

核心：找到收錄選取的最小值
首號映射 → 意義

單詞的意義是由上下文決定

- * { One-Hot Encoding: 單詞 → vector/array (其中只有個位置為 1, 其他是 0)
 - 於監督學習中的 label, 還能從 label mapping 到 meaning
- * Bag-of-Words (BoW) model: 不同 sentence 和 document 之間的關係
 - 將搜集到的單詞包中分類統計, 1 個 document 的統計量也可形成 vector
 - TF-IDF: 比較每個單詞在文章的重要性, 但要詞袋的大才能看出來, 用於大文件/論文
 - 每的 word 都是 1D dimension
- * Neural Network Based (NNLM): One-Hot Encoding 和 BoW 與的 vector 很少假設正獨立
相似意義的字彼此可能 related, 可用規律判斷, 但上工之是重疊
 - 少量的詞會歸類, 人工重新標定 dimension → Distributed Representation, 以表達間存在的性質另外 dimension (word embedding)
 - ∴ 在對不同的問題, dimension sets 和意思映射的詞也不一樣
- * GAN: Auto-encoder Network
讓計算機在大量文本中找到 dimension 規律 (非監督, 無 label)
 - 文本 → embedding → hidden layer → z node decoder → output 文本 (與原文本比較)
- * NNLM: Neural Network Language Model: 依據上文來推測下文, 経 One-Hot Encoding 到 Hidden Layer 作為處理, 再由 softmax 顯示該文單詞。
算是監督學習
 - feature, 文字 label
 - 主要是從上文分析下文, Word embedding 是副產物
- * Word2Vec: Word embedding 的 model
 - Continuous Bag-of-Words (CBOW) —— 從單詞中找出規律 (單詞的意義是由上下文決定)
 - Continuous Skip-Gram (Skip-Gram) —— 給定單詞推出上下文
 - 都是將單詞經過 embedding, 再進入 Hidden layer 分析 → 再由 softmax 進行轉化
 - 但 training too expensive → Huffman encoding → Hierarchical Softmax
 - Negative Sampling: 為了解決複雜的 softmax 最終是改變一個 word 的 embedding, 訓練過程添加不相干的單詞, 方便一起訓練, 加快次數

※ Softmax: 決策函數, 常用在分類問題 (單詞推測就是分類問題)

※ GloVe: Global Vector: 結構與 Word2Vec 相近, 並非利用上文找規律, 而是像 BM25 分析整篇文章, 所以訓練都是 offline training

※ ELMo: Embedding from Language Model 解決 NLP 的多義詞問題 (靜態 embedding 定量 → 電動 embedding 變量)

一個詞根據上文可能同時有多個 embedding 輸出, 再選最合適的 embedding
單獨訓練後會產生 3 個 embedding, 再依需要改變 embedding 的校準組成。

↗ 需要 2 layers 產生 3 個 embedding
 ↘ Pre-training: 得到 3 個 embedding (v_1, v_2, v_3)
 ↗ Fine-tuning: 分別訓練权重 w_1, w_2, w_3
 $N_1v_1 + N_2v_2 + N_3v_3$

* BERT (Bidirectional Encoder Representation from Transformer)

→ ELMo 一样也是双向，但把 ELMo 中用的 LSTM & RNN 换成 Transformer

← 从上下文推定單詞: Word2Vec
双層双向: ELMo

以 Transformer 替代 RNN: GPT (Generative Pre-training)

① Masked Language Model (遮掩詞模型), inspired by Cloze Task (完形填空)

遮掩一些詞，再重新通过上下文推断 output，目的是避免訓練权重集中在某幾個單詞，分散权重

BERT → vector → vector 算

② 特殊 tokens:

torch.tensor 包含单一数据类型的矩阵矩阵

[CLS]: 每個 sequence 的第一個 token

[SEP]: 多句子之間的分割 token

[UNK]: 未知 token 及代替在 dict 出現的 token

[PAD]: zero padding - 用來的長度不同 sequence 補齊做 batch 算 —— refer to "Transformer"

[MASK]: 未知遮罩，在 pre-training 階段用到

* Text Corpora:

1. Monolingual corpus: 單語料庫

只包含一種語言的文本，該語料庫通常被標記為語言的一部分，且被廣泛地用於高反復用的任務。用來檢查單詞的正確方法或查找最自然的單詞組合

2. Parallel corpus: 平行語料庫

由 2 個 Monolingual corpus 組成，一種語言是另一種語言的翻譯，用戶可以搜索一種語言的單詞或短語，將和另一種語言的 example 一起顯示

3. Multilingual corpus:

4. Comparable corpus: 兩個以上的 Monolingual corpus，文本句主題相關，但來自同一個原始 corpus

5. Learner corpus: 由 language learners 生成，對於學習外語出現的錯誤和問題

b. Diachronic corpus: 歷時語料庫，包含不同時期的文本語料，用於研究語言的變化和發展

* Transformer (具前饋層和自注意力層)

① sequence 可以是句子、段落或篇章

1. 用注意机制、全连接層處理文本

2. self-attention: 搜尋 sequence 內部，

分析內不同位置的注意力机制並能計算 → Transformer 可以減少運算量、Multi-head Attention
內的表徵 → 用於搜尋序列內部的隱藏關係

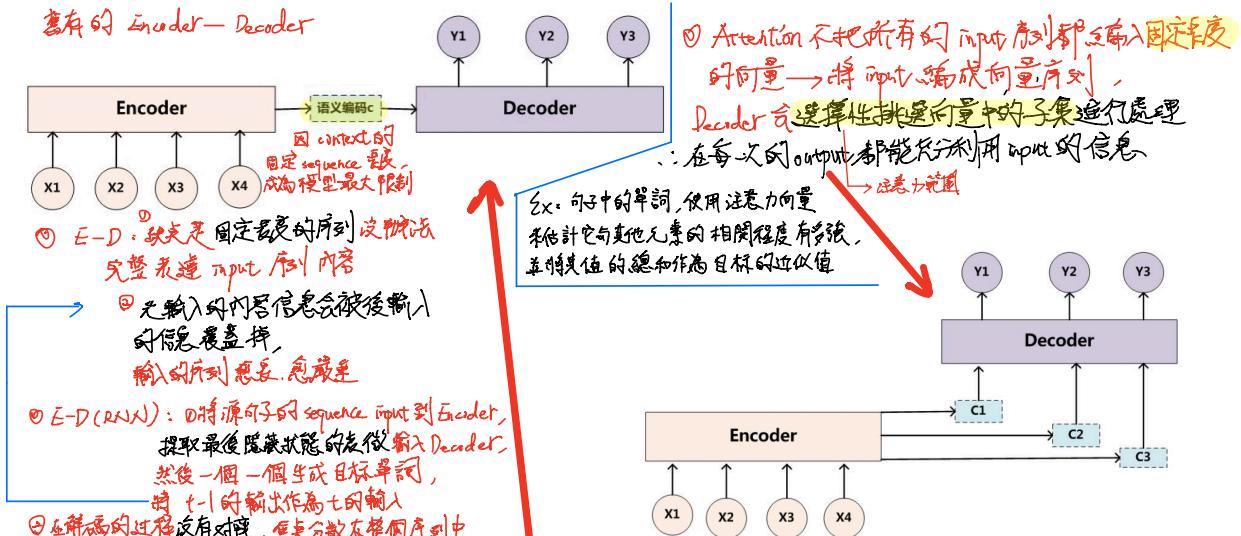
3. 取消用 CNN/RNN 的序列對接，只用自注意力 → 以線性變換取代舊有的深度網路提取特徵

4. Encoder-Decoder: x_1, x_2, \dots, x_m \rightarrow 過饋串後 $C = F(x_1, x_2, \dots, x_m) \rightarrow$ 輸出序列
(循環神經網路 RNN)

中間變量 C

$y_i = g(C, x_i, x_{i-1})$





4.1 Seq2Seq：字符串序列到序列，另一个字符串序列 (和用 Encoder-Decoder)

- ① sequence 到 sequence 之間的映射問題 (不規則長度)
② input sequence → **固定長度的** 中間 context → output 且為 Sequence
(向量)



⑤ 在 **RNN Encoder-Decoder** 中，用 RNN 模擬大腦讀入，以確定 **問題** 的問題模擬記憶，再以另一個 RNN 模擬大腦得到答案

The diagram shows the mapping from an input sequence (x_1, x_2, \dots, x_m) to an output sequence $(y_1, y_2, \dots, y_{m-1})$ via an intermediate variable $C = F(x_1, x_2, \dots, x_m)$. The input sequence is labeled "Encode" and the output sequence is labeled "Decode". The intermediate variable C is labeled "中間變量 C".

- ④ 在 backpropagation 的過程中以 RNN 實現，但訓練時會遇到 gradient exploding 或 gradient vanishing，所以實際應用以改良後的 LSTM RNN 或 GRU RNN.....

