Unit 6

Deep Learning

課程總覽

單元一、深度學習技術發展現況

單元二、深度學習模型的概念與理論基礎

單元三、深度學習模型簡介與應用

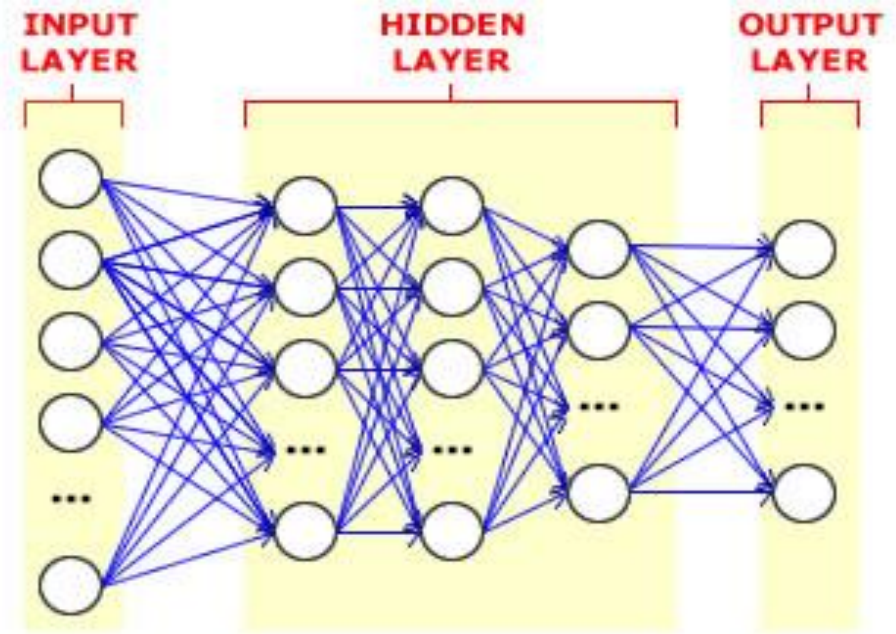
單元四、深度學習模型的設計與最佳化

單元五、如何有效的訓練深度學習模型

單元一、深度學習技術發展現況

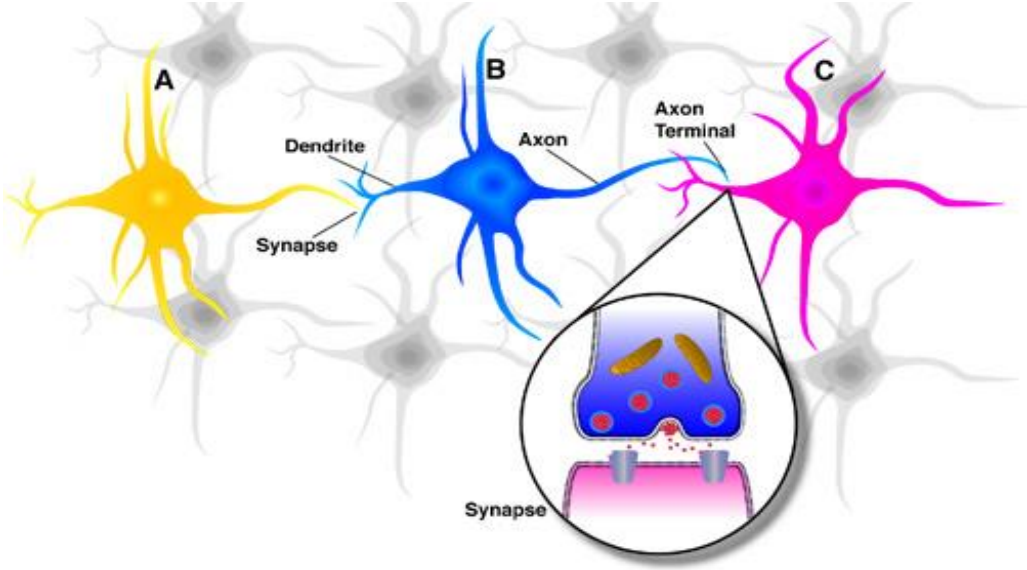
Deep Learning Overview

- Deep Learning is a new area of Machine Learning research.
- Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text.



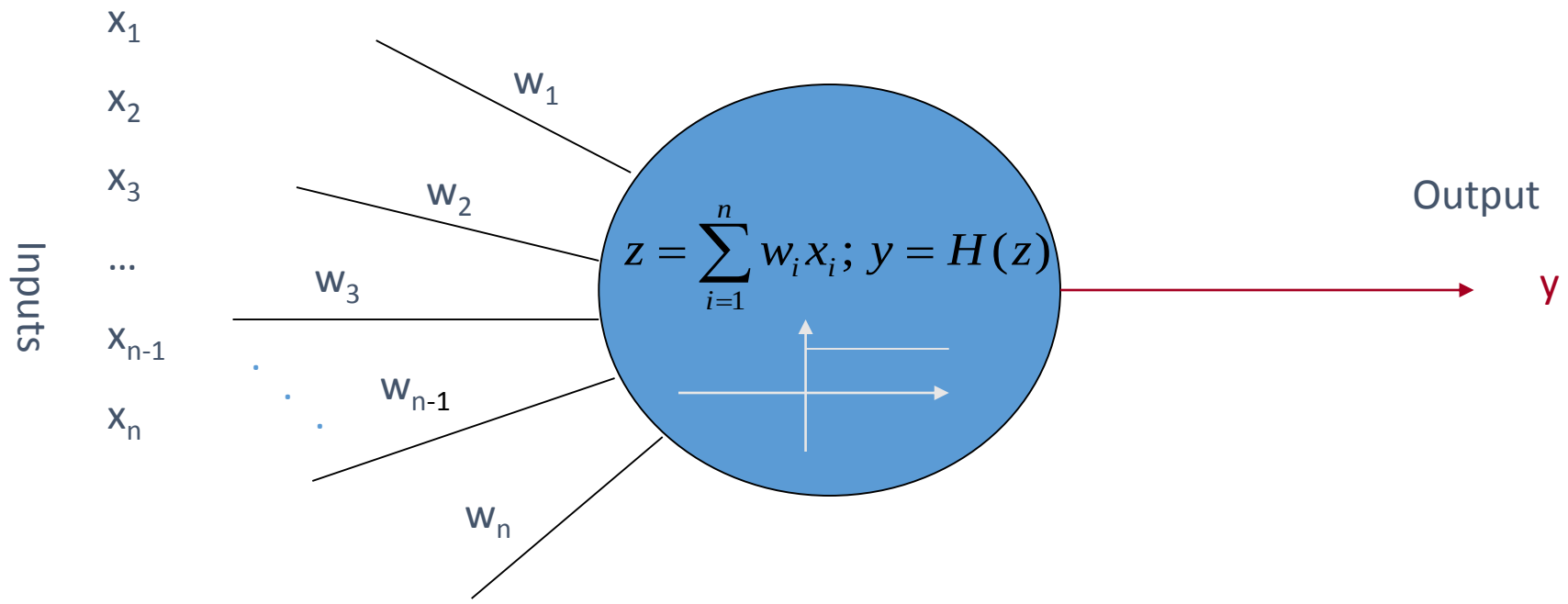
Biological Inspiration

- At a low level, the brain is composed of neurons
  - A neuron receives input from other neurons (generally thousands) from its synapses.
  - Inputs are approximately summed
  - When the input exceeds a threshold the neuron sends an electrical spike that travels that travels from the body, down the axon, to the next neuron(s)



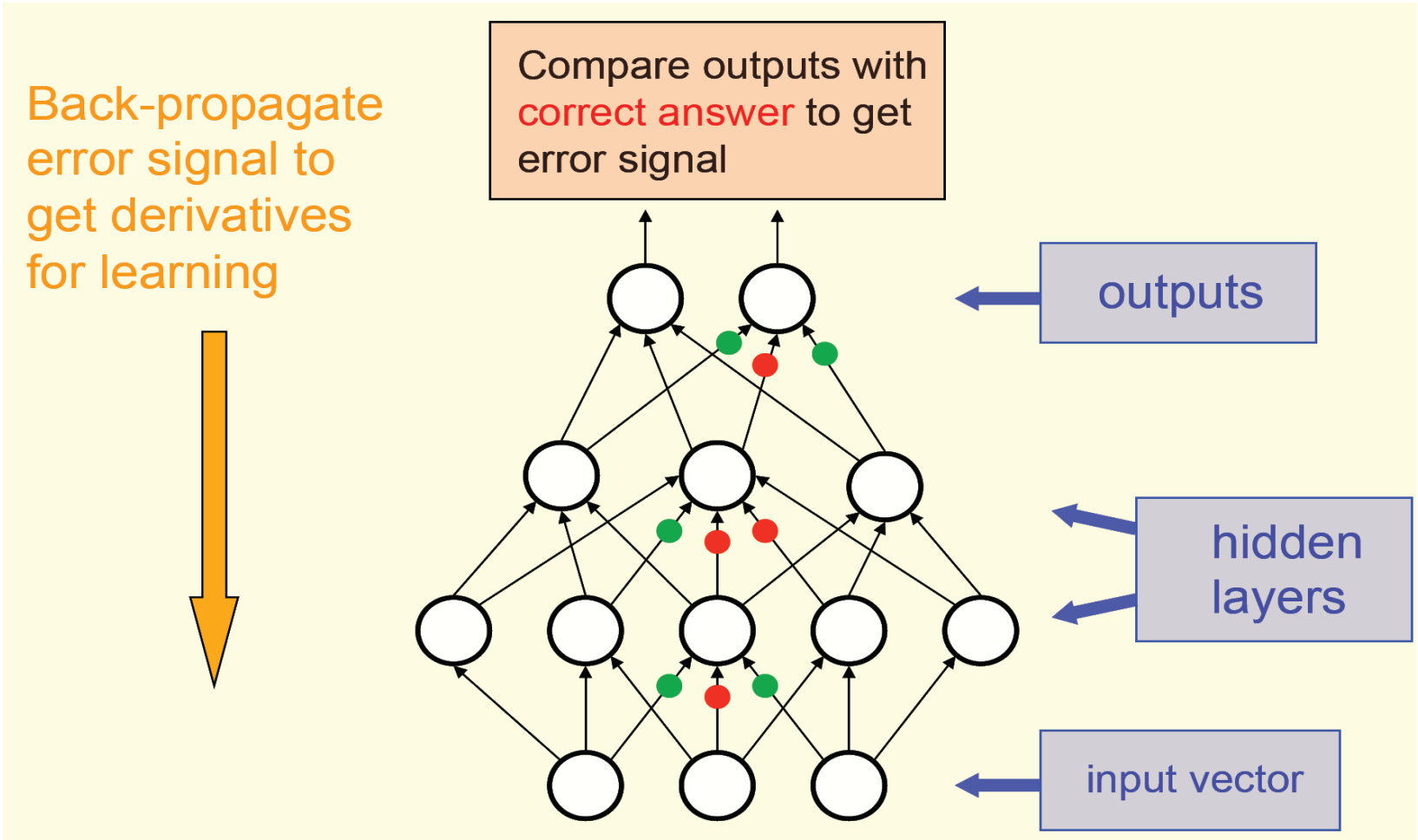
Artificial Neurons

- Neurons work by processing information. They receive and provide information in form of spikes.



The McCullogh-Pitts model

History of Deep Learning



What is wrong with Neural Network?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.
  - Researchers reported positive experimental results with typically two or three levels, but training deeper networks consistently yielded poorer results.
- Overfitting happens frequently.
- Training parameters is sometimes tricky.

How Many Computers to Identify a Cat?

- Google Brain
  - Andrew Ng and Jeff Dean
  - 16000 CPU Core of Deep Neural Networks
  - 10億節點 V.S. 人腦 150億神經元

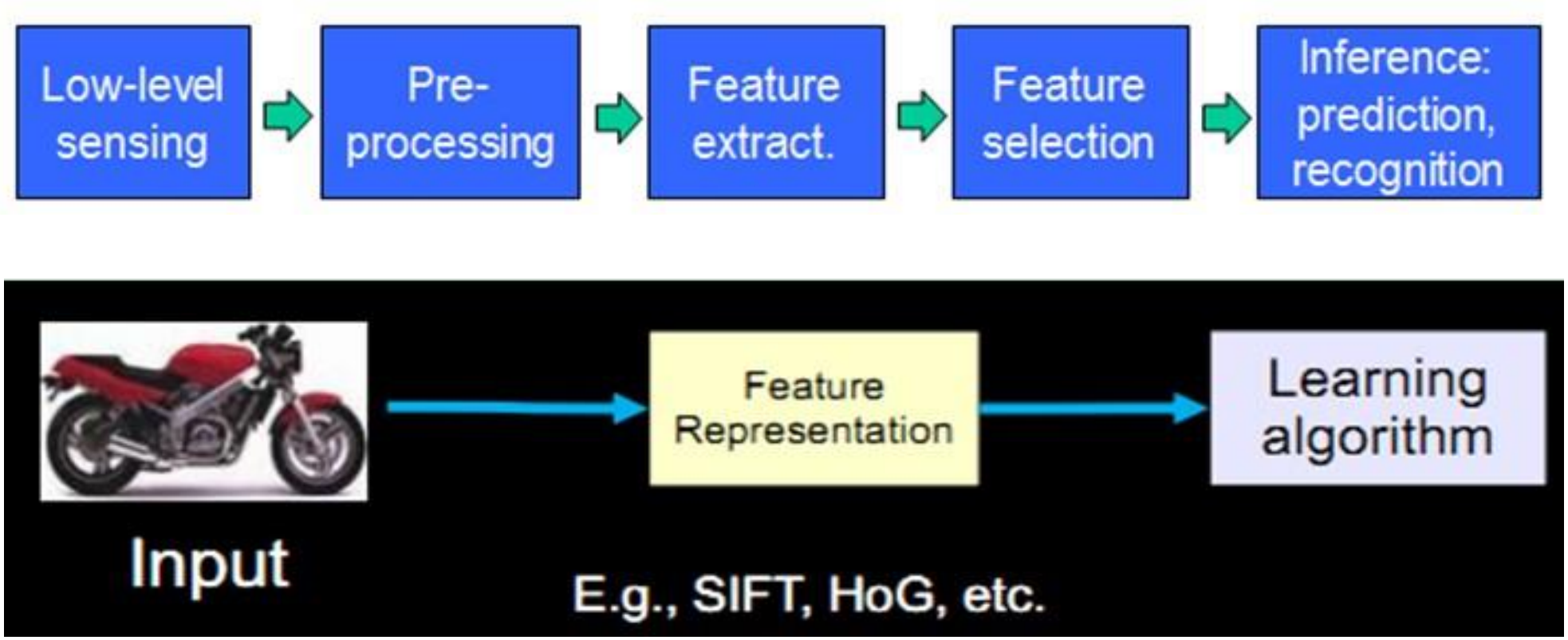
- Let data speaks for itself
  - The concept of ‘cat’ is not taught
  - It realizes ‘cat’ by itself

Object Recognition

- Over 1,000,000 images and 1,000 categories on ImageNET (2 GPU)

Rank	Name	Error rate	Description
1	**U. Toronto**	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

Traditional Recognition Approach

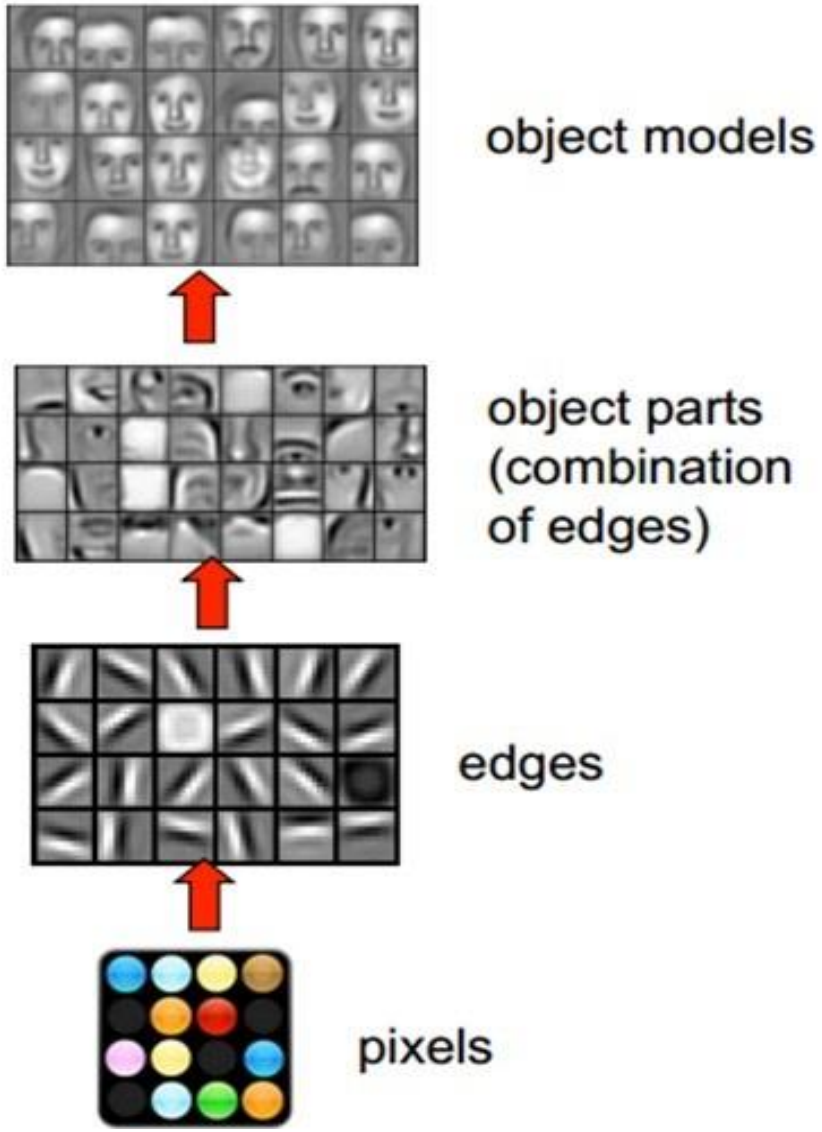


Problem with feature learning

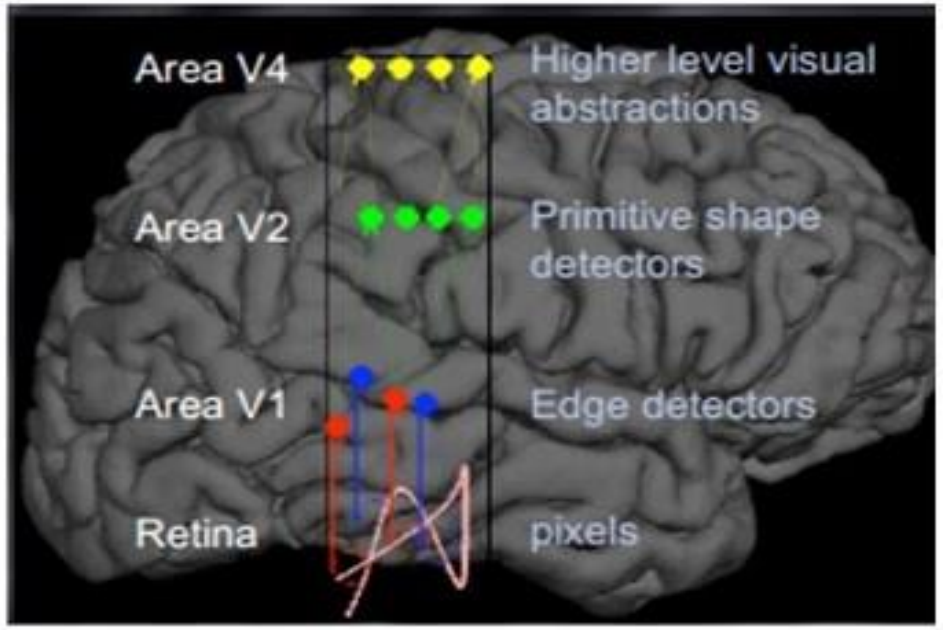
- Handcrafted features are very exhaustive
  - Need expertise
  - Require much time for adjustment
  - Sometimes depends on experiences and a little luck!

- Can we learn feature automatically?
  - The answer is ‘Deep Learning’ !
  - Also called Unsupervised Feature Learning

Motivation of Deep Learning



- 刺激－神經－中樞－大腦程序是一種**不斷迭代、不斷抽象**的過程
  - 抽象：原始訊號，建立低層抽象，發展高層概念
  - 迭代：反覆這樣的過程學習認知



AI 研發方向的推動浪潮

- 科技部 投下 50 億 AI 計畫
  - 「AI創新研究中心」- 台大、交大、清大及成大
  - 科技部「AI創新研究中心專案計畫」
- 經濟部工業局
  - 「 AI智慧製造產創平台」
  - 「智慧機械創新創業推動計畫」
- 教育部
  - 補助「人工智慧技術與應用領域系列課程」計畫
  - 推動「全國智慧製造大數據分析競賽」

產業界 - 定義「智慧製造」的未來

這一次 我們再定義
智慧製造的未來

二〇一四年底，鼎新已開始積極佈局「智慧製造、全通路零售、微企互聯網」三大戰略方向，全力打造以提升客戶「應用價值」為核心之的發展路徑，於二〇一五年七月正式發布：「智通天下」的戰略目標，並升級成「一線、三環、互聯」的戰略目標。

基於「一線三環互聯」的全景戰略，實現產業資源分享與協同，與智慧製造系統整合聯盟商，共同打造產業聯盟生態系，彈性支撐企業在網路發展下跨界融合的創新營運模式轉變；致力成為台灣乃至亞洲的工業4.0智慧製造方案整合者與規劃者。

研華科技
蔡奇男 副總經理

智慧製造最重要的目的，到底是什麼？

勞力成本提高等因素迫使著製造業要迎接未來挑戰，將物聯網技術融入了製造，就變成智慧製造。

當產品在市場上有任何回饋，可以從生產領域去追蹤到產品，是在什麼時候、是在哪一企業、哪一條生產線、哪一設備、當時生產條件如何，能夠把這個資訊有效串聯。智慧製造能確實達到管理更及時、品質更提高，帶來客戶滿意，當然最重要目的是帶來企業競爭力。

三惟科技
鄭逸陽 總經理

很多企業提出的要求，就是往工業4. 0方向走，其追求的本質為何？

工業4.0的議題被熱烈討論，而背後代表即是勞動力減少，以自動化設備取代高重複性或高危性的工作，解決人力成本調漲、技術斷層以及確保品質穩定的未來趨勢，因此ＳＩ系統整合商也是這波轉型變革的一環。

工業4.0時代機台不僅要會說話，彼此之間的訊息還要聯通。如何達到共通，就是靠雲瑞或是靠整合資料庫的收集，而目前企業所追求的，仍是回歸管理的本質議題。

東毓油壓
楊楨彬 董事長

客製風暴來襲，企業必須要擁有什麼樣的思維？

「現在客製化接單，一定要有的思維，就是製造服務化！」製造服務化的思維轉變已經勢不可擋，改變並非單一個人或者單一職能所能支撐，東毓以研發作為轉型創新的關鍵第一步，串聯全廠資訊化連結，從採購、生管、製造更整合到產銷與生產，建置通透的資訊整合的應用環境。而效益主要顯現在文管圖面流程的簡化，透過模組化管理，加快設計、採購、發包、生產速度，效率提升六十％。

深度學習的研發趨勢

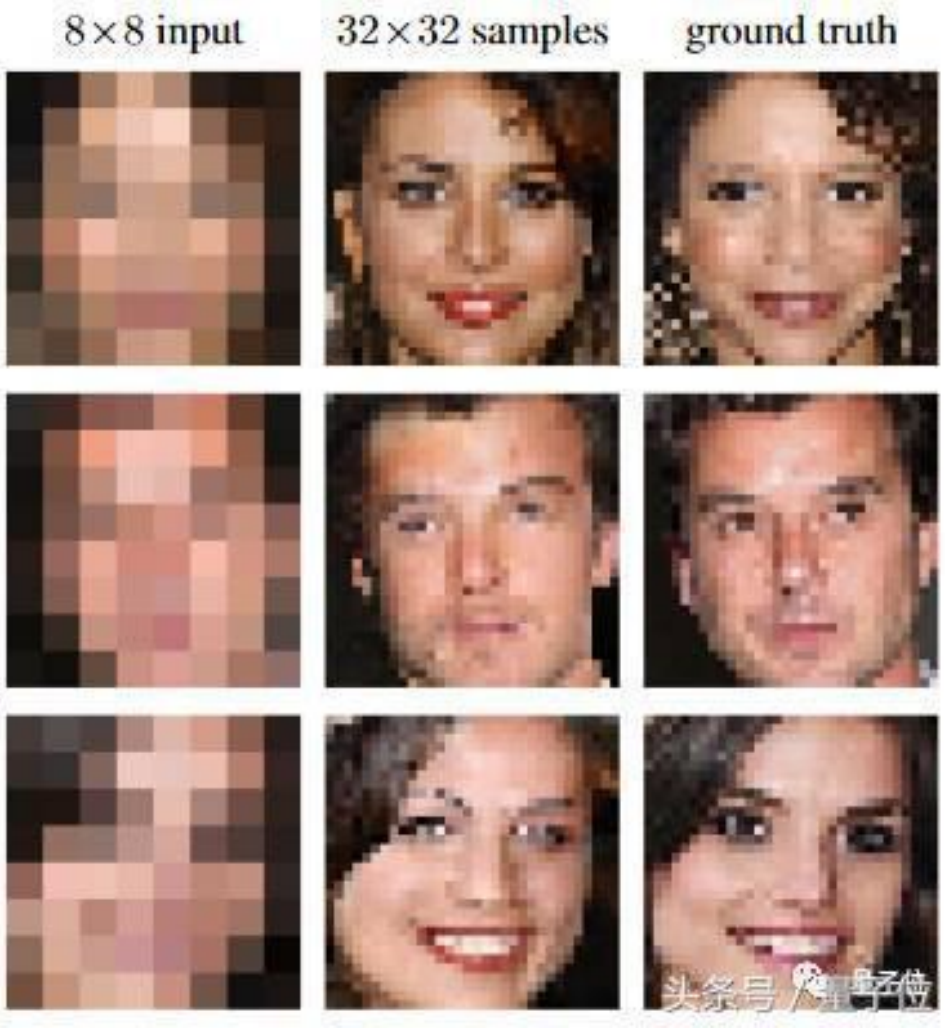
- 資訊工程領域
  - 深度學習應用於影像、視訊、文字、語音、音樂、虛擬實境
  - AI深度學習正在改寫SoC晶片設計
  - 結合視覺與深度學習邁向AI的嵌入系統新紀元
  - 深度解讀深度學習在IoT大資料和流分析中的應用
  - JeffDean用深度學習顛覆計算機系統結構設計
- 其他領域
  - 醫學、農業、交通、空汙檢測防治、機械、美術、文學

Face2Face：扮演川普

- 把你的面部表情實時移植到視頻里正在發表演講的美國總統身上。

Pixel Recursive Super Resolution：告別馬賽克

- 解析度極低的人臉圖像來預測這些面孔真實的樣子



多人動作預估辨識

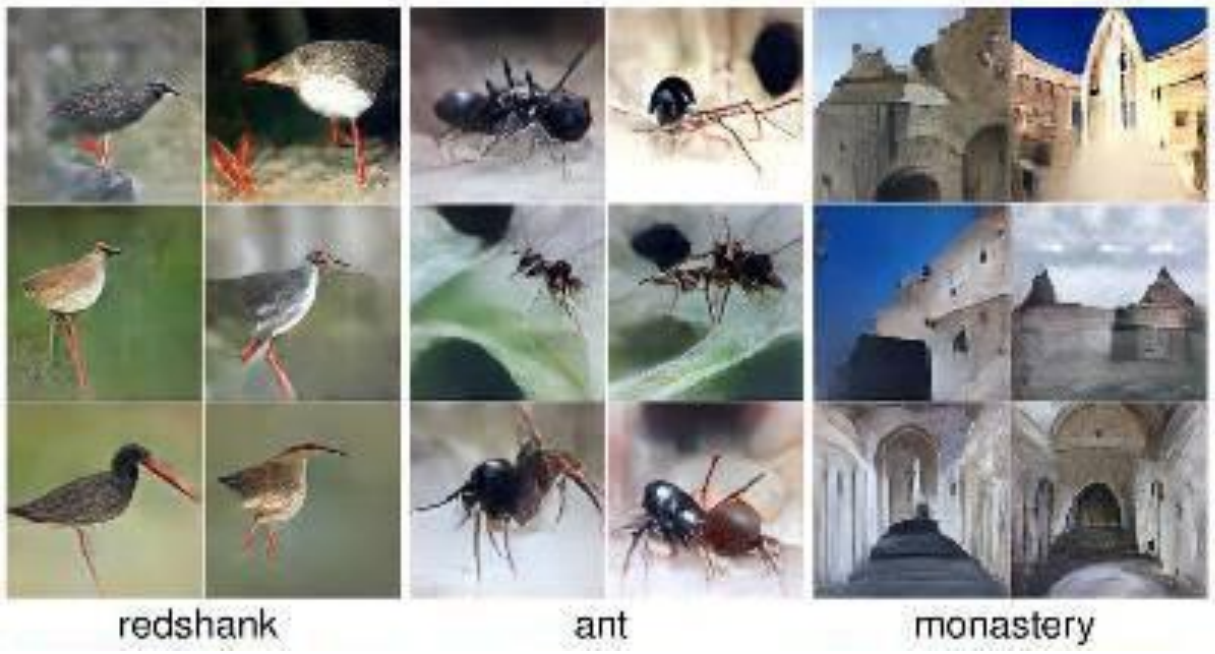
- 神經網絡估算人類骨架位置的變化。

Neural Talk：描述照片

- 對照片中不同區域內元素進行識別，並用一句話來描述照片的深度學習系統。

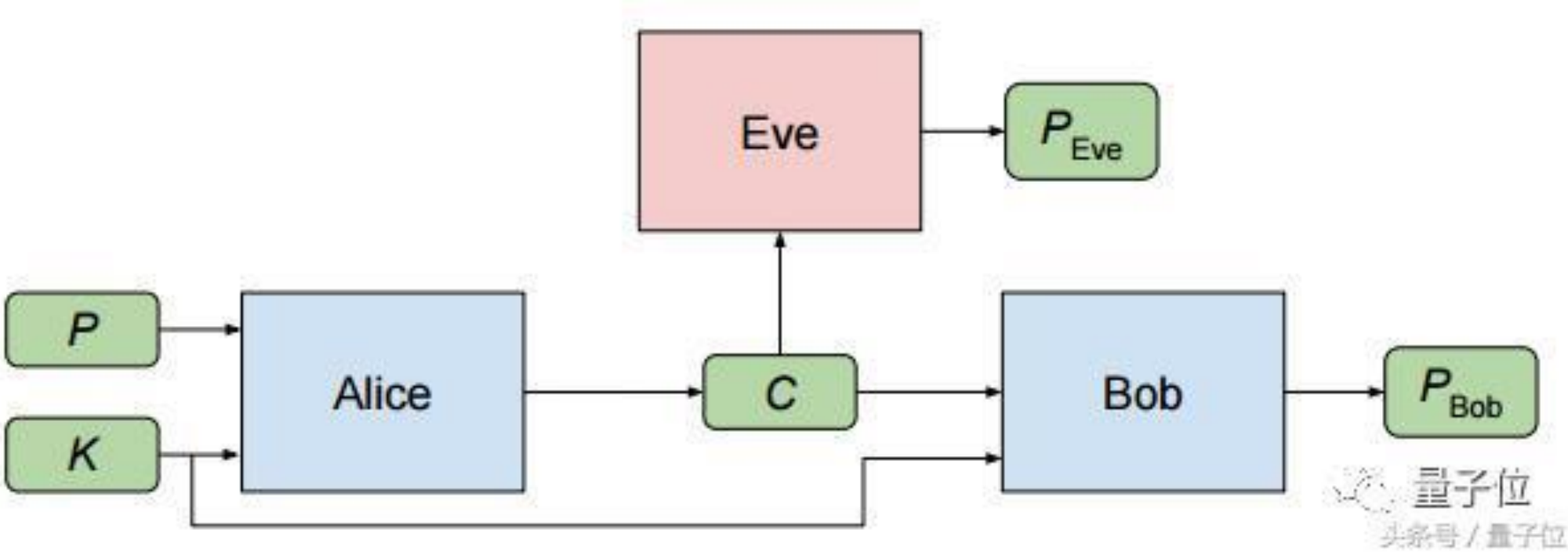
生成新照片

- 能用疊代的方式從已有照片中合成全新照片的深度學習網路



反覆攻破和修補自己的防火牆

- 一個不斷創造自己的加密算法，然後讓另一個網絡去盡力攻破它。在兩套系統的反覆纏鬥。



AI人工智能少女寫情詩

- 「微軟小冰」經過6000分鐘訓練，學習519位現代詩人的作品，現在她只要從一幅圖像獲取靈感，就能寫詩。

我尋夢失眠

康橋
新鮮的
未經三月之蕙風已不追蹤
在夢裏我尋夢失眠
我是一座長橋
你可以找到我新鮮的愛情
將希望之光投射到你
也不知道是風

全球首次！AI畫作於紐約拍賣

- 人工智慧技術創作了這幅名為《Edmond de Belamy》的繪畫，畫中是一位名為愛德蒙·貝拉米的男子肖像，充滿迷幻氣息
- 全球首幅被拍賣的AI畫作，而佳士得給這幅作品的估價在7000到10000美元之間。
- 不同與其他人物肖像畫，這幅作品可以模仿人類提供的一系列圖像，並將14世紀至20世紀的數千幅肖像錄入，讓AI系統自己學習。

AI 音樂創作

- AIVA (Artificial Intelligence Virtual Artist)
  - 1 hour music collection

AI書法字體生成與撰寫

原始	科	技	部
生成	科	技	部

原瘦金體

AI生成瘦金體

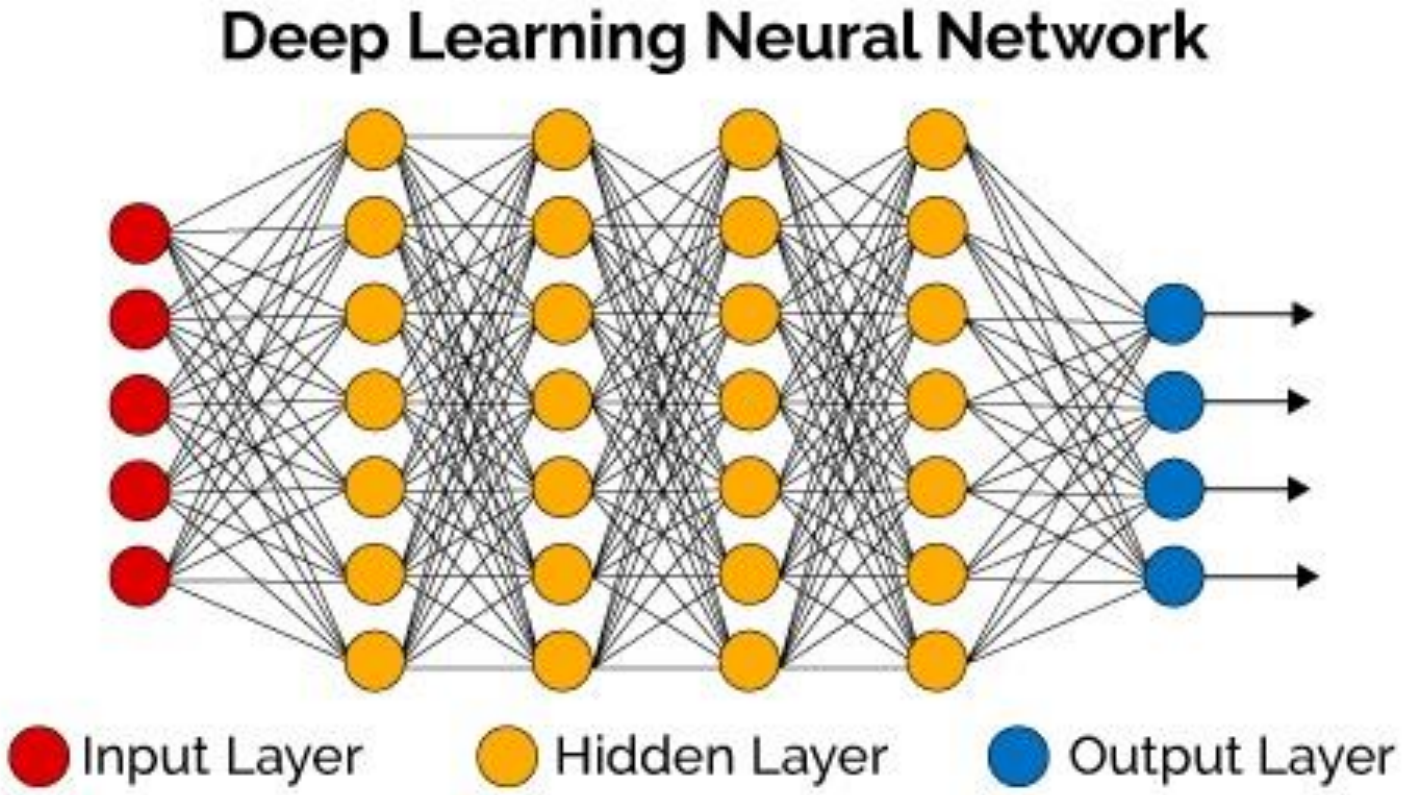
骨架擷取

機器人書寫

單元二、深度學習模型的概念與理論基礎

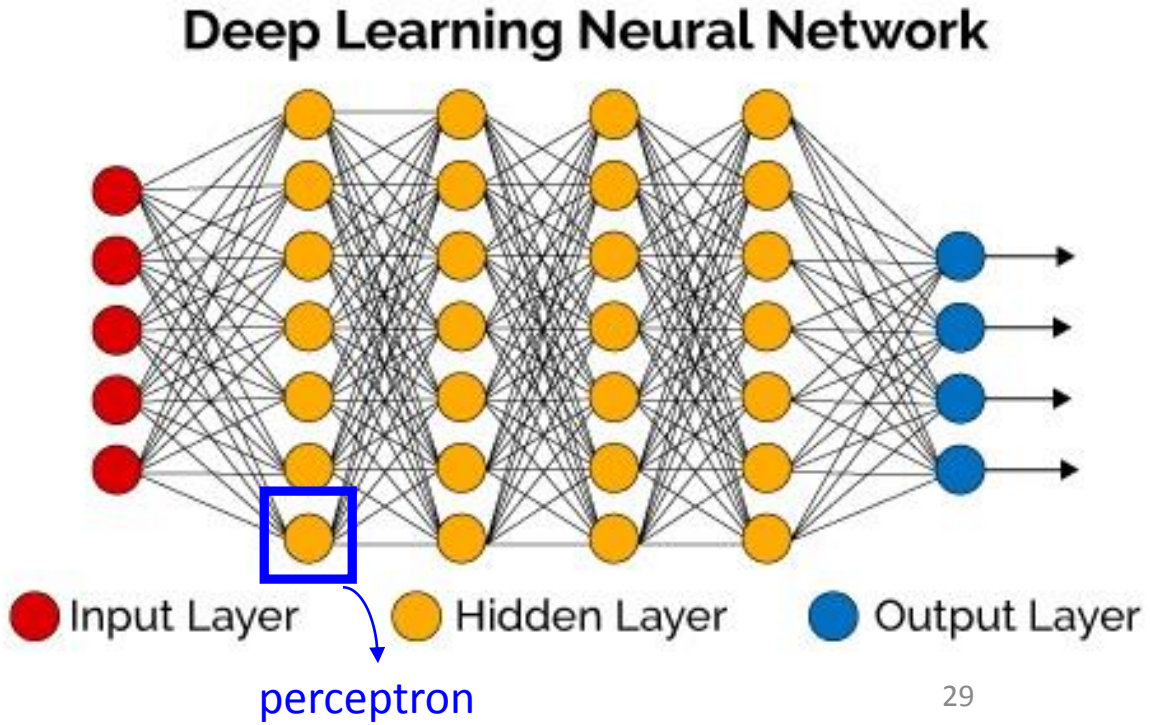
深度學習的架構－點、線、面

- 深度學習模型架構
  - ‘單一節點’的簡易功能
  - ‘單一階層’的基礎能力
  - ‘多重階層’的進階整合
  - ‘深度堆疊’的強大表現

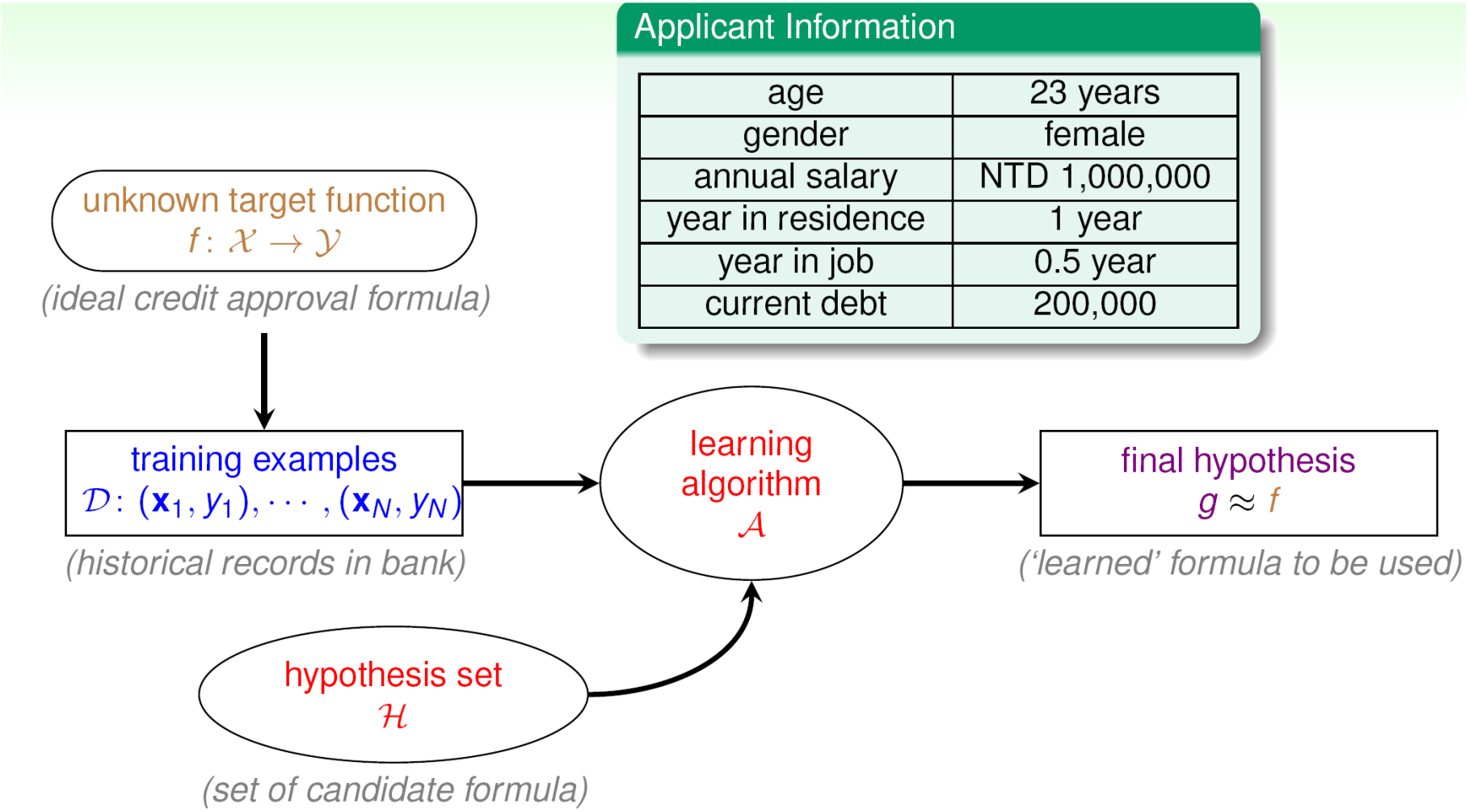


深度學習架構最基礎的模型 – Perceptron

- Perceptron (深度學習模型的單一節點)
  - Perceptron Hypothesis Set
  - Perceptron Learning Algorithm (PLA)
  - Guarantee of PLA
  - Dealing with Non-separable data



Credit Approval Problem Revisited

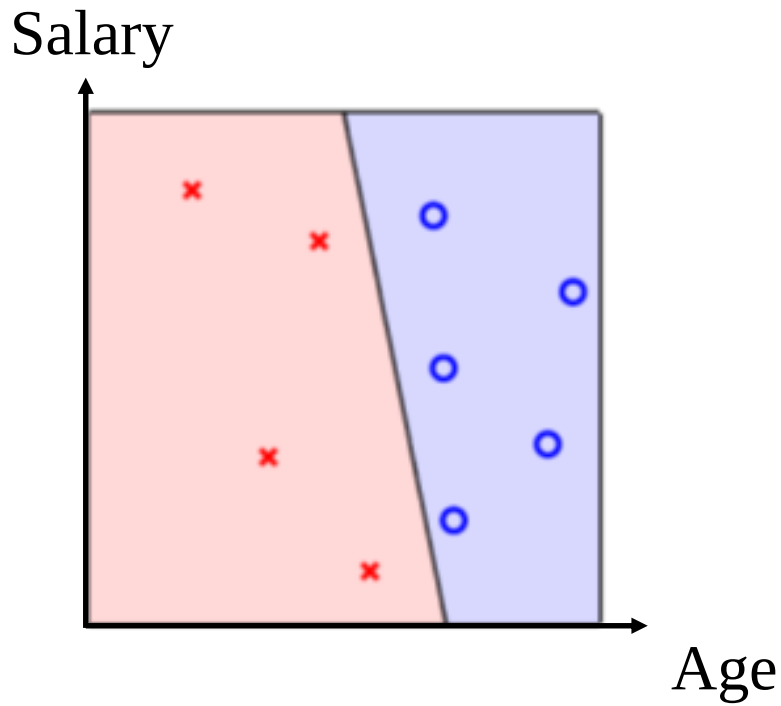


what hypothesis set can we use?

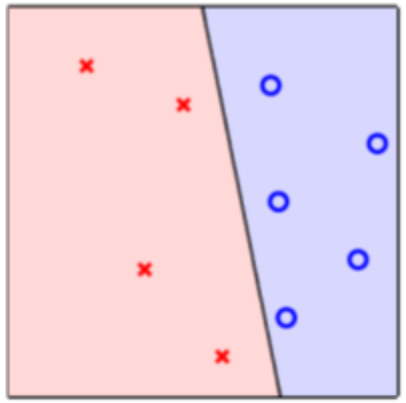
Let’s simplify our data to 2-dimension…

- If we only consider Credit Approval Problem by (age, salary)…

	Age	Salary	Approval
Customer 1	23	22,000	N
Customer 2	45	75,000	Y
Customer 3	31	60,000	Y
:	:	:	:
Customer n	26	25,000	N



Perceptron (感知器)



- customer features **x**: points on the plane (or points in $\mathbb{R}^d$)
- labels y: ○ (+1), × (-1)
- hypothesis h: **lines** (or hyperplanes in $\mathbb{R}^d$)
  —positive on one side of a line, negative on the other side
- different line classifies customers differently

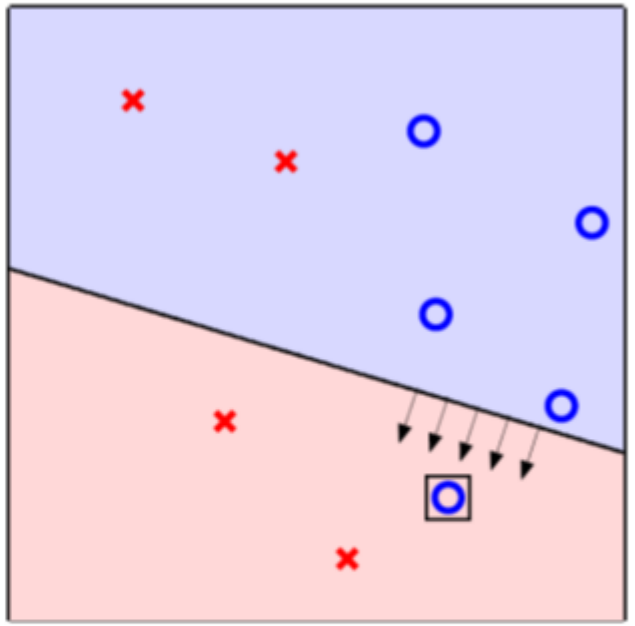
感知器

perceptrons ⇔ **linear (binary) classifiers**

如何選出正確的Perceptron?

$\mathcal{H}$ = all possible perceptrons, g =?

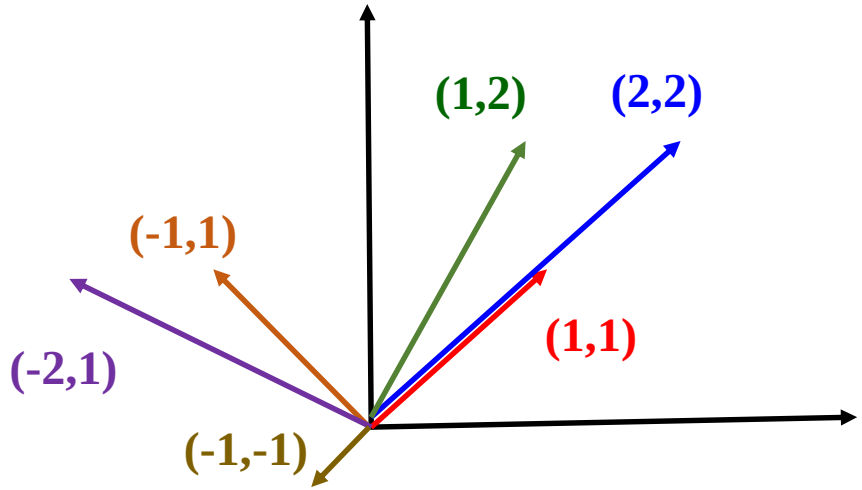
- want: $g \approx f$ (hard when f unknown)
- almost necessary: $g \approx f$ on $\mathcal{D}$, ideally $g(\mathbf{x}_n) = f(\mathbf{x}_n) = y_n$
- difficult: $\mathcal{H}$ is of **infinite** size
- idea: start from some g_0, and 'correct' its mistakes on $\mathcal{D}$



will represent g_0 by its weight vector $\mathbf{w}_0$

Recall line function and some properties…

- The line function in 2d space: $ax + by + c = 0$
  - (a, b) 為直線的法向量



- Property of Inner Product
  - 兩向量方向完全相同，向量的cos角度為1
  - 兩向量方向完全相反，向量的cos角度為-1
  - 兩向量方向垂直，向量的cos角度為0
  - 兩向量夾角差異小於90度時為正、大於90度為負、等於九十度為0

設 $\vec{v}_1 = (x_1, y_1), \vec{v}_2 = (x_2, y_2)$，且 $\vec{v}_1, \vec{v}_2$ 的夾角為 θ，

則 $\cos\theta = \dfrac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1 \| \vec{v}_2|} = \dfrac{x_1x_2 + y_1y_2}{\sqrt{x_1^2 + y_1^2}\sqrt{x_2^2 + y_2^2}}$ 。

符號定義

- 原本的二維資料 (x, y)，重新寫成 (x_1, x_2)
- 原本的直線方程式 $ax+by+c=0$，重新寫成 $w_0+w_1x_1+w_2x_2=0$

- 直線方程式可以視為(w_0, w_1, w_2) 與 $(1, x_1, x_2)$ 的內積=0。
  - $w_0+w_1x_1+w_2x_2 = (w_0, w_1, w_2) \cdot (1, x_1, x_2) = \boldsymbol{w} \cdot \boldsymbol{x} = 0$

- 平面上的點落在直線右邊，$w \cdot x > 0$；否則 $w \cdot x < 0$
  - 分類器可以用 **sign($\boldsymbol{w} \cdot \boldsymbol{x}$)** 表示
  - 資料以 $\boldsymbol{x}$ 表示、資料的label以 $\boldsymbol{y}$ 表示

Perceptron Learning Algorithm

start from some $\mathbf{w}_0$ (say, $\mathbf{0}$), and 'correct' its mistakes on $\mathcal{D}$

For $t = 0, 1, \ldots$

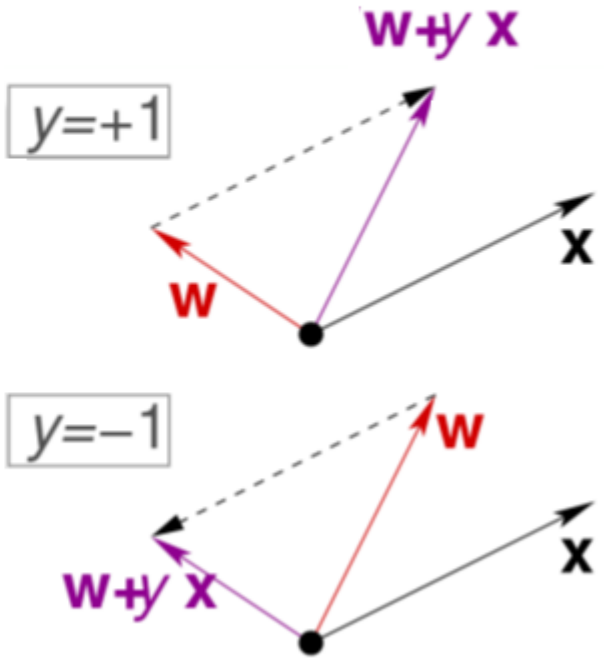
1. find a mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$

$$\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. (try to) correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$

. . . until no more mistakes
return last $\mathbf{w}$ (called $\mathbf{w}_{PLA}$) as g



Practical Implementation of PLA

start from some $\mathbf{w}_0$ (say, $\mathbf{0}$), and 'correct' its mistakes on $\mathcal{D}$

Cyclic PLA

For $t = 0, 1, \ldots$

1. find **the next** mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$

$$\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

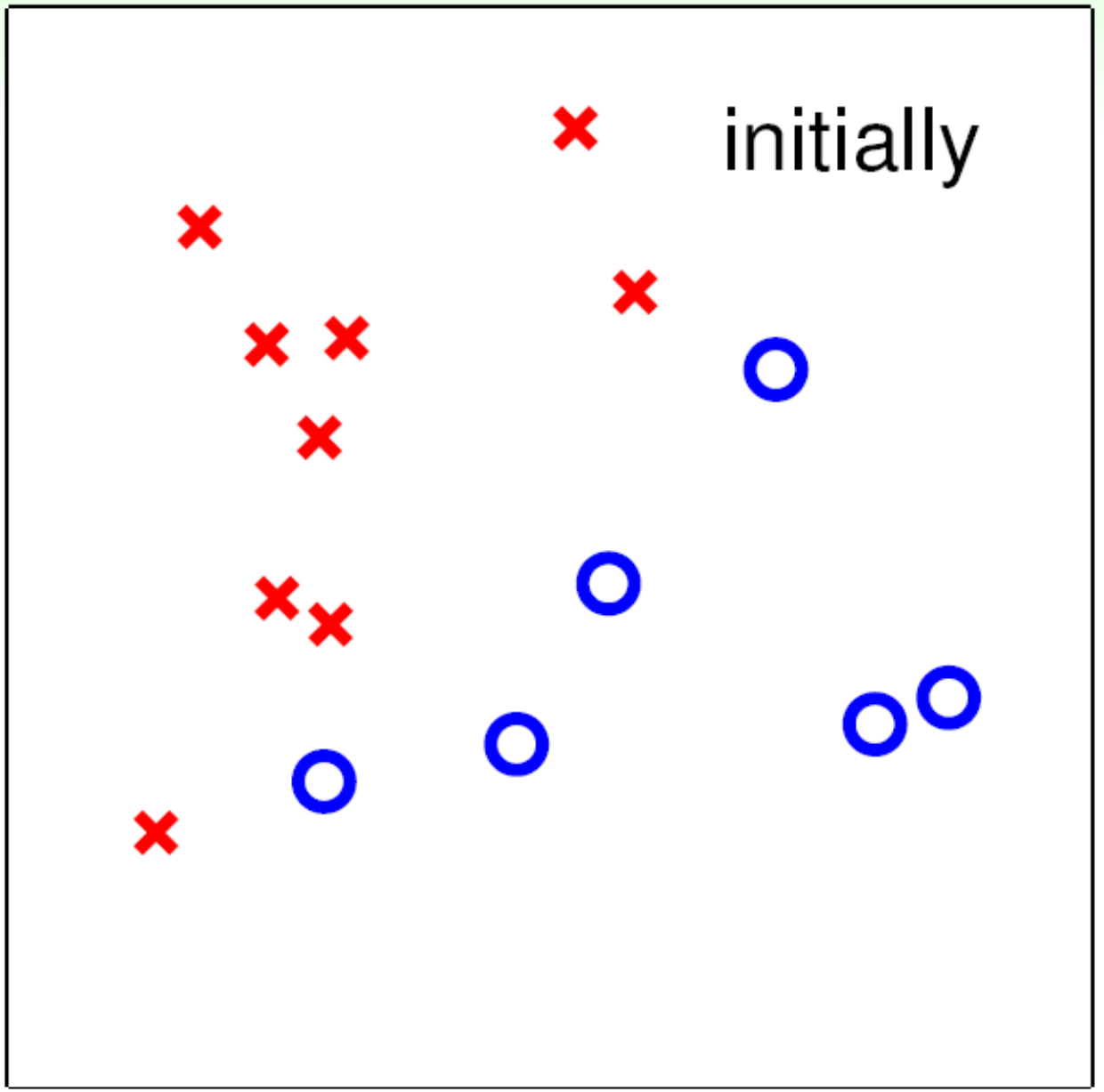
2. correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$

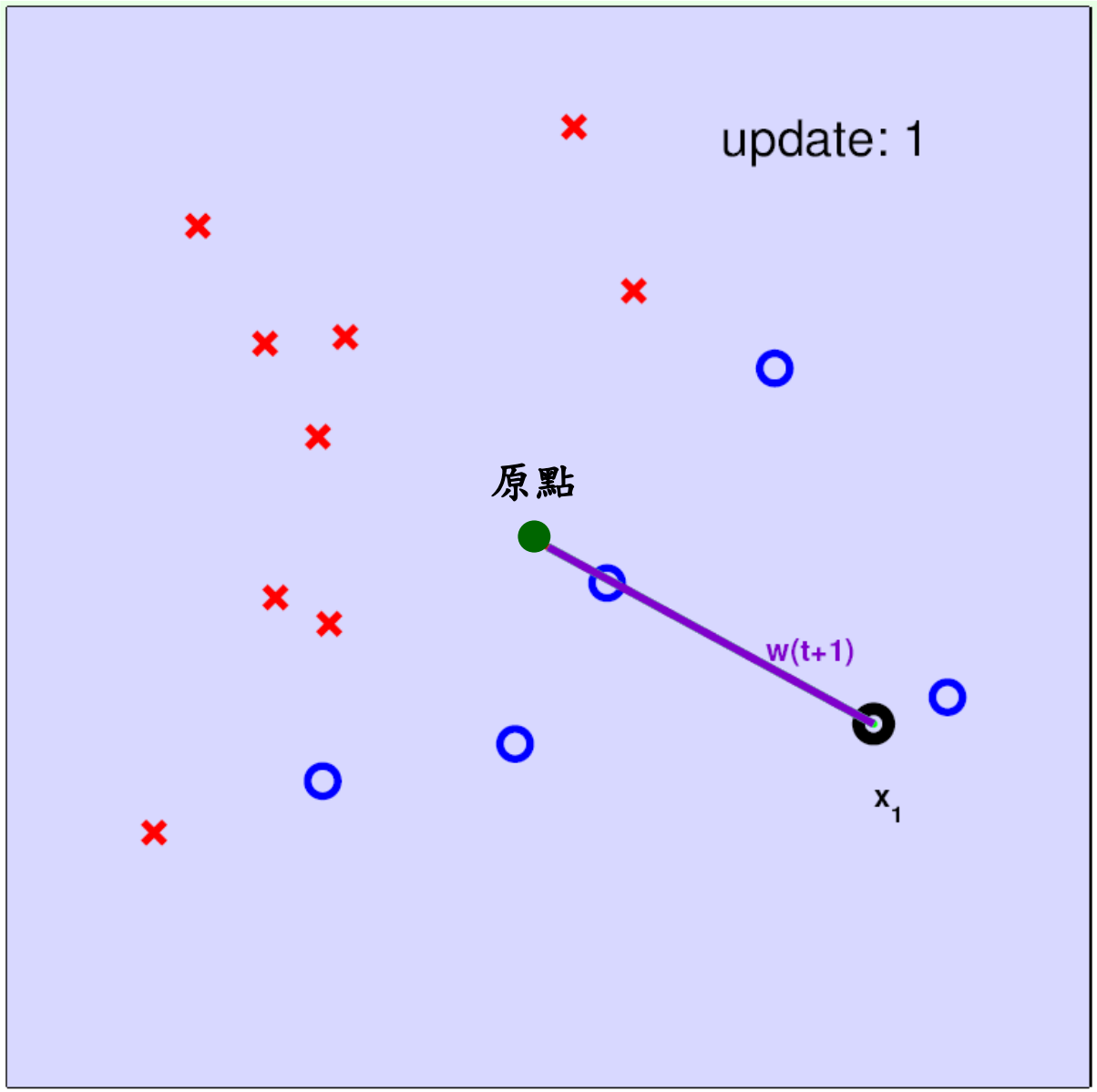
. . . until **a full cycle of not encountering mistakes**

next can follow naïve cycle $(1, \cdots, N)$
or precomputed random cycle

Seeing is Believing

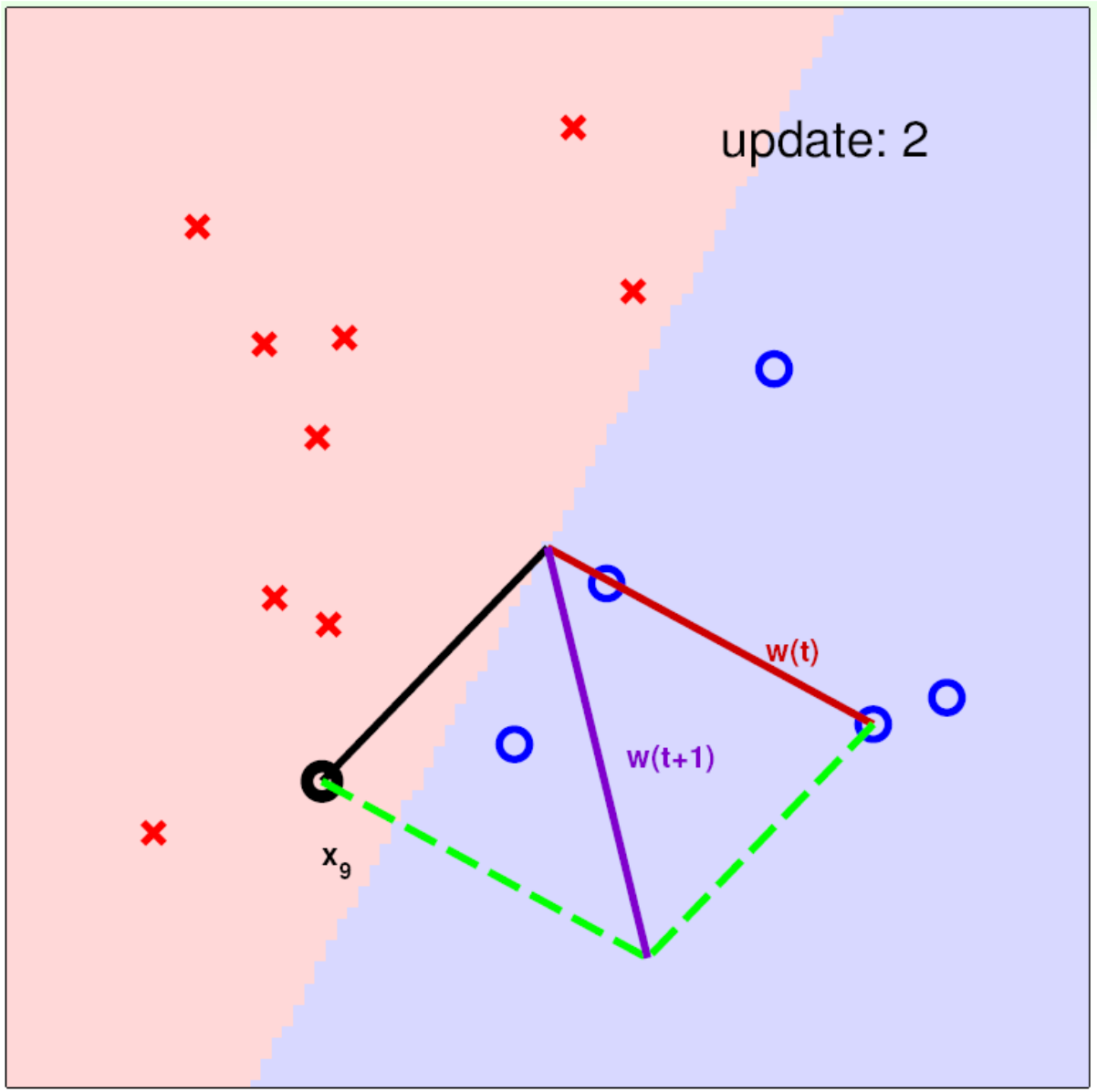


Seeing is Believing

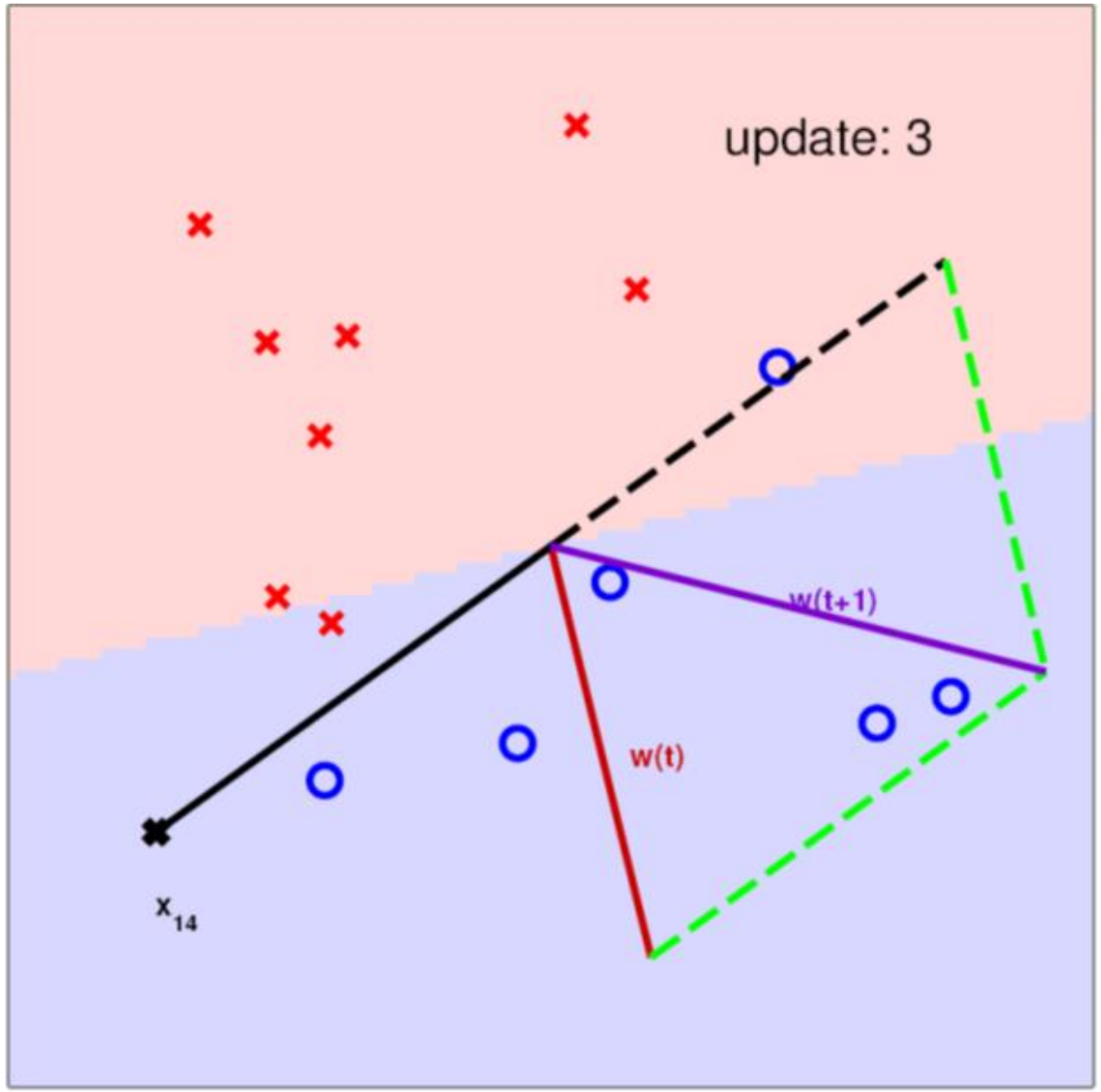


一開始沒有任何線，任取一點當錯誤點

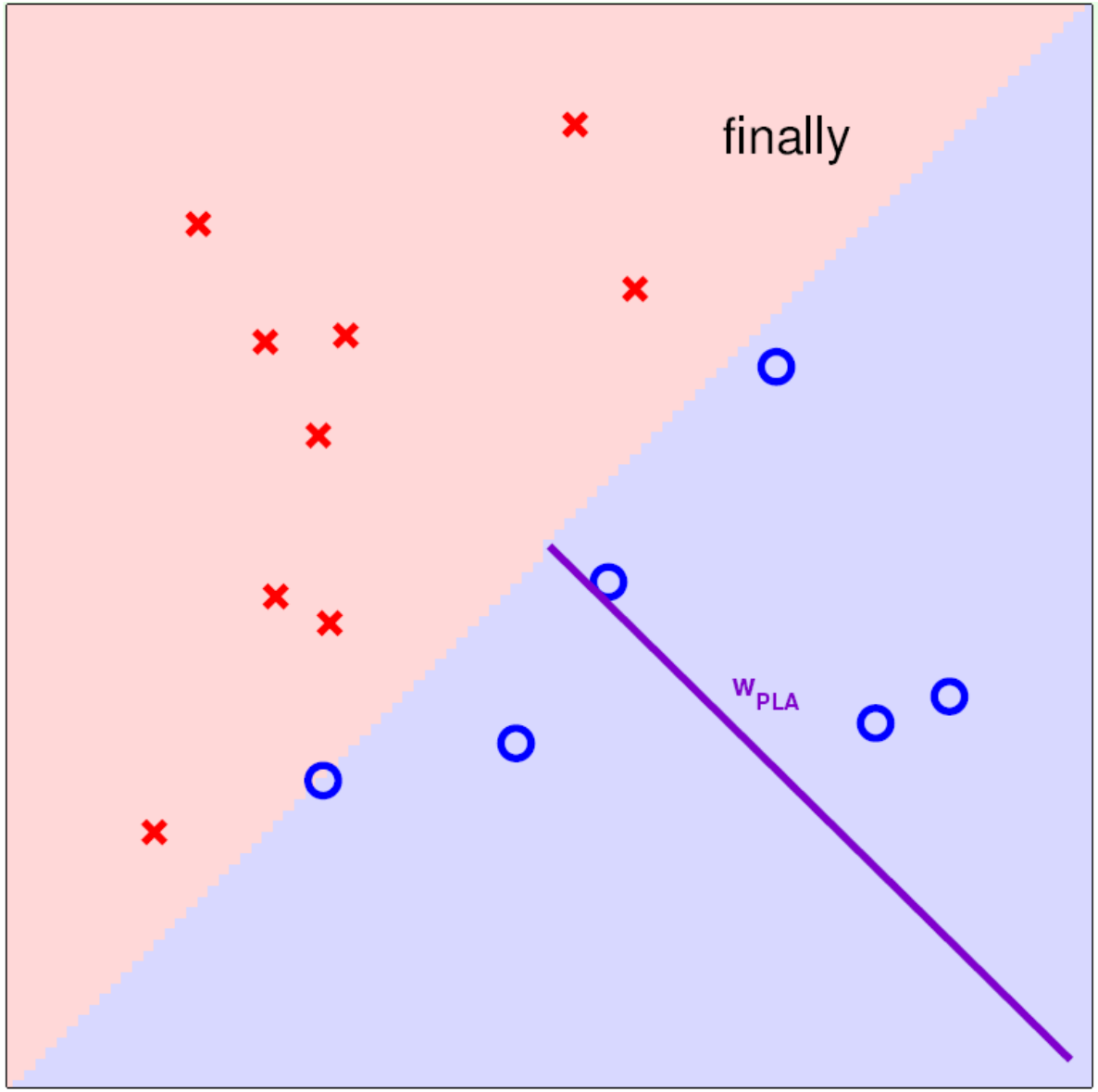
Seeing is Believing



Seeing is Believing



Seeing is Believing



How About High-dimensional Data?

age	23 years
annual salary	NTD 1,000,000
year in job	0.5 year
current debt	200,000

- For $\mathbf{x} = (x_1, x_2, \cdots, x_d)$ 'features of customer', compute a weighted 'score' and

$$\text{approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold}$$

$$\text{deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}$$

- $\mathcal{Y}$: $\{+1(\text{good}), -1(\text{bad})\}$, 0 ignored—linear formula $h \in \mathcal{H}$ are

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

Vector Form of Perceptron Hypothesis

$$
\begin{aligned}
h(\mathbf{x}) &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \\
&= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + \underbrace{(-\text{threshold})}_{w_0} \cdot \underbrace{(+1)}_{x_0}\right) \\
&= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right) \\
&= \text{sign}\left(\mathbf{w}^T \mathbf{x}\right)
\end{aligned}
$$

- each 'tall' **w** represents a hypothesis h & is multiplied with 'tall' **x** —**will use tall versions to simplify notation**

Fun Time

- Consider using a perceptron to detect spam messages.
- Assume that each email is represented by the frequency of keyword occurrence, and output +1 indicates a spam. Which keywords below shall have large positive weights in a **good perceptron** for the task?

1. coffee, tea, hamburger, steak
2. free, drug, fantastic, deal
3. machine, learning, statistics, textbook
4. national, Taiwan, university, courser

Some Remaining Issues of PLA

'correct' mistakes on $\mathcal{D}$ **until no mistakes**

Algorithmic: halt (with no mistake)?

- naïve cyclic: ??
- random cyclic: ??
- other variant: ??

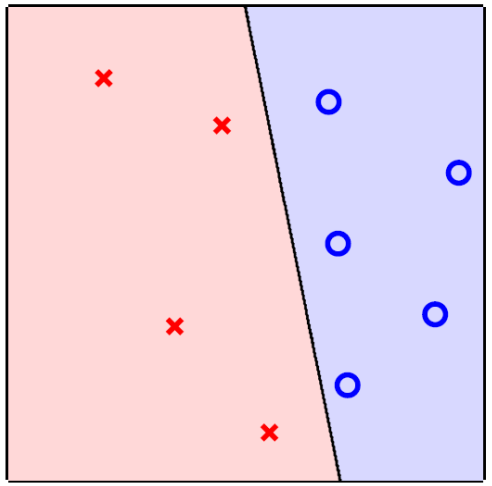
Learning: $g \approx f$?

- on $\mathcal{D}$, if halt, yes (no mistake)
- outside $\mathcal{D}$: ??
- if not halting: ??

