Machine Learning

Lecture 6
Neural Network

Chen-Kuo Chiang (江 振 國)
ckchiang@cs.ccu.edu.tw
中正大學 資訊工程學系

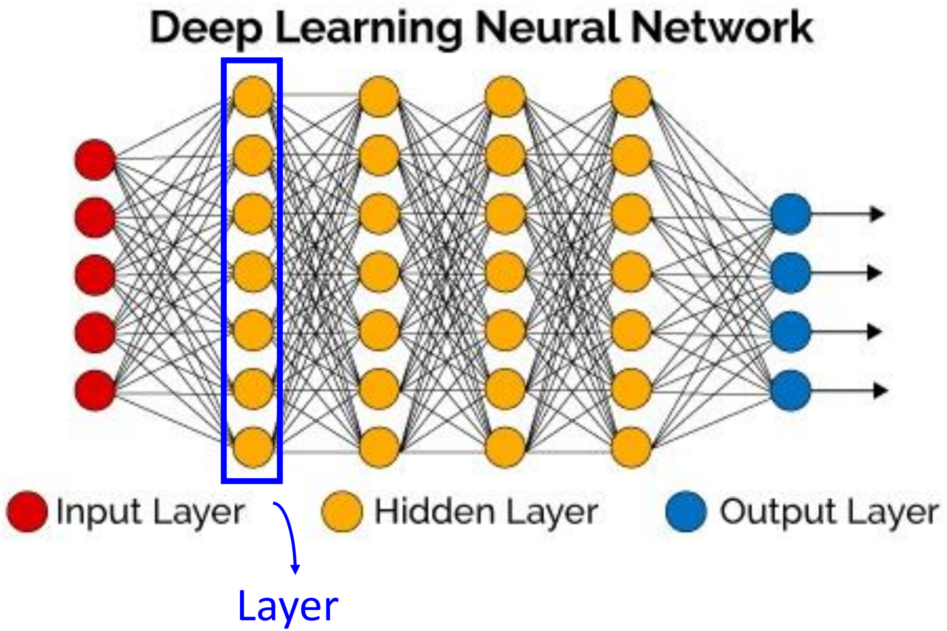
The Storyline

- Distilling Implicit Features: Extraction Models

Neural Network

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

深度學習模型-由節點擴展到層



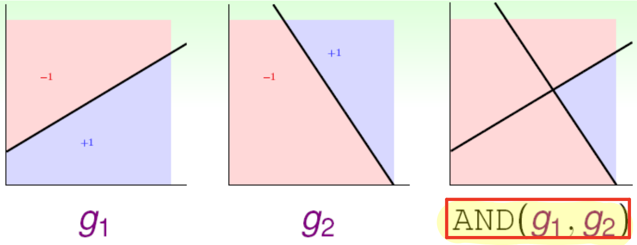
Recall : Vector Form of Perceptron Hypothesis

特徵x權重

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

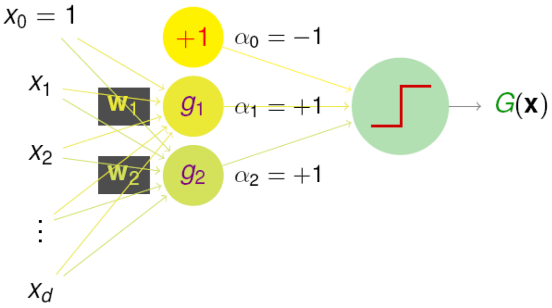
- each 'tall' $\mathbf{w}$ represents a hypothesis h & is multiplied with 'tall' $\mathbf{x}$ —**will use tall versions to simplify notation**