輸入 sequence 的最終表示為最後一個 word 的 hidden state vector

- ⑤ Decode 的過程是以 Encoder 生成的 hidden state vector 做輸入解碼生成目標 sequence，最常見用 Recurrent Neural Network Language Model (RNNLM)

→ 如果要改進 Seq2Seq 結構，只能從 Encoder 的所有隱藏層狀態、解決 context 的限制問題（而不是只用 Encoder 的最後一層隱藏層狀態）

⑤ 在 Encoder-Decoder 做出 神經翻譯 和 對齊聯合學習：Attention

生成 Target sequence 的每個詞群，E-D 中間的 context 向量是源序列透過五個隱藏層的加權和輸出。

不像係統直接以序列的最後一個隱藏狀態輸出，能保證在 Decode 過程中，源 sequence 對目前 Decod 的貢獻度不一樣

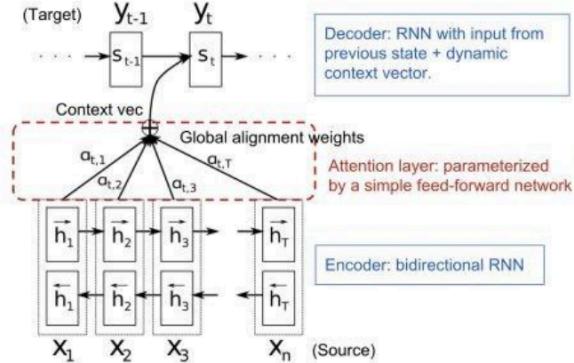
→更貼近人類翻譯

→ Source sequence 的每個詞和 Target sequence 的每個詞建立關聯，翻譯詞的語意向量成為 input sequence 透過 Encoder 後的加权和，可看出翻譯時 Source 的每個詞在目前要翻譯詞的重要性分布

被叫/接收，启动向发送者回传消息。

5. Transformer 解碼 | 編碼

- * $x = [x_1, x_2, \dots, x_n], y = [y_1, y_2, \dots, y_m]$, s_t 是 Decoder 在時間 t 的隱藏層狀態, c_i 是輸出的內容向量
- * $s_t = f(s_{t-1}, y_{t-1}, c_t), t=1\dots m$, y_t 是目標輸出
- * $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$ 加熱係數(注意力) 在以前只有一個內容向量 C
- * C_t 是 Encoder 時, 跳過序列向量 $(h_1 \dots h_{T_x})$ 加入得到 透過 bi-directional RNN, h_i 可包含輸入序列的第 i 個詞及其前後信息
- * $h_i = [h_i^T; \tilde{h}_i^T]^T, i=1\dots n$ 透過权重相加, 表示生成第 i 個輸出的注意力分配自己不同, α_{ij} 值越大, 表示第 j 的輸出在第 i 輸入分配的注意力越多, 在生成第 i 輸出時受第 j 輸入影响越多
- * 以串聯方式將前向和後向向量接作為 Encoder 後的隱藏層狀態 \tilde{h}_i^T 代表單詞本身和之前的一些信息, h_i^T 代表單詞本身和之後的一些信息



5.1 Self-Attention 自注意力机制

又稱“intra-attention”(內部注意力), 是關聯序列不同位置的注意力机制, self-attention 机制可以學習當前單詞和句子前一部分詞之間的相關性

b. Transformer 架構: 多層 self-attention、層級歸一化, Encoder 和 Decoder 都使用 主連接層、殘差連接

Encoder: 6 個模塊堆疊, 每個模塊有 2 個子層級

⇒ Multi-Head 自注意力机制。

自注意力机制的 input、output 是同一條

② 主連接層: 注意子層級的特徵,

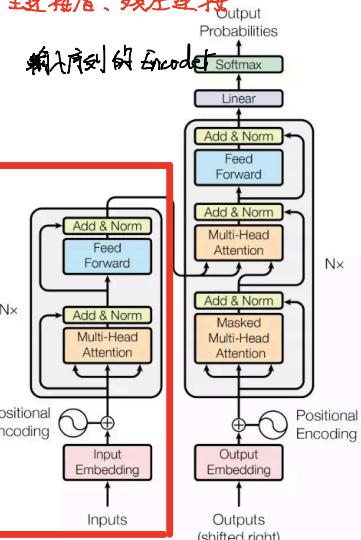
每個子層級都會添上 殘差連接和層級歸一化

輸入序列 → 詞嵌入向量 → 各位置統計向量相加
輸入 Multi-Head Attention
Multi-Head Attention 輸出 + 輸入
→ 層級歸一化 → 放到主連接層

Decoder:

建立自注意力模塊 + Encoder 輸出

→ Multi-Head Attention



⑤ 残差连接: skip connection (from skip-block in ResNet)

将 x 输入到 $F(x)$ 输出时，可用 $G(x)$ 插值 input, output.

且可拆分为 $T(x)$ 和 x 的线性叠加

$$\text{LSTM: } y = H(x, WH) \cdot T(x, WT) + X \cdot (1 - T(x, WT))$$

$$\text{当 } T(x, WT) = 0, y = x$$

$$T(x, WT) = 1, y = H(x, WH)$$

→ 网路越深，表达能力越强，但随着深度增加，会出现 gradient explode, gradient vanishing

→ 利用 skip-connection 替代用过但失败的初始化、BN层、ReLU等激活函数

DL依靠 残差的链式反向传播(BP) 进行参数更新 → skip-connection:

$$y = H(x, WH) + X$$

$$f' = f(x, w_f) + \text{卷积 cost 对于 f 的导数:}$$

$$g' = g(f') \quad g \text{ 激活} \quad \frac{d(f')}{d(w_f)} \times \frac{d(g')}{d(f')} \times \frac{d(y')}{d(g')} \times \frac{d(\text{cost})}{d(y')}$$

$$y' = k(g') \quad k \text{ 分类器}$$

$\text{cost} = \text{criterion}(y, y')$ → 其中一个导数很小，越来越小 → gradient vanishing，造成深层网路传到浅层就没了

通过残差，每个導数加上恒等项 1, $\frac{df}{dx} = \frac{d(f+x)}{dx} = 1 + \frac{df}{dx}$ ，就算 $\frac{df}{dx}$ 很小，仍能进行反向传播

→ skip-connect 改善反向传播中的 gradient vanishing

④ 深层神经网路训练失败：权重矩阵的退化

$$\text{以: } G(1) = 1.1$$

$$G: \text{非残差 } H(1) = 1.1, H(1) = F(1) + 1, F(1) = 0.1$$

$$H: \text{残差 } t+1,$$

$$G(1) = 1.2$$

$$H(1) = 1.2, H(1) = F(1) + 1, F(1) = 0.2$$

$$\rightarrow G \text{ 的 gradient} = \frac{1.2 - 1.1}{1.1}$$

$$F \text{ 的 gradient} = \frac{0.2 - 0.1}{0.1}$$

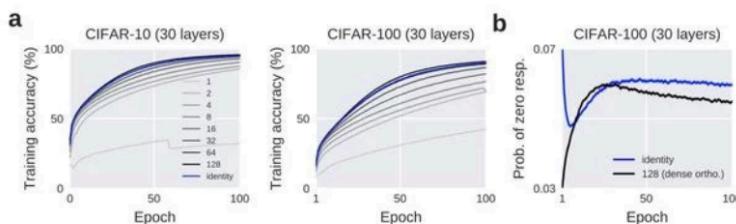
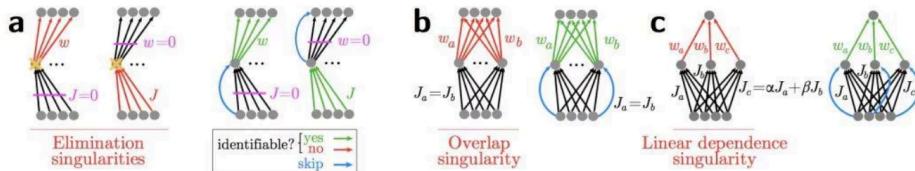


Figure 7: Random dense orthogonal skip connectivity matrices work slightly better than identity skip connections. (a) Increasing the non-orthogonality of the skip connectivity matrix reduces the performance (represented by lighter shades of gray). The results shown are averages over 10 independent runs of the simulations. (b) Probability of zero responses for residual networks with identity skip connections (blue) and dense random orthogonal skip connections (black), averaged over all hidden units and all training examples.

→ 作者認為神经网路的退化才是难以训练深层网路的主要原因，不是权重矩阵的退化
而是 gradient vanishing

在梯度大的情况下，每层中只有少部分隐藏单元通过 Activation Function 改变激活值，大部分值原本维持不变，会造成整个权重矩阵的秩不高，层权愈多，秩更低

→ 形成網路退化：高維矩陣中大部分的權重沒有信息，表達能力不符期待
殘差連接正是強制打破了網絡的對稱性。



第1種(圖a)，輸入權重矩陣(灰色部分)完全退化為0，則輸出W已經失去鑑別能力，此時加上殘差連接(藍色部分)，網絡又恢復了表達能力。第2種(圖b)，輸入對稱的權重矩陣，那輸出W一樣不具備這兩部分的鑑別能力，添加殘差連接(藍色部分)可打破對稱性。第3種(圖c)是圖b的變種，不再說明。

* 激活函數：Activation Function (AF)

要選擇何微分的函數

① 數據大多是非線性分布，但一般神經網路的計算是線性，引入AF是在神經網路加入非線性，解決非線性問題，並強化網路的學習能力。

神經網路可近似逼近任何非線性函數，並用於非線性模型

$$Y = \sum (\text{weight} * \text{input}) + \text{bias}$$

(特定权重的加权值 + 1个偏置值)
<映射函数>
→ 特徵提取

② 連接：神經元之間的連接帶有 weight，訓練的目的
是要更新 weight 以降低損失

③ 偏置(offset)：神經元額外的輸入，為1，有自己連接的权重，
確保當z為0時，還存在1個 AF

④ Activation function：為神經網路加入非線性特徵，
把值壓到很小的範圍(e.g. sigmoid 用[0,1])

⑤ 在反向伝播(取梯度)中，可能產生
gradient vanishing, gradient exploding

→ 用 Residual Connection

在某層梯度變得特別大，回後會產生鏈式效應，
導火線會越來越長，網器崩壞

$$1. z = x_1 * w_1 + x_2 * w_2 + \dots + x_n * w_n + b * 1, \quad \sigma = a_{\text{out}} = \text{sigmoid}(z)$$

$$\text{sigmoid: } f(z) = \frac{1}{1 + \exp(-z)}$$

(Logistic 函數)

取值範圍: [0,1]，將實數映射到 [0,1]

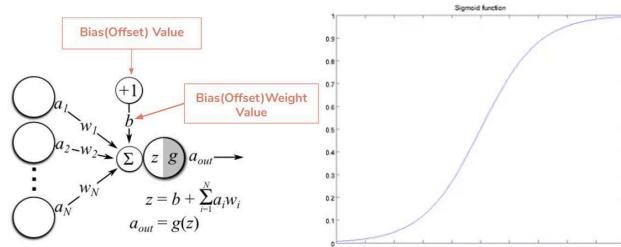
用來做二分類 (伯努利分布)