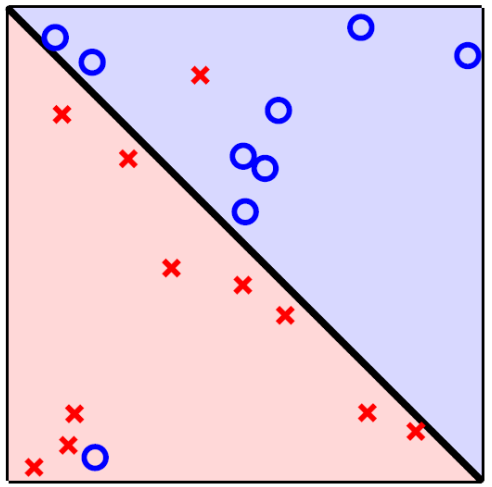
[to be shown] if (...), after 'enough' corrections,
any PLA variant halts

Linear Separability

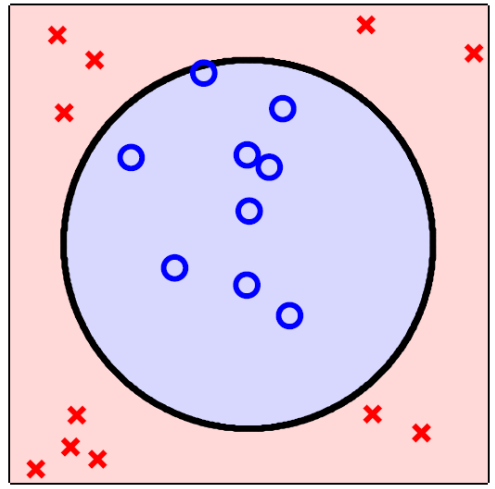
- **if** PLA halts (i.e. no more mistakes),
  (necessary condition) $\mathcal{D}$ allows some **w** to make no mistake
- call such $\mathcal{D}$ **linear separable**



(linear separable)



(not linear separable)



(not linear separable)

assume linear separable $\mathcal{D}$,
does PLA always **halt**?

More about PLA

Guarantee

as long as linear separable and correct by mistake

- inner product of $\mathbf{w}_f$ and $\mathbf{w}_t$ grows fast; length of $\mathbf{w}_t$ grows slowly
- PLA 'lines' are more and more aligned with $\mathbf{w}_f \Rightarrow$ halts

Pros

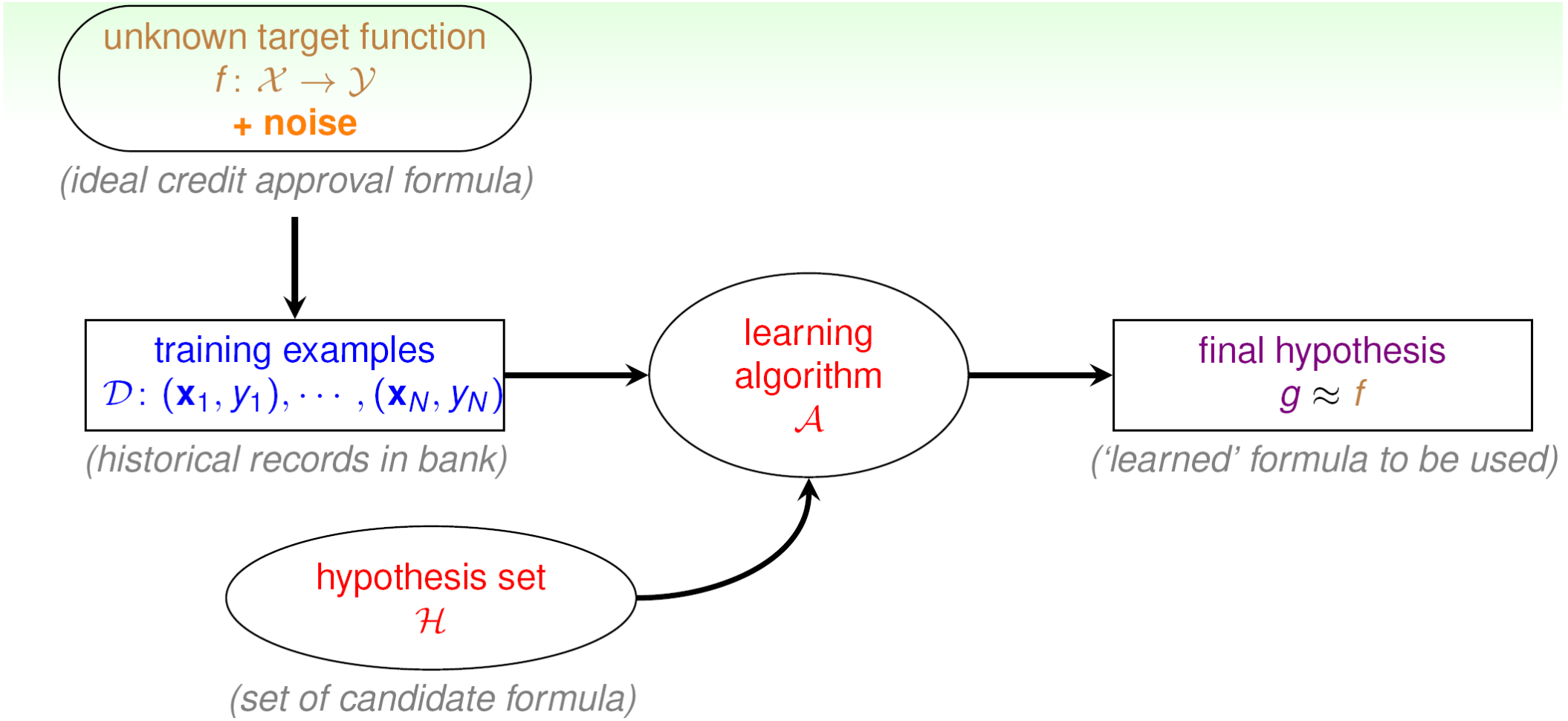
simple to implement, fast, works in any dimension d

Cons

- 'assumes' linear separable $\mathcal{D}$ to halt
  —property unknown in advance (no need for PLA if we know $\mathbf{w}_f$)
- not fully sure how long halting takes
  —though practically fast

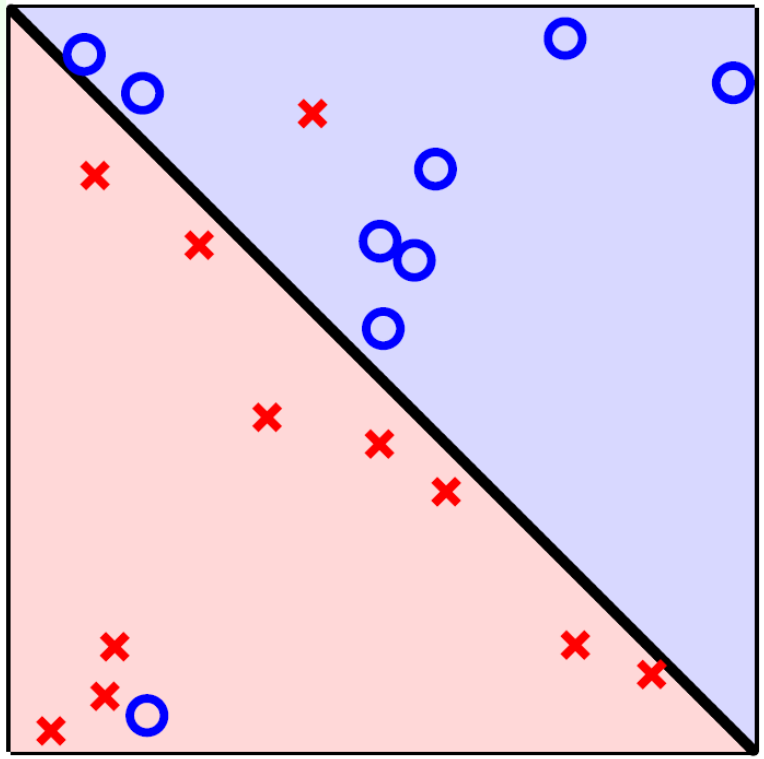
what if $\mathcal{D}$ not linear separable?

Learning with **Noisy Data**



how to at least get $g \approx f$ on **noisy** $\mathcal{D}$?

Line with Noise Tolerance



- assume 'little' noise: $y_n = f(\mathbf{x}_n)$ **usually**
- if so, $g \approx f$ on $\mathcal{D} \Leftrightarrow y_n = g(\mathbf{x}_n)$ **usually**
- how about

$$\mathbf{w}_g \leftarrow \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} \left[\!\left[y_n \neq \operatorname{sign}(\mathbf{w}^T \mathbf{x}_n) \right]\!\right]$$

—**NP-hard to solve, unfortunately**

can we modify PLA to get
an 'approximately good' g?

Pocket Algorithm

modify PLA algorithm (black lines) by **keeping best weights in pocket**

initialize pocket weights $\hat{\mathbf{w}}$

For $t = 0, 1, \cdots$

1. find a (random) mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$
2. (try to) correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$

3. **if $\mathbf{w}_{t+1}$ makes fewer mistakes than $\hat{\mathbf{w}}$, replace $\hat{\mathbf{w}}$ by $\mathbf{w}_{t+1}$**

...until **enough iterations**

return **$\hat{\mathbf{w}}$ (called $\mathbf{w}_{\text{POCKET}}$) as g**

a simple modification of PLA to find (somewhat) 'best' weights

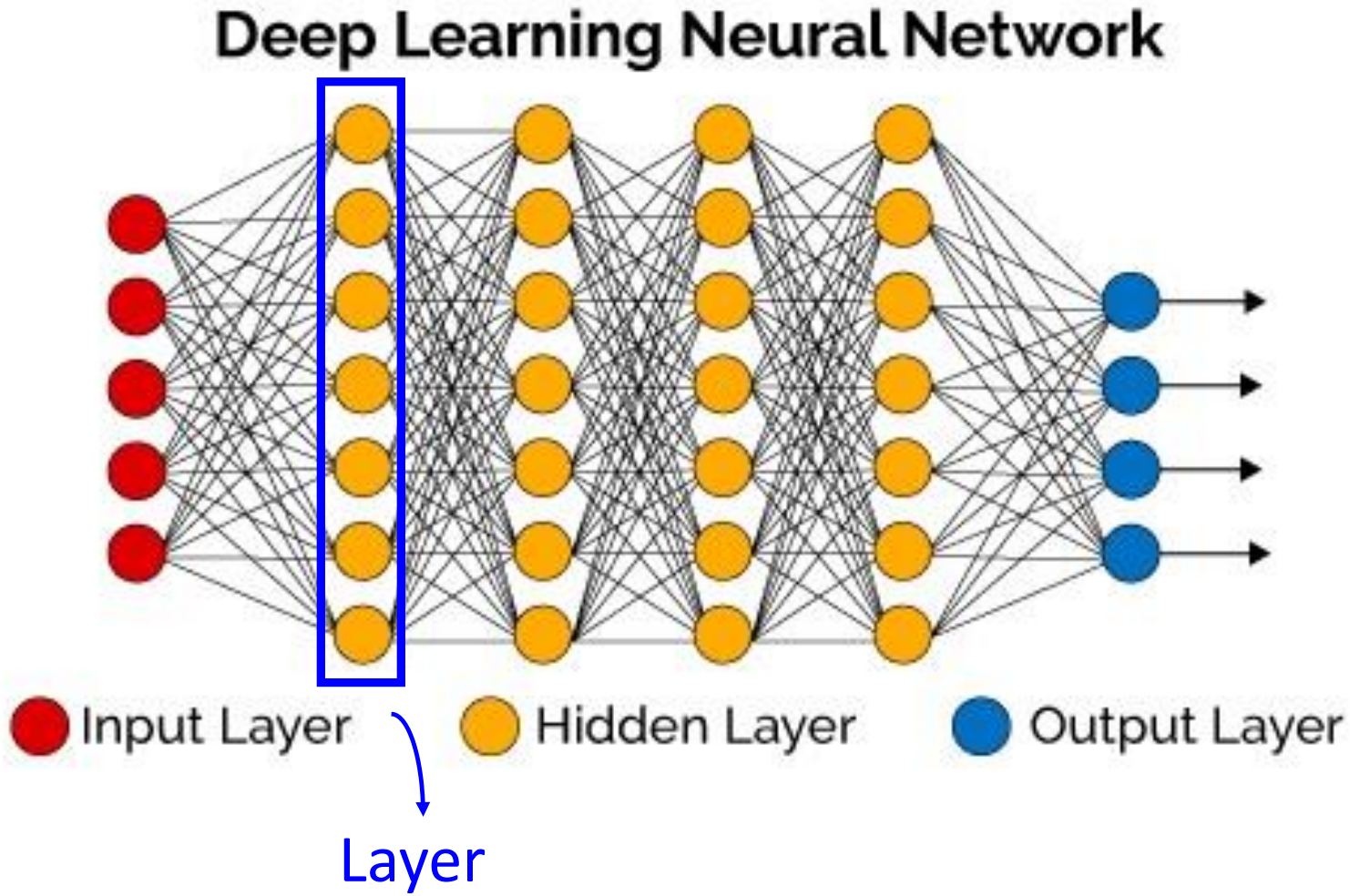
Fun Time

Should we use pocket or PLA?

Since we do not know whether $\mathcal{D}$ is linear separable in advance, we may decide to just go with pocket instead of PLA. If $\mathcal{D}$ is actually linear separable, what's the difference between the two?

1. pocket on $\mathcal{D}$ is slower than PLA
2. pocket on $\mathcal{D}$ is faster than PLA
3. pocket on $\mathcal{D}$ returns a better g in approximating f than PLA
4. pocket on $\mathcal{D}$ returns a worse g in approximating f than PLA

深度學習模型-由節點擴展到層



The Storyline

- Distilling Implicit Features: Extraction Models

Neural Network

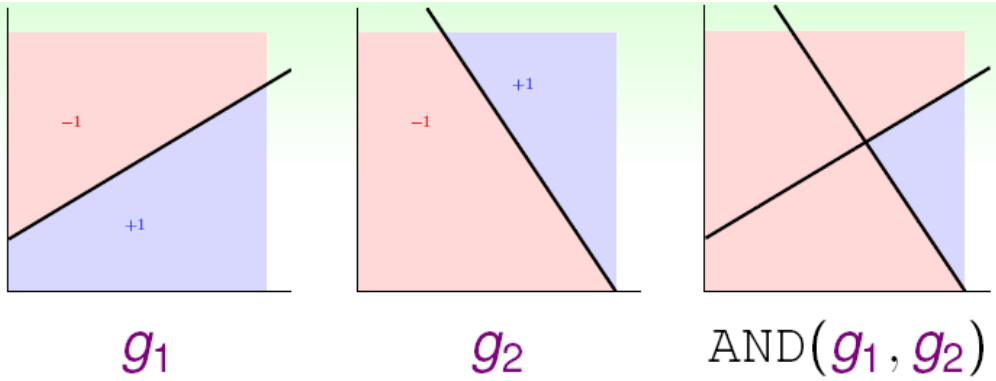
- Motivation
- Neural Network Hypothesis
- Neural Network Learning
- Optimization and Regularization

Recall : Vector Form of Perceptron Hypothesis

$$
\begin{aligned}
h(\mathbf{x}) &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \\
&= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + \underbrace{(-\text{threshold})}_{w_0} \cdot \underbrace{(+1)}_{x_0}\right) \\
&= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right) \\
&= \text{sign}\left(\mathbf{w}^T \mathbf{x}\right)
\end{aligned}
$$

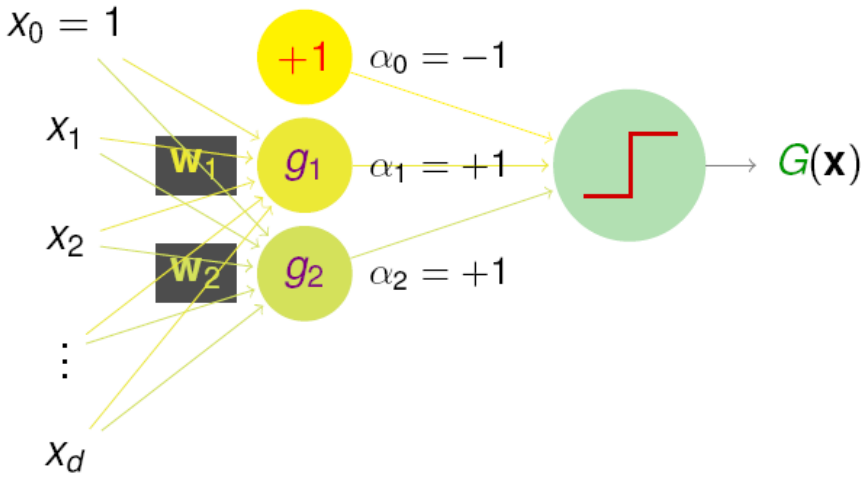
- each 'tall' **w** represents a hypothesis h & is multiplied with 'tall' **x** —**will use tall versions to simplify notation**

Logic Operations with Aggregation



	$g_2(x)$		
$g_1(x)$	G(x)	1	-1
	1		
	-1		

Replace $h(x)$ by $g(x)$ …

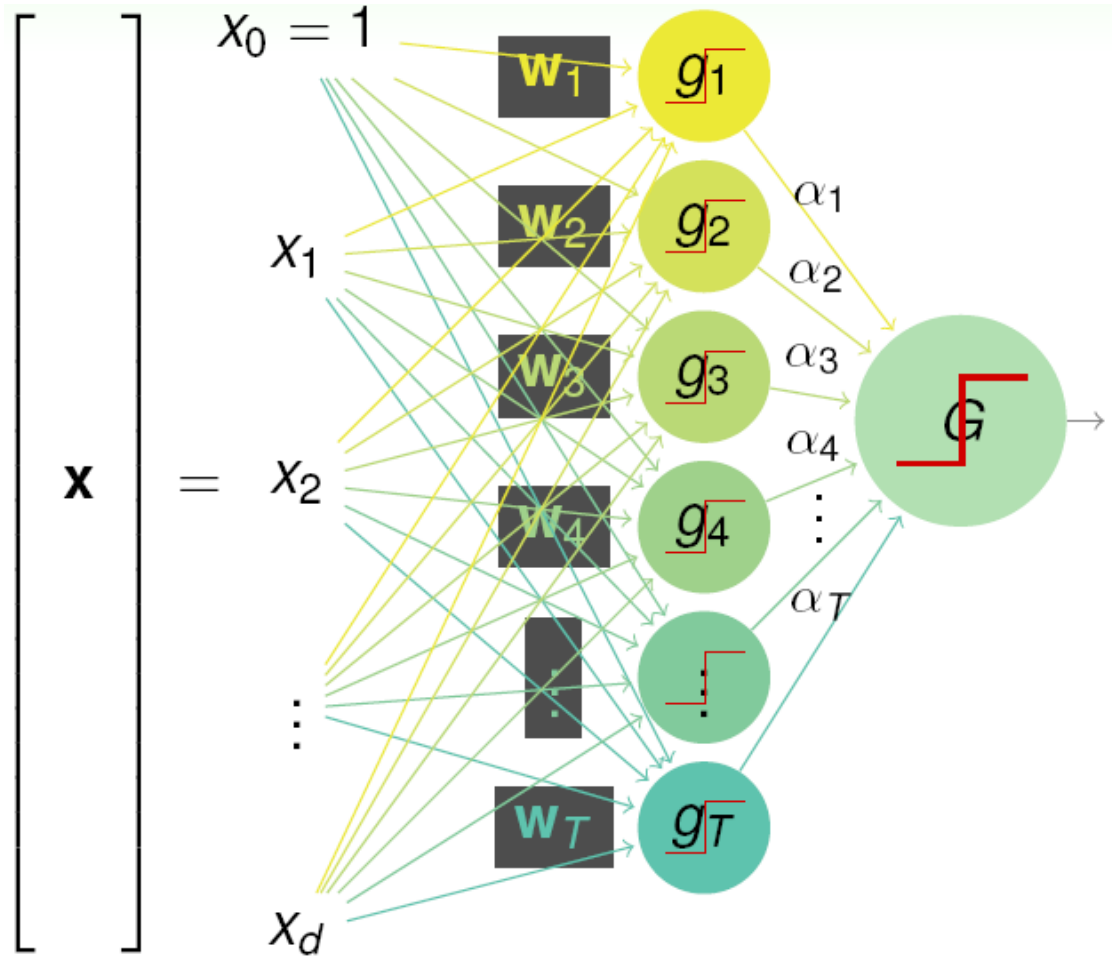


$G(\mathbf{x}) = \text{sign}\,(-1 + g_1(\mathbf{x}) + g_2(\mathbf{x}))$

- $g_1(\mathbf{x}) = g_2(\mathbf{x}) = +1$ (TRUE): $G(\mathbf{x}) = +1$ (TRUE)
- otherwise: $G(\mathbf{x}) = -1$ (FALSE)
- $G \equiv \text{AND}(g_1, g_2)$

OR, NOT can be **similarly implemented**

Linear Aggregation of Perceptrons

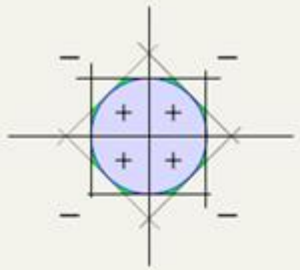


$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \underbrace{\text{sign}\left(\mathbf{w}_t^\top \mathbf{x}\right)}_{g_t(\mathbf{x})}\right)$$

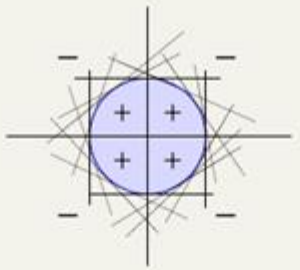
- two layers of weights: $\mathbf{w}_t$ and α
- two layers of sign functions: in g_t and in G

what boundary can G **implement**?

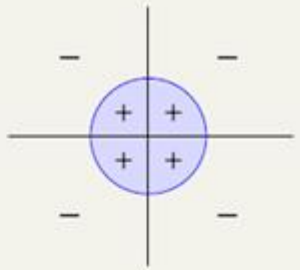
Powerfulness and Limitation



8 perceptrons

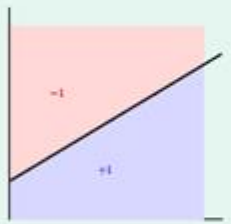


16 perceptrons

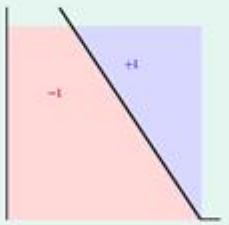


target boundary

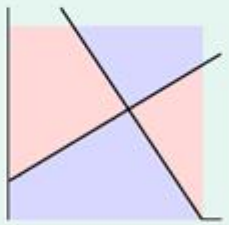
- powerfulness: enough perceptrons $\approx$ **smooth boundary**



g_1



g_2



$\texttt{XOR}(g_1, g_2)$

- limitation: XOR **not 'linear separable'** under $\phi(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}))$

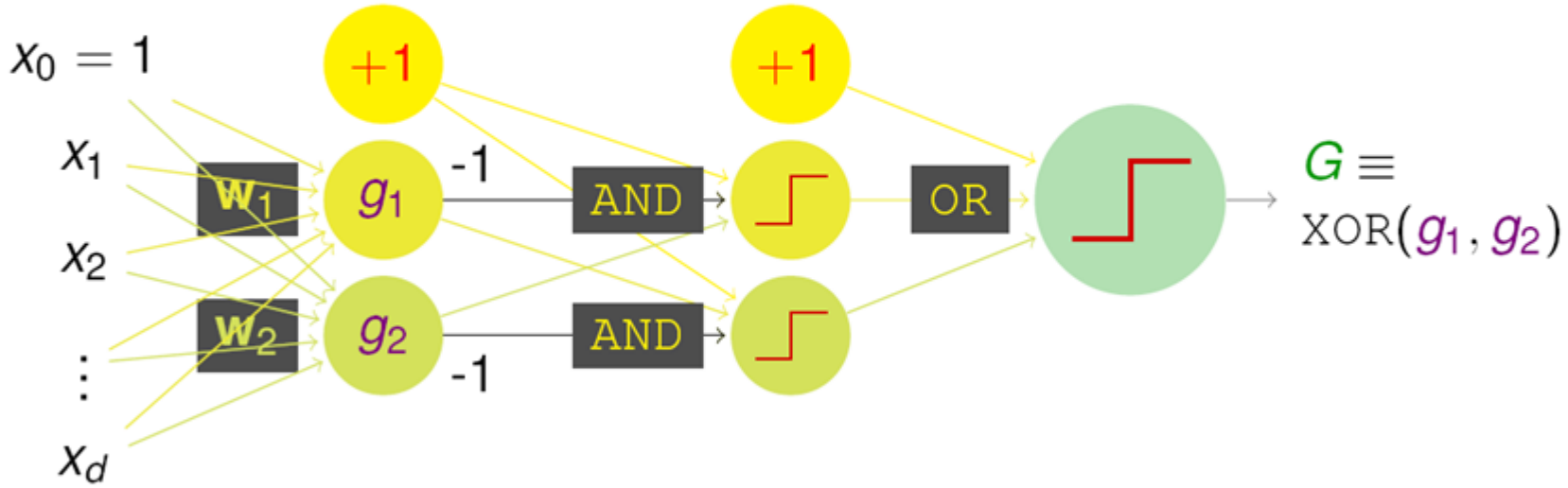
how to implement $\texttt{XOR}(g_1, g_2)$?

Multi-Layer Perceptrons (MLP): Basic Neural Network

- non-separable data: can use more **transform**
- how about one more layer of **AND transform**?

$$\text{XOR}(g_1, g_2) = \text{OR}(\text{AND}(-g_1, g_2), \text{AND}(g_1, -g_2))$$

	$g_2(x)$		
$g_1(x)$	G(x)	1	-1
	1		
	-1		



perceptron (simple)
$\Longrightarrow$ aggregation of perceptrons (powerful)
$\Longrightarrow$ **multi-layer perceptrons (more powerful)**

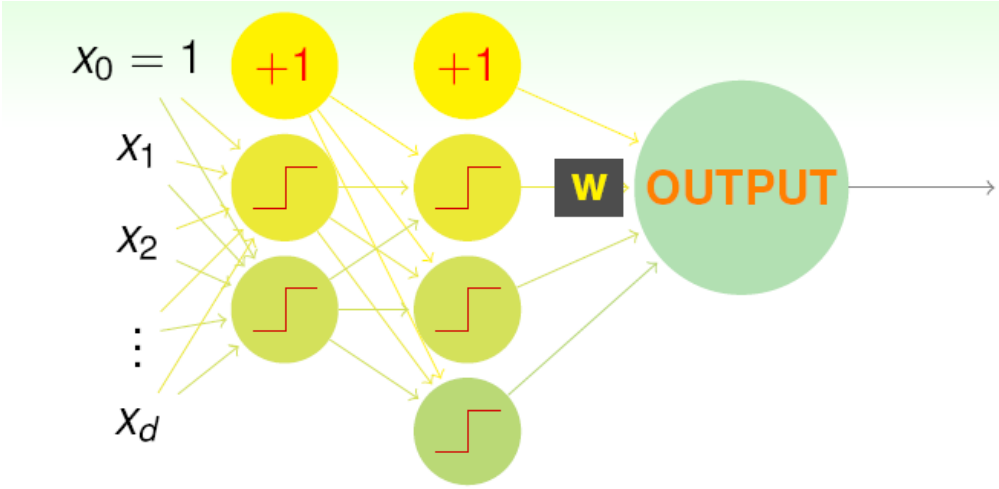
Fun Time

Let $g_0(\mathbf{x}) = +1$. Which of the following $(\alpha_0, \alpha_1, \alpha_2)$ allows $G(\mathbf{x}) = \text{sign}\left(\sum_{t=0}^{2} \alpha_t g_t(\mathbf{x})\right)$ to implement $\text{OR}(g_1, g_2)$?

1. $(-3, +1, +1)$
2. $(-1, +1, +1)$
3. $(+1, +1, +1)$
4. $(+3, +1, +1)$

	$g_2(x)$		
$g_1(x)$	G(x)	1	-1
	1		
	-1		

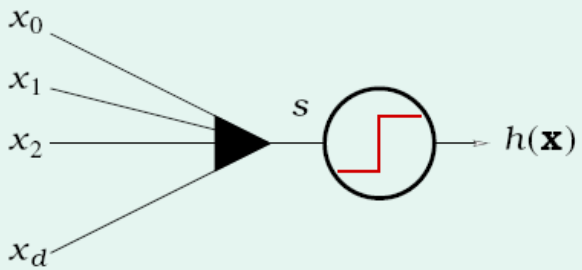
Neural Network Hypothesis: Output



- **OUTPUT**: simply a **linear model** with $s = \mathbf{w}^T \phi^{(2)}(\phi^{(1)}(\mathbf{x}))$
- any linear model can be used—**remember? :-)**

linear classification

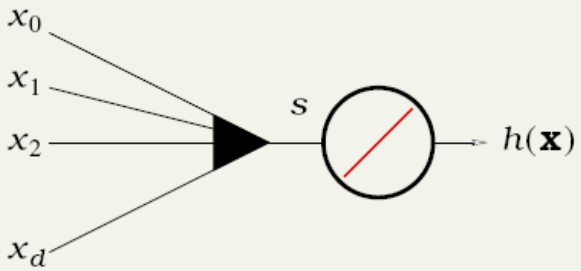
$h(\mathbf{x}) = \text{sign}(s)$



err = 0/1

linear regression

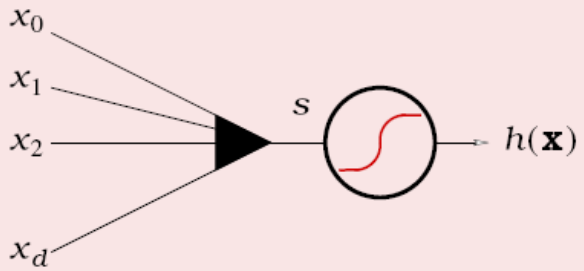
$h(\mathbf{x}) = s$



err = squared

logistic regression

$h(\mathbf{x}) = \theta(s)$

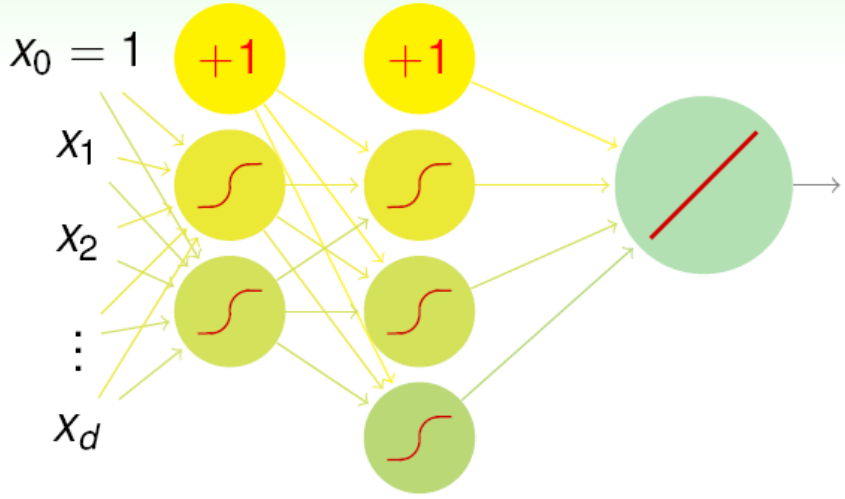


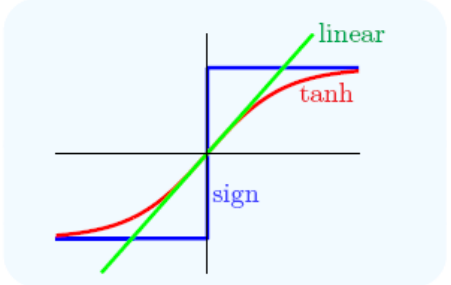
err = cross-entropy

will discuss **'regression' with squared error**

Neural Network Hypothesis: Transformation

- ⌐¬ : **transformation** function of score (signal) s
- any transformation?
  - / : whole network linear & thus **less useful**
  - ⌐¬ : discrete & thus **hard to optimize** for **w**
- popular choice of transformation: ∫ $= \tanh(s)$
  - 'analog' approximation of ⌐¬ : **easier to optimize**
  - somewhat **closer to biological** neuron
  - **not that new! :-)**

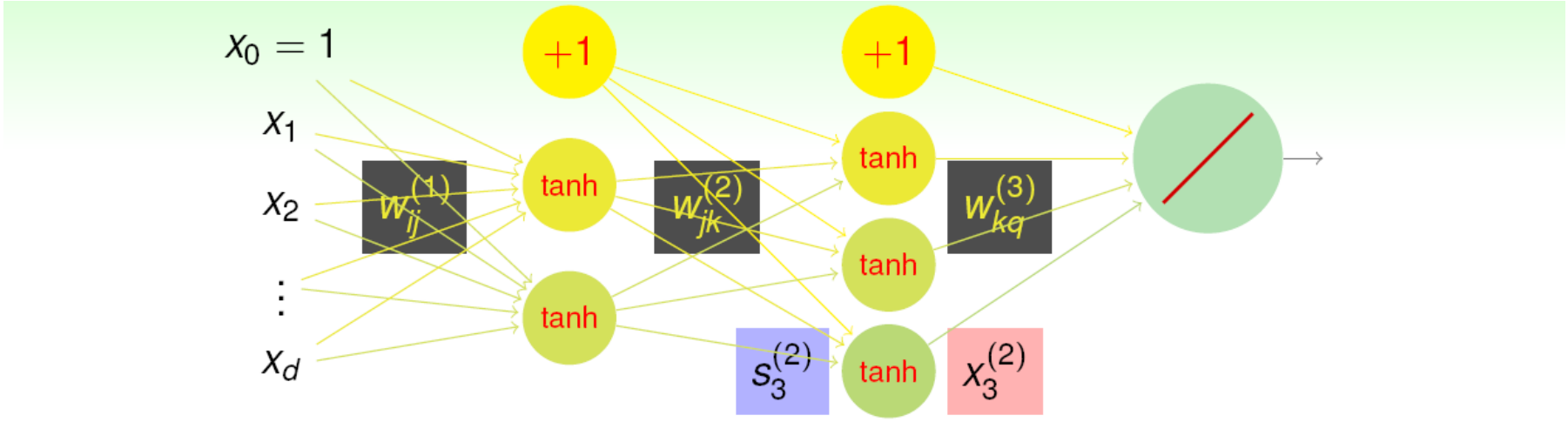




$$\tanh(s) = \frac{\exp(s) - \exp(-s)}{\exp(s) + \exp(-s)} = 2\theta(2s) - 1$$

will discuss with tanh as transformation function

Neural Network Hypothesis



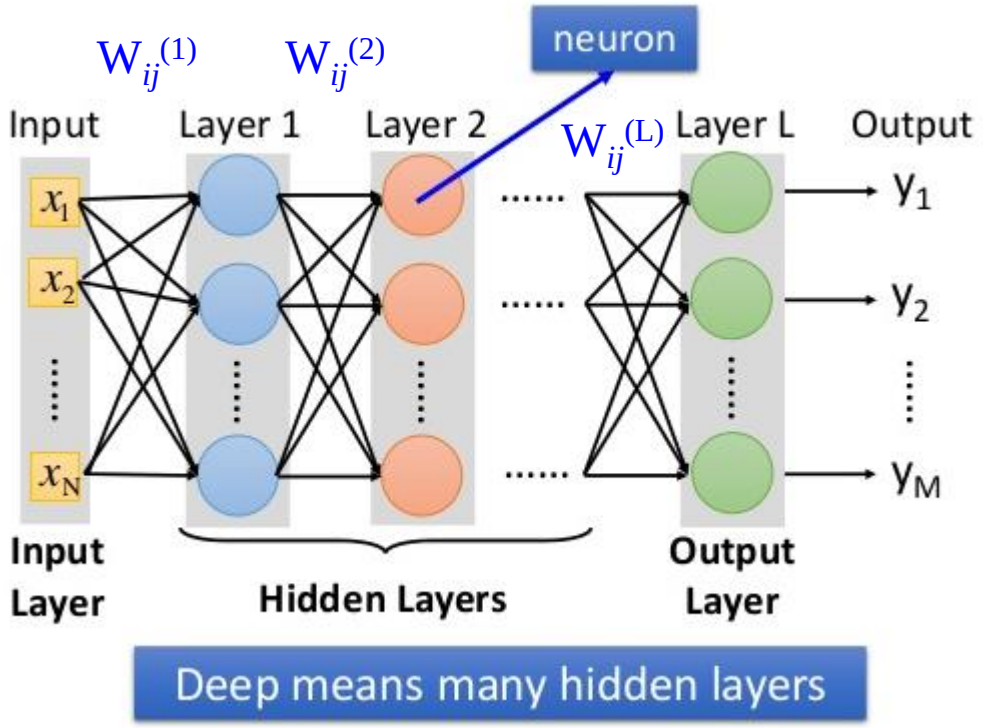
$d^{(0)}$-$d^{(1)}$-$d^{(2)}$-$\cdots$-$d^{(L)}$ Neural Network (NNet)

$$w_{ij}^{(\ell)}: \begin{cases} 1 \le \ell \le L & \text{layers} \\ 0 \le i \le d^{(\ell-1)} & \text{inputs} \\ 1 \le j \le d^{(\ell)} & \text{outputs} \end{cases}, \text{ score } s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} x_i^{(\ell-1)},$$

$$\text{transformed } x_j^{(\ell)} = \begin{cases} \tanh\left(s_j^{(\ell)}\right) & \text{if } \ell < L \\ s_j^{(\ell)} & \text{if } \ell = L \end{cases}$$

apply $\mathbf{x}$ as input layer $\mathbf{x}^{(0)}$, go through hidden layers to get $\mathbf{x}^{(\ell)}$, predict at output layer $x_1^{(L)}$

深度學習模型– 從一層開始、堆疊到很深…

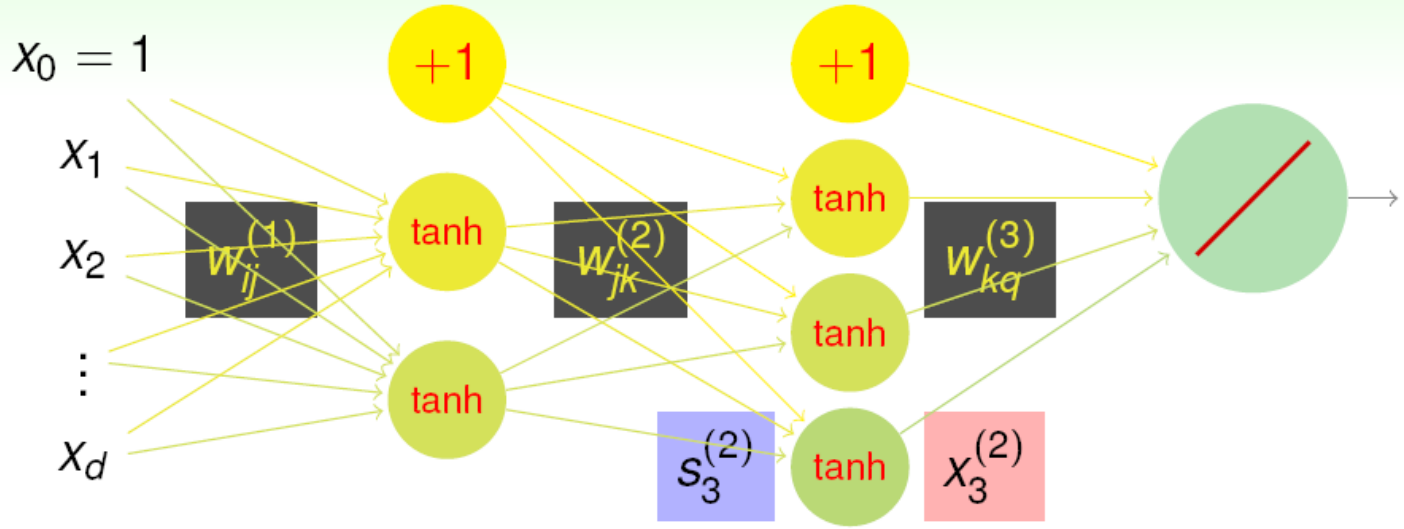


- 神經網路就是一堆函數集
  - 丟進去一堆數值x，整個網路運算出一個最好的解y出來
  - 假設某個簡單函數 $f(x)$ = 無能
  - 神經網路是一個複雜的函數

- 模型要學的是什麼?
  - 給定一堆輸入 x 和對應的輸出 y
  - 學習整個模型的權重

$$W_{ij}^{(d)},\ d = 1, \ldots, L.$$

深度學習是個黑盒子??



- each layer: **transformation** to be **learned** from data
- $\phi^{(\ell)}(\mathbf{x}) = \tanh\left(\begin{bmatrix} \sum_{i=0}^{d^{(\ell-1)}} w_{i1}^{(\ell)} x_i^{(\ell-1)} \\ \vdots \end{bmatrix}\right)$

  —whether **x** 'matches' weight vectors in pattern

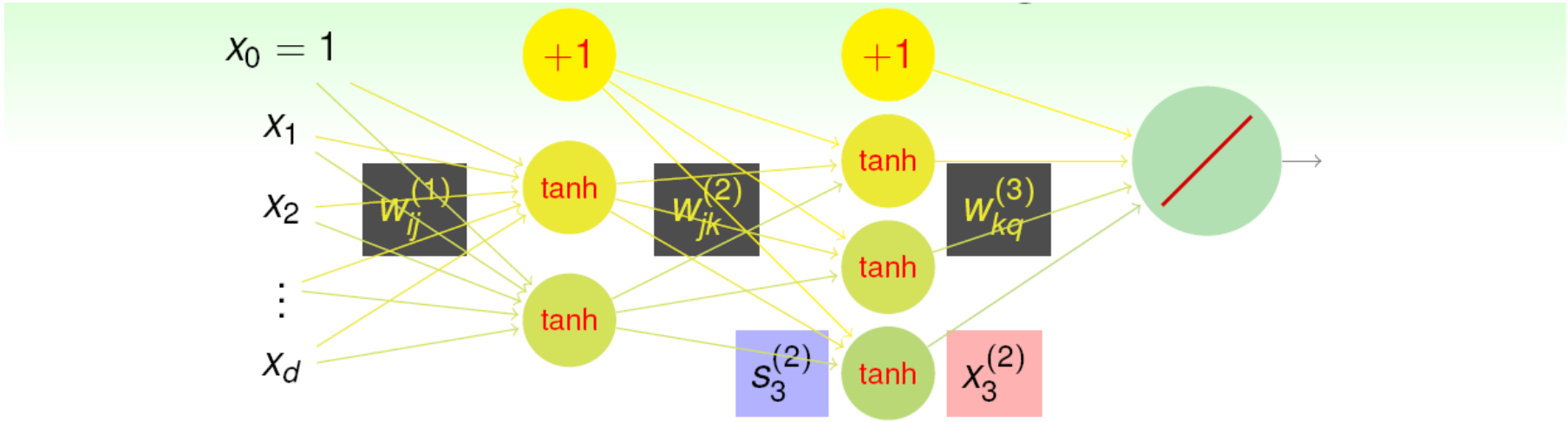
NNet: **pattern extraction** with layers of connection weights

Fun Time

How many weights $\{w_{ij}^{(\ell)}\}$ are there in a 3-5-1 NNet?

1. 9
2. 15
3. 20
4. 26

How to Learn the Weights?



- goal: learning all $\{w_{ij}^{(\ell)}\}$ to **minimize** $E_{\text{in}}\left(\{w_{ij}^{(\ell)}\}\right)$
- one hidden layer: simply **aggregation of perceptrons**
  —**gradient boosting** to determine hidden neuron one by one
- multiple hidden layers? **not easy**
- let $e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2$:
  can apply **(stochastic) GD** after computing $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$!

next: efficient computation of $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$

Computing (Output Layer) $\frac{\partial e_n}{\partial w_{i1}^{(L)}}$

$$e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2 = \left(y_n - s_1^{(L)}\right)^2 = \left(y_n - \sum_{i=0}^{d^{(L-1)}} w_{i1}^{(L)} x_i^{(L-1)}\right)^2$$

specially (output layer) $(0 \le i \le d^{(L-1)})$

$$\begin{aligned} & \frac{\partial e_n}{\partial w_{i1}^{(L)}} \\ = & \frac{\partial e_n}{\partial s_1^{(L)}} \cdot \frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}} \\ = & -2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right) \end{aligned}$$

generally $(1 \le \ell < L)$ $(0 \le i \le d^{(\ell-1)}; 1 \le j \le d^{(\ell)})$

$$\begin{aligned} & \frac{\partial e_n}{\partial w_{ij}^{(\ell)}} \\ = & \frac{\partial e_n}{\partial s_j^{(\ell)}} \cdot \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} \\ = & \delta_j^{(\ell)} \cdot \left(x_i^{(\ell-1)}\right) \end{aligned}$$

$\delta_1^{(L)} = -2\left(y_n - s_1^{(L)}\right)$, how about **others**?

Computing $\delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}}$

$$s_j^{(\ell)} \xrightarrow{\tanh} x_j^{(\ell)} \xrightarrow{w_{jk}^{(\ell+1)}} \begin{bmatrix} s_1^{(\ell+1)} \\ \vdots \\ s_k^{(\ell+1)} \\ \vdots \end{bmatrix} \Longrightarrow \cdots \Longrightarrow e_n$$

$$\delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}} = \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}}$$

$$= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\tanh'\left(s_j^{(\ell)}\right)\right)$$

$\delta_j^{(\ell)}$ can be computed **backwards** from $\delta_k^{(\ell+1)}$

Backpropagation (Backprop) Algorithm

Backprop on NNet

initialize all weights $w_{ij}^{(\ell)}$

for $t = 0, 1, \ldots, T$

1. stochastic: randomly pick $n \in \{1, 2, \cdots, N\}$
2. forward: compute all $x_i^{(\ell)}$ with $\mathbf{x}^{(0)} = \mathbf{x}_n$
3. backward: compute all $\delta_j^{(\ell)}$ subject to $\mathbf{x}^{(0)} = \mathbf{x}_n$
4. gradient descent: $w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} - \eta x_i^{(\ell-1)} \delta_j^{(\ell)}$

return $g_{\text{NNET}}(\mathbf{x}) = \left(\cdots \tanh \left(\sum_j w_{jk}^{(2)} \cdot \tanh \left(\sum_i w_{ij}^{(1)} x_i \right) \right) \right)$

$$\delta_1^{(L)} = -2 \left(y_n - s_1^{(L)} \right)$$

$$\delta_j^{(\ell)} = \sum_k \left(\delta_k^{(\ell+1)} \right) \left(w_{jk}^{(\ell+1)} \right) \left(\tanh' \left(s_j^{(\ell)} \right) \right)$$

$$w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} - \eta \frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$$

sometimes (1) to (3) is (parallelly) done many times and average($x_i^{(\ell-1)} \delta_j^{(\ell)}$) taken for update in (4), called **mini-batch**

Fun Time

According to $\frac{\partial e_n}{\partial w_{i1}^{(L)}} = -2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right)$ when would $\frac{\partial e_n}{\partial w_{i1}^{(L)}} = 0$?

1. $y_n = s_1^{(L)}$
2. $x_i^{(L-1)} = 0$
3. $s_i^{(L-1)} = 0$
4. all of the above

Reference Answer: 4

Note that $x_i^{(L-1)} = \tanh(s_i^{(L-1)}) = 0$ if and only if $s_i^{(L-1)} = 0$.

Neural Network Optimization

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \text{err}\left(\left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_{n,i}\right)\right)\right), y_n\right)$$

- generally **non-convex** when multiple hidden layers
  - not easy to reach **global minimum**
  - GD/SGD with **backprop** only gives **local minimum**
- different initial $w_{ij}^{(\ell)} \Longrightarrow$ different **local minimum**
  - somewhat '**sensitive**' to initial weights
  - **large weights** $\Longrightarrow$ **saturate** (small gradient)
  - advice: try **some random** & **small** ones

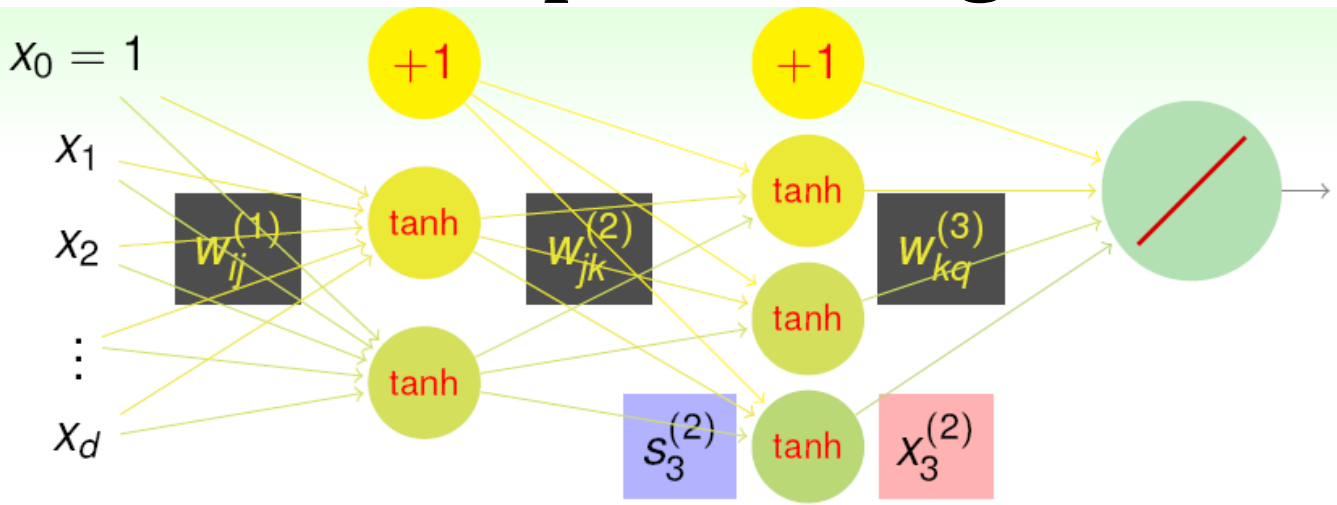
NNet: **difficult to optimize**,
but **practically works**

Challenges and Key Techniques for Deep Learning

- difficult **structural decisions**:
  - subjective with **domain knowledge**: like **convolutional NNet** for images
- high **model complexity**: 如何train得出模型?
  - no big worries if **big enough data**
  - **regularization** towards noise-tolerant: like
    - **dropout** (tolerant when network corrupted)
    - **denoising** (tolerant when input corrupted)
- hard **optimization problem**: 如何train得好模型?
  - **careful initialization** to avoid bad local minimum: called **pre-training**
- huge **computational complexity** (worsen with **big data**):
  - novel hardware/architecture: like **mini-batch with GPU**

IMHO, careful **regularization** and **initialization** are key techniques

Regularization in Deep Learning



watch out for overfitting, remember? :-)

high **model complexity**: **regularization** needed

- structural decisions/**constraints**
- weight decay or weight elimination **regularizers**
- **early stopping**

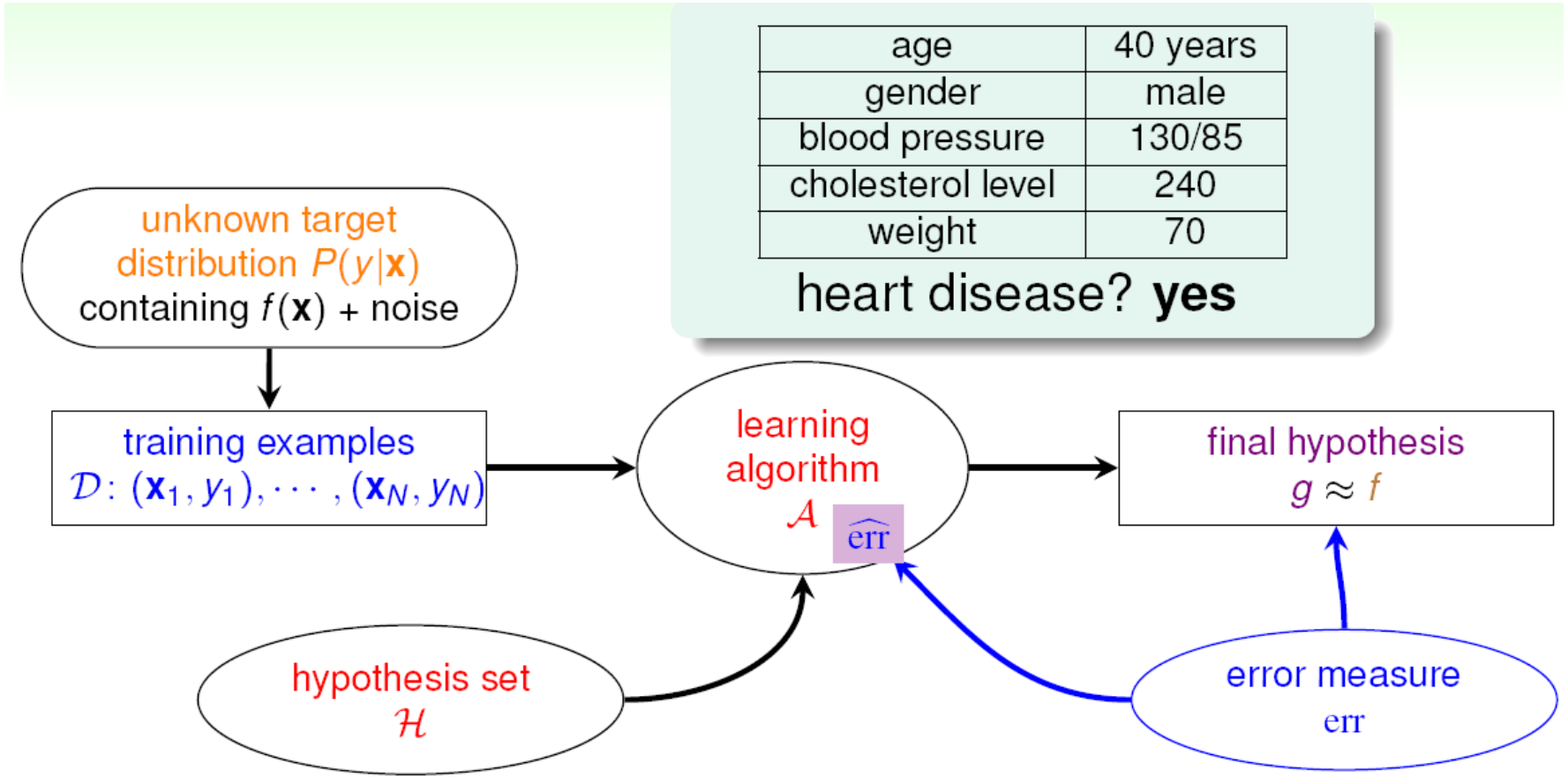
next: another **regularization** technique

Introduction of Logistic Regression

Logistic Regression

- Logistic Regression Problem
- Logistic Regression Error
- Gradient of Logistic Regression Error
- Gradient Descent

Heart Attack Prediction Problem (1/2)

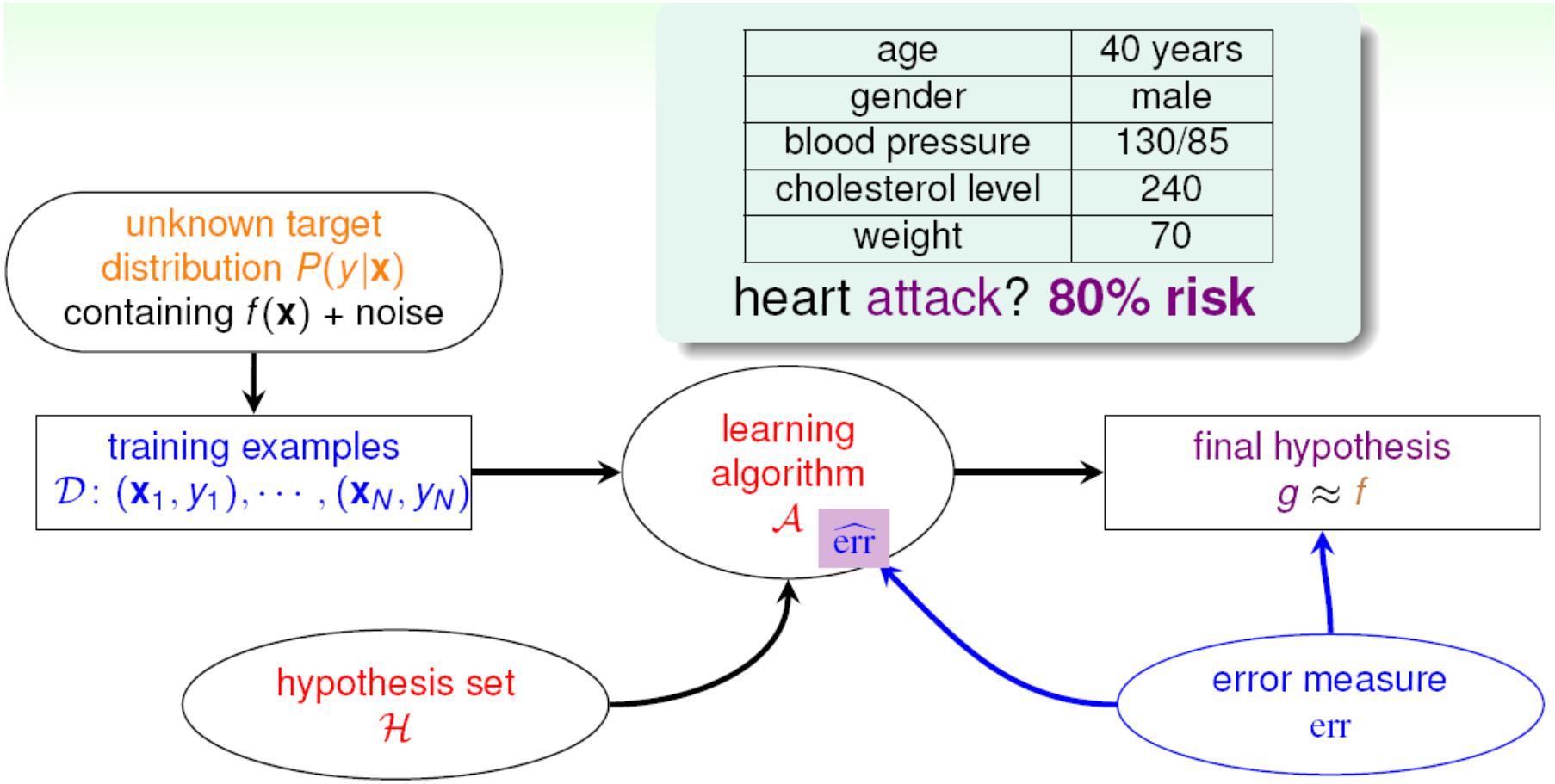


binary classification:

ideal $f(\mathbf{x}) = \text{sign}\left(P(+1|\mathbf{x}) - \frac{1}{2}\right) \in \{-1, +1\}$

because of classification err

Heart Attack Prediction Problem (2/2)



'soft' binary classification:

$$f(\mathbf{x}) = P(+1|\mathbf{x}) \in [0, 1]$$

Soft Binary Classification

target function $f(\mathbf{x}) = P(+1|\mathbf{x}) \in [0, 1]$

ideal (noiseless) data

$$\begin{pmatrix} \mathbf{x}_1, y'_1 & = 0.9 & = P(+1|\mathbf{x}_1) \end{pmatrix}$$
$$\begin{pmatrix} \mathbf{x}_2, y'_2 & = 0.2 & = P(+1|\mathbf{x}_2) \end{pmatrix}$$
$$\vdots$$
$$\begin{pmatrix} \mathbf{x}_N, y'_N & = 0.6 & = P(+1|\mathbf{x}_N) \end{pmatrix}$$

actual (noisy) data

$$\begin{pmatrix} \mathbf{x}_1, y_1 & = \circ & \sim P(y|\mathbf{x}_1) \end{pmatrix}$$
$$\begin{pmatrix} \mathbf{x}_2, y_2 & = \times & \sim P(y|\mathbf{x}_2) \end{pmatrix}$$
$$\vdots$$
$$\begin{pmatrix} \mathbf{x}_N, y_N & = \times & \sim P(y|\mathbf{x}_N) \end{pmatrix}$$

same data as hard binary classification,
different **target function**

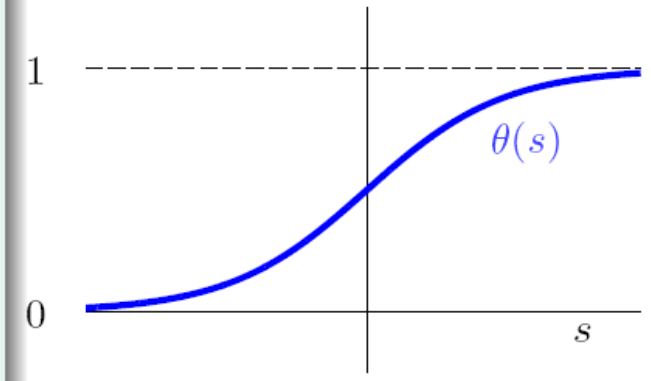
Logistic Hypothesis

age	40 years
gender	male
blood pressure	130/85
cholesterol level	240

- For $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)$ 'features of patient', calculate a weighted 'risk score':

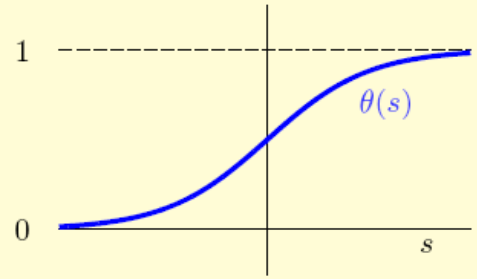
$$s = \sum_{i=0}^{d} w_i x_i$$

- convert the score to estimated probability by logistic function $\theta(s)$



logistic hypothesis: $h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x})$

Logistic Function



$\theta(-\infty) = 0$; $\theta(0) = \frac{1}{2}$; $\theta(\infty) = 1$

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

—smooth, monotonic, sigmoid function of s

logistic regression: use

$$h(\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$$

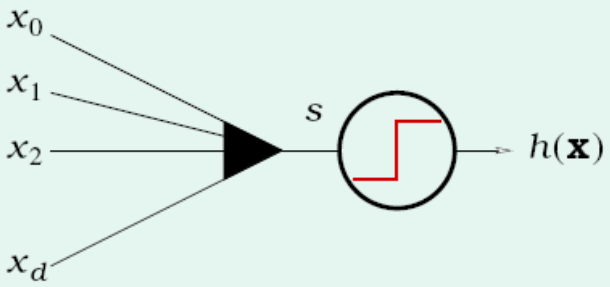
to approximate target function $f(\mathbf{x}) = P(+1|\mathbf{x})$

Three Linear Models

linear scoring function: $s = \mathbf{w}^T\mathbf{x}$

linear classification

$h(\mathbf{x}) = \text{sign}(s)$

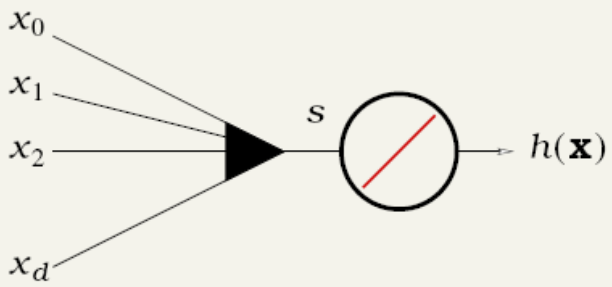


plausible err = 0/1

(small flipping noise)

linear regression

$h(\mathbf{x}) = s$

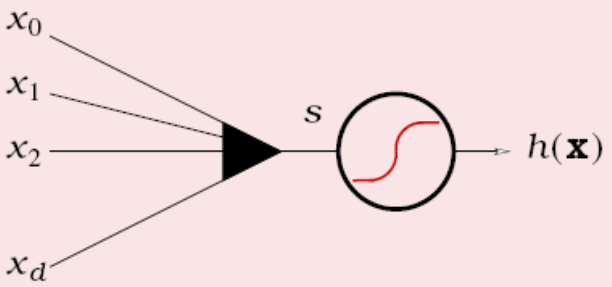


friendly err = squared

(easy to minimize)

logistic regression

$h(\mathbf{x}) = \theta(s)$



err = ?

how to define

$E_{\text{in}}(\mathbf{w})$ **for logistic regression**?

Likelihood

target function $f(\mathbf{x}) = P(+1|\mathbf{x})$ $\Leftrightarrow$ $P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases}$

consider $\mathcal{D} = \{(\mathbf{x}_1, \circ), (\mathbf{x}_2, \times), \ldots, (\mathbf{x}_N, \times)\}$

probability that f generates $\mathcal{D}$

$$P(\mathbf{x}_1)P(\circ|\mathbf{x}_1) \times$$
$$P(\mathbf{x}_2)P(\times|\mathbf{x}_2) \times$$
$$\ldots$$
$$P(\mathbf{x}_N)P(\times|\mathbf{x}_N)$$

likelihood that h generates $\mathcal{D}$

$$P(\mathbf{x}_1)h(\mathbf{x}_1) \times$$
$$P(\mathbf{x}_2)(1 - h(\mathbf{x}_2)) \times$$
$$\ldots$$
$$P(\mathbf{x}_N)(1 - h(\mathbf{x}_N))$$

- if $h \approx f$,
  then likelihood(h) $\approx$ probability using f
- probability using f usually **large**

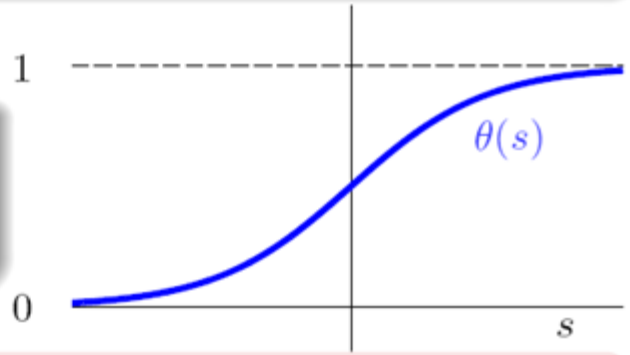
Likelihood of Logistic Hypothesis

$$\text{likelihood}(h) \approx (\text{probability using } f) \approx \textbf{large}$$

$$g = \underset{h}{\text{argmax}} \ \ \text{likelihood}(h)$$

when logistic: $h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x})$

$$1 - h(\mathbf{x}) = h(-\mathbf{x})$$



$$\text{likelihood}(h) = P(\mathbf{x}_1)h(\mathbf{x}_1) \times P(\mathbf{x}_2)(1 - h(\mathbf{x}_2)) \times \ldots P(\mathbf{x}_N)(1 - h(\mathbf{x}_N))$$

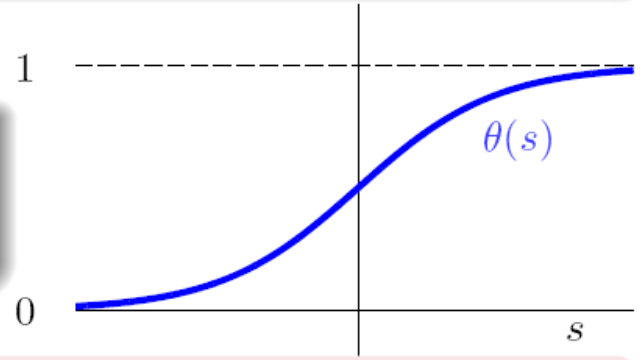
Likelihood of Logistic Hypothesis

$$\text{likelihood}(h) \approx (\text{probability using } f) \approx \textbf{large}$$

$$g = \underset{h}{\text{argmax}} \ \ \text{likelihood}(h)$$

when logistic: $h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x})$

$$1 - h(\mathbf{x}) = h(-\mathbf{x})$$



$$\text{likelihood}(h) = P(\mathbf{x}_1)h(+\mathbf{x}_1) \times P(\mathbf{x}_2)h(-\mathbf{x}_2) \times \ldots P(\mathbf{x}_N)h(-\mathbf{x}_N)$$

$$\text{likelihood}(\text{logistic } h) \propto \prod_{n=1}^{N} h(y_n\mathbf{x}_n)$$

Cross-Entropy Error

$$\max_{h} \quad \text{likelihood(logistic } h\text{)} \propto \prod_{n=1}^{N} h(y_n \mathbf{x}_n)$$

$$\max_{\mathbf{w}} \quad \text{likelihood}(\mathbf{w}) \propto \prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)$$

$$\max_{\mathbf{w}} \quad \ln \prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)$$

Cross-Entropy Error

$$\min_{\mathbf{w}} \quad \frac{1}{N}\sum_{n=1}^{N} \; - \ln\theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)$$

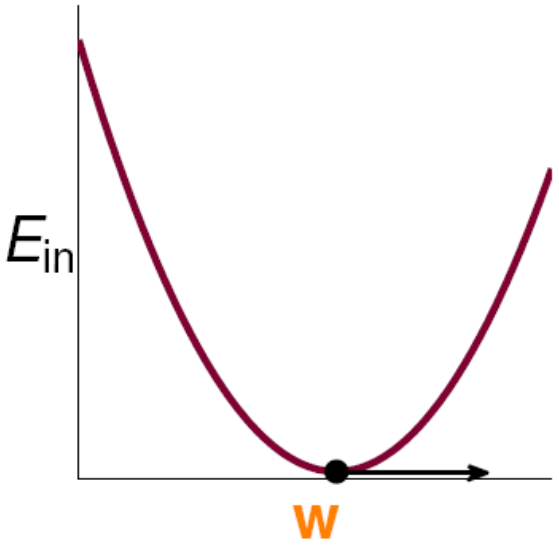
$$\theta(s) = \frac{1}{1+\exp(-s)} \quad : \quad \min_{\mathbf{w}} \quad \frac{1}{N}\sum_{n=1}^{N} \ln\left(1+\exp(-y_n \mathbf{w}^T \mathbf{x}_n)\right)$$

$$\implies \min_{\mathbf{w}} \quad \underbrace{\frac{1}{N}\sum_{n=1}^{N} \mathrm{err}(\mathbf{w}, \mathbf{x}_n, y_n)}_{E_{\text{in}}(\mathbf{w})}$$

$\mathrm{err}(\mathbf{w}, \mathbf{x}, y) = \ln\left(1+\exp(-y\mathbf{w}\mathbf{x})\right)$:
cross-entropy error

Minimizing $E_{in}(\mathbf{w})$

$$\min_{\mathbf{w}} \quad E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)\right)$$



- $E_{\text{in}}(\mathbf{w})$: continuous, differentiable, twice-differentiable, **convex**
- how to minimize? locate **valley**

want $\nabla E_{\text{in}}(\mathbf{w}) = \mathbf{0}$

first: derive $\nabla E_{\text{in}}(\mathbf{w})$

The Gradient $\nabla E_{\text{in}}(\mathbf{w})$

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(\underbrace{1+\exp(\overbrace{-y_n \mathbf{w}^T \mathbf{x}_n}^{\bigcirc})}_{\square}\right)$$

$$\frac{\partial E_{\text{in}}(\mathbf{w})}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N} \left(\frac{\partial \ln(\square)}{\partial \square}\right)\left(\frac{\partial(1+\exp(\bigcirc))}{\partial \bigcirc}\right)\left(\frac{\partial -y_n \mathbf{w}^T \mathbf{x}_n}{\partial w_i}\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N} \left(\frac{1}{\square}\right)\left(\exp(\bigcirc)\right)\left(-y_n x_{n,i}\right)$$

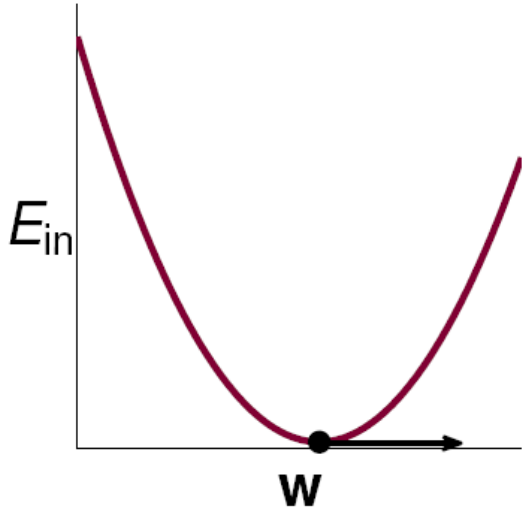
$$= \frac{1}{N}\sum_{n=1}^{N} \left(\frac{\exp(\bigcirc)}{1+\exp(\bigcirc)}\right)\left(-y_n x_{n,i}\right) = \frac{1}{N}\sum_{n=1}^{N} \theta(\bigcirc)\left(-y_n x_{n,i}\right)$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \theta\left(-y_n \mathbf{w}^T \mathbf{x}_n\right)\left(-y_n \mathbf{x}_n\right)$$

Minimizing $E_{in}(\mathbf{w})$

$$\min_{\mathbf{w}} \quad E_{\text{in}}(\mathbf{w}) \quad = \quad \frac{1}{N}\sum_{n=1}^{N} \ln\left(1+\exp(-y_n \mathbf{w}^T \mathbf{x}_n)\right)$$

$$\text{want } \nabla E_{\text{in}}(\mathbf{w}) \quad = \quad \frac{1}{N}\sum_{n=1}^{N} \theta\left(-y_n \mathbf{w}^T \mathbf{x}_n\right)\left(-y_n \mathbf{x}_n\right) = \mathbf{0}$$



scaled θ-weighted sum of $-y_n \mathbf{x}_n$

- all $\theta(\cdot) = 0$: only if $y_n \mathbf{w}^T \mathbf{x}_n \gg 0$
  —linear separable $\mathcal{D}$
- weighted sum = $\mathbf{0}$:
  non-linear equation of $\mathbf{w}$

closed-form solution? no :-(

PLA Revisited: Iterative Optimization

PLA: start from some $\mathbf{w}_0$ (say, $\mathbf{0}$), and 'correct' its mistakes on $\mathcal{D}$

For $t = 0, 1, \ldots$

1. find a mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$

$$\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. (try to) correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$

1. (equivalently) pick some n, and update $\mathbf{w}_t$ by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + [\![\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_n\right) \neq y_n]\!]\, y_n \mathbf{x}_n$$

when stop, return last $\mathbf{w}$ as g

PLA Revisited: Iterative Optimization

PLA: start from some $\mathbf{w}_0$ (say, $\mathbf{0}$), and 'correct' its mistakes on $\mathcal{D}$

For $t = 0, 1, \ldots$

1. (equivalently) pick some n, and update $\mathbf{w}_t$ by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \underbrace{1}_{\eta} \cdot \underbrace{\left([\![\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_n\right) \neq y_n]\!] \cdot y_n \mathbf{x}_n \right)}_{\mathbf{v}}$$

when stop, return last $\mathbf{w}$ as g

choice of $(\eta, \mathbf{v})$ and stopping condition defines **iterative optimization approach**

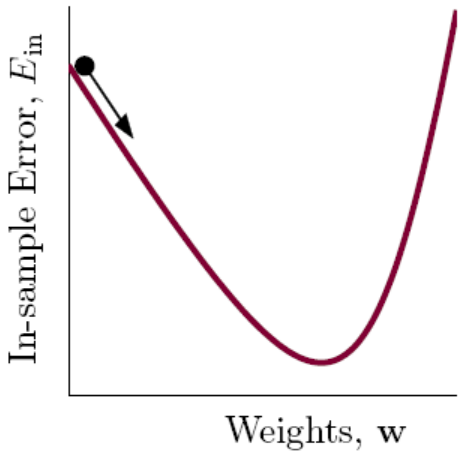
Iterative Optimization

For $t = 0, 1, \ldots$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \mathbf{v}$$

when stop, return last $\mathbf{w}$ as g

- PLA: $\mathbf{v}$ comes from mistake correction
- smooth $E_{\text{in}}(\mathbf{w})$ for logistic regression: choose $\mathbf{v}$ to get the ball roll '**downhill**'?
  - direction $\mathbf{v}$: (assumed) of unit length
  - step size η: (assumed) positive



a greedy approach for some given $\eta > 0$:

$$\min_{\|\mathbf{v}\|=1} E_{\text{in}}(\underbrace{\mathbf{w}_t + \eta \mathbf{v}}_{\mathbf{w}_{t+1}})$$

Linear Approximation

a greedy approach for some given $\eta > 0$:

$$\min_{\|\mathbf{v}\|=1} \quad E_{\text{in}}(\mathbf{w}_t + \eta\mathbf{v})$$

- still non-linear optimization, now **with constraints**
  —not any easier than $\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w})$
- local approximation by linear formula makes problem easier

$$E_{\text{in}}(\mathbf{w}_t + \eta\mathbf{v}) \approx E_{\text{in}}(\mathbf{w}_t) + \eta\mathbf{v}^T\nabla E_{\text{in}}(\mathbf{w}_t)$$

if η really small (Taylor expansion)

an **approximate** greedy approach for some given **small** η:

$$\min_{\|\mathbf{v}\|=1} \quad \underbrace{E_{\text{in}}(\mathbf{w}_t)}_{\text{known}} + \underbrace{\eta}_{\text{given positive}} \mathbf{v}^T \underbrace{\nabla E_{\text{in}}(\mathbf{w}_t)}_{\text{known}}$$

Gradient Descent

an **approximate** greedy approach for some given **small** η:

$$\min_{\|\mathbf{v}\|=1} \underbrace{E_{\text{in}}(\mathbf{w}_t)}_{\text{known}} + \underbrace{\eta}_{\text{given positive}} \mathbf{v}^T \underbrace{\nabla E_{\text{in}}(\mathbf{w}_t)}_{\text{known}}$$

- optimal $\mathbf{v}$: opposite direction of $\nabla E_{\text{in}}(\mathbf{w}_t)$

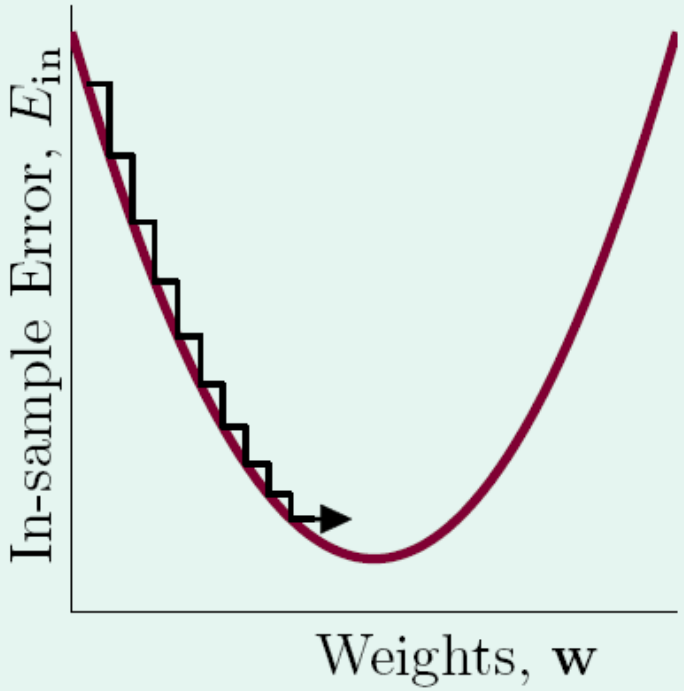
$$\mathbf{v} = -\frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$$

- gradient descent: for **small** η, $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$

gradient descent:
a simple & popular optimization tool

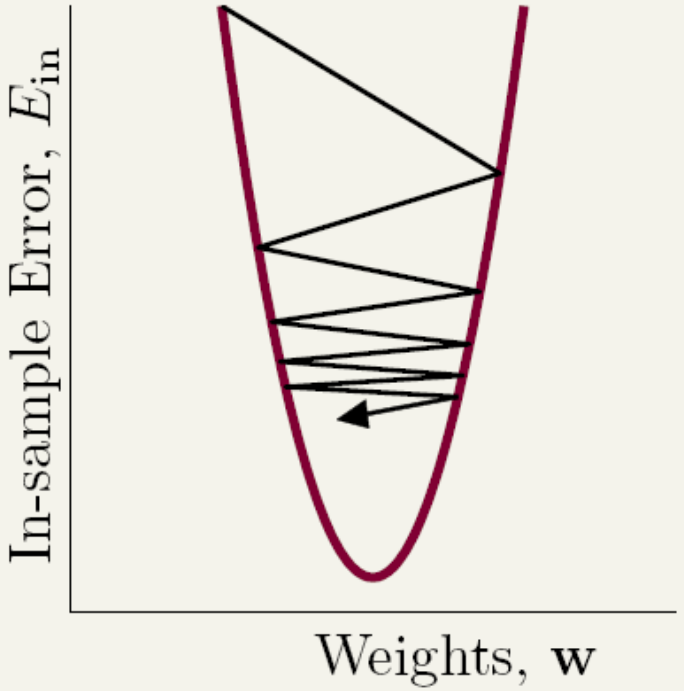
Choice of η

too small



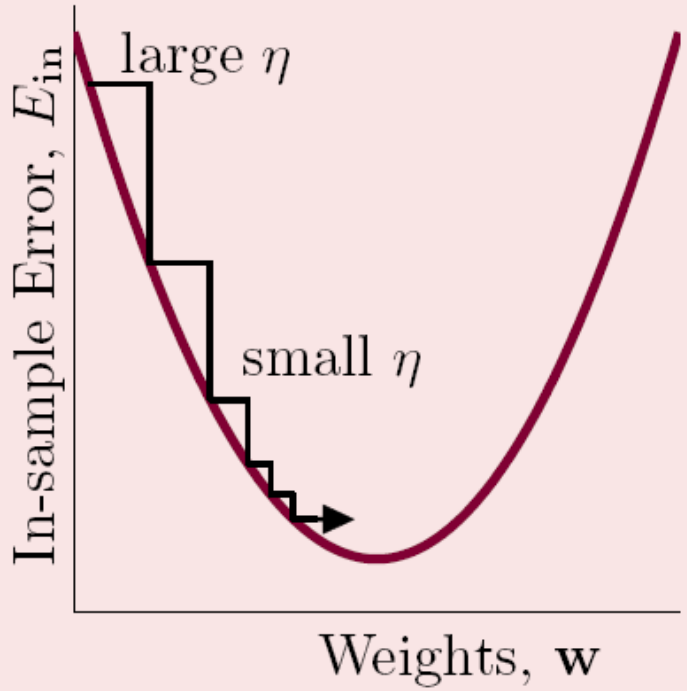
too slow :-(

too large



too unstable :-(

just right



use changing η

η better be **monotonic of** $\|\nabla E_{\text{in}}(\mathbf{w}_t)\|$

Simple Heuristic for **Changing** η

η better be **monotonic of** $\|\nabla E_{\text{in}}(\mathbf{w}_t)\|$

- if red $\eta \propto \|\nabla E_{\text{in}}(\mathbf{w}_t)\|$ by ratio purple η

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$$

$$\Big\| \quad \mathbf{w}_t - \eta \nabla E_{\text{in}}(\mathbf{w}_t)$$

- call purple η the fixed learning rate

fixed learning rate gradient descent:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla E_{\text{in}}(\mathbf{w}_t)$$

Putting Everything Together

Logistic Regression Algorithm

initialize $\mathbf{w}_0$

For $t = 0, 1, \cdots$

1. compute

$$\nabla E_{\text{in}}(\mathbf{w}_t) = \frac{1}{N}\sum_{n=1}^{N} \theta\left(-y_n \mathbf{w}_t^T \mathbf{x}_n\right)\left(-y_n \mathbf{x}_n\right)$$

2. update by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla E_{\text{in}}(\mathbf{w}_t)$$

...until $\nabla E_{\text{in}}(\mathbf{w}_{t+1}) = 0$ or enough iterations

return last $\mathbf{w}_{t+1}$ as g

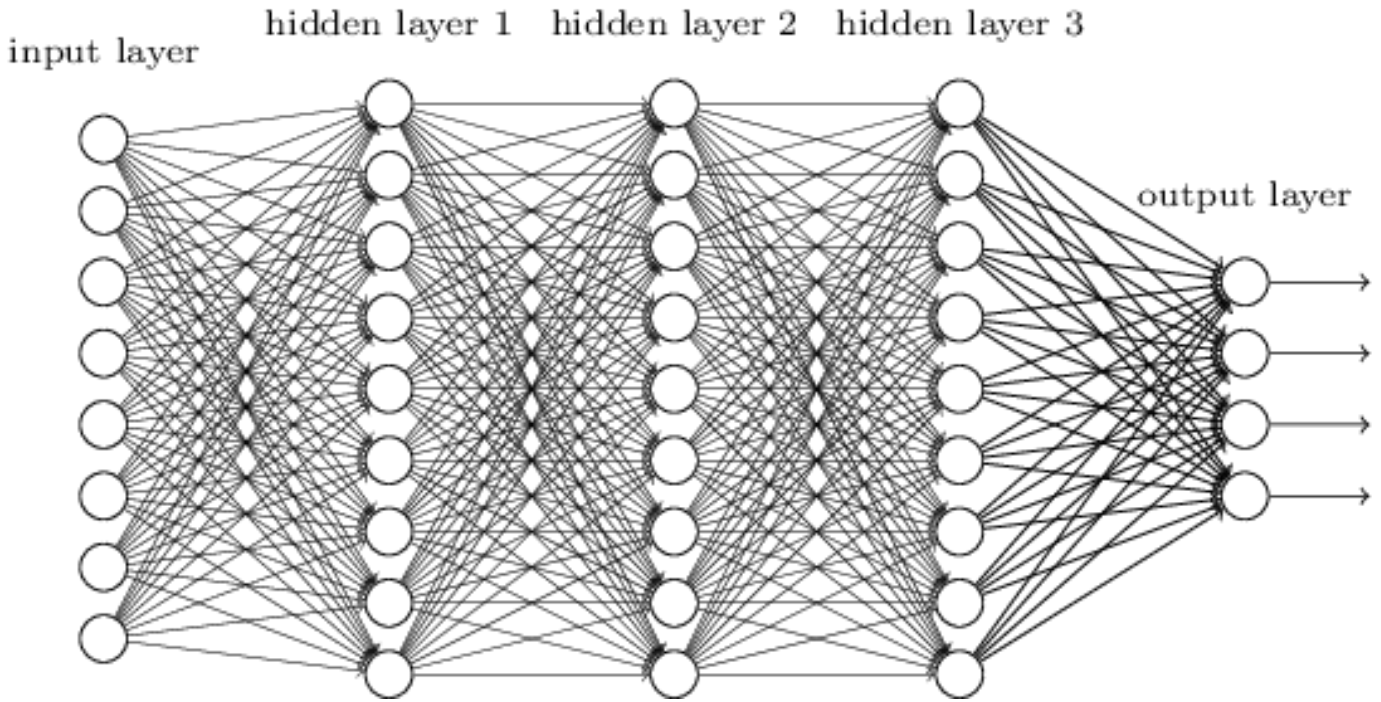
similar time complexity to **pocket** per iteration

單元三、深度學習模型簡介與應用

深度捲積神經網路
Convolutional Neural Network (CNN)

Motivation of Convolutional Neural Network

- We know it is good to learn a small model.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?



