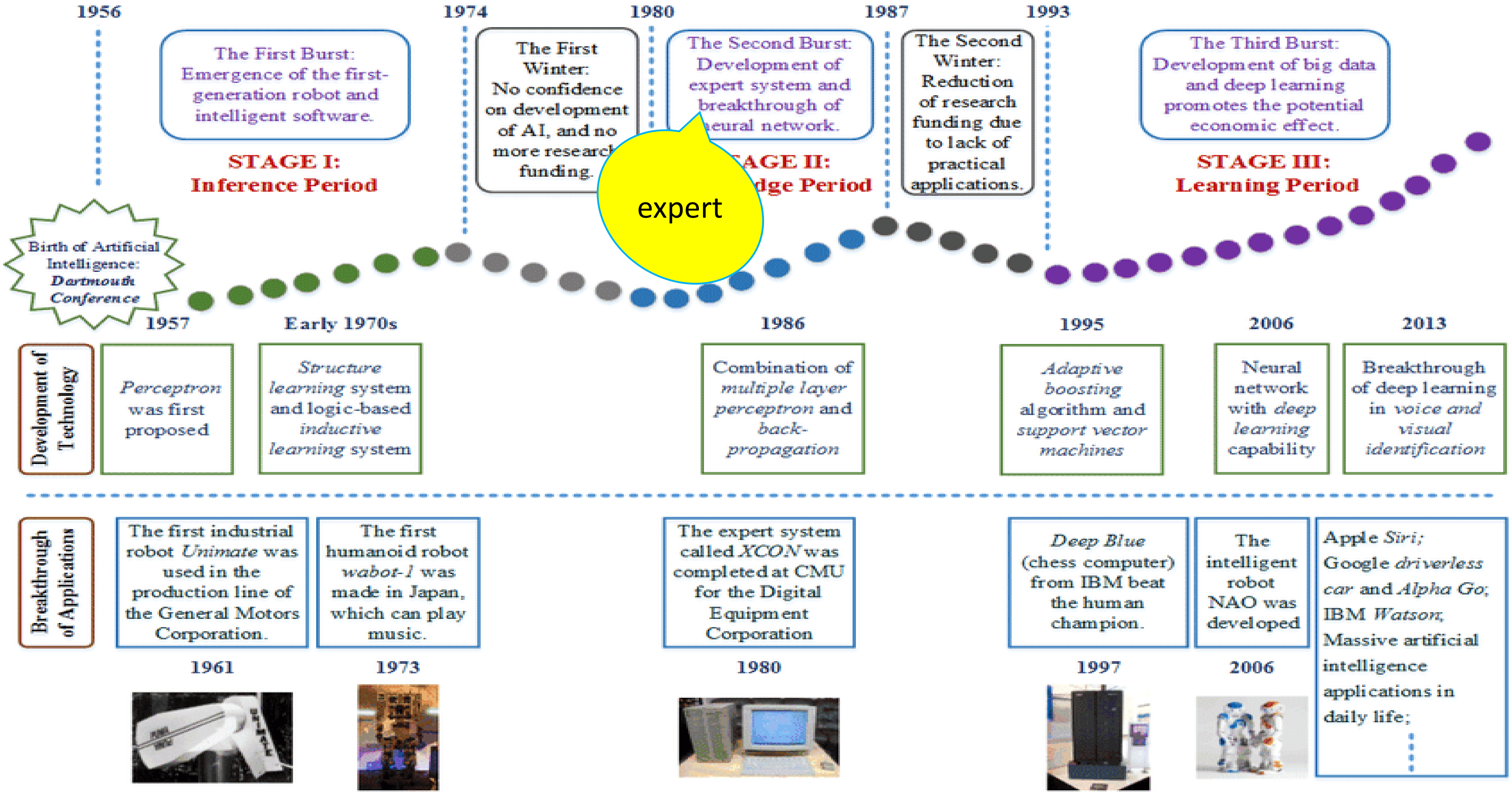
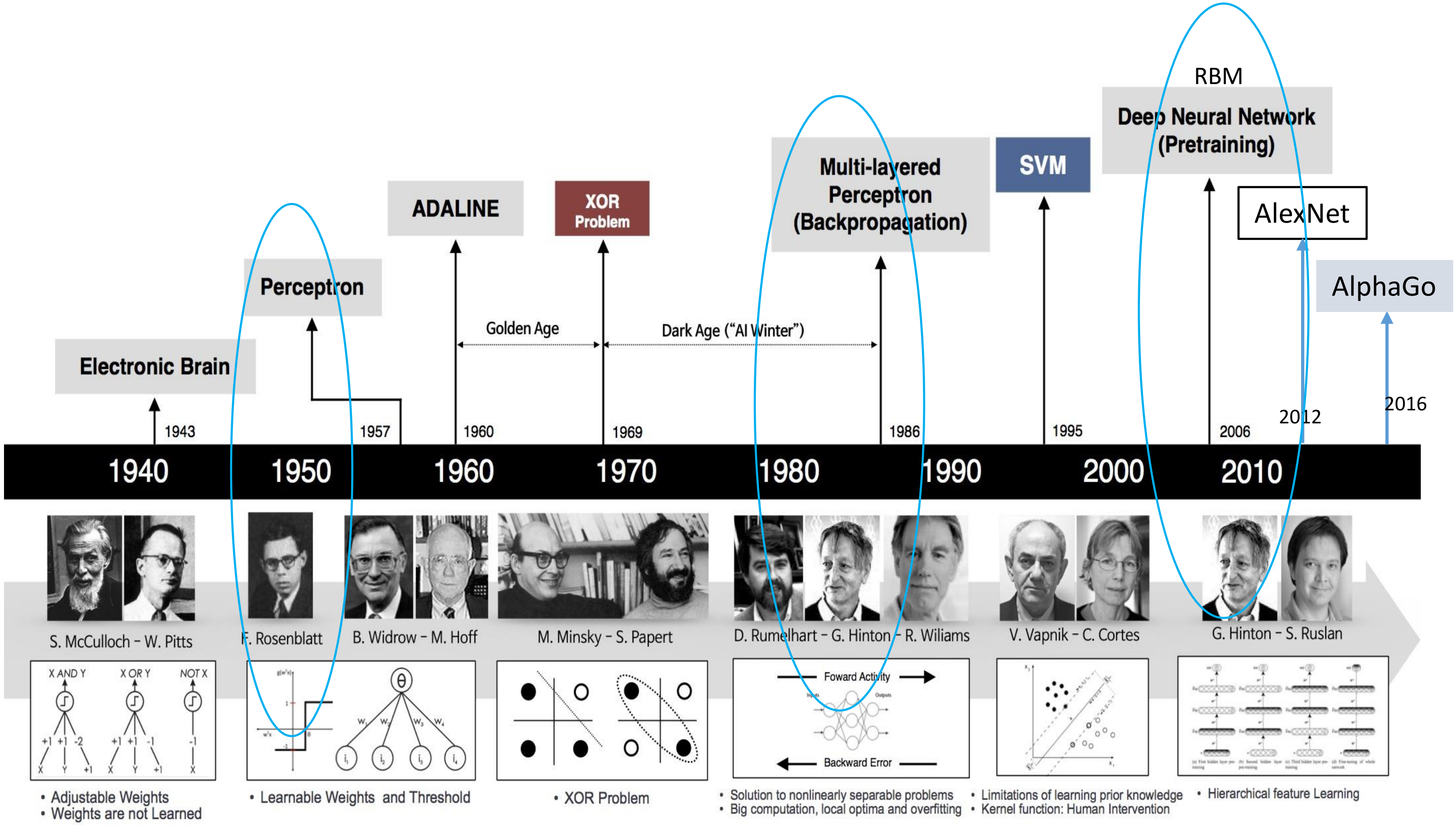
Introduction

