



計算圖、反向傳播及自然 語言處理之分散式表示法

報告人: 蘇佳益

指導老師: 陳聰毅

國立高雄科技大學建功校區電子工程系

<https://github.com/chaiyisu/Natural-Language-Processing>

Agenda

- 計算圖及反向傳播
- **Tricks and Tips on Neural Network**
- 自然語言處理之分散式表示法
- 向量間的相似度
- 降維 (Dimension Reduction)



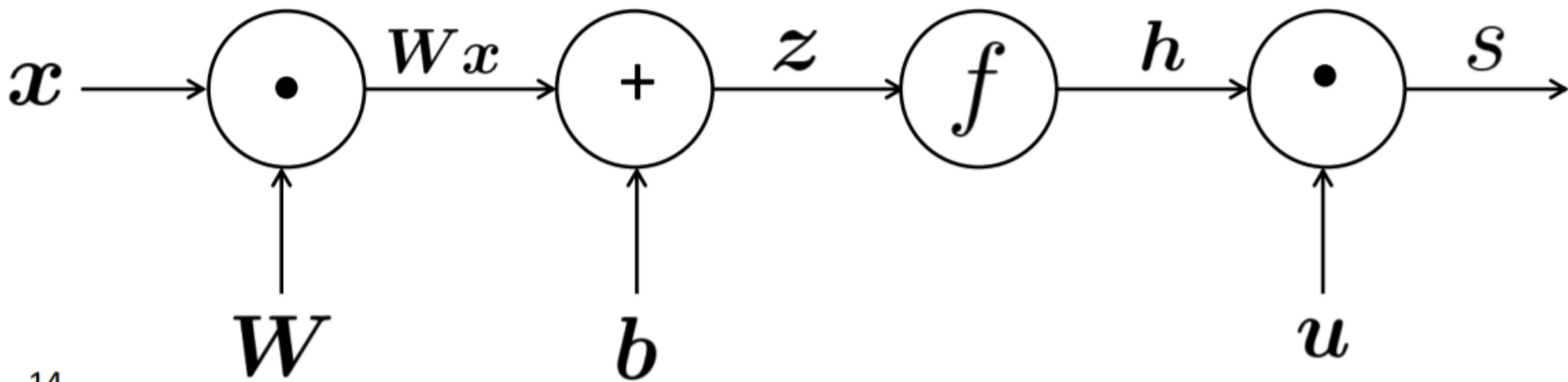
計算圖及反向傳播

Introduction

- 微分
- 連鎖率
- 通常我們會反覆利用上層網路所微分出來的值，以減少運算量及避免重複計算。

計算圖及前向傳播

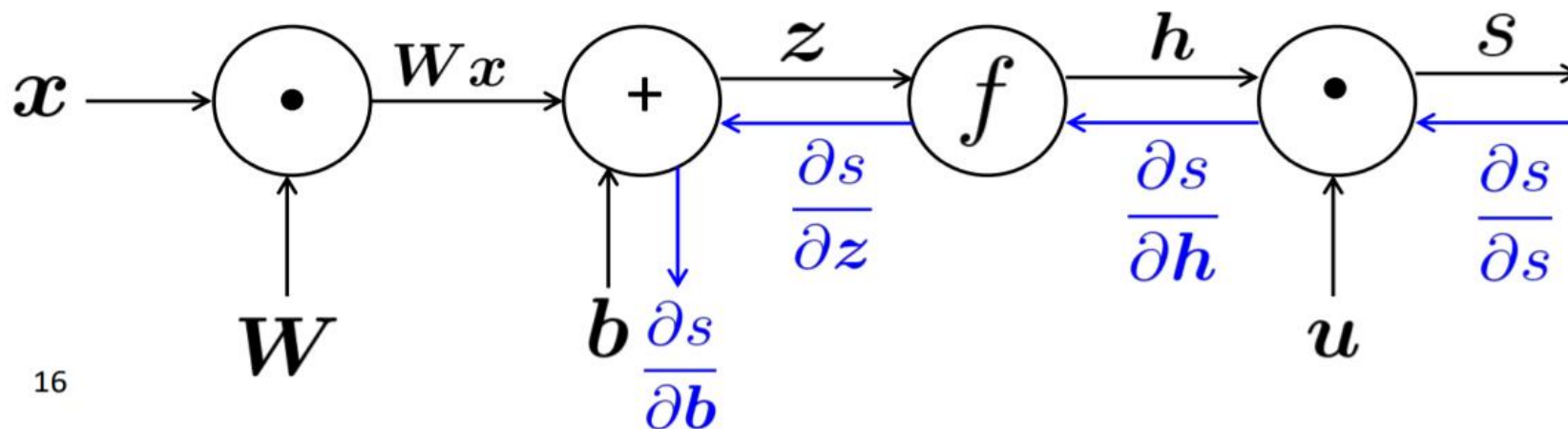
- 前向傳播
 - x (input)
 - $z = Wx + b$
 - $h = f(z)$
 - $s = u^T h$



14

計算圖及反向傳播

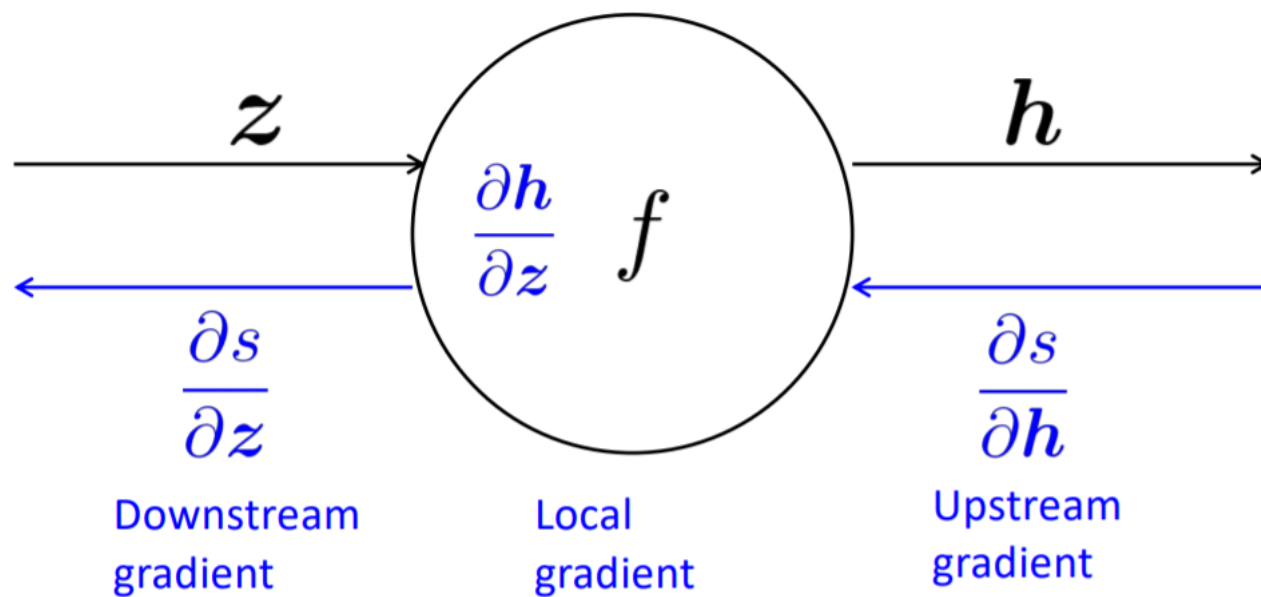
- 前向傳播
 - x (input)
 - $z = Wx + b$
 - $h = f(z)$
 - $s = u^T h$



16

單一節點的反向傳播

- 每一個節點都有
 - 上游梯度
 - Local Gradient



如何計算 $\frac{\partial s}{\partial z}$?