輸出恆正且嚴格遞增

$$\text{微分: } f'(x) = f(x)(1-f(x))$$

在特徵比較複雜或相差不大的效果比較好

但在反向伝播時容易出現 gradient vanishing



前向網路通常使用反向伝播法訓練

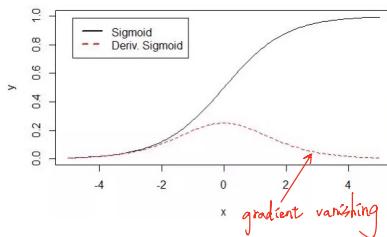
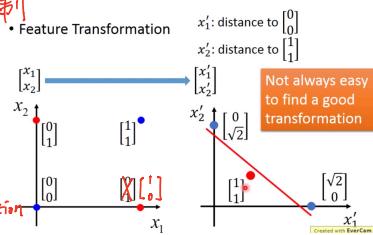
→ 反向传播，对 Activation Function 有要求

$$\phi'(x) = \phi(x)(1 - \phi(x))$$

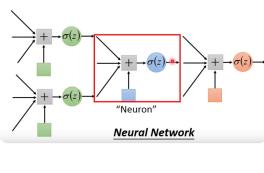
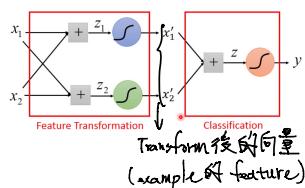
② Limitation：線性分類的限制

1. Feature Transformation

重新做线性软硬，但不是要用那种
Feature Transformation，非机器学习，是
人工选择
→ 需要让机器自行产生 Feature Transformation



• Cascading logistic regression models



→ 某個 logistic regression 的 input 和 X 是這個 LG 的 output，
某個 LG 的 output 也可以是接下来 LG 的 input！

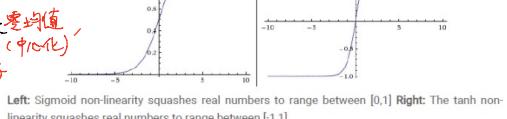
2. Hyperbolic tangent function $\tanh(x)$

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad \tanh(x) = 2\text{sigmoid}(2x) - 1$$

取值範圍 [-1, 1]

在梯度相差明顯時效果很好，
在後續過程中會不斷擴大梯度效果

与 sigmoid 不同，tanh 是 零均值
(中心化)，
實際應用會比 sigmoid 更好



3. ReLU (Rectified Linear Unit) 用於 Hidden Layer 輸出

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

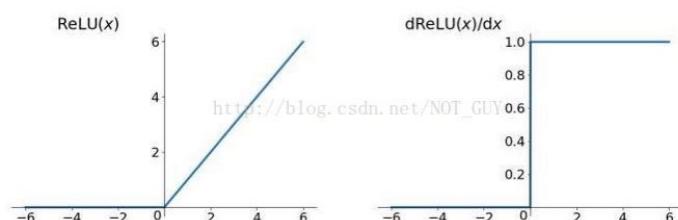
輸入信號 < 0 時，輸出都是 0，> 0 時 輸出 = 輸入

→ 發現使用 ReLU，得到的 SGD (隨機梯度下降) 的收斂速度
比 sigmoid / tanh 快很多，且可抵抗 gradient vanishing

→ 如果值有大於 0 的話是 1，沒有的話是 0

③ Dead ReLU Problem：死掉的部分永遠不會被激活，

梯度永遠是 0



缺點：① 訓練時很容易 die，假如有一個很大的梯度經過 ReLU，重新更新參數後

，ReLU 不再對其據激活，→ ReLU 的梯度會永遠為 0

→ 一旦輸出變成 0，該節點之後就會一直是 0，該節點就沒用了，可降低所需要的精度→ 提高收斂速度

② 學習率 (Learning Rate) 太大造成無法收斂，很可能大部分 ReLU 都 dead 了，可見 (補)

4. Softplus function：連續版本的 ReLU，但表現較 ReLU 差

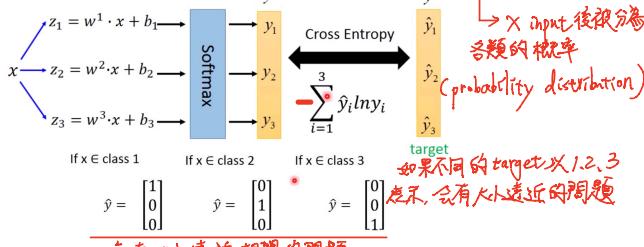
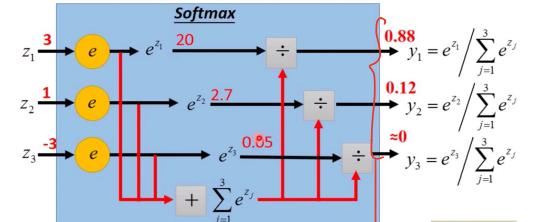
6. Softmax : 用於多分類神經網路輸出 (多項式分布) (Multi-class Classifier)

$$Y = \sum (\text{weight} * \text{input}) + \text{bias}$$

$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$ for $j = 1, \dots, K$

Probability:
 $C_1: w^1, b_1 \quad z_1 = w^1 \cdot x + b_1 \quad 1 > y_1 > 0$
 $C_2: w^2, b_2 \quad z_2 = w^2 \cdot x + b_2 \quad \sum_i y_i = 1 \quad (\because \text{有做} N \text{orm})$
 $C_3: w^3, b_3 \quad z_3 = w^3 \cdot x + b_3 \quad \text{softmax}(x) = \text{softmax}(x + C)$

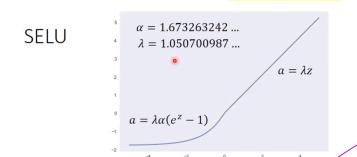
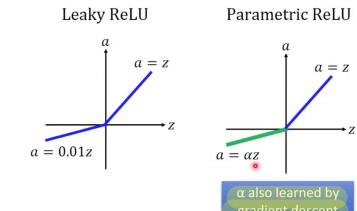
乙為任意值，取exp後歸正
 ① Softmax 會對 max 做強化，取 exp 後大小之間的差距拉更開
 ② exp? → Maximum Entropy
 1. (要讓大的更大)
 2. (需要可取導數的函式)



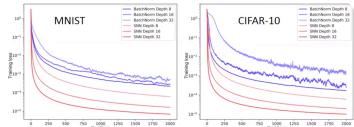
沒有大小靠近相關的問題

7. ReLU (補)

ReLU - variant

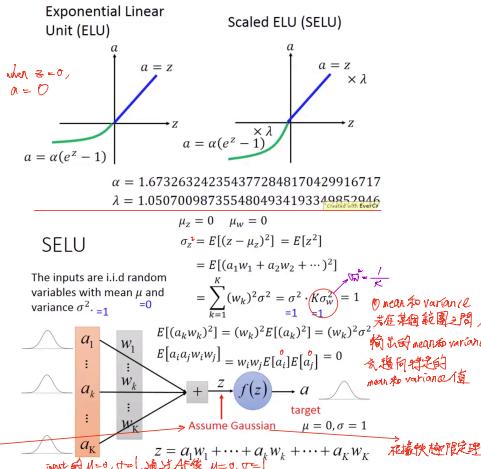
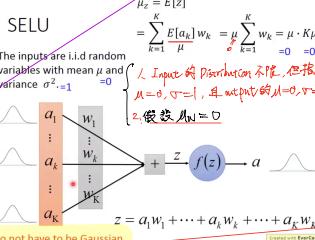
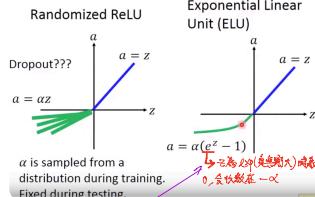


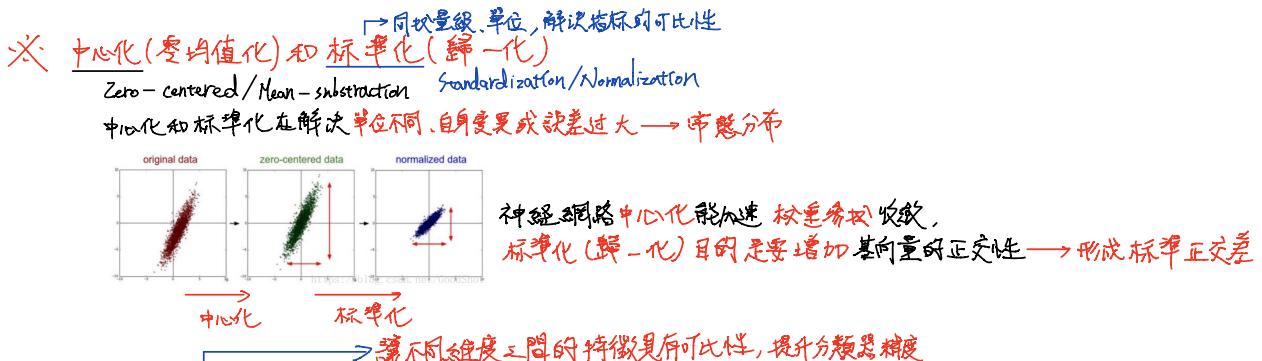
Positive and negative values
 ▶ The whole ReLU family has this property except the original ReLU.
 Saturation region
 ▶ ELU also has this property (∴ ELU又多了一個特點)
 Slope larger than 1
 ▶ 只有多了一次，小變化的改變會很大，因此叫



FNN method comparison			ML method comparison		
Method	avg. rank diff.	p-value	Method	avg. rank diff.	p-value
SNN	-0.756		SNN	-6.7	
MSRAinit	-0.240*	2.7e-02	SVM	-6.4	5.8e-01
LayerNorm	-0.16	1.3e-03	RandomForest	-5.9	2.4e-01
Highway	0.021*	1.9e-03	MSRAinit	-5.4*	4.5e-03
ResNet	0.273*	5.4e-04	LayerNorm	-5.3	7.1e-02
WeightNorm	0.397*	7.8e-07	Highway	-4.6*	1.7e-03
BatchNorm	0.504*	3.5e-06			

