ENGO 697

Remote Sensing Systems and Advanced Analytics

Session 9: How does deep learning fit into remote sensing systems and fundamental concepts

Dr. Linlin (Lincoln) Xu
Linlin.xu@ucalgary.ca
Office: ENE 221

	(1) Direct inversion	(2) LUT approach	(3) Numerical Approach	(4) Simulation & ML	(5) ML	**(6) DL**
f(.) is known	yes	yes	yes	yes	yes	**yes**
f(.) is partially known, i.e., form known, but with some unknown parameters U	no	no	Yes, estimate X and U together	no	no	**no**
f(.) unknown, (X,Y) known	no	no	no	no	yes	**yes**
f(.) unknown, (X,Y) unknown	no	no	no	no	no	**no**
If both f(.) and (X,Y) known, can accommodate both?	no	yes?	Yes? Use (X,Y) to estimate parameters in f(.)	yes?	Yes, use both simulated and observed data	**Yes, use both simulated and observed data**
Can use prior information? e.g., spatial prior and value prior	no	Yes? Use value prior for sampling	Yes? Use value prior of X in Bayesian estimation	Yes, Use value prior in sampling and spatial prior in Random fields	Yes, spatial prior in Random field approaches	**Yes, similar to ML**
Advantages	Knowledge-driven; Simple, easy	Knowledge-driven; Intuitive, easy, discrete fitting;	Knowledge-driven; estimate U; Efficient for simple f(.) in convex problems	Knowledge-driven; flexible; continuous fitting; good inter/extrapolation; faster than LUT	Data-driven; flexible; Classic;	**Strong modeling capability; automatic feature learning;**
Disadvantages	Unrealistic; rely on simple f(.)	Sensitive to accuracy of f(.), similarity metrics, sampling density and range; slow if LUT is large; bad for extrapolation;	Rely on efficiency of nonlinear solver; Slow; Local optimum;	Overfitting and underfitting risk to simulated data; difficult model selection; Sensitive to accuracy of f(.), similarity metrics, sampling density and range;	Weak modeling capability; Rely on "good" engineered features; Black-box; Overfitting, underfitting; Feature and model selection is difficult and slow	**Overfitting and underfitting; Black-box;**

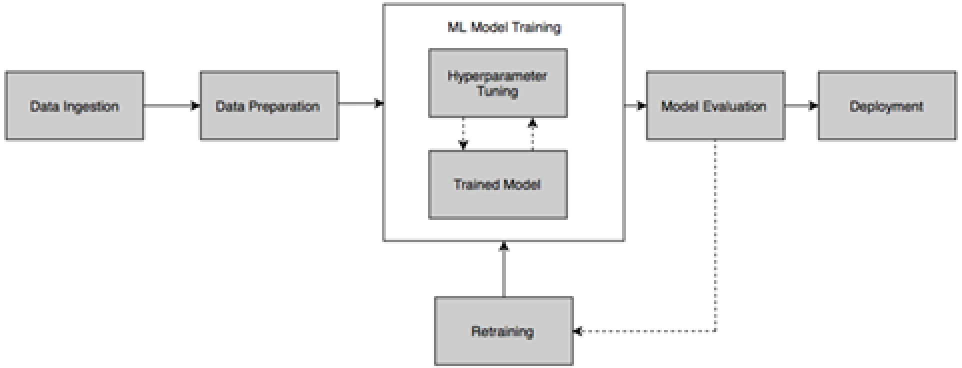
Machine Learning (ML) Approaches

All previous approaches assume that radiative transfer model f(.) is known. What if f(.) is unknown? How do we solve inverse problems? In this case, we need to collect both X (ground truth) and Y (remote sensing data) to build X and Y pairs, i.e., $\{(X_j,Y_j) \mid j=1,2,\ldots,T\}$, based on which we establish the inverse function $X=g(Y,\theta)$, where g(.) is a statistical or ML model, which is called empirical model.

No forward model: $Y = f(X)$, where f(.) is unknown.

We need to obtain some remote sensing data Y
and the associated ground truth data Y for building some (X,Y) pairs,
to be used as training data to train ML model.

X_1 : Y_1
X_2 : Y_2
X_3 : Y_3
….
X_4 : Y_4



Based on **$\{(X_j,Y_j) \mid j=1,2,\ldots,T\}$**, we build the following objective function:

$J(\theta) = \sum \|X_i - g(Y_i)\|$
$\theta = \min J(\theta)$

where θ is the unknown parameters in g(.). Once we know θ, we can establish the inverse function g(.), and use it to estimate the X value of an observed Y value by $X=g(Y)$.

Comparing with the data simulation & ML approach in (4), here the only difference is that the data is not simulated but observed for both X and Y. The ML approach is known as data-driven empirical approaches that are more and more widely used in remote sensing.

Deep Learning (DL) Approaches

Deep learning (DL) approaches are also ML approaches, and as such they can be used for data inversion through (4) and (5), i.e.,

--- if f(.) is known, we simulate $\{(X_j,Y_j) \mid j=1,2,...,T\}$ using f(.) and use them to train DL models for obtaining the inverse function X=g(Y);
--- if f(.) is unknown, we obtain remote sensing data Y and ground truth data X to build X and Y pairs, i.e., $\{(X_j,Y_j) \mid j=1,2,...,T\}$, and use them to train DL models for obtaining the inverse function X=g(Y);

Based on **training data**, we build the following objective function:

$J(\theta) = \sum||X_i-g(Y_i)||$
$\theta = \min J(\theta)$

where θ is the unknown parameters in DL model g(.). Once we know θ, we can establish the inverse function g(.), and use it to estimate the X value of an observed Y value by X=g(Y).

Comparing with traditional ML approaches, such as SVM and random forest, the DL approaches, due to their strong modeling capability and GPU computation, are more capable of effectively and efficiently learning the complex nonlinear relationship between Y and X, and perform accurate and fast model prediction for estimating X.

True Inverse Function vs. Approximated Inverse Function

Forward model:

$Y = f(X)$

(1) Y: received radiation by the sensor

(2) X: variables that you want to know, e.g., class labels, chlorophyll content in leaves, leaf area index/density;

True inverse function:

$X = t(Y) = f^{-1}(Y)$

where $f^{-1}(.)$ is difficult/impossible to get, and the form of t(.) is usually unknown; t(.) is physical model;

Approximated inverse function:

$X = g(Y)$

Note that g(.) is only an approximation to the true inverse function t(.), and g(.) is empirical model.

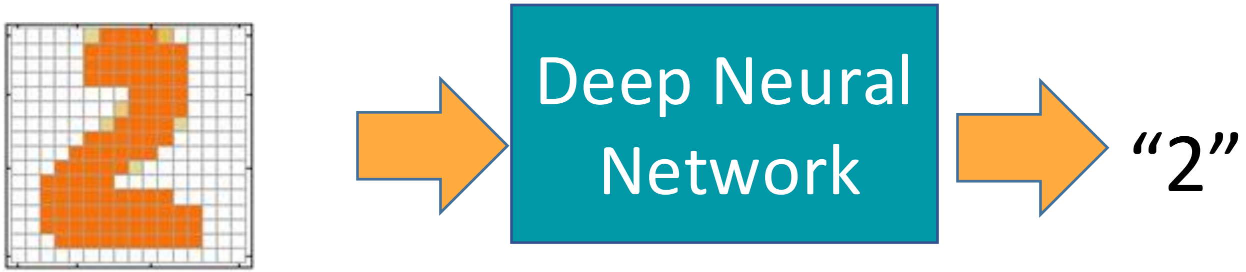
Based on $\{(X_j,Y_j) \mid j=1,2,\ldots,n\}$, we build the following objective function:

$J(\theta) = \sum||X_i-g(Y_i)||$

$\theta = \min J(\theta)$

Example Application

- Handwriting Digit Recognition



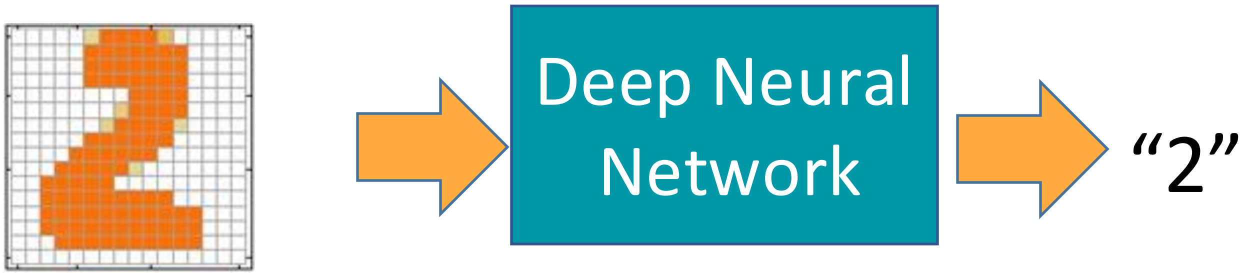
Q1: in this example, what is the observation Y?

Q2: what is underlying variable X that you try to estimate?

Q3: do you have a forward model?

Q4: how do you obtain your inverse function? Is this inverse model/function a physical model?

Inverse problem



Forward model: $Y = f(X)$

(1) Y: Digital image
(2) X: Image identity, i.e., the digit value in the image

Inverse model: $X = g(Y, \theta)$

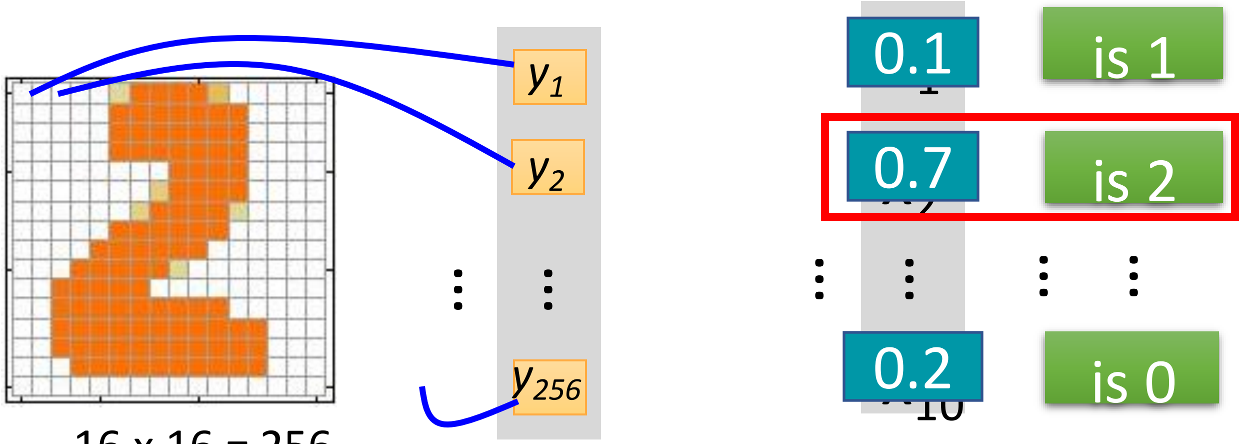
where g(.) is an unknown inverse function with unknown model parameter θ.

