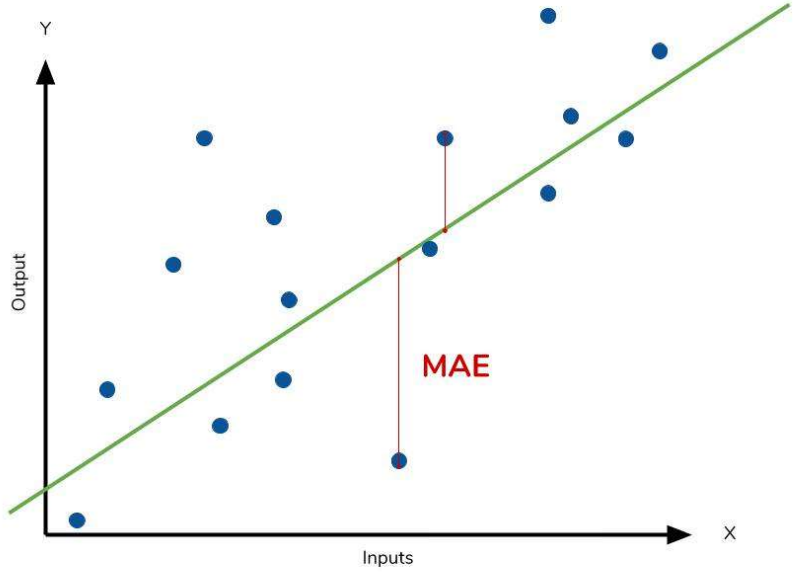
Assessing the performance of the Model

$$MAE(Mean\ Absolute\ Error) = \sum_{i}^{m} \frac{|y_i - \hat{y}_i|}{m}$$

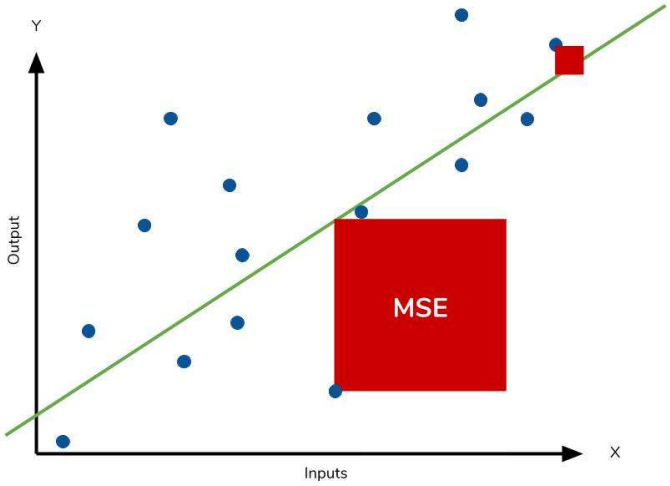


- m : data points
- Y : vector of observed values
- $\hat{Y}$: vector of predicted values

- 절대값을 취하기 때문에 직관적
- MSE보다 특이치에 robust
- Regression 평가에 사용

Assessing the performance of the Model

$$Mean\ Square\ Error = \frac{1}{m}\sum_{i=1}^{n}(\hat{Y} - Y)^2$$

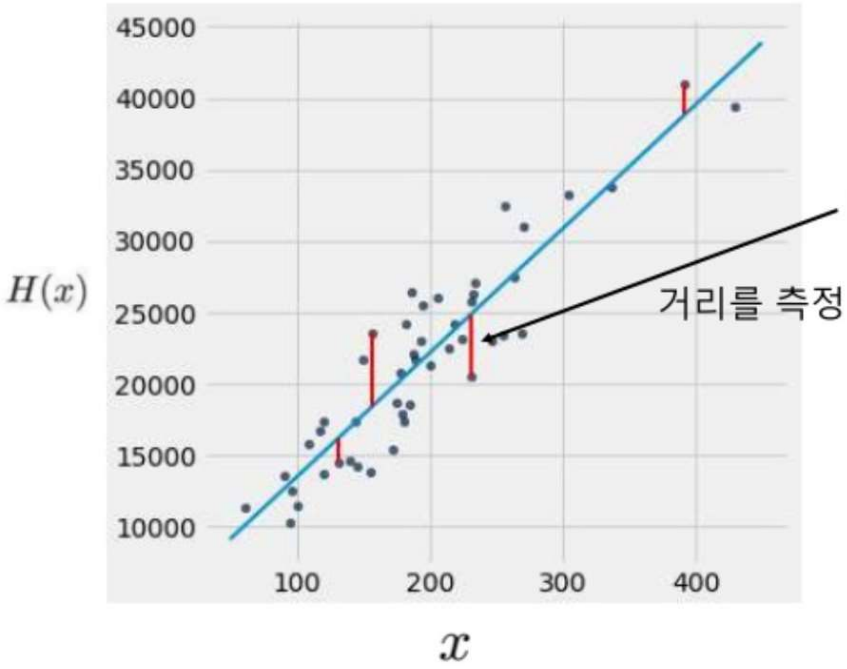


- m : data points
- Y : vector of observed values
- $\hat{Y}$: vector of predicted values

- 모델의 예측값과 실제 값 차이 이의 면적의 합
- 특이치에 민감
- Training에 사용

How to minimize the cost?

- $MSE = \frac{1}{m}\sum_{i=1}^{n}(\hat{Y} - Y)^2$



$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

(m = data size, y = 실제값)

cost 함수

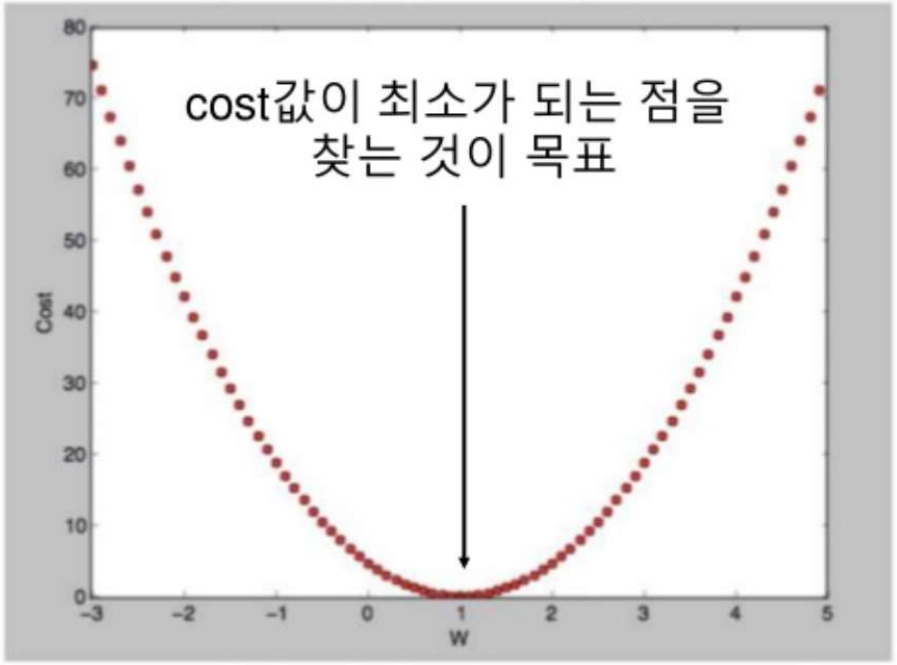
↓

W , b 의 함수

↓

Cost값을 작게 가지는 W , b 를 학습
= linear regression 의 학습

Cost function(or loss function)



(e.g. MSE)

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

무작위로 $H(x)$을 그어서
$cost(W)$ 가 최소가 되는 점을 찾는다?

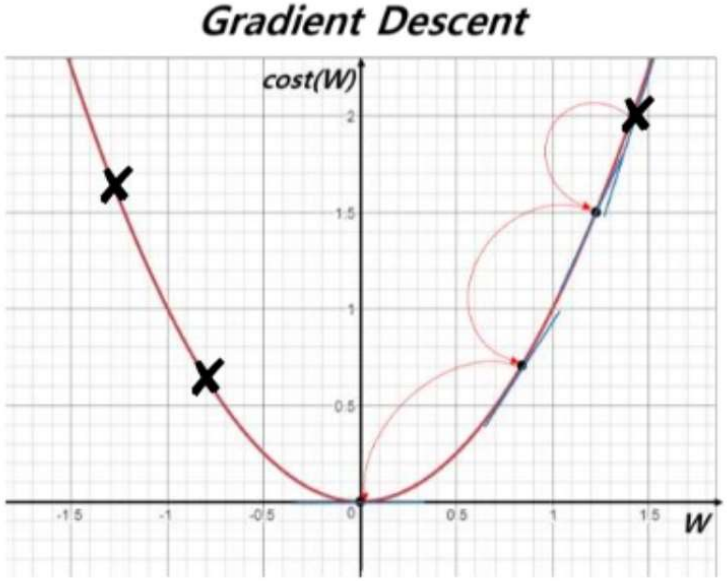
↓

$cost(W)$가 최소가 되는 점을
기계적으로 찾아내야 함

↓ optimization

Gradient descent algorithm

Gradient descent algorithm



✘ = 시작점

learning rate(step size)
= 수렴속도 조절

$$W := W - \eta \frac{1}{m} \sum_{i=1}^{m} \left(W \cdot x^{(i)} - y^{(i)} \right) \cdot x^{(i)}$$

$\frac{\partial cos\ (W)}{\partial W}$: 가파른 정도(slope)와 방향

*참고: $\frac{\partial (x^n)}{\partial x} = nx^{n-1}$

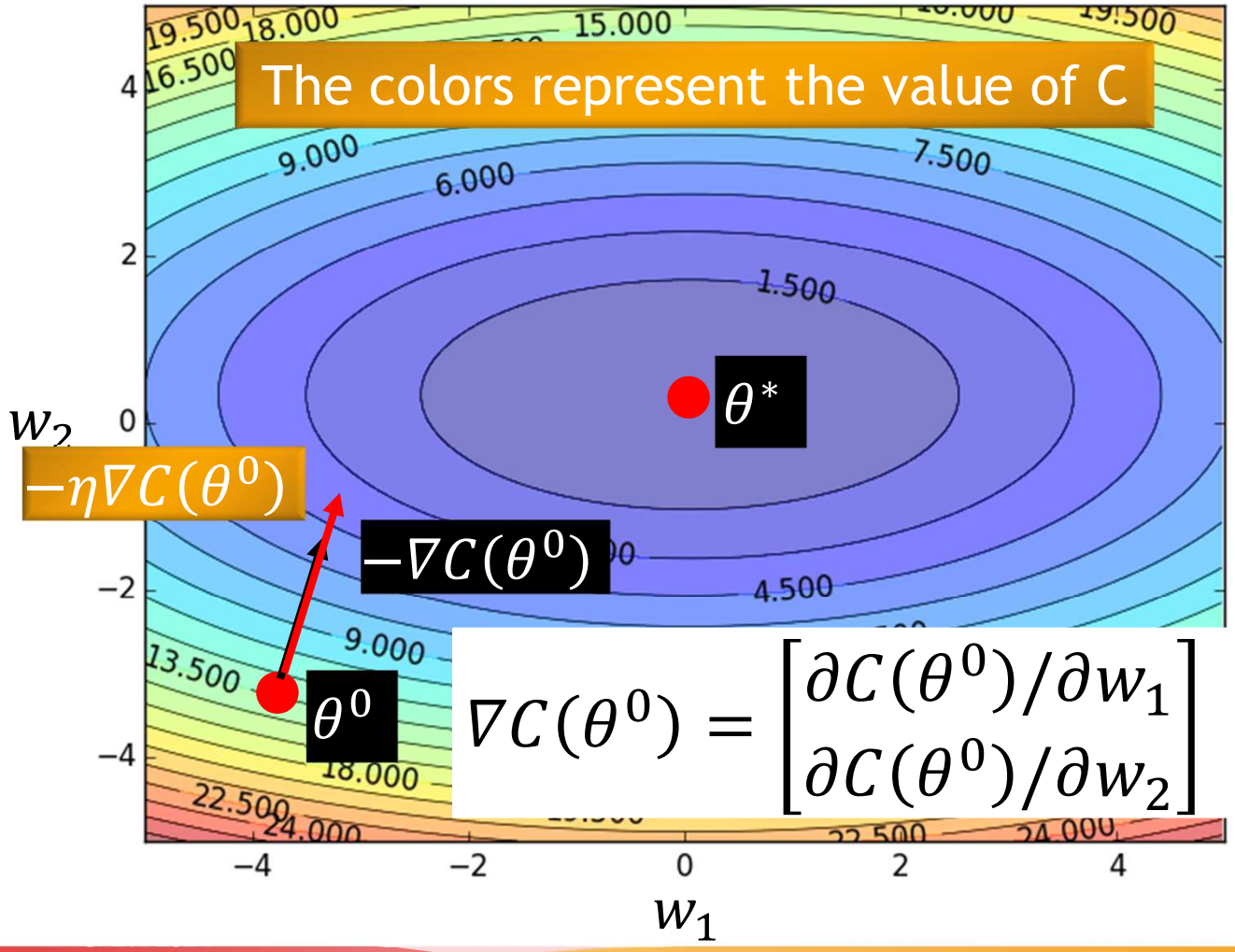
1. 시작점의 경사도에 따라 이동
2. 이동된 위치의 경사도를 따라 다시 이동
3. Cost(W)가 최소 즉 경사도 0인 지점까지 반복

Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network.

$$\theta = \{w_1, w_2\}$$

Error surface



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

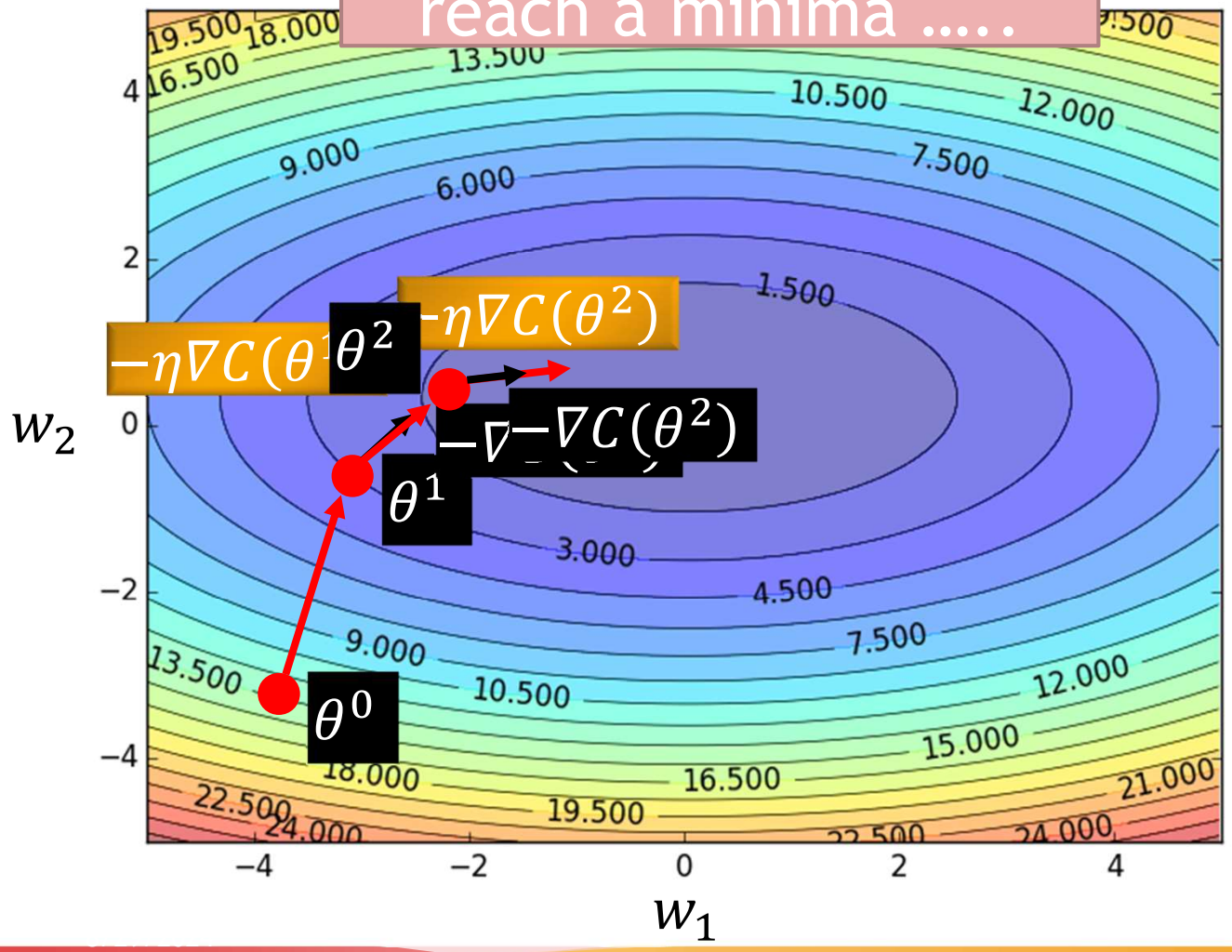
$-\nabla C(\theta^0)$

Times the learning rate η

$-\eta\nabla C(\theta^0)$

Gradient Descent

Eventually, we would reach a minima .....



