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)

Data: X

↓

Task: y

↓

Representation: : x, f()

↓

Score Function: L()

↓

Search/Optimization : argmin()

↓

Models, Parameters, metrics

X: Tabular

↓

Binary Classification

↓

Log-odds y = linear function of Xs

↓

$$\sum_{i=1}^{N}\{y_i \log(\Pr(Y=1|X=x_i))+(1-y_i)\log(\Pr(Y=0|X=x_i))\}$$

↓

Iterative (Newton) method / SGD

↓

Logistic weights

/ Accuracy / F1

Last: Logistic Regression **Classifier**

- View I: logit(y) as linear of Xs
- View II: model Y as Bernoulli with p(y=1|x) as p(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

Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters, metrics

X: Tabular / 2D / 1D / 3D / Graph /…

Classification / Regression

Multilayer Network topology

neg Log-likelihood , Cross-Entropy / MSE / Many more

SGD / Backprop

Weights and biases in NN Layers

/ Accuracy / F1

Roadmap: DNN Basics

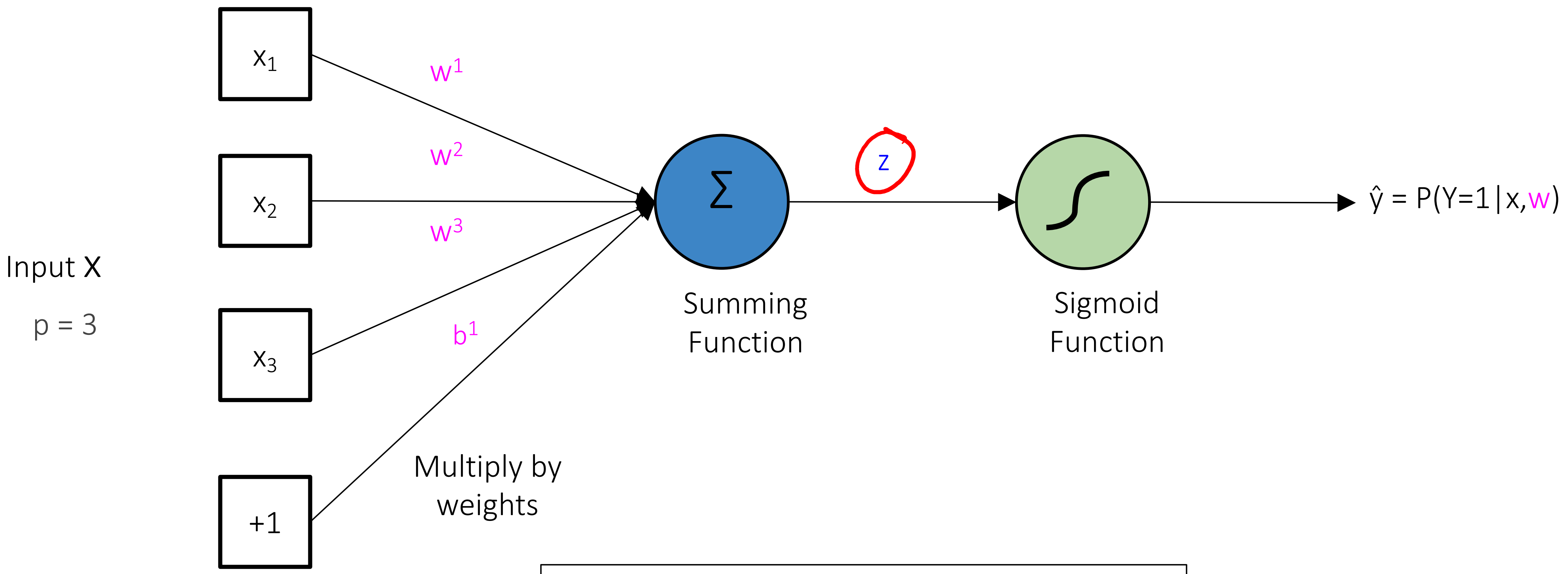
- Basics of Neural Network (NN)
  - single neuron, e.g. logistic regression unit
  - multilayer perceptron (MLP)
  - various loss function
    - E.g., when for multi-class classification, softmax layer
  - training NN with backprop algorithm
    - A few advanced tricks

ReWrite Logistic Regression as two stages:

First:
Summing $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$

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

One "Neuron": Expanded Logistic Regression

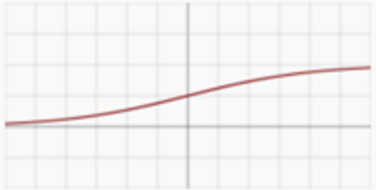
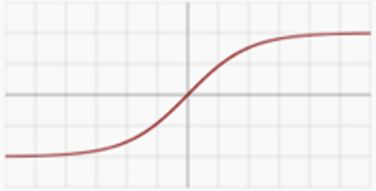
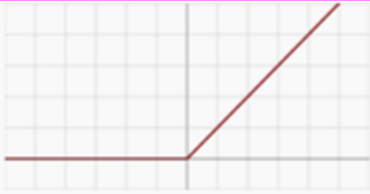


$z = w^T \cdot x + b$

$y = \text{sigmoid}(z) = \dfrac{e^z}{1 + e^z}$

E.g., Many Possible Nonlinearity Functions

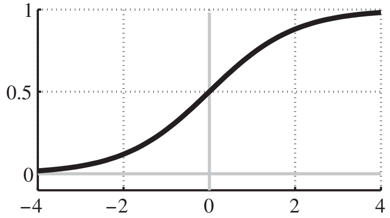
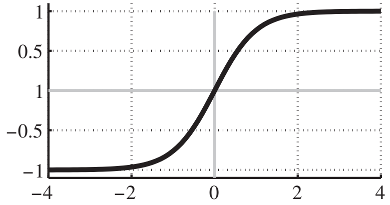
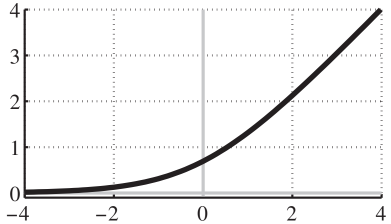
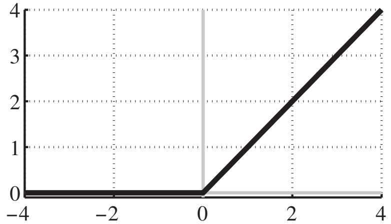
(aka transfer or activation functions)

Name	Plot	Equation	Derivative (w.r.t x)
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
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)[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$

usually works best in practice

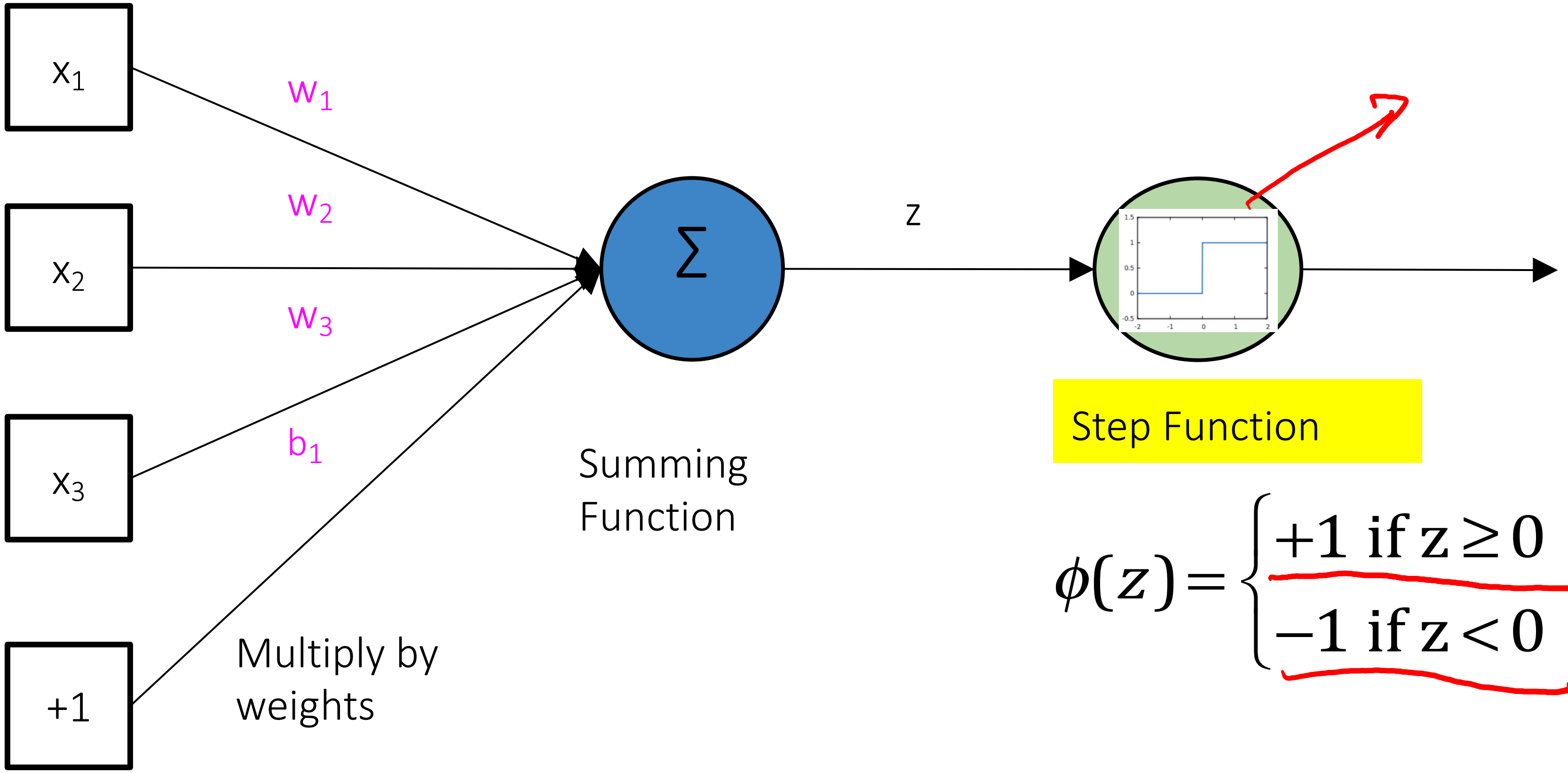
https://en.wikipedia.org/wiki/Activation_function#Comparison_of_activation_functions

Activation functions

Name and Graph	**Function**	**Derivative**
sigmoid(x) 	$h(x) = \dfrac{1}{1 + \exp(-x)}$	$h'(x) = h(x)[1 - h(x)]$
tanh(x) 	$h(x) = \dfrac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$	$h'(x) = 1 - h(x)^2$
softplus(x) 	$h(x) = \log(1 + \exp(x))$	$h'(x) = \dfrac{1}{1 + \exp(-x)}$
rectify(x) 	$h(x) = \max(0, x)$	$h'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$

History ➔ Perceptron: 1-Neuron Unit with Step

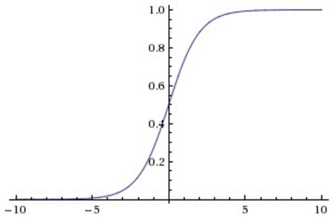
- First proposed by Rosenblatt (1958)
- A simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function**



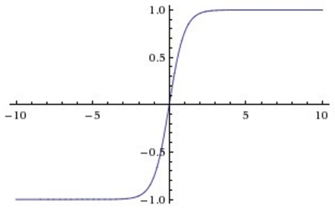
$$\phi(z) = \begin{cases} +1 \text{ if } z \geq 0 \\ -1 \text{ if } z < 0 \end{cases}$$

Sigmoid

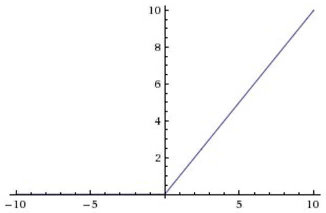
$\sigma(x) = 1/(1+e^{-x})$



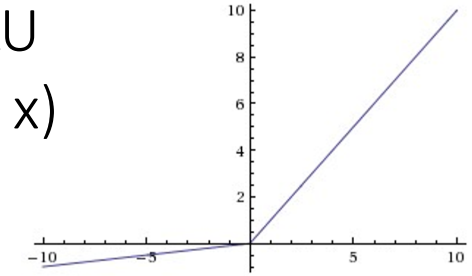
tanh   tanh(x)



ReLU   max(0,x)

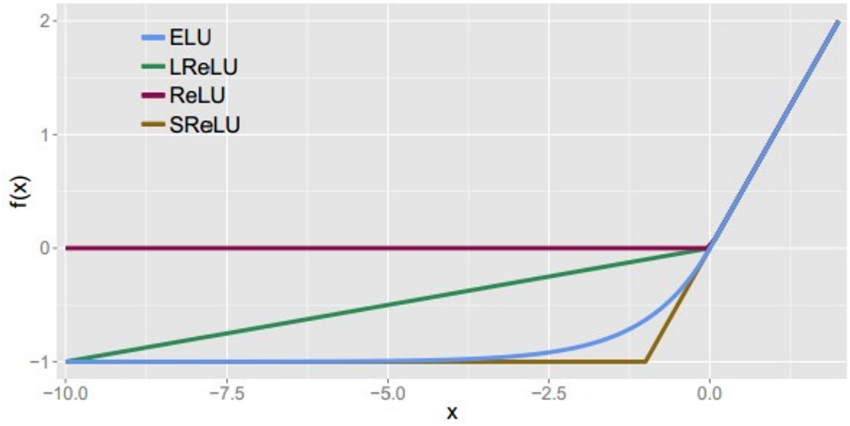


Leaky ReLU
max(0.1x, x)



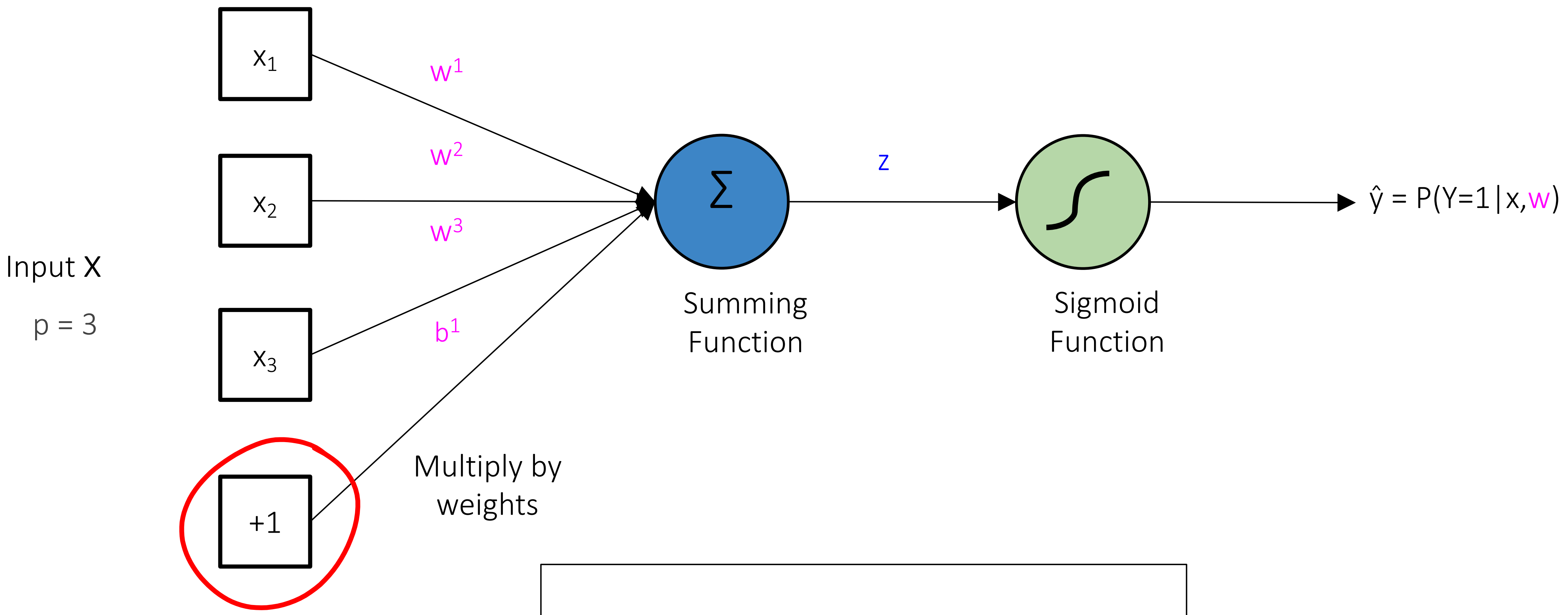
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

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



From Feifei Li Stanford Cousre

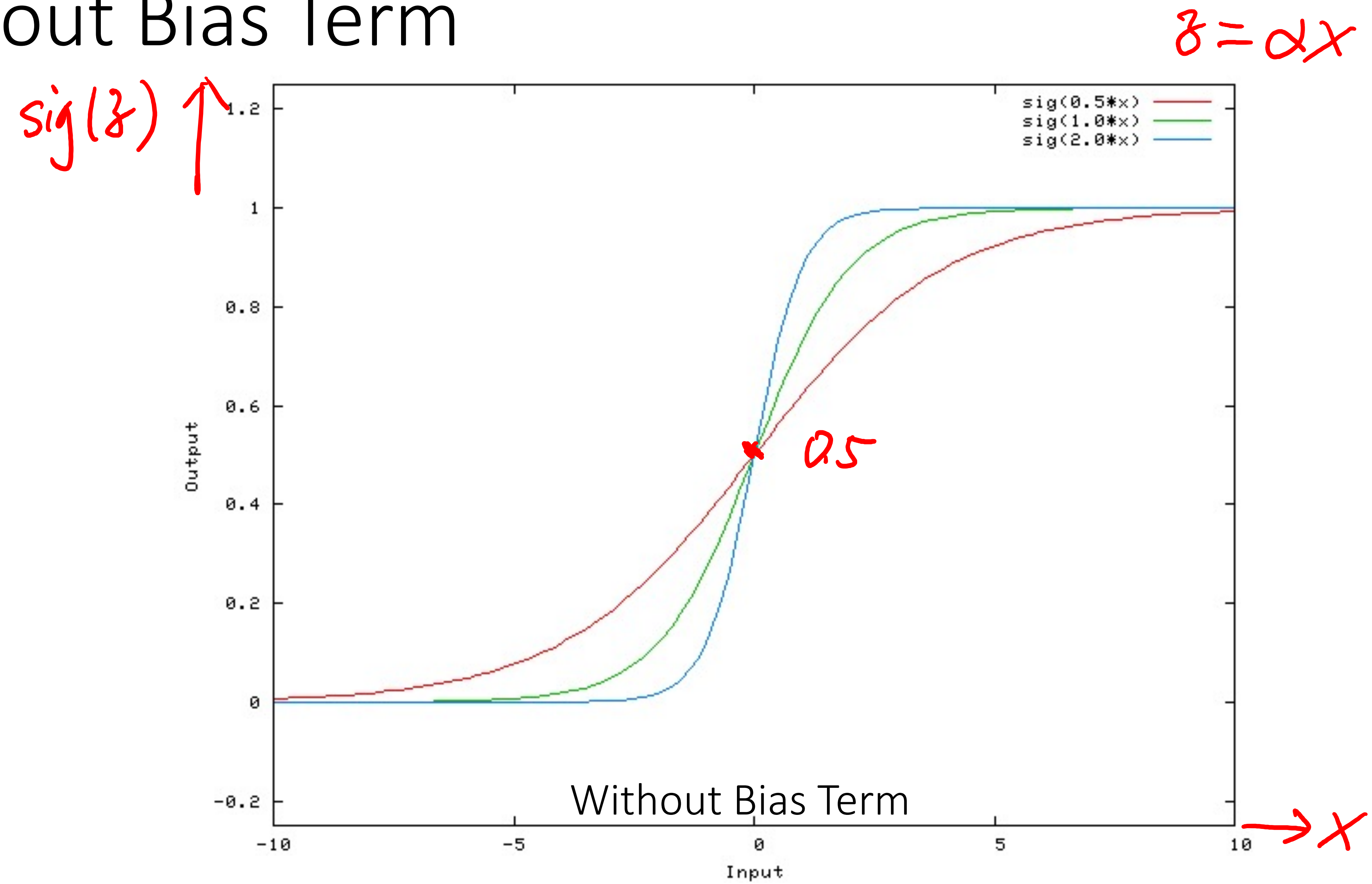
Bias Term?