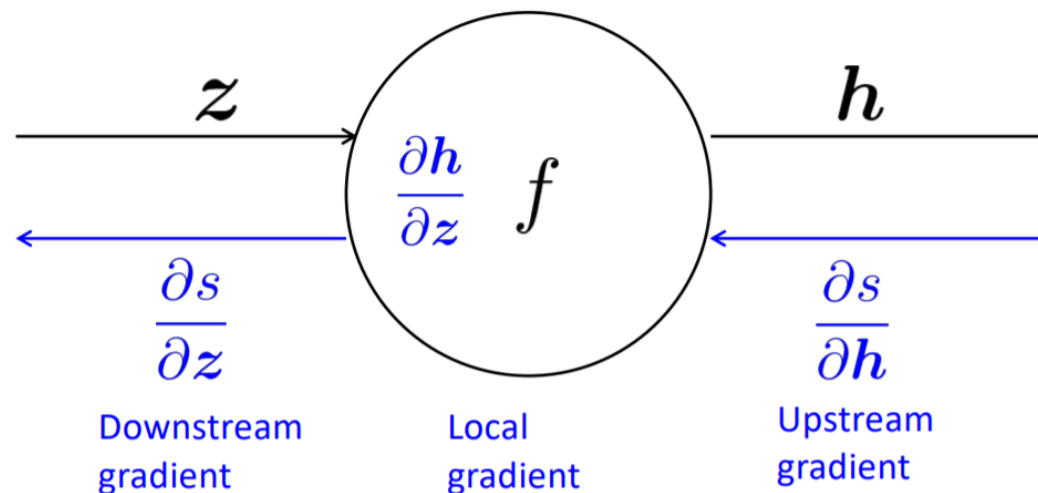
- $h = f(z)$

- $s = u^T h$

- $\frac{\partial s}{\partial z} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z}$

- (downstream gradient) = (upstream gradient) (local gradient)