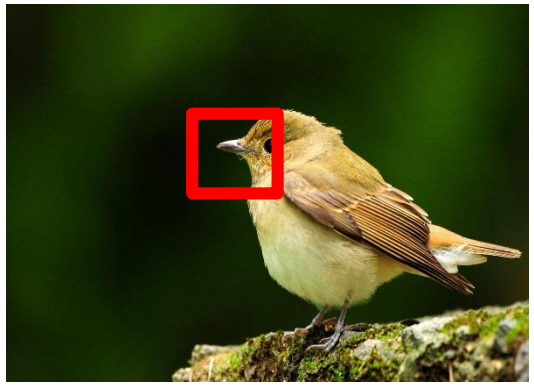
Consider Learning ‘Beak Detector’

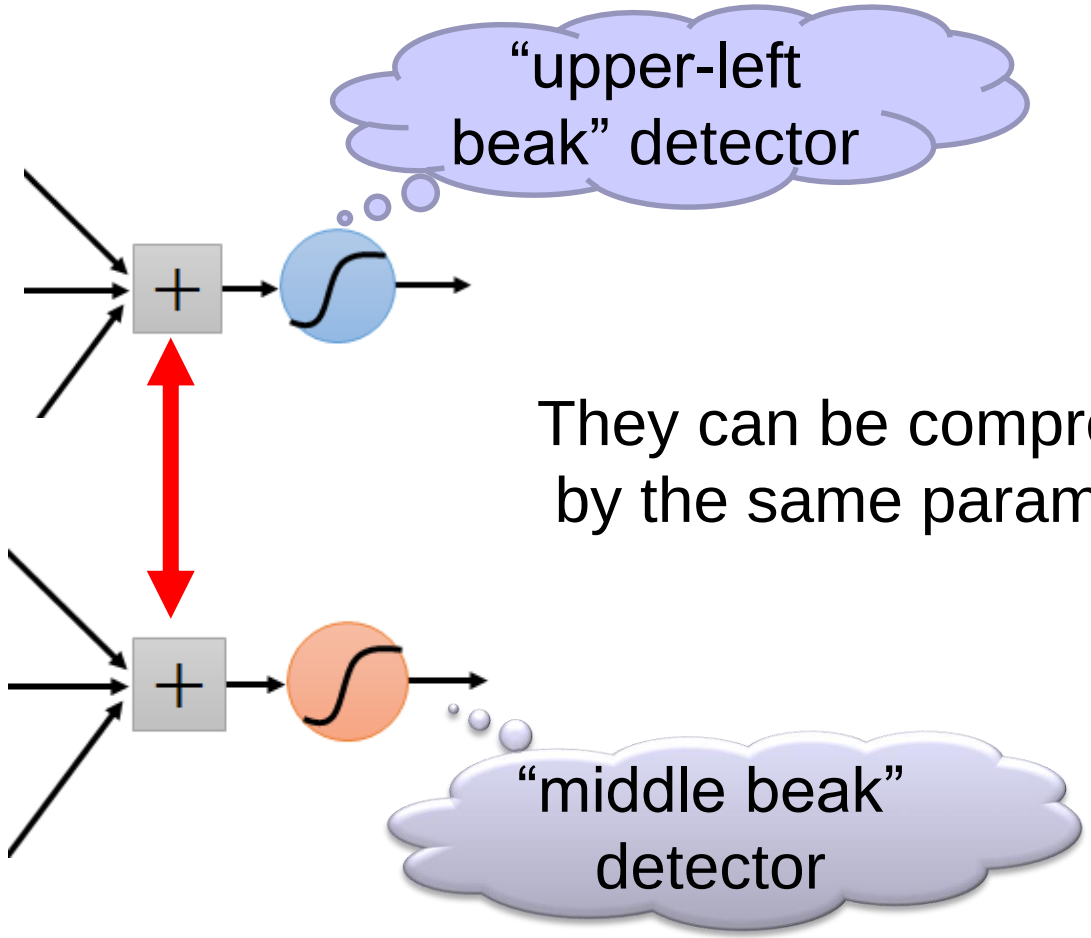
- Beak patterns are much smaller than the whole image.

Can represent a small region with fewer parameters

Same pattern appears in different places…

- They can be compressed!
- What about training a lot of such "small" detectors and each detector must "move around".

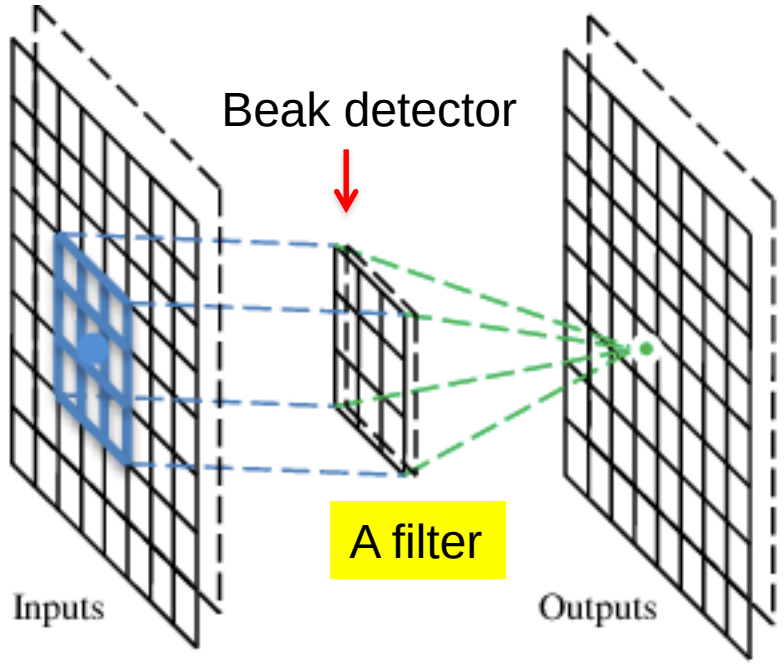




They can be compressed by the same parameters.

Convolutional Layer

- A CNN is a neural network with some convolutional layers (and some other layers).
- A convolutional layer has a number of filters that does convolutional operation.



Convolution Operation

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

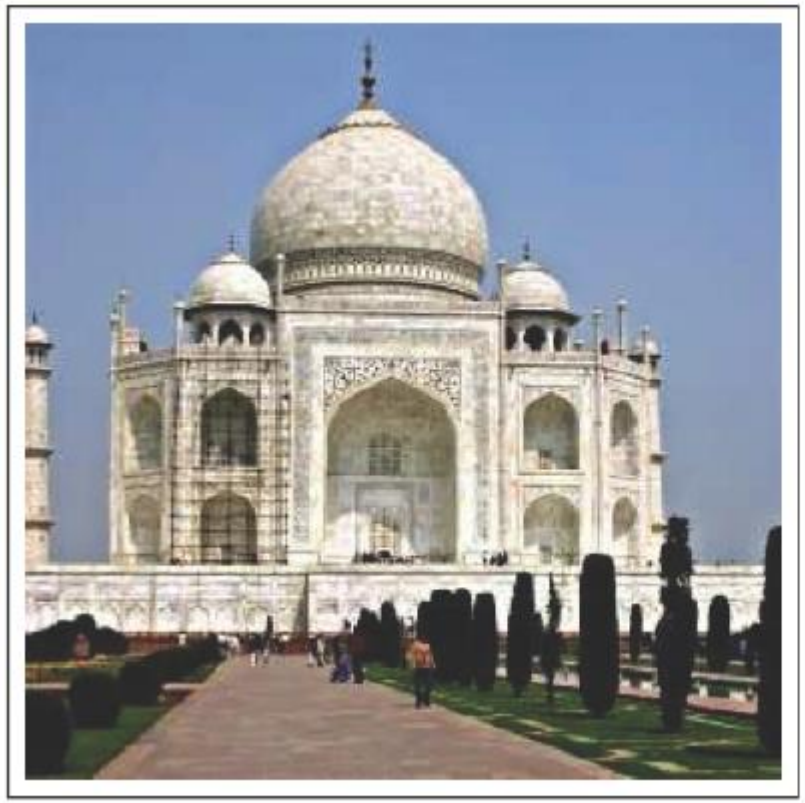
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

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

Example of a Kernel



Edge enhance Filter

0	0	0
-1	1	0
0	0	0

How to Apply 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

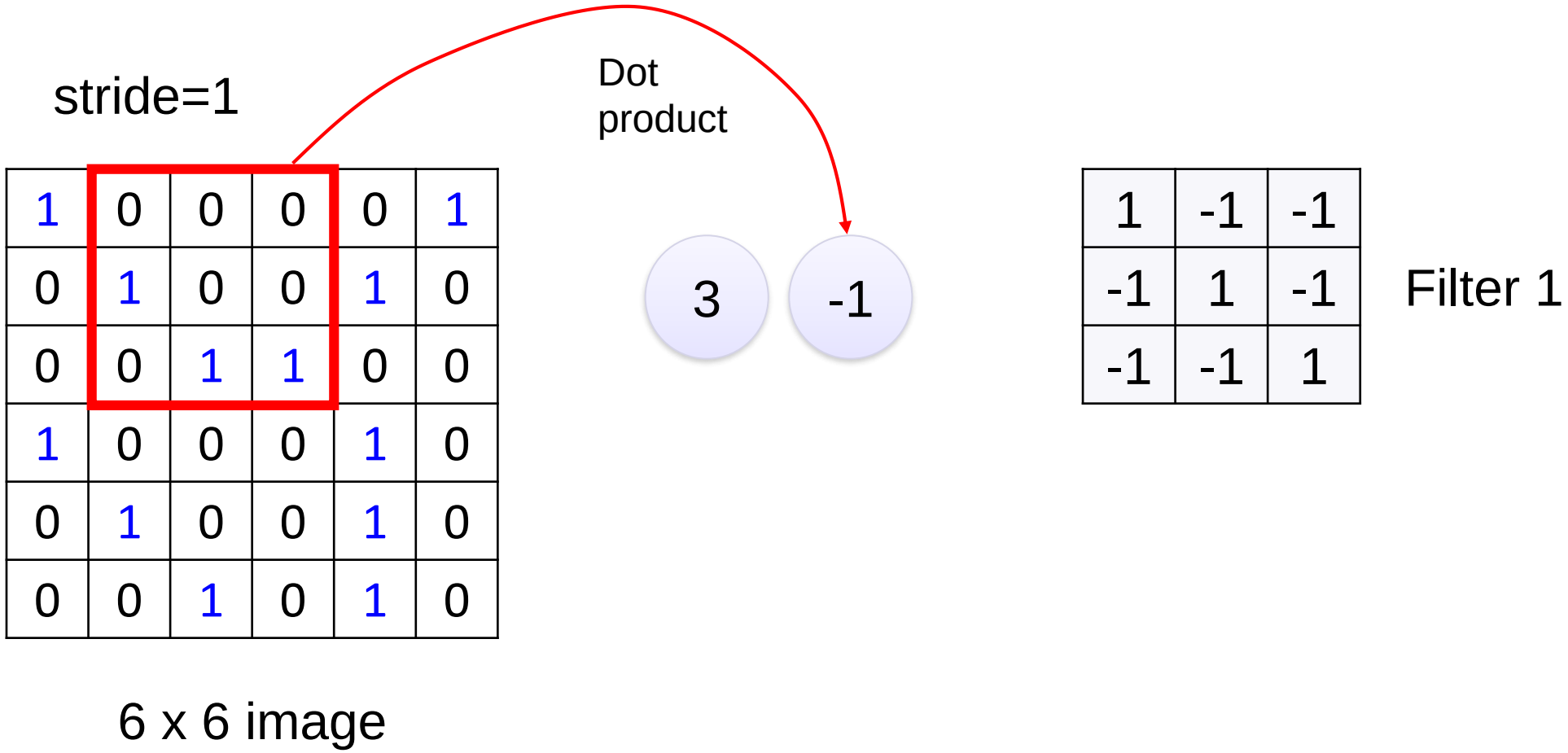
6 x 6 image

Dot product → 3

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

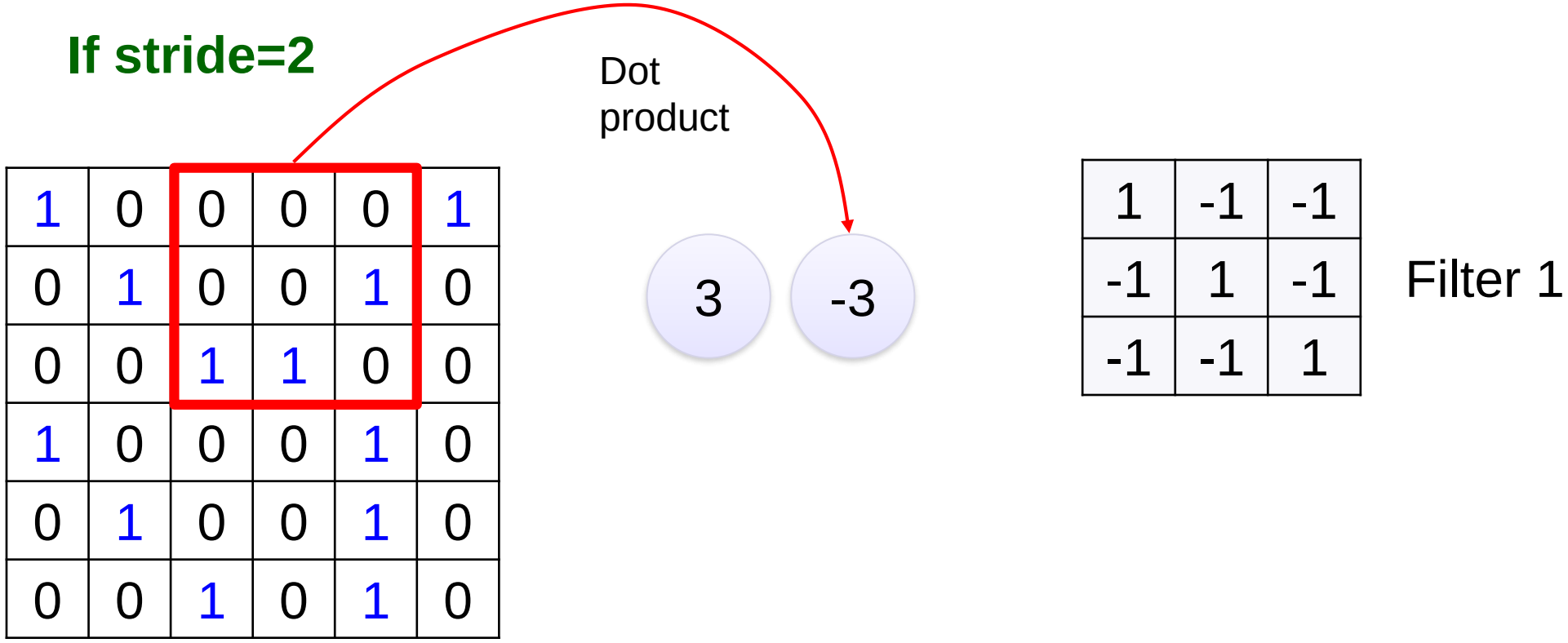
Filter 1

How to Apply Convolution ?



How to Apply Convolution ?

If stride=2



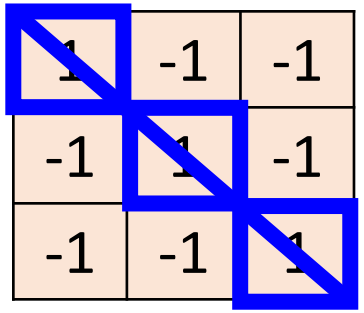
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

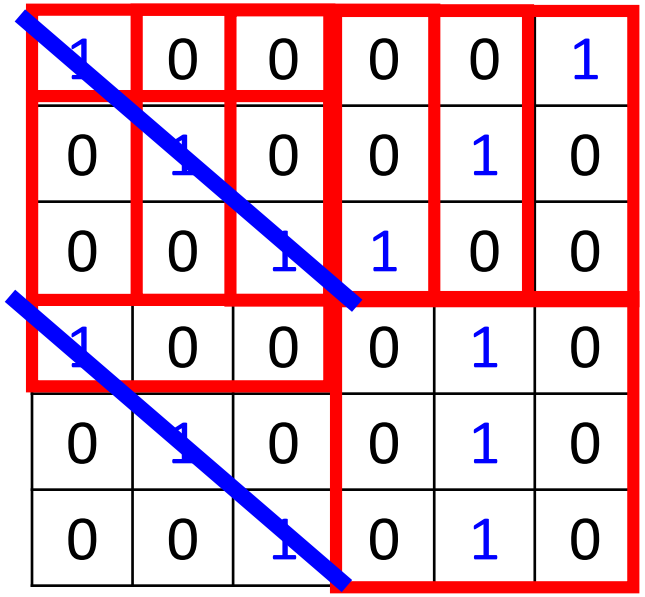
Filter 1

Convolution

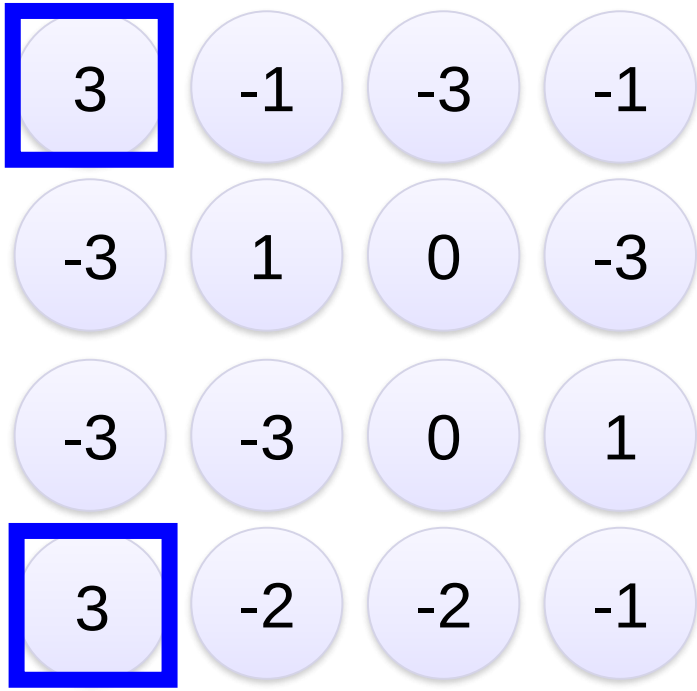


Filter 1

stride=1



6 x 6 image



Convolution

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

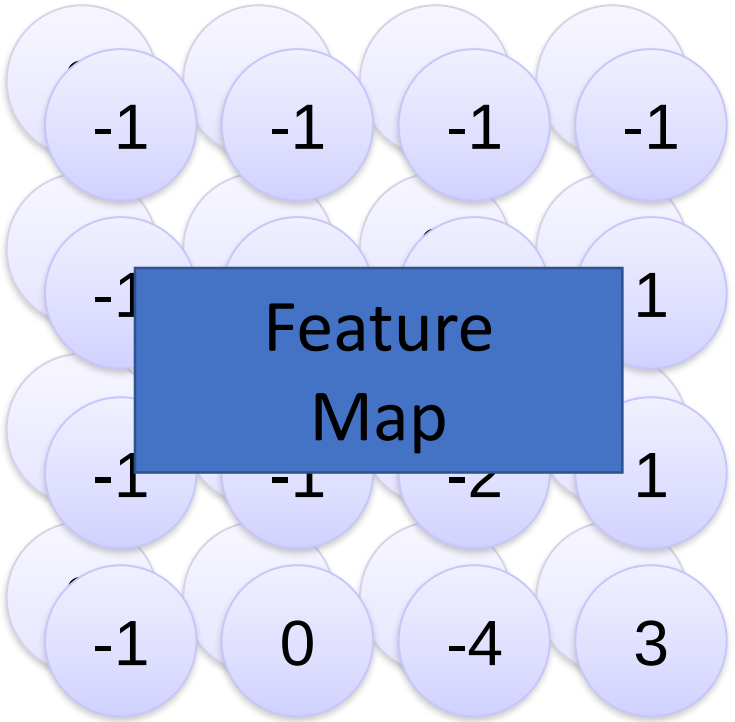
Filter 2

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

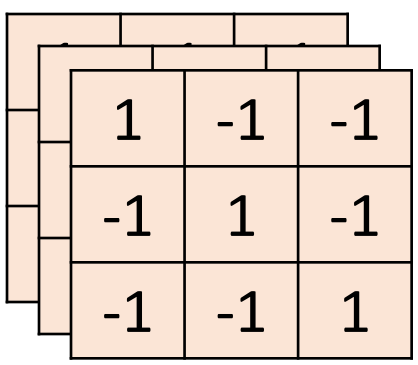
6 x 6 image

Repeat this for each filter

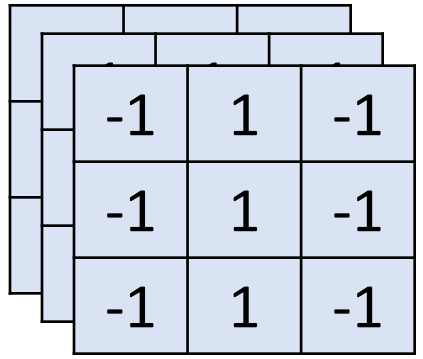


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

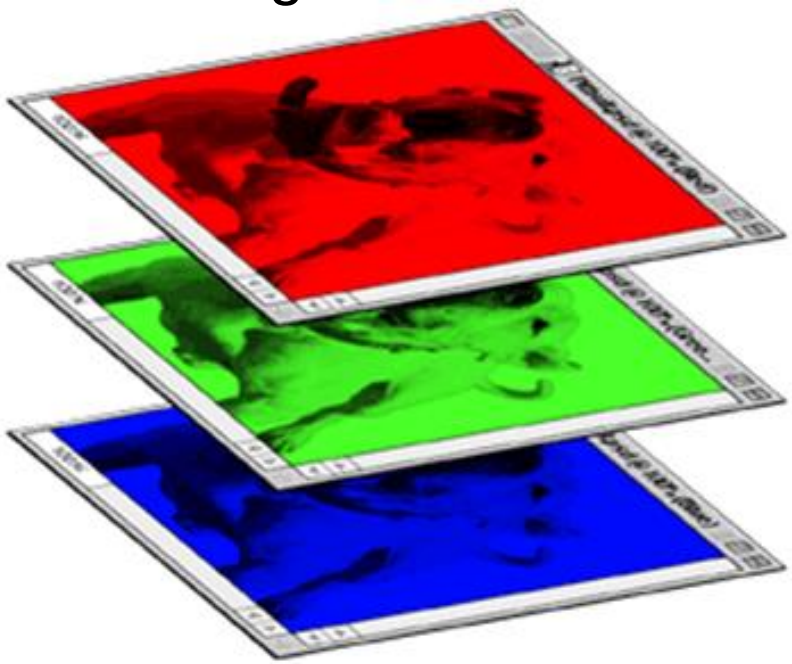


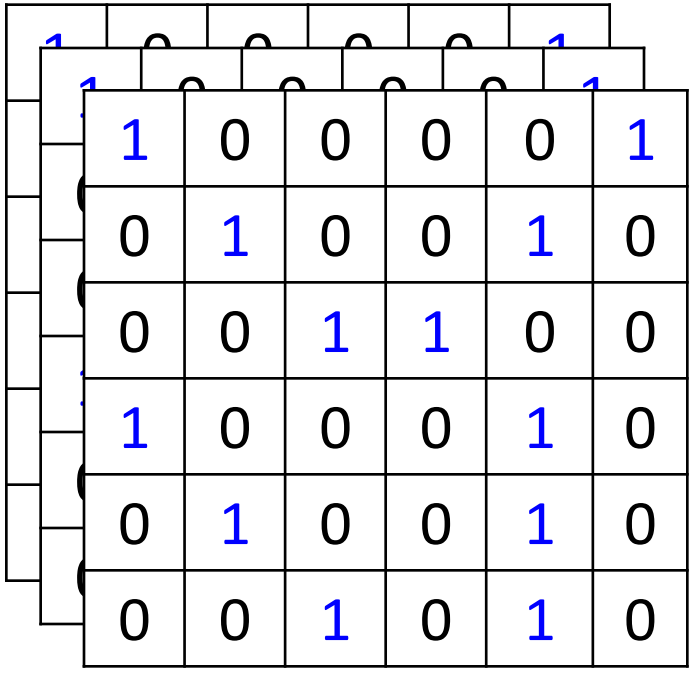
Filter 1



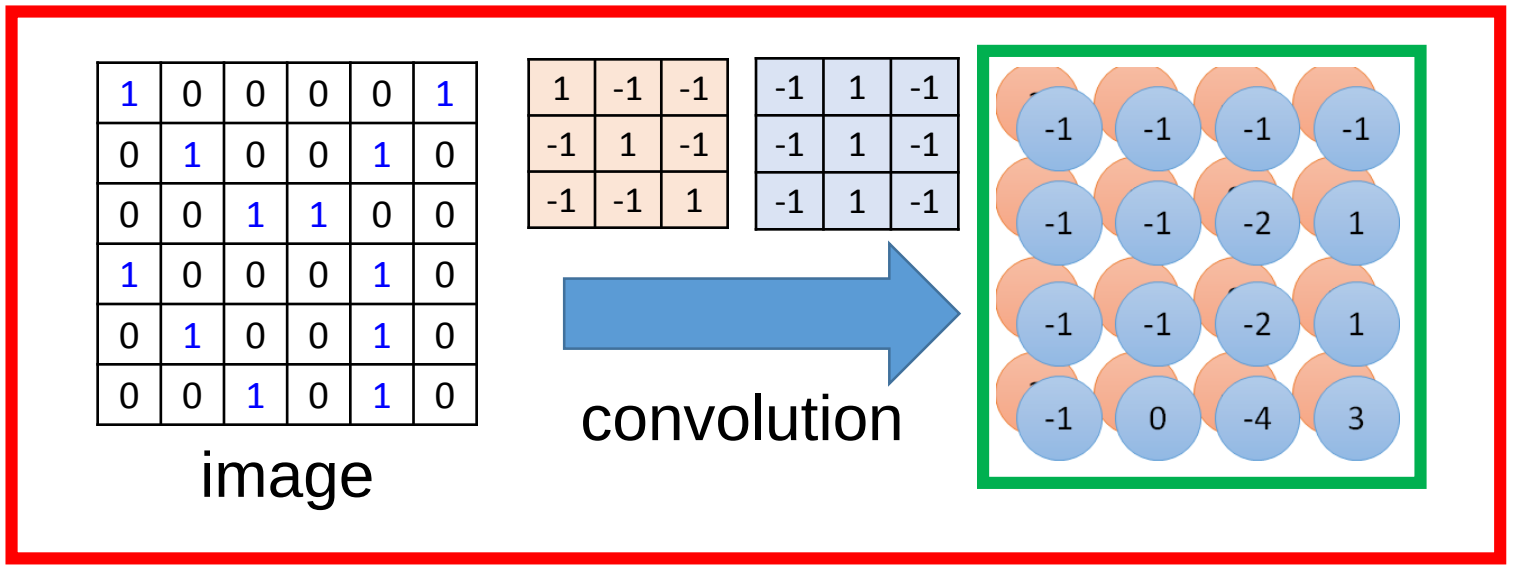
Filter 2

Color image

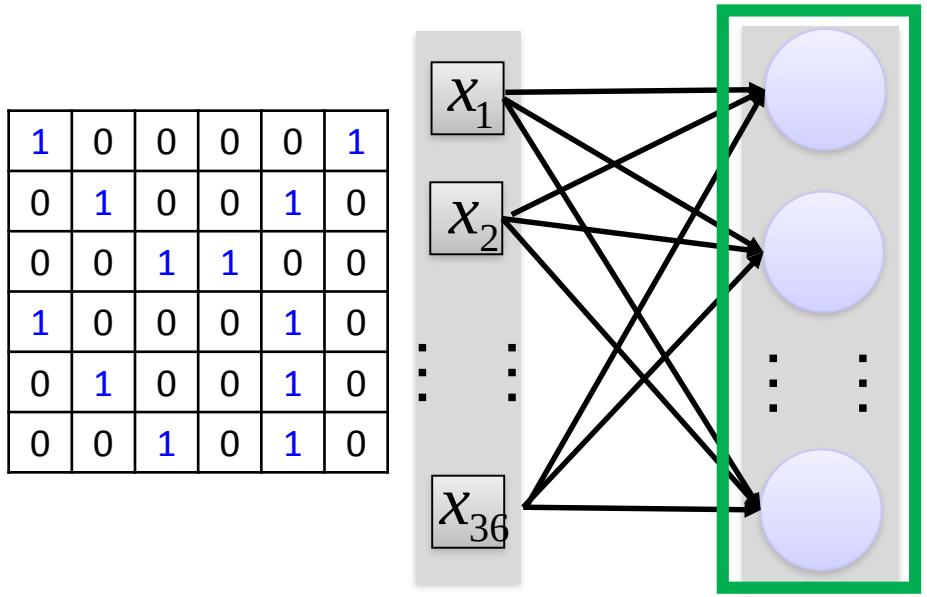


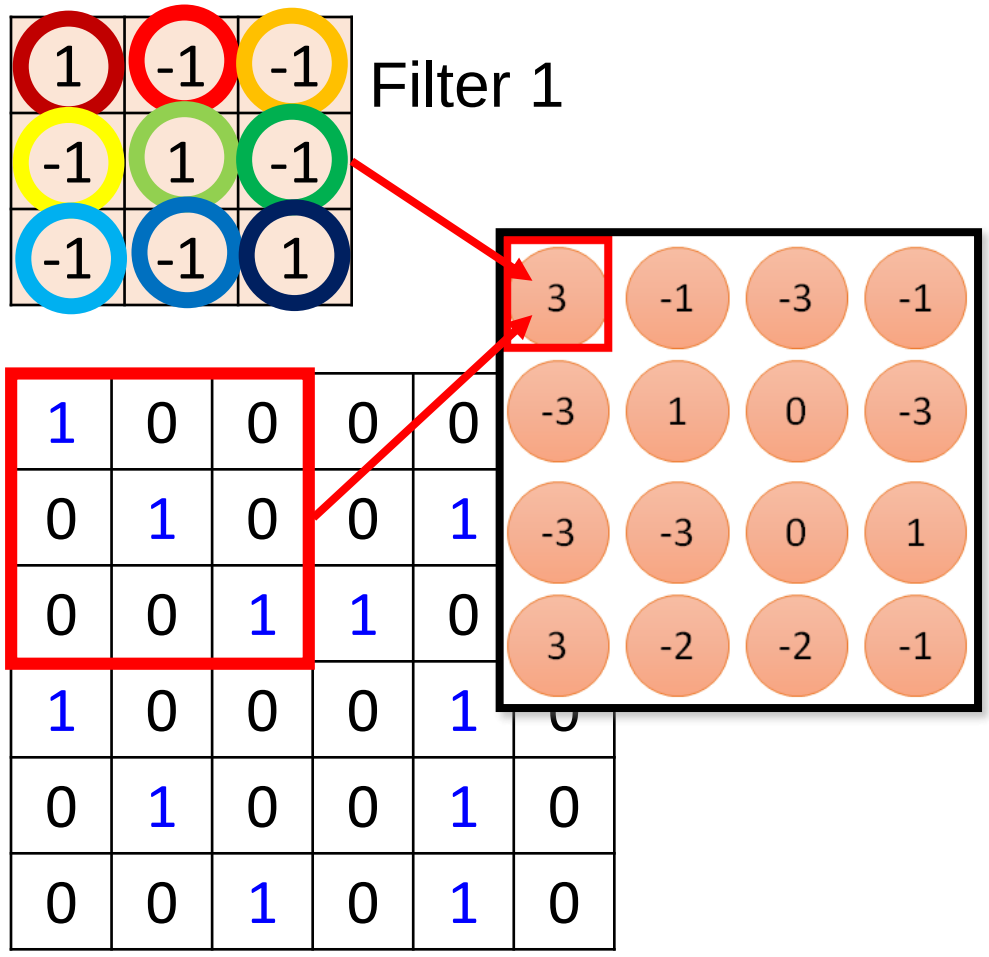


Convolution v.s. Fully Connected

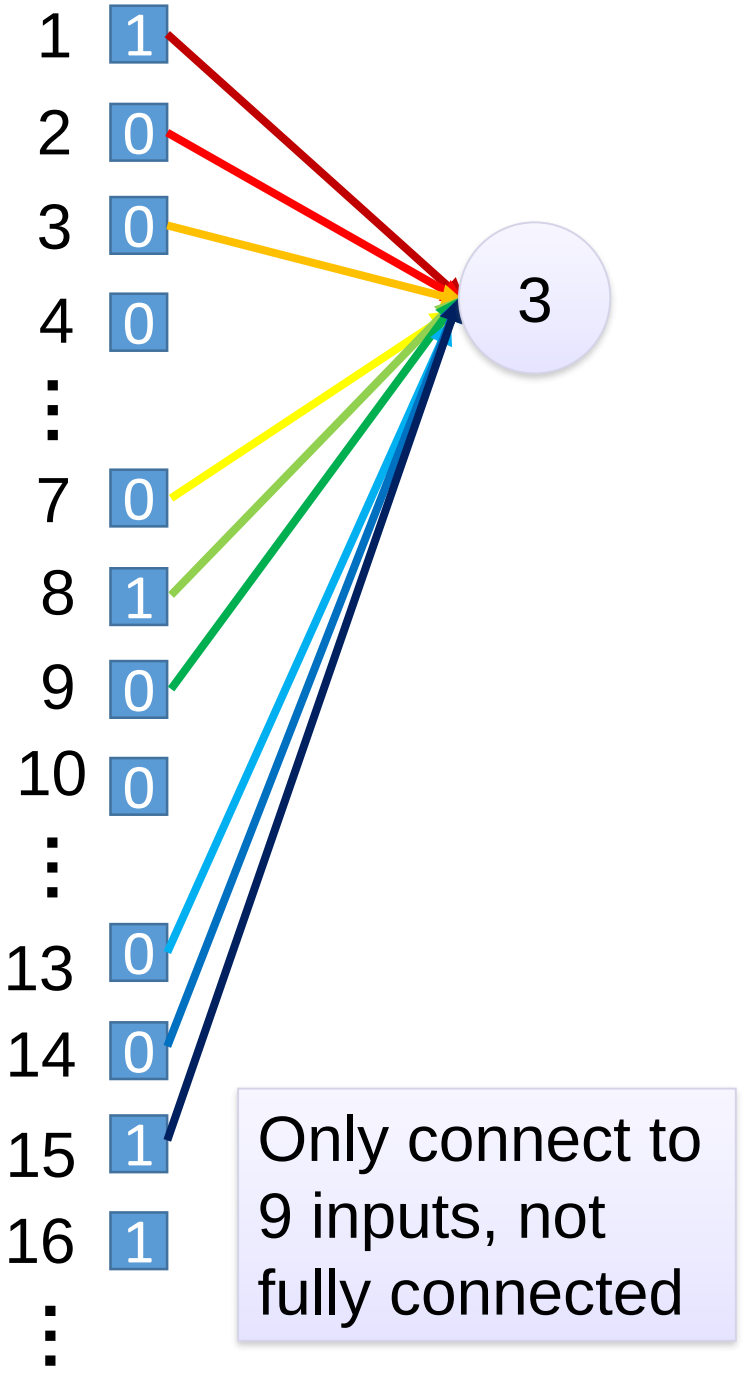


Fully-connected

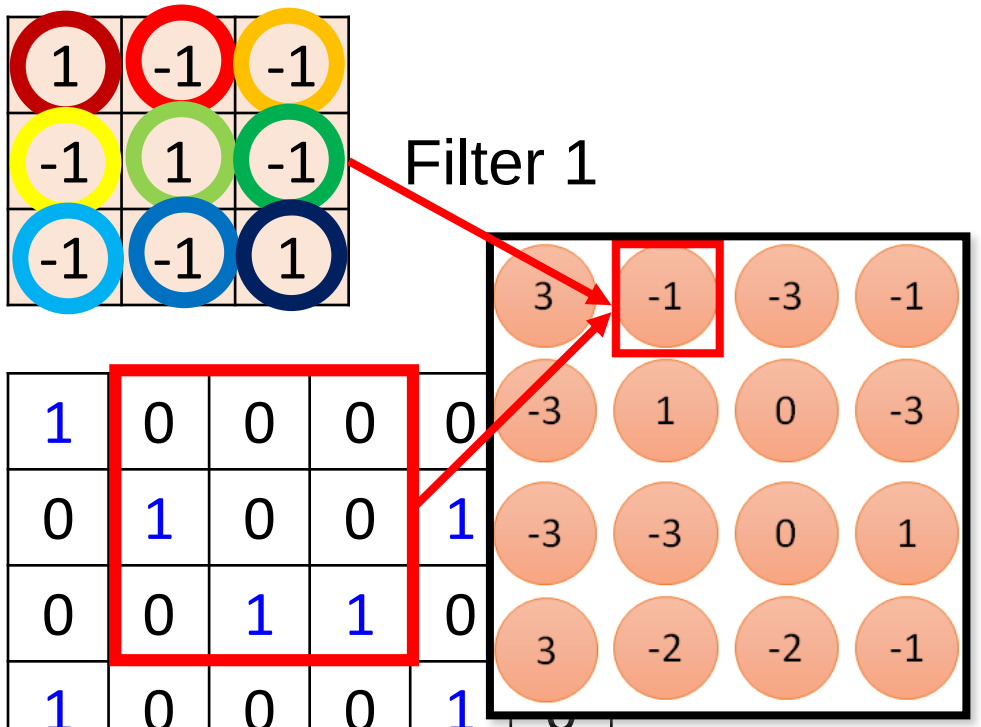




Fewer parameters !

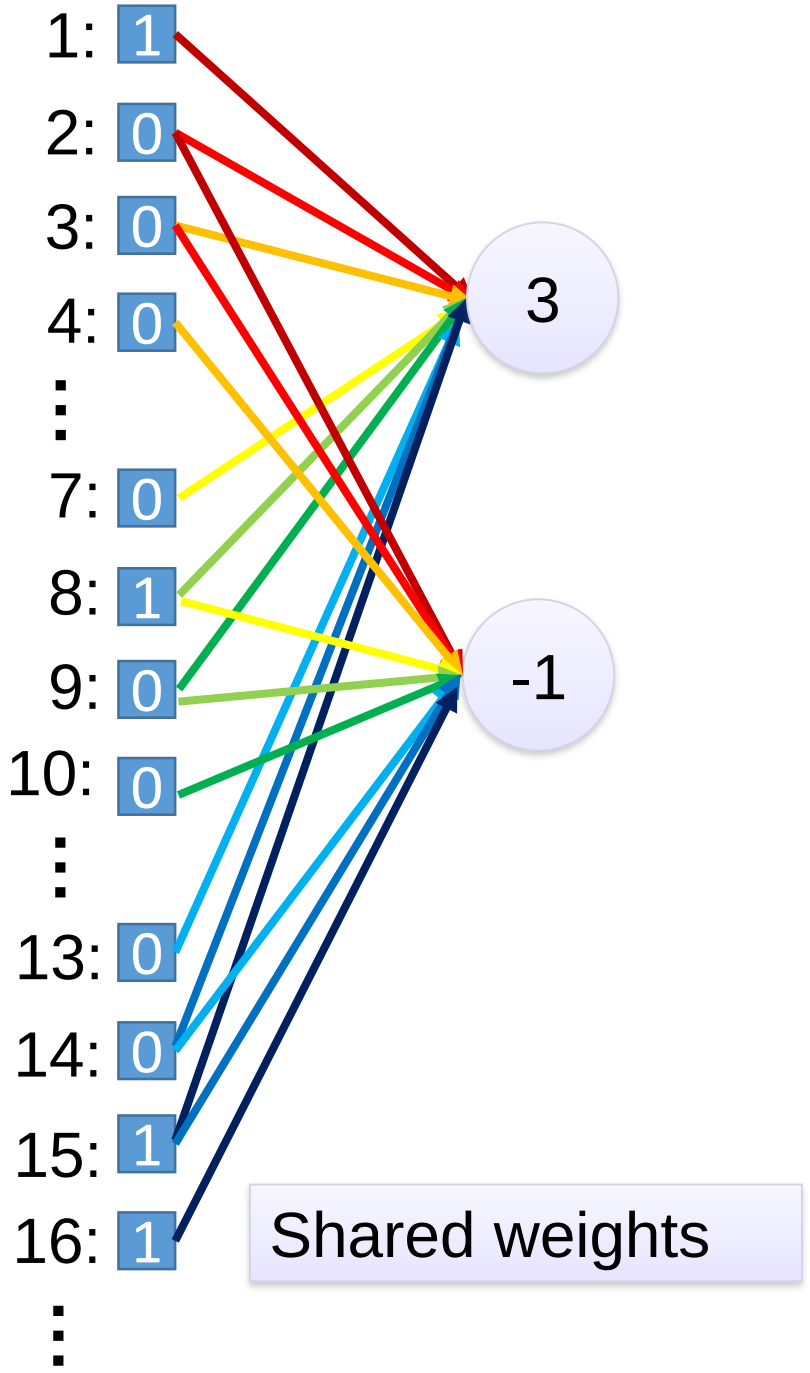


Only connect to 9 inputs, not fully connected



Fewer parameters

Even fewer parameters



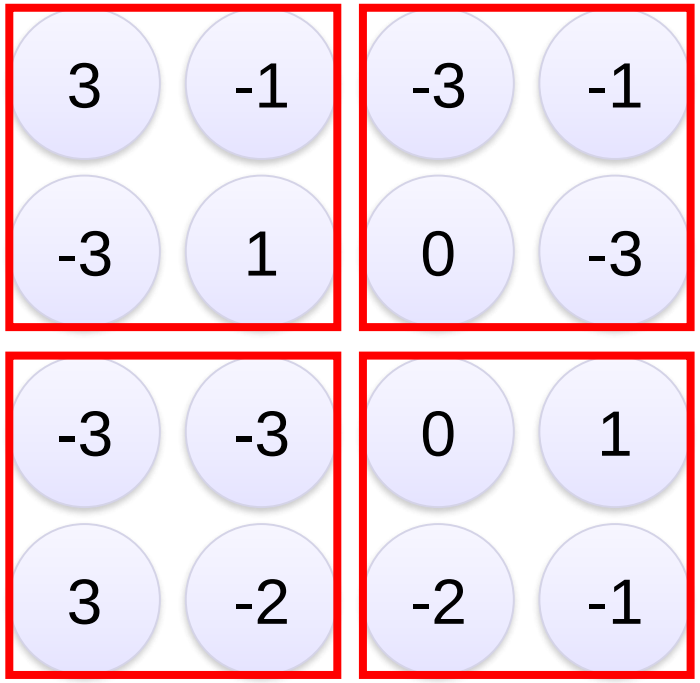
Max Pooling

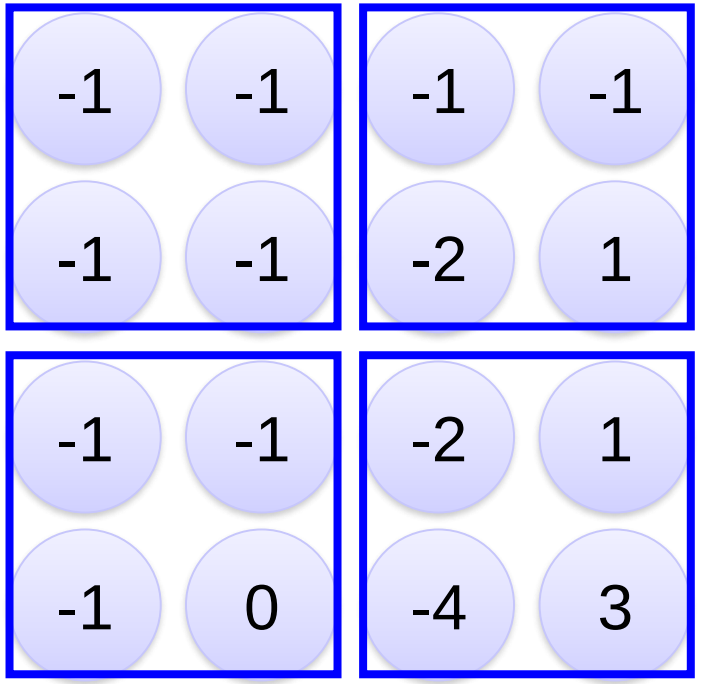
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

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

Filter 2





Why Pooling

- Subsampling pixels will not change the object.

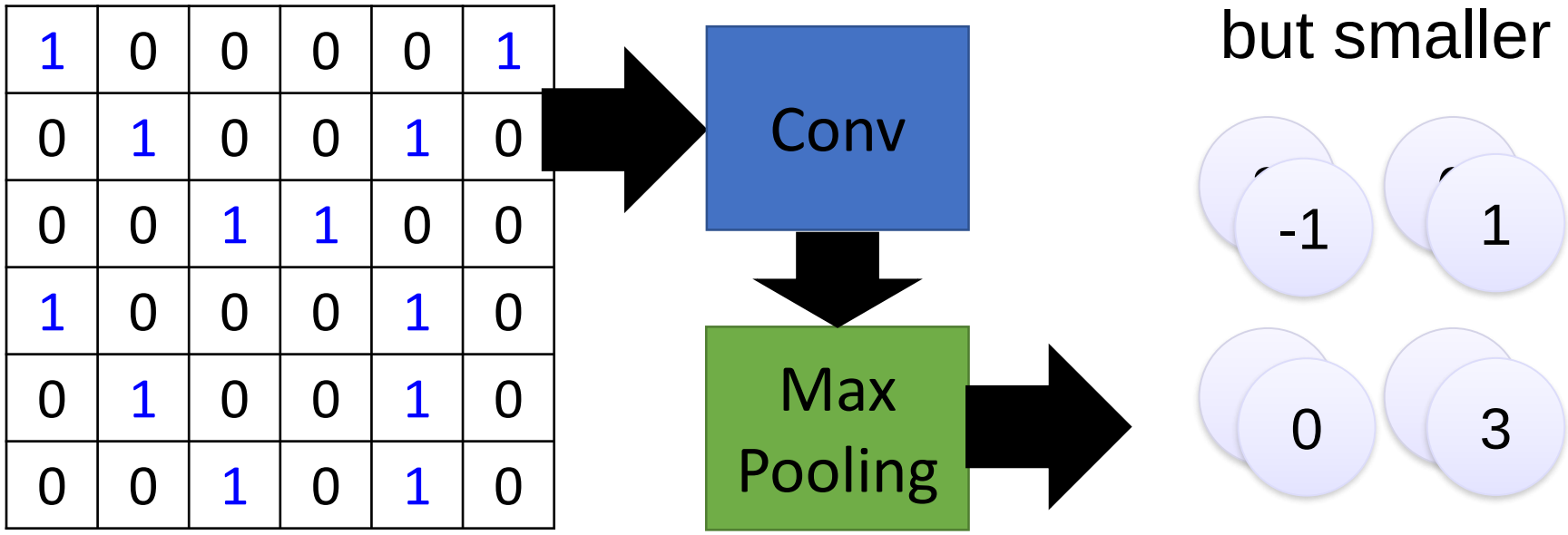
bird

Subsampling

bird

- We can subsample the pixels to make image smaller.
- This means fewer parameters to characterize the image.

Max Pooling



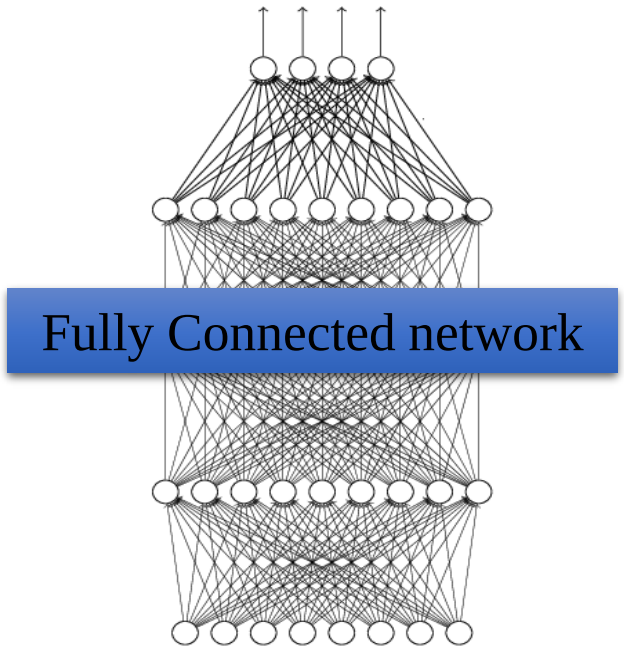
6 x 6 image

Each filter is a channel

The whole CNN

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity.

cat dog ……



Fully Connected network

Convolution

Max Pooling

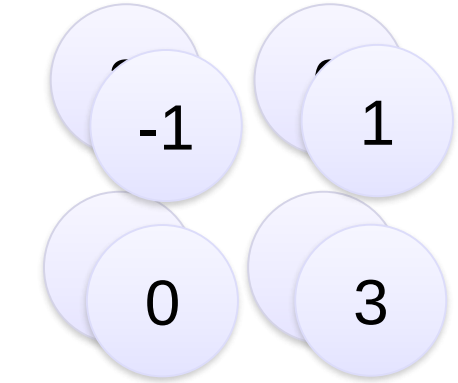
Convolution

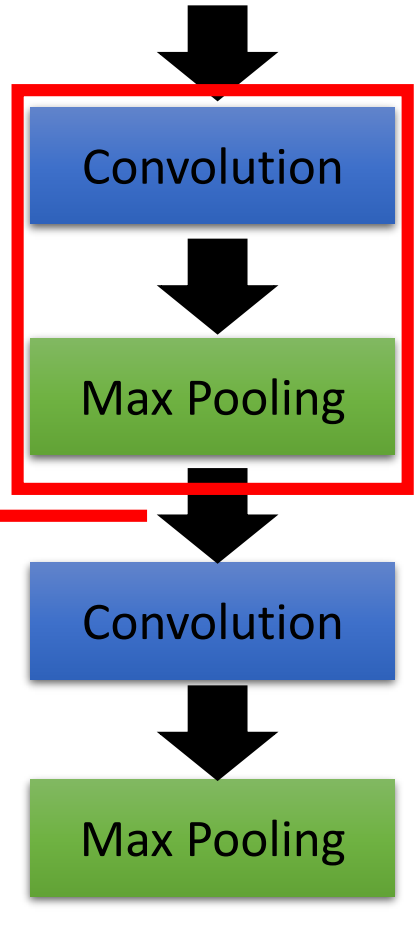
Max Pooling

Can repeat many times…

Flattened

The whole CNN

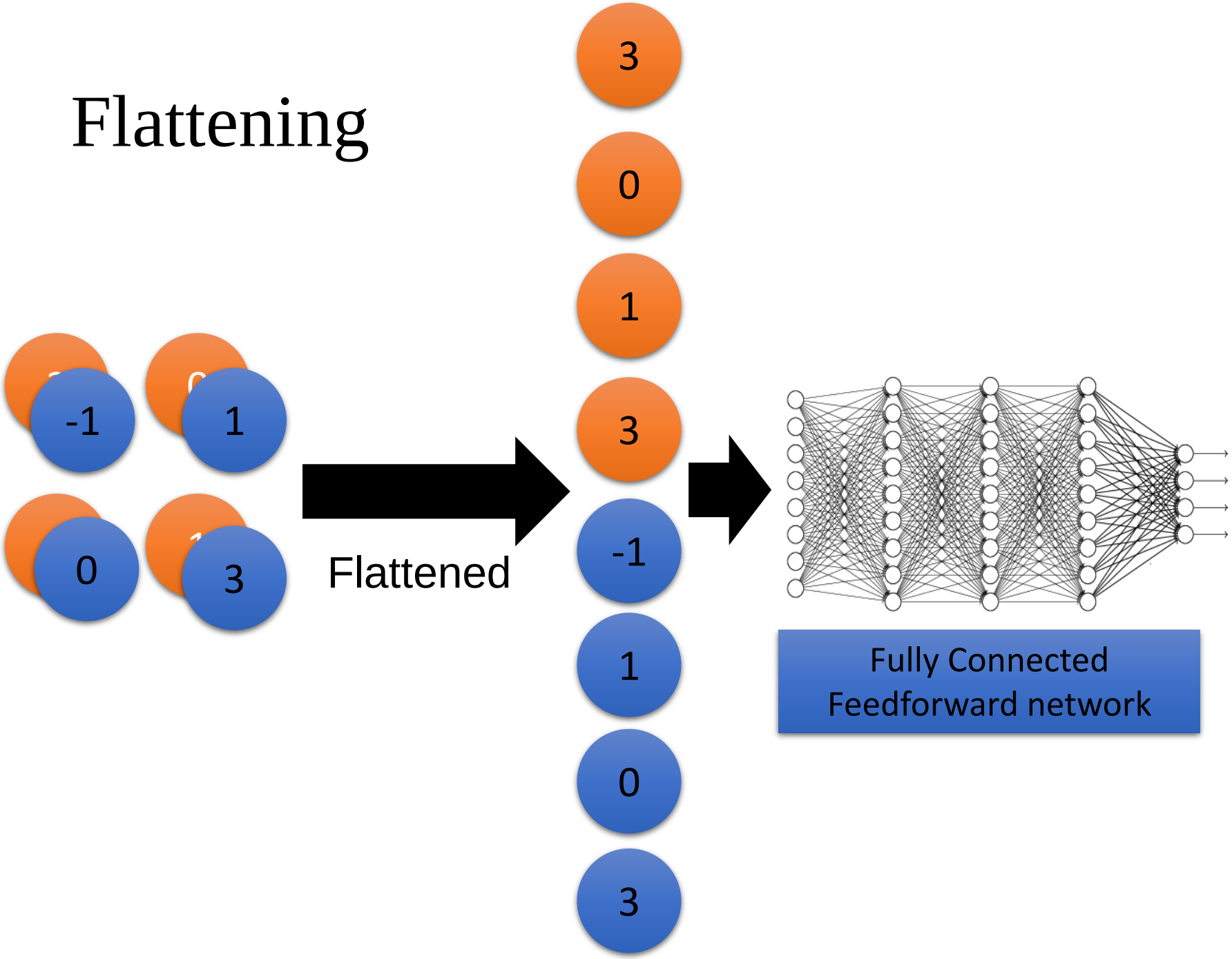




Can repeat many times

- Smaller than the original image
- The number of channels is the number of filters

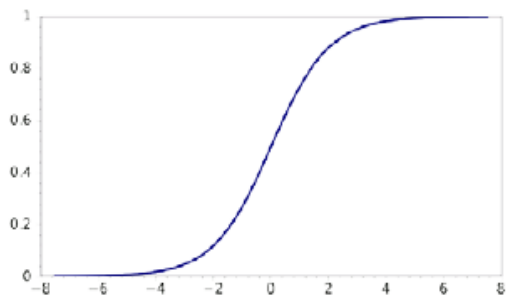
Flattening


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

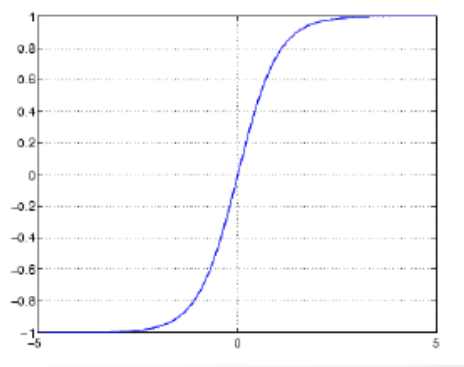
Non-Linear Activation Function

- Sigmoid: $S(t) = \frac{1}{1 + e^{-t}}.$
- Tanh: $\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Rectified Linear Unit (ReLU):

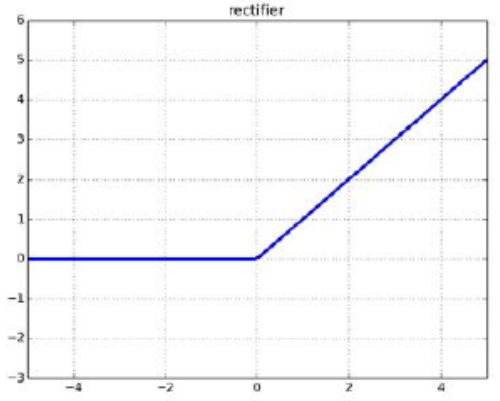
$$f(x) = \max(0, x)$$



Sigmoid



Tanh



ReLU

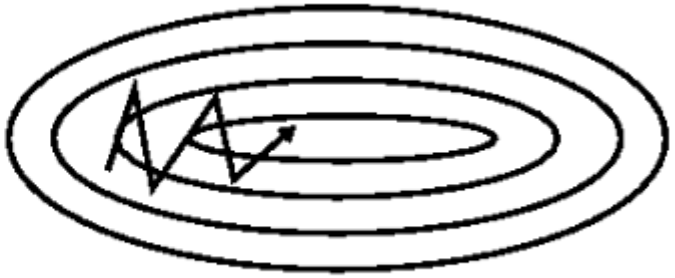
ReLU is the most popular activation function for DNN in 2015, avoiding saturation issues.

Backpropagation in Deep Learning

- Three Important Parameters for Weight Update $\omega_i \leftarrow \omega_i - \eta\frac{\partial E}{\partial w_i} + \alpha\omega_i - \lambda\eta\omega_i$

  η- learning rate,
  α - momentum,
  λ - weight decay

SGD without momentum

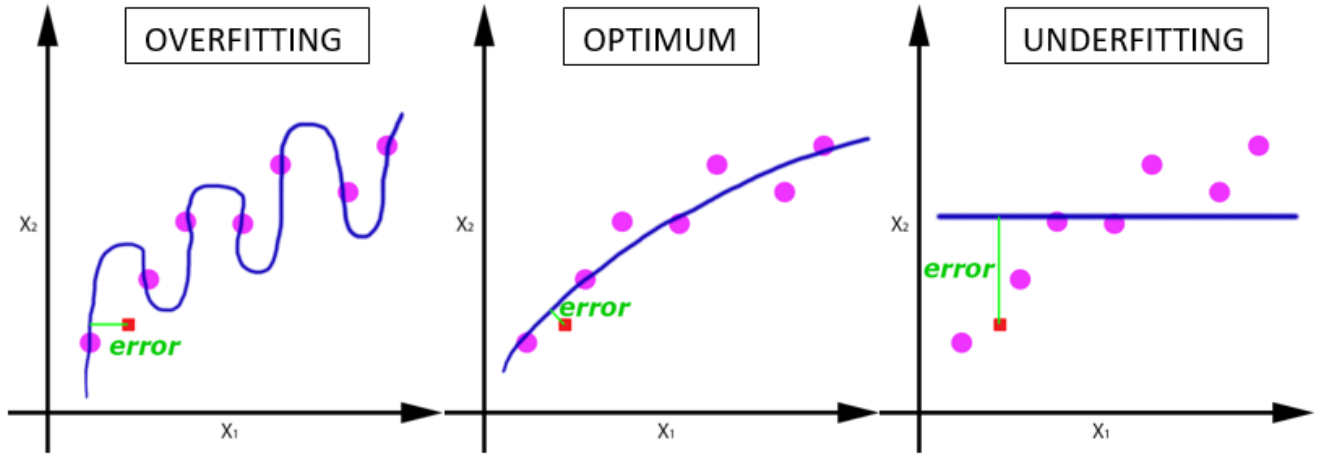


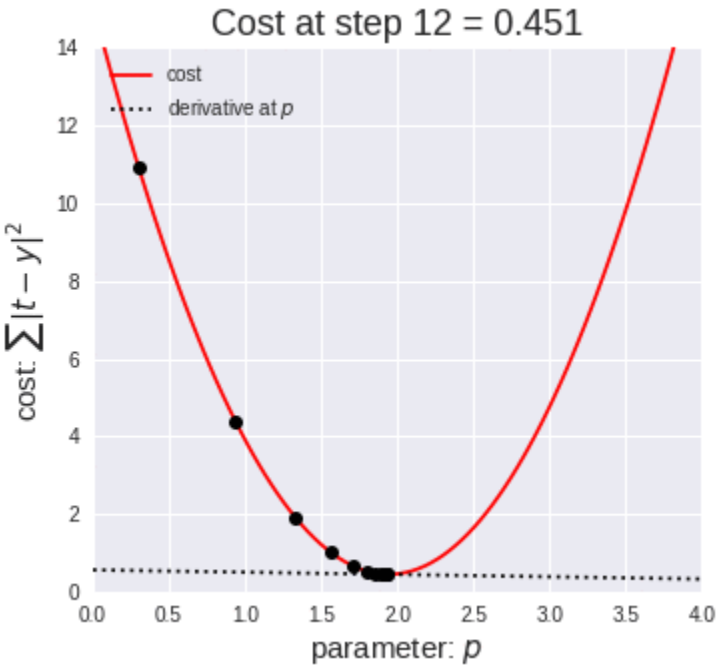
SGD with momentum

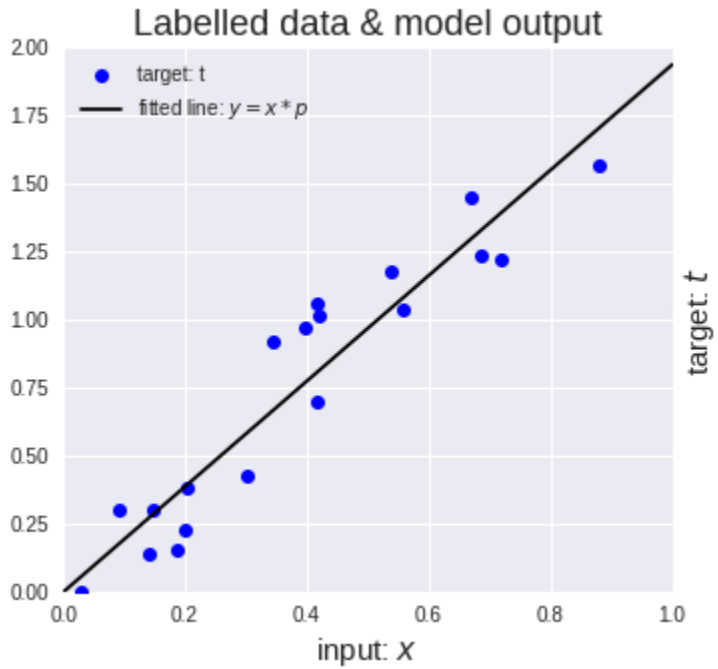
- Momentum (動量)
  - 一般Gradient Descent是朝負梯度方向下降。當考慮Momentum，如果上一次的Momentum(即w_i)與這次負梯度方向相同，這次下降的幅度就會更大。
  - 這可以在梯度平坦區，降低來回震動次數，加速收斂

Epoch vs Batch Size vs Iterations

- Recall: **Gradient Descent** is an iterative optimization algorithm to find the best results.







- The algorithm is iterative- we need to get the results multiple times to get the most optimal result.
- The iterative quality helps a under-fitted graph to make the graph fit optimally to the data.

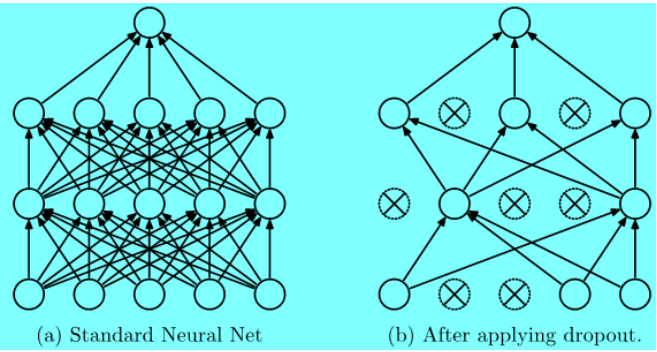
Epoch vs Batch Size vs Iterations

- **Epochs**
  - One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.
  - When using a limited dataset optimize the learning by Gradient Descent, updating the weights with single pass or one epoch is not enough.
- **Batch Size**
  - Since you can't pass the entire dataset into the neural net at once to calculate the sum of gradient, you divide dataset into Number of Batches (mini-batches).
- **Iterations**
  - The number of batches needed to complete one epoch.

- Example: divide the dataset of 2000 examples into batches of 500. Then, it will take 4 iterations to complete 1 epoch.