Pau-Choo Chung (Julia)

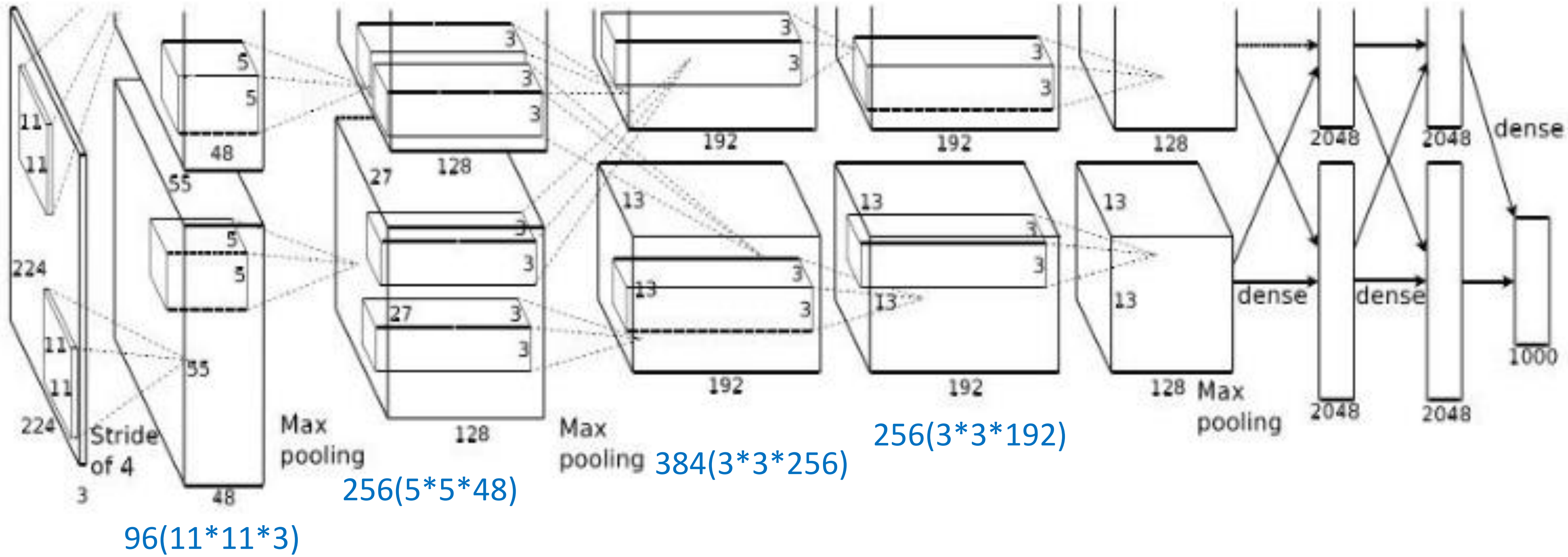

1956
1974
1980
1987
1993
The First Burst: Emergence of the first-generation robot and intelligent software.
STAGE I: Inference Period
The First Winter: No confidence on development of AI, and no more research funding.
The Second Burst: Development of expert system and breakthrough of neural network.
AGE II:
dge Period
expert
The Second Winter: Reduction of research funding due to lack of practical applications.
The Third Burst: Development of big data and deep learning promotes the potential economic effect.
STAGE III: Learning Period
Birth of Artificial Intelligence: Dartmouth Conference
1957
Early 1970s
1986
1995
2006
2013
Development of Technology
Perceptron was first proposed
Structure learning system and logic-based inductive learning system
Combination of multiple layer perceptron and back-propagation
Adaptive boosting algorithm and support vector machines
Neural network with deep learning capability
Breakthrough of deep learning in voice and visual identification
Breakthrough of Applications
The first industrial robot Unimate was used in the production line of the General Motors Corporation.
1961
The first humanoid robot wabot-1 was made in Japan, which can play music.
1973
The expert system called XCON was completed at CMU for the Digital Equipment Corporation
1980
Deep Blue (chess computer) from IBM beat the human champion.
1997
The intelligent robot NAO was developed
2006
Apple Siri; Google driverless car and Alpha Go; IBM Watson; Massive artificial intelligence applications in daily life;



