Introduction to Deep Learning

Jianwen Zhang

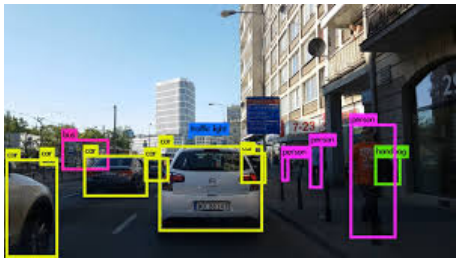
August 18, 2018

Outline

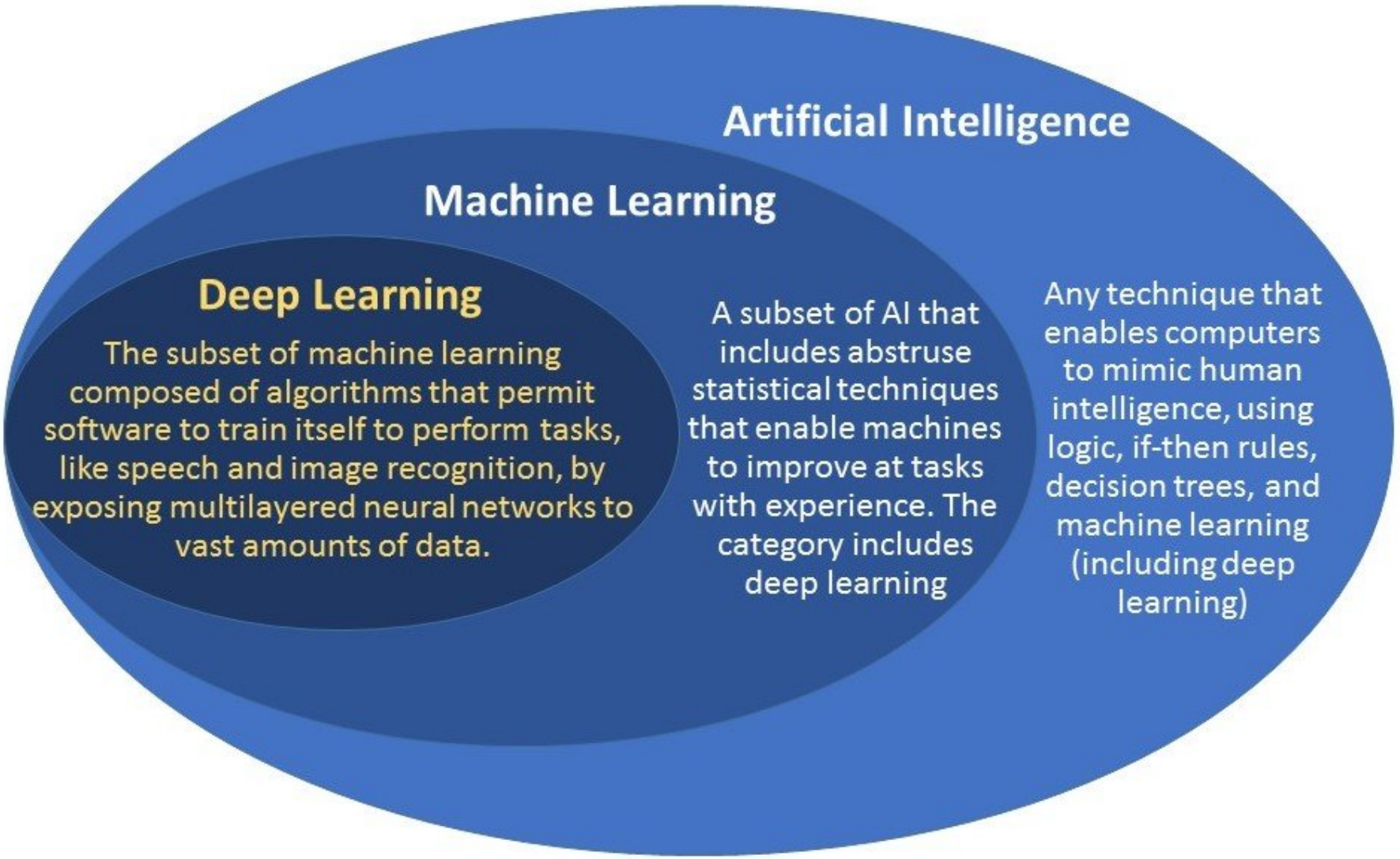
- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others

Outline

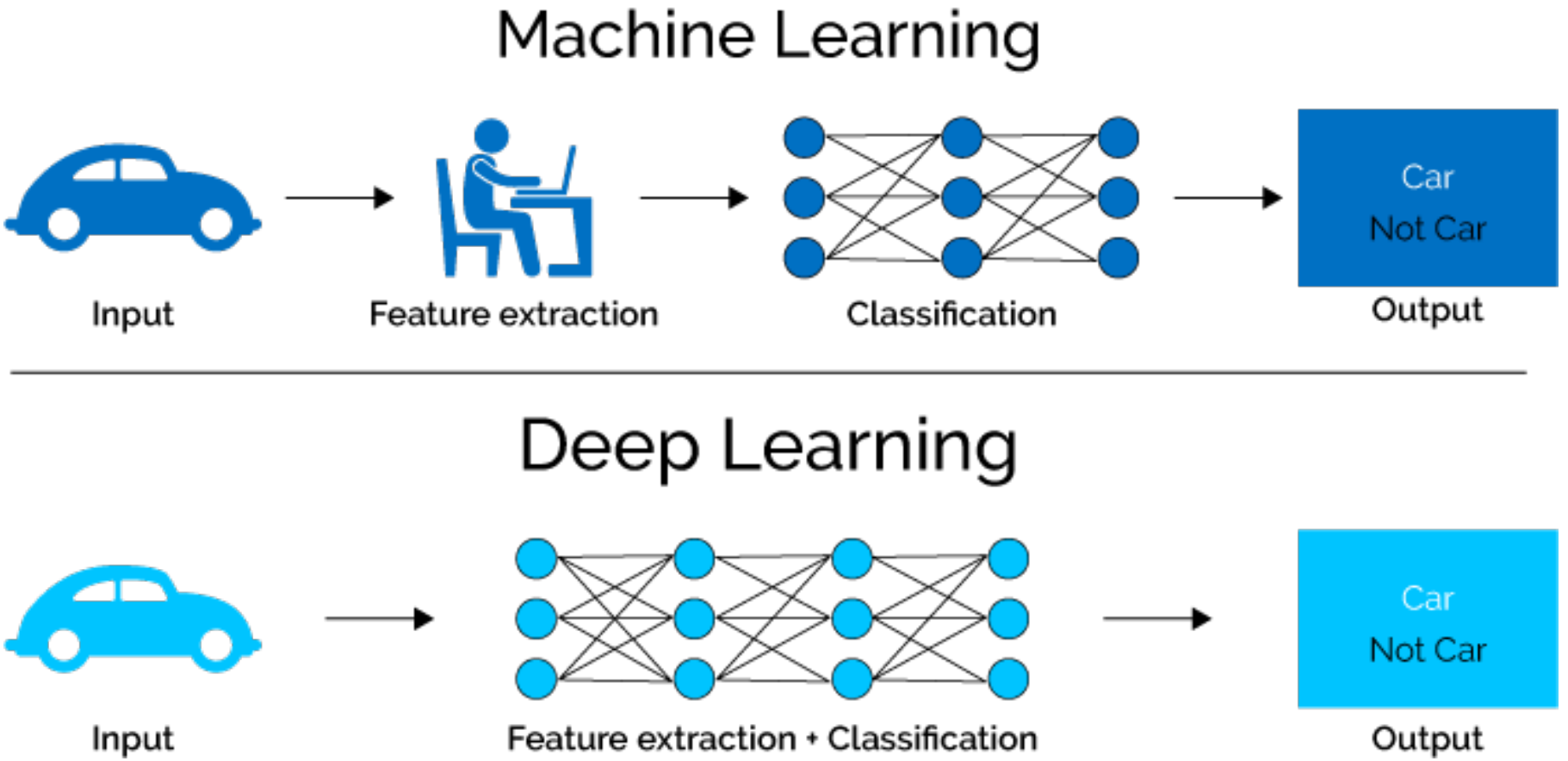
Introduction – What can AI do?



Introduction – What is AI



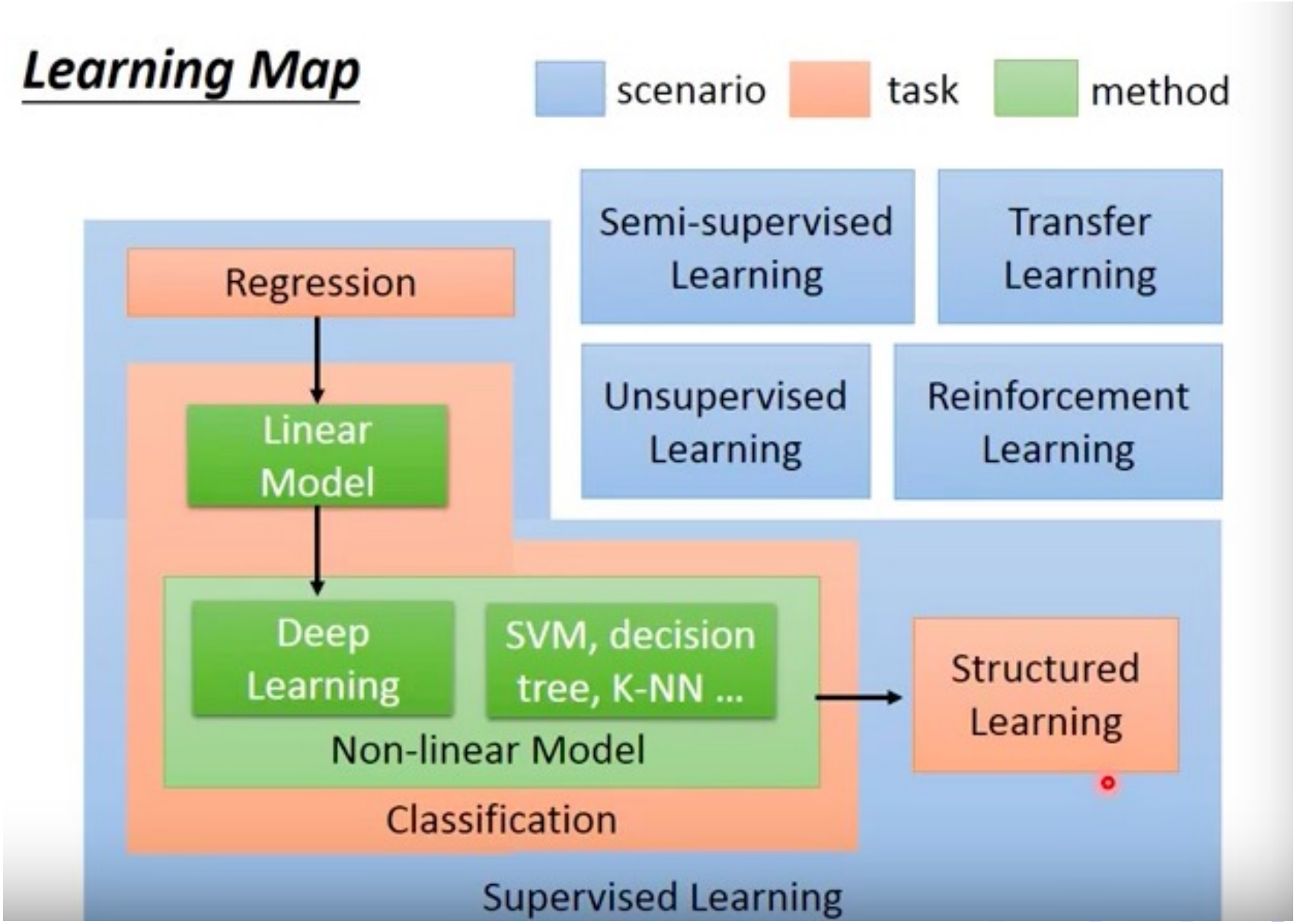
Machine Learning vs Deep Learning



Machine Learning

- Definition given by T. Mitchell: A computer program is said to learn from experience E with respect to some task T and some performance measure P, is its performance on T, as measured by P, improves with experience E.
- Supervised learning: "right answers" given, labeled data, including regression and classification
- Unsupervised learning: without any labeled data, eg. clustering

Learning Map

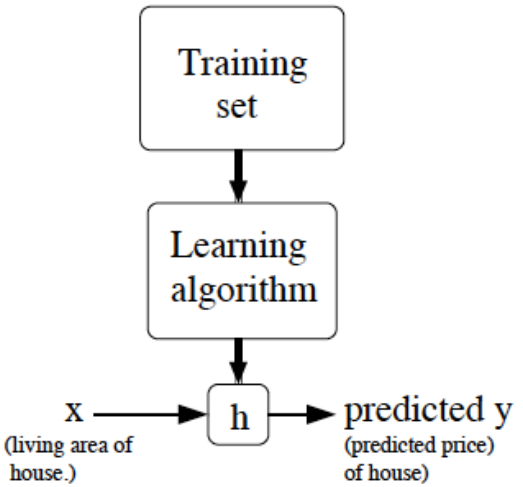


Linear Regression

- notations: m: no. of training data, x: input or features, y: output, (x, y): one training example, $(x^{(i)}, y^{(i)})$: ith training example.
- Formally definition: Given a training set, our goal is to learn a function $h : \mathcal{X} \to \mathcal{Y}$ so that $h(x)$ is a "good" predictor for the corresponding value of y.
- linear hypothesis: $h_w(x) = w_0 + w_1 x_1 + \cdots + w_n x_n = \mathbf{w}^T \mathbf{x}$
- performance measurement (cost function):

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right)^2$$

- Our goal is to find $\mathbf{w}$ so that $J(\mathbf{w})$ is small enough or to make $h_w(x)$ close to y for all training samples.



Gradient Descent

- Problem formulation: $\min_w J(\mathbf{w}), s.t. h_w(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$
- Gradient descent:

  Repeat until convergence:

  $$w_j = w_j - \alpha \sum_{i=1}^{m} (\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)} \quad \forall j$$

  where α is the learning rate
- This method looks at every example in the entire training set on every step, and is called batch gradient descent.
- Very complex with large training set!!!

Gradient Descent

- Stochastic gradient descent (SGD):

  Loop:

  for $i = 1 : m, \{w_j = w_j - \alpha(\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)} \quad \forall j\}$
- Close to the minimum much faster
- Never converge
- Mini-batch gradient descent

Probabilistic Interpretation

- Assume: $y^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + \epsilon^{(i)}$, where $\epsilon^{(i)}$ is the error term and assume $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$.
- Then,

$$p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2}{2\sigma^2}\right)$$

- The likelihood function is then given as

$$L(\mathbf{w}) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2}{2\sigma^2}\right)$$

- The least-squares regression corresponds to finding the maximum likelihood estimate of $\mathbf{w}$.

Outline

- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others
- How to Build a DNN in Keras

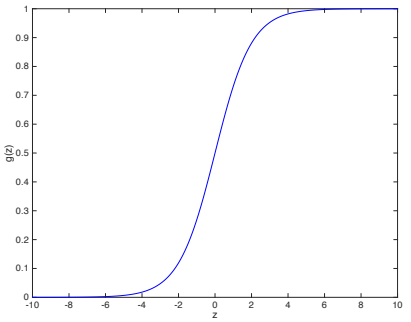
Logistic Regression

- If the output $y \in \{0, 1\}$ or other discrete values, the problem becomes a logistic regression problem or classification.
- Directly apply the above algorithm to this problem always lead to a poor performance.

- Change the hypothesis $h_w(x)$ to

$$h_w(x) = g(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x}\right)}$$

where $g(z) = \frac{1}{1+\exp(-z)}$ called sigmoid function.



Logistic Regression

- Assume

$$p(y = 1|x, w) = h_w(x)$$

$$p(y = 0|x, w) = 1 - h_w(x)$$

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

- The ML problem can be formulated as

$$\max_w \log L(\mathbf{w}) = \sum_{i=1}^{m} y^{(i)} \log h_w(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_w(x^{(i)}))$$

- Gradient descent update:

$$w_j = w_j - \alpha(h_w(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)} \quad \forall j$$

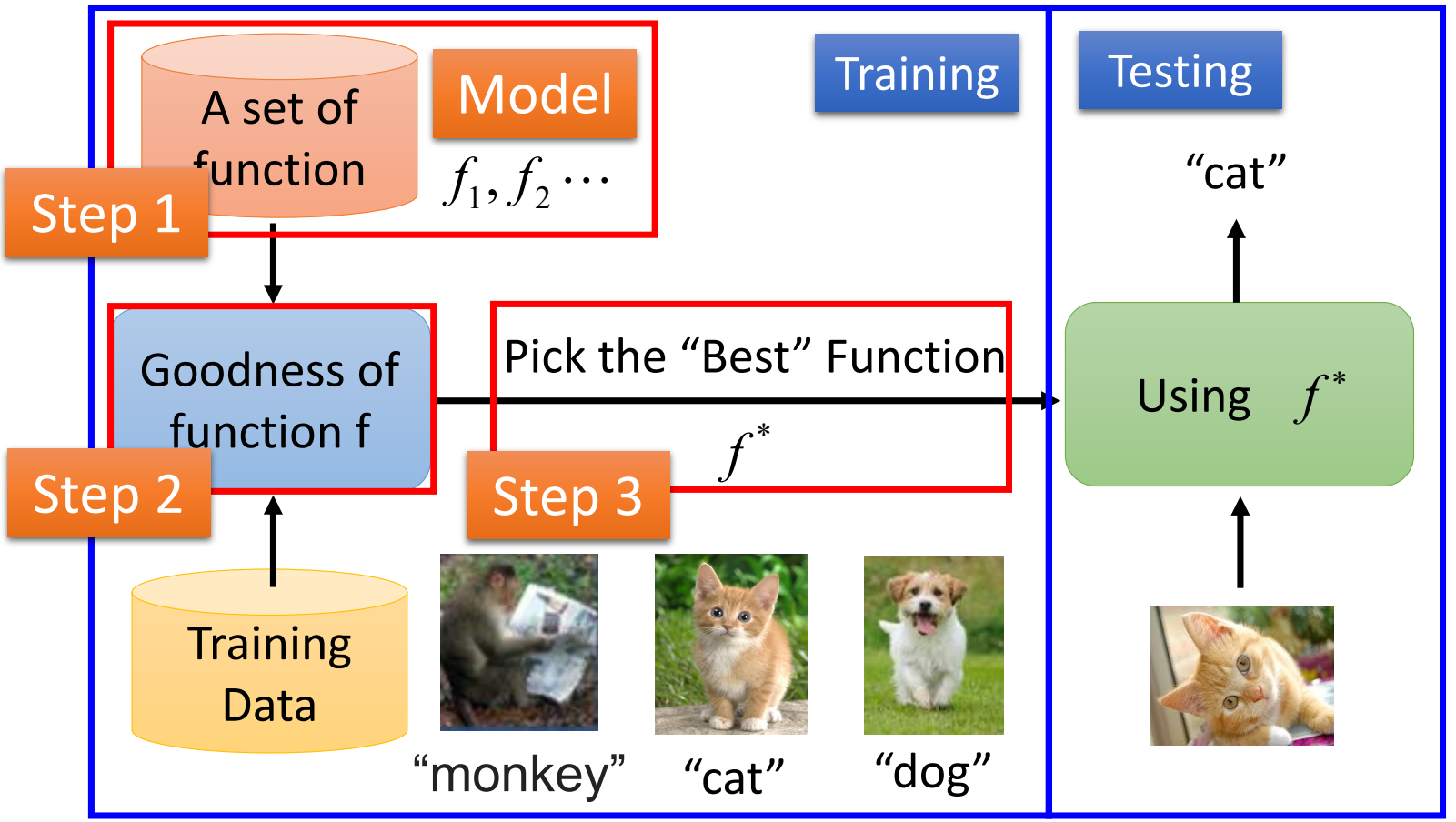
Outline

- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others

Framework

Image Recognition:

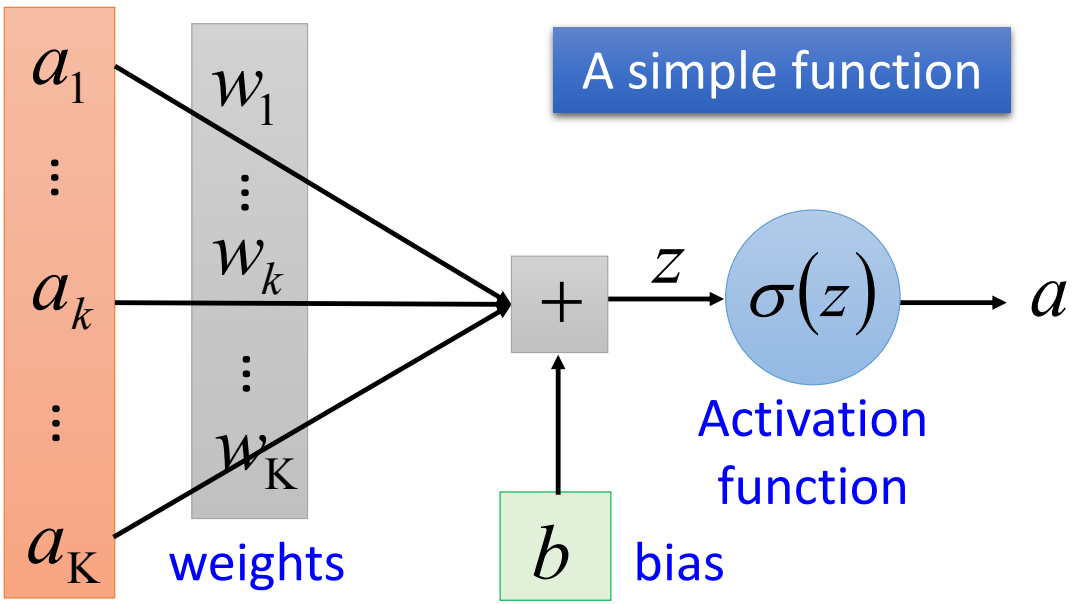
$f($  $) =$ "cat"



Neuron

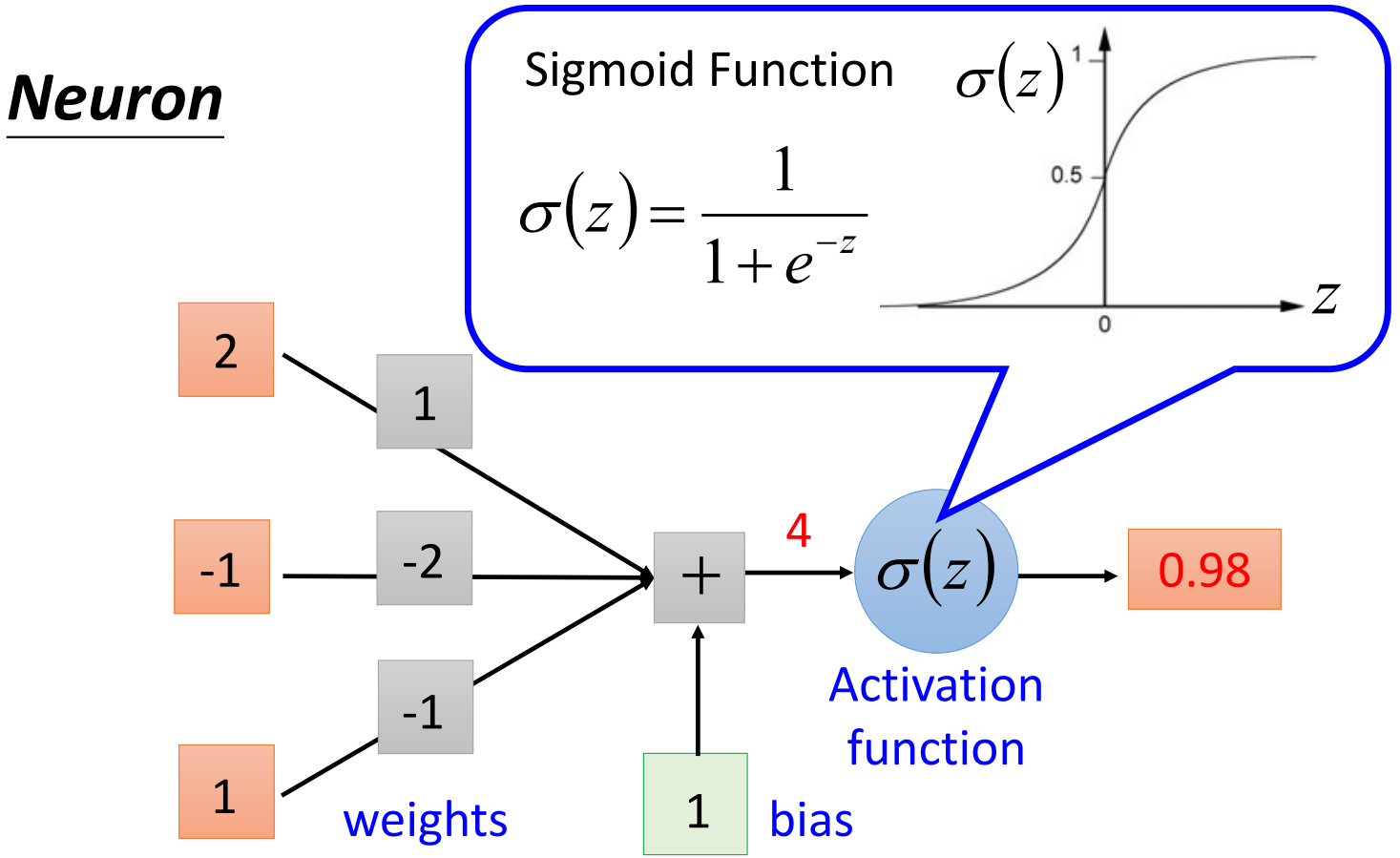
- $\{a_k\}$ are the inputs, $\{w_k\}, b$ are the network parameters.

$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$



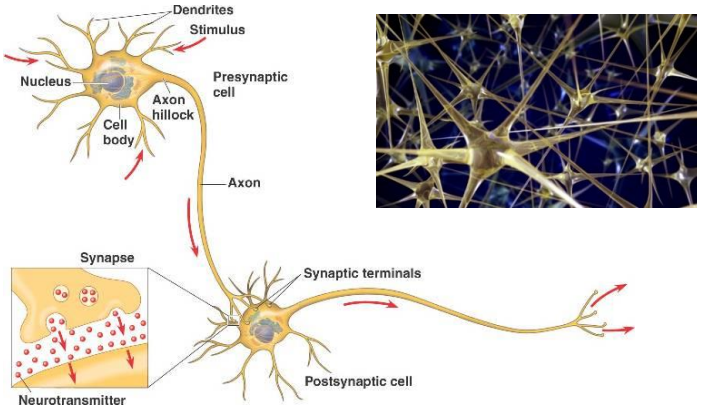
Neuron

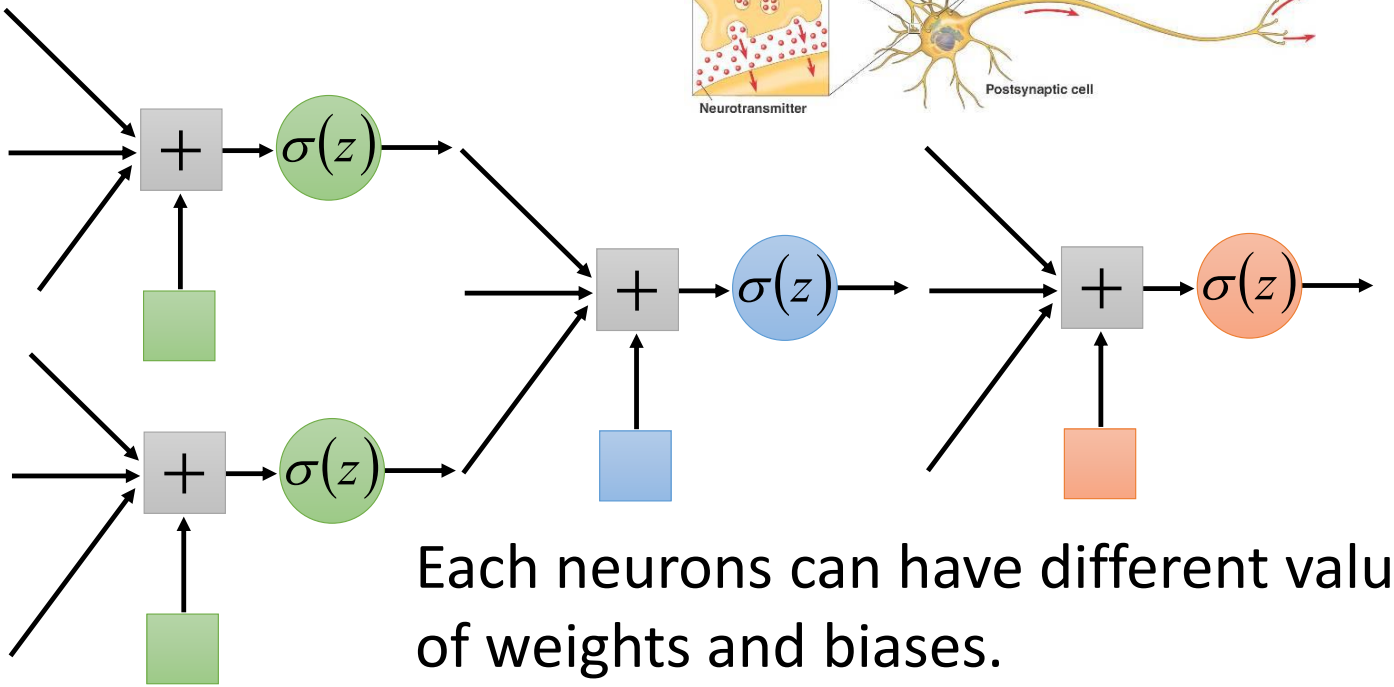
- $\{a_k\}$ are the inputs, $\{w_k\}, b$ are the network parameters.



Neural Network

Different connections leads to different network structure

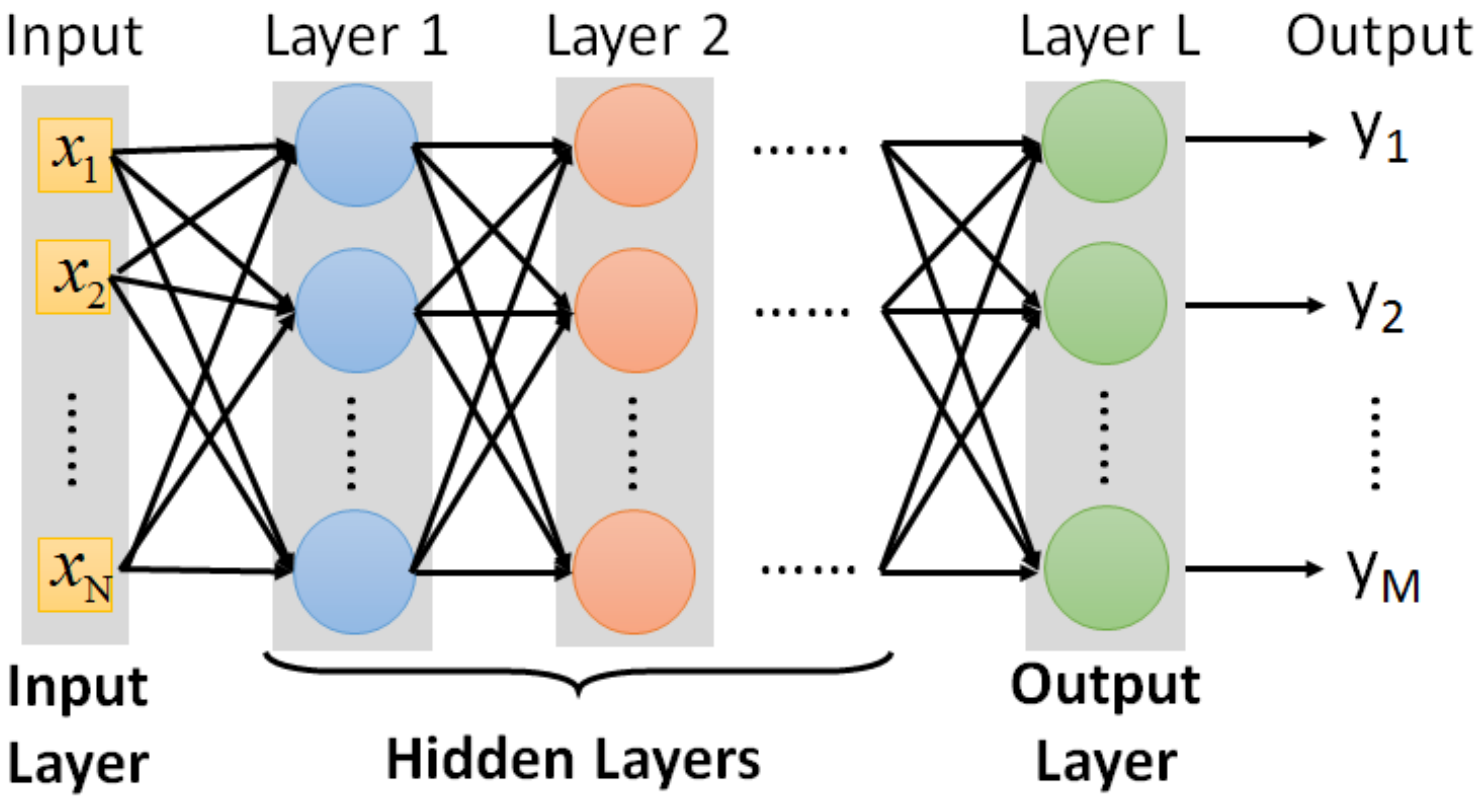




Each neurons can have different values of weights and biases.

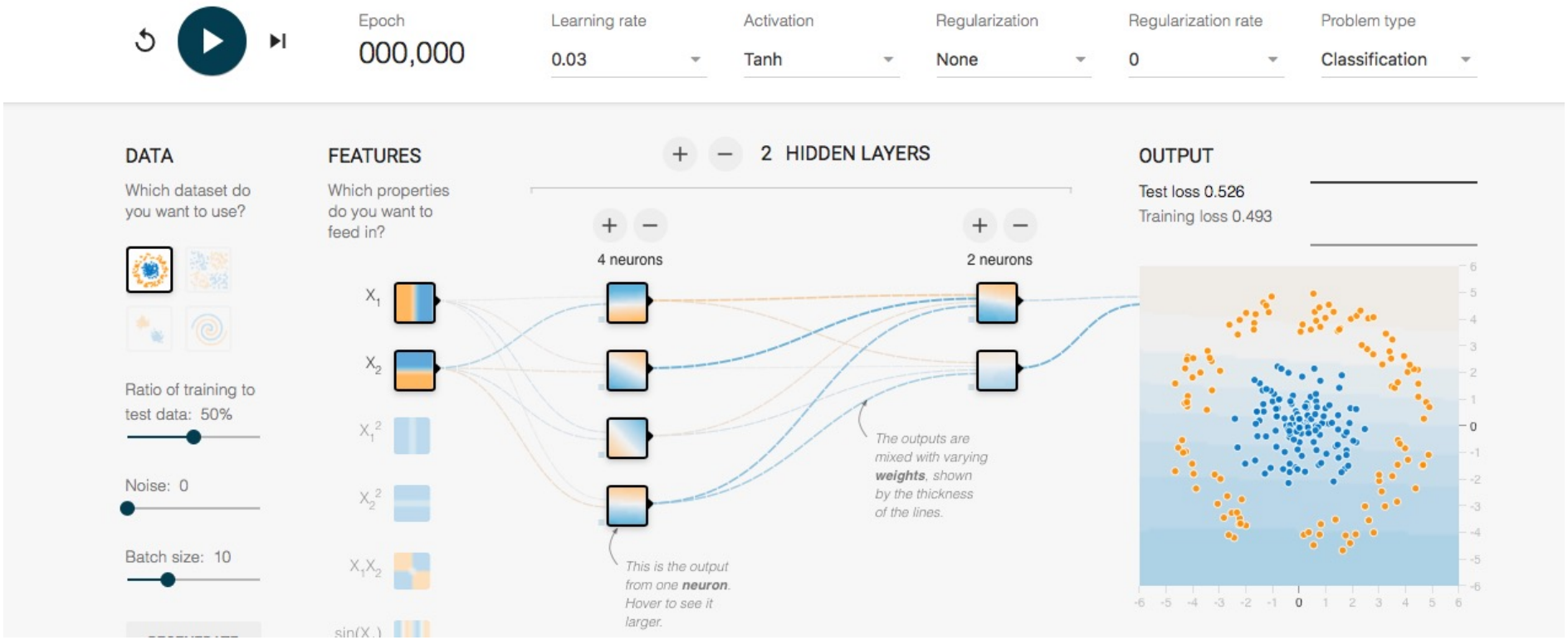
Weights and biases are network parameters θ

Fully Connect Network



A Demo

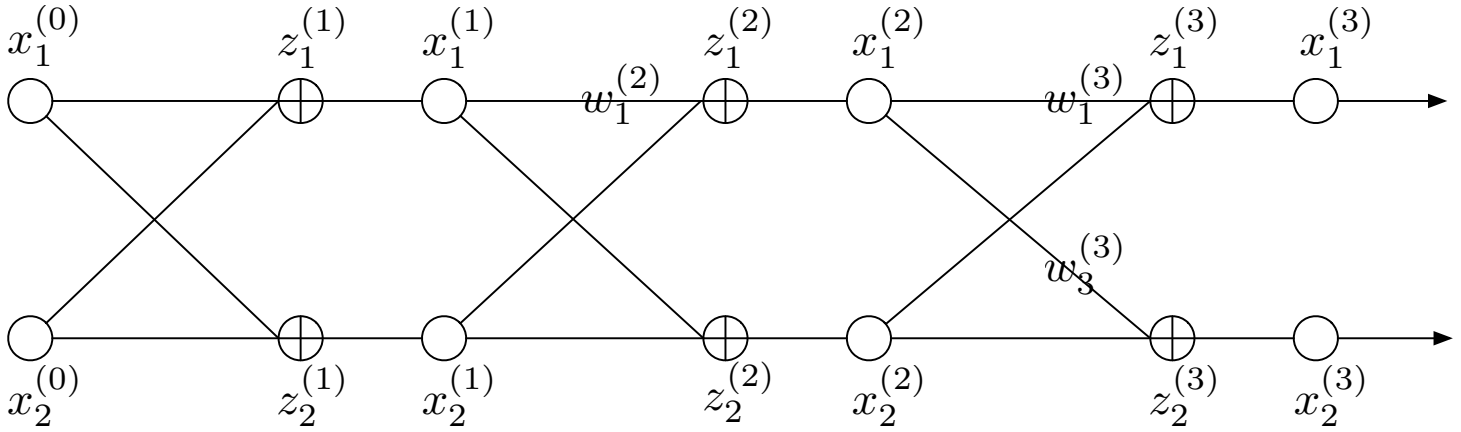
- Demo


Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
4 neurons
2 neurons
The outputs are mixed with varying weights, shown by the thickness of the lines.
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.526
Training loss 0.493