Logic Operations with Aggregation



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

Replace $h(x)$ by $g(x)$ …

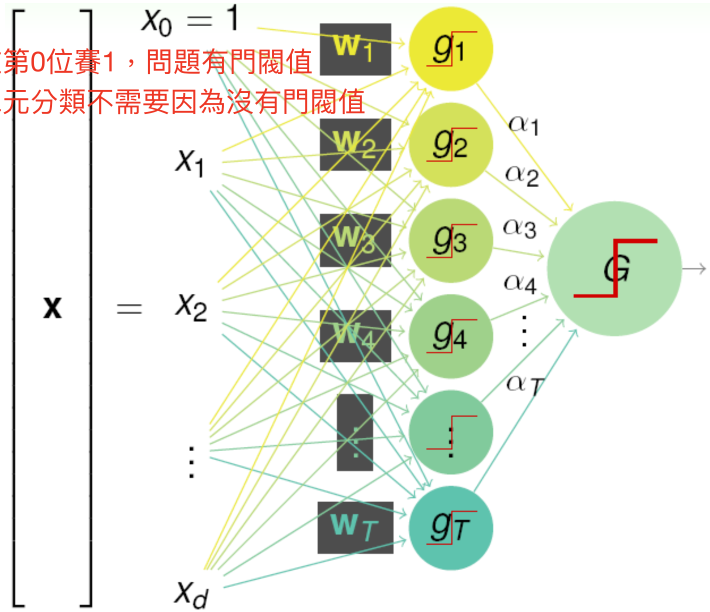


$G(\mathbf{x}) = \text{sign}\,(-1 + g_1(\mathbf{x}) + g_2(\mathbf{x}))$

- $g_1(\mathbf{x}) = g_2(\mathbf{x}) = +1$ (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



在第0位賽1，問題有門閥值
二元分類不需要因為沒有門閥值

權重

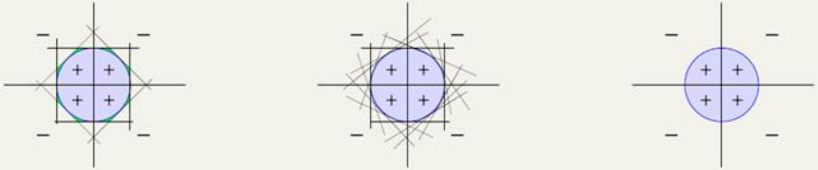
$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \underbrace{\text{sign}\left(\mathbf{w}_t^T \mathbf{x}\right)}_{g_t(\mathbf{x})}\right)$$

Input

- 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**?

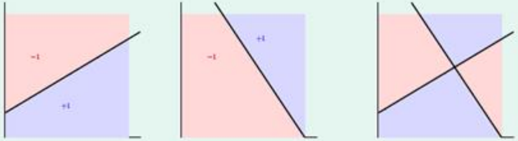
Powerfulness and Limitation



8 perceptrons | 16 perceptrons | target boundary

線性夠多

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



g_1 | g_2 | $\mathrm{XOR}(g_1, g_2)$

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

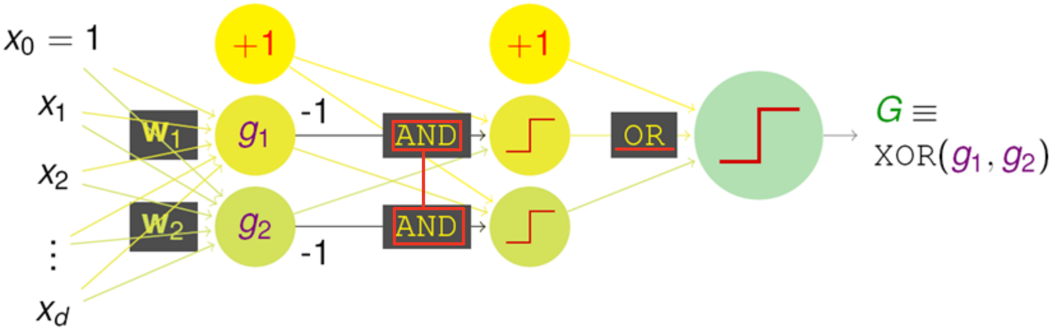
how to implement $\mathrm{XOR}(g_1, g_2)$?