ReLU - variant

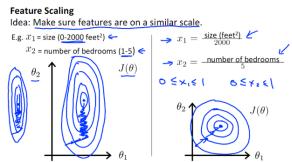




⑤ 標準化(歸一化)：把數據映射到 $[0, 1]$

① 加快梯度下降求最優解的速度(提升模型收斂的速度)：

如下图， x_1 的取值为0-2000，而 x_2 的取值为1-5，假如只有这两个特征，对其进行优化时，会得到一个窄长的椭圆形，导致在梯度下降时，梯度的方向为垂直等高线的方向而走之字形路线。这样会使迭代很慢，相比之下，右图的迭代就会很快（理解：也就是步长走多走少方向总是对的，不会走偏）



→ ① min-max 標準化：

$$x^* = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

② Z-score : $[0, 1]$ 映射

$$x^* = \frac{x - \mu}{\sigma}$$

② 提升模型的精度：

避免取值範圍小者在距離計算的影響力小於取值範圍大者，且即使各特徵的貢獻度相同

{ 同趨化處理：解決不同性質指標問題，因為直接加總會忽略重要性和作用力，
利用次實值指標指標性質，讓所有指標的作用力同趨化，再加總

無量綱化處理：提升指標的可比性，將不同坡量級的指標映射到 $[0, 1]$

(有量綱 \rightarrow 無量綱)

$$z_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

五三

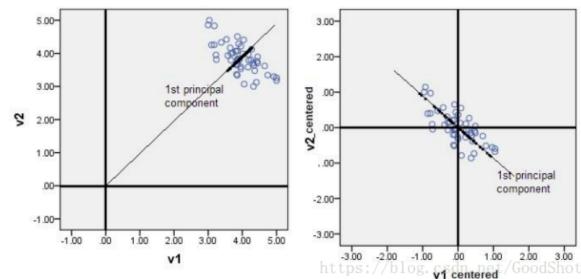
* 均值歸一化可防止梯度爆炸



④ 中心化(零均值化):

平稳的过程，但取出其中的特徵向量(主成分)平移到原点

→ 才能概括和看出数据的分布方向



※ ELMo, GPT, BERT

1. Word Embedding: 根據上下文分析 token，並依據相關性給予向量，同類、相近的 token 距離會愈近，每 One-Hot Encoding 不同，0-H-E每個 token 之間彼此獨立。每 Bag of Words (BoW)不同，是個 BoW 有分類但還是太粗略

2. 一詞多義 (不同的 token，但同样的 type)，以前只要 1 個 type 就會對應到同一個 embedding，但無法用於一詞多義。

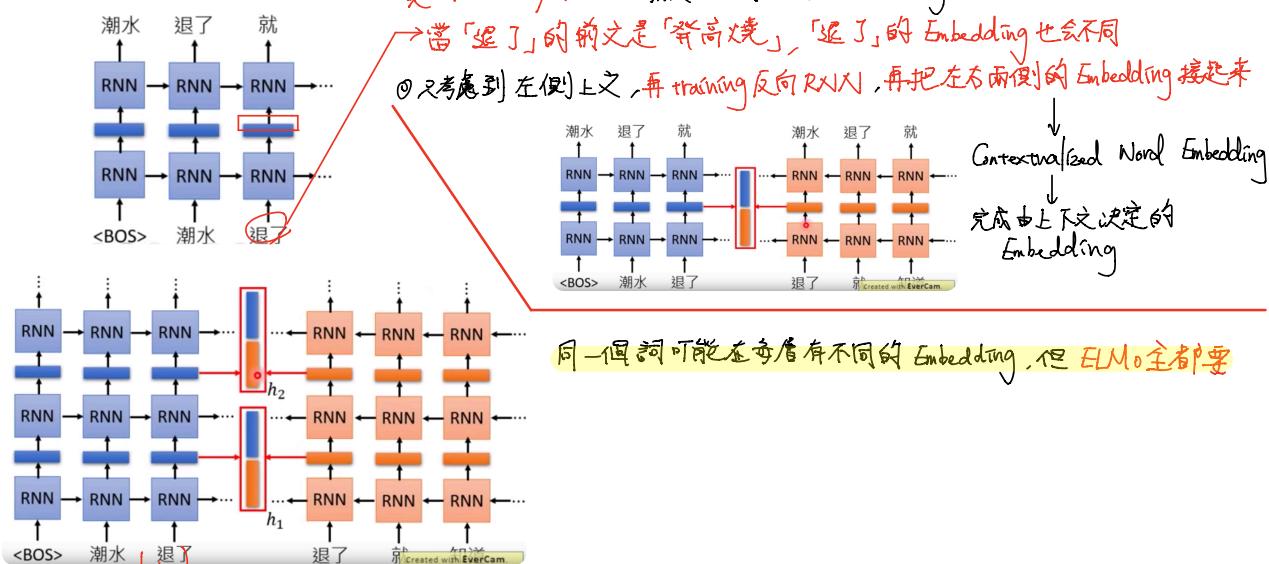
1 個 type → 1 個 embedding → 1 個 type → 固定多個 embedding → 每個 token → 不同的 embedding

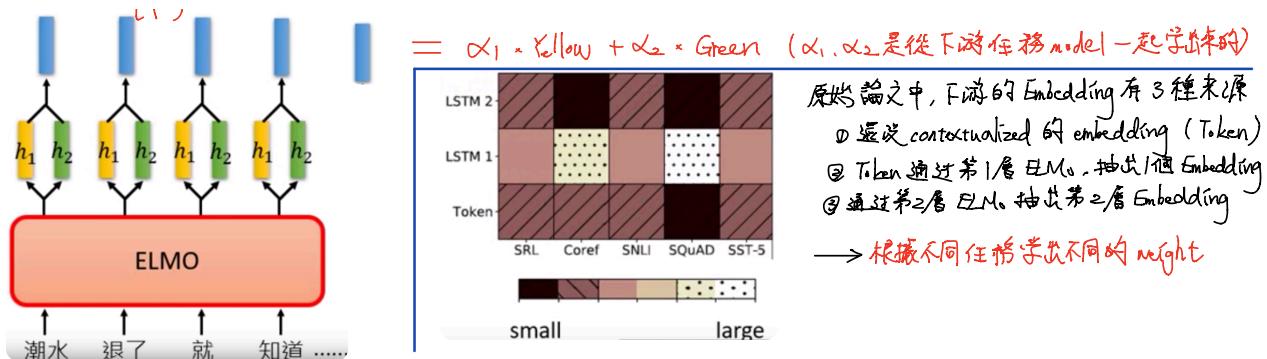
3. Contextualized Word Embedding: 每個 token 都有不同的 embedding，由上下文決定，愈相近，向量會愈近

4. Embedding from Language Embedding (ELMo):

RNN-based Language Model: 蒐集未標注的句子，再由 RNN 訓練，學習的過程是預測下一個 token

→ Hidden Layer: 由前文預測生成的 Embedding





5. Bidirectional Encoder representation from Transfomers (BERT):

① BERT = Encoder of Transformer

② 只需搜集一堆句子，不需 annotation，就可 train Encoder

→ 句子輸入，吐出 Embedding

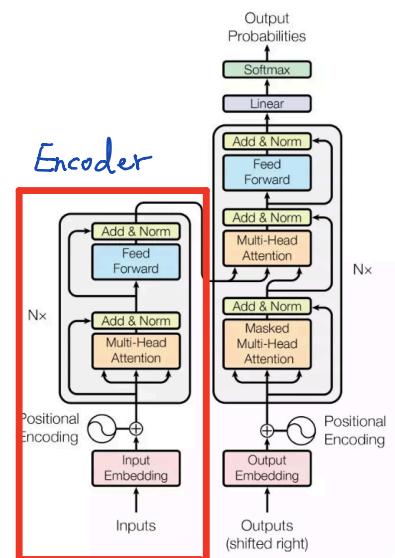
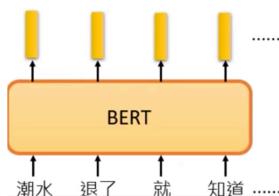
③ Encoder + self-attention layer:
→ input sequence, → output sequence

④ BERT: input word sequence, 輸出一半 embedding，
每個 embedding 再反應到不同的 word

⑤ 輸入問題：詞 vs. 字

如果 token 是詞組成 One-Hot encoding
的维度会太大。

實作方式用中文的字作 O-H-E 会比較合適



⑥ Pre-training方法: ① MLM 的 mask 問題: ① [MASK] 在實際預測不會出現，訓練時用太多 [MASK] 會造成 overfitting
② 每個 batch 只有 15% 被預測，沒做比 left-to-right

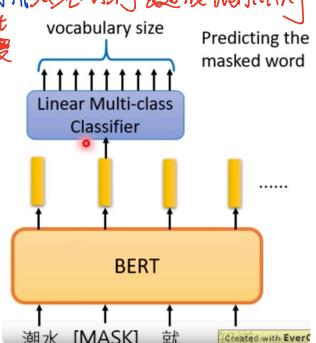
1. Masked LM: 輸入的句子隨機屏蔽 15% 的詞，變成 [MASK]，邊慢

BERT 的任務是要猜出來 [MASK] 的詞

→ 將 [MASK] 的 embedding (向量)，丟到

Linear Multi-class Classifier 預測 [MASK] 可能是什麼詞
爛，所以 BERT 需要很多層的 embedding 加校正丟入
Classifier 才能有比較好的預測

→ 如果 2 個詞填在 [MASK] 處都很合理，就有
類似 ST embedding，因為 embedding 的向量很近，
所以被 Classifier 訂在一起



2. Next Sentence Prediction (NSP): 給 BERT 2 個句子，判定是否為連續句或毫無關聯

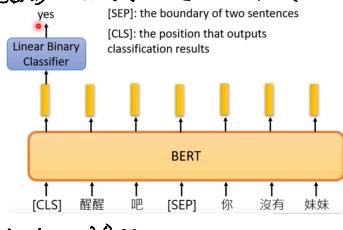
[SEP]: 句子間的分界

[CLS]: 放在句子開頭，告訴 Encoder 要做 Classification

[CLS] 輸出的 embedding，丟入 Linear Binary Classifier.