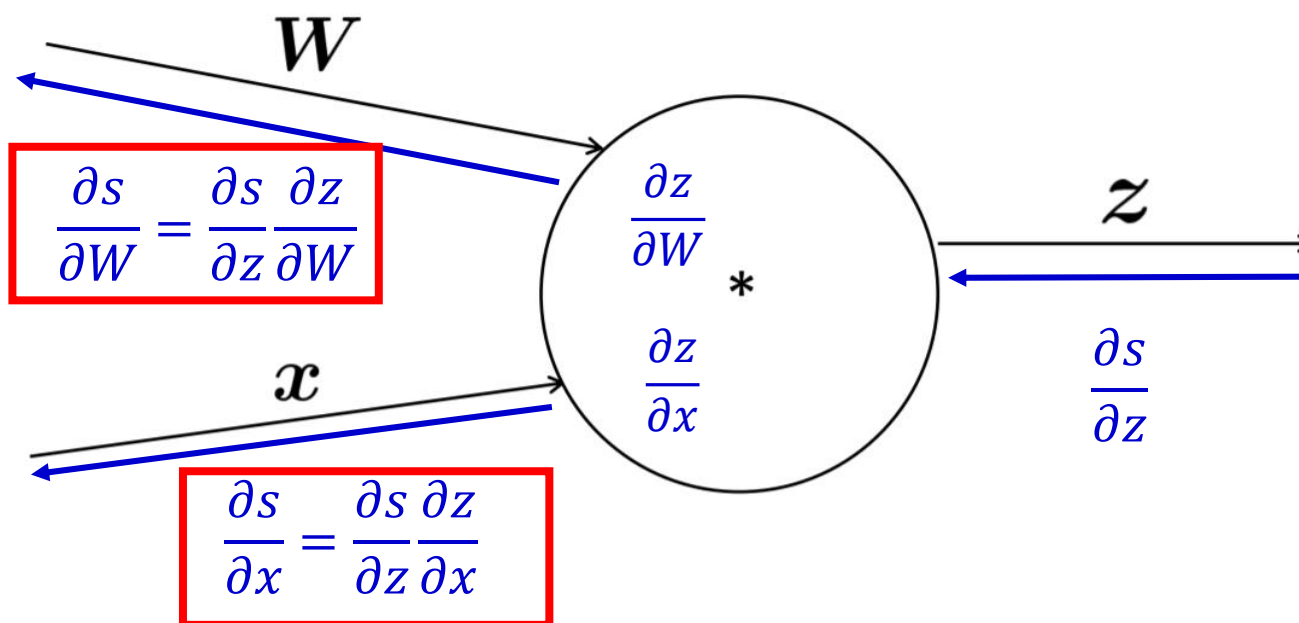
連鎖率



多輸入的節點

- $z = Wx$
- 多輸入 = 多個 Local Gradients

連鎖率

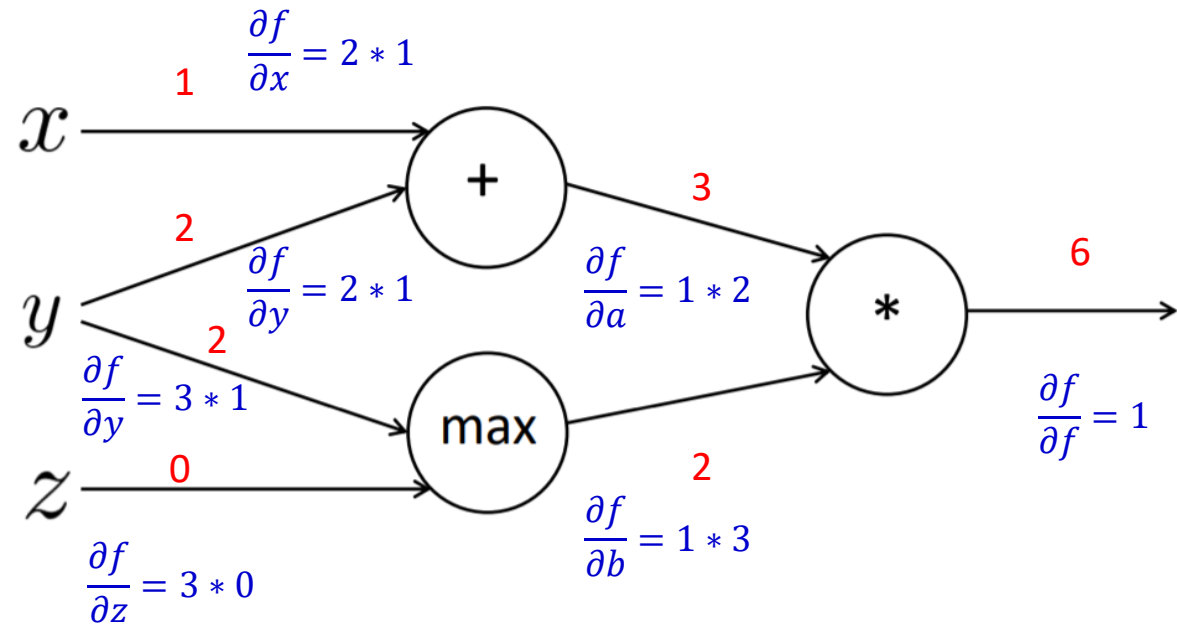


Example

- Forward Propagation

- $a = x + y = 3$
- $b = \max(y, z) = 2$
- $f = ab = 6$
- $x = 1, y = 2, z = 0$

- $\frac{\partial f}{\partial x} = 2$
- $\frac{\partial f}{\partial y} = 2 + 3 = 5$
- $\frac{\partial f}{\partial z} = 0$

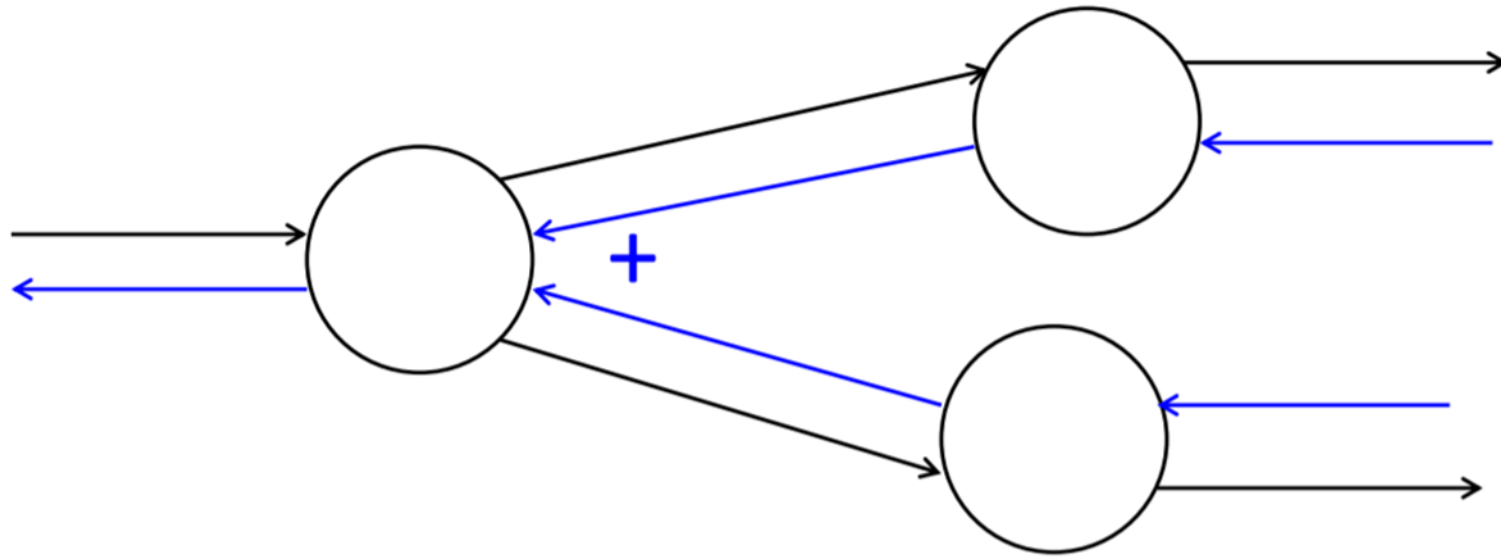


多個分支的梯度

- 多個分支



將梯度相加

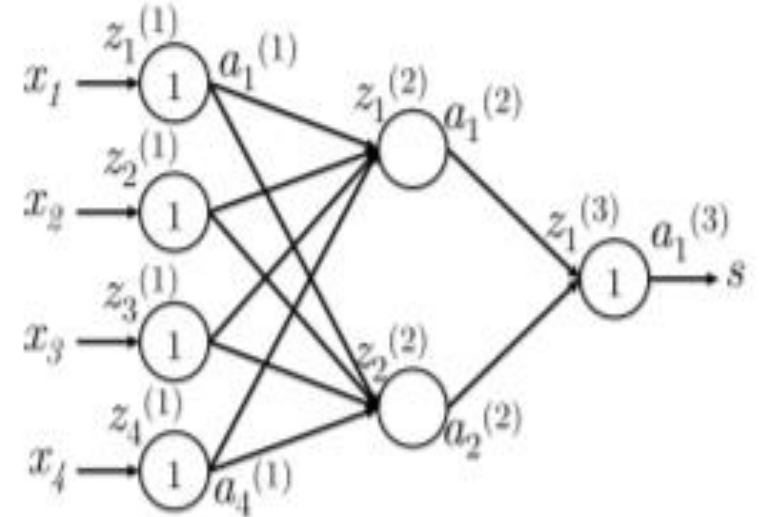


Real Network Example - What is $\frac{\partial s}{\partial W_{ij}^{(1)}}$?

- what is $\frac{\partial s}{\partial W_{ij}^{(1)}}$? $x \in R^{1 \times 4}$, $W^{(0)} \in R^{4 \times m_0}$, $a^{(1)} \in R^{1 \times m_0}$, $W^{(1)} \in R^{m_0 \times m_1}$,
 $a^{(2)} \in R^{1 \times m_1}$, $W^{(2)} \in R^{m_1 \times m_2}$, $a^{(3)} \in R^{1 \times m_2}$

- Forward Propagation

- $a^{(1)} = f(xW^{(0)})$
- $a^{(2)} = f(a^{(1)}W^{(1)})$
- $a^{(3)} = a^{(2)}W^{(2)}$



Real Network Example - What is $\frac{\partial s}{\partial W_{ij}^{(1)}}$?