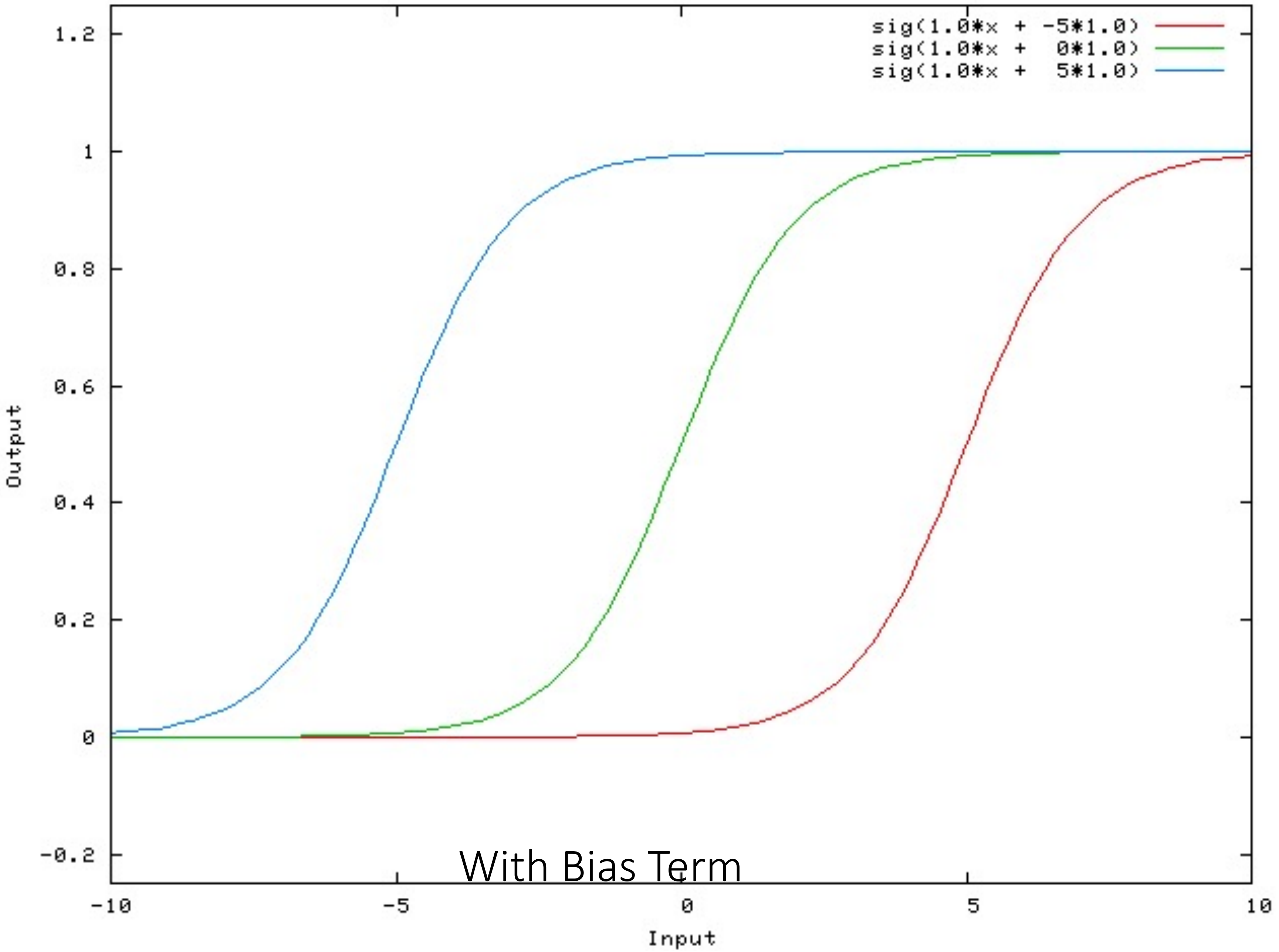
$z = W^T \cdot X + b$

$y = \text{sigmoid}(z) = \frac{e^z}{1 + e^z}$

Without Bias Term



With Bias Term

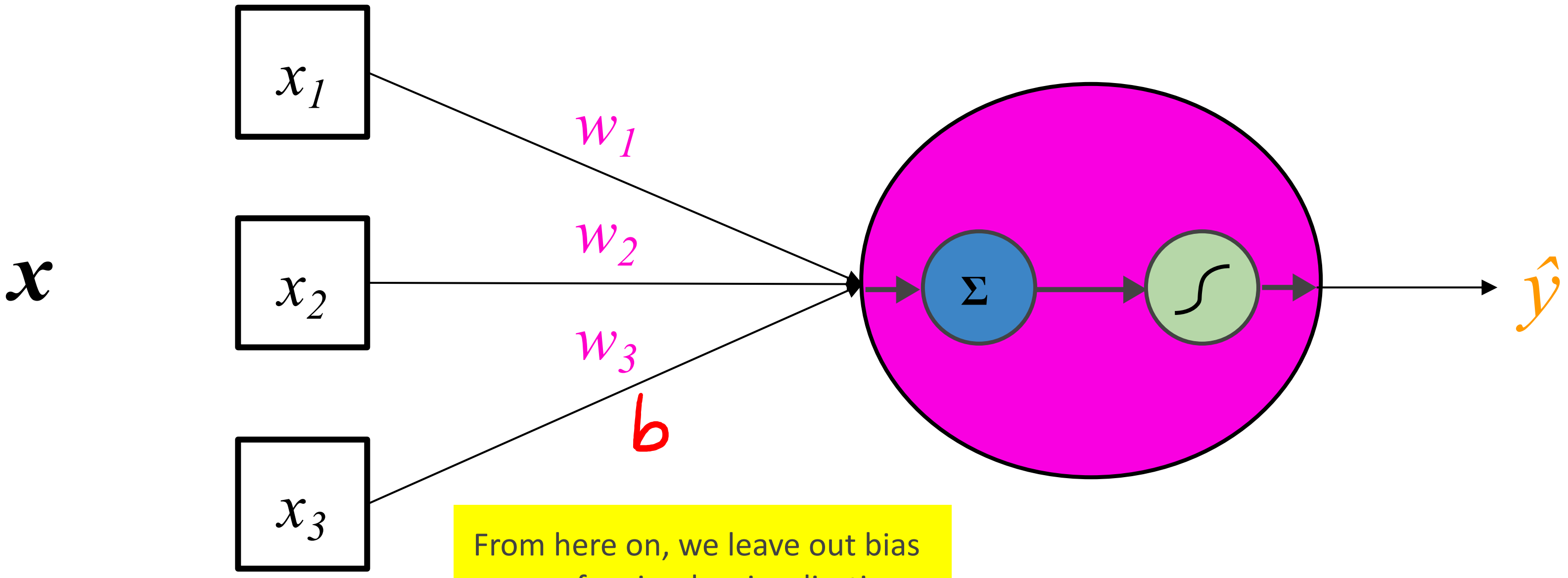


With Bias Term

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

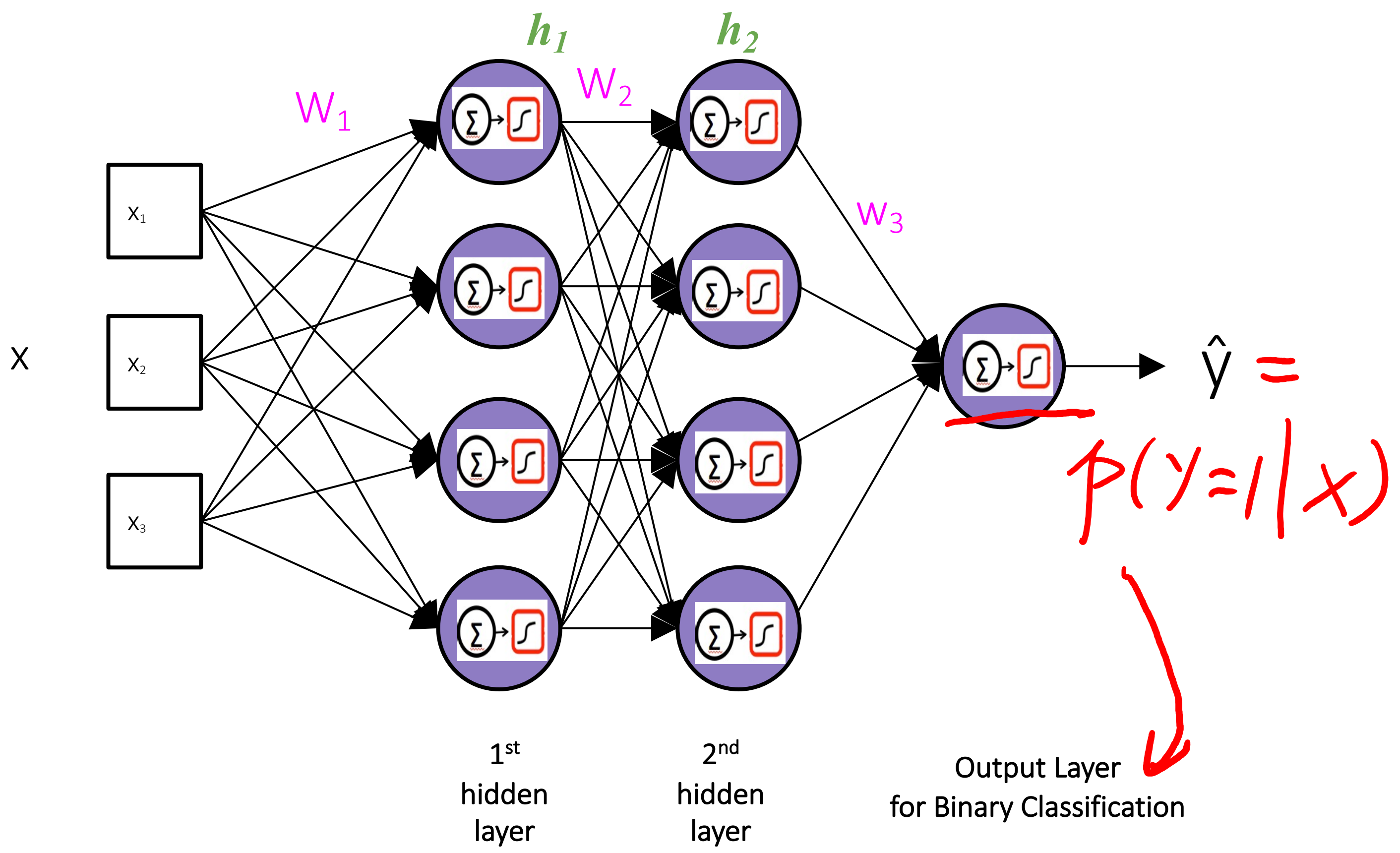
Neuron Representation



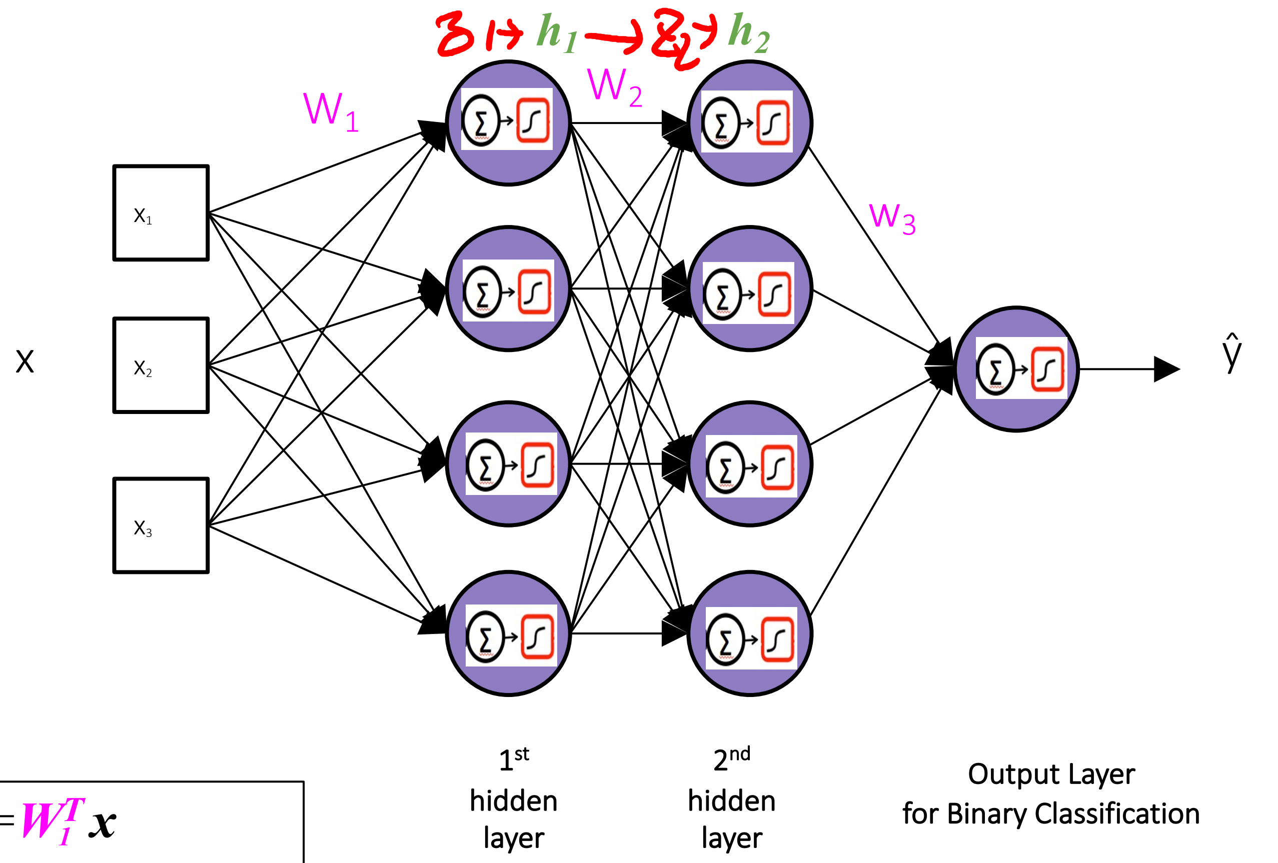
From here on, we leave out bias square for simple visualization

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)

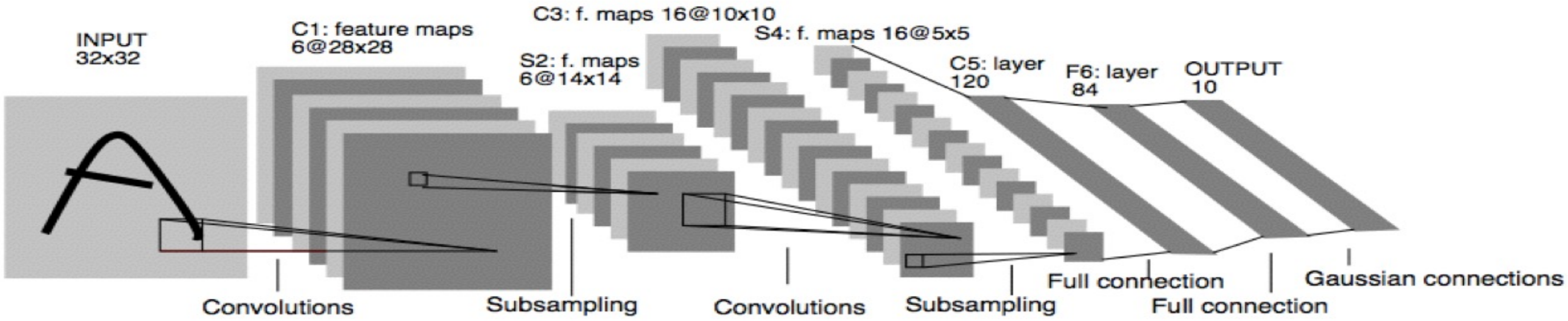


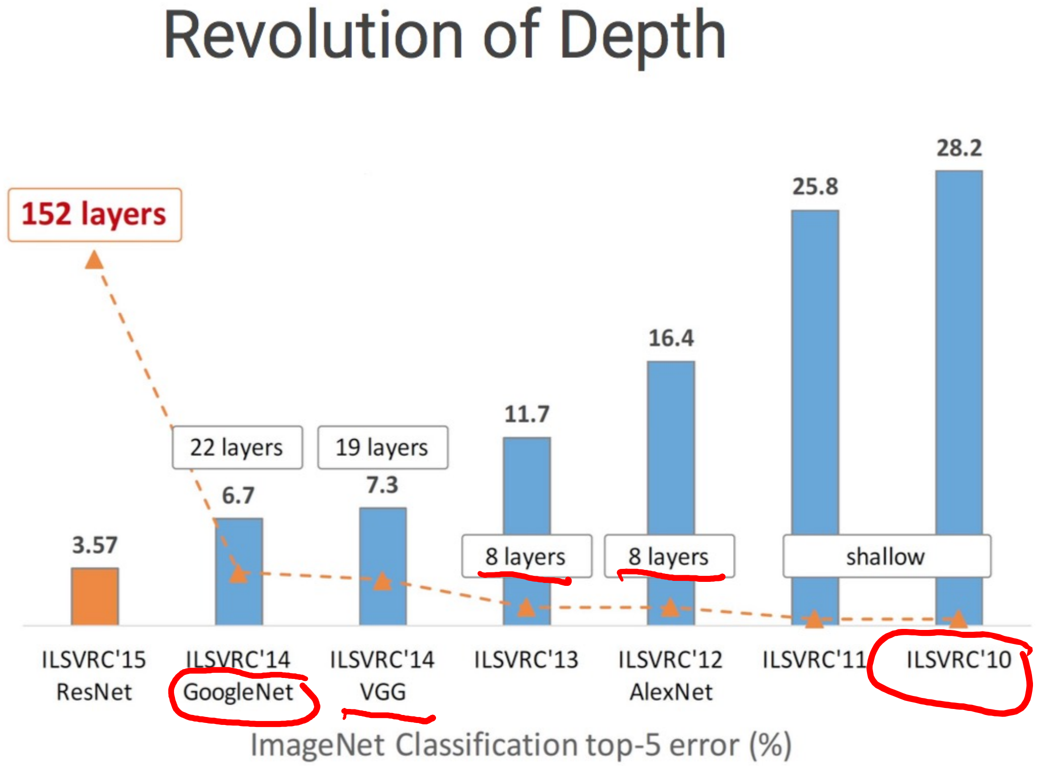
hidden layer 1 output →
hidden layer 2 output →

$$z_1 = W_1^T x$$
$$h_1 = sigmoid(z_1)$$
$$z_2 = W_2^T h_1$$
$$h_2 = sigmoid(z_2)$$
$$z_3 = w_3^T h_2$$
$$\hat{y} = sigmoid(z_3)$$

$$x \xrightarrow{w_1} z_1 \to h_1 \xrightarrow{w_2} z_2 \to h_2 \xrightarrow{w_3} \hat{y}$$

"Deep" Neural Networks (i.e. many hidden layers)



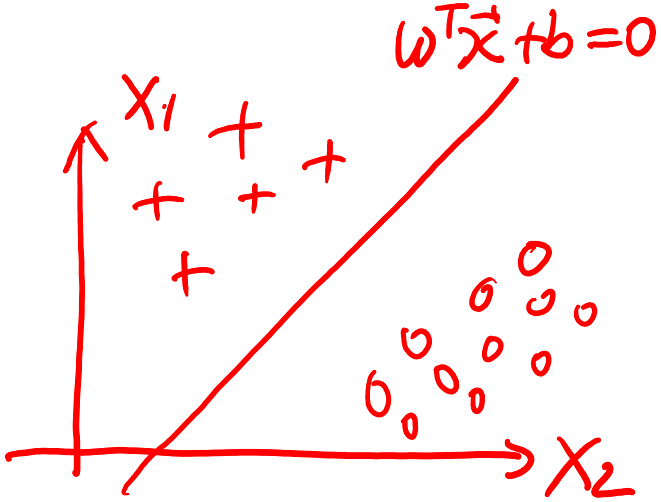


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

View IV: Logistic Regression models a linear classification boundary!

P(y=1|x) = P(y=0|x) = 0.5

Boundary
no decision



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

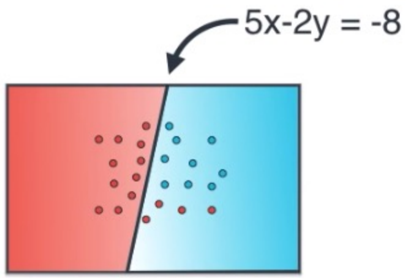
$$\ln\left[\frac{P(y|x)}{1-P(y|x)}\right]=\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_p x_p$$

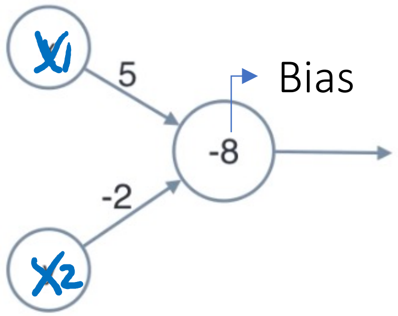
Decision Boundary ➔ equals to zero

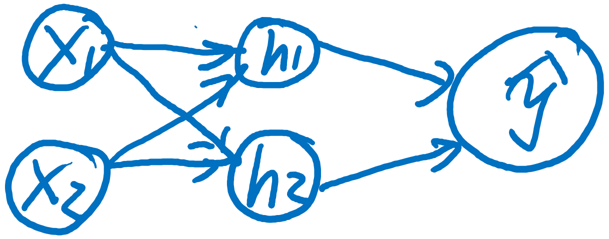
$$\ln\left[\frac{P(y=1|x)}{P(y=0|x)}\right]=\ln\left[\frac{P(y=1|x)}{1-P(y=1|x)}\right]=\alpha+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_p x_p$$

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

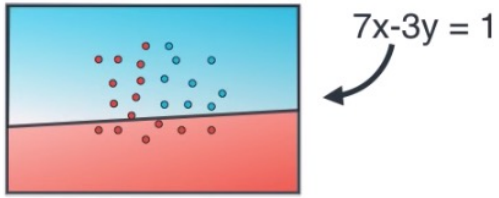
One neuron h1

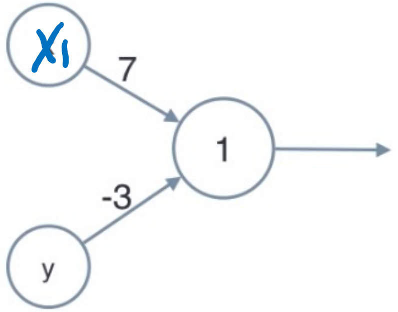




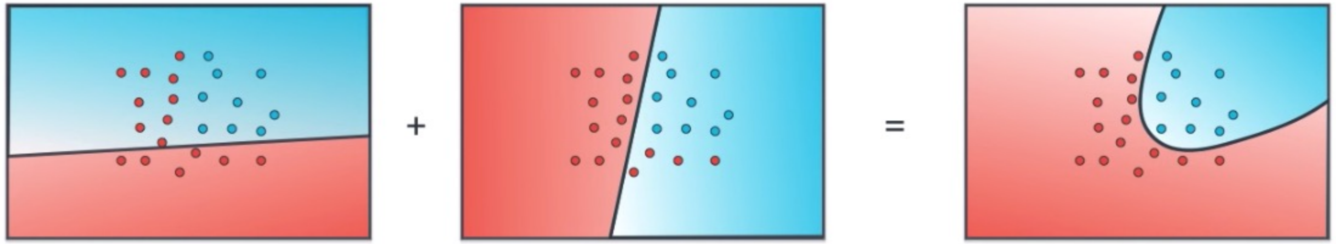


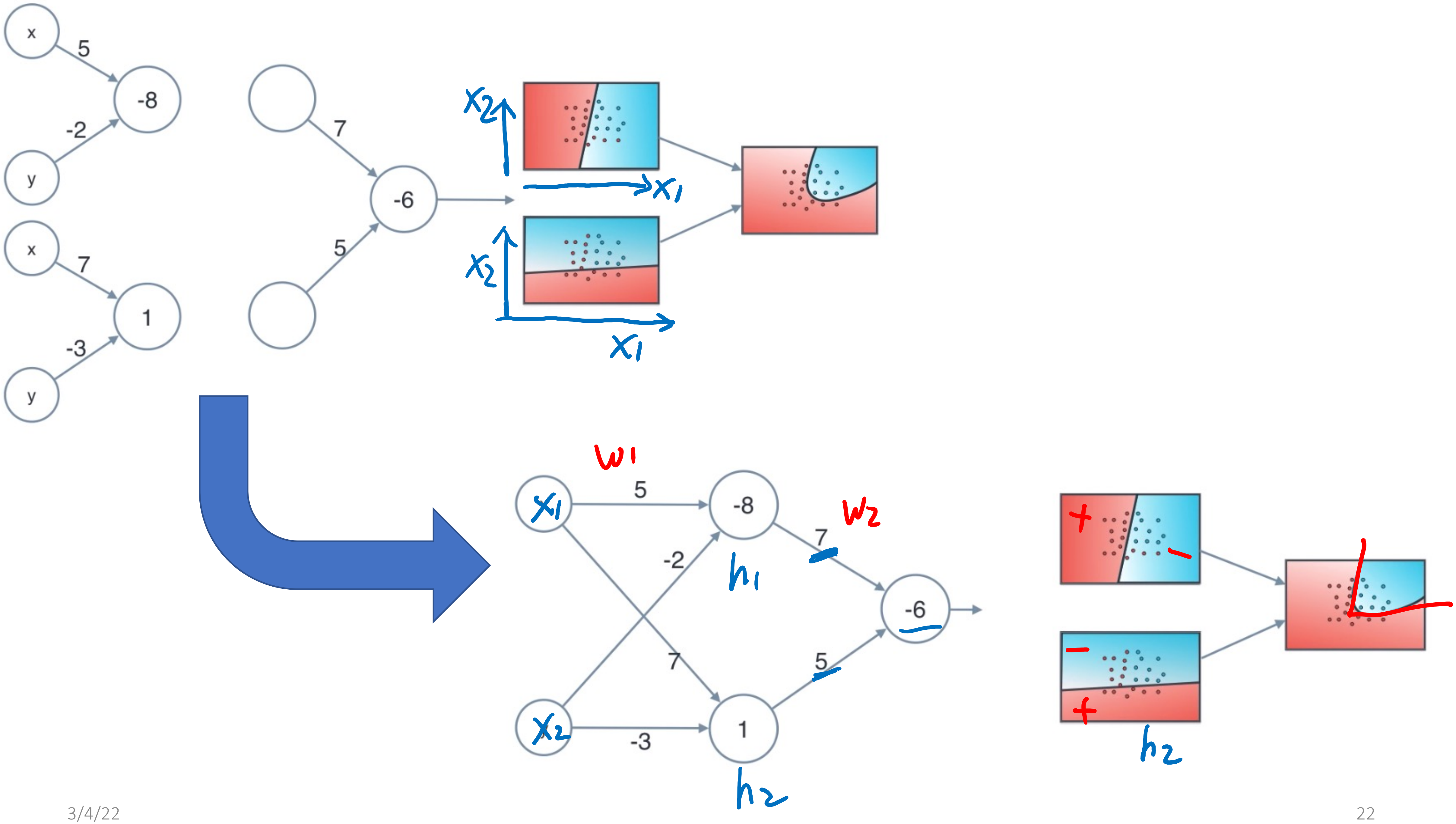
Another neuron h2





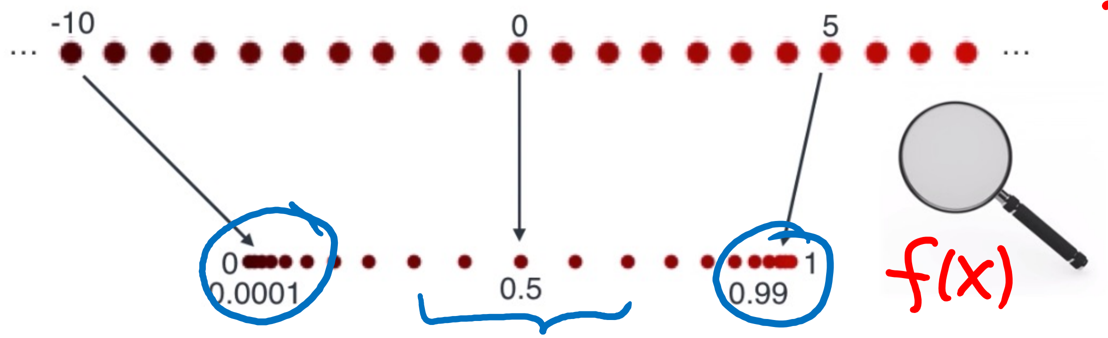
How to combine to get nonlinear decision boundary?



