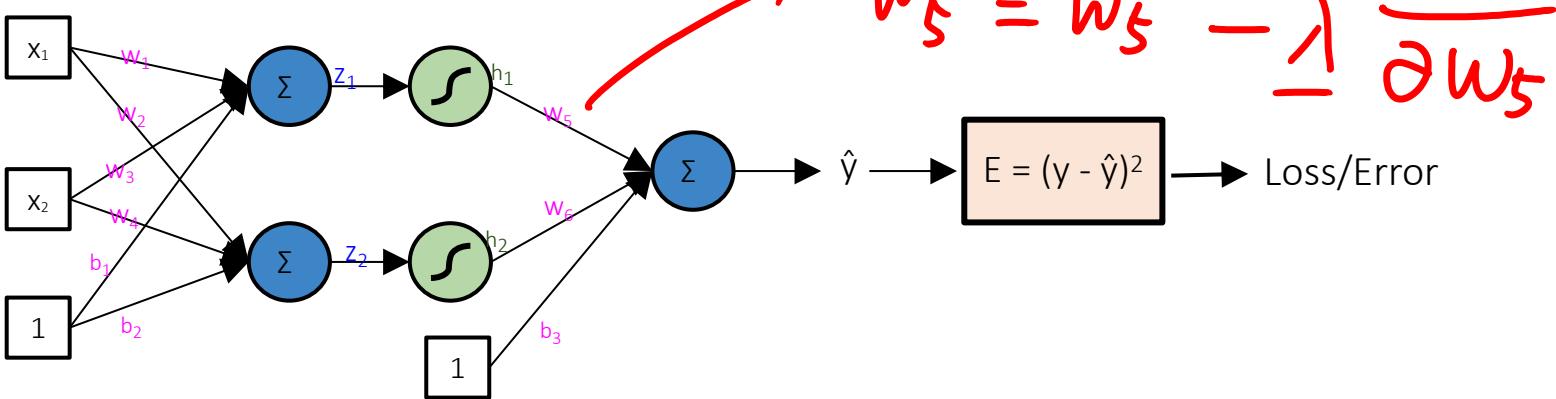
$$\text{argmin}_w \{ f_4(f_3(f_2(f_1(\cdot)))) \}$$

↓

	Input	Output	Local Gradients = $\frac{\partial \text{Output}}{\partial \text{Input}}$
$f_1$	$x_1 w_1 + x_2 w_3 + b_1$ $z_2 = x_1 w_2 + x_2 w_4 + b_2$	$z_1, z_2$	$\frac{\partial z_1}{\partial x_1} = w_1$
$f_2$	$h_1 = \frac{\exp(z_1)}{1 + \exp(z_1)}$ $h_2 = \frac{\exp(z_2)}{1 + \exp(z_2)}$	$z_1, z_2$	$\frac{\partial h_1}{\partial z_1} = h_1(1-h_1)$
$f_3$	$\hat{y} = h_1 w_5 + h_2 w_6 + b_3$	$w_5, h_1, h_2$	$\frac{\partial \hat{y}}{\partial h_1} = w_5$
$f_4$	$E = (y - \hat{y})^2$	$\hat{y}$	$\frac{\partial E}{\partial \hat{y}} = -2(y - \hat{y})$
		loss $E$	

$$\begin{aligned}
 \frac{\partial E}{\partial w_1} &= \frac{\partial E}{\partial \hat{y}} = \frac{\partial f_4}{\partial f_3} \frac{\partial f_3}{\partial f_2} \frac{\partial f_2}{\partial f_1} \frac{\partial f_1}{\partial w_1} \\
 &= -2(y - \hat{y}) \frac{\partial (h_1 w_5 + h_2 w_6 + b_3)}{\partial w_1} \\
 &= -2(y - \hat{y}) (w_5 \frac{\partial h_1}{\partial w_1} + w_6 \frac{\partial h_2}{\partial w_1}) \\
 &= -2(y - \hat{y}) w_5 \frac{\partial h_1}{\partial w_1} = -2(y - \hat{y}) w_5 \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_1} \\
 &= \frac{-2(y - \hat{y})}{f_4 \text{ local}} \frac{w_5}{f_3 \text{ local}} \frac{h_1(1-h_1)}{f_2 \text{ local}} \frac{x_1}{f_1 \text{ local}} \frac{\frac{\partial f_1}{\partial w_1}}{\frac{\partial z_1}{\partial w_1}}
 \end{aligned}$$