Knowledge, data and prior information?
--- Knowledge f(.) too complex and nonlinear, unknown; true inverse function $X = t(Y)=f^{-1}(Y)$ unknown
---- Data (X, Y) pairs abundant;
---- Prior information (e.g., spatial prior) ambiguous; pixels are spatially correlated to form the digit signature;

Handwriting Digit Recognition

Input

Output



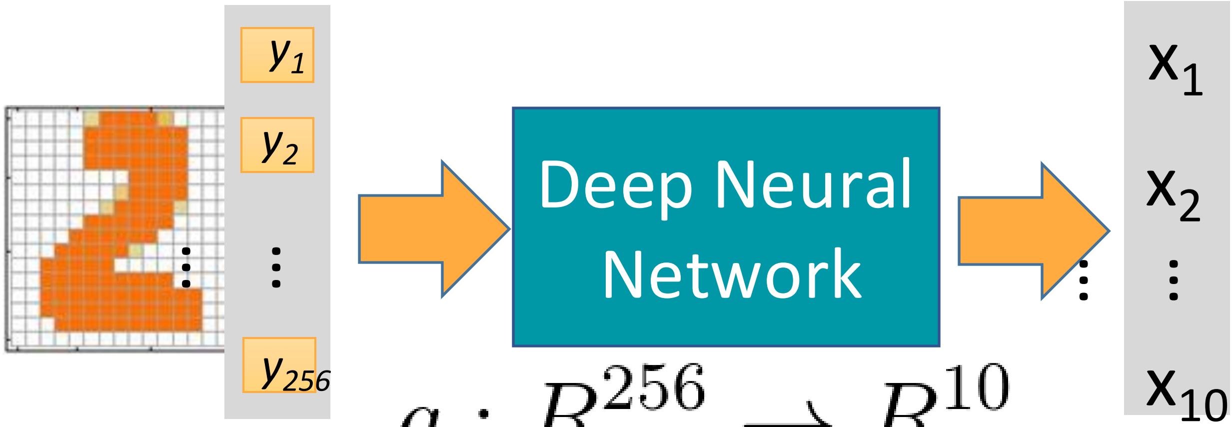
16 x 16 = 256

Ink → 1

No ink → 0

Example Application

- Handwriting Digit Recognition R256

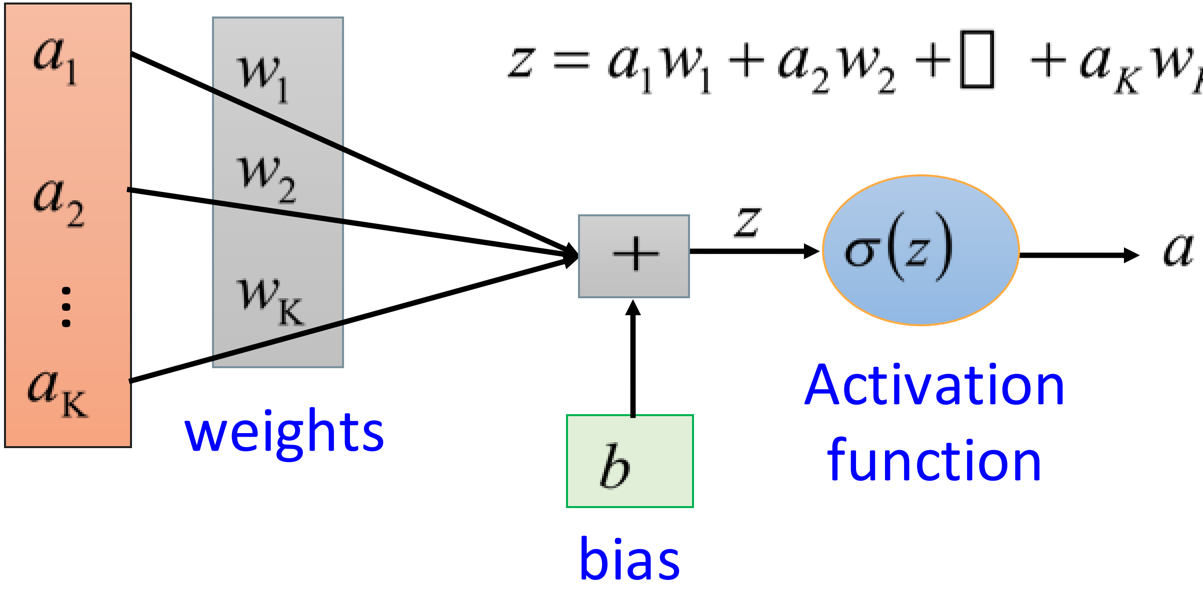


$$g : R^{256} \rightarrow R^{10}$$

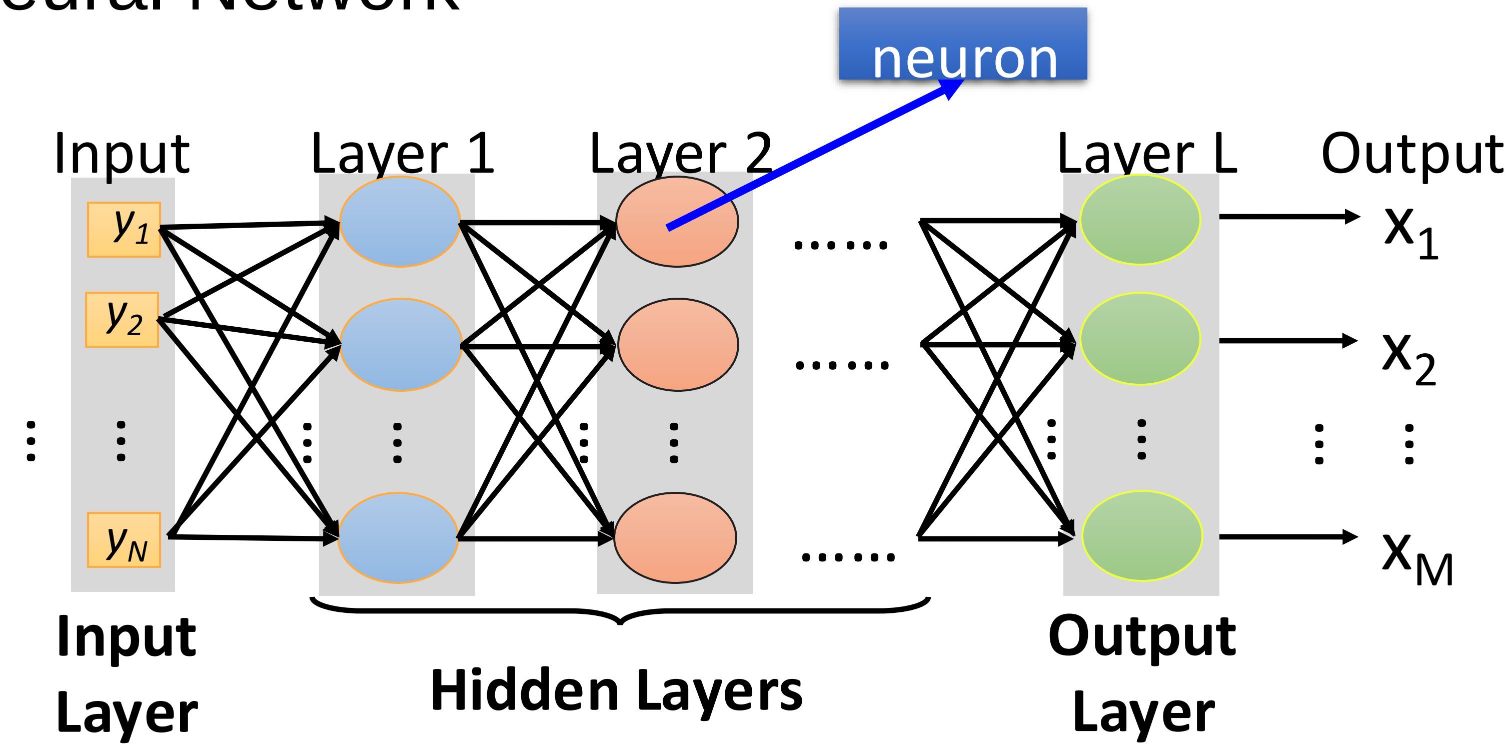
Element of Neural Network

Neuron $g : R^K \rightarrow R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