Measures to Reduce Overfitting

- These measures help in keeping the number of free parameters in the network low reducing complexity and thus over-fitting
- Dropout Learning
  - Randomly set the output value of network neurons to 0 with a dropout ratio of 0.5 (50% chance)
- Weight Decay
  - Used to keep the magnitude of weights close to zero
- Data Augmentation
  - Took random crop of 227x227 from image of 256x256 and randomly mirrored it in each forward-backward training pass



Error Function – Classification Error v.s Cross Entropy

- Classification Error = Count of error items / Count of all items
- Example: computed 預測結果，targets 標準答案

模型 1

computed	targets	correct?
0.3 0.3 0.4	0 0 1 (democrat)	yes
0.3 0.4 0.3	0 1 0 (republican)	yes
0.1 0.2 0.7	1 0 0 (other)	no

$$\text{classification error} = 1/3 = 0.33$$

模型 2

computed	targets	correct?
0.1 0.2 0.7	0 0 1 (democrat)	yes
0.1 0.7 0.2	0 1 0 (republican)	yes
0.3 0.4 0.3	1 0 0 (other)	no

$$\text{classification error} = 1/3 = 0.33$$

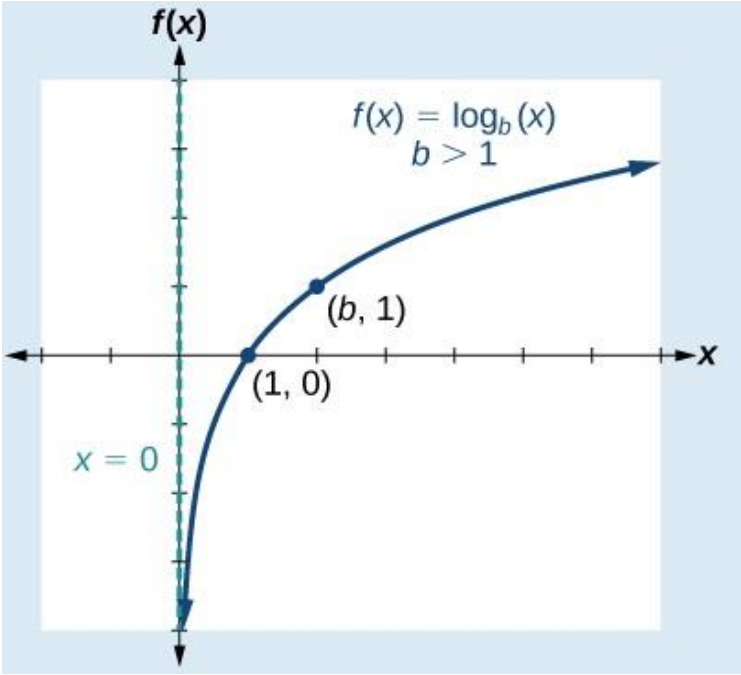
模型一前兩項都是險勝，模型二前兩是大勝；模型一最後一項是大敗，模型二最後一項是惜敗

Classification Error 無法分出兩個模型的差異!!!

Cross Entropy

- In neural network, $s_j = \sum_i w_{ij} x_i$, $a = \sigma(s)$ where σ(.) is the activation function by sigmoid function. 其中a為預測結果，y為標準答案

Cross Entropy $C = -\frac{1}{n}\sum_x [y ln a + (1-y)\ln(1-a)]$



y	*a*	*lna*	*ln*(1-*a*)	*ylna*	(1-*y*)*ln*(1-*a*)	*C*
1	1	0	負很大	0	0	0
1	0	負很大	0	負很大	0	正很大
0	1	0	負很大	0	負很大	正很大
0	0	負很大	0	0	0	0

答對的時候C為0，答錯的時候C很大 = 最小化C可以得到正確答案!

Cross Entropy

- Example:

模型 1

computed	targets	correct?
0.3 0.3 0.4	0 0 1 (democrat)	yes
0.3 0.4 0.3	0 1 0 (republican)	yes
0.1 0.2 0.7	1 0 0 (other)	no

classification error $= 1/3 = 0.33$

模型 2

computed	targets	correct?
0.1 0.2 0.7	0 0 1 (democrat)	yes
0.1 0.7 0.2	0 1 0 (republican)	yes
0.3 0.4 0.3	1 0 0 (other)	no

classification error $= 1/3 = 0.33$

第一个模型中第一项的 cross-entropy 是：

-((ln(0.3)*0) + (ln(0.3)*0) + (ln(0.4)*1)) = -ln(0.4)

所以，第一个模型的 ACE (average cross-entropy error) 是

-(ln(0.4) + ln(0.4) + ln(0.1)) / 3 = 1.38

第二个模型的 ACE 是：

-(ln(0.7) + ln(0.7) + ln(0.3)) / 3 = 0.64

用Cross Entropy 可以辨别出模型二優於模型一 !!!

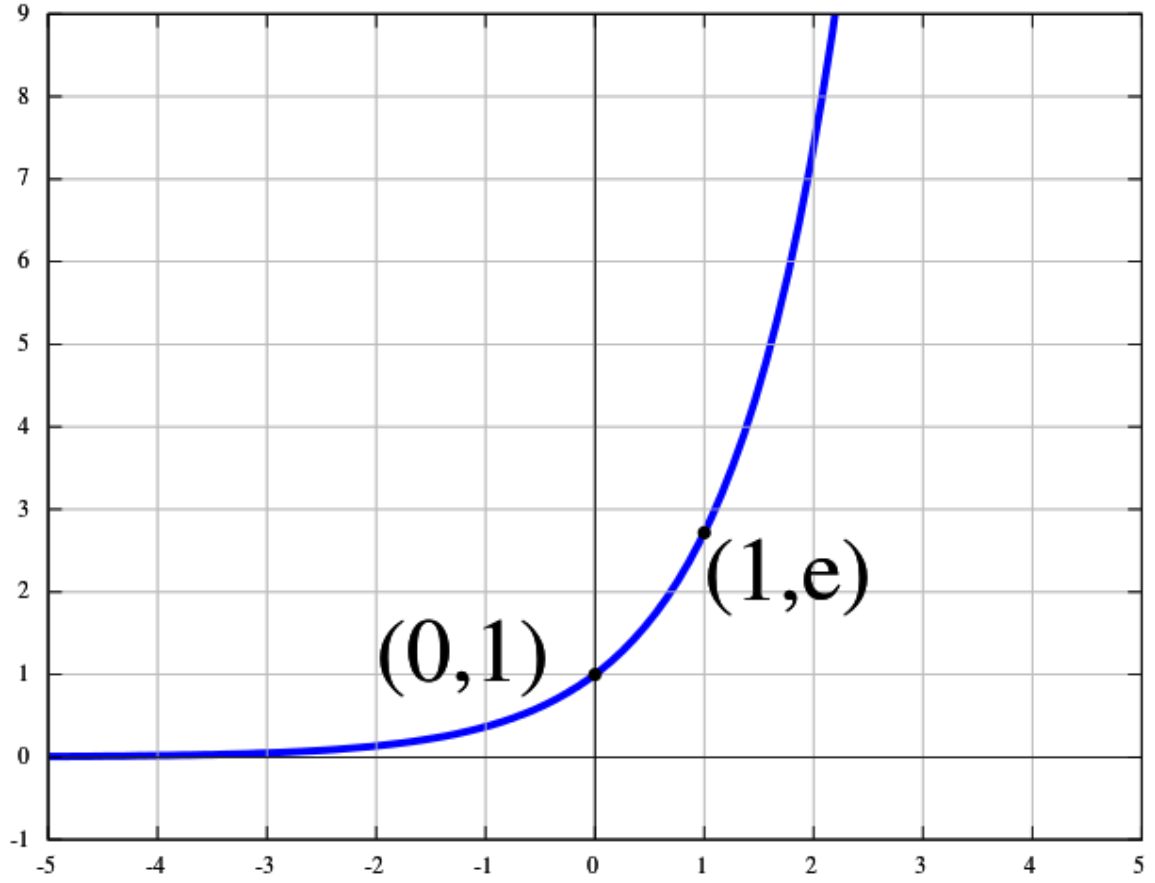
Softmax Function

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} exp(\theta_j^T x)} \begin{bmatrix} exp(\theta_1^T x) \\ exp(\theta_2^T x) \\ \vdots \\ exp(\theta_K^T x) \end{bmatrix}$$

- 特性
  - 輸出每一個類別的機率值
  - 所有類別的機率值總合為1
- Softmax_cross_entropy_with_logits 使用方式
  - 分數值不作sigmoid，直接轉成機率值
  - 可以用於binary classification problem
  - 可以用於multi-class classification problem
  - 不可用於multi-label classification problem
- 原因
  - 對於multi-class問題，每一個類別的機率和為一，帶入Entropy公式結果正確
  - 對於multi-label，類別的機率和不會是一，無法計算Entropy

Why use Exp function for Normalization?

- If x is 0 then y=1, if x is 1, then y=2.7, and if x is 2, now y=7!
- A huge step! A non-linear transformation of unnormalized log scores.
- The interesting property of the exponential function combined with the normalization
  - In the softmax, high scores in x become much more probable than low scores.



An Example of Softmax

- Say your log score *x* is vector [2,4,2,1].
- The simple argmax function outputs: [0,1,0,0]
- A simple normalization, which is differentiable, outputs the following probabilities:

  [0.2222,0.4444,0.2222,0.1111]

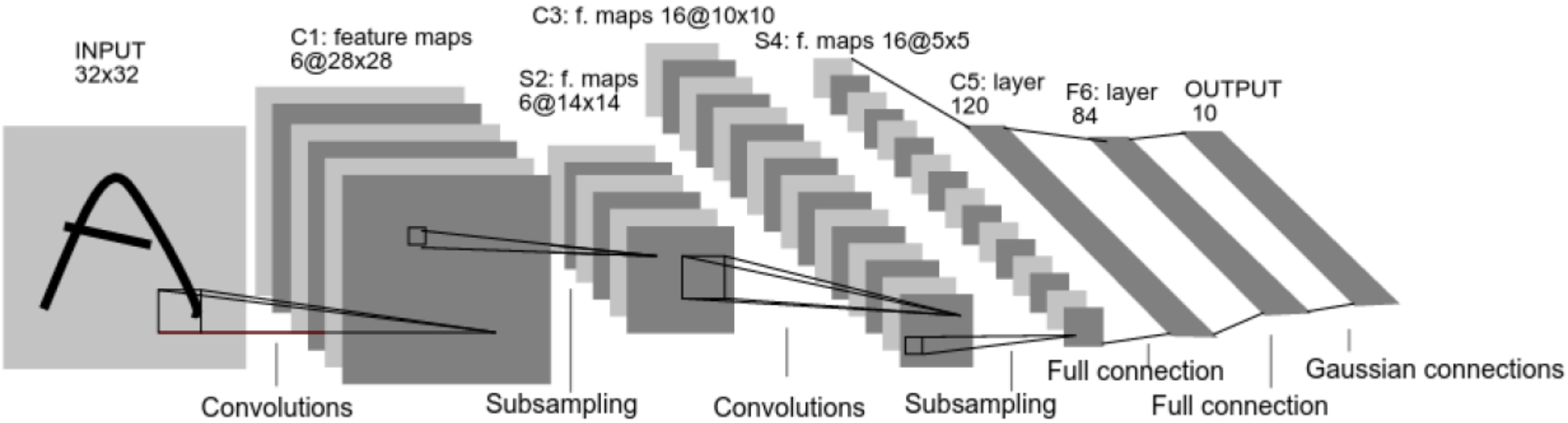
- That's really far from the argmax! Whereas the softmax outputs:

  [0.1025,0.7573,0.1025,0.0377]

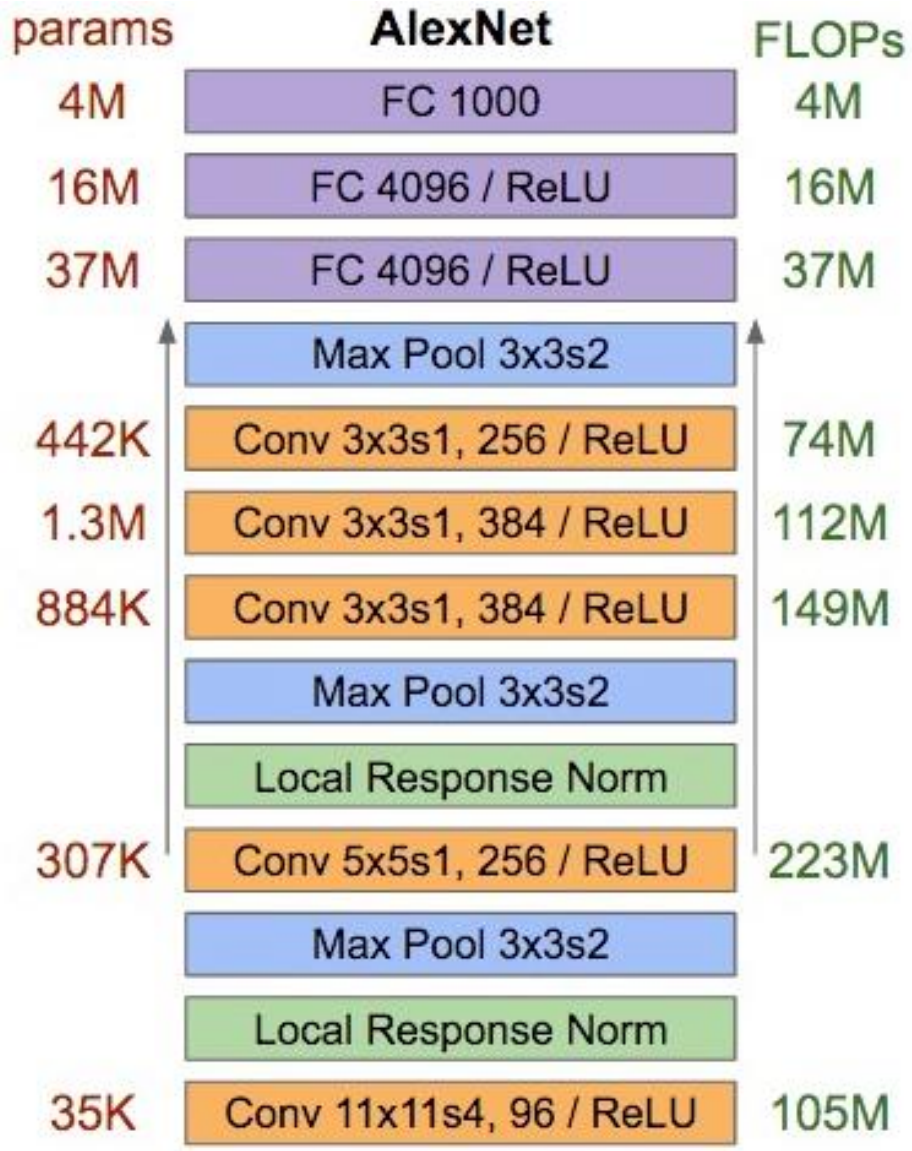
- That's much closer to the argmax! Because we use the natural exponential, we hugely increase the probability of the biggest score and decrease the probability of the lower scores when compared with standard normalization. Hence the "max" in softmax.

A simple version of the CNN (LeNet5 Architecture)

- Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

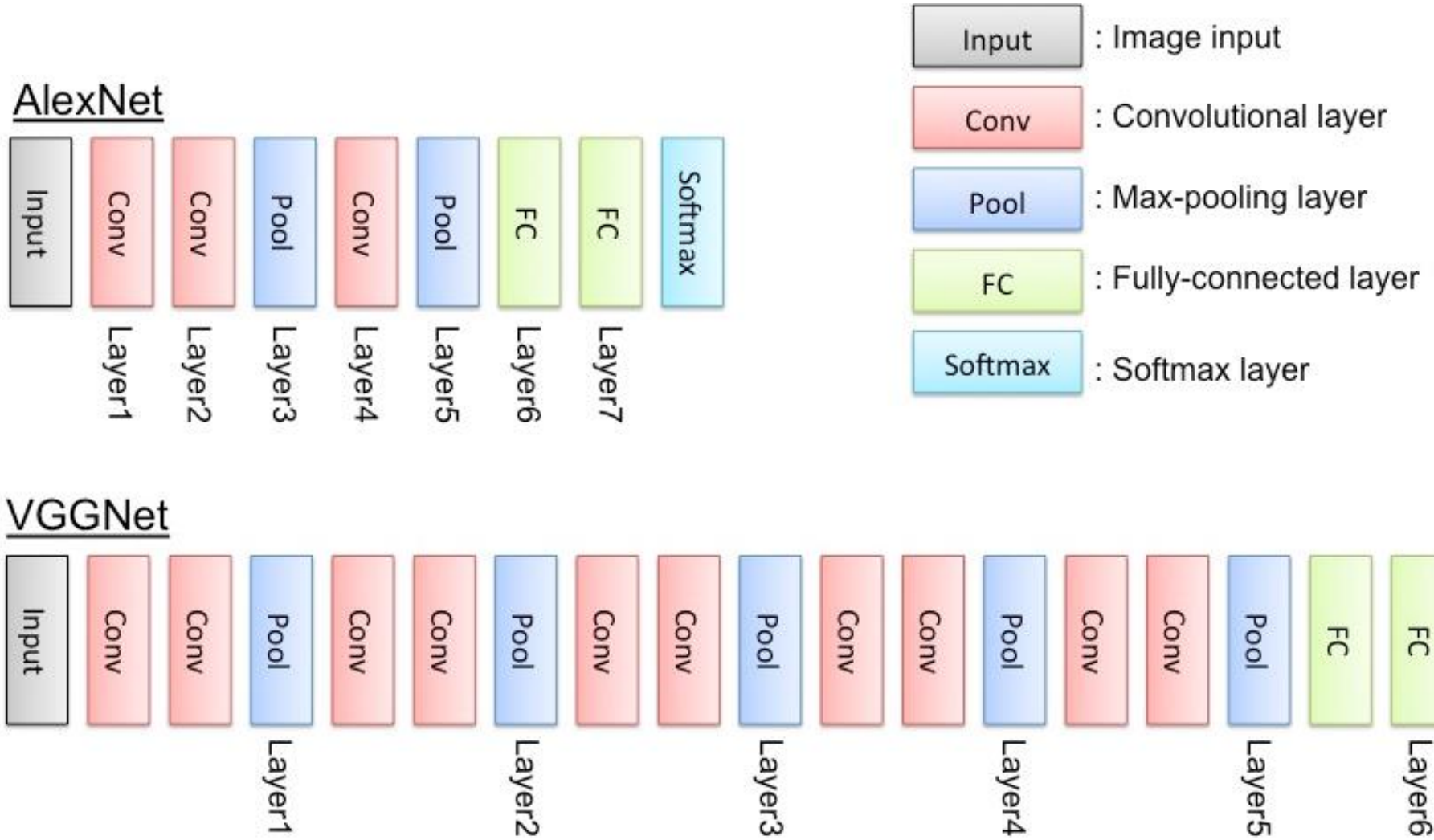


Structure of AlexNet

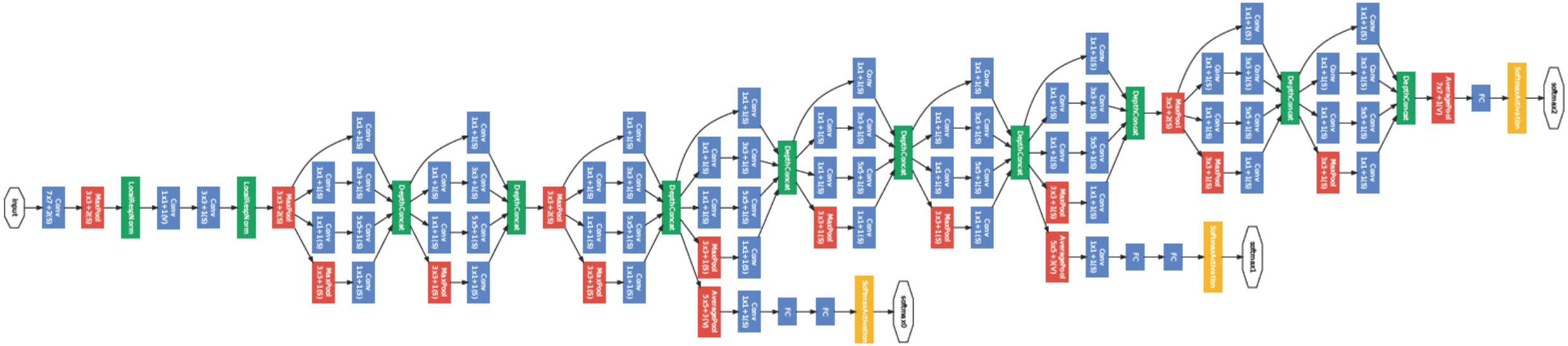


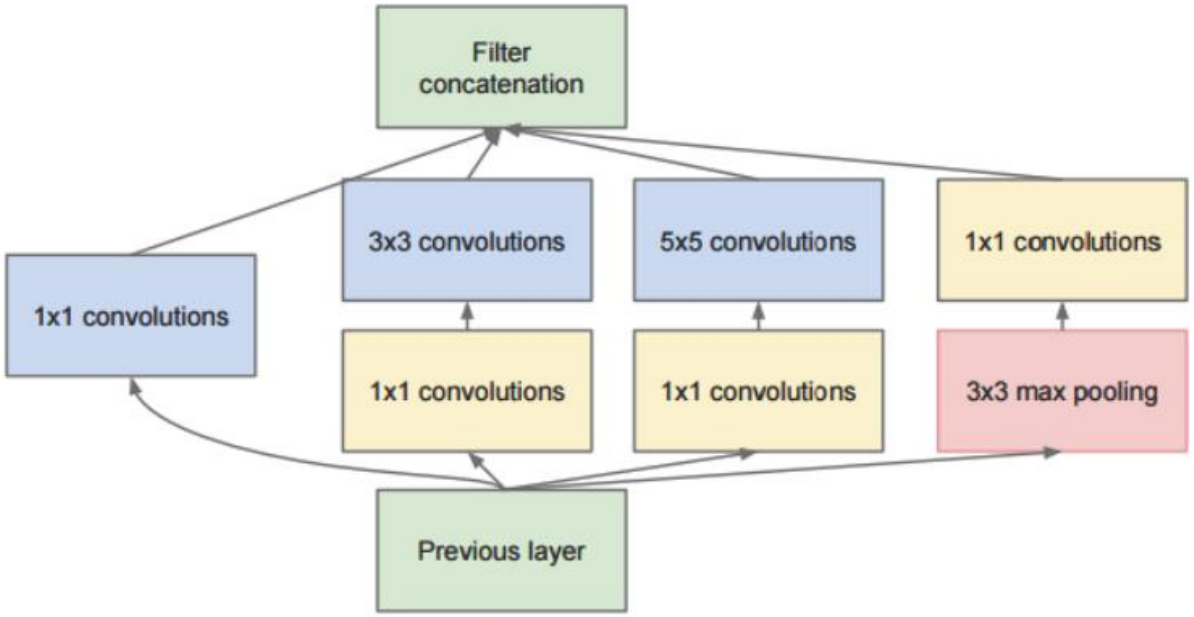
- Note Pooling considered part of the layer
- 96 convolution kernels, then 256, then 384
- Stride of 4 for first convolution kernel, 1 for the rest
- Pooling layers with 3x3 receptive fields and stride of 2 throughout
- Finishes with fully connected (fc) MLP with 2 hidden layers and 1000 output nodes for classes
- Local Response Norm - the outputs of corresponding units of equivalent convolution layer from different kernels are used to normalize using neighbors of points or an accumulative quantity (say mean) over convolution output.

Structure of VGGNet



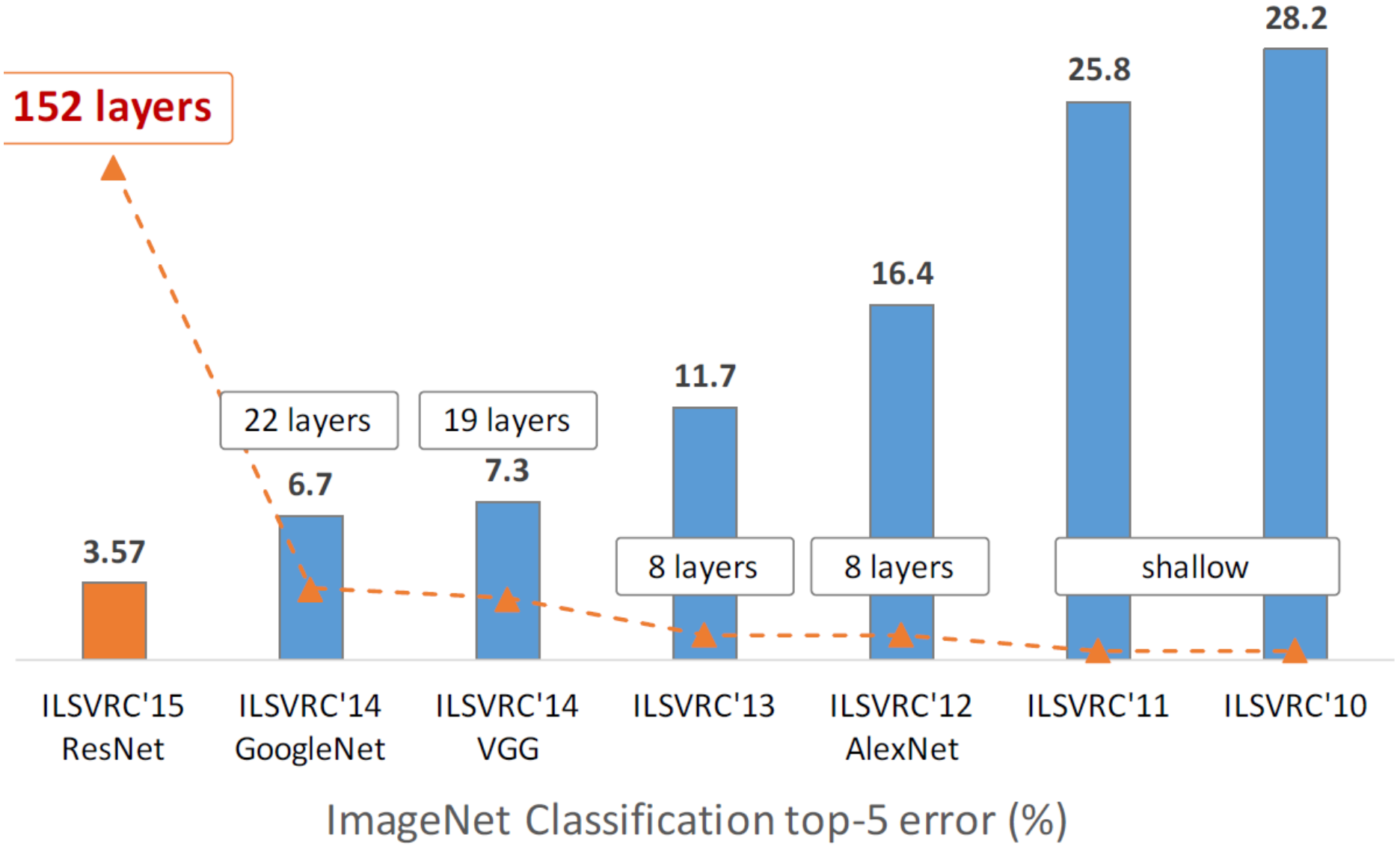
Structure of GoogleNet





The inception module (convolutions and max pooling)

The Revolution of Depth Increasing



AlphaGo

19 x 19 matrix

Black: 1

white: -1

none: 0

Neural Network

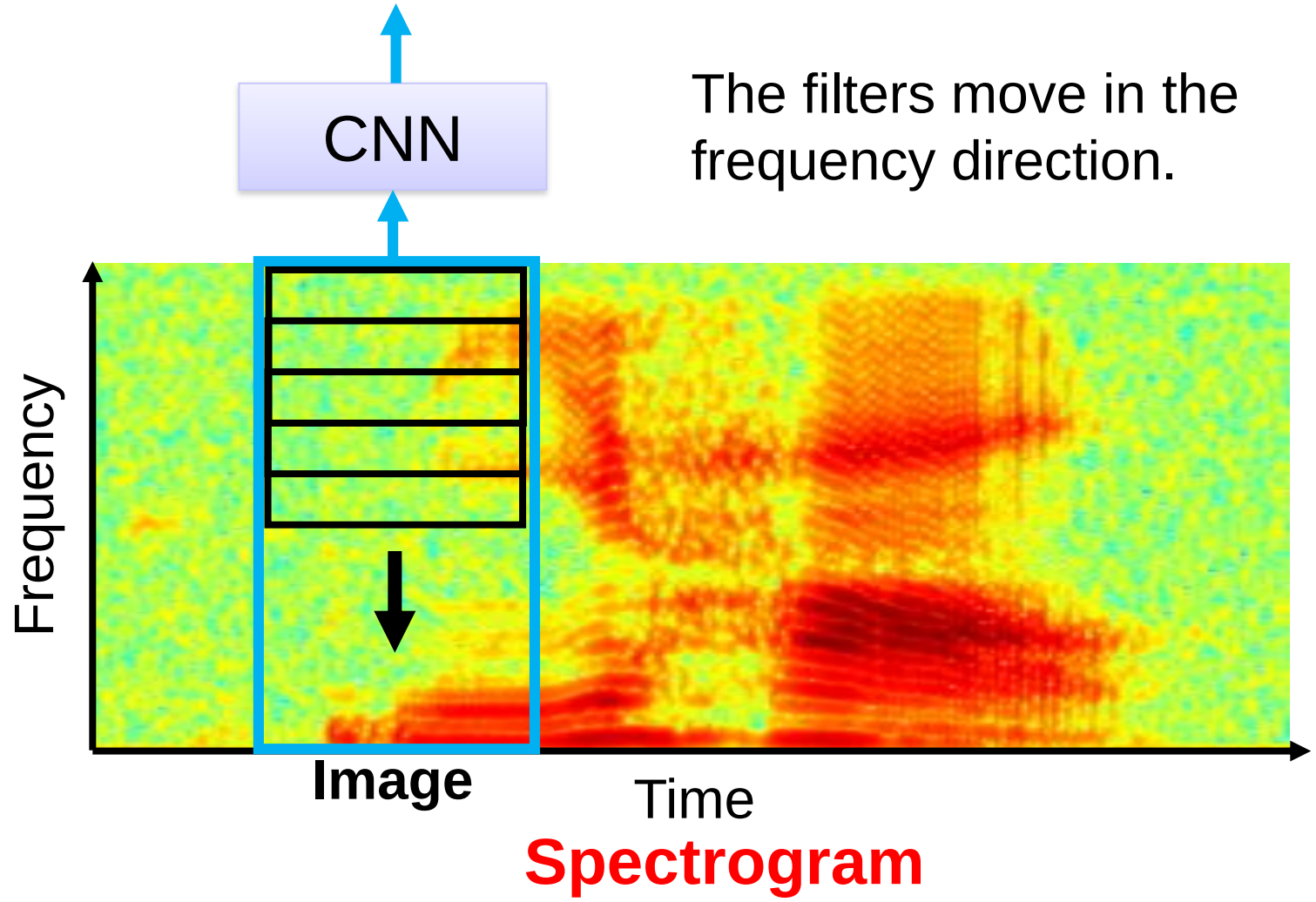
Next move

(19 x 19 positions)

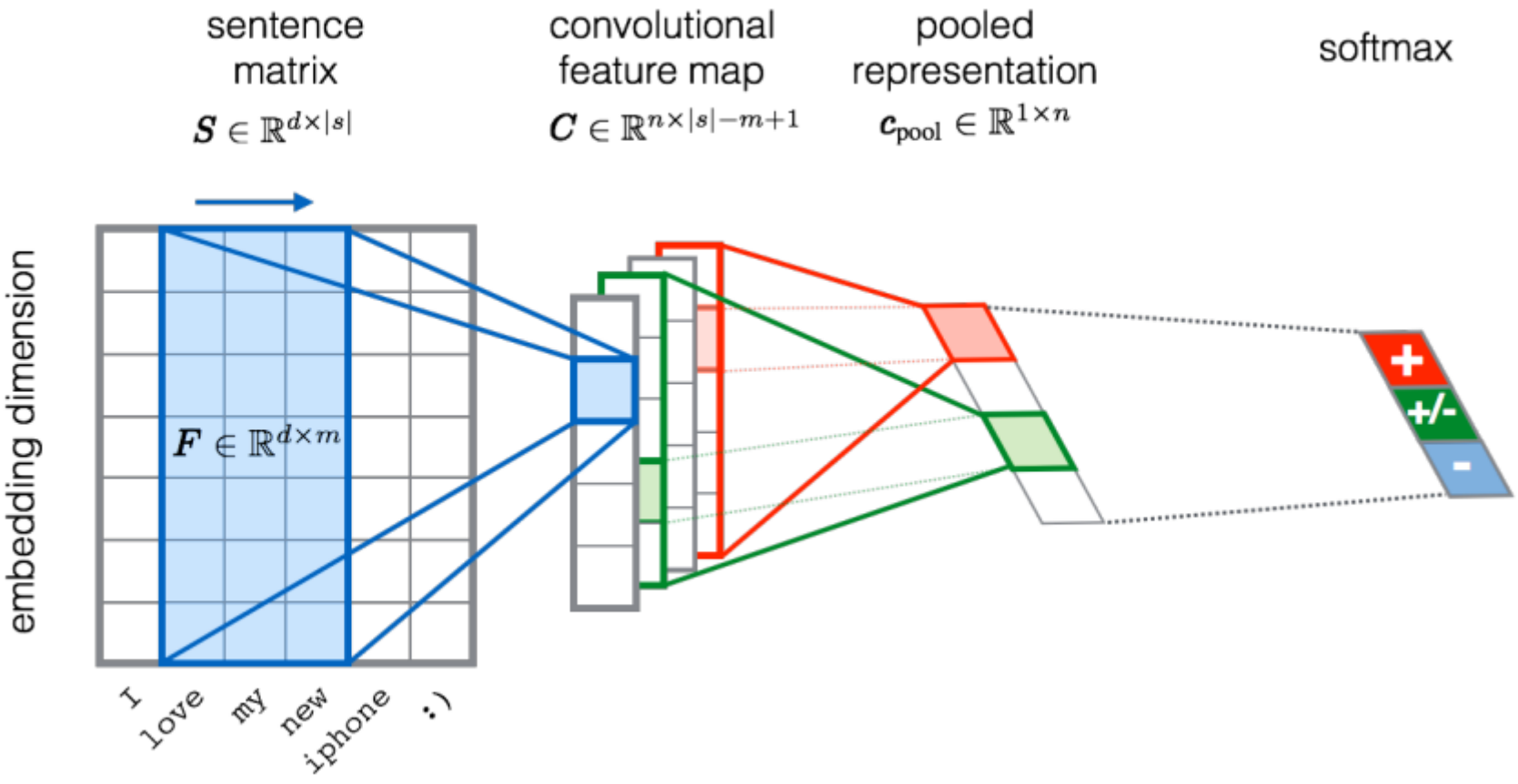
Fully-connected feedforward network can be used

But CNN performs much better

CNN in Speech Recognition



CNN in Text Classification



http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

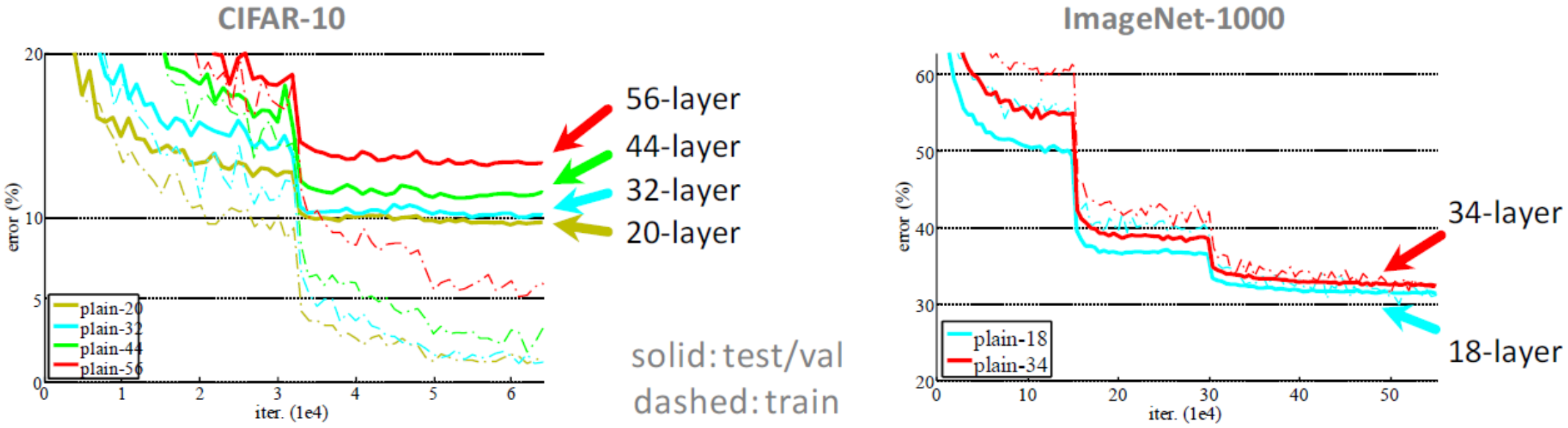
深度學習工具軟體比較表

- Pytorch v.s Keras

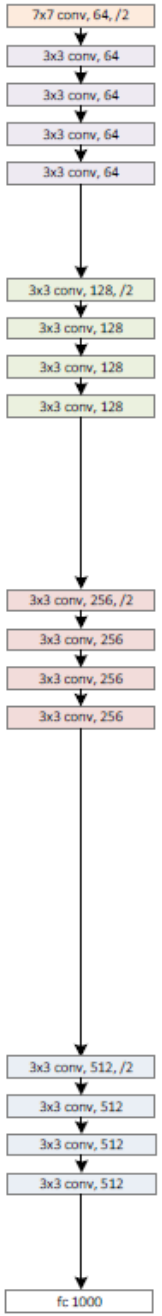
比較項目	PyTorch	Keras	說明
抽象層次	較低	較高	Keras 提供更高階的 API，代表更方便快速開發，架設簡單 (simple) 模型的情況下，使用 Keras 的程式碼行數會明顯比較少；PyTorch 提供較低階的 API，可以更有彈性的使用自定義的模型
入門難度	較高	較低	PyTorch 抽象層次較低，必須對深度模型清楚理解才能夠成功使用，相較之下 Keras 更容易入門
除錯能力	較好	較差	PyTorch 引進動態計算圖機制 (Dynamic Computational Graph) 使得可以逐行建立神經網路模型，在出錯的情況下可以逐行除錯
效能	較好	較差	Ref: 1 (https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/)

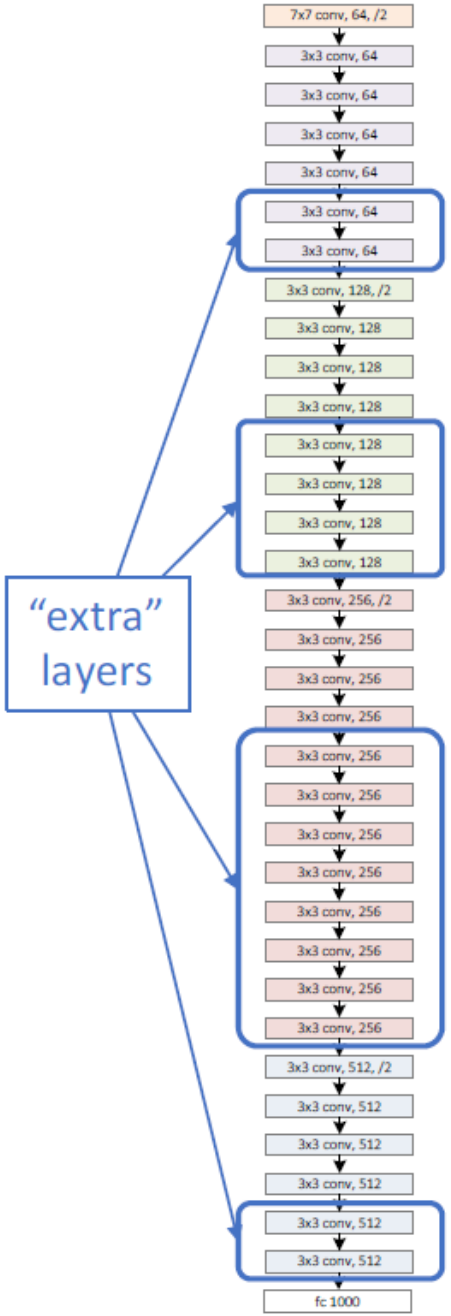
Simply stacking layers?

- *Plain* nets: stacking 3x3 conv layers…
- 56-layer net has **higher training error** and test error than 20-layer net
- "Overly deep" plain nets have **higher training error**
- A general phenomenon, observed in many datasets



a shallower model (18 layers)



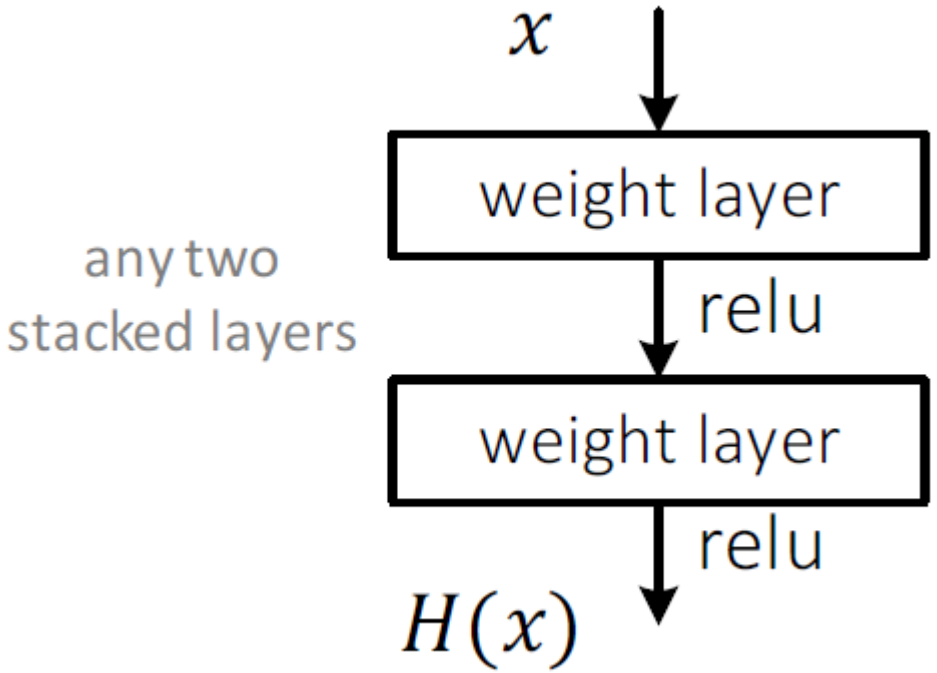


a deeper counterpart (34 layers)

- Richer solution space
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…

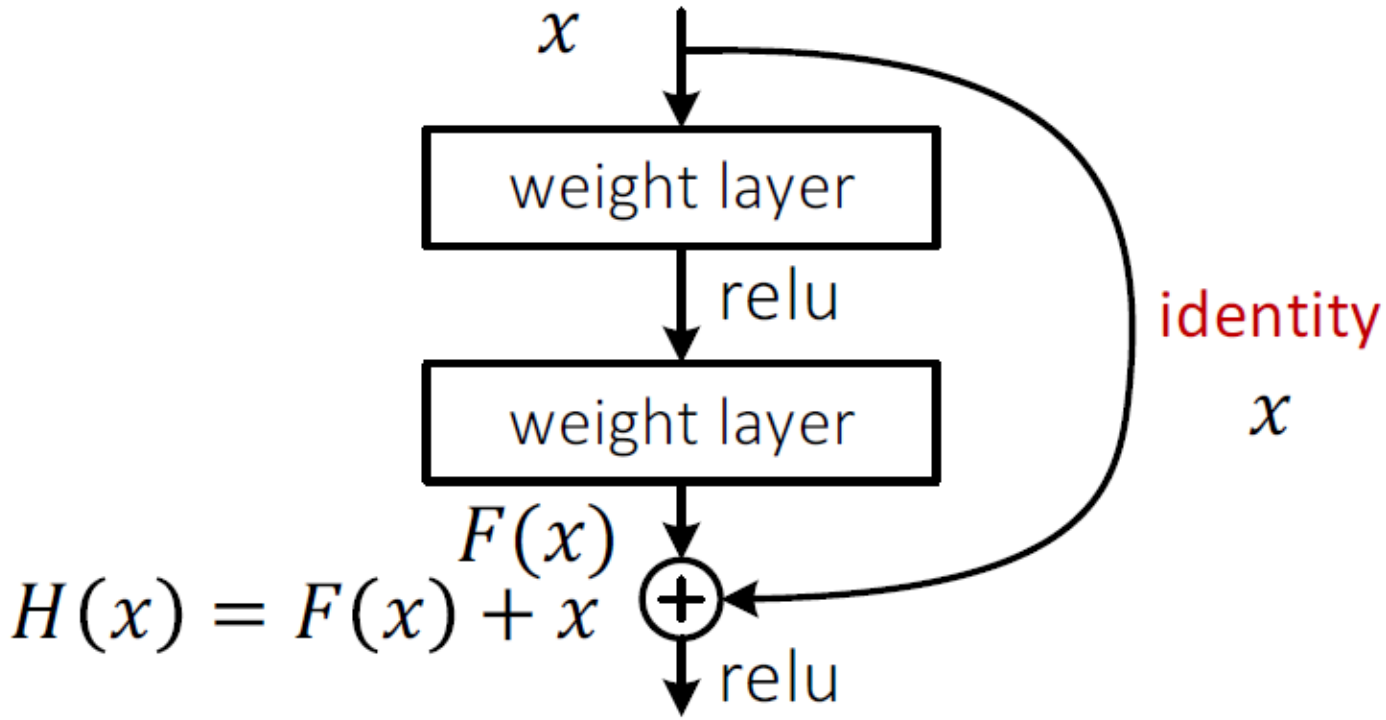
Deep Residual Learning

- Plaint net



- $H\ x$ is any desired mapping,
- Hope the 2 weight layers fit $H(x)$

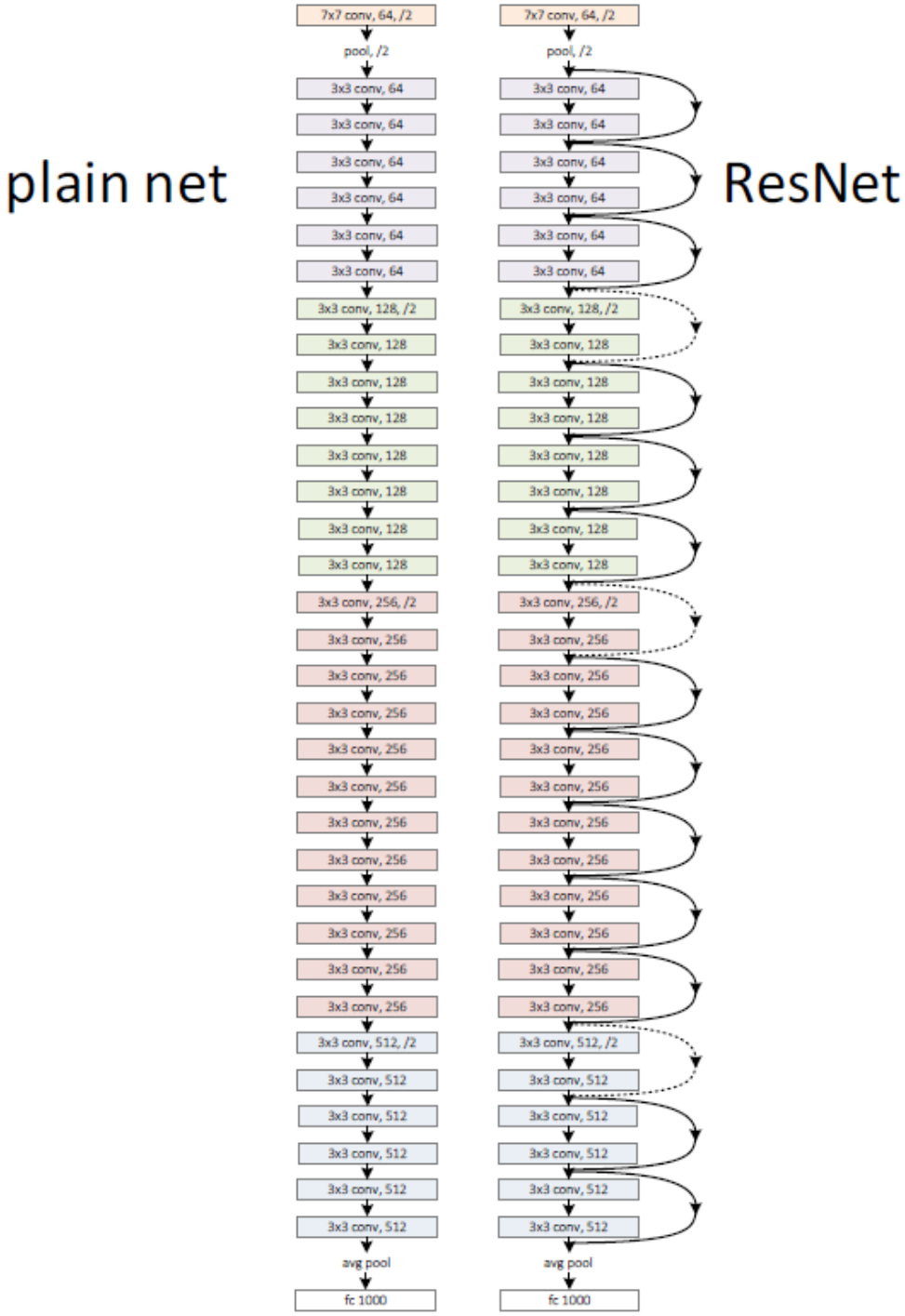
- Residual net



- Hope the 2 weight layers fit $F(x)$
- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

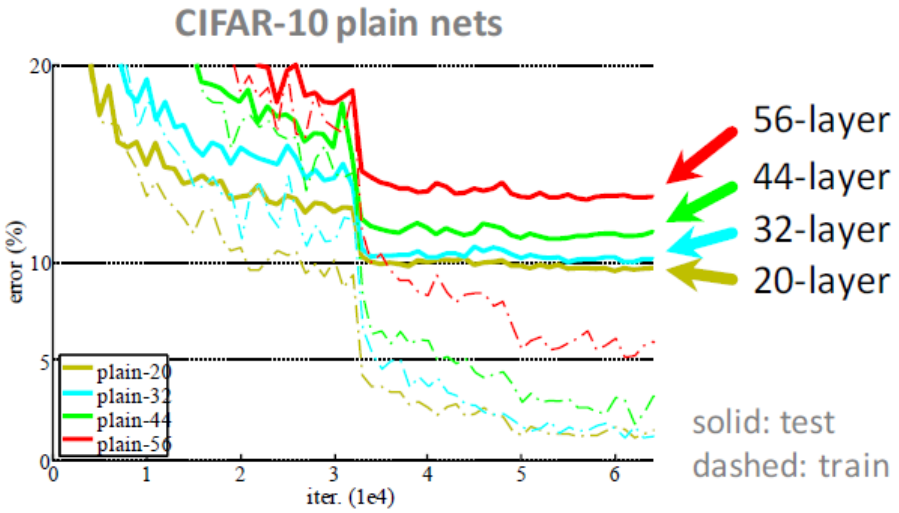
Network Design

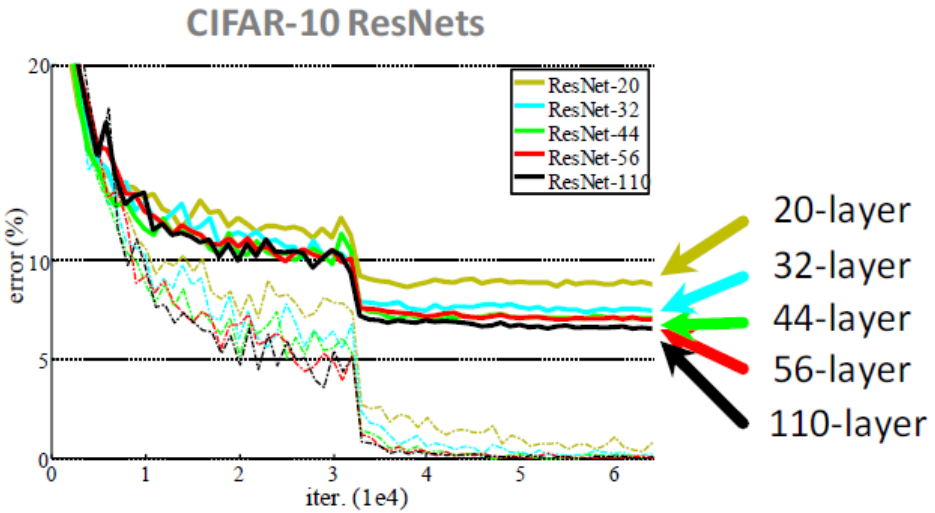
- Keep it simple
- Basic design (VGG-style)
  - all 3x3 conv (almost)
  - spatial size /2 => # filters x2
    (~same complexity per layer)
  - Simple design, just deep!

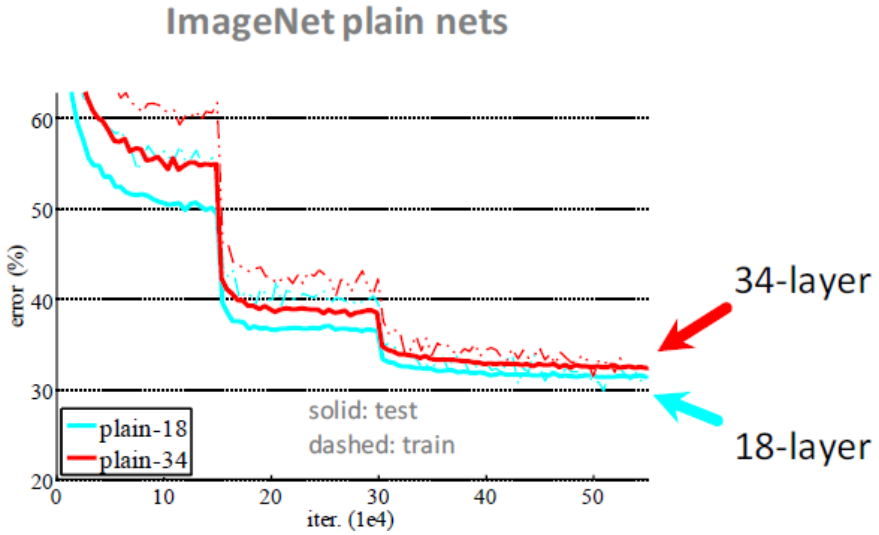


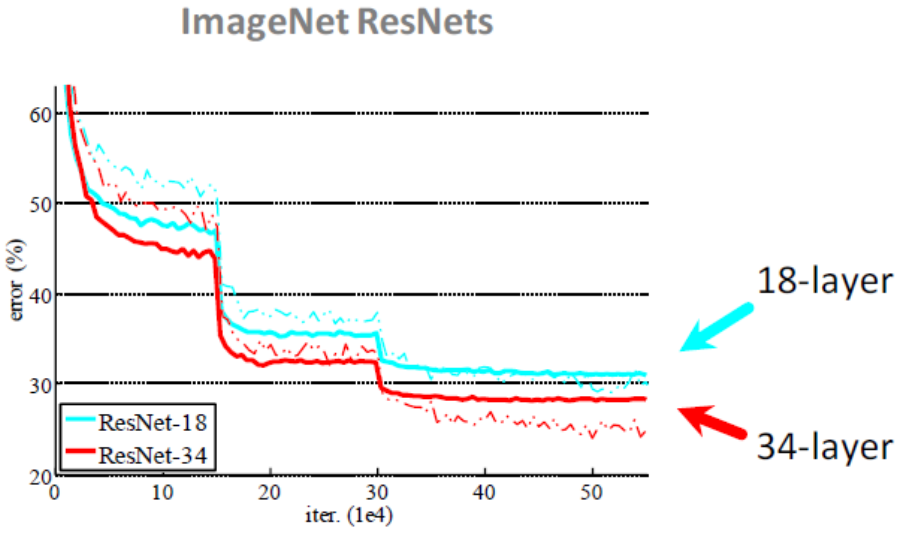
Error Measure

- Deeper ResNets have **lower training error**, and also lower test error

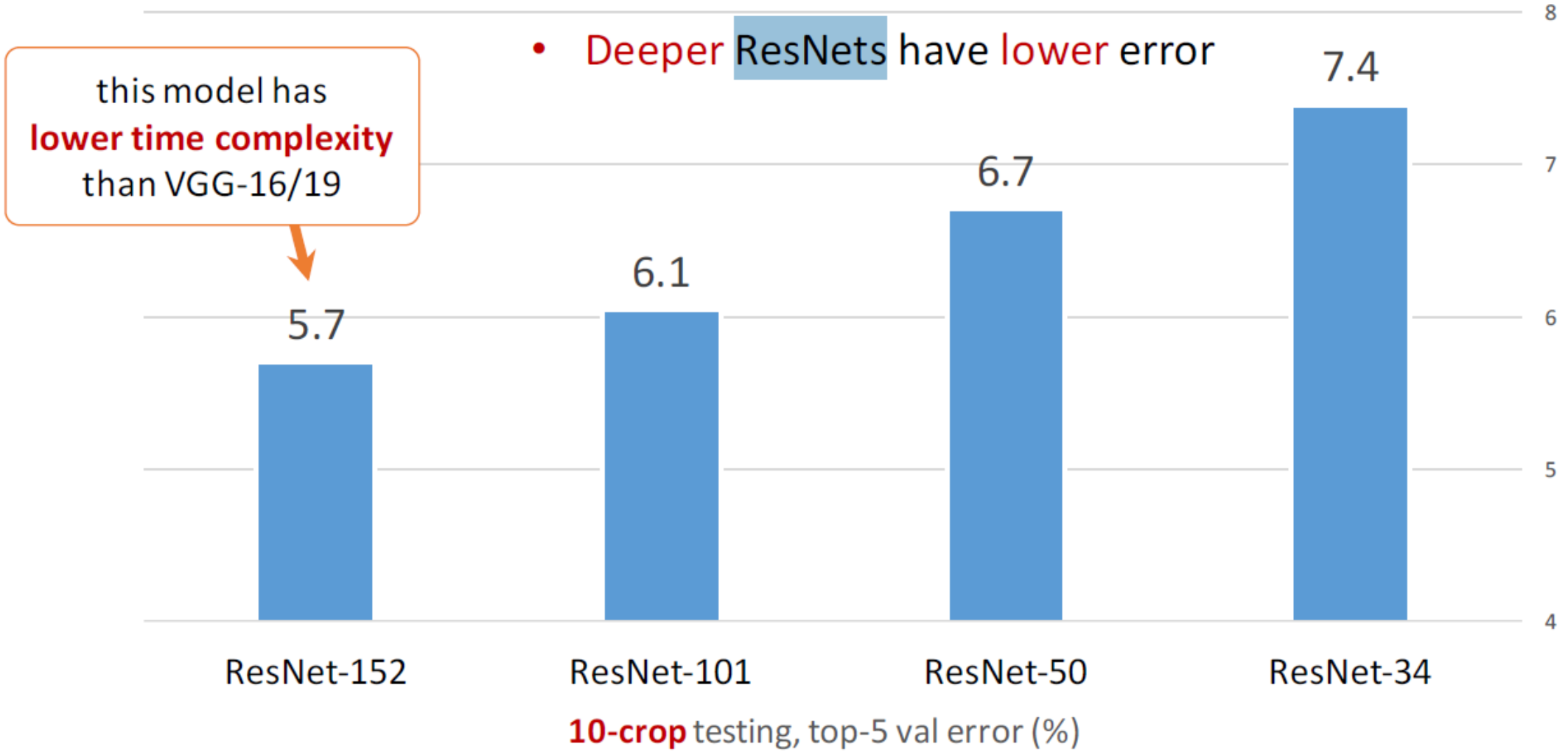






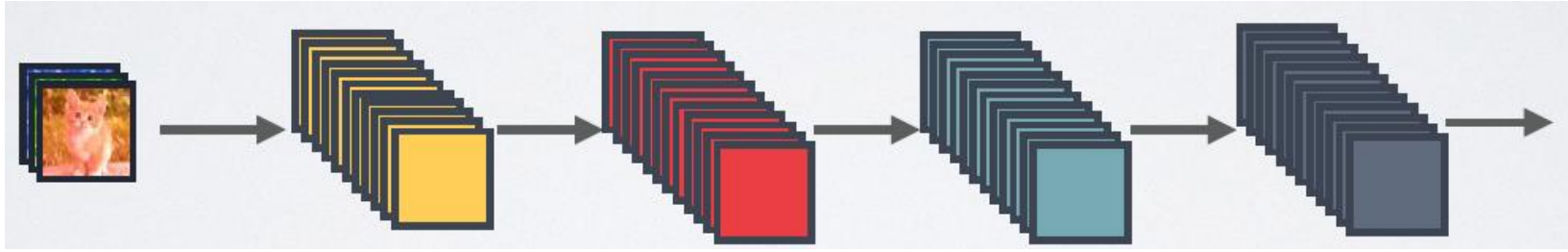


ResNets Go Deeper…

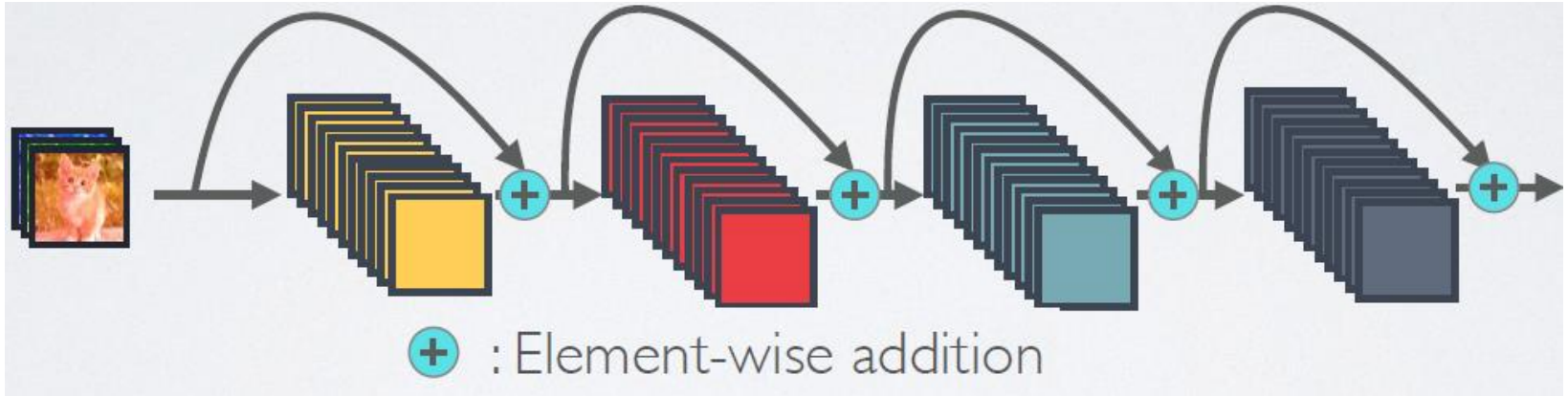


Densely Connected Convolutional Networks

- Network connection
  - Standard connection

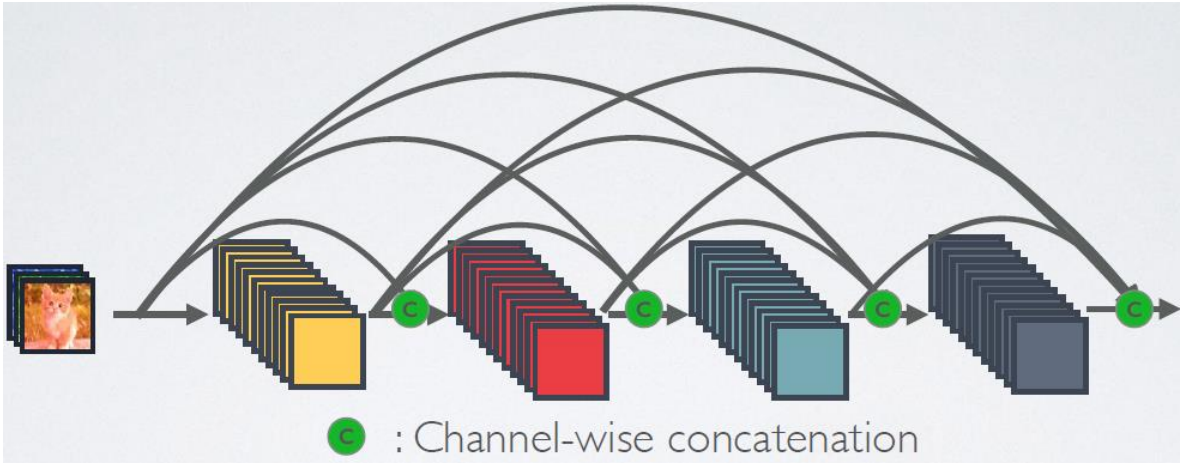


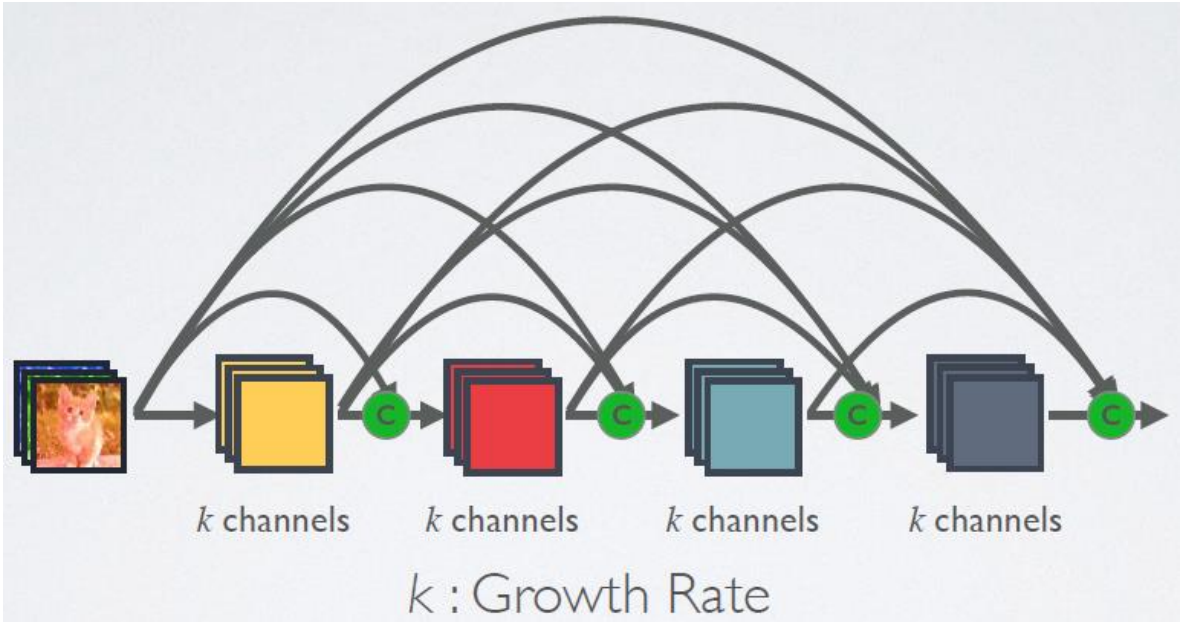
  - ResNet



Densely Connected Convolutional Networks

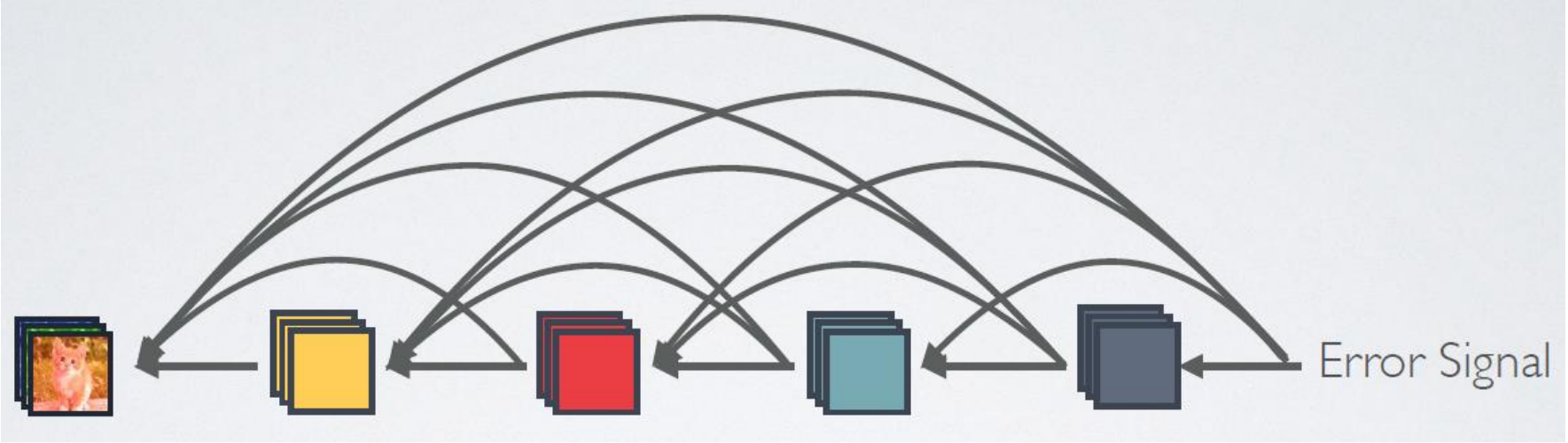
- DenseNet





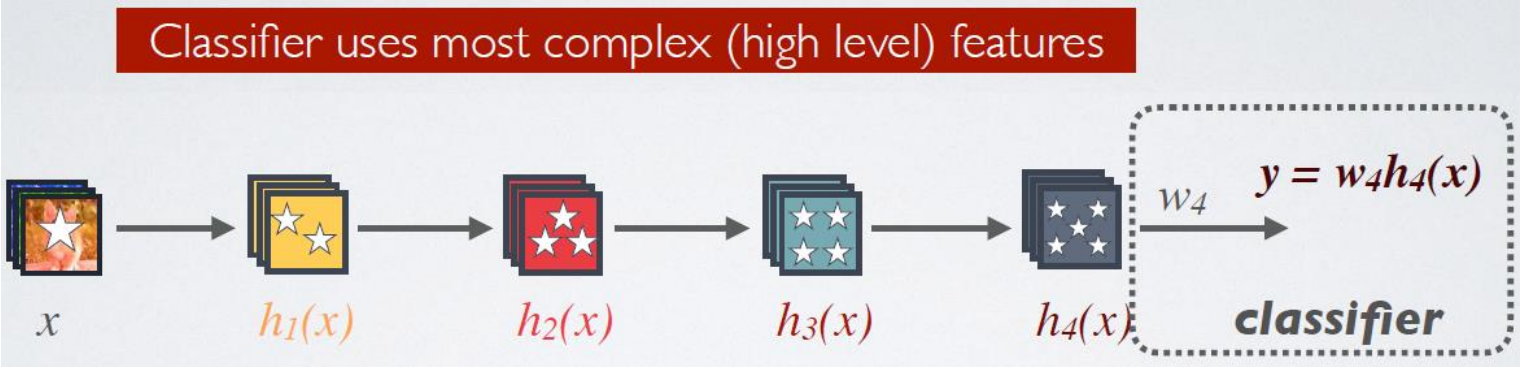
Advantages of DenseNet

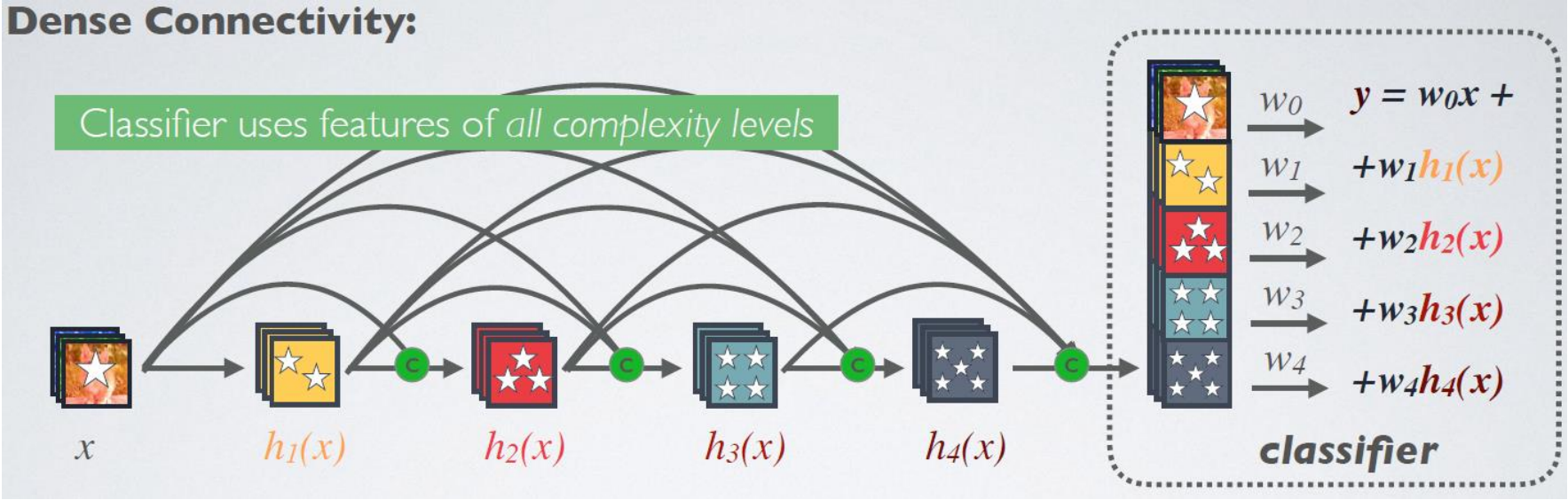
- Strong gradient flow
  - Alleviate the vanishing-gradient problem
- Strengthen feature propagation.



Advantages of DenseNet

- Encourage feature reuse

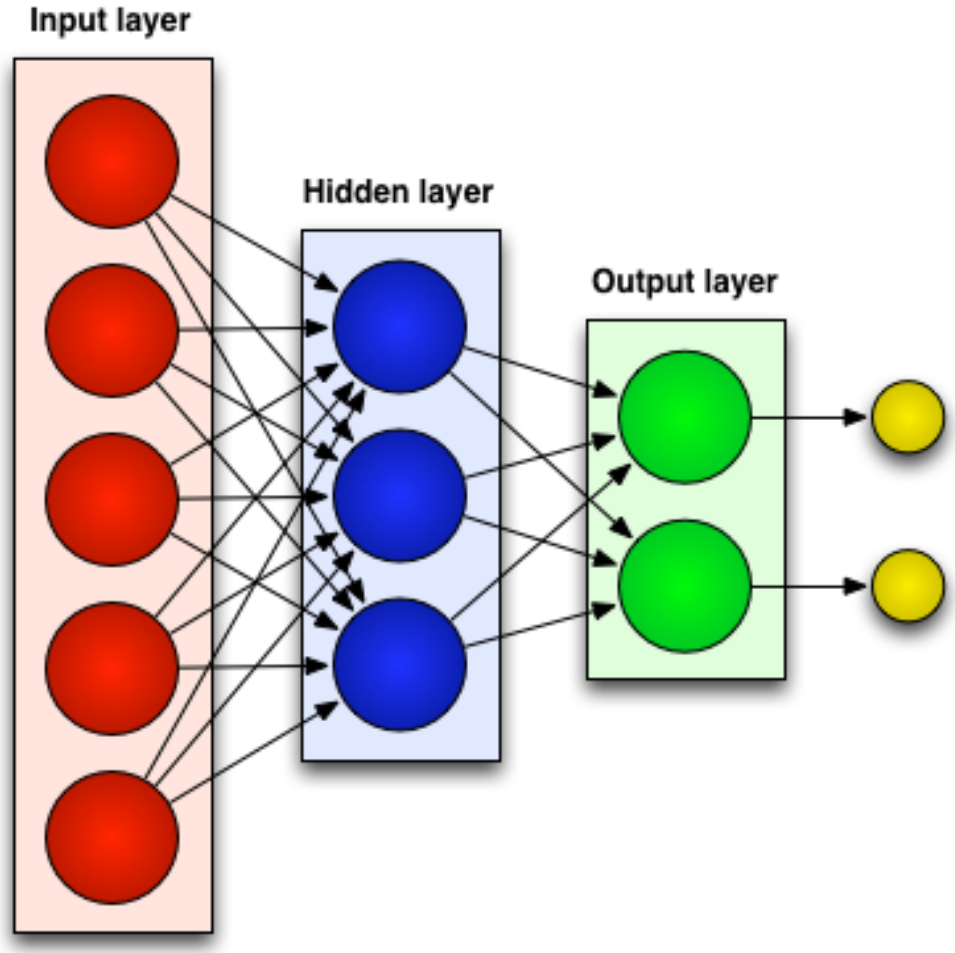




Recurrent Neural Network (RNN) & Long Short-Term Memory

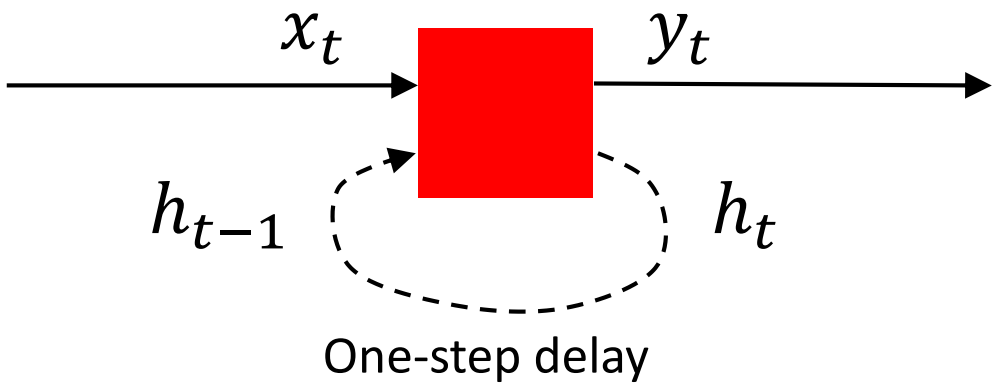
Feed-Forward Neural Networks

- Feedforward Neural Networks:
  - Connections between the units do not form a cycle.
  - The topological ordering is used for activation propagation, and for gradient back-propagation.