Outline

- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others

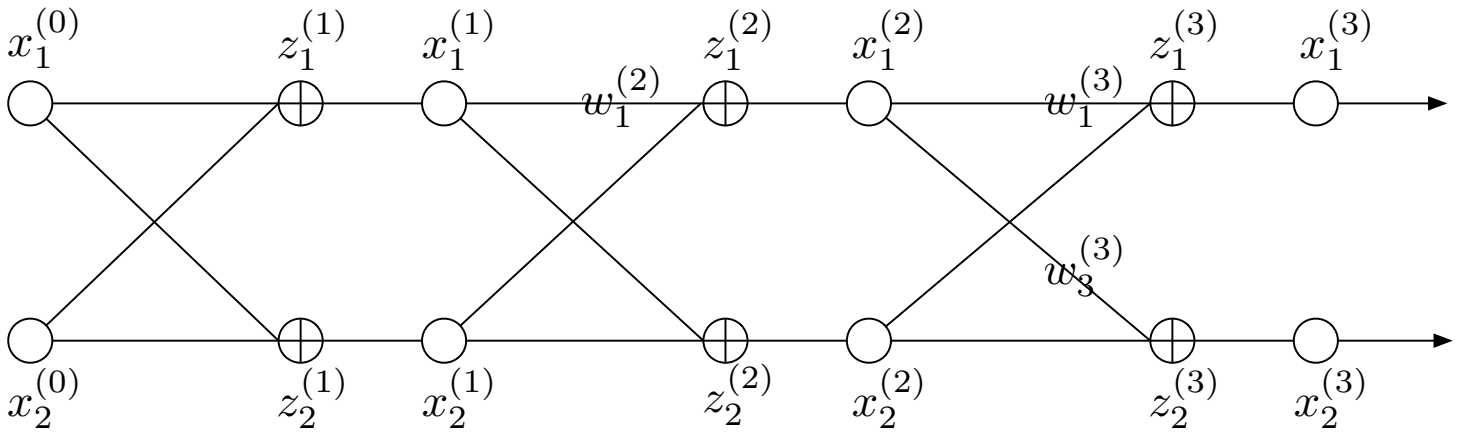
Back-Propagation



- Consider $\partial L/\partial w_1^{(2)}$, where L stands for loss function:

$$\begin{aligned}\frac{\partial L}{\partial w_1^{(2)}} &= \frac{\partial L}{\partial x_1^{(3)}} \cdot \frac{\partial x_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial x_1^{(2)}} \cdot \frac{\partial x_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_1^{(2)}} \\ &+ \frac{\partial L}{\partial x_2^{(3)}} \cdot \frac{\partial x_2^{(3)}}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial x_1^{(2)}} \cdot \frac{\partial x_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_1^{(2)}} \\ &= (\sigma'(z_1^{(3)})w_1^{(3)} + \sigma'(z_2^{(3)})w_3^{(3)})\sigma'(z_1^{(2)})x_1^{(1)}\end{aligned}$$

Back-Propagation



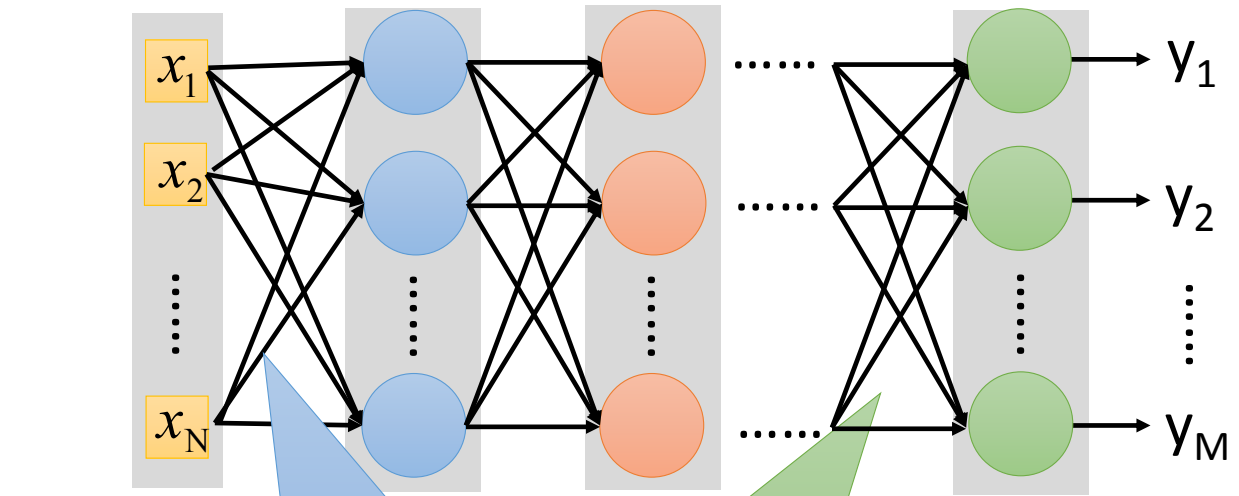
- $\partial L/\partial w_1^{(2)} = (\sigma'(z_1^{(3)})w_1^{(3)} + \sigma'(z_2^{(3)})w_3^{(3)})\sigma'(z_1^{(2)})x_1^{(1)}$.
  Note that $\sigma'(z)$ is a constant because z is already determined in the forward pass.
- x and z are computed by the forward pass, and $\sigma'(z)$ are computed by the backward pass.

Outline

- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others
- How to Build a DNN in Keras

Activation Function – sigmoid

- The main drawback of the sigmoid function is vanishing gradient problem.



Smaller gradients

Learn very slow

Almost random

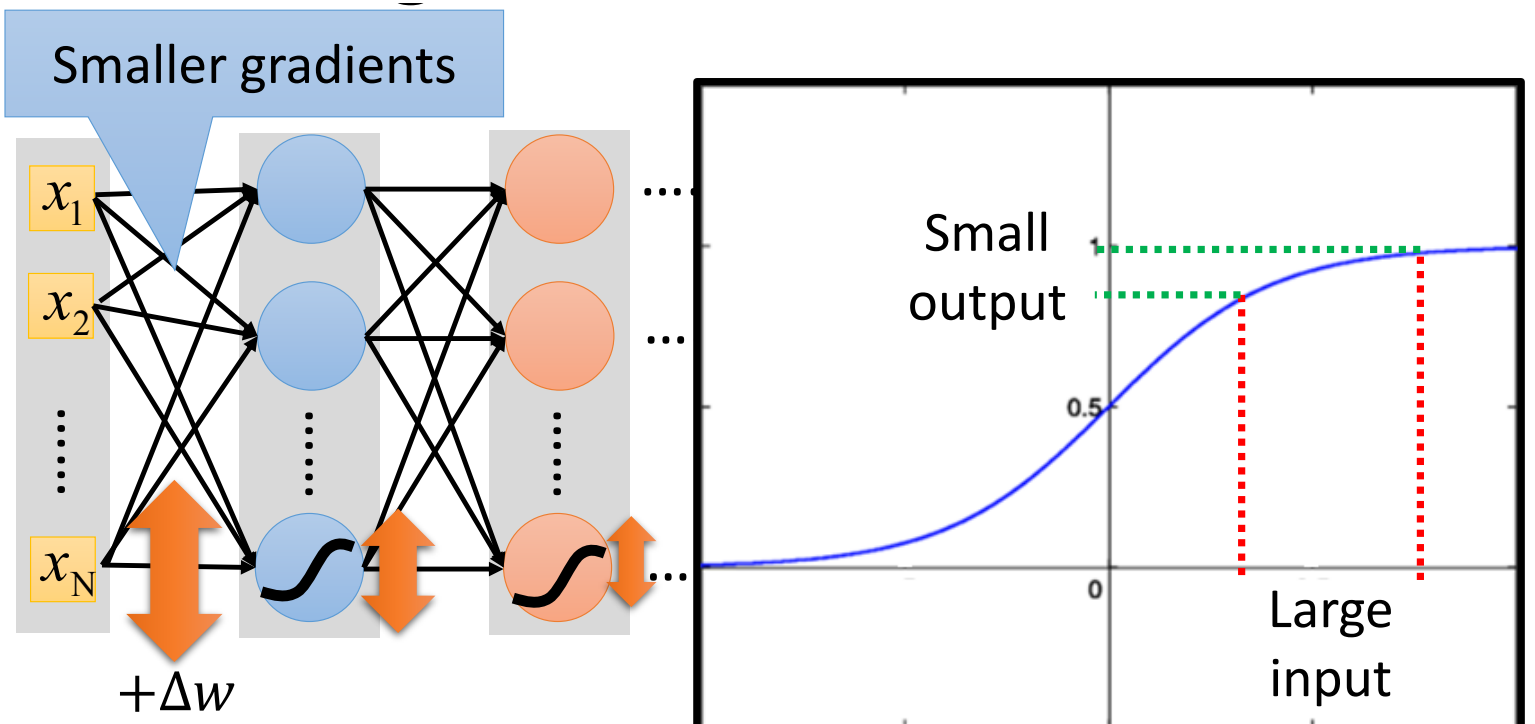
Larger gradients

Learn very fast

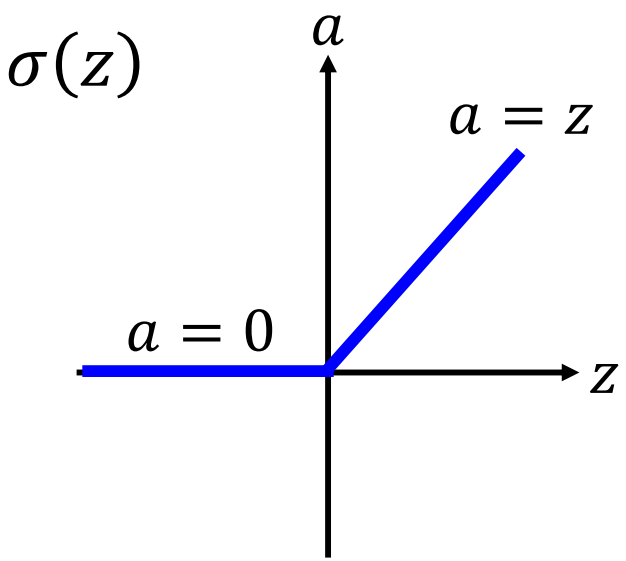
Already converge

based on random!?

Activation Function – sigmoid



Activation Function – ReLU (Rectified Linear Unit)

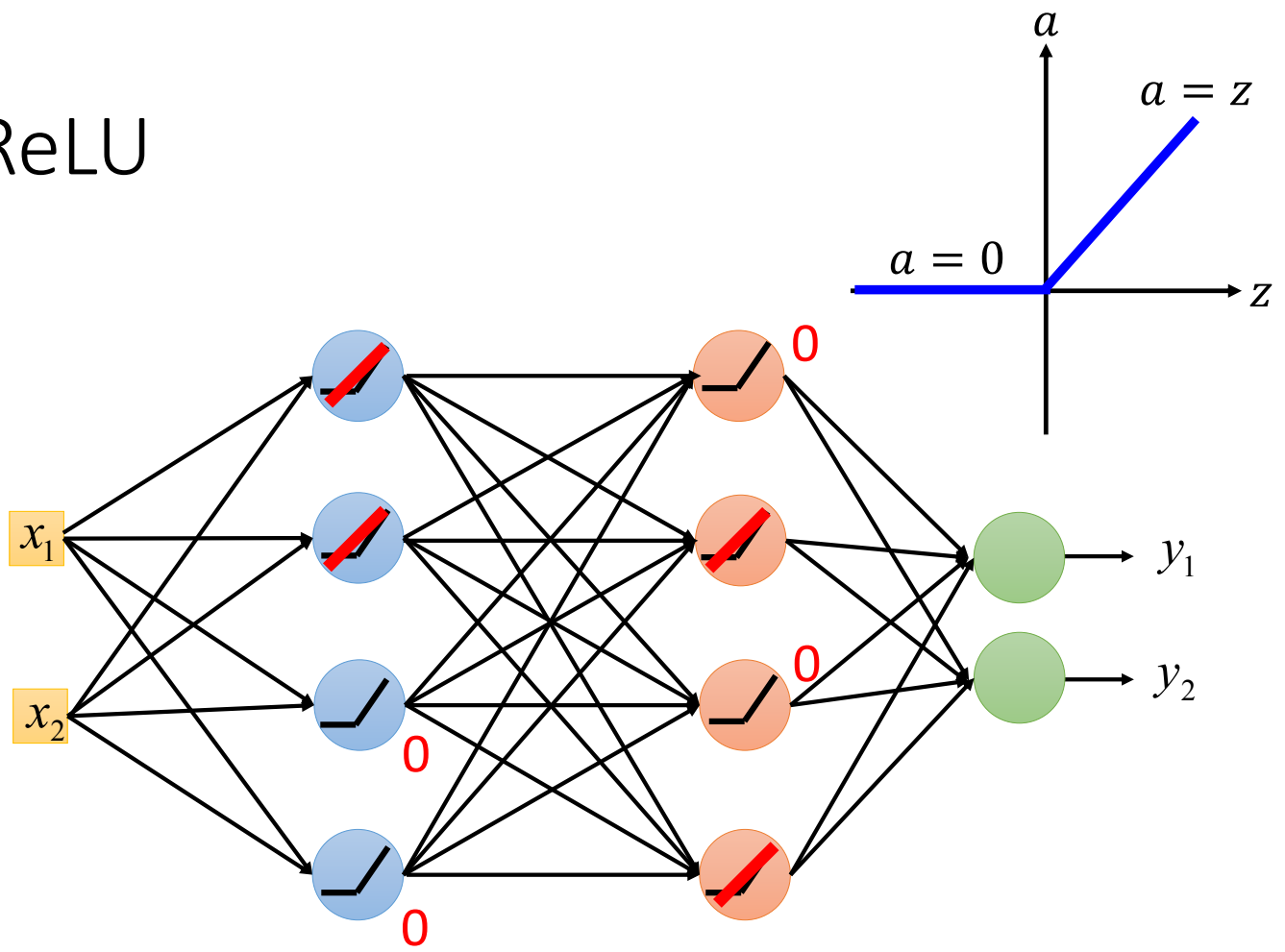


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

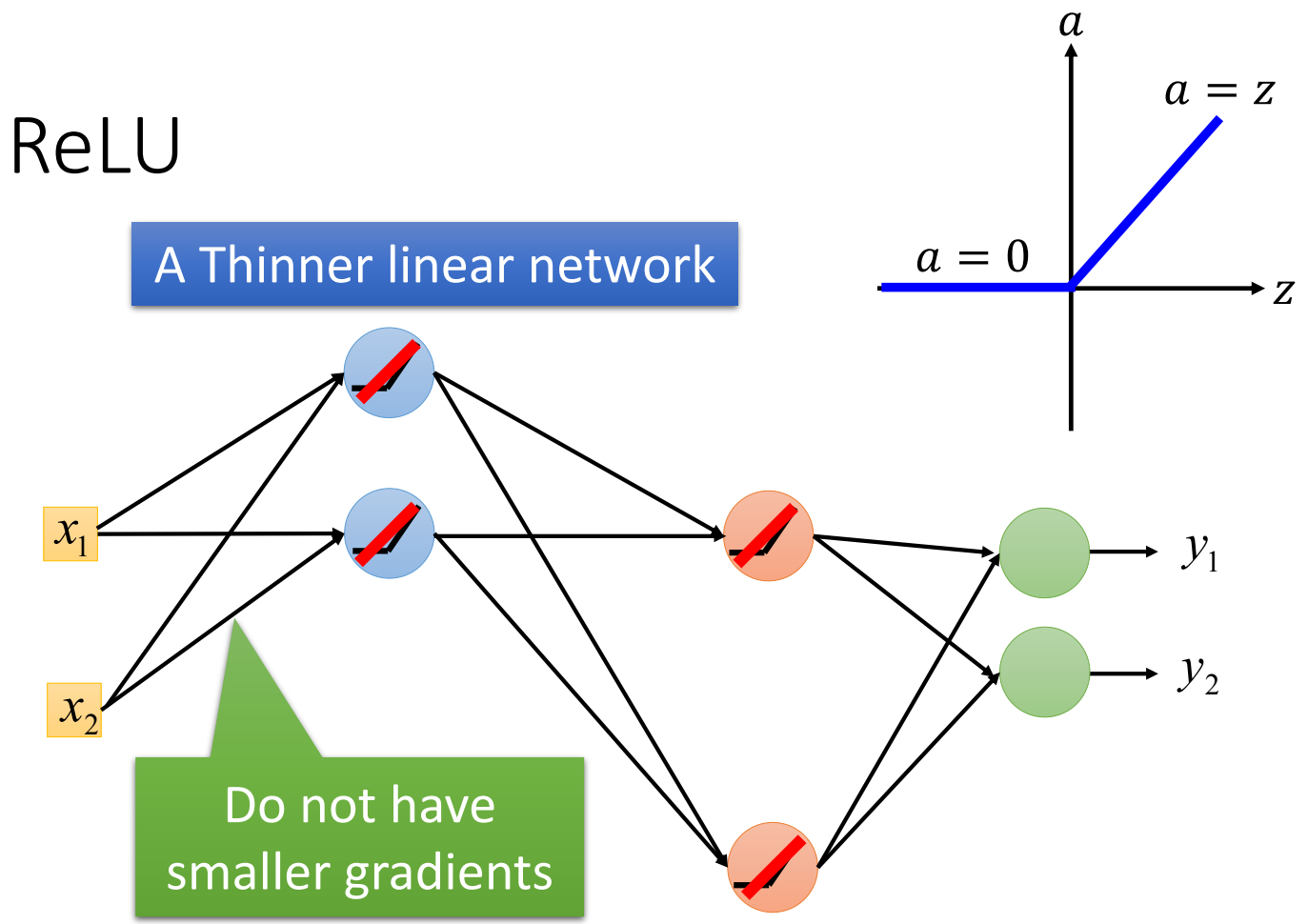
Reason:

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

Activation Function – ReLU



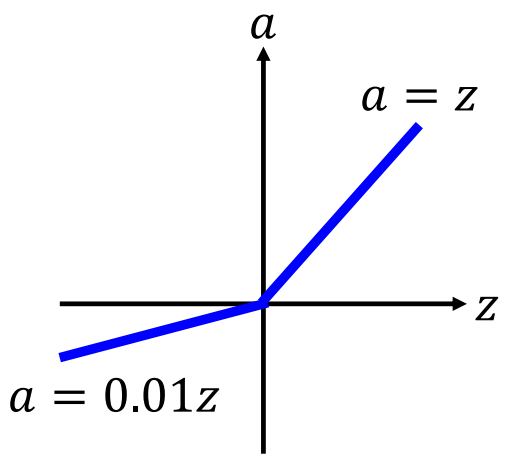
Activation Function – ReLU



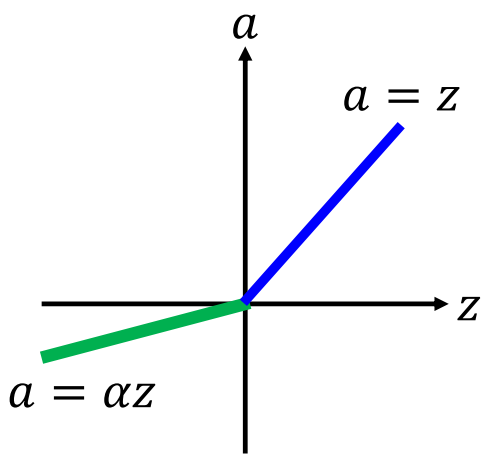
Activation Function – ReLU

ReLU - variant

Leaky ReLU



Parametric ReLU



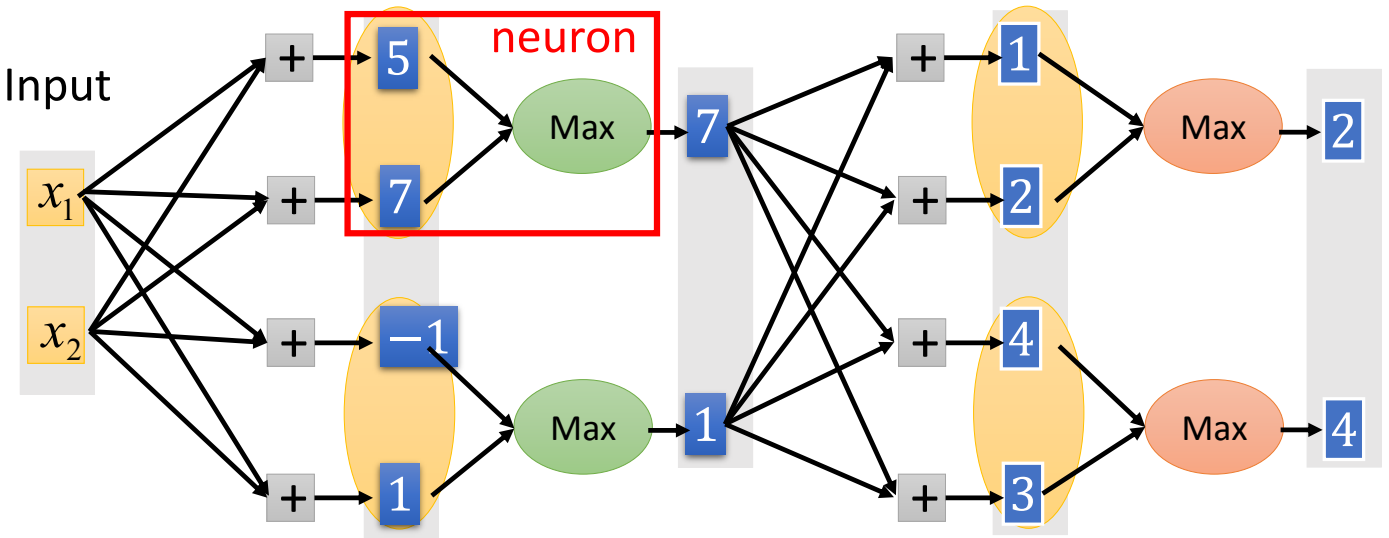
α also learned by gradient descent

Activation Function – MaxOut

Maxout

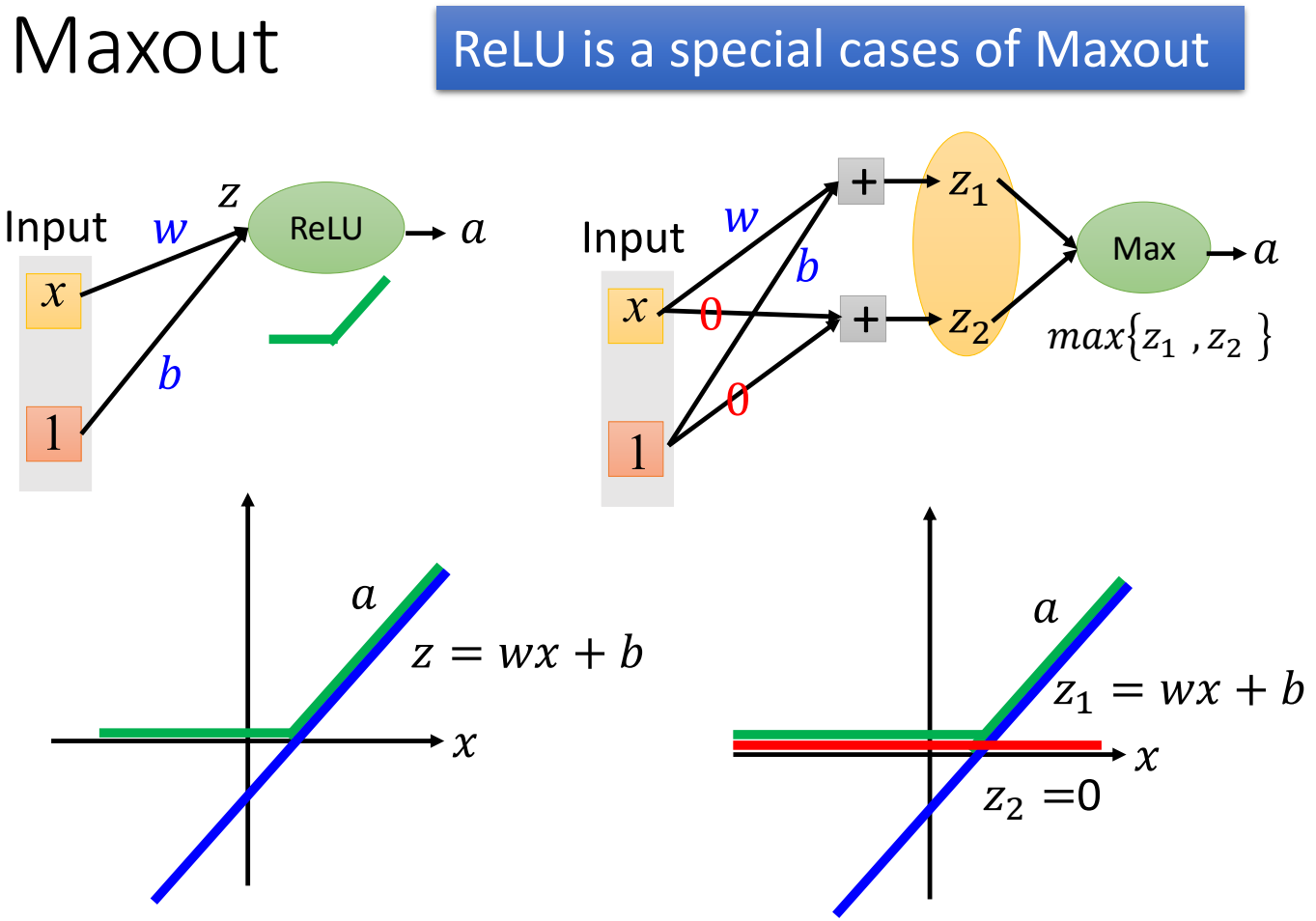
ReLU is a special cases of Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]

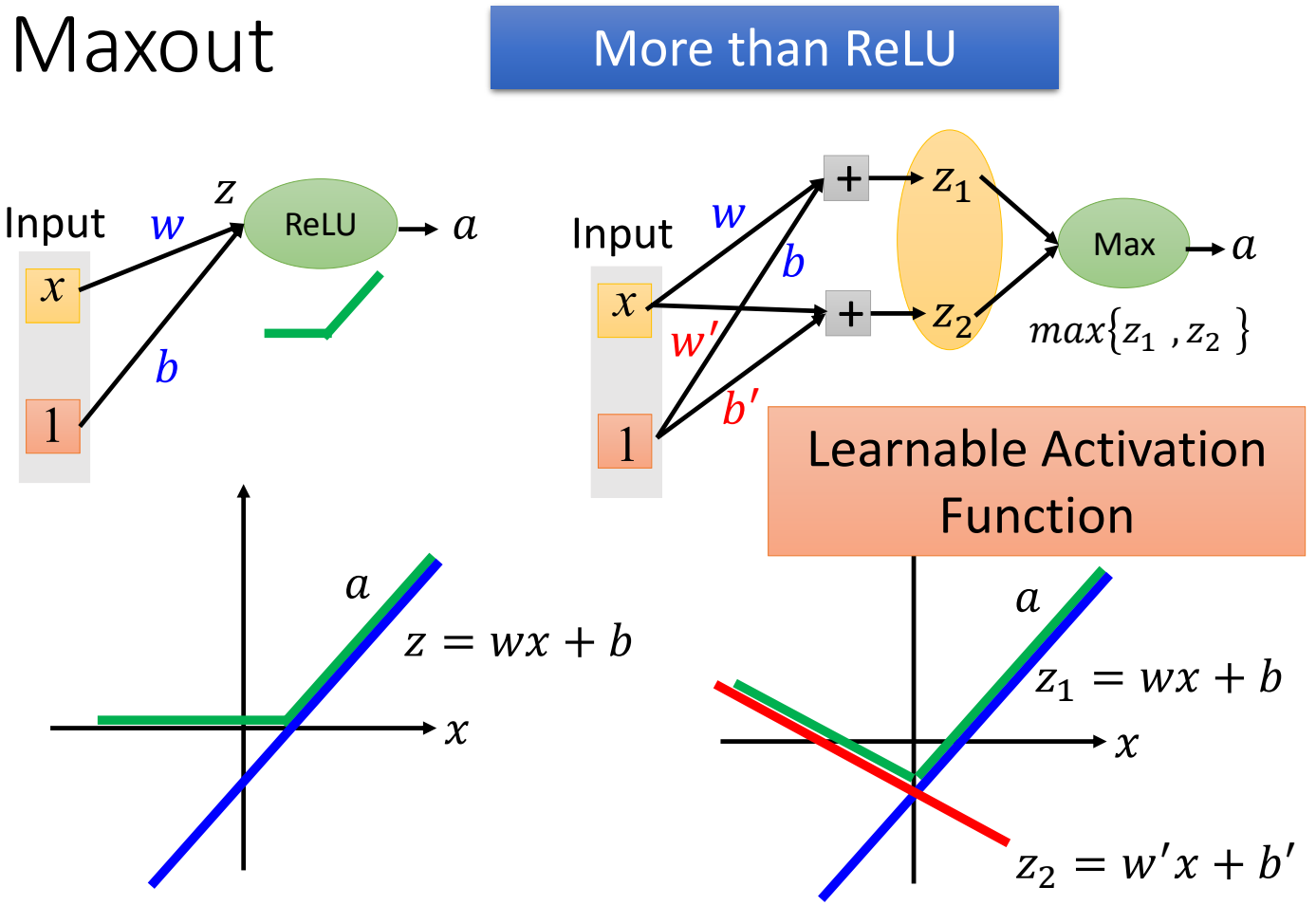


You can have more than 2 elements in a group.

Activation Function – MaxOut



Activation Function – MaxOut

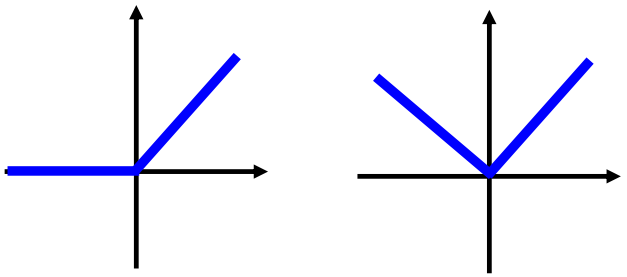


Activation Function – MaxOut

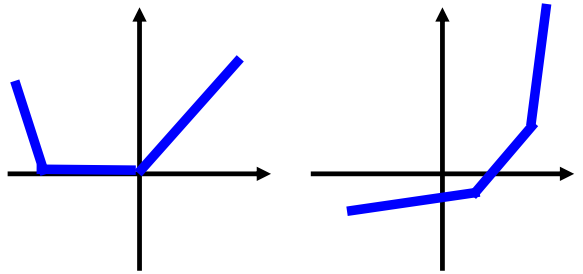
Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]
  - Activation function in maxout network can be any piecewise linear convex function
  - How many pieces depending on how many elements in a group

2 elements in a group



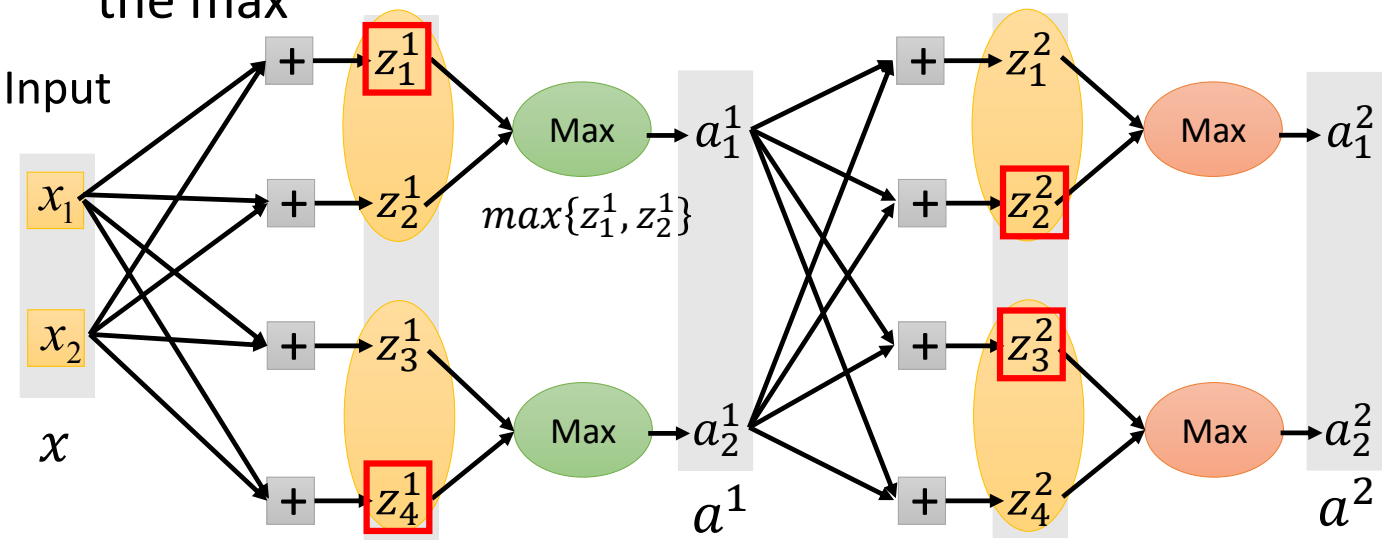
3 elements in a group



Activation Function – MaxOut

Maxout - Training

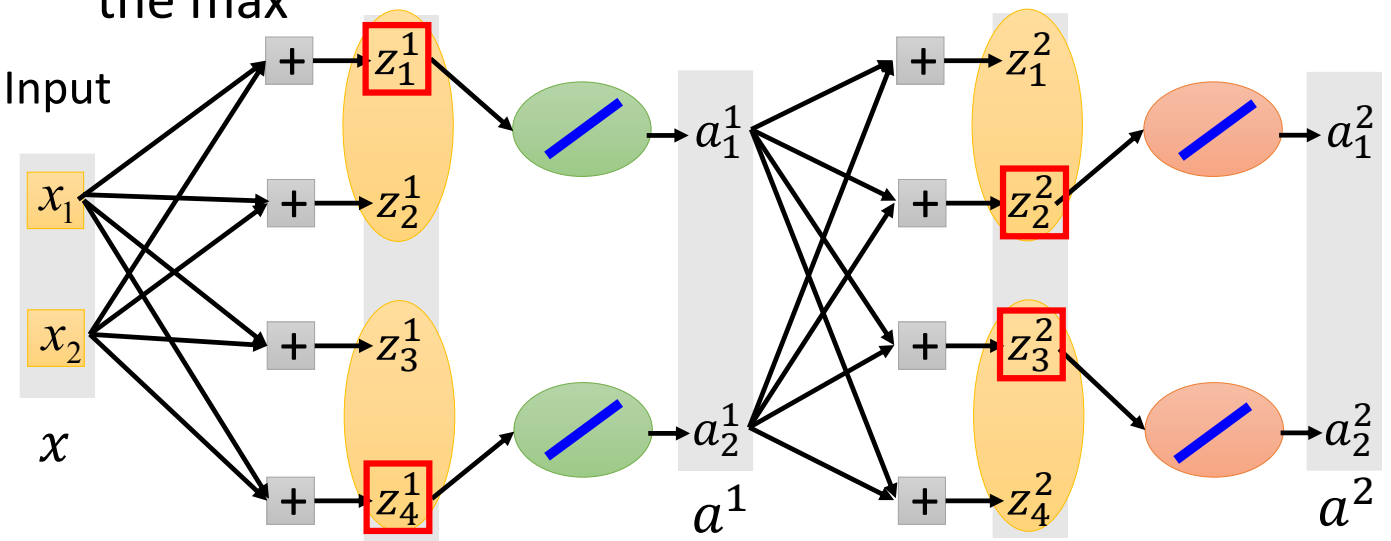
- Given a training data x, we know which z would be the max



Activation Function – MaxOut

Maxout - Training

- Given a training data x, we know which z would be the max



- Train this thin and linear network

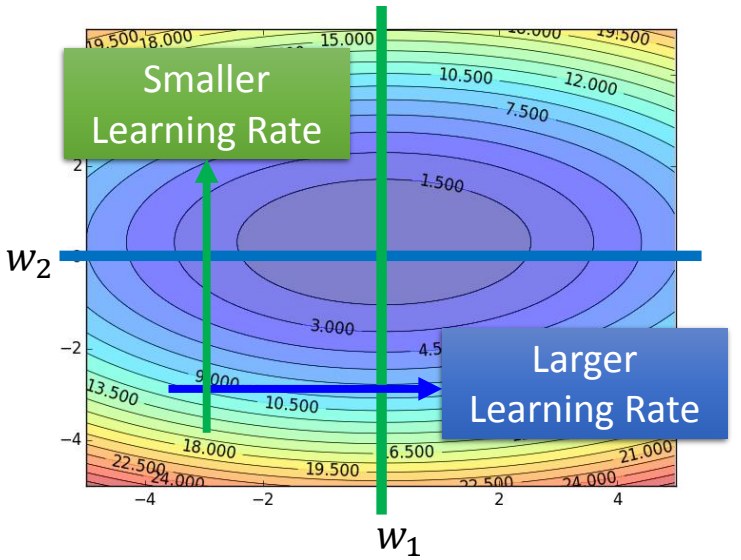
Different thin and linear network for different examples

Optimization – AdaGrad

- The AdaGrad algorithm adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient.

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

- Performs well for convex problem.

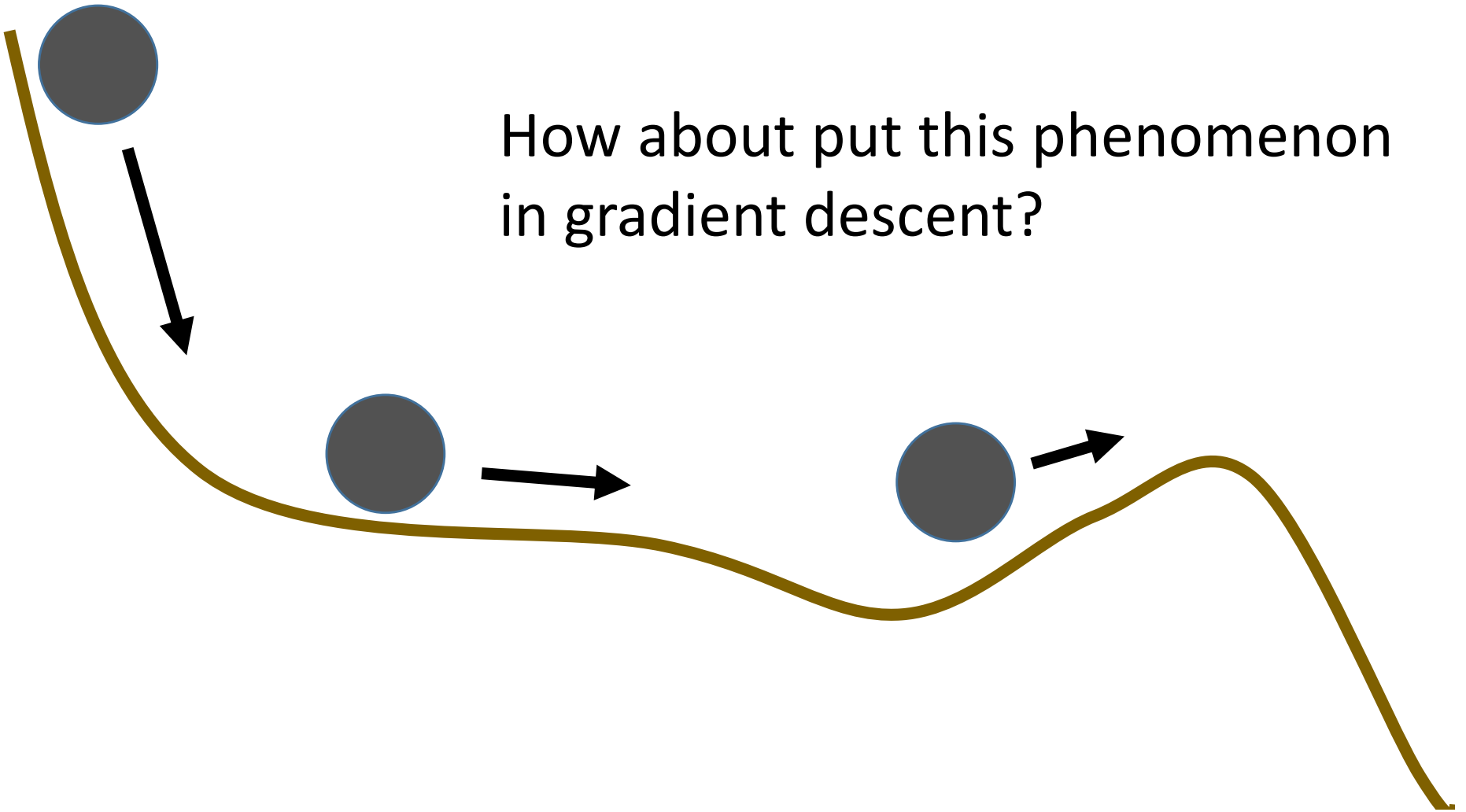


Optimization – RMSProp

- Modifies AdaGrad by changing the gradient accumulation into an exponentially weighted moving average.