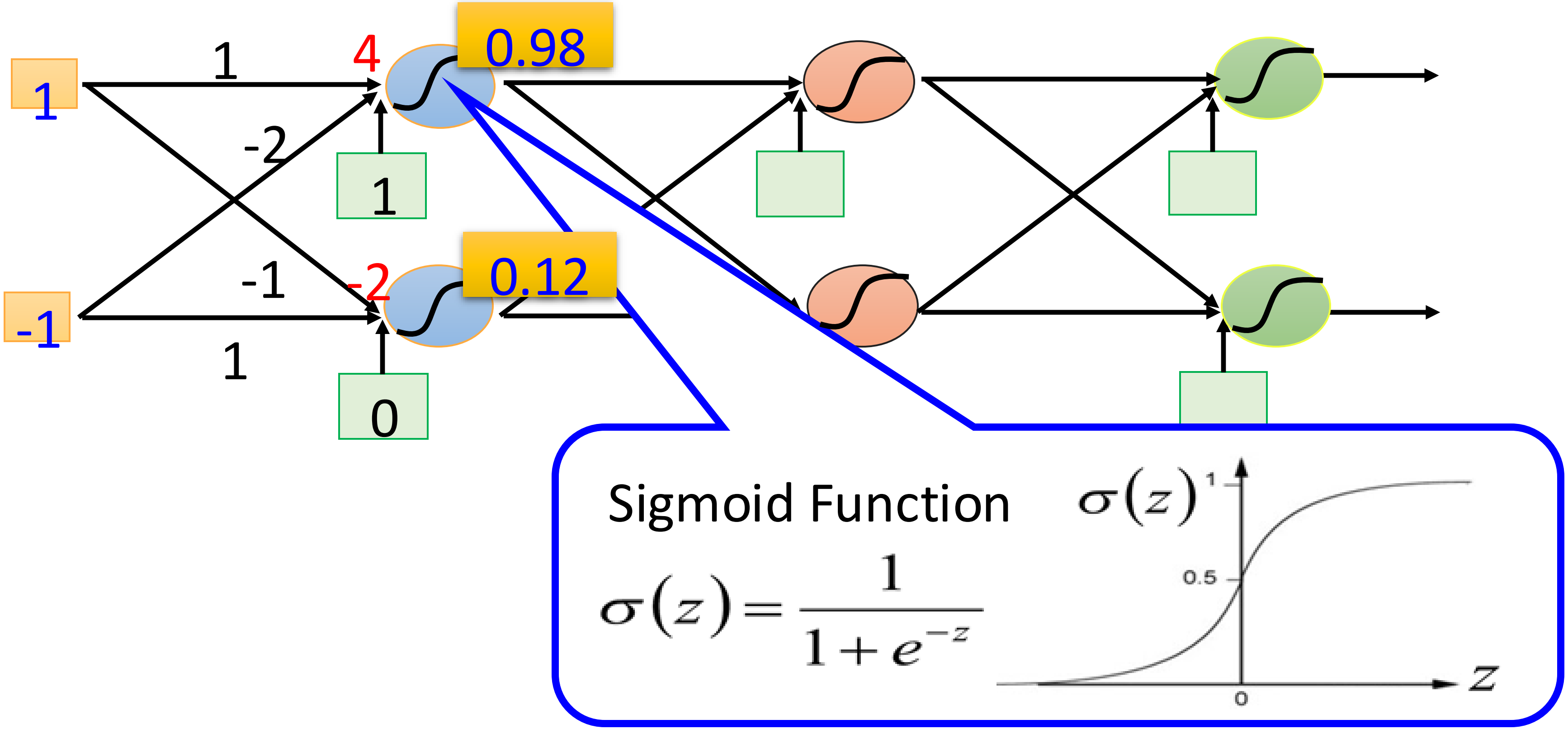


Neural Network

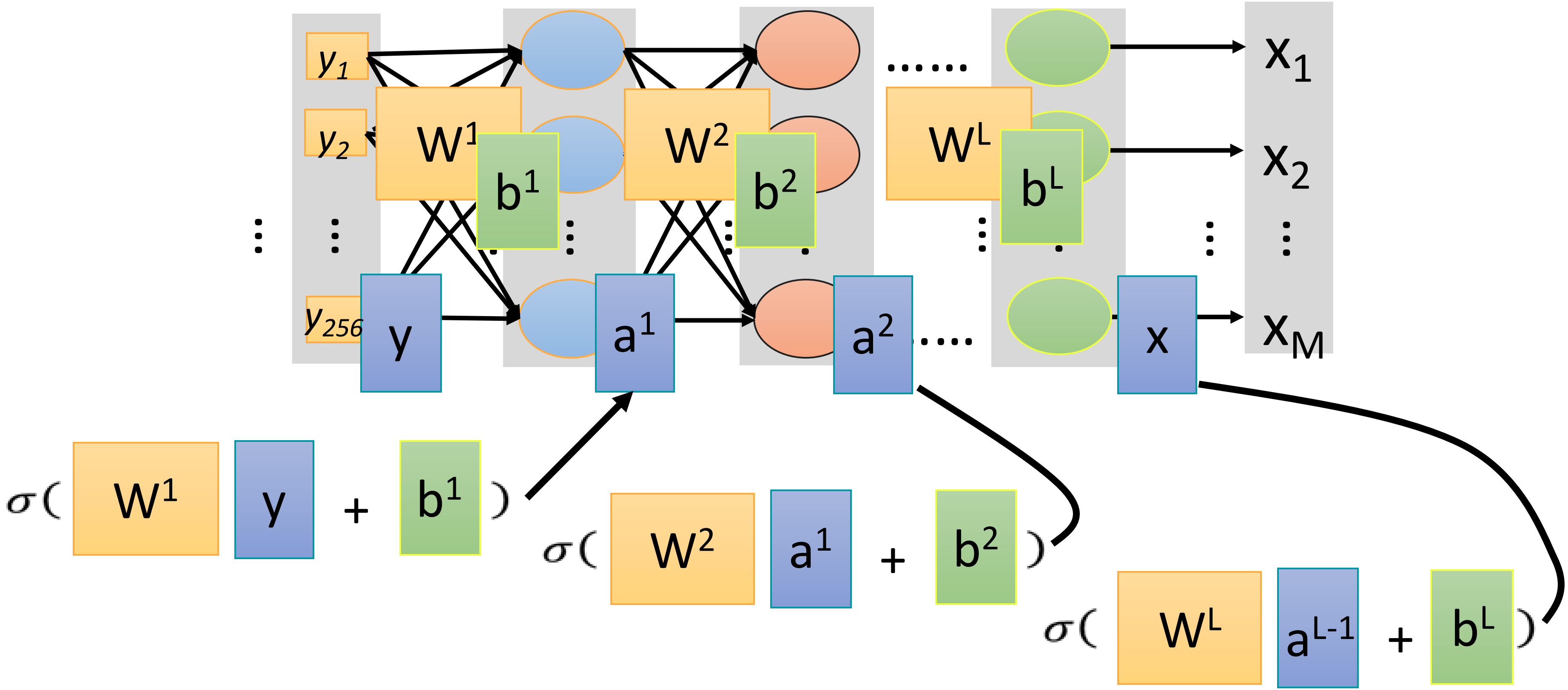


Deep means many hidden layers

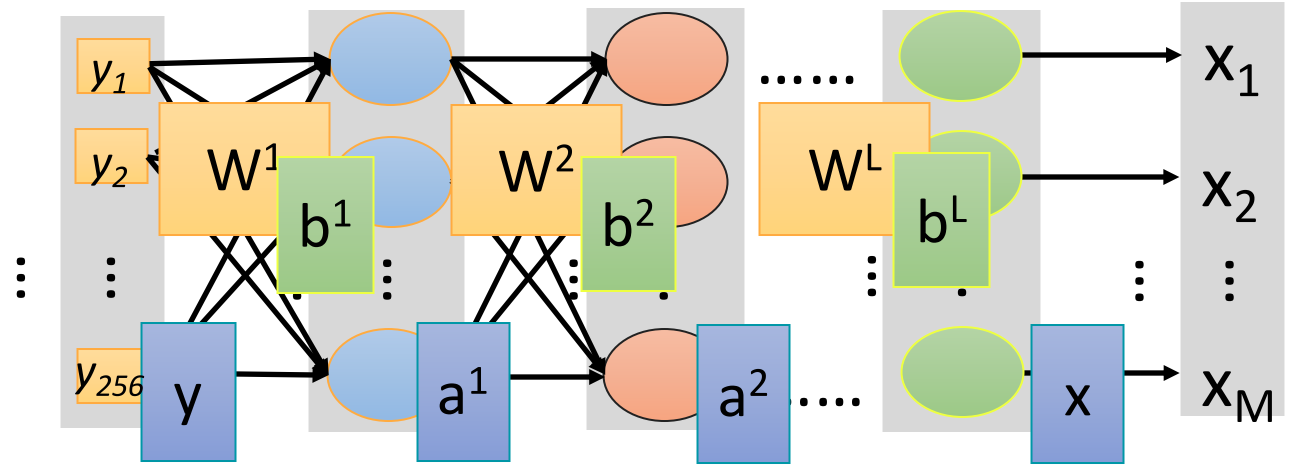
Example of Neural Network



Neural Network



Neural Network



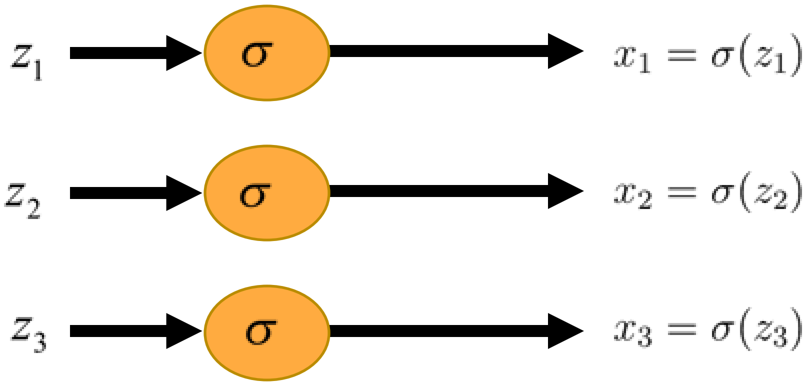
$$x = g(y)$$

$$= \sigma(W^L \ldots \sigma(W^2 \, \sigma(W^1 y + b^1) + b^2) \ldots + b^L)$$

Softmax

- Softmax layer as the output layer

<u>Ordinary Layer</u>



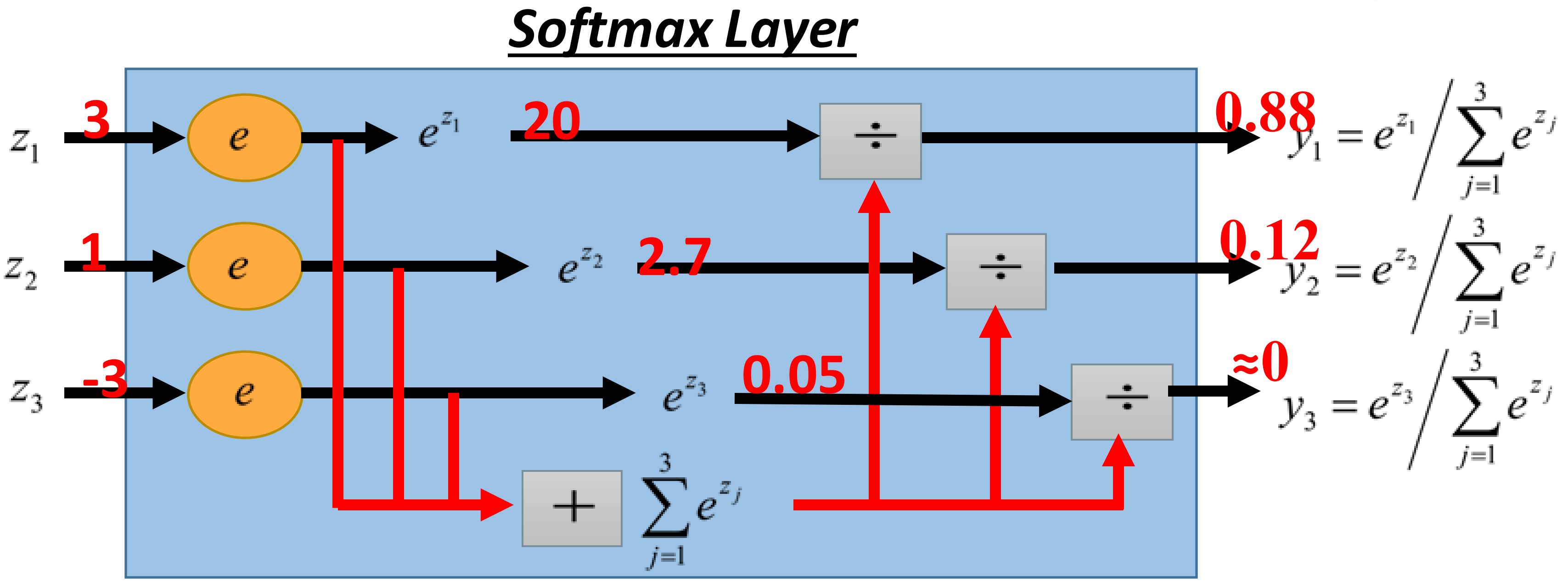
In general, the output of network can be any value.

May not be easy to interpret

Softmax

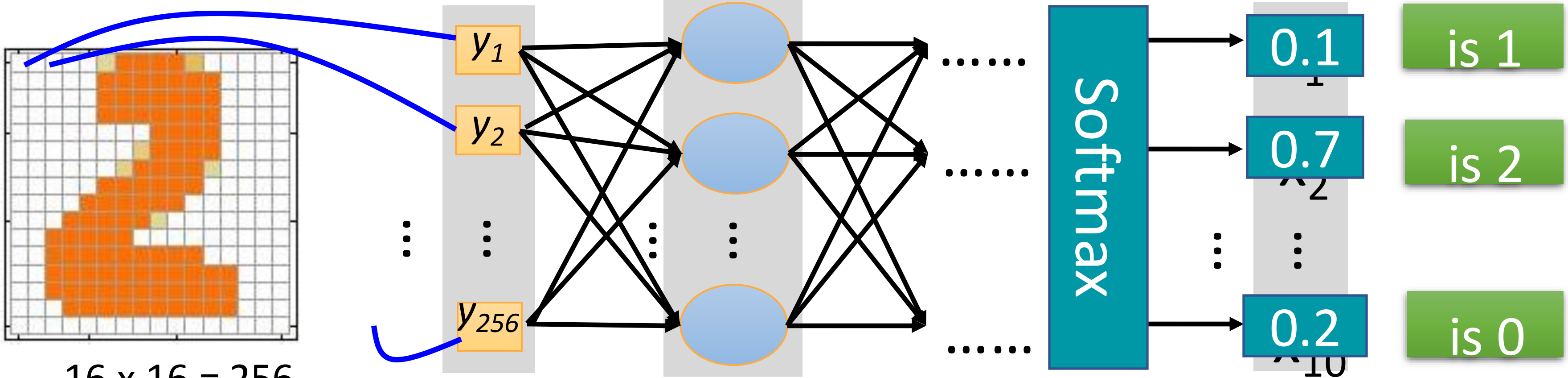
- Softmax layer as the output layer

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



How to set network parameters

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

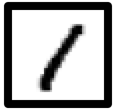


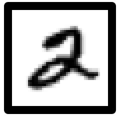
16 x 16 = 256

Ink → 1

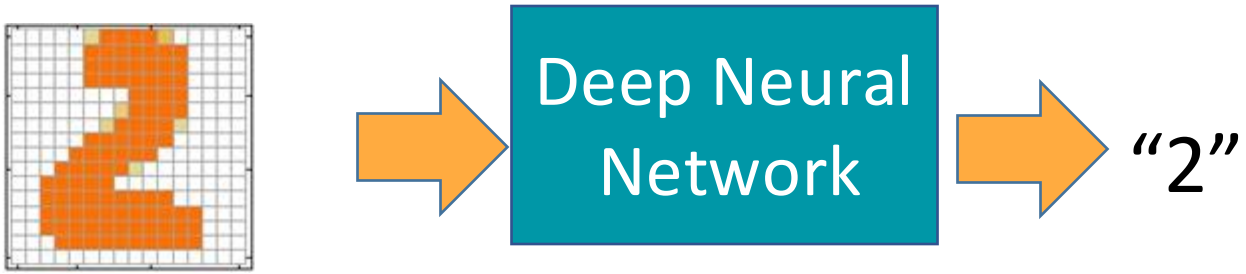
No ink → 0

Set the network parameters θ such that ......

Input:  → x_1 has the maximum value

Input:  → x_2 has the maximum value

Inverse problem



Inverse model: $X = g(Y, \theta)$

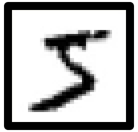
where g(.) is an unknown inverse function with unknown model parameter θ.

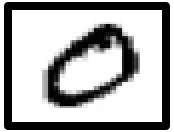
Now, the first “unknown” is known, because we assume that g(.) can be expressed as a neural network.

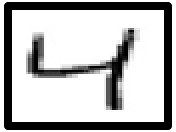
How do we address the second “unknown”?