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

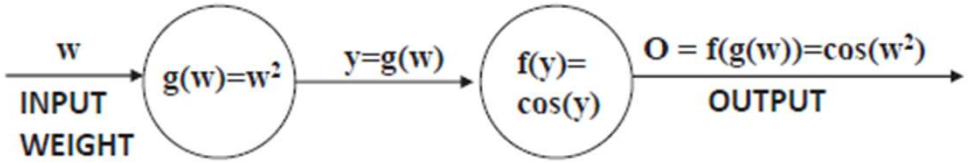
$-\nabla C(\theta^0)$

Times the learning rate η

$-\eta\nabla C(\theta^0)$

Backpropagation

- To compute the partial derivative of the loss function wrt. each intermediate weight
  - Not a simple matter with multi-layer architectures



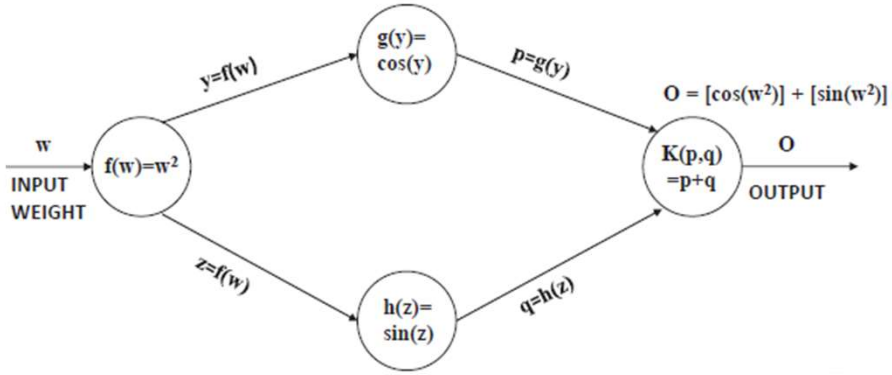
- In the univariate chain rule, we compute product of local derivatives

$$\frac{\partial f(g(w))}{\partial w} = \underbrace{\frac{\partial f(y)}{\partial y}}_{-sin(y)} \cdot \underbrace{\frac{\partial g(w)}{\partial w}}_{2w} = -2w \cdot \sin(y) = -2w \cdot \sin(w^2)$$

- Local derivatives are easy to compute because they care about their own inputs and outputs

Backpropagation

- Multivariate chain rule



$$\frac{\partial o}{\partial w} = \underbrace{\frac{\partial K(p,q)}{\partial p}}_{1} \cdot \underbrace{g'(y)}_{-\sin(y)} \cdot \underbrace{f'(w)}_{2w} + \underbrace{\frac{\partial K(p,q)}{\partial q}}_{1} \cdot \underbrace{h'(z)}_{\cos(z)} \cdot \underbrace{f'(w)}_{2w}$$

$$= -2w \cdot \sin(y) + 2w \cdot \cos(z)$$

$$= -2w \cdot \sin(w^2) + 2w \cdot \cos(w^2)$$

- Product of local derivatives along all paths from ***w*** to ***o***

Backpropagation

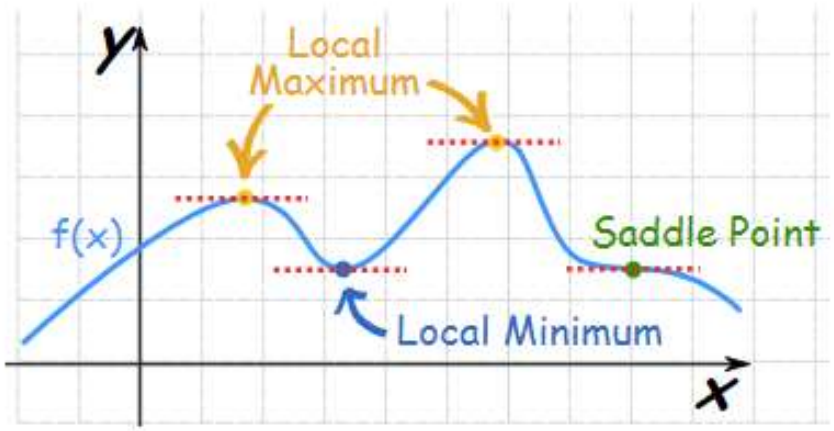
- A neural network can have millions of parameters
  - The size of today's NNs tends to increase
    - E.g., GPT-3 consists of 175B parameters
- Many toolkits can compute the gradients automatically

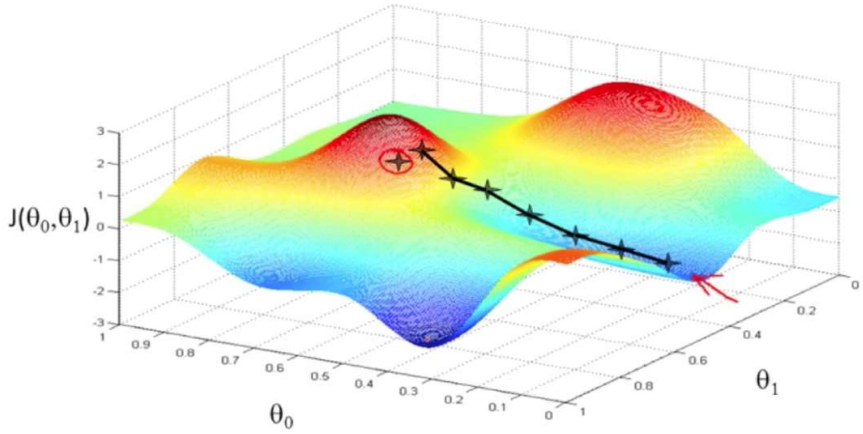
TensorFlow PyTorch Keras theano

- Pytorch: 연구자들이 선호, 많은 연구자 커뮤니티 사이트, HuggingFace에 등록된 모델의 92%가 Pytorch(`23.3)
- TensorFlow : 산업계에서 주로 활용, Model Serving, Debugging 유리

Local Minima

- Gradient descent never guarantee global minima

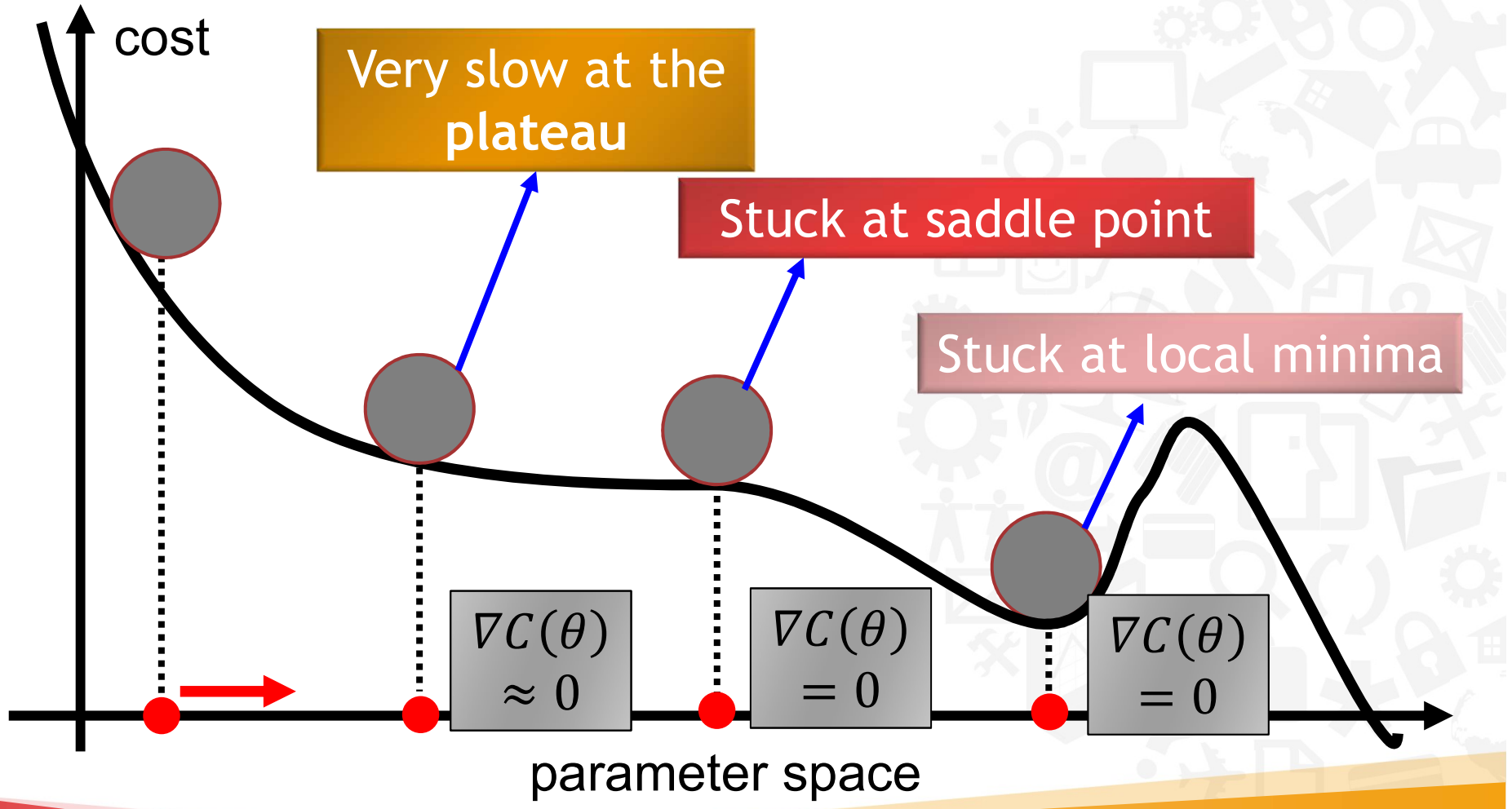




Different initial point θ^0

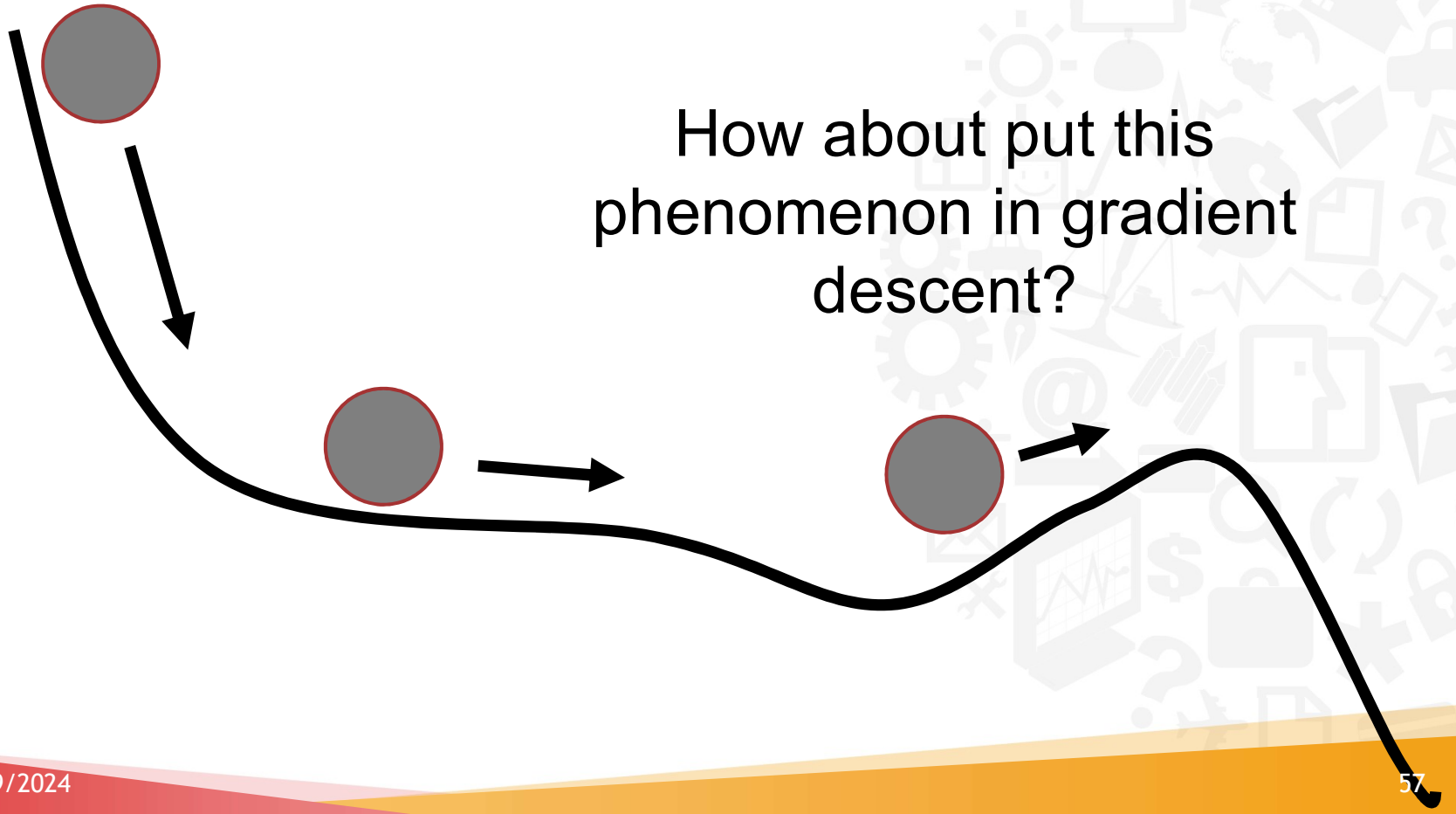
Reach different minima, so different results

Besides local minima .....