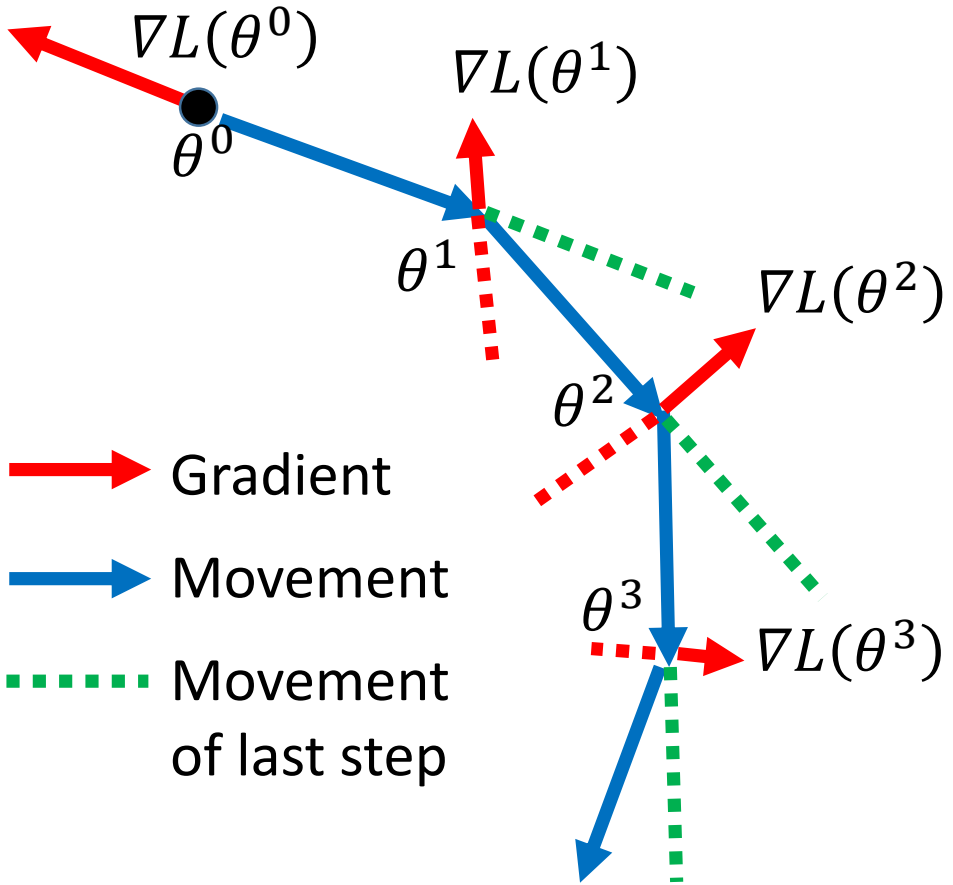
$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \quad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

Optimization – Momentum



Optimization – Momentum

Movement: movement of last step minus gradient at present



Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta \nabla L(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

Movement $v^2 = \lambda v^1 - \eta \nabla L(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

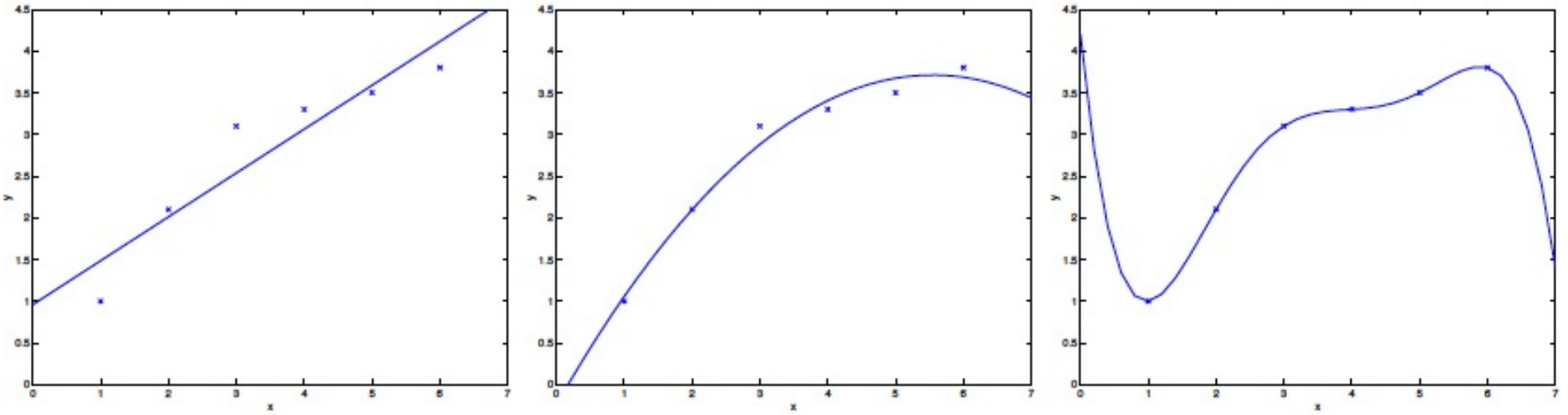
Movement not just based on gradient, but previous movement.

Optimization – Adam

- Adam (adaptive moments): RMSProp + Momentum
- How to choose the right optimization algorithm? Try!!!

Dropout

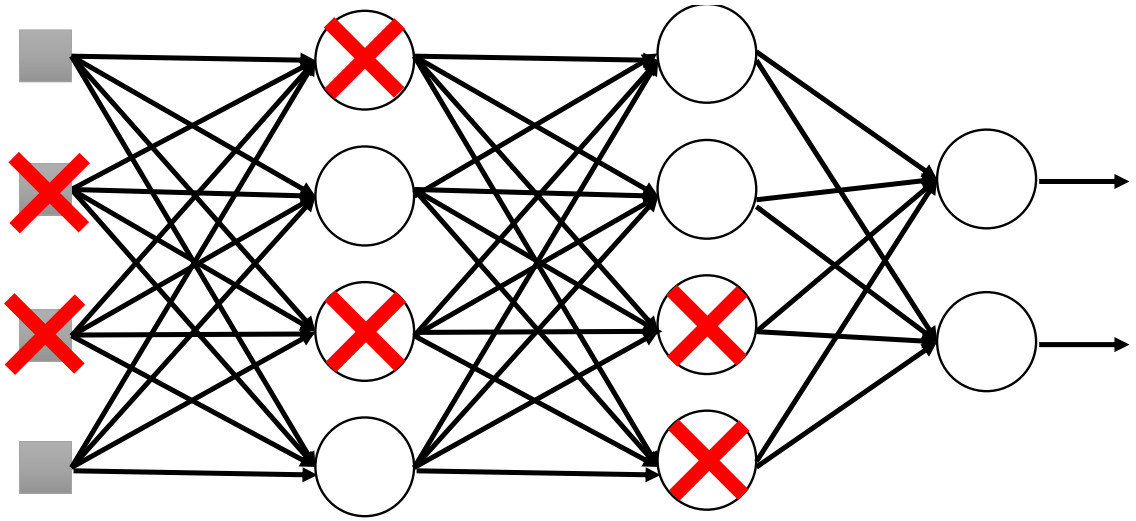
- What is overfitting?



- Overfitting has good performance in training set but poor performance for test set.
- For example: $y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5$
- If we restrict $w_3, w_4, w_5 = 0$, then no overfitting. How?

Dropout

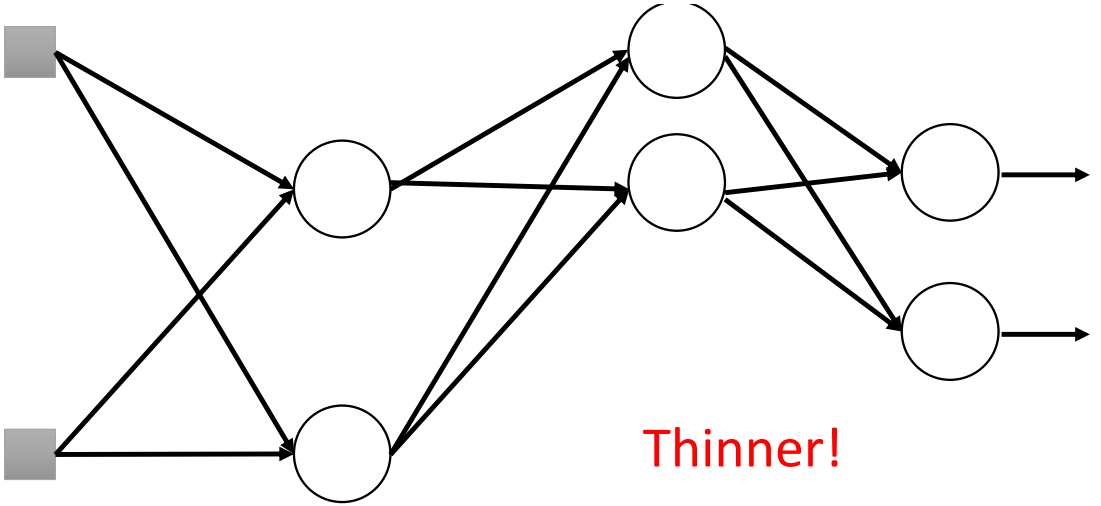
Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Dropout

Training:

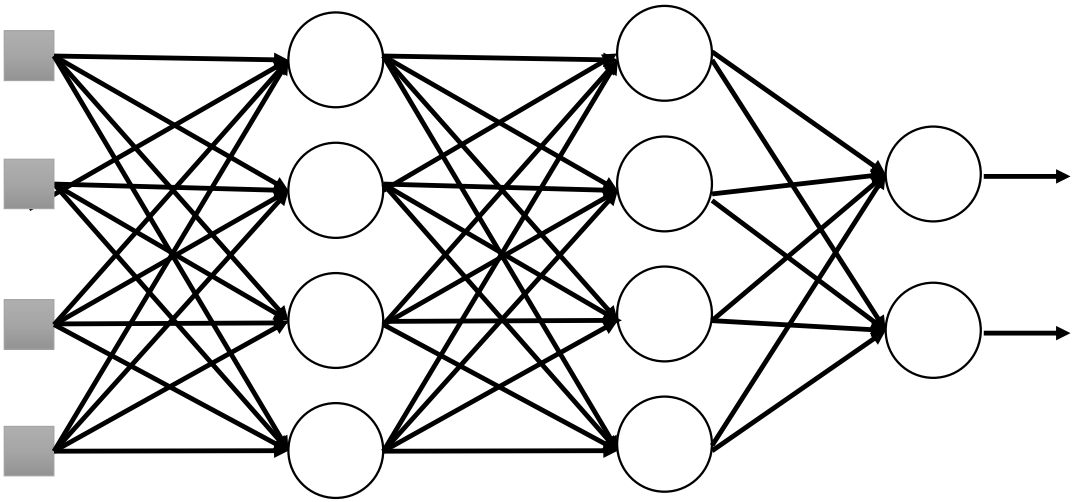


- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

Testing:



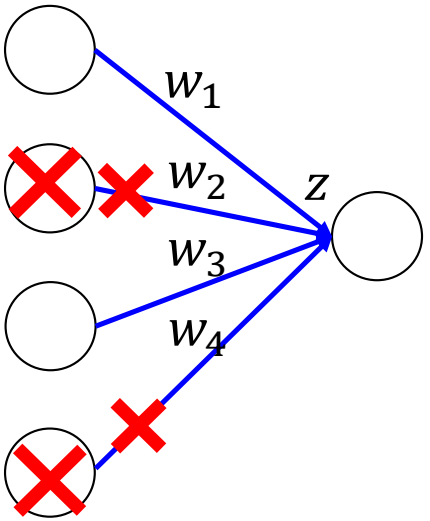
- **No dropout**
  - If the dropout rate at training is p%, all the weights times 1-p%
  - Assume that the dropout rate is 50%. If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout

- Why the weights should multiply (1-p)% (dropout rate) when testing?

Training of Dropout

Assume dropout rate is 50%

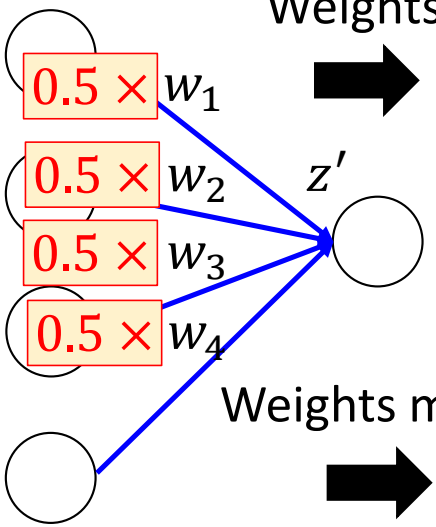


Testing of Dropout

No dropout

Weights from training

$z' \approx 2z$



Weights multiply 1-p%

$z' \approx z$

- Dropout works better for linear network. Hence, better for ReLU and MaxOut.

Outline

- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others

Why CNN for Image

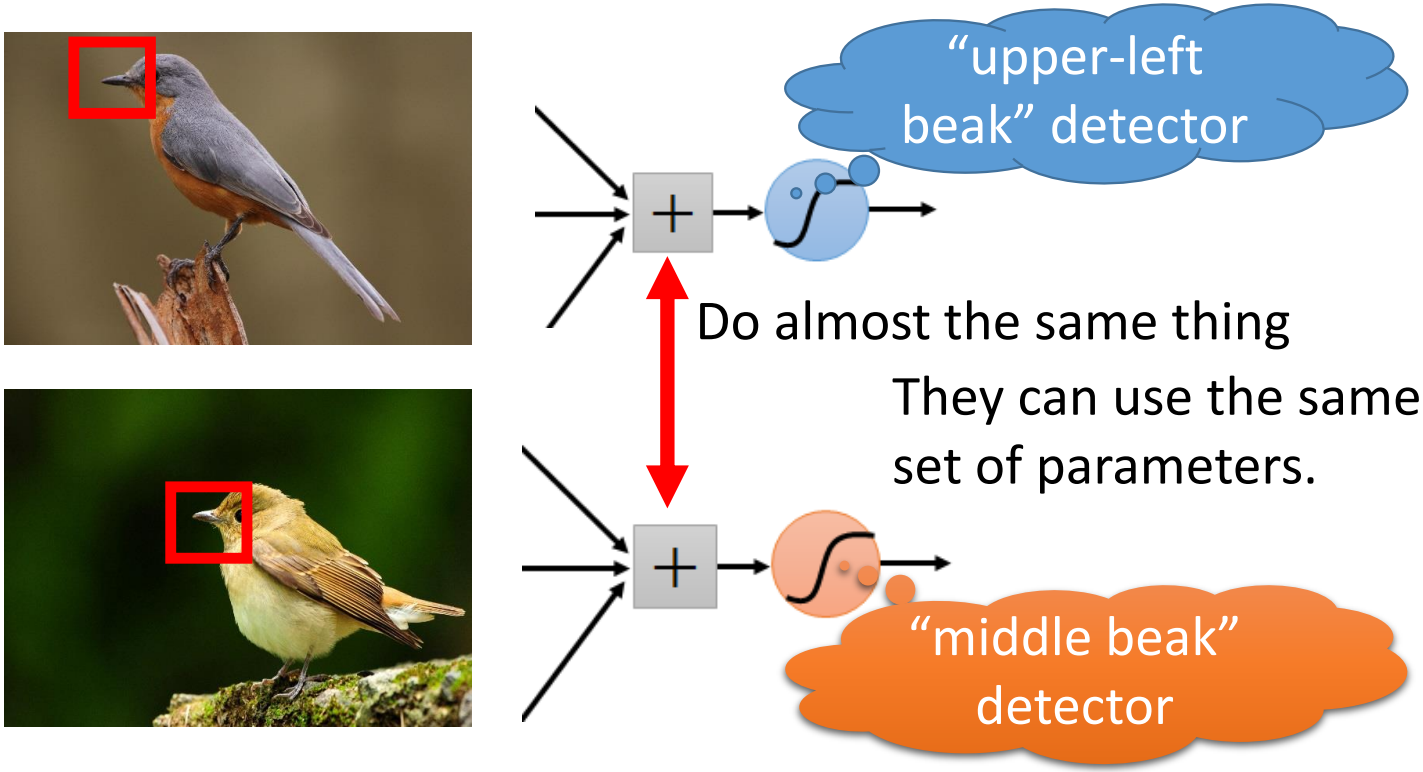
- Some patterns are much smaller than the whole image

A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters

Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

- Subsampling the pixels will not change the object

bird

subsampling

bird

We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

CNN – Convolution

Those are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

⋮

Property 1 Each filter detects a small pattern (3 x 3).

CNN – Convolution

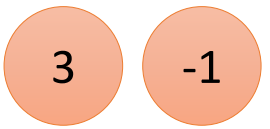
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



CNN – Convolution

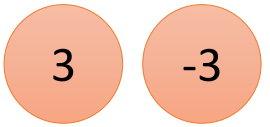
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

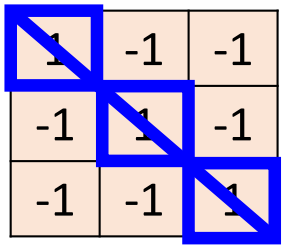
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



