DNN Introduction

講者：Isaac

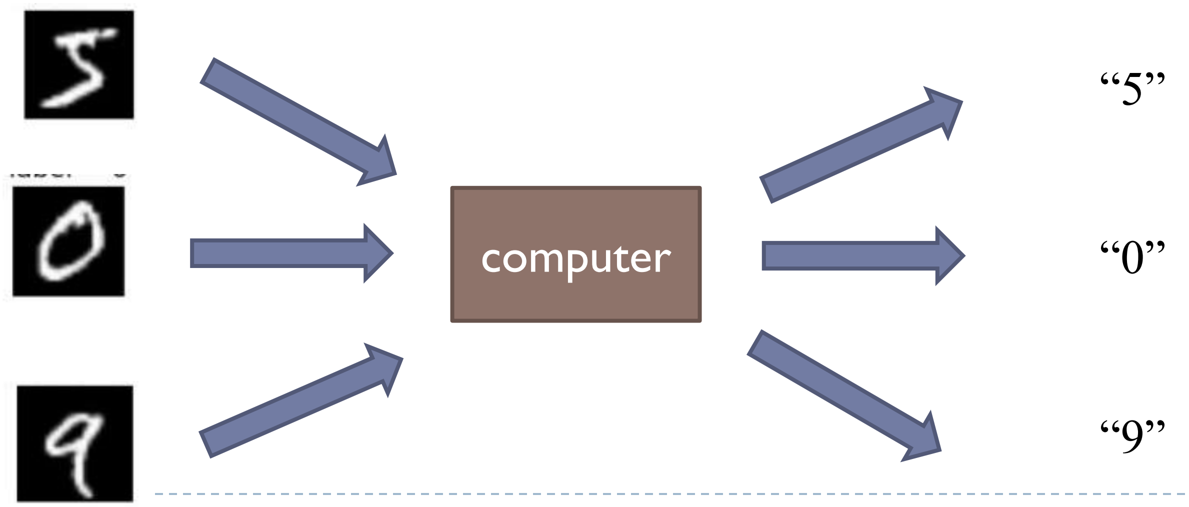
Outline

- Build DNN Model
- Define Loss Function
  - Mean square
  - Cross entropy with softmax
- Optimization
  - Gradient decent with moment
  - Adagrad, RMS, and Adam
- Predict/Validate Result

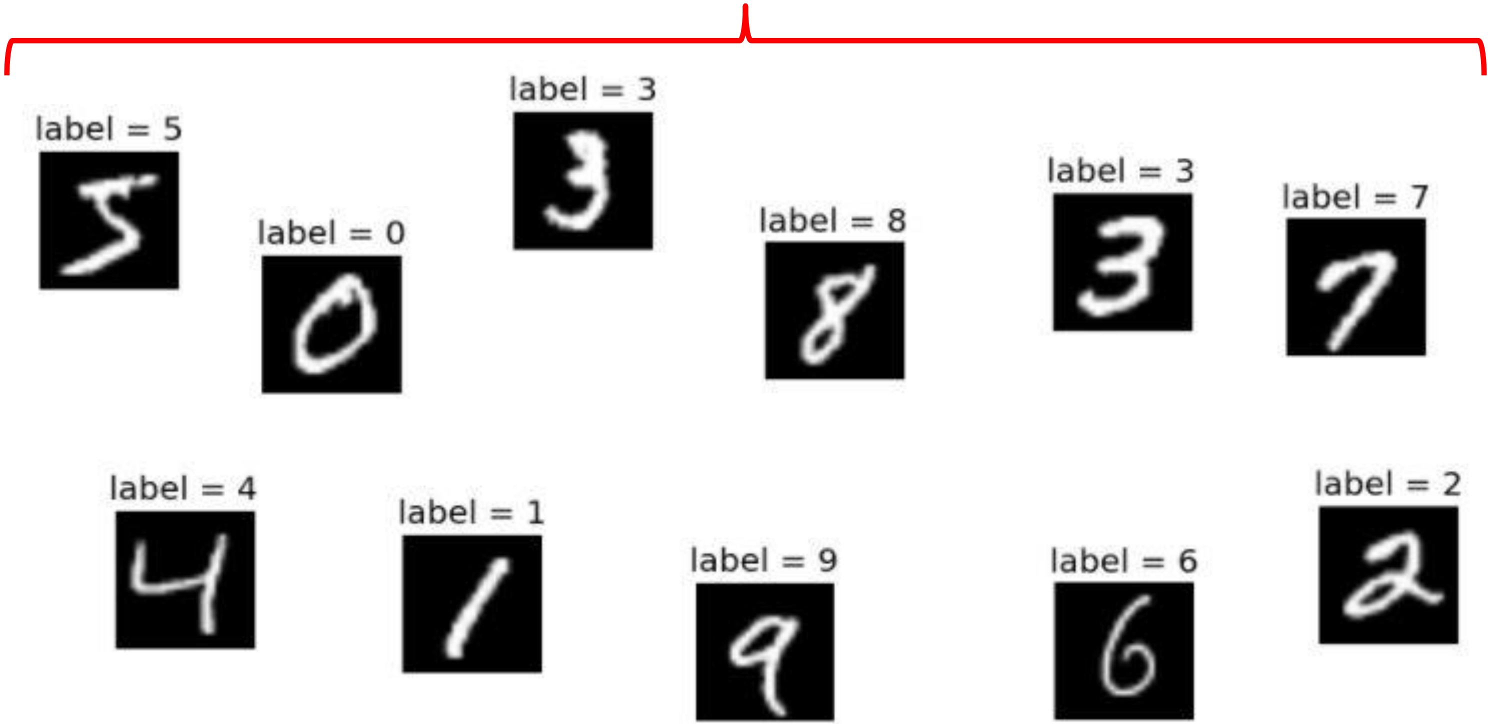
Big Picture

Same logic when implementation in TensorFlow

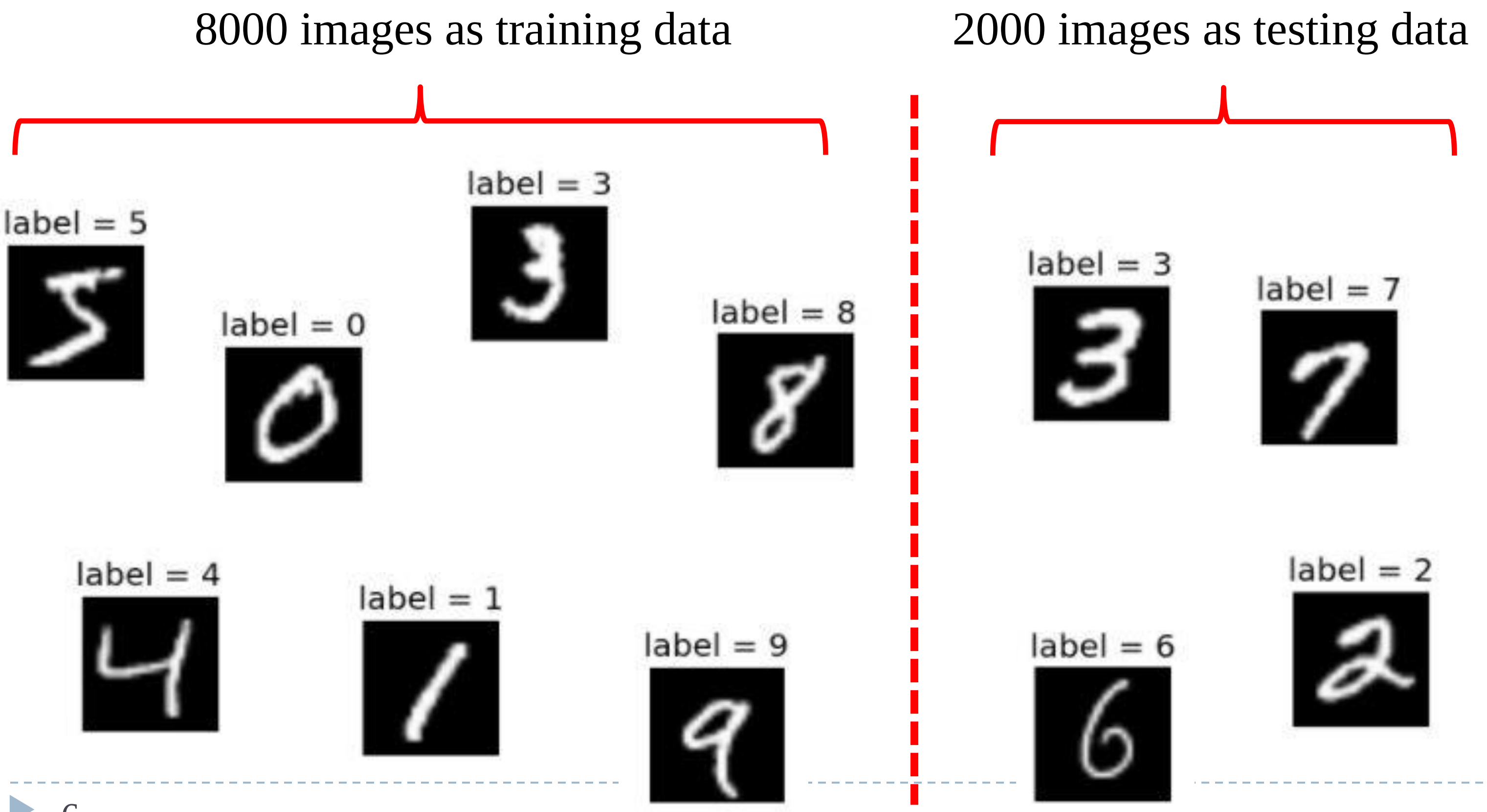
Assume we want to build a handwritten system to recognize image from 0 to 9



Total 10000 Images with labels



Split the data



Encode labels

8000 images as training data

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

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

One-hot encoding

One-hot encoding

What we want

$$f(x) \Rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \end{bmatrix} \Longleftrightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We want this two as close as possible

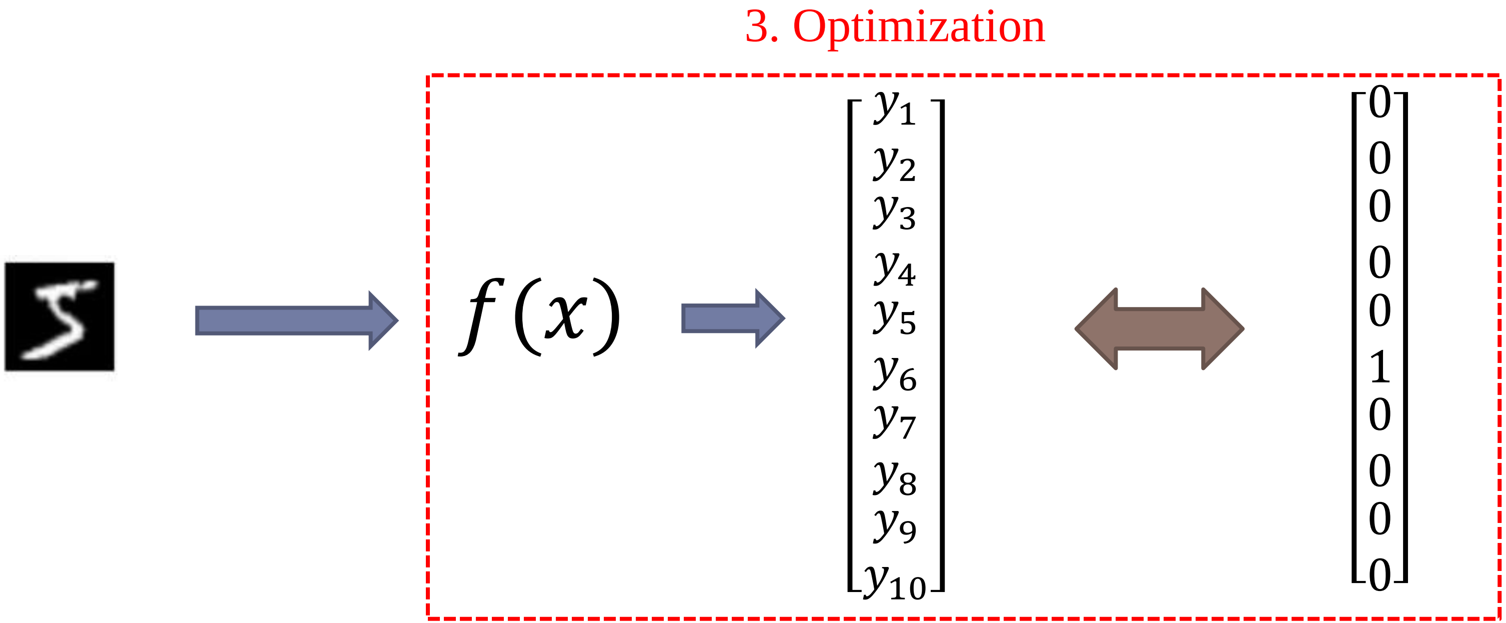
What we want

$f(x)$

1. Build DNN Model

2. Define Loss

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \end{bmatrix} \Longleftrightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$


3. Optimization
$f(x)$
$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \end{bmatrix}$
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

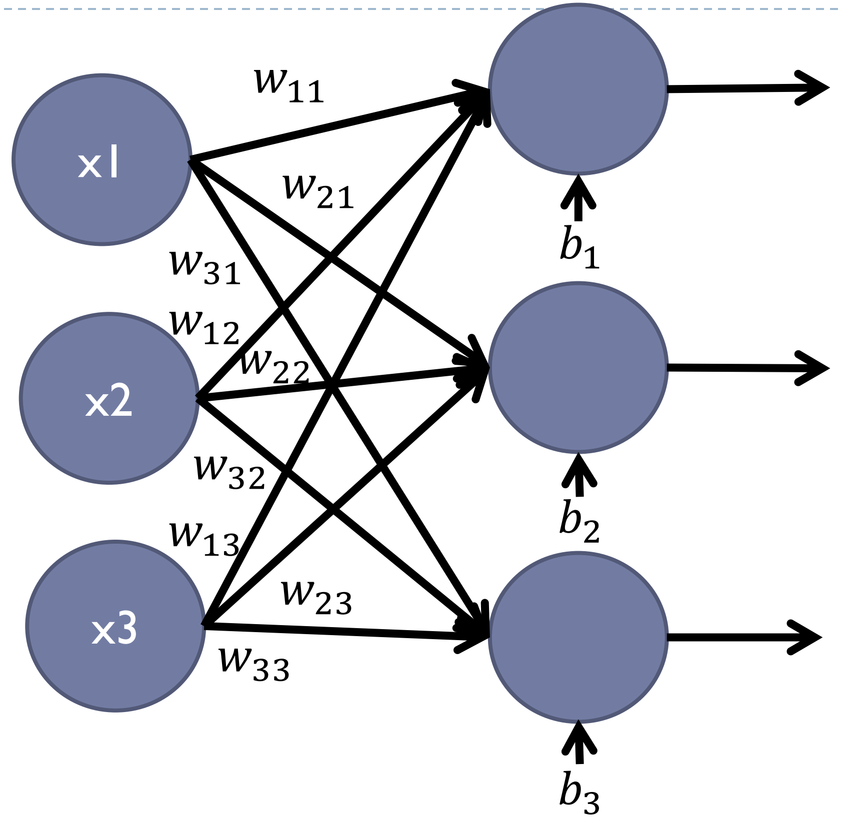
Finally, we get

$$f^*(x) \Rightarrow \begin{bmatrix} 0 \\ 0.2 \\ 0 \\ 0.1 \\ 0 \\ \boxed{0.8} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We get it is ‘5’

