

# Machine Learning, Spring 2020

## Deep Learning Part I

Instructor: Prof. Yi Fang

yfang@nyu.edu

Reading Assignment: Chapter 10,11,12 & 13

Python tutorial: <http://learnpython.org/>

TensorFlow tutorial: <https://www.tensorflow.org/tutorials/>

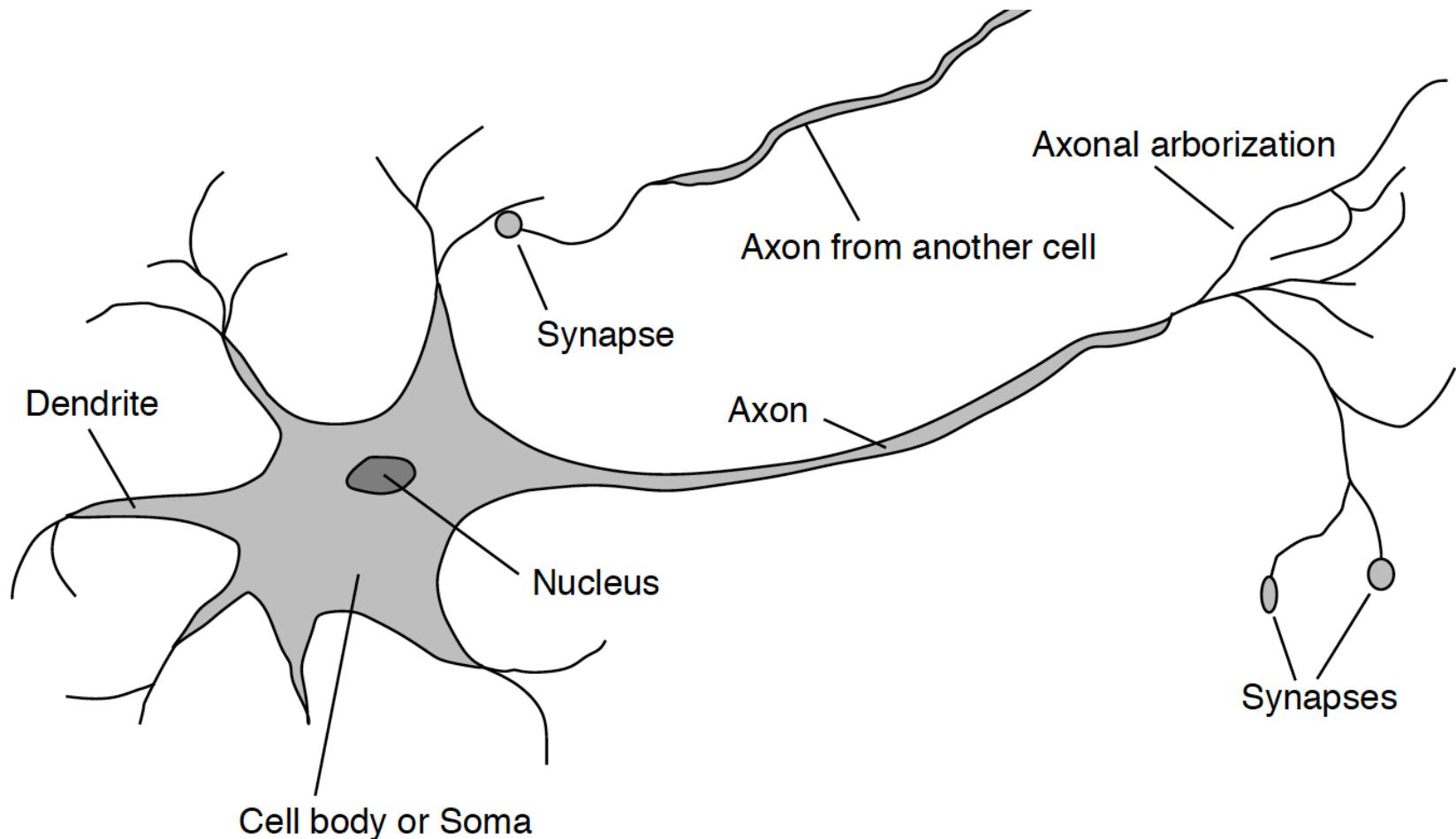
PyTorch tutorial: <https://pytorch.org/tutorials/>

# Overview

- Human Brain
- Neural Network
- Perceptrons
- Multilayer Perceptrons
- Convolution Neural Network
  - Overview architecture
  - Convolution operation
  - Convolution layer
  - Pooling layer
  - Fully connection layer