https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

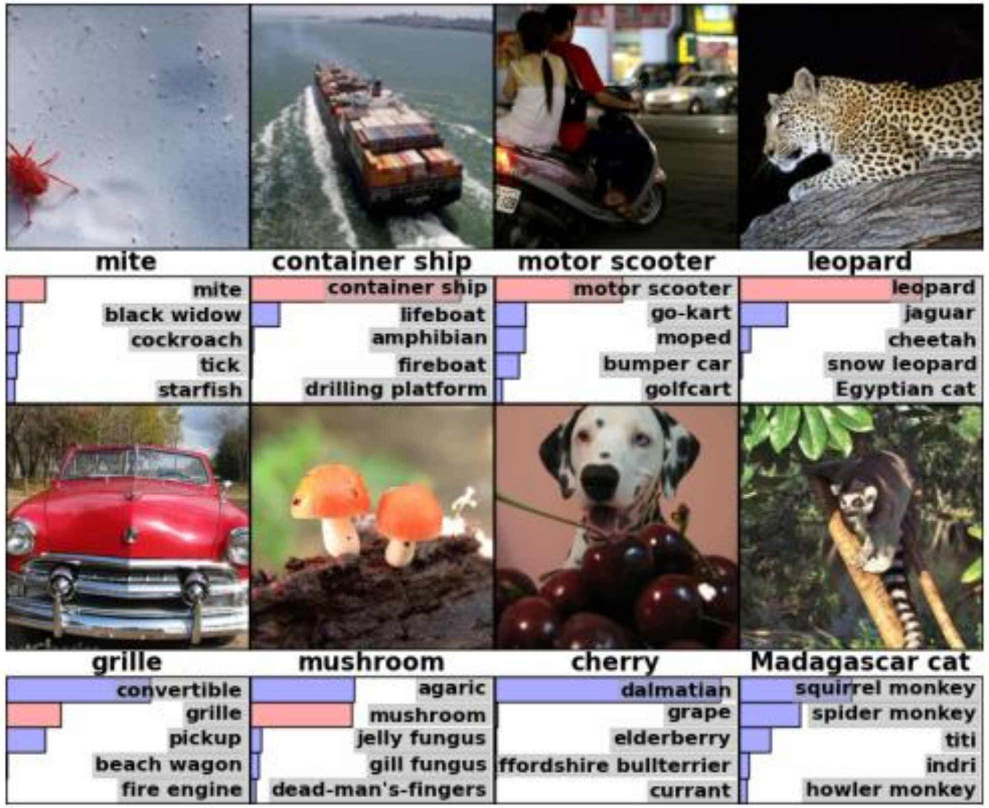
AlexNet

- 5 convolutional layers and 2 fully connected layers
- Max pooling layer after first, second and fifth layers
- Use ReLU as activation function



Source: [ImageNet Classification with Deep Convolutional Neural Networks] (NIPS 2012)

- 1.3 million high-resolution images in the LSVRC-2010(ImageNet Large Scale Visual Recognition Competition)
- GPU implementation of convolutional nets



Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

Source: ImageNet Classification with Deep Convolutional Neural Networks (NIPS 2012)

VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- 11, 13, 16 and 19 layers implementation
- 3*3 and 1*1 kernel sizes for deep model

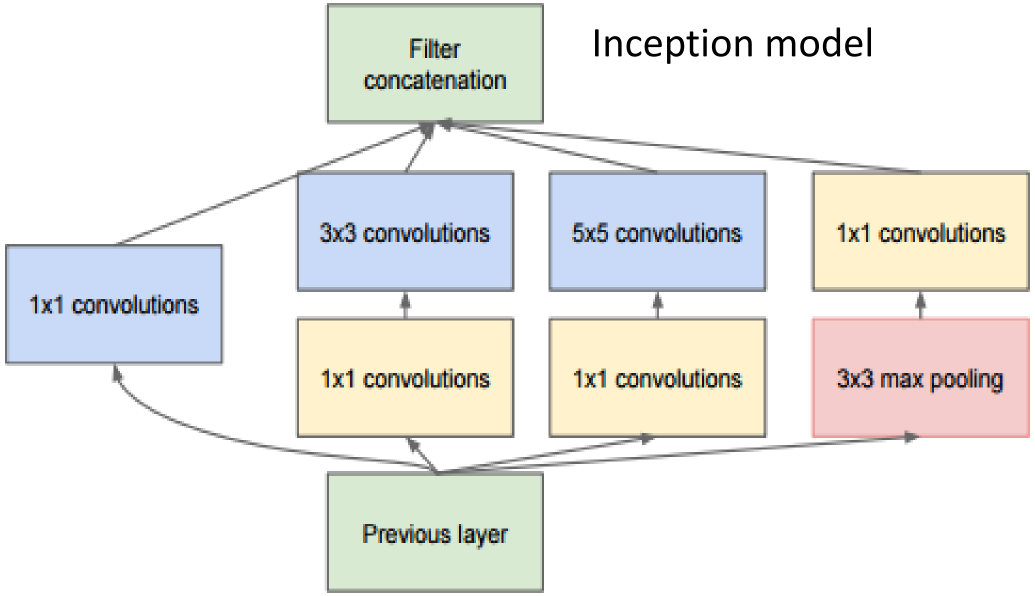
Using smaller masks
The first using 1*1 kernel size

Simplifying networks to afford deeper

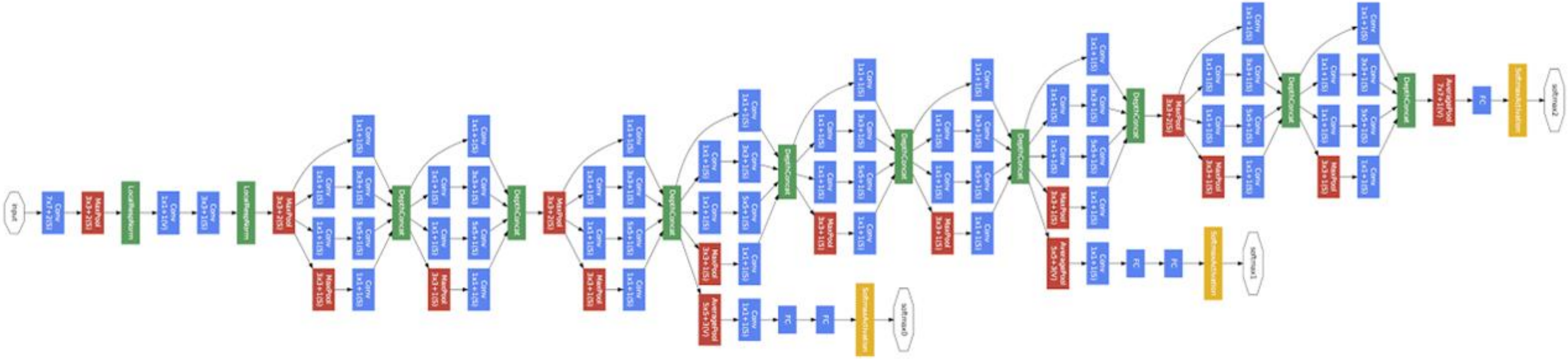
Source: Very Deep Convolutional Networks for Large-Scale Image Recognition (ICLR 2014)