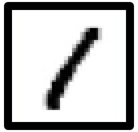
Training Data

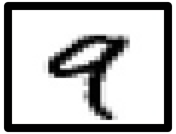
- Preparing training data: images and their labels

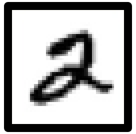

“5”

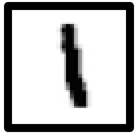

“0”

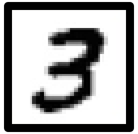

“4”


“1”


“9”


“2”

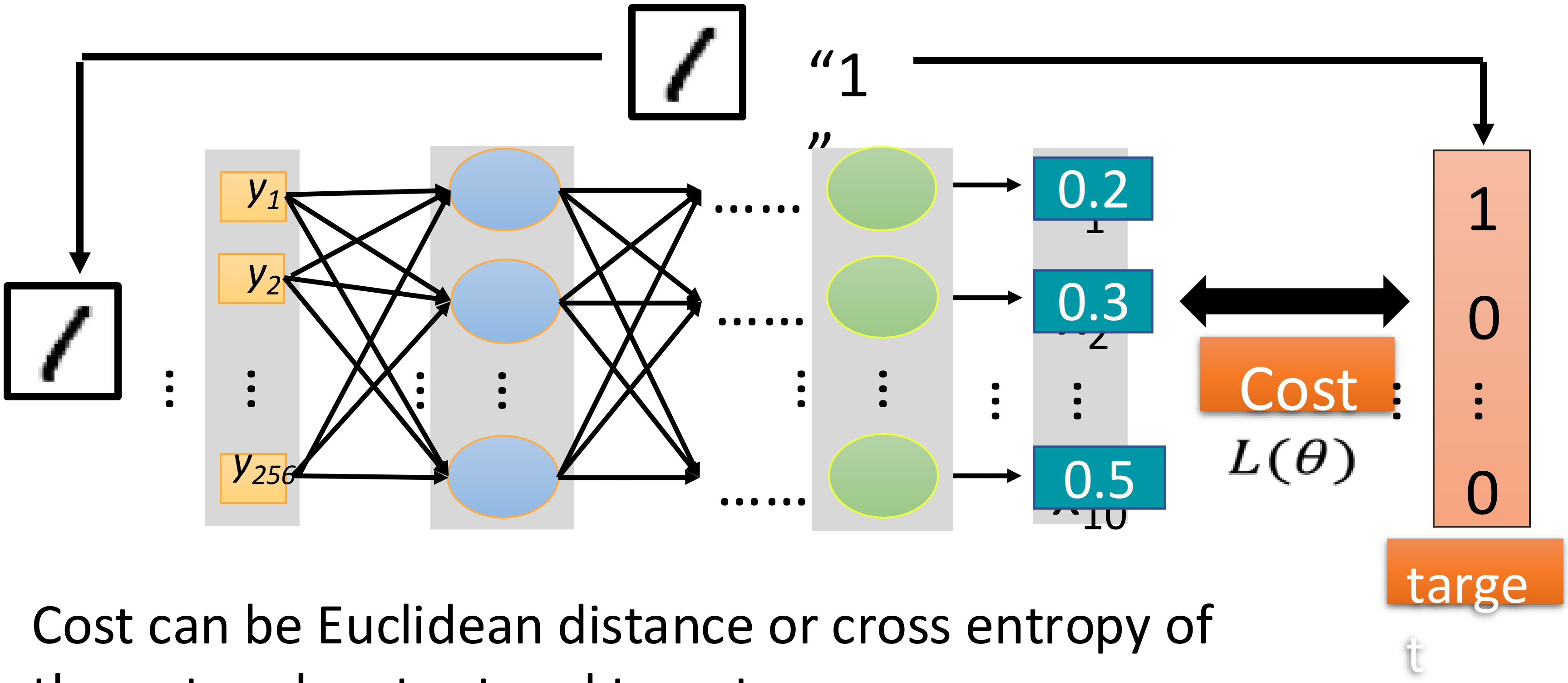

“1”


“3”

Using the training data to find the network parameters.

Cost

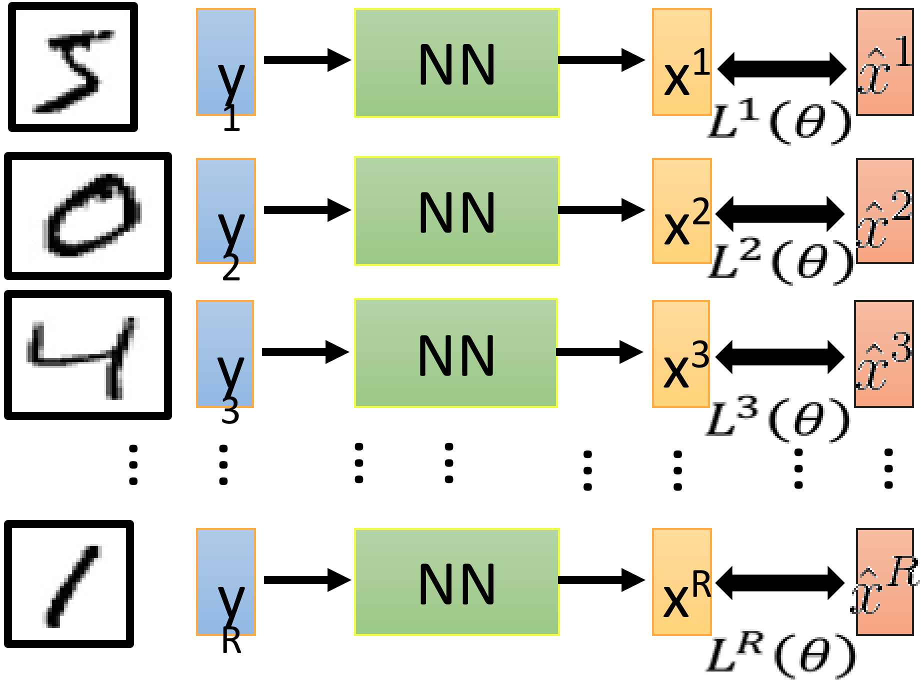
Given a set of network parameters θ, each example has a cost value.



Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

For all training data …



Total Cost:

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

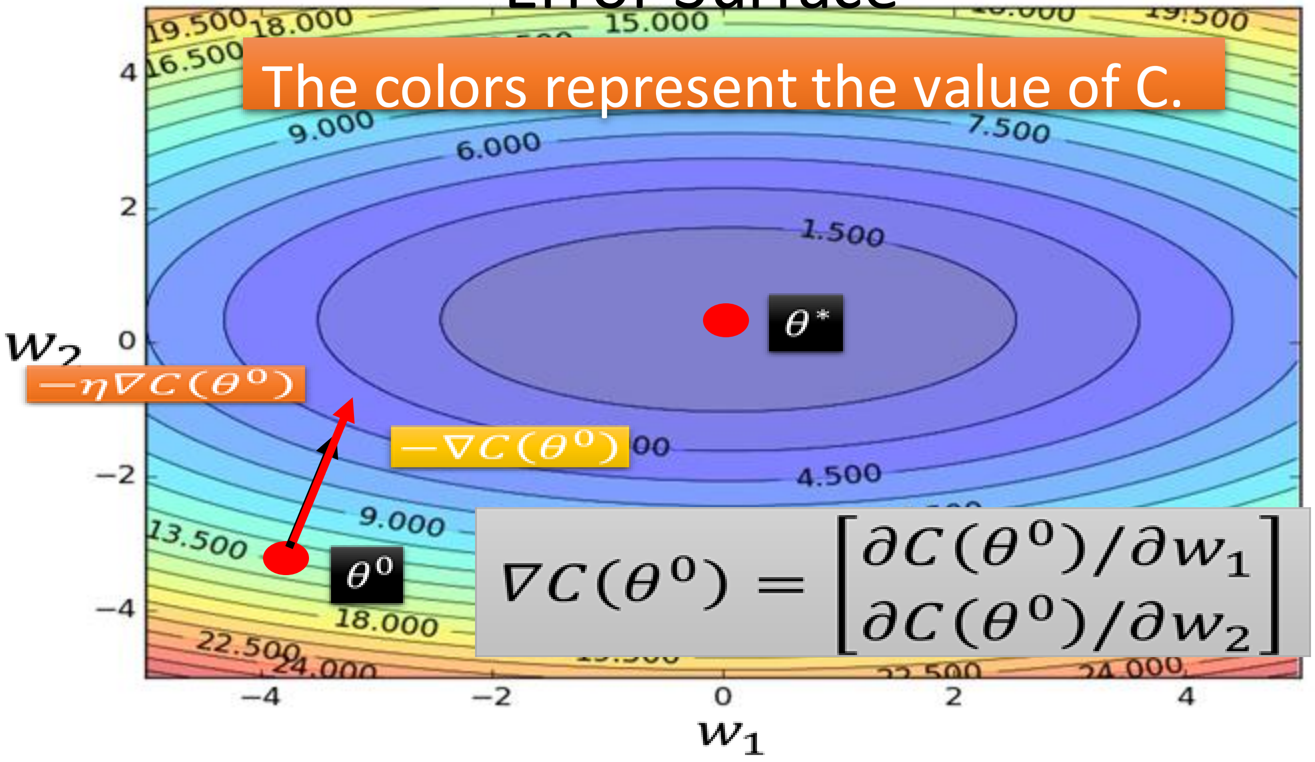
How bad the network parameters θ is on this task

Find the network parameters θ^* that minimize this value

Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network. $\theta = \{w_1, w_2\}$

Error Surface



$$\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta^0)/\partial w_1 \\ \partial C(\theta^0)/\partial w_2 \end{bmatrix}$$