# Human Brain

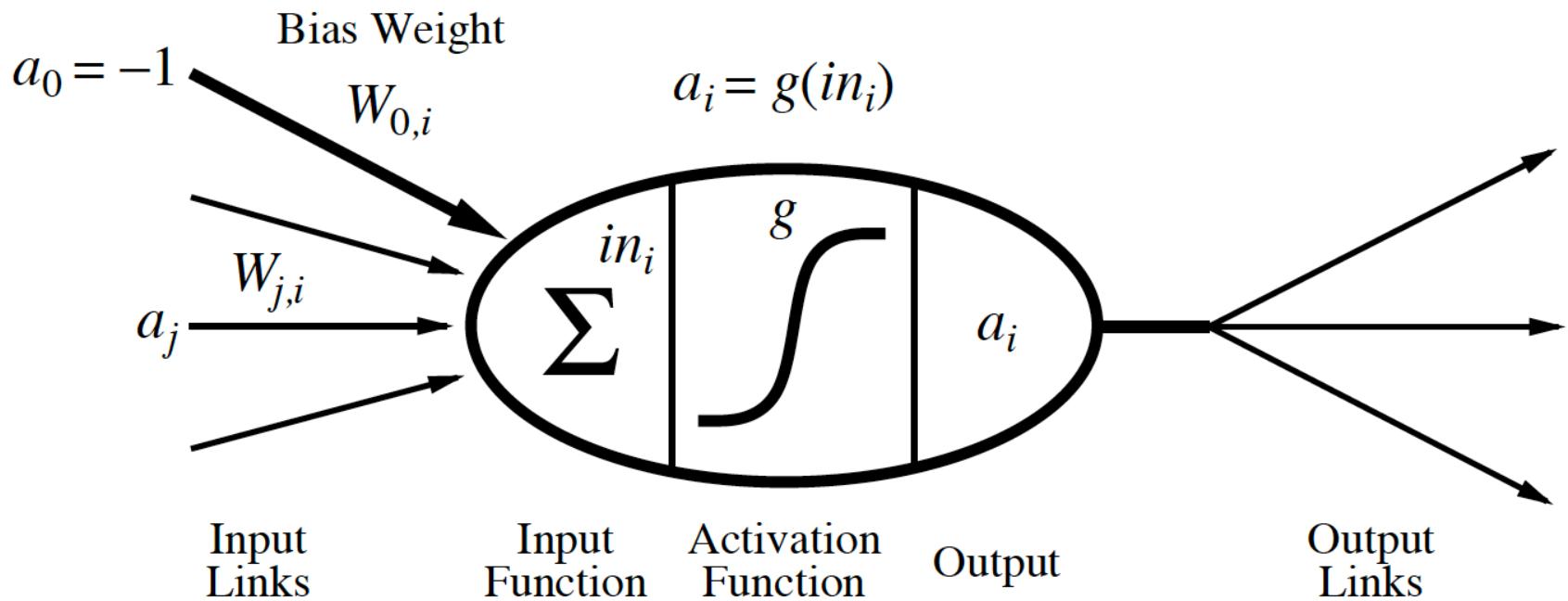
- $10^{11}$  neurons of > 20 types,  $10^{14}$  synapses, 1ms-10ms cycle time
- Signals are noisy “spike trains” of electrical potential



# Neurons

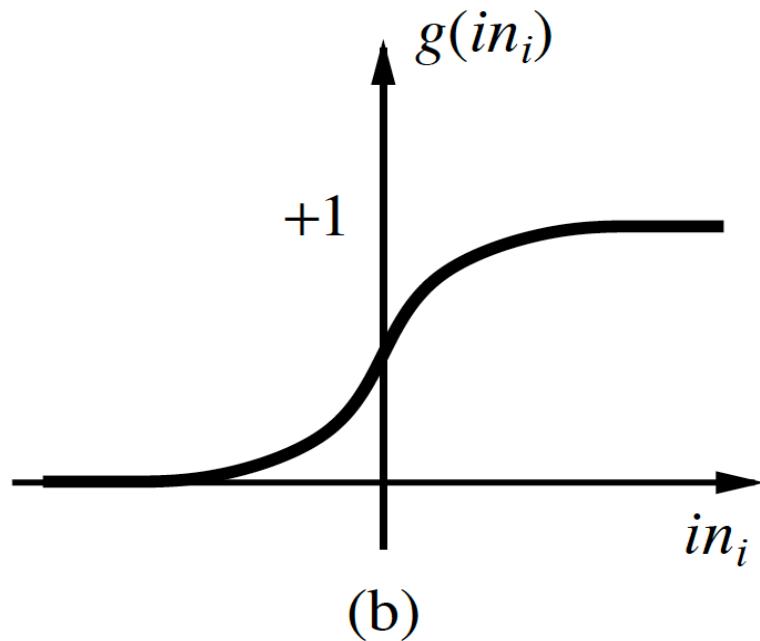
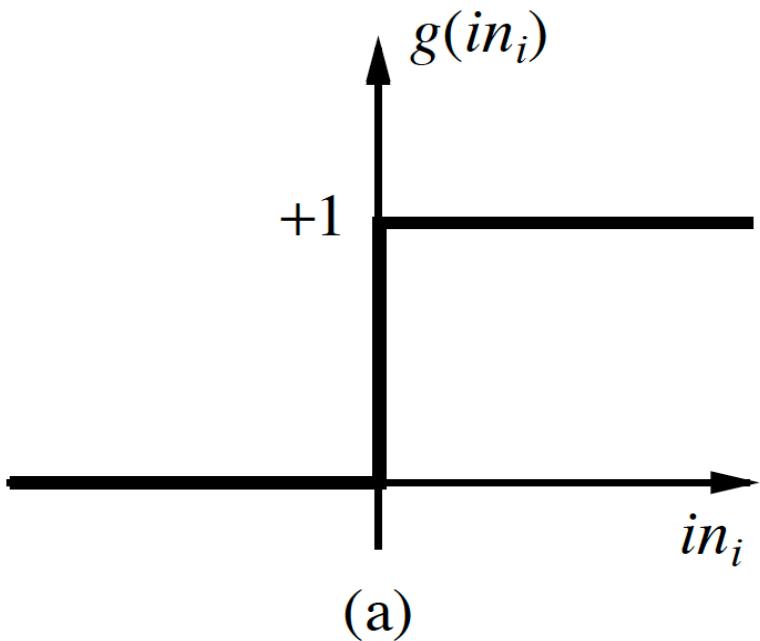
- A gross oversimplification of real neurons