GoogleNet

- Inception
- compute various convolution kernel sizes in same network
- 1*1 convolutions for reducing the number of maps

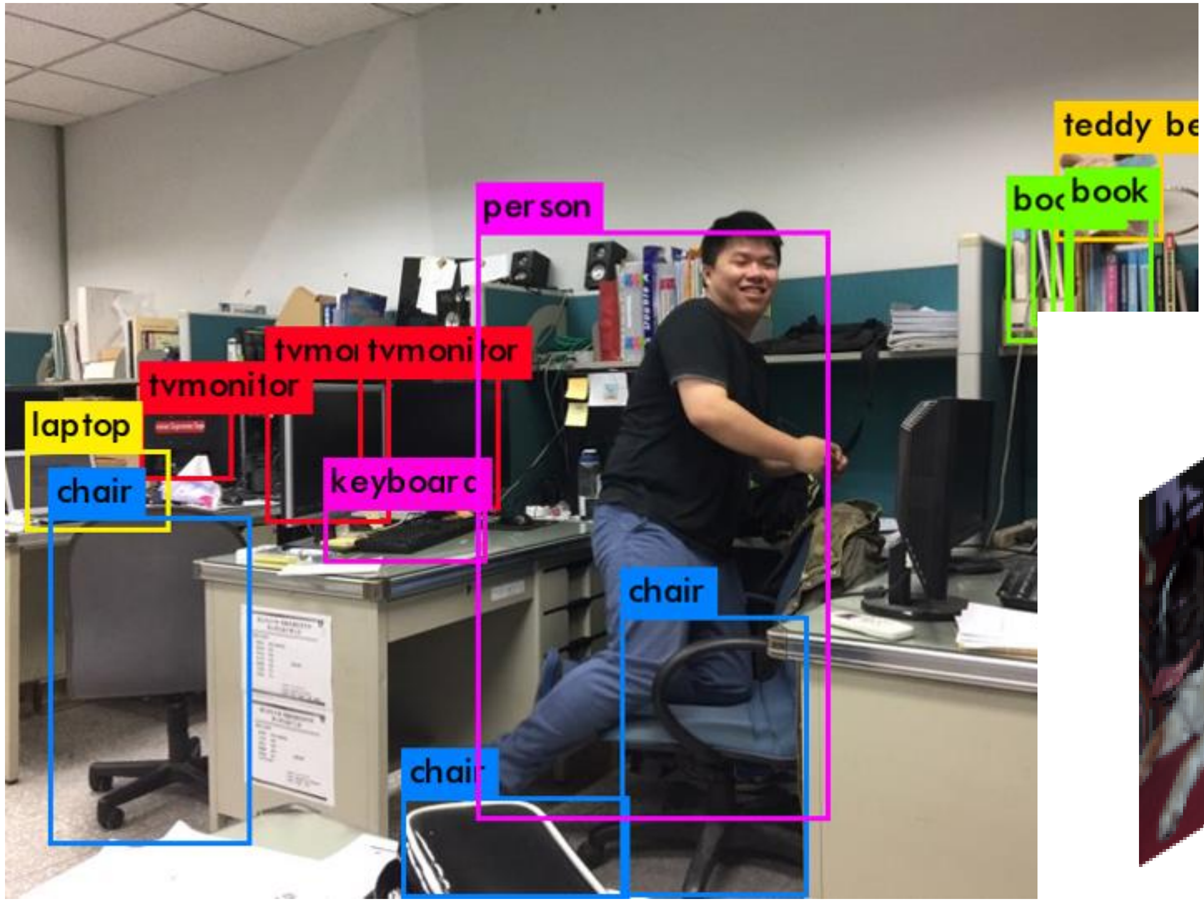


Inception model



Source: Going deeper with convolutions (CVPR 2015)

Object detection (Yolo V2)