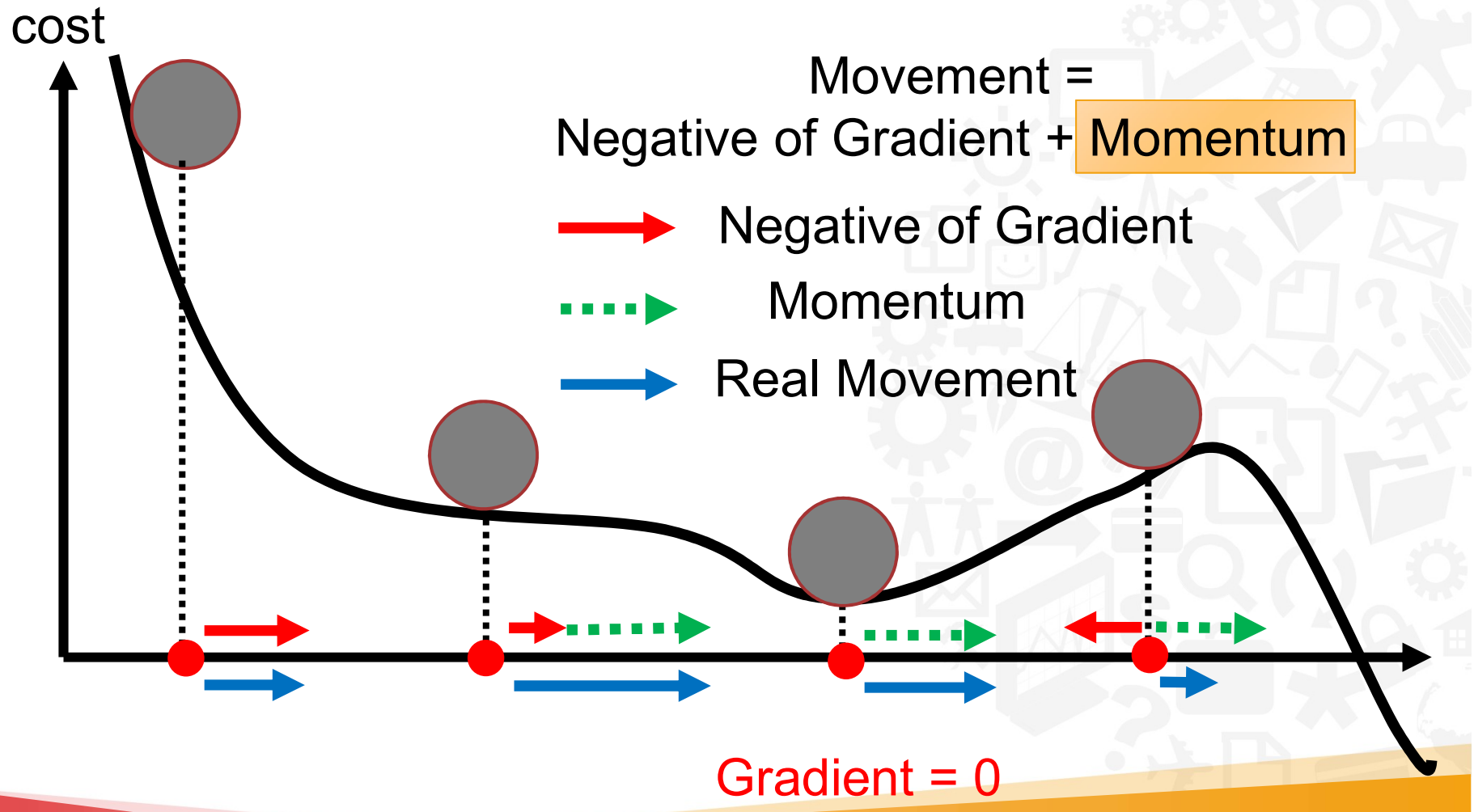
In physical world ……

- Momentum

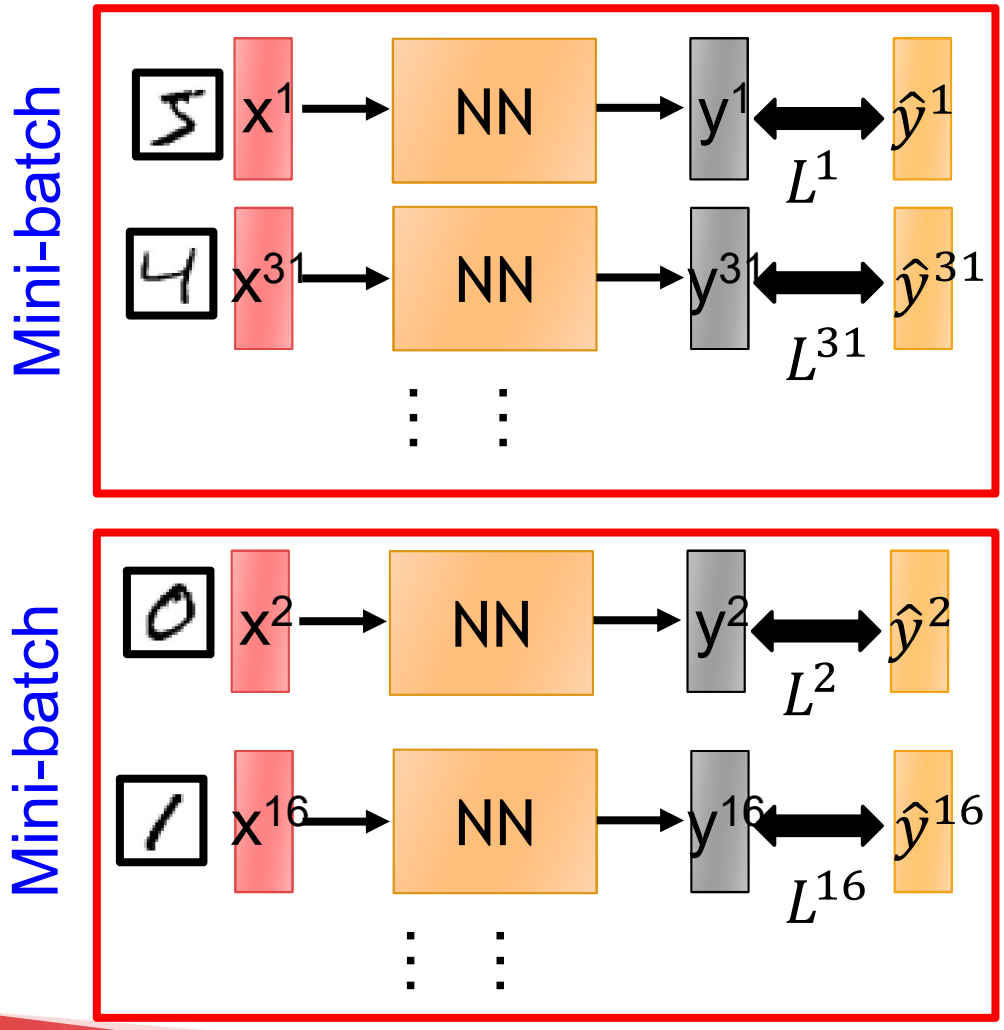


Momentum

Still not guarantee reaching global minima, but give some hope ……



Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch

$$C = L^1 + L^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

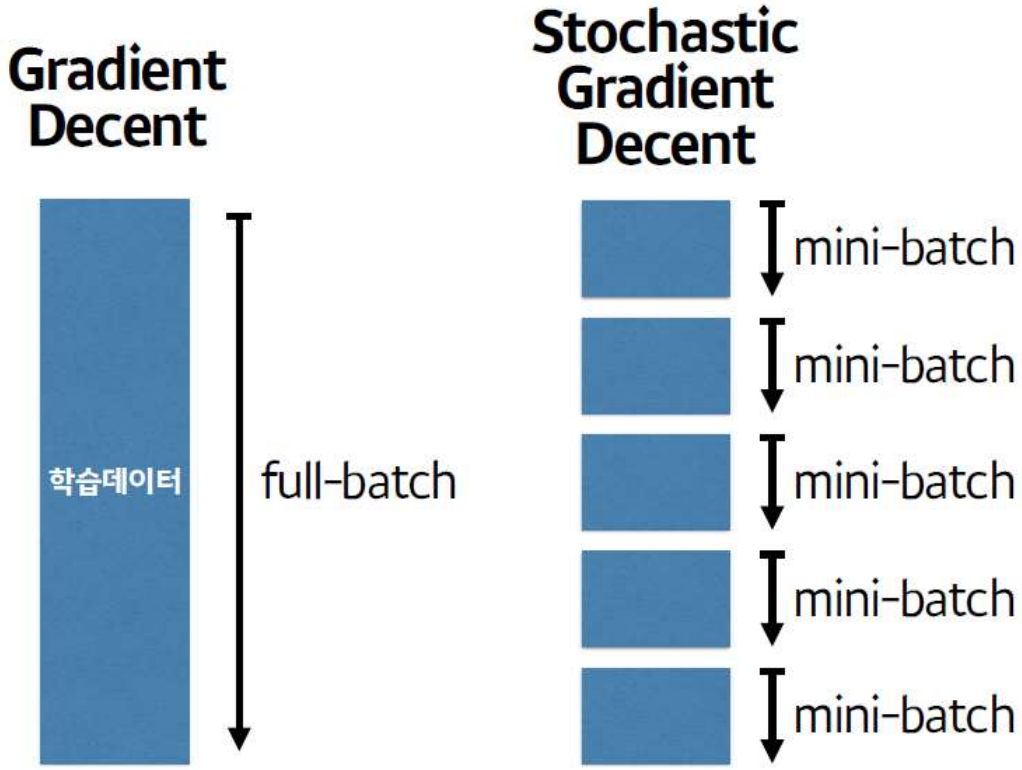
$$C = L^2 + L^{16} + \cdots$$

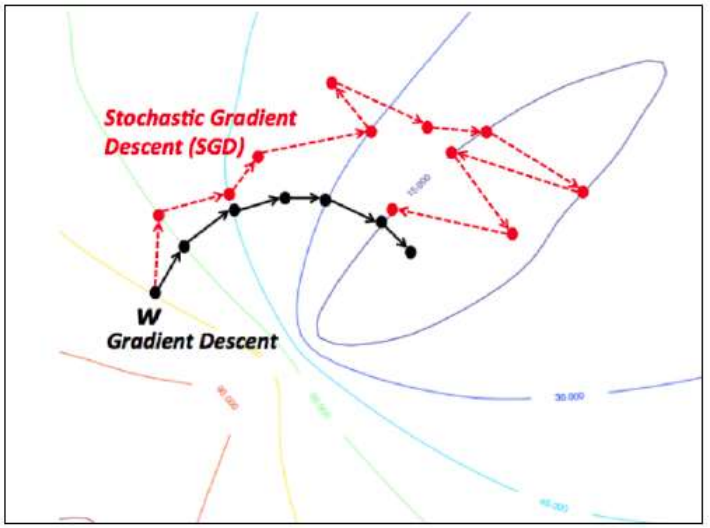
$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

⋮

C is different each time when we update parameters!

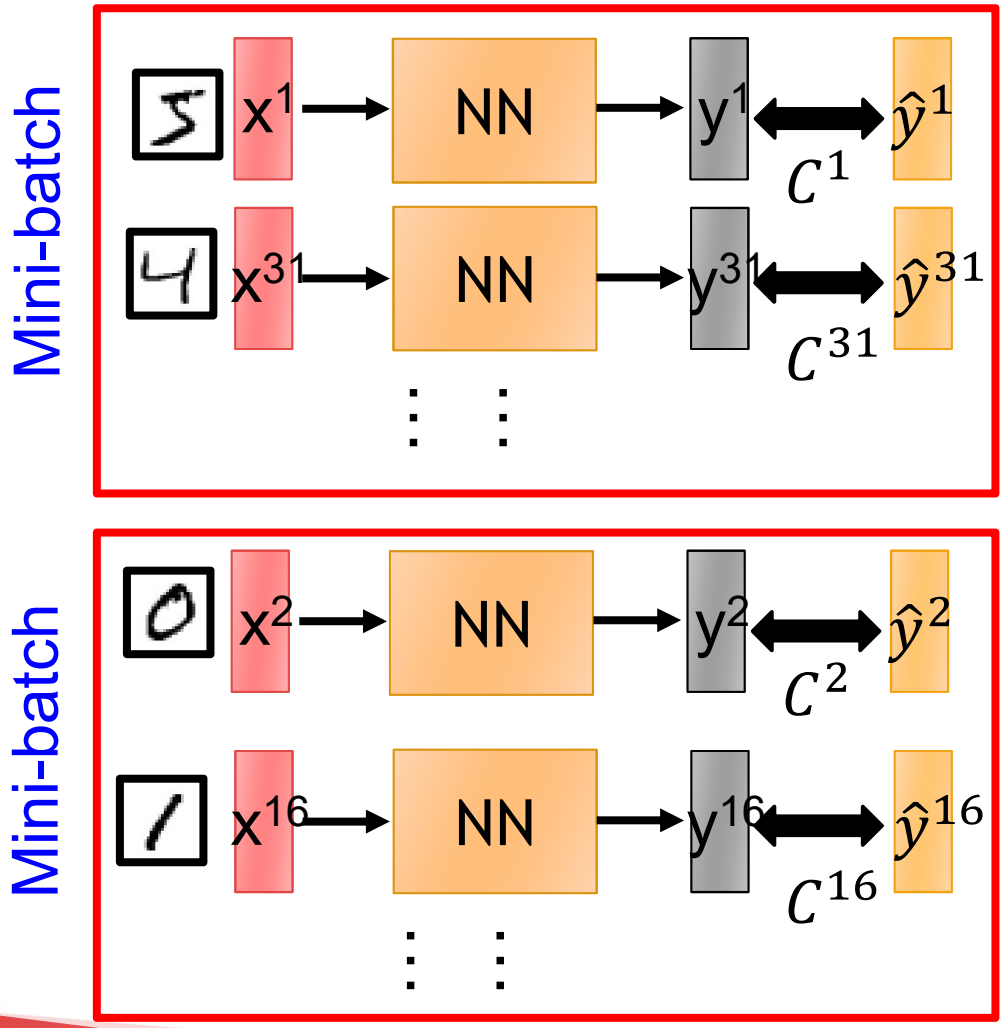
Mini-batch


Gradient Decent
학습데이터
full-batch
Stochastic Gradient Decent
mini-batch
mini-batch
mini-batch
mini-batch
mini-batch