Randomly pick a starting point θ^0

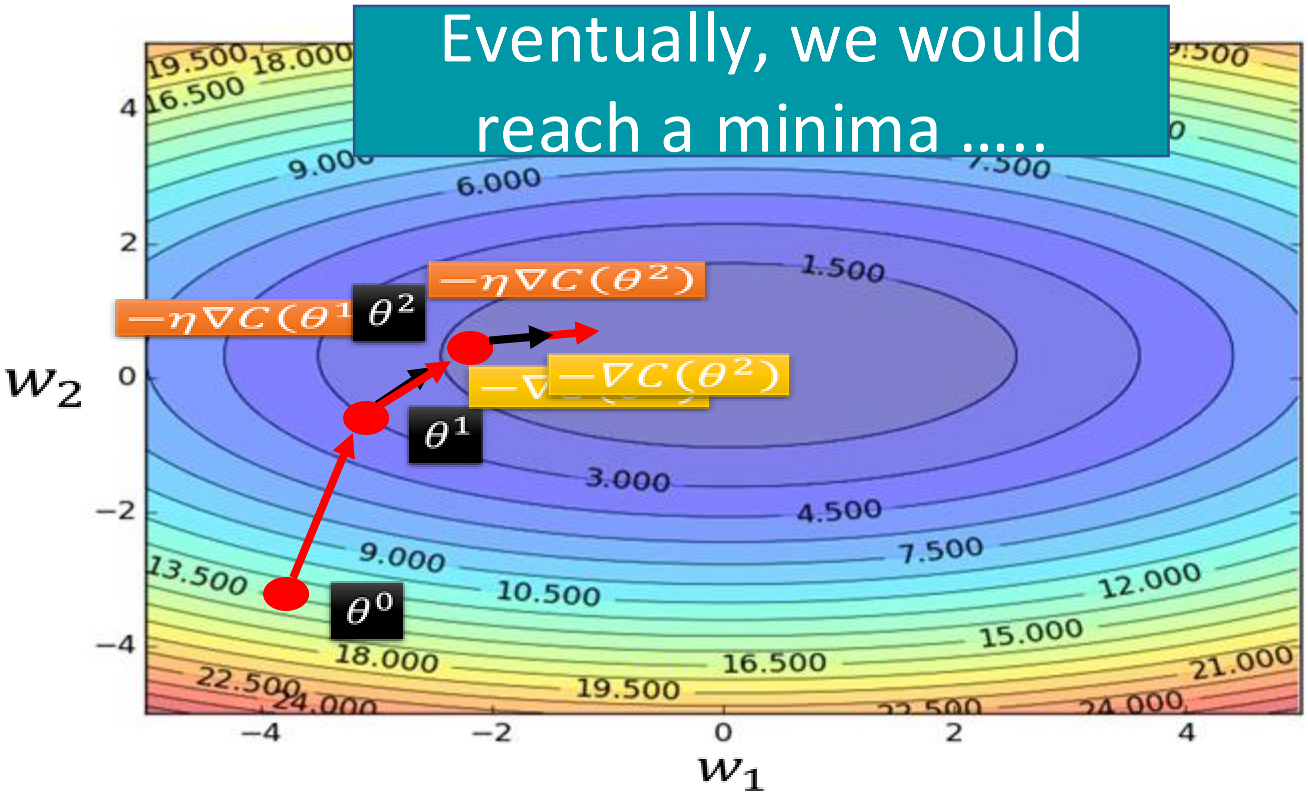
Compute the negative gradient at θ^0

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

Times the learning rate η

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

Gradient Descent



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

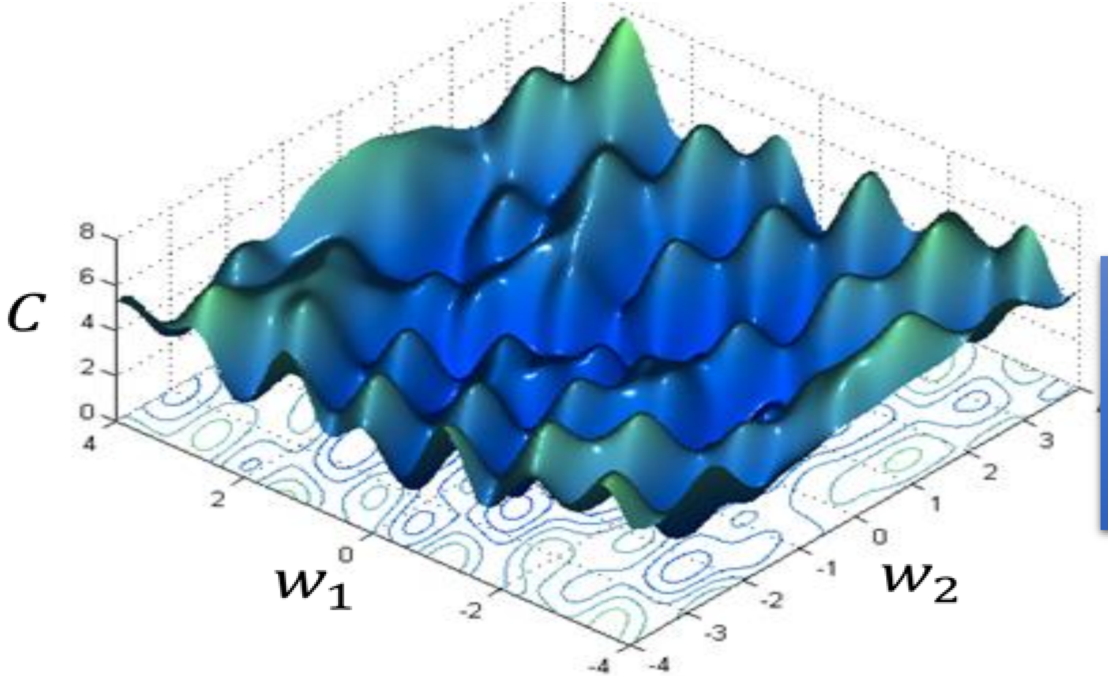
$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

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

Local Minima

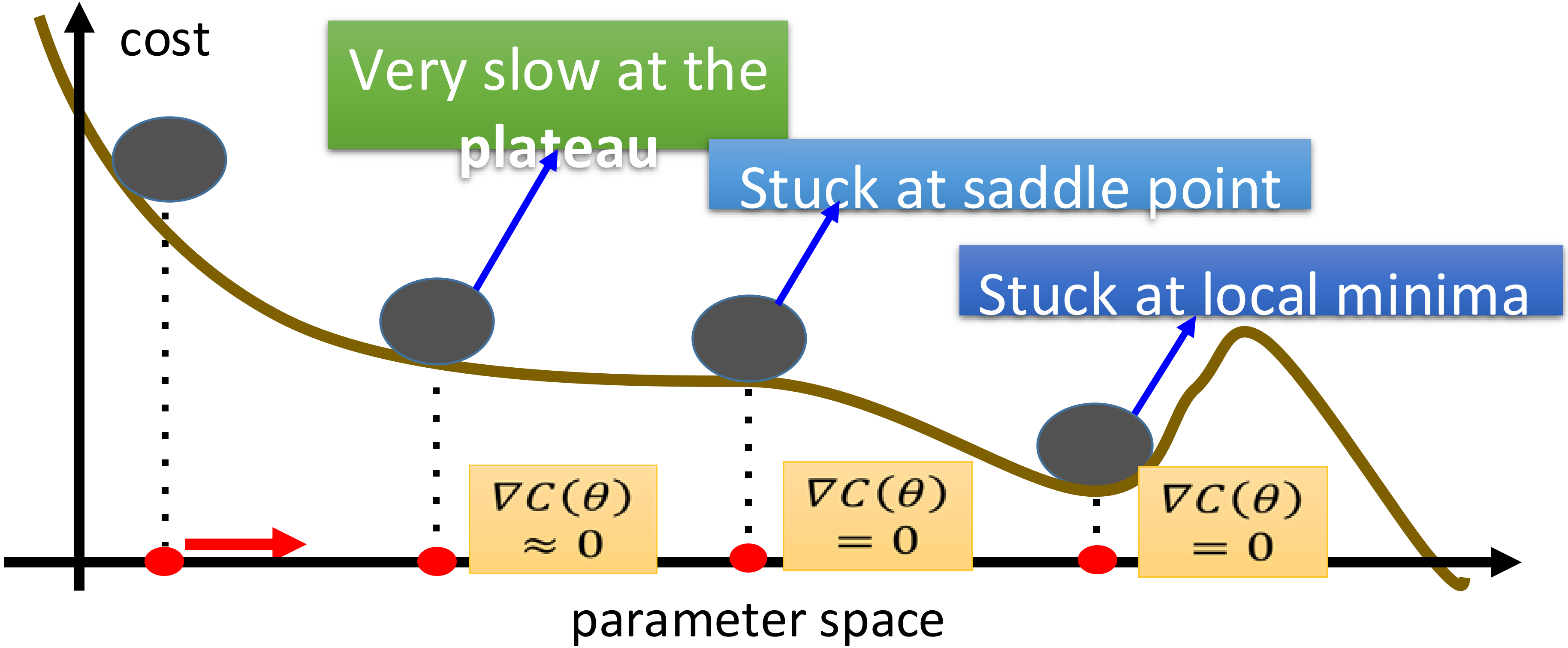
- Gradient descent never guarantee global minima



Different initial point θ^0

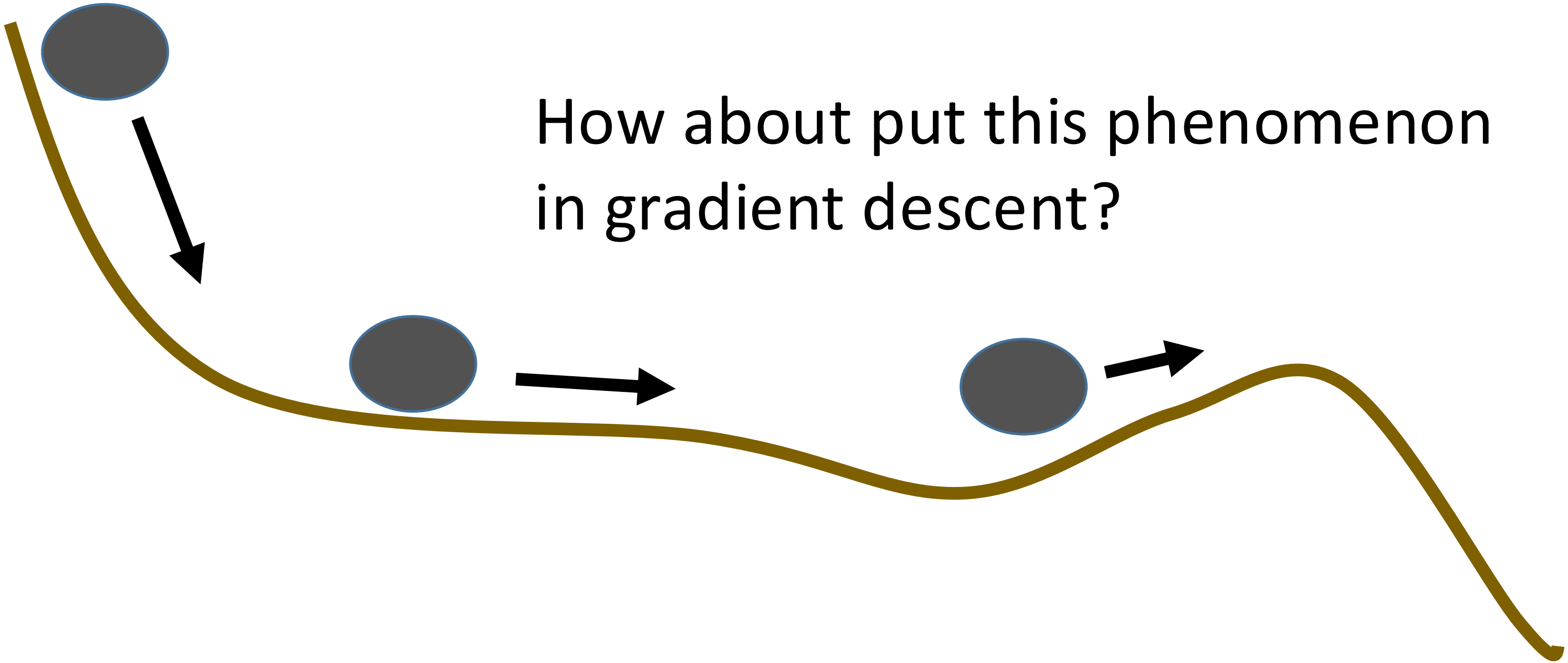
Reach different minima, so different results

Besides local minima ……



In physical world ……

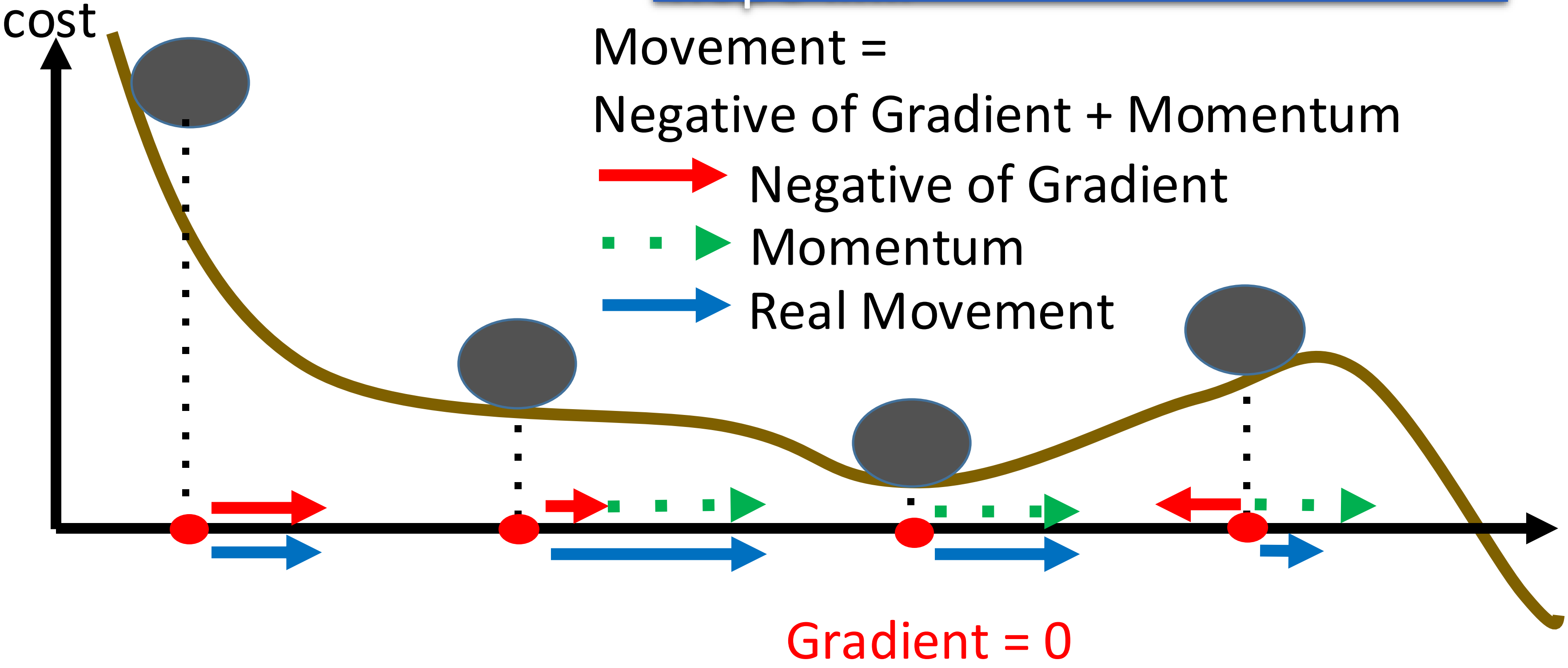
- Momentum



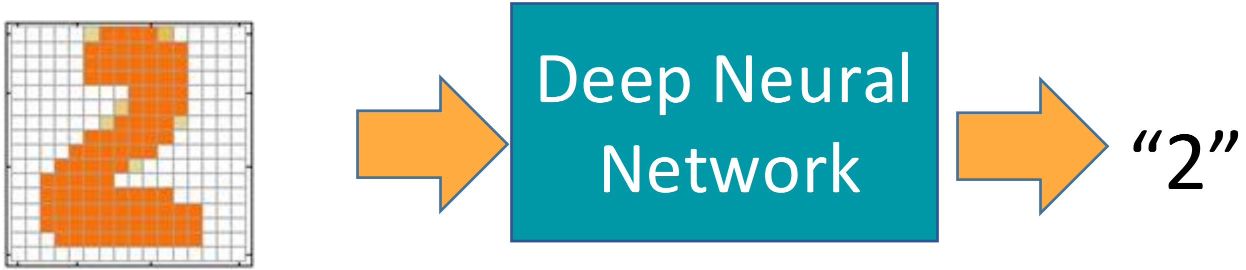
How about put this phenomenon in gradient descent?