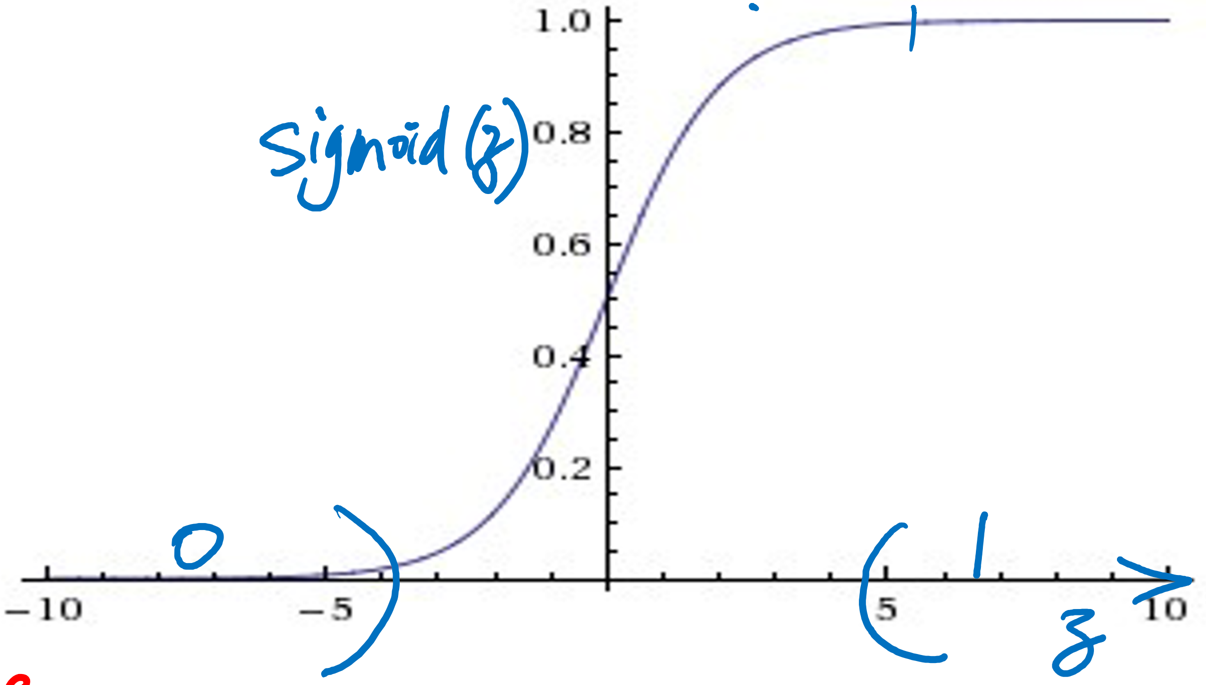

x
5
-8
-2
y
7
-6
5
x
7
1
-3
y
x2
x1
x2
x1
w1
x1
5
-8
-2
h1
7
w2
-6
7
5
x2
-3
1
h2
+
-
-
+
h2

From https://www.youtube.com/watch?v=BR9h47Jtqyw&list=PLs8w1Cdi-zvavXlPXEAsWlh4Cgh83pZPO&index=3

Activation function

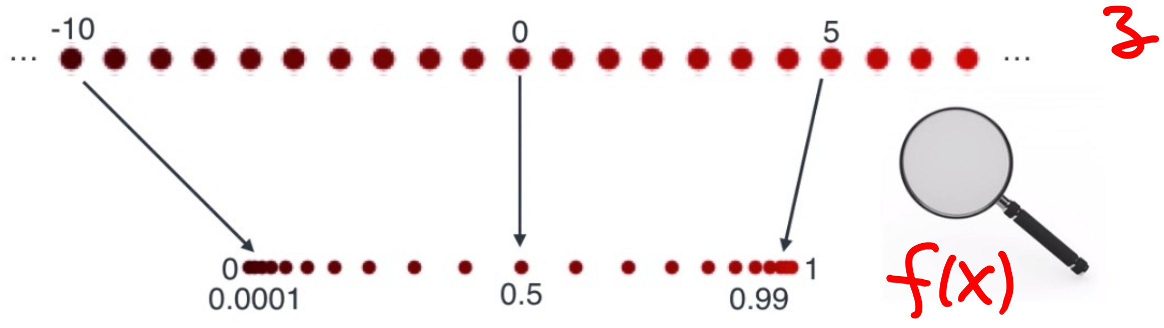


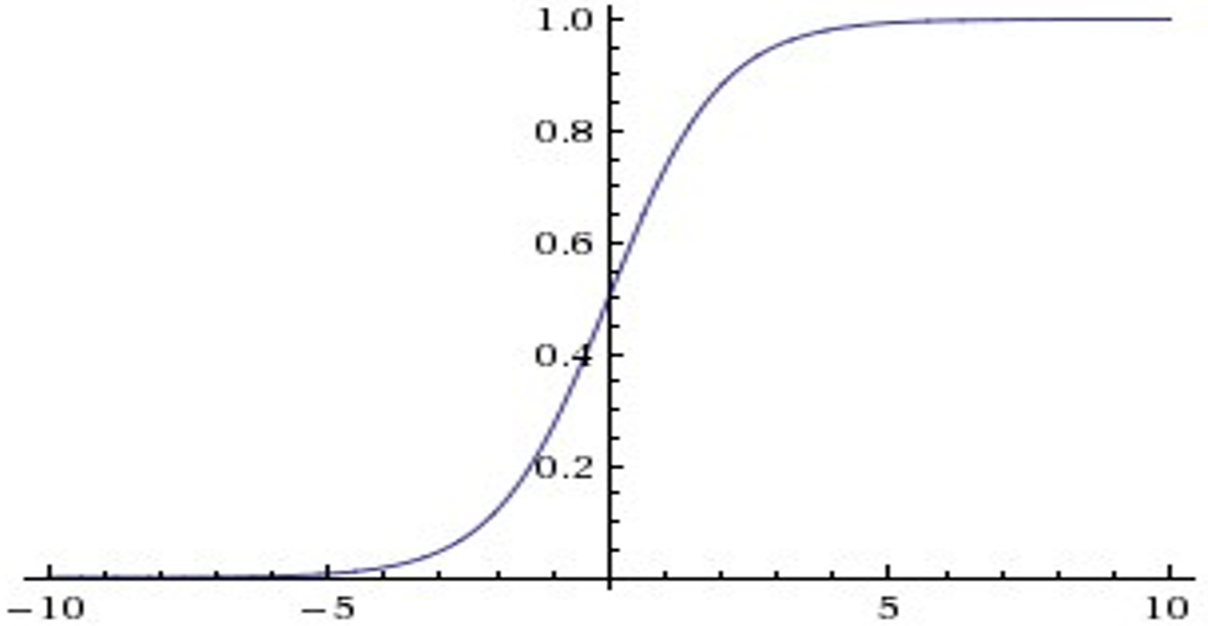
$$f(x) = \frac{1}{1 + e^{-x}}$$



[From https://www.youtube.com/watch?v=BR9h47Jtqyw&list=PLs8w1Cdi-zvavXlPXEAsWlh4Cgh83pZPO&index=3](https://www.youtube.com/watch?v=BR9h47Jtqyw&list=PLs8w1Cdi-zvavXlPXEAsWlh4Cgh83pZPO&index=3)

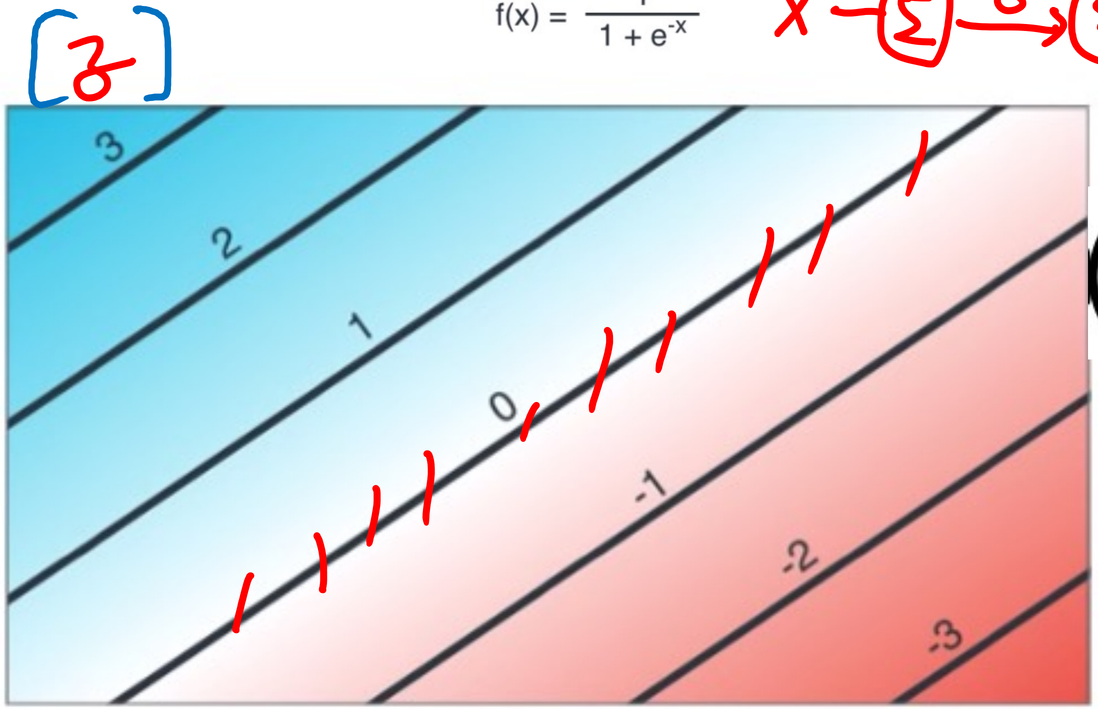
Activation function

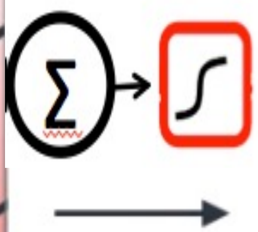


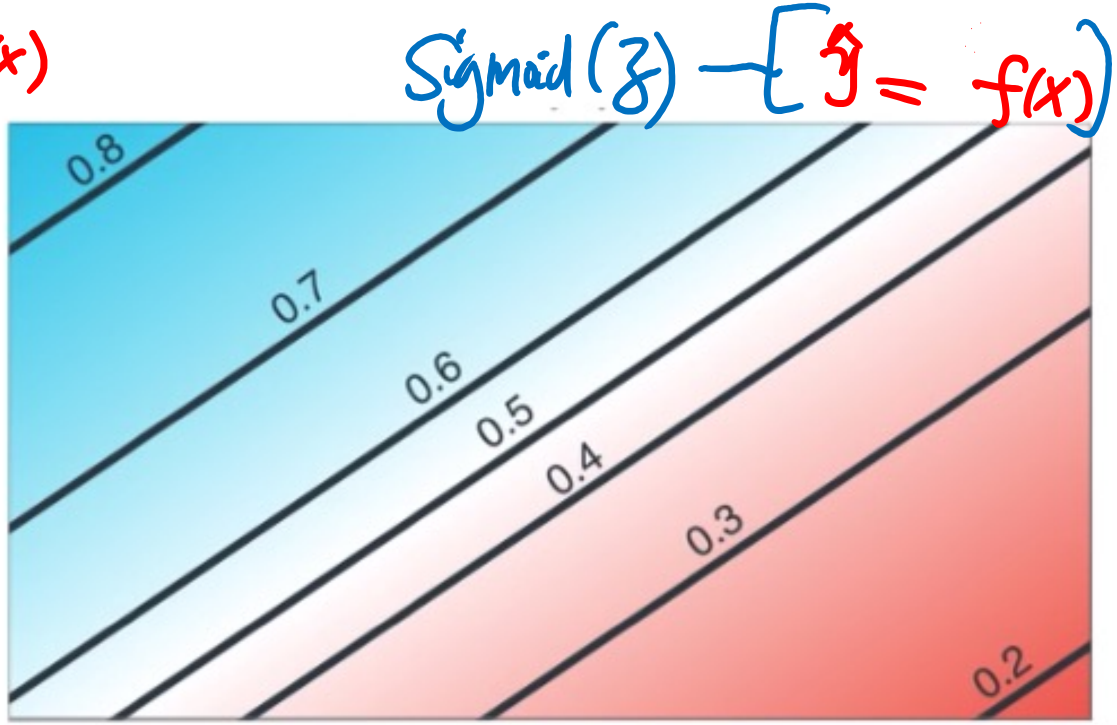


$$f(x) = \frac{1}{1 + e^{-x}}$$

X → Σ → z → sig → f(x)







Today: Basic Neural Network Models

Data: X
↓
Task: y
↓
Representation: : x, f()
↓
Score Function: L()
↓
Search/Optimization : argmin()
↓
Models, Parameters, metrics

X: Tabular / 2D / 1D / 3D / Graph /…
↓
Classification / Regression
↓
Multilayer Network topology
↓
neg Log-likelihood , Cross-Entropy / MSE / Many more
↓
SGD / Backprop
↓
Weights and biases in NN Layers / Accuracy / F1

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

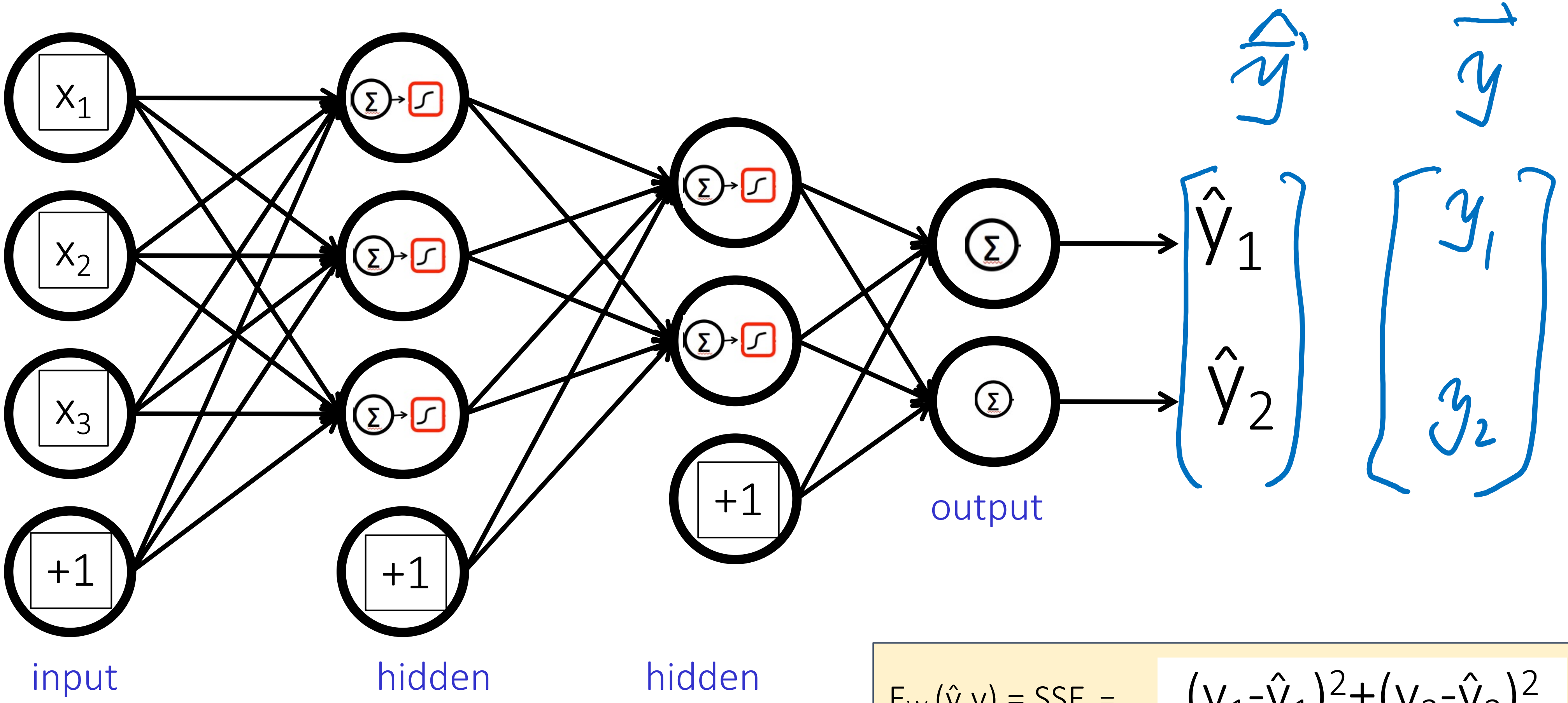
Module II

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

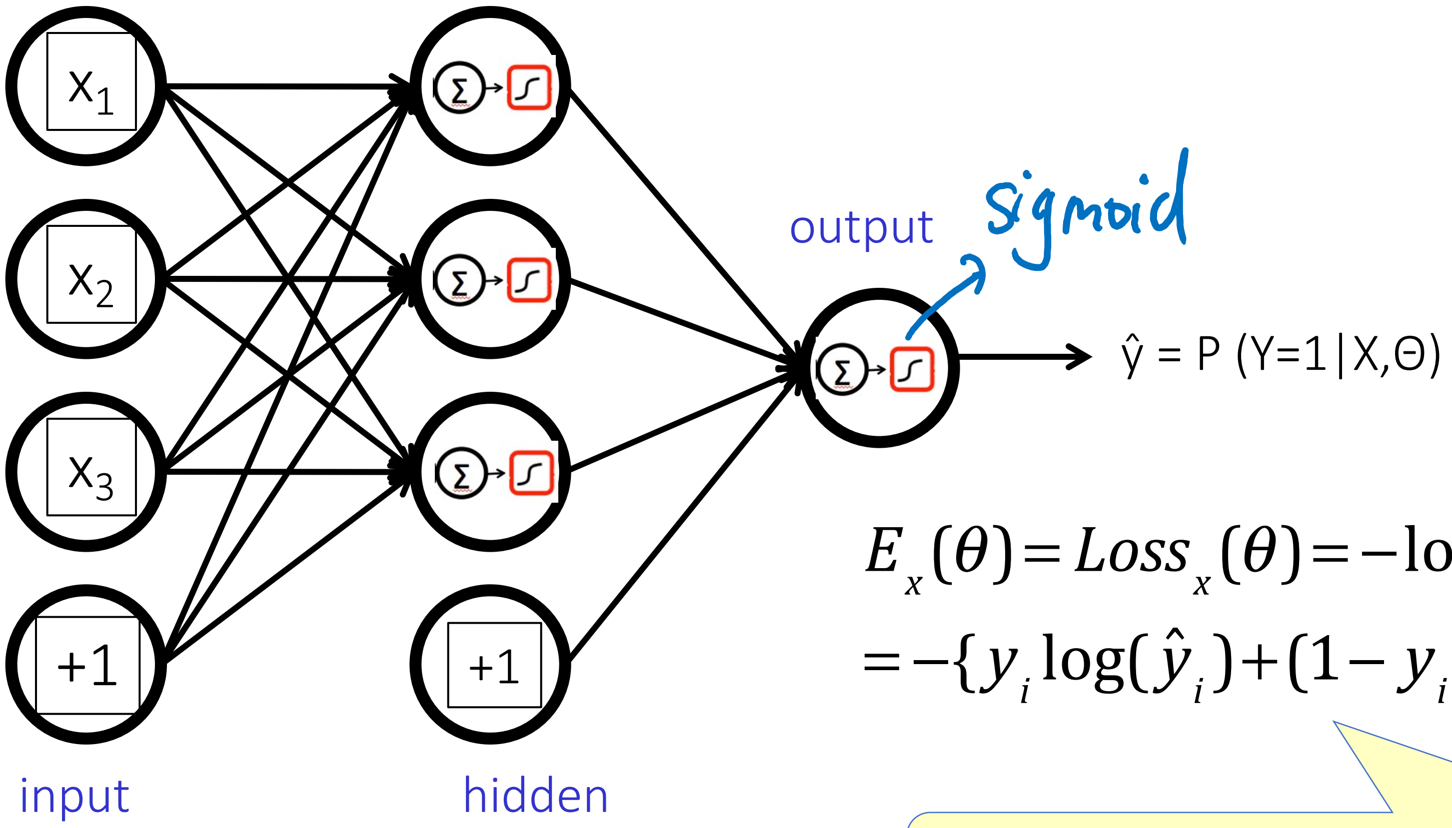
E.g., SSE loss on Multi-Layer Perceptron (MLP) for Regression