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



# Human Brain

- Activation function



(a) is a step function or threshold function

(b) is a sigmoid function  $1/(1 + e^{-x})$

Changing the bias weight  $W_{0,i}$  moves the threshold location

# Network structure

- Feed-forward Networks
  - Single-layer perceptrons
  - Multi-layer perceptrons
- Recurrent Network
  - Hopfield networks
  - Boltzmann machines

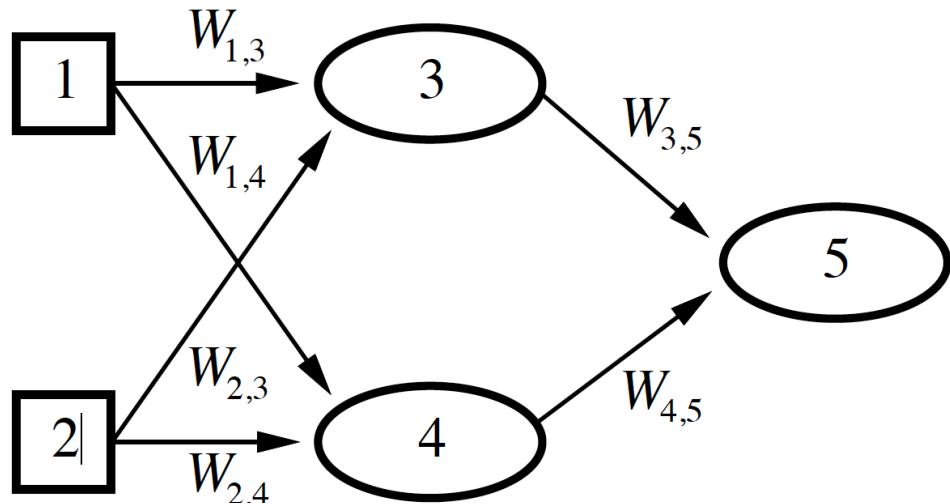
Feed-forward networks implement functions which have no internal state

Recurrent neural nets have directed cycles with delays and have internal state (like ip-ops)

# Feed-forward Networks

- Feed-forward network = a parameterized family of nonlinear functions
- Adjusting weights changes the function

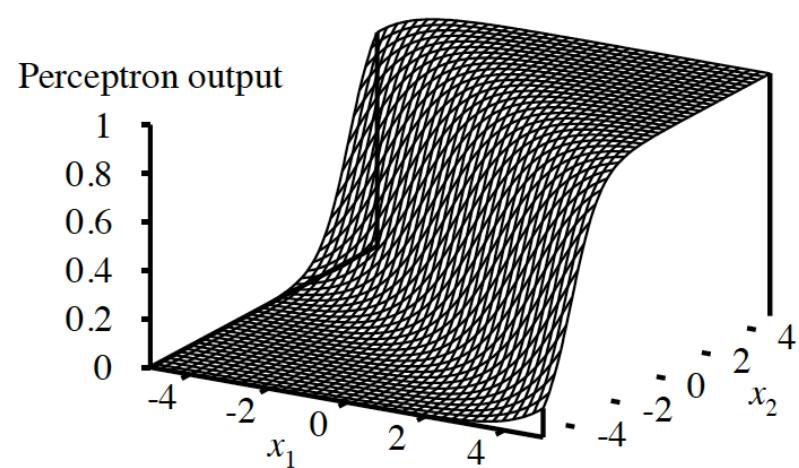
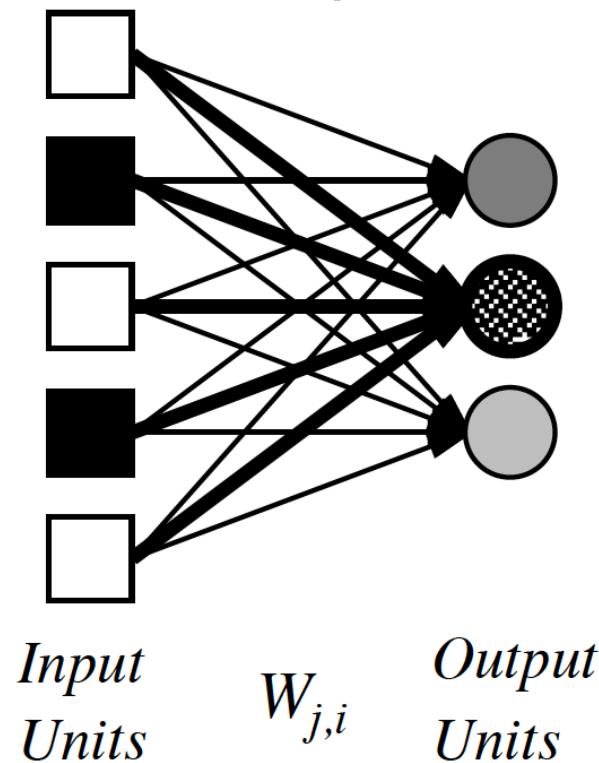
An easy example:



$$\begin{aligned}a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\&= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))\end{aligned}$$