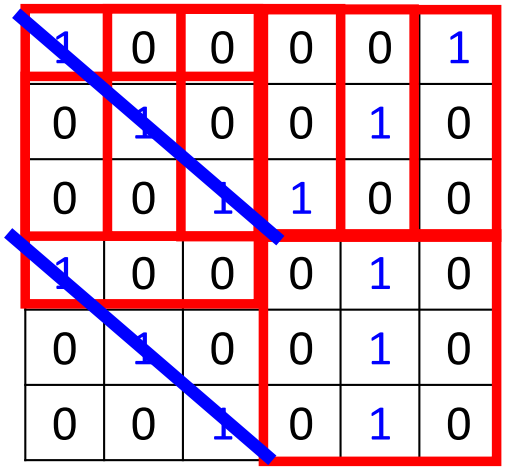
We set stride=1 below

CNN – Convolution

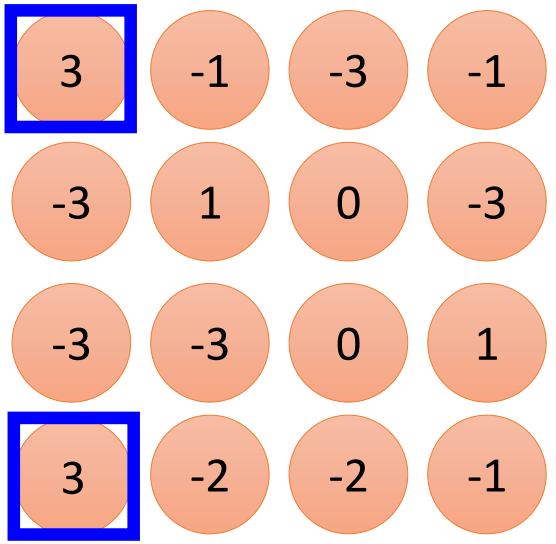


Filter 1

stride=1



6 x 6 image

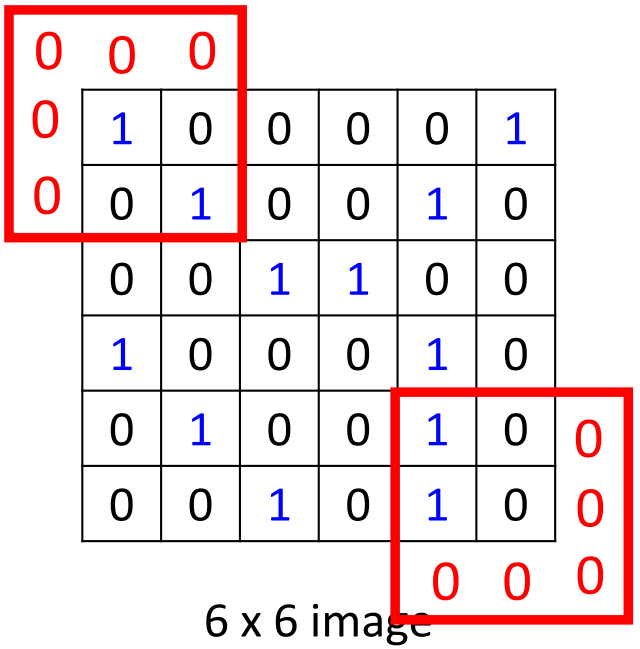


Property 2

CNN – Zero Padding

1	-1	-1
-1	1	-1
-1	-1	1

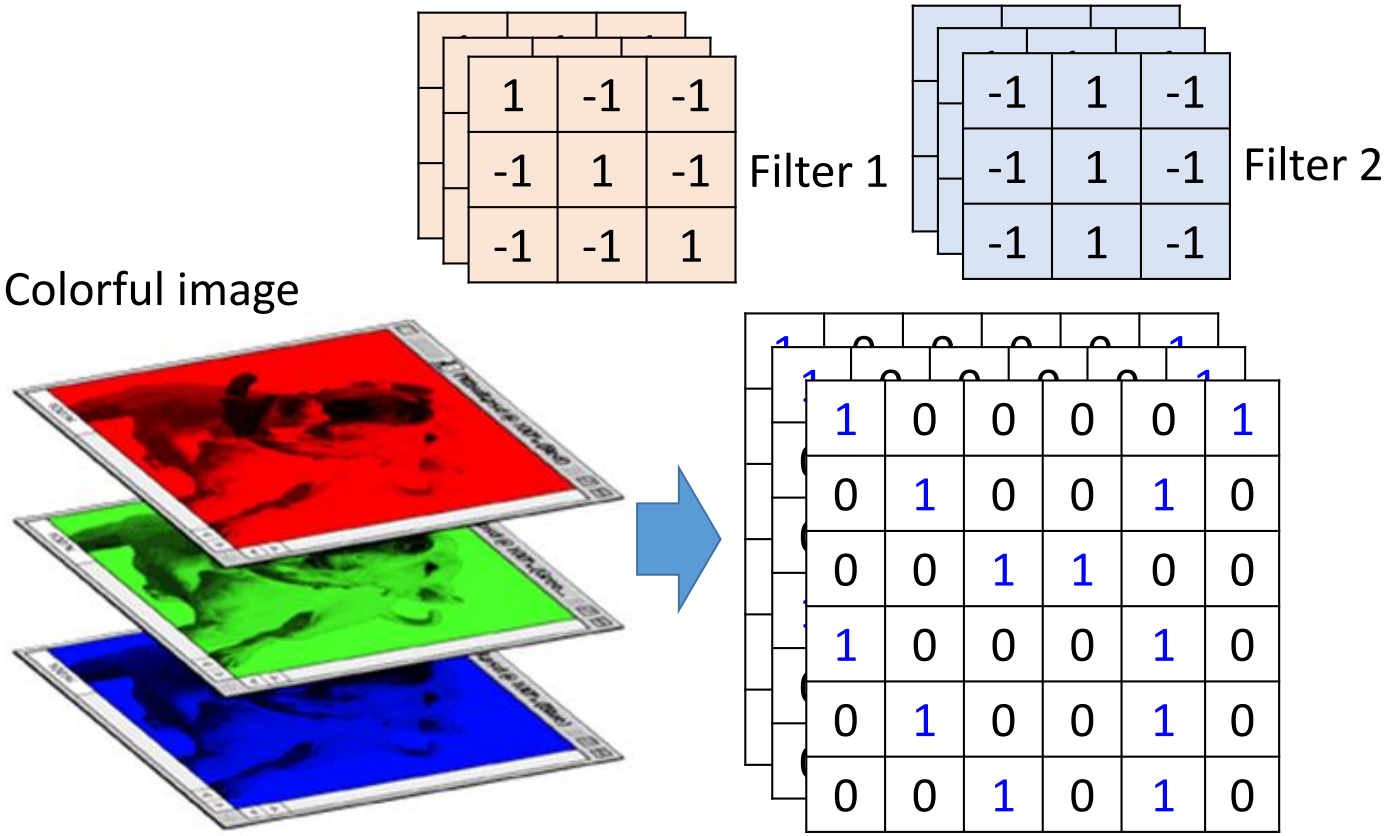
Filter 1



You will get another 6 x 6 images in this way

Zero padding

CNN – Colorful image



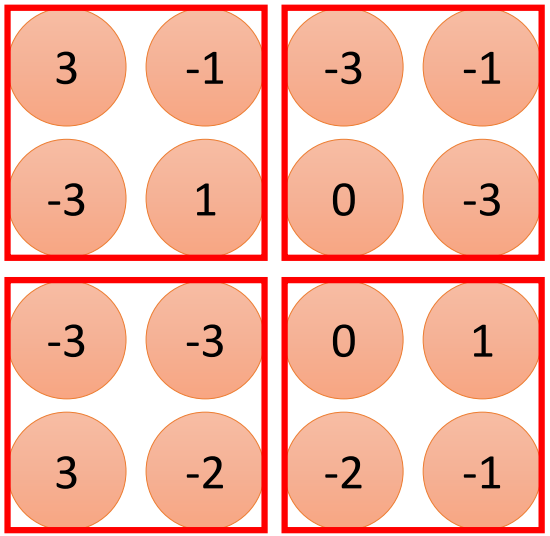
CNN – Max Pooling

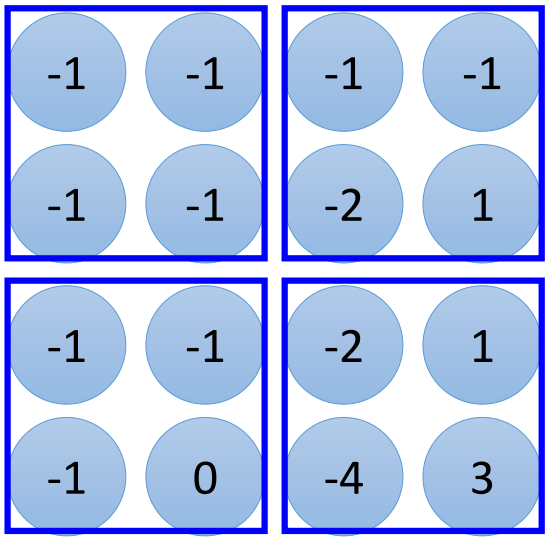
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

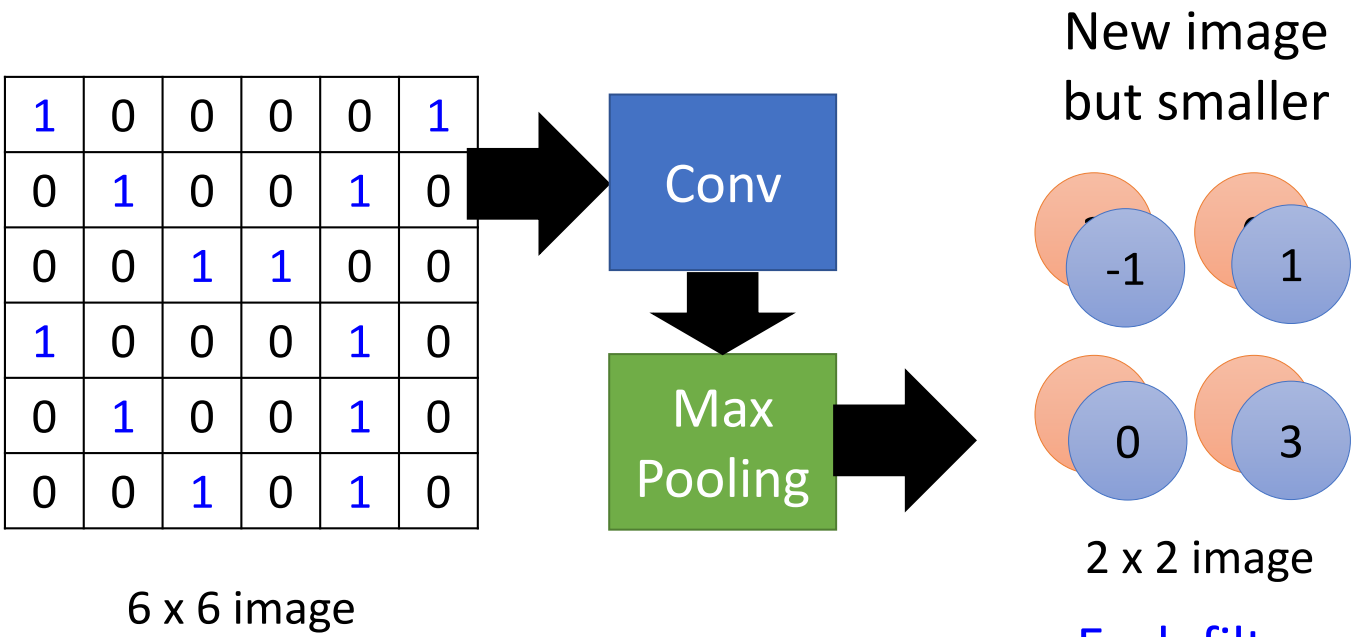
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





CNN – Max Pooling

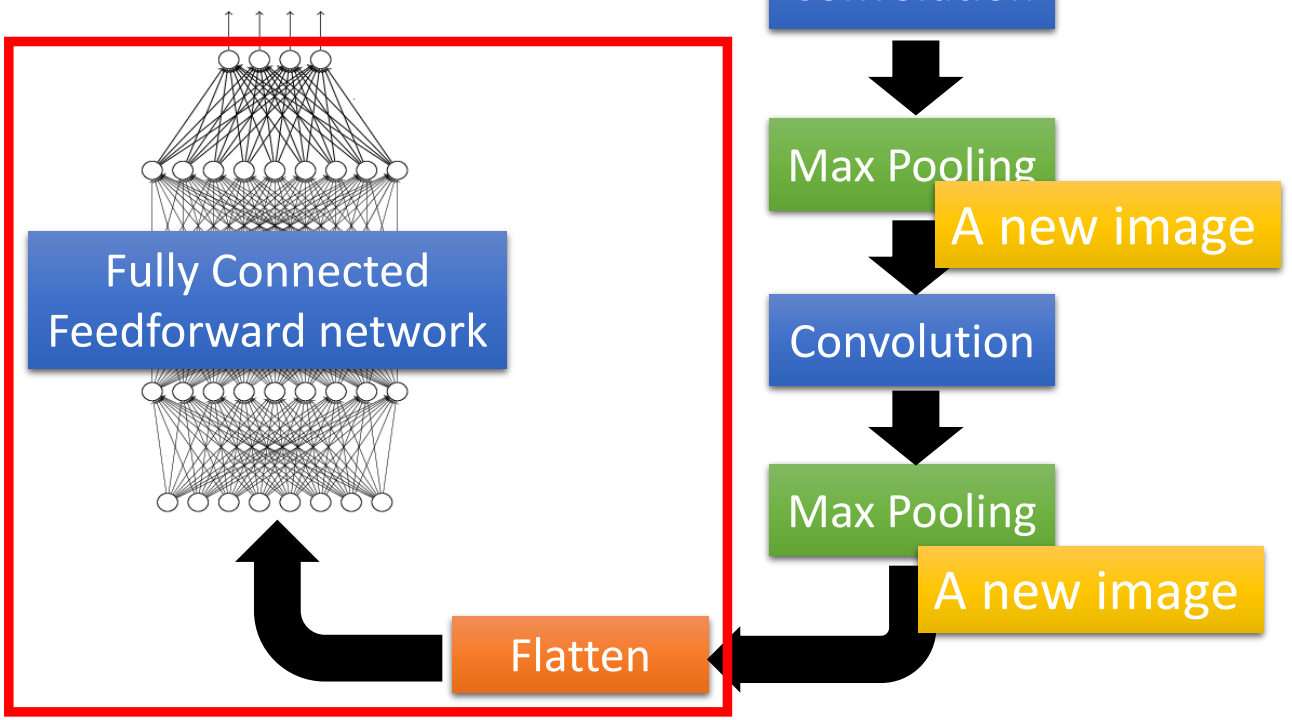


6 x 6 image

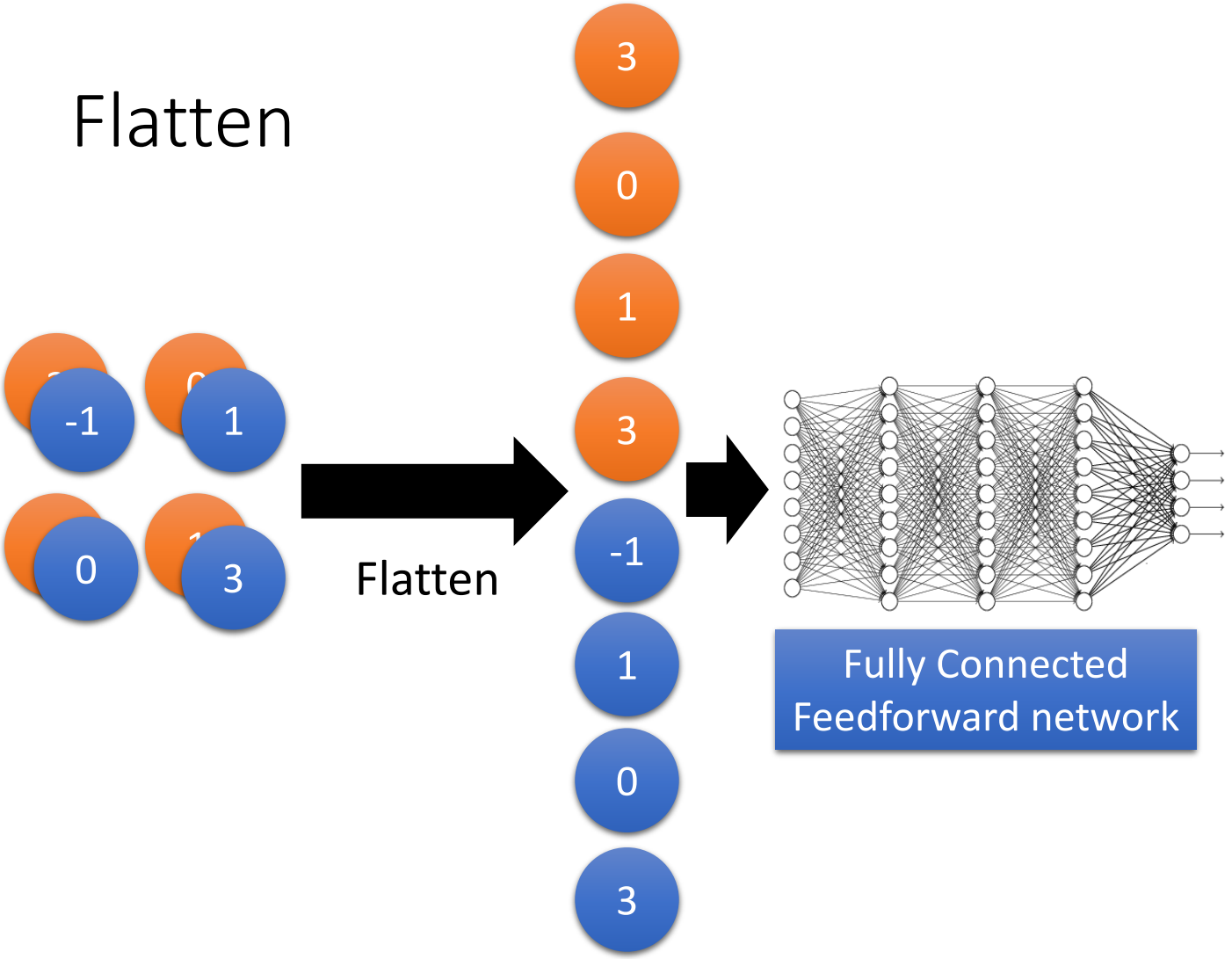
2 x 2 image

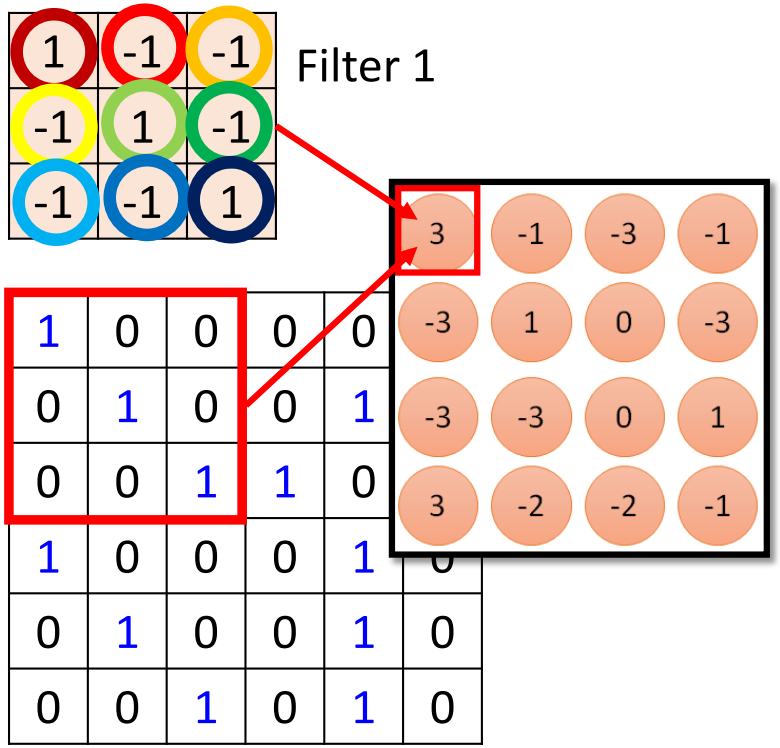
Each filter is a channel

The whole CNN

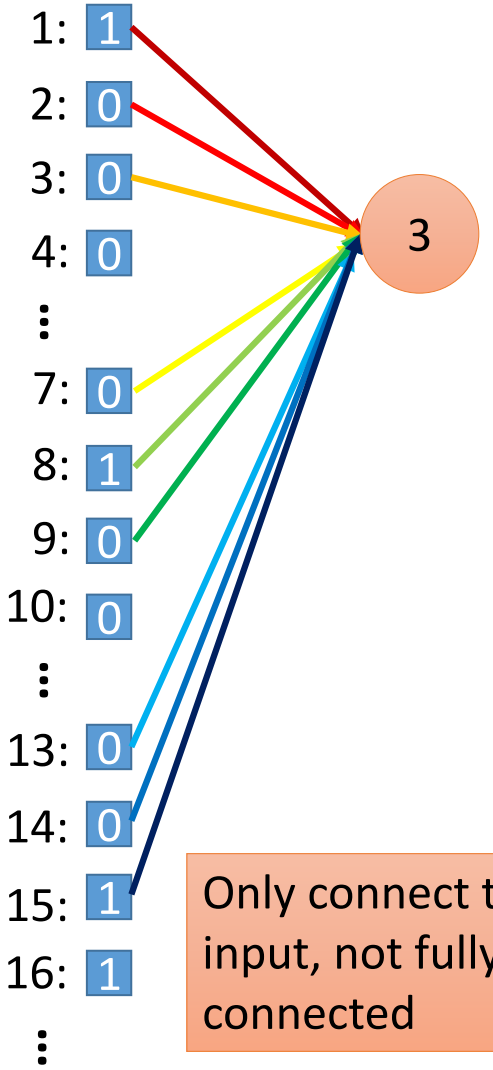


Flatten


-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network



Less parameters!