Multi-Layer Perceptrons (MLP): Basic Neural Network

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

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

	$g_2(x)$		
$g_1(x)$	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

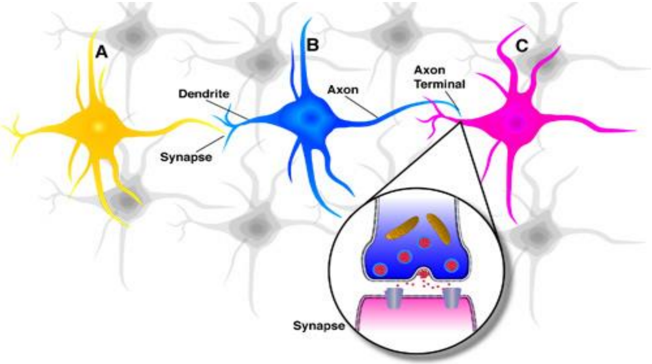
$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=0}^{2} \alpha_t g_t(\mathbf{x})\right)$$ to implement $\text{OR}(g_1, g_2)$?

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		

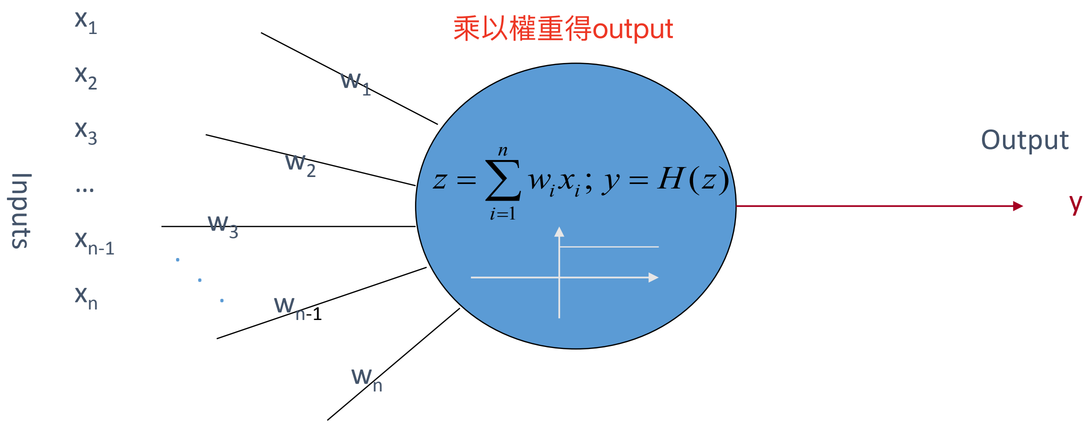
Neural Network - Biological Inspiration

- 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)



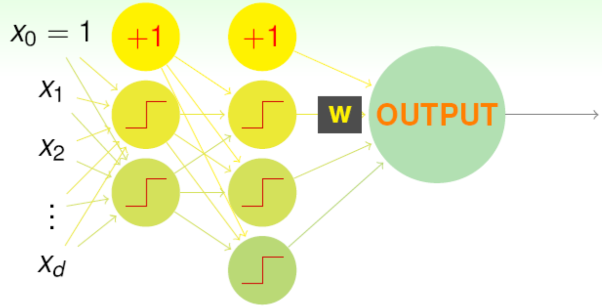
Artificial Neurons

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



The McCullogh-Pitts model

Neural Network Hypothesis: Output



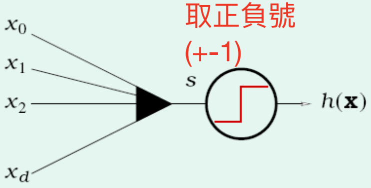
- **OUTPUT**: simply a **linear model** with $s = \mathbf{w}^T \phi^{(2)}(\phi^{(1)}(\mathbf{x}))$
- any linear model can be used—**remember? :-)**

linear classification

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

取正負號

(+-1)