Recurrent Neural Network (RNN)

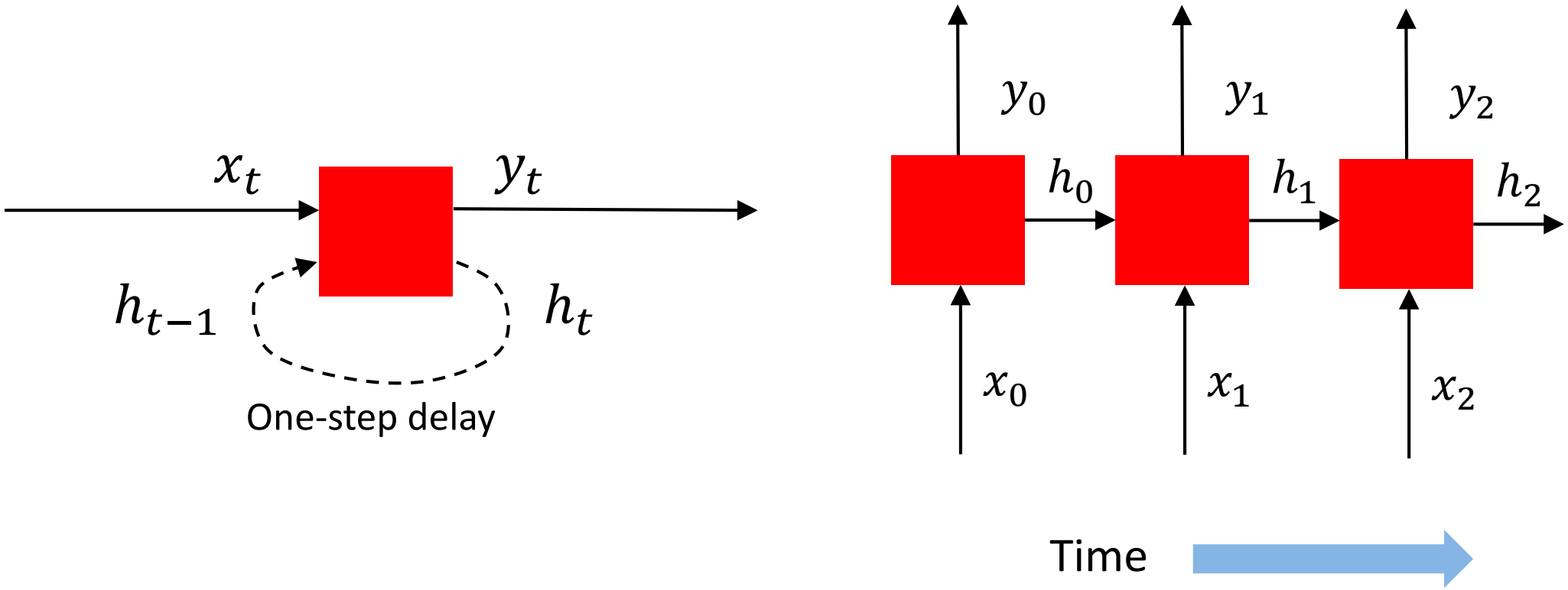
- We now will input one x_i at a time, and re-use the same edge weights.



- Recurrent networks introduce cycles and a notion of time.
  - They are designed to process sequences of data $x_1, \ldots, x_n$ and can produce sequences of outputs $y_1, \ldots, y_m$.

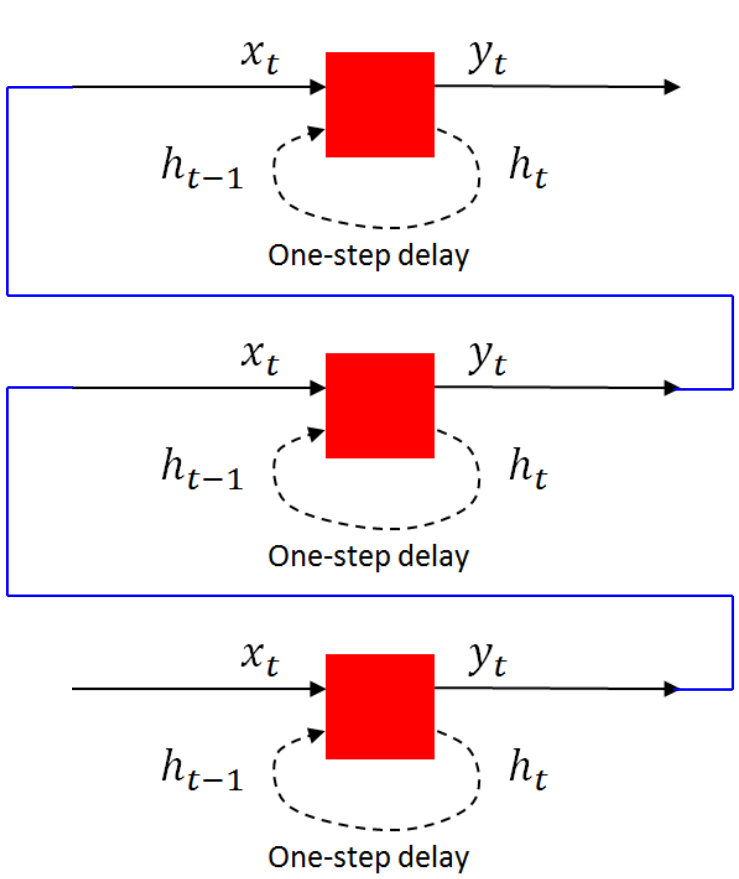
Recurrent Neural Network (RNN)

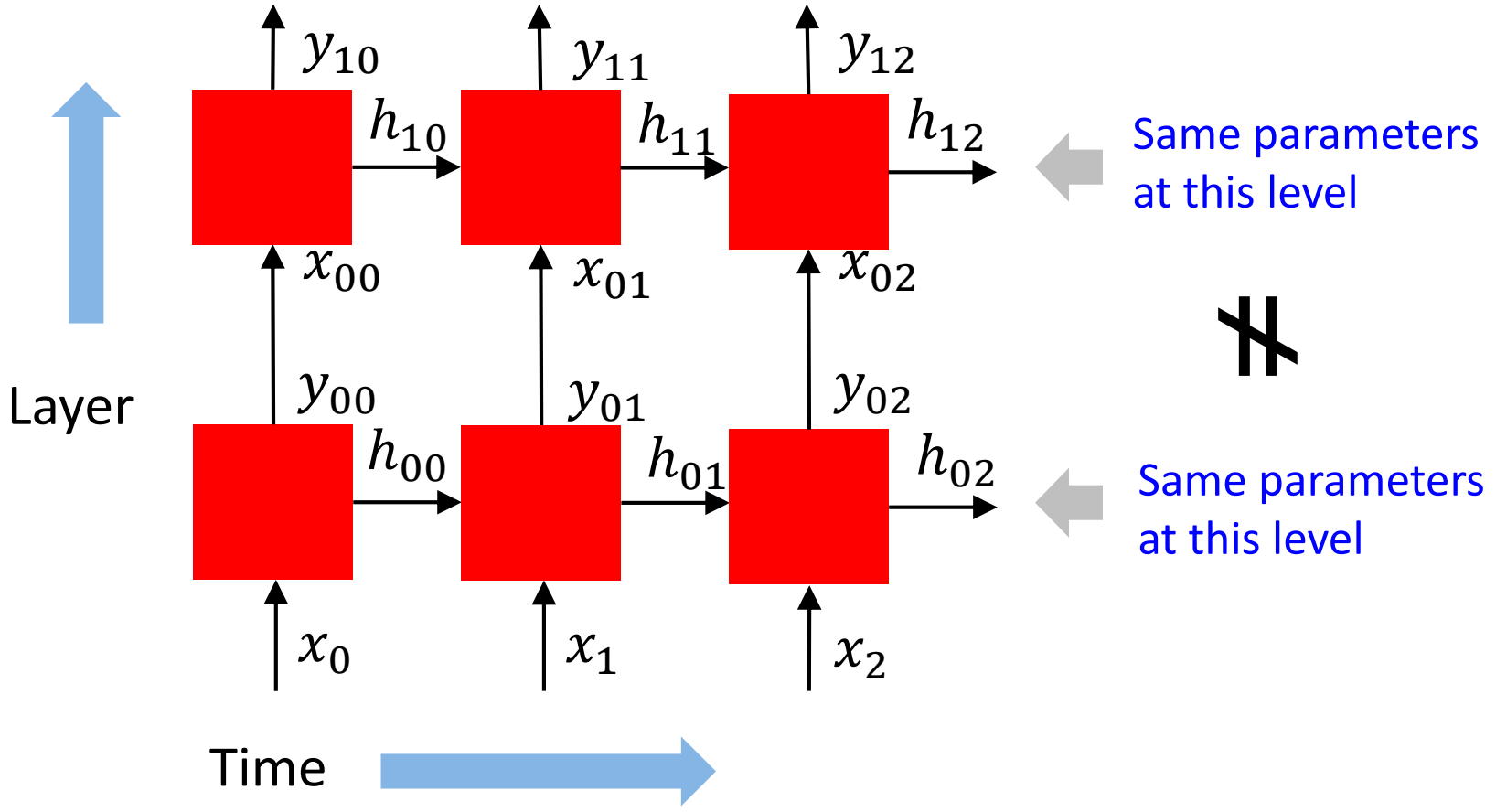
- RNNs can be unrolled across multiple time steps.



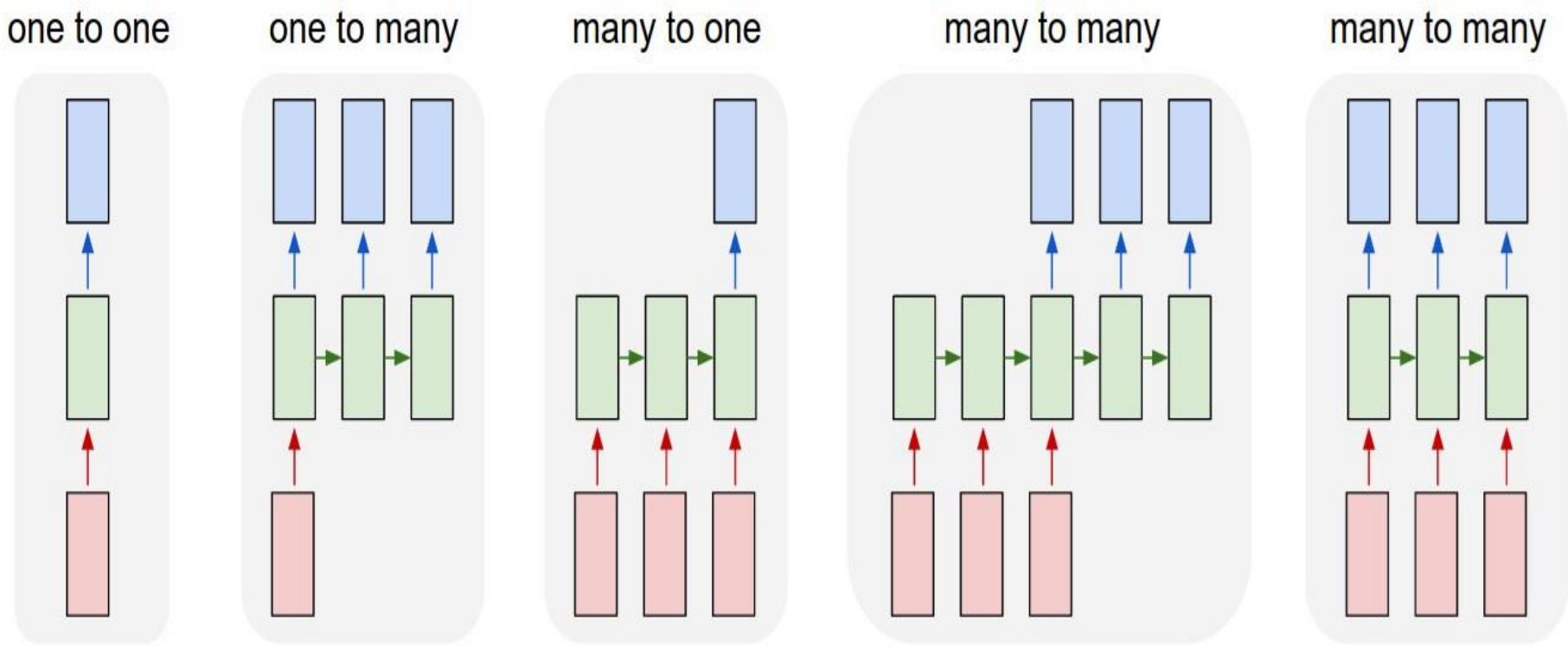
RNN Structure

- Layers can be stacked vertically (deep RNNs):





Flexibility of Recurrent Networks



Vanilla Neural Networks

Flexibility of Recurrent Networks

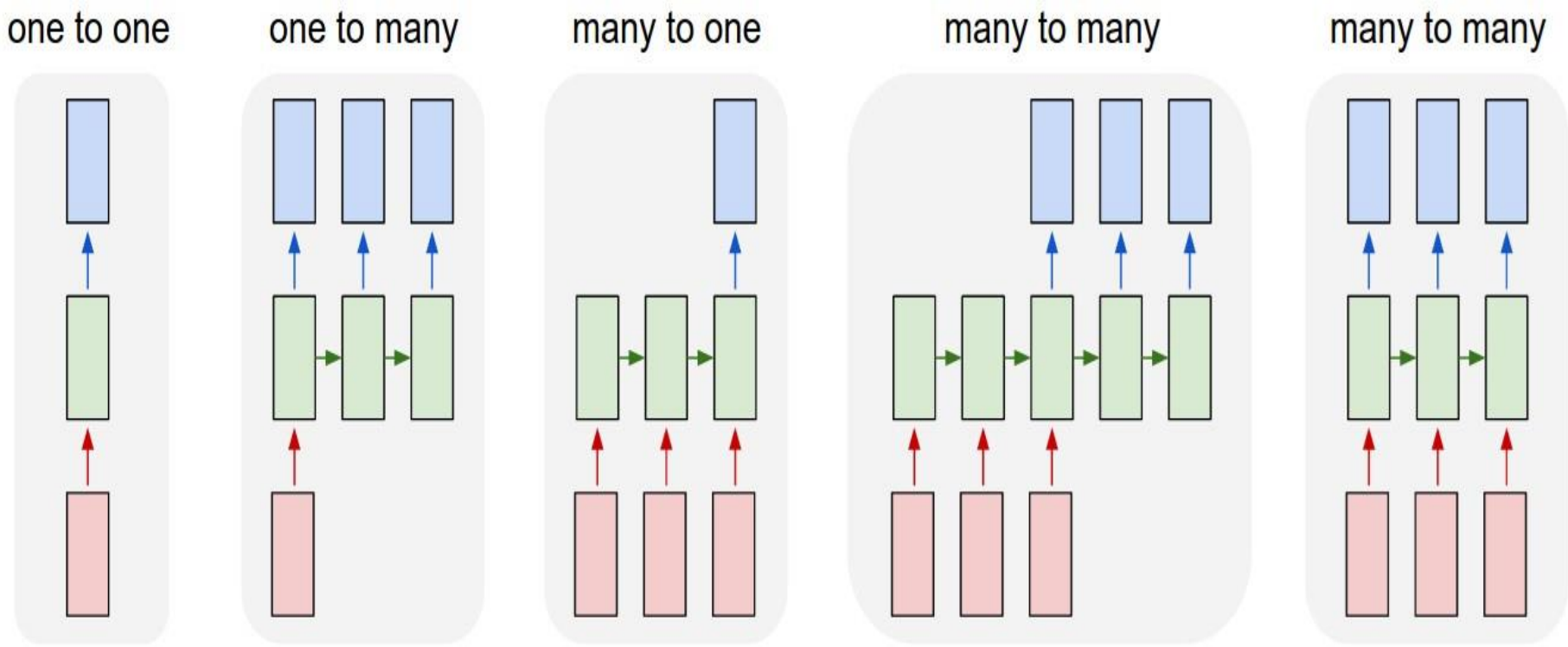
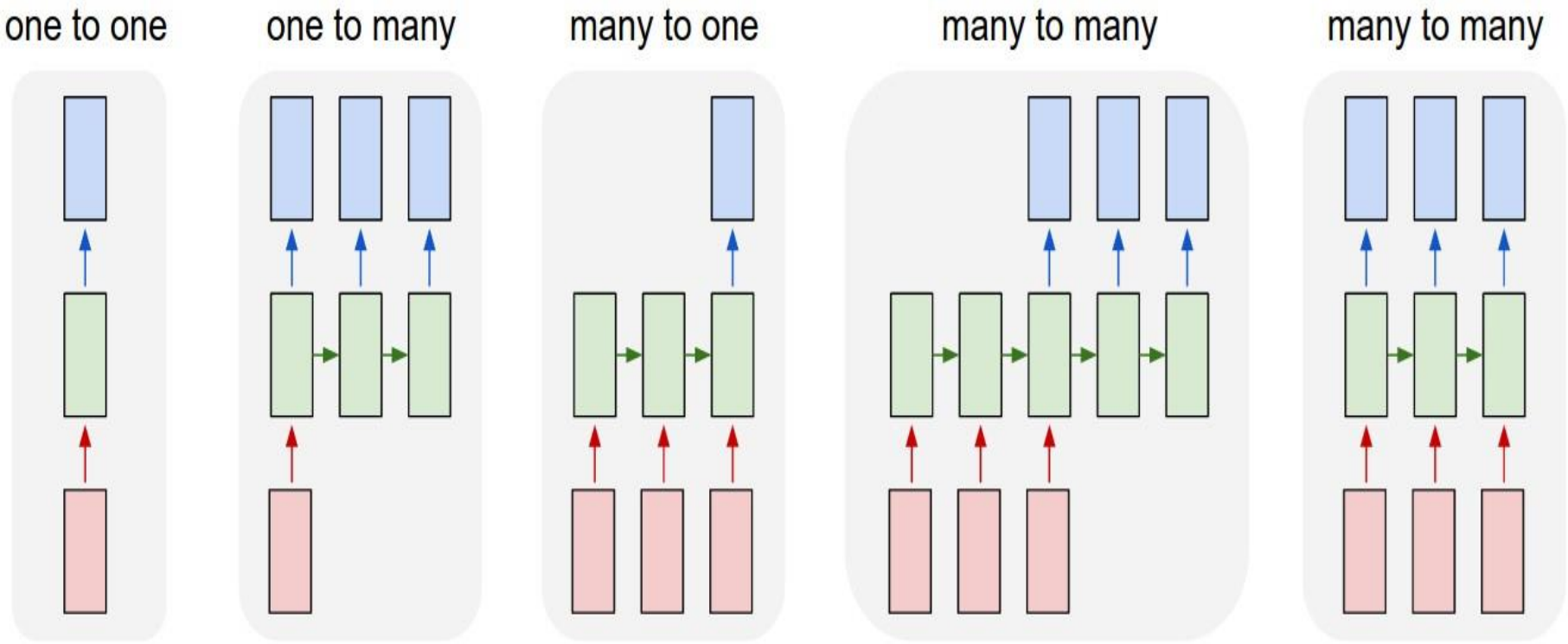


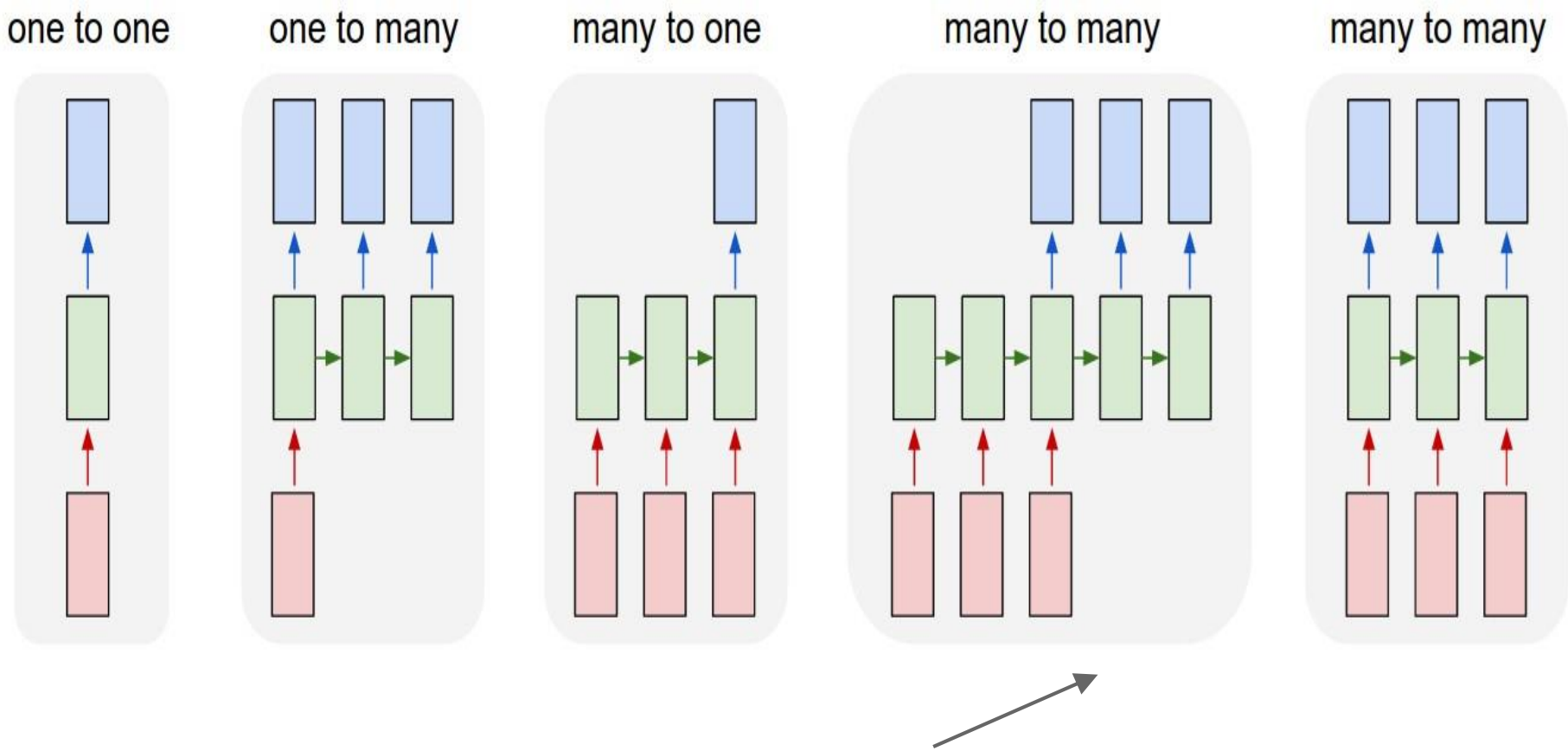
Image Captioning
image -> sequence of words

Flexibility of Recurrent Networks



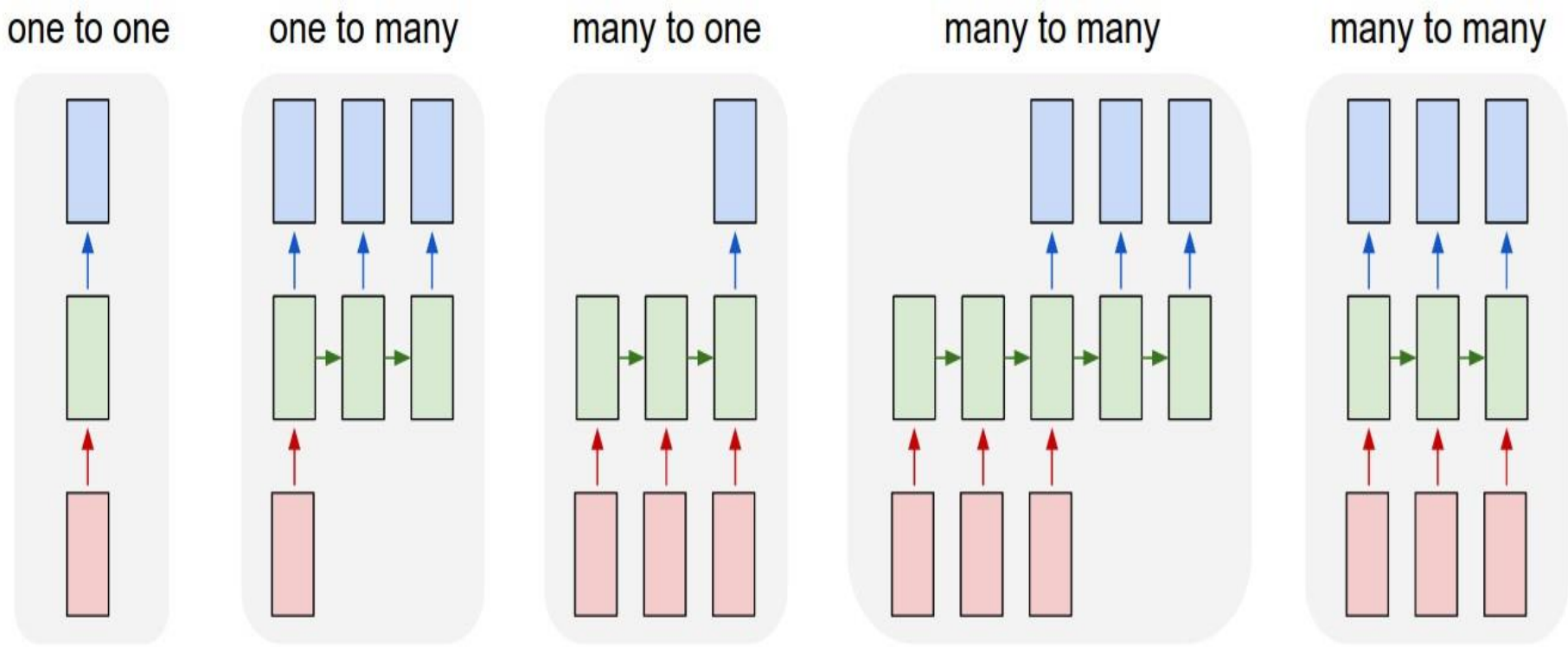
Sentiment Classification
sequence of words -> sentiment

Flexibility of Recurrent Networks



e.g. **Machine Translation**
seq of words -> seq of words

Flexibility of Recurrent Networks



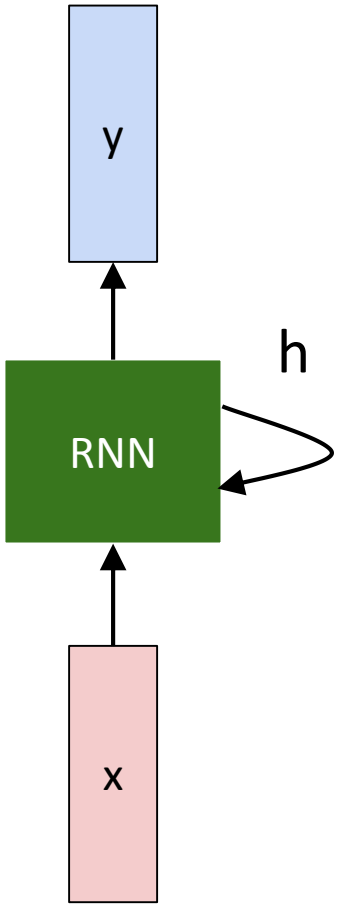
Video classification on frame level

Property of Recurrent Neural Network

- RNNs are a neural network with memory.
- Recurrent since they receive inputs, update the hidden states depending on the previous computations, and make predictions for every element of a sequence.
- RNNs are very powerful for sequence tasks, such as speech recognition, since they maintain a state vector that implicitly contains information about the history of all the past elements of a sequence.

Formulation of Recurrent Neural Network

- We can process a sequence of vectors x by applying a recurrence formula at every time step:



$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

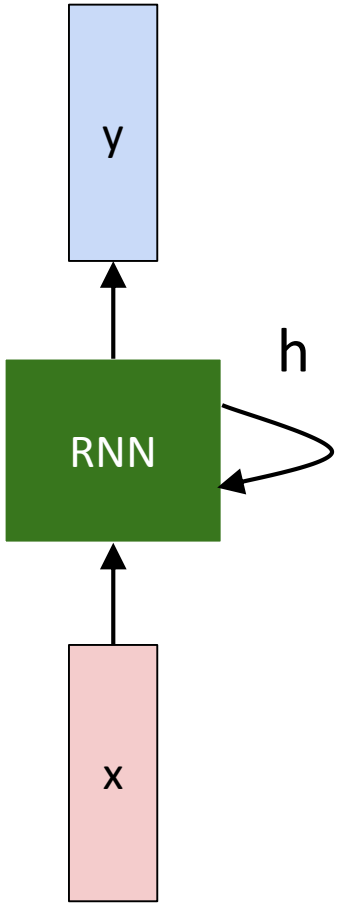
old state

input vector at some time step

Notice: the same function and the same set of parameters are used at every time step.

Formulation of Recurrent Neural Network

- The state consists of a single “hidden” vector h:



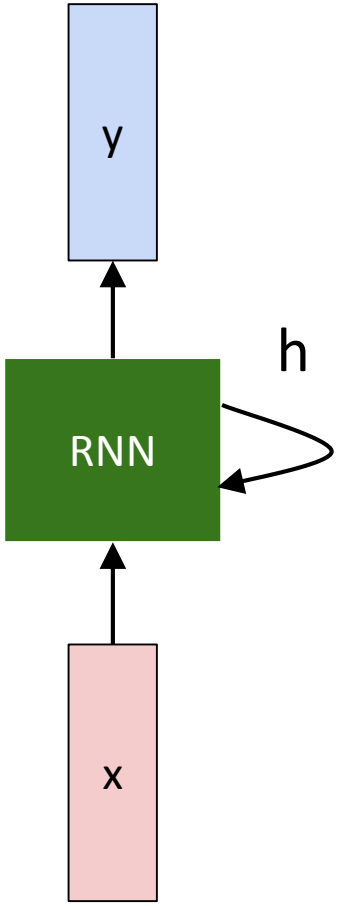
$$y_t = W_{hy} h_t$$

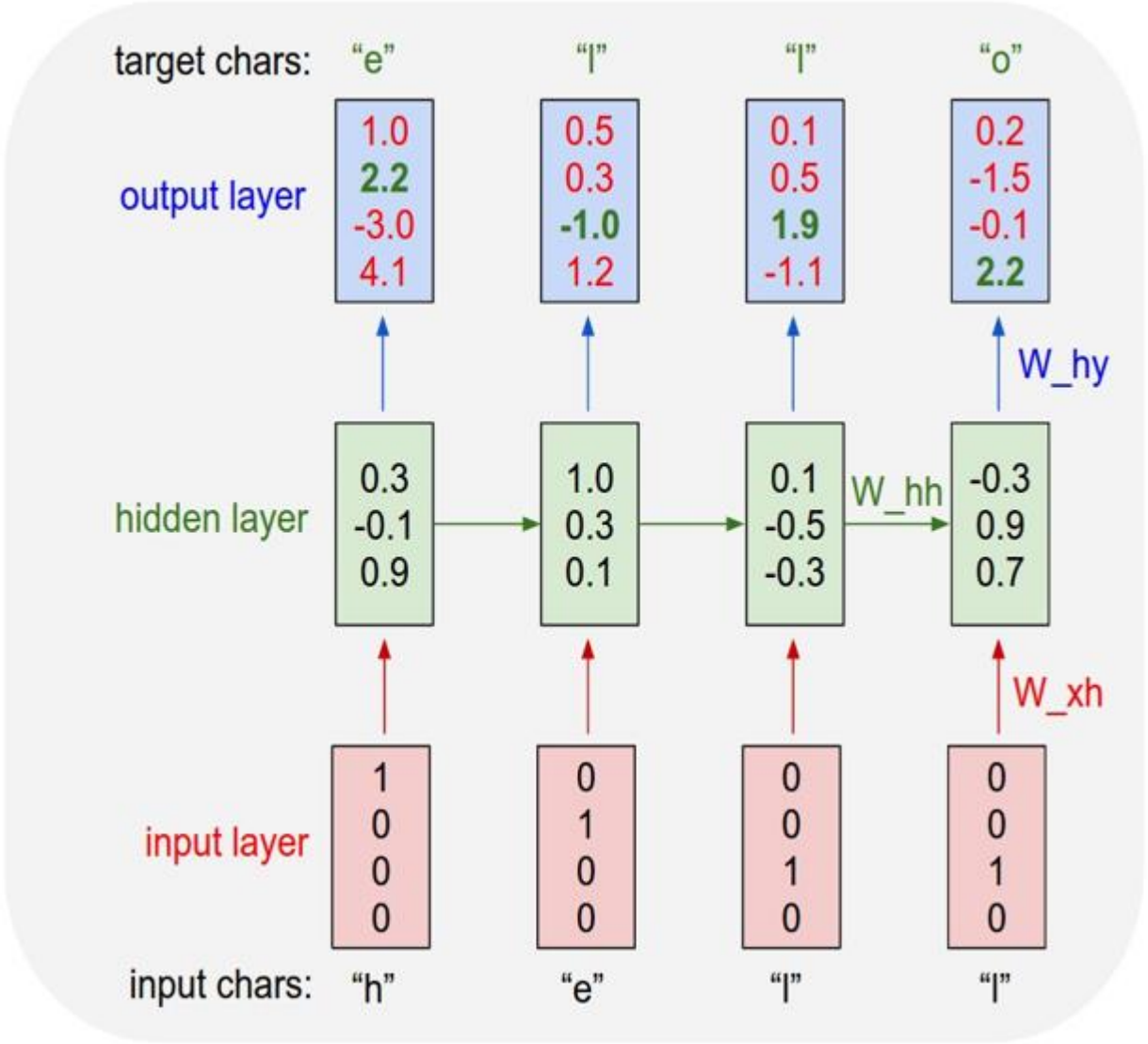
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$h_t = f_W(h_{t-1}, x_t)$$

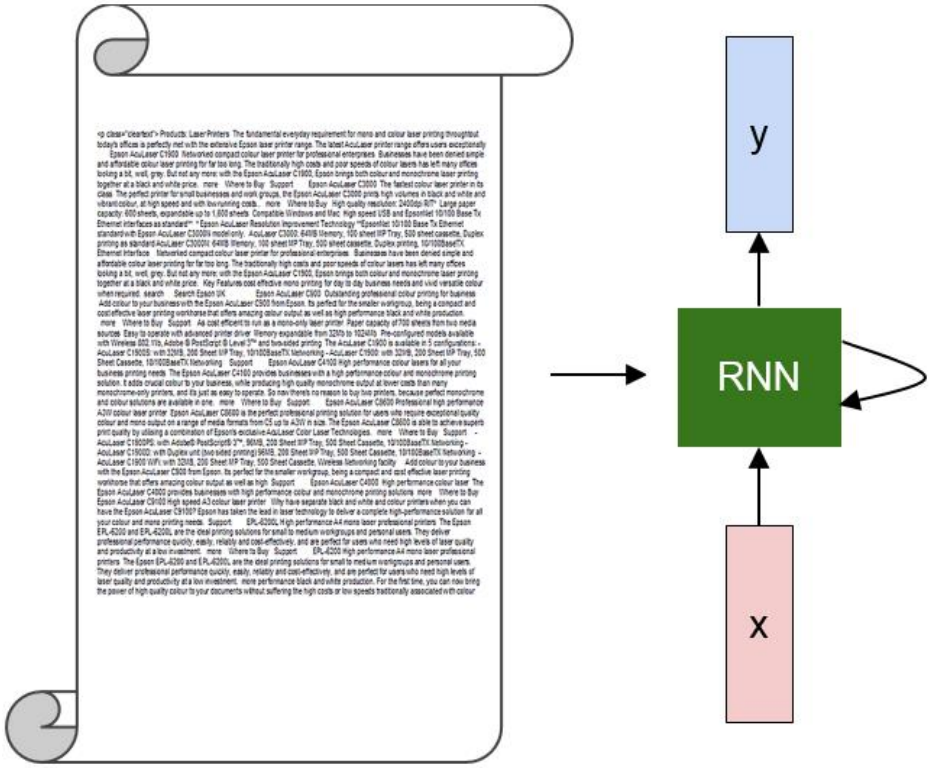
Applications : Character-level Language Model

- Use Vocabulary [h,e,l,o] to train sequence “hello”





Applications : Text Generation Model



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Applications : Image Captioning

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

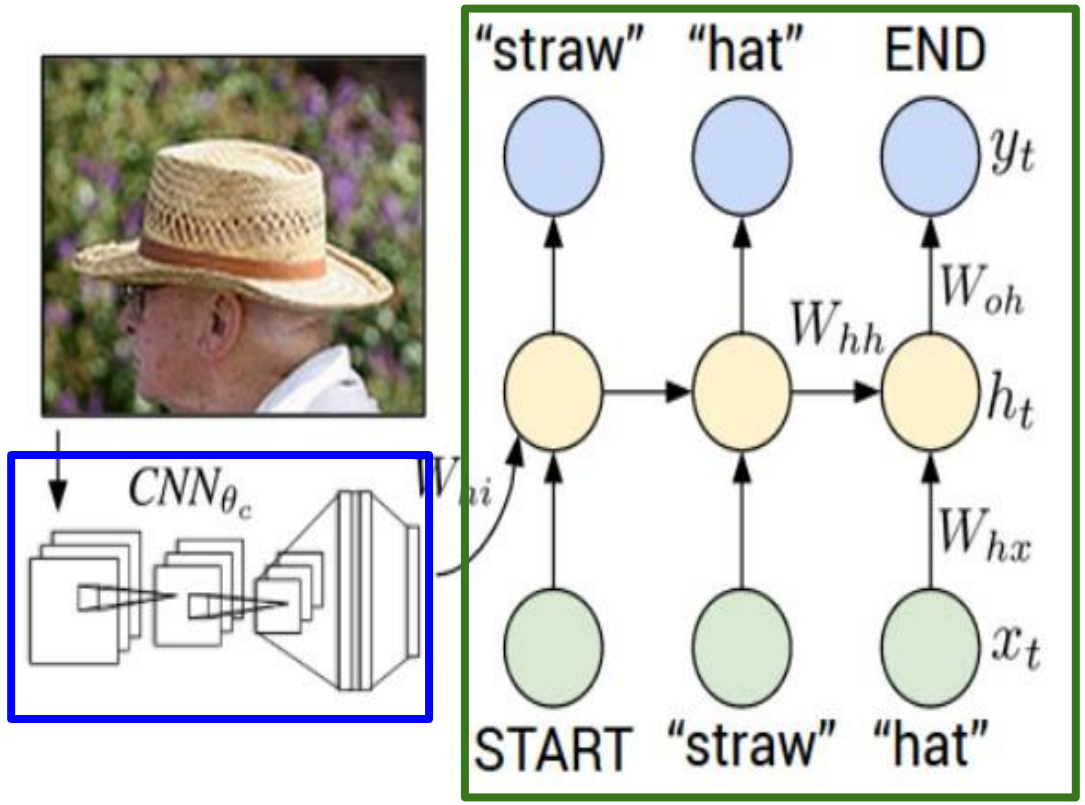
"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

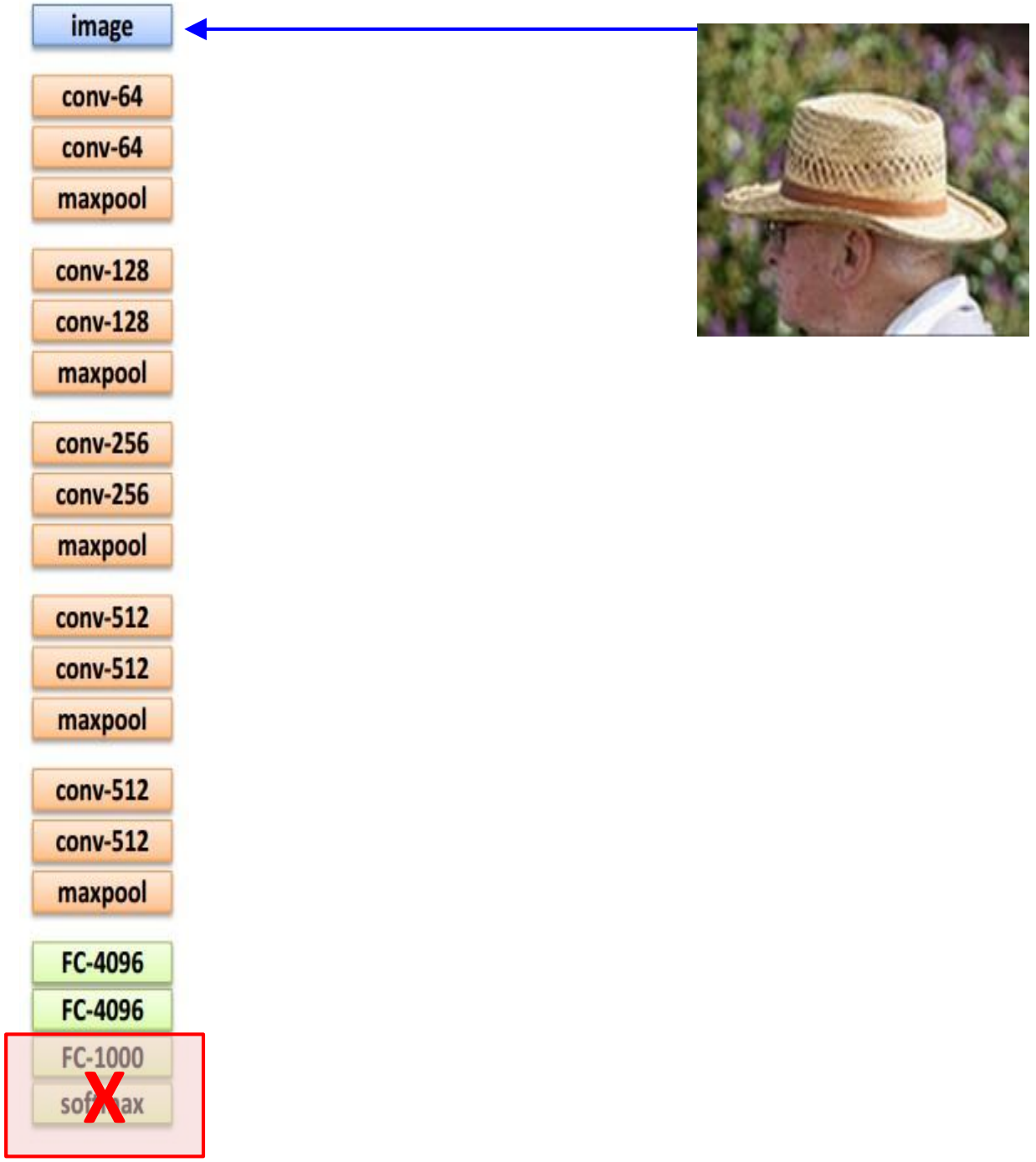
Applications : Image Captioning

- Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
- Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei Li.
- Show and Tell: A Neural Image Caption Generator, Vinyals et al.
- Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
- Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick.

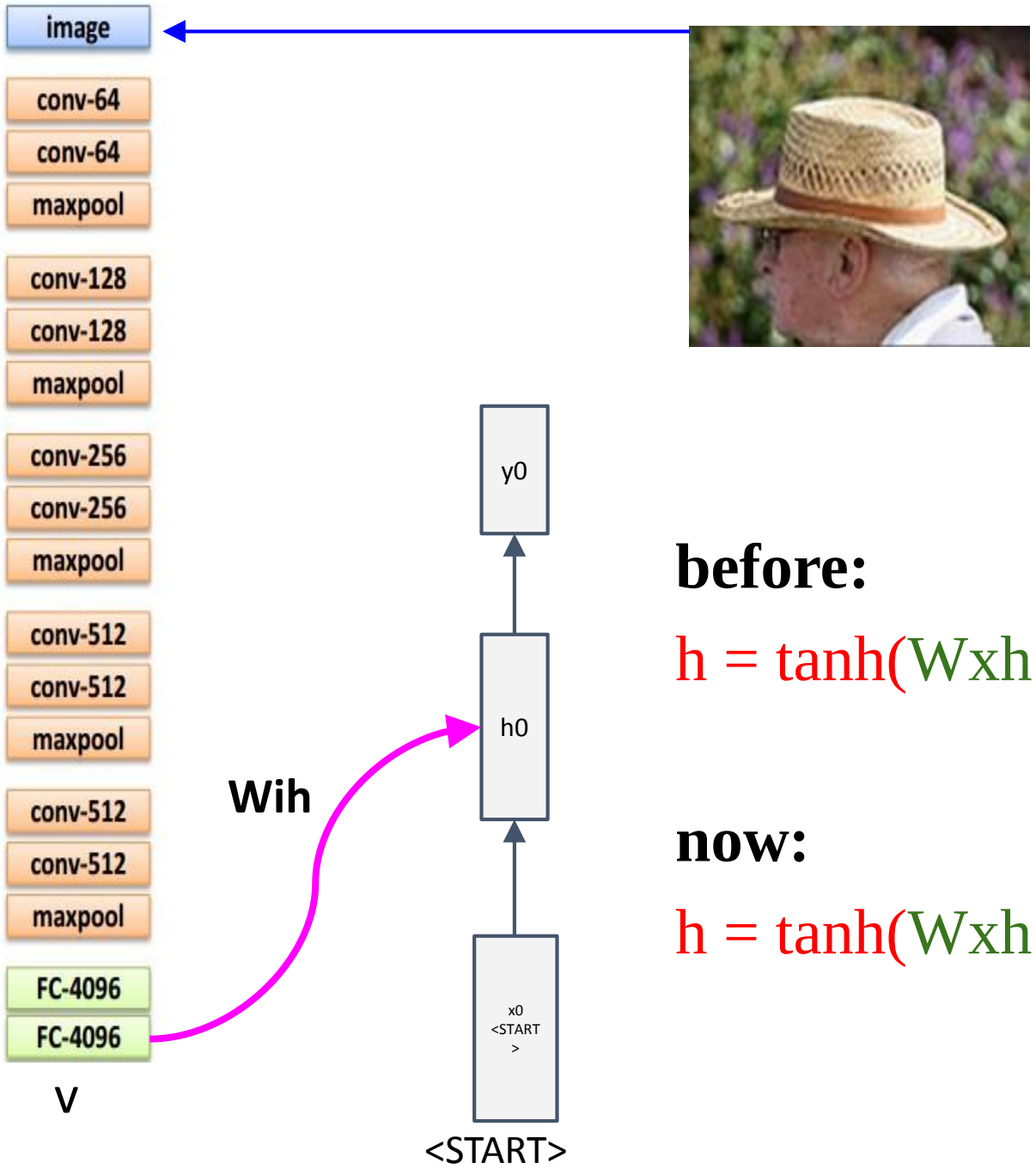
Recurrent Neural Network



Convolutional Neural Network



Training Image



Training Image

before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih * v**)

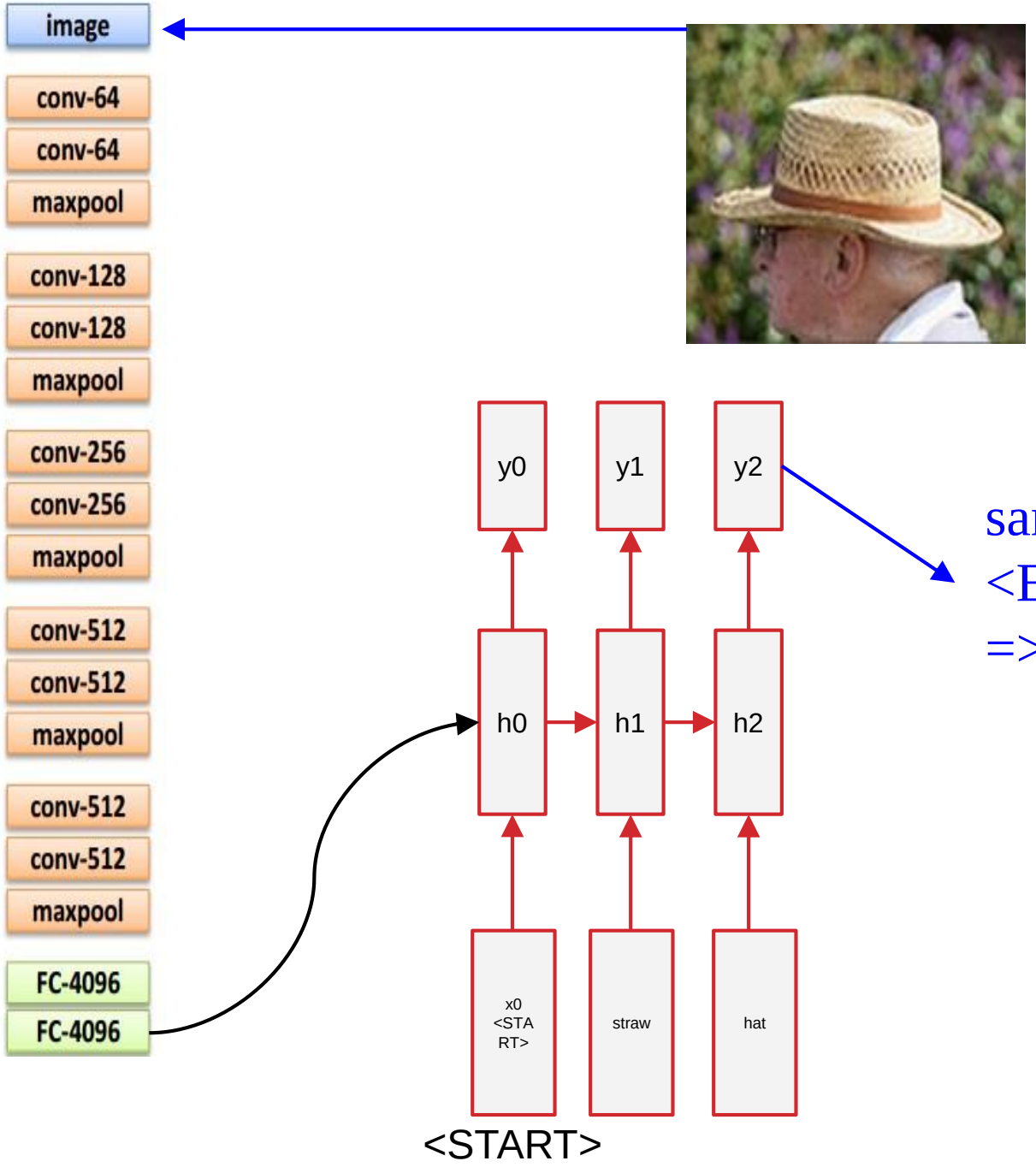


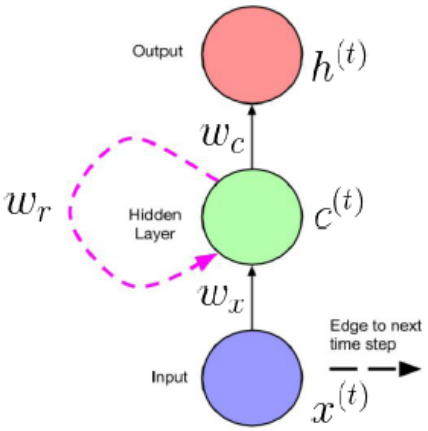
image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
y2
h0
h1
h2
x0 <START>
straw
hat
<START>
Training Image
sample <END> token => finish.

Problems with RNN Model

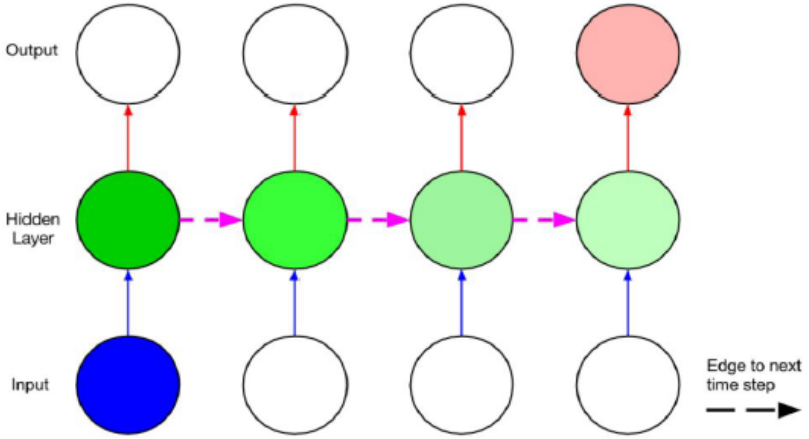
- When dealing with a time series, it tends to **forget old information**.
  - In practice, the range of contextual information that standard RNNs can access are limited to approximately 10 time steps between the relevant input and target events.

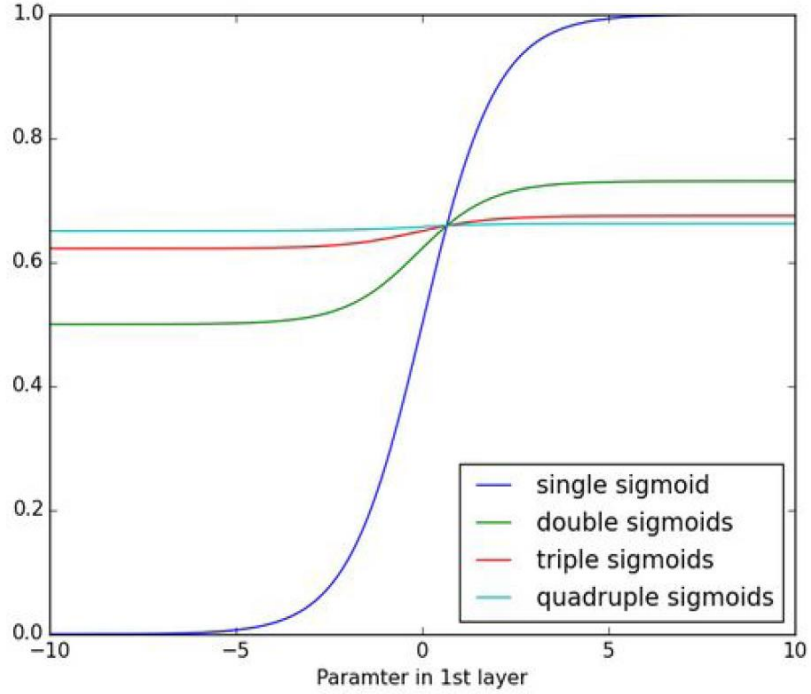
- **Vanishing gradient problem**.
  - The influence of a given input on the hidden layer, and therefore on the network output, either decays or grows exponentially as it propagates through an RNN.

Problems with RNN Model



UNFOLD



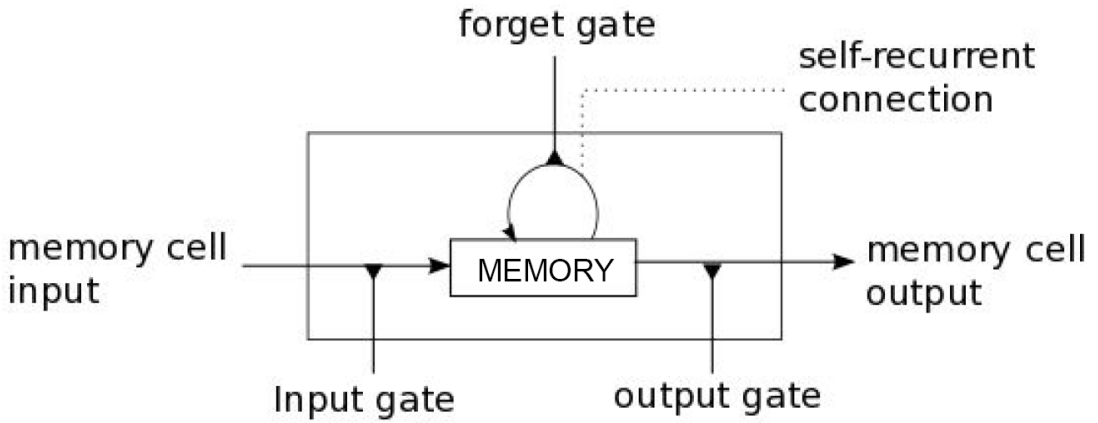


$$h^{(t)} = \sigma(w_c \cdot c^{(t)})$$
$$c^{(t)} = \sigma(w_r \cdot c^{(t-1)} + w_x \cdot x^{(t)})$$

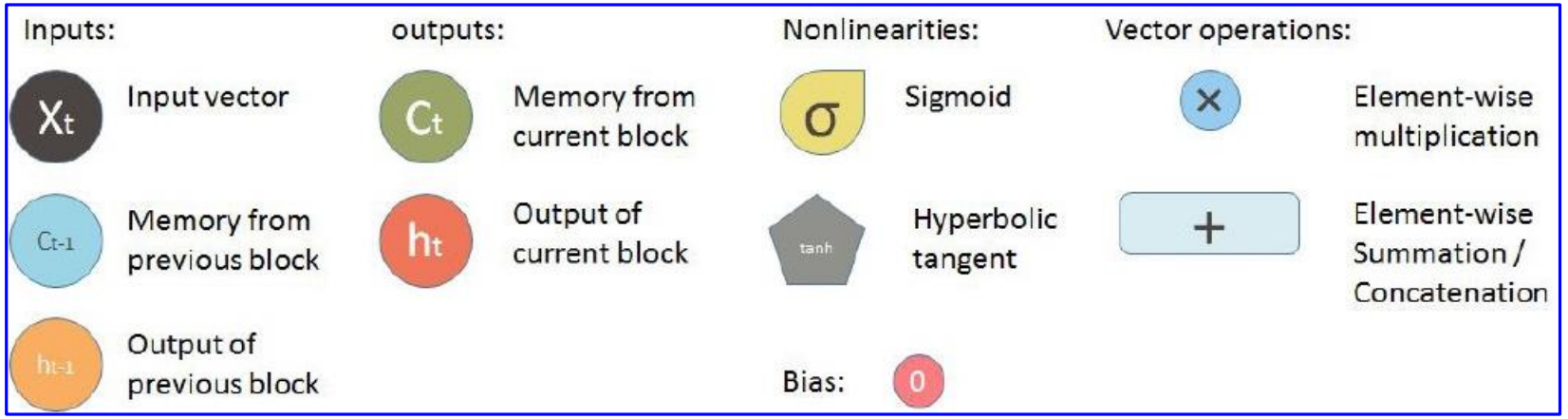
$$\begin{aligned} h^{(3)} &= \sigma(w_c \cdot c^{(3)}) \\ &= \sigma(w_c \cdot \sigma(w_x \cdot x^{(3)} + w_r \cdot c^{(2)})) \\ &= \sigma(w_c \cdot \sigma(w_x \cdot x^{(3)} + w_r \cdot \sigma(w_x \cdot x^{(2)} + w_r \cdot c^{(1)})))) \\ &= \sigma(w_c \cdot \sigma(w_x \cdot x^{(3)} + w_r \cdot \sigma(w_x \cdot x^{(2)} + w_r \cdot \sigma(w_x \cdot x^{(1)} + w_r \cdot c^{(0)}))))) \end{aligned}$$

Solution : Long Short-Term Memory

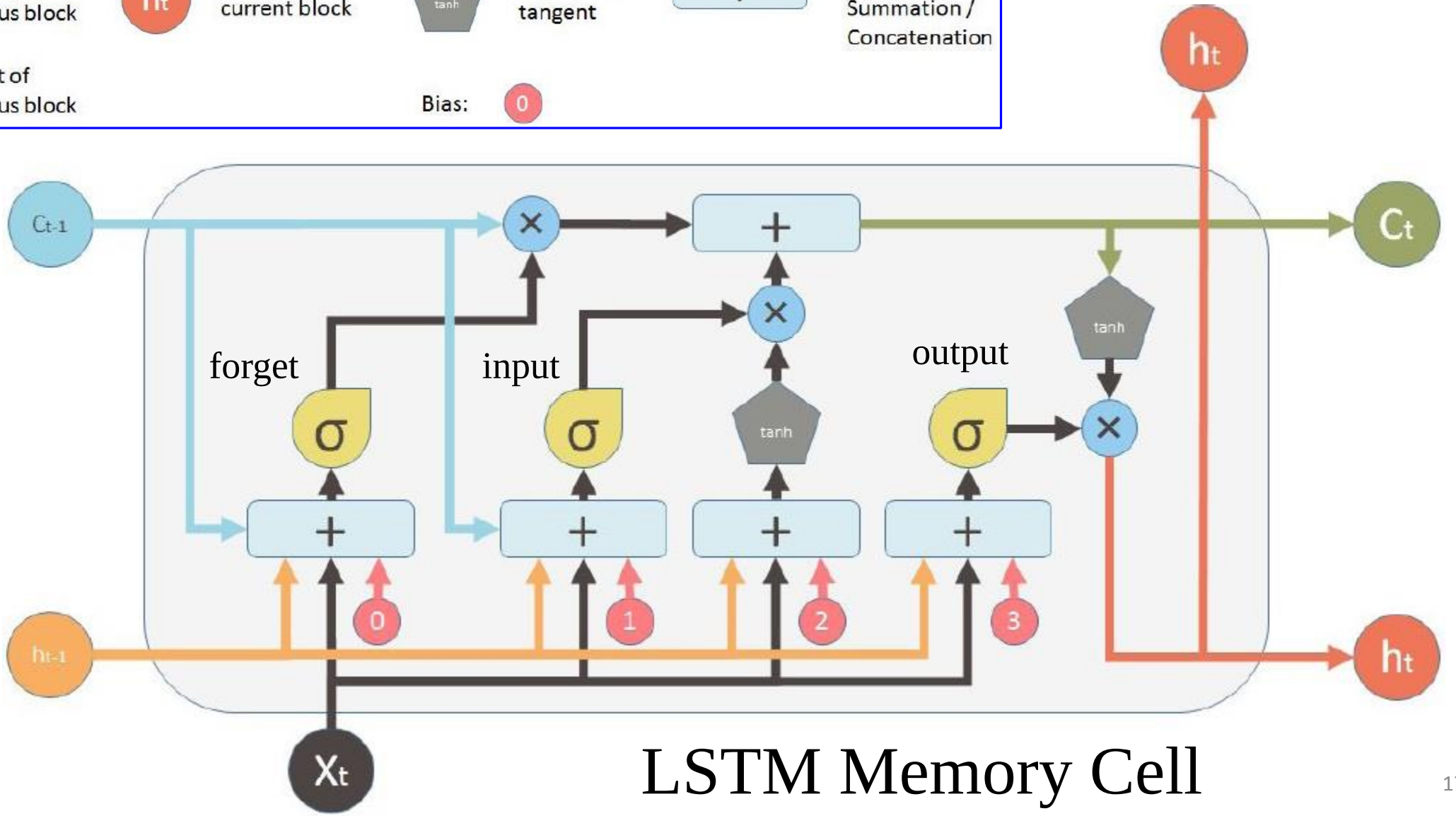
- When there is a distant relationship of unknown length, we wish to have a “memory” to it.
- **Idea**: Design a memory cell which can maintain its state over time, consisting of an explicit memory (i.e the cell state vector) and gating units which regulate the information flow into and out of the memory.



LSTM Memory Cell



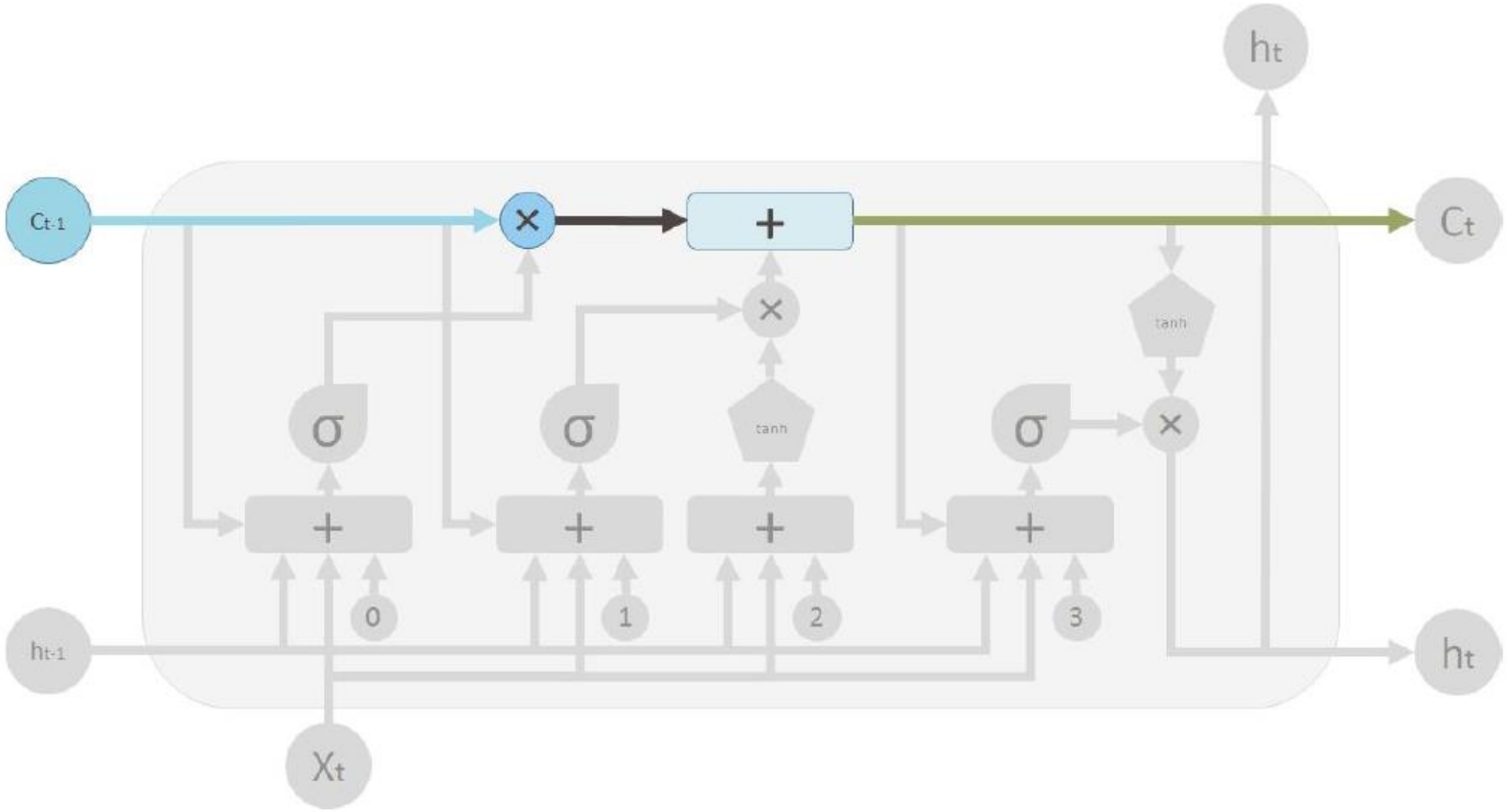
- **Three components:**
  forget gate, input gate and output gate.



LSTM Memory Cell

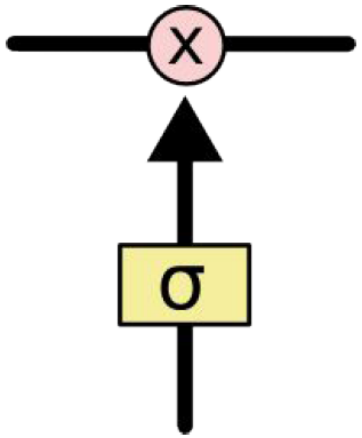
Memory - Cell State Vector

- New concept to RNN model, representing the memory of the LSTM.
- Undergoes changes via forgetting of old memory (forget gate) and addition of new memory (input gate) Cell



Gates

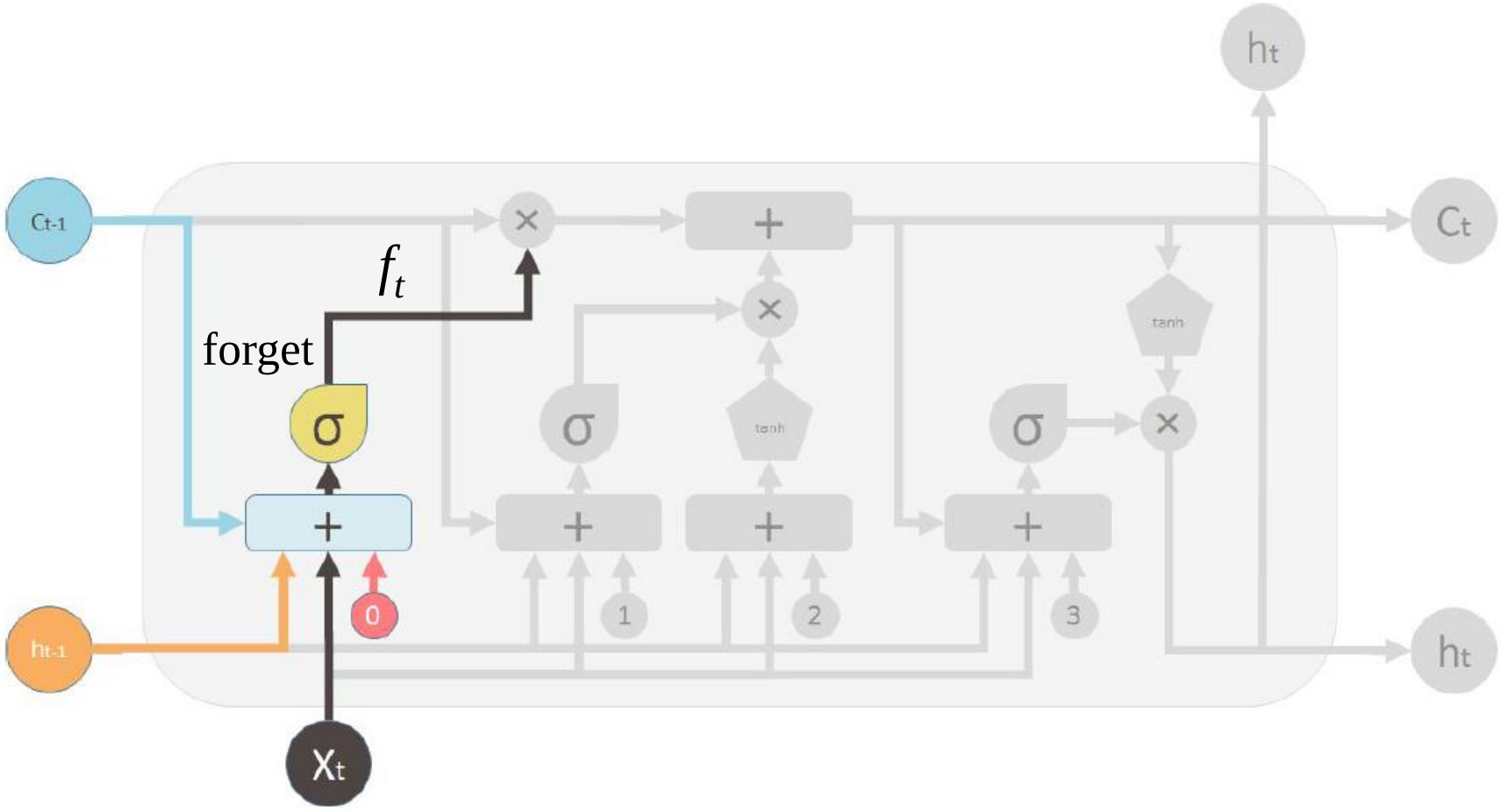
- Gate: sigmoid neural net layer followed by pointwise multiplication operator.
  - Recall sigmoid outputs values from 0 to 1.
  - Values are discarded if 0 is used for pointwise multiplication .
- Gates control the flow of information to/from the memory
- Gates are controlled by a concatenation of the output from the previous time step and the current input and optionally the cell state vector.



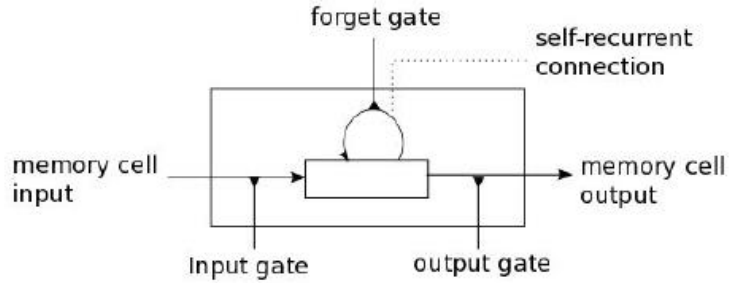
Gate (sigmoid layer followed by pointwise multiplication)

Forget Gate

- Controls what information to throw away from memory.