Segmentation

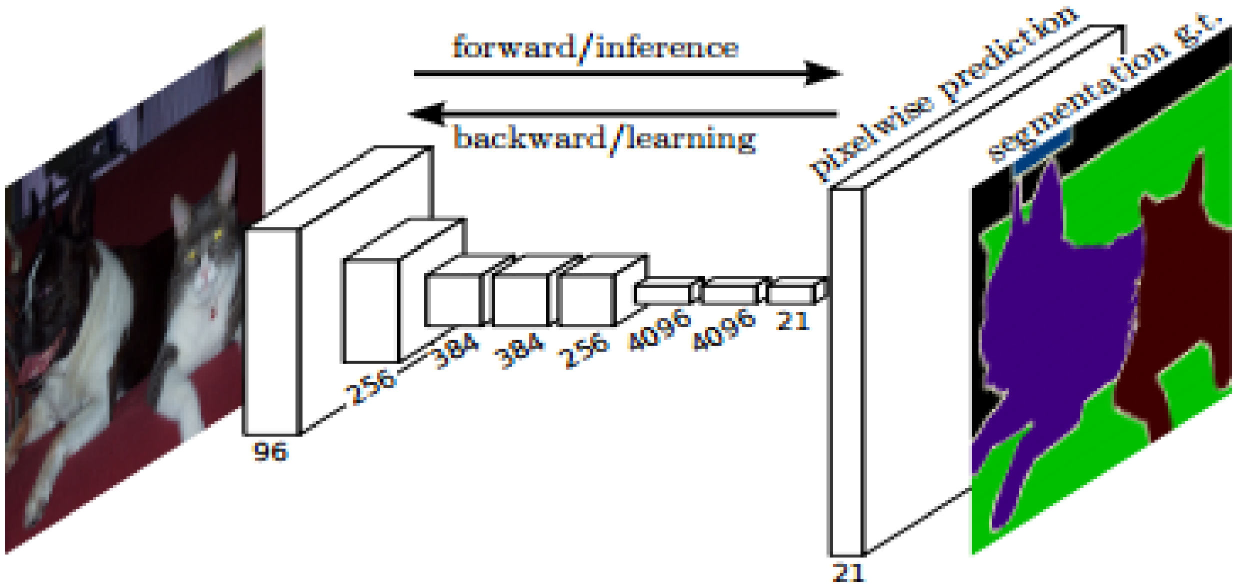
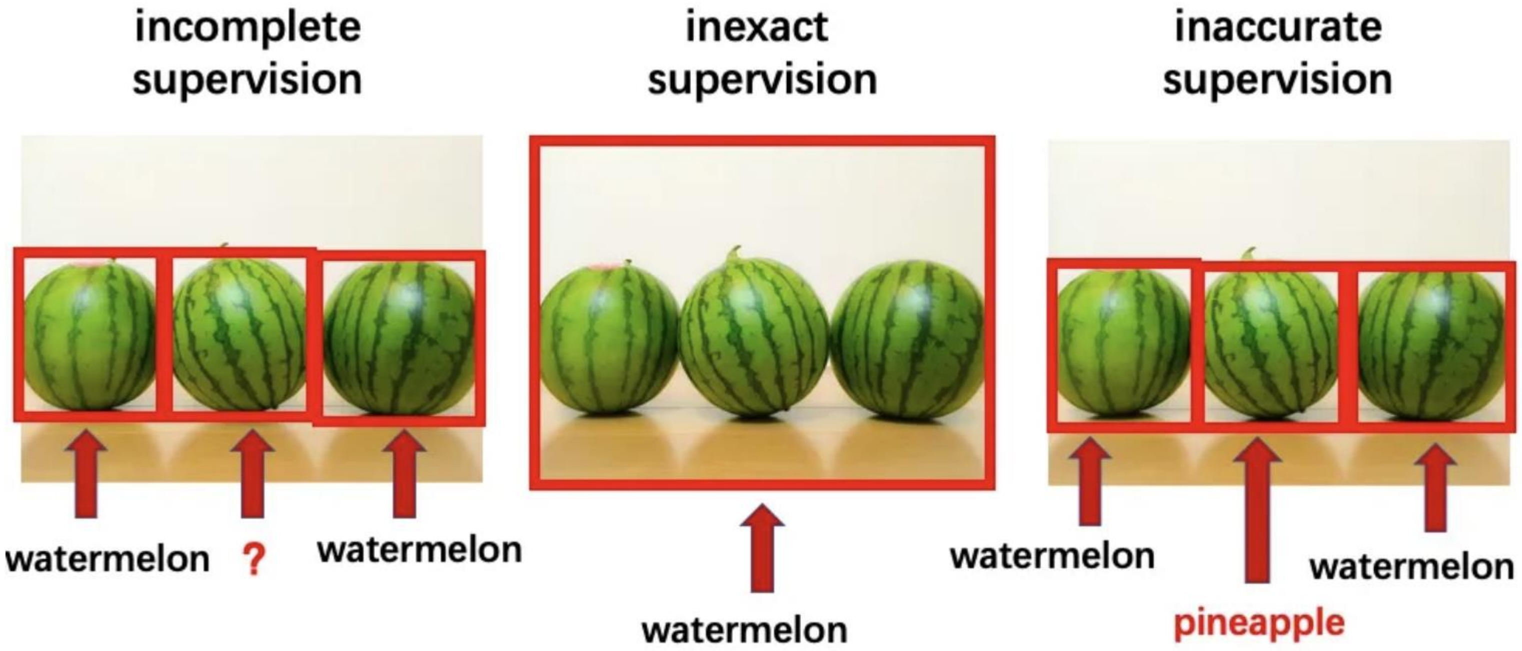


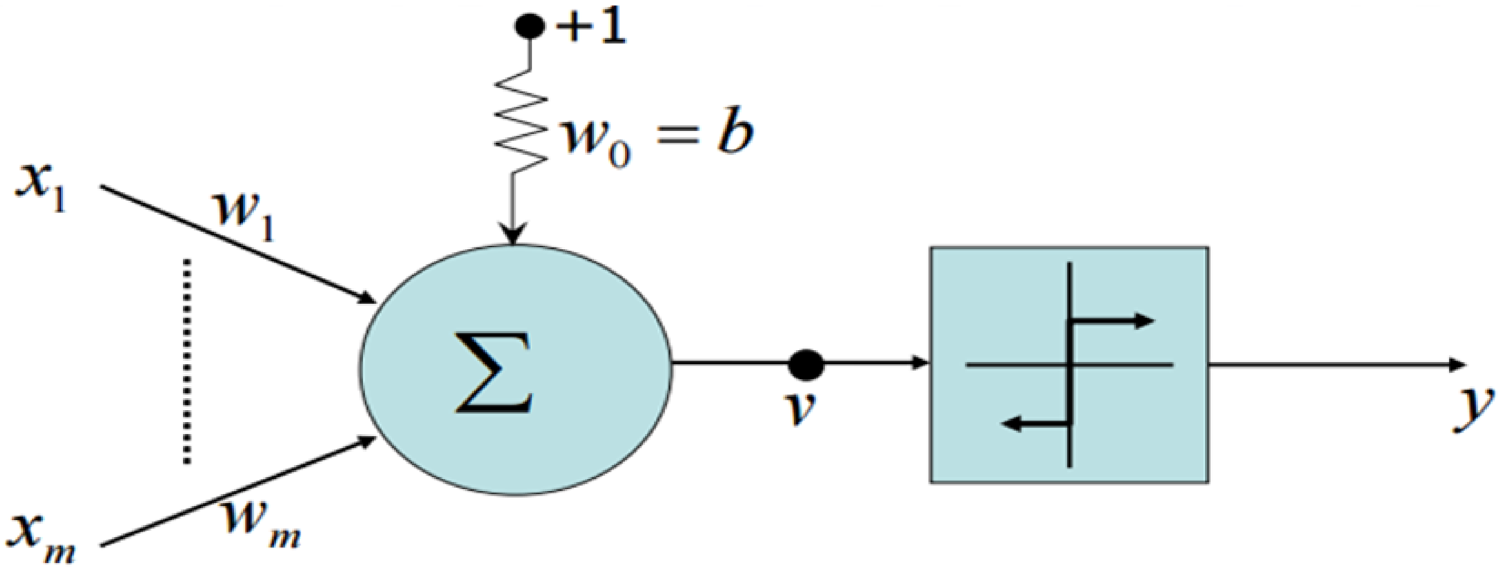
Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Learning approaches

- Supervised : need labeled ground truth
- Unsupervised: no labels
- Weakly supervised:
  - Incomplete
  - Inexact
  - inaccurate
-