Momentum

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



Do we really need a global optimum?



Inverse model: $X = g(Y, \theta)$

where g(.) is an unknown inverse function with unknown model parameter θ.

True inverse function $X = t(Y)=f^{-1}(Y)$ unknown;

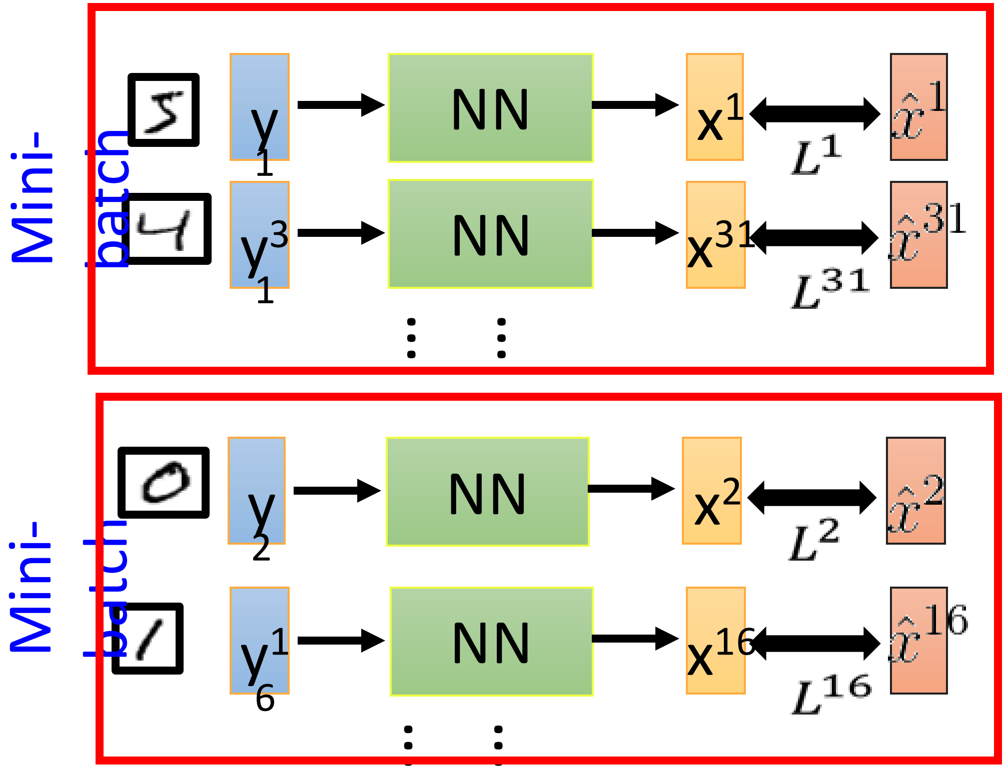
Use approximated inverse function $X = g(Y)$, where the form of g(.) is expressed as a neural network;

Use data pairs to fit $X = g(Y, \theta)$, in order to estimate θ;

The “goodness” of θ depends on the “goodness” of g(.):

---- if g(.) is very close to t(.), then we probably want a global optimum according to g(.) standard is useful;
---- if g(.) is strongly biased, and very different from t(.), then the standard for estimating θ is also biased;

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

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

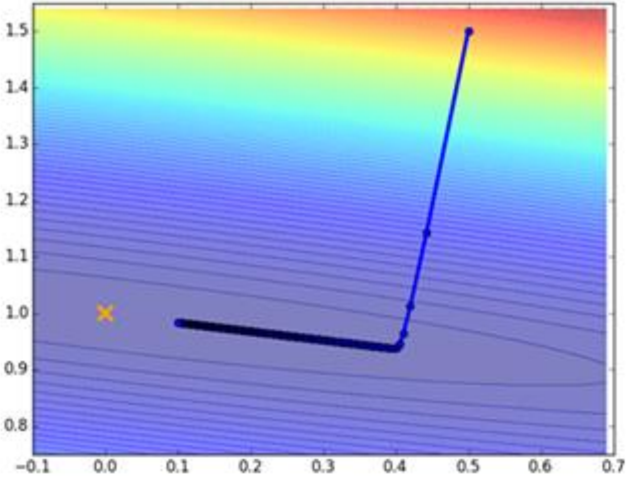
  $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

  ⋮

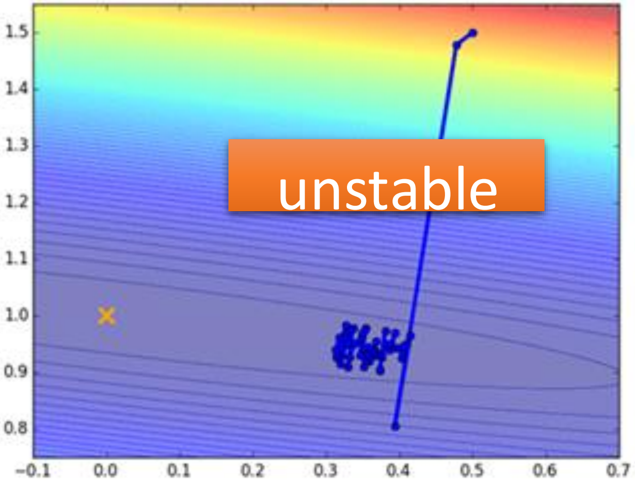
C is different each time when we update parameters!

Mini-batch

Original Gradient Descent



With Mini-batch

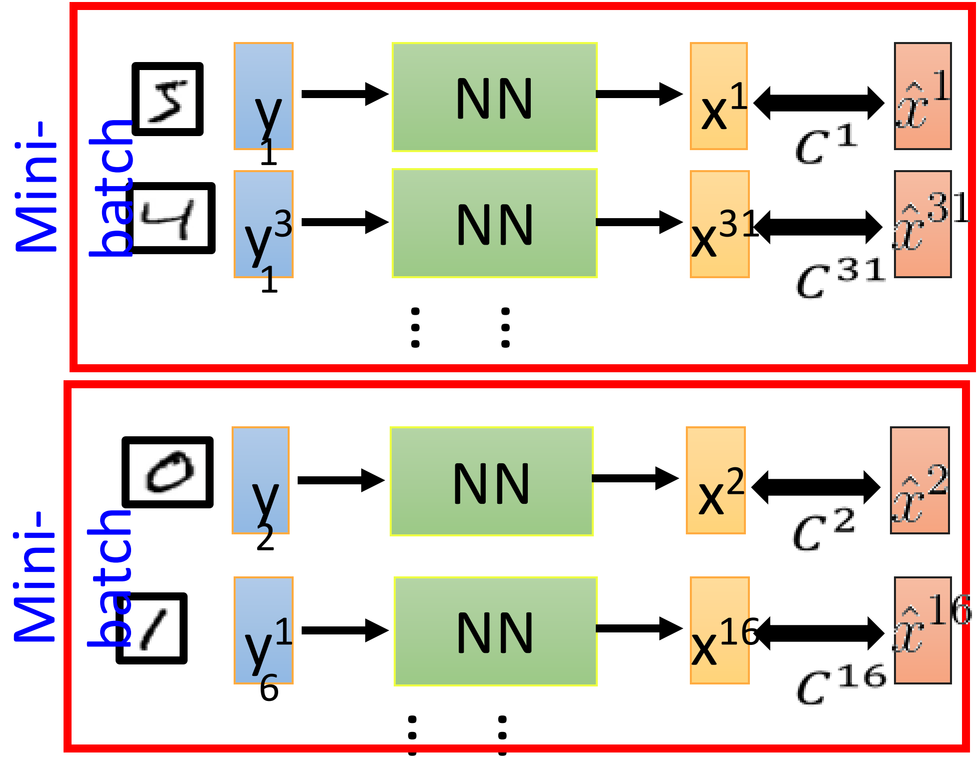


The colors represent the total C on all training data.

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

True inverse function vs. approximated inverse function

Forward model:

$Y = f(X)$

(1) Y: received radiation by the sensor

(2) X: variables that you want to know, e.g., class labels, chlorophyll content in leaves, leaf area index/density;

True inverse function:

$X = t(Y) = f^{-1}(Y)$

where $f^{-1}(.)$ is difficult to get and

the form of t(.) is usually unknown;

Approximated inverse function:

$X = g(Y)$

Note that g(.) is only an approximation to the true inverse function t(.)

Appropriate fitting: when the complexity of g(.) is close to t(.);

Overfitting: when the complexity of g(.) is larger than t(.);

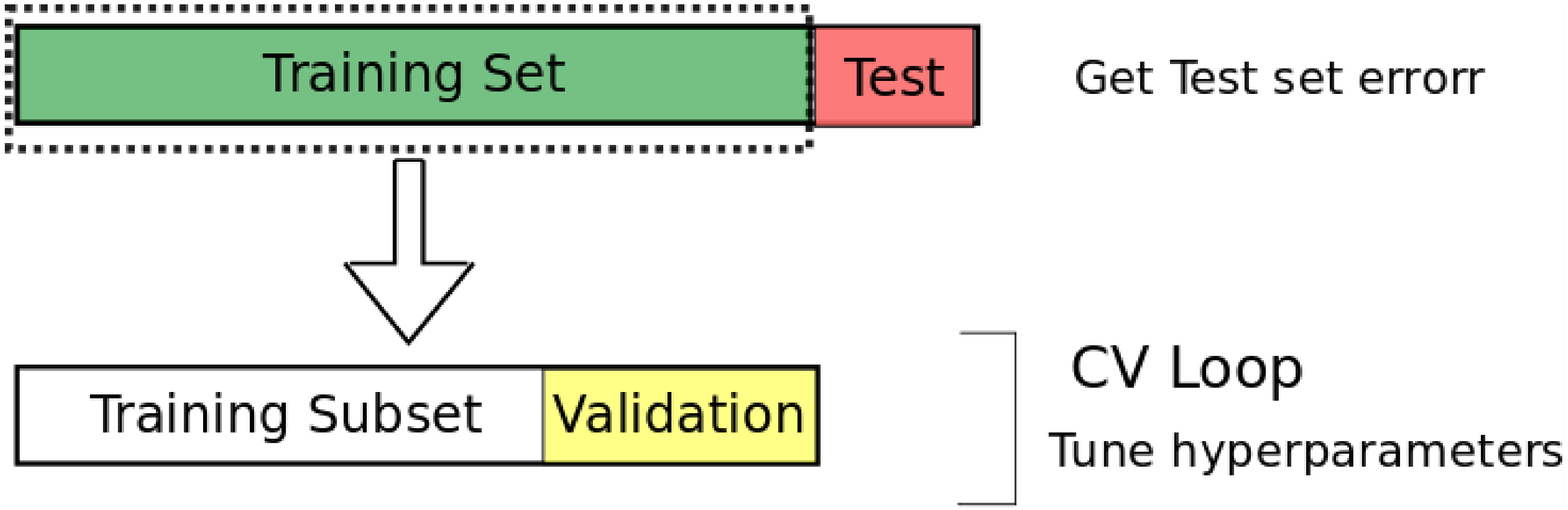
Underfitting: when the complexity of g(.) is smaller than t(.);

Based on $\{(X_j,Y_j) \mid j=1,2,...,n\}$, we build the following objective function:

$J(\theta) = \sum||X_i-g(Y_i)||$

$\theta = \min J(\theta)$

How do you choose a good g(.)?