Build DNN Model

Model Overview

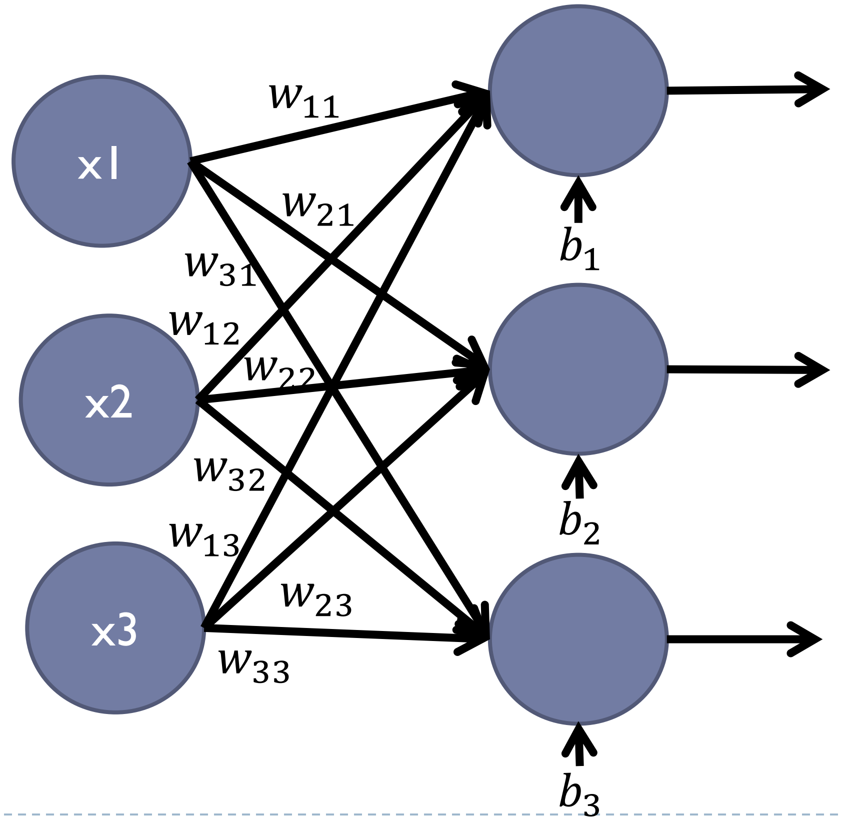


$$\sigma(x1 * w_{11} + x2 * w_{12} + x3 * w_{13} + b_1)$$

$$\sigma(x1 * w_{21} + x2 * w_{22} + x3 * w_{23} + b_2)$$

$$\sigma(x1 * w_{31} + x2 * w_{32} + x3 * w_{33} + b_3)$$

Model Overview

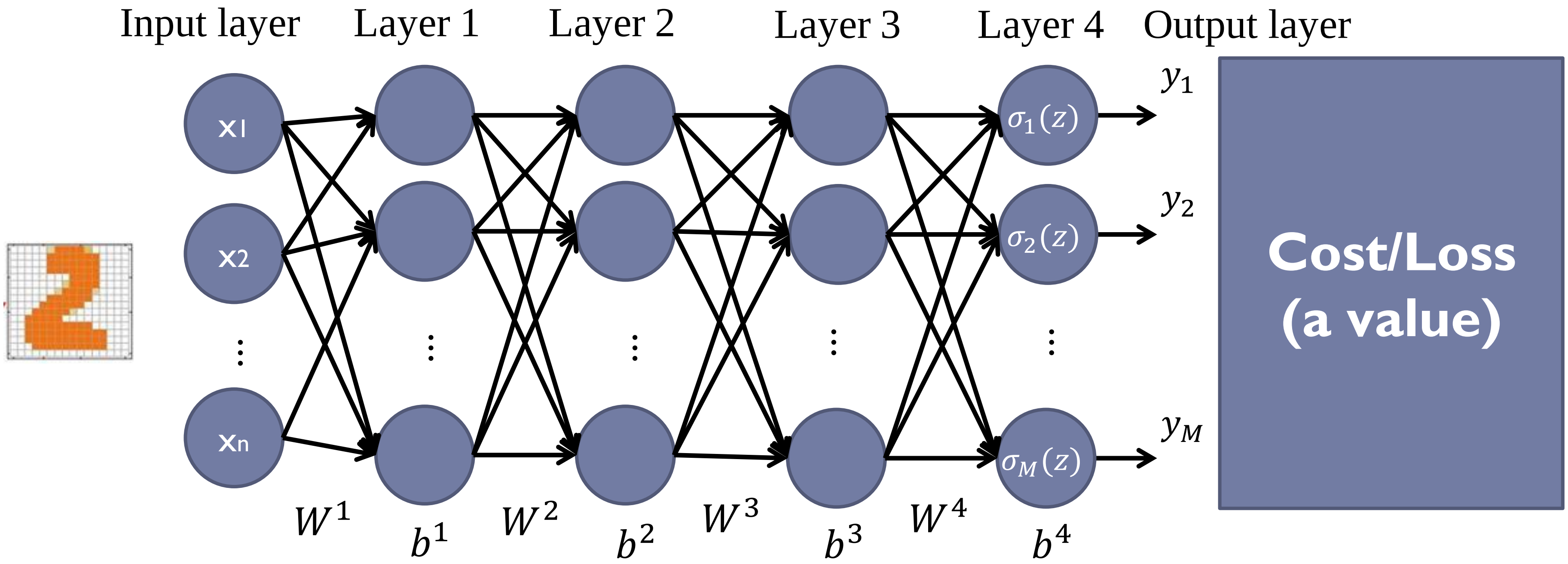


$$\sigma(W * X + b)$$

$$X = \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

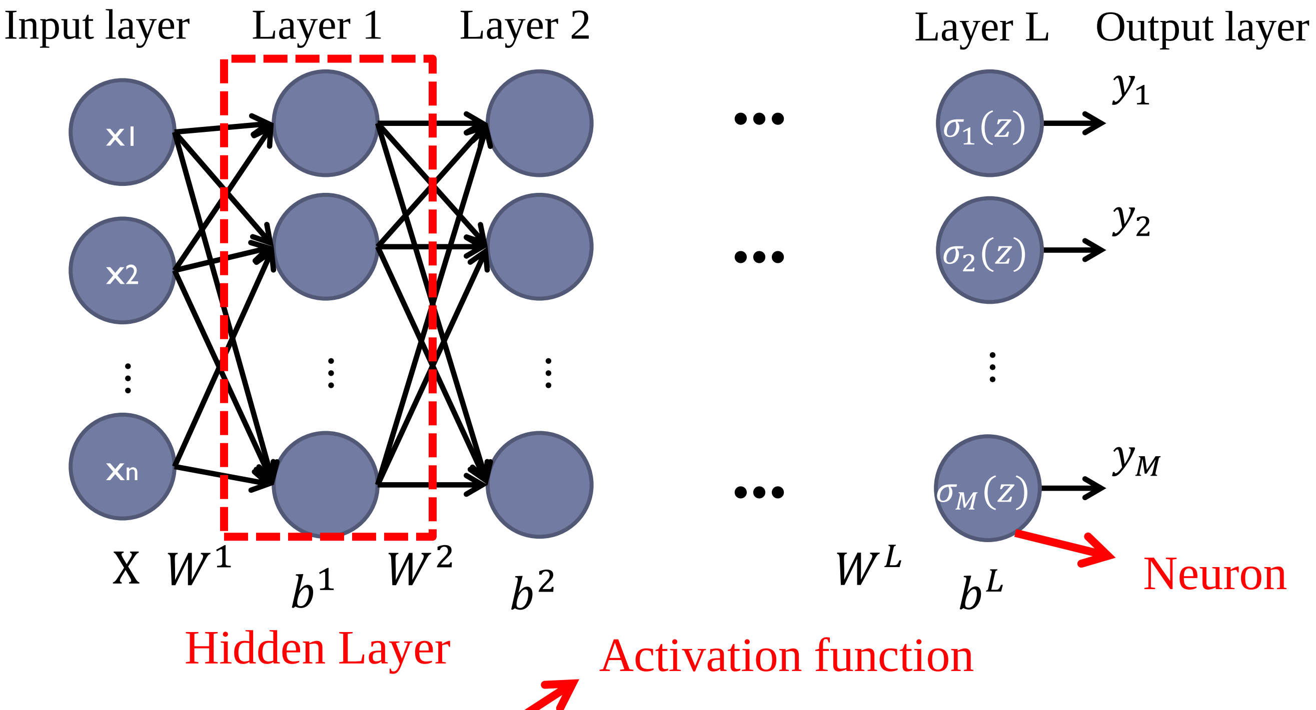
$$b = \begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}$$

Model Overview



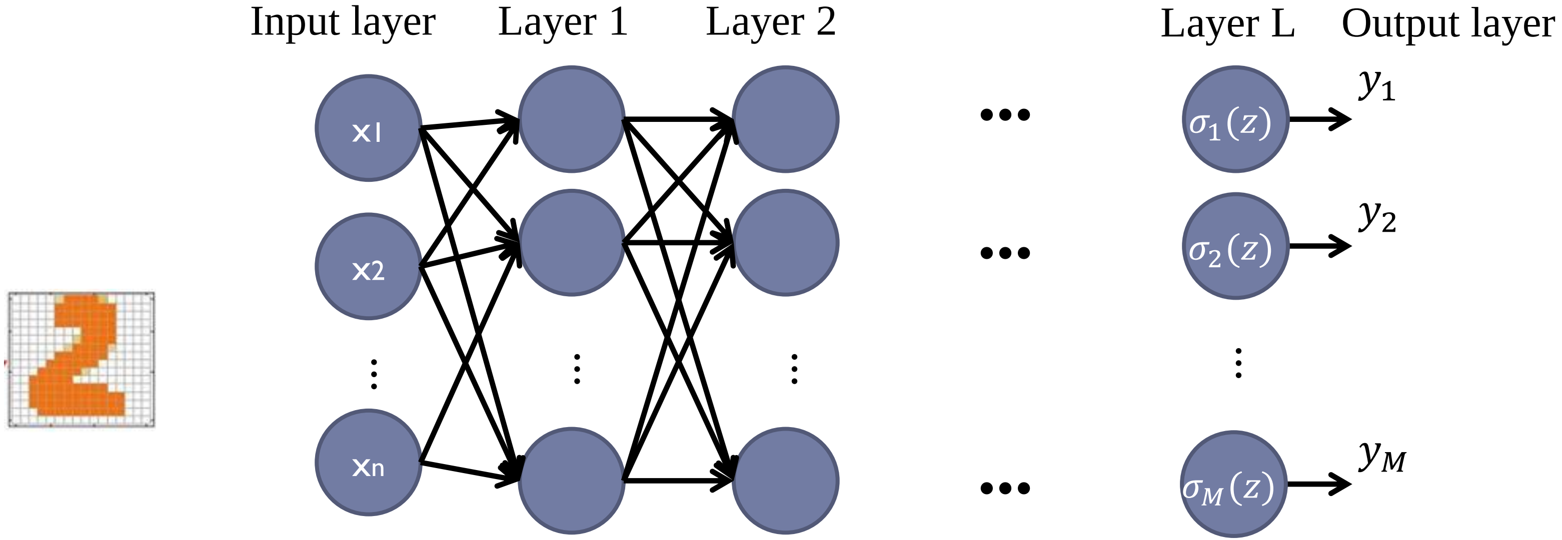
DNN also called “Multilayer perceptron”

Model Overview



$$y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 X + b^1) + b^2) \ldots + b^L)$$

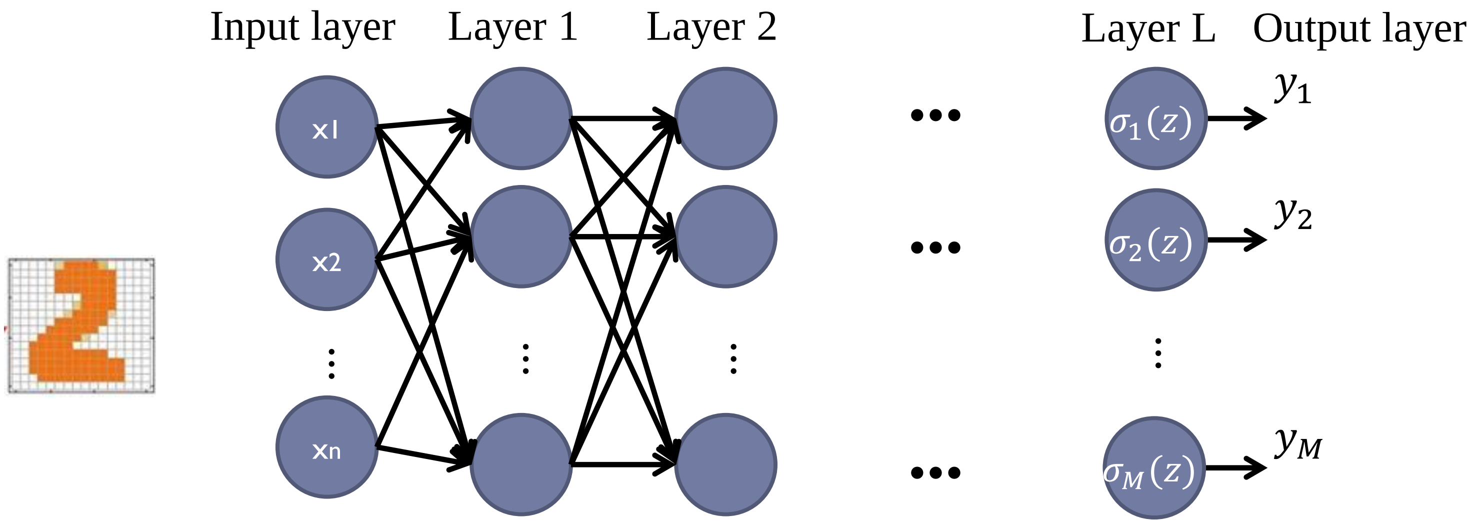
Model Overview



$$y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 X + b^1) + b^2) \ldots + b^L)$$

Weight (variable)   Bias (variable)

Model Overview



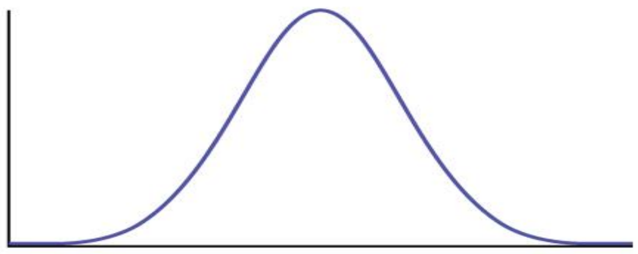
$$y = f(x) = \sigma(W^L \ldots \sigma(W^2\sigma(W^1X + b^1) + b^2) \ldots + b^L)$$

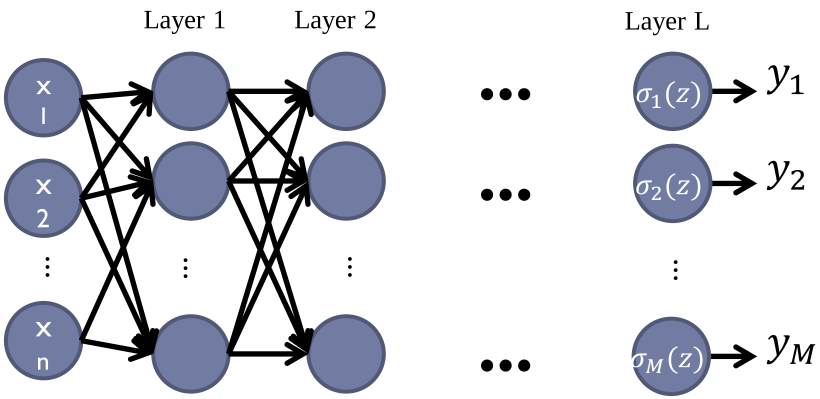
We want to find the best parameter set :

$$\theta = \{W^1, b^1, W^2, b^2, \ldots, W^L, b^L\}$$

Parameters Initialization

- The most common way are
  - Use random normal distribution with zero mean and small standard deviation
- In complex model, it is very important to find better way to initialize the parameters

