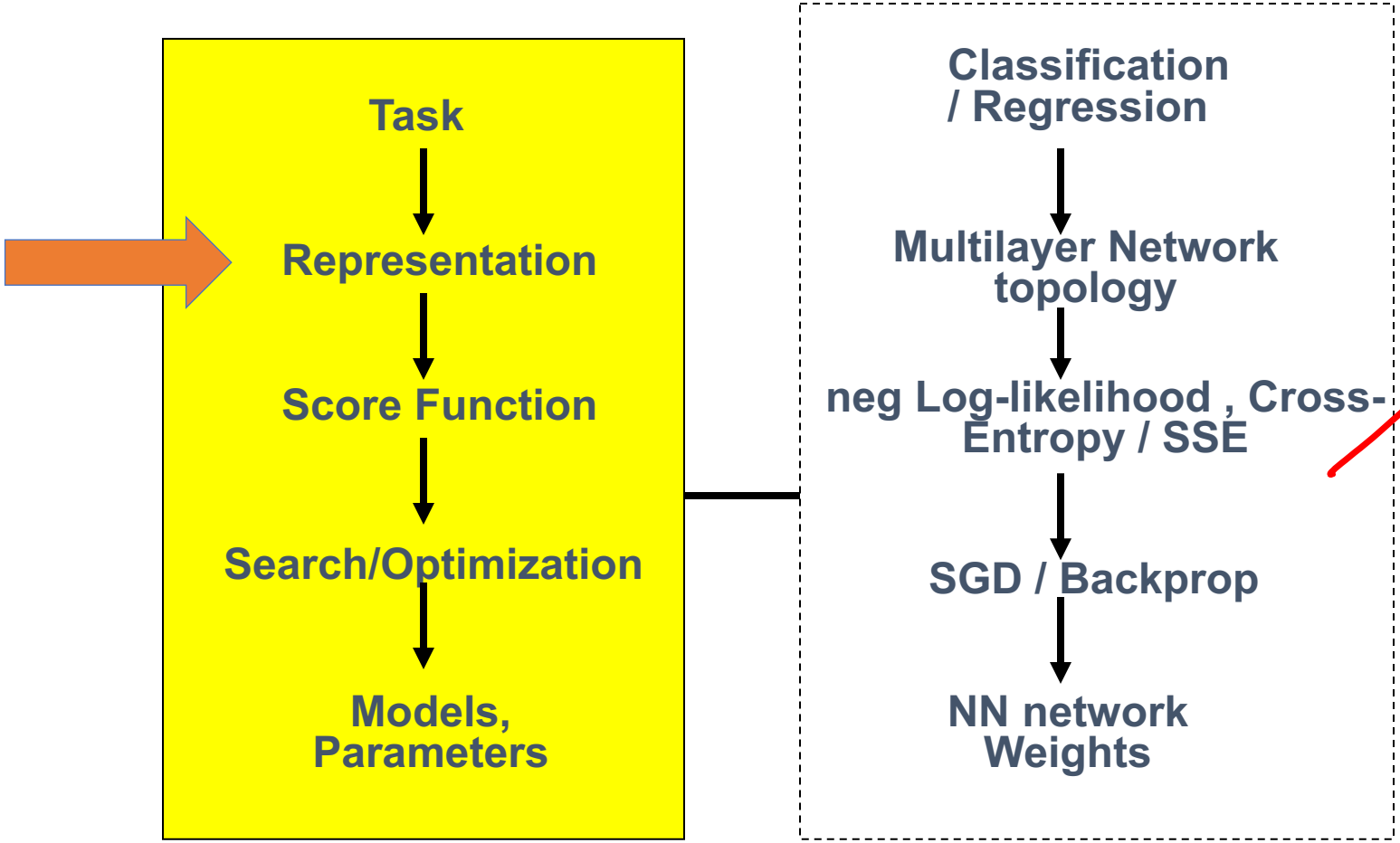
UVA CS 6316: Machine Learning

Lecture 15b: Recap of Neural Network (NN) and More BackProp

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Basic Neural Network



Binary Classification
{ Hinge loss (sum)
negative loglikehood (MLE / cross entropy)

$y_i \log p(y=1|x_i)$

$\hat{y}_i$

Basics

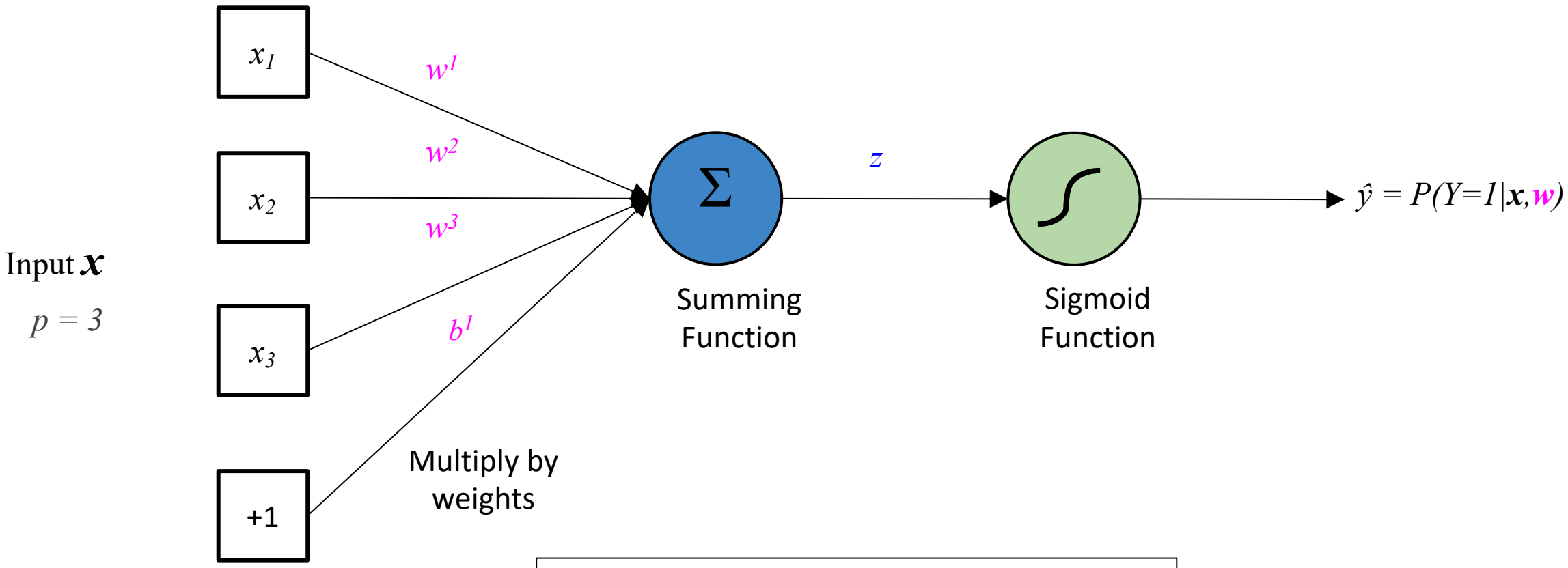
- Basic Neural Network (NN)
  - single neuron, e.g. logistic regression unit
  - multilayer perceptron (MLP)
  - various loss function
    - E.g., when for multi-class classification, softmax layer
  - training NN with backprop algorithm

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

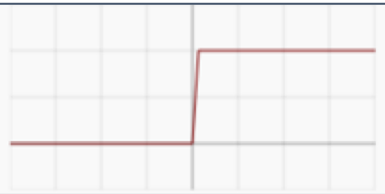
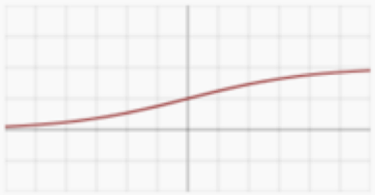
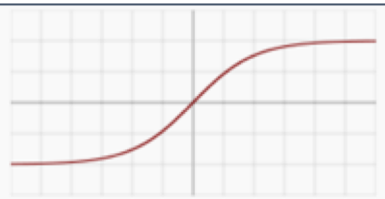
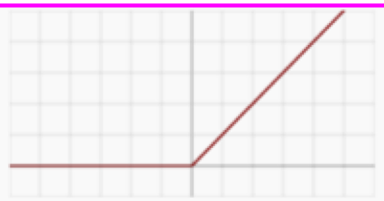


$$z = \boldsymbol{w}^T \cdot \boldsymbol{x} + \boldsymbol{b}$$

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

E.g., Many Possible Nonlinearity Functions

(aka transfer or activation functions)

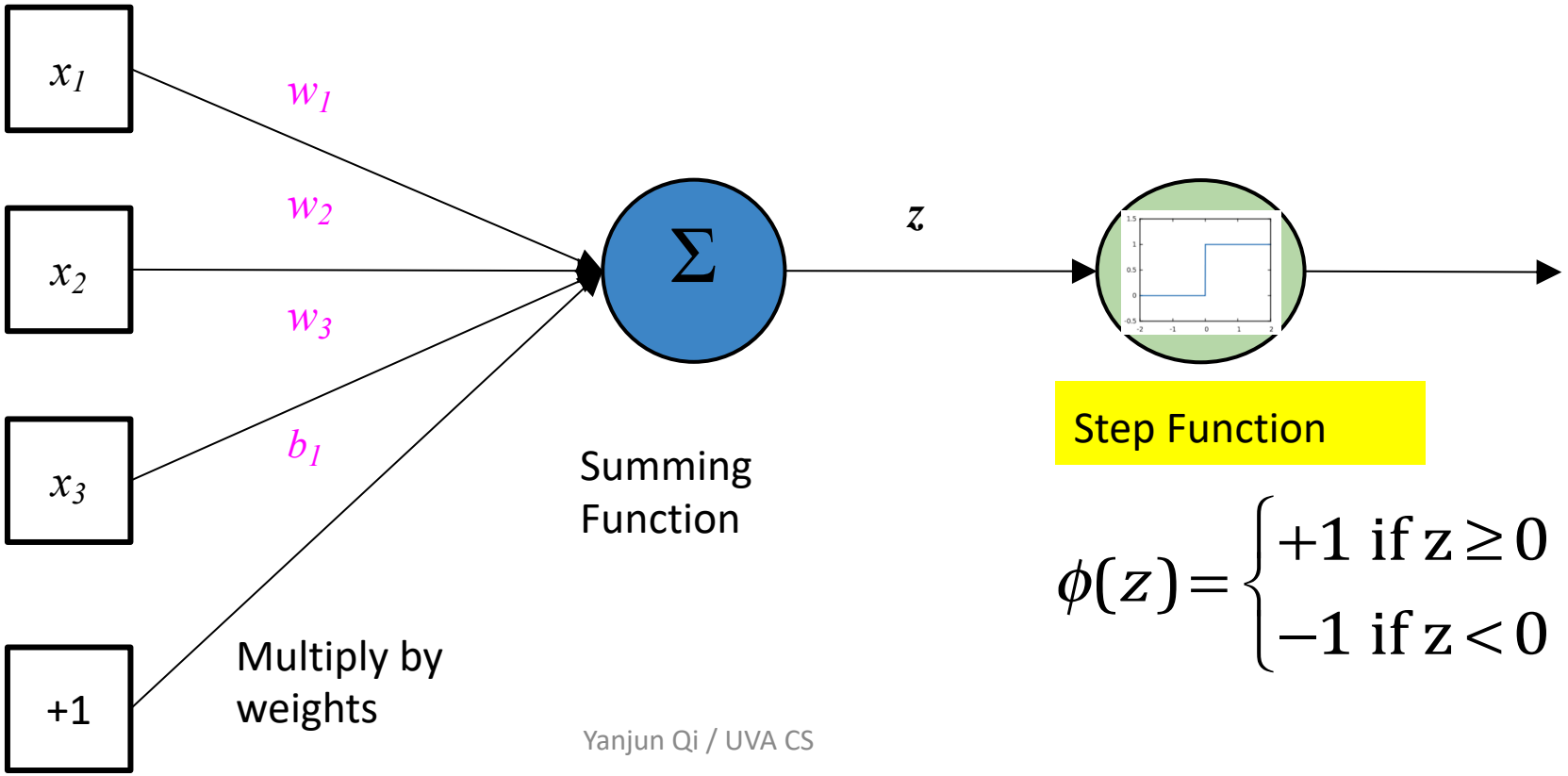
Name	Plot	Equation	Derivative (w.r.t x)
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
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

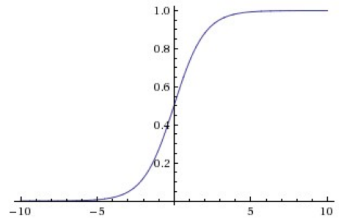
History ➔ Perceptron: 1-Neuron Unit with Step

- 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**

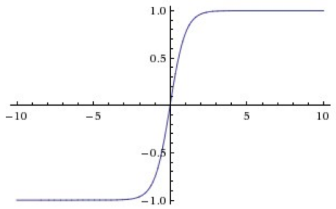


Sigmoid

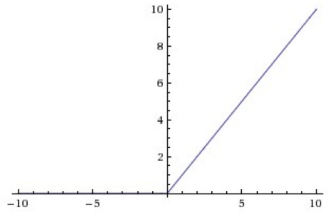
$$\sigma(x) = 1/(1+e^{-x})$$



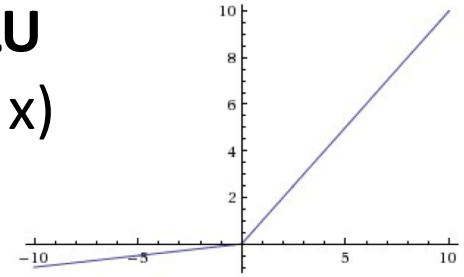
tanh tanh(x)



ReLU max(0,x)



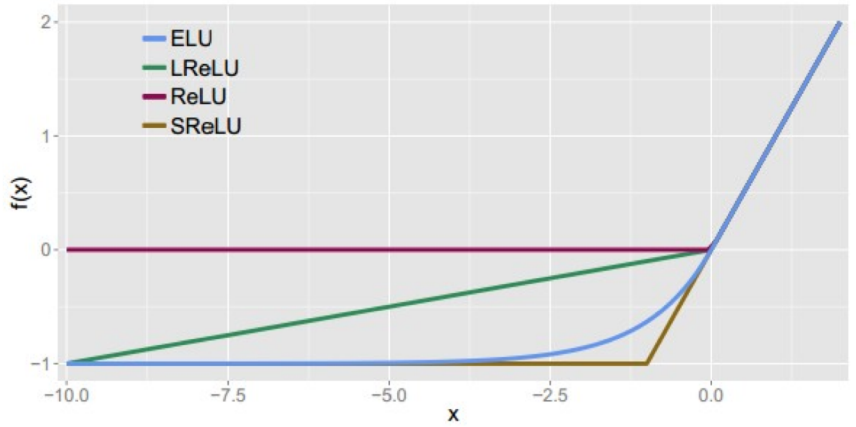
Leaky ReLU
max(0.1x, x)