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



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

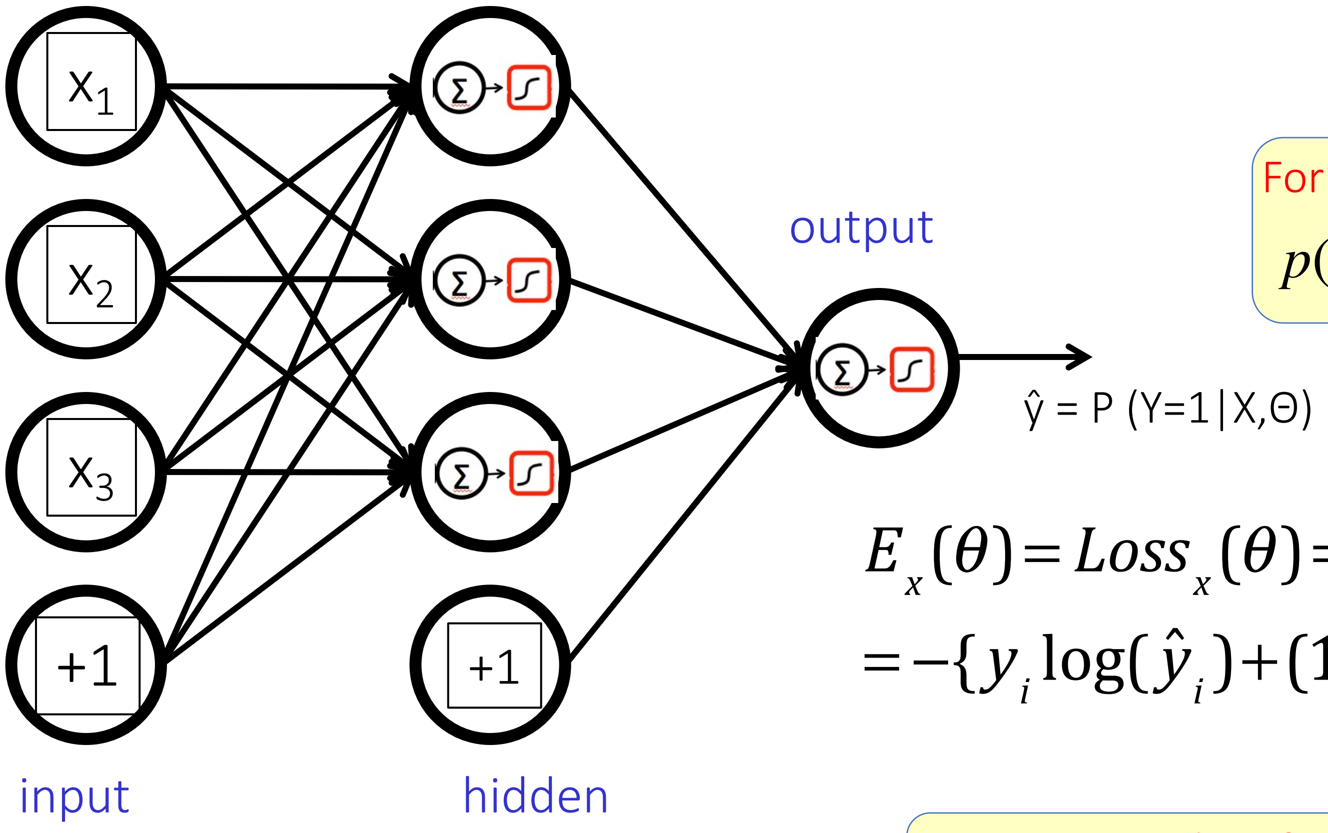
e.g., Cross-Entropy loss for Multi-Layer Perceptron (MLP) for Binary Classification

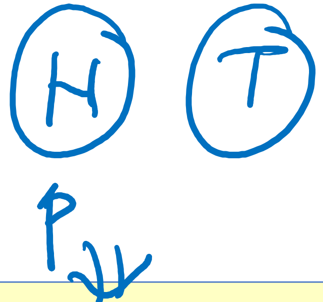


$$E_x(\theta) = Loss_x(\theta) = -\log\Pr(Y = y \mid X = x)$$
$$= -\{y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\}$$

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,

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

$$E_x(\theta) = Loss_x(\theta) = -\log\Pr(Y=y|X=x)$$

$$= -\{y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\}$$

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

Basic Bernoulli | Logistic / Bernoulli

LIKELIHOOD:

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

↑ function of p=Pr(head)

$$L(\beta) = \prod_{i=1}^{n} p(y_i=1|x_i)^{y_i} (1-p(y_i=1|x_i))^{1-y_i}$$

$$= \prod_{i=1}^{n} \hat{y}_i^{y_i} (1-\hat{y}_i)^{1-y_i}$$

$$\log(L(p)) = \log\left[\prod_{i=1}^{n} p^{z_i} (1-p)^{1-z_i}\right]$$

$$= \sum_{i=1}^{n} (z_i \log p + (1-z_i)\log(1-p))$$

Log likelihood

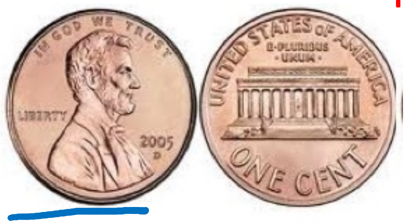
$$ll(\beta) = \sum_{i=1}^{n} \left(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right)$$

Binary Classification ➔ Multi-Class Classification

$p(c|x)$

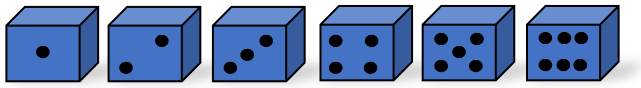
models the target binary random variable with Bernoulli whose parameter p=p(y=1|x) predefined as function on x

1-p(y=1x)



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

Multinoulli Distribution, e.g.,



k

$p(y=1|x)$
$p(y=2|x)$
$p(y=3|x)$
⋮
$p(y=6|x)$

Multi-class target variable representation

K=4

Total Class

y_K → $k=4$

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

one-hot

- Multi-class output variable ➔

An indicator basis vector representation

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

$$P(y = C_k \mid x)$$

$G: \{1, 2, 3, 4\} \rightarrow k=4$

$C: \{$Poli, fin, Health, Enter, tech$\} \rightarrow k=5$

Review: Multi-class variable representation

Class y_K →

N ↓

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

- How to classify to multi-class ?
  - First: learn K different regression (x)
  - Then: Softmax using all K outputs as input

Review: Multi-class variable representation

Class

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
  - Then: ➡ $\hat{G}(x) = \underset{k \in g}{\mathbf{argmax}}\, \hat{f}_k(x)$

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

Review: Multi-class variable representation

Class y_K

N

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

- How to classify to multi-class ?
  - First: learn K different regression
  - Then: Softmax using all K outputs as input
  - Then:

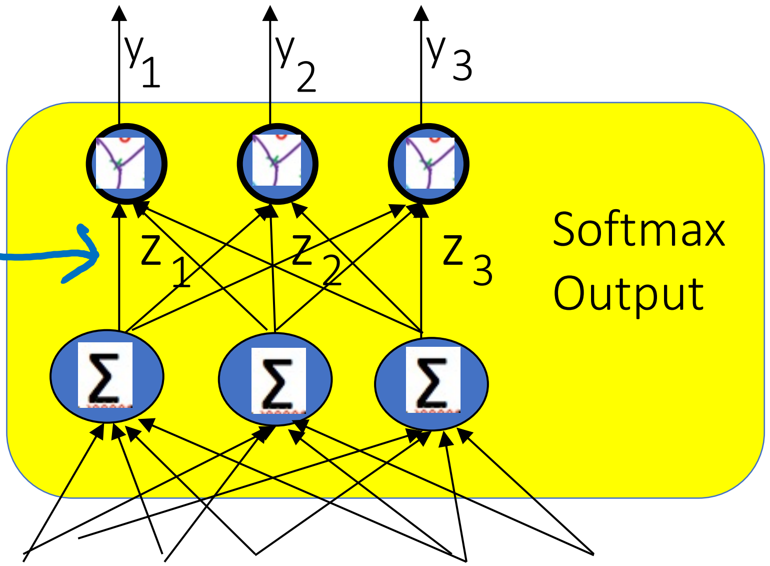
$$\hat{G}(x) = \underset{k \in g}{\mathbf{argmax}}\, \hat{f}_k(x) \quad p_k(x)$$

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

$p(y=k|x)$

discriminative classifier

Strategy : Use "softmax" layer function for multi-class classification



$z_1, z_2, .., z_k$ → regression

$p(y_k|x)$

$f_1(x)+f_2(x) + \dots + f_k(x)=1$

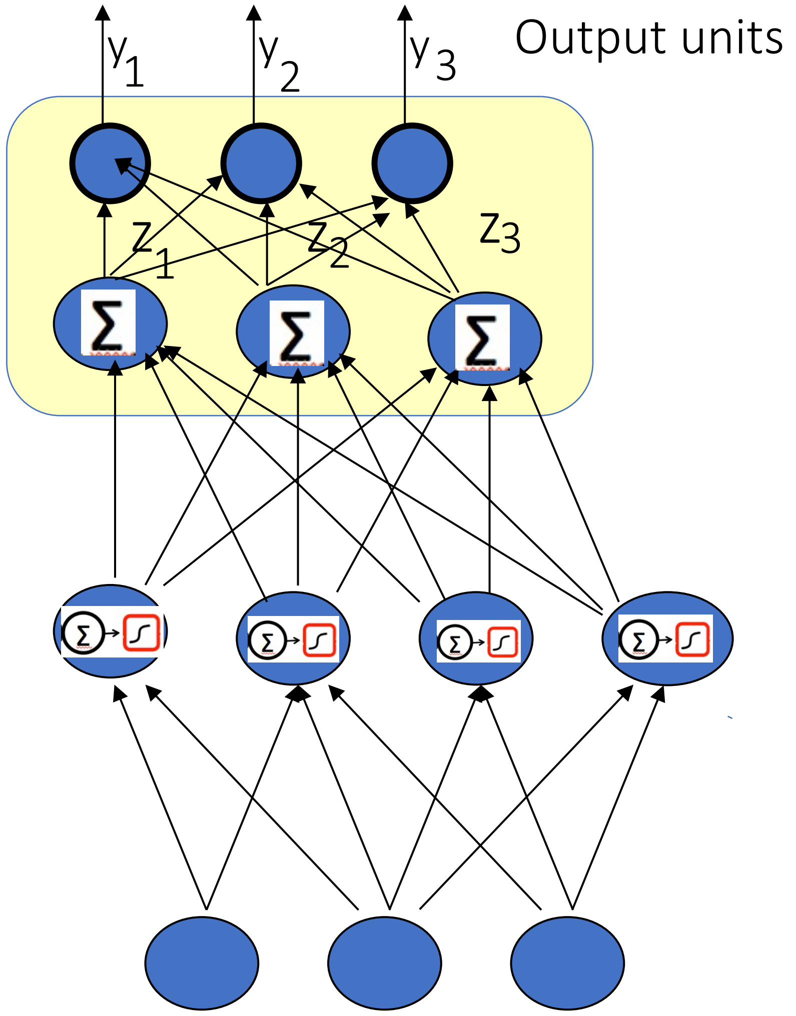
$0 \le f_i(x) \le 1$

$$Pr(G = k \mid X = x) = Pr(Y_k = 1 \mid X = x)$$

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

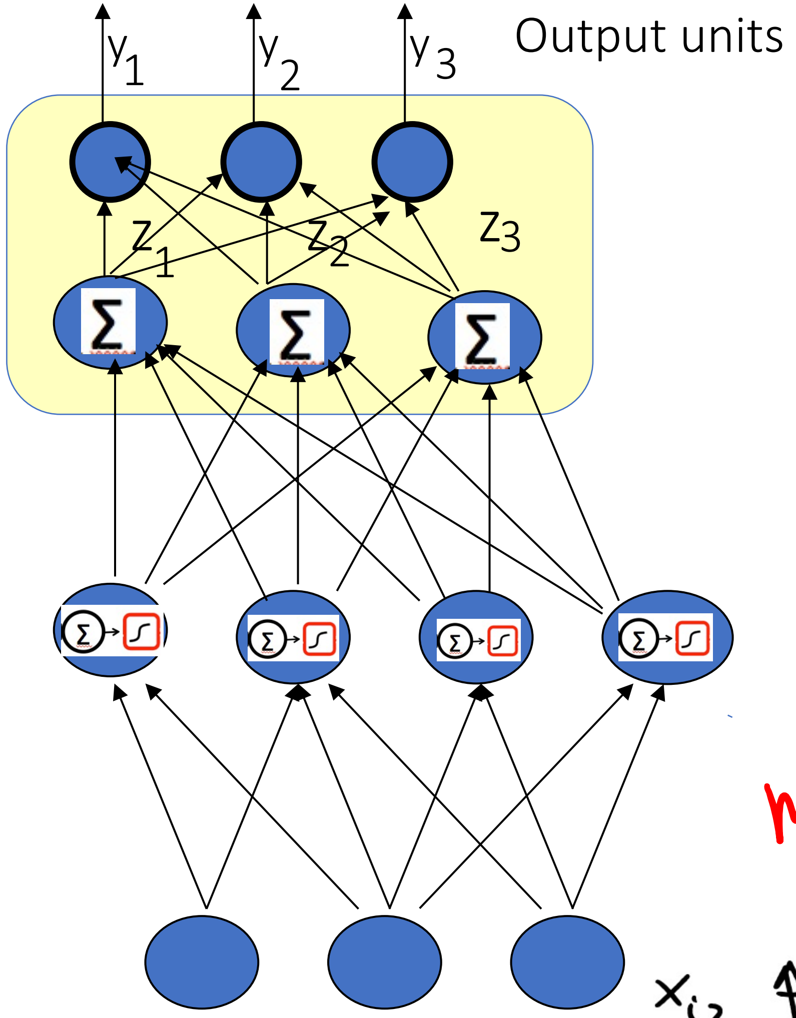
"Softmax" functio: Normalizing function which converts each class output to a probability.

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



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

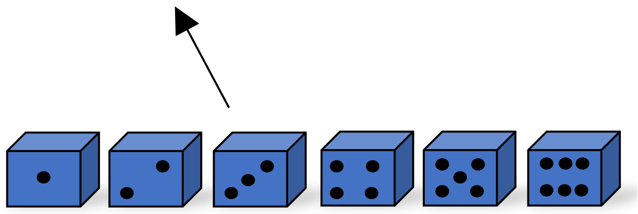
last layer is softmax output layer ➔
a Multinoulli logistic regression unit



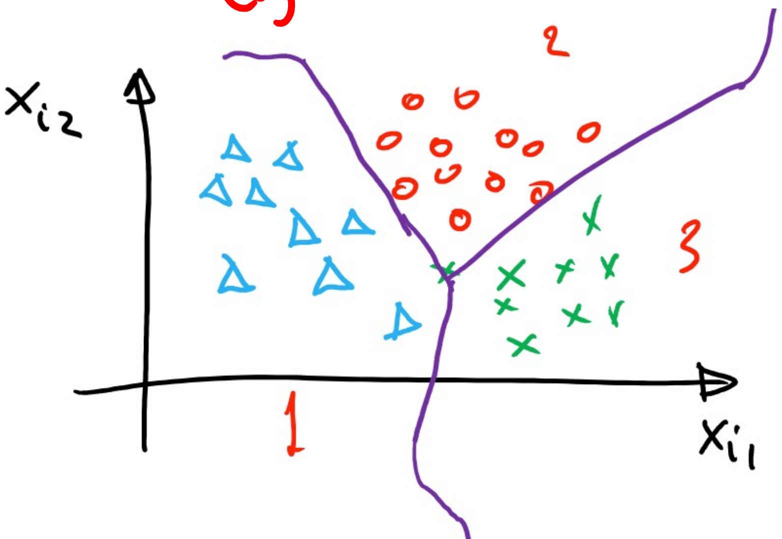
When for multi-class classification (last output layer: softmax layer)

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

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



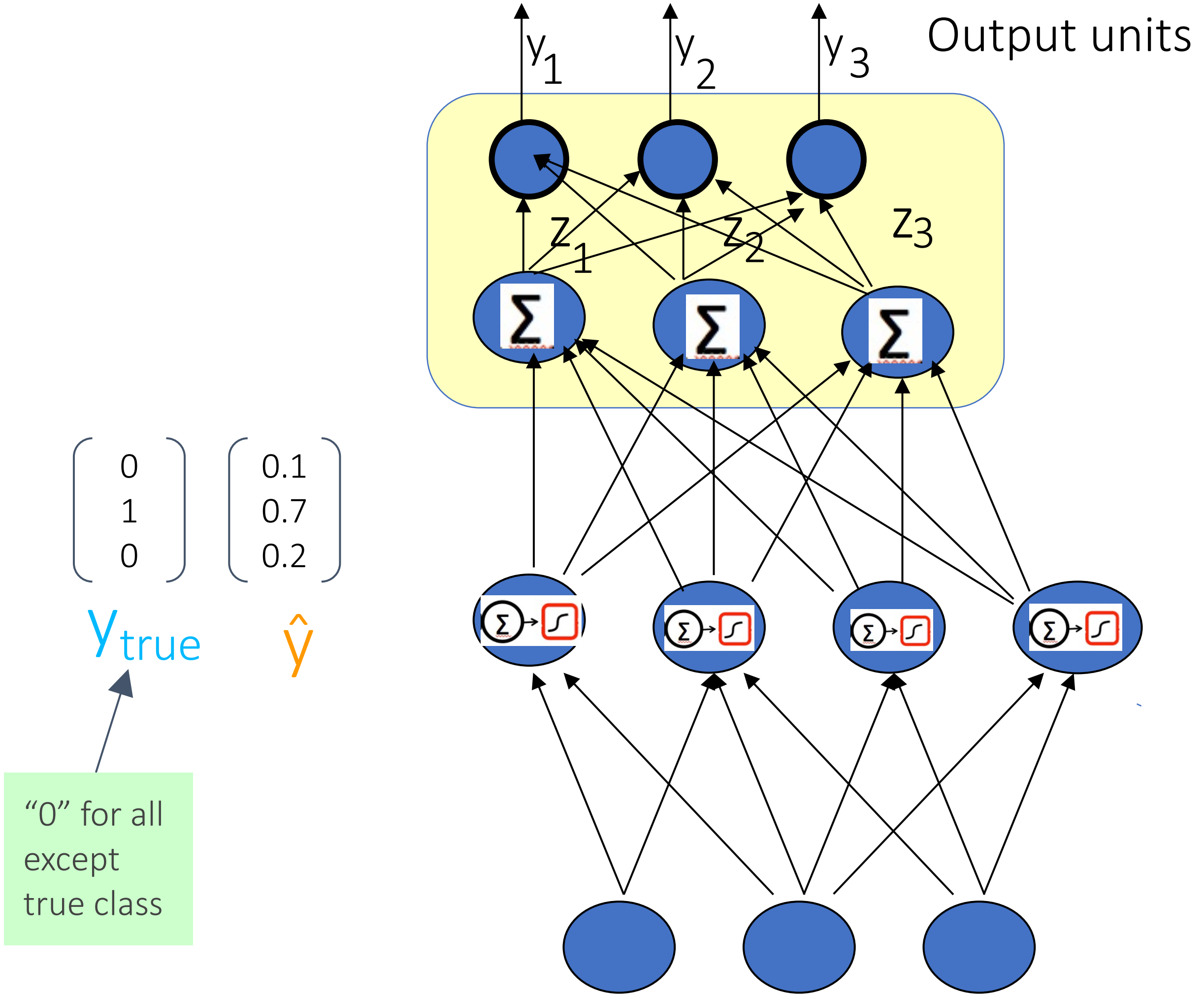
multiclass logistic regression



y_{i1}	y_{i2}	y_{i3}	
0	1	0	Class 2
1	0	0	Class 1
0	0	1	Class 3

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

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



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

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

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

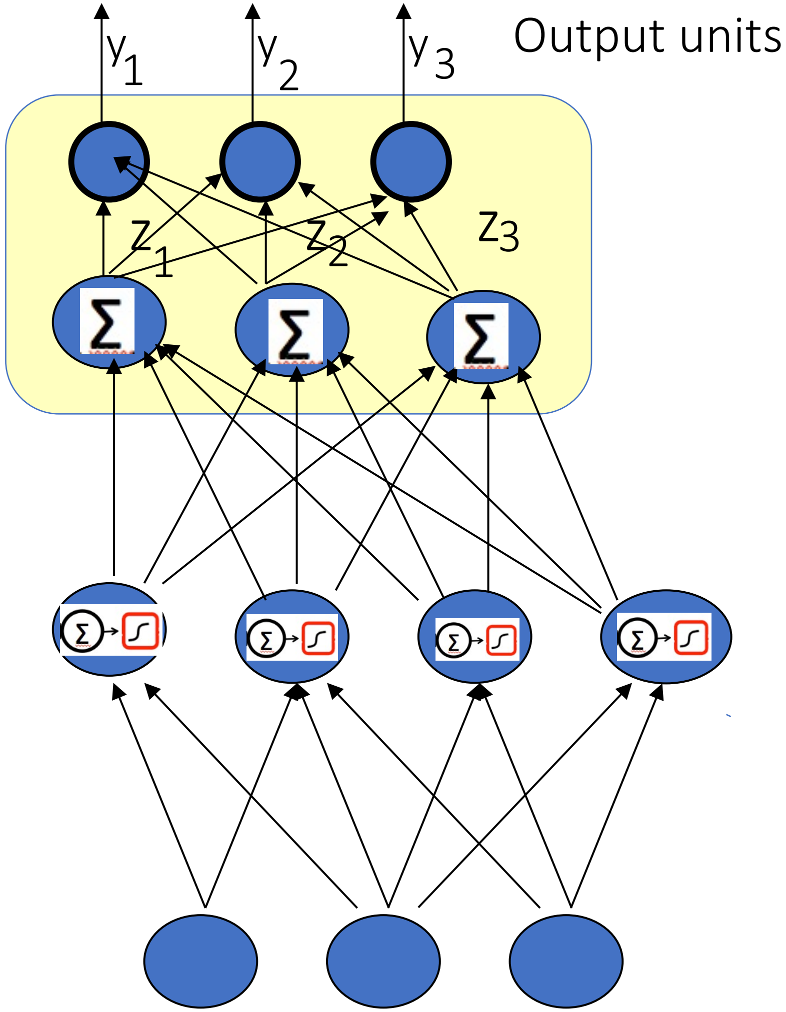
last layer is softmax output layer ➔ a Multinoulli logistic regression unit

one hot

$$K\left\{\begin{pmatrix}0\\1\\0\end{pmatrix}\right. \quad \begin{pmatrix}0.1\\0.7\\0.2\end{pmatrix}$$