判定是否為連續句

→ [CLS] 因為 BERT 的內部是透過 Transformer 執行，透過



self-attention，在不考慮 position encoding 的影響之下，詞和詞或字和字的距離都~~都不~~不影響
 → [CLS]的位置不影響分類結果，但傳統的 RNN 必須把整個句子讀完，[CLS]可能要放在句尾
 → MLM 和 NSP 在文獻上是要同時使用效果比較好

6 Generative Pre-training (GPT):

GPT = Decoder of Transformer

① 預測新的 token: self-attention

② Zero-shot Learning:

• Reading Comprehension

$d_1, d_2, \dots, d_N, "Q:", q_1, q_2, \dots, q_M, "A:"$

• Summarization $d_1, d_2, \dots, d_N, "TL;DR":$ (效果類似 random)

• Translation

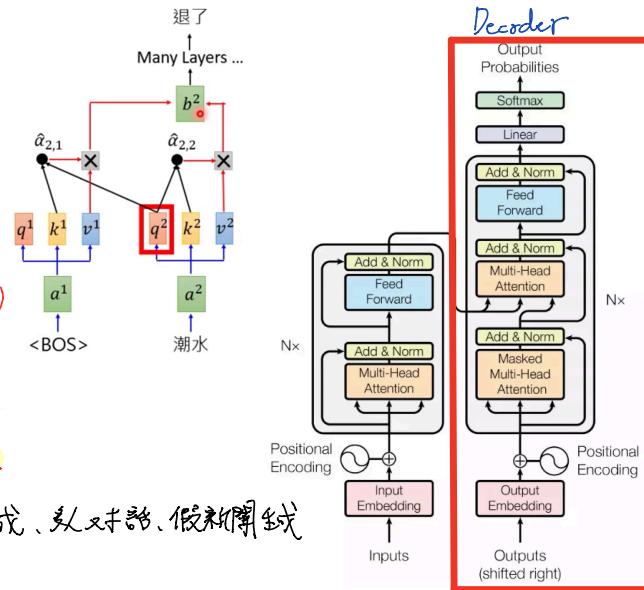
(效果差)

English sentence 1 = French sentence 1
 English sentence 2 = French sentence 2

可在輸入的資料未經過訓練，就能透過

self-attention 自動生成

③ OpenAI 有用來作寫作方面的文章生成、交叉對話、假新聞識別



※ Transformer

1. RNN 常被用來處理 input 是一個 sequence，可用單向/雙向 RNN 完成 Seq2Seq 轉換輸出

2. 但 RNN 無法做到平行處理，只能依序處理 → 用 CNN 整很多層達到平行處理，缺點是需要很多層才能得到長期的資訊

3. self-attention: 輸入輸出和双向 RNN 相同，輸出時也會遍歷整個 input 的 sequence，但能同時被計算出來，不像 RNN 需要依序計算 → 用 RNN 能做的部分，現在都用 self-attention 完成

4. Self-attention

q : query (to match others)

$$q^i = W^q a^i$$

k : key (to be matched)

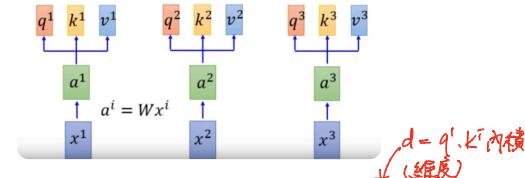
$$k^i = W^k a^i$$

v : value (information to be extracted)

$$v^i = W^v a^i$$

→ 將 q 對 k 做 attention:

$$\hat{a}_{1,i} = \exp(\alpha_{1,i}) / \sum_j \exp(\alpha_{1,j})$$



$$\text{Scaled Dot-Product Attention: } \alpha_{1,i} = \frac{\text{dot product}}{d = q^1 \cdot k^i / \sqrt{d}}$$

dot product

Soft-max

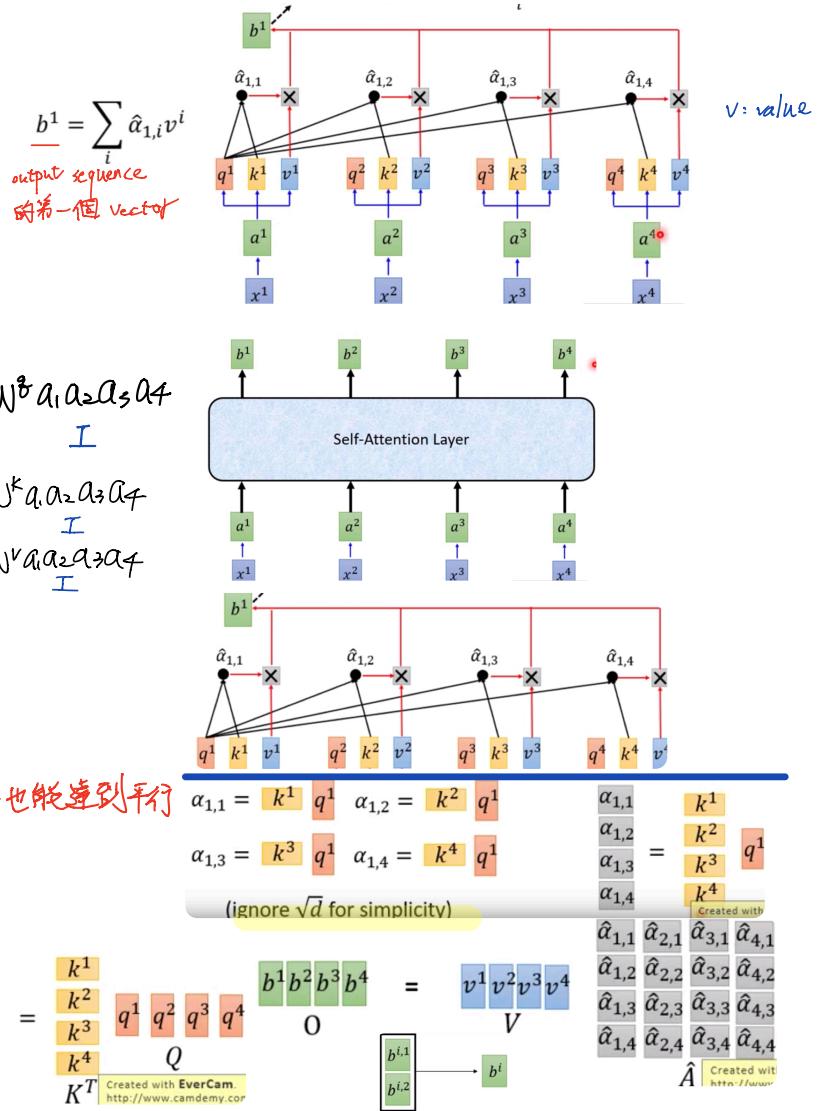
Weighted Summation

Output

Value

Key

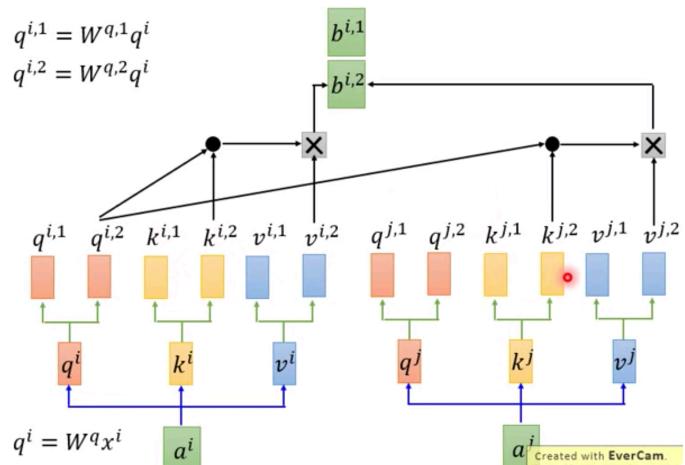
Query



5. Multi-head Self-attention

① 2 heads as example

② 可能不同的 head 閱讀的資訊不一樣，
有些取 local 的資訊，或取長期 (global)
的資訊



b. Position Encoding

- ① No position information in self-attention.
- ② Original paper: each position has a unique position vector e^i (not learned from data)
 - Input 的 vector 都做 self-attention, 正确遠近都一样, 但因為沒有位置的資訊
 - a^i 打了 $b = b$ 打了 a
 - e^i 值自定義 (但 dimension 要和 a^i 一样才能相加)

③ In other words, each x^i appends a one-hot vector p^i

$$\rightarrow \begin{matrix} W^I & \cdot & x^i \\ W^P & \cdot & p^i \end{matrix} = \begin{matrix} W^I & \cdot & x^i \\ & \oplus & a^i \\ & \cdot & e^i \\ + & \cdot & W^P \cdot p^i \end{matrix}$$