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

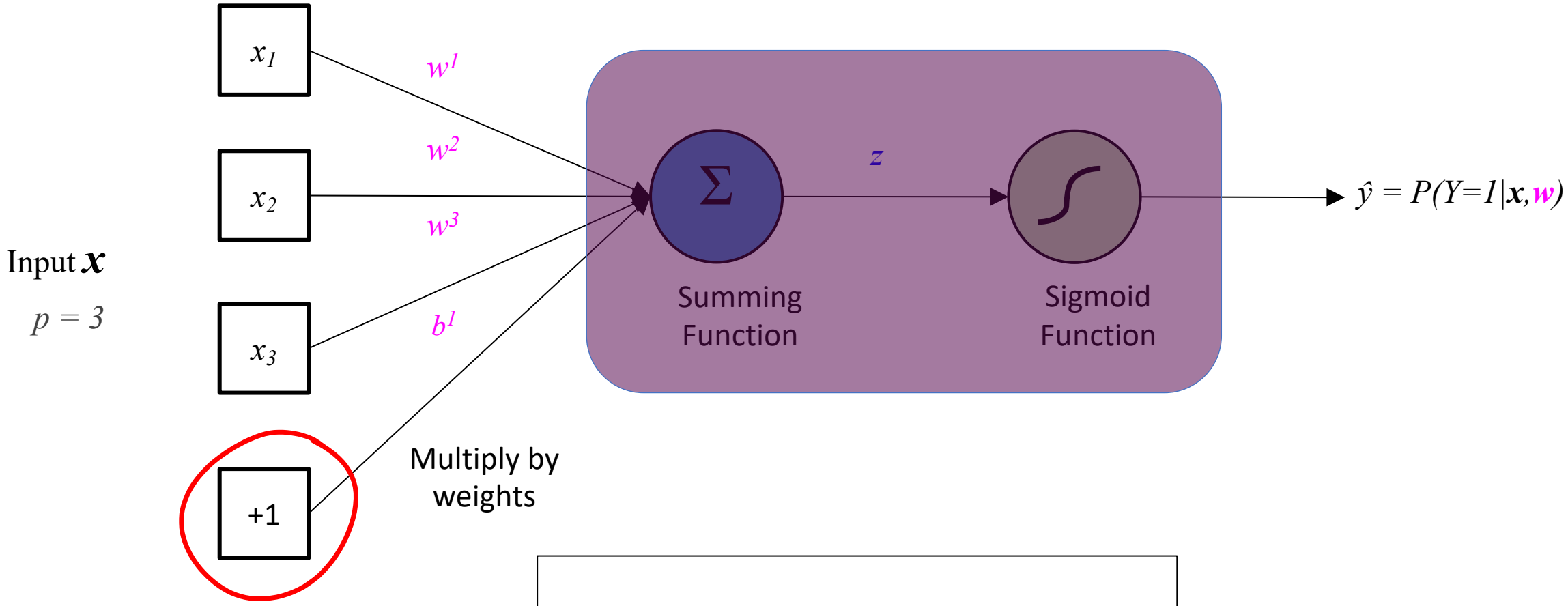
ELU

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



From Feifei Li Stanford Cousre

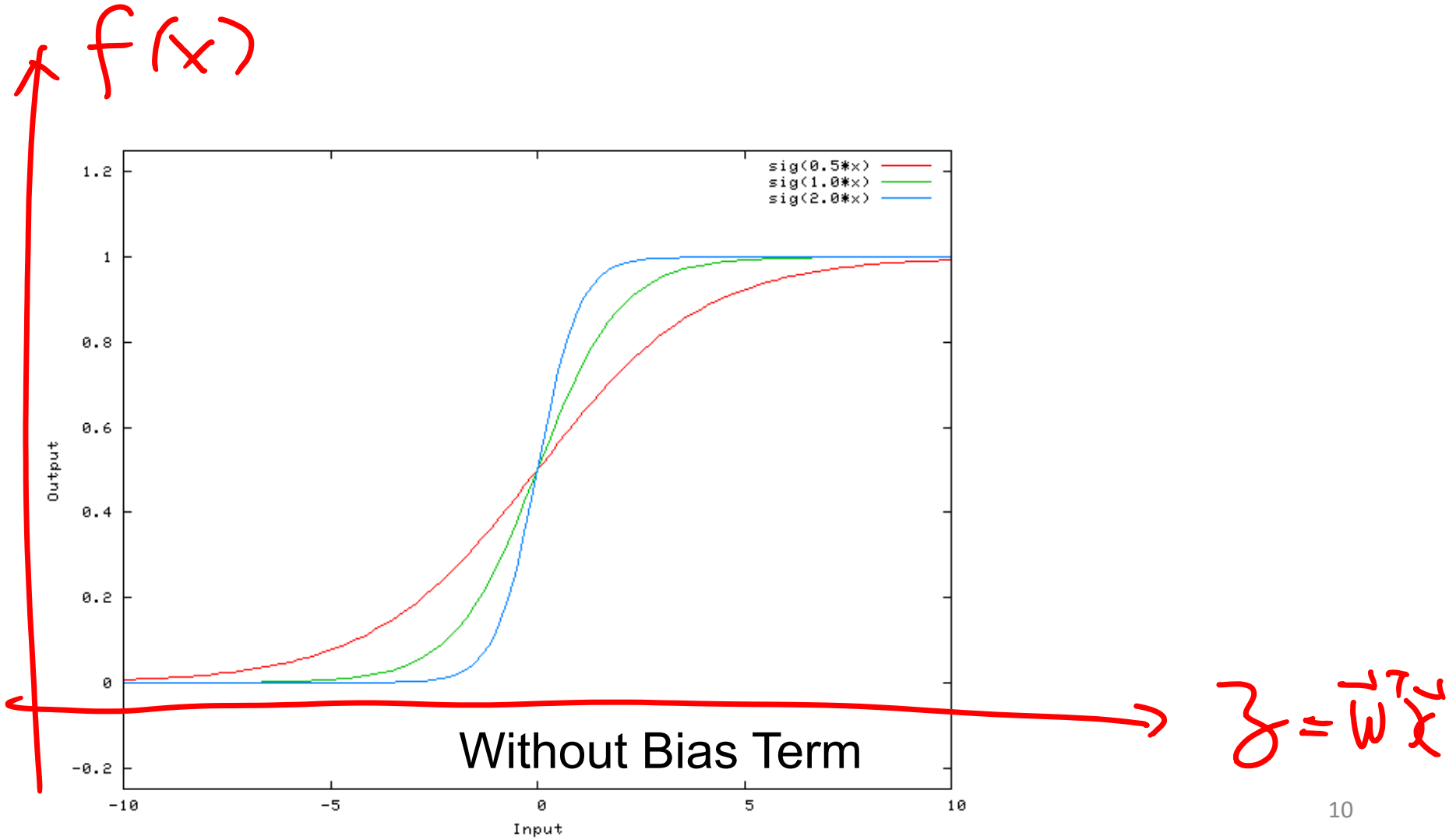
Bias Term?



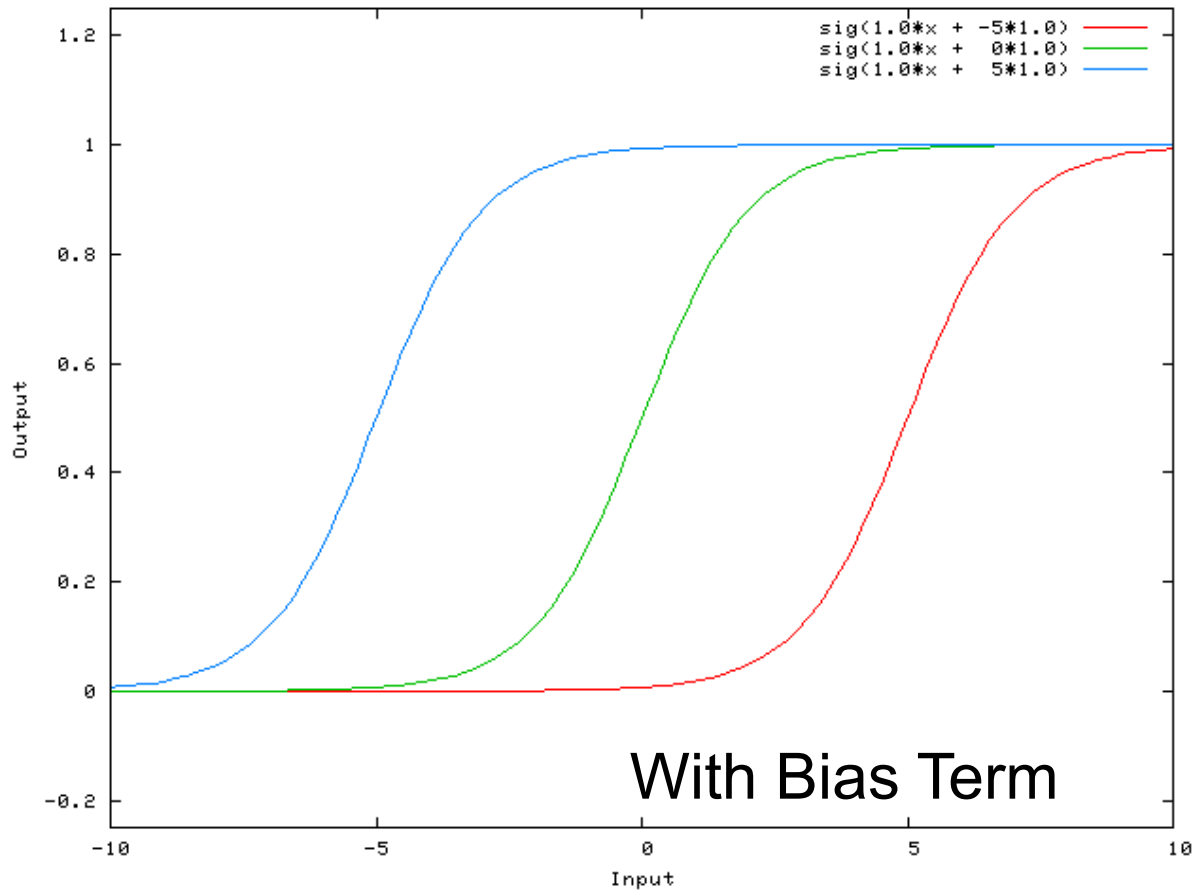
$$z = \boldsymbol{w}^T \cdot \boldsymbol{x} + \boldsymbol{b}$$

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

Without Bias Term



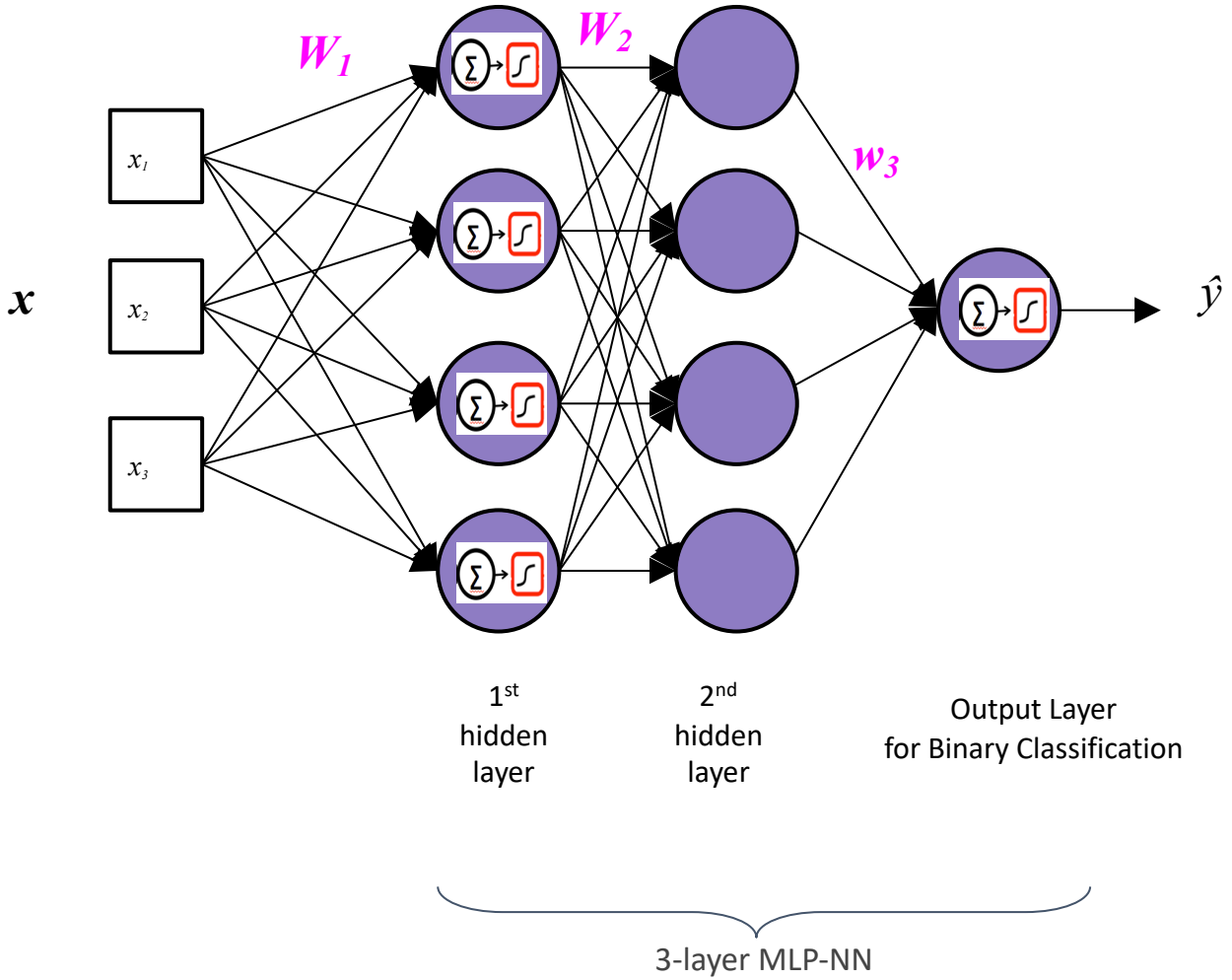
With Bias Term



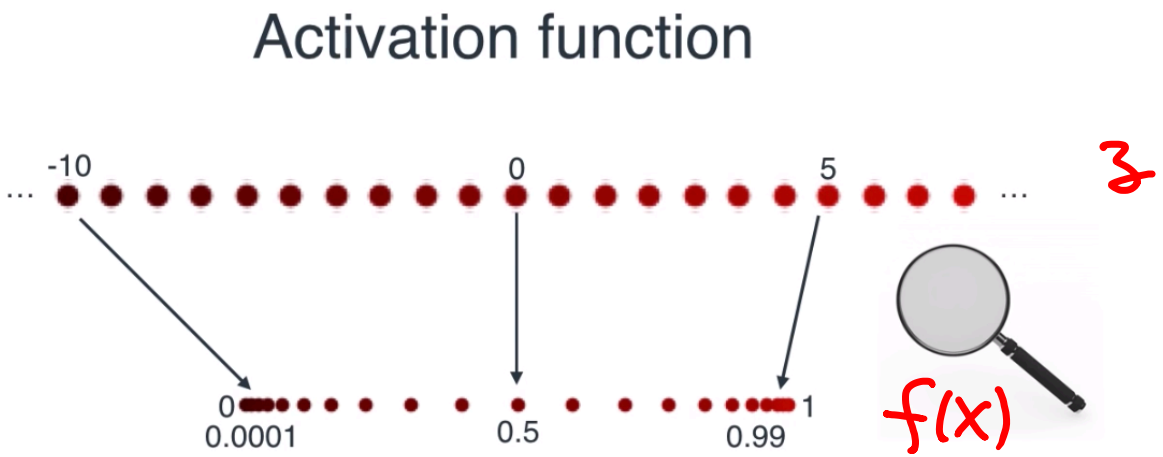
Basics

- Basic 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

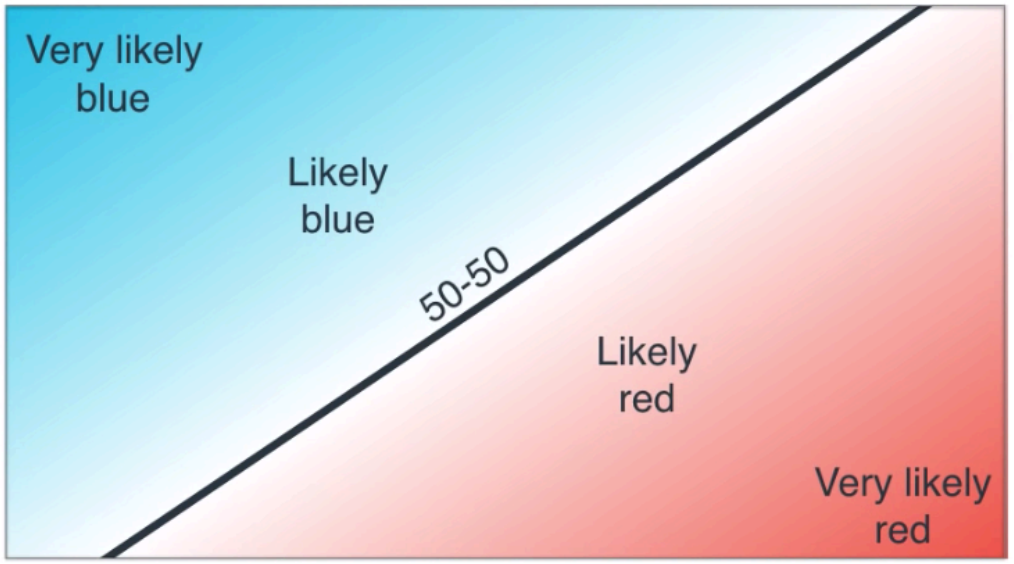
Multi-Layer Perceptron (MLP)- (Feed-Forward NN)



From https://www.youtube.com/watch?v=BR9h47Jtqyw&list=PLs8w1Cdi-zvavXlPXEAsWIh4Cgh83pZPO&index=3