# Single-layer perceptrons

- Output units all operate separately and no shared weights
- Adjusting weights moves the location, orientation, and steepness of cliff



# Perceptron learning

Learn by adjusting weights to reduce **error** on training set

The **squared error** for an example with input  $\mathbf{x}$  and true output  $y$  is

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

Perform optimization search by gradient descent:

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j} (y - g(\sum_{j=0}^n W_j x_j)) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

Simple weight update rule:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

# Multilayer perceptrons

- Layers are usually fully connected
- Numbers of hidden units typically chosen by hand

Output units

$a_i$

$W_{j,i}$

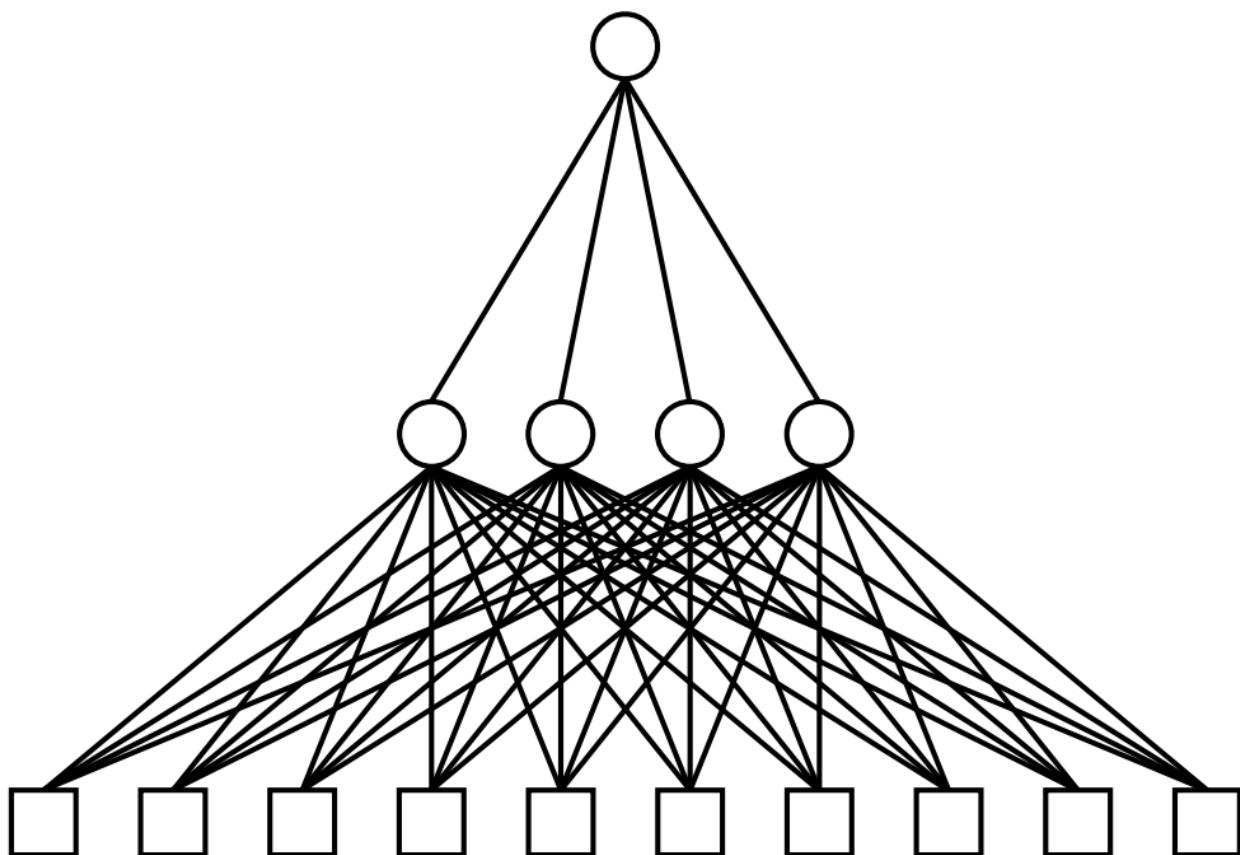
Hidden units

$a_j$

$W_{k,j}$

Input units

$a_k$



# Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where  $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j .$$

# Back-propagation example

- Loss is defined as:

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2 ,$$

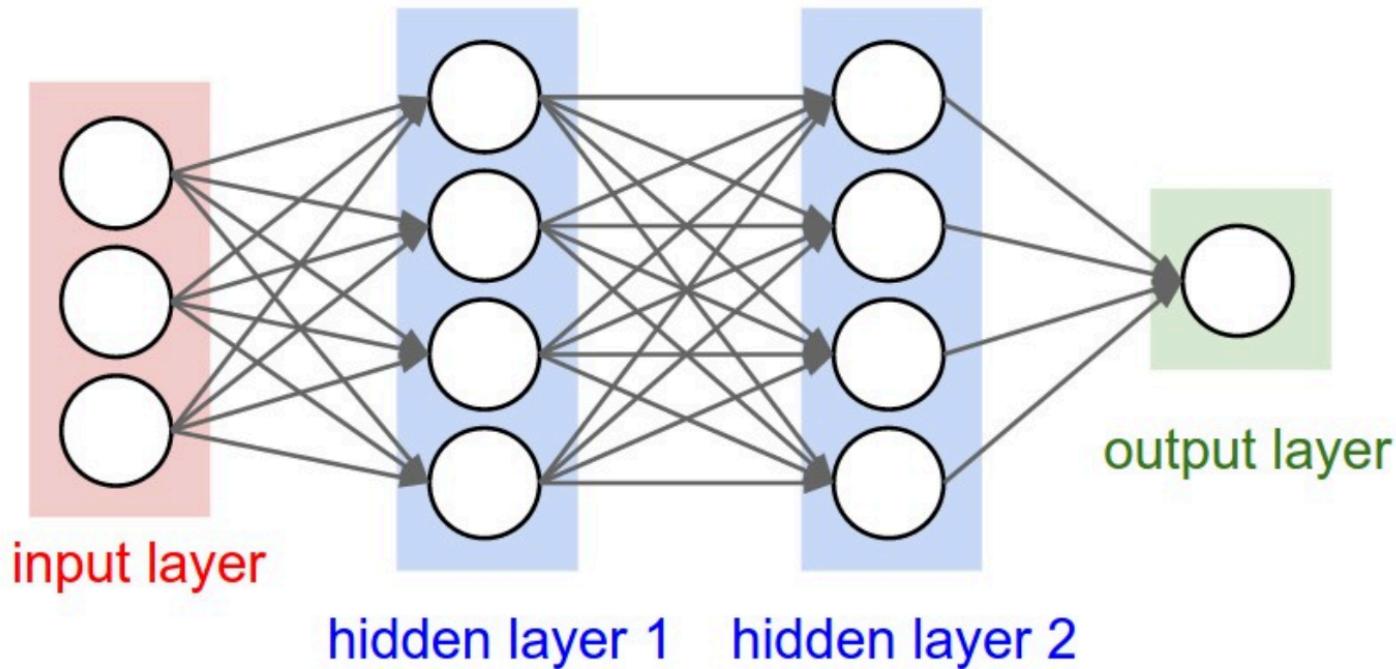
- where the sum is over the nodes in the output layer

$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i) g'(in_i) \frac{\partial}{\partial W_{j,i}} \left( \sum_j W_{j,i} a_j \right) \\ &= -(y_i - a_i) g'(in_i) a_j = -a_j \Delta_i\end{aligned}$$

# Convolution Neural Network (CNN)

# Why CNN?

- A regular 3-layer Neural Network.

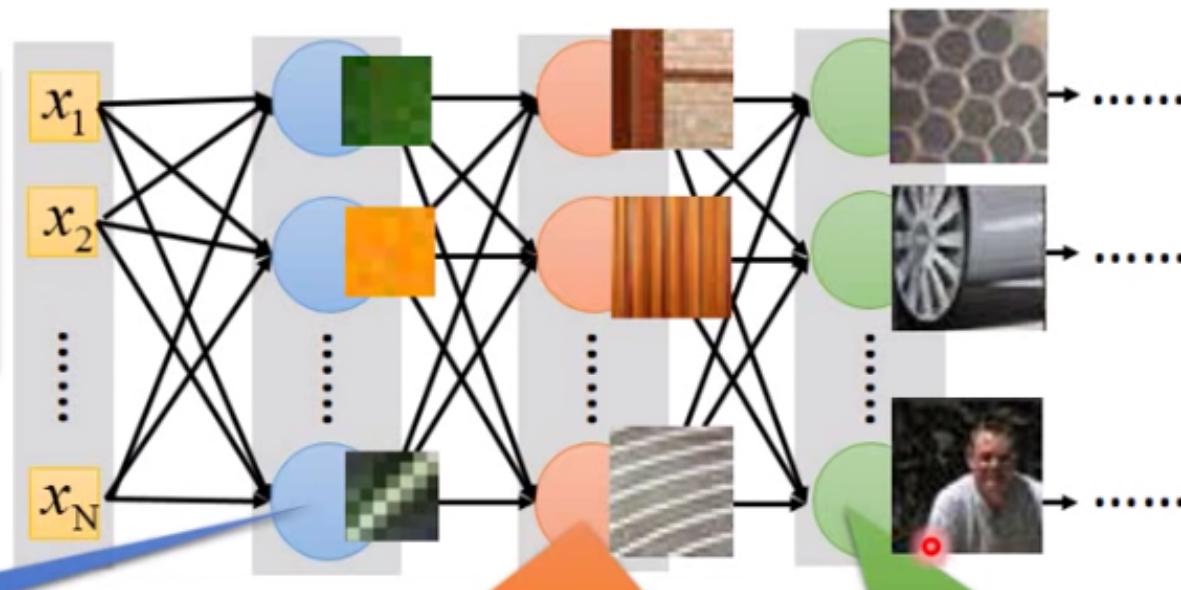


# Why CNN for Image?

[Zeiler, M. D., ECCV 2014]



Represented  
as pixels



The most basic  
classifiers

Use 1<sup>st</sup> layer as module  
to build classifiers

Use 2<sup>nd</sup> layer as  
module .....