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

"0" for all except true class



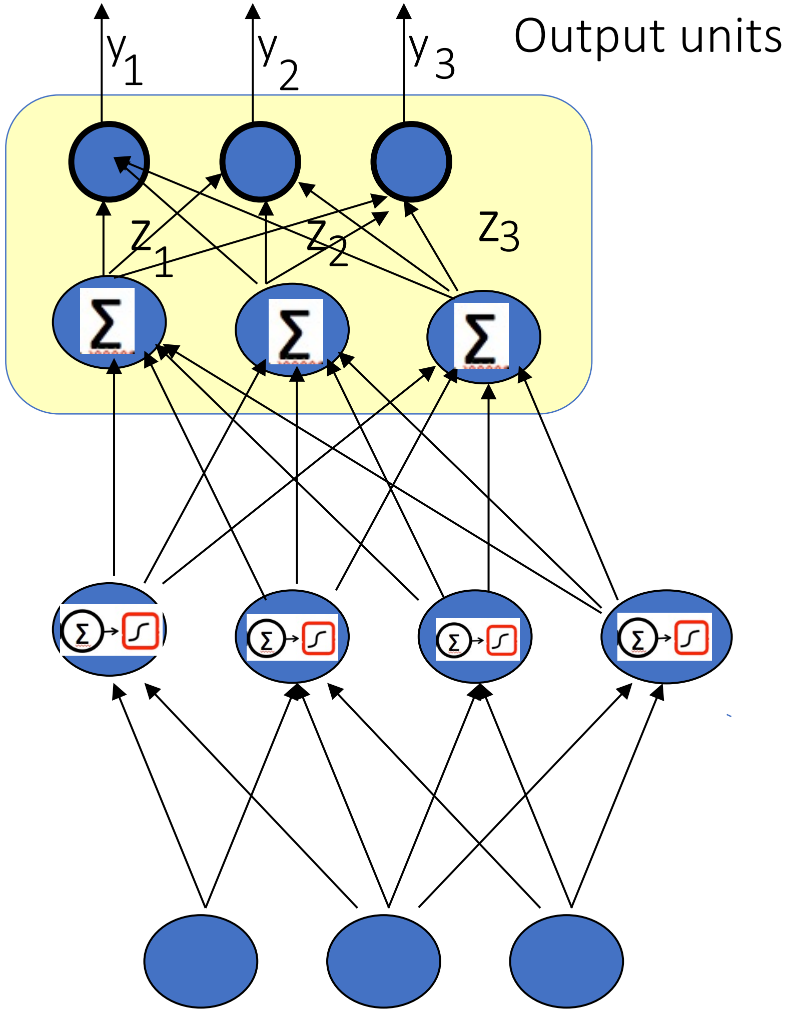
$$E_W(\hat{y},y) = \text{cross-E} = -\sum_{j=1...K} y_j \ln \hat{y}_j$$

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

$$\underset{w}{\operatorname{argmin}}\left\{-\sum_{k=1}^{K} y_k \log \hat{y}_k\right\} = \left\{-\log \hat{y}_{true\ class}\right\} \Rightarrow \operatorname{argmax}_w \left\{\log \hat{y}_{true}\right\}$$

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

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



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

y_{true} $\hat{y}$

"0" for all except true class

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

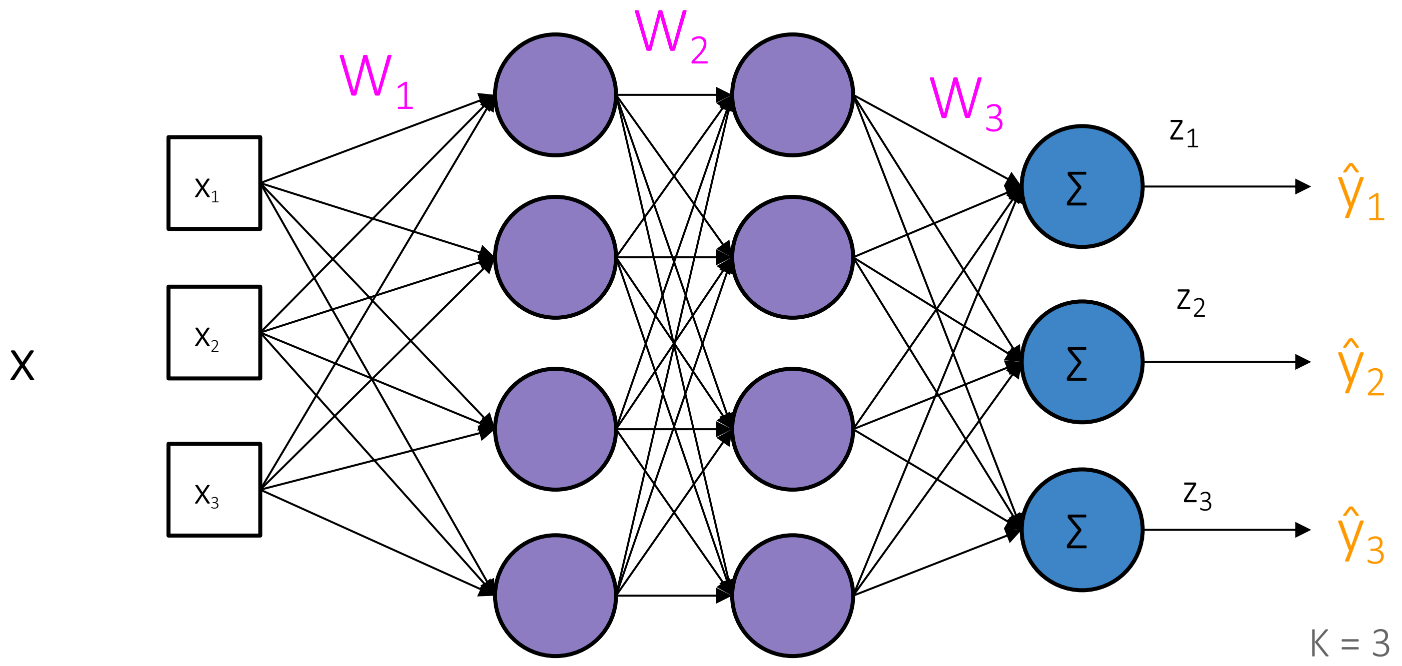
MLE / the negative log probability of the right answer / Cross entropy loss function :

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

Error calculated from predicted Output vs. true

Summary Recap: Multi-Class Classification Loss

Cross Entropy Loss



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

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

$$y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \hat{y} = \begin{pmatrix} 0.1 \\ 0.7 \\ 0.2 \end{pmatrix}$$

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

"0" for all except true class

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 z1 and z0 has no effect.
  - The over-parameterization of the softmax is because the probabilities must add to 1.

Today: Basic Neural Network Models

Data: X
↓
Task: y
↓
Representation: : x, f()
↓
Score Function: L()
↓
Search/Optimization : argmin()
↓
Models, Parameters, metrics

X: Tabular / 2D / 1D / 3D / Graph /…
↓
Classification / Regression
↓
Multilayer Network topology
↓
neg Log-likelihood , Cross-Entropy / MSE / Many more
↓
SGD / Backprop
↓
Weights and biases in NN Layers / Accuracy / F1


Thank You

UVA CS 4774: Machine Learning

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

Dr. Yanjun Qi

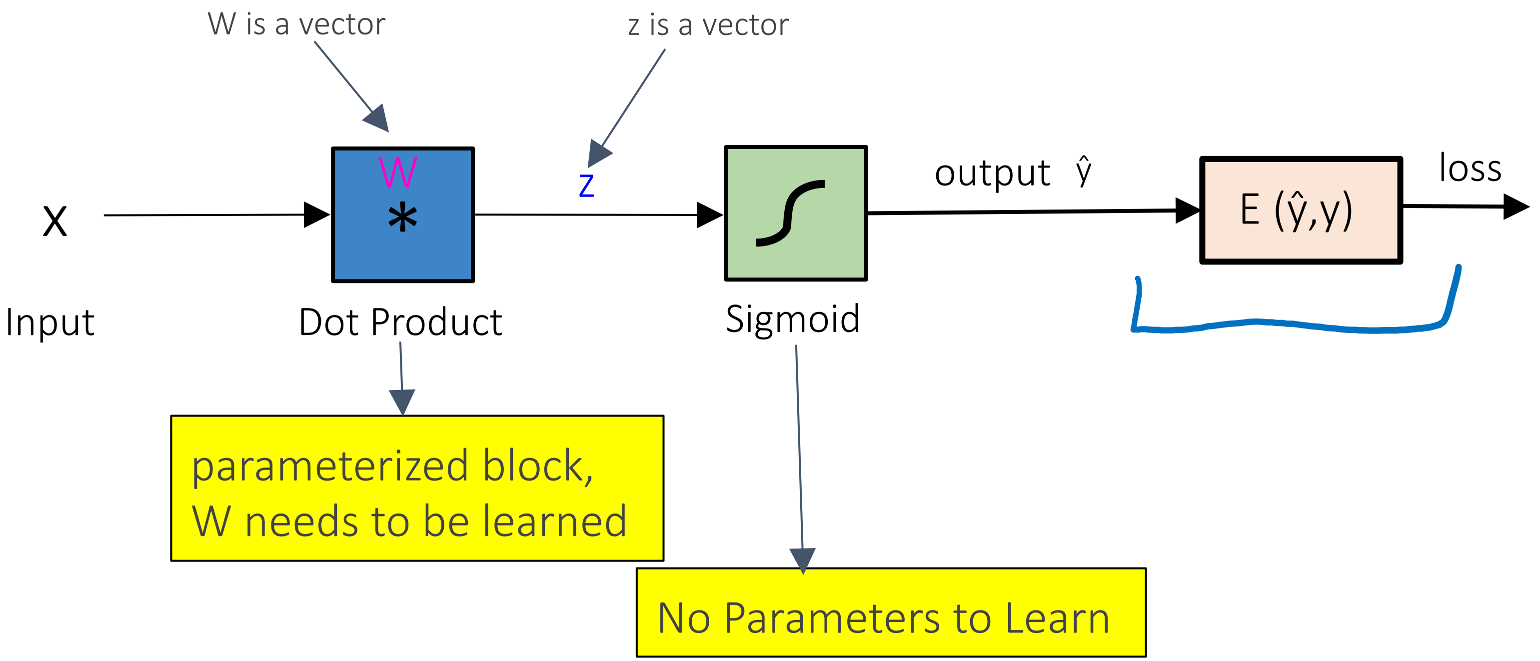
University of Virginia
Department of Computer Science

Module III

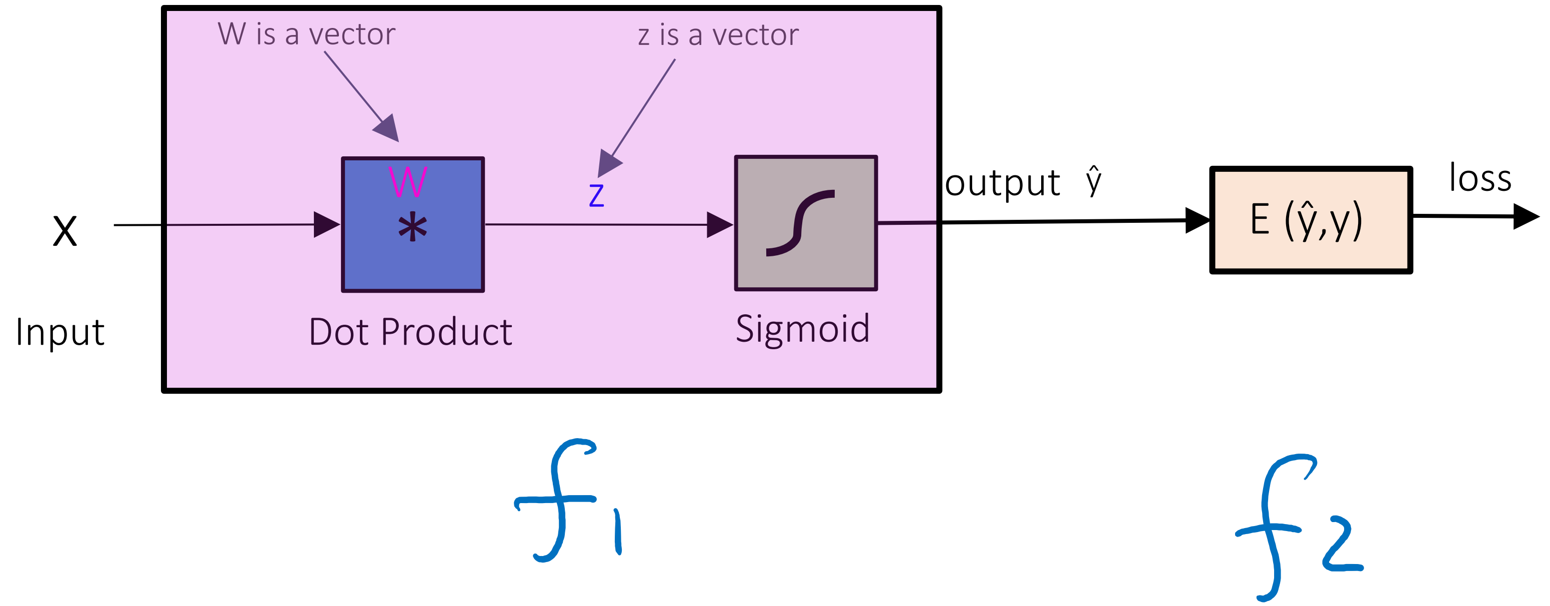
Roadmap: DNN Basics

- 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

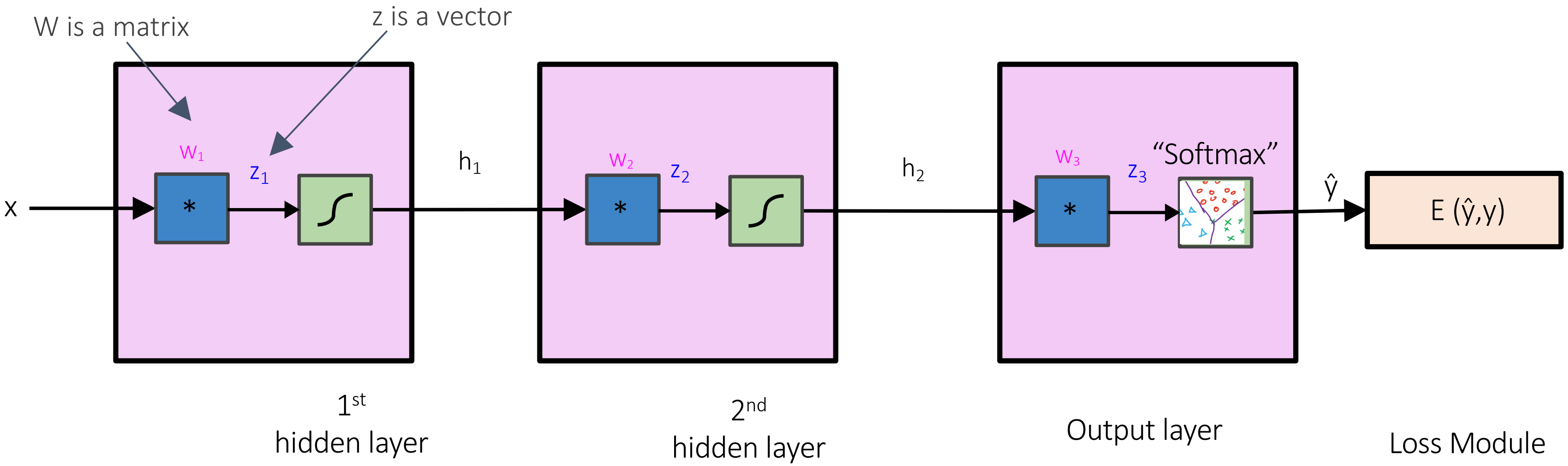
e.g., “Block View” of Logistic Regression



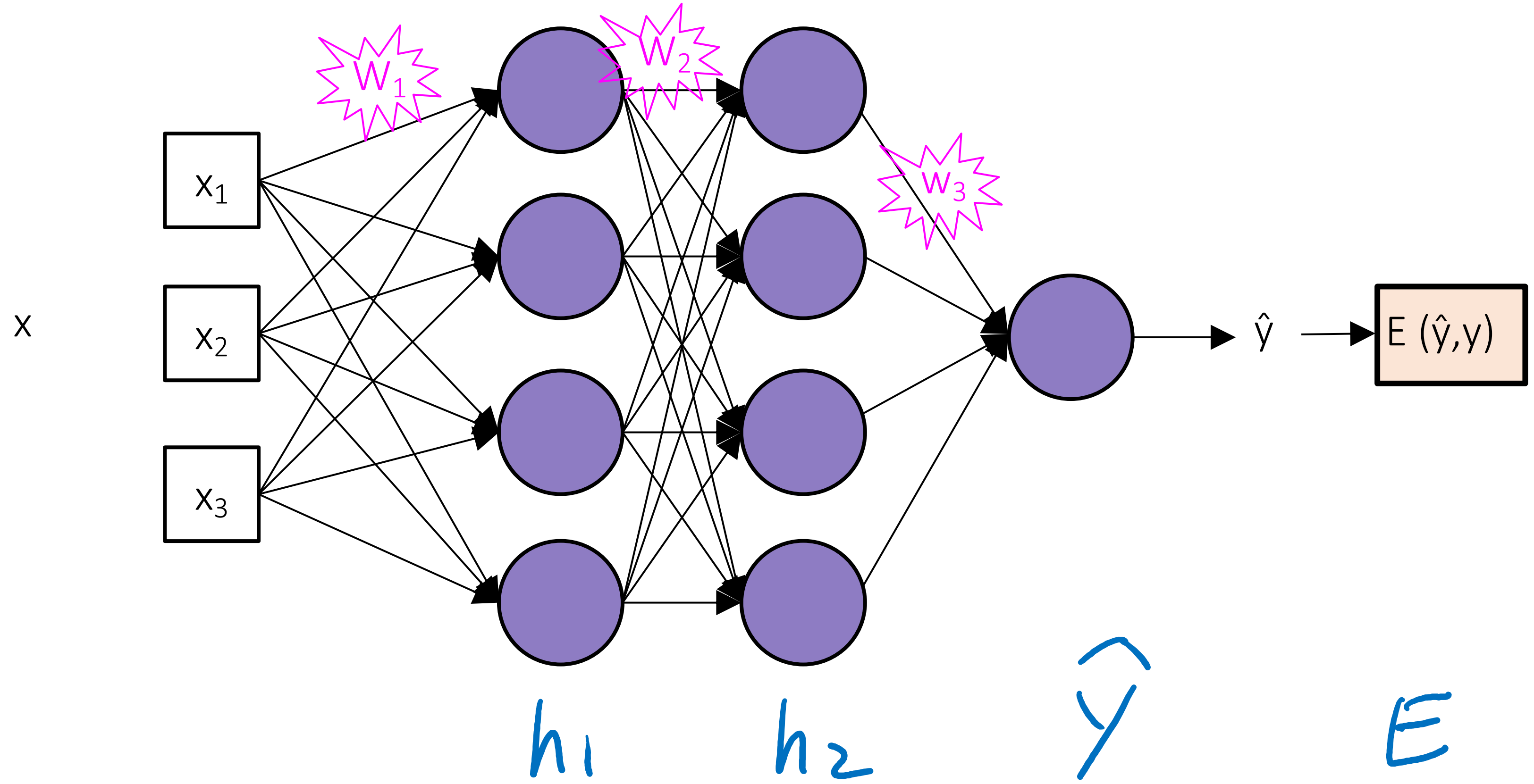
e.g., "Block View" of Logistic Regression



e.g., “Block View” of multi-class NN



$x \xrightarrow{W_1} h_1 \xrightarrow{W_2} h_2 \xrightarrow{W_3} \hat{y} \longrightarrow E(\hat{y}, y)$


x
x_1
x_2
x_3
W_1
W_2
W_3
ŷ
E (ŷ,y)
h_1
h_2
ŷ
E

Review: Stochastic GD ➔

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

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

$E(\hat{y}, y)$

- ➔ For neural network, we have the delta rule

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

$W = \{w^1, w^2, \dots, w^T\}$

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

Backpropagation

- 1. Initialize network with random weights
- 2. For 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))

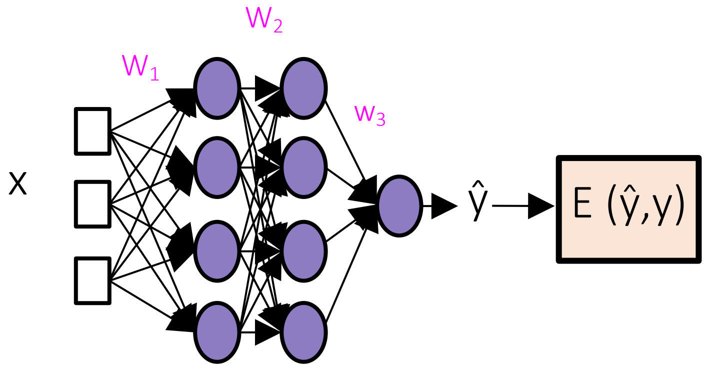
  - **Backward:** For all layers (starting with the output layer, back to input layer):
    - Propagate local gradients layer by layer from final layer, until back to input layer to calculate each layer's gradient $\frac{\partial E}{\partial {W_l}^t}$ → Need to calculate these!
    - Adapt weights in current layer $${W_l}^{t+1} = {W_l}^t - \eta \frac{\partial E}{\partial {W_l}^t}$$

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:

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



But how do we get gradients of lower layers?