- Backward Propagation

$$\Rightarrow \frac{\partial s}{\partial W_{ij}^{(1)}} = \frac{\partial a^{(2)} W^{(2)}}{\partial W_{ij}^{(1)}} = \frac{\partial a_i^{(2)} W_i^{(2)}}{\partial W_{ij}^{(1)}} = W_i^{(2)} \frac{\partial a_i^{(2)}}{\partial W_{ij}^{(1)}}$$

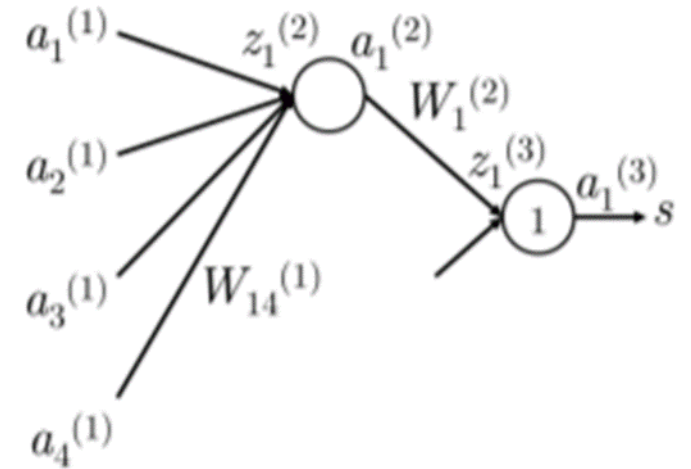
$$\Rightarrow = W_i^{(2)} \frac{\partial a_i^{(2)}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial W_{ij}^{(1)}}$$

$$\Rightarrow = W_i^{(2)} \frac{\partial f(z_j^{(2)})}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial W_{ij}^{(1)}}$$

$$\delta_i^{(2)}$$

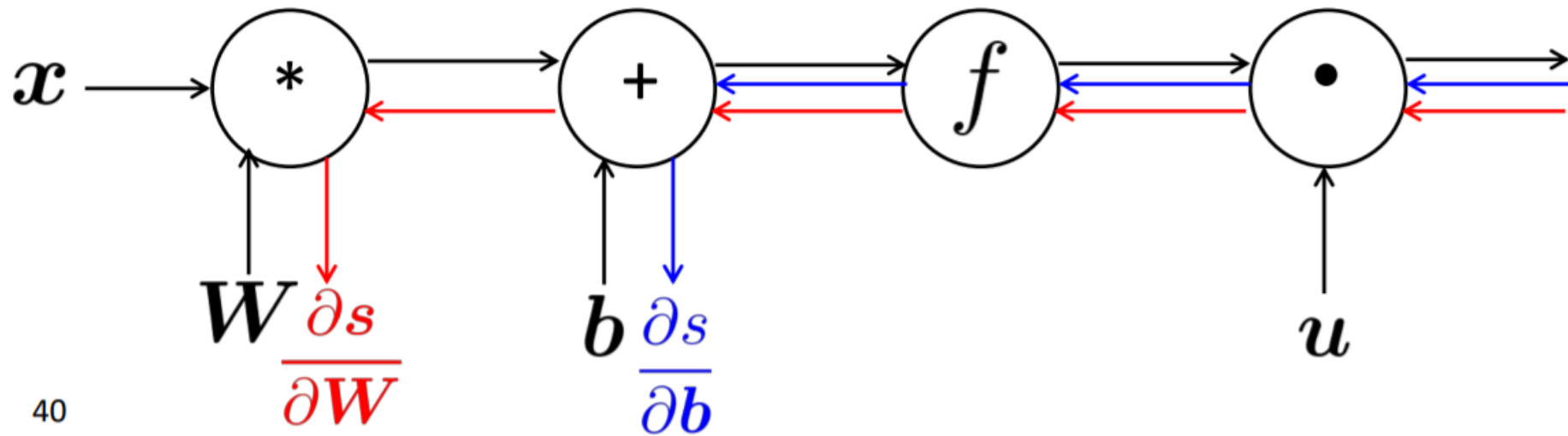
How to calculate $\frac{\partial z_j^{(2)}}{\partial w_{ij}^{(1)}}$?

- $\frac{\partial z_j^{(2)}}{\partial w_{ij}^{(1)}} = \frac{\partial}{\partial w_{ij}^{(1)}} \left(a_1^{(1)} w_{1j}^{(1)} + a_2^{(1)} w_{2j}^{(1)} + a_3^{(1)} w_{3j}^{(1)} + a_4^{(1)} w_{4j}^{(1)} \right)$
- $= \frac{\partial}{\partial w_{ij}^{(1)}} \left(\sum_k a_k^{(1)} w_{kj}^{(1)} \right)$
- $= a_i^{(1)}$