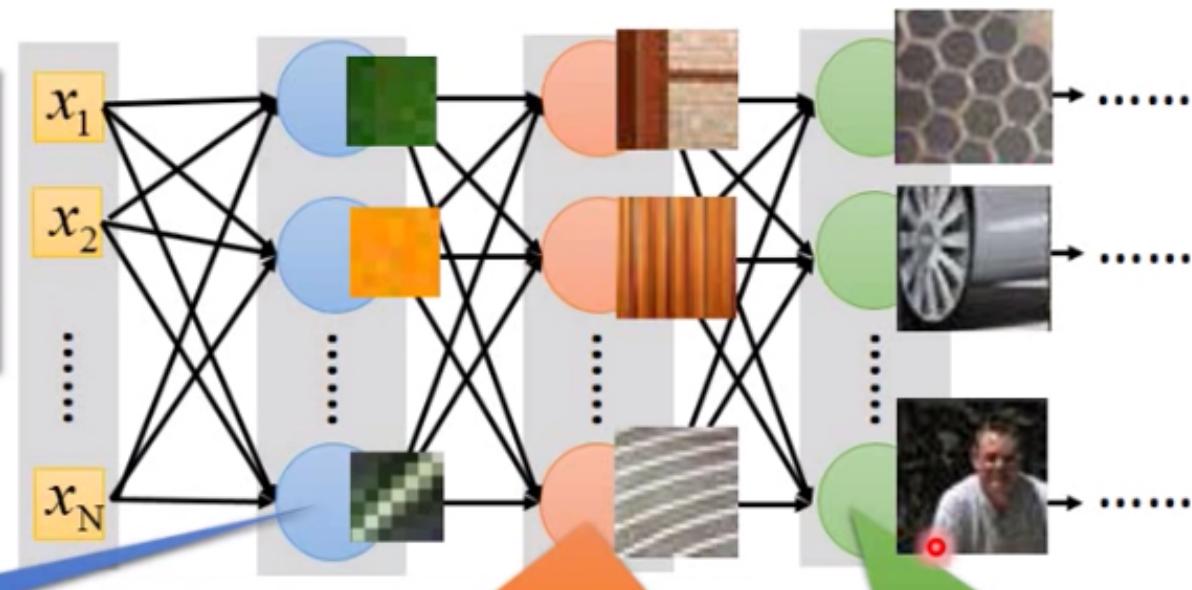
# Redundant Parameters?

## Why CNN for Image?

[Zeiler, M. D., ECCV 2014]



Represented  
as pixels



The most basic  
classifiers

Use 1<sup>st</sup> layer as module  
to build classifiers

Use 2<sup>nd</sup> layer as  
module .....

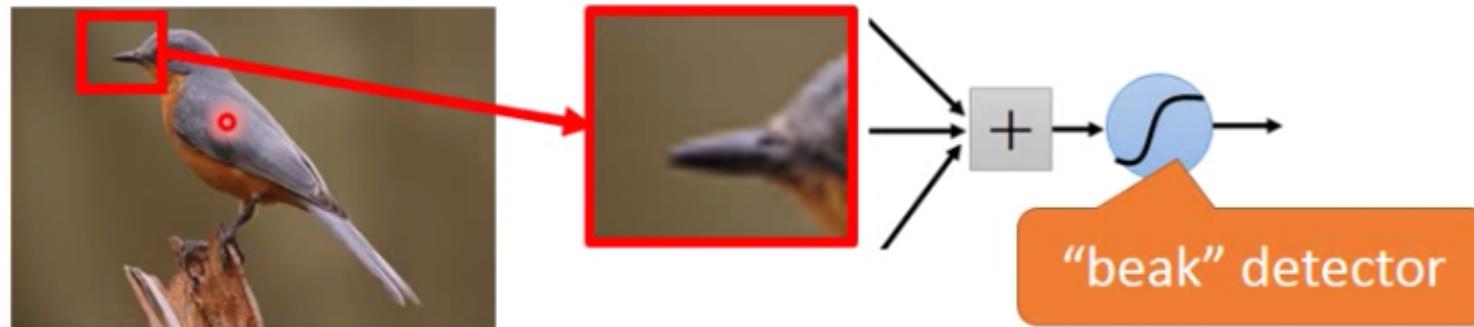
# Can the Network be simplified?