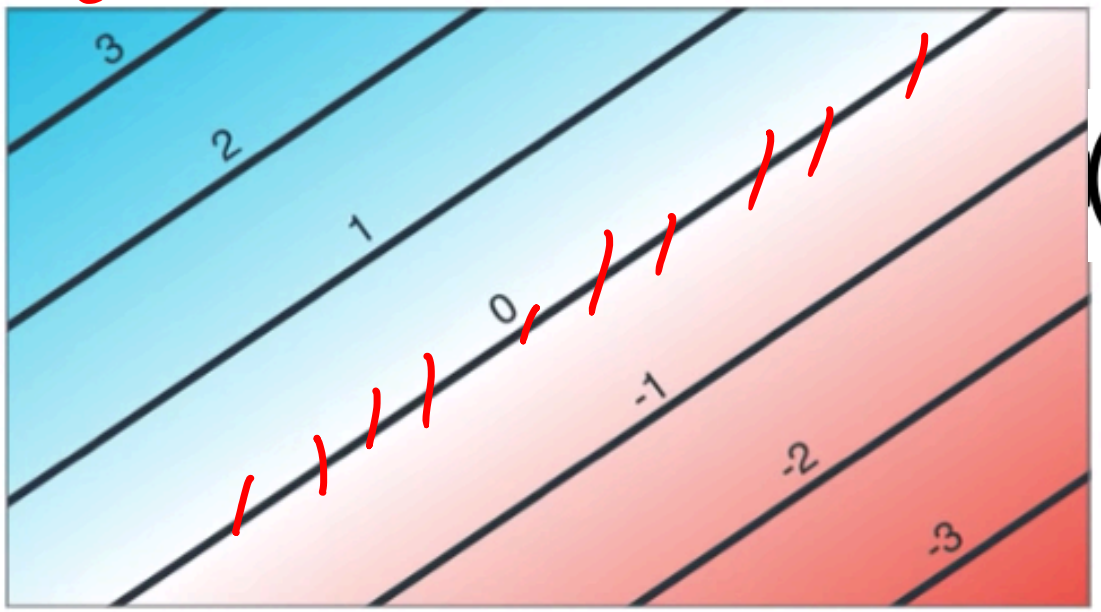
z

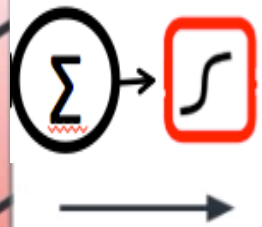


$$f(x) = \frac{1}{1 + e^{-x}}$$

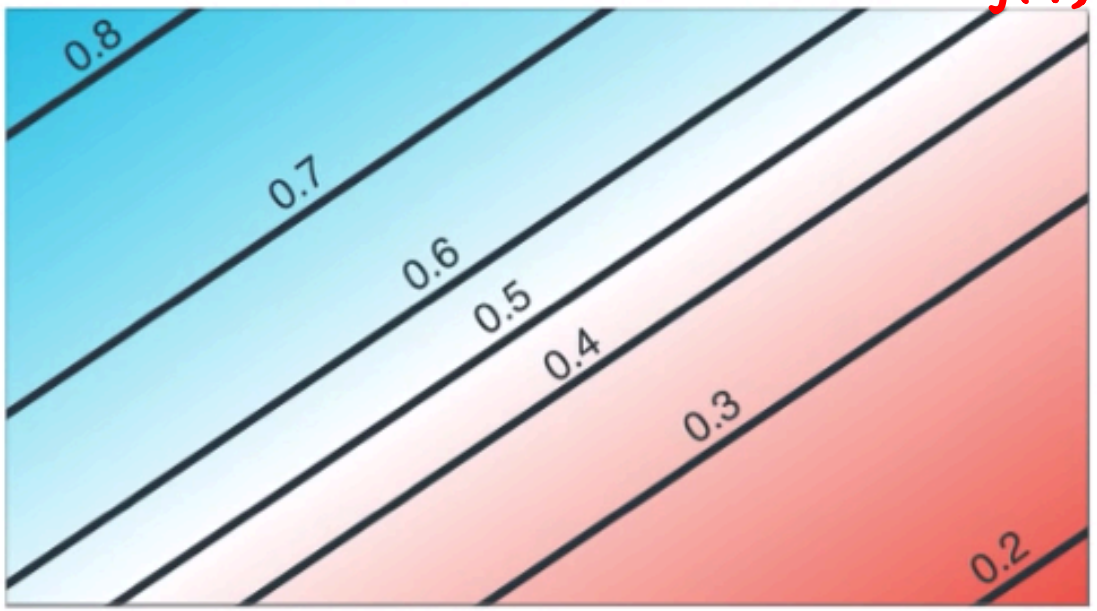
x → Σ → z → sig → f(x)

z

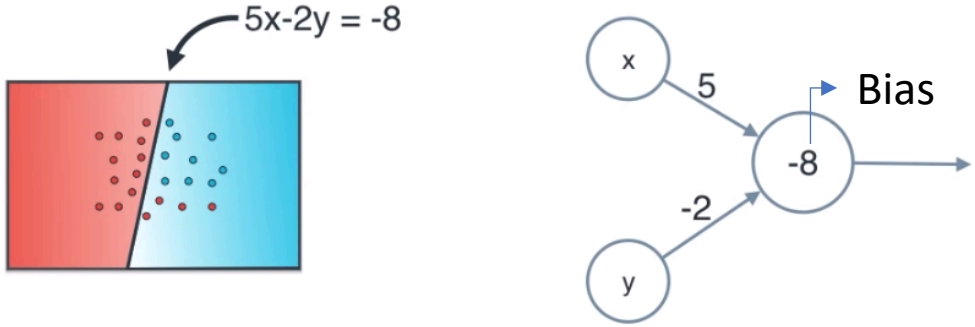




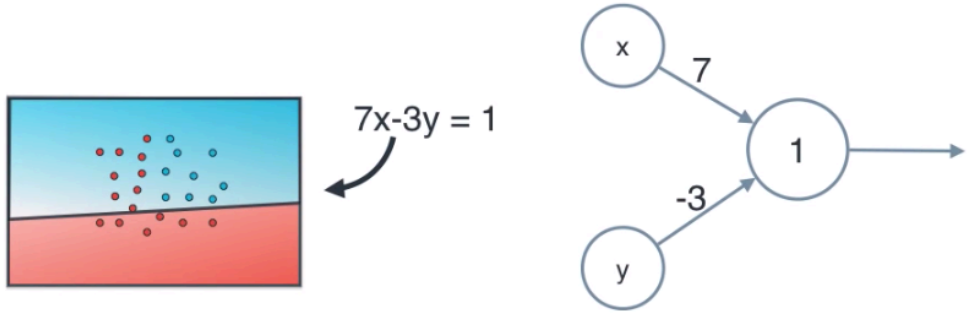
y = f(x)



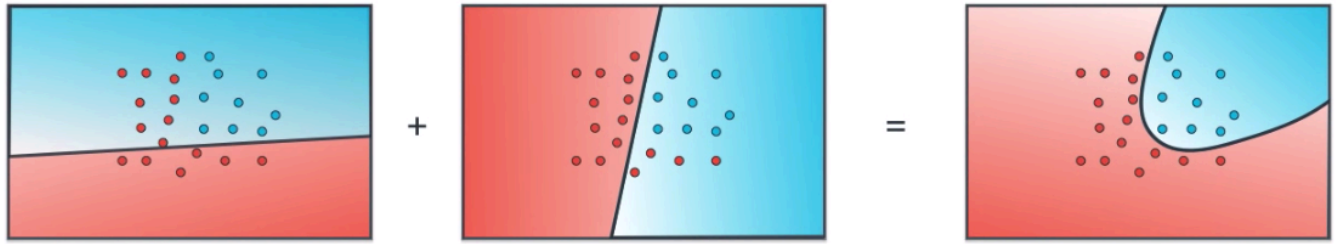
One neuron

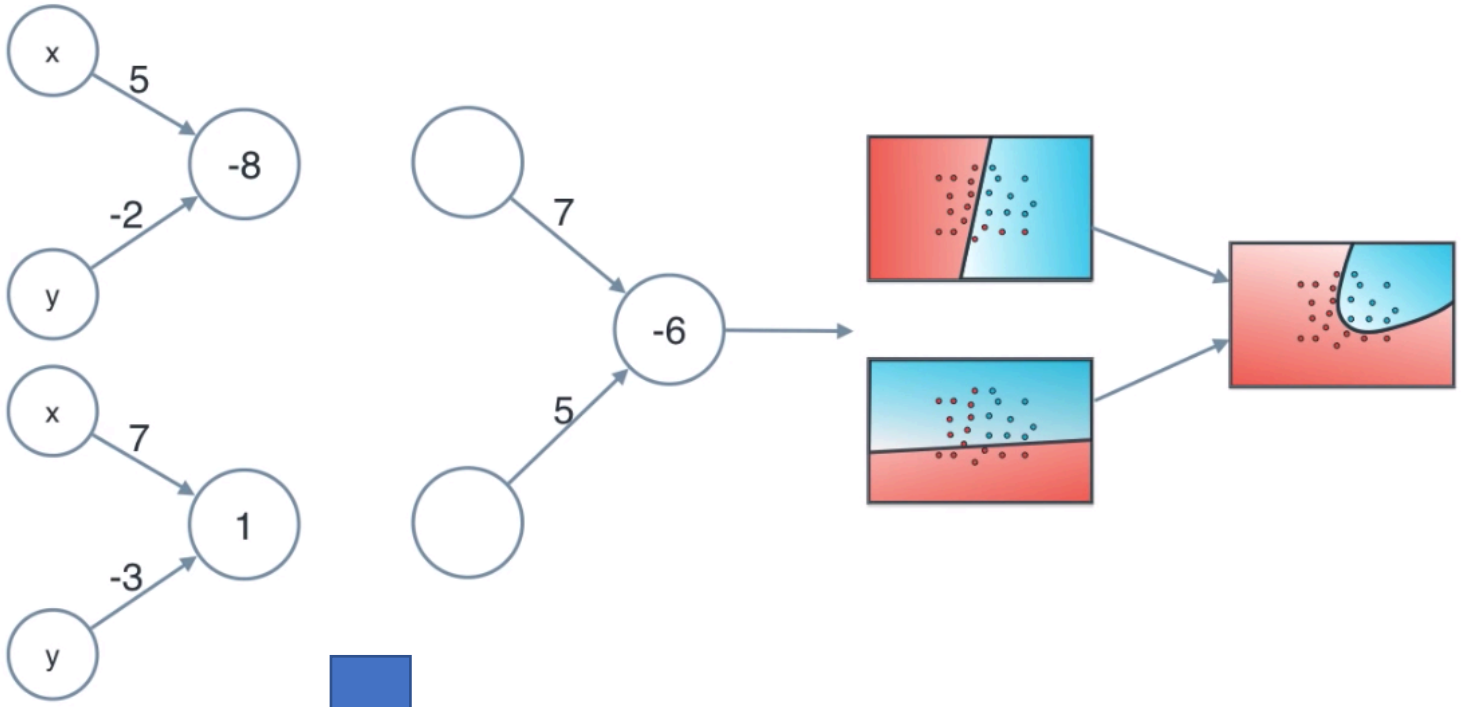


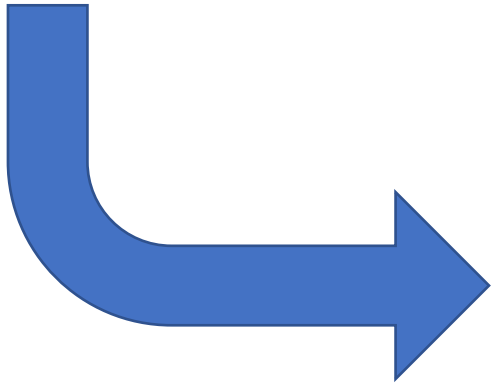
Another neuron

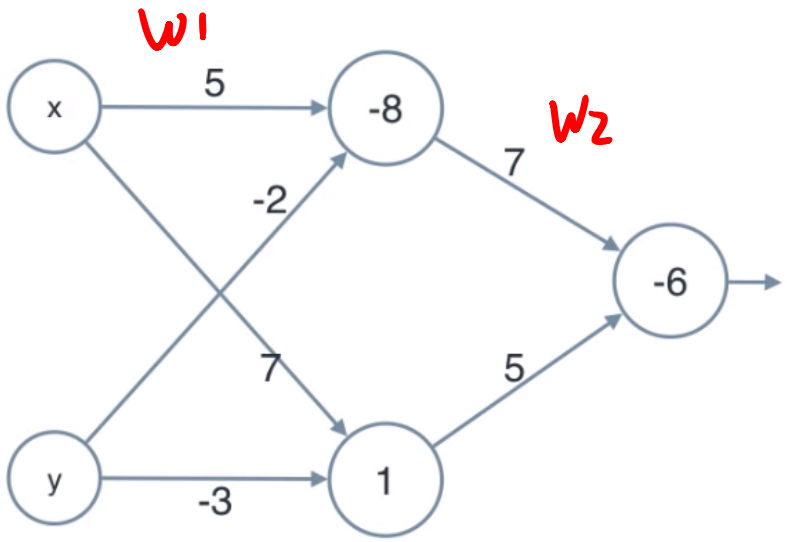


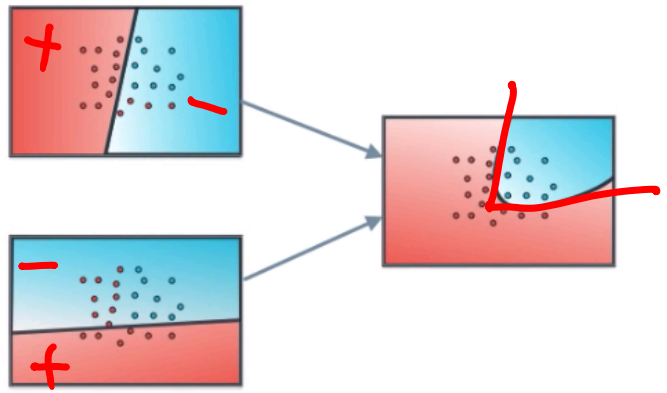
How to combine to get nonlinear decision boundary?