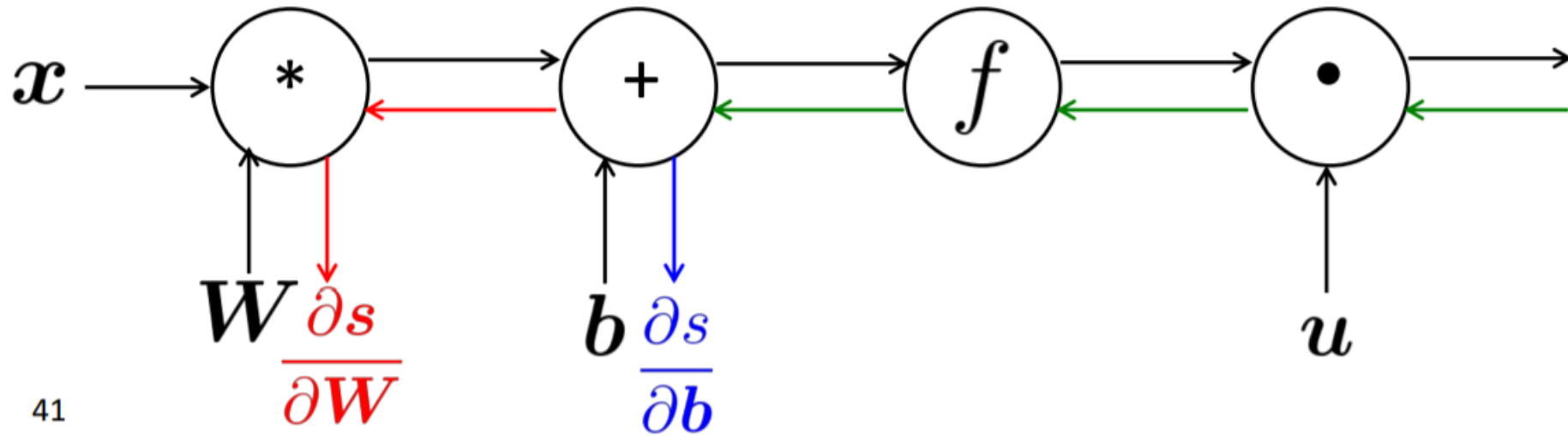
Inefficient Gradient Computation

- 未重複使用已經算過的梯度



Efficient Gradient Computation

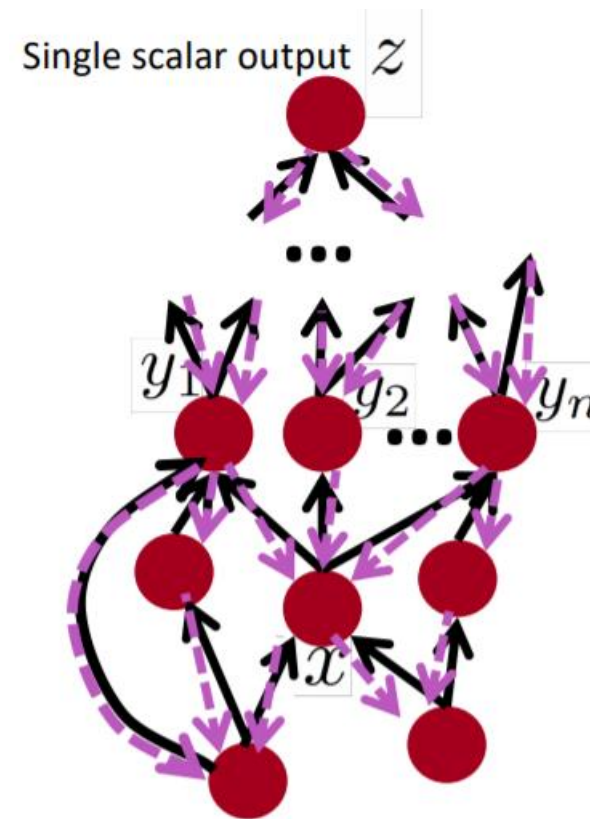
- 重複使用以計算過的梯度



41

計算圖及反向傳播總節

- 每個節點以拓樸排序
- 前向傳播
 - 以拓樸排序的方向計算每個值
- 反向傳播
 - 輸出的梯度初始化成1
 - 每個節點將以反拓樸方向走訪
- 如果計算正確前向傳播及反向傳播的演算法複雜度相同





Tricks and Tips on Neural Network

Regularization

- 為了避免Overfitting，損失函數將修改成下列公式：

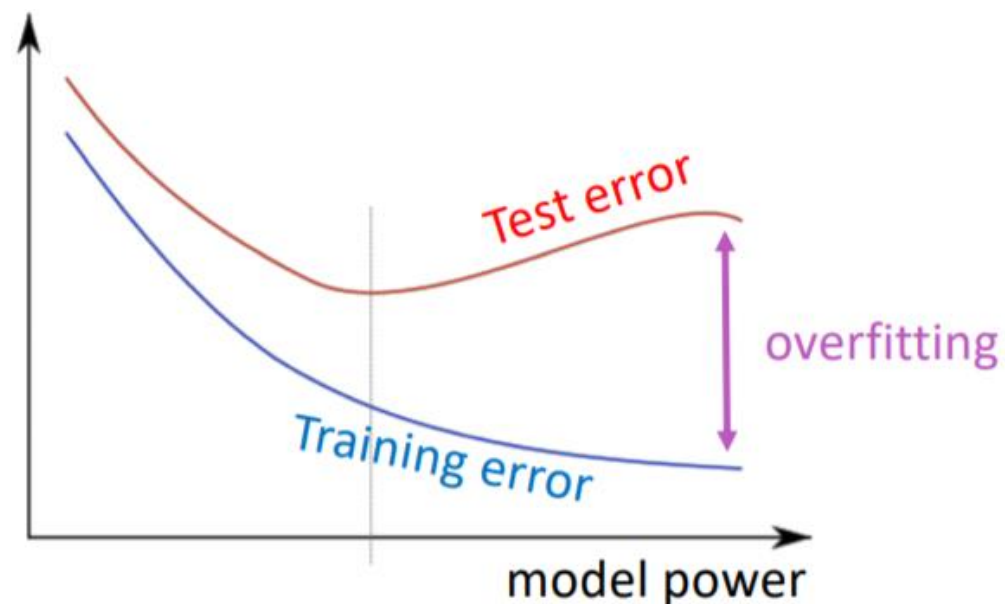
$$J(\theta) = \frac{1}{N} \sum_{i=1}^N -\log \left(\frac{\exp(f_{yi})}{\sum_{c=1}^C \exp(f_c)} \right) + \lambda \sum_k \theta_k^2$$

← L2 Regularization

↑
Hyper-Parameter

- Is it better to use less parameters?

- BERT Base 110 Million Parameters
- BERT Large 340 Million Parameters
- GPT-2 1.5 Billion Parameters
- GPT-3 175 Billion Parameters

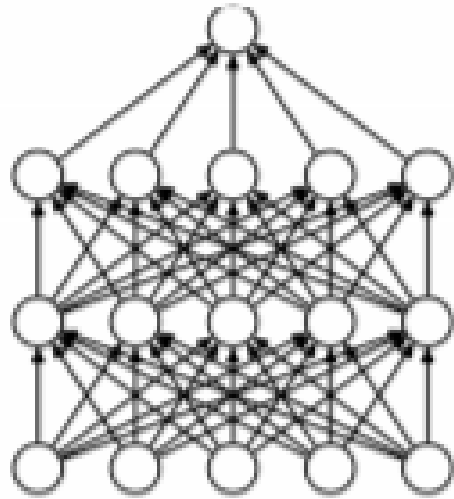


Regularization: λ

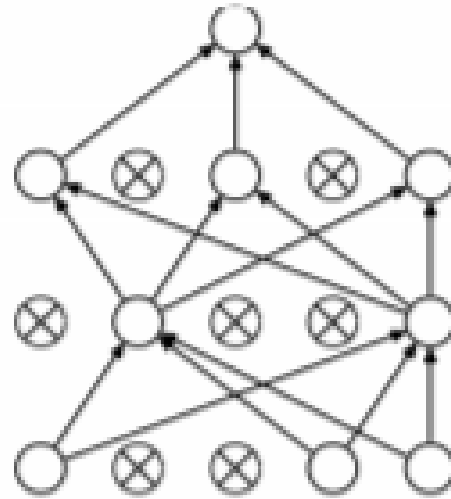
- 如果 λ
 - 太大：幾乎等於沒有訓練，因為權重幾乎接近於0
 - 太小：模型會在一次**Overfitting**，因為幾乎等於沒有正規化
- λ 為超參數，需藉由實驗找出有結果最佳的超參數。

Dropout

- 一種正規化的方法
- 訓練階段：只有 $P\%$ 個神經元被激活
- 預測階段：所有神經元都會被激活



(a) Standard Neural Net



(b) After applying dropout.

Vectorization

```
from numpy import random
```

```
N = 100  
d = 300  
C = 5  
W = random.rand(C,d)  
wordvectors_list = [random.rand(d,1) for i in range(N)]  
wordvectors_one_matrix = random.rand(d,N)  
%timeit [W.dot(wordvectors_list[i]) for i in range(N)]  
%timeit W.dot(wordvectors_one_matrix)
```

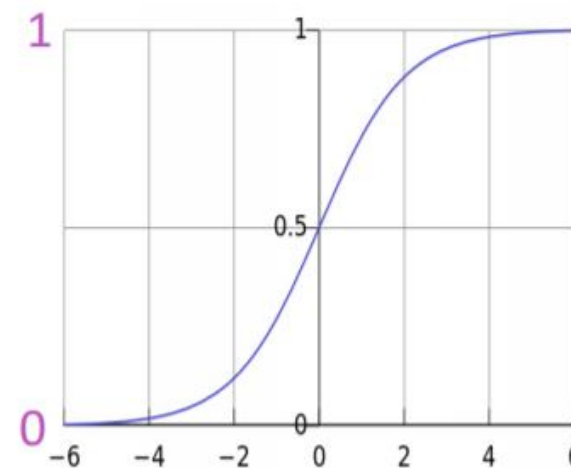
114 μ s \pm 2.76 μ s per loop (mean \pm std. dev. of 7 runs, 10000 loops each)
8.22 μ s \pm 267 ns per loop (mean \pm std. dev. of 7 runs, 100000 loops each)