Perceptron

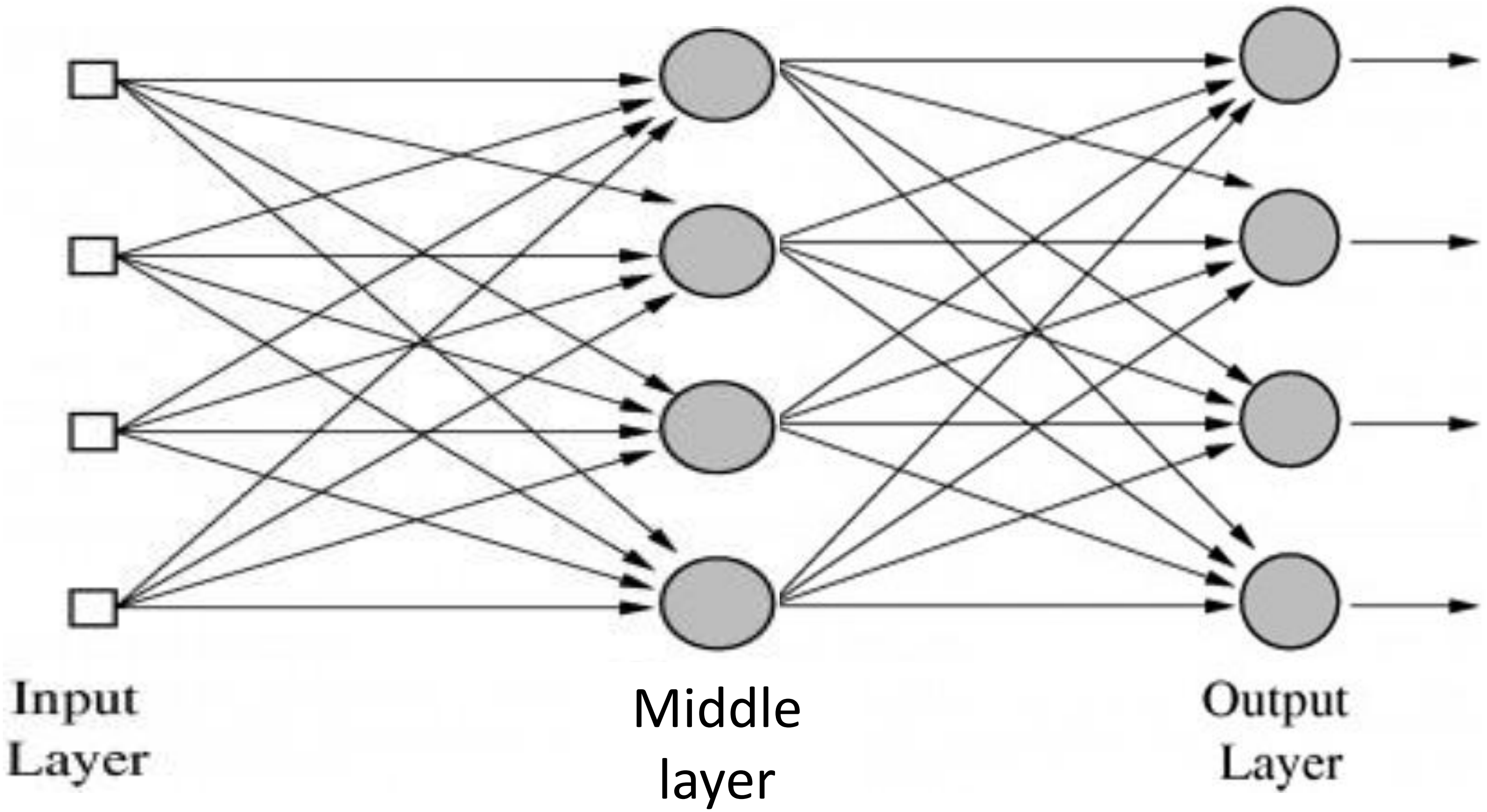


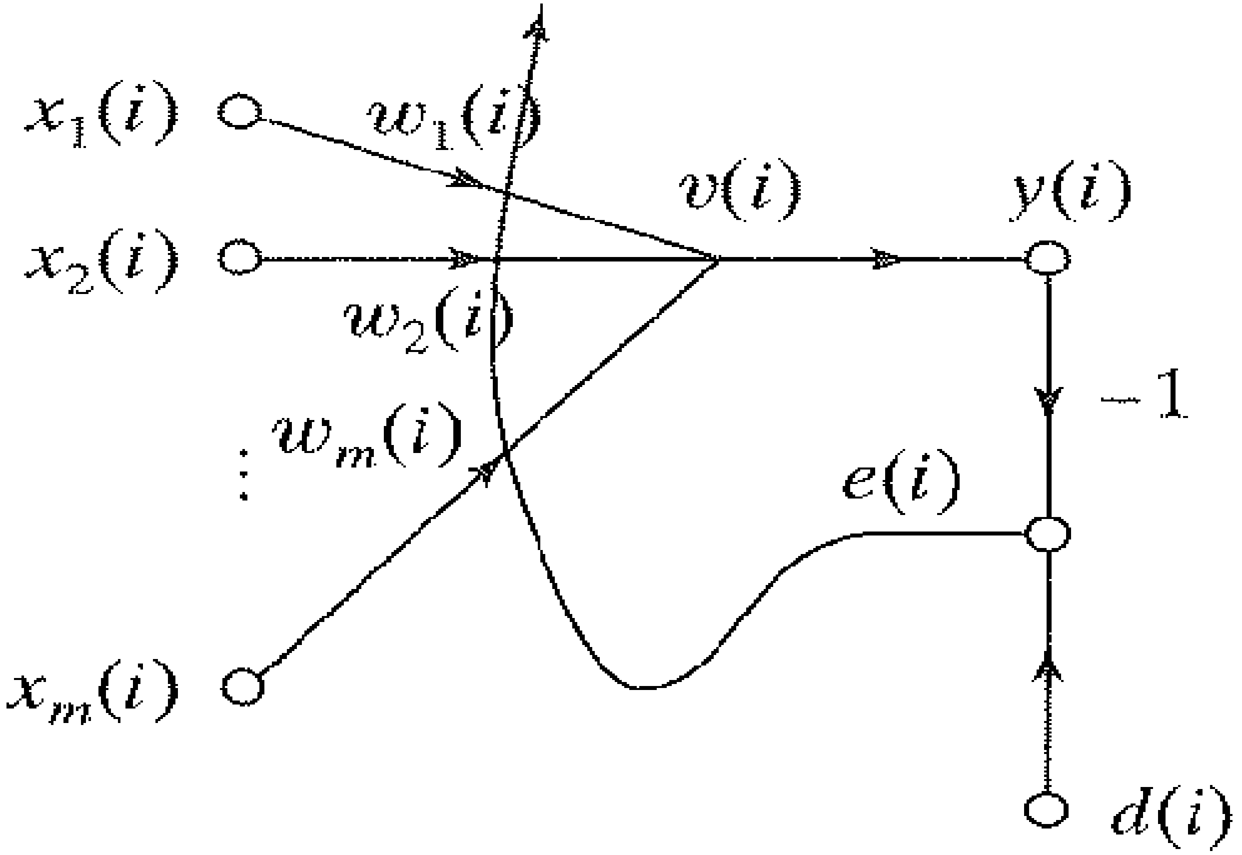
$$y = f(v)$$

$$v = \sum_{i=0}^{m} x_i w_i = \vec{w}^T(n)\vec{x}(n)$$

$f(v)$: $non-linear\ activation\ function$

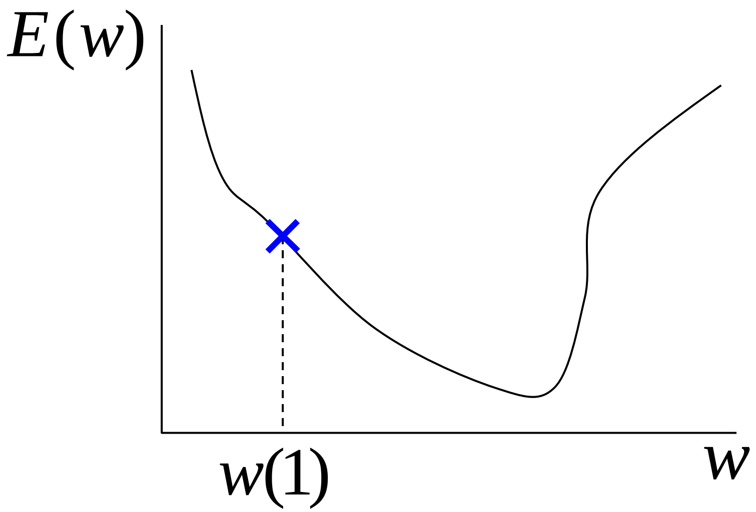
Multi-layer Perceptron





$x_1(i)$
$w_1(i)$
$v(i)$
$y(i)$
$x_2(i)$
$w_2(i)$
-1
$w_m(i)$
$e(i)$
$x_m(i)$
$d(i)$

Gradient-Decent



設目前 $w = w(1)$，請將 w 加一值

使得 $E(w(2)) < E(w(1))$

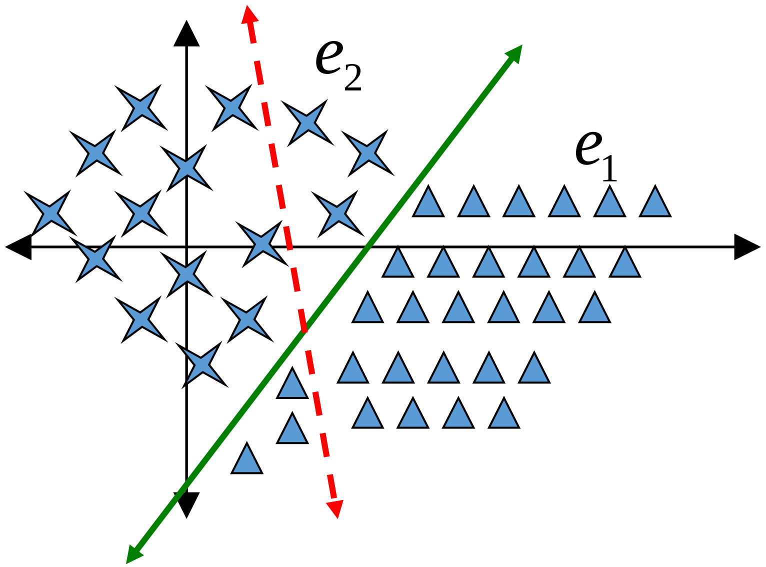