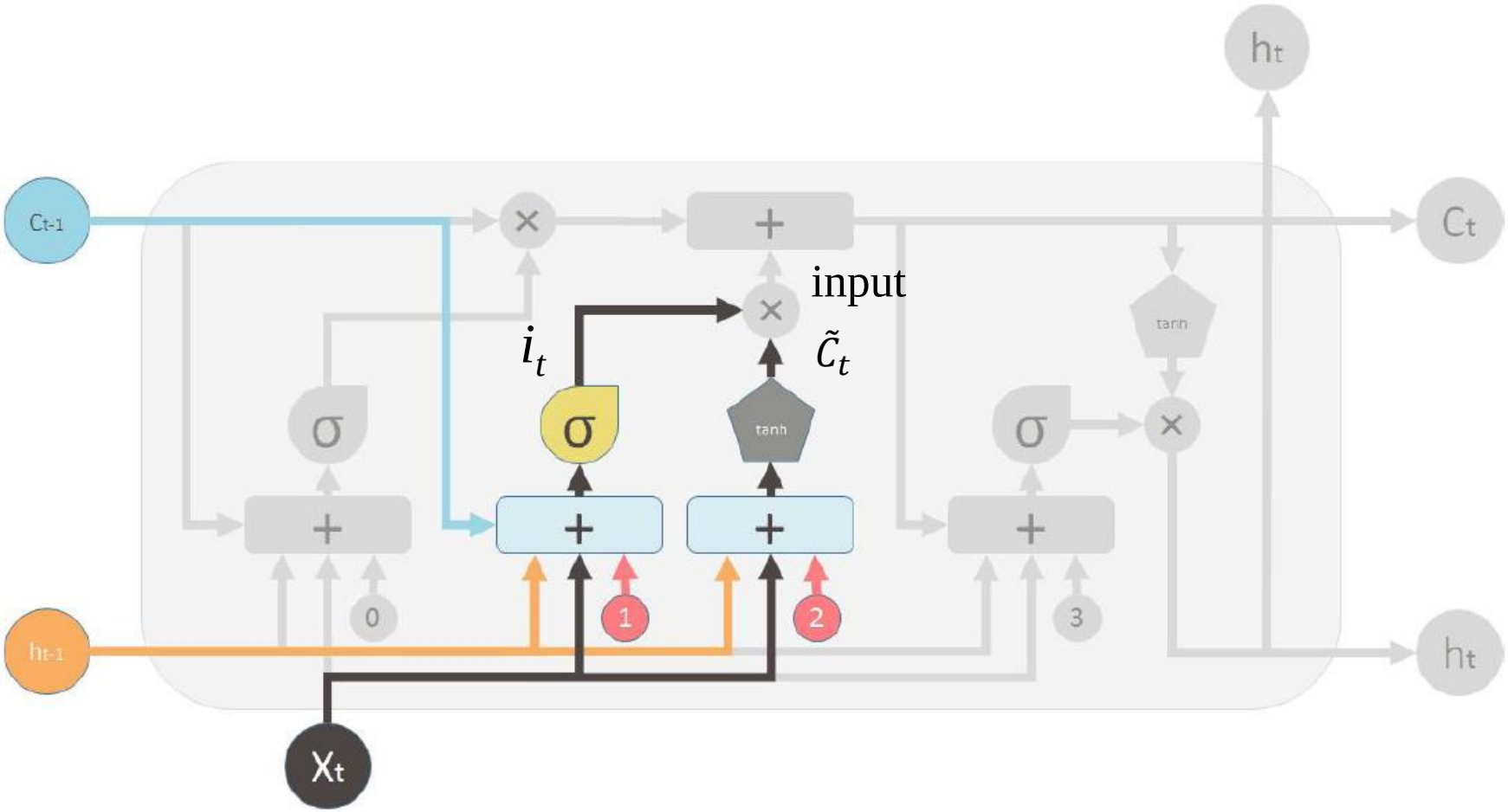


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



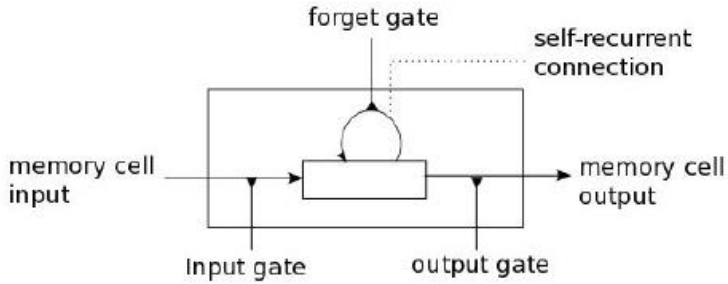
Input Gate

- Controls what new information is added to cell state from current input.



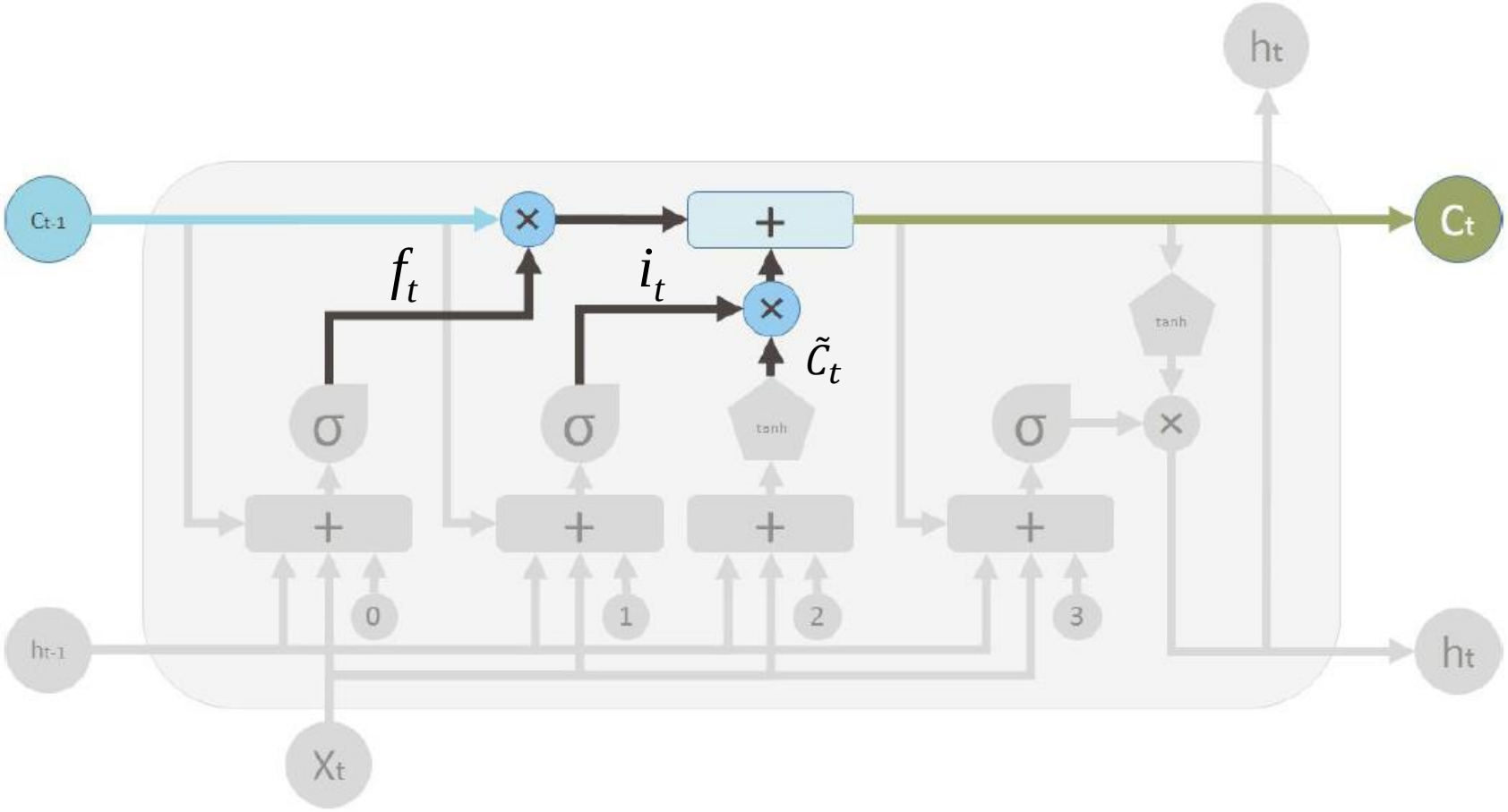
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

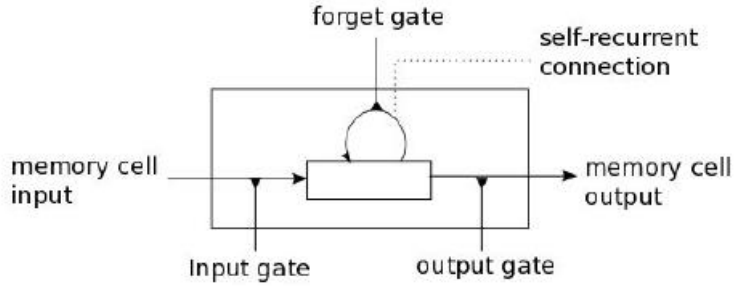


Memory Update

- The cell state vector aggregates the two components (old memory via the forget gate and new memory via the input gate).

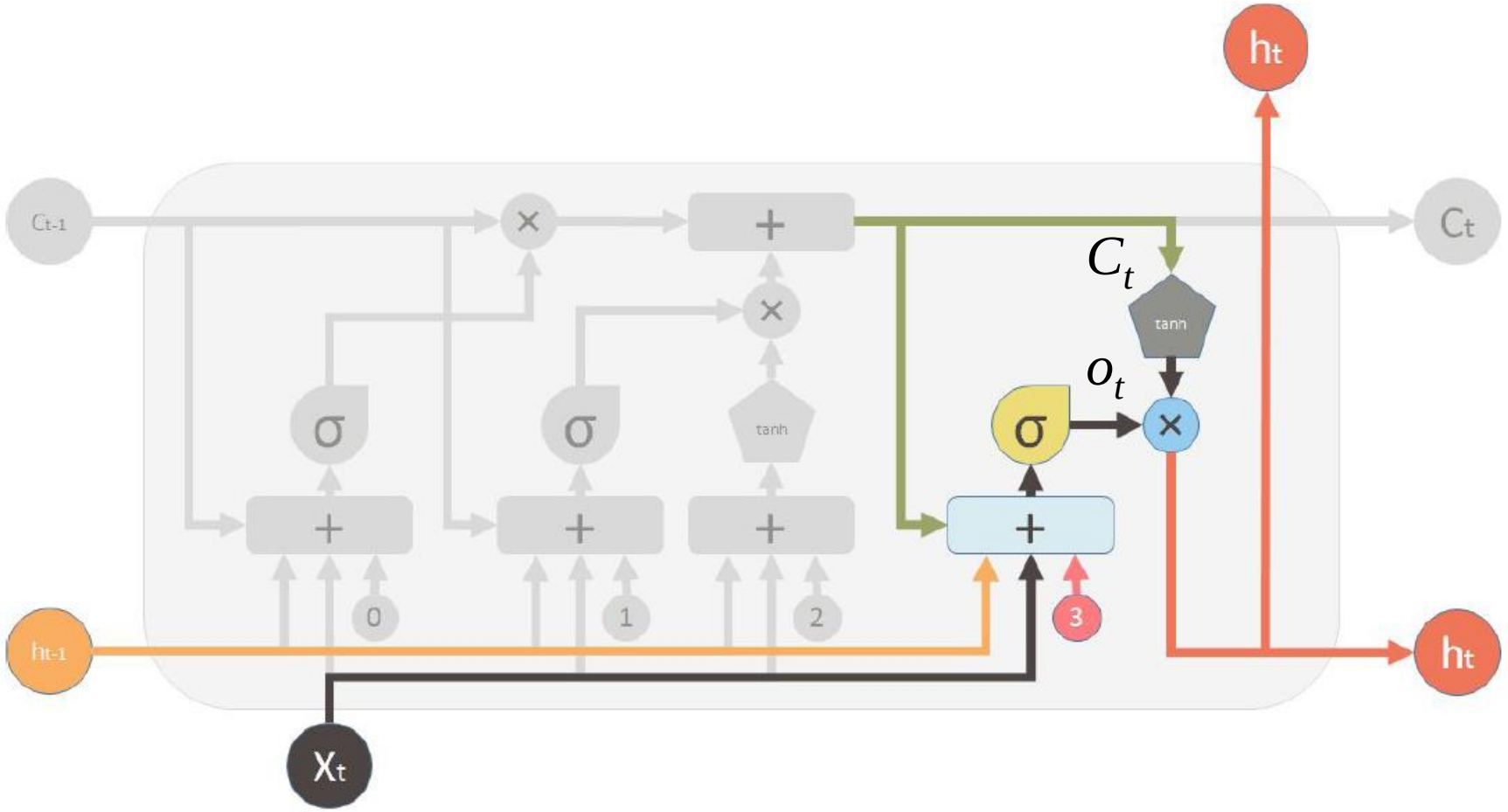


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



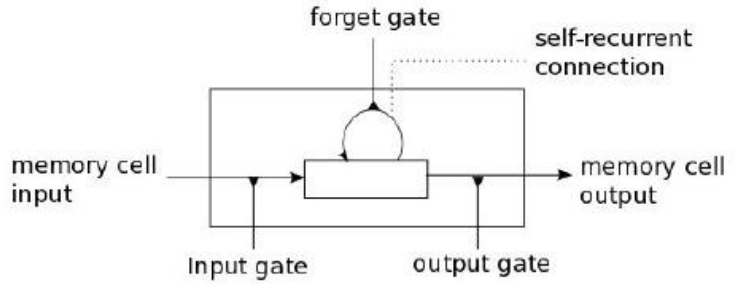
Output Gate

- Conditionally decides what to output from the memory.

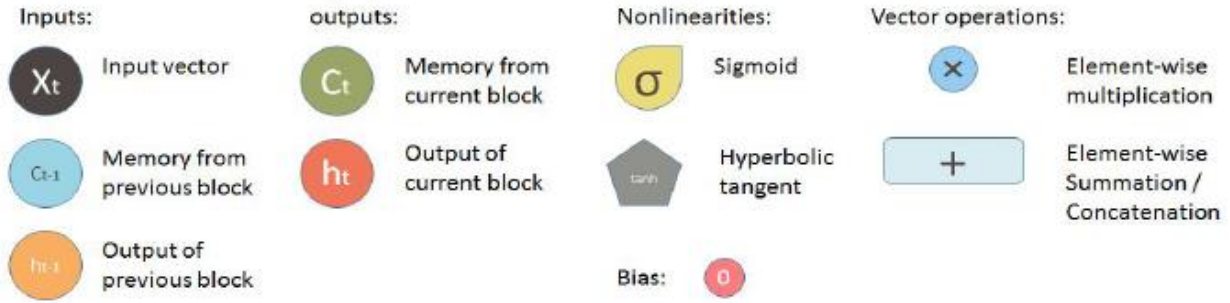


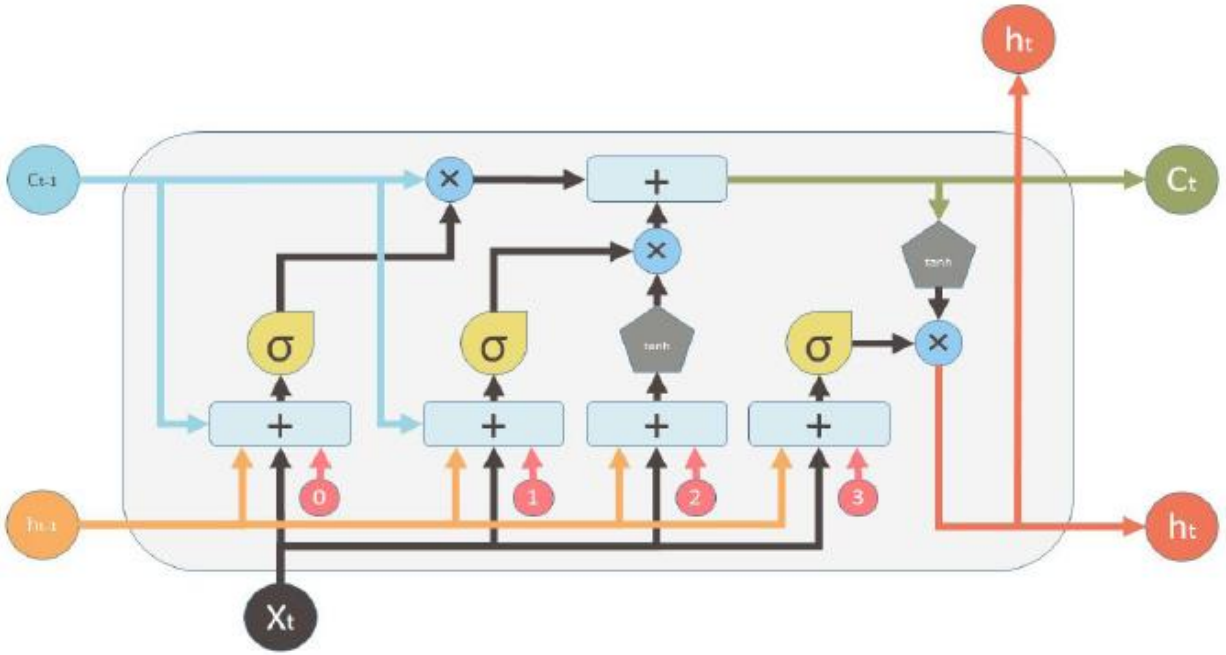
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



LSTM Memory Cell Summary





$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

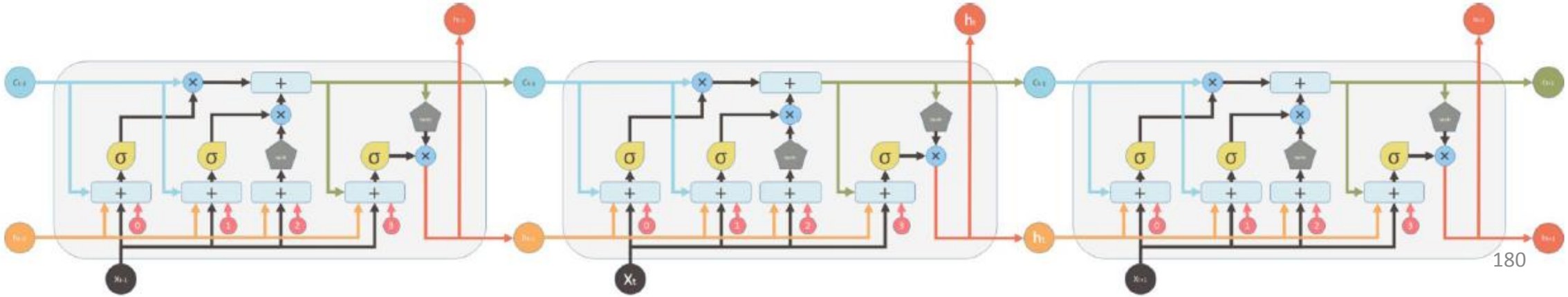
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

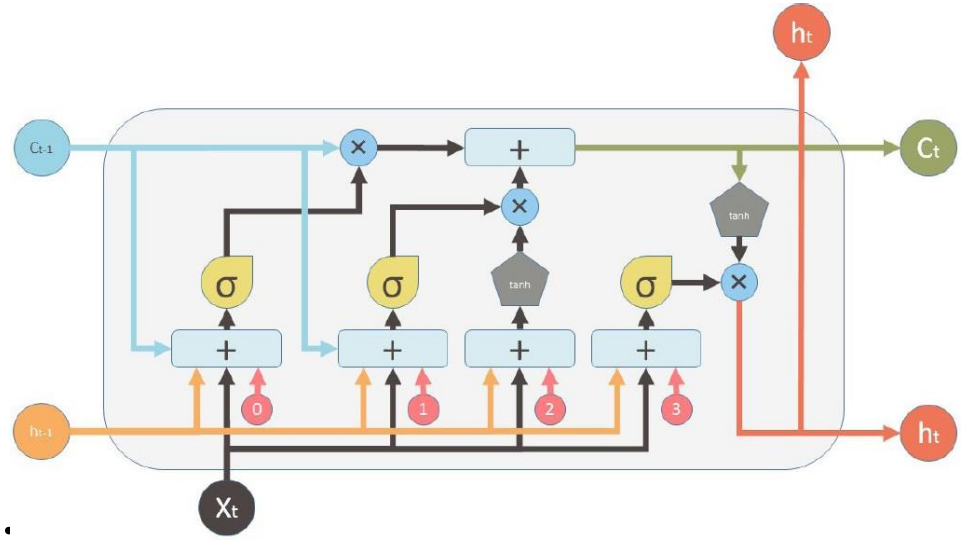
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



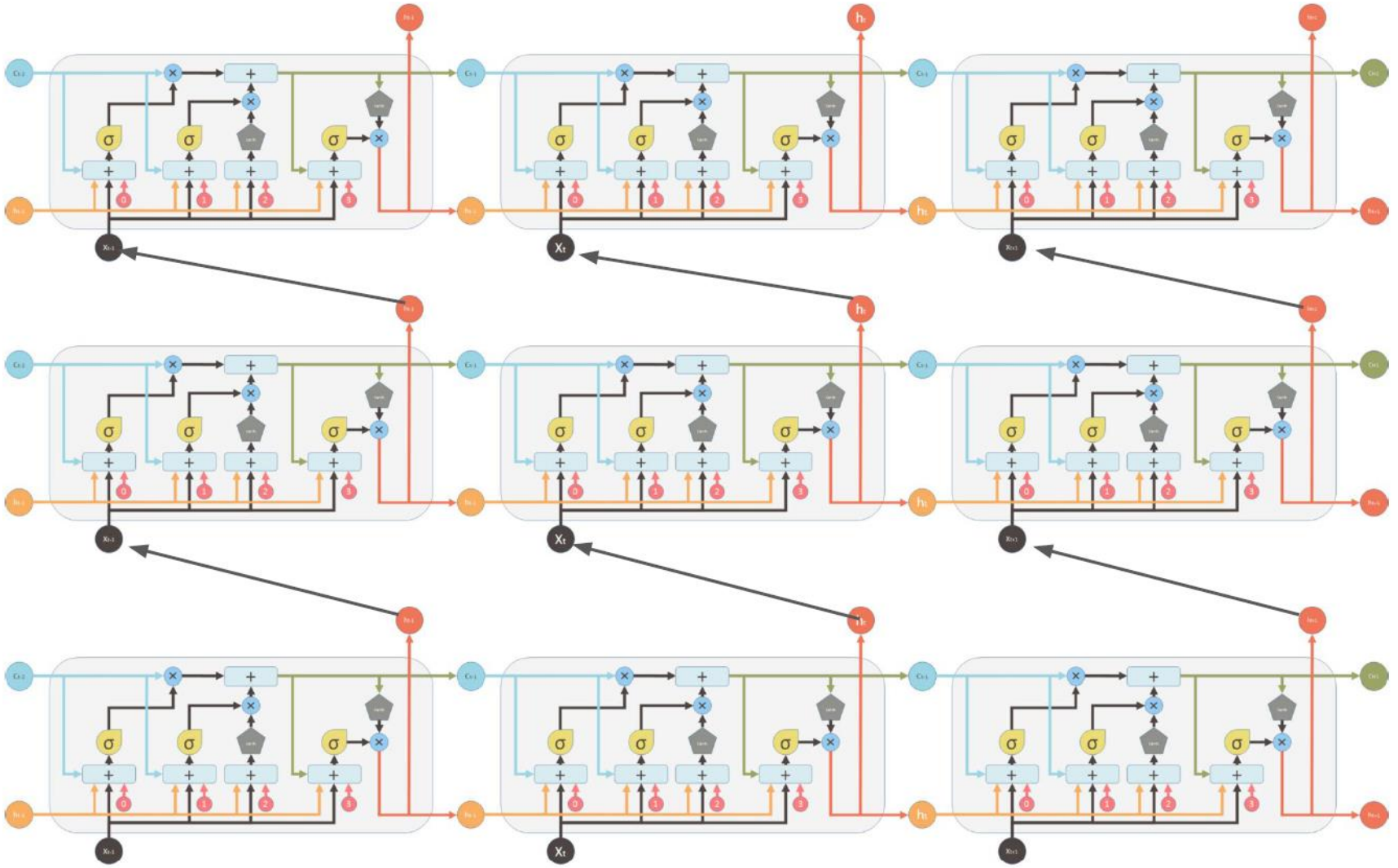
LSTM Training

- Backpropagation Through Time (BPTT) is most common.
- What weights are learned?
  - Gates (input/output/forget)
  - Input tanh layer
- Outputs depend on the task:
  - Single output prediction for the whole sequence.
  - One output at each time step (sequence labeling).

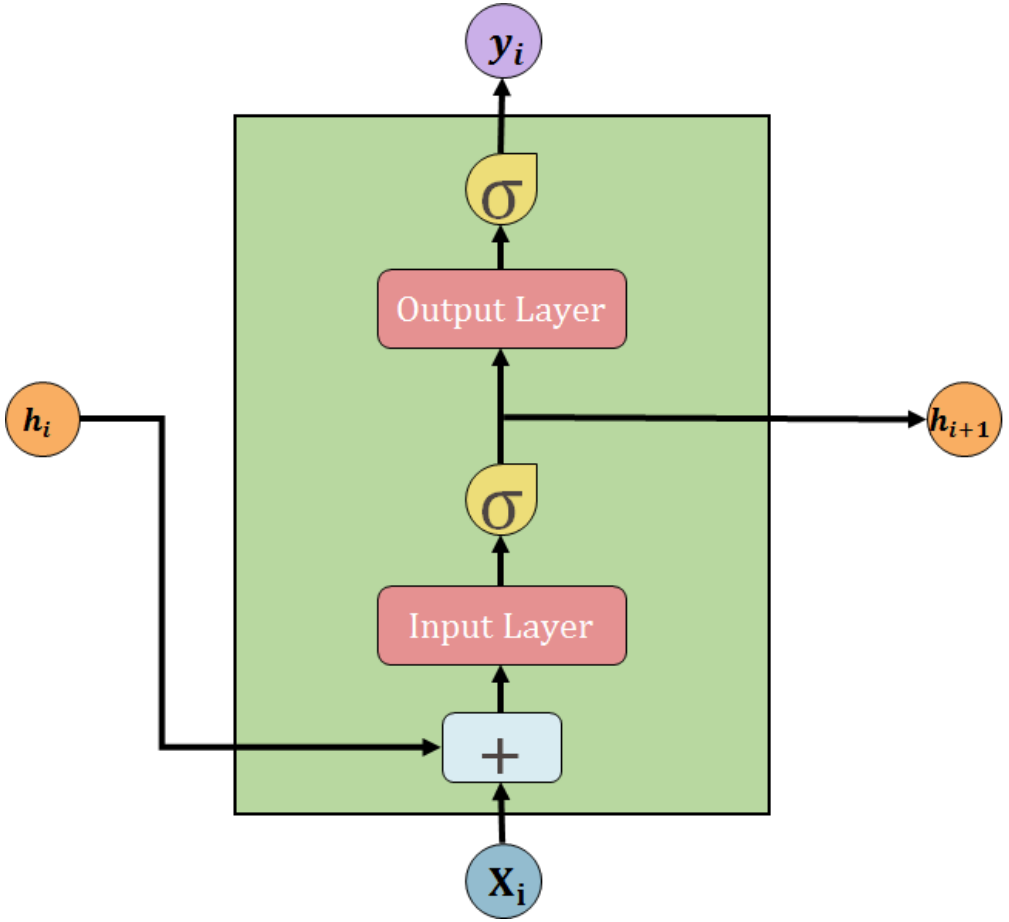


Deep LSTMs

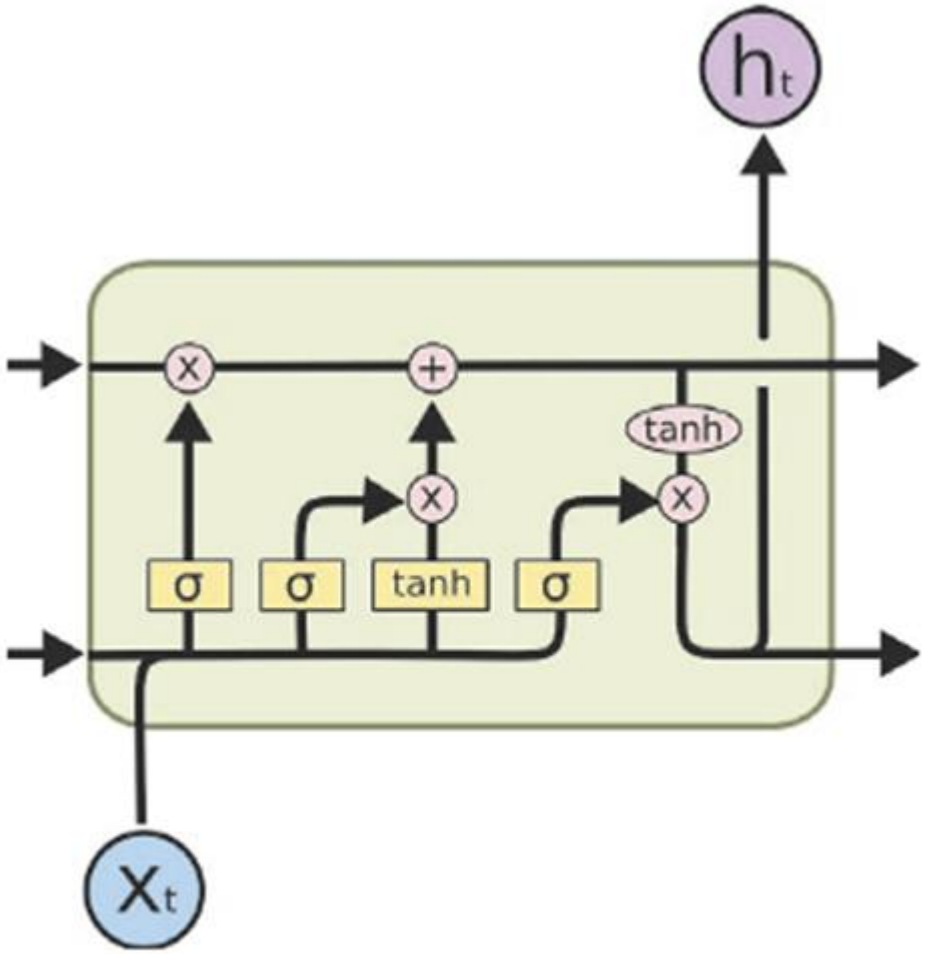
- Deep LSTMs can be created by stacking multiple LSTM layers vertically, with the output sequence of one layer forming the input sequence of the next.
- Increases the number of parameters - but given sufficient data, performs significantly better than single-layer LSTMs.



RNN vs LSTM



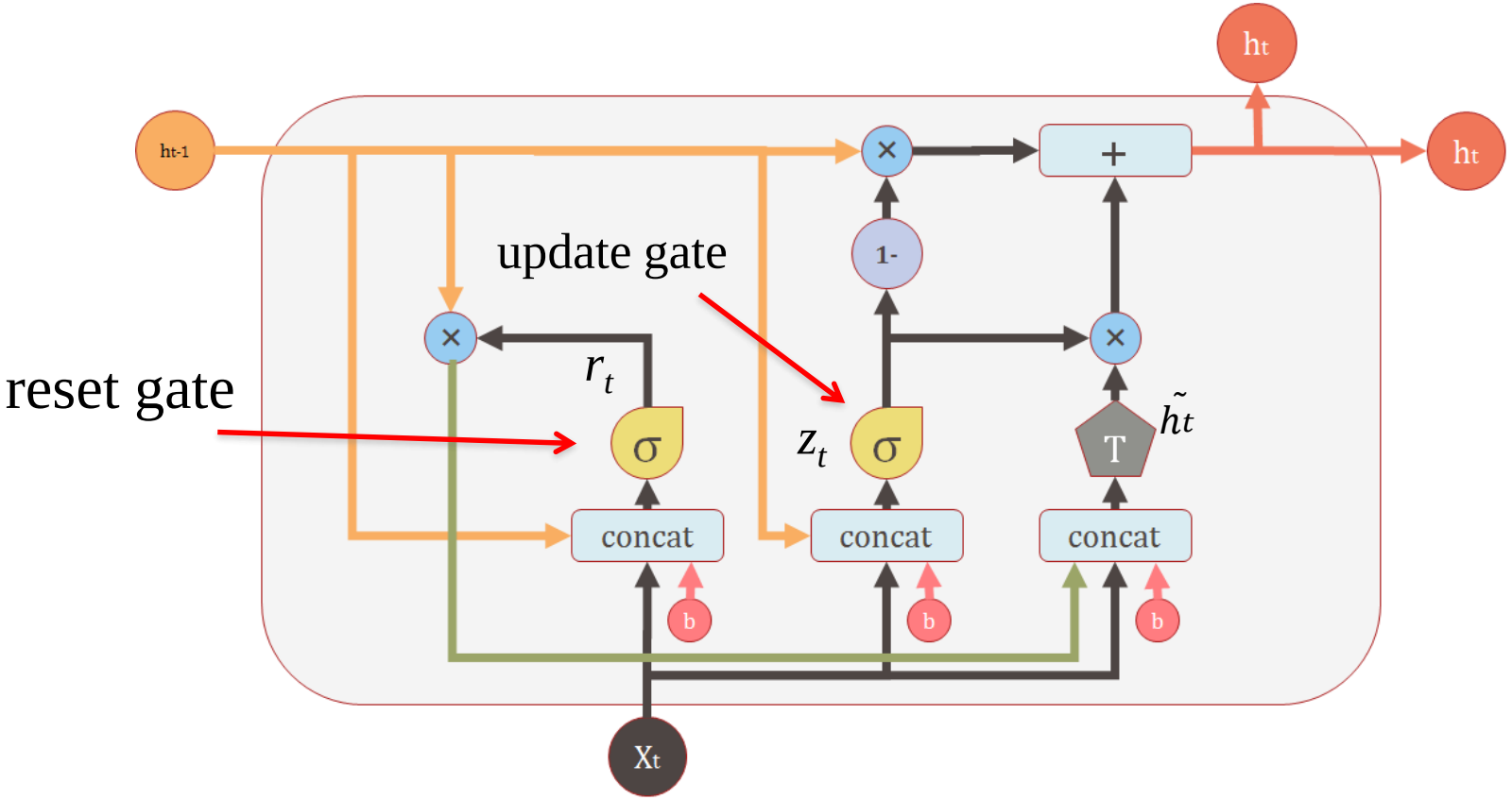
(a) RNN



(b) LSTM

GRU – Gated Recurrent Unit

- It also merges the cell state and hidden state.
- It combines the forget and input into a single update gate.
- Simpler and more compressed than LSTM.



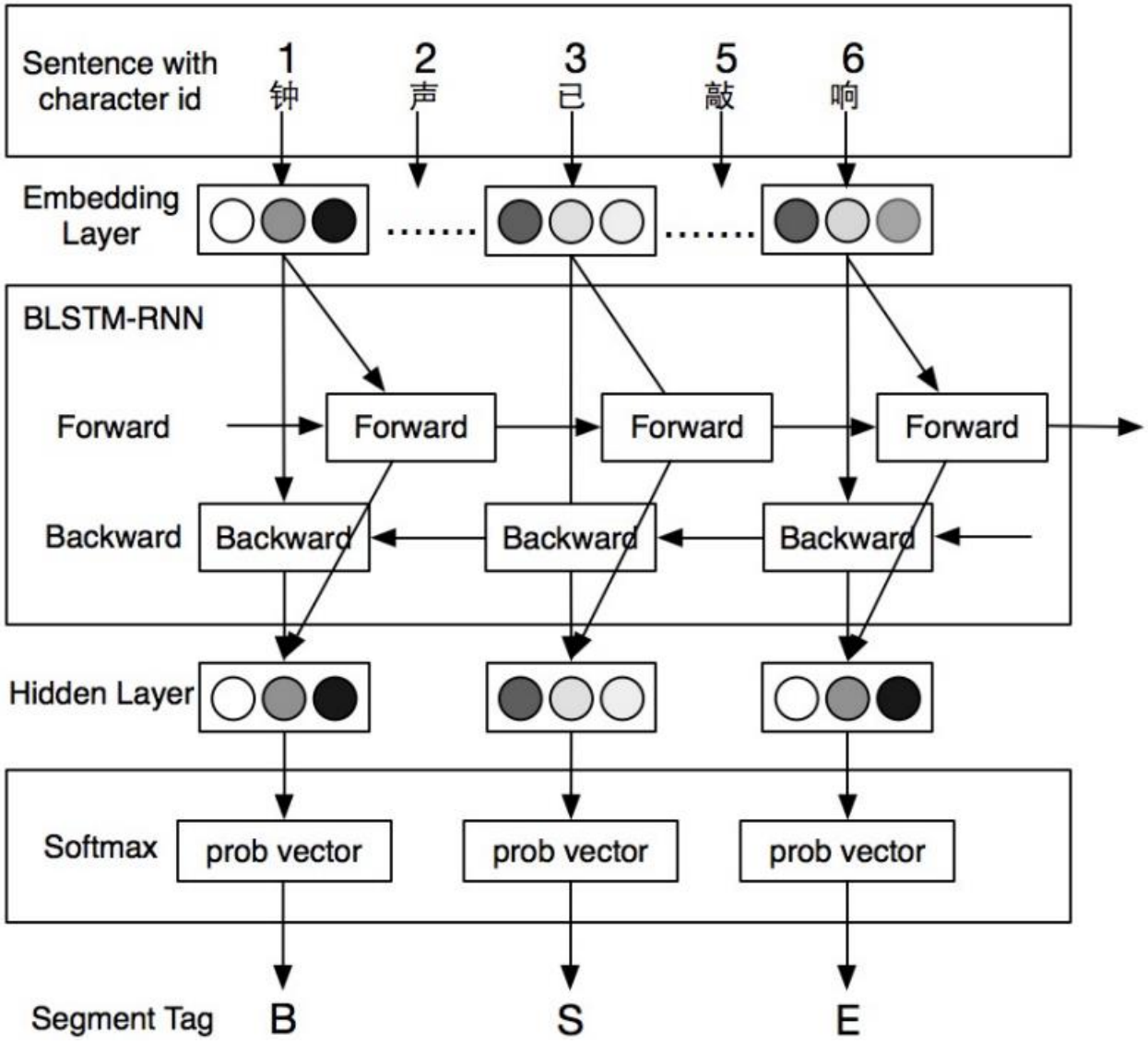
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

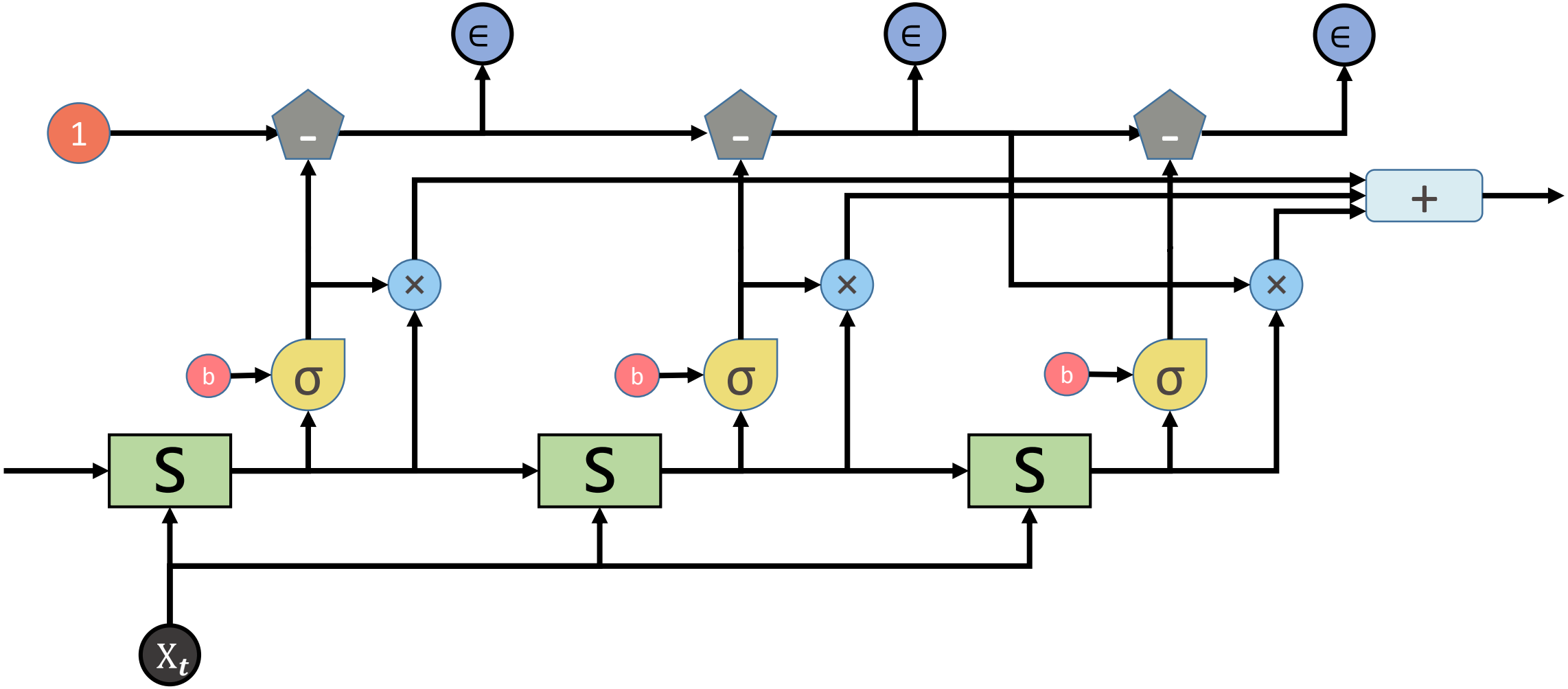
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Bidirectional LSTM (Bi-LSTM)



Google DeepMind, A Graves – Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition, 2005.

Adaptive computation Time RNN (ACT RNN)



Google DeepMind, A Graves – Adaptive Computation Time for Recurrent Neural Networks, 2016.

針對不同應用問題，深度學習模型的選擇…

- Artificial Neural Network (ANN)
  - 數據資料預測分析
  - 例如: 以氣溫、氣壓、濕度，預測明天降雨機率
- Convolutional Neural Network (CNN)
  - 影像/數據資料的分類或分析
  - 文字、音樂、語音、數據資料，皆可套用一維捲積運算
- Recurrent Neural Network/Long Short-Term Memory (RNN/LSTM)
  - 具有時序性的數據資料或影像

Generative Adversarial Network

生成對抗網路模型

Generative Adversarial Network

Generative Models

Training samples

Model samples

Training samples

Generative Adversarial Network

- First introduced by Ian Goodfellowet al. in 2014
- GANs have been used to generate images, videos, poems, and conversation .

Generator:

- Generates candidates/images (from a probability distribution)
- It's objective is to 'fool' the discriminator by producing novel synthesized instances that appear to come from the true data

Discriminator:

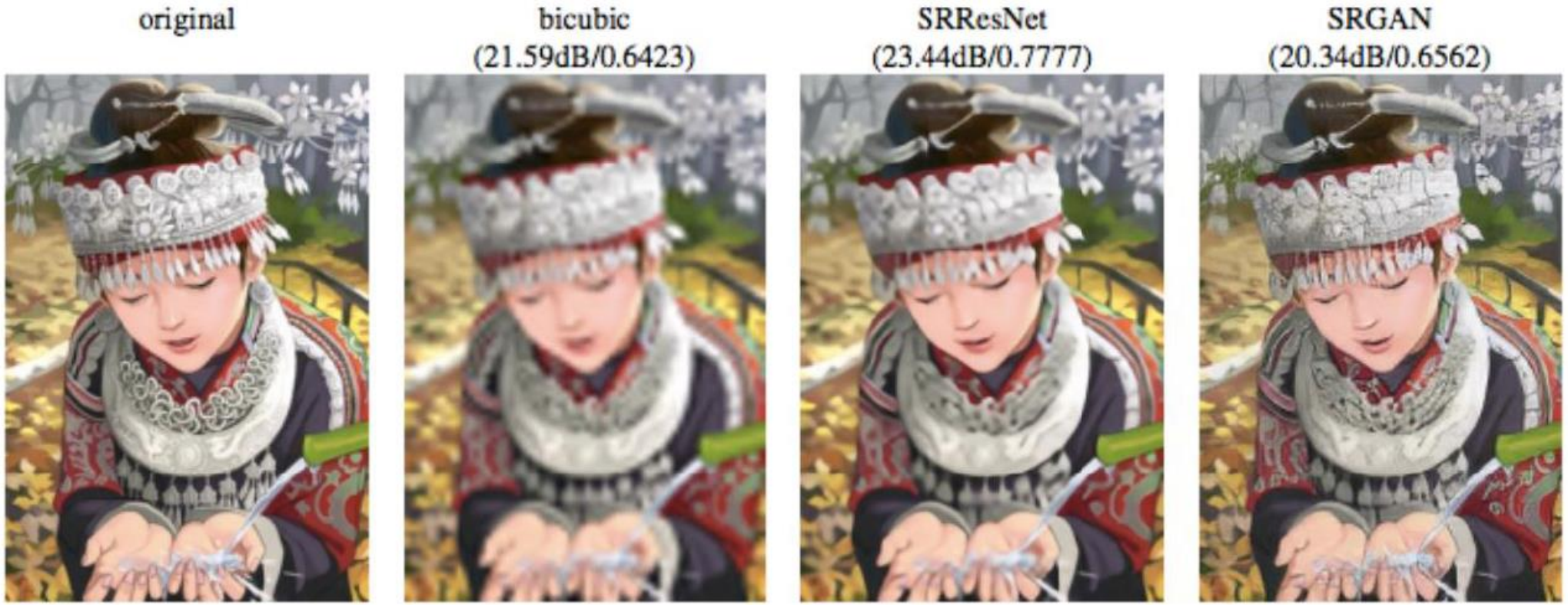
- Evaluates the generated images to see if they come from the true data or not

Backpropagation applied to both networks:

- Generator to produce better images
- Discriminator to be more skilled at evaluating generated images

Applications of GAN

- **Image generation tasks**
  - Example: single-image super-resolution



Text-to-Image Translation

this small bird has a pink breast and crown, and black primaries and secondaries.

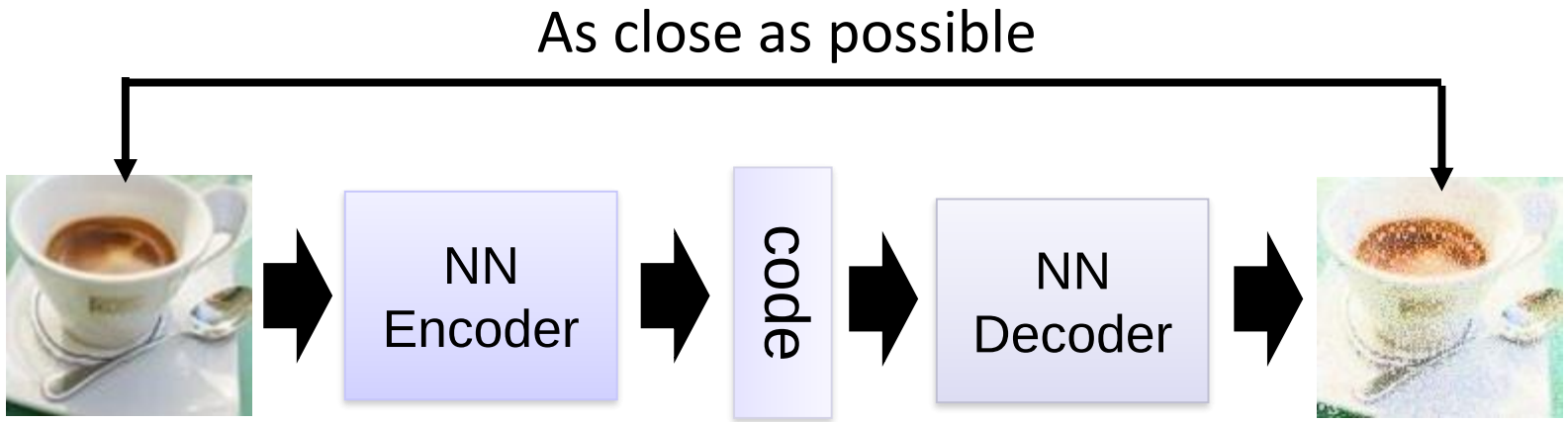
this magnificent fellow is almost all black with a red crest, and white cheek patch.

the flower has petals that are bright pinkish purple with white stigma

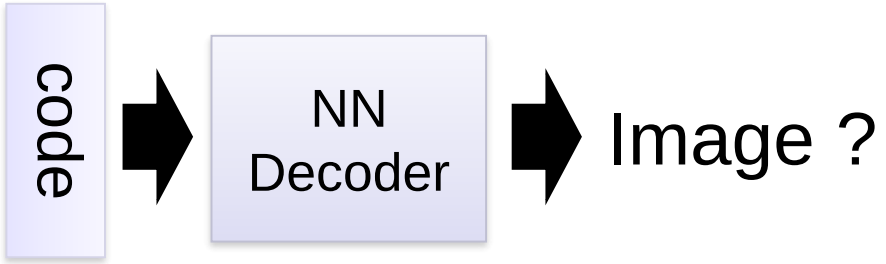
this white and yellow flower have thin white petals and a round yellow stamen

Image-to-Image Translation

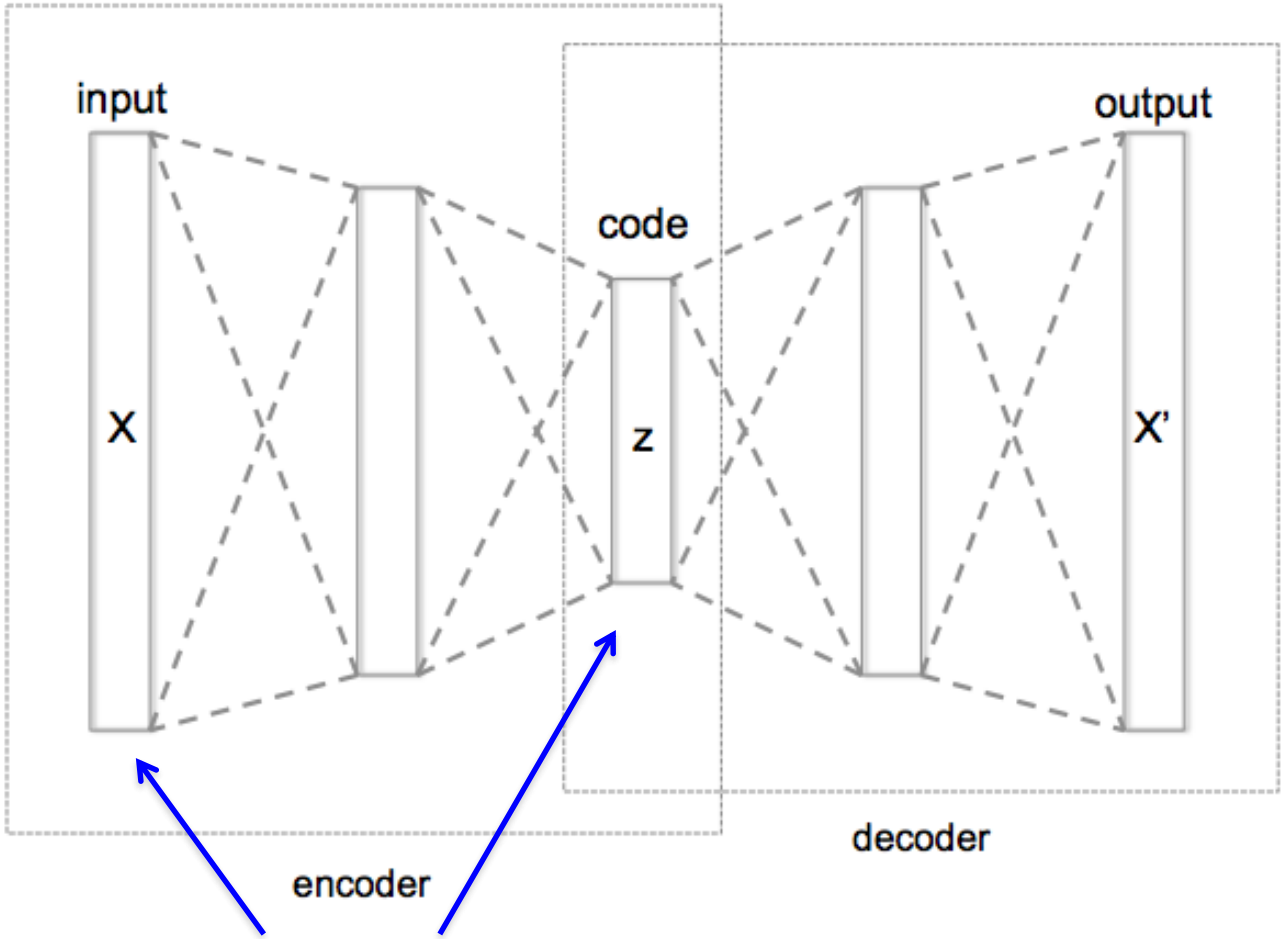
Motivation - Autoencoder



Randomly generate a vector as code



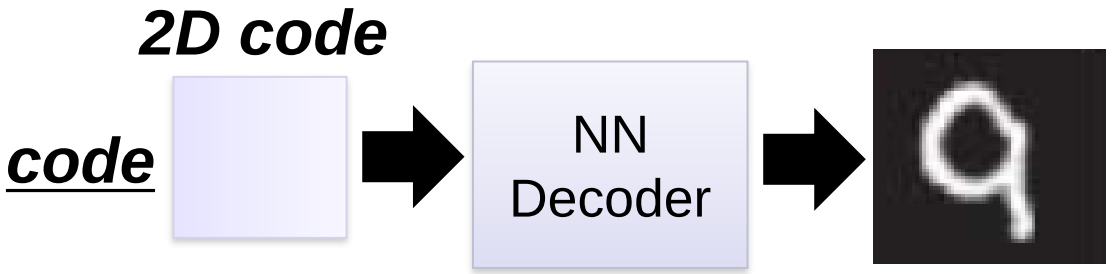
Autoencoder with 3 fully connected layers

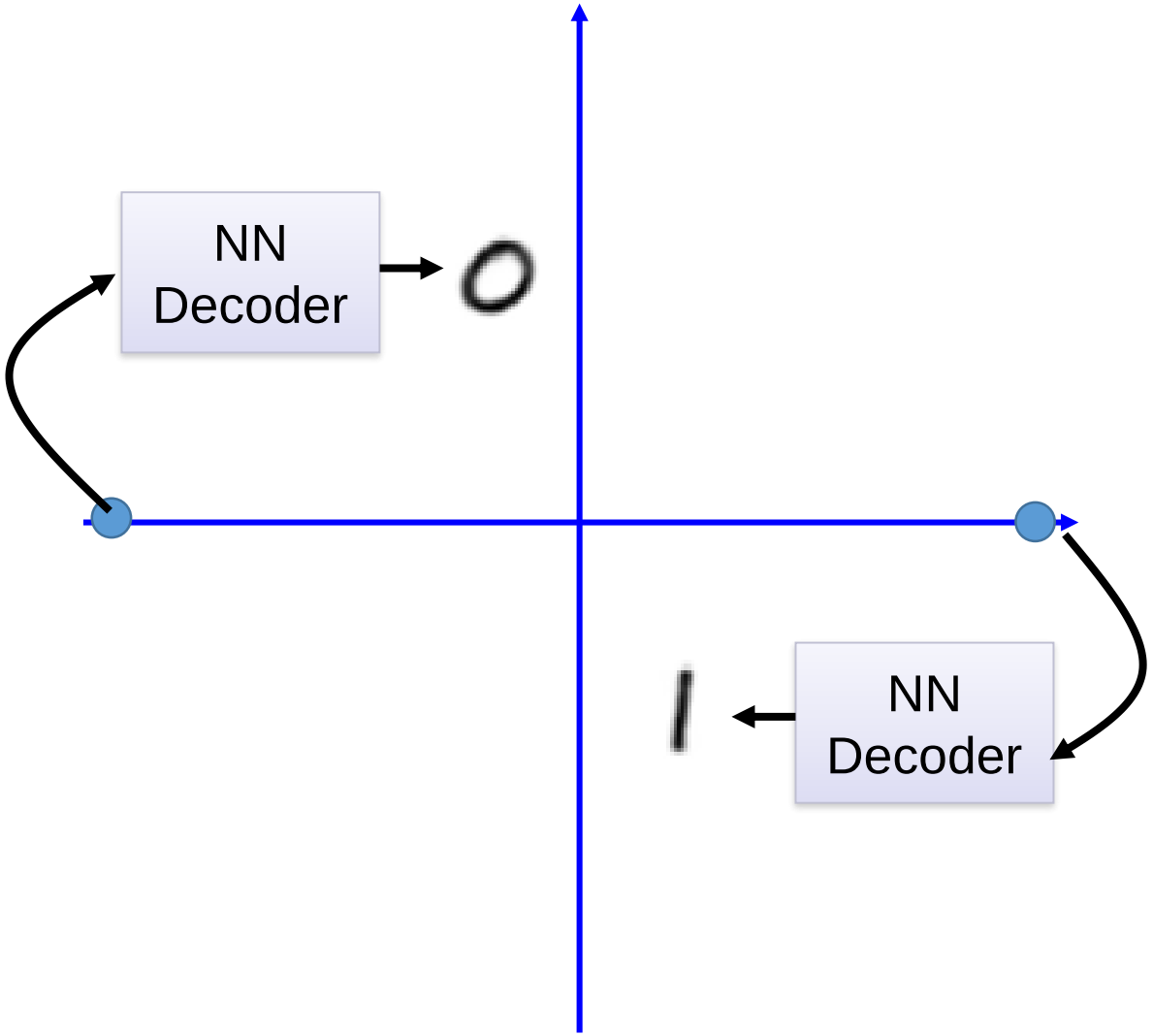


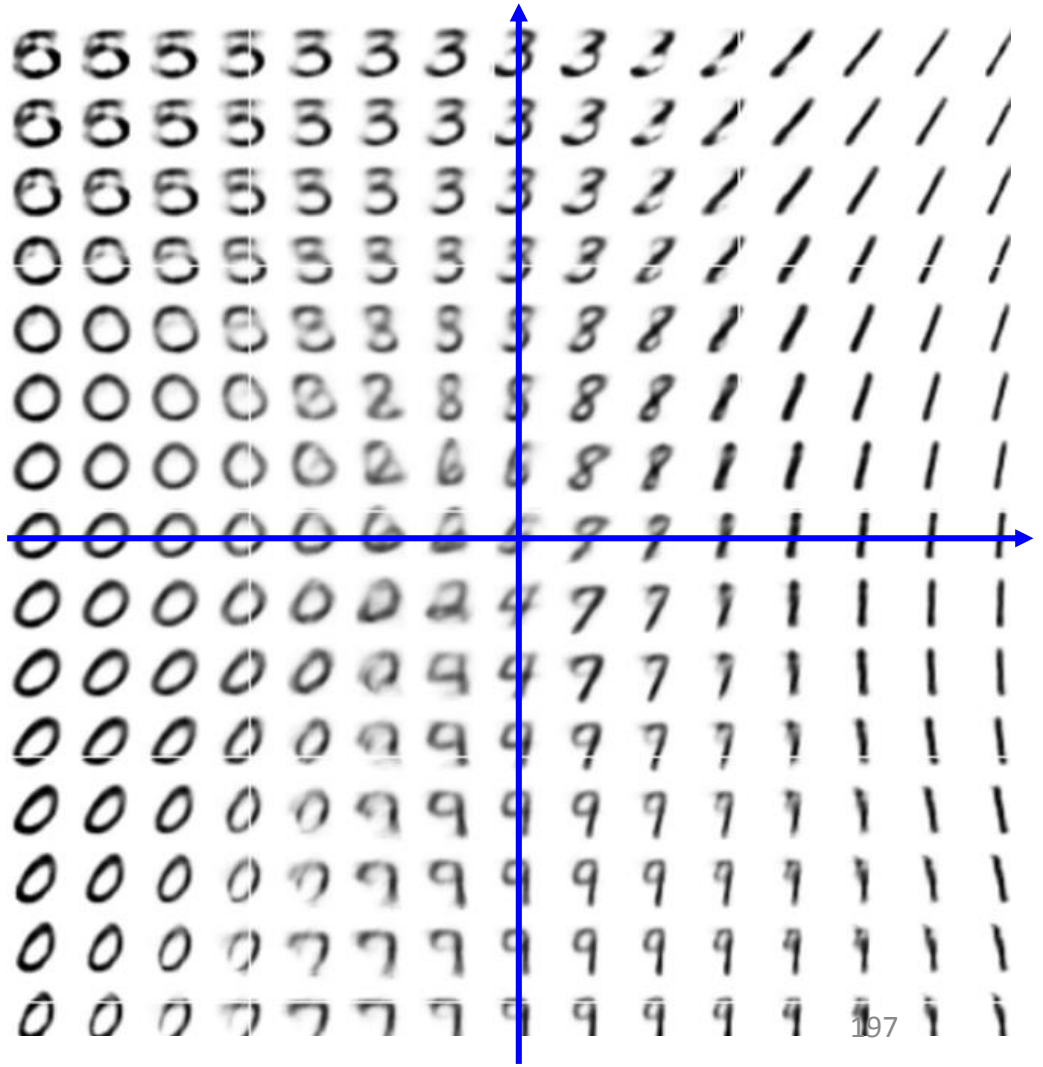
Training: model.fit(X,X')
Cost function: $\Sigma_{k=1..N} (x_k - x'_k)^2$

Large → small, learn to compress

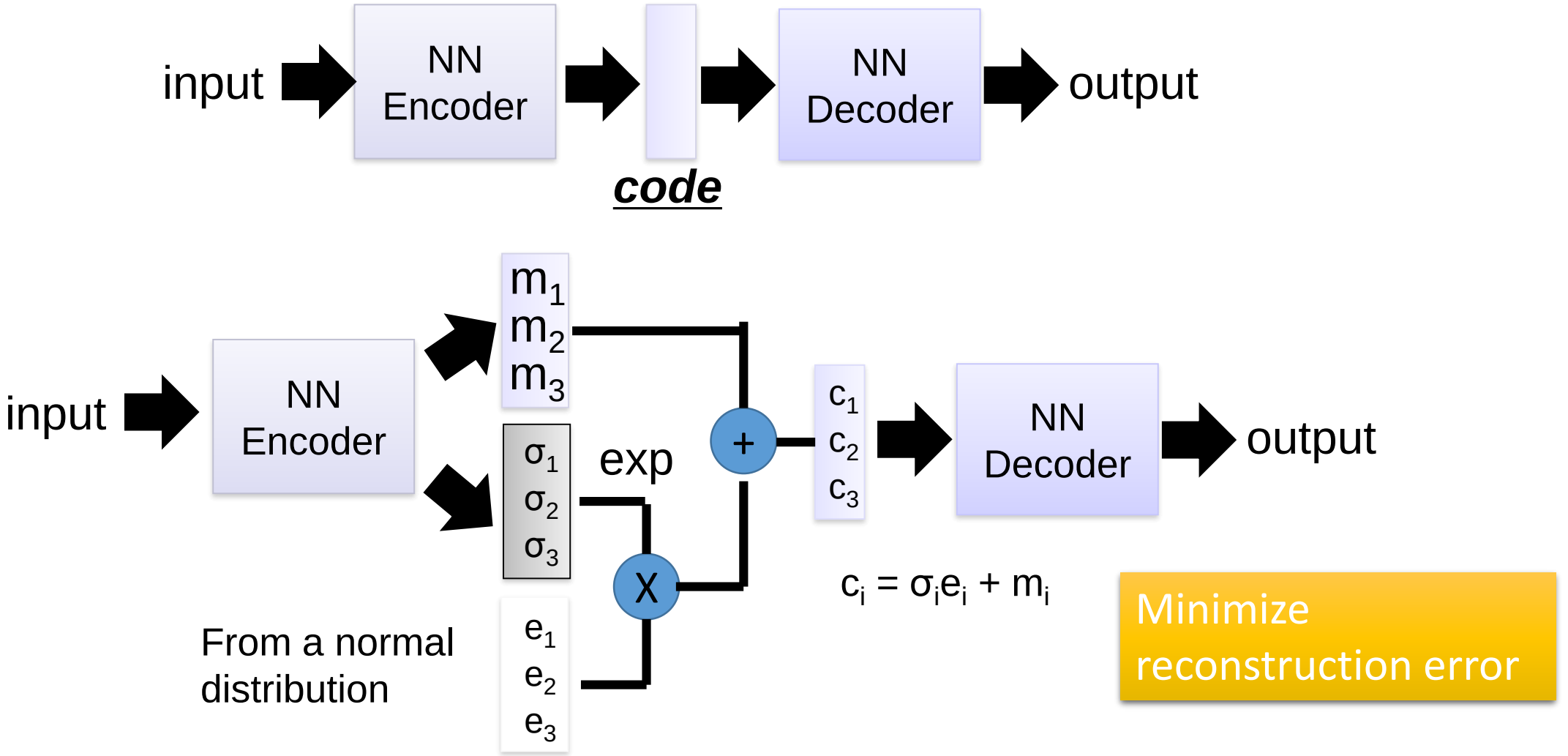
Auto-encoder







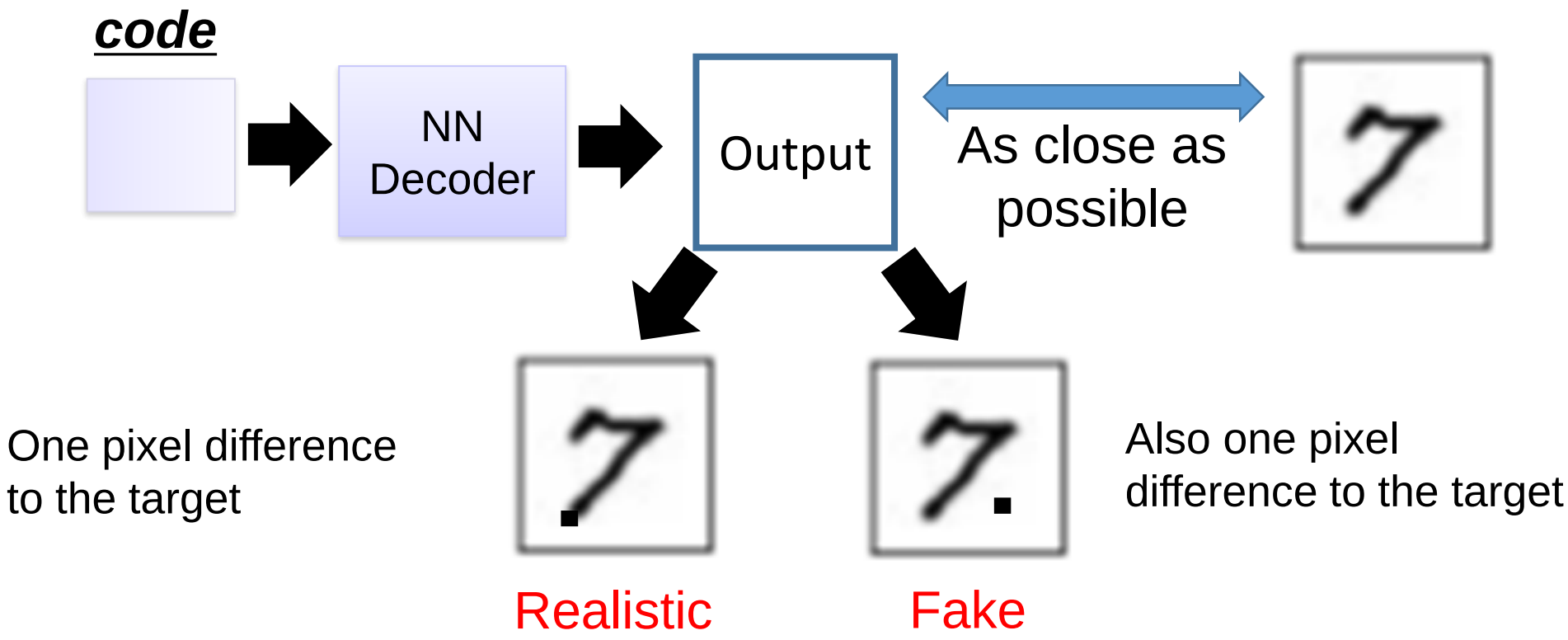
Variational Autoencoder



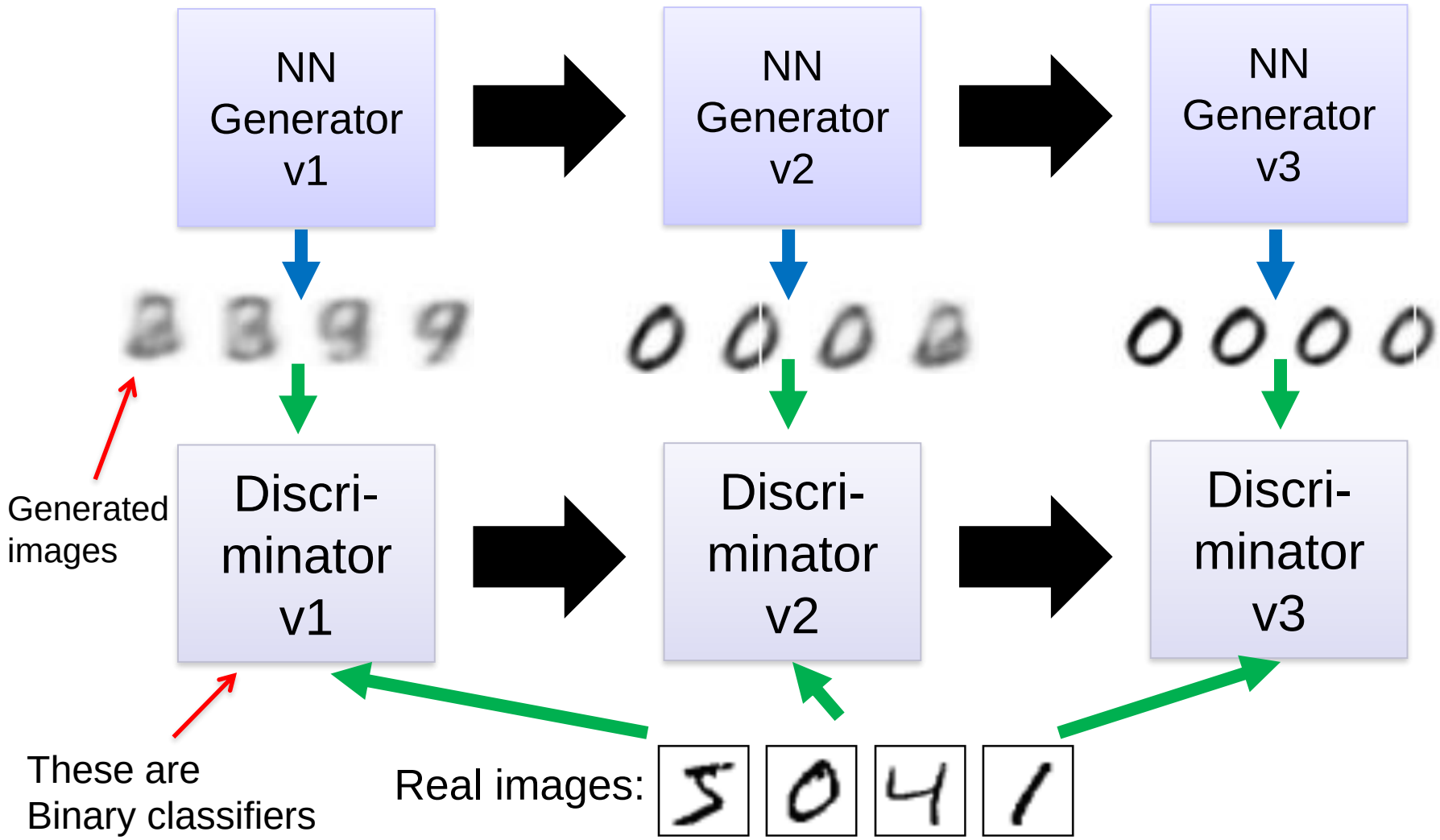
Auto-Encoding Variational Bayes, https://arxiv.org/abs/1312.6114

Problems of VAE

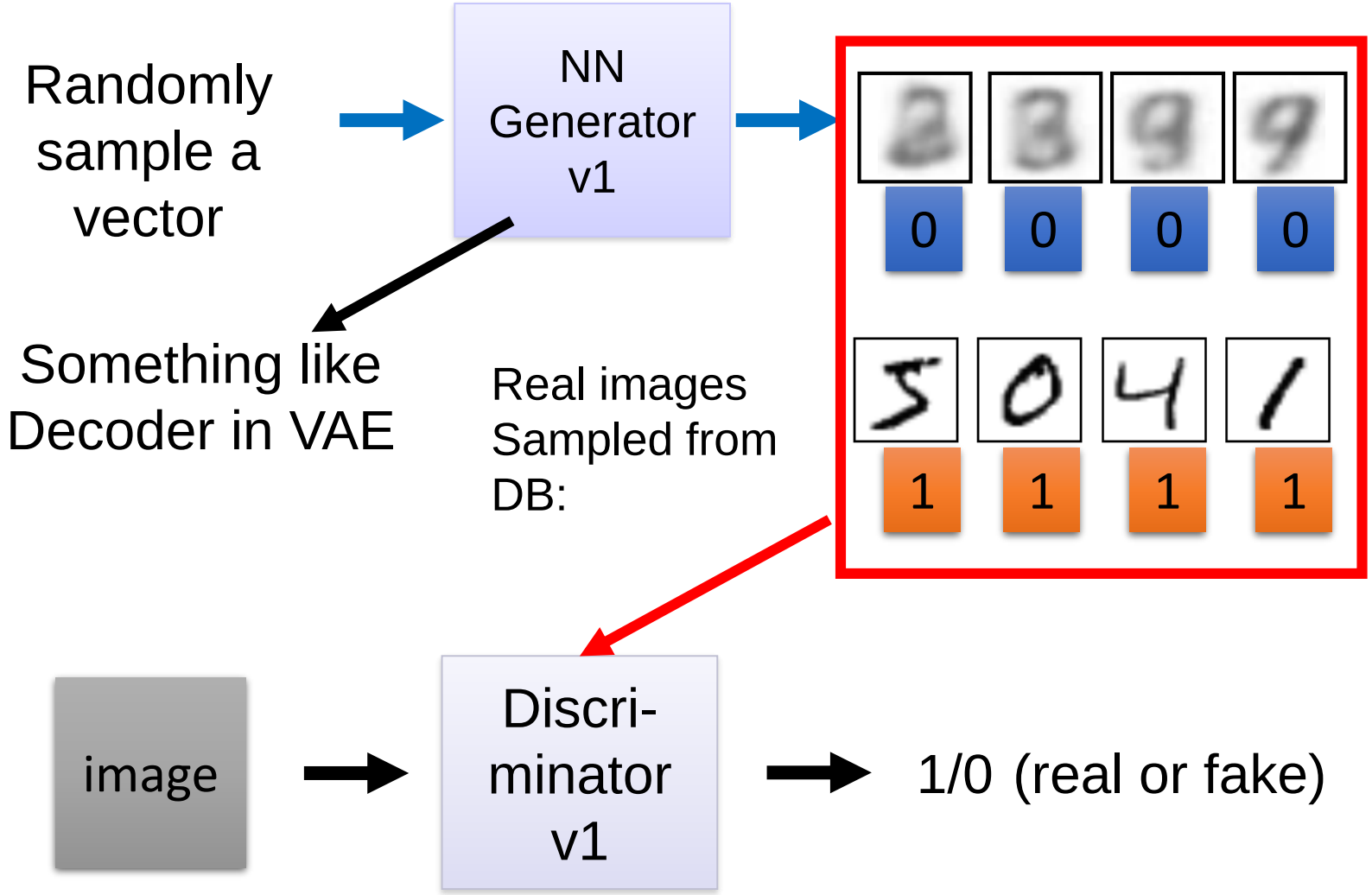
- It does not really try to simulate real images



Gradual and Step-wise Generation



GAN – Learn a Discriminator



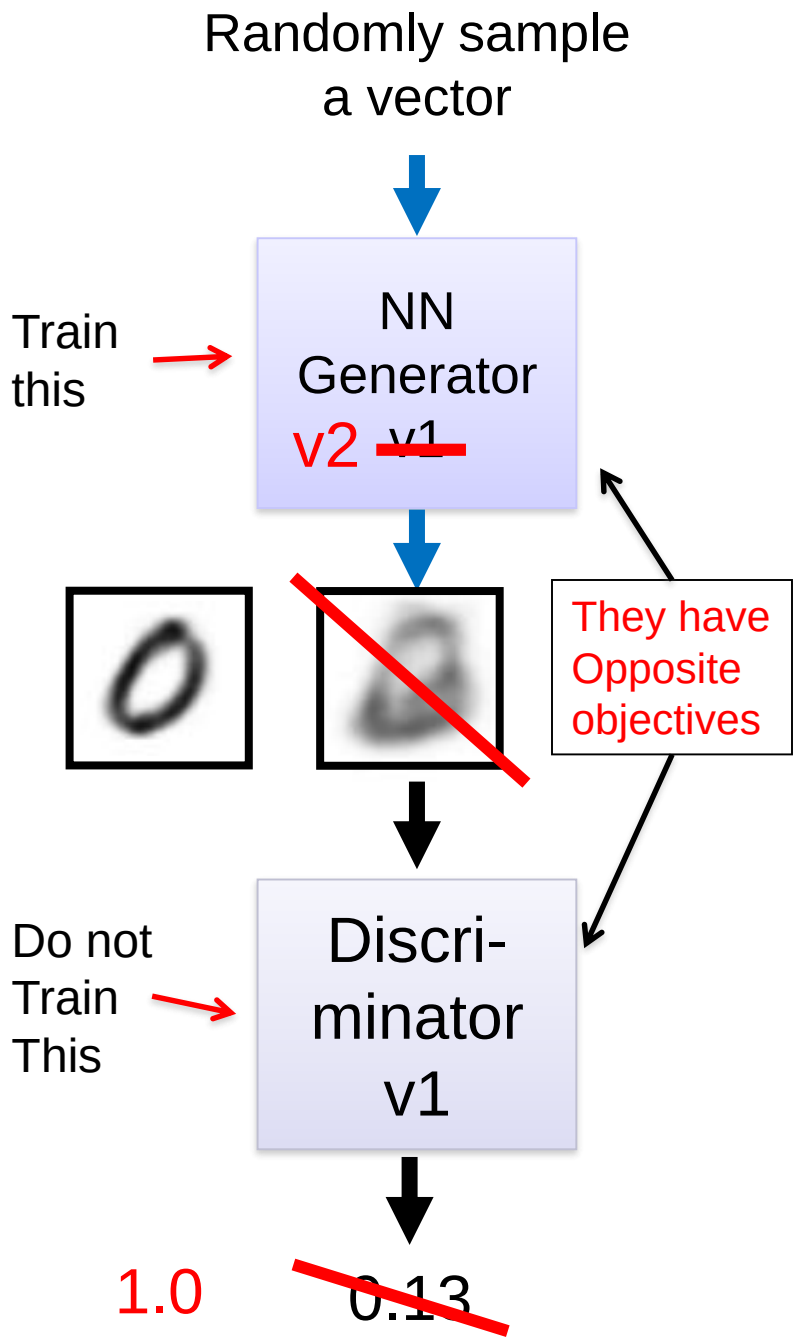
GAN – Learn a Generator

Updating the parameters of generator

➡ The output be classified as “real” (as close to 1 as possible)

Generator + Discriminator = a network

Using gradient descent to update the parameters in the generator, but fix the discriminator



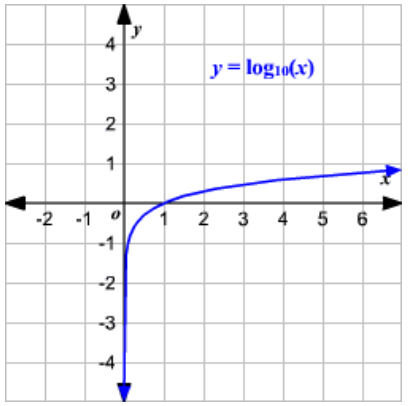
Generative Adversarial Networks

- The generator and discriminator networks compete:
  - Discriminator network trains to classify real vs. generated images.
    - Tries to maximize probability of real images, minimize probability of sampled images.
    - A standard supervised learning problem.
  - Generator network adjust parameters so samples fool the discriminator.
    - It never sees real data.
    - Trains using the gradient of the discriminator network.
      - Backpropagated through the network so samples look more like real images.