## Backpropagation Example



$f_1$	$z_1 = x_1 w_1 + x_2 w_3 + b_1$
	$z_2 = x_1 w_2 + x_2 w_4 + b_2$
<hr/>	
$f_2$	$h_1 = \frac{\exp(z_1)}{1 + \exp(z_1)}$
	$h_2 = \frac{\exp(z_2)}{1 + \exp(z_2)}$
<hr/>	
$f_3$	$\hat{y} = h_1 w_5 + h_2 w_6 + b_3$
<hr/>	
$f_4$	$E = (y - \hat{y})^2$

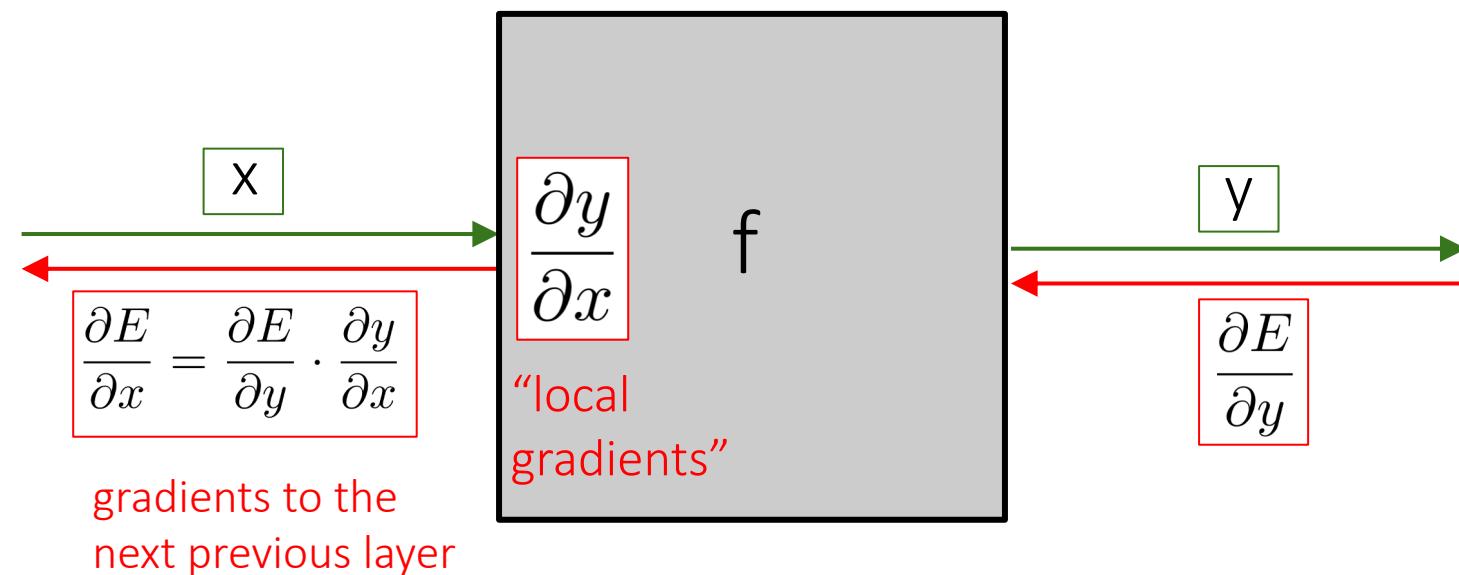
$$\text{argmin}_w \{ f_4 ( f_3 ( f_2 ( f_1 ( ) ) ) ) \}$$