其中 $w(2) = w(1)$ + 此值(Δw)

令 $E'(w(1))$ 為 $E(\cdot)$ 在 $w(1)$ 之微分

由圖知 $E'(w(1)) < 0$，但明顯地 $\Delta w > 0$

因此 $\Delta w = -\eta \, E'(w(1)) = -\eta \left. \frac{\partial f(w)}{\partial w} \right|_{w=w(1)}$

perceptron



Given $$E = \sum_{k=1}^{\phi} (e_k(n))^2$$ >=0

當pattern $\vec{x}(n)$ 代入

若 $d(n) \neq y(n)$

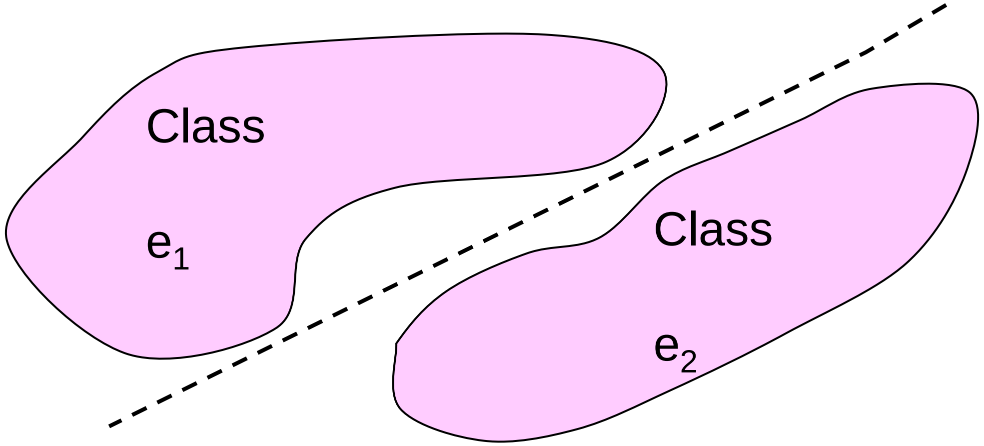
➔希望改 $\vec{w}$ 使 $E(n)$ 小

$$\Delta w = -\eta \frac{\partial E(n)}{\partial w} = +\eta [d(n) - y(n)] \vec{x}(n)$$