Generative Model of Natural Image

- Notation of Generative Model:
  - The distribution of real data : $x \sim p_{data}(x)$
  - Sample from $p_{data}(x)$: X
  - The distribution of generated data : $z \sim p(\mathrm{z})$
  - Sample from $P(z)$: Z
  - Generator Network : $G(z)$
  - Discriminator Network : $D(x)$
  - Real data X : $x \sim P_{data}$
- Unsupervised representation learning
  - Transfer learned representation to discriminative tasks, retrieval, clustering, etc.
  - Semi-supervised learning: very little labeled data, regularization, etc.
- Understand data; Density estimation ...

Objective Function



- Minimax value function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Discriminator pushes up

Discriminator's ability to recognize data as being real

Discriminator's ability to recognize generator samples as being fake

Generator pushes down

Objective Function - Generator

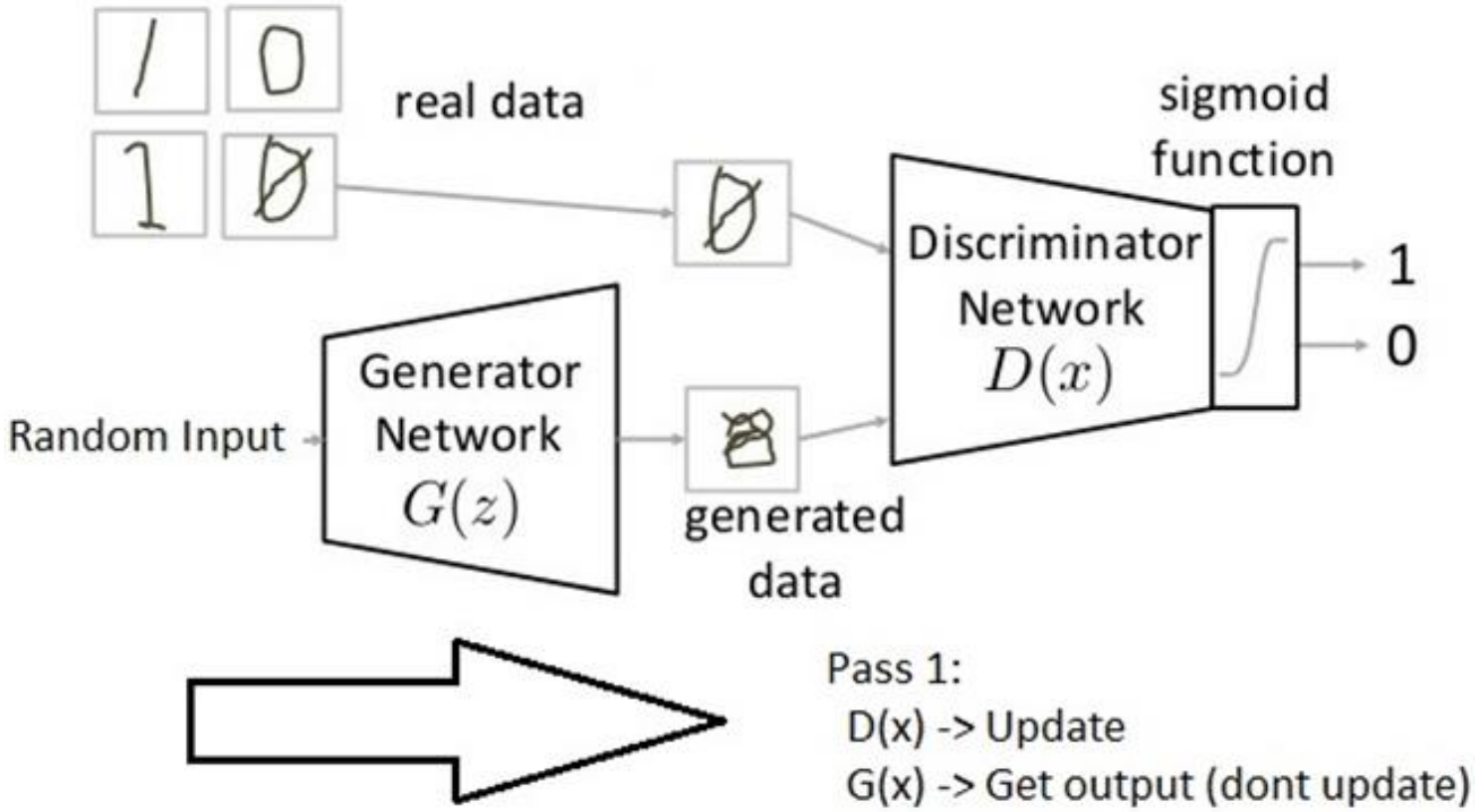
- While the previous objective function is theoretically elegant, in practice the gradient for *G* vanishes before reaching best solution.
- Use same objective for *D* but change objective for *G* to:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$$

- Sometimes better if instead of using one minibatch at a time to compute gradient and do batch normalization, we also have a fixed subset of training set, and use combination of fixed subset and current minibatch.

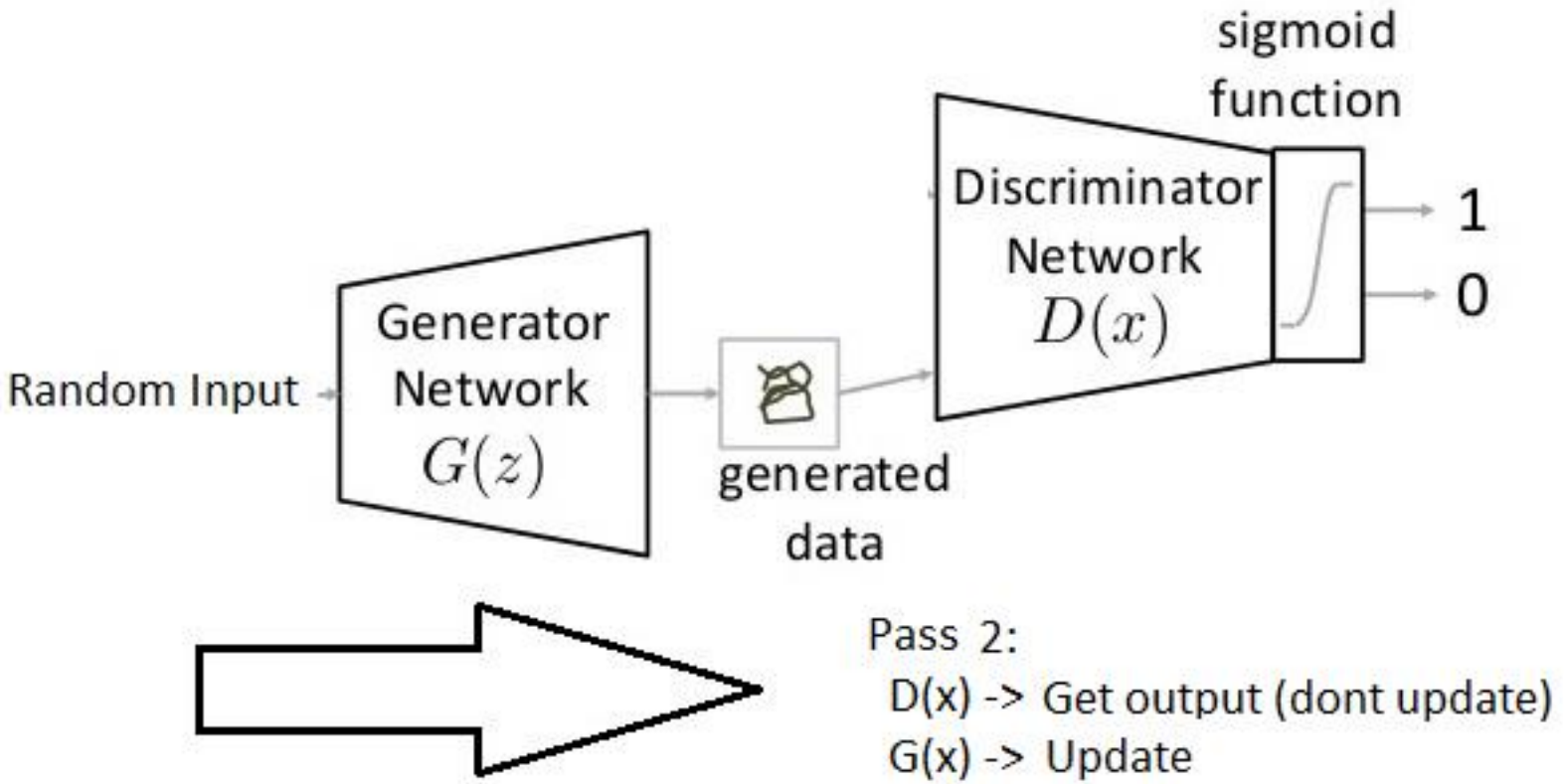
Training GAN (1)

- **Pass 1**: Train discriminator and freeze generator.



Training GAN (2)

- **Pass 2**: Train generator and freeze discriminator



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

---

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

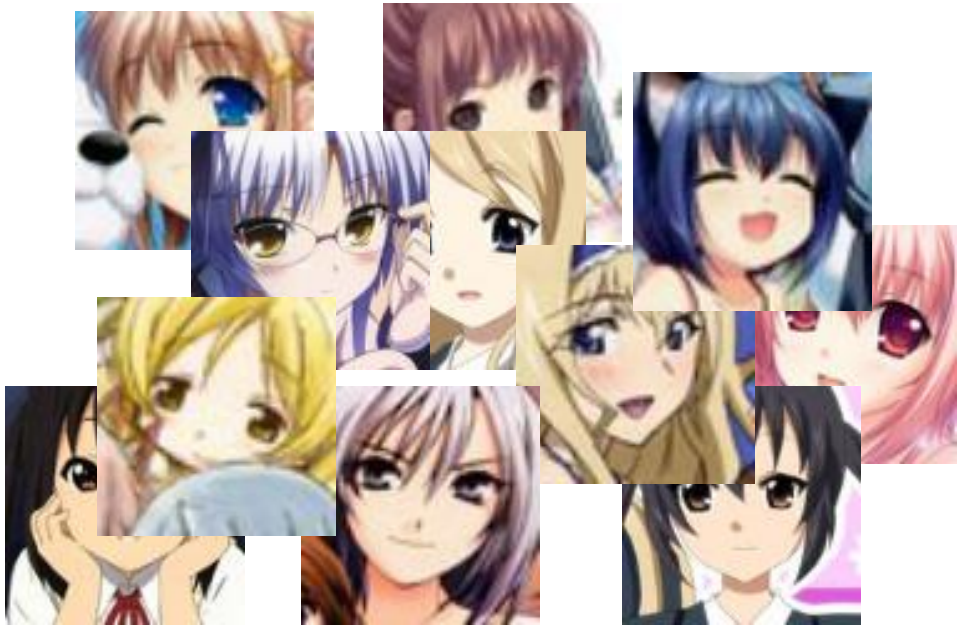
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Steps to train a GAN

- **Step 1: Define the problem.** Do you want to generate fake images or fake text. Here you should completely define the problem and collect data for it.
- **Step 2: Define architecture of GAN.** Define how your GAN should look like. Should both your generator and discriminator be multi layer perceptrons, or convolutional neural networks? This step will depend on what problem you are trying to solve.
- **Step 3: Train Discriminator on real data for *n* epochs.** Get the data you want to generate fake on and train the discriminator to correctly predict them as real. Here value *n* can be any natural number between 1 and infinity.
- **Step 4: Generate fake data from Generator and train Discriminator on fake data.** Get generated data and let the discriminator correctly predict them as fake.
- **Step 5: Train Generator with the output of Discriminator.** Now when the discriminator is trained, you can get its predictions and use it as an objective for training the generator. Train the generator to fool the discriminator.
- **Step 6: Repeat step 3 to step 5 for a few epochs.**
- **Step 7: Check if the fake data manually if it seems legit. If it seems appropriate, stop training, else go to step 3.**

Generating 2nd element figures

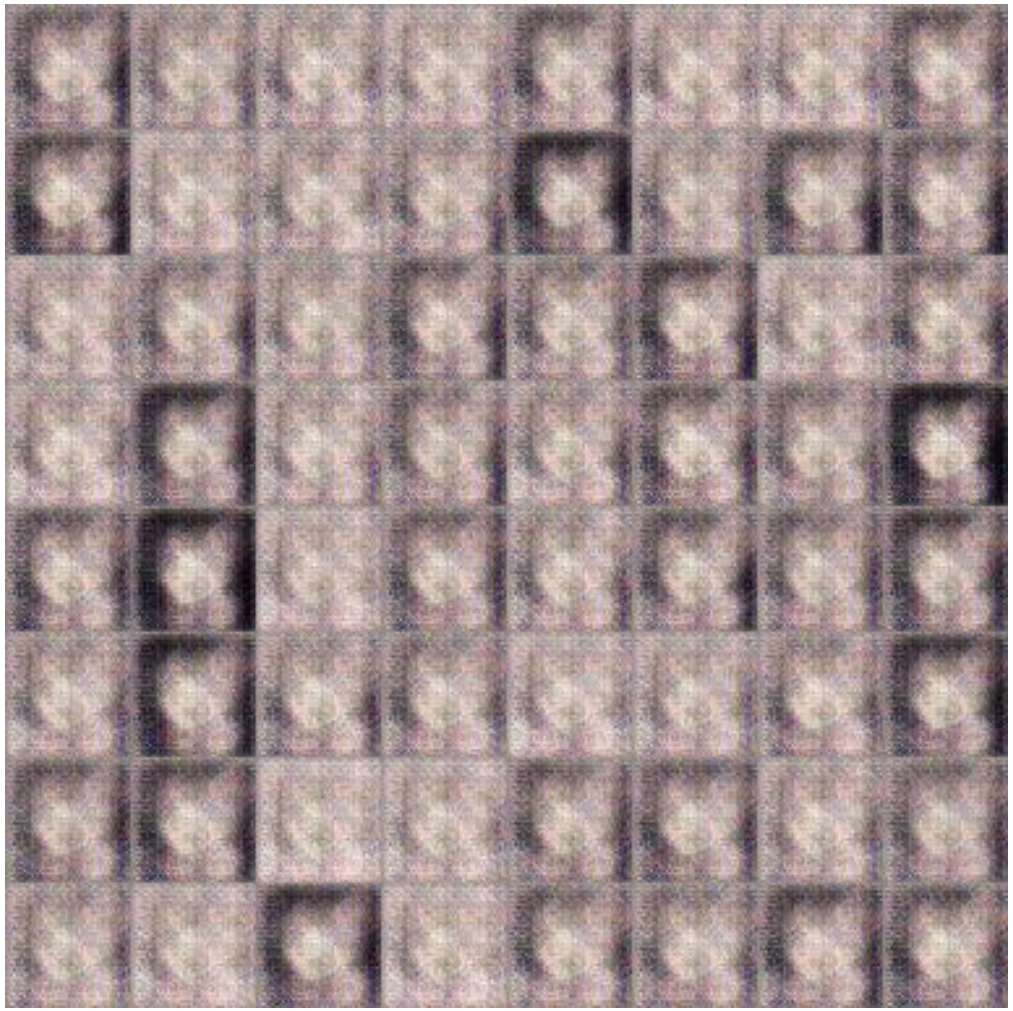


You can use the following to start a project (but this is in Chinese):

Source of images: https://zhuanlan.zhihu.com/p/24767059
From Dr. HY Lee's notes.

DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

GAN – generating 2nd element figures



100 rounds

This is fast, I think you can use your CPU

GAN – generating 2nd element figures

1000 rounds

GAN – generating 2nd element figures

2000 rounds

GAN – generating 2nd element figures

5000 rounds

GAN – generating 2nd element figures

10,000 rounds

GAN – generating 2nd element figures

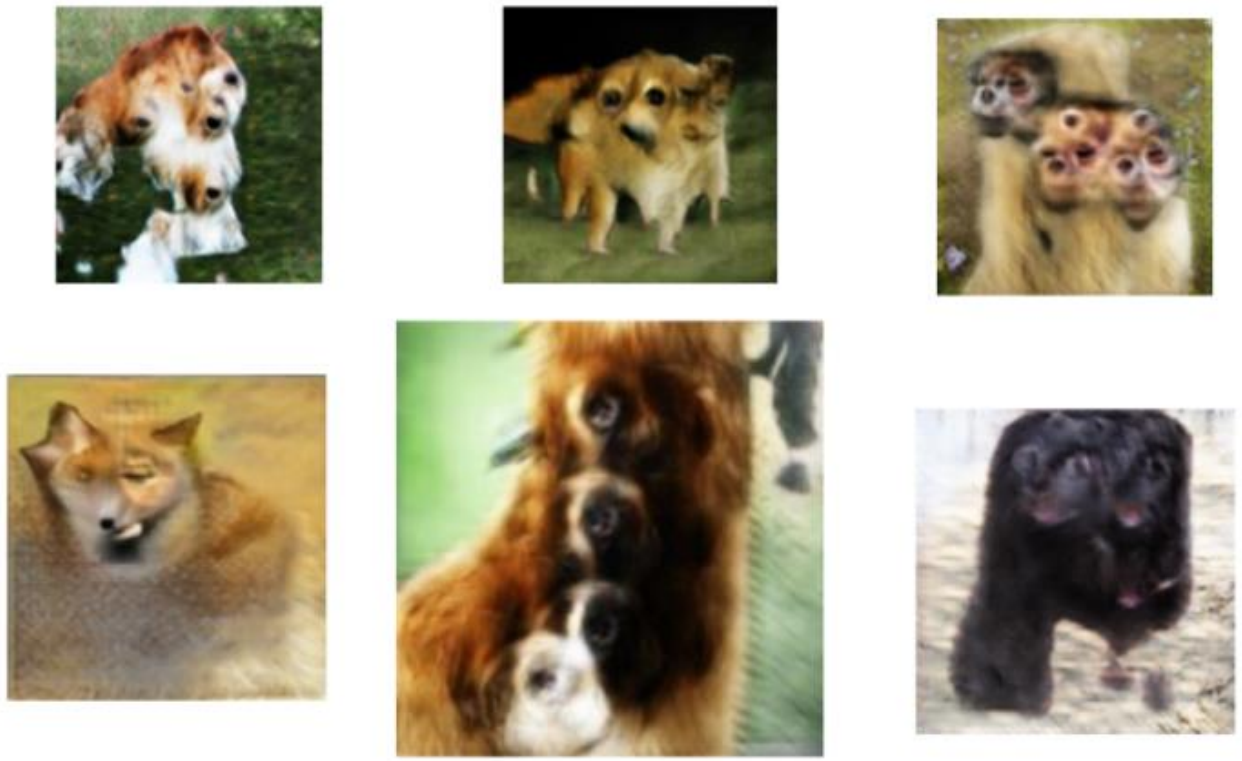
20,000 rounds

GAN – generating 2nd element figures

50,000 rounds

Challenges with GANs

- **Problem with Counting**: GANs fail to differentiate how many of a particular object should occur at a location. As we can see below, it gives more number of eyes in the head than naturally present.

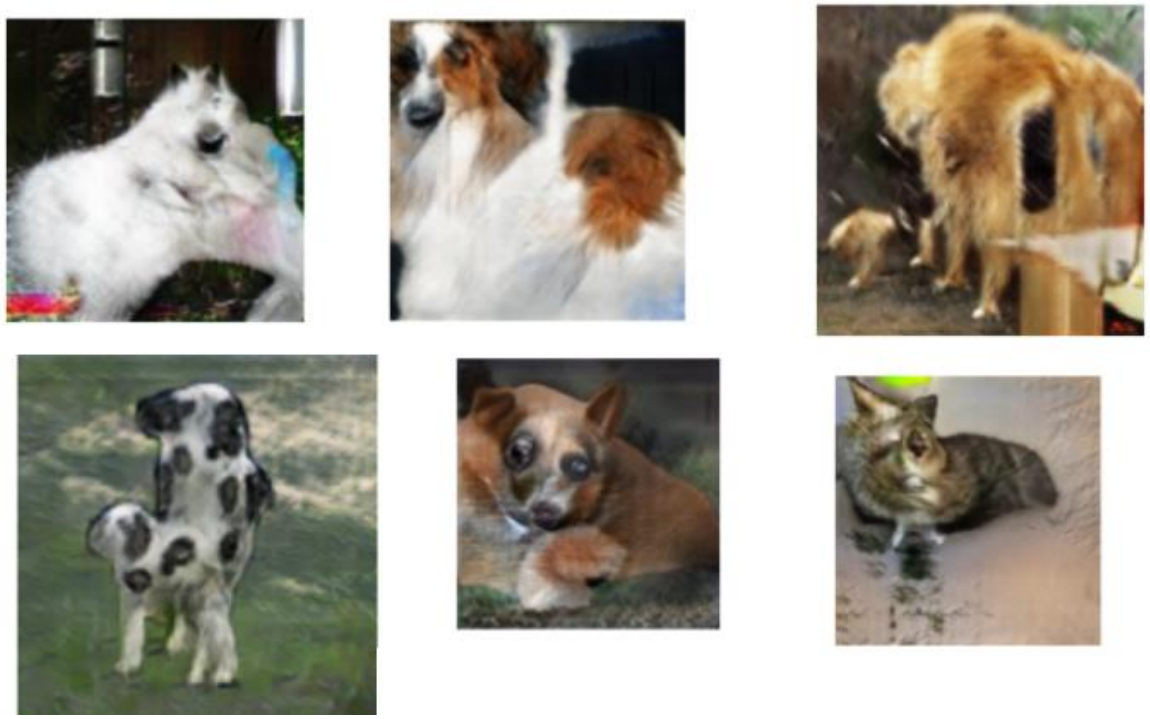


Challenges with GANs

- **Problems with Perspective**: GANs fail to adapt to 3D objects. It doesn't understand perspective, i.e.difference between frontview and backview. As we can see below, it gives flat (2D) representation of 3D objects.

Challenges with GANs

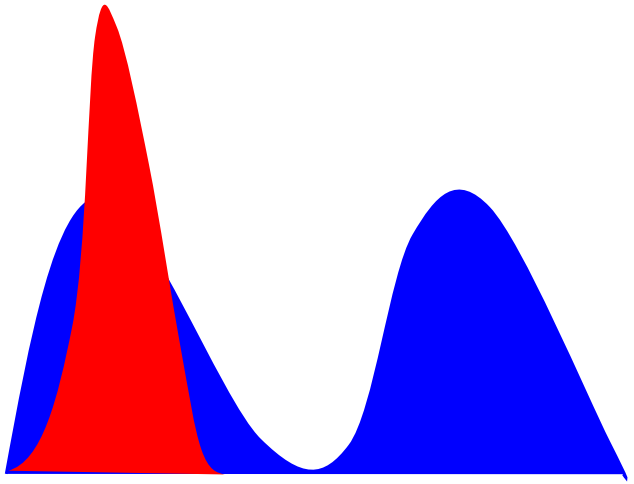
- **Problems with Global Structures:** Same as the problem with perspective, GANs do not understand a holistic structure. For example, a cow standing on its hind legs and simultaneously on all four legs. That is definitely not possible in real life!



Mode Collapse

Converge to same faces

Generated Distribution

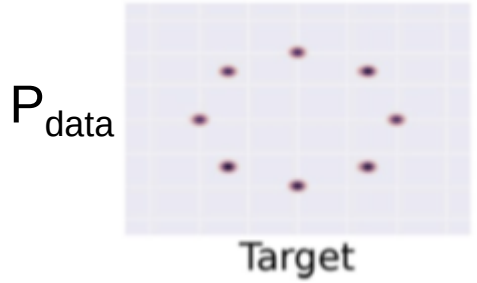


Data Distribution

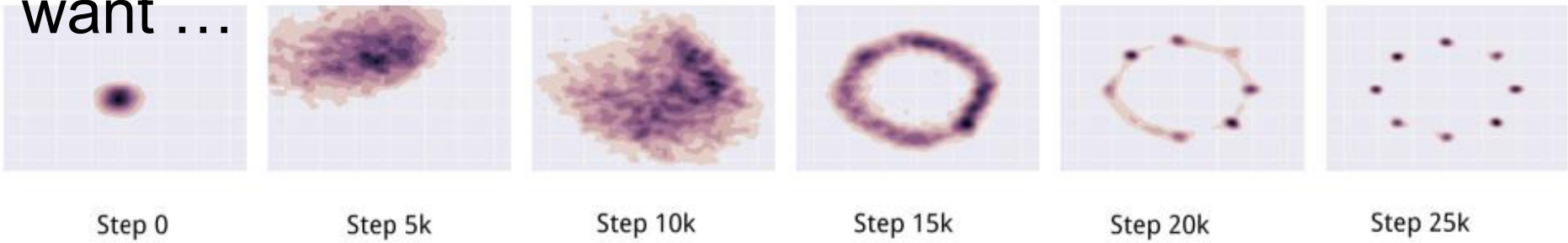
Sometimes, this is hard to tell since
one sees only what's generated, but not what's missed.

Mode Collapse Example

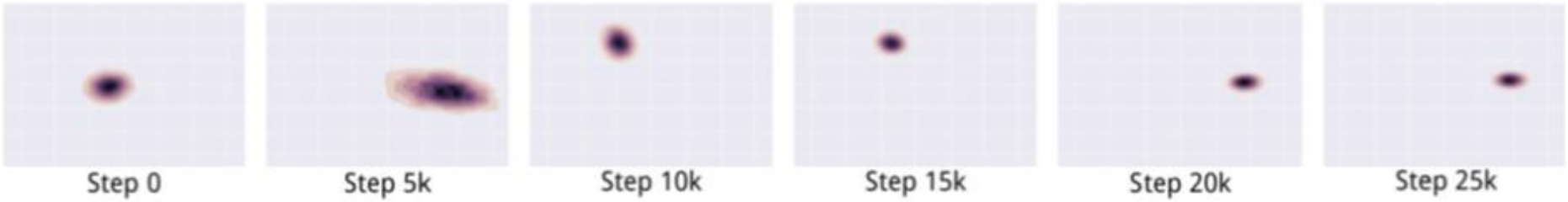
8 Gaussian distributions:



What we want …



In reality …



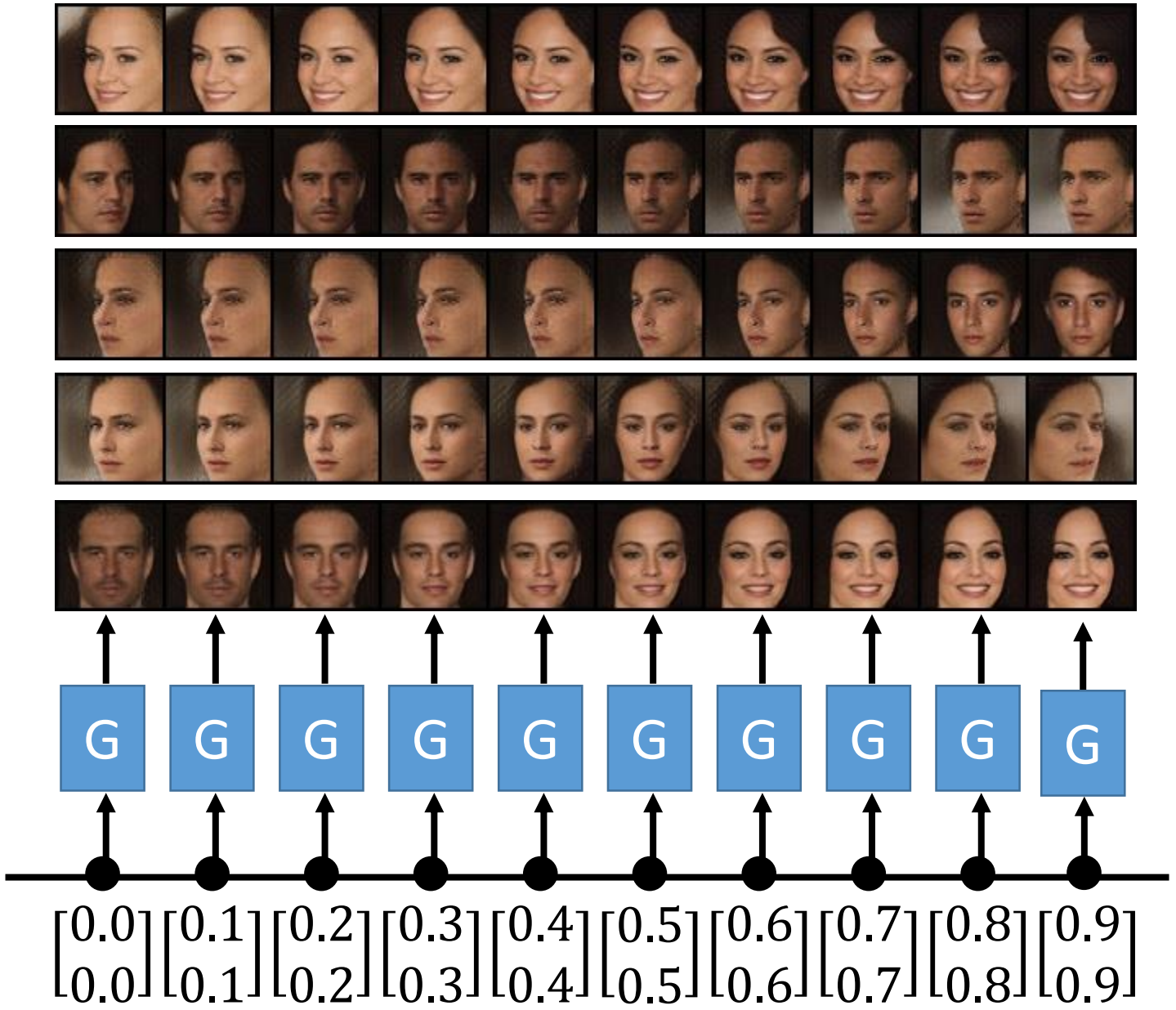
Deep Convolutional Generative Adversarial Network (DCGAN)

- Use deep CNN for generator and discriminator instead of MLP.
  - Replace any pooling layers with strided convolution.
  - Use batchnorm in both the generator and the discriminator.
  - Remove fully connected hidden layers for deeper architectures.
  - Uses Tanh for the output (and sigmod).
  - Use LeakyReLU in the discriminator and ReLU in the generator.
- Use the trained discriminators for image classification tasks.

Image Generation Results

Figure : Generated bedrooms on LSUN

Face Generation



Conditional Generative Adversarial Nets

- Condition generation on additional info y (e.g. class label)
- Objective function:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z|y)|y))]$$

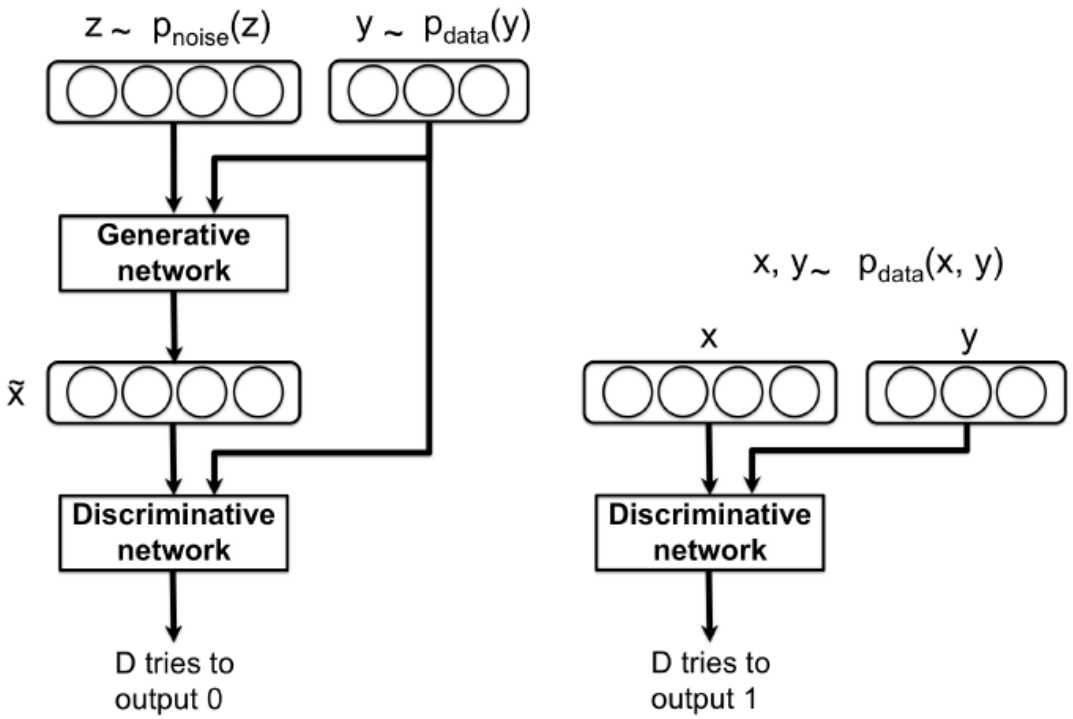


Figure : Model structure for conditional GAN

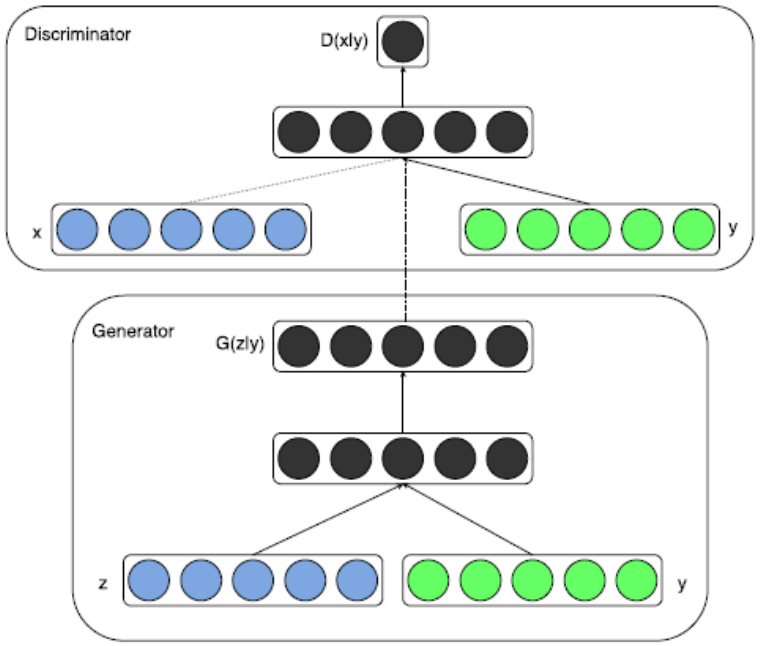
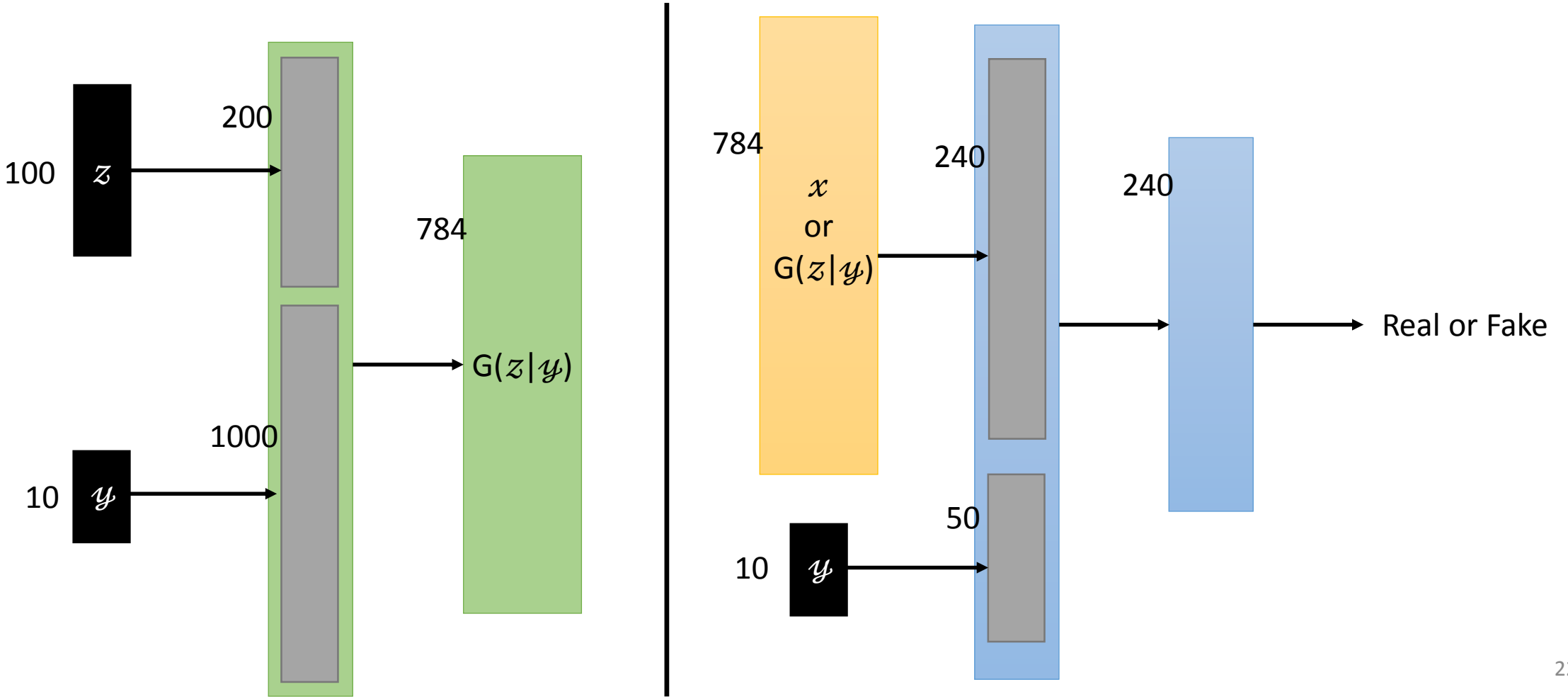


Figure : Conditional GAN with MLP

Experiment

- Handwritten digit generating (on MNIST)



Results on CIFAR-100 Generated

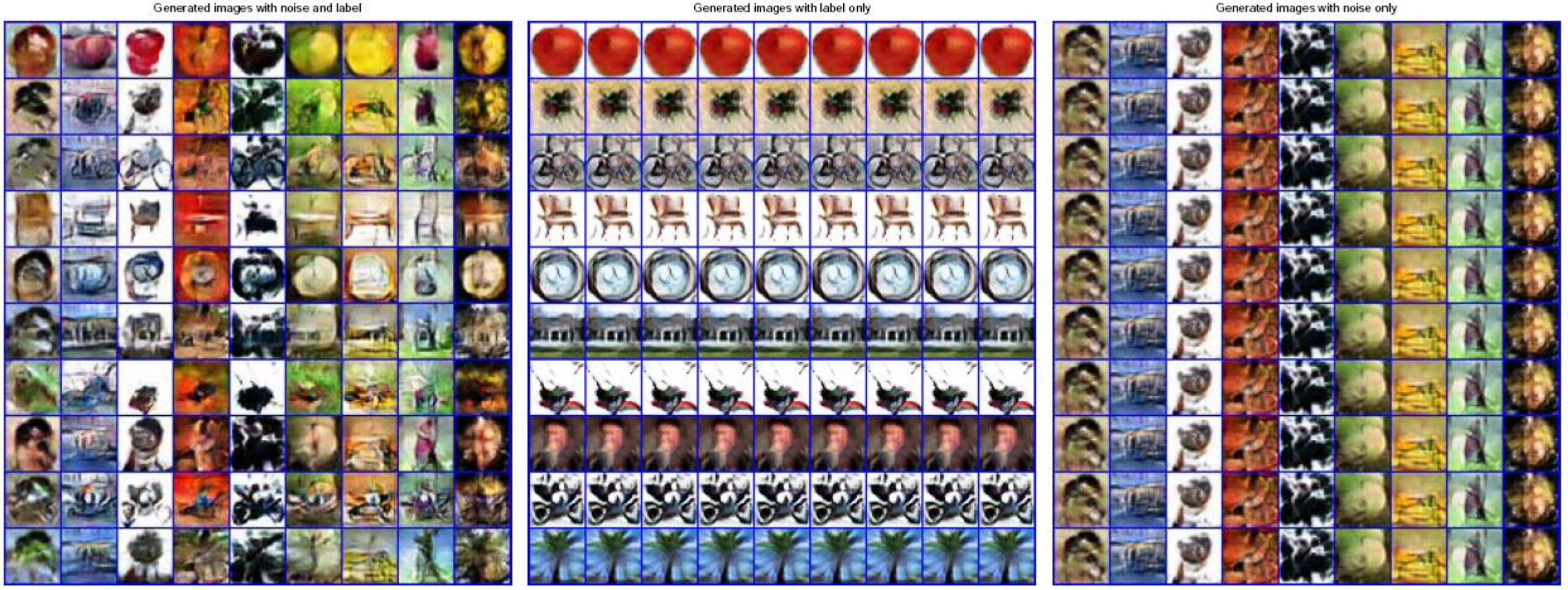
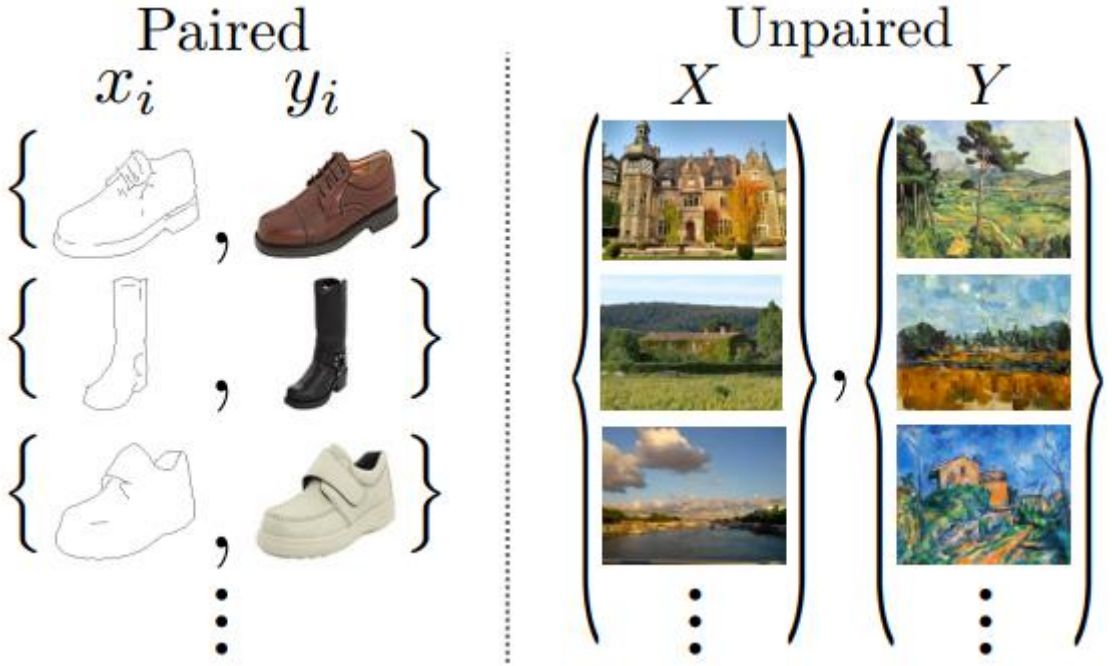


Figure : (Left) Each row has the same label y and each column has the same noisy vector z (Middel) Set noisy vector as zero (Right) Set label vector as zero

Cycle-Consistent Adversarial Networks (**CycleGAN**)

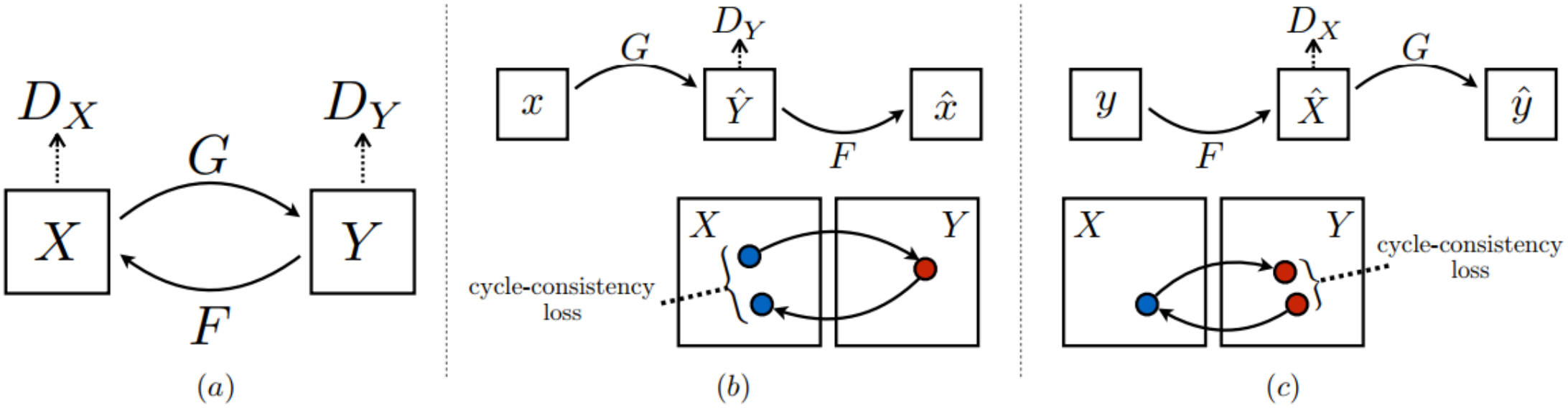
- Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs.
- However, for many tasks, paired training data will not be available.



The Problem of Adversarial Loss

- Adversarial training can, in theory, learn mappings G and F that produce outputs identically distributed as target domains Y and X respectively.
- However, with large enough capacity, a network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can **induce an output distribution that matches the target distribution**.
- Thus, adversarial losses alone **cannot guarantee that the learned function can map an individual input x_i to a desired output y_i** .

Cycle Consistency Loss



Cycle Consistency Loss

- For each image x from domain X, the image translation cycle should be able to bring x back to the original image,
  i.e. **x → G(x) → F(G(x)) ≈ x**.
  We call this forward cycle consistency
- Similarly, for each image y from domain Y , G and F should also satisfy backward cycle consistency:
  i.e. **y → F(y) → G(F(y)) ≈ y**.

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) =& \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] \\ &+ \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].\end{aligned}$$

Full Objective

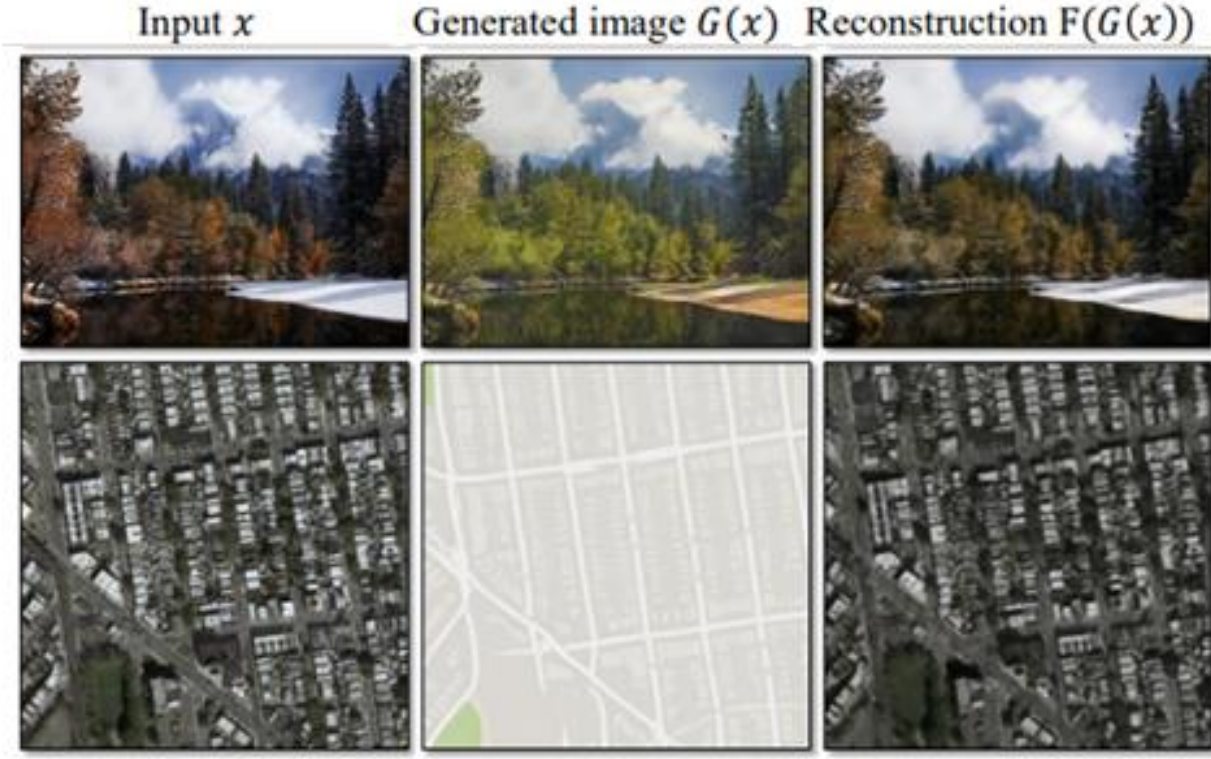
- Full objective is:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) =& \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ &+ \lambda \mathcal{L}_{\text{cyc}}(G, F), \qquad (3)\end{aligned}$$

- Aim to solve:

$$G^*, F^* = \arg\min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y). \qquad (4)$$

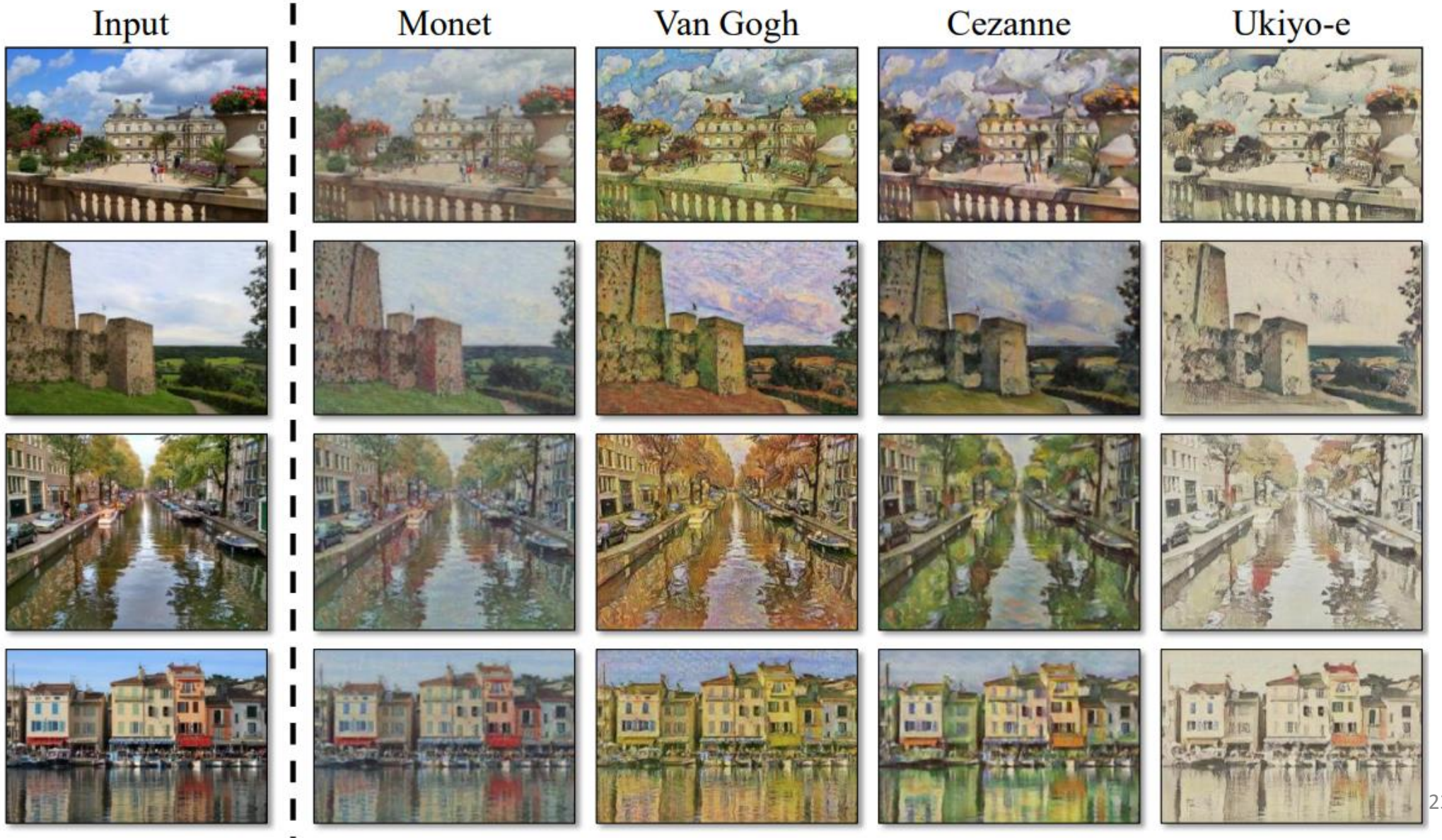
Cycle Consistency Loss



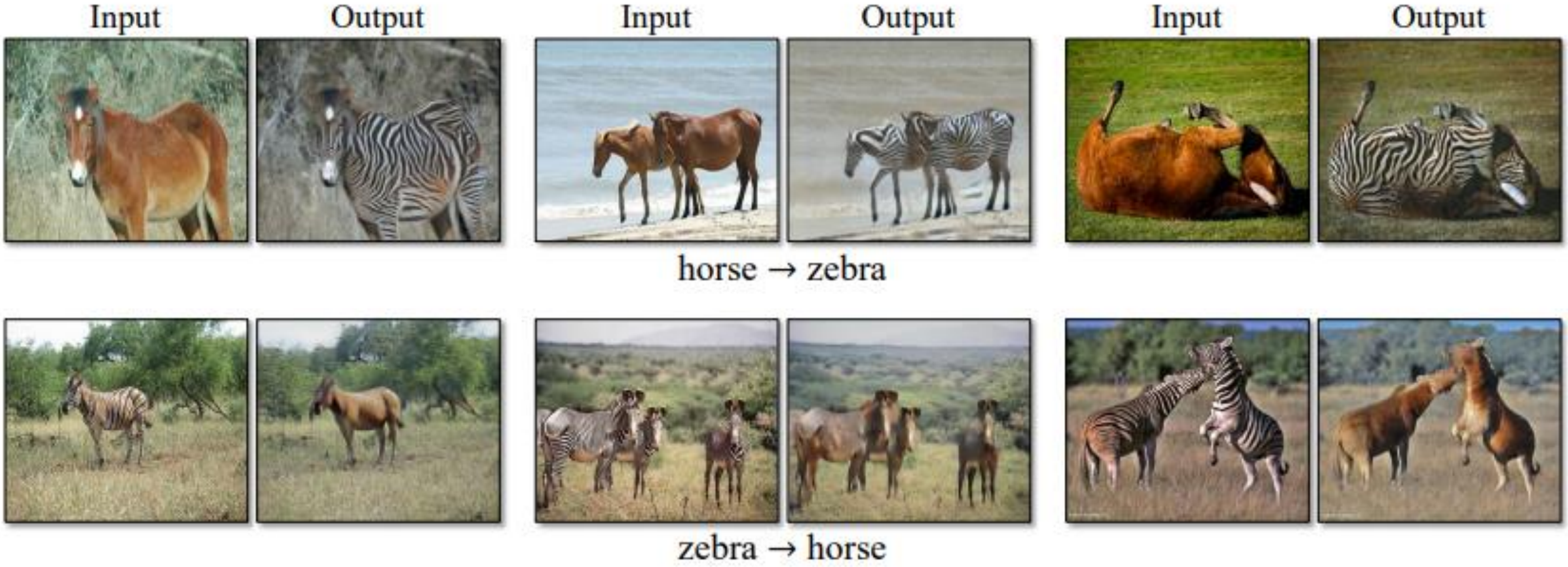
Applications

- Collection style transfer
- Object transfiguration
- Season transfer
- Photo generation from paintings
- Photo enhancement

Collection Style Transfer



Object Transfiguration



Season Transfer

winter Yosemite → summer Yosemite

summer Yosemite → winter Yosemite

Photo Generation from Paintings

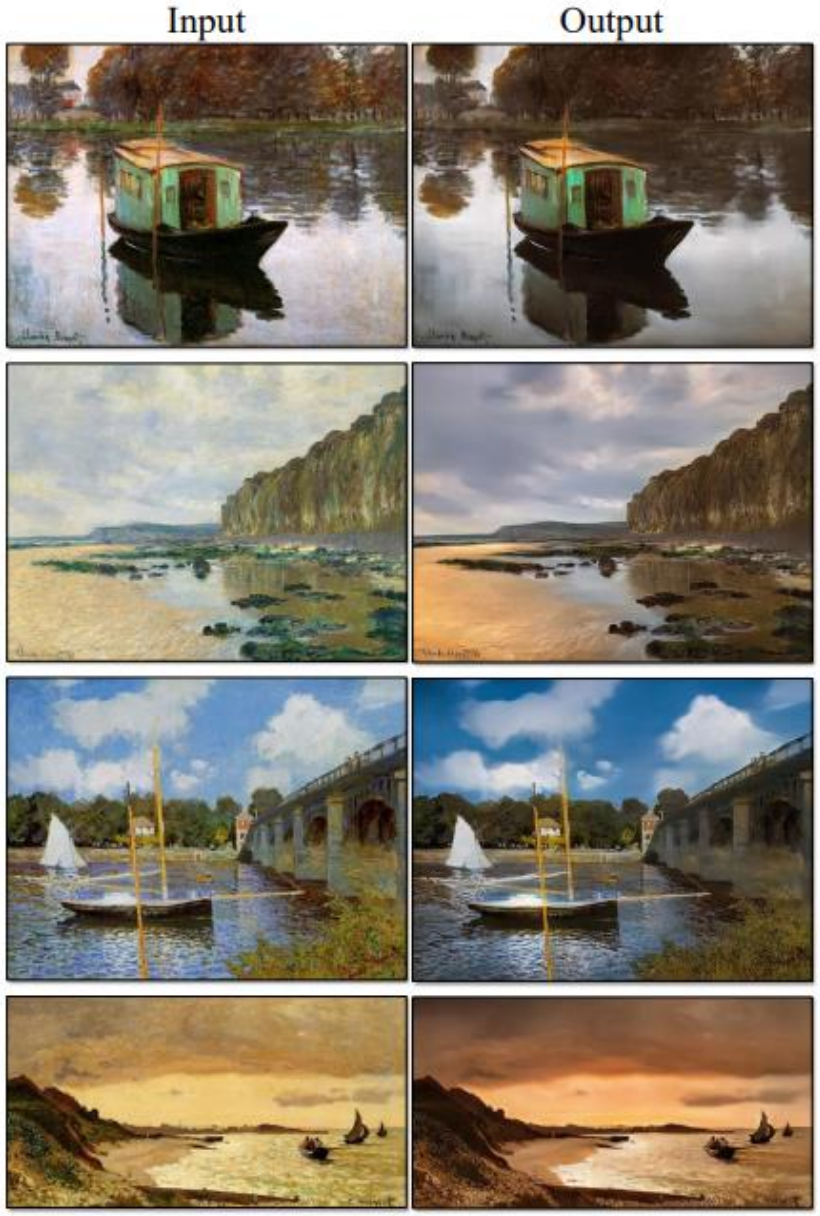


Photo Enhancement – Simulation of Depth Camera

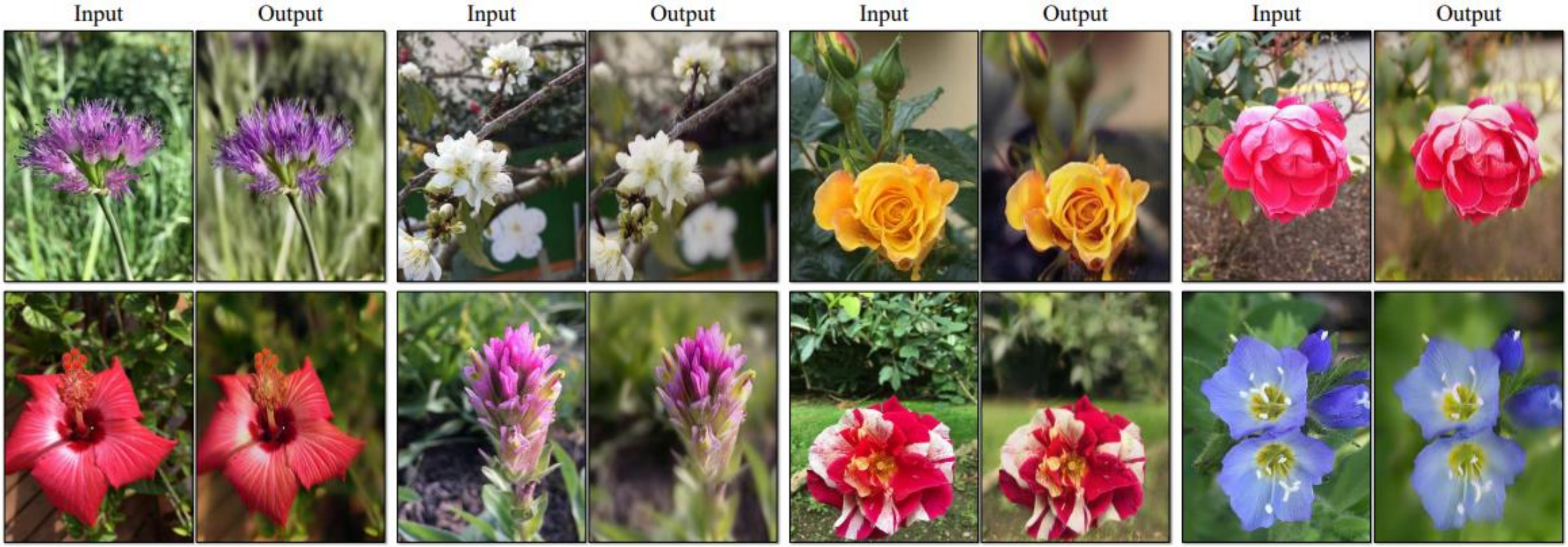


Figure 14: Photo enhancement: mapping from a set of iPhone snaps to professional DSLR photographs, the system often learns to produce shallow focus. Here we show some of the most successful results in our test set – average performance is considerably worse. Please see our website for more comprehensive and random examples.

Comparison with Gatys et al.

Failure Cases

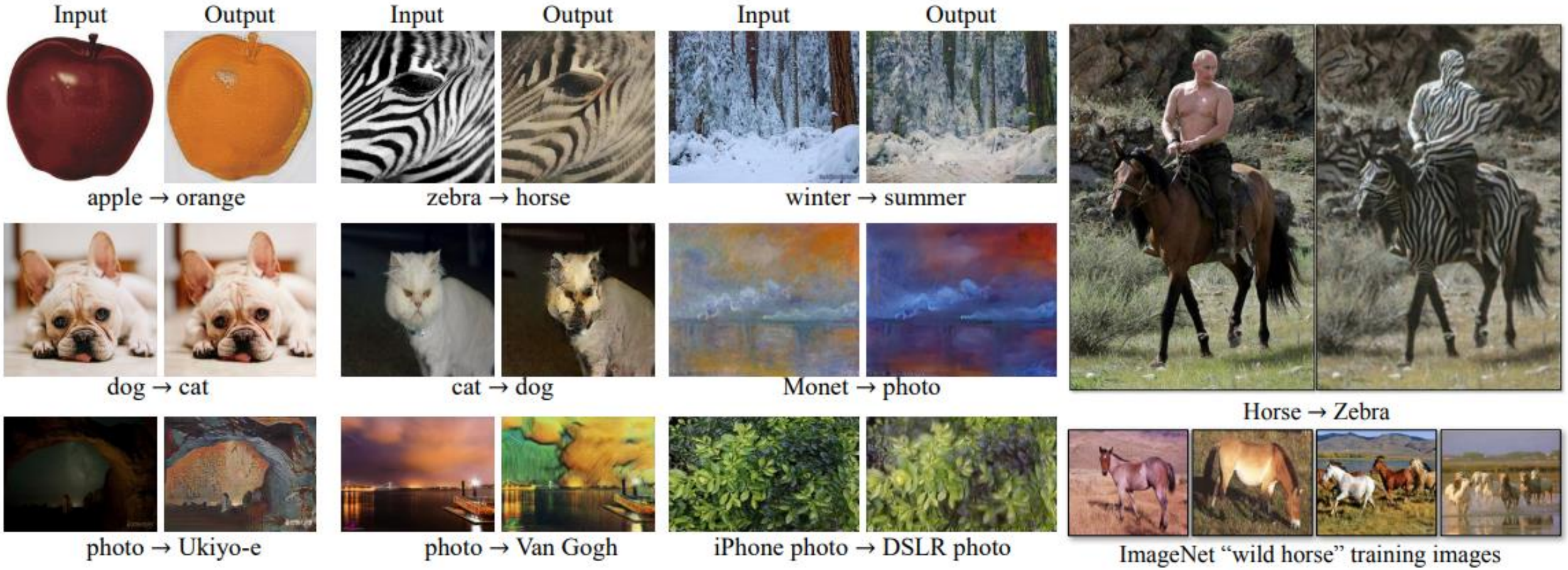


Figure 17: Typical failure cases of our method. Please see our website for more comprehensive results.

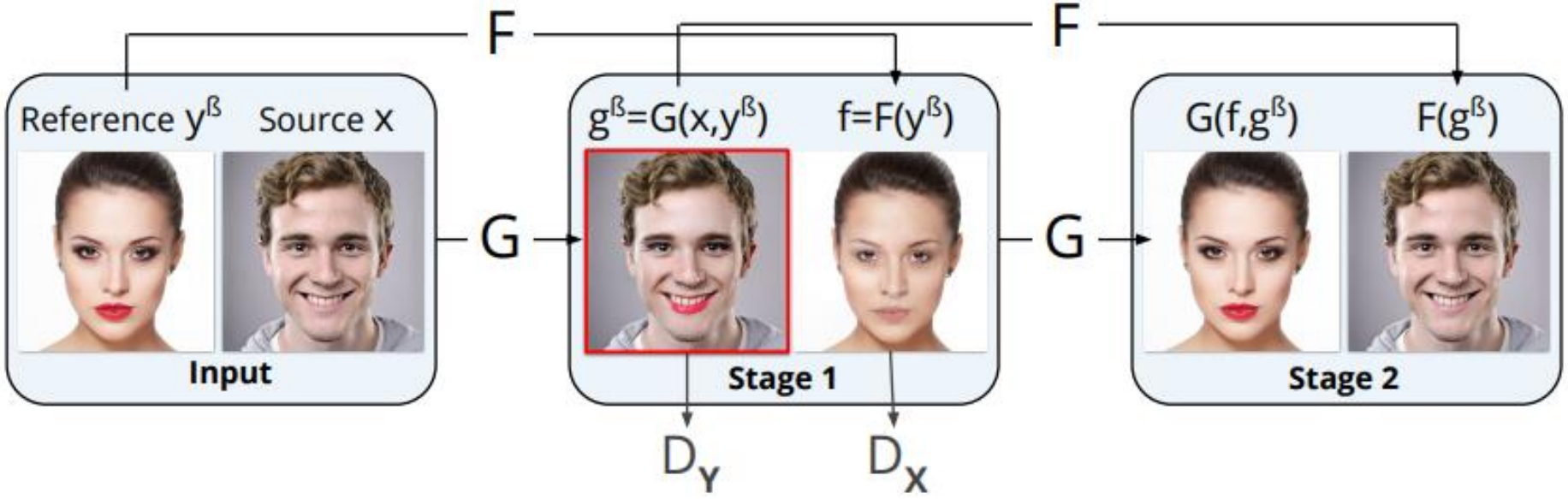
Limitations and Discussion

- On translation tasks that involve color and texture changes, like many of those reported above, the method often succeeds.
- Some failure cases are caused by the distribution characteristics of the training datasets. For example, the horse → zebra example (Figure 17, right) has got confused, because our model was trained on the wild horse and zebra synsets of ImageNet, which does not contain images of a person riding a horse or zebra.
- Handling more varied and extreme transformations, especially **geometric changes**, is an important problem for future work.

PairedCycleGAN : Makeup Transfer and Removal

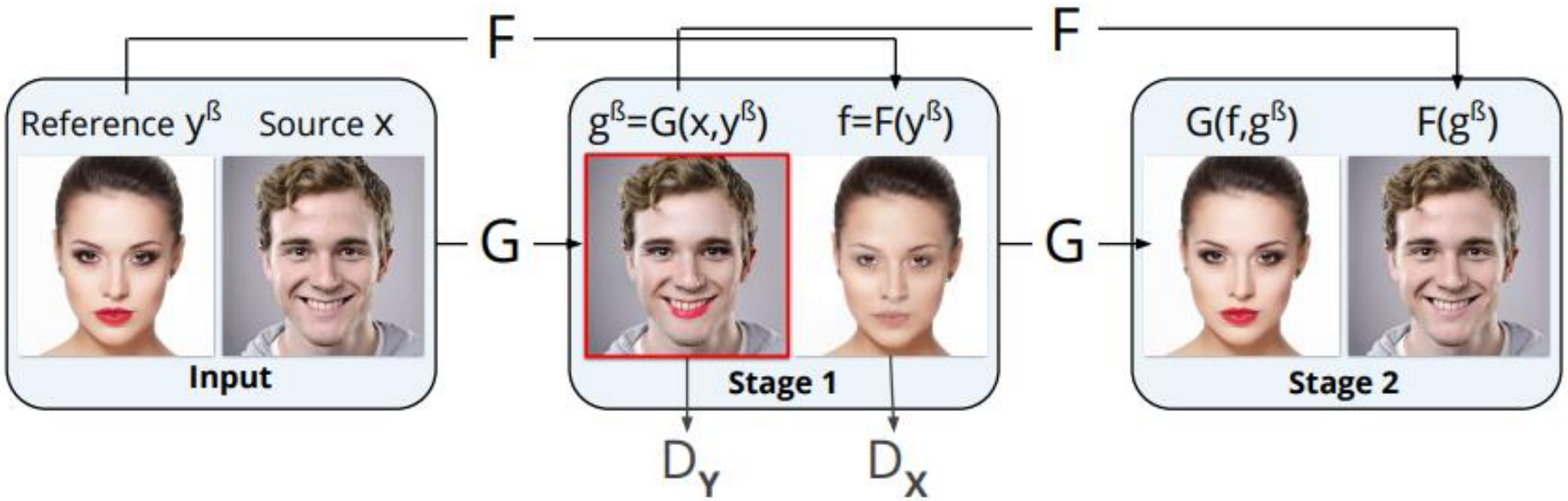
CycleGAN would not replicate a specific example makeup style.