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_5}$$

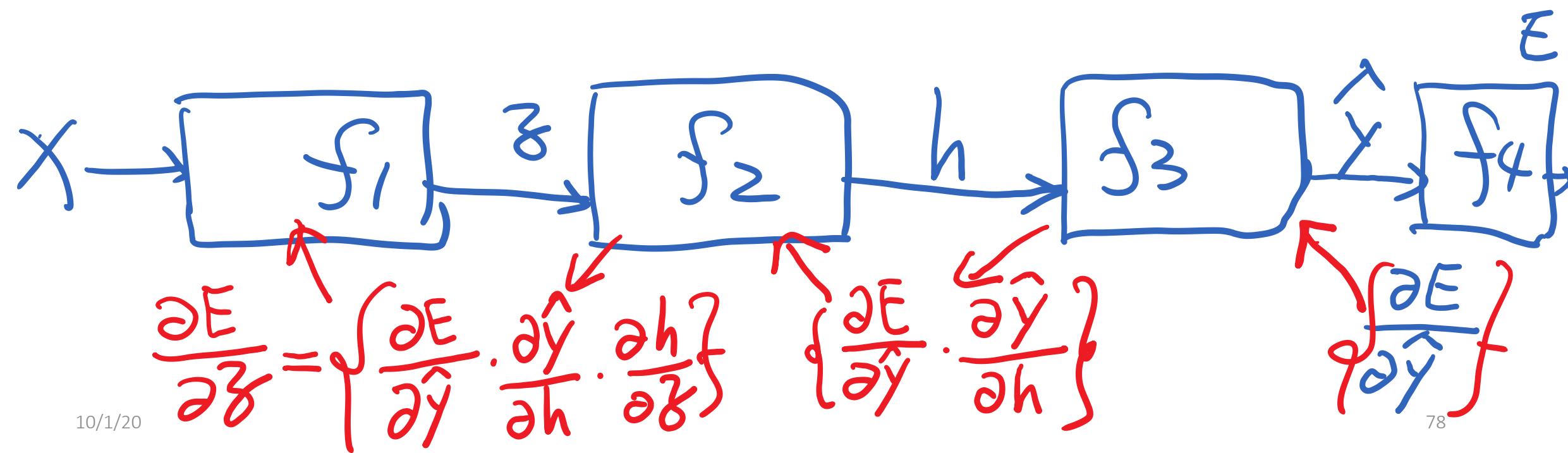
$$= -2(y - \hat{y}) \underline{h_1}$$

$$\frac{\partial E}{\partial w_5} =$$

$$\frac{\partial E}{\partial w_1} =$$

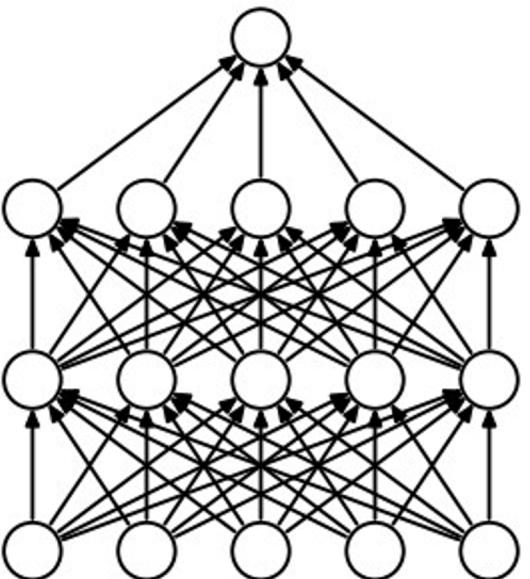


gradients to the  
next previous layer

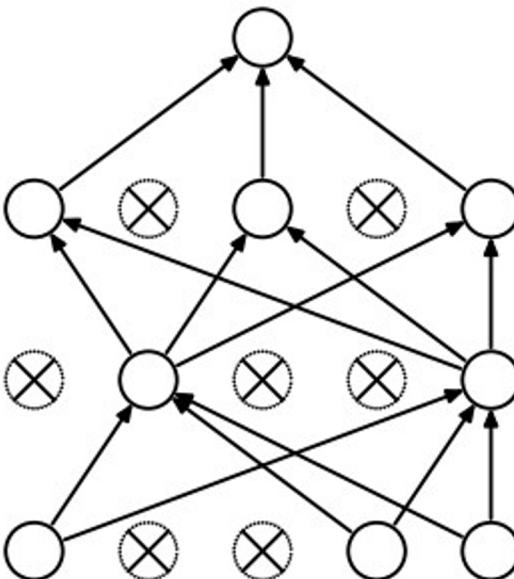


# Regularization by Dropout

“randomly set some neurons to zero in the forward pass”



(a) Standard Neural Net

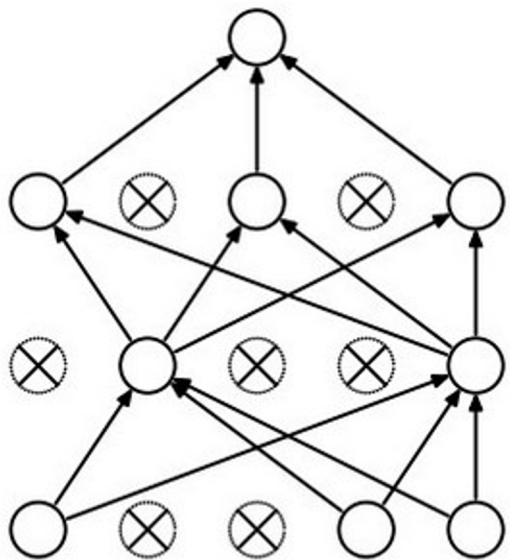


(b) After applying dropout.

[Srivastava et al., 2014]

Dropout is training a large ensemble of models (that share parameters).  
Each binary mask is one model

# Dropout At test time....



Ideally:  
want to integrate out all the noise

Monte Carlo approximation:  
do many forward passes with different  
dropout masks, average all predictions