Non-Linearity : Sigmoid

- 輸出的值介於0到1 之間
- 目前較少使用Sigmoid除非輸出值需要為機率分布

logistic (“sigmoid”)

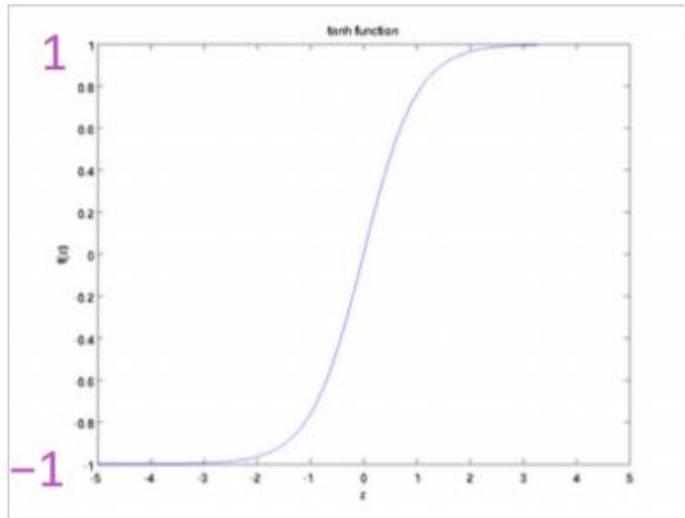
$$f(z) = \frac{1}{1 + \exp(-z)}.$$



Non-Linearities : tanh and hard tanh

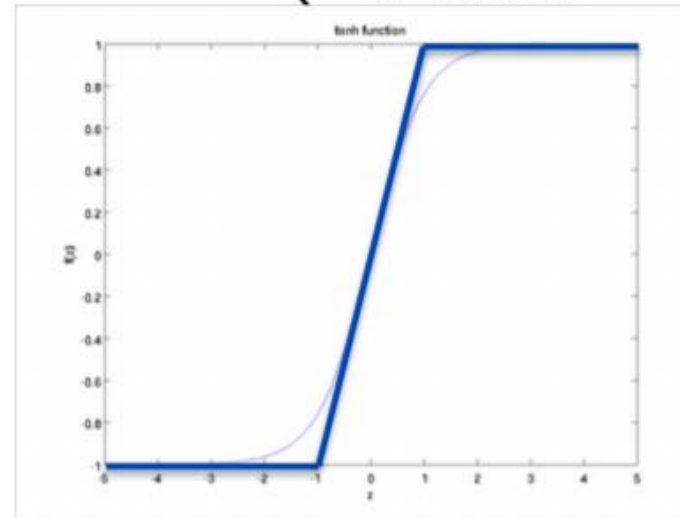
tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



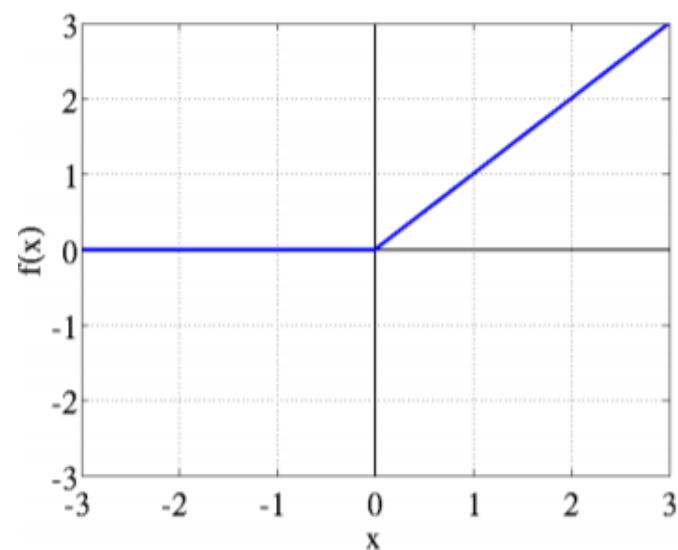
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$

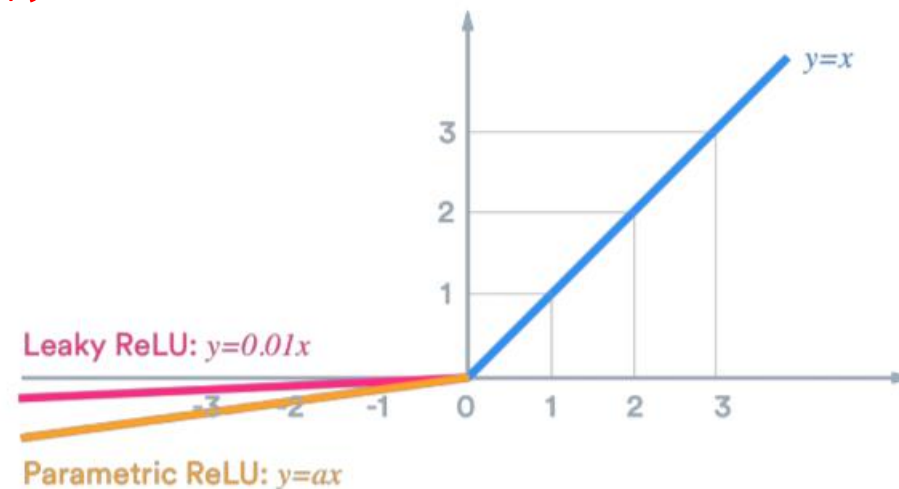


Better Non-Linearity

- $\text{ReLU} = \max(z, 0)$
- Computational Efficiency
- Dead Network Sometimes
- Leaky ReLU and Parametric ReLU



可以先嘗試使用Relu

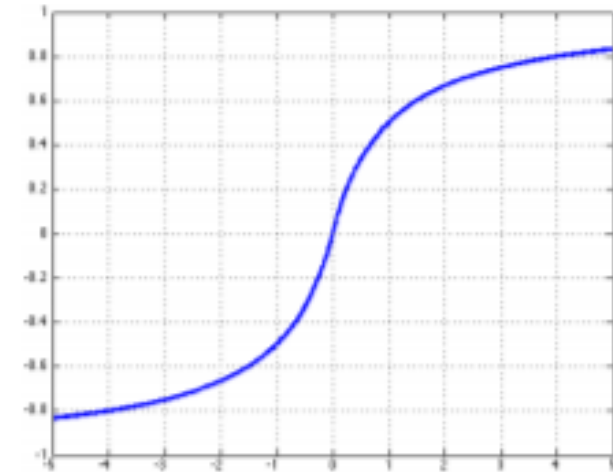


Non-Linearity : Maxout

- $maxout = \max(w_1^T x + b_1, w_2^T x + b_2)$
- 好處
 - 不會有Dead Network的問題
- 壞處
 - 需要兩倍的參數