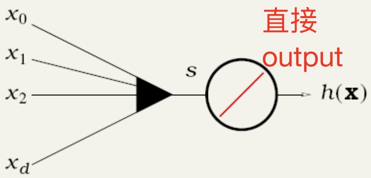


err = 0/1

linear regression

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

想知道期末成績直接output

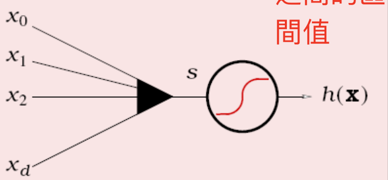


err = squared

logistic regression

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

得到0~1之間的區間值



err = cross-entropy

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

Neural Network Hypothesis: Transformation

- : **transformation** function of score (signal) s

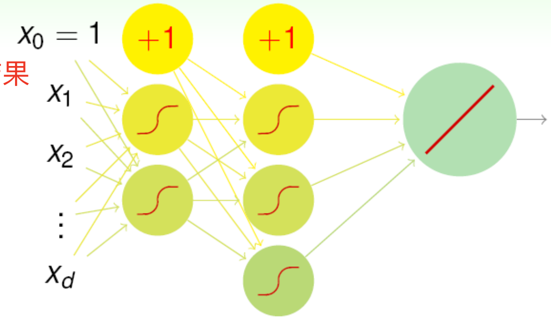
1. 多次線性轉換結果相當於一次

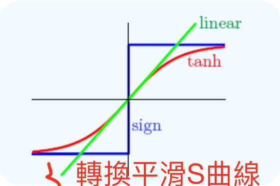
- any transformation?
  - : whole network linear & thus **less useful**
  - : discrete & thus **hard to optimize** for $\mathbf{w}$

2. 加入非線性轉換函數

2. 難做最佳化（不可微分）

- popular choice of transformation: $\stackrel{\text{}}{=} \tanh(s)$
  - 'analog' approximation of : **easier to optimize**
  - somewhat **closer to biological** neuron
  - **not that new! :-)**



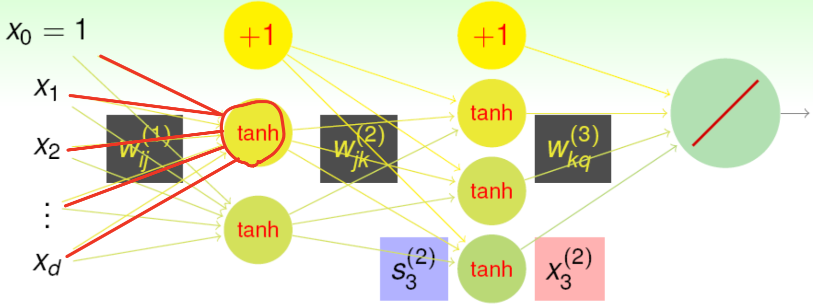


3. 轉換平滑S曲線（可微分）

$$\tanh(s) = \boxed{\begin{aligned} &\frac{\exp(s) - \exp(-s)}{\exp(s) + \exp(-s)} \\ &= 2\theta(2s) - 1 \end{aligned}}$$