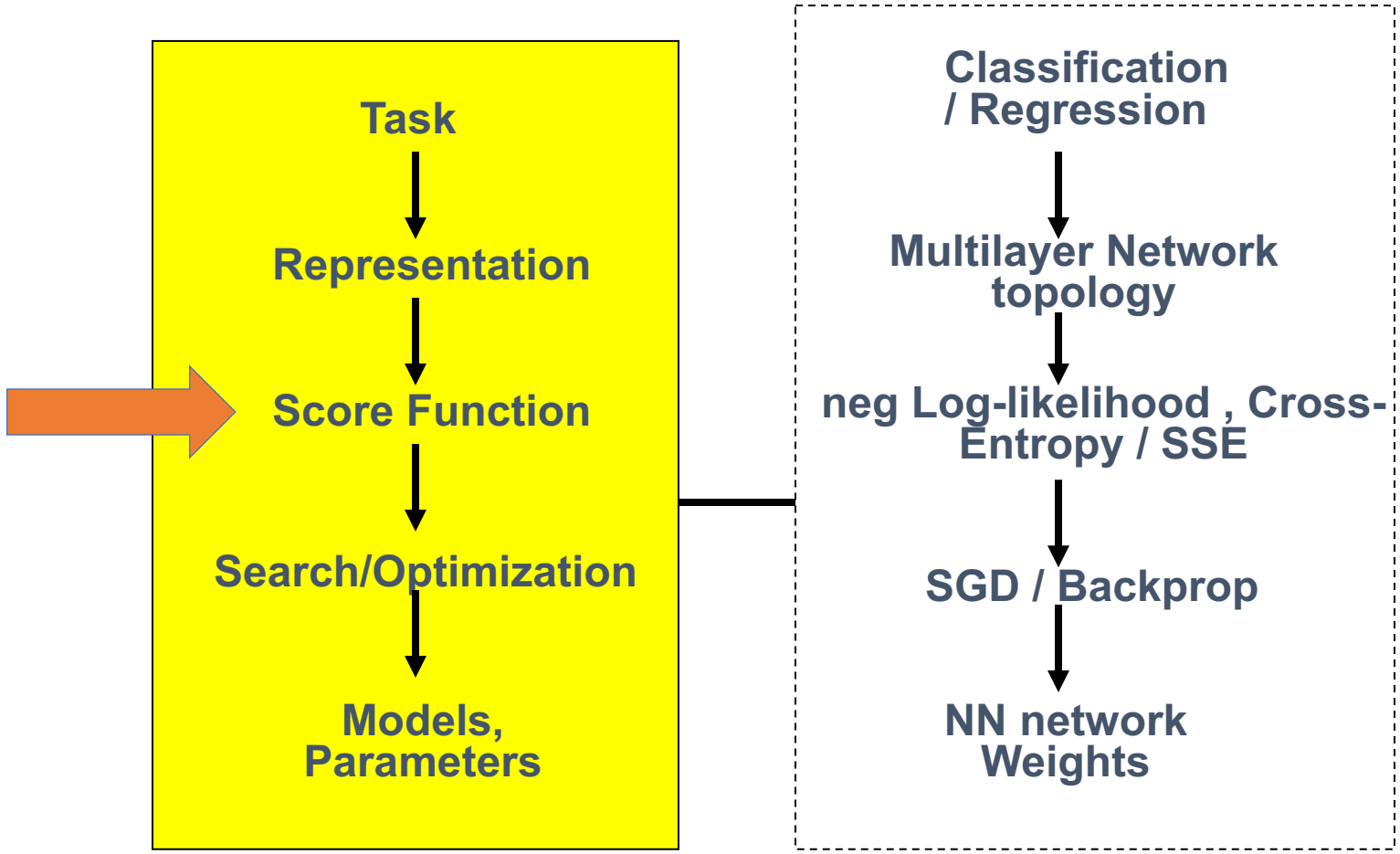

x
y
5
-2
-8
7
5
-6
x
y
7
-3
1




W1
W2
x
y
5
-2
7
-3
-8
1
7
5
-6


+
−
−
+

Basic Neural Network

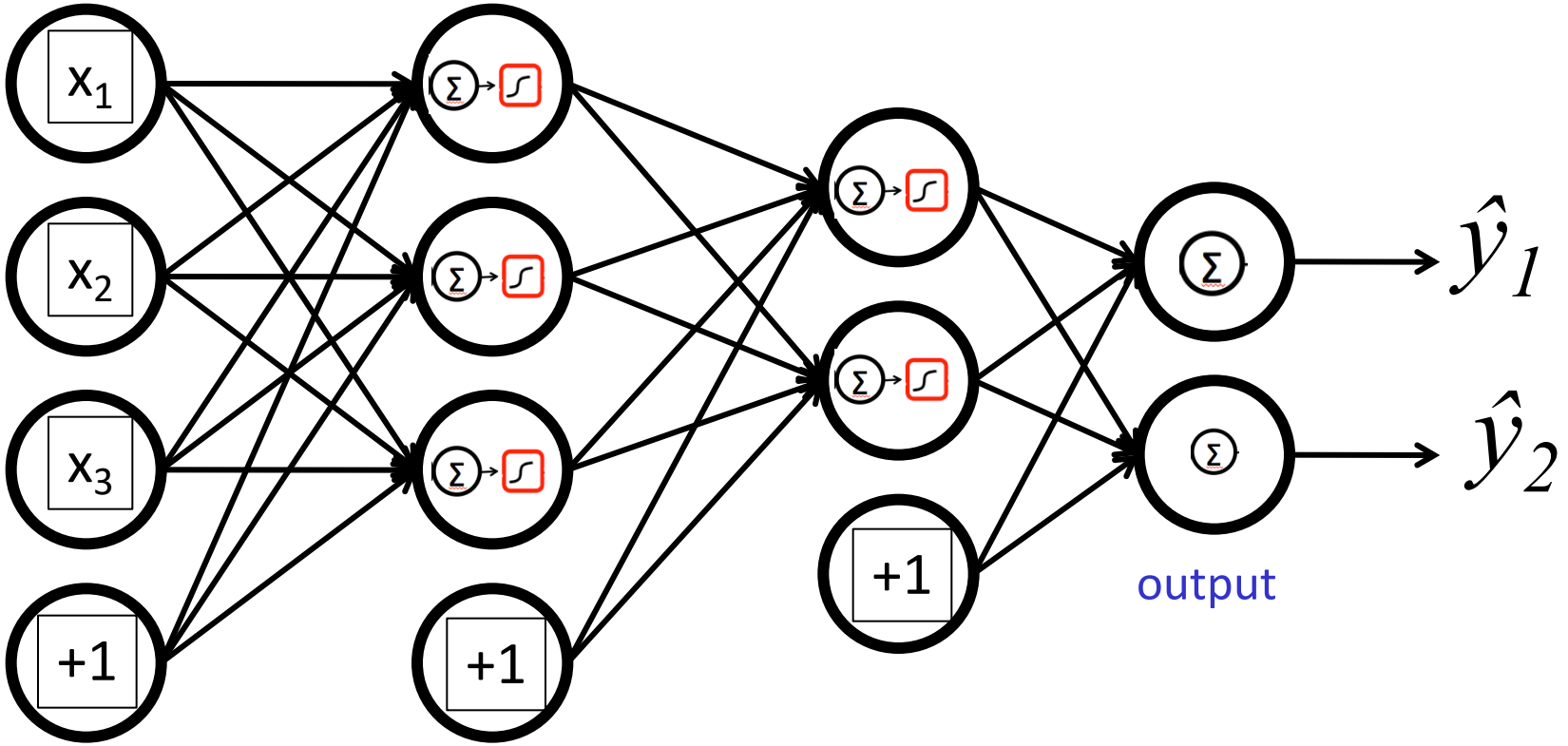


Basics

- Basic 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

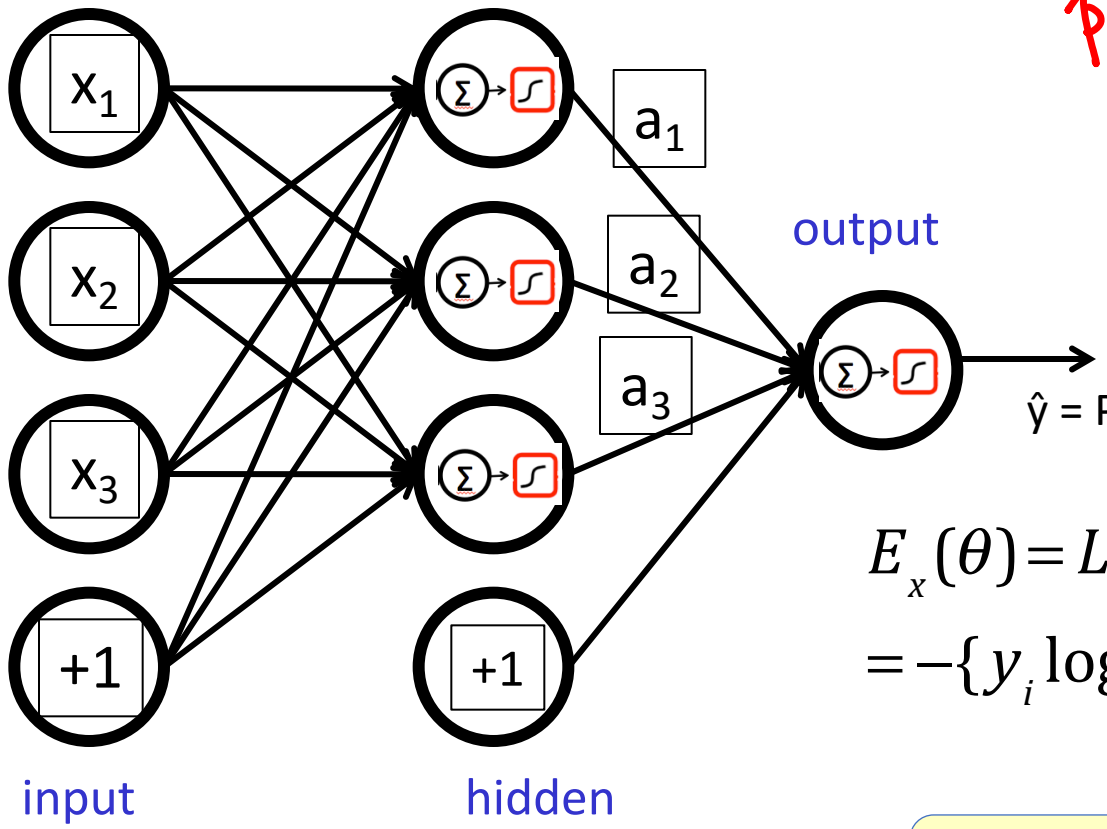
Example: 4 layer network with 2 output units:



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

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

Example: 3 layer network:



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

$p(\text{Head})$

For Bernoulli distribution,

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

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

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

data likelihood

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(head)

Basic Bernoulli

Logistic / Bernoulli

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

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

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

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

Log likelihood

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

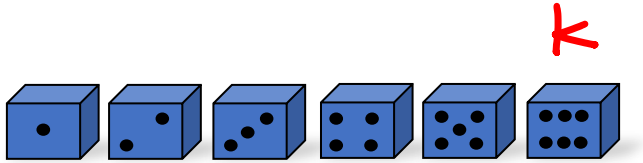
Binary Classification ➔ Multi-Class Classification

Bernoulli distribution to model the target variable with its Bernoulli parameter p=p(y=1|x) predefined as function on x

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

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

Multinoulli Distribution, e.g.,



$p(y=1|x)$

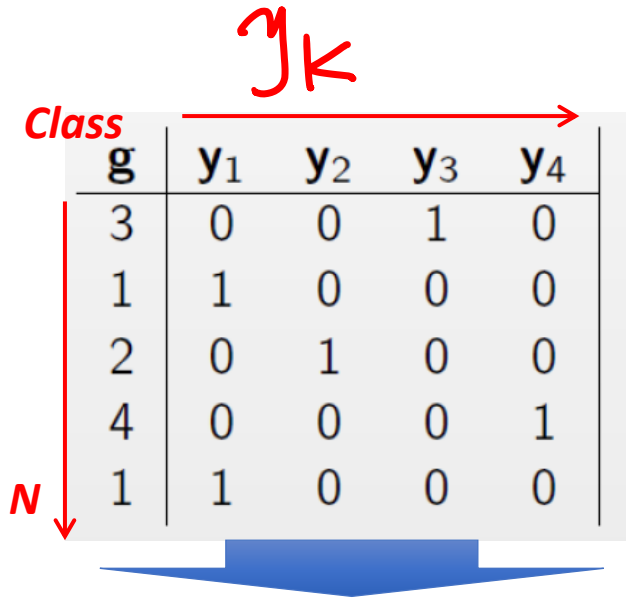
$p(y=2|x)$

$p(y=3|x)$

$\vdots$

$p(y=6|x)$

Review: Multi-class variable representation



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

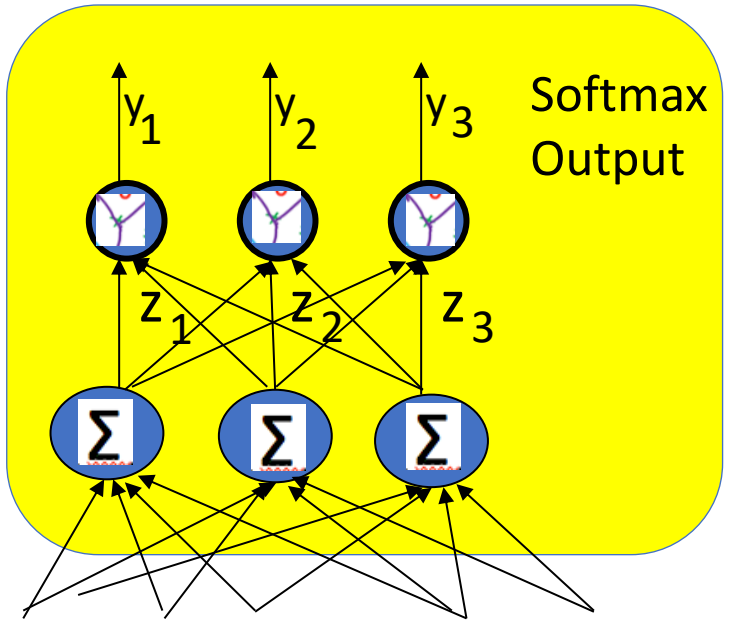
- How to classify to multi-class ?
  - First: learn K different regression
  - Then:

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

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

discriminative classifier $p(y_1 \cdots y_K | x)$ $p_k(x)$

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



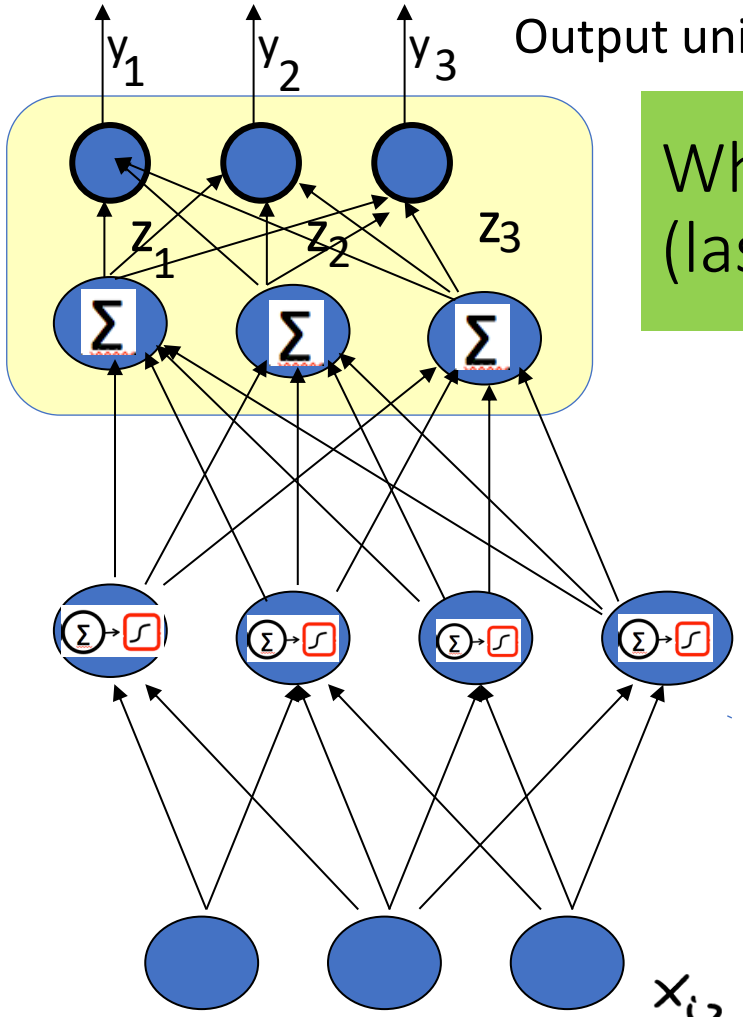
$$Pr(G = k \mid X = x) = Pr(Y_k = 1 \mid X = x)$$

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

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

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

$f_1(x)+f_2(x)$
$+ \cdots + f_k(x)=1$
$0 \le f_i(x) \le 1$

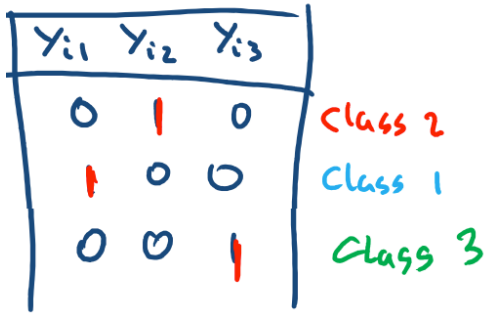


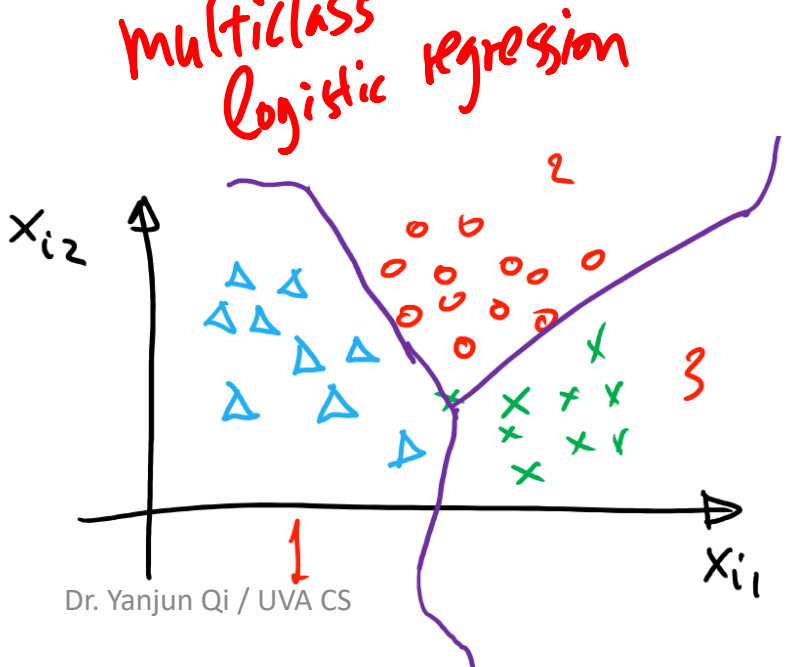
When for multi-class classification (last output layer: softmax layer)

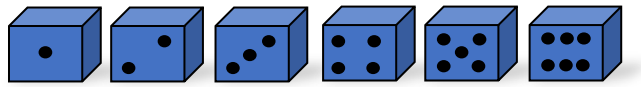
last layer is softmax output layer ➔
a Multinoulli logistic regression unit

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

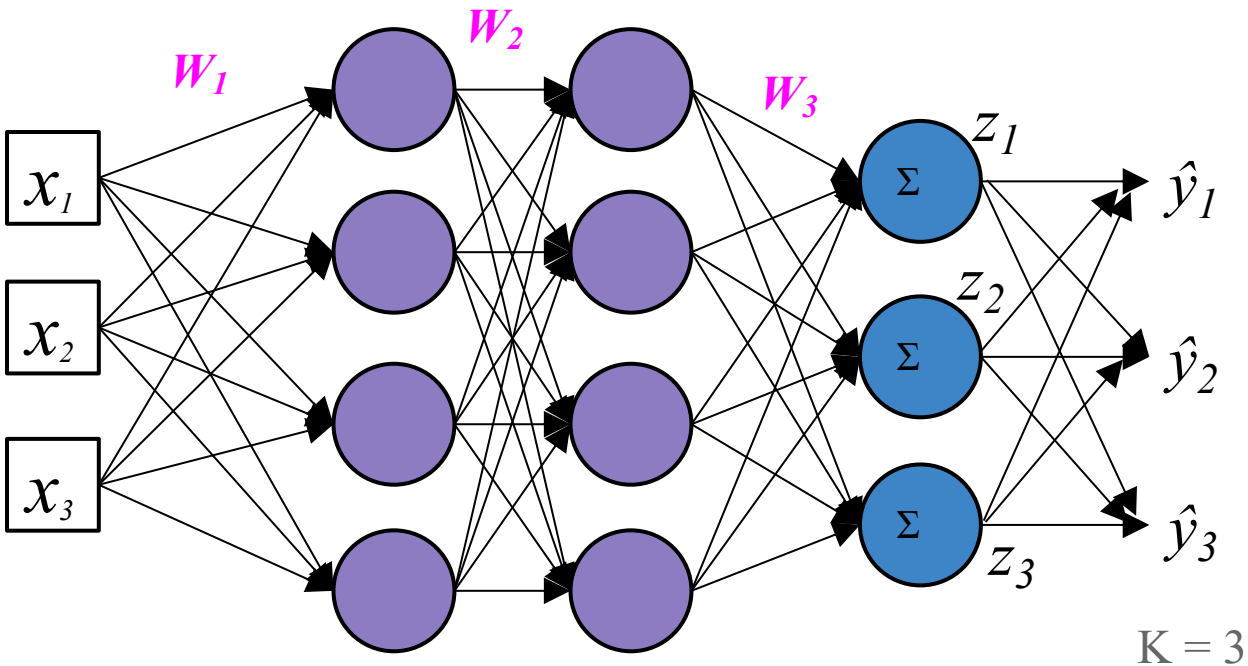
"Softmax" function. Normalizing function which converts each class output to a probability.