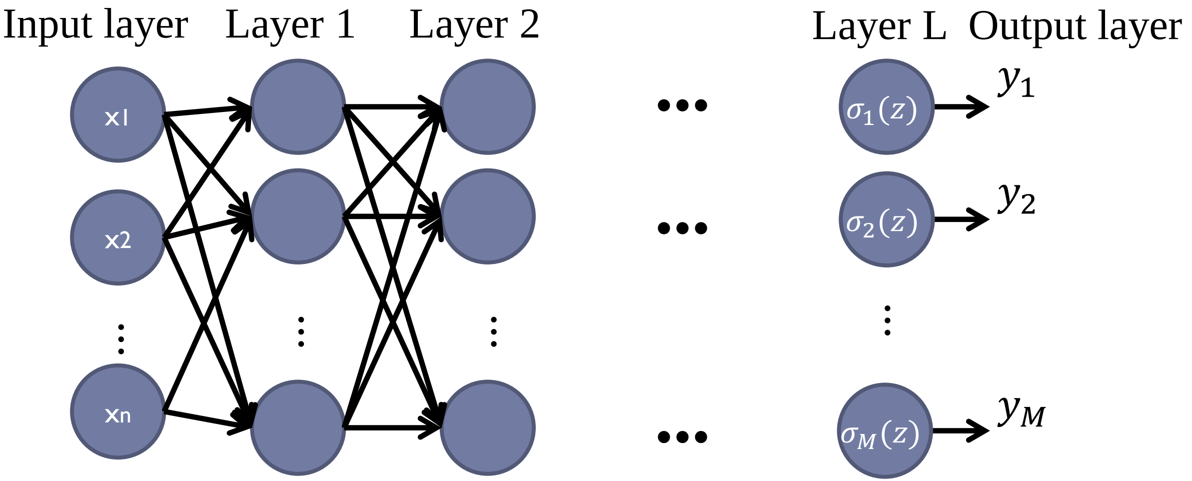




$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

Activation Function

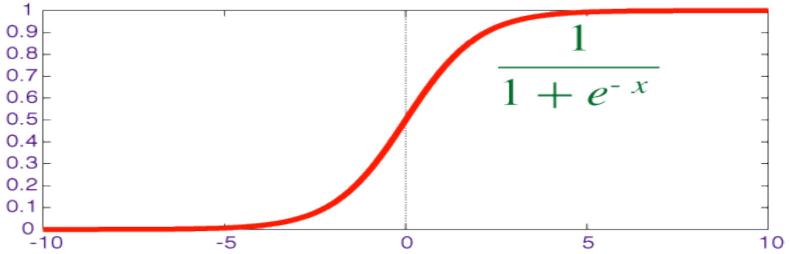
- Give NN nonlinearity property
- Can be regarded as "ON" (1) or "OFF" (0)



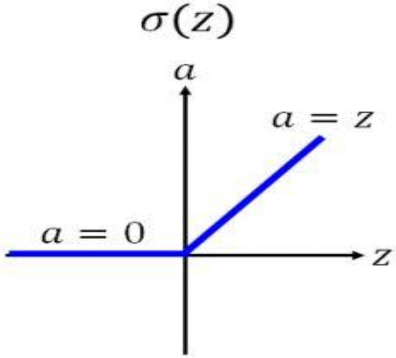
$$y = f(x) = \sigma(W^L \dots \sigma(W^2\sigma(W^1X + b^1) + b^2) \dots + b^L)$$

Activation Function

- There are many kinds of activation
  - relu, sigmoid, elu, etc……
- Usually, we use relu as first try on building neural network



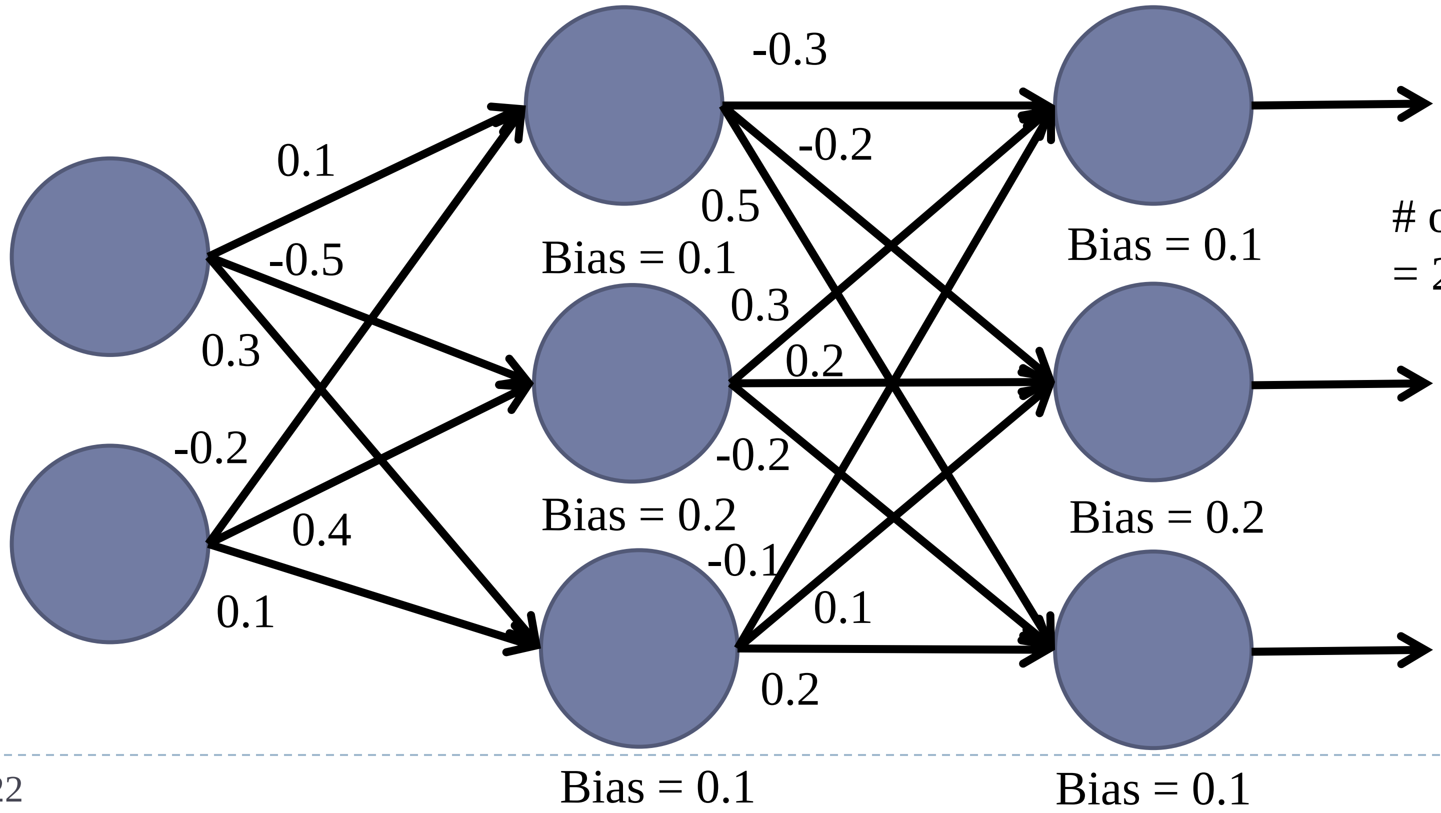
Sigmoid function



RELU

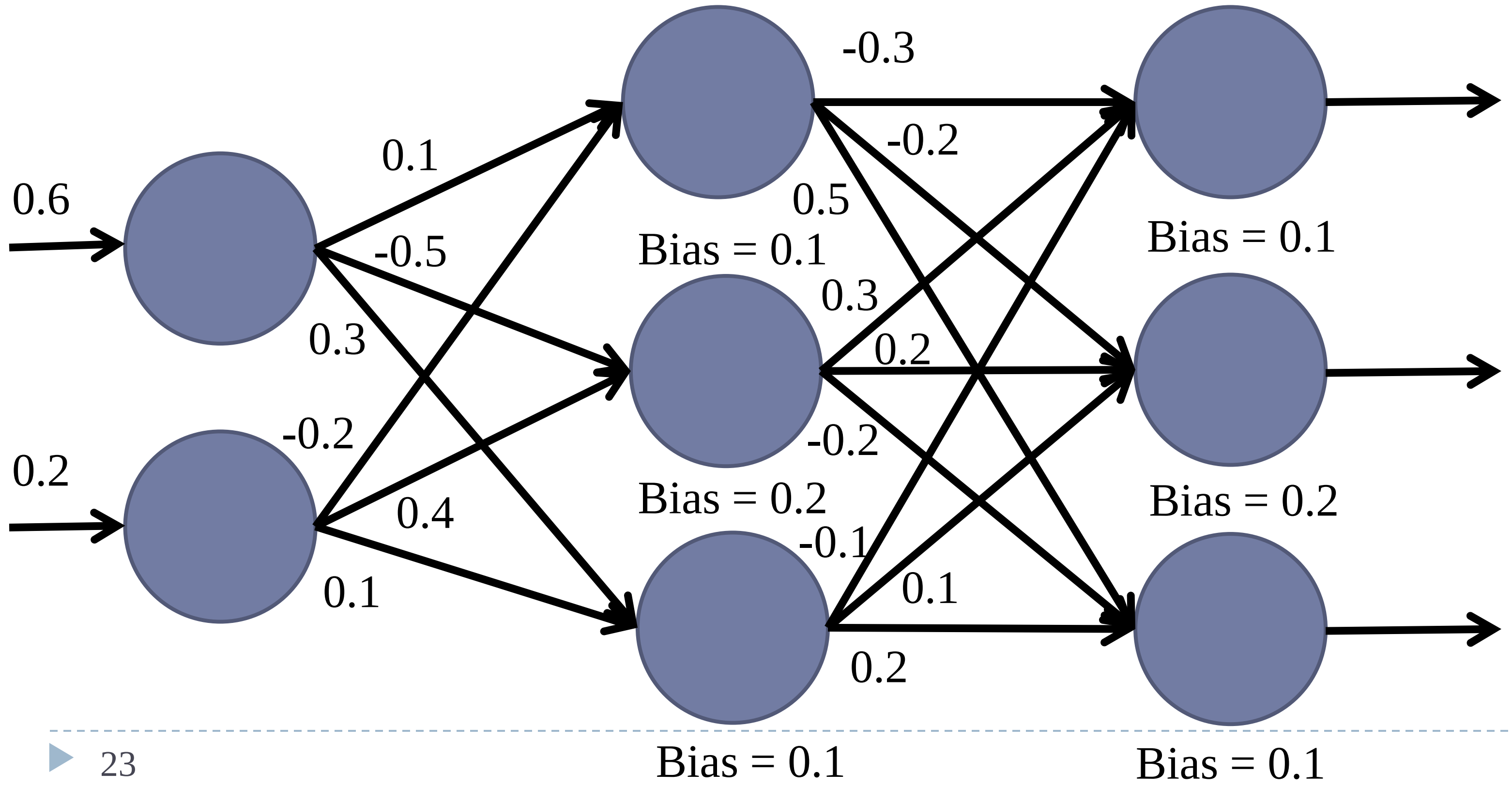
We usually use this now !

Example-initialization

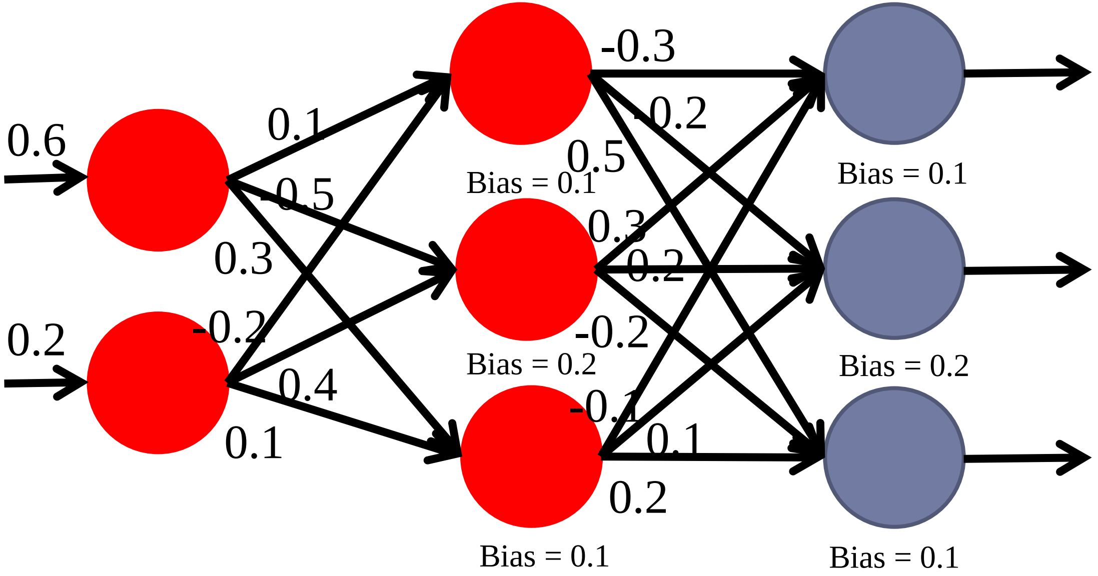


# of parameters = 21

Example-feed data



Example-forward pass

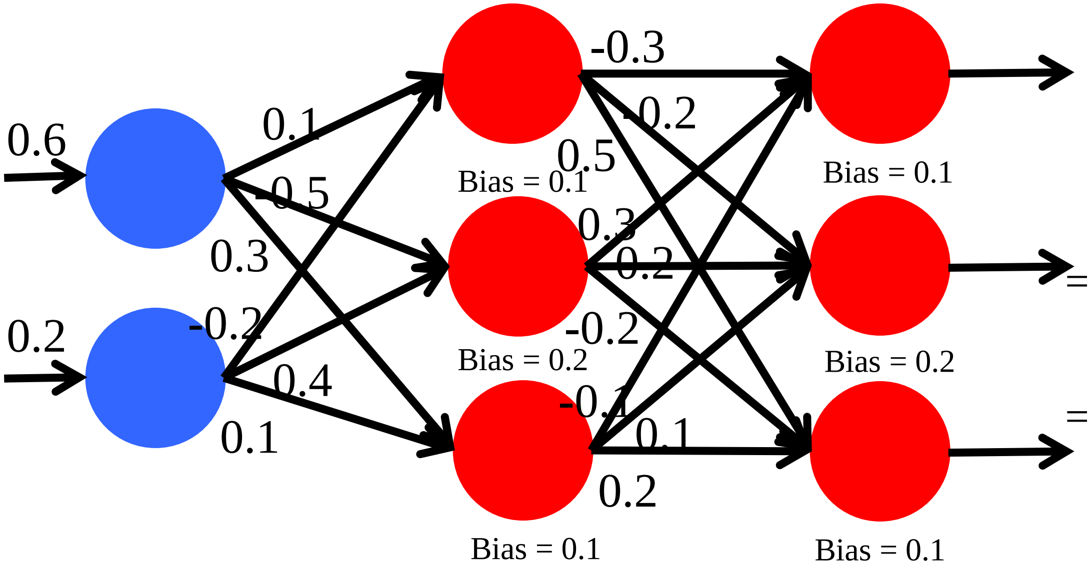


$$relu\left(\begin{bmatrix} 0.1 & -0.2 \\ -0.5 & 0.4 \\ 0.3 & 0.1 \end{bmatrix}\begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix} 0.12 \\ -0.02 \\ 0.3 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.12 \\ 0 \\ 0.3 \end{bmatrix}$$