Network Pipeline



G: markup transfer function
F: markup removal function
X: no-makeup image domain
Y: with-makeup image domain
β: specific makeup style

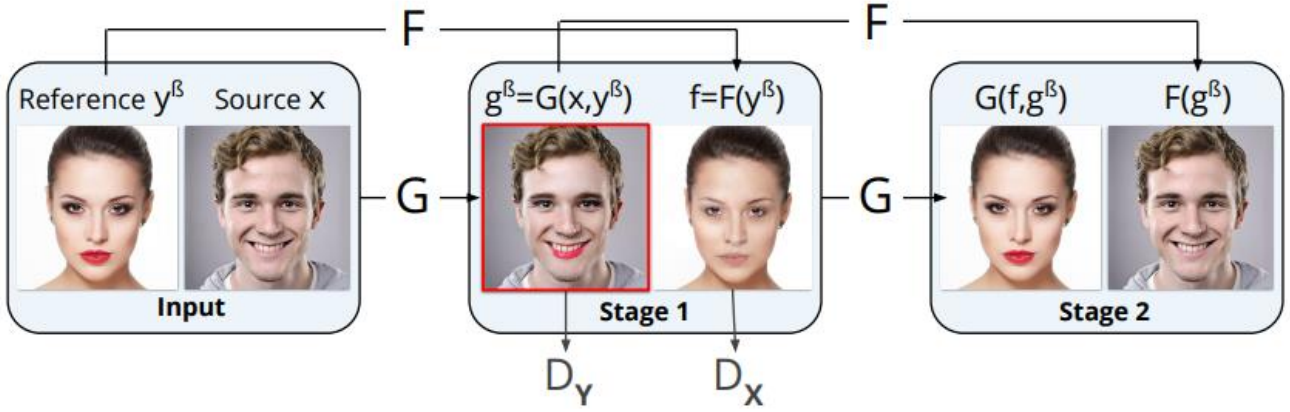
Total Loss



$$\min_{G,F} \max_{D_X,D_Y,D_S} L$$

$$L = \lambda_G L_G + \lambda_F L_F + L_I + L_S + \lambda_P L_P$$

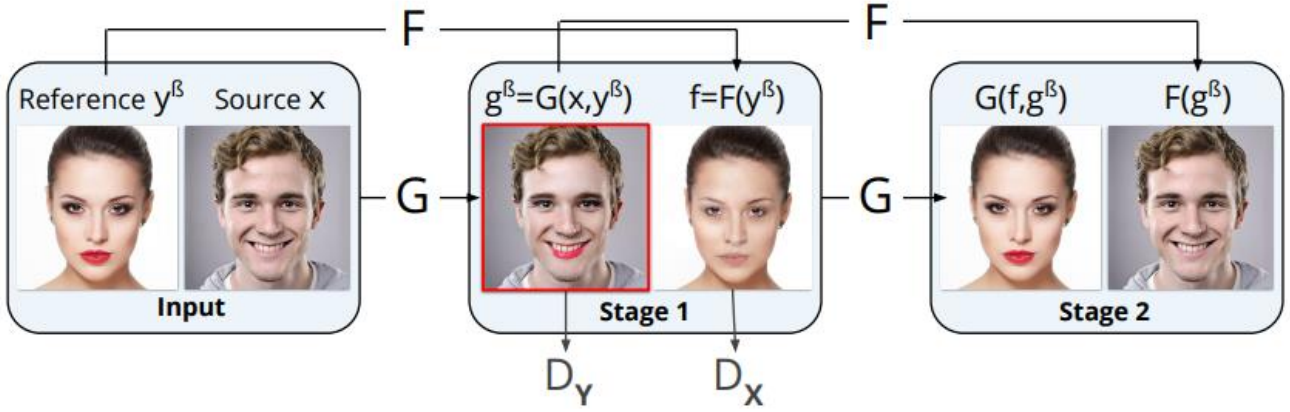
Adversarial Loss for G



$$L = \lambda_G L_G + \lambda_F L_F + L_I + L_S + \lambda_P L_P$$

$$L_G(G, D_Y) = \mathbb{E}_{y \sim \mathcal{P}_Y}[log D_Y(y)] + \mathbb{E}_{x \sim \mathcal{P}_X, y \sim \mathcal{P}_Y}[log(1 - D_Y(G(x, y)))] \quad (1)$$

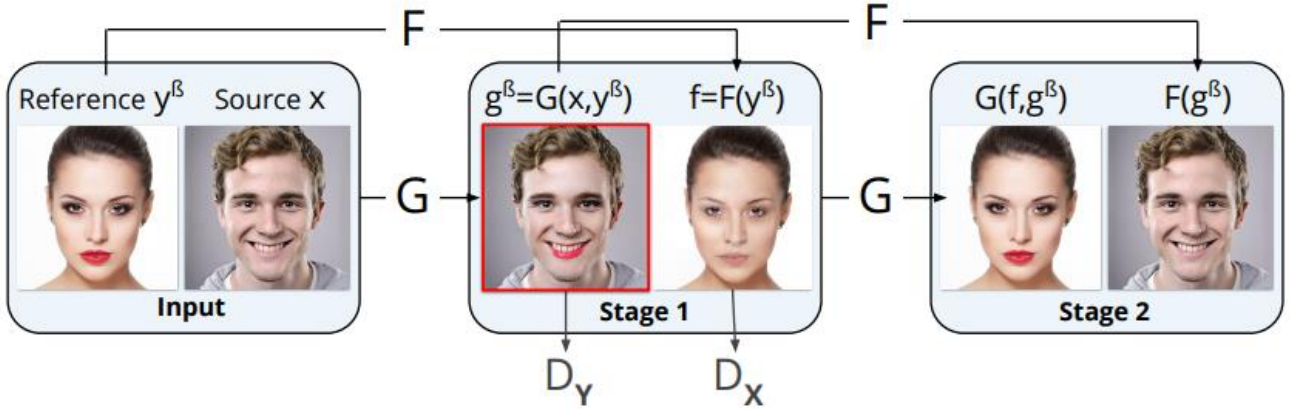
Adversarial Loss for F



$$L = \lambda_G L_G + \lambda_F L_F + L_I + L_S + \lambda_P L_P$$

$$L_F(F, D_X) = \mathbb{E}_{x \sim \mathcal{P}_X}[log D_X(x)] + \mathbb{E}_{y^\beta \sim \mathcal{P}_Y}[log(1 - D_X(F(y^\beta)))] \quad (2)$$

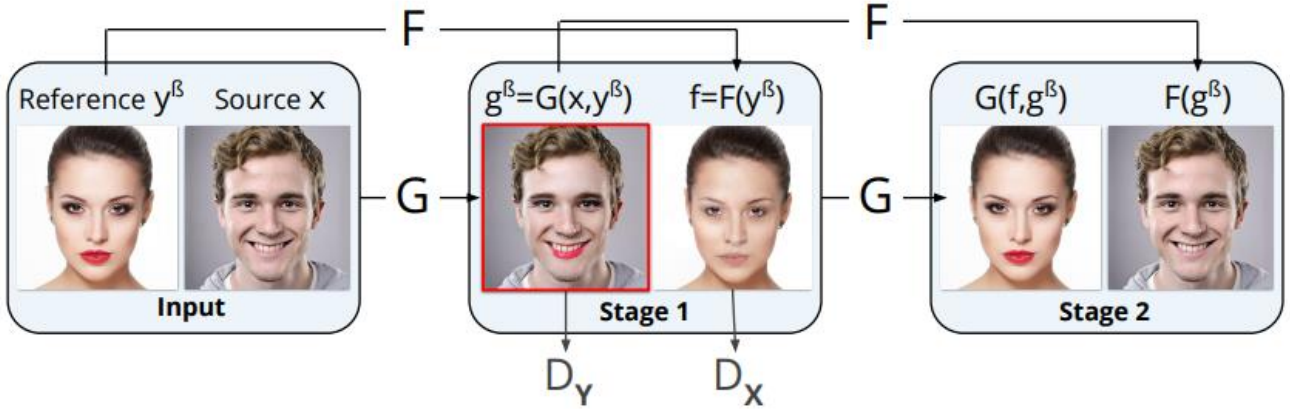
Identity Loss



$$L = \lambda_G L_G + \lambda_F L_F + L_I + L_S + \lambda_P L_P$$

$$L_I(G, F) = \mathbb{E}_{x \sim \mathcal{P}_X, y^\beta \sim \mathcal{P}_Y} [||F(G(x, y^\beta)) - x||_1] \quad (3)$$

Style Losses: L1 Reconstruction Loss

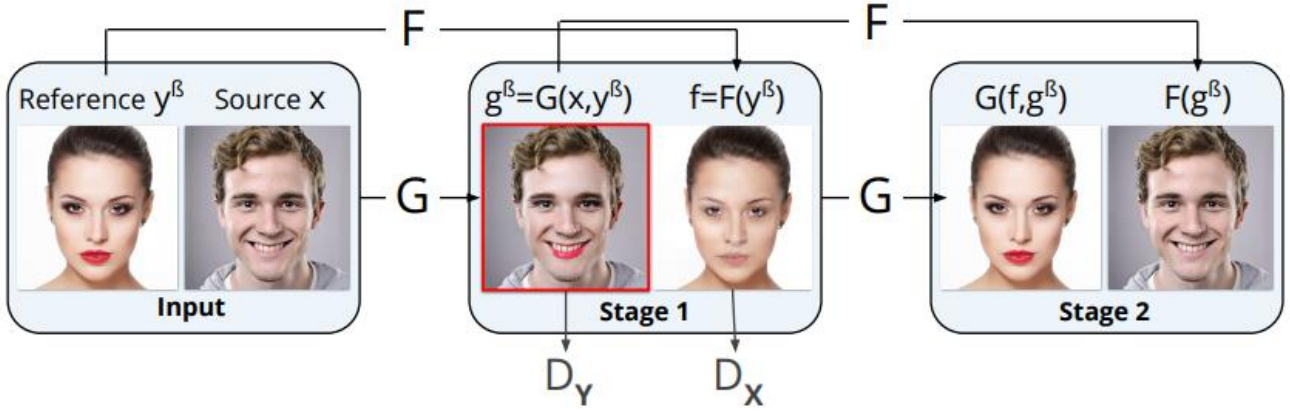


$$L = \lambda_G L_G + \lambda_F L_F + L_I + \boxed{L_S} + \lambda_P L_P$$

$$L_S(G, F) = \mathbb{E}_{x \sim \mathcal{P}_X, y^\beta \sim \mathcal{P}_Y} [||G(F(y^\beta), G(x, y^\beta)) - y^\beta||_1] \quad (4)$$

But it leads to blurry results incapable of transferring sharp edges (e.g. eyelashes and eyeliners)

Style Losses: Style Discriminator Loss



$$L = \lambda_G L_G + \lambda_F L_F + L_I + L_S + \lambda_P L_P$$

$$L_P(G, D_S) = \mathbb{E}_{x \sim \mathcal{P}_X, y^\beta \sim \mathcal{P}_Y}[log D_S(y^\beta, W(x, y^\beta))] + \mathbb{E}_{x \sim \mathcal{P}_X, y^\beta \sim \mathcal{P}_Y}[log(1 - D_S(y^\beta, G(x, y^\beta)))] \quad (5)$$

Synthetic Groundtruth *W*

Warp the reference makeup face to match the detected facial landmarks in the source face.

Network Architecture and Loss Analysis

Without loss for generator,
G could apply the eyeshadow anywhere around the eye.

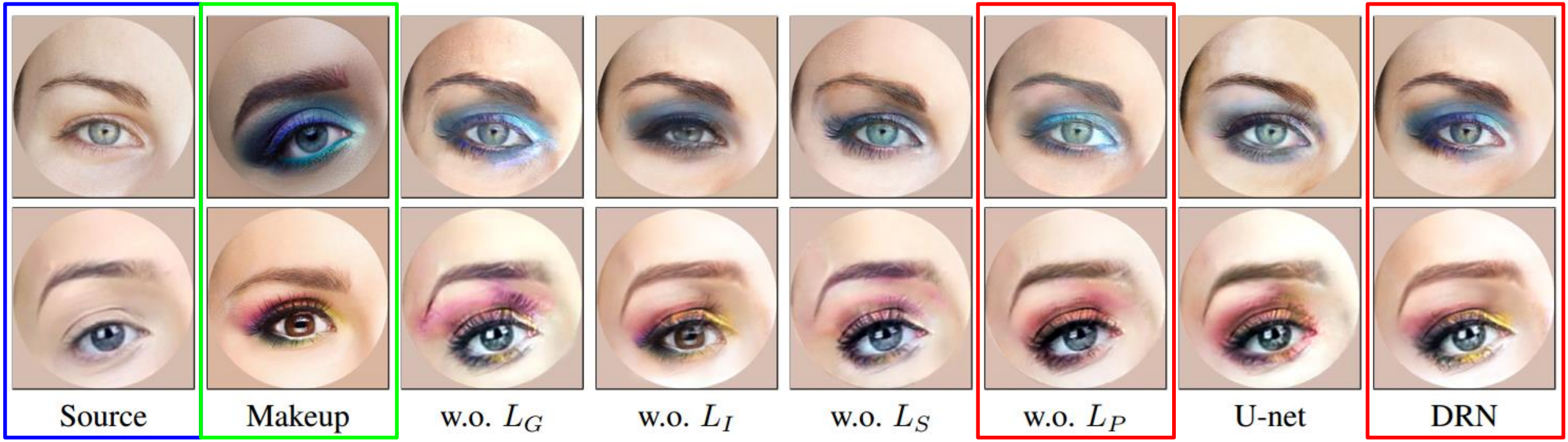
Network Architecture and Loss Analysis

Without identity loss,
G could encodes the characteristics of eyes in both source and reference.

Network Architecture and Loss Analysis

Without style loss,
the results look less saturated and the makeup style is not fully transferred.

Network Architecture and Loss Analysis



Without style discriminator loss,
the results become more vivid, but some eyelashes get neglected.

Network Architecture and Loss Analysis

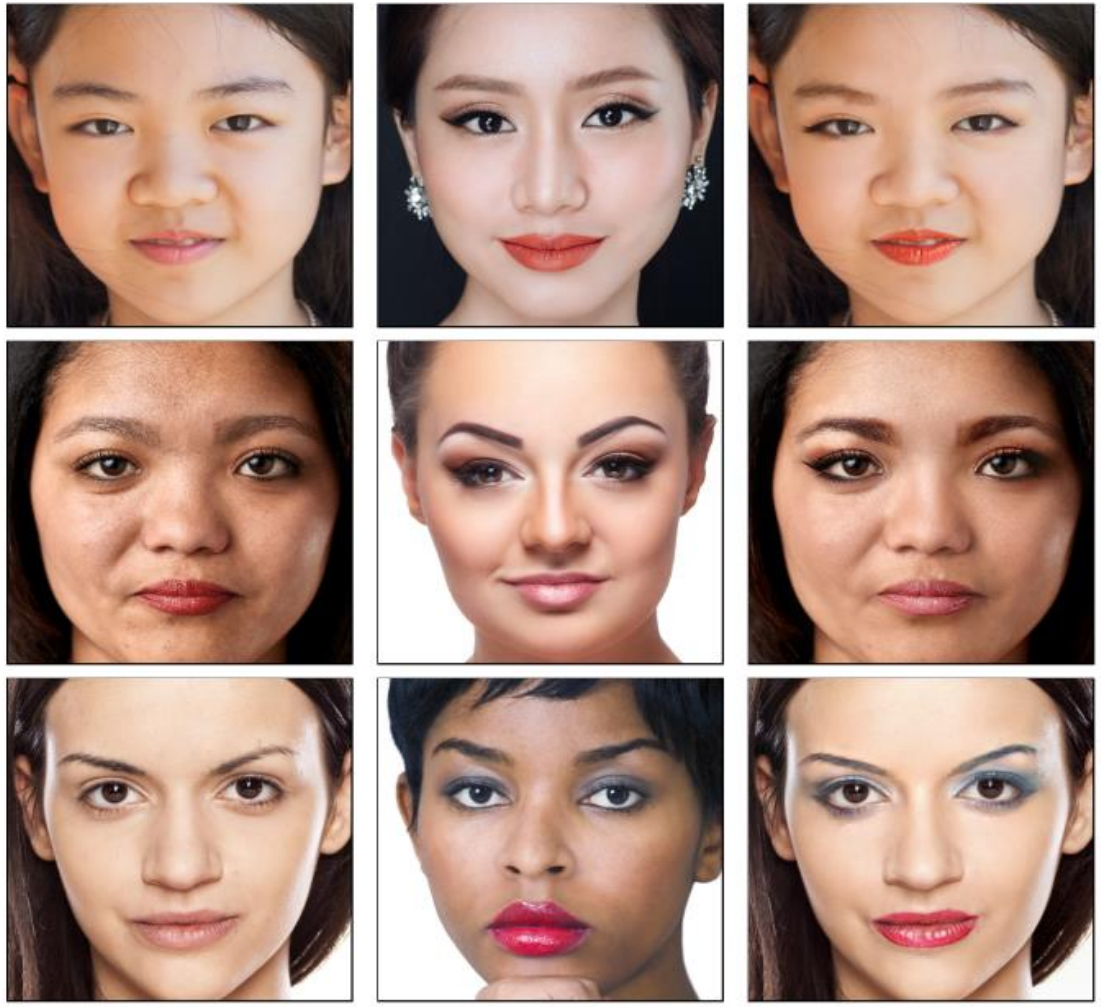
DRN > U-net

Results of the Lip and Eye Regions

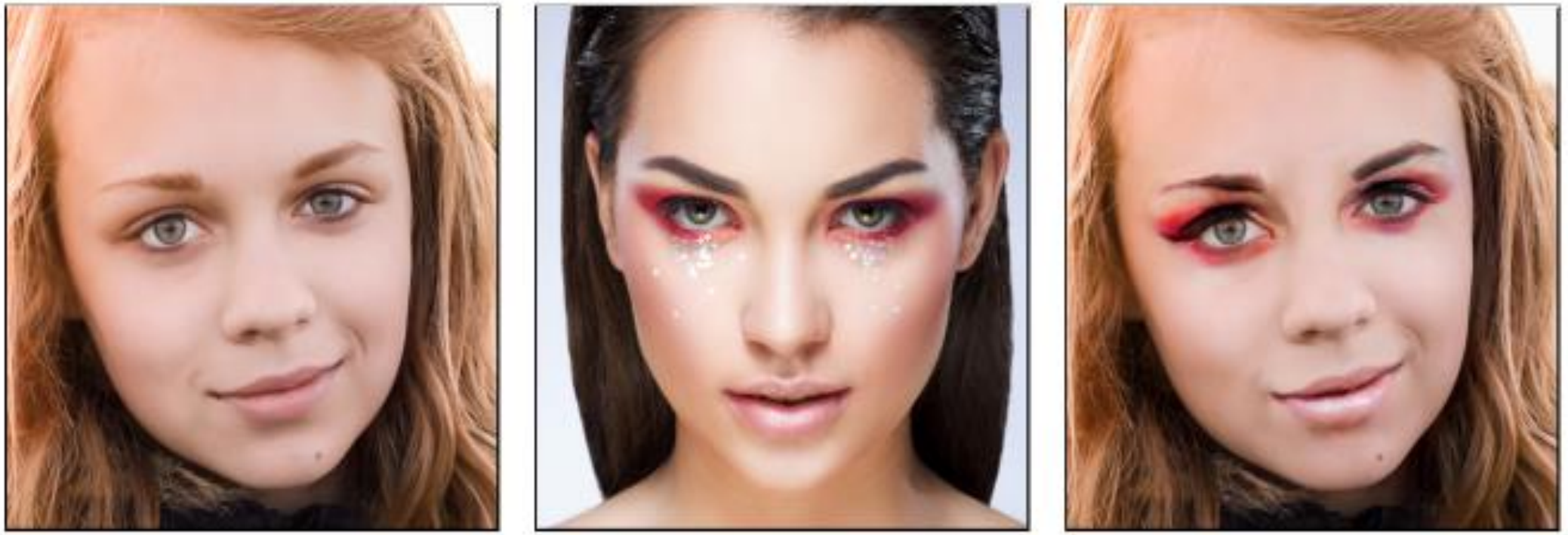
The generated lips exhibit plausible colors and inherit the shiny appearance from the reference.

When the distance between eyes and eyebrows or the orientation of eyebrows are very different between the source and the reference, it can still synthesize plausible result.

Results on the Entire Face



Limitations: Makeup Style Unseen during Training



Source Reference Our result

Makeup Transfer Comparison

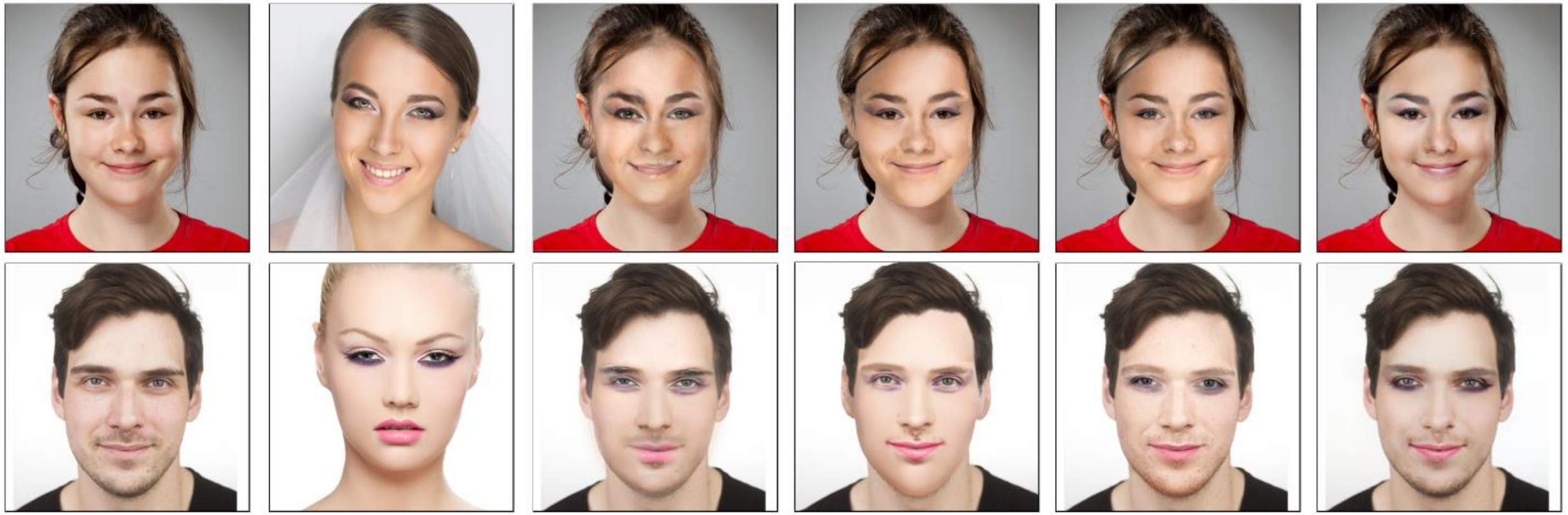


Figure 11: Makeup Transfer Results. We compare with makeup and portrait style transfer work [18, 5, 16].

Makeup Removal Comparison

Input Wang *et al*. 16 Ours

Figure 12: Demakup Results. We compare with makeup removal work by Wang et al. *[24]. Our demakeup network F can remove the detected makeup to virtually recover the original face.*

單元四、深度學習模型的設計與最佳化

Network Design and Optimization

- Question ?
  - How can the performance of a ConvNet representation be *maximized* for a particular target task?

- The ways a deep ConvNet representation can be learned and adjusted to allow better transfer learning from a source task producing a generic representation to a specific target task.

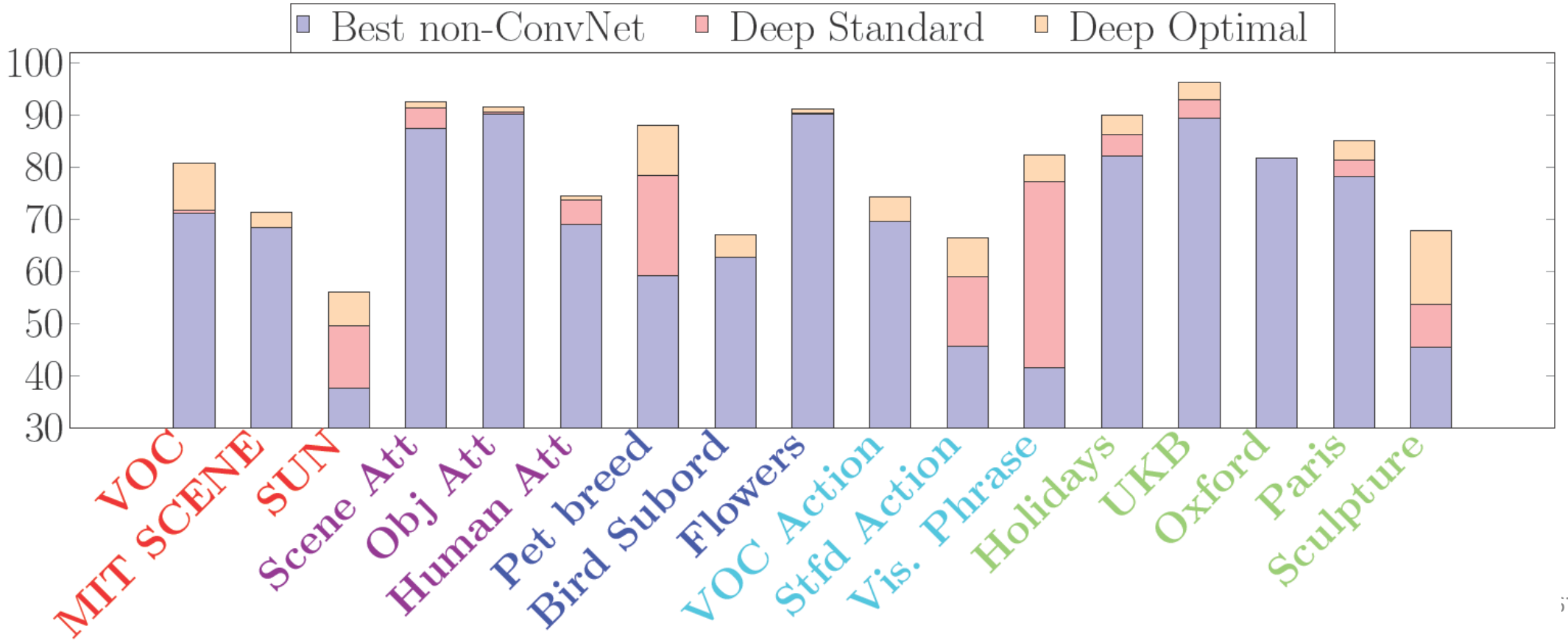
Network Design and Optimization

- Answer
  - Factors of transferability

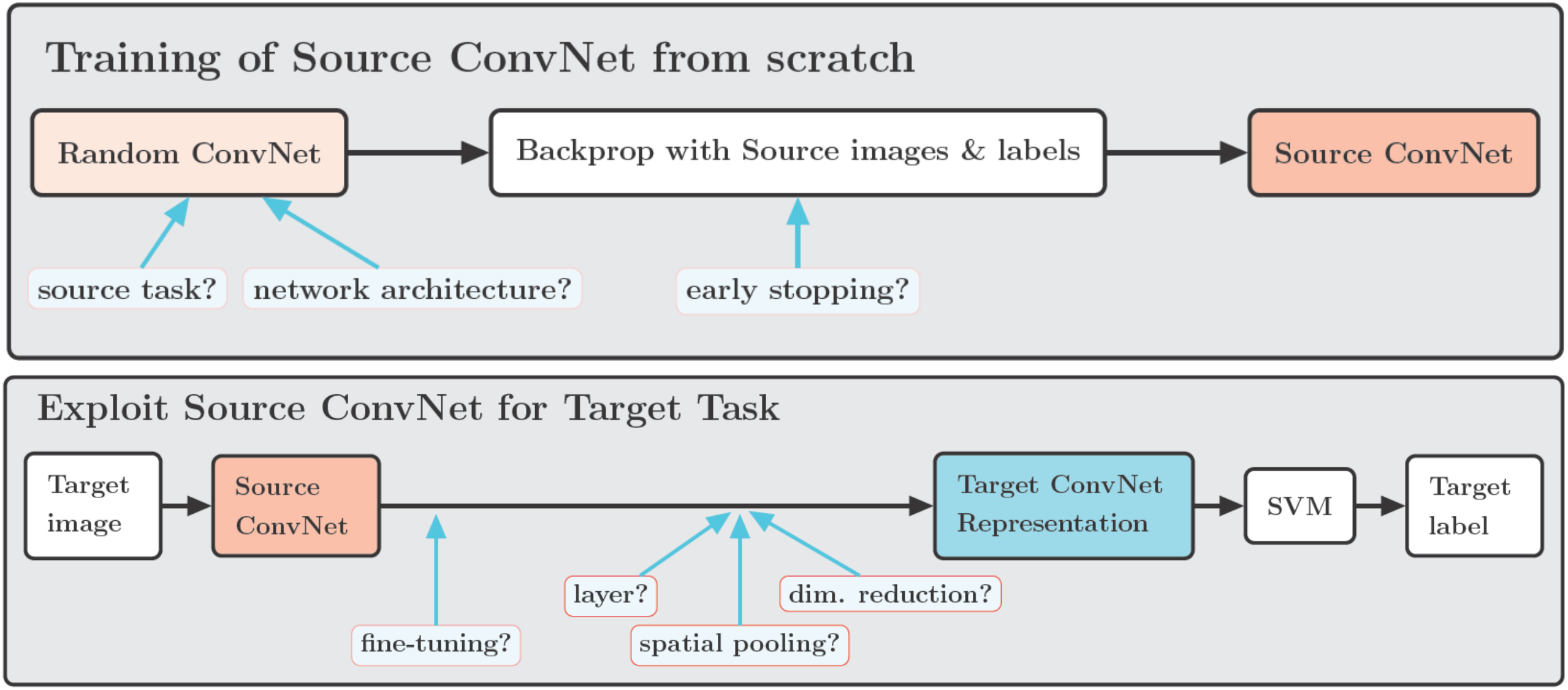
    - VGGNet, GoogleNet or AlexNet are usually trained by ImageNet dataset. How to exploit such model to your own research problem?

Transferability Factors

- Best non-ConvNet : non-ConvNet state of the art systems
- Deep Standard : standard ConvNet features with a linear SVM classifier
- Deep Optimal : optimizing the **transferability factors** for each task



Transferring ConvNet Representation



Range of Target Tasks

- The order these target tasks are based on their similarity to the source task of object image classification as defined by ILSVRC12.
- Instance retrieval as the least similar to the source task. Each task in this set has no explicit category information and is solved by explicit matching to exemplar images
- Compositional- how specific objects interact with one another. Play/hold.

Decreasing similarity to ImageNet →

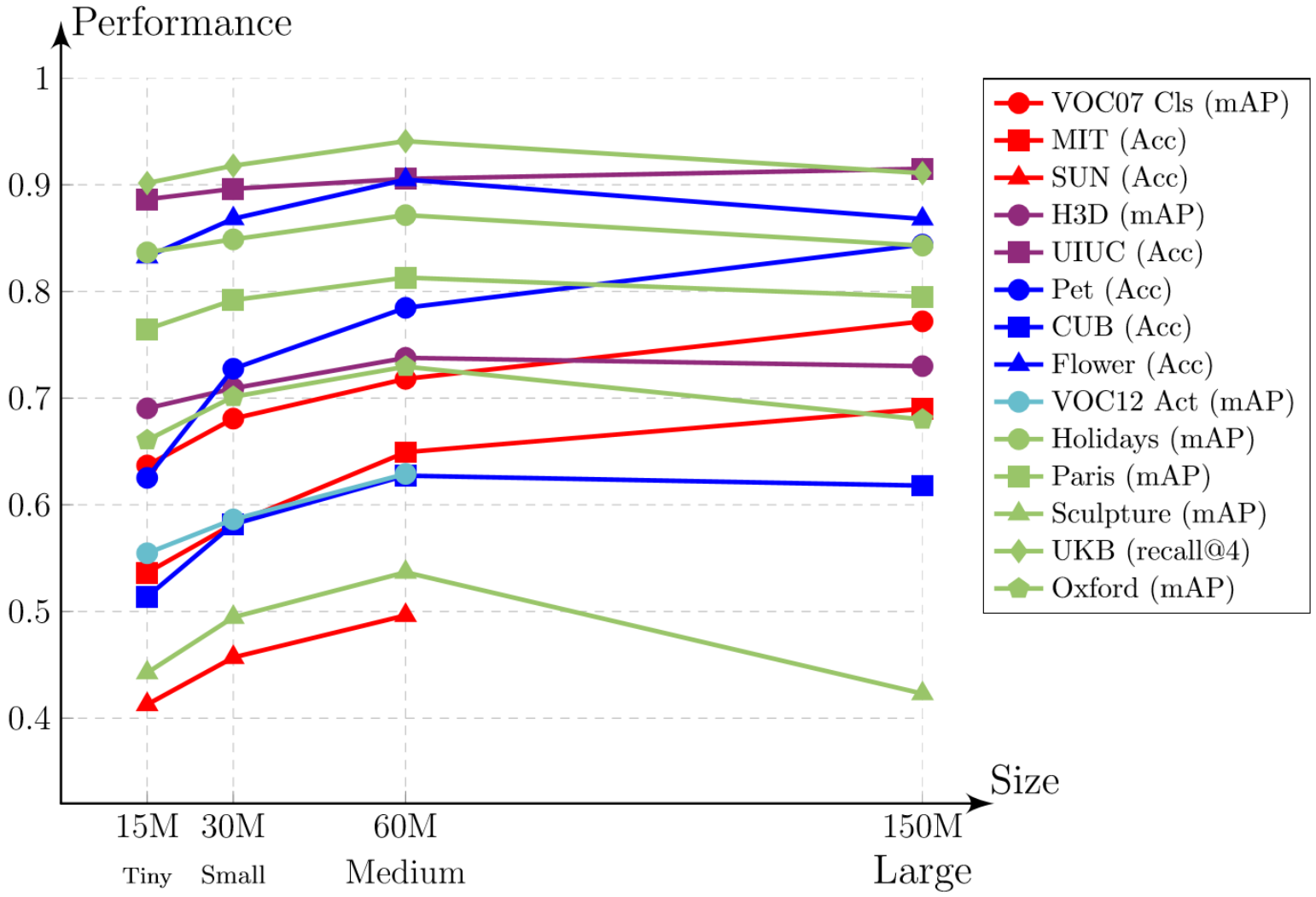
Decreasing similarity to ImageNet ↓	Image Classification	Attribute Detection	Fine-grained Recognition	Compositional	Instance Retrieval
	PASCAL VOC Object	H3D human attributes	Cat & Dog breeds	VOC Human Action	Holiday scenes
	MIT 67 Indoor Scenes	Object attributes	Bird subordinate	Stanford 40 Actions	Paris buildings
	SUN 397 Scene	SUN scene attributes	102 Flowers	Visual Phrases	Sculptures

Network Width

Convolutional layers				FC layers
Network	N_T	n_k per layer	output dim	n_h per layer
Tiny (A)	14M	(24, 64, 96, 96, 64)	6x6x64	(4096, 1024, 1000)
Small (C)	29M	(48, 128, 192, 192, 128)	6x6x128	(4096, 2048, 1000)
Medium(E)	59M	(96, 256, 384, 384, 256)	6x6x256	(4096, 4096, 1000)
Large (F)	138M	(96, 256, 512, 512, 1024, 1024)	5x5x1024	(4096, 4096, 1000)

- By keeping the network depth fixed, examine the impact of the network width on different tasks:
  - N_T : the total number of weights parameters in the network
  - n_k : the number of kernels at a convolutional layer
  - n_h : the number of nodes in a fully connected layer.

Results of Various Network Width



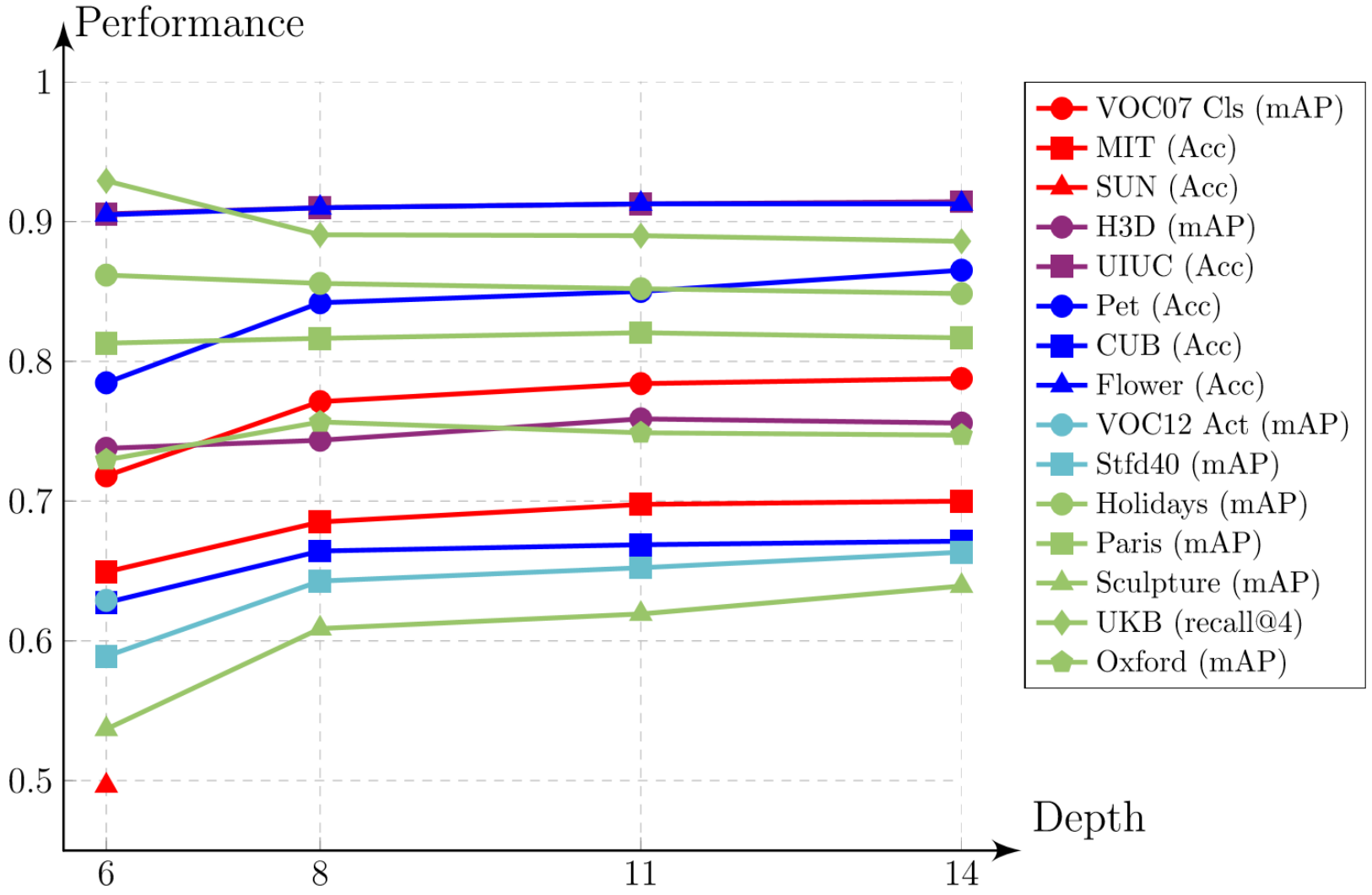
- 對於model的width設計來說參數量Large時(150M)，只有Target Dataset和Source Dataset很相像時，效果才會提升(**VOC07, MIT, SUN**)。
- 與Source Dataset較不相似時，適合中大型Width Network的(**Medium 59M**)。
- **Tiny和Small**的表現和Large的表現差異不大。因此在移動式裝置 或是需要簡化運算量時，Width不是一個很重要的參數，甚至可以設計的小一點，表現更好。

Network Depth

Convolutional layers				FC layers
Network	N_T	n_k per layer	output dim	n_h per layer
Deep8 (~E)	85M	(1×64, 1×128, 3×256)	8×8×256	(4096, 4096, 1000)
Deep11 (H)	86M	(1×64, 3×128, 4×256)	8×8×256	(4096, 4096, 1000)
Deep13 (I)	86M	(2×64, 4×128, 4×256)	8×8×256	(4096, 4096, 1000)
Deep16 (J)	87M	(2×64, 5×128, 6×256)	8×8×256	(4096, 4096, 1000)

- By keeping the number of network parameters fixed, examine the impact of the network depth on different tasks:
  - N_T : the total number of weights parameters in the network
  - n_k : the number of kernels at a convolutional layer
  - n_h : the number of nodes in a fully connected layer.

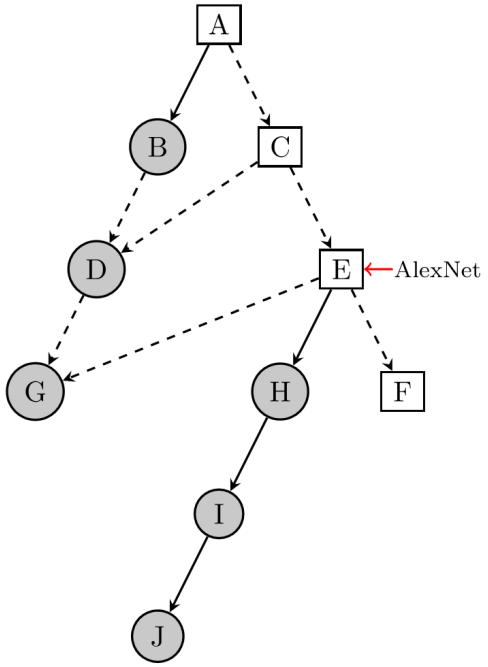
Results of Network Depth



- 不論和Source Dataset相似與否，Network的Depth影響全部Target Dataset的效果。
- 因此Network的Depth是設計的越深越好!

Width versus Depth

Convolutional layers				FC layers
Network	N_T	n_k per layer	output dim	n_h per layer
Deep Tiny(B)	85M	(13x64)	6x6x64	(4096, 1024, 1000)
Deep Small (D)	86M	(13x128)	6x6x128	(4096, 2048, 1000)
Deep Medium(G)	86M	(13x256)	6x6x256	(4096, 4096, 1000)

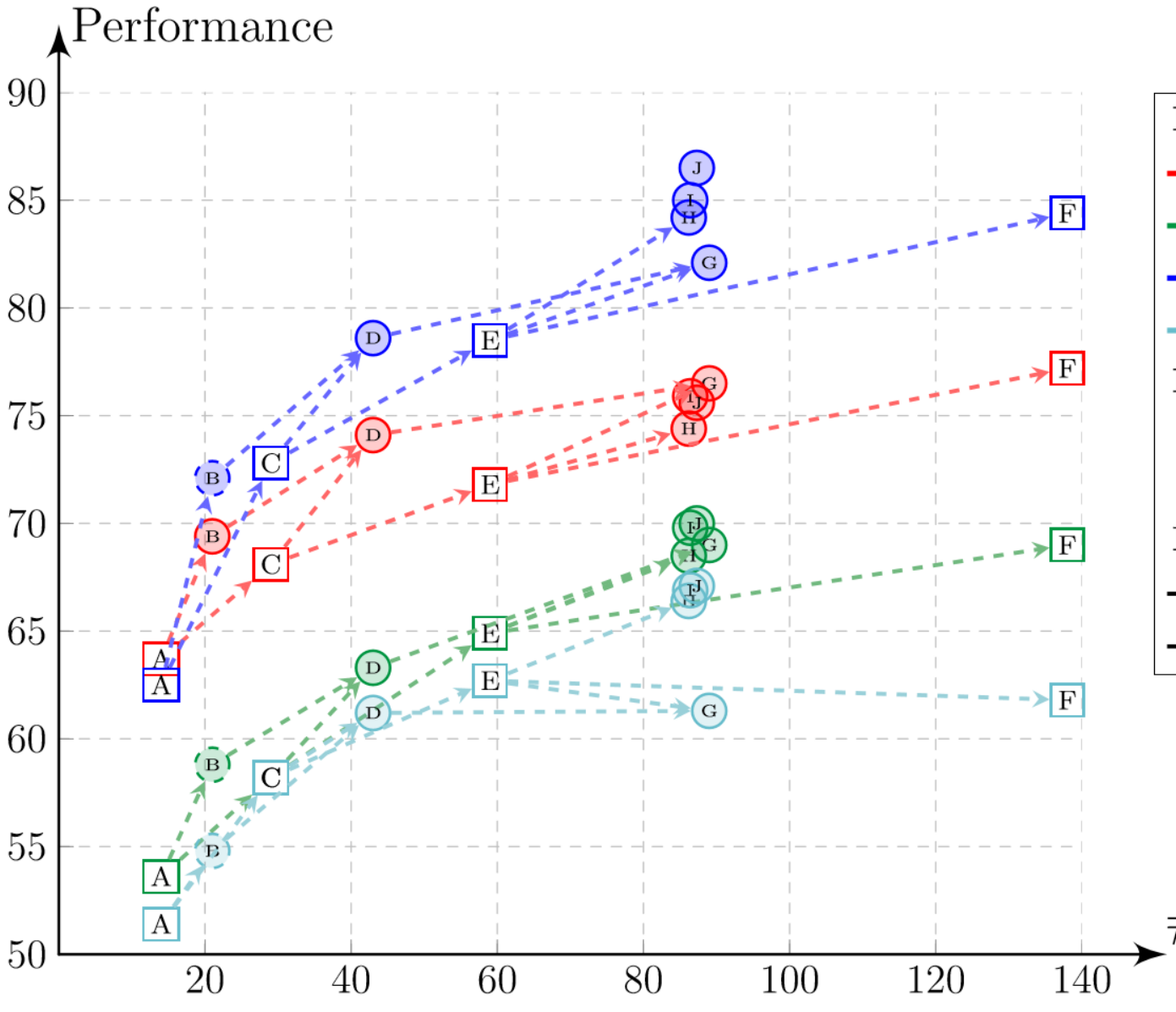


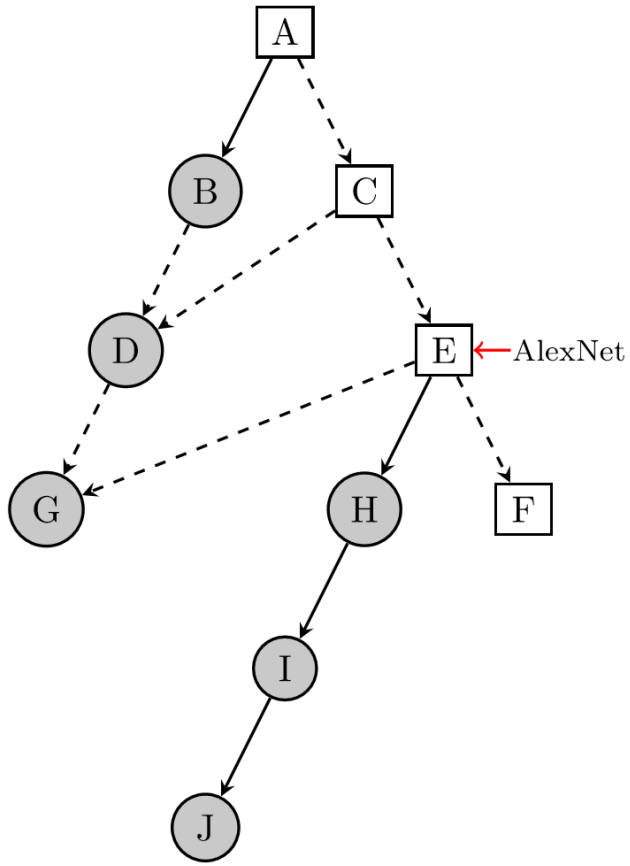
How network altered:

→ increase depth

- - → increase width

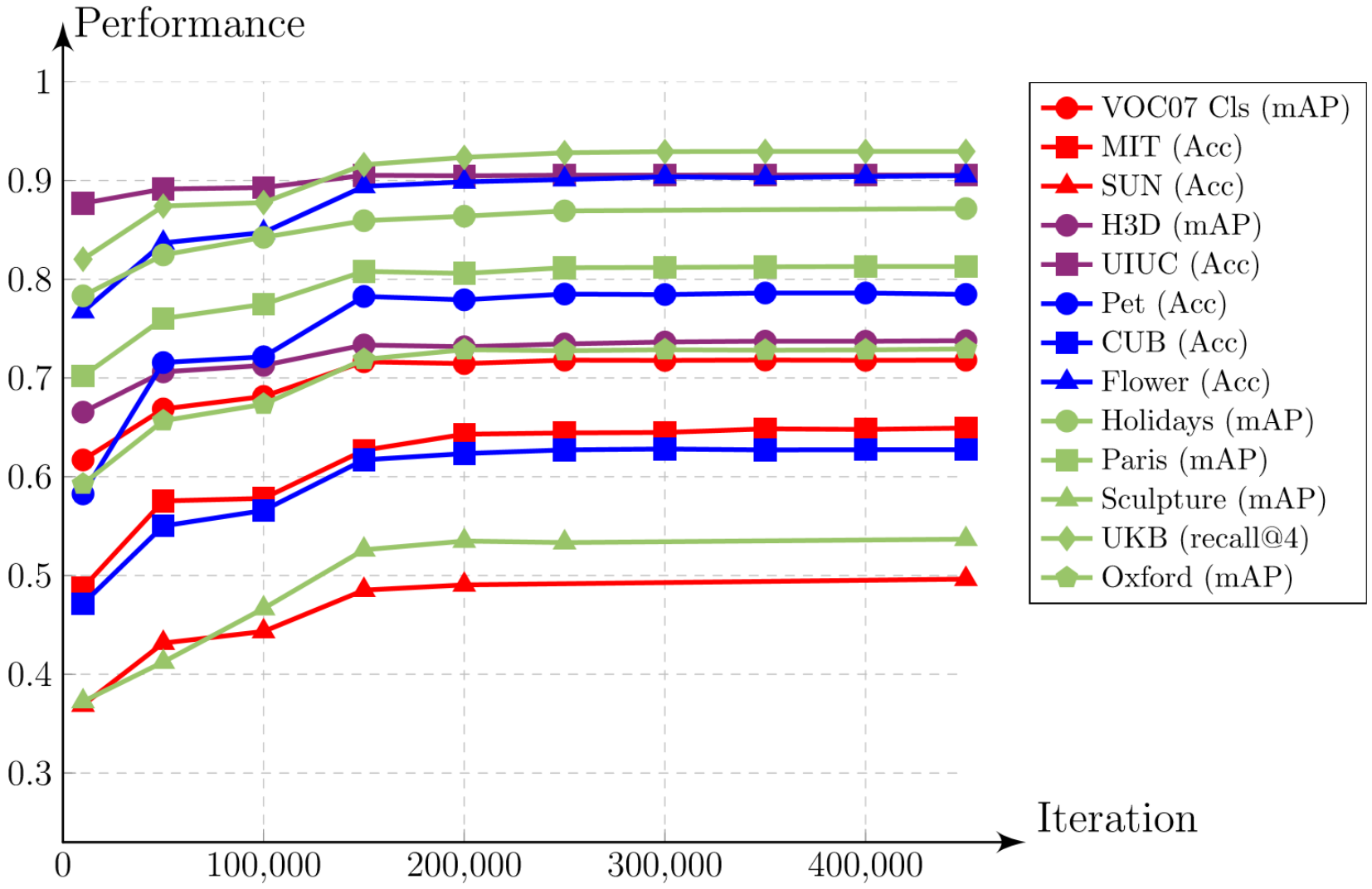
Results of Width versus Depth





- 觀察A-B與A-C，E-F與E-H，改變Depth得到的效果提升比改變Width來的明顯且重要。

Early Stopping

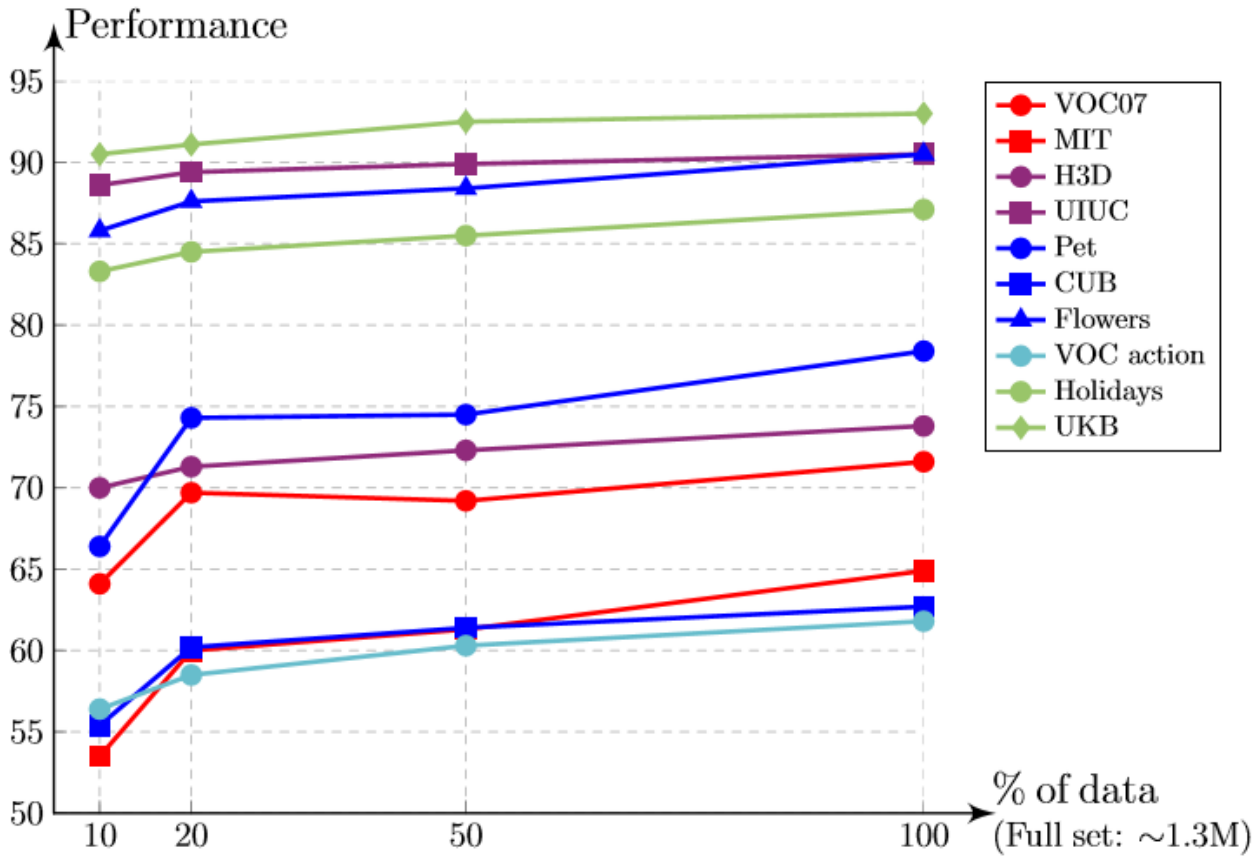


- 大約在200000的iteration(約40 epoch)就無法再有效的提升效果。
- 因此不需要提早停止，因為絕大部分的training的epoch都大約在這個區間。
- 從這個例子並未看到overfitting，因次並不代表一定不需要提早停止。

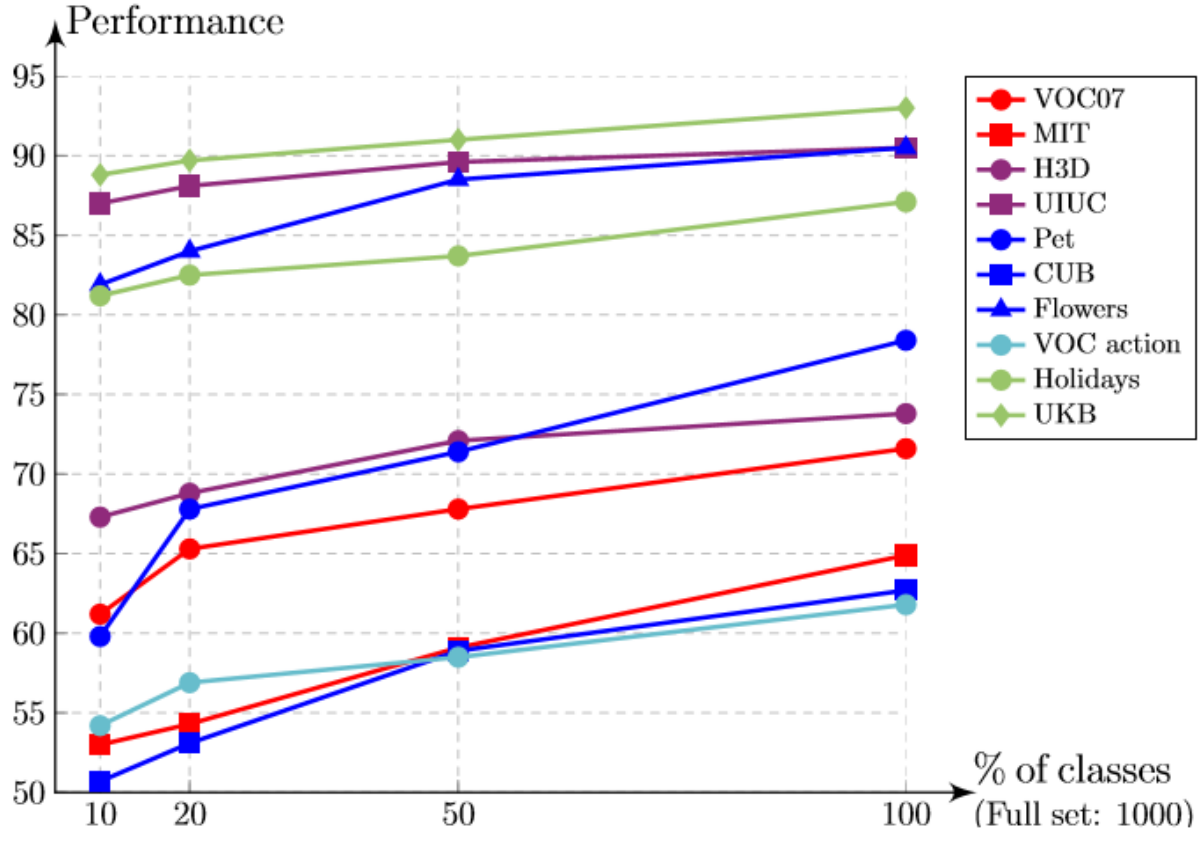
Data Density & Data Diversity

- Data Density – 單一class中資料量變多
- Data Diversity – class的數量變多

1. 圖左單一class的data數量越多越好
2. 圖右dataset的class種類數量越多越好



Increase datasets 10%, 20%, and 50% of the 1.3 million images in ILSVRC12.



Increased the number of classes from 100 to 1000, but kept the number of images the same as in ILSVRC 2012.

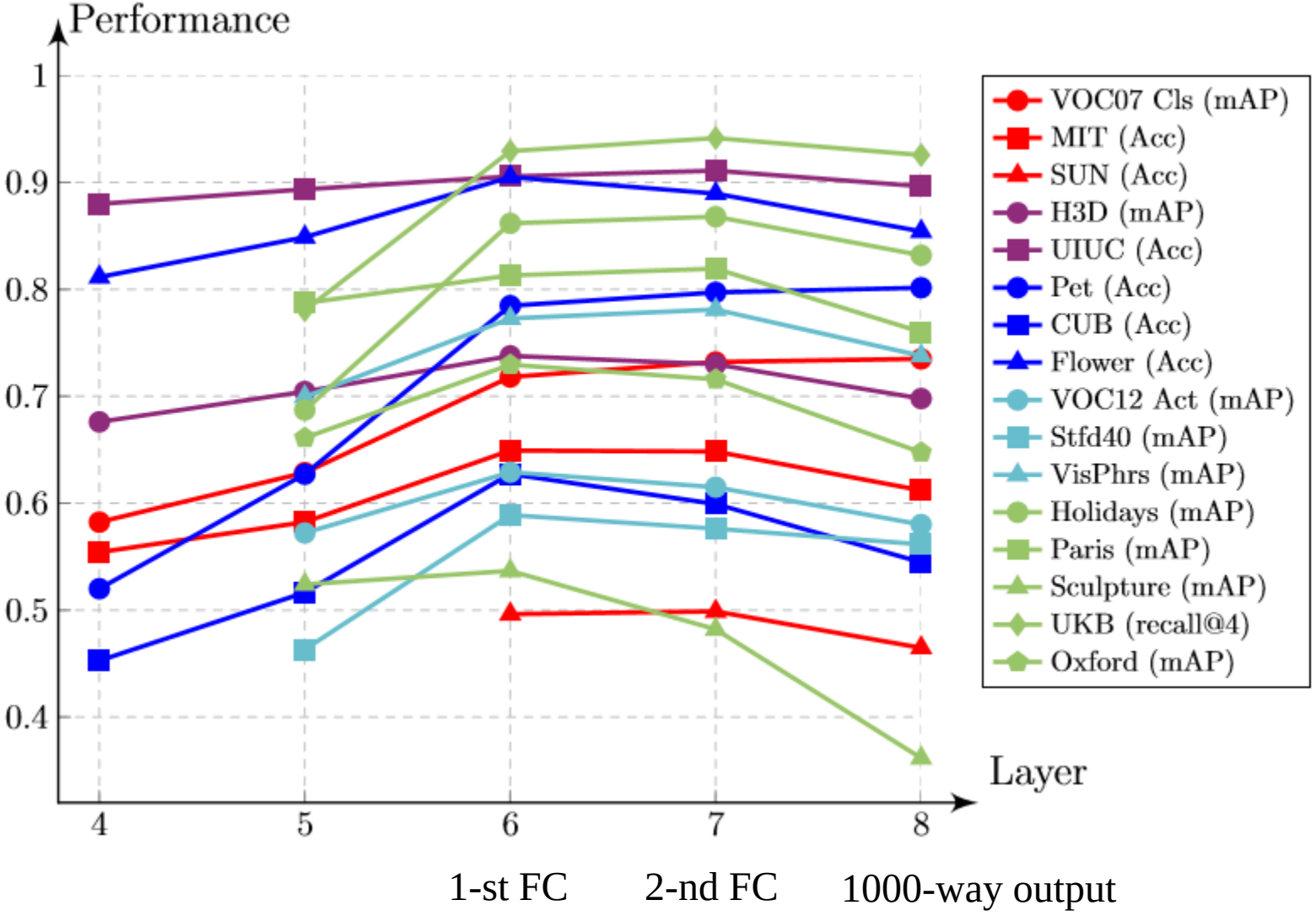
Source Task

	Image Classification			Attribute Detection		Fine-grained Recognition			Compositional	Instance Retrieval		
Source task	VOC07	MIT	SUN	H3D	UIUC	Pet	CUB	Flower	Stanf. Act40	Oxf.	Scul.	UKB
ImageNet	71.6	64.9	49.6	73.8	**90.4**	**78.4**	**62.7**	**90.5**	58.9	71.2	52.0	93.0
Places	68.5	69.3	55.7	68.0	88.8	49.9	42.2	82.4	53.0	70.0	44.2	88.7
Hybrid	72.7	69.6	56.0	72.6	90.2	72.4	58.3	89.4	58.2	**72.3**	52.3	92.2
Concat	**73.8**	**70.8**	**56.2**	**74.2**	**90.4**	75.6	60.3	90.2	**59.6**	72.1	**54.0**	**93.2**

- ImageNet: 1.3M張影像 (Object)
- Places: 3.5M張影像 (Scene)
- Hybrid: 4.8M張影像(ImageNet+Places)
- Concat: 將ImageNet和Places分別訓練完後的model，取得兩組feature concatenate作為特徵

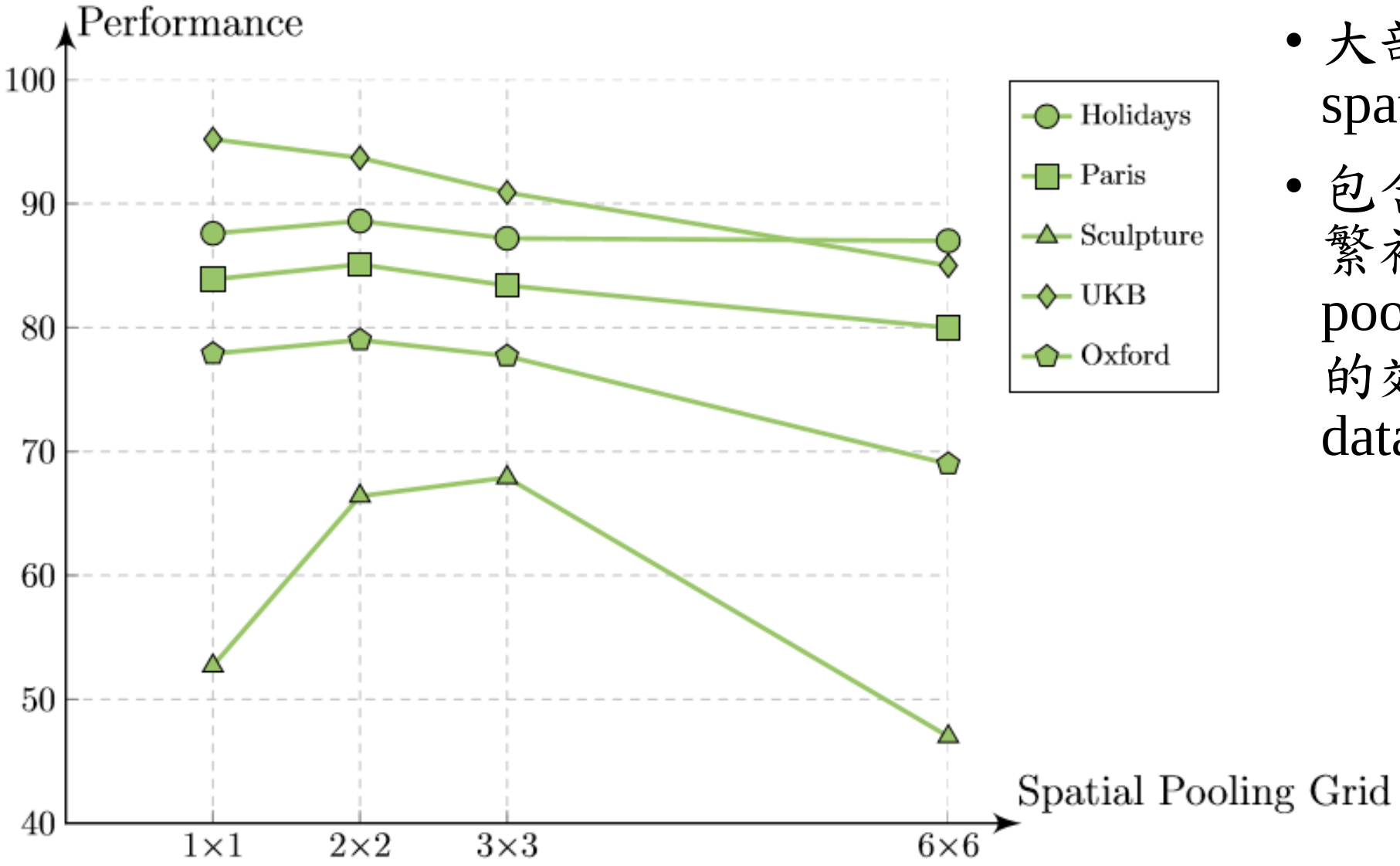
1. Hybrid特徵對於相似於Source Dataset的有較好的效果。對於不相似的dataset，因為Source Dataset的Label多樣性ImageNet的Label大於Places，增加類別數量是好事
2. Places雖然Dataset的影像數量是ImageNet的數量近3倍，但是效果都不理想，證實類別量的效果>資料量
3. Hybrid的結果不總是最好，可以歸咎於Places數量多，訓練產生了bias。當使用Concat設定，除了Fine-grained，效果皆得到提升。對Fine-grained而言，Places的資料並不相關，徒增加資料維度。

Network Layer as Feature Representation



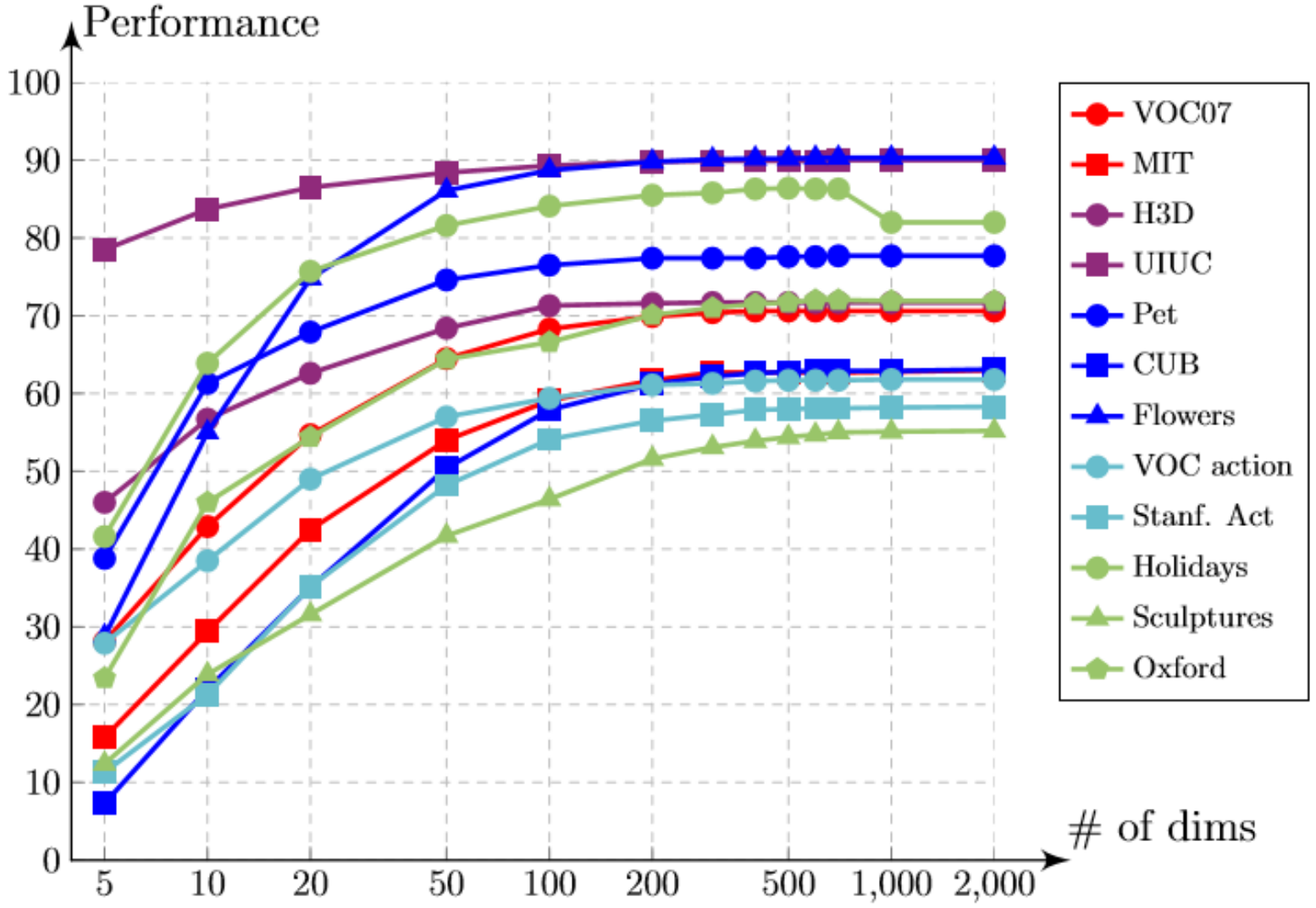
- 對Source Dataset最相似的幾個Target Dataset，取最後一層當Feature會最好。
- 對於和Source Dataset 稍微相似的則是取倒數第二層最好。
- 對於大多數其餘的Target Dataset來說，取自第六層的Feature，也就是第一層的FC來說效果會最好
- 建議全部取自第六層也就是第一層FC來當作Feature的效果會是普遍來說最好

Spatial Pooling



- 大部分的情形，2x2的spatial pooling就夠用了。
- 包含複雜細節或是紋路繁複的圖，需要大的pooling才能得到較好的效果，如Sculpture dataset.

Dimensionality Reduction



- 做PCA降維可以減少Feature的維度，來增加運算速度
- 幾乎所有的Target Dataset都在500維之後效果就沒太多提升
- 以原始4096維的Feature來說，降到500維會是平均表現維持較佳的維度

Fine-tuning

	Representation	MIT	CUB	Flower
Original FC7 feature	Medium FC7	65.9	62.9	90.4
FT: Find-tune feature	Medium FT	**66.3**	**66.4**	**91.4**

勝！

- Fine-tuning is done by initializing a network with weights optimized for ILSVRC12 and then updating the network's weights using the target task training set.
- The learning rate used for fine-tuning is typically set to be less than the initial learning rate used to optimize the ConvNet for ILSVRC12.
  - This ensures the features learnt from the larger dataset are not forgotten.

Increasing Training Data

- To measure the PASCAL VOC 2007 object detection, fine-tune the AlexNet network using samples from the Oxford Pet and Caltech-UCSD birds datasets.
- Even though there already exists a large number of samples for those classes in ImageNet (more than 100,000 dogs), adding around 3000 dogs from the Oxford Pet dataset improves performance significantly.

Representation	bird	cat	dog
ConvNet [14]	38.5	51.4	46.0
ConvNet-FT VOC [14]	50.0	60.7	56.1
ConvNet-FT VOC+CUB+Pet	**51.3**	**63.0**	**57.2**

Different Classifiers

Classifier	VOC07	MIT	Pet
Linear SVM	72.8	63.7	79.6
Logistic regression	71.7	63.6	79.5
Neural Network ReLU	-	61.3	81.9
Neural Network exponentiated	-	63.0	80.4
Perceptron (no ReLU, no dropout)	-	59.6	78.2

- 幾乎各個分類器在不同情況下得到的效果都不太一樣

Optimized Accuracy

	Image Classification			Attribute Detection			Fine-grained Recognition			Compositional			Instance Retrieval				
	VOC07	MIT	SUN	SunAtt	UIUC	H3D	Pet	CUB	Flower	VOCa.	Act40	Phrase	Holid.	UKB	Oxf.	Paris	Scul.
non-ConvNet	[41] 71.1	[27] 68.5	[47] 37.5	[32] 87.5	[45] 90.2	[53] 69.1	[31] 59.2	[12] 62.7	[21] 90.2	[30] 69.6	[49] 45.7	[36] 41.5	[43] 82.2	[54] 89.4	[43] **81.7**	[43] 78.2	[4] 45.4
Deep Standard	71.8	64.9	49.6	91.4	90.6	73.8	78.5	62.8	90.5	69.2	58.9	77.3	86.2	93.0	73.0	81.3	53.7
Deep Optimized [d]	**80.7**	**71.3**	**56.0**	**92.5**	**91.5**	**74.6**	**88.1**	**67.1**	**91.3**	**74.3**	**66.4**	**82.3**	**90.0**	**96.3**	79.0	**85.1**	**67.9**
Err. Reduction	32%	18%	13%	13%	10%	4%	45%	12%	8%	17%	18%	22%	28%	47%	22%	20%	31%
Source Task	ImgNet	Hybrid	Hybrid	Hybrid	ImgNet	ImgNet	ImgNet	ImgNet	ImgNet	ImgNet	ImgNet	ImgNet	Hybrid	ImgNet	ImgNet	ImgNet	ImgNet
Network Width	Medium	Medium	Medium	Medium	Large	Medium	Medium	Medium	Medium	Medium	Medium	Medium	Medium	Medium	Medium	Medium	Medium
Network Depth	16	8	8	8	8	16	16	16	16	16	16	16	8	8	16	16	16
Rep. Layer	last	last	last	last	2nd last	2nd last	2nd last	3rd last	3rd last	3rd last	3rd last	3rd last	4th last	4th last	4th last	4th last	4th last
PCA	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
Pooling	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	1×1	1×1	2×2	2×2	3×3

How to Design and Optimize your Network?

Factor	Target task: Source task ImageNet	...	FineGrained recognition	...	Instance retrieval
Early stopping	Don't do it				
Network depth	As deep as possible				
Network width	Wider		Moderately wide →		
Diversity/Density	More classes better than more images per class				
Fine-tuning	Yes, more improvement with more labelled data				
Dim. reduction	Original dim		Reduced dim →		
Rep. layer	Later layers		Earlier layers →		

Summary

- Factors of transferability
  - Range of Target Tasks –the relationships of pre-train/target model
  - Network Width versus Network Depth
  - Data Density versus Data Diversity
  - Feature Representation – which layer? hybrid? concatenate?
  - Spatial Pooling
  - Dimension Reduction
  - Early Stopping

2019年深度學習新趨勢 – Learning to Learn

- Learning to Learn (Meta Learning)
  - 深度學習的九陽神功
  - **學會如何學習 – 從過去學習的經驗，快速學習適用於新任務的深度學習模型。**
- Meta Learning 的範圍
  - 訓練超參數Hyper Parameters：包括Learning rate，Batch Size，input size等等目前要人為設定的參數
  - 神經網絡的結構
  - 神經網絡的初始化
  - 優化器Optimizer的選擇。比如SGD，Adam，RMSProp
  - 依據不同任務，學習損失函數的定義。

Reference & Slides Courtesy

- Deep Residual Learning for Image Recognition, Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun, CVPR 2016.
- ICML 2016 Tutorial – Deep Residual Network, Kaiming He.
- Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, CVPR 2017.
- Factors of Transferability for a Generic ConvNet Representation, Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, Stefan Carlsson, 2015 TPAMI.
- Based on notes from Andrej Karpathy, Fei-Fei Li, Justin Johnson.
- Slides of ‘Recurrent Neural Network Introduction’, Yun-Zhing Lu.
- Slides of ‘Long Short-Term Memory’, Akshay Sood.
- Online Course Materials (機器學習基石), 林軒田.