Try different models, g1(.), g2(.), …, gn(2), and select the one that with highest accuracy on the validation set.

Overfitting vs. Underfitting

Overfitting:

---- ML model is so flexible and complex that it accommodates the noise effect in the training data and treats it as signal, and the learnt noise characteristics cannot generalize well to the test data;

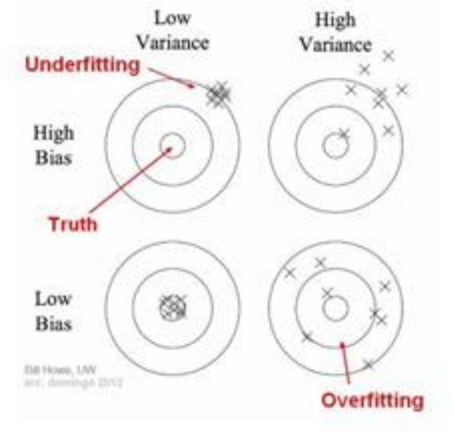
---- very high training accuracy but low validation/test accuracy; small Bias but big variation in prediction;

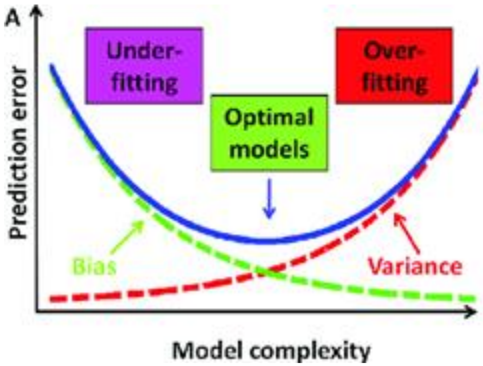
Underfitting:

---- ML model is so simple and rigid that it does not has enough capacity to accommodate signal in the training data, and the learnt biased/parcial information cannot generalize well to the test data;

---- low training accuracy & low validation/test accuracy; big Bias but small variation in prediction;

Trade-off between Bias and Variance





True inverse function: $X = t(Y) = f^{-1}(Y)$ unknow;

Approximated inverse function: $X = g(Y)$ is only an approximation to t(.);

Appropriate fitting: when the complexity of g(.) is close to t(.);

Overfitting: when the complexity of g(.) is larger than t(.);

Underfitting: when the complexity of g(.) is smaller than t(.);

Bias is the difference between the average prediction of our model and the true value which we are trying to predict.

Variance is the variability of model prediction.

<u>Why increasing model complexity lead to small bias in prediction?</u>

---- increasing model complexity -> g(.) to be universal approximator -> stronger accommodating/modeling capability to learn the genuine nonlinear relationship between X and Y in X = t(Y) -> less bias;

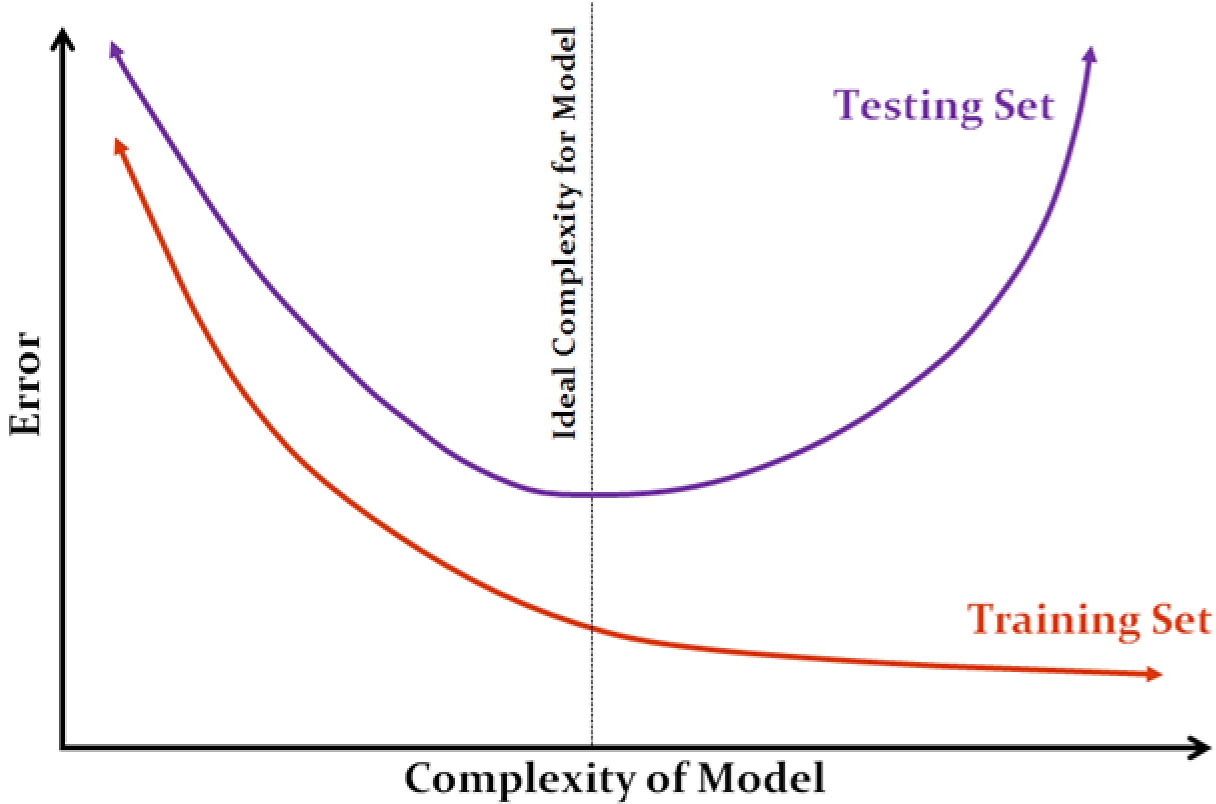
<u>Why increasing model complexity lead to larger variance in prediction?</u>

---- increasing model complexity -> g(.) to be universal approximator -> stronger accommodating/modeling capability to learn both the genuine nonlinear relationship between X and Y and irrelevant factors (i.e., noise and even errors in the data) -> larger variance;

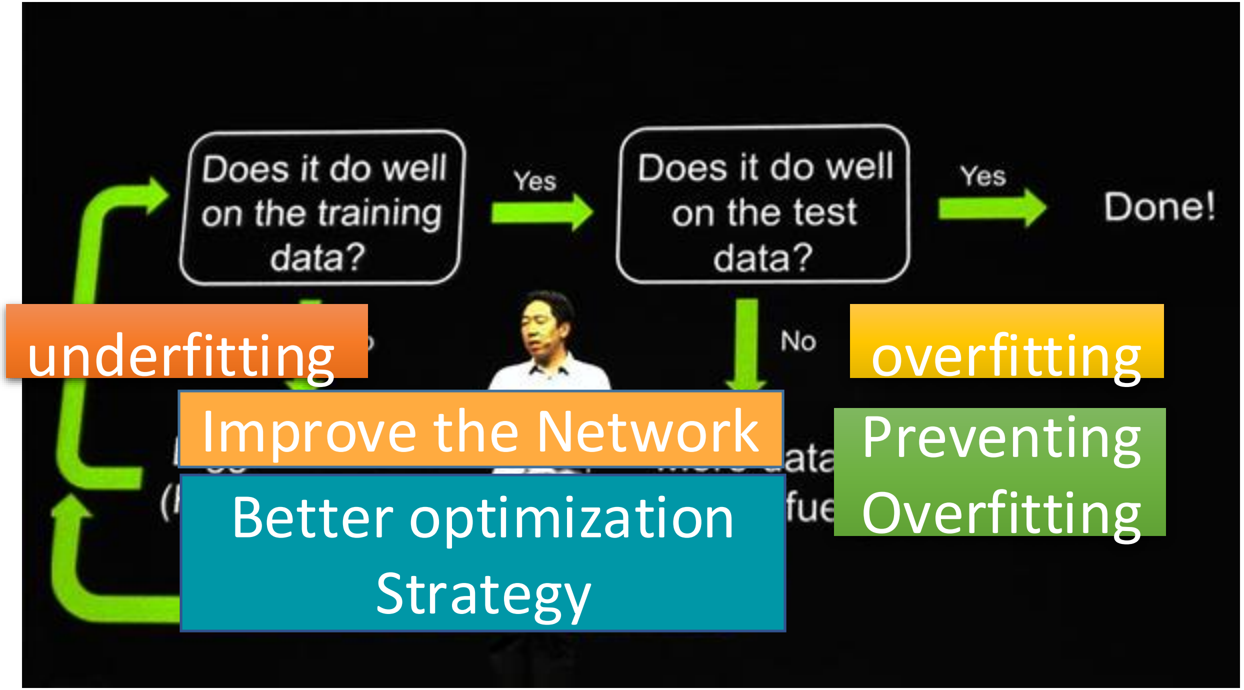
<u>Why decreasing model complexity lead to larger bias in prediction?</u>

<u>Why increasing model complexity lead to smaller variance in prediction?</u>

Training error vs. test error as model complexity changes

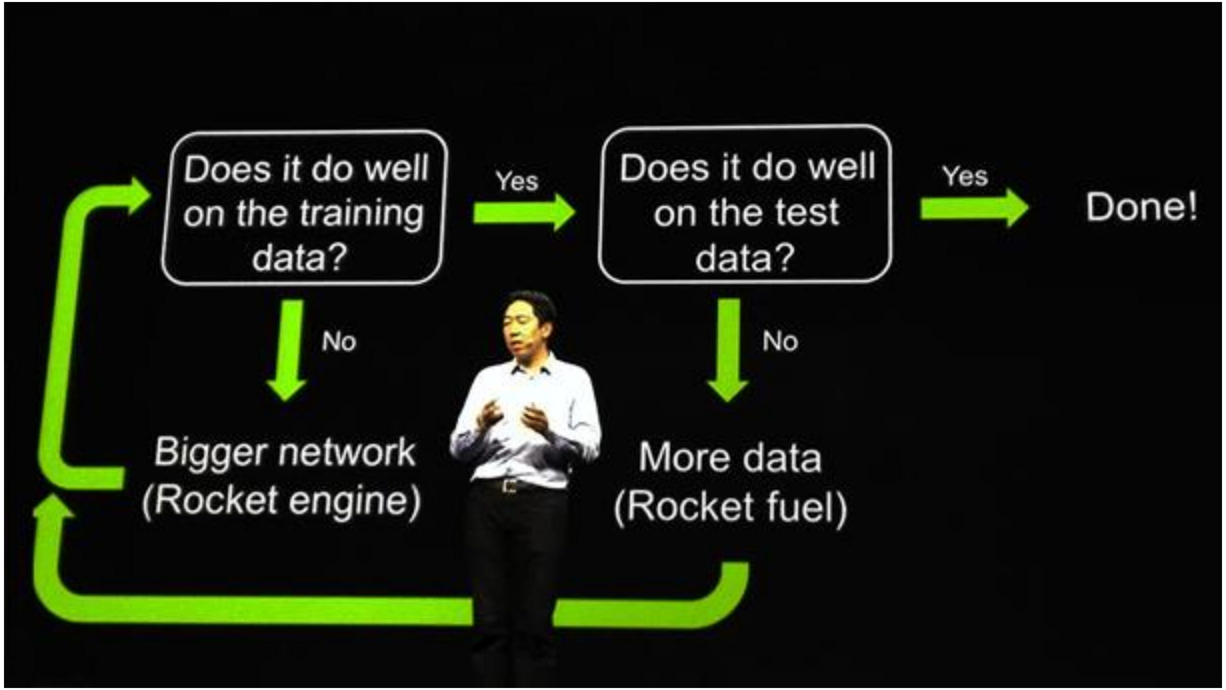


Recipe for Learning



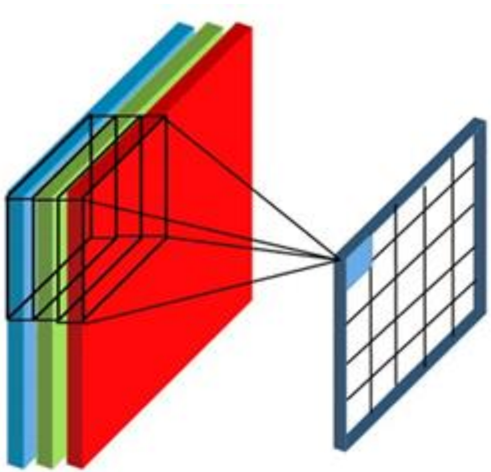
http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/