will discuss with tanh as transformation function

Neural Network Hypothesis



D-2-3-1NNet

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

input

index

link

第L層節點權重

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

與前面連結相
乘與相加

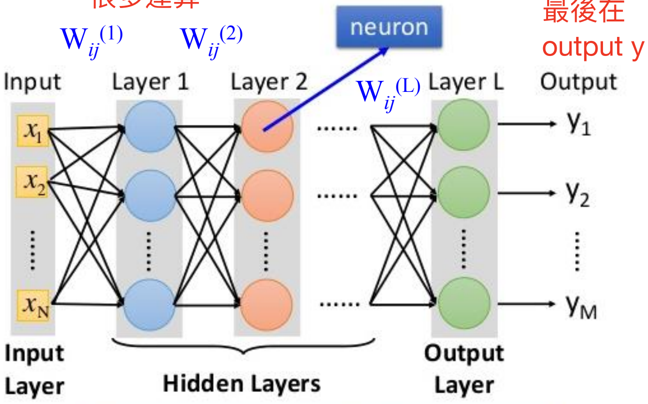
$$\text{transformed } x_j^{(\ell)} = \begin{cases} \tanh\left(s_j^{(\ell)}\right) & \text{if } \ell < L \\ s_j^{(\ell)} & \text{if } \ell = L \end{cases}$$

在非最後一層在做
（activation
function）

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

What is Neural Network?

很多運算



Deep means many hidden layers

學每一個node的權重值

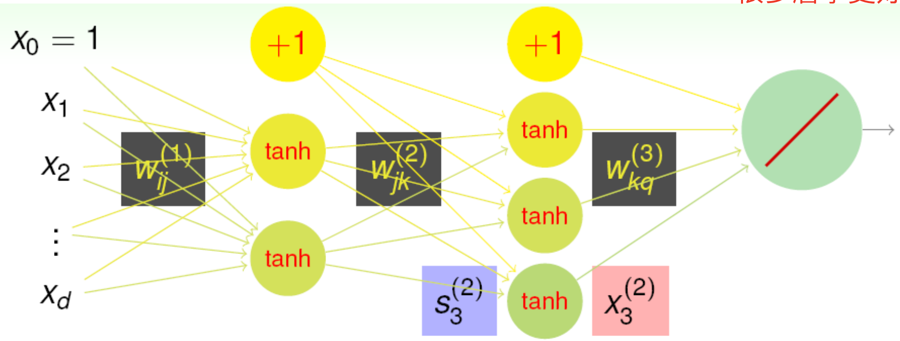
- 神經網路就是$\boxed{\text{一堆函數集}}$
  - 丟進去一堆數值x，整個網路運算出一個最好的解y出來
  - 假設某個簡單函數 $f(x)$ = 無能
  - 神經網路是一個複雜的函數

- 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??

很多node的二元分類，
很多層學更好



- 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

X會與每層相層與相加，做向量內稽，很多相同方向（符合學習狀況低中高資料表示），得到高分的layer。

特徵表示法

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

+

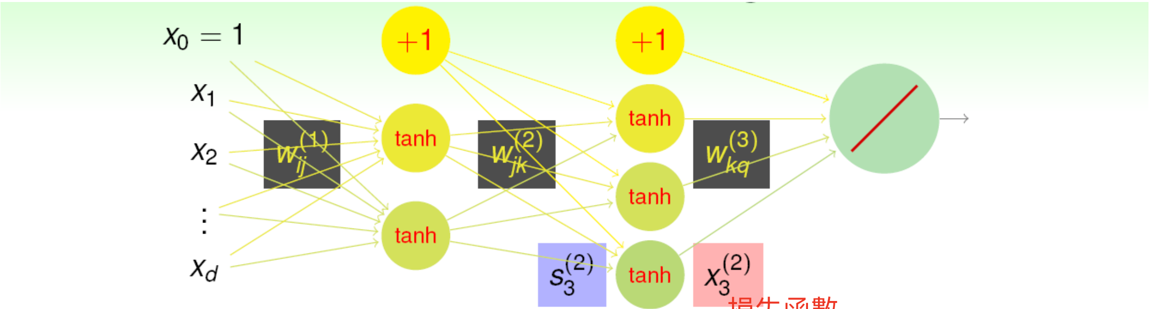
3 $\boxed{1 = 4}$

5 $\boxed{4 \times 5 = 20}$

1 5+1 = 6

$\boxed{20+6 = 26}$

How to Learn the Weights?