Stochastic Gradient Descent (SGD)
W
Gradient Descent

Mini-batch

Faster **Better!**



- Randomly initialize θ^0
- Pick the 1st batch

  $C = C^1 + C^{31} + \cdots$

  $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
- Pick the 2nd batch

  $C = C^2 + C^{16} + \cdots$

  $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

  .
- Until all mini-batches have been picked

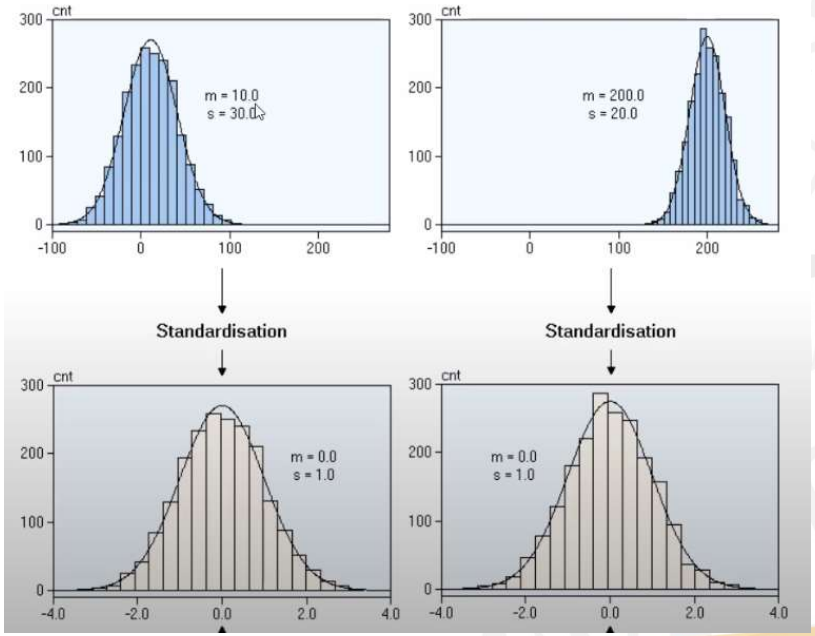
one epoch

Repeat the above process

Standardization

- 각 관찰값이 평균을 기준으로 어느정도 떨어져 있는지를 나타냄
  - 값의 scale이 다른 두 변수가 있을 때 scale 차이를 제거해주는 효과

$$x_{new} = \frac{x - \mu}{\sigma}$$

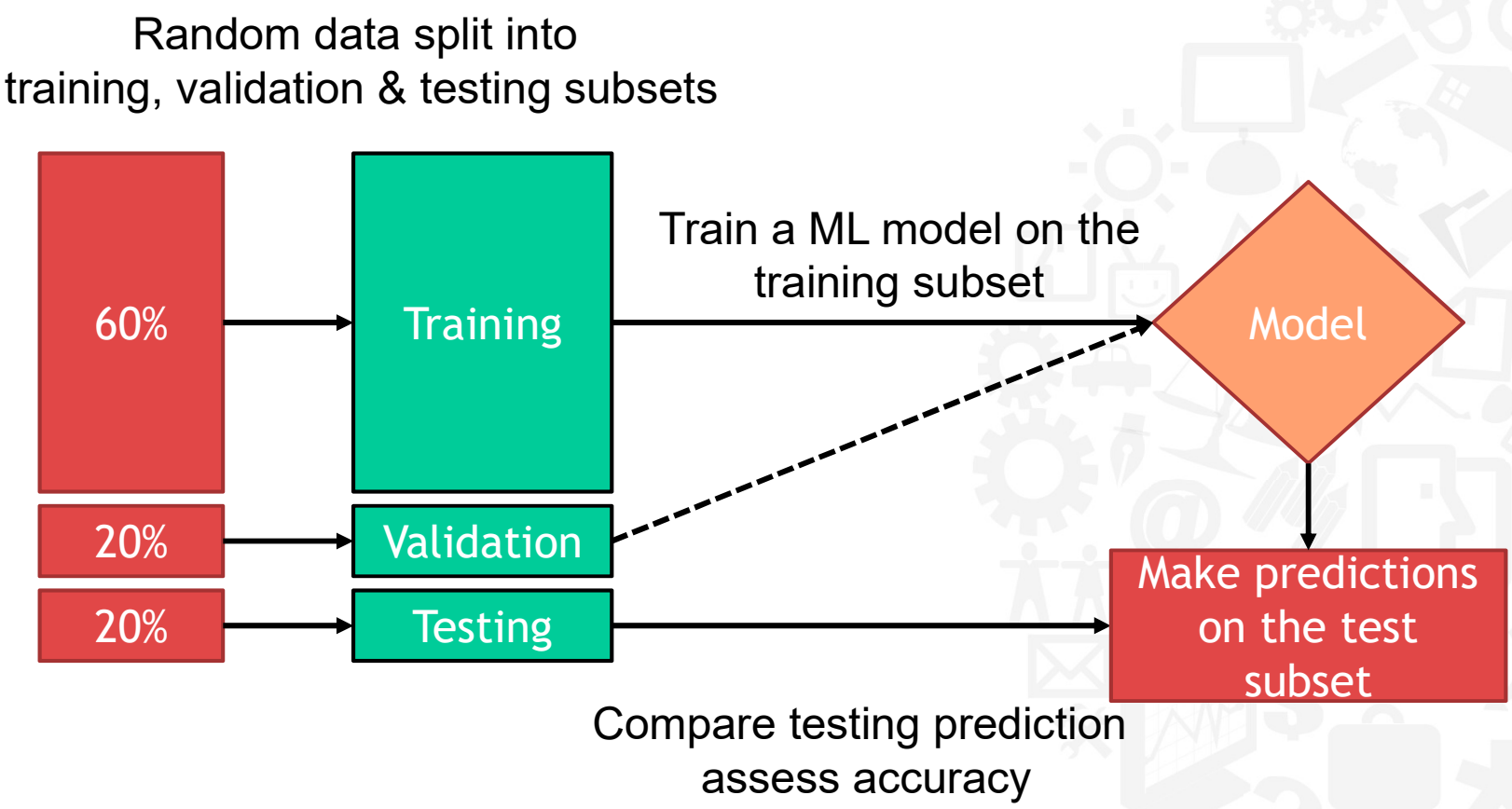


Normalization

- 정규화는 데이터의 범위를 0과 1로 변환하여 데이터 분포를 조정하는 방법
  - (해당 값- 최소값) / (최대값-최소값)
  - MinMax, Feature scaling 이라고도 함

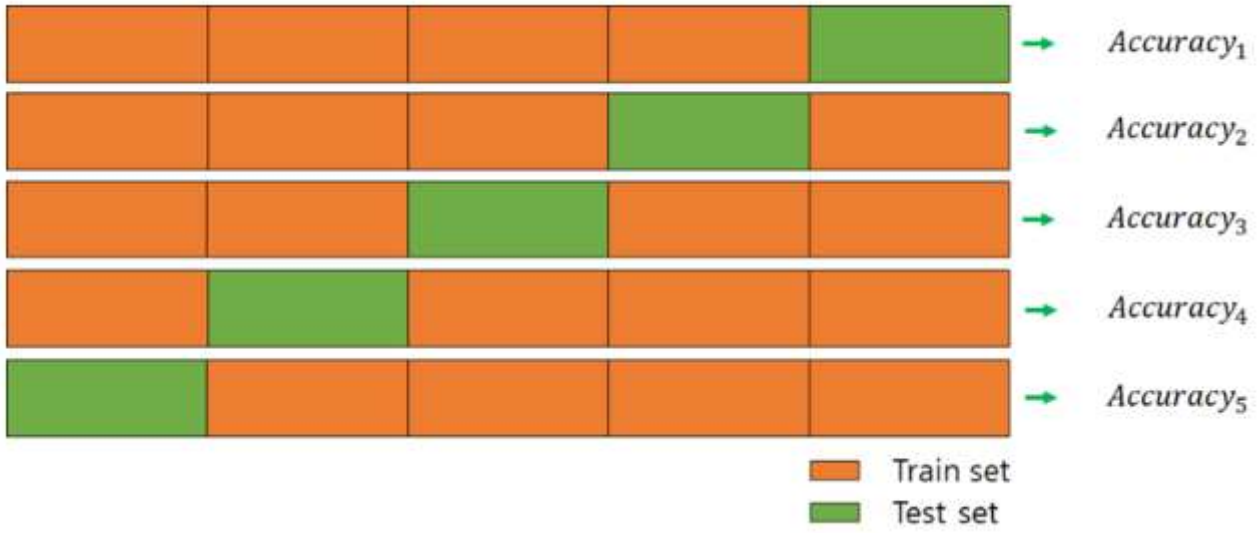
$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Training and testing splitting

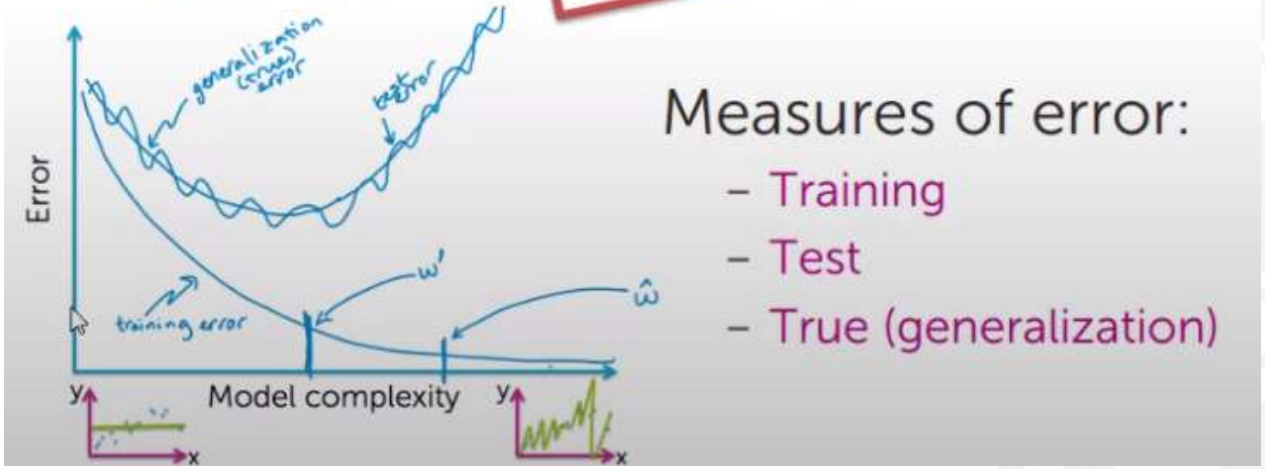


K-Folds Cross Validation

- 사용 이유
  - Test data 가 고정되어 있는 경우 test set에 overfit 될 수 있음
  - Dataset이 작은 경우 validation과 test set으로 데이터를 빼면 학습 데이터가 줄어드는 경우
-



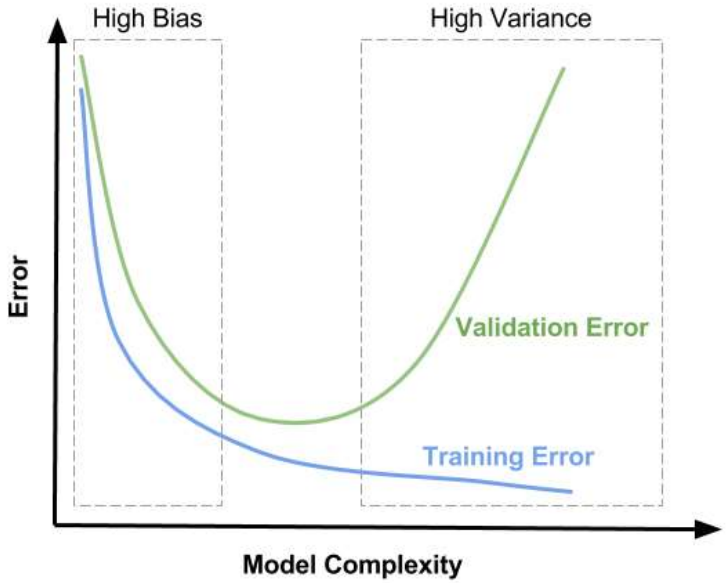
How do we determine the best fit line



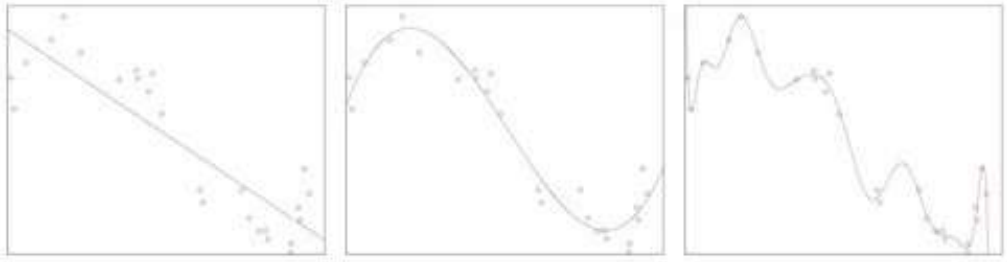
Measures of error:

- Training
- Test
- True (generalization)

Bias-Variance tradeoff



Model complexity



Error vs. amount of data

Bias-Variance tradeoff

- Bias (underfitting)
  - 추정값의 평균과 참값의 차이
  - 데이터 내에 있는 모든 정보를 고려하지 않음으로 인해 발생
- Variance (Overfitting)
  - 추정값의 평균과 추정값들간의 차이
  - 데이터 내에 있는 error나 noise까지 잘 잡아내도록 model fitting하여, 실제 관계없는 것까지 학습하는 경향

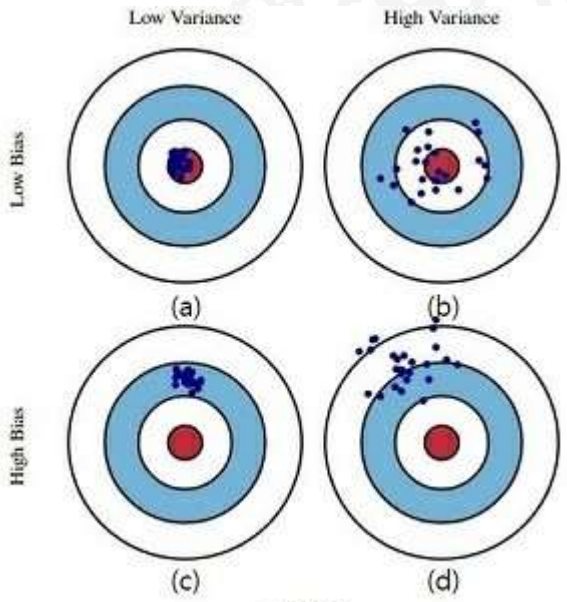


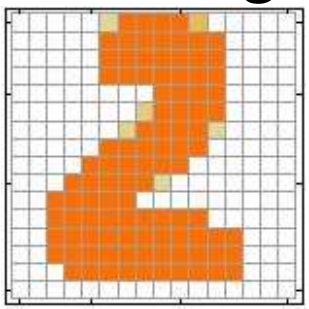
그림 1

LAB 1 : Housing Price Prediction

Classification Example

Deep learning example

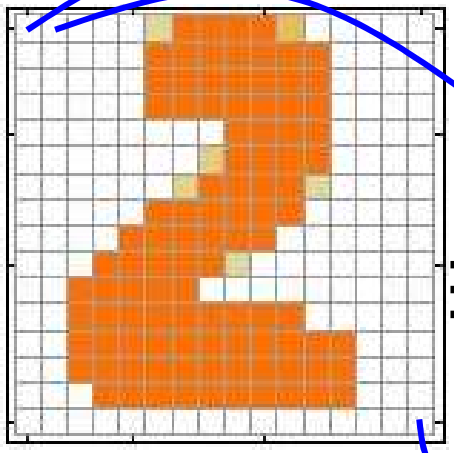
- Handwriting digit recognition



Machine

"2"