Example-forward pass



$$relu\left(\begin{bmatrix} -0.3 & 0.3 & -0.1 \\ -0.2 & 0.2 & 0.1 \\ 0.5 & -0.2 & 0.2 \end{bmatrix}\begin{bmatrix} 0.12 \\ 0 \\ 0.3 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix} -0.066 \\ 0.006 \\ 0.12 \end{bmatrix}\right)$$

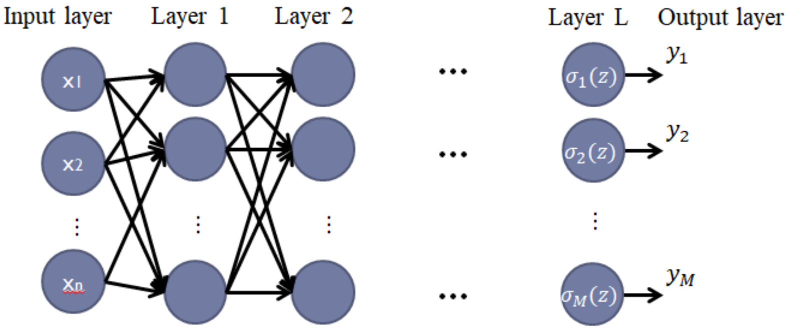
$$= \begin{bmatrix} 0 \\ 0.006 \\ 0.12 \end{bmatrix}$$

Define Loss Function

Loss Function

- There are many kinds of loss function
- Usually, it is a function that map multi-variables to a single value
- We will introduce two loss function in DNN
  - Mean square
  - Cross-entropy

What is one-hot encode



$$\begin{array}{cc} \begin{bmatrix} 0.15 \\ 0.2 \\ 0.3 \\ \vdots \end{bmatrix} & \begin{array}{c} \text{“0”} \\ \text{“1”} \\ \text{“2”} \\ \end{array} \end{array} \qquad \begin{array}{cc} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} & \begin{array}{c} \text{“0”} \\ \text{“1”} \\ \text{“2”} \\ \end{array} \end{array}$$

Output of NN look like this

What we want (one hot encode)

Mean Square

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

$$\begin{bmatrix} 0.15 \\ 0.2 \\ 0.3 \\ \vdots \end{bmatrix} \begin{array}{l} \text{“0”} \\ \text{“1”} \\ \text{“2”} \\ \end{array} \qquad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{array}{l} \text{“0”} \\ \text{“1”} \\ \text{“2”} \\ \end{array}$$

Cross-entropy with Softmax

- Information
  - $log\left(\frac{1}{p_i}\right)$ *where* p_i *is probability of an event*

Sun rises in the east tomorrow

It will rain tomorrow in Taiwan

Which is more informative?

Cross-entropy with Softmax

- Entropy
  - Expected value(mean) of information contained in each message
- Entropy can be seen as index of uncertainty
  - Bigger mean more chaos
- Cross-entropy
  - Measurement on the difference between two probability distribution
  - Different distribution apply on entropy
  - Cross-entropy is greater than entropy

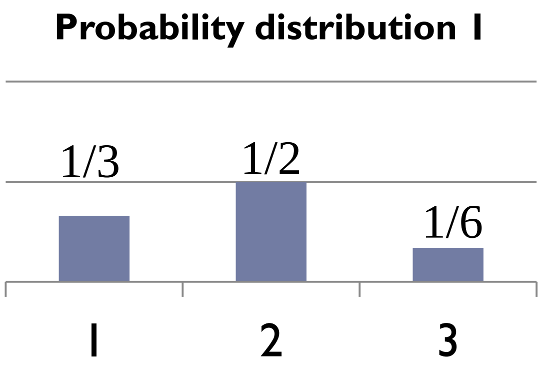
$$H(y) = \sum_i y_i \log \frac{1}{y_i} = -\sum_i y_i \log y_i$$

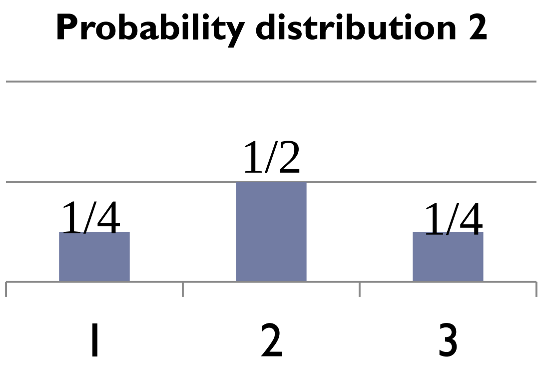
Entropy

$$\boldsymbol{H(y) = -\sum_i y_i log \hat{y}_i}$$

Cross-entropy

Example





Entropy on distribution 1
= 1/3 * log(3) + 1/2 * log(2) + 1/6 * log(6)

Entropy on distribution 2
= 1/4 * log(4) + 1/2 * log(2) + 1/4 * log(4)

Cross-entropy on distribution 1 over distribution 2
= 1/3 * log(4) + 1/2 * log(2) + 1/6 * log(4)

Cross-entropy on distribution 2 over distribution 1
= 1/4 * log(3) + 1/2 * log(2) + 1/4 * log(6)

Example

Entropy on distribution 1
= 1/3 * log(3) + 1/2 * log(2) + 1/6 * log(6)
= 0.439

Entropy on distribution 2
= 1/4 * log(4) + 1/2 * log(2) + 1/4 * log(4)
=0.452

Cross-entropy on distribution 1 over distribution 2
= 1/3 * log(4) + 1/2 * log(2) + 1/6 * log(4) = 0.456

Cross-entropy on distribution 2 over distribution 1
= 1/4 * log(3) + 1/2 * log(2) + 1/4 * log(6) = 0.464

- Cross-entropy is greater than entropy

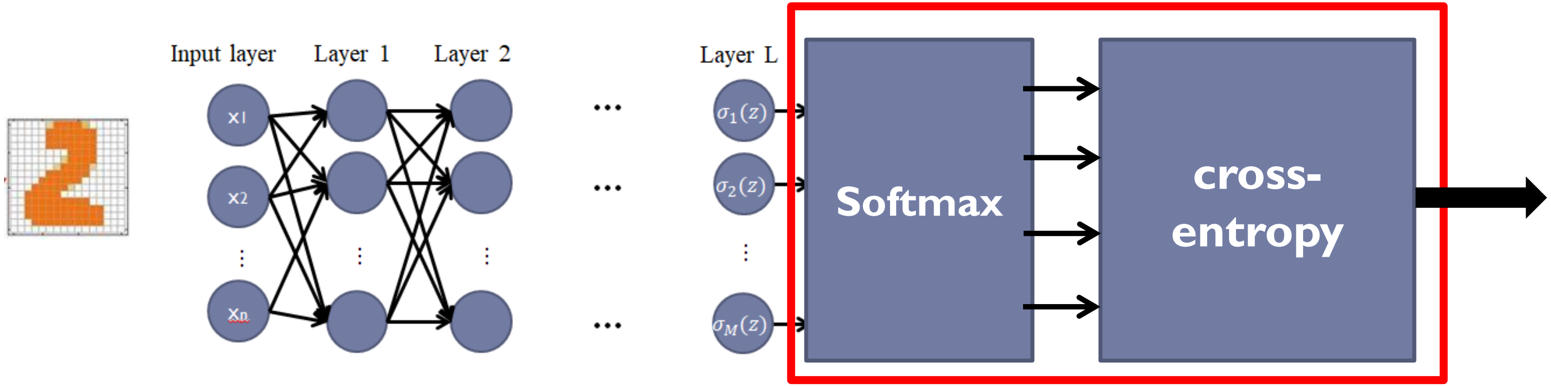
  Cross-entropy on distribution 1 over 2 > Entropy on distribution 1

  Cross-entropy on distribution 2 over 1 > Entropy on distribution 2
- If two distribution become closer
  - Value of cross-entropy is closer to entropy

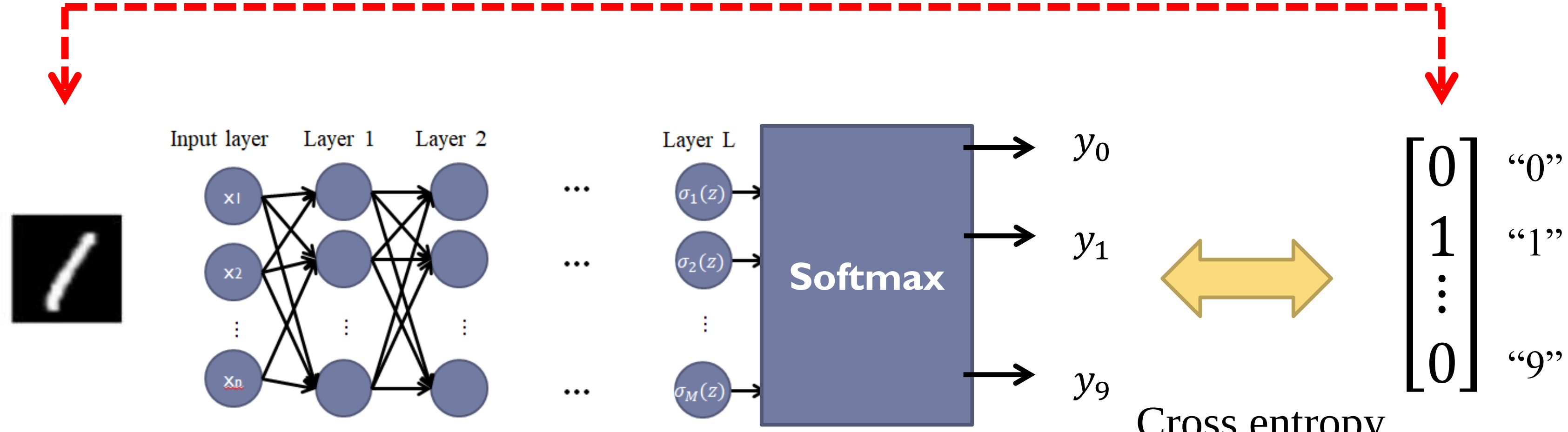
Cross-entropy with Softmax

- Cross-entropy usually come with softmax layer in NN
- Softmax function squash all of elements in vector to [0, 1]

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



Cross-entropy with Softmax

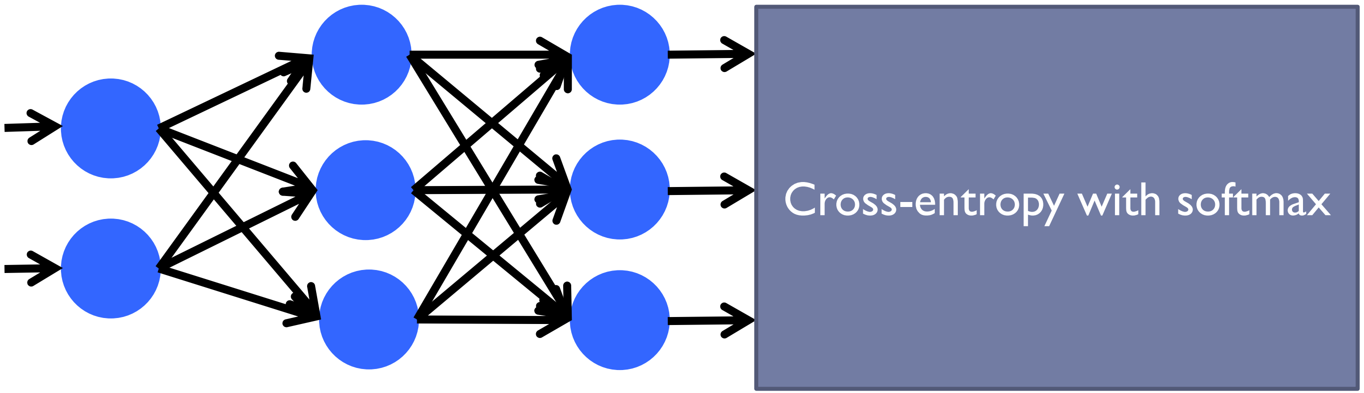


Cross entropy

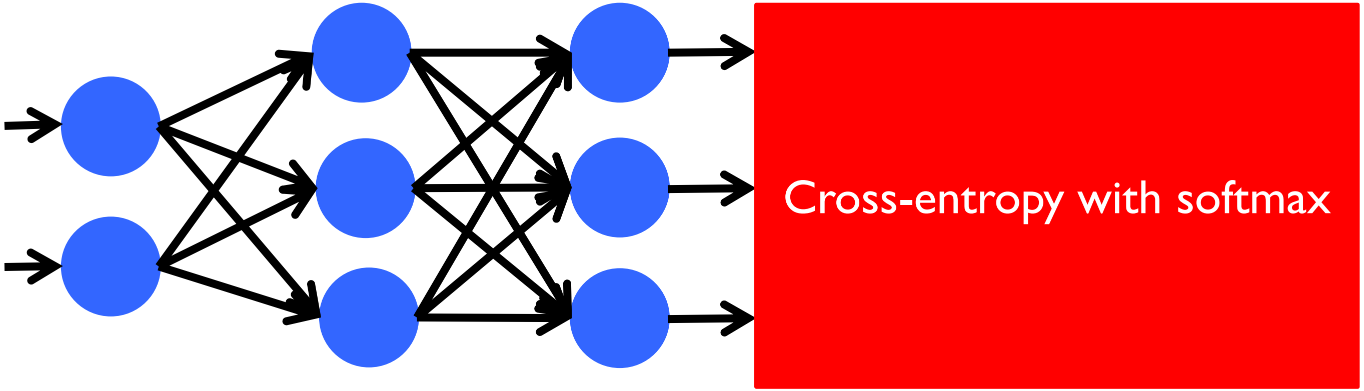
$$\boldsymbol{H(y) = -\sum_i y_i log \hat{y}_i}$$

$$Cost = -(0 * \log(y_0) + 1 * \log(y_1) + 0 * \log(y_2) + \cdots + 0 * \log(y_9))$$

Example-forward pass



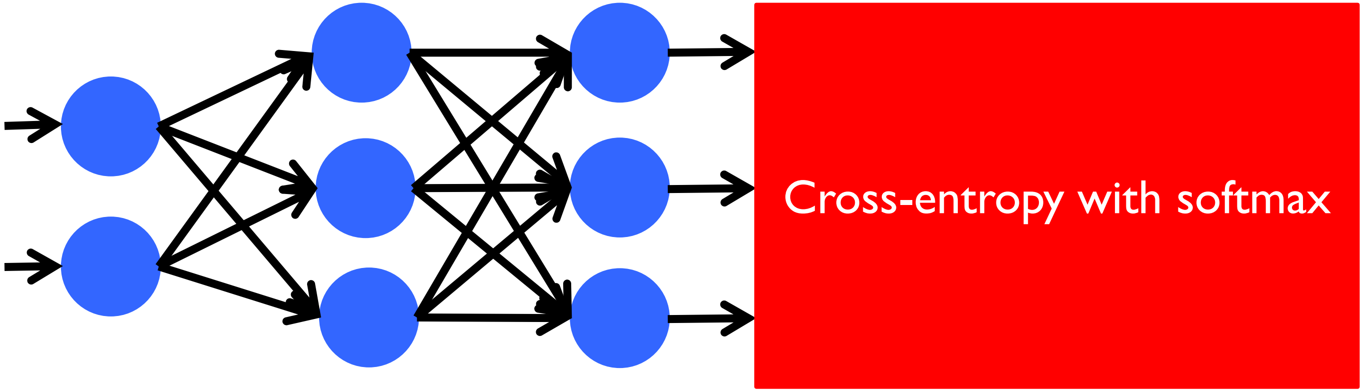
Example-forward pass



$$\text{Softmax}\left(\begin{bmatrix} 0 \\ 0.006 \\ 0.12 \end{bmatrix}\right) = \begin{bmatrix} 0.319 \\ 0.321 \\ 0.36 \end{bmatrix}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Example-forward pass



What we expect (label)

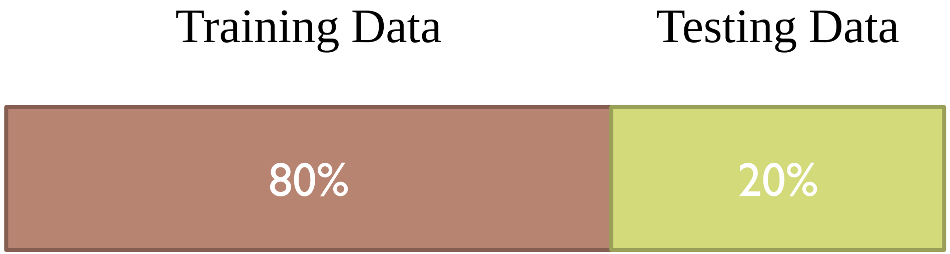
$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- 0 * ln(0.319) – 1 * ln(0.321) – 0 * ln(0.36)
= 1.1363

$$-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i)$$

This is large at the beginning. During training, this should decrease.