Non-Linearity : Soft Sign

- $\text{softsign}(z) = \frac{z}{1+|z|}$
- 由於Softsign為未飽和函數，所以通常tanh會被取代成softsign



Derivative of Non-Linearities

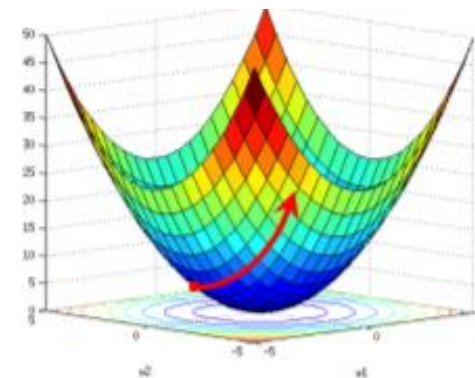
- $\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$
- $\text{hardtanh}(z) = \begin{cases} -1 & : z < -1 \\ z & : -1 < z < 1 \\ 1 & : z > 1 \end{cases}$
- $\text{softsign}(z) = \frac{z}{1+|z|}$
- $\text{ReLU}(z) = \max(z, 0)$
- $\text{Leaky ReLU}(z) = \max(z, k \cdot z)$
- $\tanh'(z) = 1 - \tanh^2(z)$
- $\text{hardtanh}'(z) = \begin{cases} 1 & : -1 < z < 1 \\ 0 & : \text{otherwise} \end{cases}$
- $\text{softsign}'(z) = \frac{1}{(1+z)^2}$
- $\text{rect}'(z) = \begin{cases} 1 & : z > 0 \\ 0 & : \text{otherwise} \end{cases}$
- $\text{leaky}'(z) = \begin{cases} 1 & : z > 0 \\ k & : \text{otherwise} \end{cases}$

Optimizer

- 能夠自己調整Learning Rate的Optimizer如下：
 - Adagrad
 - RMSprop
 - Adam
 - SparseAdam
 - ...

Learning Rate

- Learning Rate 的大小探討
 - 太大：無法收斂
 - 太小：收斂速度太慢
 - 可以朝10的次方來修正Learning Rate
- 如果使用會自我修正的Optimizer一開始可以使用較大的Learning Rate





自然語言處理之分散式表示法

詞庫

- 類似同義詞辭典
- 將意思相似或相近的語詞分類在相同群組
- 範例：Car = Auto Automobile Machine Motorcar
- 最有名的詞庫為詞網 (WordNet)

詞庫的問題

- 難以因應時代變遷
 - Crowdfunding (群眾募資)
 - Covidiot (防疫豬隊友)
 - LOL (大聲地笑)
- 人工作業成本昂貴
 - 英文單字總數約 1000 萬個，目前WordNet登入大約20萬個
- 無法展現出語詞的細微語感差異
 - Vintage：過去最好的
 - Retrospective：過往的

計數手法

- 讓電腦自動從文本中找出語詞之間的知識
- 計數手法的演算法
 - 共生矩陣
 - 點間互資訊

分布假說

- 一個語詞的意思是根據周圍的語詞而生成。

- 範例

- 我想喝飲料。
 - 他想喝果汁。

- 目前最新的方法BERT, GPT-1, GPT-2, GPT-3並未根據分布假說。
- 他們強調根據整個句子來做編碼。
- 他們的表現遠優於採用分布假說的演算法。
- 然而：他們所需的參數量也較多，運算量也較大。

上下文 (Context)

- 中心字詞左右兩邊的語詞
- 大小通常為對稱
- 一般會以視窗大小 (**Windows Size**)來表示上下文大小
- 範例
 - 如果視窗大小為**1**，代表上下文為左右各**1**個語詞
 - 如果大小為**2**，代表上下文為左右各**2**個語詞
 - ...

共生矩陣

- 假設我們有下列句子：
 - I like deep learning.
 - I like NLP.
 - I enjoy flying.
- 假設視窗大小為1 (通常為5~10)
- 共生矩陣如下圖

上述句子之共生矩陣

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

共生矩陣的缺點

- 容易被停用字影響

- 範例：

- The
- Car
- Drive

- 因此：我們需要一種比較不受停用字影響的方法

點間互資訊

- $PMI(x, y) = \log_2 \frac{p(x,y)}{p(x)p(y)}$, $p(x)$ 為字詞 x 所出現的機率, $p(x,y)$ 為字詞 x 與 y 共同出現的機率

$$= \log_2 \frac{\frac{c(x,y)}{N}}{\frac{c(x)}{N} \frac{c(y)}{N}}, c(x) = \text{字詞 } x \text{ 出現的次數}, N \text{ 為語料庫內所有語詞的數量}$$

$$= \log_2 \frac{c(x,y) * N}{c(x)c(y)}$$

上下同乘 N 的平方

點間互資訊的問題

- $\text{PMI}(x, y) = \log_2 \frac{p(x,y)}{p(x)p(y)}$
- $\log 0$ 為負的無窮大
- 所以公式可以修改為： $\text{PPMI}(x, y) = \max(0, \text{PMI}(x, y))$

點間互資訊範例

- 假設 $N = 10000$, $C(\text{the})$ 為 1000, $C(\text{car})$ 為 20, $C(\text{drive})$ 為 10, $C(\text{the, car})$ 為 10, $C(\text{car, drive})$ 為 5, 點間互資訊如下：
- $\text{PMI}(\text{the, car}) = \log_2 \frac{10 \cdot 10000}{1000 \cdot 20} \approx 2.32$
- $\text{PMI}(\text{car, drive}) = \log_2 \frac{5 \cdot 10000}{20 \cdot 10} \approx 7.97$