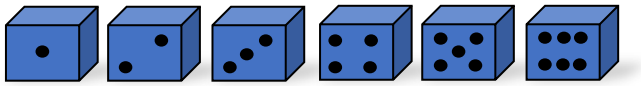






E.g., Cross-Entropy Loss for Multi-Class Classification

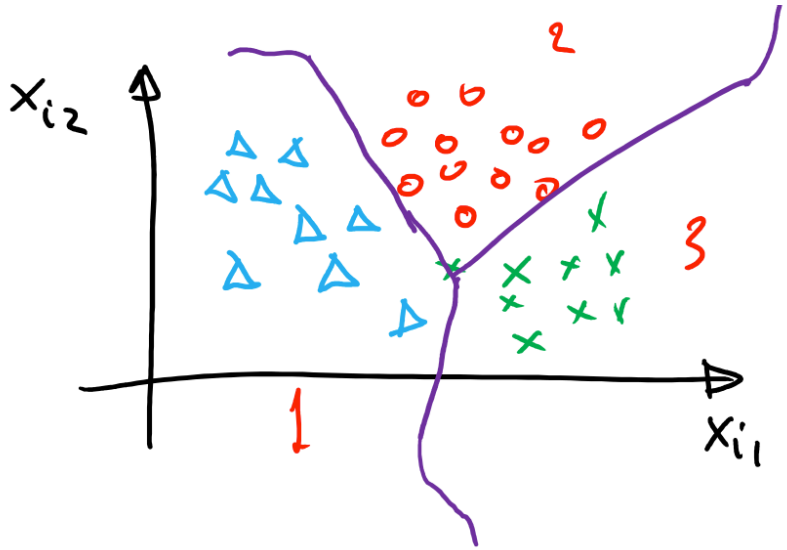




$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}} = P(y_i = 1 \mid \boldsymbol{x})$$

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

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



MLE ⇓ min. negative likelihood

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

$y \log y + (1-y) \log(1-y)$ Binary

Cross-entropy loss

“softmax” layer function for multi-class classification

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

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

y_{true} $\hat{y}$

“0” for all except true class

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

$p_1, p_2 \ldots p_k$

“0” for all except true class

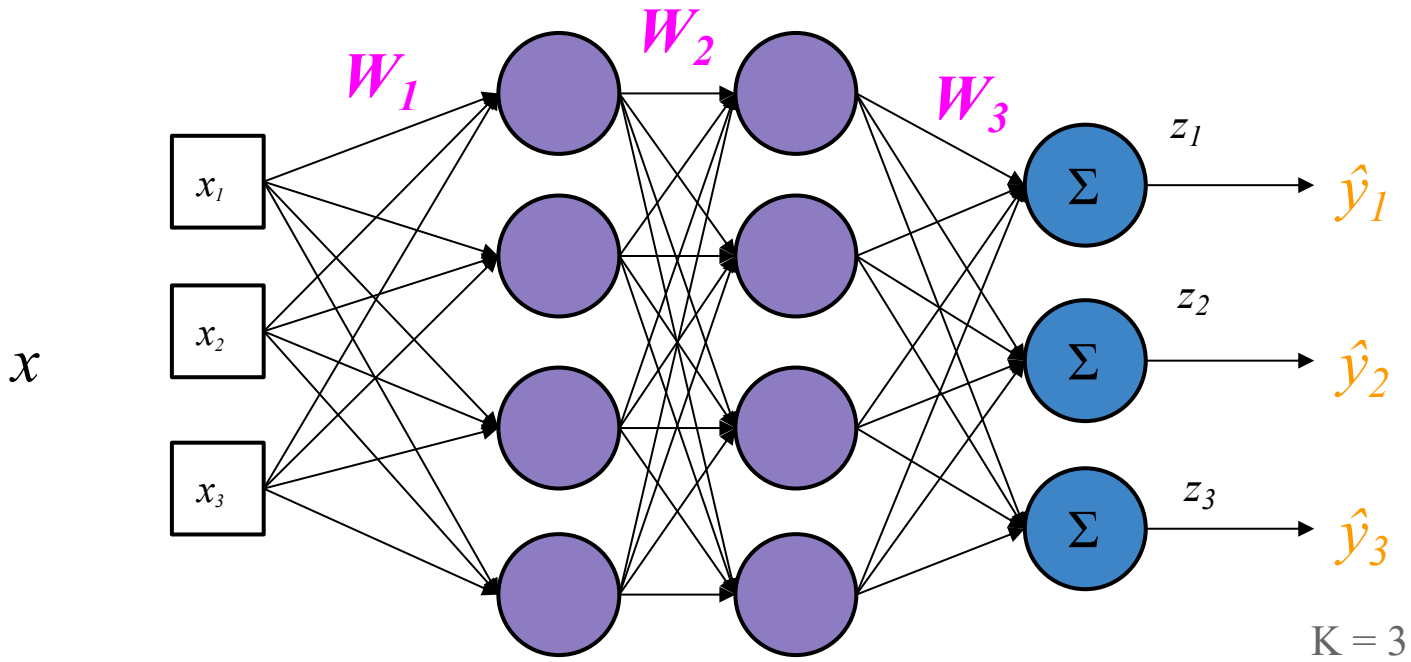
① $$\frac{\partial E}{\partial z_i} = \sum_{j=1\ldots K} \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_i} = \hat{y}_i - y_{true,i}$$

Error calculated from predicted Output vs. true

② $E = (y-\hat{y})^2$ $\quad \frac{\partial E}{\partial y} = 2(y-\hat{y})$

Summary Recap: Multi-Class Classification Loss

Cross Entropy Loss



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

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

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

y $\hat{y}$

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

"0" for all except true class

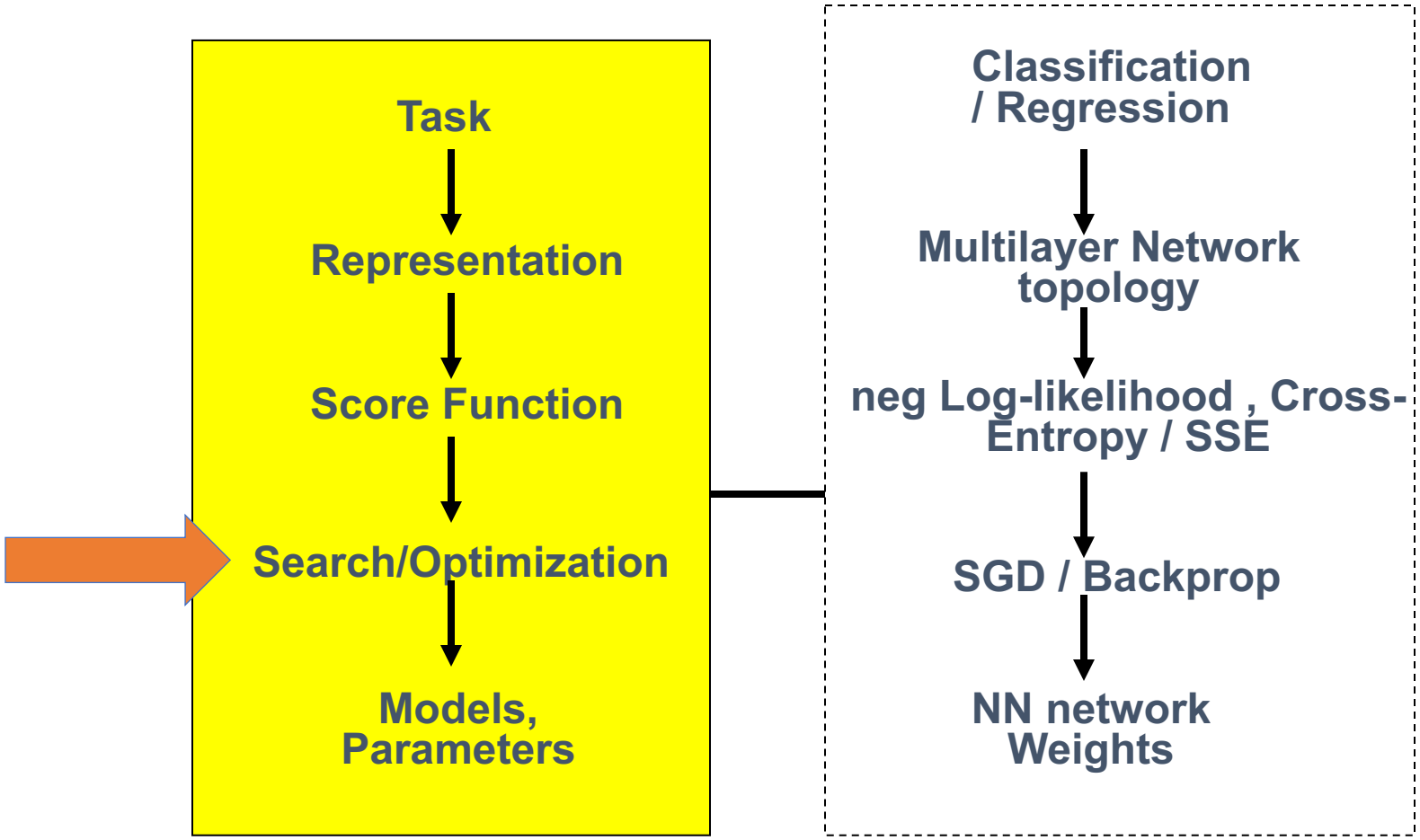
Logistic: a special case of softmax for two classes

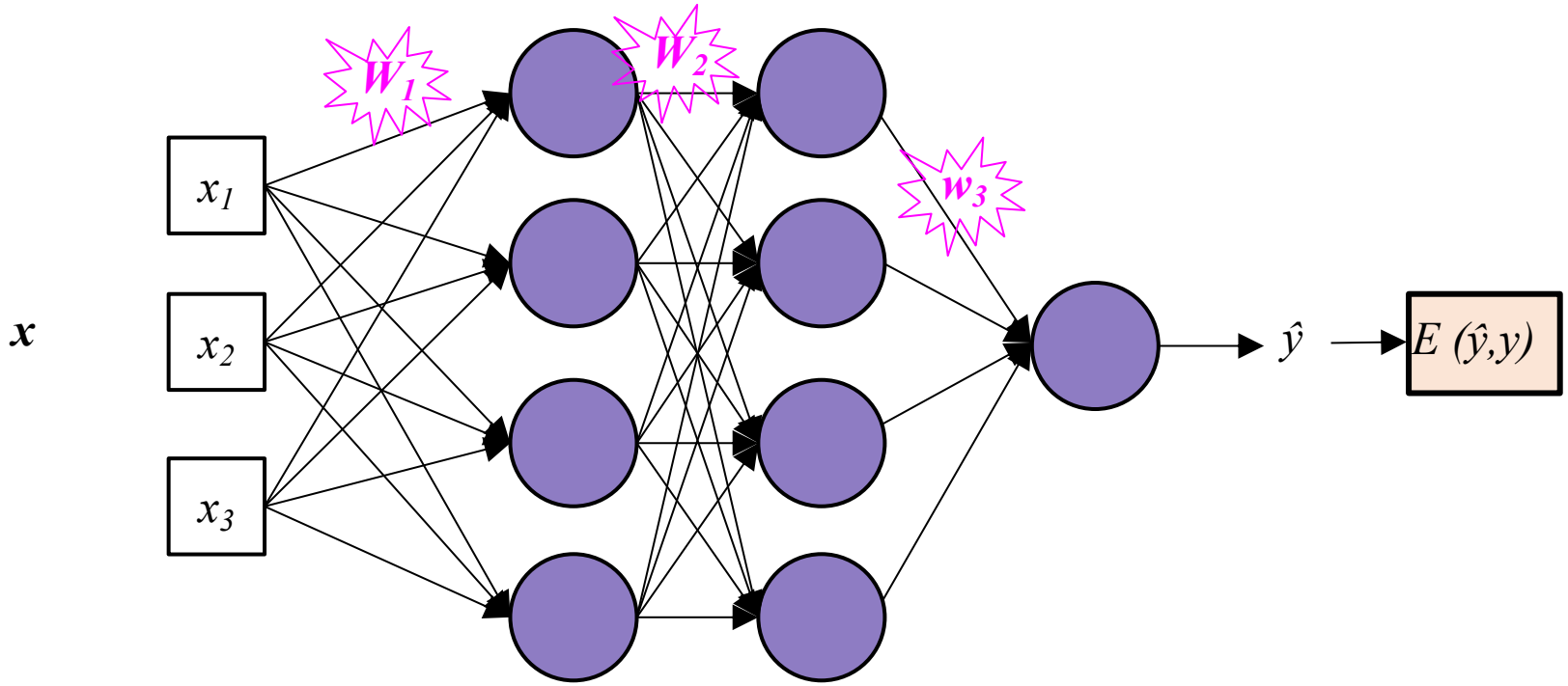
{H, T}

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

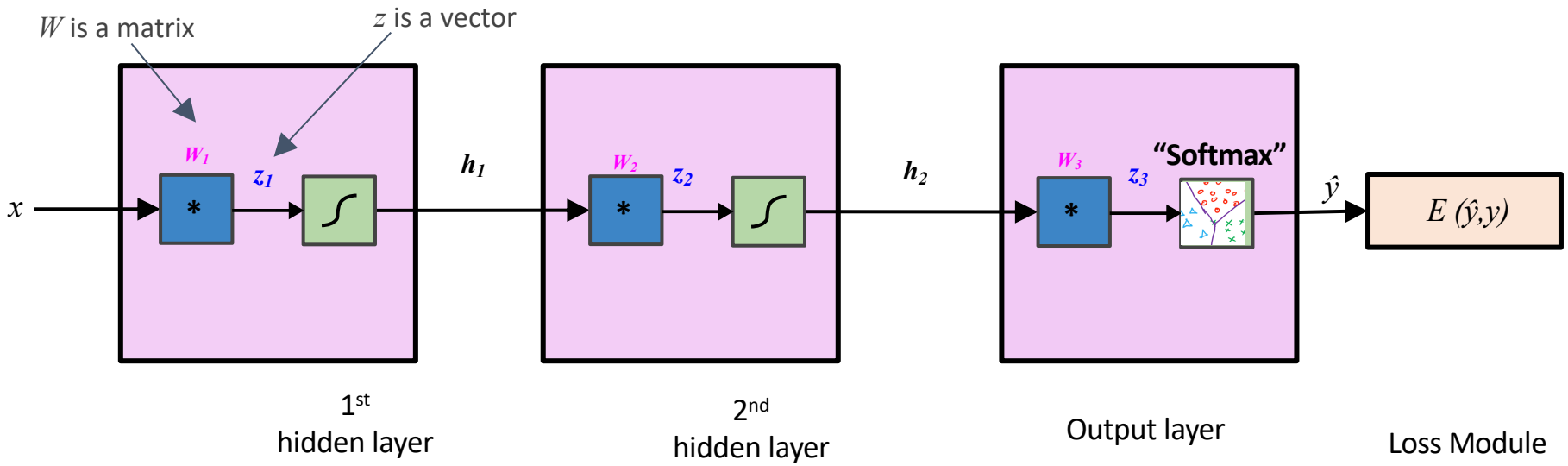
- So the logistic binary case is just a special case that avoids using redundant parameters:
  - Adding the same constant to both z1 and z0 has no effect.
  - The over-parameterization of the softmax is because the probabilities must add to 1.