How to Feed data



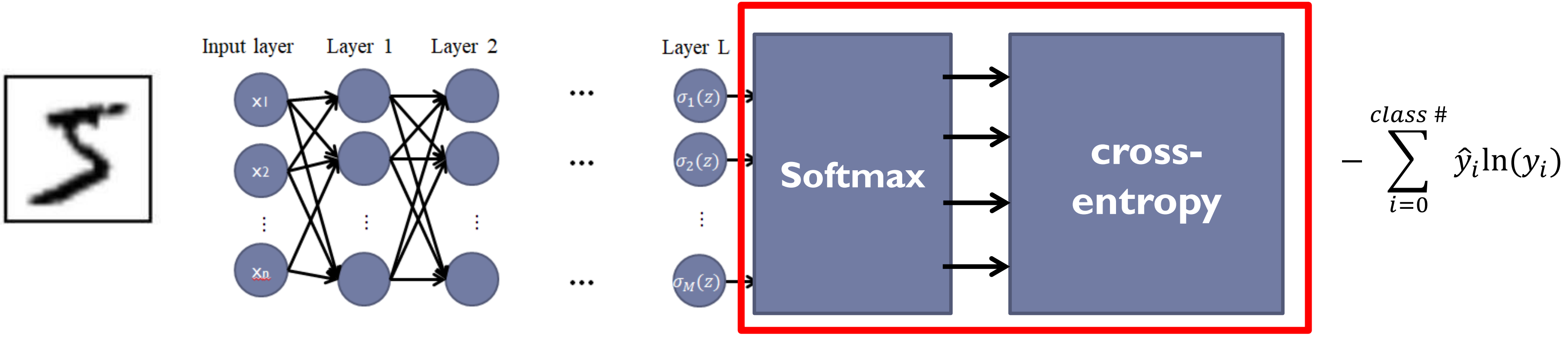
x:  **y: “2” (label)**

How to Feed data

How to feed data	Mean Square	Cross-entropy
All data at a time	$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$	$\frac{1}{N}\sum_{j=1}^{N}(-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i))$
One data at a time	$(y - \hat{y})$	$-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i)$
Batch of data a time (B < N)	$\frac{1}{B}\sum_{i=1}^{B}(y_i - \hat{y}_i)^2$	$\frac{1}{B}\sum_{j=1}^{B}(-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i))$

How to Feed data

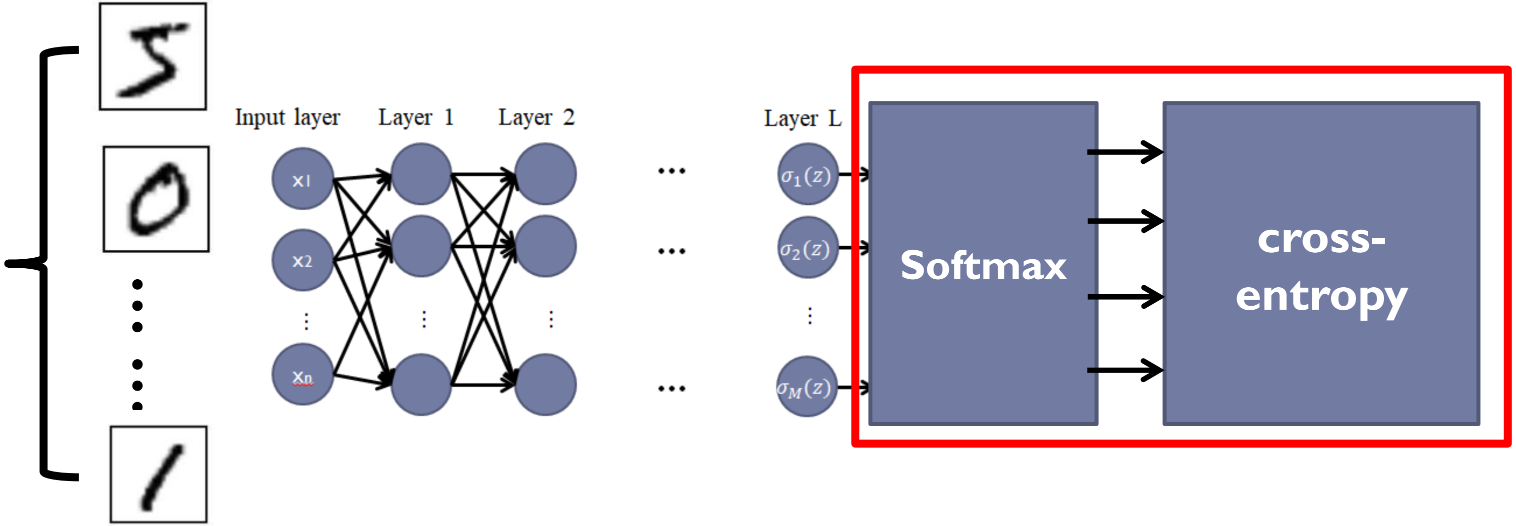
Pick one training data at a time



How to Feed data

Pick all training data at a time

All training data

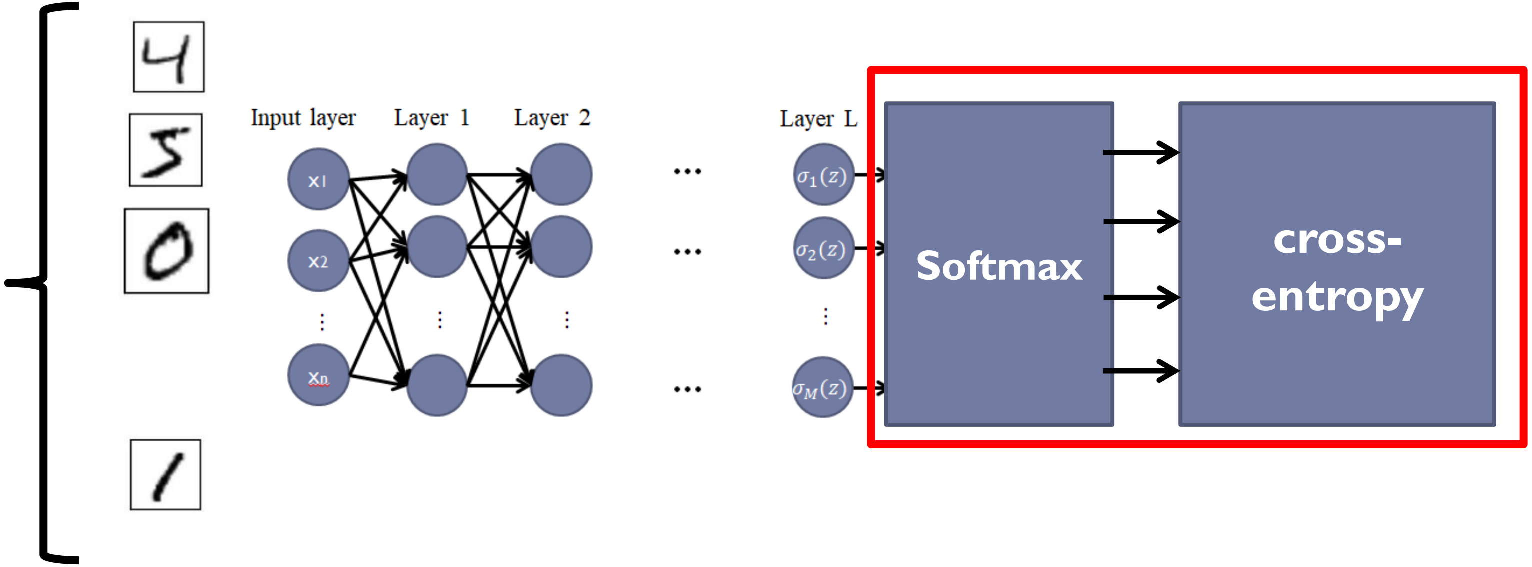


$$\frac{1}{N}\sum_{j=1}^{N}\left(-\sum_{i=0}^{class\ \#}\hat{y}_i \ln(y_i)\right)$$

How to Feed data

A batch of data
(you can define)

Pick a batch of data at a time



$$\frac{1}{B}\sum_{j=1}^{B}\left(-\sum_{i=0}^{class\,\#}\hat{y}_i \ln(y_i)\right)$$

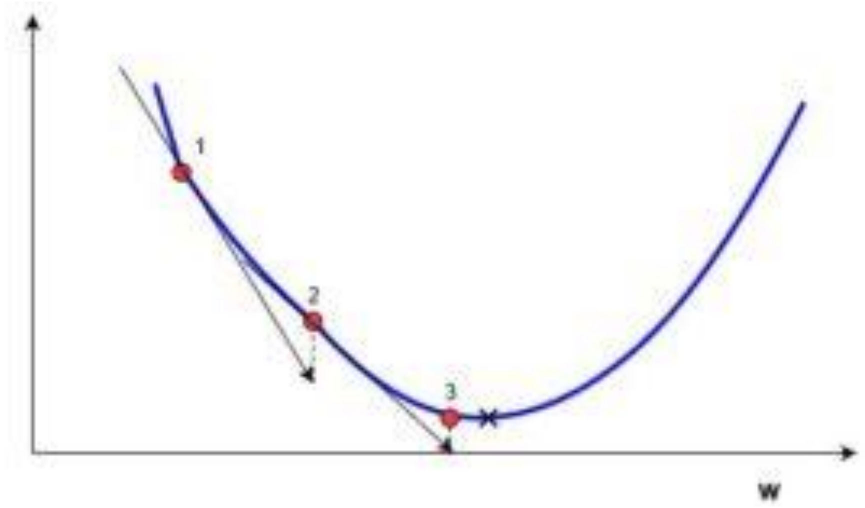
Optimization

Optimization

- We will introduce these Optimization methods
  - Gradient decent
  - Gradient decent with moment
  - Adagrad
  - RMSprop
  - Adam

Gradient Descent

- An algorithm that find the minimum of a function



$Randomly\ select\ \theta_1 as\ start\ point$

$$Compute\ \frac{dC(\theta_1)}{d\theta}$$

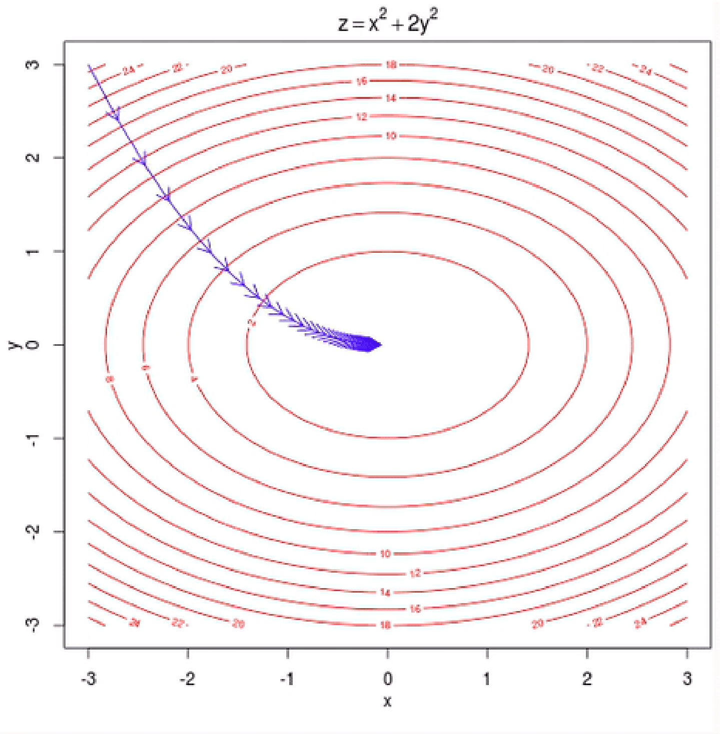
$$\theta_2 \leftarrow \theta_1 - \eta \frac{dC(\theta_1)}{d\theta}$$

$$Compute\ \frac{dC(\theta_2)}{d\theta}$$

$$\theta_3 \leftarrow \theta_2 - \boxed{\eta} \frac{dC(\theta_2)}{d\theta}$$

Learning rate

Gradient Descent



$$\theta = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \nabla C(\theta) = \begin{bmatrix} dz/dx \\ dz/dy \end{bmatrix}$$

$Randomly\ select\ \theta_1 as\ start\ point$

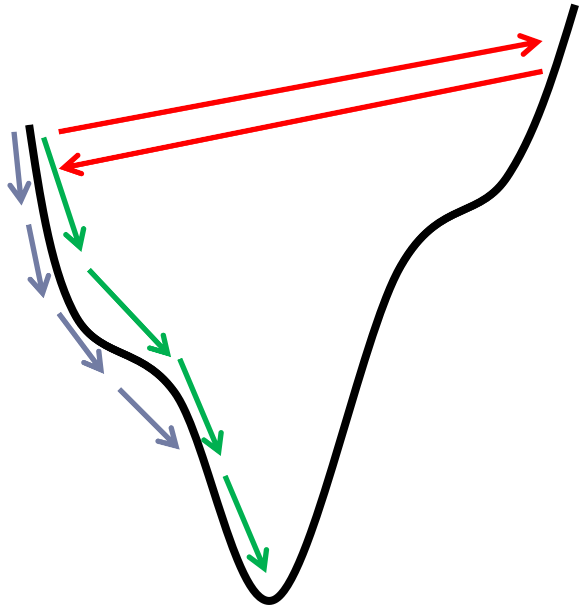
$Compute\ \nabla C(\theta_1)$

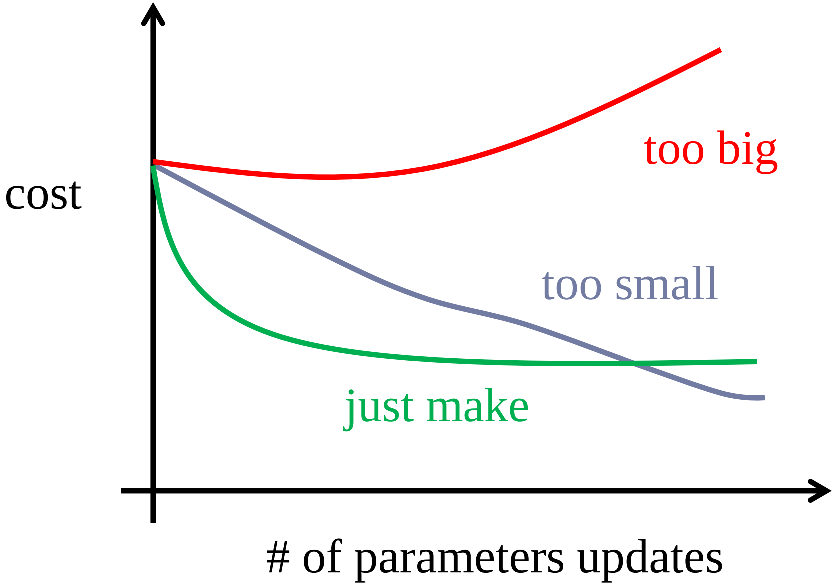
$$\theta_2 \leftarrow \theta_1 - \eta \nabla C(\theta_1)$$