其中 W^P 也是自定義

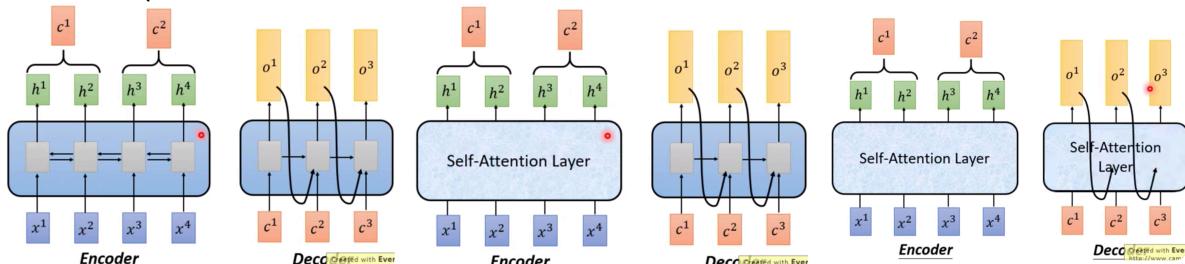
$$p^i = \begin{matrix} \dots \\ 0 \\ 1 \\ 0 \\ \dots \end{matrix} \leftarrow i\text{-th dim}$$

$$q^i, k^i, v^i$$

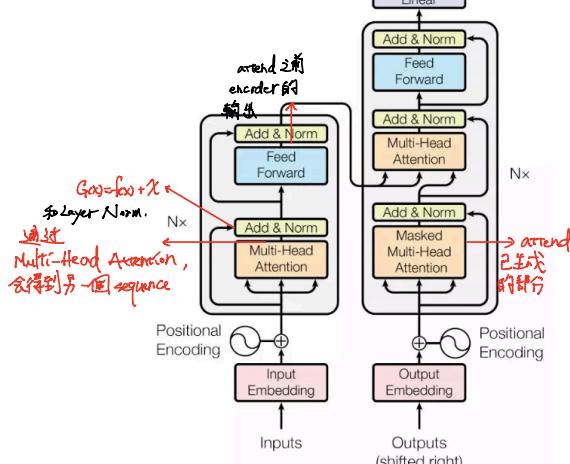
$$e^i + a^i$$

$$x^i$$

7. Seq2Seq with Attention (看到 RNN, 就用 self-attention 替代)



8.



c. Batch Normalization

$$Y = \sum (\text{weight} * \text{input}) + \text{bias}$$

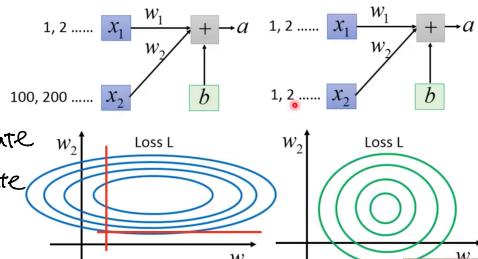
1. Feature Scaling: Make different features have the same scaling

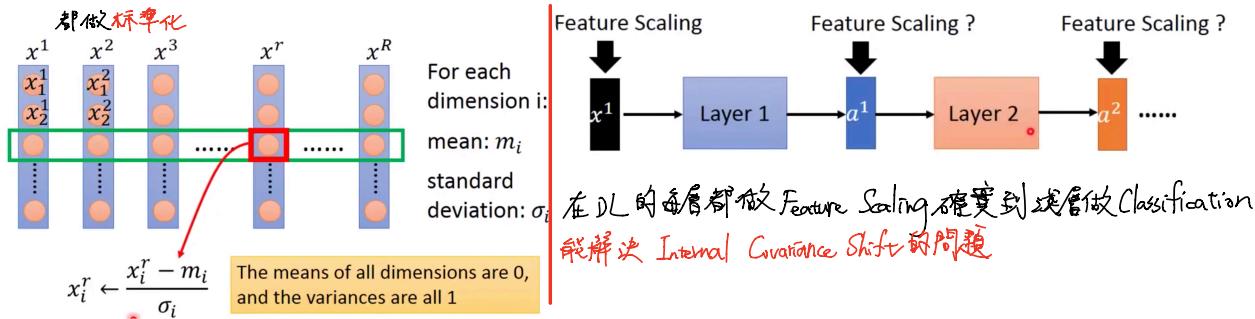
站在 N_1 的角度, 权重比較小, 影响力小 → 学习率较小, 要给较大的 learning rate

N_2 , 权重较大, 影响力大 → 学习率较大, 给较小的 learning rate

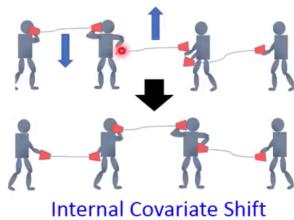
→ 但在 Training 的过程中, 给不同的 learning rate 不容易

在各数据的同一维度





② Internal Covariance Shift :



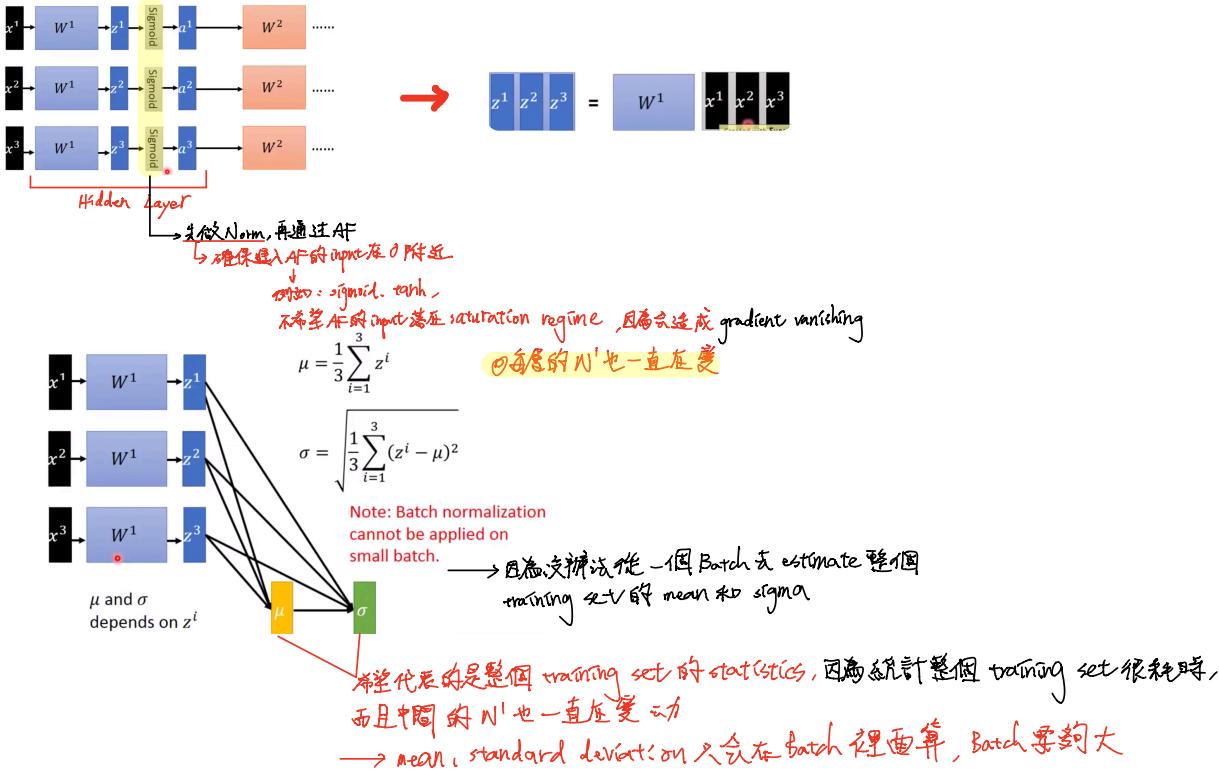
傳統：把 learning rate 設小一隻，但欲更就是去慢慢

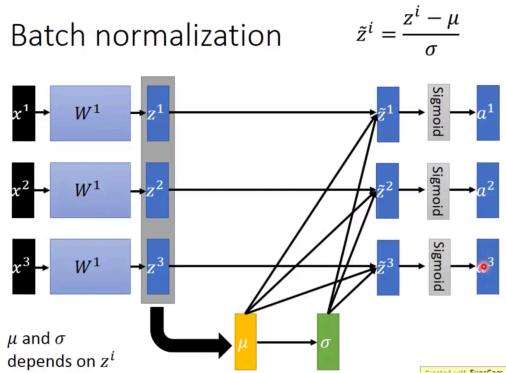
現今：Feature Scaling：對每個 layer 的 output 做 Normalization，每個 layer 而言，前一個 layer 的 output 都會在固定的範圍內，但每個 Hidden layer 的 output 也會一直在變
 → 伝統 RNN Decoder 的 schedule sampling，其中 time step 是把前一個 output 接過來，又因為後面的 time step 和前面的學好後，前面的又變，
 → Schedule Sampling 在深層網路也是，當後面的 layer 按前面的參數學好後，前面的又變，因為前面的 layer 也一直在變

① Difficulty: their statistics change during training

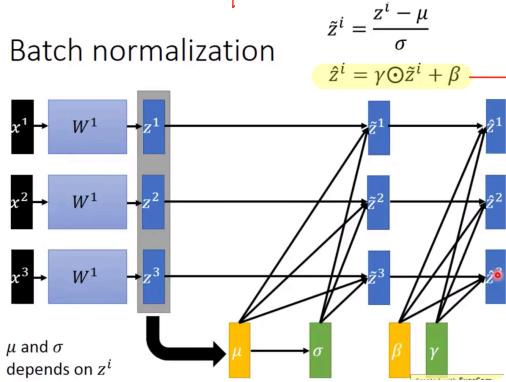
training 的過程中，參數一直在變，每個 hidden layer 的 output 也一直在變，mean 和 variance 也一直在變，很難做 Normalization
 → 下層的變化經過反向傳播可能導致上層落入 saturation regime

2. Batch Norm

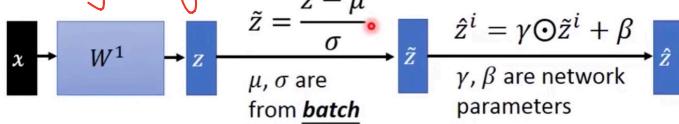




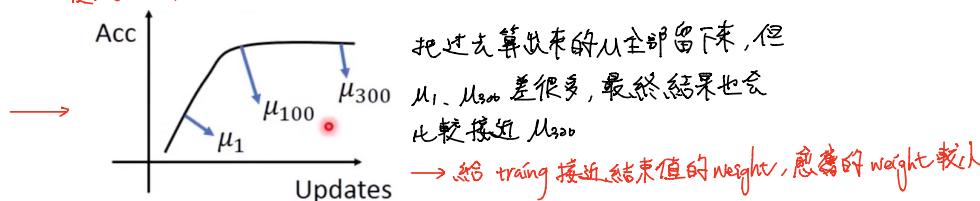
① 有些 AF 可能不希望 input 是 mean 0, variance 1



Testing stage



→ 但 training set 太大会因为 training set 可能算 Batch
之后没留下来



Batch normalization - Benefit

- BN reduces training times, and make very deep net trainable.