- Input



x_1

x_2

x_{256}

16 x 16 = 256

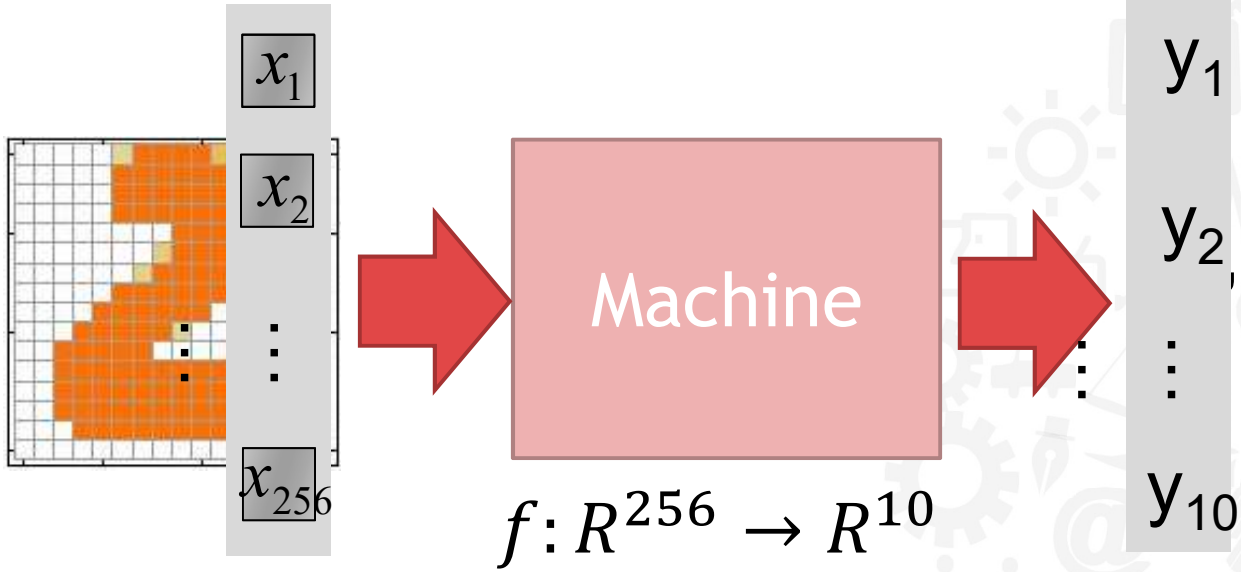
Output

0.1	is 1
0.7	is 2
0.2	is 0

The image is "2"

Each dimension represents the confidence of a digit.

Handwriting digit recognition

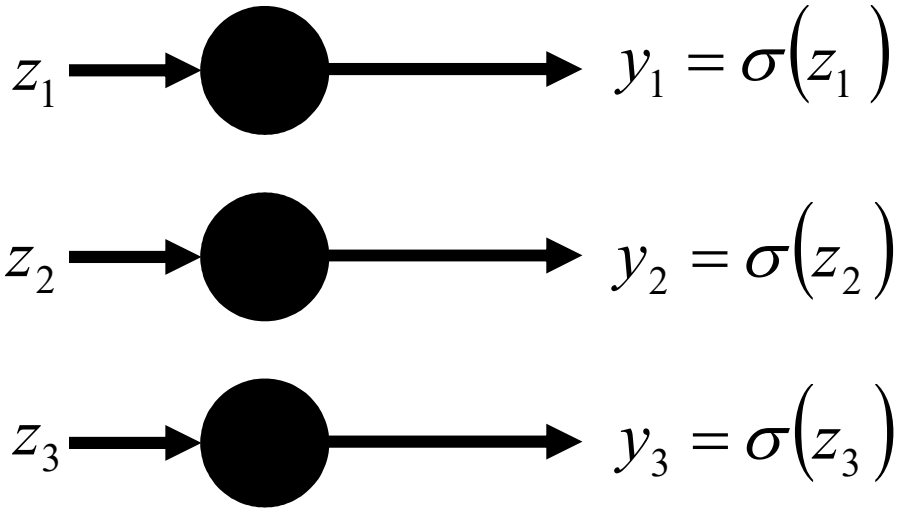


In deep learning, the function f is represented by neural network

Softmax

- Softmax layer as the output layer

Ordinary Layer



In general, the output of network can be any value

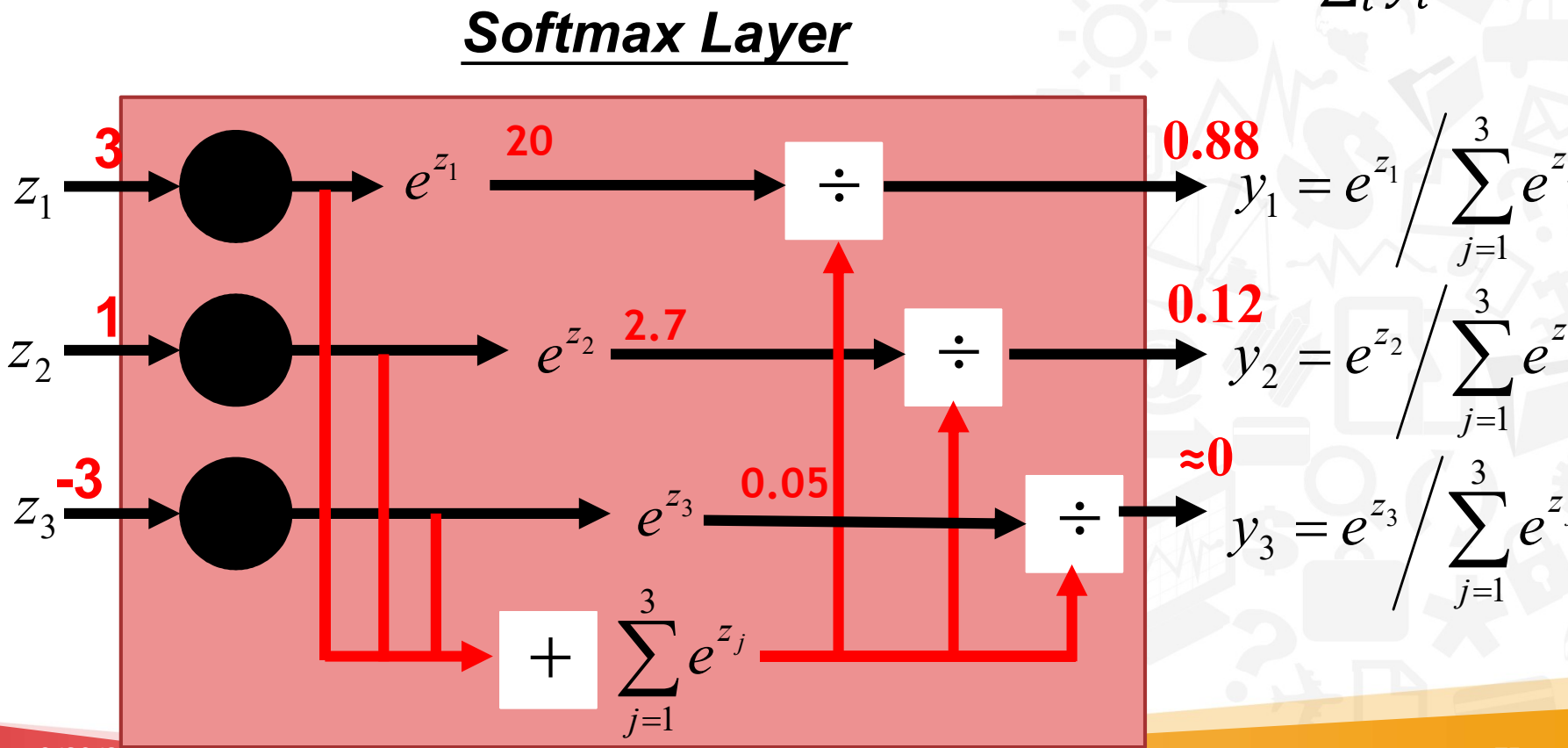
may not be easy to interpret

softmax

- Softmax as the output layer

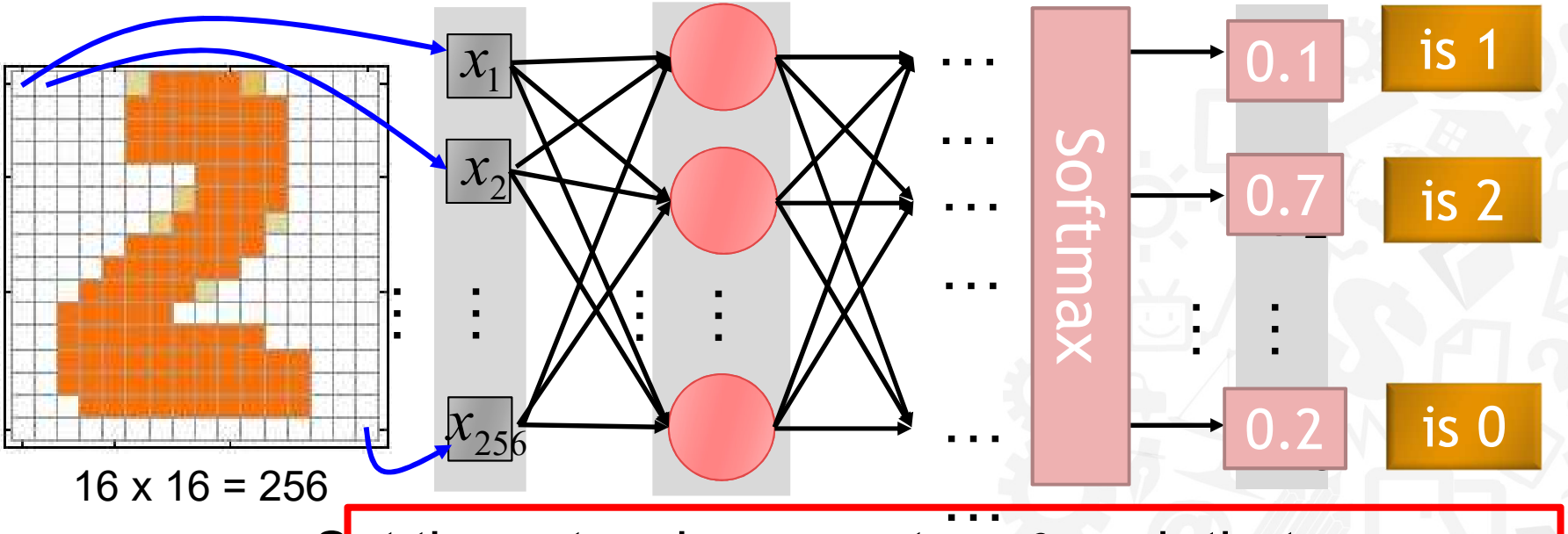
Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$



How to set network parameters

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



16 x 16 = 256

Ink → 1

No ink → 0

Set the network parameters θ such that ……

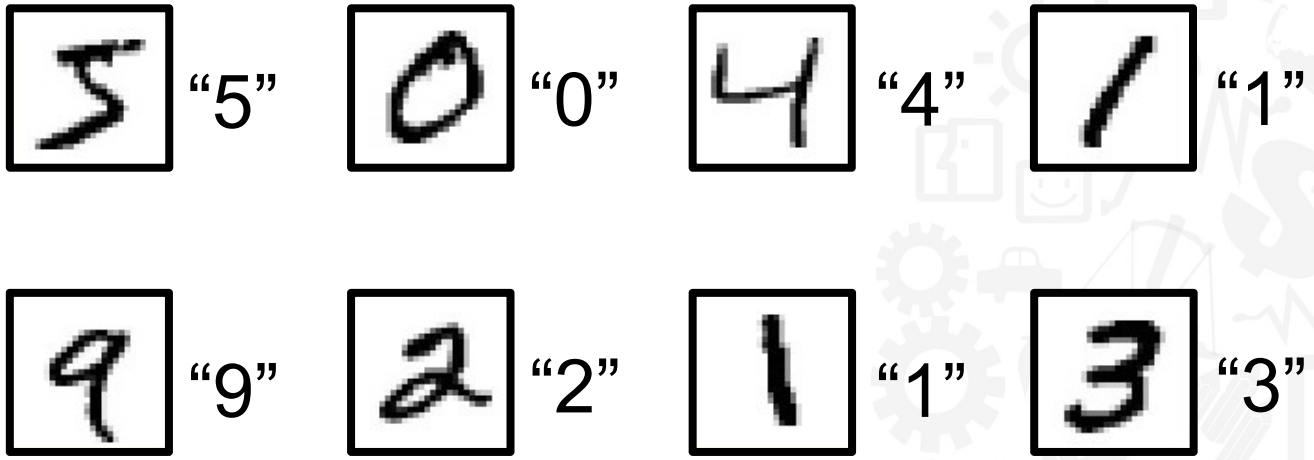
Input: m value

Input: y_2 has the maximum value

How to let the neural network achieve this

Training Data

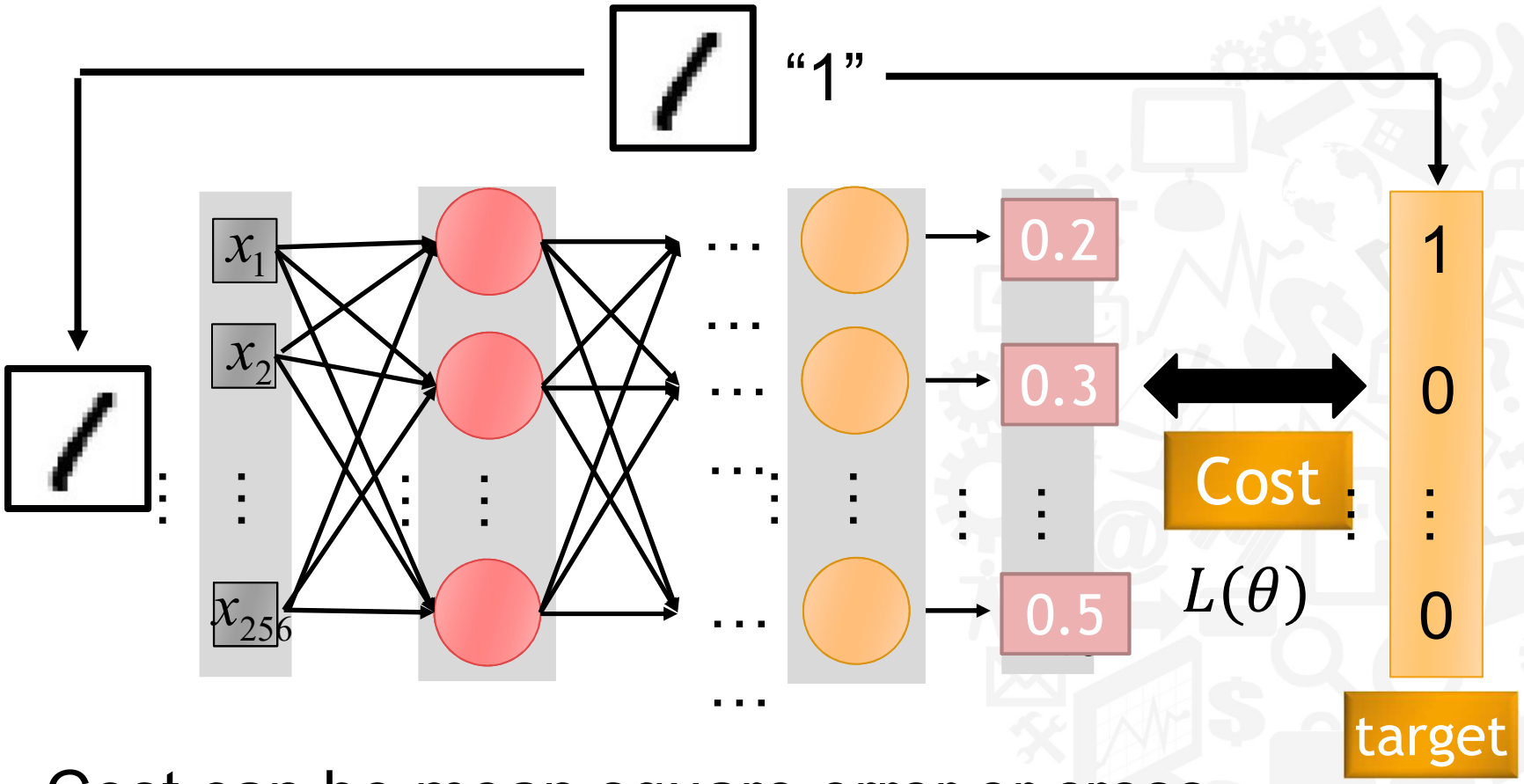
- Preparing training data: images and their labels



Using the training data to find the network parameters.