- Focus on local region
- Insensitive to location
- Sub-sample operator

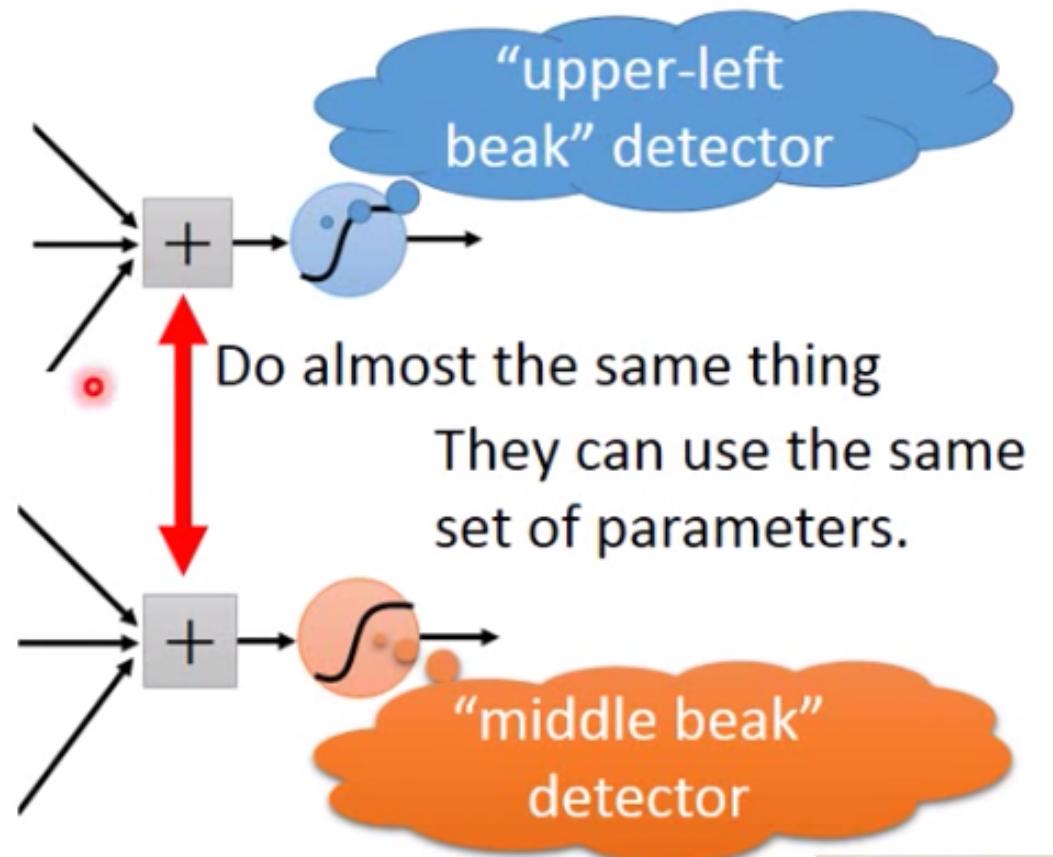
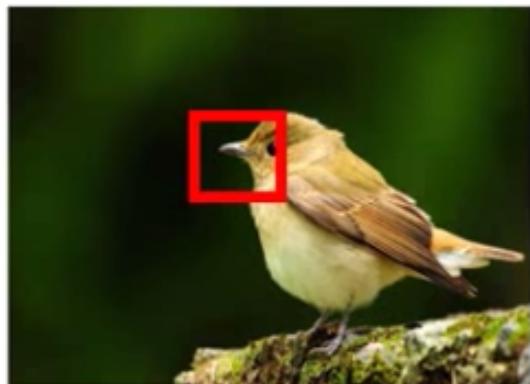
# Focus on local region

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

Connecting to small region with less parameters



# Inensitive to location



# Sub-sampling

- Subsampling the pixels will not change the object



We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

# The whole CNN



## Property 1

- Some patterns are much smaller than the whole image

## Property 2

- The same patterns appear in different regions.