Basic Neural Network

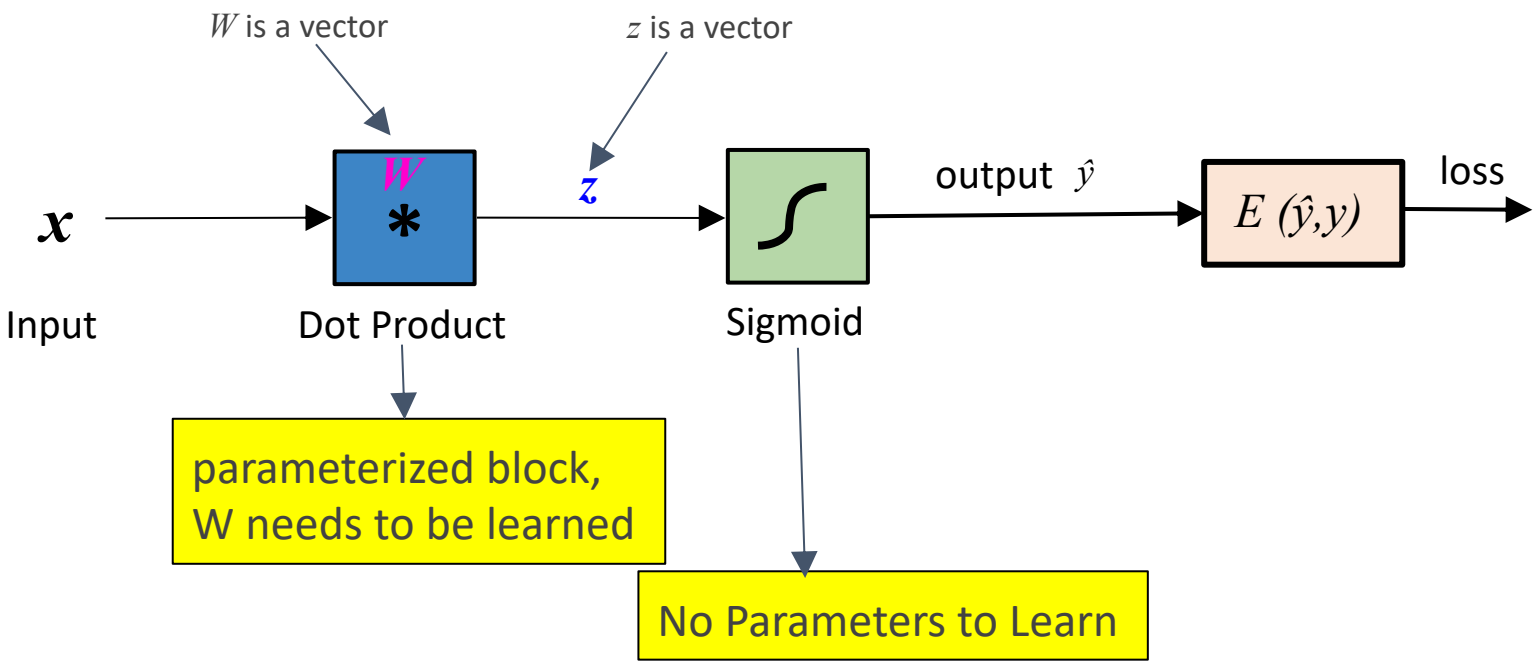



x
x_1
x_2
x_3
w_1
w_2
w_3
$\hat{y}$
$E(\hat{y},y)$

e.g., “Block View” of multi-class NN



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



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

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

$W = \{W^1, W^2, \cdots, W^T\}$

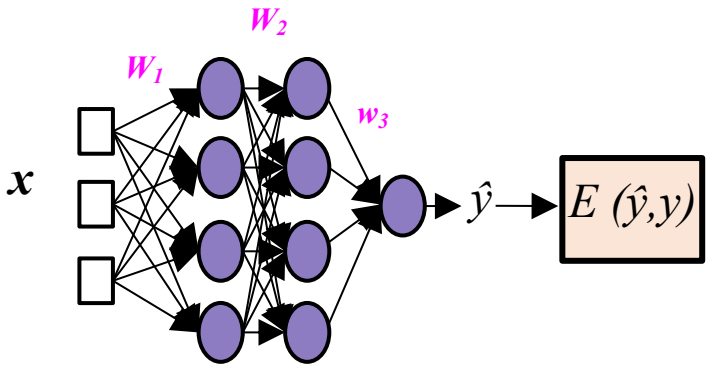
$$W^{t+1} = W^t - \eta \frac{\partial E}{\partial W^t} = W^t + \Delta w$$

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:**

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



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

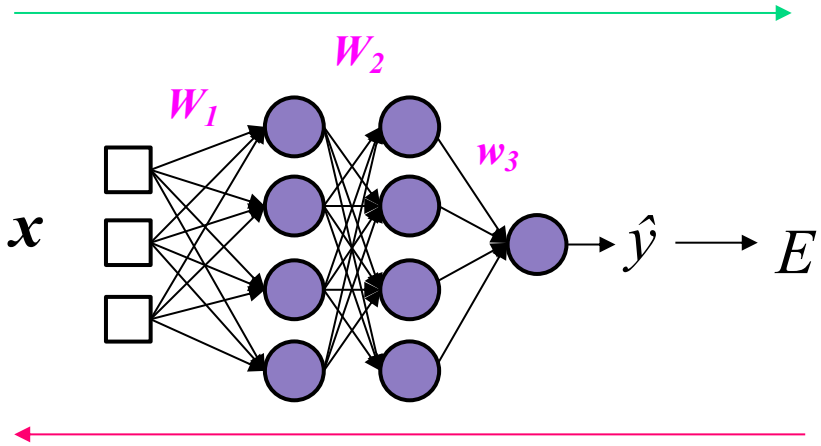
LeCun et. al. *Efficient Backpropagation*. 1998

Backpropagation

- 1. Initialize network with random weights
- 2. For all training examples:
  - **Forward:** Feed training 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 $\frac{\partial E}{\partial W_l{}^t}$
    - Adapt weights in current layer $W_l{}^{t+1} = W_l{}^t - \eta \frac{\partial E}{\partial W_l{}^t}$

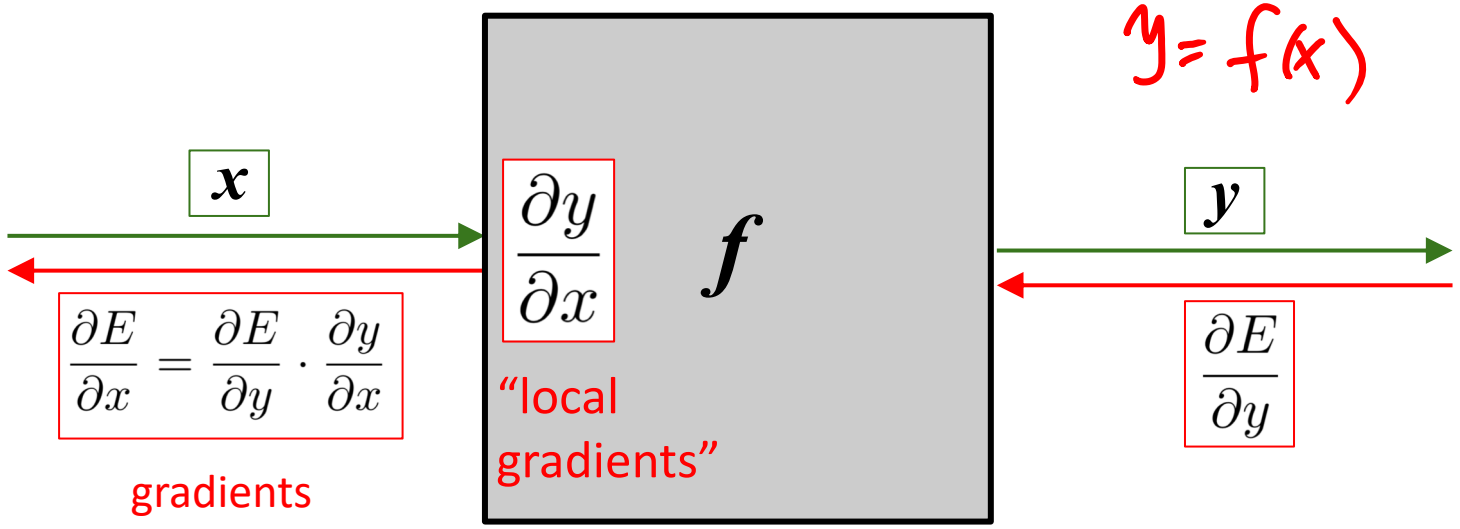
Need to calculate these!

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

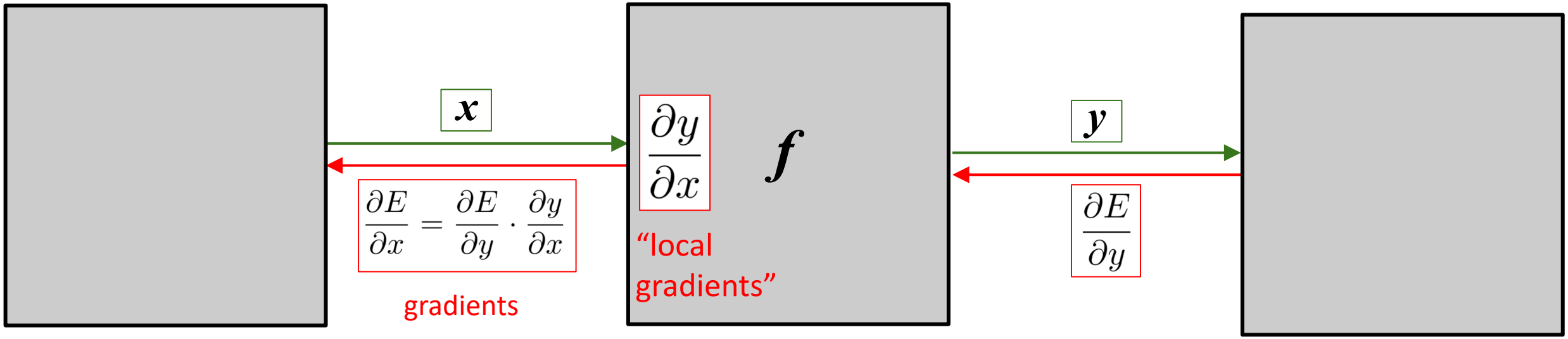
“Local-ness” of Backpropagation



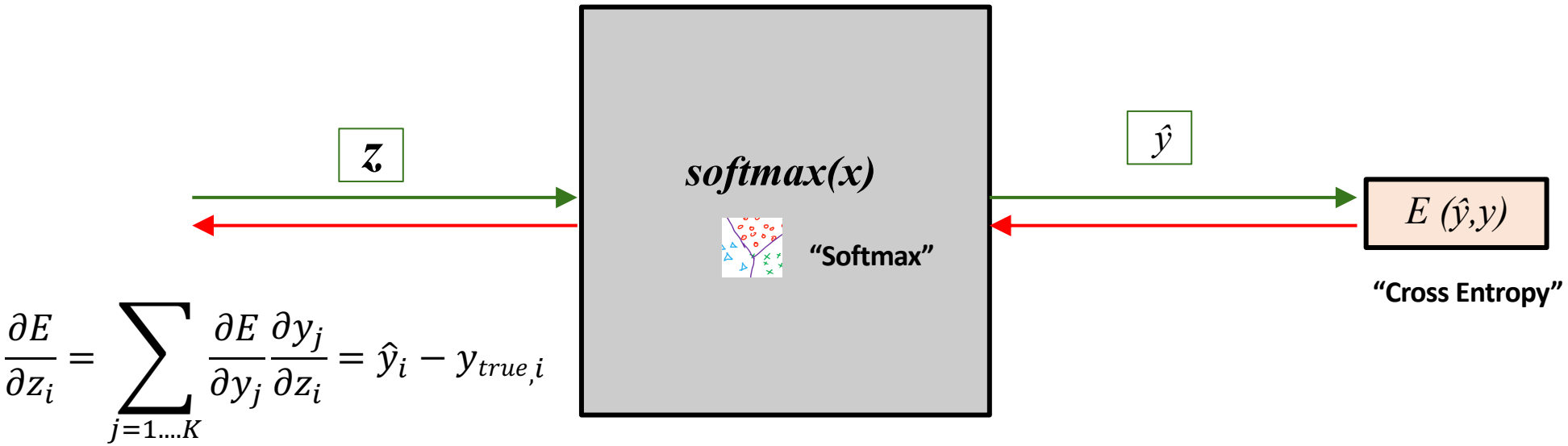
$E = g(y) = g(f(x)) \Rightarrow \frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial x}$

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

“Local-ness” of Backpropagation

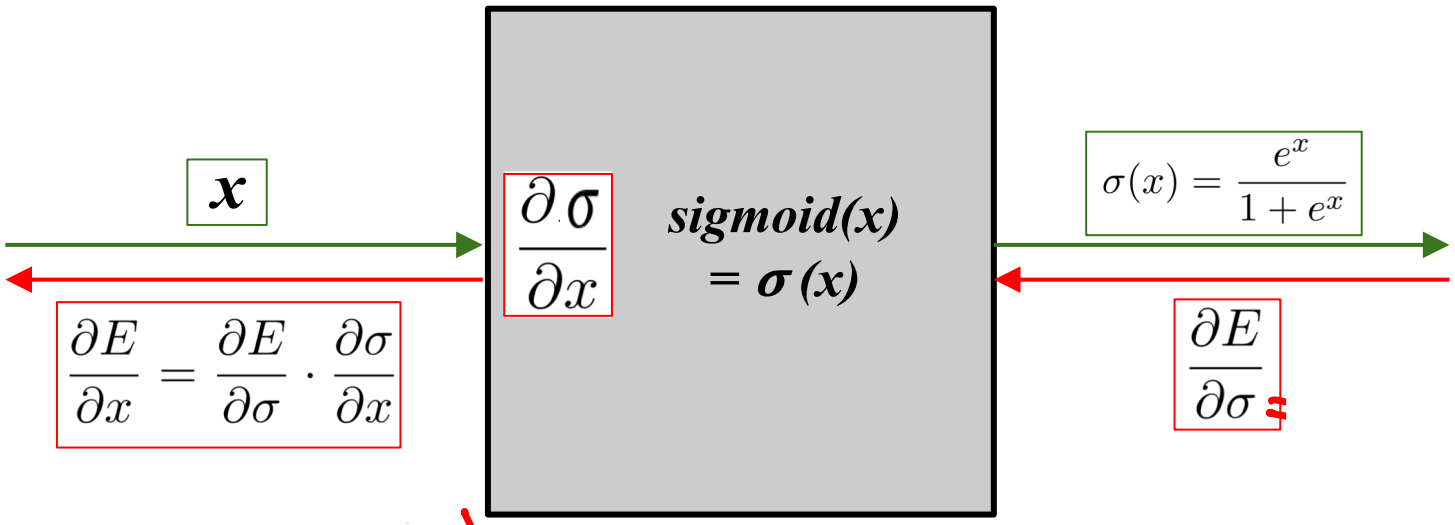


Example: Softmax Block (right before loss layer)

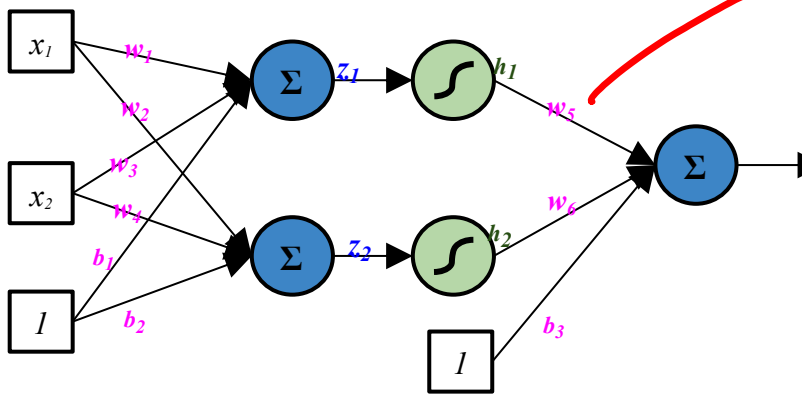


$$\frac{\partial E}{\partial z_i} = \sum_{j=1....K} \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_i} = \hat{y}_i - y_{true,i}$$

Example: Sigmoid Block



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	$E = (y - \hat{y})^2$

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

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

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

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

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

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
$x_1, x_2, w_1, ..$	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	$\partial E / \partial \hat{y} = -2(y-\hat{y})$

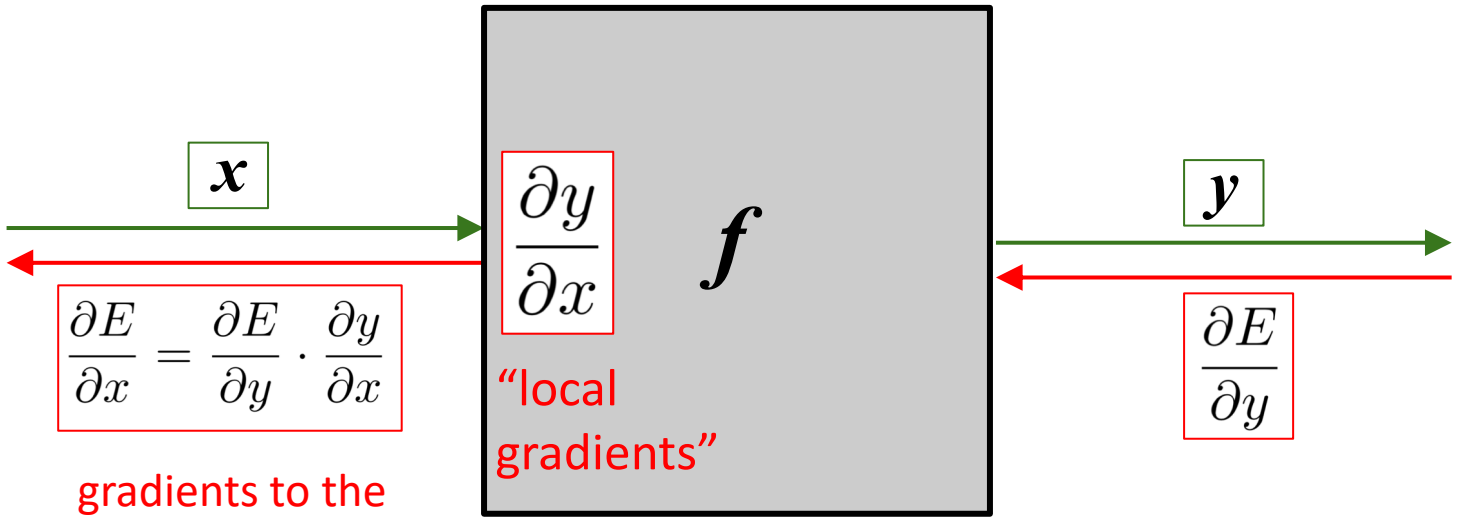
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial w_1} = \frac{\partial f_4}{\partial f_3} \frac{\partial f_3}{\partial f_2} \frac{\partial f_2}{\partial f_1} \frac{\partial f_1}{\partial w_1}$$