- Because of less Covariate Shift, we can use larger learning rates.
- Less exploding/vanishing gradients → 提高 learning rate, 加快收敛速度

- Learning is less affected by initialization.



- BN reduces the demand for regularization.

BN 主要針對每層每批樣本進行標準化, 但標準化的 input 繼過網路層後已經不是標準化, 偏差会越来越大, 而反向传播必須考慮到 Internal Covariance Shift, 只能降低 learning rate 防止 gradient vanishing/explode

通過 N 後的 z_1, z_2, z_3 , 會經過 Normalization 得到 $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3$, 再通過 sigmoid(AF), 得到 a_1, a_2, a_3 , 再放到下個 Hidden layer

→ 每個 layer 都會做, 進入 AF 之前會做 Norm

◎ Backpropagation 時, 會通過 $\frac{\partial L}{\partial \mu}, \frac{\partial L}{\partial \sigma}$ 去 update \tilde{z} , μ, σ 的值也會再變動

→ 乙对 μ, σ 的影响在 training 時会被考慮進去

→ 把目前的 mean 和 variance 幫更新, 再把 \tilde{z} 通過 sigmoid 做為

下一個 layer 的 input,
 γ, β 在 training 的過程中也能學習出來

→ 當 $\gamma = 1, \beta = 0$, 会跟原本一样,
但 γ, β 是獨立的, γ, β 是依 data 而定
↓ 不受 input 的 data 影響

在 testing stage 沒有从 γ, β 可用

→ 把模型套到 training set 上估算出 μ, σ ,
之前从 γ, β 一直在變, 但 train 完就能固定

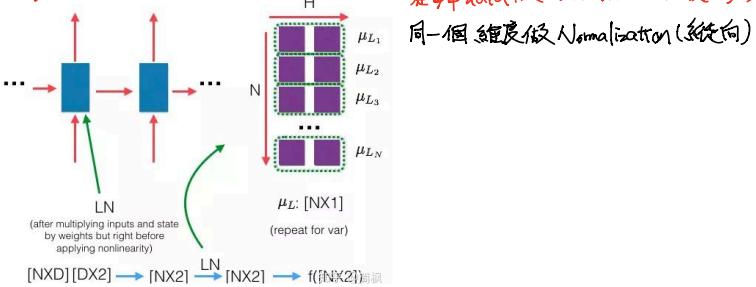
◎ 在 training 的過程中, 會 SELL 起來, 因 BN 是用 Batch 來估測整個 statistics, 會不太穩定, 產生一些振盪

用 BN 可保證 input 在 0 附近, 斜率較大的地方, 比較不會有 gradient vanishing
對 sigmoid, tanh 的 situation region 的問題

很有幫助

→ 可抵抗 overfitting 的問題, 就算有心想造成 data 的 mean 有 shift, BN 也能

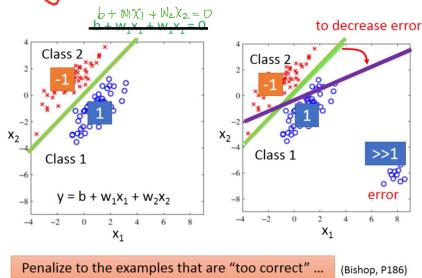
※ Layer Normalization



在每筆 data 做 Normalization (横向), 不像 Batch Normalization 是取所有 data 的同一個維度做 Normalization (縱向)

~~X~~ Classification

① Regression as Classification:



- Multiple class: Class 1 means the target is 1; Class 2 means the target is 2; Class 3 means the target is 3 problematic

多類標籤較為較容易有大小近的相關問題

- Function (Model): $f(x)$

$x \rightarrow g(x) > 0 \quad \text{Output} = \text{class 1}$
 $\quad \quad \quad \text{else} \quad \quad \quad \text{Output} = \text{class 2}$

- Loss function:

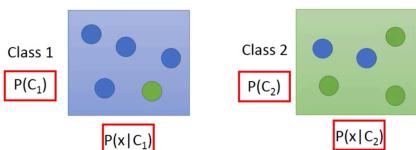
$L(f) = \sum_n \delta(f(x^n) \neq \hat{y}^n)$ 實際情況
The number of times f get incorrect results on training data.

The number of times f
get incorrect results on
training data

or Perceptron, SVM ...
感知机

Two Classes

Estimating the Probabilities From training data



Given an x , which class does it belong to?

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

→ Generative Model $P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$

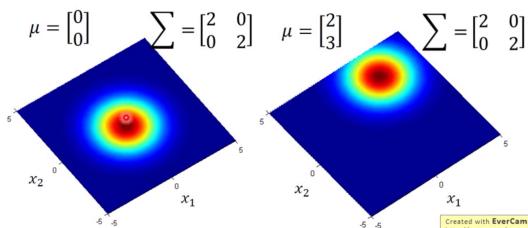
Gaussian Distribution (One of the probability Distribution) <https://blog.slinuxer.com/tag/pca>

Input: vector x , output: probability of sampling x
 The shape of the function determines by mean μ and

covariance matrix Σ

Input: vector x , output: probability of sampling x
The shape of the function determines by mean μ and variance σ^2

covariance matrix Σ vector

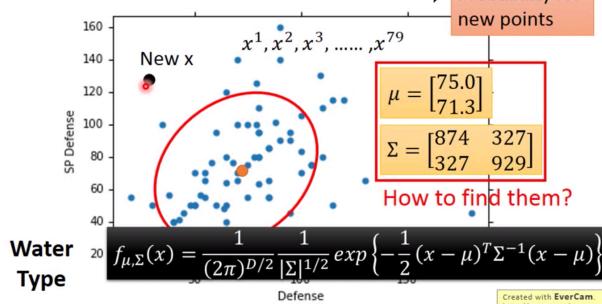


有了 $p(x|C_1), p(C_1), p(x|C_2), p(C_2)$ 就能知道 x 在 C_1, C_2 出現的机率，
 x 出現哪就会被分到 x 出現机率比較大的 class

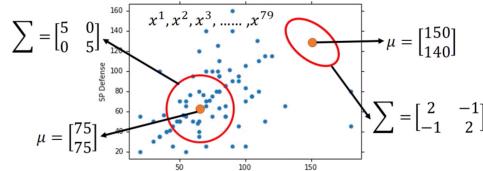
Probability from Class

Assume the points are sampled from a Gaussian distribution

Find the Gaussian distribution behind them → Probability for



Maximum Likelihood $f_{\mu, \Sigma}(x) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right\}$



The Gaussian with any mean μ and covariance matrix Σ can generate these points.

Different Likelihood

Likelihood of a Gaussian with mean μ and covariance matrix Σ

= the probability of the Gaussian samples $x^1, x^2, x^3, \dots, x^{79}$

$$L(\mu, \Sigma) = f_{\mu, \Sigma}(x^1)f_{\mu, \Sigma}(x^2)f_{\mu, \Sigma}(x^3) \dots f_{\mu, \Sigma}(x^{79})$$

Maximum Likelihood

We have the "Water" type Pokémons: $x^1, x^2, x^3, \dots, x^{79}$

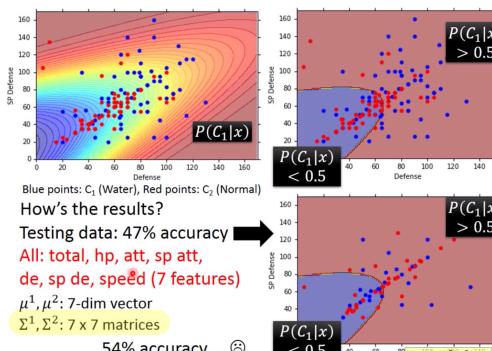
We assume $x^1, x^2, x^3, \dots, x^{79}$ generate from the Gaussian (μ^*, Σ^*) with the **maximum likelihood**

$$L(\mu, \Sigma) = f_{\mu, \Sigma}(x^1)f_{\mu, \Sigma}(x^2)f_{\mu, \Sigma}(x^3) \dots f_{\mu, \Sigma}(x^{79})$$

$$f_{\mu, \Sigma}(x) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right\}$$

$$\mu^*, \Sigma^* = \arg \max_{\mu, \Sigma} L(\mu, \Sigma)$$

$$\mu^* = \frac{1}{79} \sum_{n=1}^{79} x^n \quad \Sigma^* = \frac{1}{79} \sum_{n=1}^{79} (x^n - \mu^*) (x^n - \mu^*)^T$$



Covariance Matrix $\propto (\text{Input Feature size})^2$
Covariance Matrix 可隨着 Input Feature 快速增長

Modifying Model

Ref: Bishop, chapter 4.2.2

- Maximum likelihood

"Water" type Pokémons:

$$x^1, x^2, x^3, \dots, x^{79}$$

"Normal" type Pokémons:

$$x^{80}, x^{81}, x^{82}, \dots, x^{140}$$

Find μ^1, μ^2, Σ maximizing the likelihood $L(\mu^1, \mu^2, \Sigma)$

$$L(\mu^1, \mu^2, \Sigma) = f_{\mu^1, \Sigma}(x^1)f_{\mu^1, \Sigma}(x^2) \dots f_{\mu^1, \Sigma}(x^{79}) \\ \times f_{\mu^2, \Sigma}(x^{80})f_{\mu^2, \Sigma}(x^{81}) \dots f_{\mu^2, \Sigma}(x^{140})$$