$Compute \nabla C(\theta_2)$

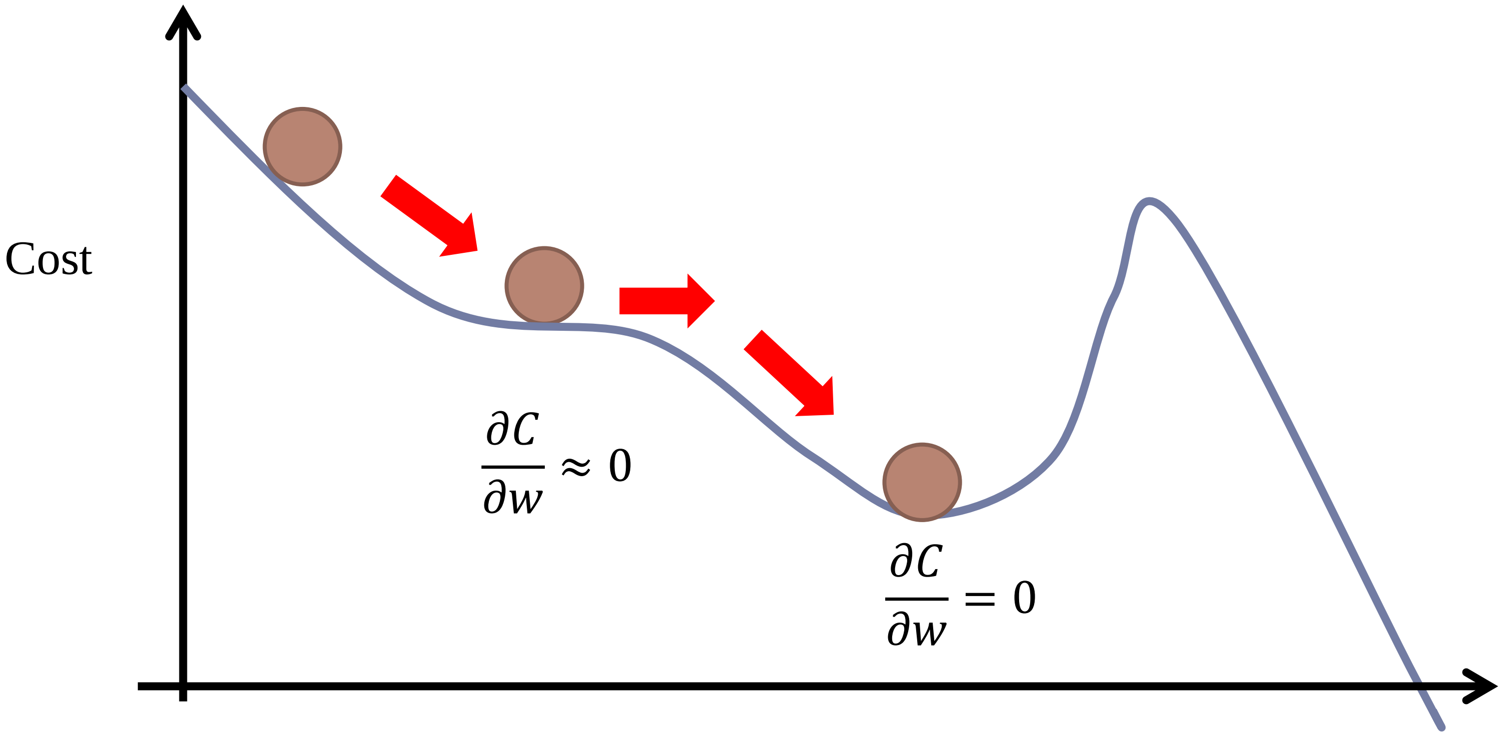
$$\theta_3 \leftarrow \theta_2 - \eta \nabla C(\theta_2)$$

Gradient Descent





Gradient Descent



Gradient Descent with Momentum

$Randomly\ select\ \theta_1 as\ start\ point$

$Compute\ \nabla\theta_1$

$\theta_2 \leftarrow \theta_1 - \eta\nabla\theta_1$

$Compute\ \nabla\theta_2$

$\theta_3 \leftarrow \theta_2 - \eta\nabla\theta_2$

$\vdots$

Without momentum

$Randomly\ select\ \theta_1 as\ start\ point\ , initialize\ v_1 = 0$

$Compute\ \nabla\theta_1, v_2 = \lambda v_1 - \eta\nabla\theta_1$

$\theta_2 \leftarrow \theta_1 + v_2$

$Compute\ \nabla\theta_2, v_3 = \lambda v_2 - \eta\nabla\theta_2$

$\theta_3 \leftarrow \theta_2 + v_3$

$\vdots$

With momentum

Gradient Descent

- Drawback of gradient descent
  - Learning rate would not decay over time
  - Direction of learning rate is fixed
- Adaptive learning rate is needed

Adagrad

$$w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0 \qquad \sigma^0 = \sqrt{(g^0)^2}$$

$$w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\frac{1}{2}[(g^0)^2 + (g^1)^2]}$$

$$\eta^t = \frac{\eta}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

$$w^3 \leftarrow w^2 - \frac{\eta^2}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\frac{1}{3}[(g^0)^2 + (g^1)^2 + (g^2)^2]}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$

RMSProp

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \qquad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2}$$

$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

Adam

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector) → for RMSprop
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

What is one epoch/step

- One epoch
  - One pass of all the training data
- One step
  - Pass a batch of training data (batch size is user defined)

Feed first batch to train → One step
Update parameters
Feed second batch to train
Update parameters
⋮
Feed final batch to train
Update parameters

→ One epoch

Backpropagation

- An efficient way to compute gradient given a set of variable

$$\frac{\partial C}{\partial w_{ij}^{l}} \quad ?$$

Backpropagation

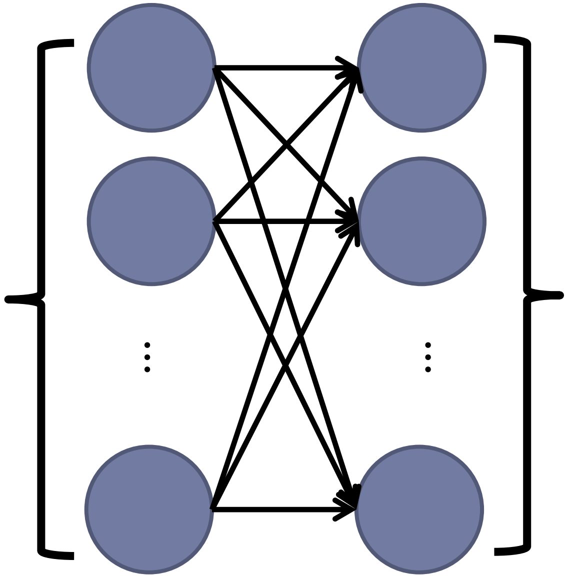
$$y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 X + b^1) + b^2) \ldots + b^L)$$

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} * \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$z^1 = W^1 X + b^1$$
$$a^1 = \sigma(z^1)$$
$$z^2 = W^2 a^1 + b^2$$
$$a^2 = \sigma(z^2)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$

1. Calculate $\frac{\partial z_i^l}{\partial w_{ij}^l}$
2. Calculate $\frac{\partial C}{\partial z_i^l}$ (**error signal** δ_i^l)

Backpropagation



$$\frac{\partial z_i^l}{\partial w_{ij}^l}$$

$if\ l = 1\ (input\ layer\ \rightarrow layer\ 1)$

$$z_i^1 = \sum_j w_{ij}^1 x_j + b_i^1 \qquad \frac{\partial z_i^1}{\partial w_{ij}^1} = x_j$$

$if\ l > 1 (layer\ l - 1\ \rightarrow layer\ 1)$

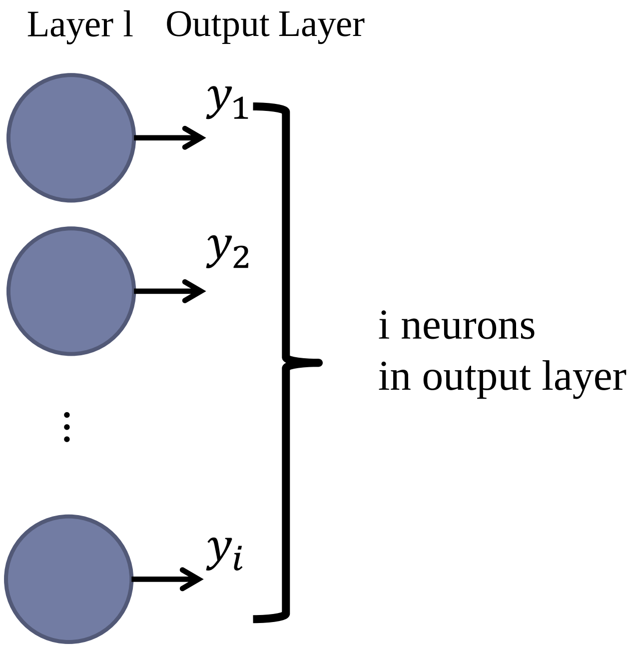
$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l \qquad \frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1}$$

Backpropagation

$$\frac{\partial C}{\partial z_i^l}$$ (**error signal** δ_i^l)

1. Compute δ^L (L layer error signal)
2. Compute relation between δ^l and δ^{l+1}

Backpropagation

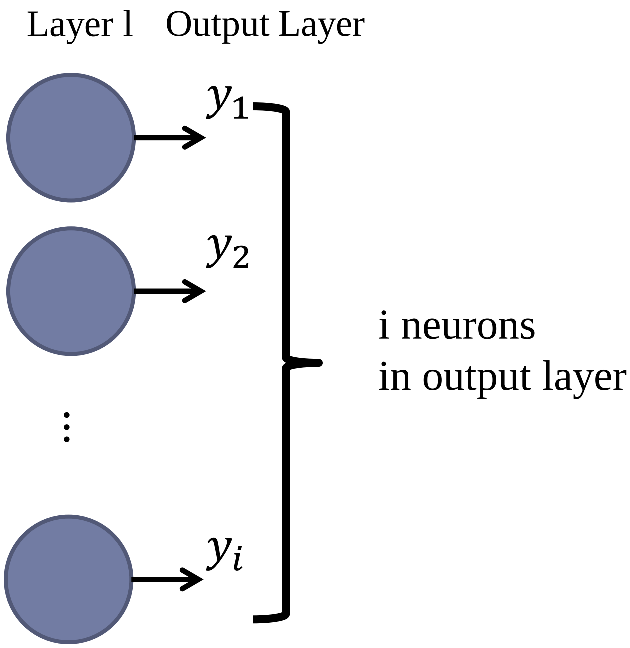


$$\frac{\partial C}{\partial z_i^l}$$ (**error signal** δ_i^l)

1. Compute δ^L (L layer error signal)

$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = \frac{\partial y_i}{\partial z_i^L} * \frac{\partial C}{\partial y_i} = \sigma'(z_i^L) * \frac{\partial C}{\partial y_i}$$

Backpropagation



$$\frac{\partial C}{\partial z_i^l} \quad (\textbf{error signal} \quad \delta_i^l \ )$$

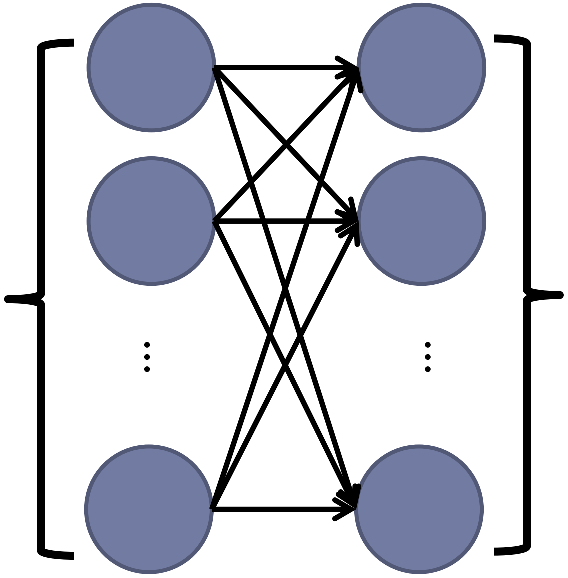
1. Compute δ^L (L layer error signal)

$$\delta^L = \sigma'(z^L) \bullet \nabla C(y)$$

$$\sigma'(z^L) = \begin{bmatrix} \sigma'(z_1^L) \\ \sigma'(z_2^L) \\ \vdots \\ \sigma'(z_n^L) \end{bmatrix} \qquad \nabla C'(y) = \begin{bmatrix} \frac{\partial C}{\partial y_1} \\ \frac{\partial C}{\partial y_2} \\ \vdots \\ \frac{\partial C}{\partial y_n} \end{bmatrix}$$

Backpropagation

Layer l Layer l+1



i neurons in l layer

k neurons in l+1 layer

$$\frac{\partial C}{\partial z_i^l}$$ (**error signal** δ_i^l)