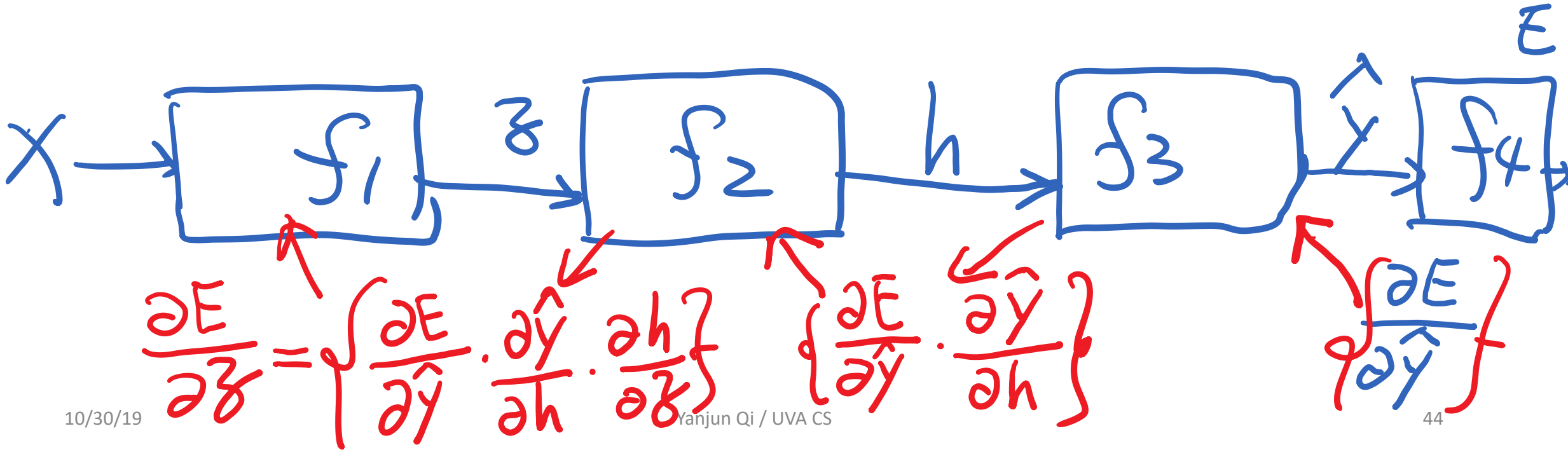
$$= -2(y-\hat{y}) \frac{\partial (h_1 w_5 + h_2 w_6 + b_3)}{\partial w_1}$$

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

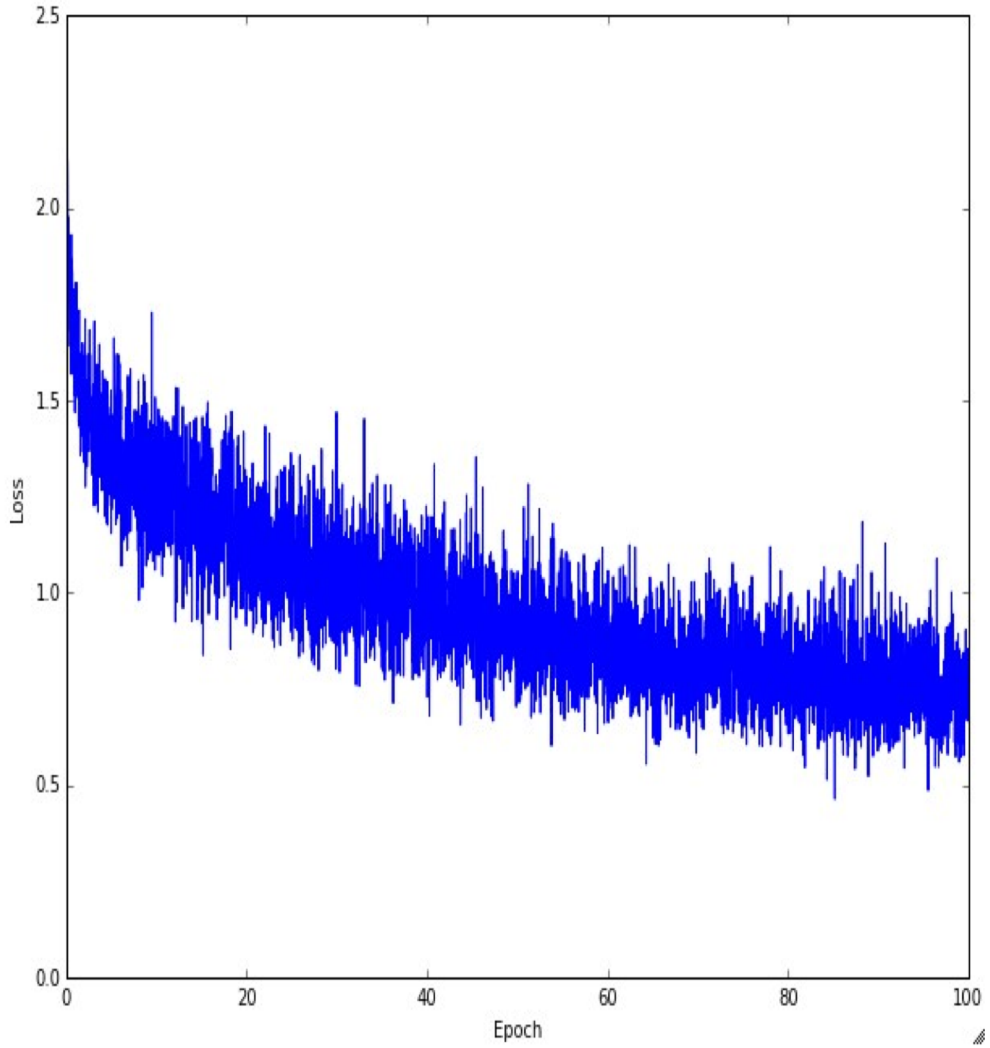
$$= -2(y-\hat{y}) w_5 \frac{\partial h_1}{\partial w_1} = -2(y-\hat{y}) w_5 \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

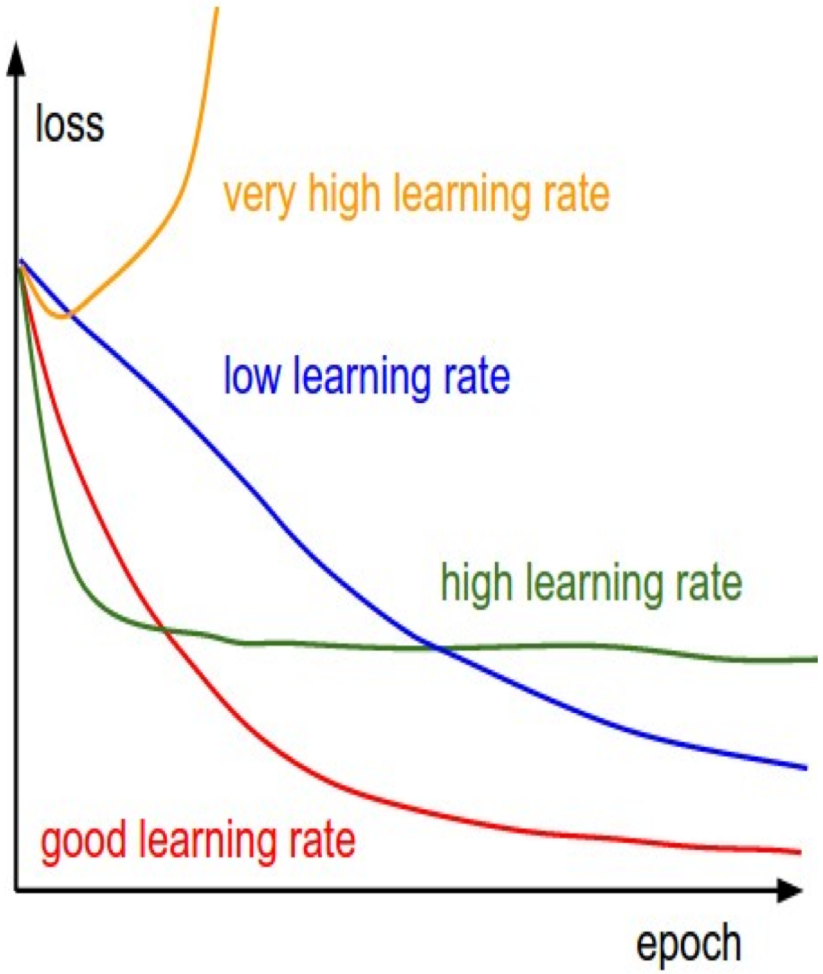
$$= \underbrace{-2(y-\hat{y})}_{f_4 \text{ local}} \underbrace{w_5}_{f_3 \text{ local}} \underbrace{h_1(1-h_1)}_{f_2 \text{ local}} x_1 \frac{\partial f_1}{\partial w_1}$$


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


X
f1
z
f2
h
f3
ŷ
f4
E
$\frac{\partial E}{\partial z} = \{\frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} \cdot \frac{\partial h}{\partial z}\}$
$\{\frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h}\}$
$\{\frac{\partial E}{\partial \hat{y}}\}$

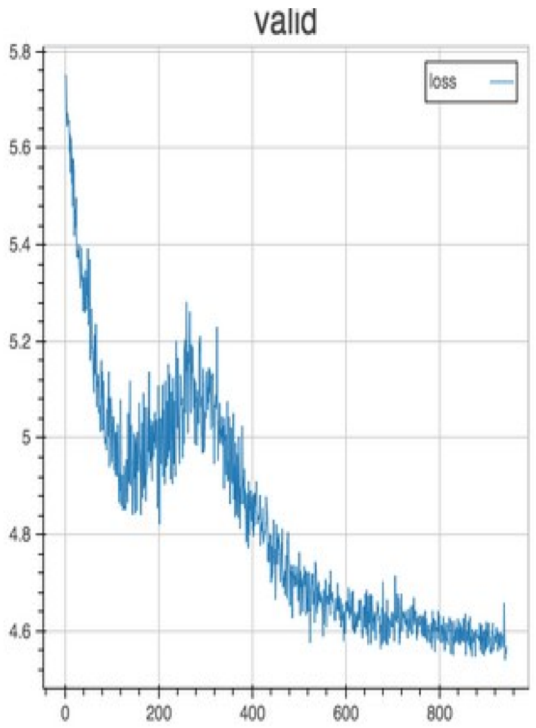
Monitor and visualize the loss curve

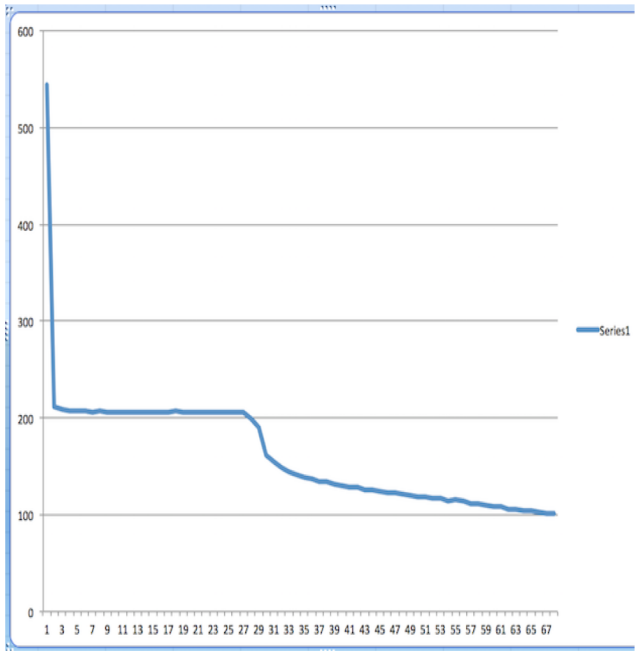


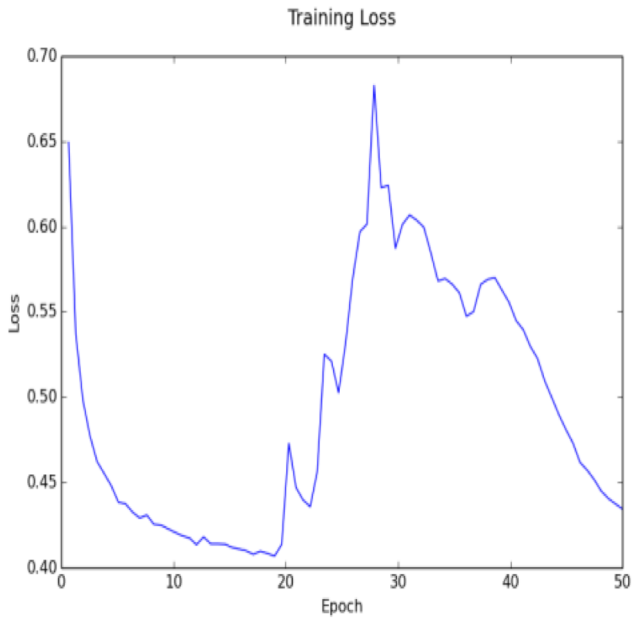


From Feifei Li Stanford Cousre

lossfunctions.tumblr.com



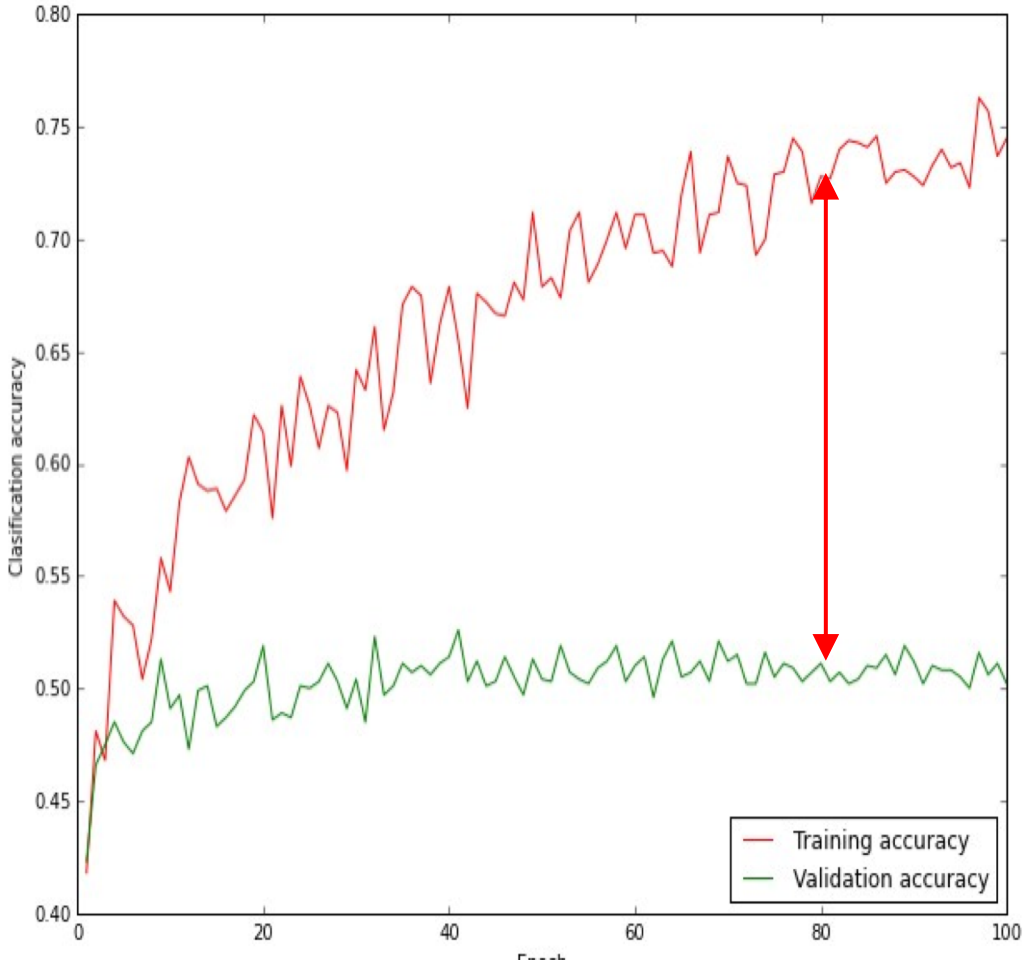




Other things to plot and check:

- 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

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



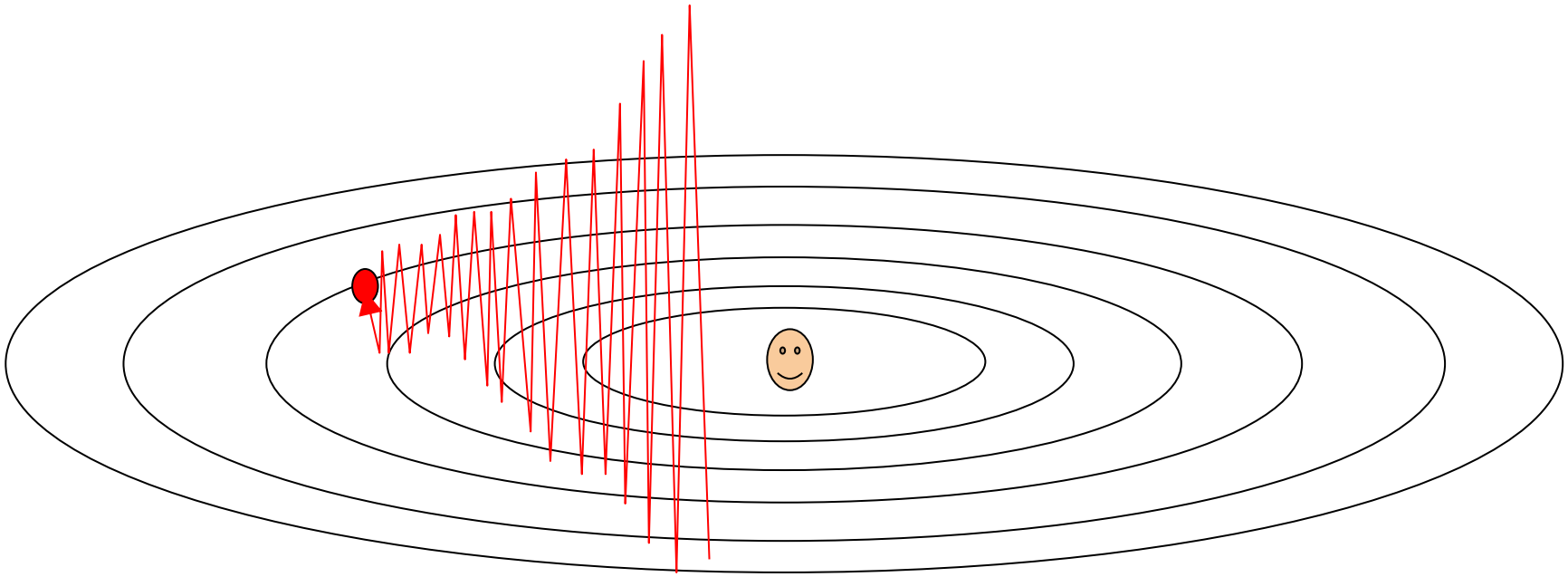
big gap = overfitting

=> increase regularization strength?

no gap, e.g. underfitting / both bad

=> increase model capacity?

Gradient Magnitudes:



Gradients too big → divergence
Gradients too small → slow convergence

Divergence is much worse!

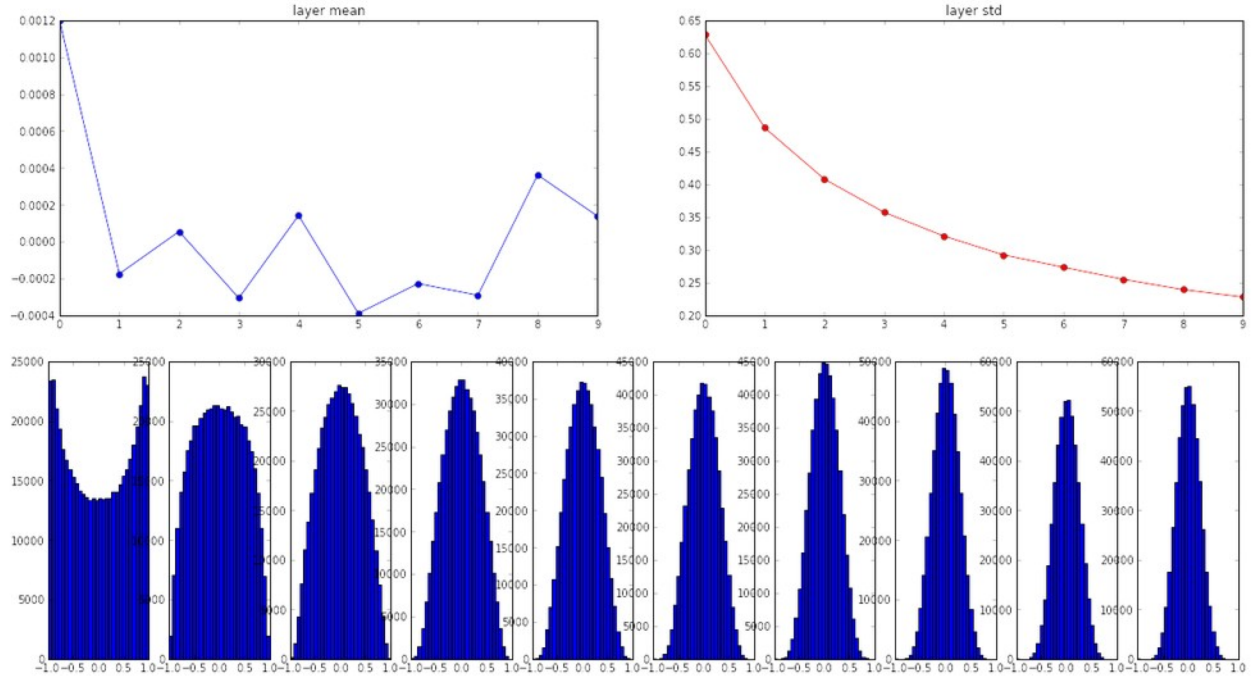
Many great tools, e.g., Adam
https://arxiv.org/abs/1609.04747

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

[Ioffe and Szegedy, 2015]

Normalize:

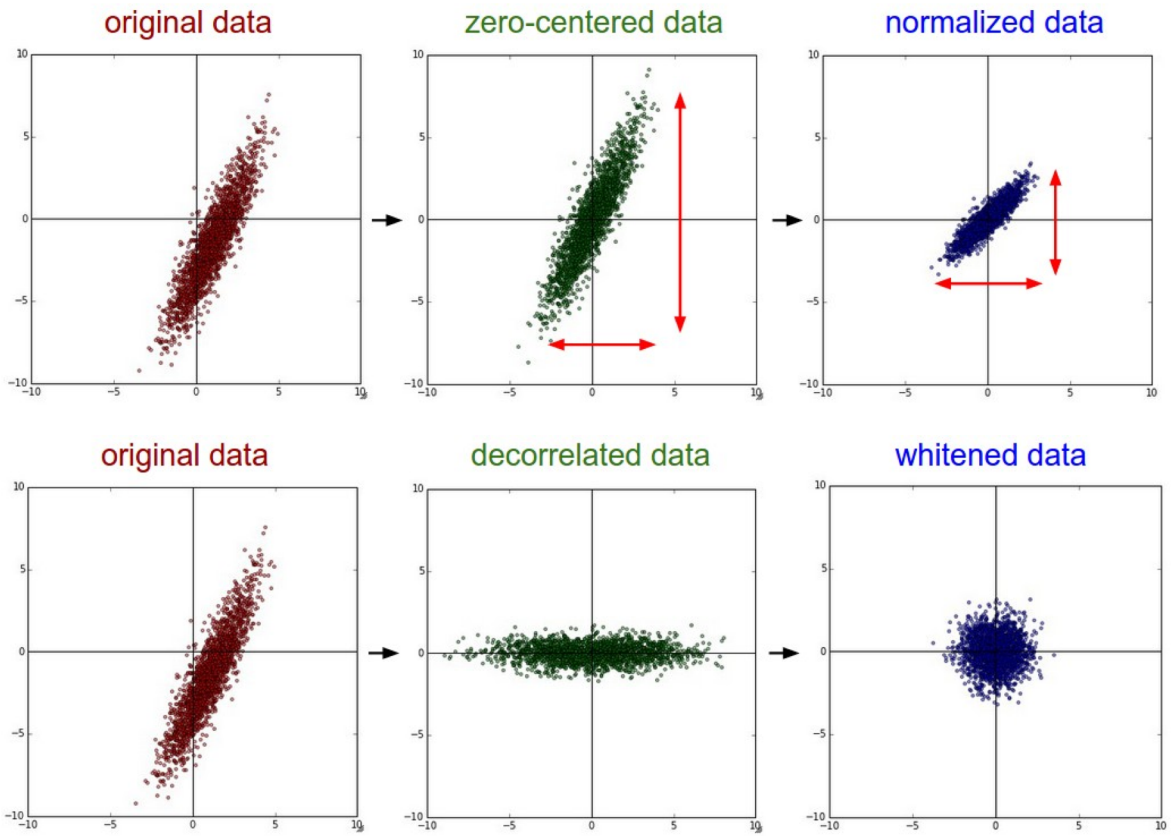
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

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

Data Preprocessing



From Feifei Li Stanford Cousre

Hyperparameters to play with:

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

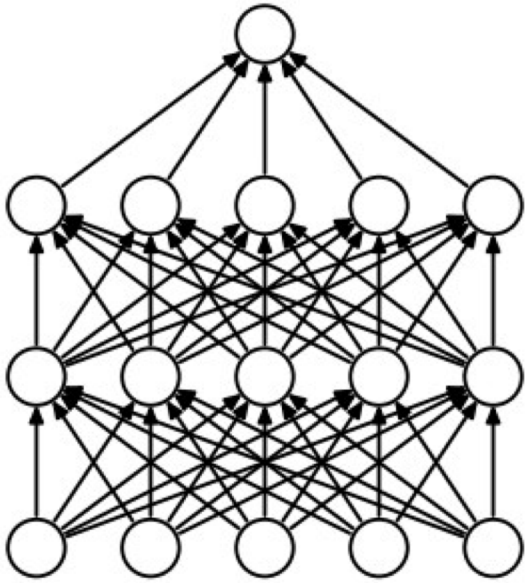
How to become a great neural networks practitioner

➔ Craft? / Talent? / Experience?

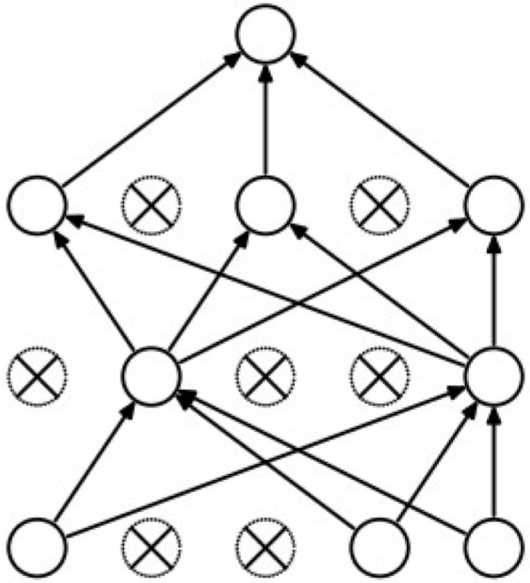
Your Friend: loss function

Regularization by Dropout

"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net

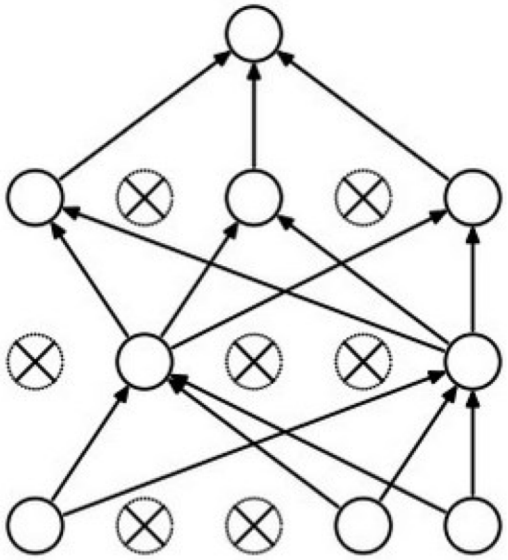


(b) After applying dropout.

[Srivastava et al., 2014]

Regularization by Dropout

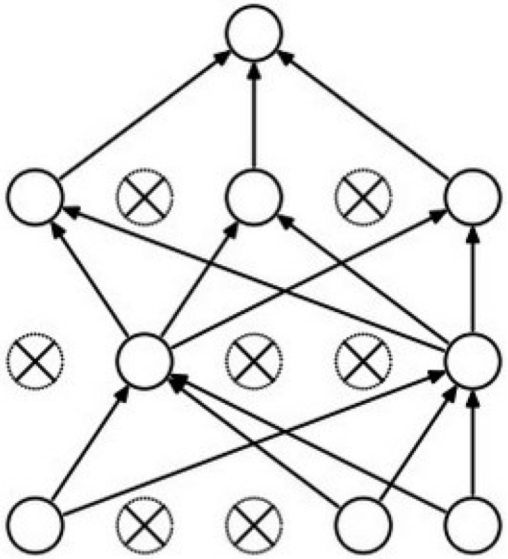
Waaaait a second…
How could this possibly be a good idea?



Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model

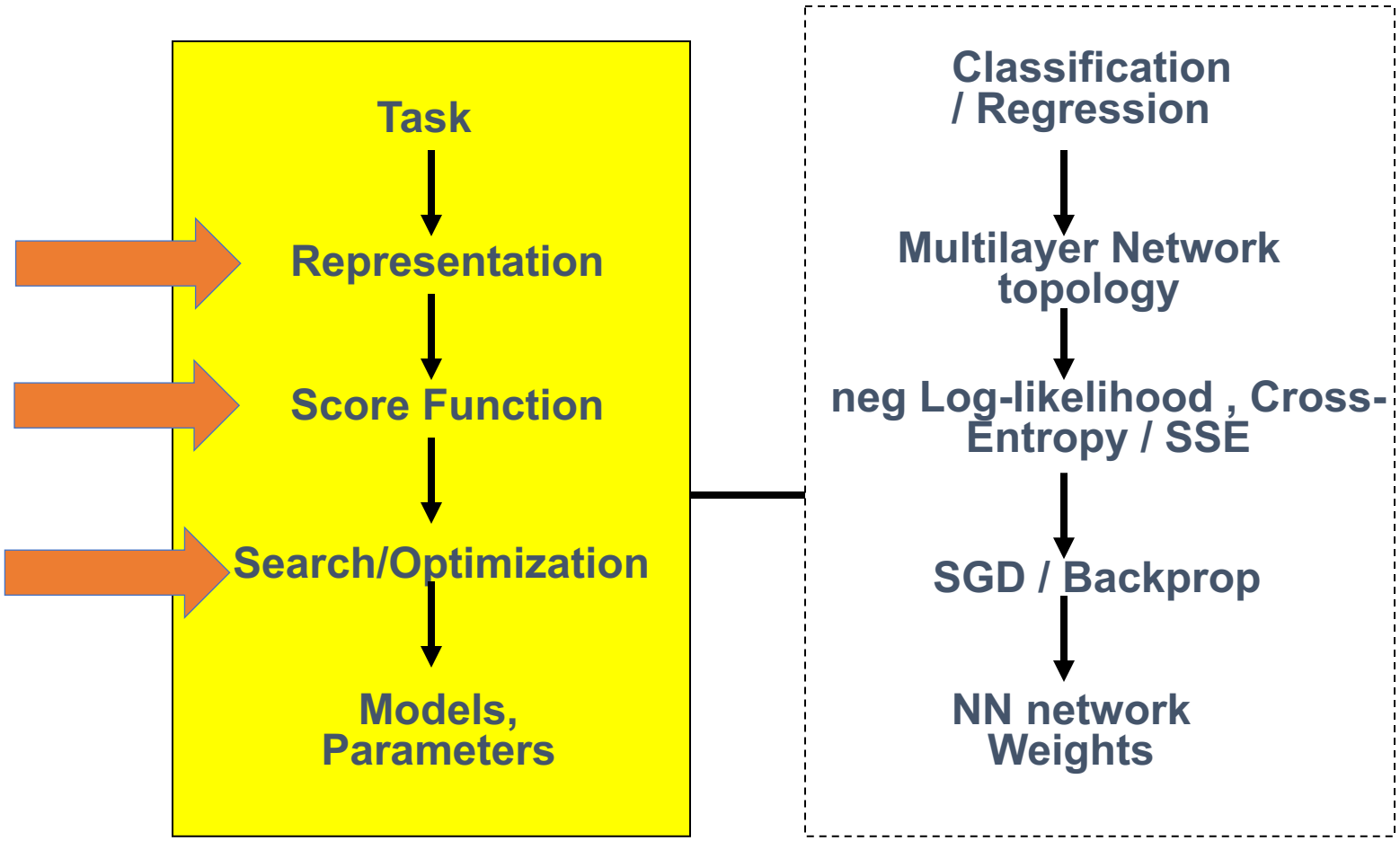
Dropout At test time….



Ideally:
want to integrate out all the noise

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

Today Recap: Basic Neural Network



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