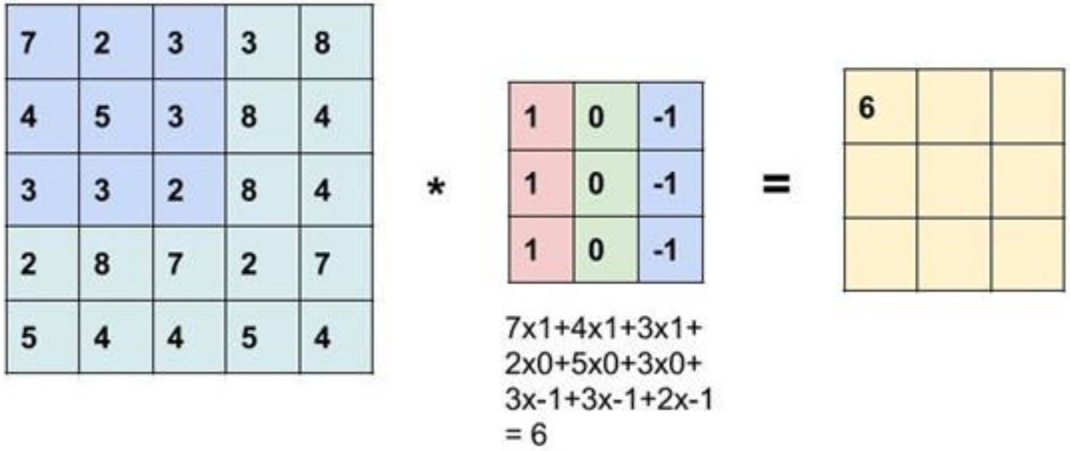
Recipe for Learning



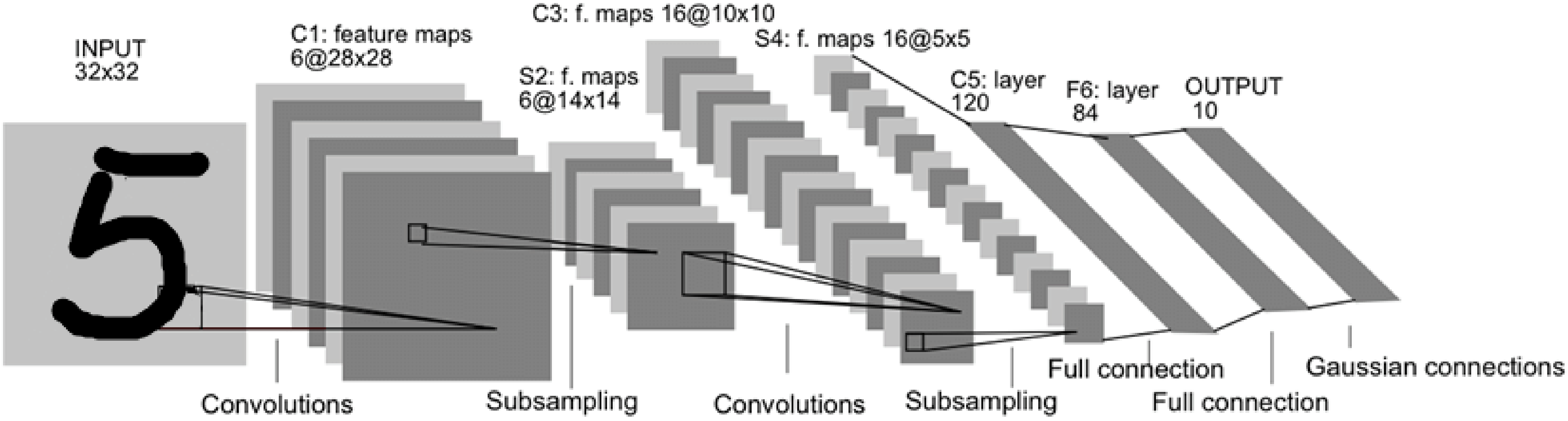
http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/

Convolutional neural network (CNN)



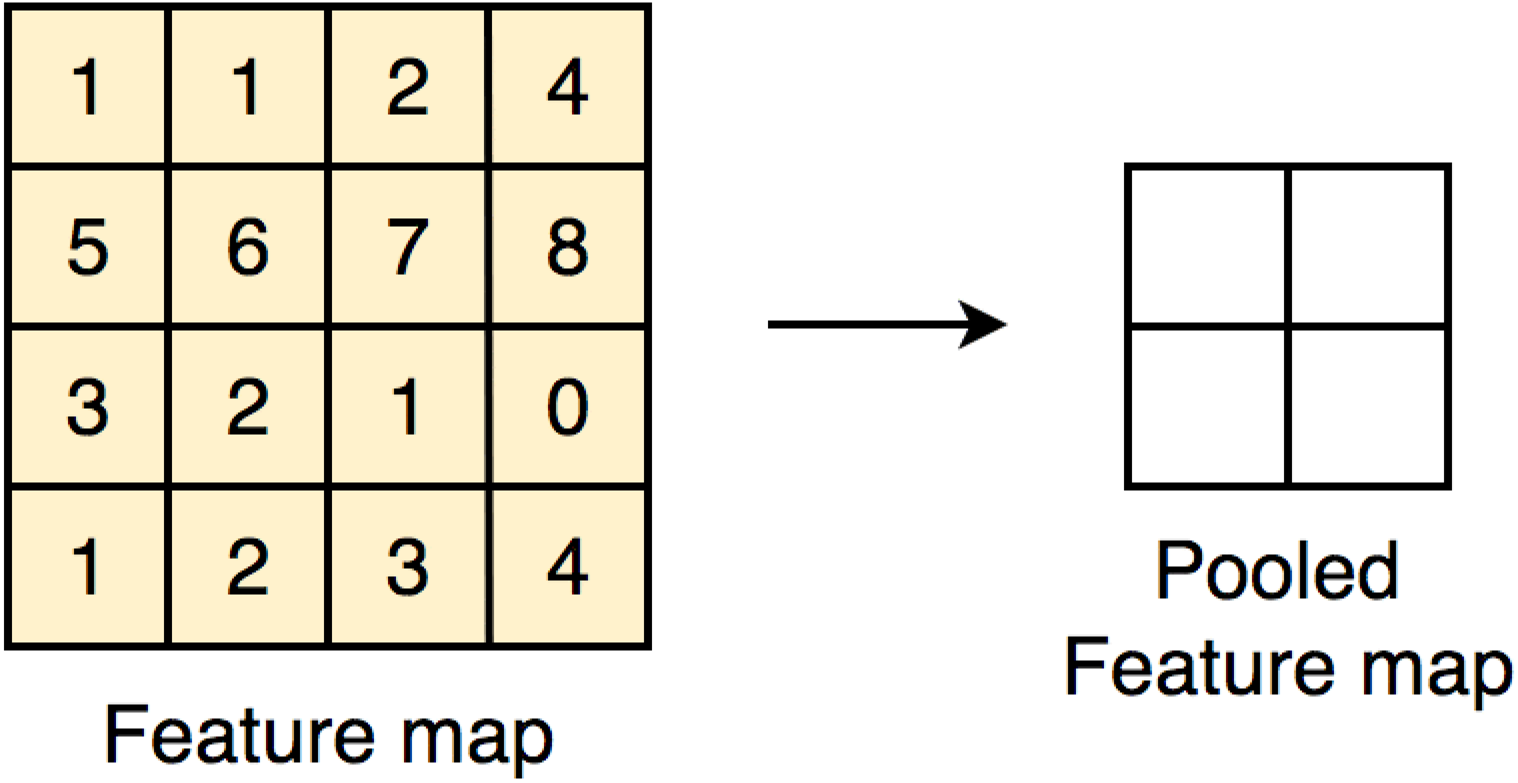


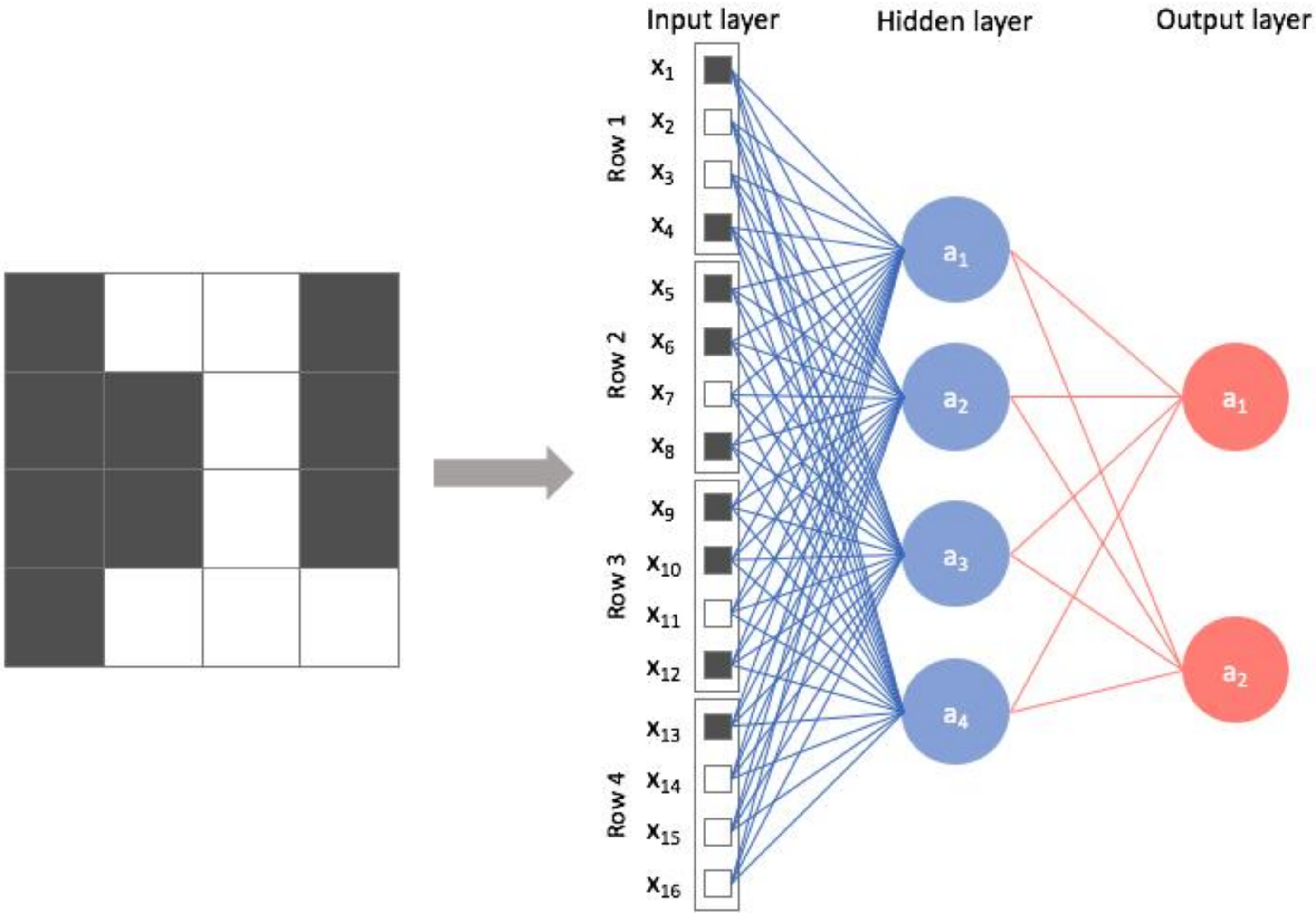
How does CNN work in digit recognition?



An early (Le-Net5) Convolutional Neural Network design, LeNet-5, used for recognition of digits

Max pooling layer





Input layer
Hidden layer
Output layer
Row 1
Row 2
Row 3
Row 4
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
a1
a2
a3
a4
a1
a2

DL frameworks

Caffe2

PyTorch

TensorFlow