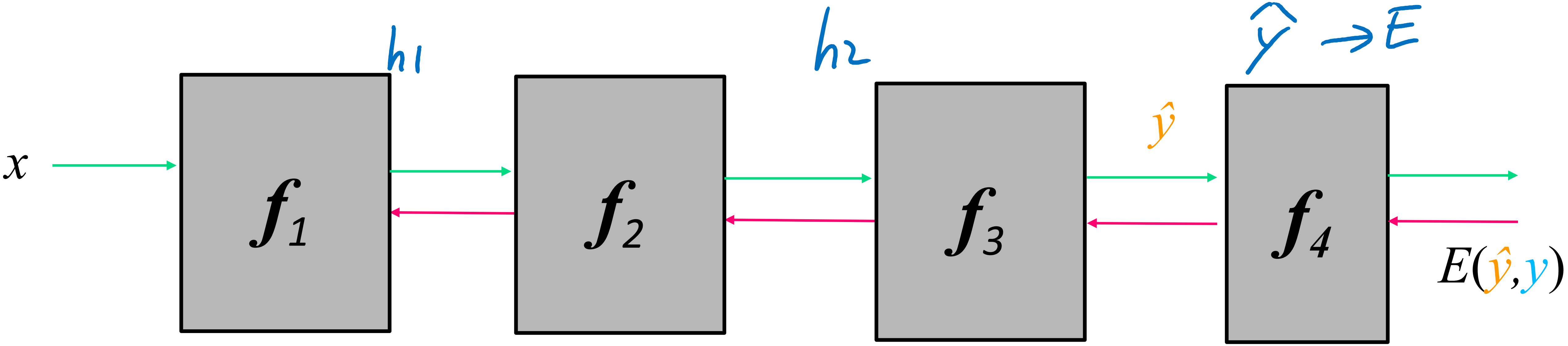
- Backpropagation!
  - Repeated application of chain rule of calculus
  - Locally minimize the objective
  - Requires all “blocks” of the network to be differentiable

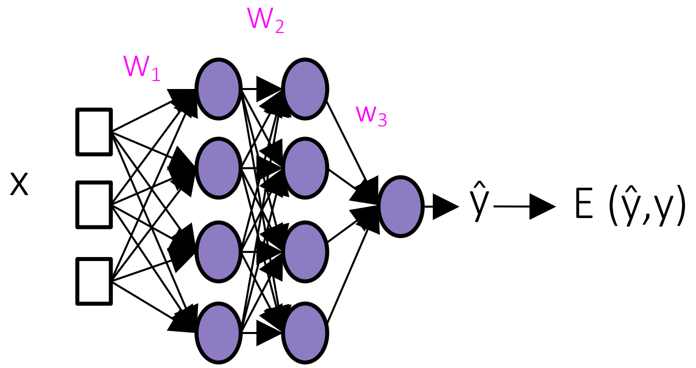
LeCun et. al. Efficient Backpropagation. 1998

Layers of Differentiable Parameterized Functions (with nonlinearities)

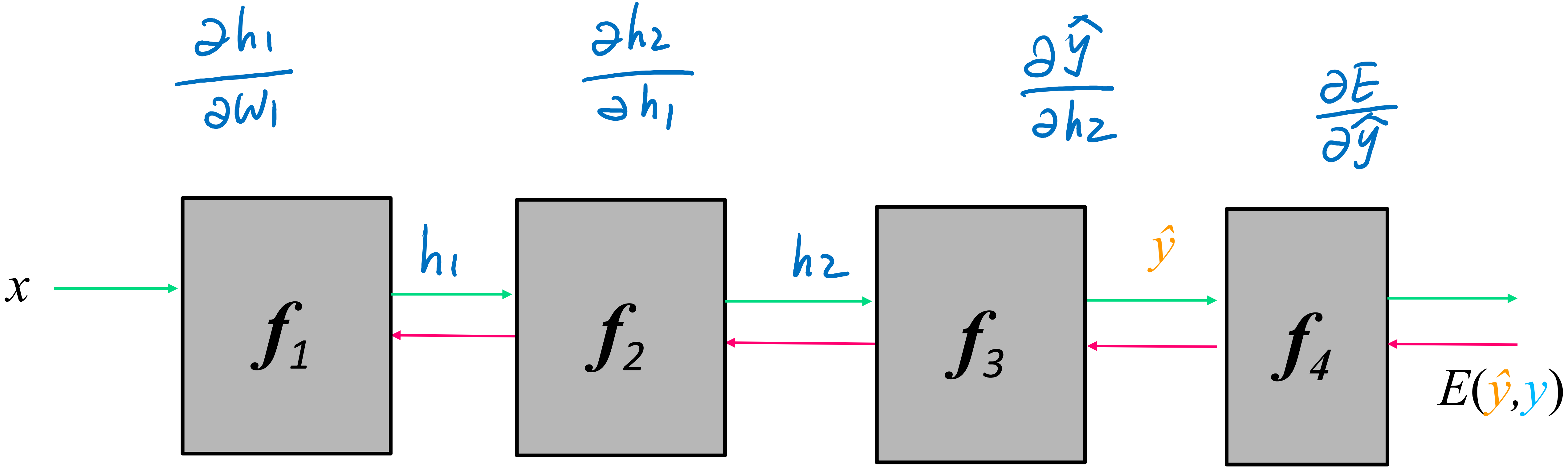
Forward: Feed inputs to network layer by layer, and calculate output of each layer (from input layer to until the final layer (error function))

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

Training Neural Networks by Backpropagation - to jointly optimize all parameters

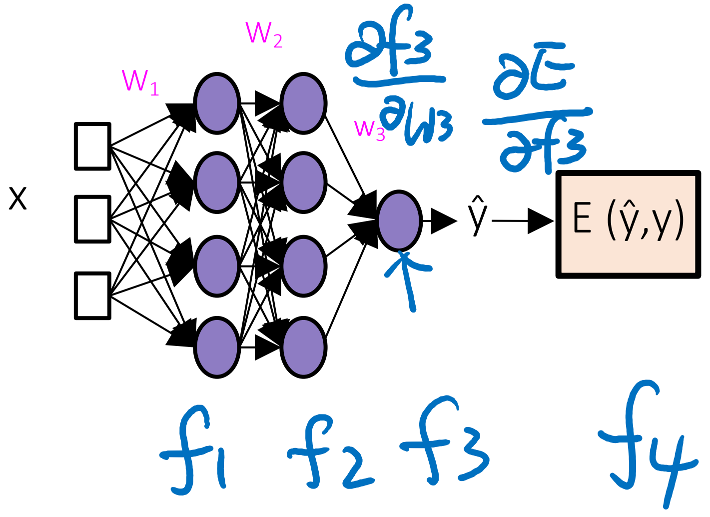
How do we learn the optimal weights W_L for our task??

- Stochastic Gradient descent:

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

But how do we get gradients of lower layers?

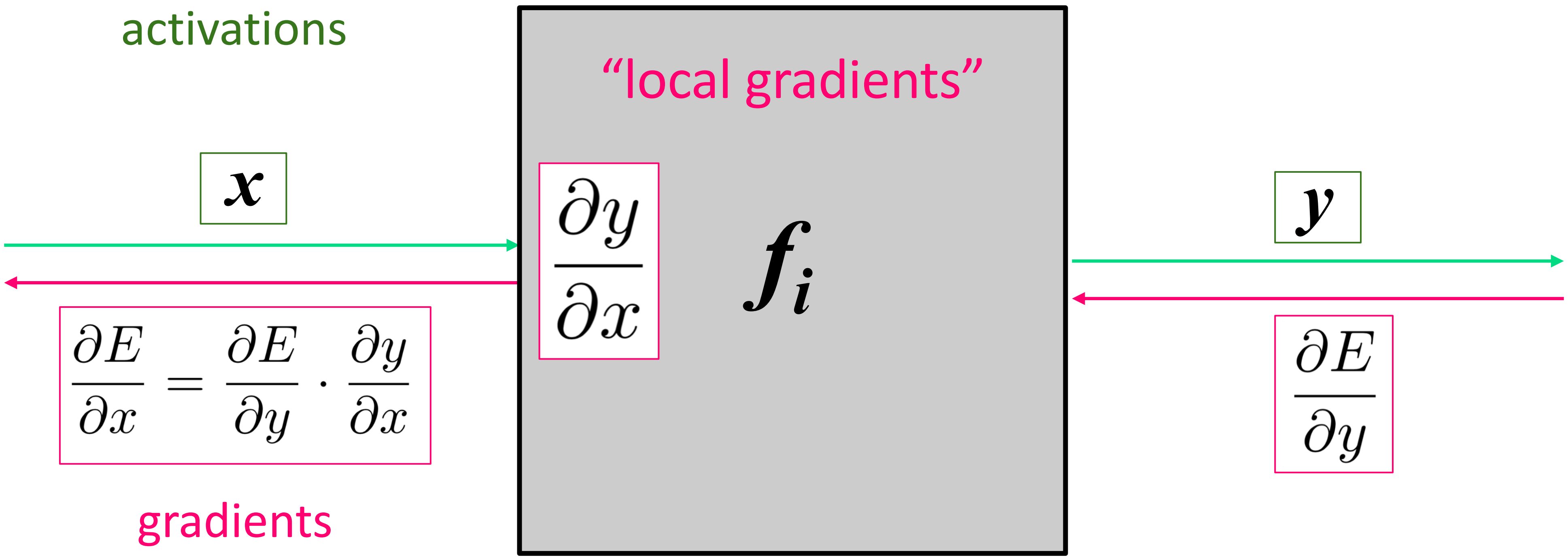
- Backpropagation!
  - Repeated application of chain rule of calculus
  - Locally minimize the objective
  - Requires all "blocks" of the network to be differentiable



$$E(x, y) = f_4(y, f_3(f_2(f_1(x))))$$

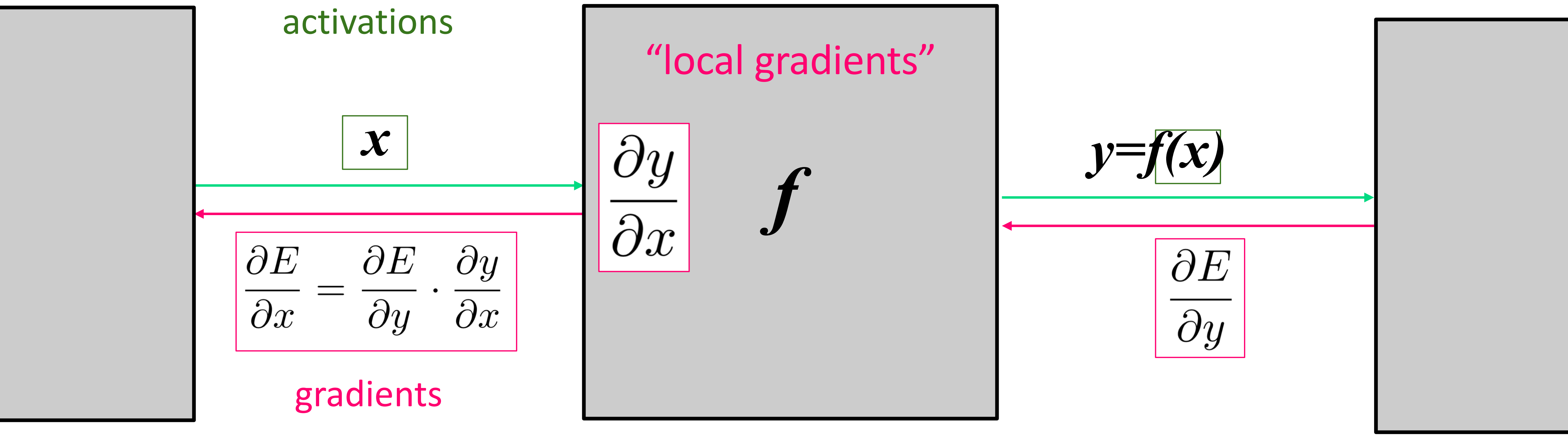
W_3 W_2 W_1

“Local-ness” of Backpropagation



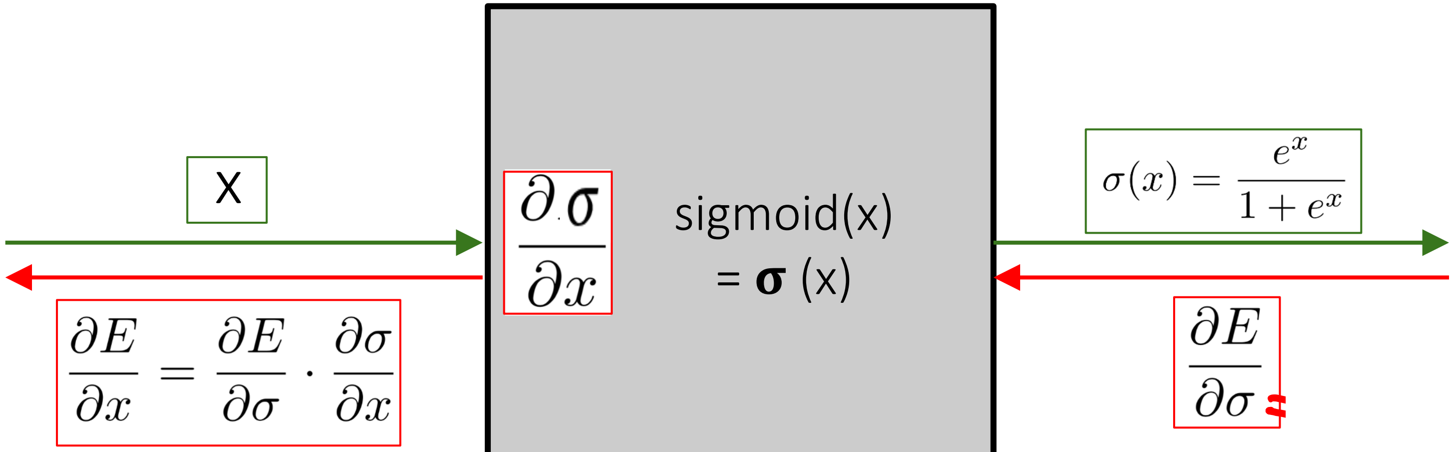
http://cs231n.stanford.edu/slides/winter1516_lecture5.pdf

“Local-ness” of Backpropagation



http://cs231n.stanford.edu/slides/winter1516_lecture5.pdf

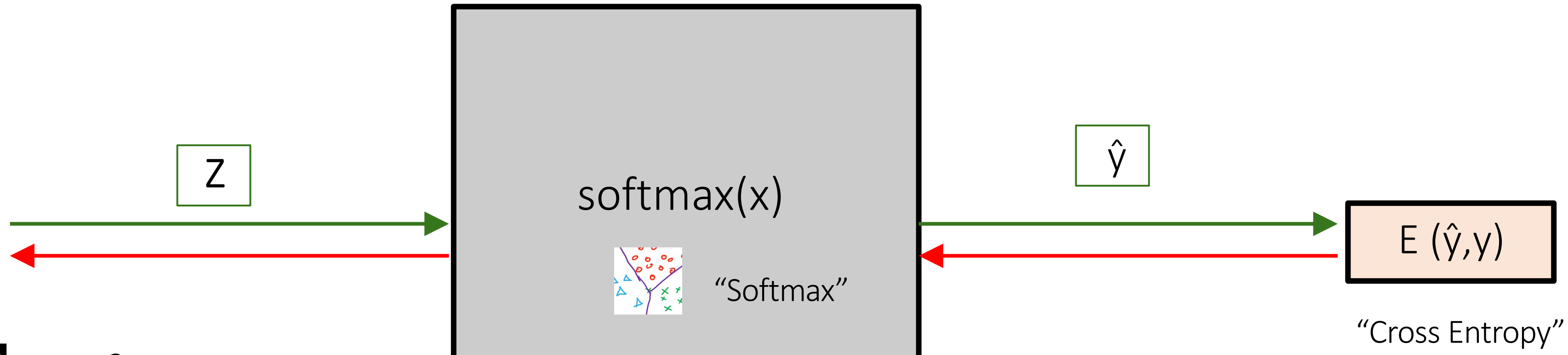
Example: Sigmoid Block



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

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

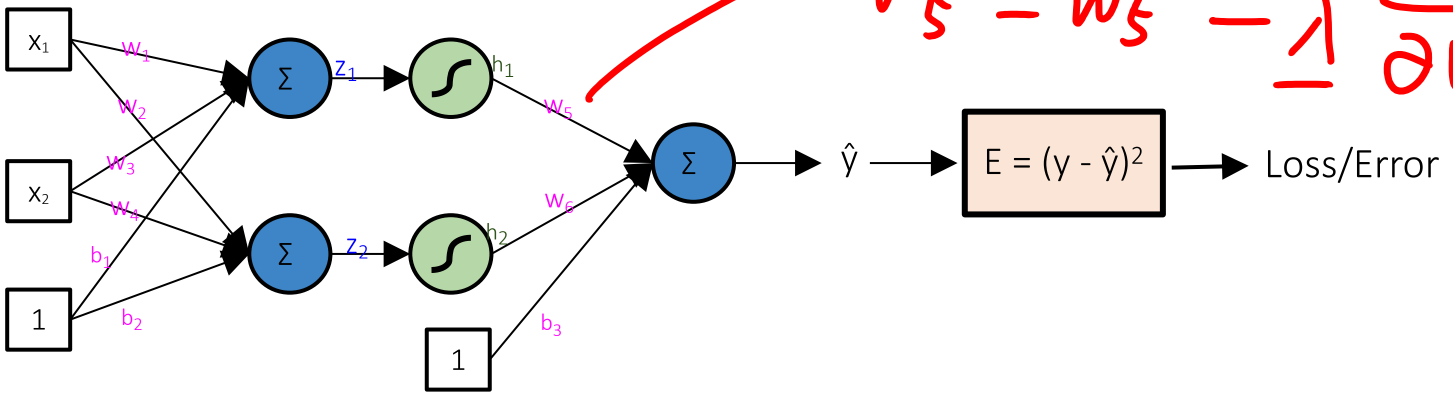
Example: Softmax Block (right before loss layer)



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

Backpropagation Example

$$w_5^t = w_5^{t-1} - \lambda \frac{\partial E}{\partial w_5}$$



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

argmin_w $\{ f_4 (f_3 (f_2 (f_1 ( )))) \}$

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

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

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

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

Extra

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

Input

Output

Local Gradients=∂Output /∂Input

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

Extra

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

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

Input	Output	Local Gradients=∂Output /∂Input
$x_1, x_2, w_1, \ldots$	z_1, z_2	$\frac{\partial z_1}{\partial x_1} = w_1$
z_1, z_2	h_1, h_2	$\frac{\partial h_1}{\partial z_1} = h_1(1-h_1)$
w_5, h_1, h_2	$\hat{y}$	$\partial \hat{y} / \partial h_1 = w_5$
$\hat{y}$	loss E	$\partial E / \partial \hat{y} = -2(y-\hat{y})$

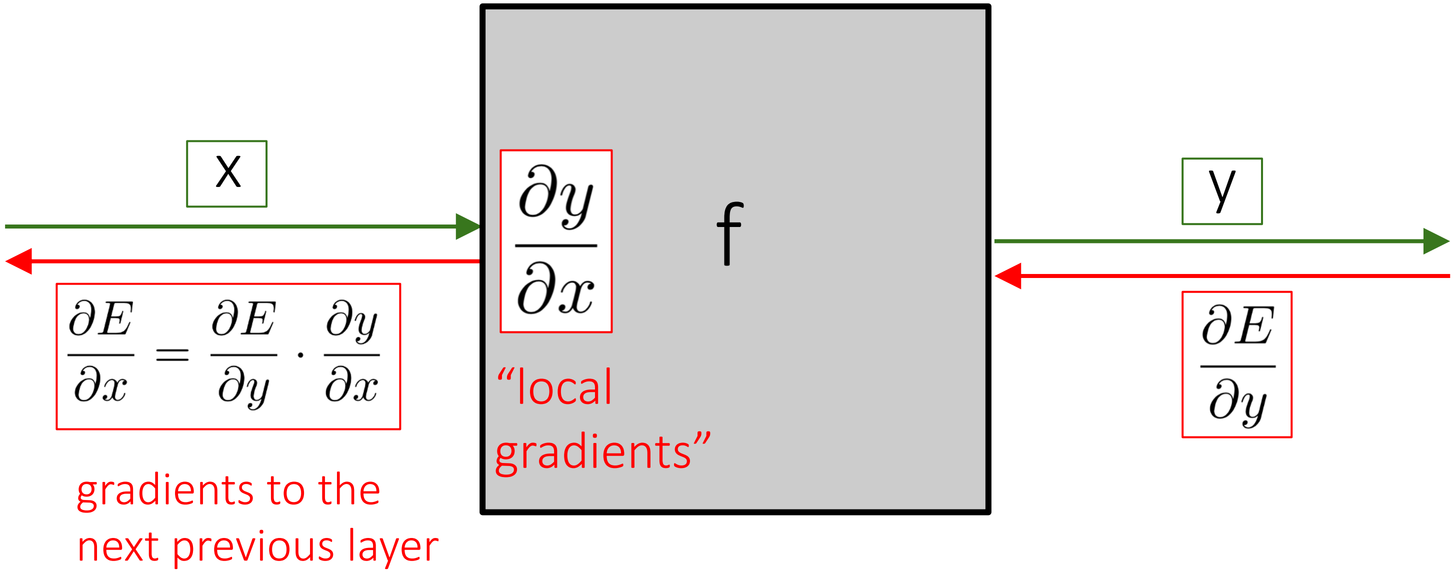
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial w_1} = \frac{\partial f_4}{\partial f_3} \frac{\partial f_3}{\partial f_2} \frac{\partial f_2}{\partial f_1} \frac{\partial f_1}{\partial w_1}$$

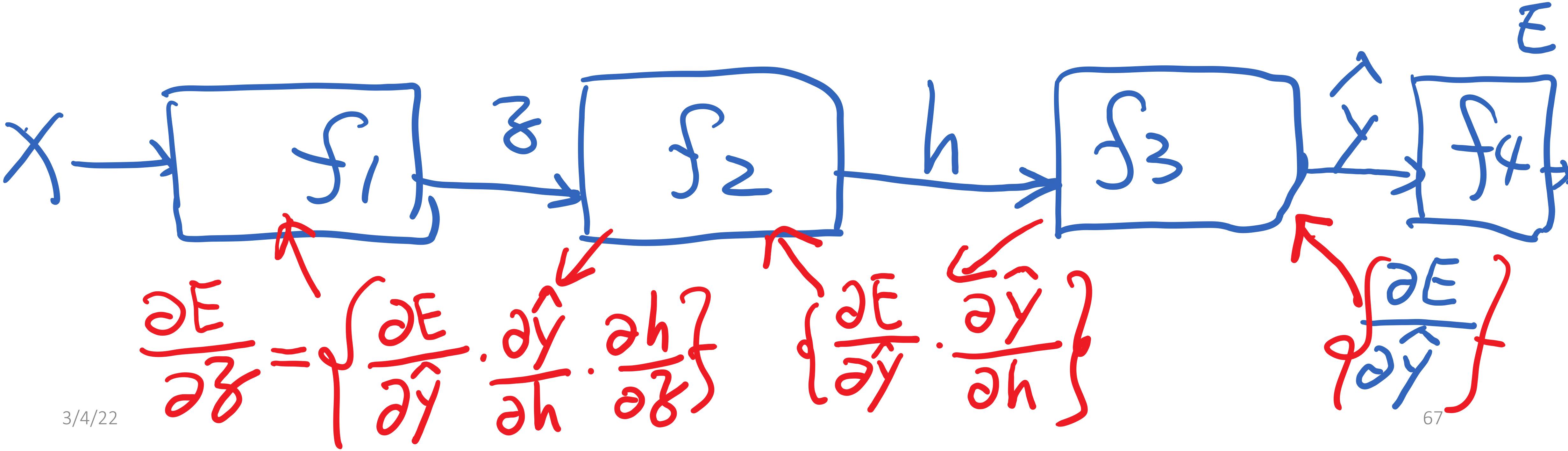
$$= -2(y-\hat{y}) \frac{\partial (h_1 w_5 + h_2 w_6 + b_3)}{\partial w_1}$$

$$= -2(y-\hat{y}) \left(w_5 \frac{\partial h_1}{\partial w_1} + w_6 \frac{\partial h_2}{\partial w_1}\right)$$