μ^1 and μ^2 is the same as original

$$\Sigma = \frac{79}{140} \Sigma^1 + \frac{61}{140} \Sigma^2$$

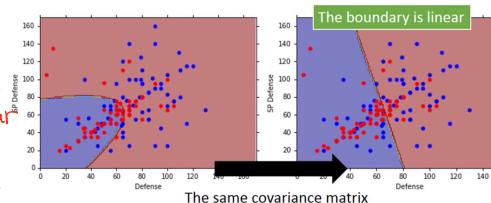
× element 權重

共用 Covariance Matrix

→ Boundary turns to linear

且提高 accuracy, 減低 overfitting 的可能性

Modifying Model



① 以 Gaussian Distribution 去做出 likelihood

當然也是 Point from Gaussian Distribution

有 test point 的 μ 和目標的 x , 就能算 likelihood
不同的 x 对應相同的 test point 会有不同的 likelihood

有很多 x 要算 likelihood 才能算 likelihood

$$L(\mu, \Sigma) = f_{\mu, \Sigma}(X_1) \times f_{\mu, \Sigma}(X_2) \times f_{\mu, \Sigma}(X_3) \times \dots \times f_{\mu, \Sigma}(X_n)$$

Likelihood 很大, 代表愈相關

∴ 寶石找到 Maximum Likelihood

Now we can do classification 😊

$$f_{\mu^1, \Sigma^1}(x) = \frac{1}{(2\pi)^{D/2} |\Sigma^1|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu^1)^T (\Sigma^1)^{-1}(x - \mu^1)\right\}$$

$$\mu^1 = \begin{bmatrix} 75.0 \\ 71.3 \end{bmatrix} \quad \Sigma^1 = \begin{bmatrix} 874 & 327 \\ 327 & 929 \end{bmatrix}$$

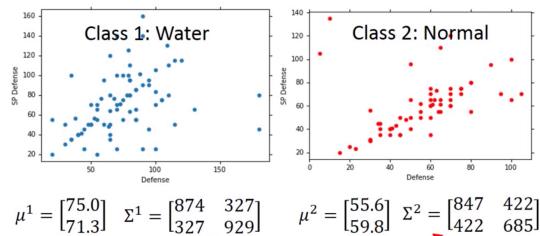
$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$f_{\mu^2, \Sigma^2}(x) = \frac{1}{(2\pi)^{D/2} |\Sigma^2|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu^2)^T (\Sigma^2)^{-1}(x - \mu^2)\right\}$$

$$\mu^2 = \begin{bmatrix} 55.6 \\ 59.8 \end{bmatrix} \quad \Sigma^2 = \begin{bmatrix} 847 & 422 \\ 422 & 685 \end{bmatrix}$$

If $P(C_1|x) > 0.5 \rightarrow x$ belongs to class 1 (Water)

Modifying Model



如果 2 個 Gaussian 都給不同的 Covariance Matrix,
Model Parameters 會過多 → 容易造成 overfitting
→ 共用 Covariance Matrix

Three Steps

- Function Set (Model):

$$x \rightarrow P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

If $P(C_1|x) > 0.5$, output: class 1
Otherwise, output: class 2

- Goodness of a function:

The mean μ and covariance Σ that maximizing the likelihood (the probability of generating data)

- Find the best function: easy

評價 function 的好壞：找最能產生 Maximum Likelihood 的 Probability Distribution.

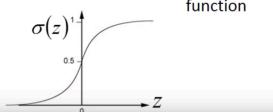
Posterior Probability

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$= \frac{1}{1 + \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}} = \frac{1}{1 + \exp(-z)} = \sigma(z)$$

Sigmoid function

$$z = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$



$$\rightarrow P(C_1|x) = \sigma(z) \quad \text{(sigmoid function)}$$

$$z = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} = \ln \frac{P(x|C_1)}{P(x|C_2)} + \ln \frac{P(C_1)}{P(C_2)}$$

$$P(x|C_1) = \frac{1}{(2\pi)^{D/2} |\Sigma^1|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu^1)^T (\Sigma^1)^{-1} (x - \mu^1) \right\}$$

$$P(x|C_2) = \frac{1}{(2\pi)^{D/2} |\Sigma^2|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu^2)^T (\Sigma^2)^{-1} (x - \mu^2) \right\}$$

$$\rightarrow P(C_1|x) = \sigma(w \cdot x + b) \rightarrow \text{重點要直接找 out } w \text{ 和 } b, \text{ 不找其他 } N_1, N_2, \mu_1, \mu_2, \Sigma$$

→ 共用 Covariance Matrix 後的 Boundary 是 linear

* Logistic Regression

Step 1: Function Set

We want to find $P_{w,b}(C_1|x)$
 $P_{w,b}(C_1|x) \geq 0.5$, output C_1
Otherwise, output C_2

$$P_{w,b}(C_1|x) = \sigma(z)$$

$$z = w \cdot x + b = \sum_i w_i x_i + b$$

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Function set:

$$f_{w,b}(x) = P_{w,b}(C_1|x) \quad \text{Including all different w and b}$$

$$\begin{array}{cccc} x^1 & x^2 & x^3 & \dots \\ \hline C_1 & C_1 & C_2 & \dots \end{array} \rightarrow \begin{array}{cccc} x^1 & x^2 & x^3 & \dots \\ \hline \hat{y}^1 = 1 & \hat{y}^2 = 1 & \hat{y}^3 = 0 & \dots \\ \hat{y}^n: 1 \text{ for class 1, 0 for class 2} \end{array}$$

$$L(w, b) = f_{w,b}(x^1)f_{w,b}(x^2)\left(1 - f_{w,b}(x^3)\right) \dots$$

$$w^*, b^* = \arg \max_{w, b} L(w, b) = w^*, b^* = \arg \min_{w, b} -\ln L(w, b)$$

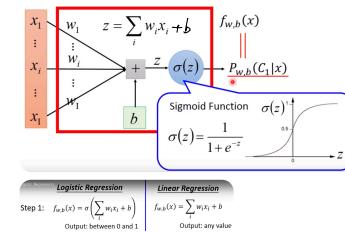
$$-\ln L(w, b)$$

$$= -\ln f_{w,b}(x^1) \rightarrow -[\hat{y}^1 \ln f(x^1) + (1 - \hat{y}^1) \ln(1 - f(x^1))]$$

$$-\ln f_{w,b}(x^2) \rightarrow -[\hat{y}^2 \ln f(x^2) + (1 - \hat{y}^2) \ln(1 - f(x^2))]$$

$$-\ln(1 - f_{w,b}(x^3)) \rightarrow -[0 \ln f(x^3) + (1 - 0) \ln(1 - f(x^3))]$$

$$\vdots$$



Step 2: Goodness of a Function

$$L(w, b) = f_{w,b}(x^1)f_{w,b}(x^2)\left(1 - f_{w,b}(x^3)\right) \dots f_{w,b}(x^N)$$

$$-\ln L(w, b) = \ln f_{w,b}(x^1) + \ln f_{w,b}(x^2) + \ln \left(1 - f_{w,b}(x^3)\right) \dots$$

$$\hat{y}^n: 1 \text{ for class 1, 0 for class 2}$$

$$= -\sum_n [\hat{y}^n \ln f_{w,b}(x^n) + (1 - \hat{y}^n) \ln(1 - f_{w,b}(x^n))]$$

Cross entropy between two Bernoulli distribution

$$\text{Distribution p: } p(x=1) = \hat{y}^n \quad \text{cross entropy}$$

$$p(x=0) = 1 - \hat{y}^n \quad \text{Distribution q: } q(x=1) = f(x^n)$$

$$q(x=0) = 1 - f(x^n) \quad H(p, q) = -\sum p(x) \ln(q(x))$$

Cross entropy: 2個 distribution 之間有多近

Step 2: Goodness of a Function

$$\begin{array}{ll} \text{Training Data} & \begin{array}{cccccc} x^1 & x^2 & x^3 & \dots & x^N \\ C_1 & C_1 & C_2 & \dots & C_1 \end{array} \end{array}$$

Assume the data is generated based on $f_{w,b}(x) = P_{w,b}(C_1|x)$

Given a set of w and b , what is its probability of generating the data?

$$L(w, b) = f_{w,b}(x^1)f_{w,b}(x^2)\left(1 - f_{w,b}(x^3)\right) \dots f_{w,b}(x^N)$$

The most likely w^* and b^* is the one with the largest $L(w, b)$.

$$w^*, b^* = \arg \max_{w, b} L(w, b)$$

求最大值 w, b

$$\text{Training data: } (x^n, \hat{y}^n)$$

$$\hat{y}^n: \text{a real number}$$

$$L(f) = \sum_n C(f(x^n), \hat{y}^n)$$

$$L(f) = \frac{1}{2} \sum_n (f(x^n) - \hat{y}^n)^2$$

Cross entropy: *

$$C(f(x^n), \hat{y}^n) = -[\hat{y}^n \ln f(x^n) + (1 - \hat{y}^n) \ln(1 - f(x^n))]$$

Question: Why don't we simply use square error as linear regression?

* Scaled Dot Product Attention

* Mask (from Transformer)

1. Padding Mask: 用於處理批次輸入序列的長度不同，在較短的序列後填充 0，但對 attention 机制，再加上非常大的負記號 → 經過 softmax 後的概率會趨近 0 (在所有的 scaled dot product 都會用到)

2. Sequence Mask: 為了讓 Transformer 的 Decoder 不能看見未來的信息
→ 在 time step 為 t 時，Decoder 的輸出只能依賴 t 之前的 output，不能依賴 t 之後的 output
 \downarrow 由左向右，由上而下，由外而內，逐層逐行全用到

→ 要把 7 之後的 tokens 丟進 decoder by sequence mask

∴ Decoder 的 self-attention 用到 att_mask 包含 sequence mask + padding mask,
其他的 att_mask 只包含 padding mask

* Position Embedding (from Transformer)

※ 神經網路架構

※ Perceptron Learning Algorithm (PLA)

※ 梯度下降、梯度爆炸、梯度消失、隨機梯度下降(GD) ⊗

※ 反向伝播、誤差反向伝播

※ 學習率与梯度下降關係

※ 一次擬合 ※ 亂數路徑 ※ 神經網路

※ Autoencoder

※ 降維(Dimension Reduction)

※ ResNet

※ 線性判別分析(Linear Discriminant Analysis)

※ 主成分分析(Principal Component Analysis, PCA)

※ Radial Basis Function(RBF)、Matrix Factorization

※ LSTM ※ NNLIN

※ Conv2D ※ Word2vec

※ Optimization

※ Regularization

※ Loss Function

※ Layer Normalization