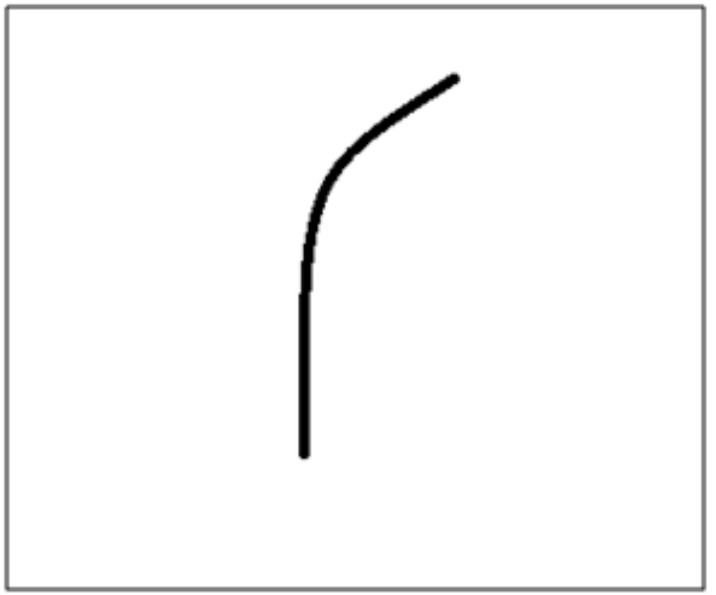
Only connect to 9 input, not fully connected

More about CNN-high level perspective

- Each filters can be thought of as feature identifiers. (like straight edges, simple colors, curves, etc.)

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

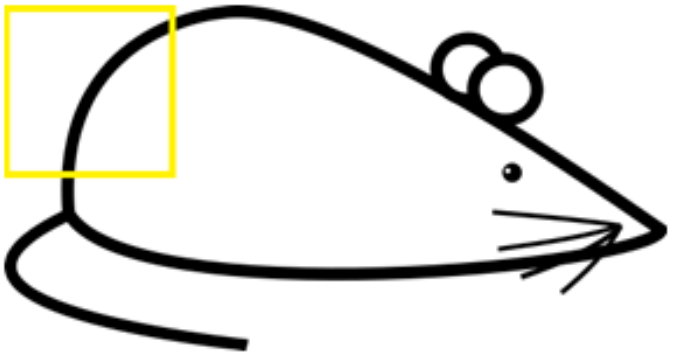


Visualization of a curve detector filter

More about CNN-high level perspective

- Visualizing the CNN

Original image



Visualization of the filter on the image

More about CNN-high level perspective

- Visualizing the CNN

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

- This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate.

More about CNN-high level perspective

- Visualizing the CNN

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

- There wasn't a curve in that region of the original image.

More about CNN-high level perspective

- Visualizing the CNN

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

- The more filters, the greater the depth of the activation map, and the more information we have about the input volume.

Something Else

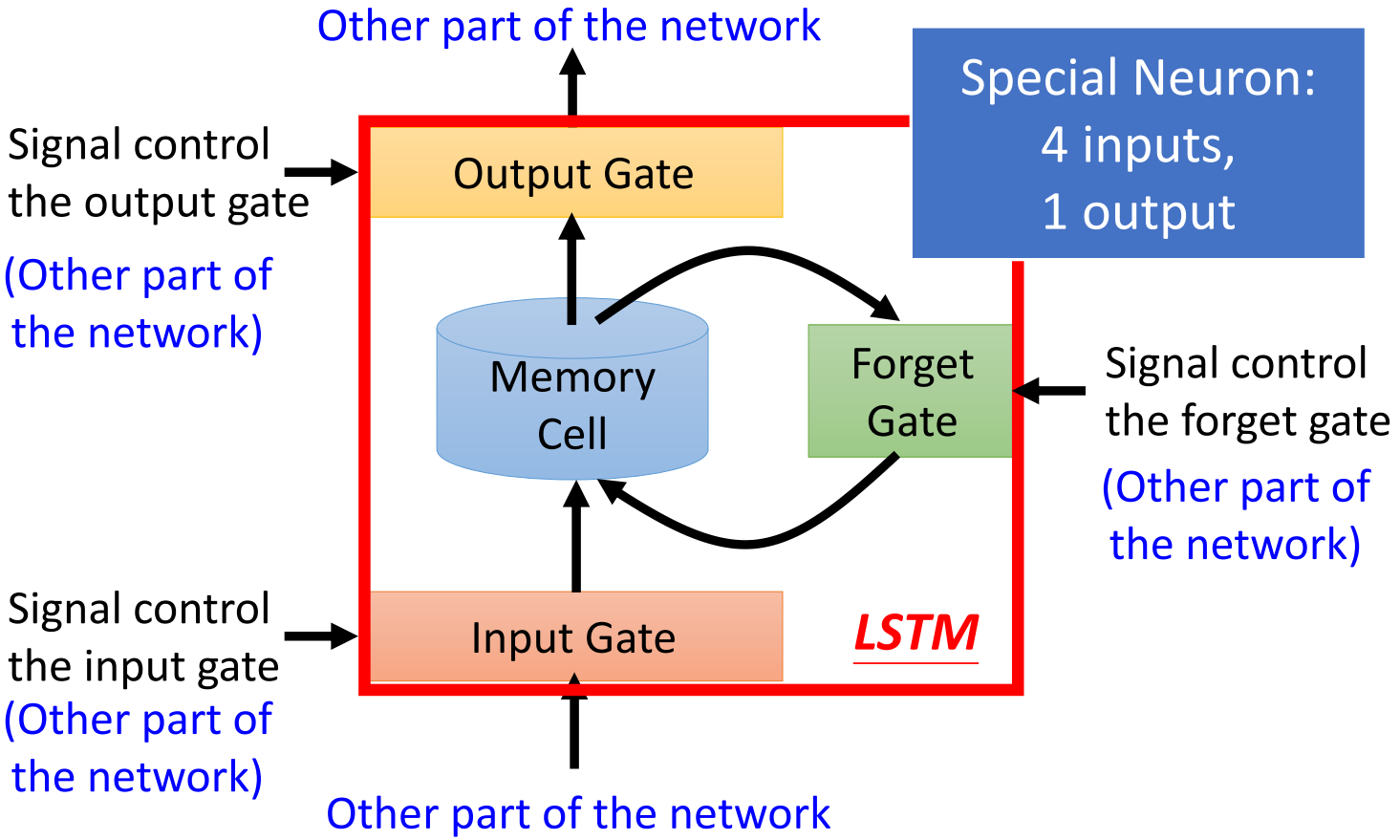
- Deep Style

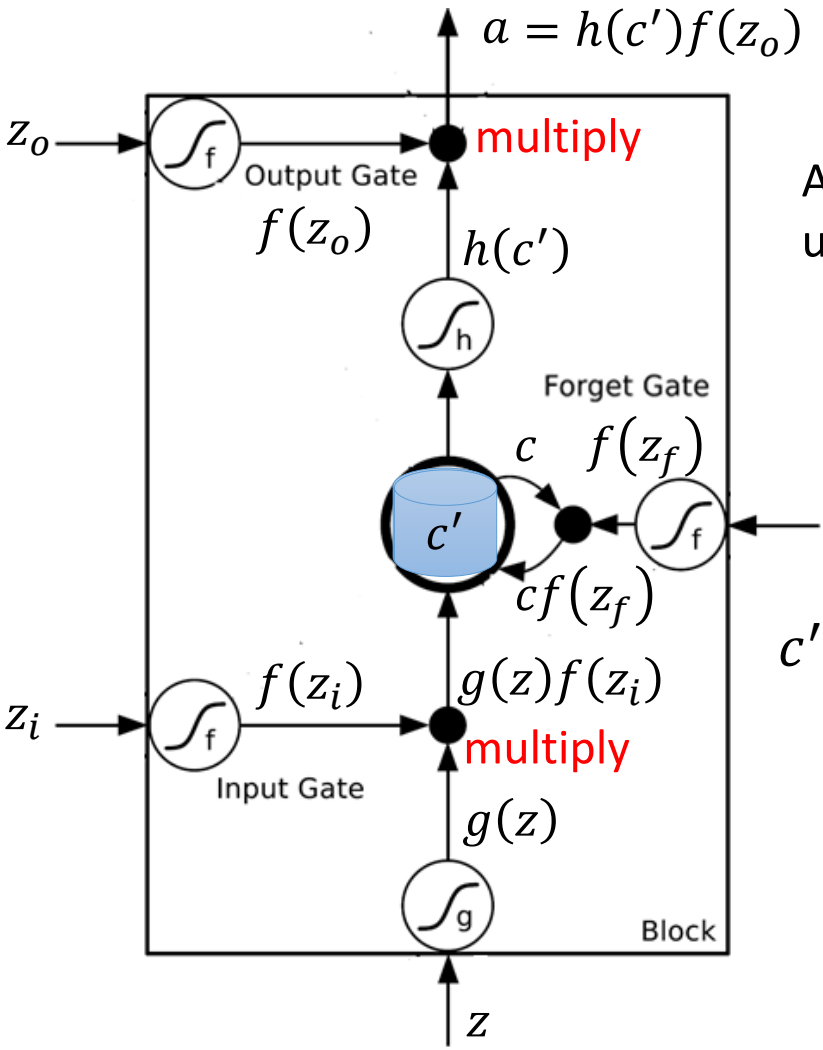
Deep Dream

Outline

- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others

Long Short-term Memory (LSTM)





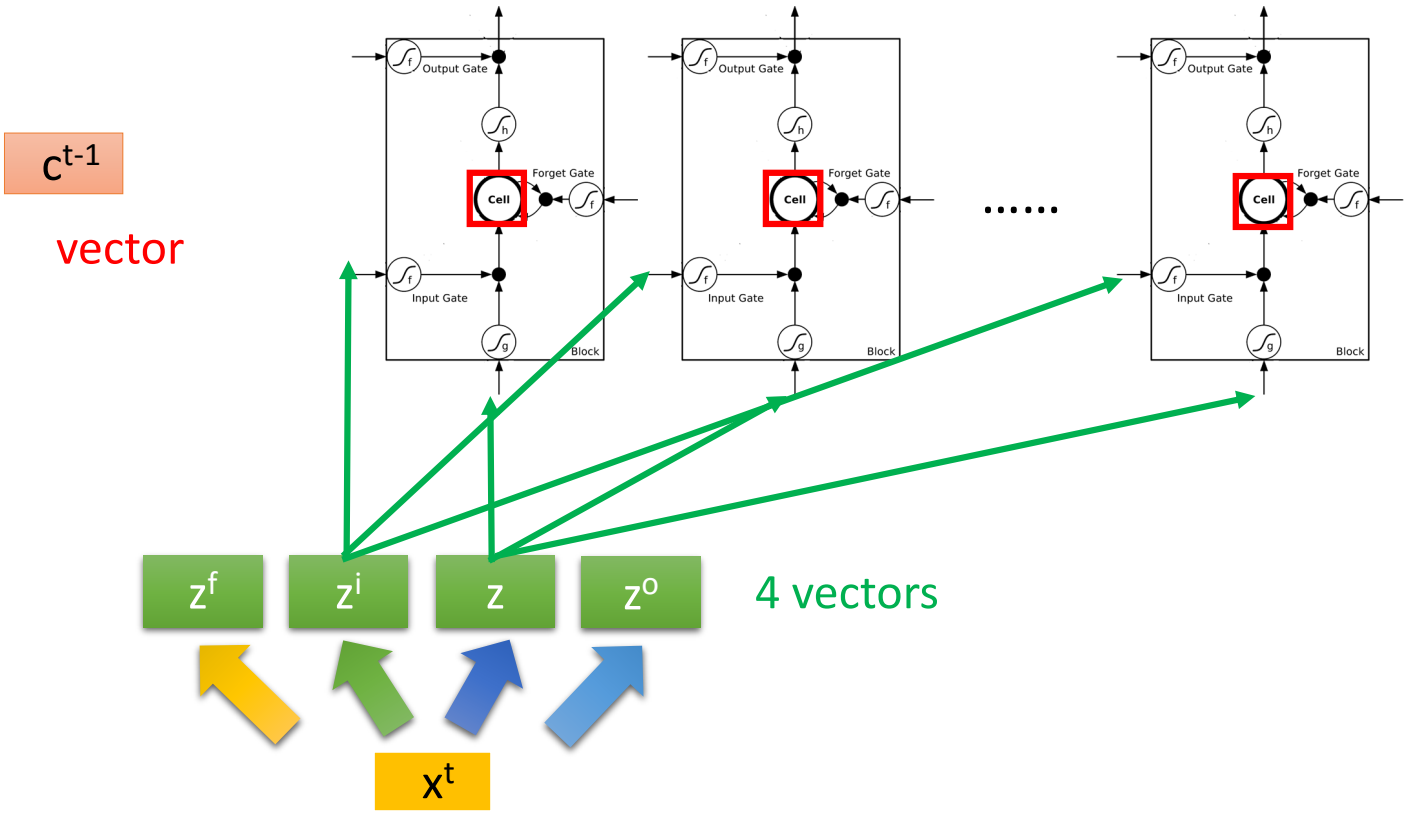
Activation function f is usually a sigmoid function

Between 0 and 1

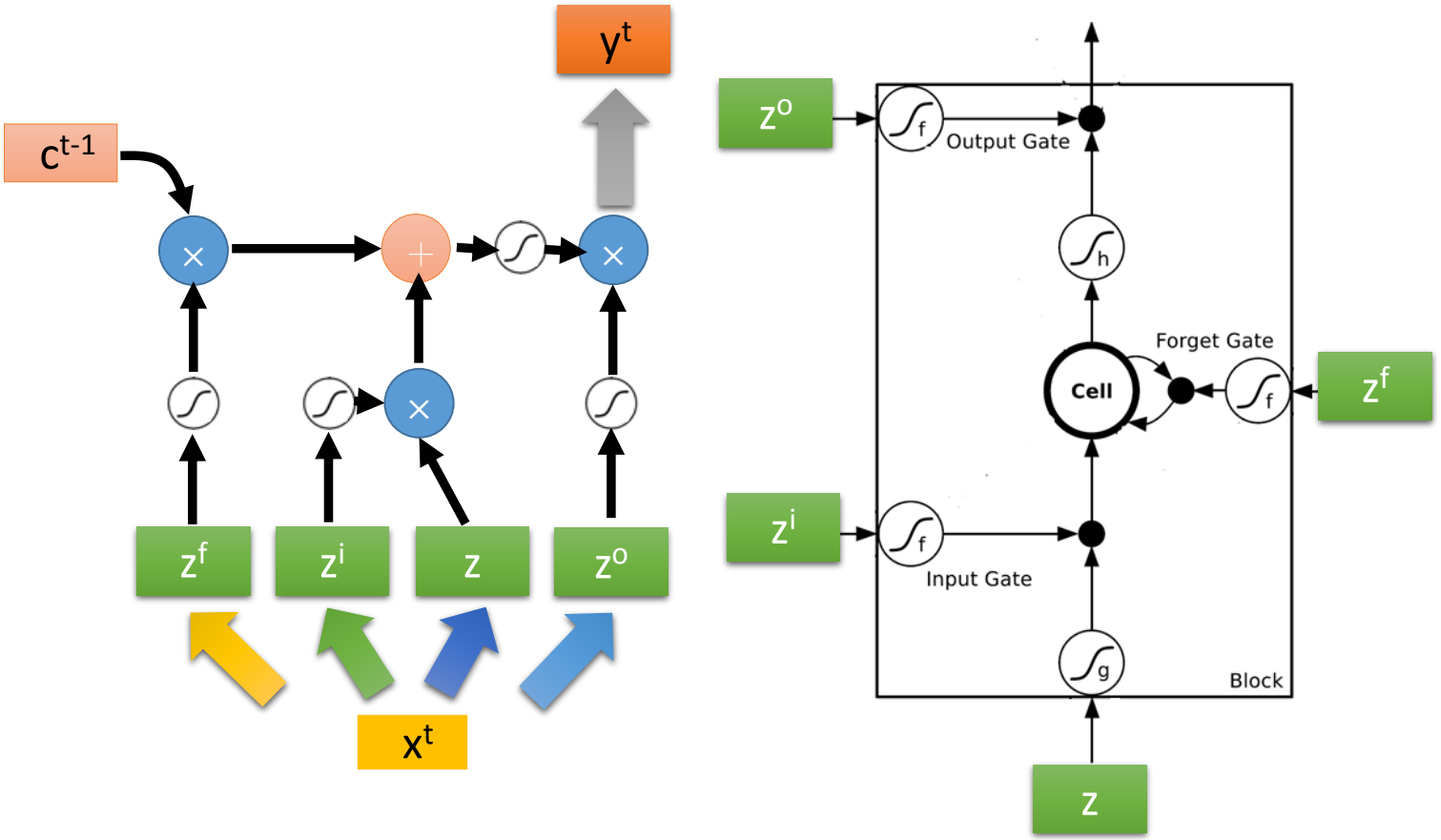
Mimic open and close gate

$$c' = g(z)f(z_i) + cf(z_f)$$

LSTM

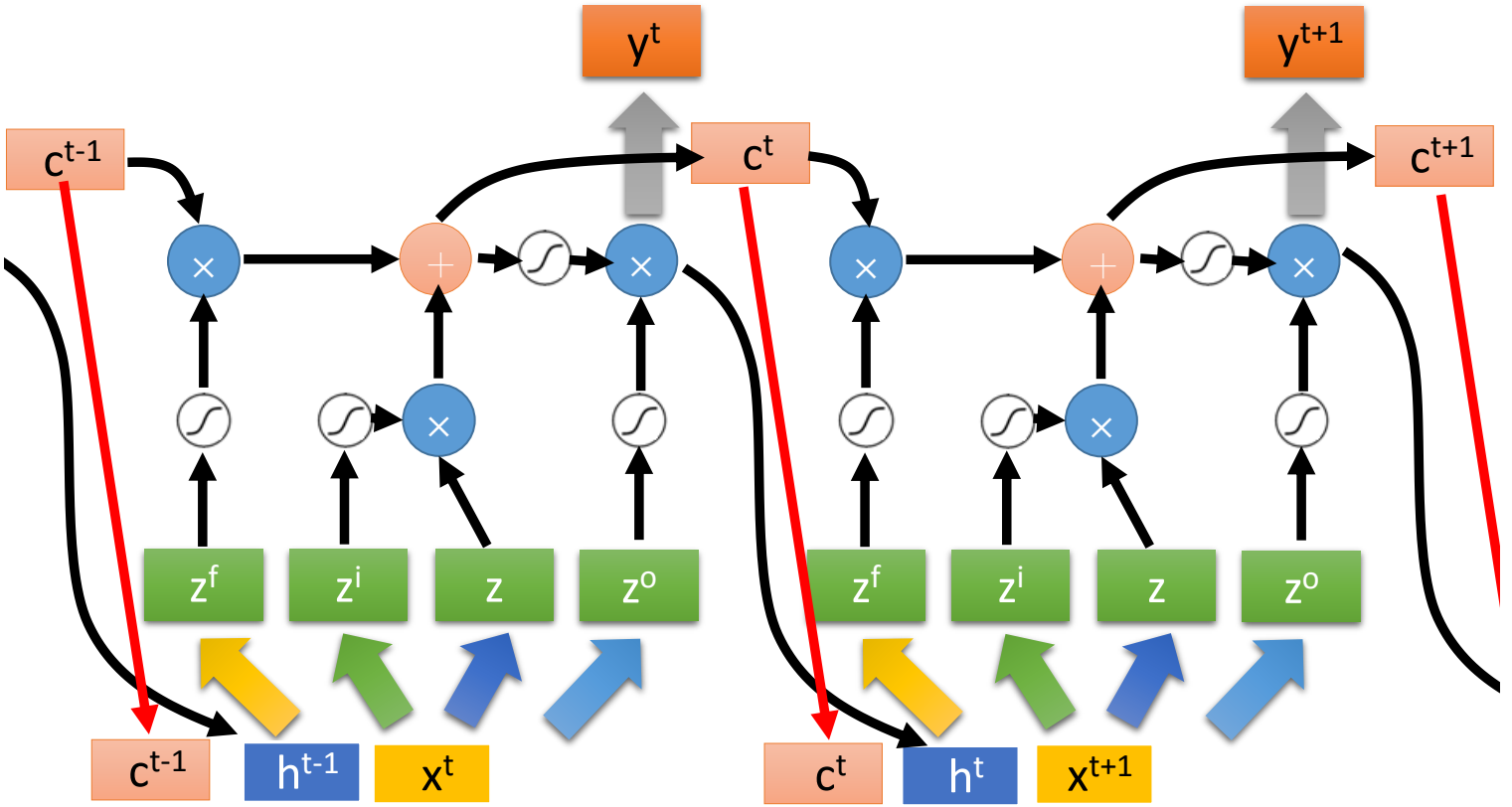

c^{t-1}
vector
Output Gate
Forget Gate
Cell
Input Gate
Block
.....
z^f
z^i
z
z^o
4 vectors
x^t