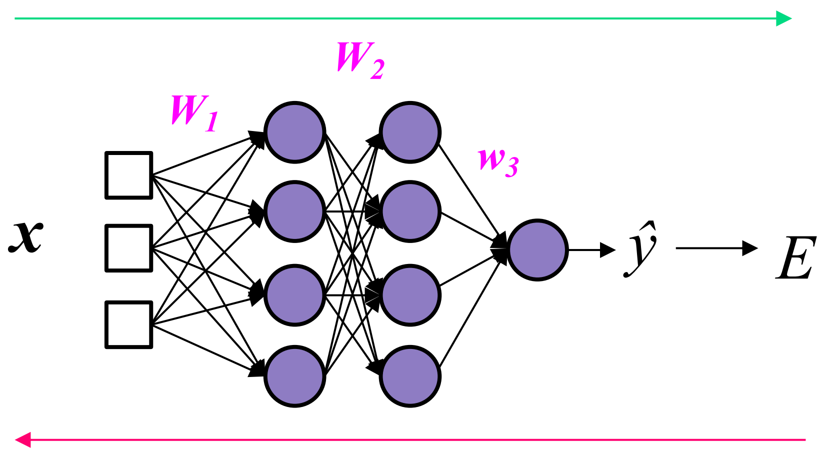
$$= -2(y-\hat{y}) w_5 \frac{\partial h_1}{\partial w_1} = -2(y-\hat{y}) w_5 \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

$$= \underbrace{-2(y-\hat{y})}_{f_4 \text{ local}} \ \underbrace{w_5}_{f_3 \text{ local}} \ \underbrace{h_1(1-h_1)}_{f_2 \text{ local}} \ \underbrace{x_1}_{\frac{\partial f_1}{\partial w_1}}$$


x
y
f
$\frac{\partial y}{\partial x}$
"local gradients"
$\frac{\partial E}{\partial y}$
$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial x}$
gradients to the next previous layer

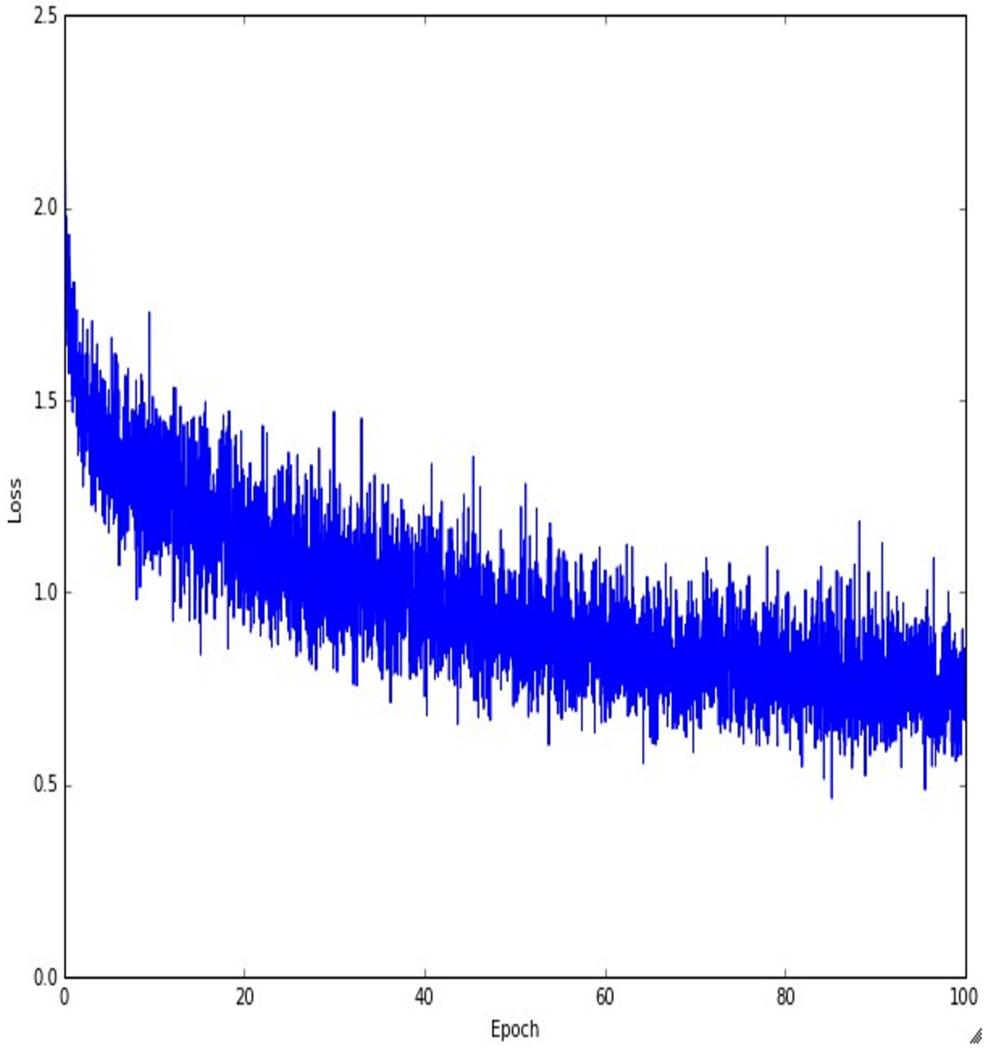

x
f_1
z
f_2
h
f_3
$\hat{y}$
f_4
E
$\frac{\partial E}{\partial z} = \left\{\frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} \cdot \frac{\partial h}{\partial z}\right\}$
$\left\{\frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h}\right\}$
$\left\{\frac{\partial E}{\partial \hat{y}}\right\}$

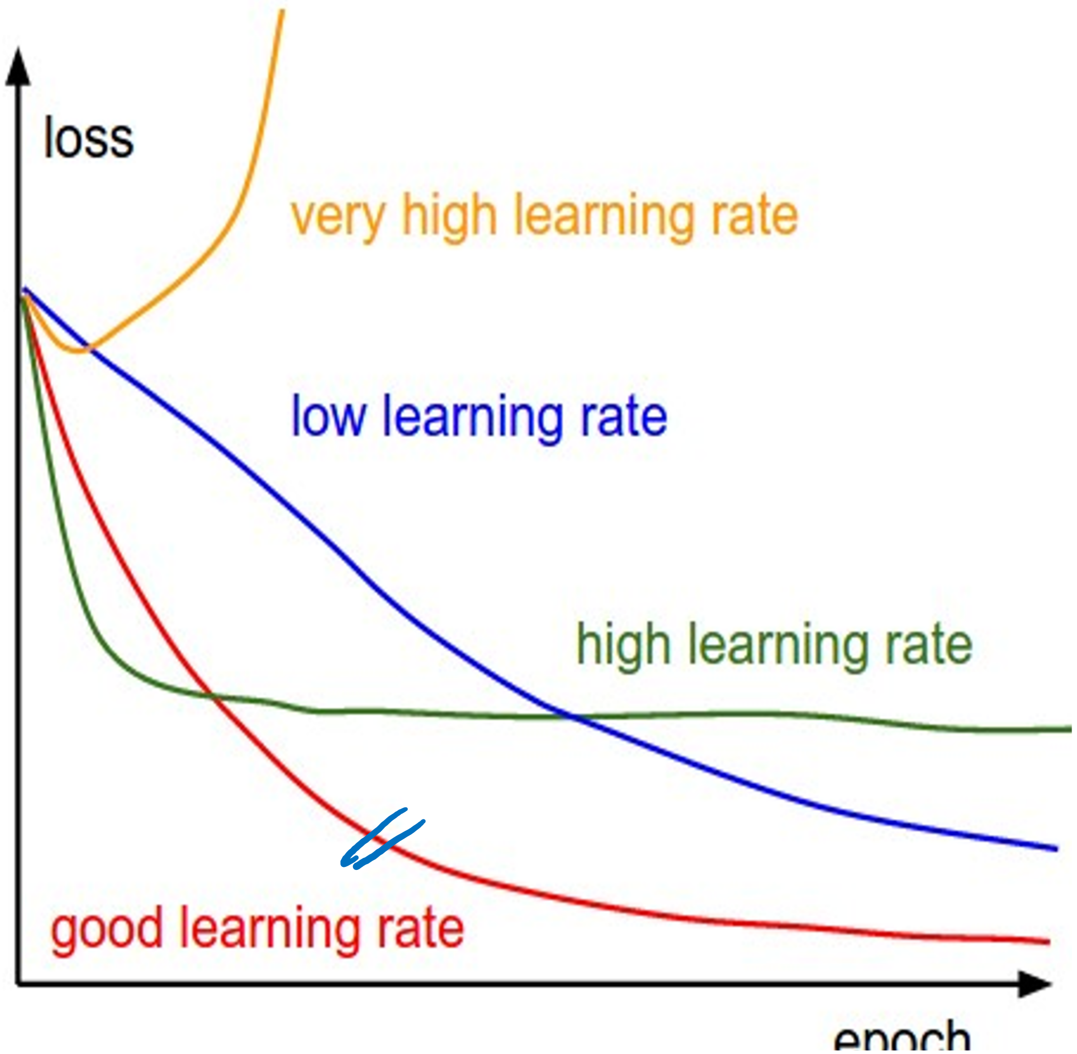
BackProp in Practice: Mini-batch SGD



1. Initialize weights
2. For each batch of input samples $S\boldsymbol{x}$:
   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

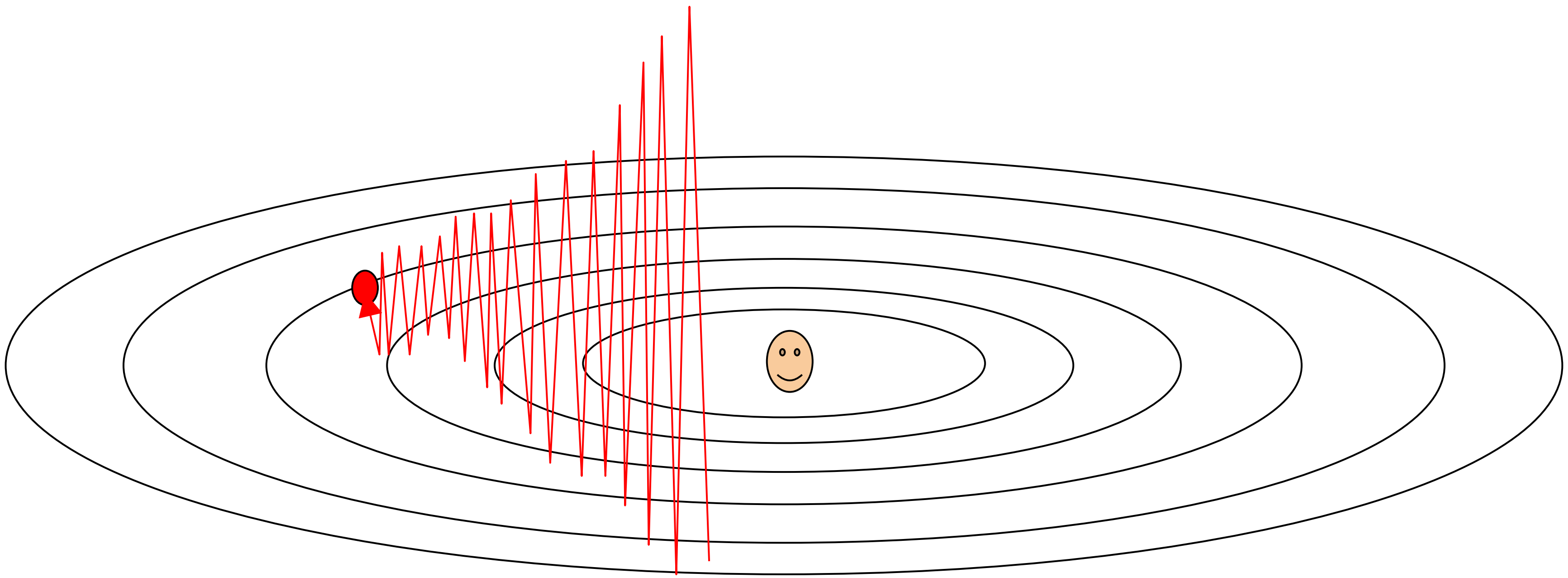
Monitor and visualize the loss curve





From Feifei Li Stanford Cousre

Gradient Magnitudes:

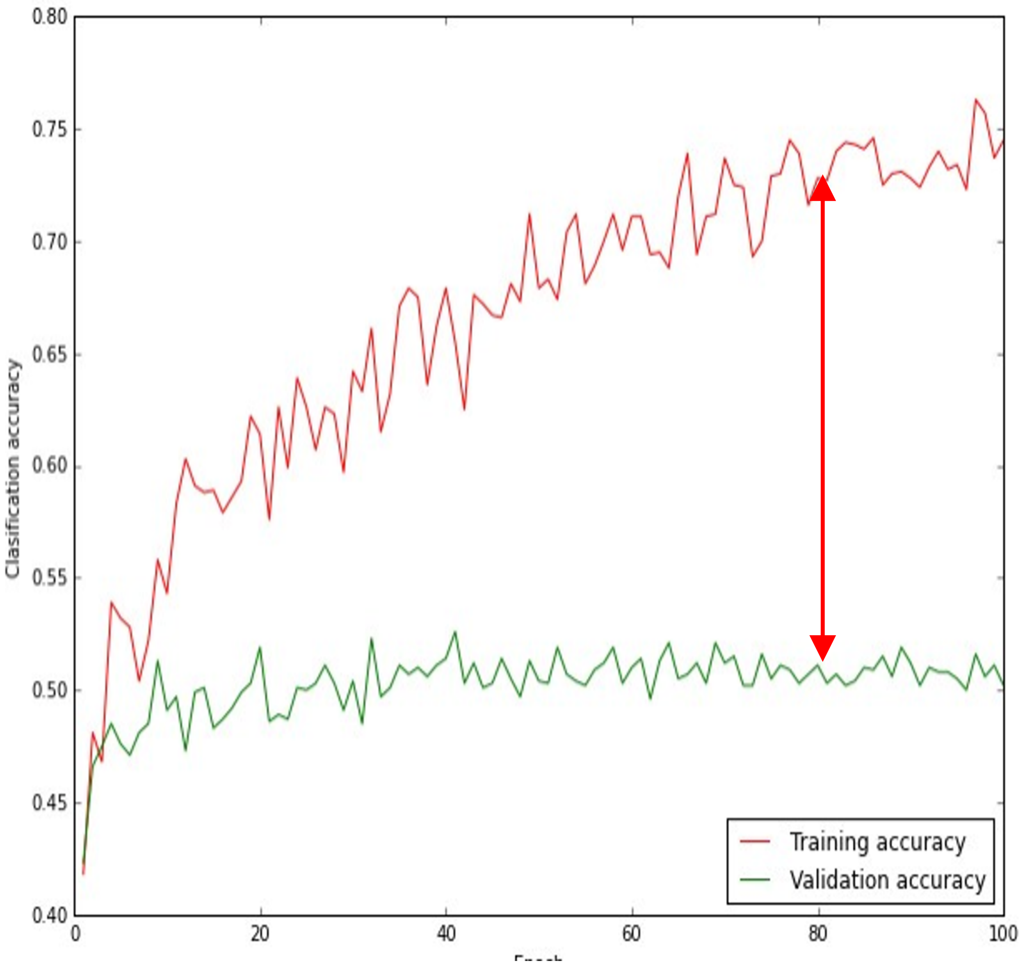


Gradients too big → divergence
Gradients too small → slow convergence

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?

From Feifei Li Stanford Cousre

Extra

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

From Feifei Li Stanford Cousre

Extra

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

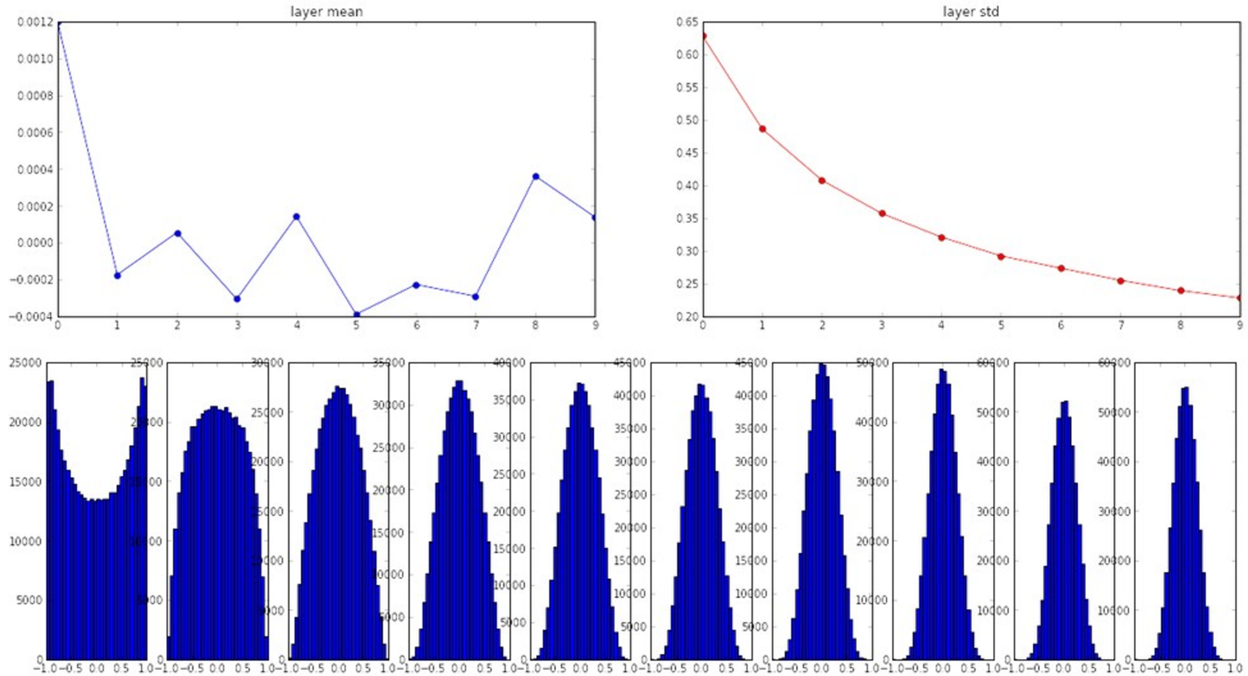
Extra

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)

Extra

Batch Normalization: implicit regularization

[Ioffe and Szegedy, 2015]

Normalize:

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

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

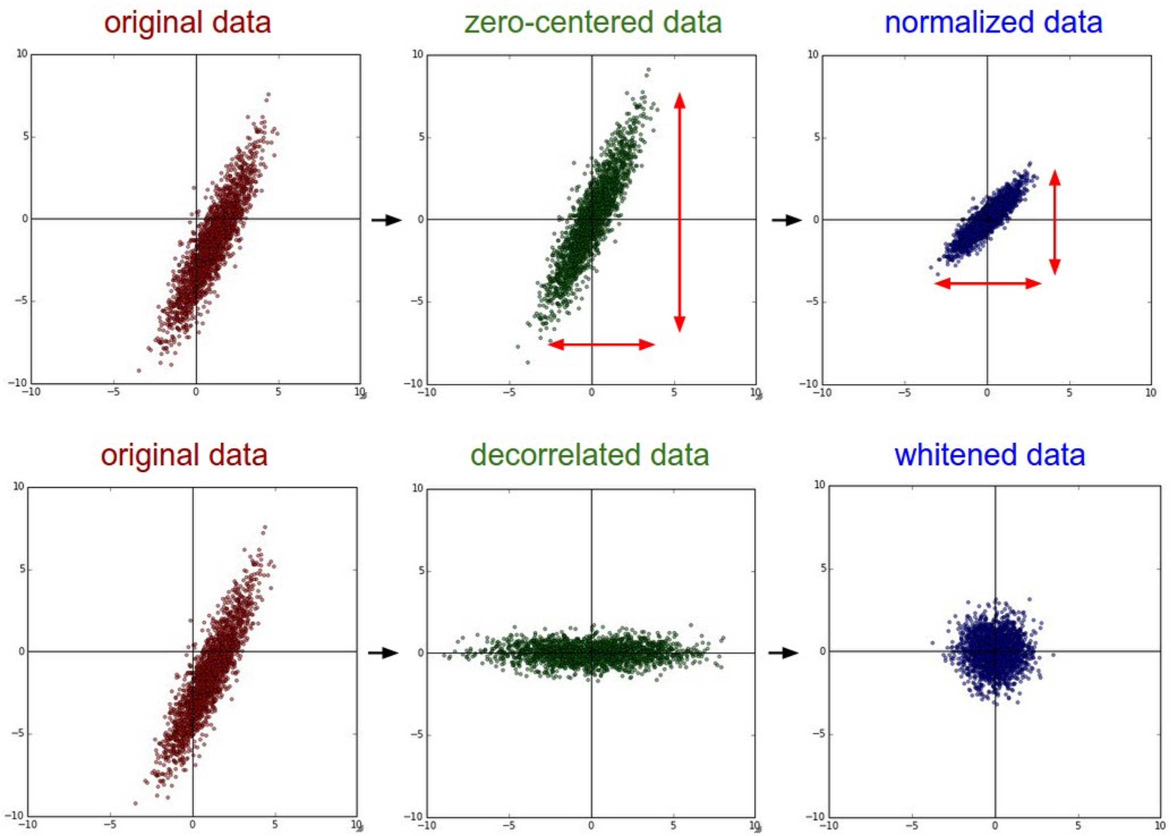
$$y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)}$$

- 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

Standardizing the activations of the prior layer means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update will not change, at least not dramatically. This has the effect of stabilizing and speeding-up the training process of deep neural networks.