## Property 3

- Subsampling the pixels will not change the object

Convolution

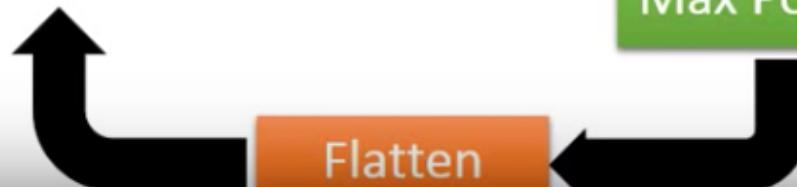
Max Pooling

Convolution

Max Pooling

Flatten

Can repeat many times



# CNN – Convolution

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

Filter 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

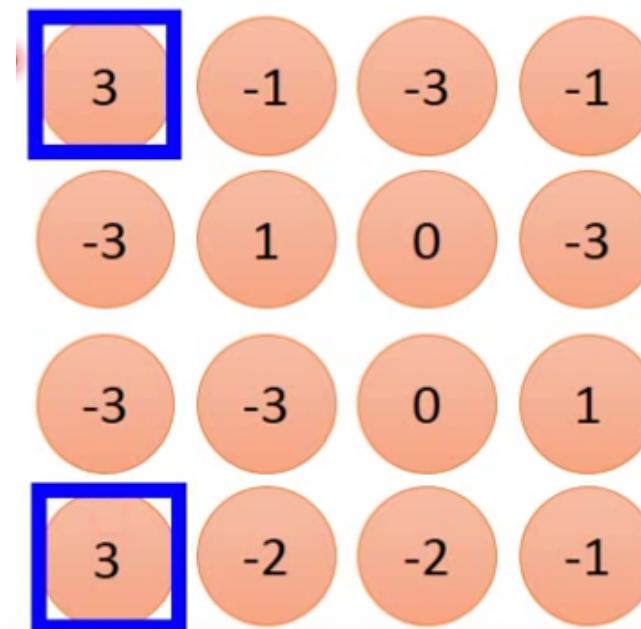
stride=1

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



Property 2

# CNN – Convolution

stride=1

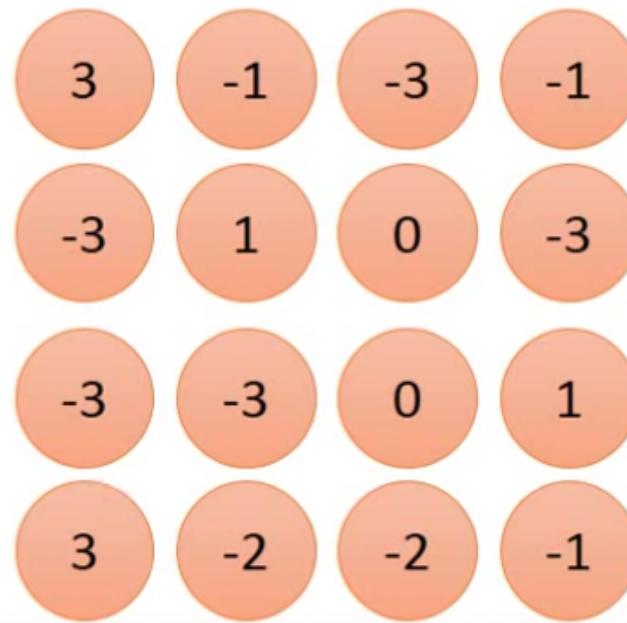
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

Filter 2

Do the same process for every filter



# CNN – Convolution

stride=1

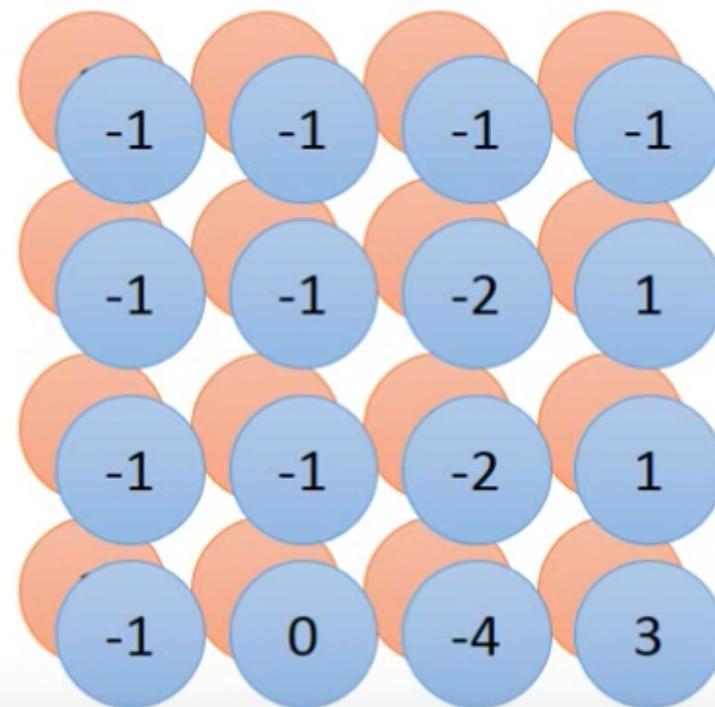
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

Filter 2

Do the same process for every filter



# CNN – Convolution

stride=1

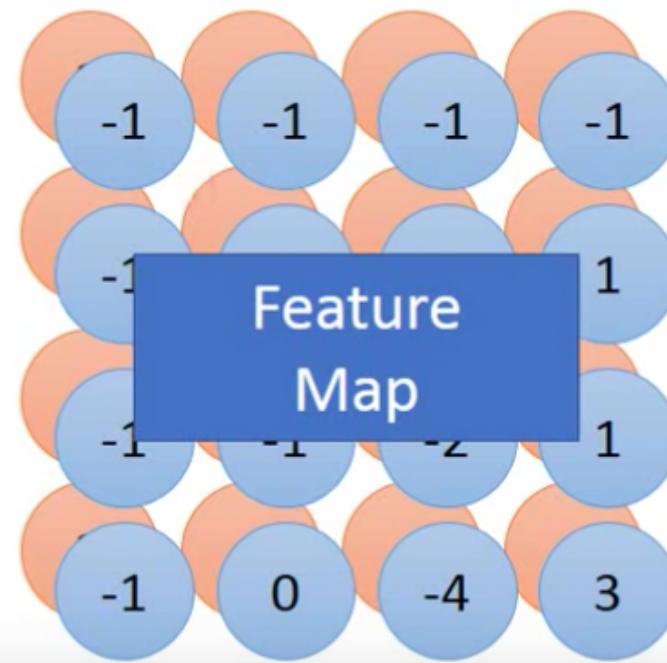
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

Filter 2

Do the same process for every filter



# References

- CNN: ML Lecture 10: Convolutional Neural Network by Hongyi Li (youtube)
- Chapter 20 from Berkeley Artificial Intelligence: A Modern Approach (link: <http://aima.eecs.berkeley.edu/>)
- Module 2: Convolutional Neural Networks from course notes of Stanford CS231n (link: <http://cs231n.github.io/convolutional-networks/>)
- Full connection blog (link:  
<https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection>)
- <http://aima.eecs.berkeley.edu/>
- <http://cs231n.github.io/convolutional-networks/>