Cost

Given a set of network parameters θ, each example has a cost value.

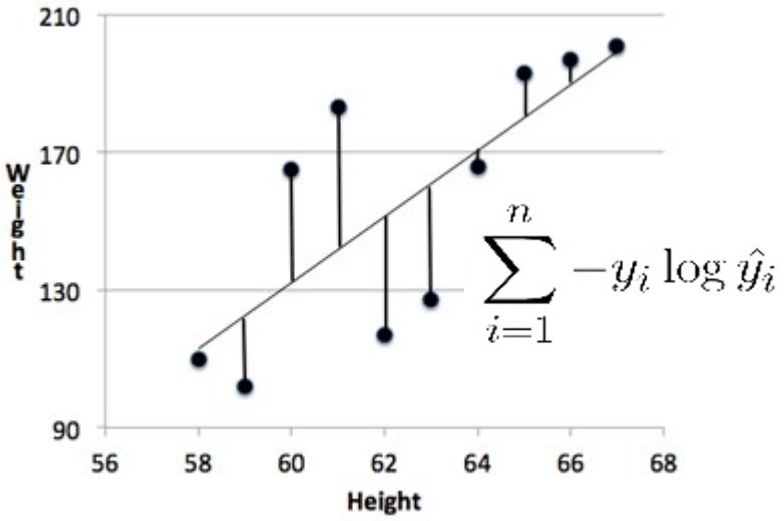


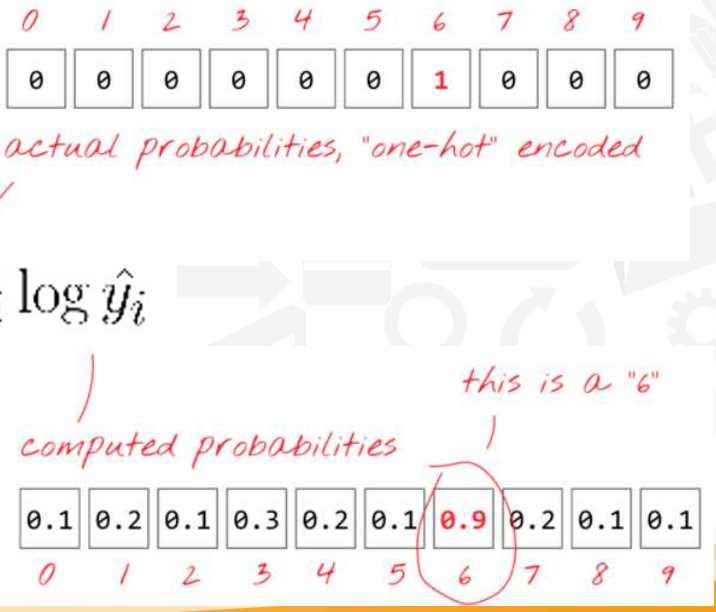
Cost can be mean square error or cross-entropy of the network output and target

Cost function

Mean Squared Error: $$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

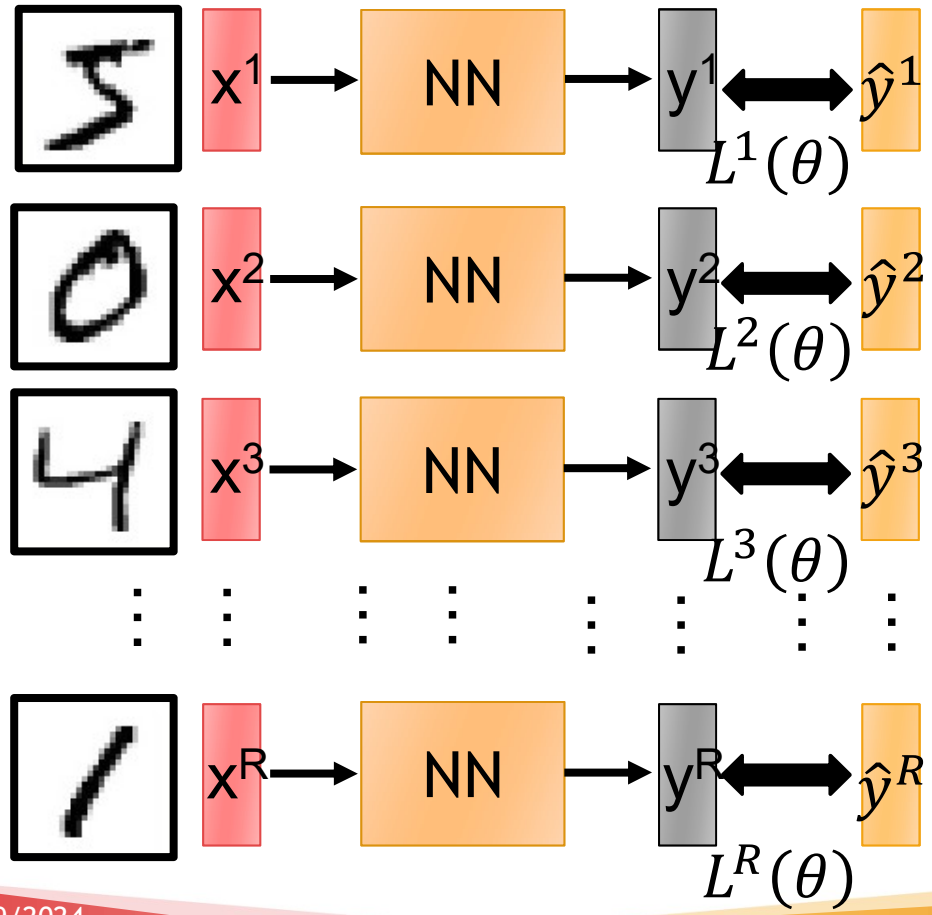
Cross Entropy: $$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{n} -y_i \log \hat{y}_i$$





Total Cost

For all training data …



Total Cost:

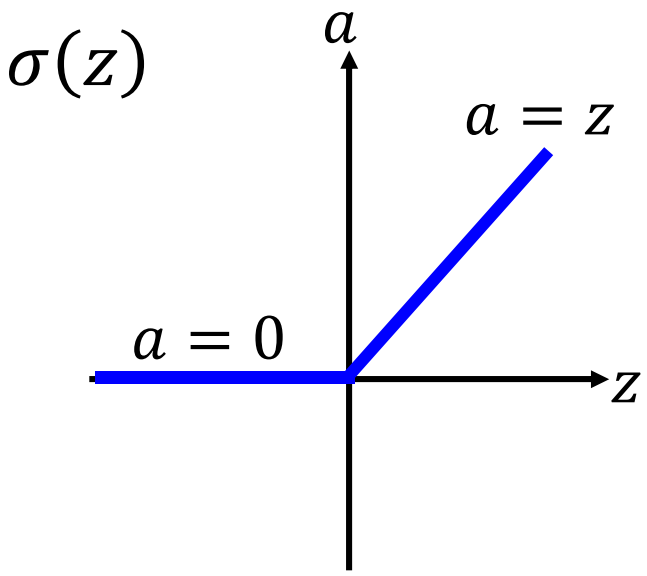
$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

How bad the network parameters θ is on this task

Find the network parameters θ^* that minimize this value

RELU

- Rectified Linear Unit(ReLU)



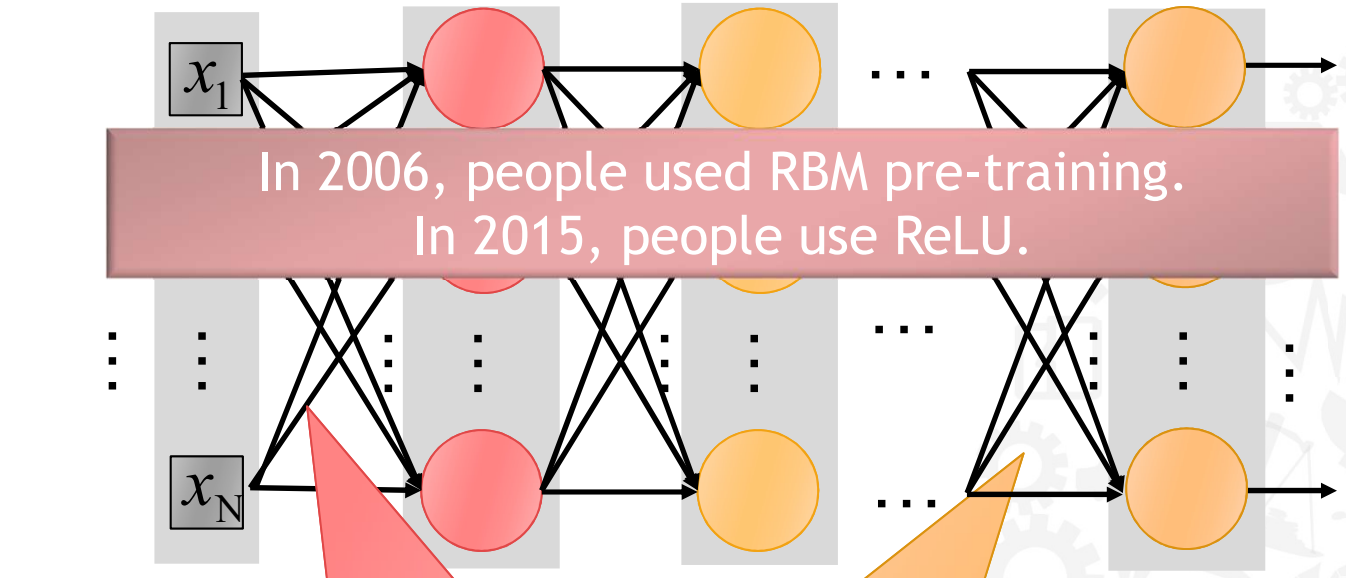
[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

Reason:

1. Fast to compute

2. Vanishing gradient problem

Vanishing gradient problem



In 2006, people used RBM pre-training.
In 2015, people use ReLU.