Extra

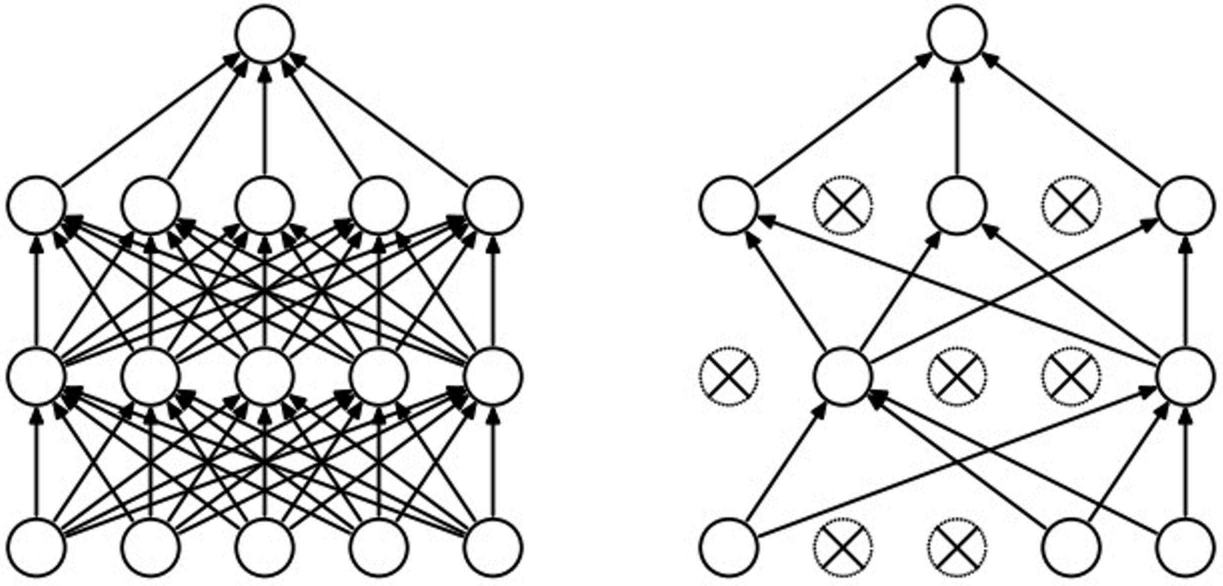
Data Preprocessing



From Feifei Li Stanford Cousre

Regularization by Dropout

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



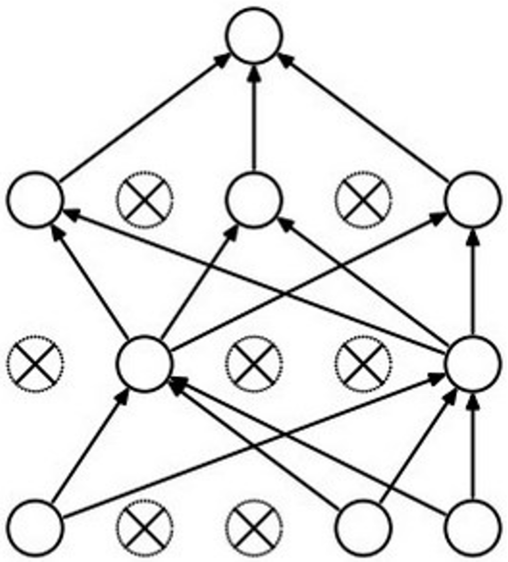
(a) Standard Neural Net (b) After applying dropout.

[Srivastava et al., 2014]

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

From Feifei Li Stanford Course

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

From Feifei Li Stanford Course

Today: Basics of Neural Network Models

Data: X

Task: y

Representation: : x, f()

Score Function: L()

Search/Optimization : argmin()

Models, Parameters, metrics

X: Tabular / 2D / 1D / 3D / Graph /…

Classification / Regression

Multilayer Network topology

neg Log-likelihood , Cross-Entropy / MSE / Many more

SGD / Backprop

Weights and biases in NN Layers

/ Accuracy / F1


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

UVA CS 4774: Machine Learning

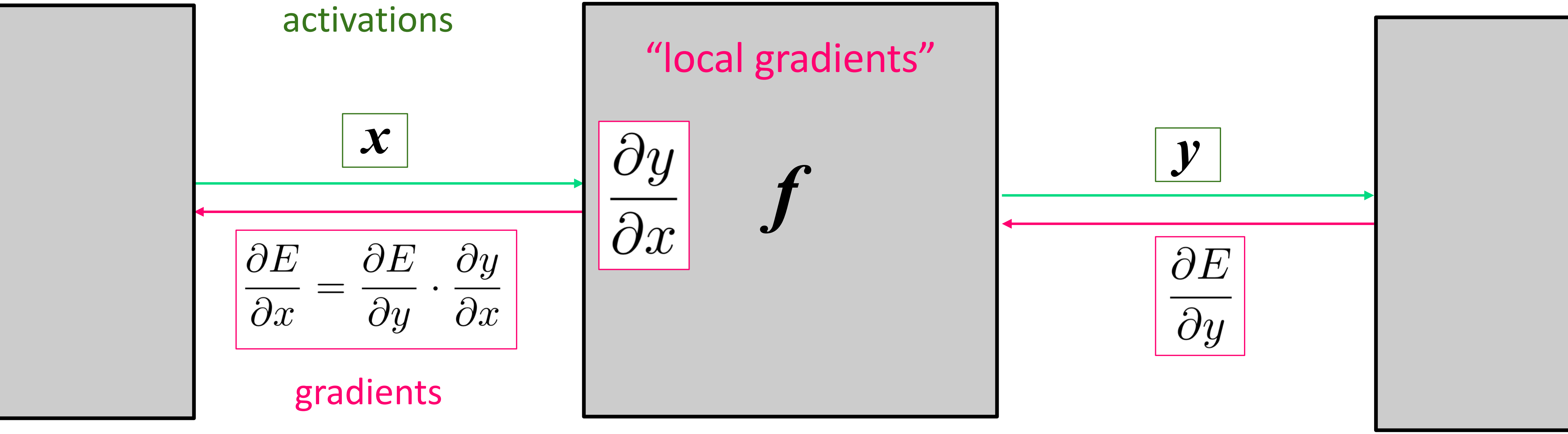
Lecture 12: Neural Network (NN) and More: BackProp

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Module IV

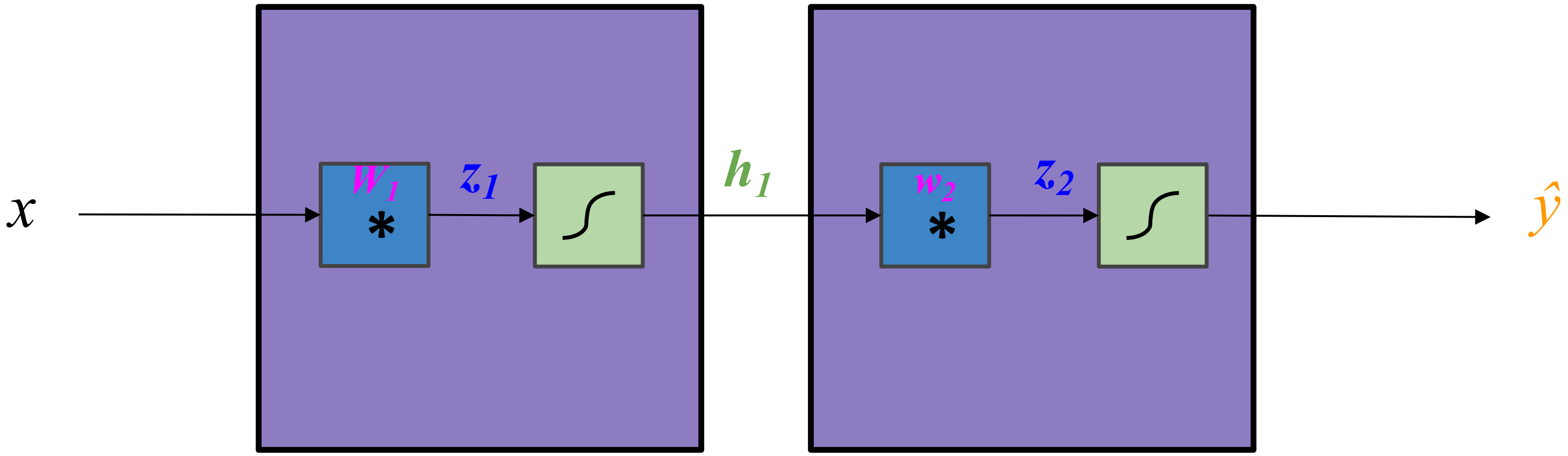
“Local-ness” of Backpropagation



http://cs231n.stanford.edu/slides/winter1516_lecture5.pdf

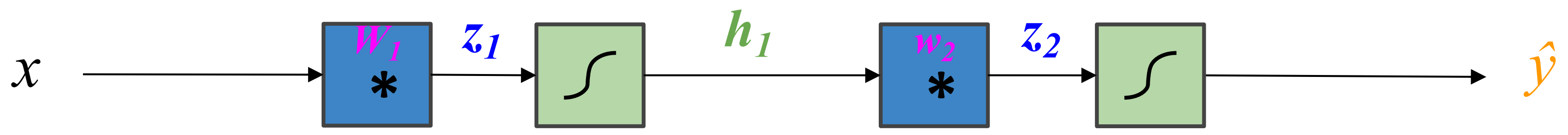
Backpropagation

(binary classification example)



Backpropagation

(binary classification example)



$$E = \text{loss} = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

Gradient Descent to Minimize loss:

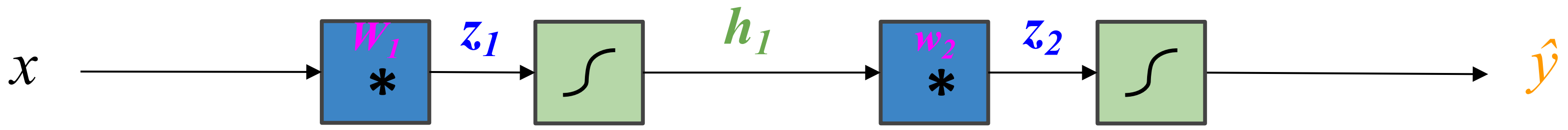
$$\mathbf{w}_2(t+1) = \mathbf{w}_2(t) - \eta \frac{\partial E}{\partial \mathbf{w}_2(t)}$$

$$W_1(t+1) = W_1(t) - \eta \frac{\partial E}{\partial W_1(t)}$$

Need to find these!

Backpropagation

(binary classification example)



$$E = f_4 = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = f_3 = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = f_3 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

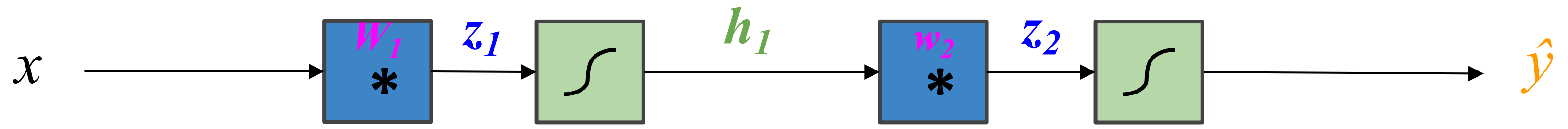
$$\boldsymbol{h}_1 = f_2 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = f_1 = W_1^T \boldsymbol{x}$$

$$E = f_4(f_3(f_2(f_1(x))))$$

Backpropagation

(binary classification example)



$$E = f_4 = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = f_3 = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = f_3 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = f_2 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = f_1 = W_1^T \boldsymbol{x}$$

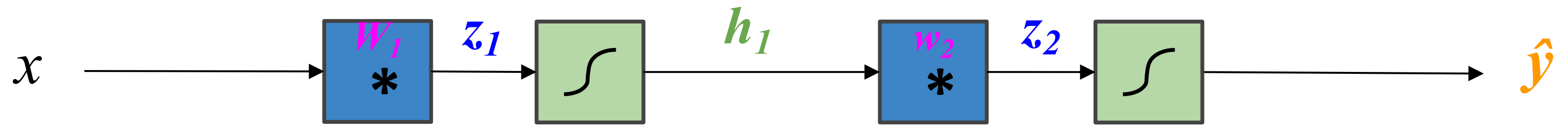
$$\frac{\partial E}{\partial \mathbf{w}_2} = ??$$

$$\frac{\partial E}{\partial \mathbf{W}_1} = ??$$

$$E = f_4(f_3(f_2(f_1(x))))$$

Backpropagation

(binary classification example)



$$E = f_4 = -y \ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = f_3 = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = f_3 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = f_2 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = f_1 = W_1^T \boldsymbol{x}$$

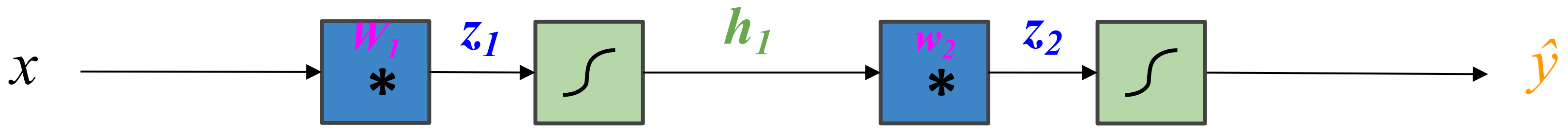
$$\frac{\partial E}{\partial \mathbf{w}_2} = ??$$

$$\frac{\partial E}{\partial \mathbf{W}_1} = ??$$

$$E = f_4(f_3(f_2(f_1(x))))$$ → Exploit the chain rule!

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

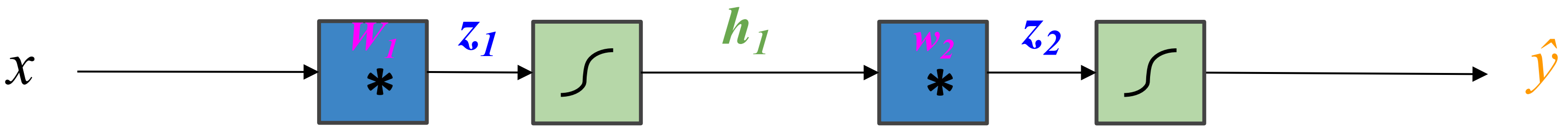
$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

$$\frac{\partial E}{\partial \boldsymbol{w}_2} =$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

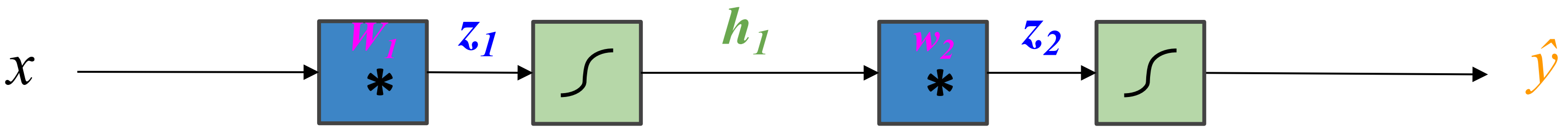
$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

chain rule

$$\frac{\partial E}{\partial \boldsymbol{w}_2} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \boldsymbol{w}_2}$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = \frac{e^{\boldsymbol{z}_1}}{1+e^{\boldsymbol{z}_1}}$$

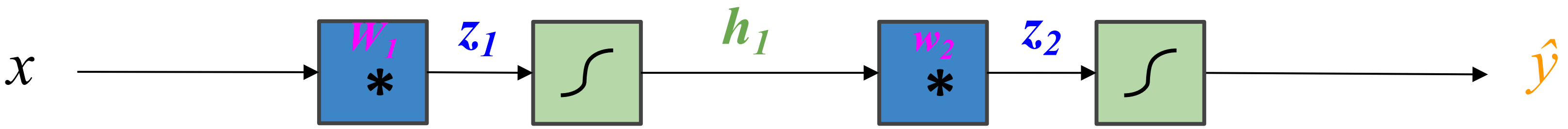
$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

$$\frac{\partial E}{\partial \boldsymbol{w}_2} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \boldsymbol{w}_2}$$

$$= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

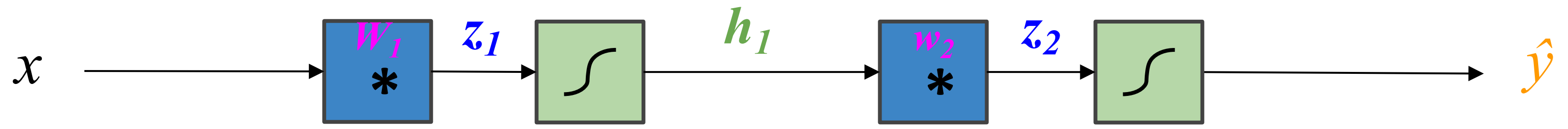
$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

$$\frac{\partial E}{\partial \boldsymbol{w}_2} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \boldsymbol{w}_2}$$

$$= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot \left(\frac{e^{z_2}}{1+e^{z_2}}\left(1 - \frac{e^{z_2}}{1+e^{z_2}}\right)\right) \cdot$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

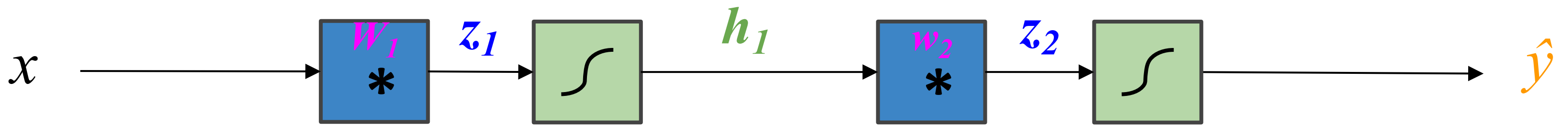
$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

$$\frac{\partial E}{\partial \boldsymbol{w}_2} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \boldsymbol{w}_2}$$

$$= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot \left(\frac{e^{z_2}}{1+e^{z_2}}\left(1 - \frac{e^{z_2}}{1+e^{z_2}}\right)\right) \cdot (\boldsymbol{h}_1)$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

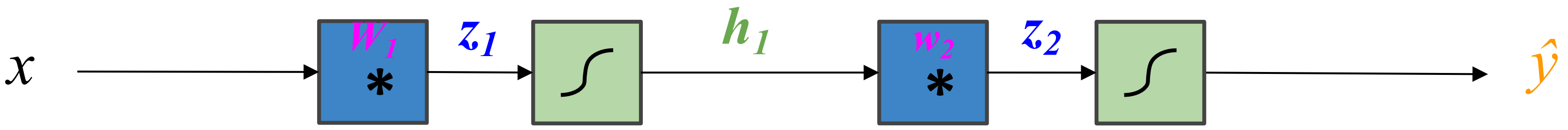
$$\frac{\partial E}{\partial \boldsymbol{w}_2} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \boldsymbol{w}_2}$$

$$= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot \left(\frac{e^{z_2}}{1+e^{z_2}}\left(1-\frac{e^{z_2}}{1+e^{z_2}}\right)\right) \cdot (\boldsymbol{h}_1)$$

$$= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot (\hat{y}(1-\hat{y})) \cdot (\boldsymbol{h}_1)$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

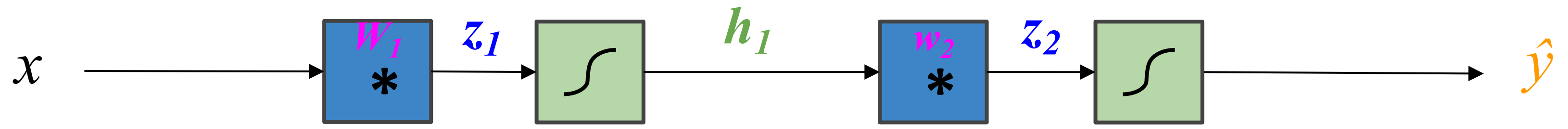
$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

$$\frac{\partial E}{\partial \boldsymbol{W}_1} =$$

Backpropagation

(binary classification example)



$$E = -\,y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

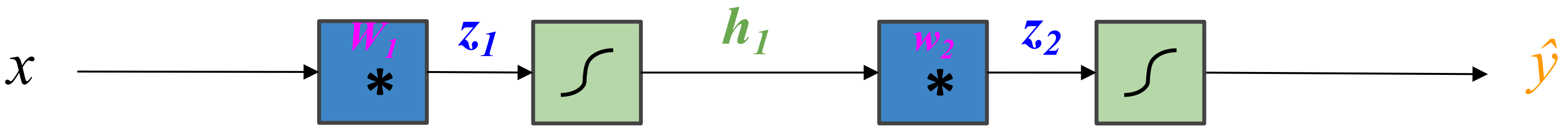
$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

$$\frac{\partial E}{\partial \boldsymbol{W}_1} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \boldsymbol{h}_1} \cdot \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{z}_1} \cdot \frac{\partial \boldsymbol{z}_1}{\partial W_1}$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = \frac{e^{\boldsymbol{z}_1}}{1+e^{\boldsymbol{z}_1}}$$

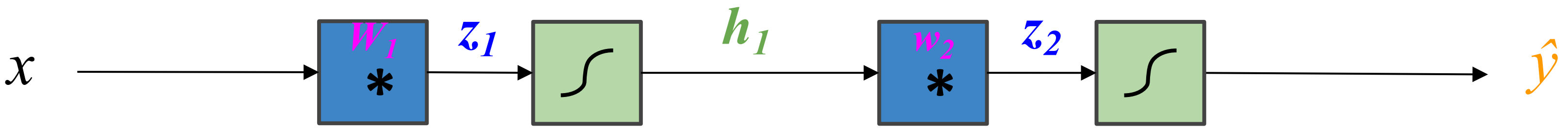
$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

$$\frac{\partial E}{\partial \boldsymbol{W}_1} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \boldsymbol{h}_1} \cdot \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{z}_1} \cdot \frac{\partial \boldsymbol{z}_1}{\partial W_1}$$

$$= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot (\hat{y}(1-\hat{y})) \cdot (\boldsymbol{w}) \cdot (\boldsymbol{h}_1(1-\boldsymbol{h}_1)) \cdot (x)$$

Backpropagation

(binary classification example)



$$E = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$$

$$\hat{y} = \frac{e^{z_2}}{1+e^{z_2}}$$

$$z_2 = \boldsymbol{w}_2^T \boldsymbol{h}_1$$

$$\boldsymbol{h}_1 = \frac{e^{z_1}}{1+e^{z_1}}$$

$$\boldsymbol{z}_1 = W_1^T \boldsymbol{x}$$

already computed!

$$\frac{\partial E}{\partial \boldsymbol{W}_1} = \boxed{\frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2}} \cdot \frac{\partial z_2}{\partial \boldsymbol{h}_1} \cdot \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{z}_1} \cdot \frac{\partial \boldsymbol{z}_1}{\partial W_1}$$

$$= \left(\frac{\hat{y}-y}{\hat{y}(1-\hat{y})}\right) \cdot (\hat{y}(1-\hat{y})) \cdot (\boldsymbol{w}) \cdot (\boldsymbol{h}_1(1-\boldsymbol{h}_1)) \cdot (x)$$