損失函數

- 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**
- 標準 模型答案
  let $e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2$:
  can apply **(stochastic) GD** after computing $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$! 求微分的最小值

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

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

最後output
節點分數

相乘相加而來

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

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

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

最後一層
前一層

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

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

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

$$\begin{aligned} \delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}} &= \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} \\ &= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\tanh'\left(s_j^{(\ell)}\right)\right) \end{aligned}$$

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

Backpropagation (Backprop) Algorithm

Backprop on NNet

initialize all weights $w_{ij}^{(\ell)}$ 初始權重值
for $t = 0, 1, \ldots, T$

1. stochastic: randomly pick $n \in \{1, 2, \cdots, N\}$ 慢慢更新
2. forward: compute all $x_i^{(\ell)}$ with $\mathbf{x}^{(0)} = \mathbf{x}_n$ 算出第一層node權重
3. backward: compute all $\delta_j^{(\ell)}$ subject to $\mathbf{x}^{(0)} = \mathbf{x}_n$ 得到最後一層，反推前一層
4. gradient descent: $w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} - \eta x_i^{(\ell-1)} \delta_j^{(\ell)}$ 最後結果

return $g_{\text{NNET}}(\mathbf{x}) = \left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_i\right)\right)\right)$ 不斷調整越來越接近

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

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

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

把error最小化

sometimes 1 to 3 is (parallelly) done many times and 每次一筆太慢
average$(x_i^{(\ell-1)} \delta_j^{(\ell)})$ taken for update in 4, called **mini-batch**

Batch 32筆平均更新Weight

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

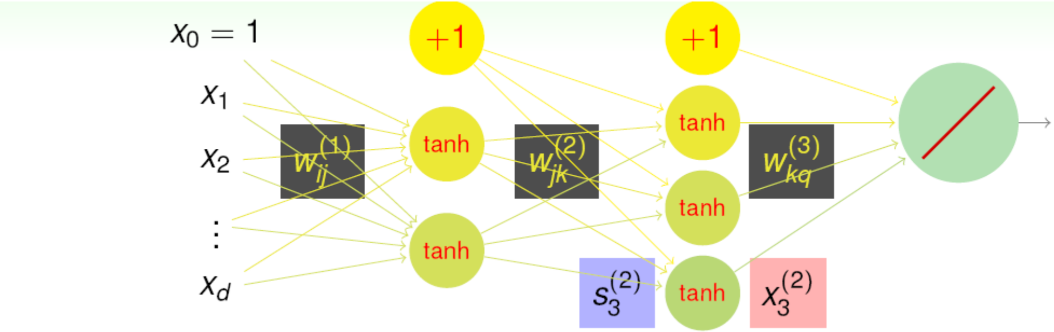
Neural Network Optimization

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

- generally **non-convex** when multiple hidden layers
  - not easy to reach **global minimum** 沒有那麼容易
  - GD/SGD with **backprop** only gives **local minimum** 透過微分
- different initial $w_{ij}^{(\ell)} \Longrightarrow$ different **local minimum** 不同w
  - somewhat '**sensitive**' to initial weights 透過不同的pretraining得到較好local minimum
  - **large weights** $\Longrightarrow$ **saturate** (small gradient)
  - advice: try **some random** & **small** ones

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

Deep Model Design



- each layer: **pattern feature extracted** from data, **remember? :-)**
- how many neurons? how many layers? 考量那些因素實驗結果較好
  —more generally, **what structure?**
  - subjectively, **your design!** 較主觀
  - objectively, **validation, maybe?**

structural decisions:
key issue for applying NNet

Shallow versus Deep Neural Networks

shallow: few (hidden) layers; deep: many layers

Shallow NNet

- more **efficient** to train (○)
- **simpler** structural decisions (○)
- theoretically **powerful enough** (○) 足夠擁有