2. Compute relation between δ^l and δ^{l+1}

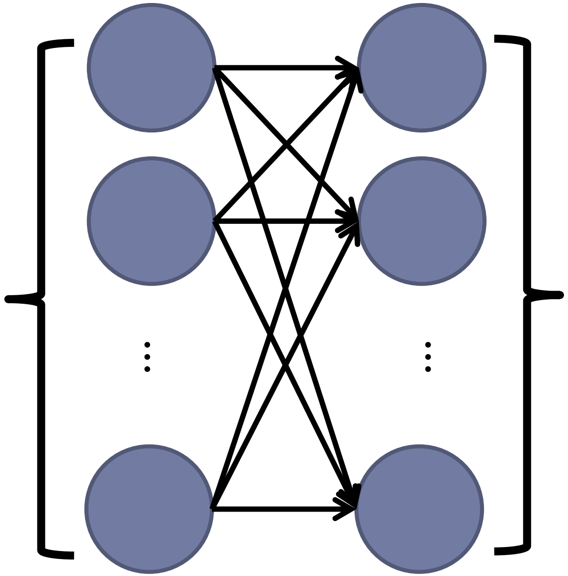
$$\delta_i^l = \frac{\partial C}{\partial z_i^l} = \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial C}{\partial z_k^{l+1}}$$

recall

$$\boxed{\frac{\partial a_i^l}{\partial z_i^l} = \sigma'(z_i^l), \quad \frac{\partial z_k^{l+1}}{\partial a_i^l} = w_{ki}^{l+1}, \quad \frac{\partial C}{\partial z_k^{l+1}} = \delta_k^{l+1}}$$

$$\delta_i^l = \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

Backpropagation



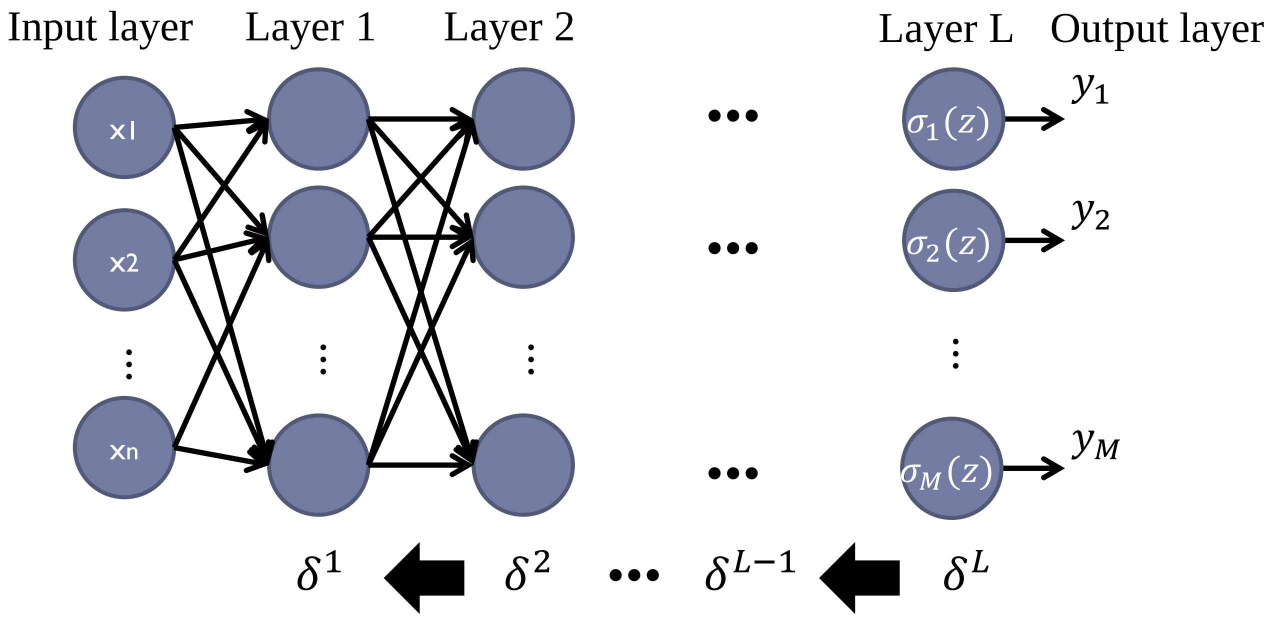
$\frac{\partial C}{\partial z_i^l}$ (**error signal** δ_i^l)

2. Compute relation between δ^l and δ^{l+1}

$$\delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1}$$

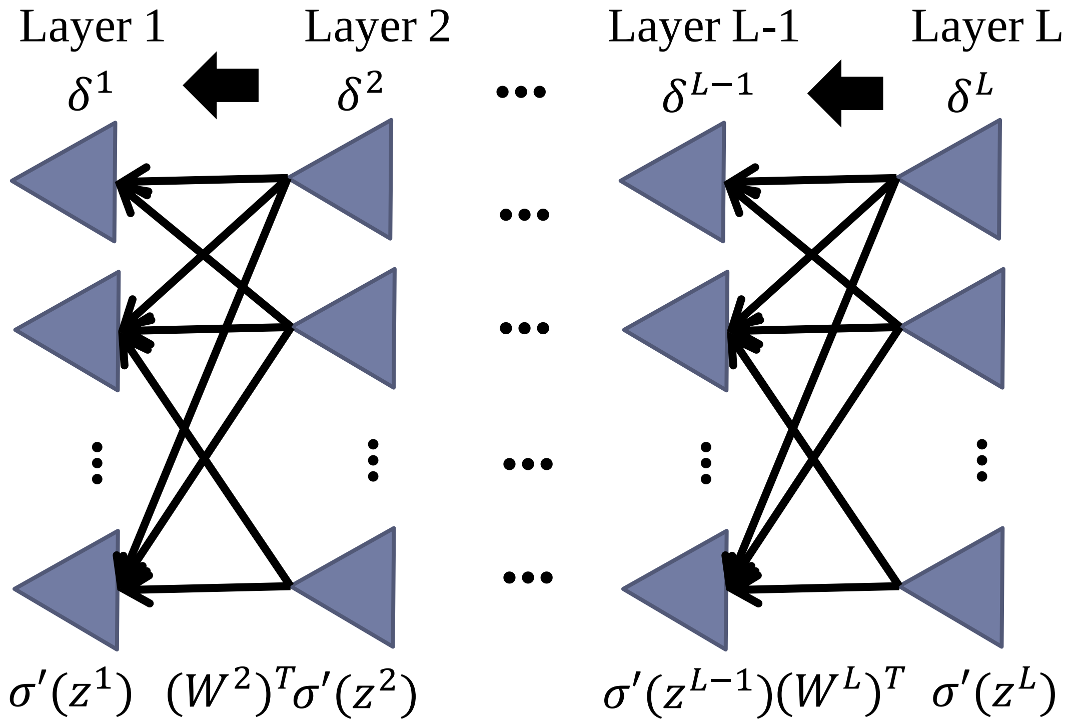
Backpropagation

$$\begin{cases} \delta^L = \sigma'(z^L) \bullet \nabla C(y) \\ \delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1} \end{cases}$$



Backpropagation

$$\begin{cases} \delta^L = \sigma'(z^L) \bullet \nabla C(y) \\ \delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1} \end{cases}$$



Backpropagation

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} * \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\delta^L = \sigma'(z^L) \bullet \nabla C(y)$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^L)^T \delta^L$$

$$\vdots$$

$$\delta^{l-1} = \sigma'(z^{l-1}) \bullet (W^l)^T \delta^l$$

$$\vdots$$

$if\ l = 1\ (input\ layer\ \rightarrow layer\ 1)$

$$z_i^1 = \sum_j w_{ij}^1 x_j^r + b_i^1 \qquad \frac{\partial z_i^1}{\partial w_{ij}^1} = x_j$$

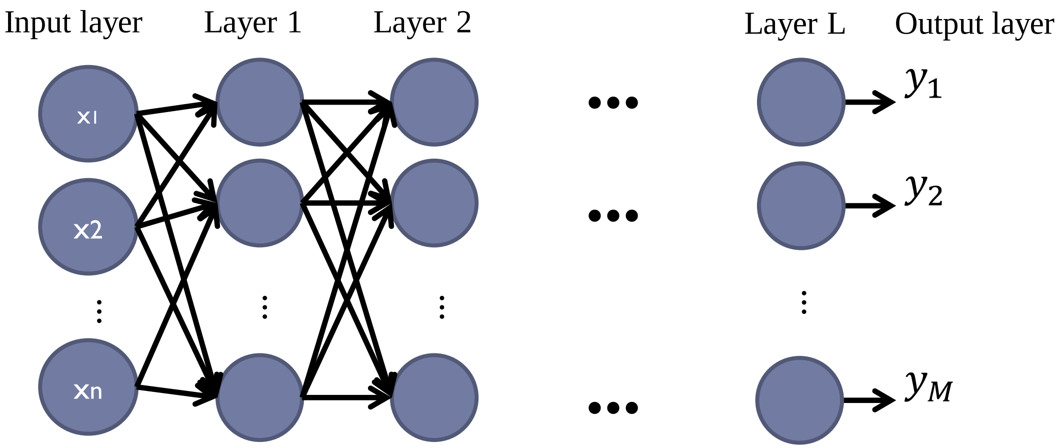
$if\ l > 1 (layer\ l-1\ \rightarrow layer\ 1)$

$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l \qquad \frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1}$$

Predict/Validate Result

Predict/Validate Result

- “*” mean a better set of parameters we have found
- Use test data to find model accuracy (supervised learning)
  - # correct predict / # total data



$$\theta = \{W^{1*}, b^{1*}, W^{2*}, b^{2*}, \dots, W^{L*}, b^{L*}\}$$

Overfitting

Training error is going lower but testing error is not

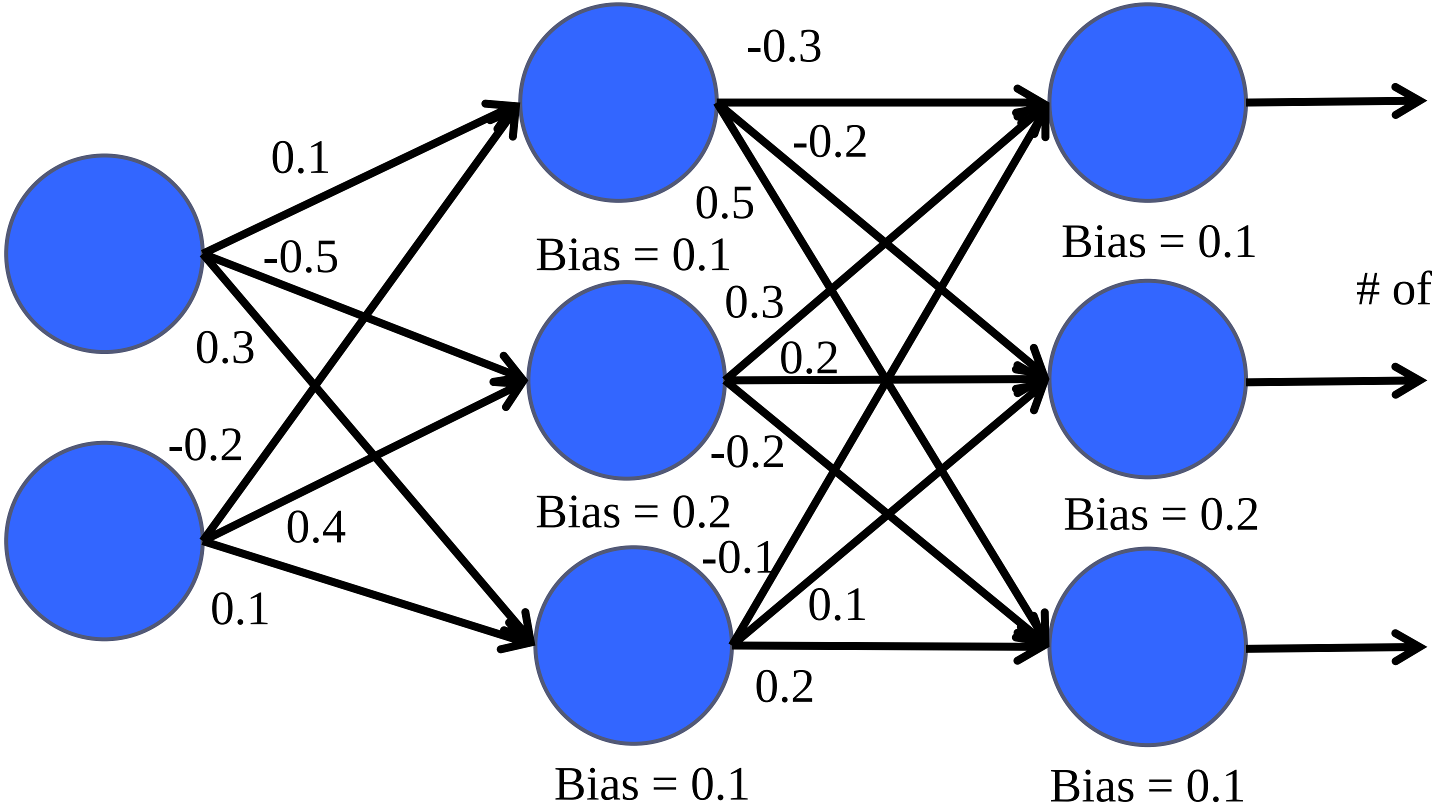
Overfitting

- Prevent overfitting
  - Use more data
  - Stronger regularization
  - Data augmentation
  - Reduce complexity of model

Quick Recap in this Lecture

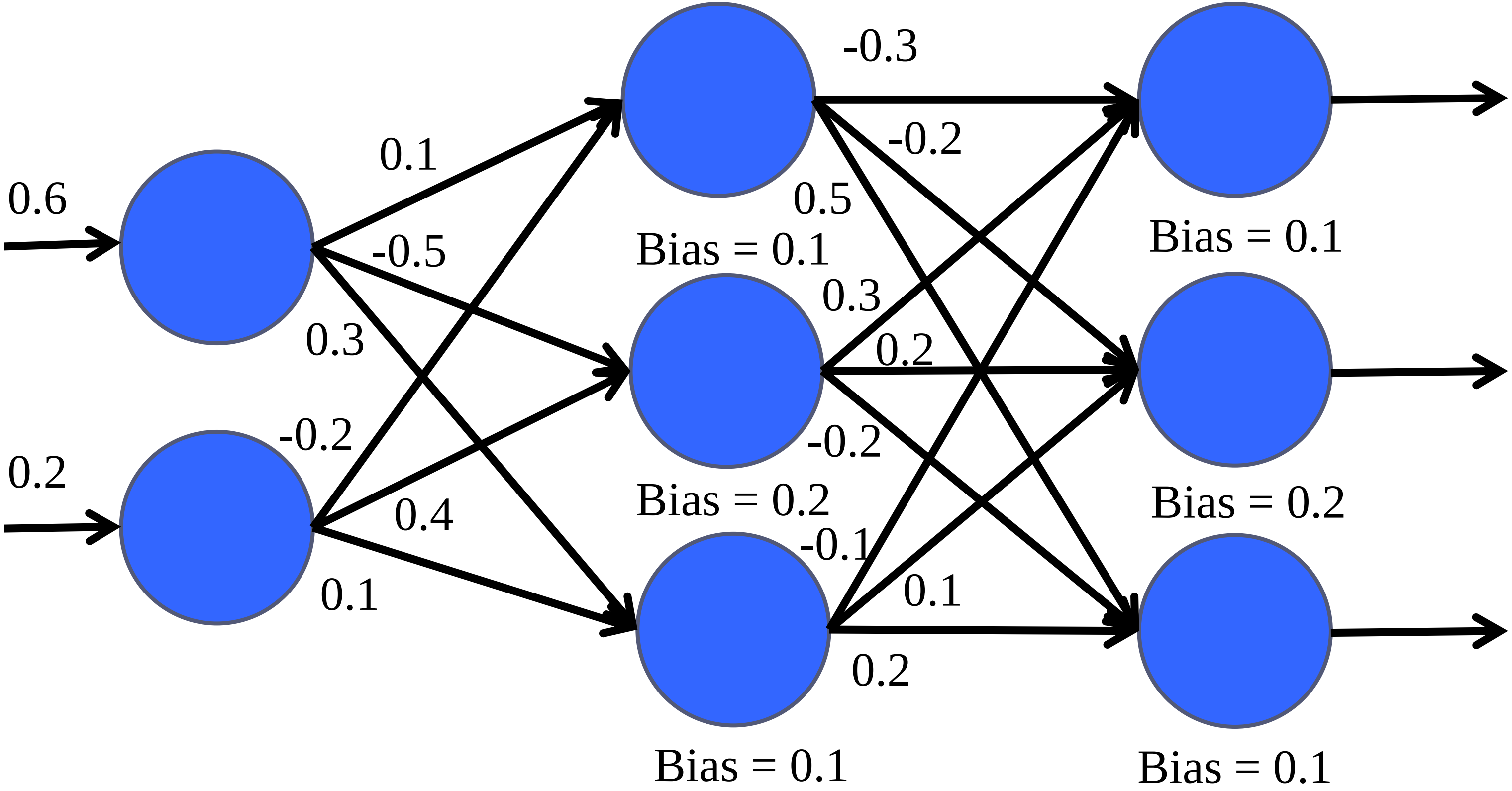
1. Build DNN model
2. Define loss (mean square/cross entropy with softmax)
3. Optimization (backpropagation)
4. Validate result