LSTM

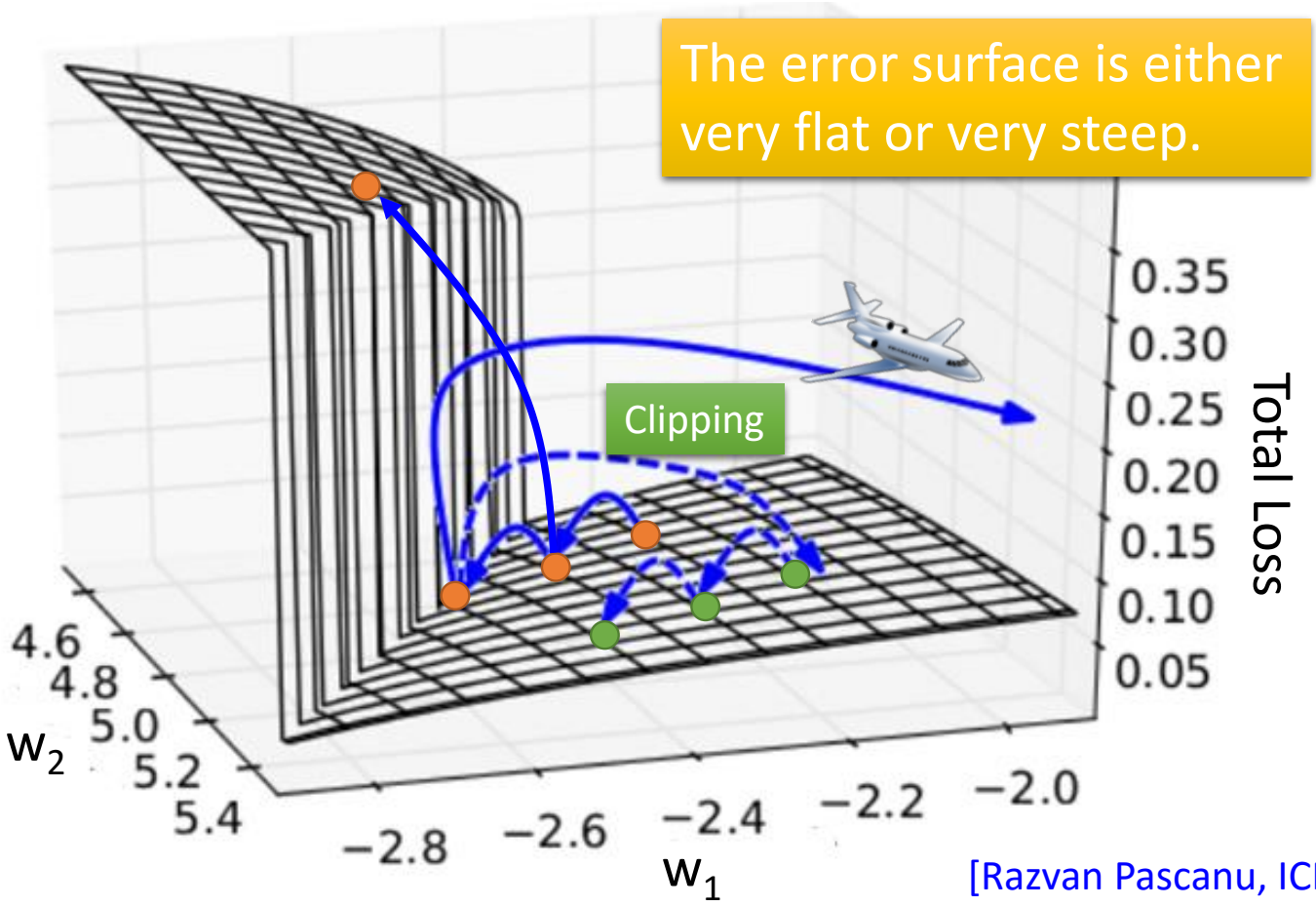

y^t
c^{t-1}
z^f
z^i
z
z^o
x^t
z^o
Output Gate
Forget Gate
Cell
z^f
z^i
Input Gate
Block
z

LSTM

Extension: “peephole”



Gradient Vanishing & Gradient Explode



[Razvan Pascanu, ICML'13]

Gradient Vanishing & Gradient Explode

- Why:

$w = 1 \Rightarrow y^{1000} = 1$

$w = 1.01 \Rightarrow y^{1000} \approx 20000$

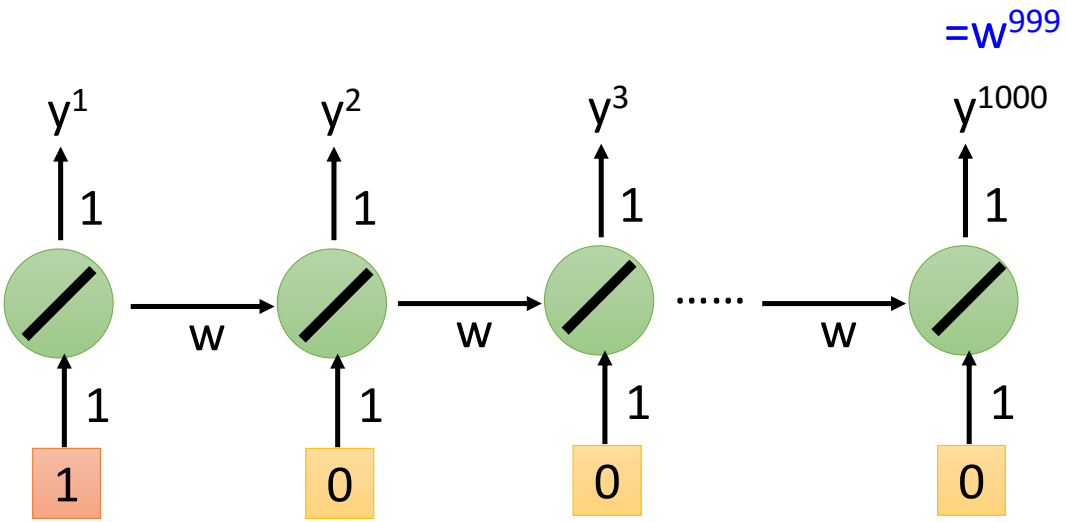
Large $\partial L/\partial w$ → Small Learning rate?

$w = 0.99 \Rightarrow y^{1000} \approx 0$

$w = 0.01 \Rightarrow y^{1000} \approx 0$

small $\partial L/\partial w$ → Large Learning rate?

Toy Example



Gradient Vanishing & Gradient Explode

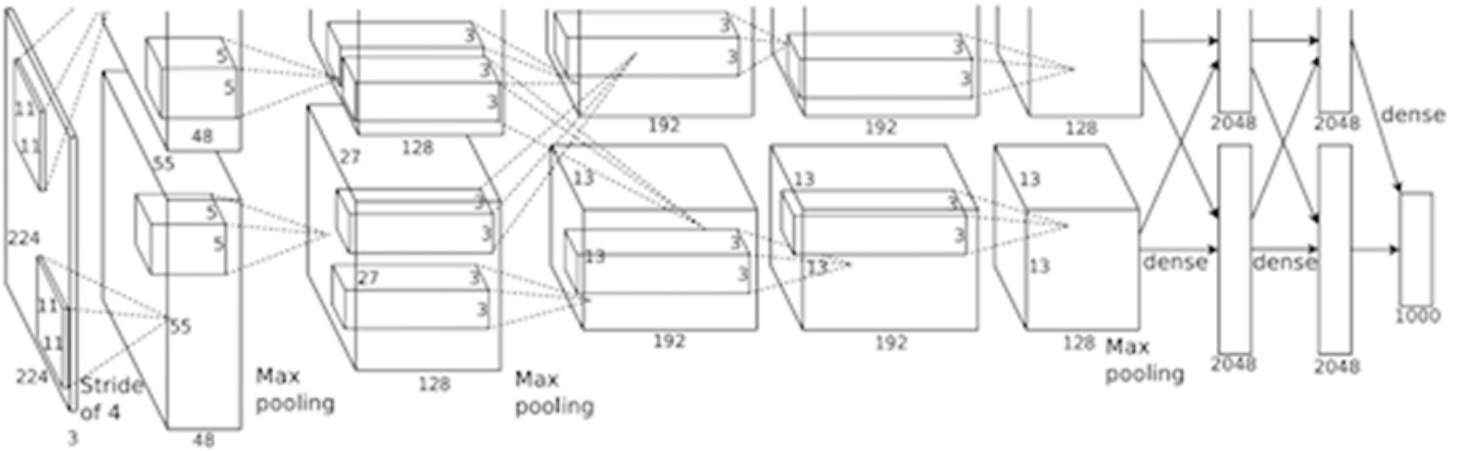
- LSTM can deal with gradient vanishing. No gradient vanishing if forget gate is opened.
- Clipping to deal with gradient explode.

Outline

- Introduction
- Linear Regression
- Classification
- Neural Networks
- Back-Propagation
- Training DNN
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN) & LSTM
- Others

Ultra Deep Network

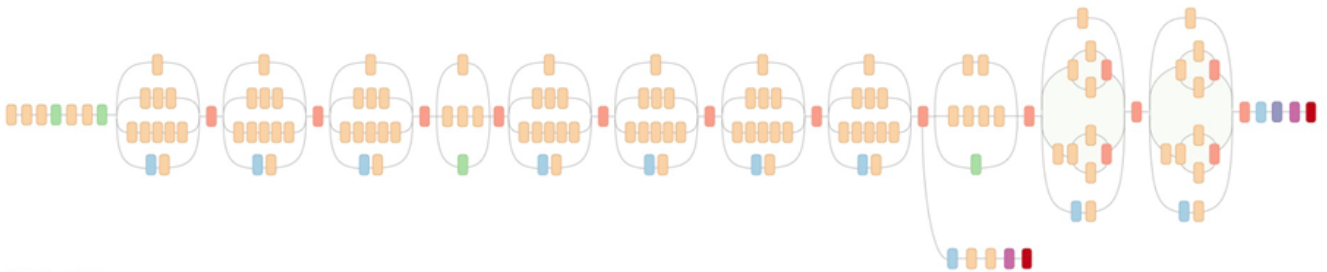
- AlexNet: top5 error rate 15.4%



AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

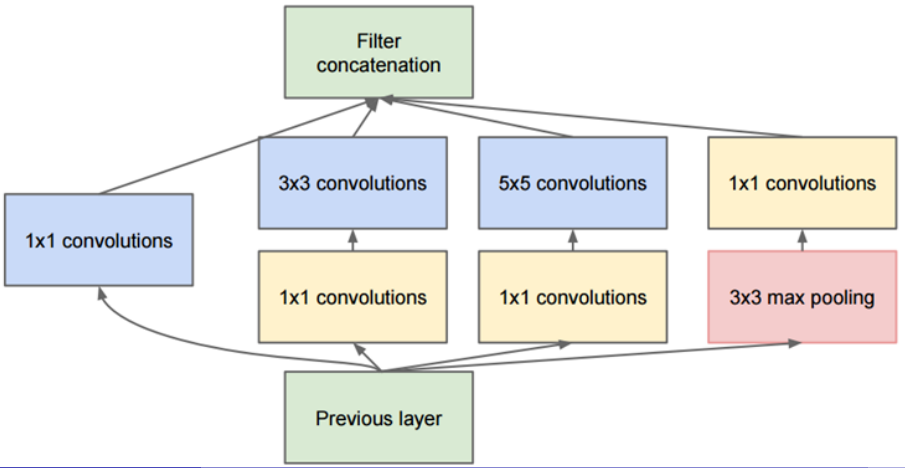
Ultra Deep Network

- GoogleNet: top5 error rate 6.7%



Another view of GoogLeNet's architecture.

- Full inception module

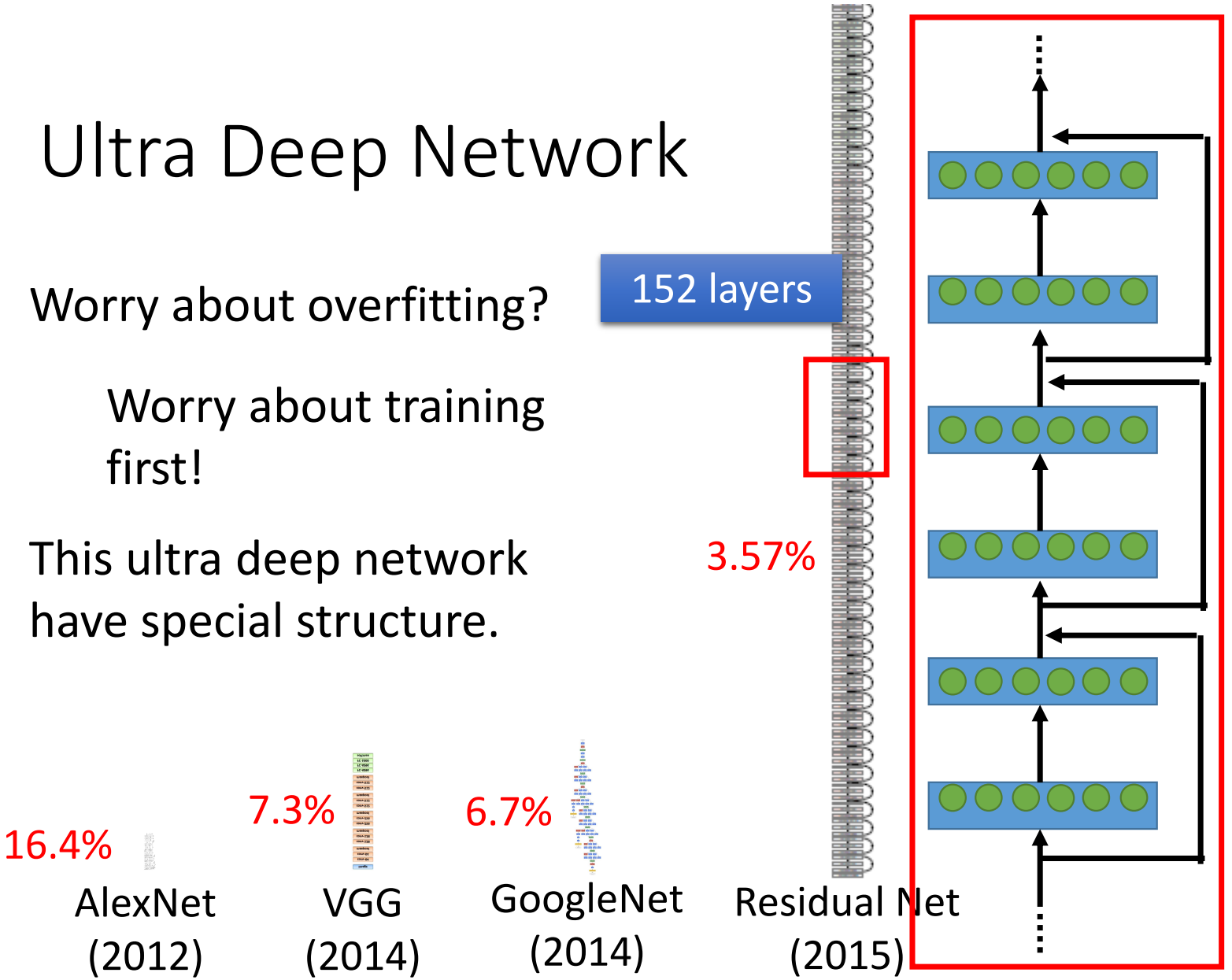


Ultra Deep Network

Worry about overfitting?

Worry about training first!

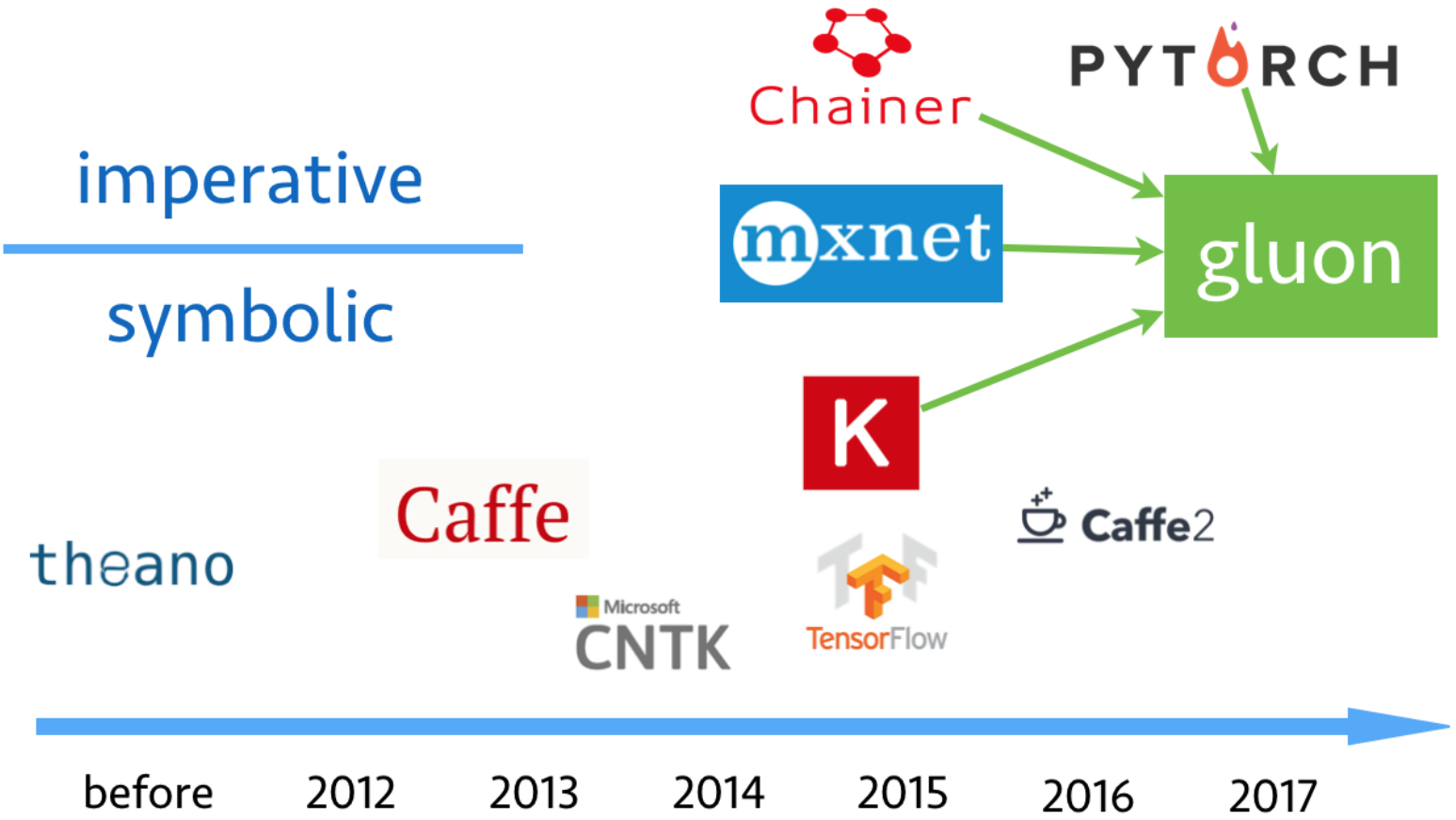
This ultra deep network have special structure.



Others

- PCA
- Auto-encoder: compression + feature + data visualization
- Clustering: K-NN, K-means, K-means++, mean-shift clustering, DB-SCAN, Hierarchical clustering
- Transfer learning: using existing networks
- Reinforcement learning
- Places Demo
- GoogleNet

Build a DNN


imperative
symbolic
Chainer
PYTORCH
mxnet
gluon
K
Caffe
theano
Caffe2
Microsoft
CNTK
TensorFlow
before
2012
2013
2014
2015
2016
2017

Build a DNN

- Download and install Anaconda
- Build a environment or just use the base environment: conda create -n tensorflow pip python=3.6
- Activate the env: source activate tensorflow
- Install tensorflow: pip install tensorflow

Thank you!