Smaller gradients

Learn very slow

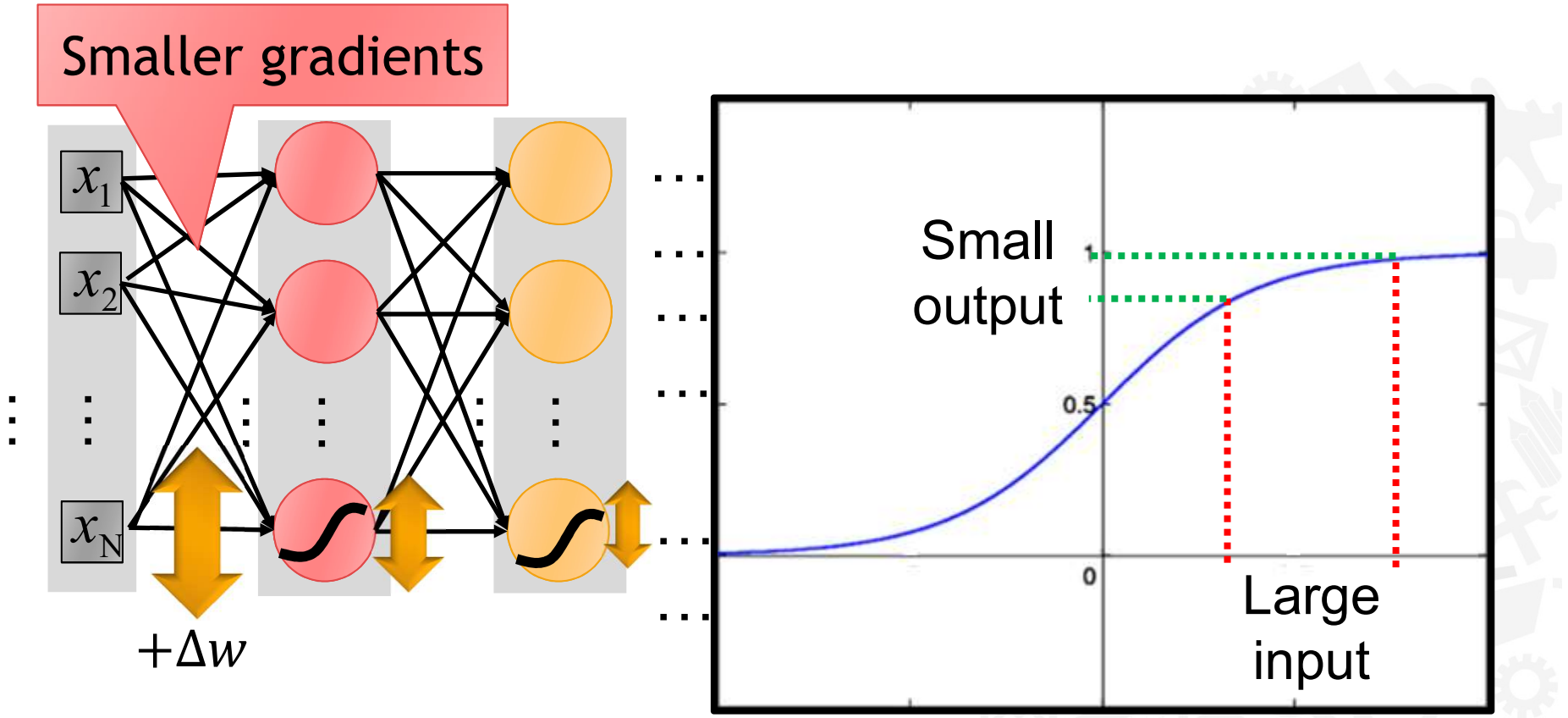
Almost random

Larger gradients

Learn very fast

Already converge

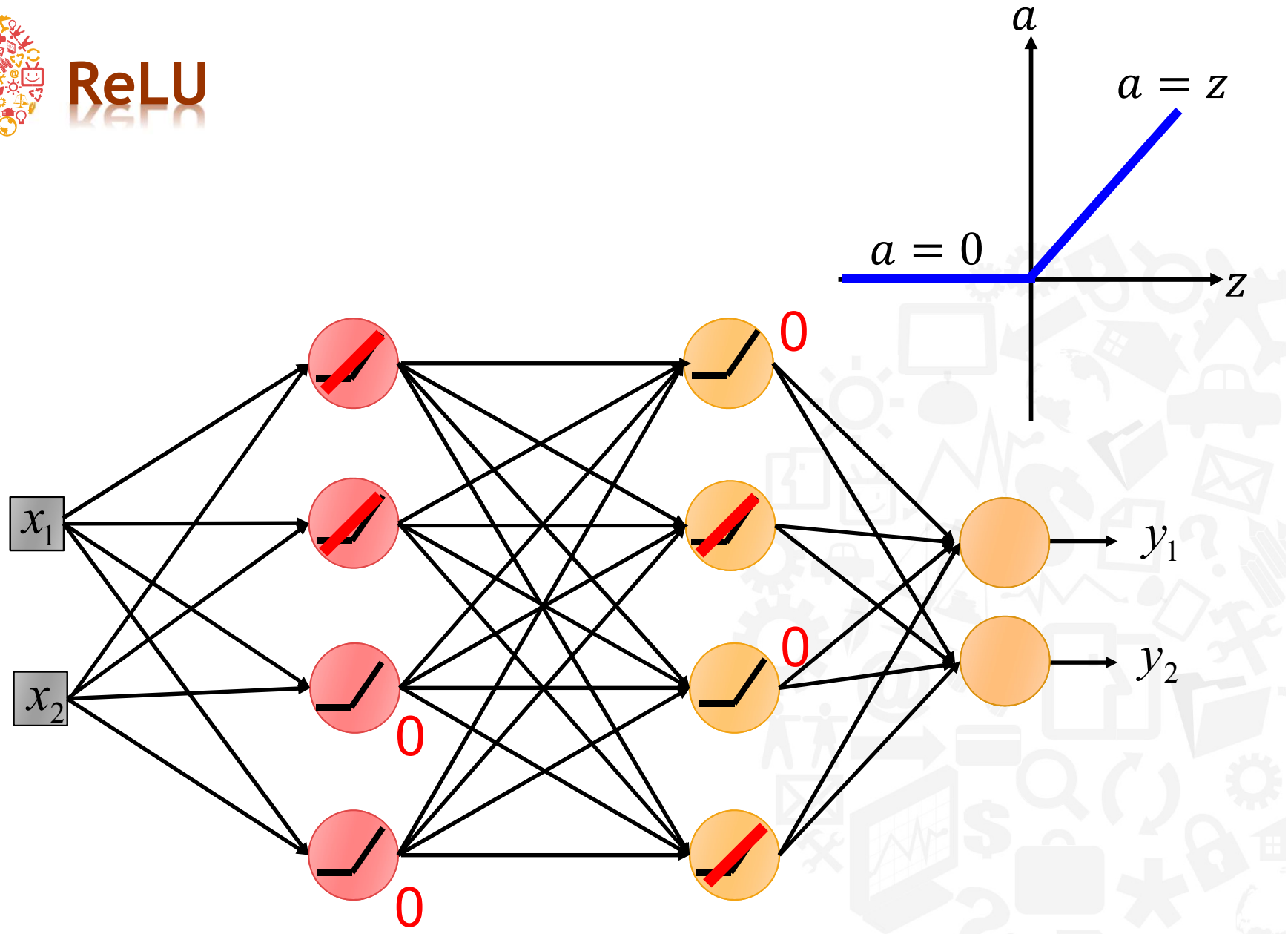
Vanishing gradient problem



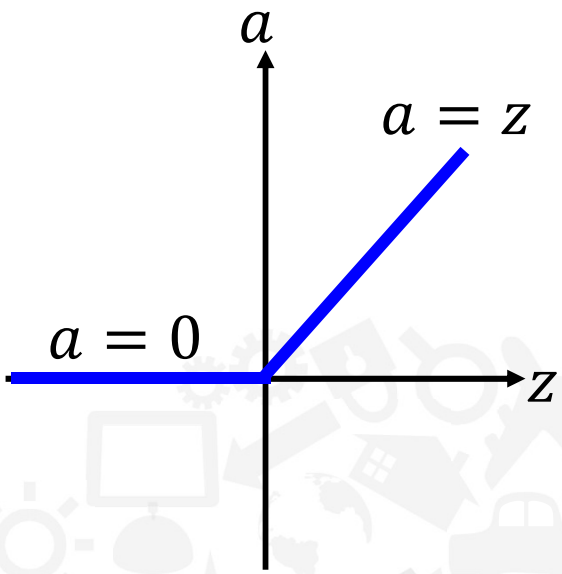
Intuitive way to compute the gradient …

$$\frac{\partial C}{\partial w} =? \frac{\Delta C}{\Delta w}$$

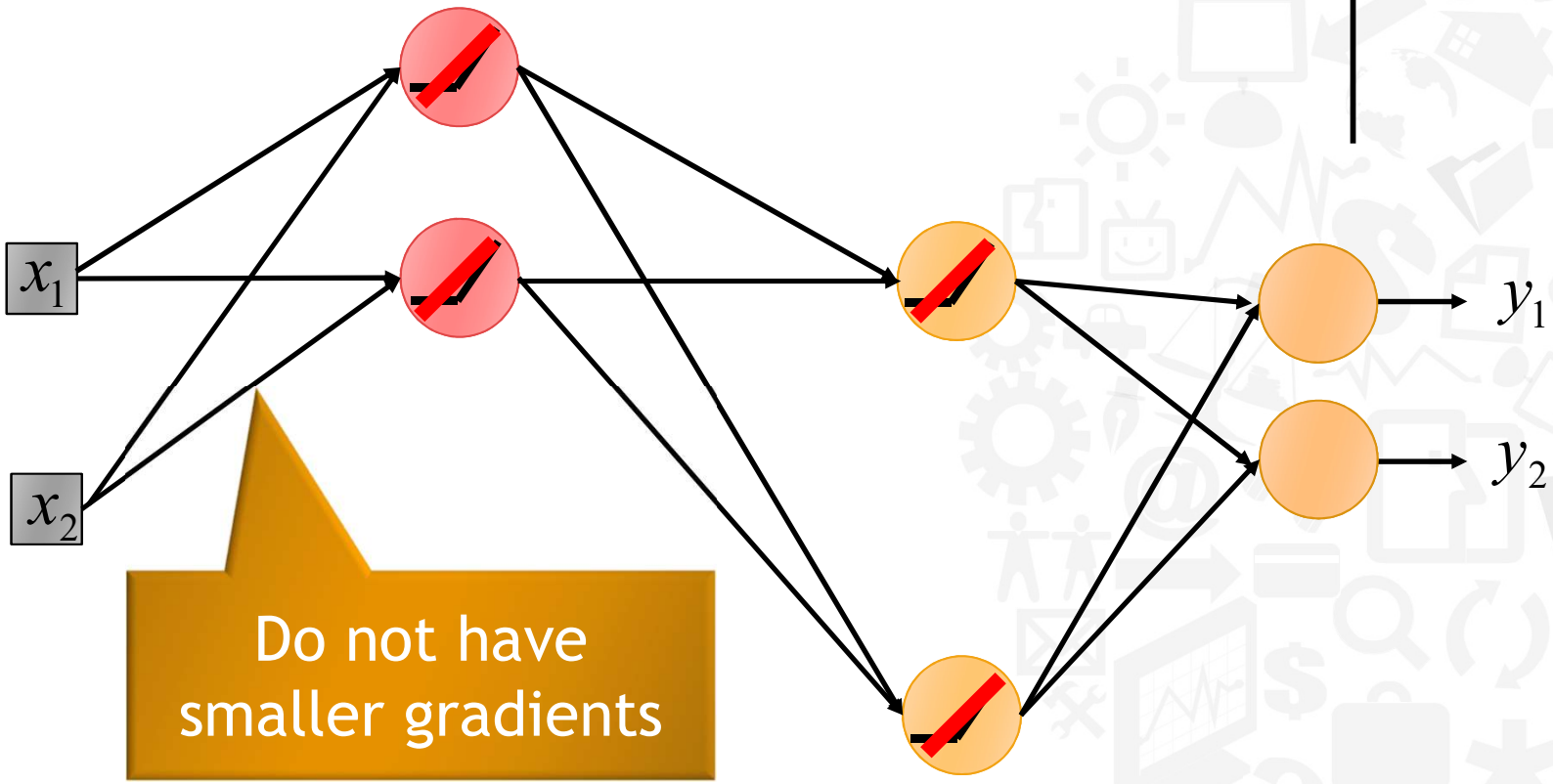
ReLU



ReLU



A Thinner linear network



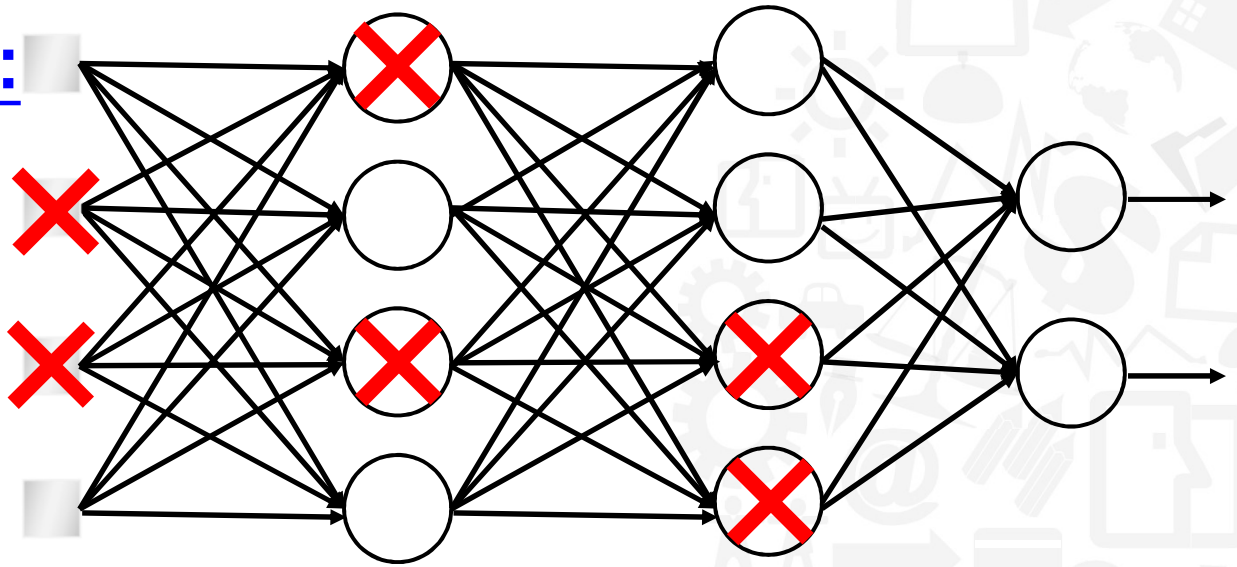
Do not have smaller gradients

DropOut

Pick a mini-batch

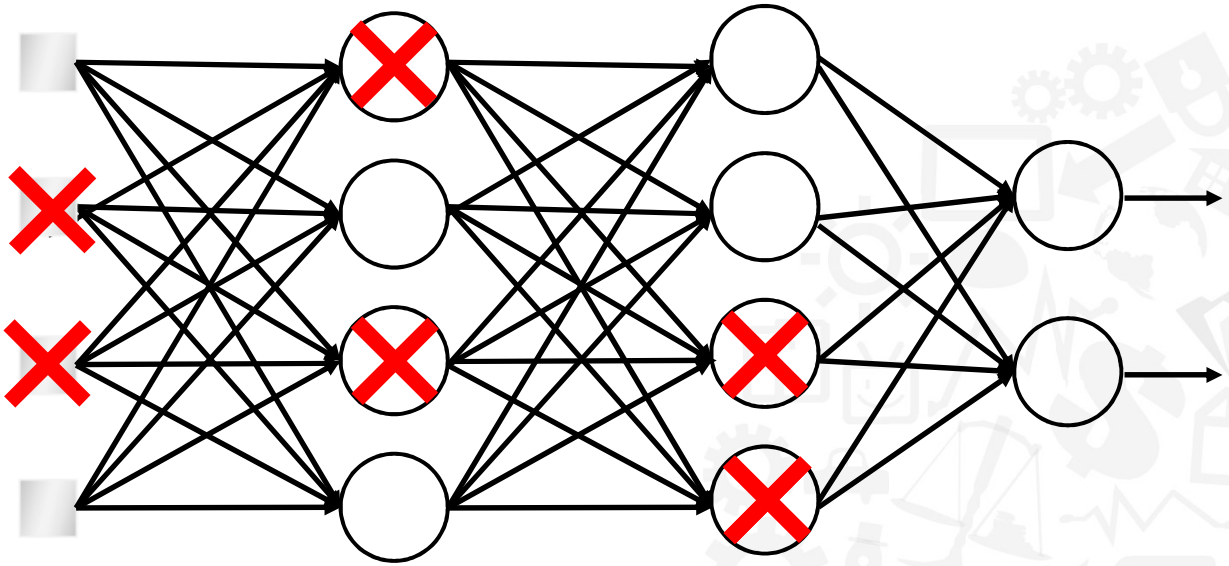
$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Training:



- **Each time before computing the gradients**
  - Each neuron has p% to dropout

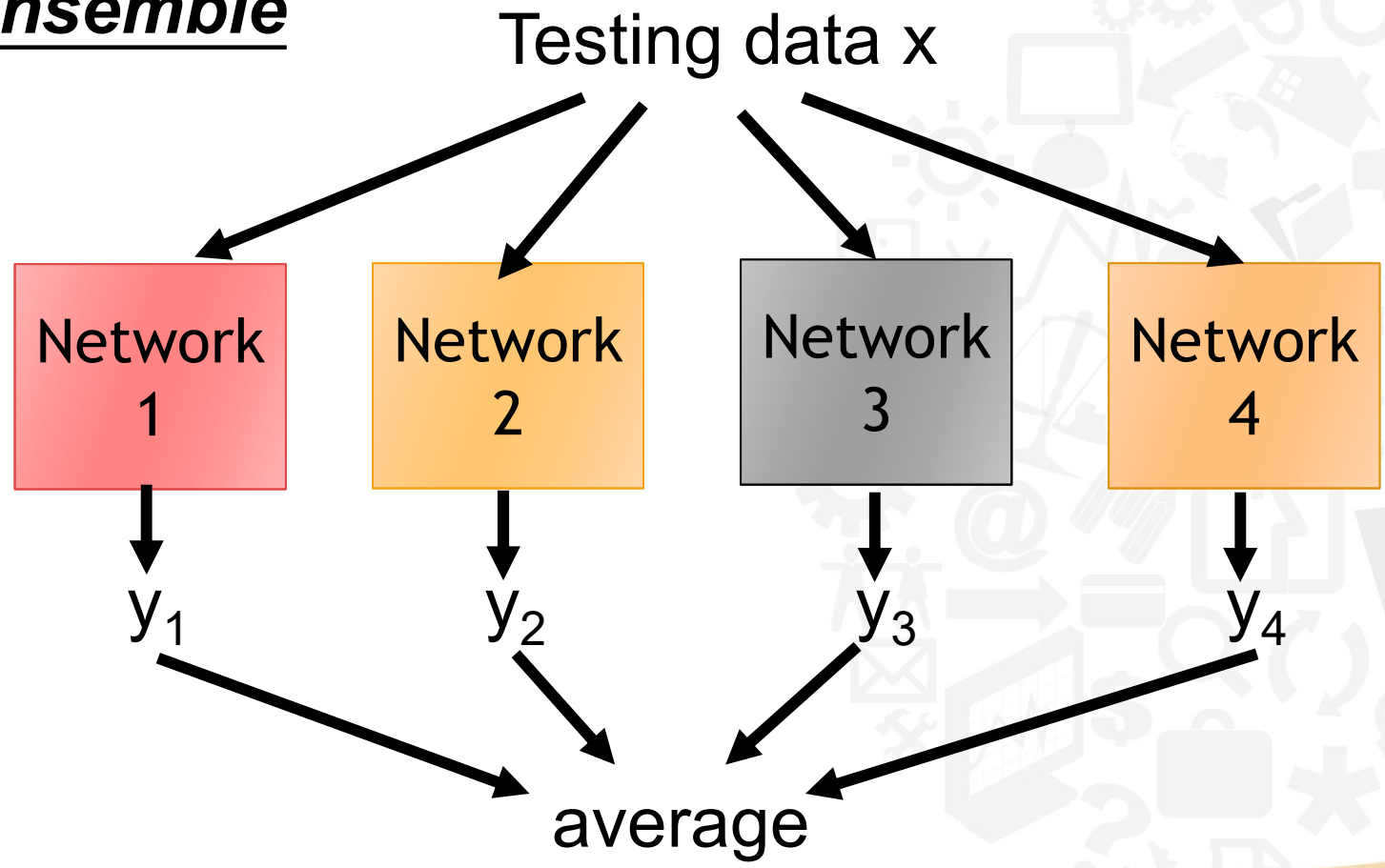
DropOut



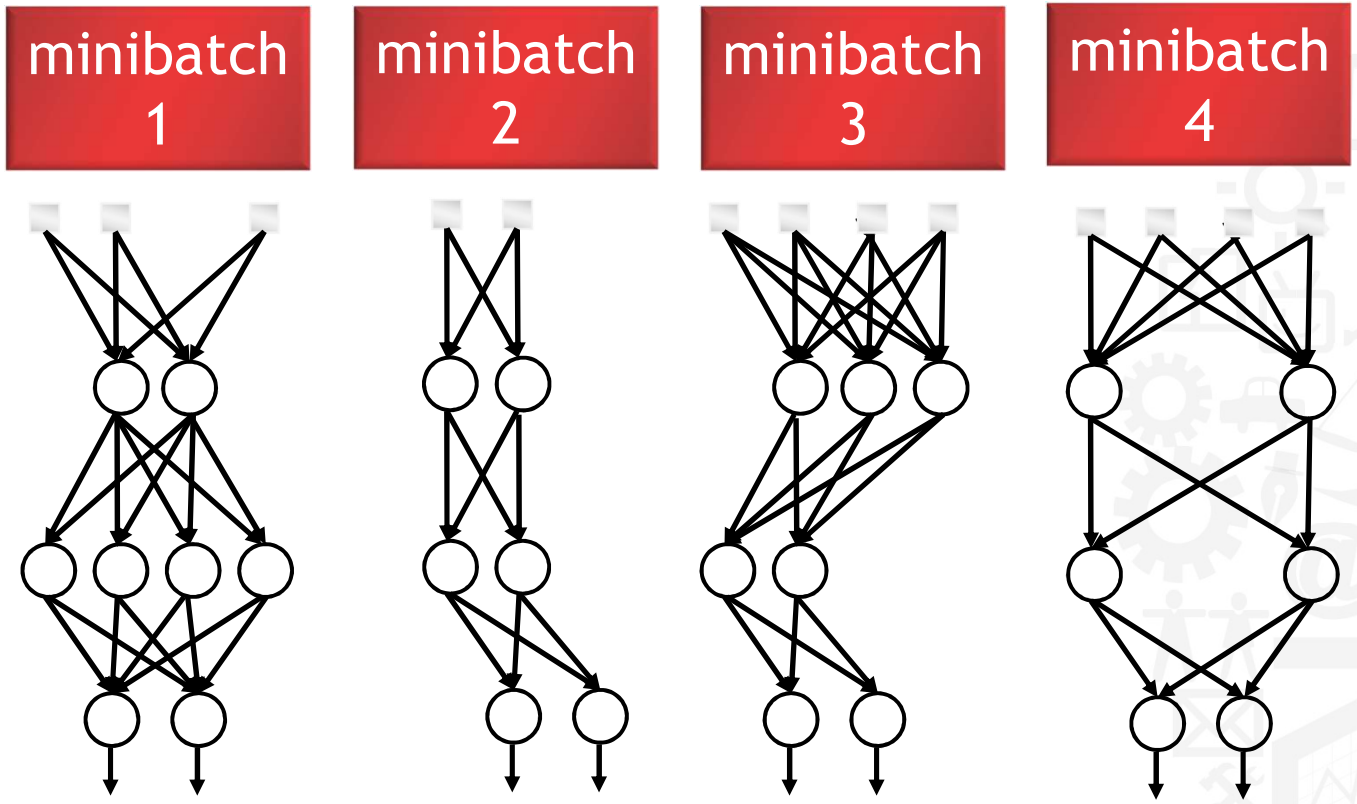
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

DropOut is a kind of ensemble

Model Ensemble



DropOut is a kind of ensemble



Training of Dropout

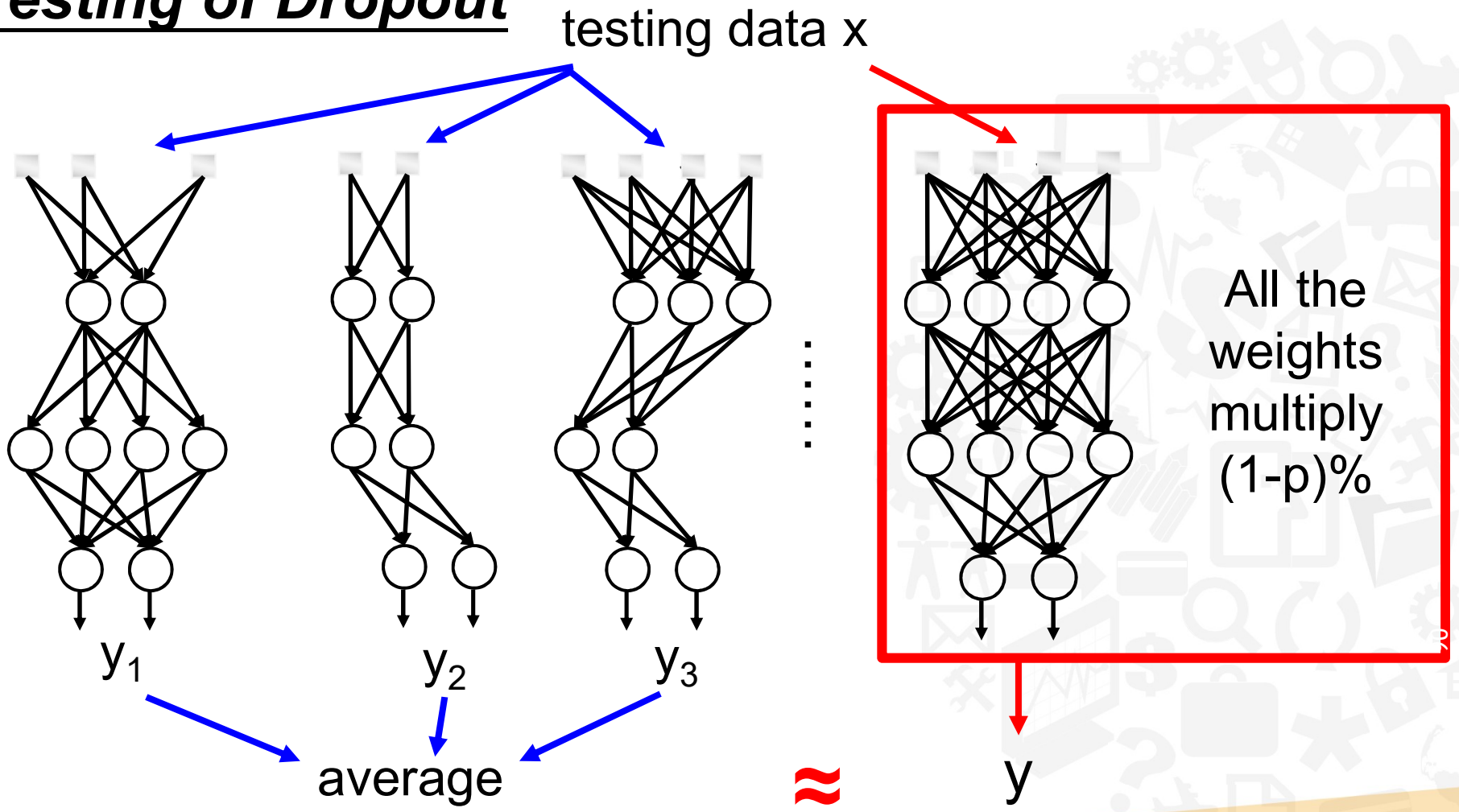
M neurons

2^M possible networks

- Using one mini-batch to train one network
- Some parameters in the network are shared

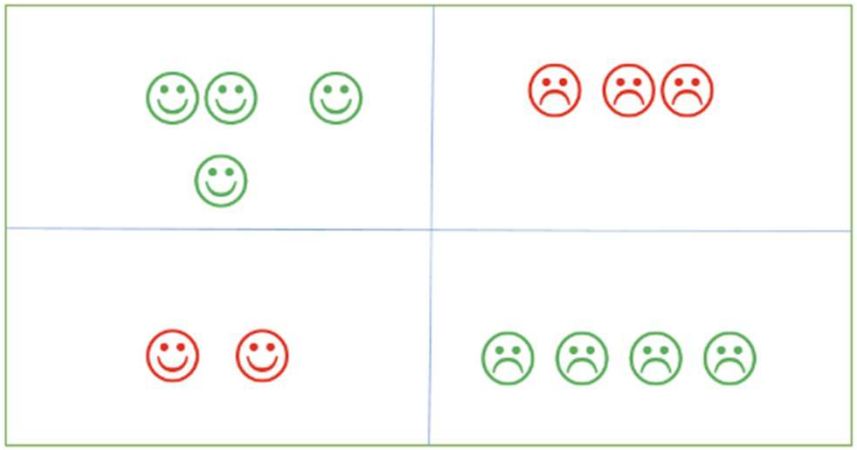
Dropout is a kind of ensemble.

Testing of Dropout



Metrics

- Precision
  - TP/(TP+FP)
- Recall
  - TP/(TP+FN)
- Accuracy
  - $\frac{TP+TN}{TP+TN+FP+FN}$
- F1 Score
  - $\frac{2 \times precision \times Recall}{precision+recal}$

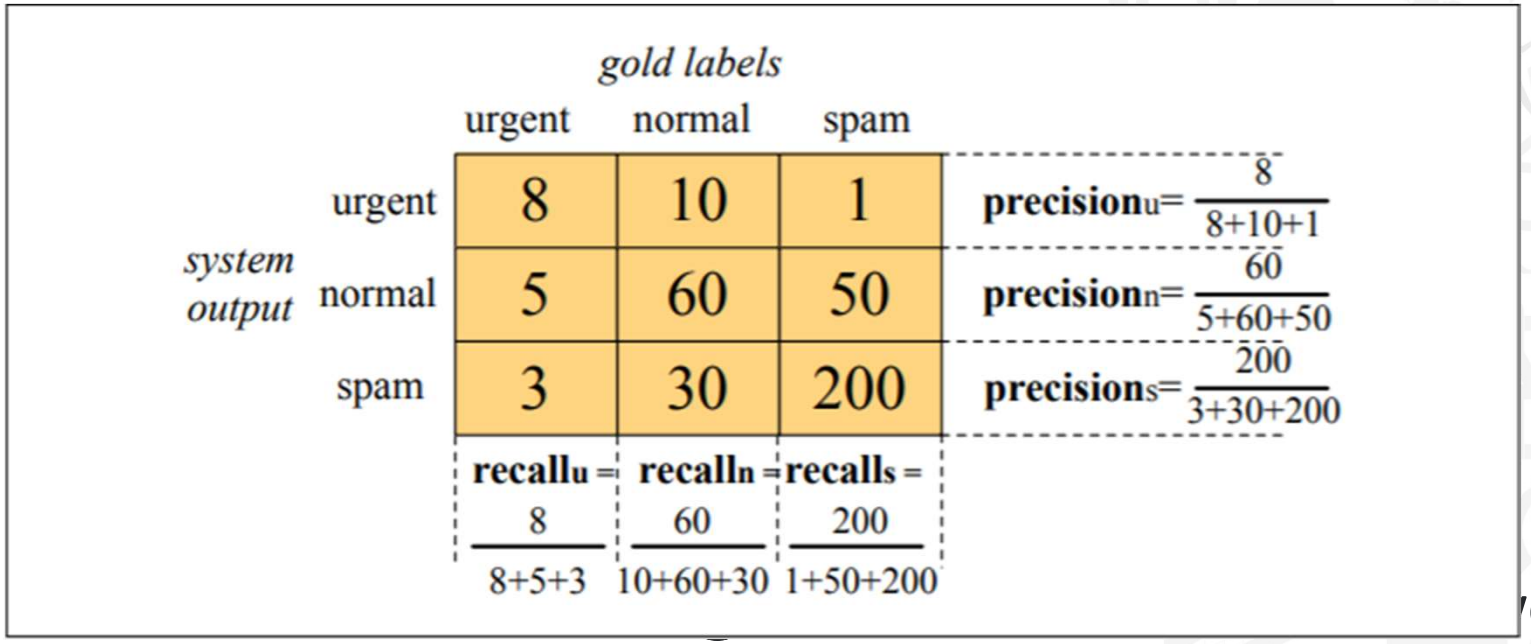


Predicted

Actual	Positive	Negative	
Positive	True positive(TP)	False Negative(FN)	Sensitivity or Recall or True Positive Rate=TP/(TP+FN)
Negative	False Positive (FP)	True Negative(TN)	Specificity or True Negative Rate=TN/(TN+FP)
	Precision or Positive Predictive Value=TP/(TP+FP)	Negative Predictive Value=FN/(FN+TN)	Accuracy=TP+TN/TP+TN+FP+FN

Metrics

- Assume that we are classifying an email into one of the three groups: urgent, normal and spam



- ell the system is doing?

Macro vs. Micro

- **Macro average**
  - we compute the performance for each class, and then average over classes.
- **Micro average**
  - we collect the decisions for all classes into a single confusion matrix, and then compute precision and recall from that table.

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Weighted Average

- Proportional Average that each score is multiplied by proportion of the class in your dataset

Label	Per-Class F1 Score	Support	Support Proportion	Weighted Average F1 Score
 Airplane	0.67	3	0.3	$(0.67 * 0.3) + (0.40 * 0.1) + (0.67 * 0.6)$ **= 0.64**
 Boat	0.40	1	0.1	
 Car	0.67	6	0.6	
Total	-	10	1.0	

LAB 2: Classifying Diabetes with NN

- Pima Indians Diabetes Database | Kaggle
  - Csv file download
    - https://vo.la/VOEZgg
- Python Notebook file download
  - https://vo.la/O7zrKO