若 $d > 0$ 則 $\Delta w = \eta_1 \vec{x}(n)$

若 $d < 0$ 則 $\Delta w = -\eta_1 \vec{x}(n)$

Summary



For every pattern

$$\vec{w}^T \vec{x} > 0 \quad \forall\, \vec{x} \in e_1$$
$$\vec{w}^T \vec{x} < 0 \quad \forall\, \vec{x} \in e_2$$

- Adapting weights:
  1. If correctly classified

$$\vec{w}(n+1) = \vec{w}(n) \quad \text{if} \quad \vec{w}^T \vec{x} > 0 \ \text{and} \ \vec{x} \in e_1$$
$$\vec{w}(n+1) = \vec{w}(n) \quad \text{if} \quad \vec{w}^T \vec{x} < 0 \ \text{and} \ \vec{x} \in e_2$$

  2. If incorrectly classified (also called Hebbian learning)

$$\vec{w}(n+1) = \vec{w}(n) - \eta(n)\vec{x}(n) \quad \text{if} \quad \vec{w}^T \vec{x} > 0 \ \text{and} \ \vec{x} \in \underline{e_2}$$
$$\vec{w}(n+1) = \vec{w}(n) + \eta(n)\vec{x}(n) \quad \text{if} \quad \vec{w}^T \vec{x} < 0 \ \text{and} \ \vec{x} \in \underline{e_1}$$

Summary of Learning

I. Initialize the weight w(0)=0

II. at t=n apply the training patterns

III. Compute the actual responses

$$y(n) = sign[\vec{w}^T(n)\vec{x}(n)]$$

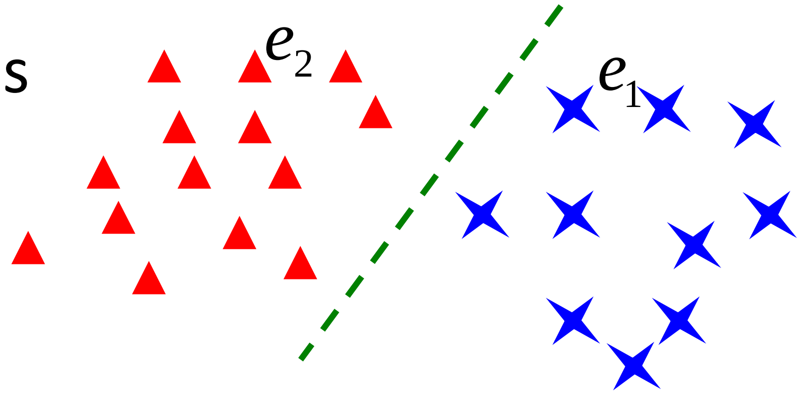
IV. Update the weight

$$\vec{w}(n+1) = \vec{w}(n) + \eta[d(n) - y(n)]\vec{x}(n)$$

$$\because d(n) = \begin{cases} 1 & if\ \vec{x} \in e_1 \\ -1 & if\ \vec{x} \in e_2 \end{cases}$$

➔if d(n)≠y(n), then $\vec{w}(n+1) = \begin{cases} \vec{w}(n) + \eta\,\vec{x}(n) & if\ \vec{x} \in e_1 \\ \vec{w}(n) - \eta\,\vec{x}(n) & if\ \vec{x} \in e_2 \end{cases}$

V. Increment n, go back to II



Learning Rate Annealing Schedules

1. $\eta = \eta_0$ for all n
2. $\eta(n) = \dfrac{c}{n}$ stochastic approximation schedule
3. $\eta(n) = \dfrac{\eta_0}{1+(n/\tau)}$ Darken & Moody (1992)

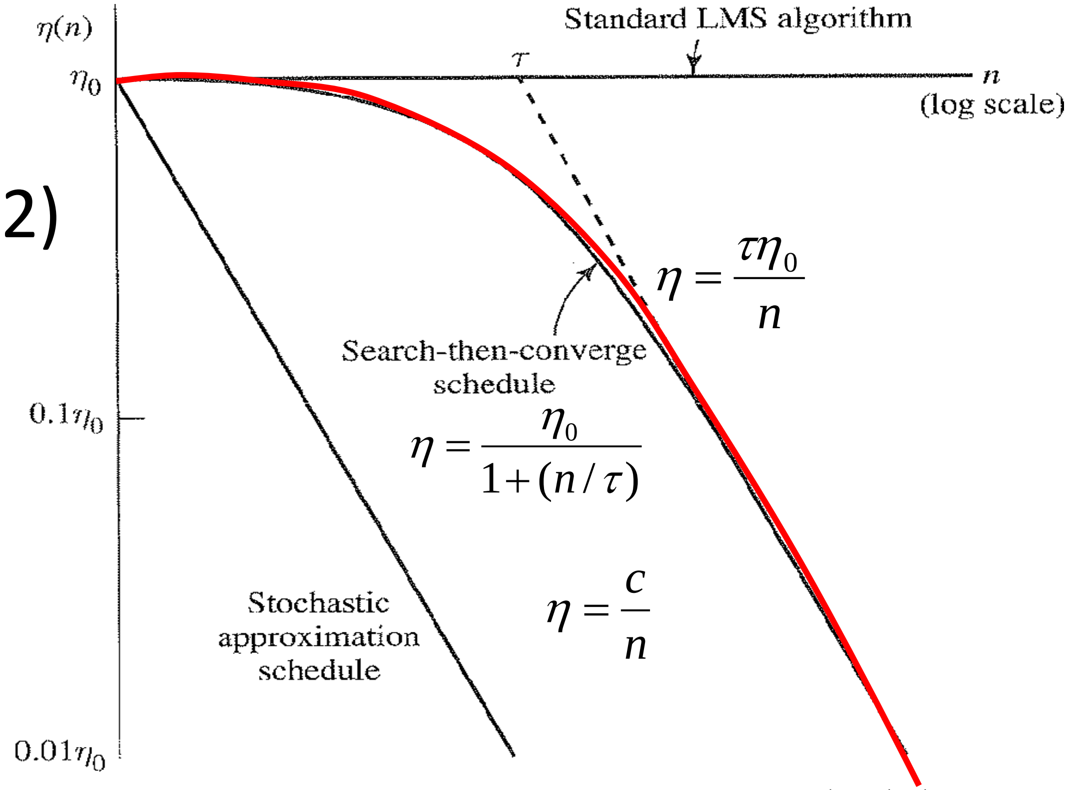


FIGURE 3.5 Learning-rate annealing schedules.