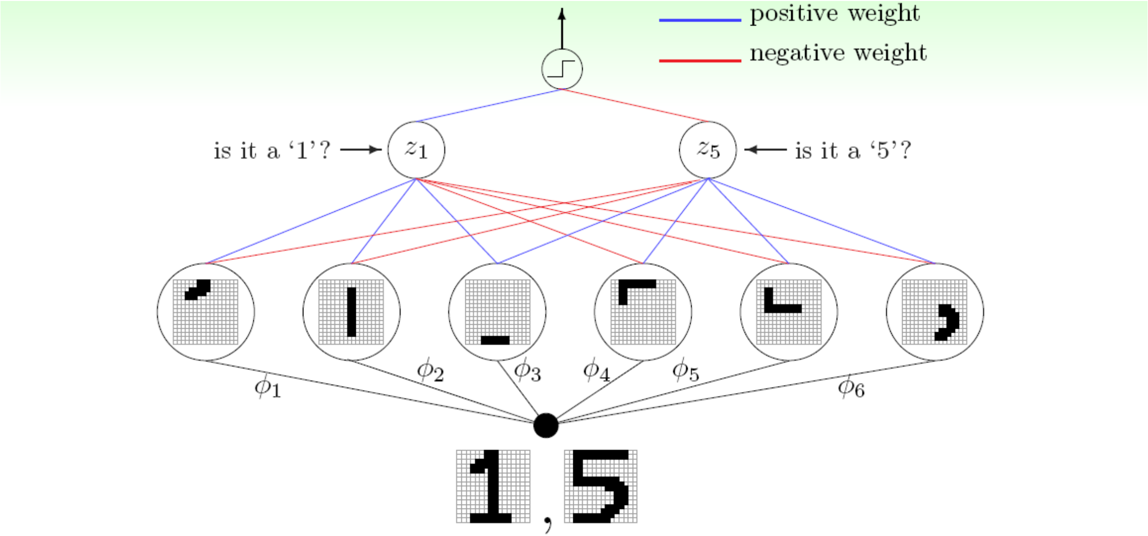
Deep NNet

- **challenging** to train (×)
- **sophisticated** structural decisions (×)
- **'arbitrarily' powerful** (○)
- more **'meaningful'?** (see next slide) 有意義