Example-initialization

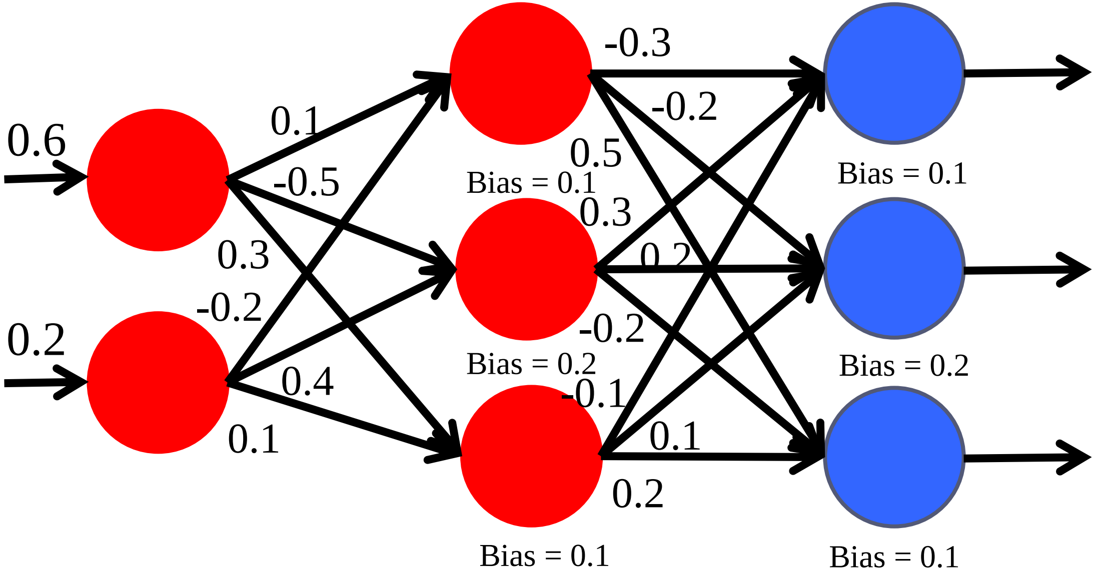


\# of parameters = 21

Example-feed data



Example-forward pass

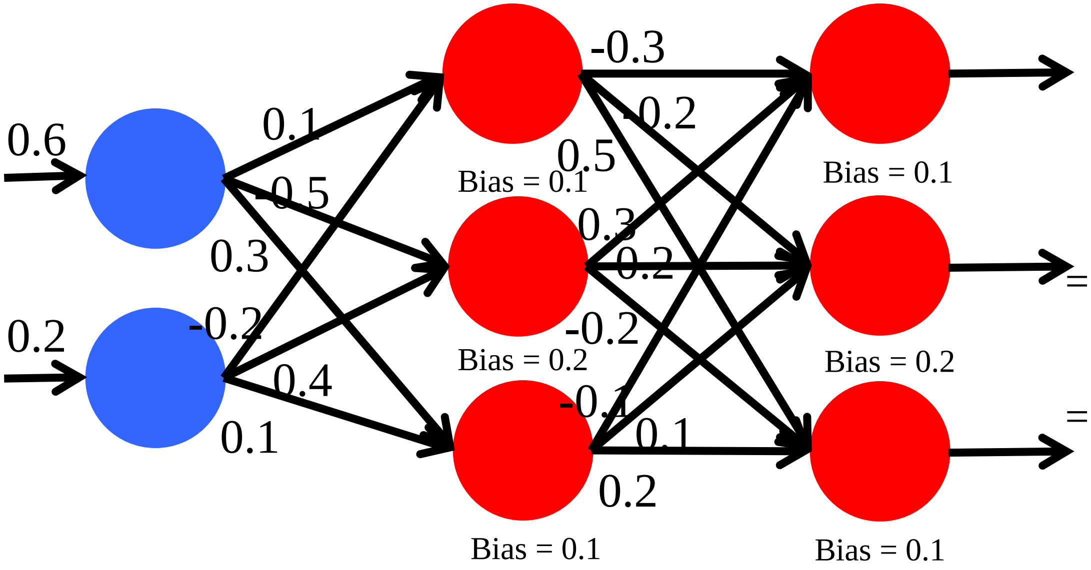


$$relu\left(\begin{bmatrix} 0.1 & -0.2 \\ -0.5 & 0.4 \\ 0.3 & 0.1 \end{bmatrix}\begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix} 0.12 \\ -0.02 \\ 0.3 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.12 \\ 0 \\ 0.3 \end{bmatrix}$$

Example-forward pass

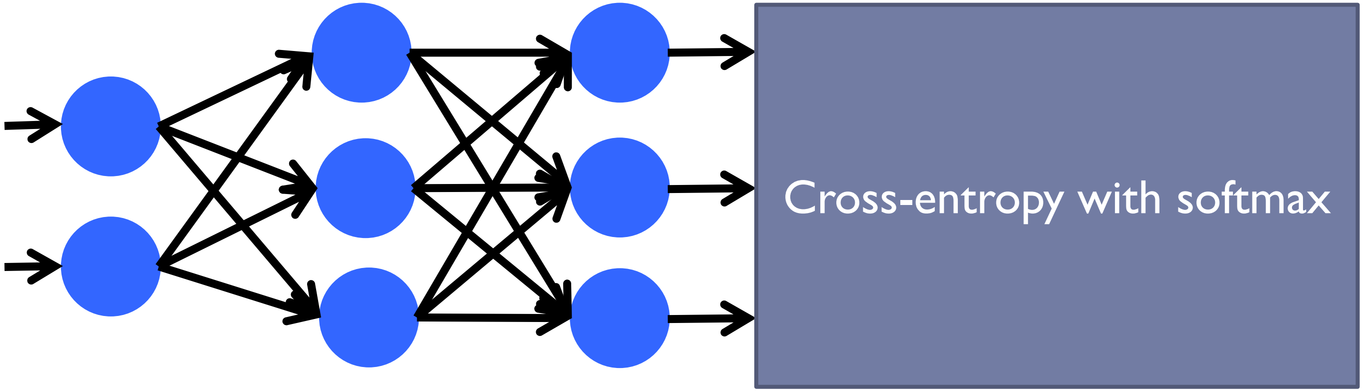


$$relu\left(\begin{bmatrix} -0.3 & 0.3 & -0.1 \\ -0.2 & 0.2 & 0.1 \\ 0.5 & -0.2 & 0.2 \end{bmatrix}\begin{bmatrix} 0.12 \\ 0 \\ 0.3 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}\right)$$

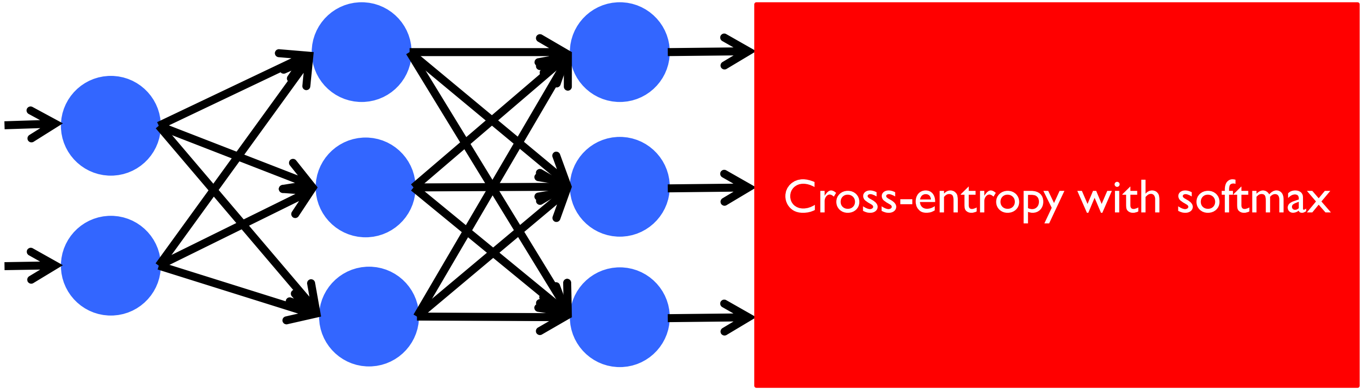
$$= relu\left(\begin{bmatrix} -0.066 \\ 0.006 \\ 0.12 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0 \\ 0.006 \\ 0.12 \end{bmatrix}$$

Example-forward pass



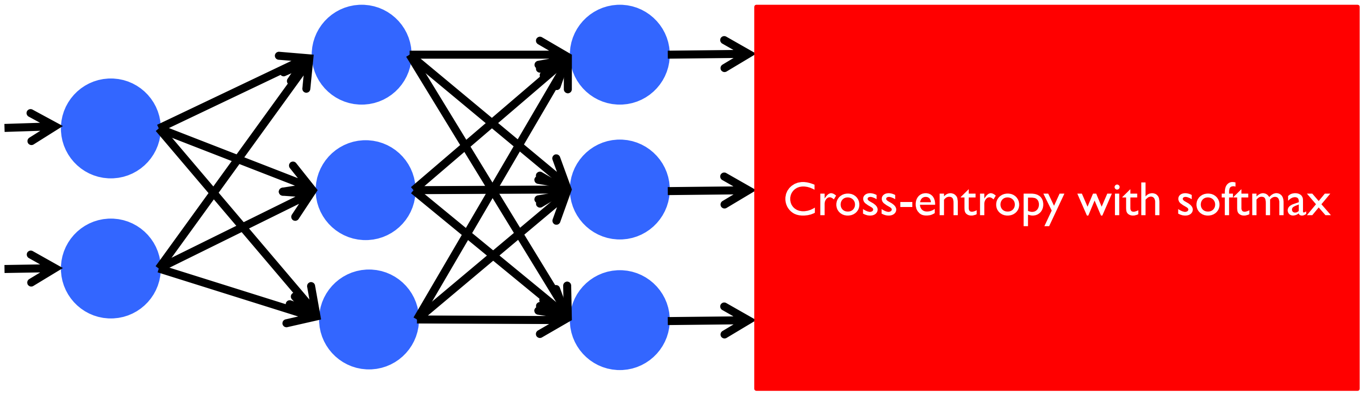
Example-forward pass



$$\text{Softmax}\left(\begin{bmatrix} 0 \\ 0.006 \\ 0.12 \end{bmatrix}\right) = \begin{bmatrix} 0.319 \\ 0.321 \\ 0.36 \end{bmatrix}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Example-forward pass



What we expect (label)

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

- 0 * ln(0.319) – 1 * ln(0.321) – 0 * ln(0.36)
= 1.1363

$$-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i)$$

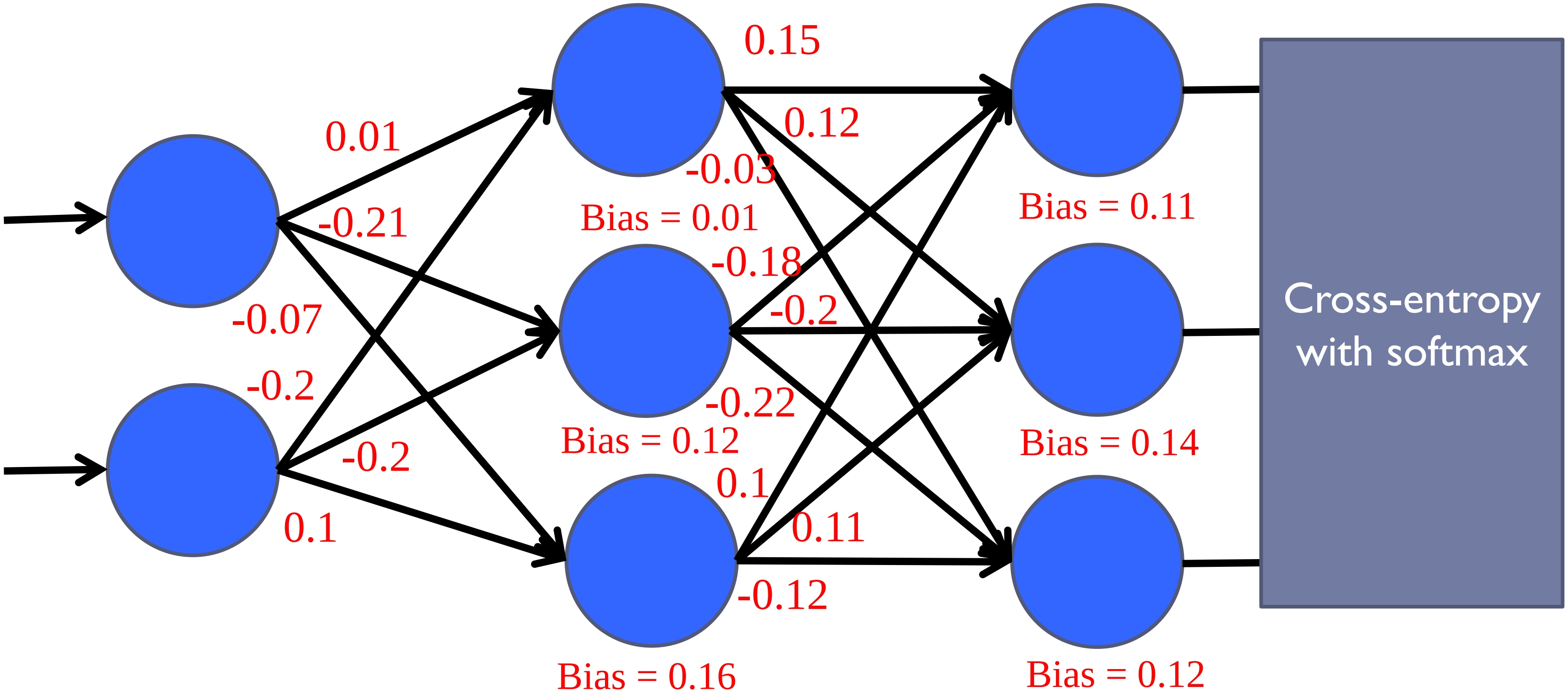
This is large at the beginning. During training, this should decrease.

Example-optimization

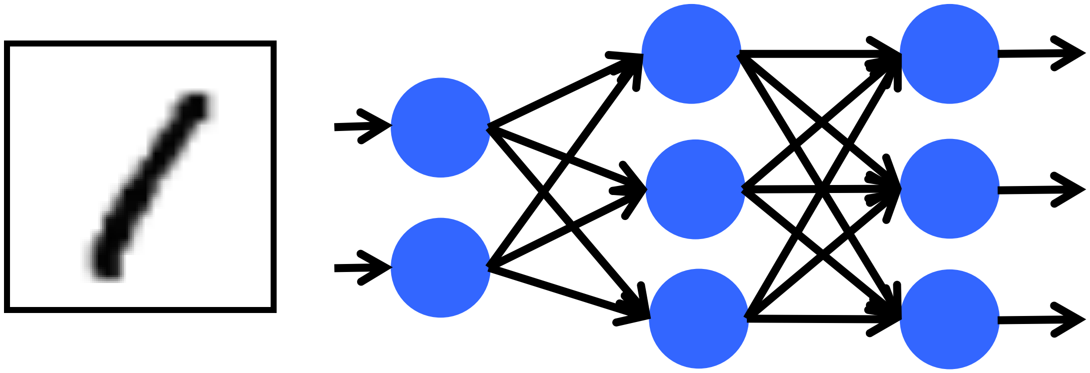
Use optimizer to optimize

and wait……

Example-after optimization



Example-predict



Feed you test data
You may get something like this

$$\begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix}$$

Model predict this is class #2
(0.5 is greater than others)