deep NNet (**deep learning**)
gaining attention in recent years

辨識與分類問題的
負擔分出去

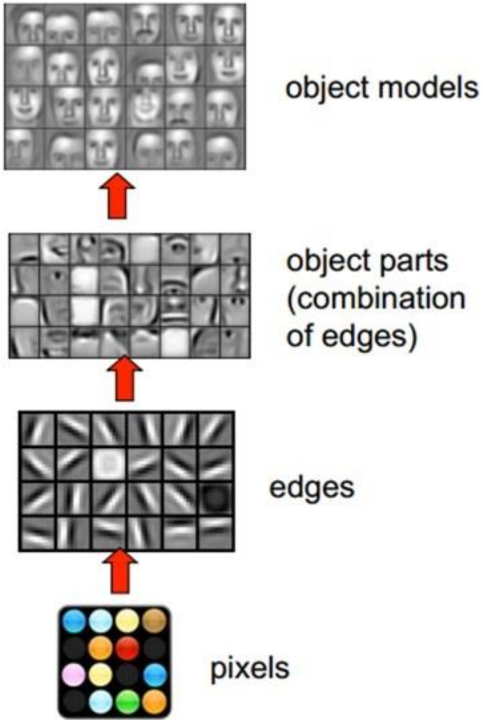
Meaningfulness of Deep Learning

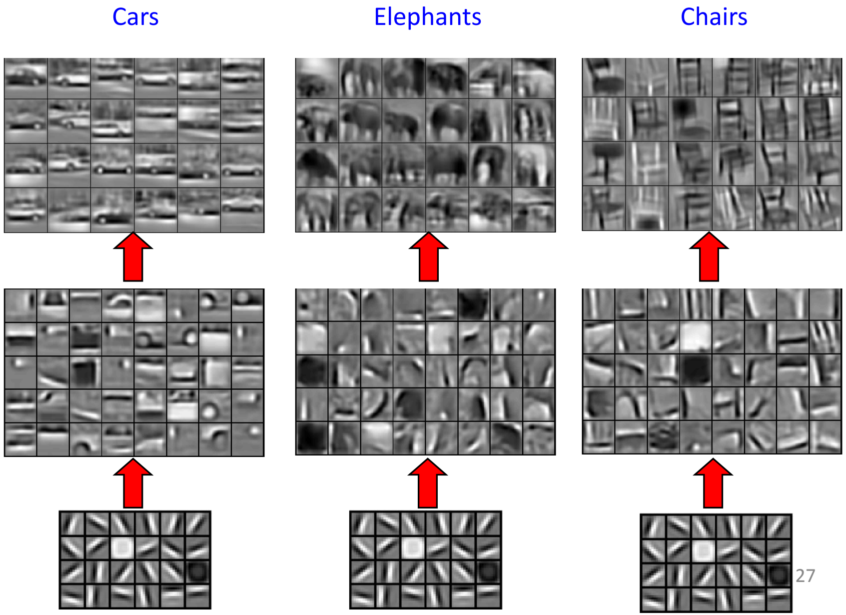


- **'less burden'** for each layer: **simple** to **complex** features
- natural for **difficult** learning task with **raw features**, like **vision**

deep NNet: currently popular in
vision/speech/...

Simple to Complex Features





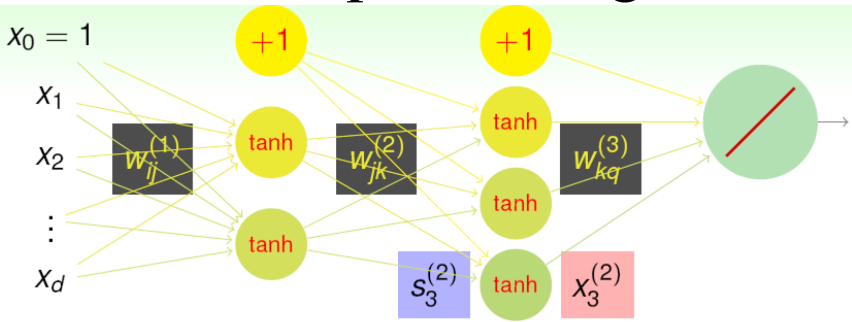
Challenges and Key Techniques for Deep Learning

- difficult **structural decisions**: 問題 複雜的結構設計
  - subjective with **domain knowledge**: like **convolutional NNet** for images 與domain有關

模型複雜度高
- high **model complexity**: 如何train得出模型?
  - no big worries if **big enough data** 資料夠多/可以訓練出來
  - **regularization** towards noise-tolerant: like 複雜度太高
    - **dropout** (tolerant when network corrupted) 隨機將node設為0
    - **denoising** (tolerant when input corrupted)
- hard **optimization problem**: 如何train得好模型? 困難最佳化問題
  - **careful initialization** to avoid bad local minimum: called **pre-training** 初始值設定的好
- huge **computational complexity** (worsen with **big data**): 運算複雜度太高
  - novel hardware/architecture: like **mini-batch with GPU**

IMHO, careful $\boxed{\textbf{regularization}}$ and $\boxed{\textbf{initialization}}$ are key techniques

Regularization in Deep Learning



watch out for overfitting, remember? :-)

high **model complexity**: **regularization** needed 讓權重個數降低

- structural decisions/**constraints**
- weight decay or weight elimination **regularizers**
- **early stopping** 在平原期一半就停下來

next: another **regularization** technique

Summary

- Distilling Implicit Features: Extraction Models

Neural Network

- Motivation

multi-layer for power with biological inspirations

- Neural Network Hypothesis

layered pattern extraction until linear hypothesis 抓出layer特徵

- Neural Network Learning

backprop to compute gradient efficiently

- Optimization and Regularization Forward/backward與regularzation

優化 正規化