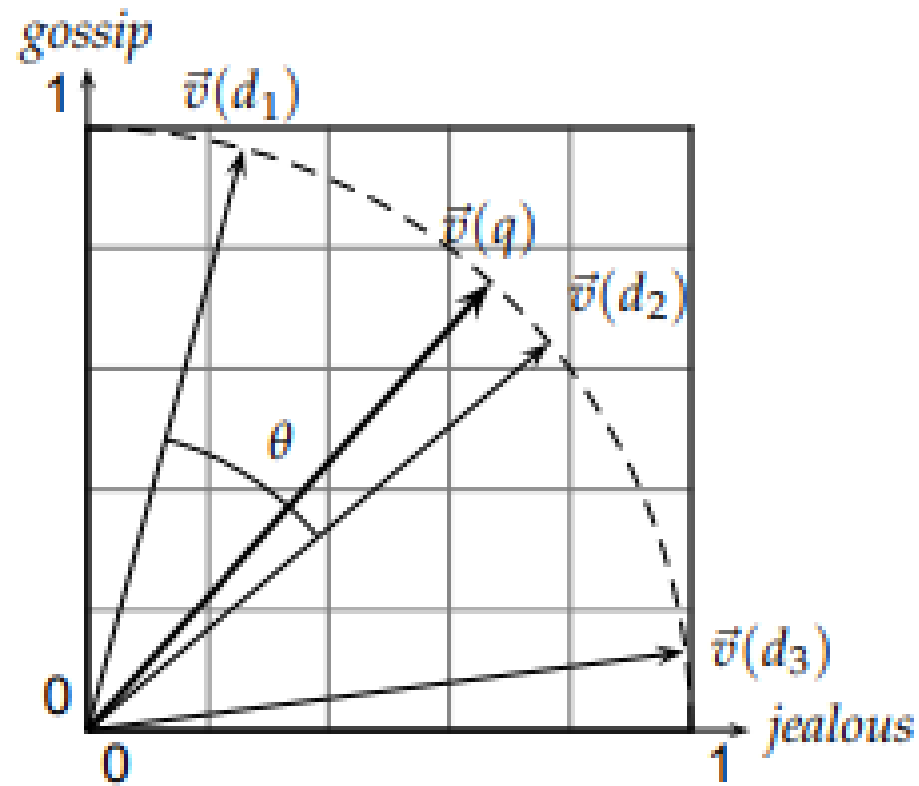
點間互資訊及共生矩陣的問題

- 儲存：隨著字數增加，需儲存的資料也增加
- 稀疏矩陣：可以由上述共生矩陣發現矩陣會有許多的0
- 穩健度：由於矩陣太稀疏也讓我們模型穩健度降低 (易受雜訊影響)
- 解決方法：
 - 降維 (不是最好的解決辦法)



向量間的相似度

Cosine Similarity

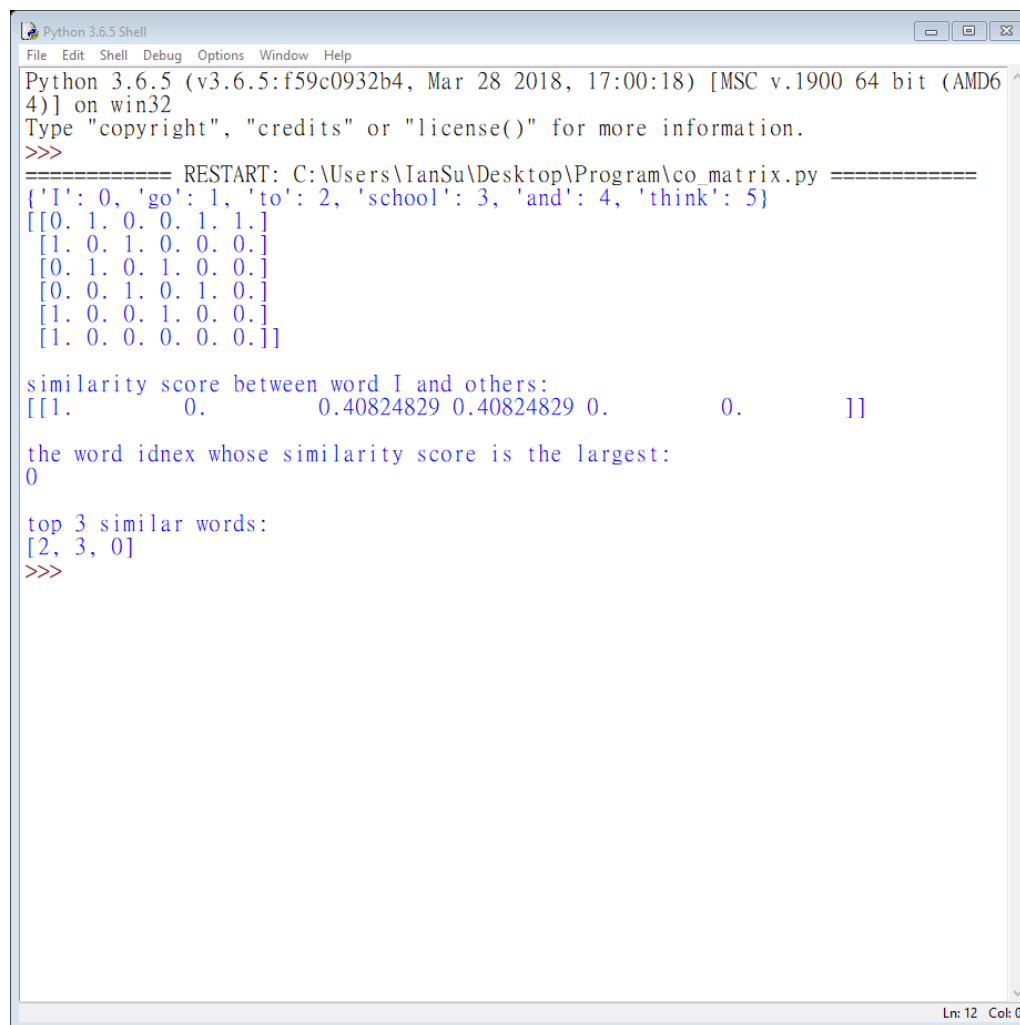


- Image Source: <https://nlp.stanford.edu/IR-book/pdf/06vect.pdf>

Formula of Cosine Similarity

- $\text{Similarity}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \sqrt{\sum_i y_i^2}}$
- 如果將每個向量變成單位向量，相似度公式修改如下：
- $\text{Similarity}(x, y) = x \cdot y = \sum_i x_i y_i$

相似度計算程式範例



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\IanSu\Desktop\Program\co_matrix.py =====
{'I': 0, 'go': 1, 'to': 2, 'school': 3, 'and': 4, 'think': 5}
[[0. 1. 0. 0. 1. 1.]
 [1. 0. 1. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0.]
 [0. 0. 1. 0. 1. 0.]
 [1. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0.]]

similarity score between word I and others:
[[1.         0.         0.40824829 0.40824829 0.         0.         ]]

the word idnex whose similarity score is the largest:
0

top 3 similar words:
[2, 3, 0]
>>>
```

Cosine Similarity 探討

- 如果相似度分數為
 - **1**：兩個向量相同，也就是兩個語詞意思或是語法相同
 - **-1**：兩個向量相反，也就是兩個語詞意思或是語法相反



降維 (Dimension Reduction)

奇異值分解 (SVD)

- 將原本的向量 X 分解成三個矩陣的乘積
- SVD的數學表示法：
- $X = USV^T$ ， U 及 V 為正交矩陣， S 為對角矩陣。
- 最後再從 U 刪除不重要的元素

奇異值分解：如何找出三個矩陣

- $X = USV^T$
- $XX^T = USV^T \boxed{V^T V} S^T U^T$
 $= \boxed{U S S^T} U^T$
- $X^T X = VS^T U^T USV^T$
 $= VS S^T V^T$

S^2

Reference:

https://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/positive-definite-matrices-and-applications/singular-value-decomposition/MIT18_06SCF11_Ses3.5sum.pdf

https://www.youtube.com/watch?v=mBcLRGuAFUk&t=2s&ab_channel=MITOpenCourseWare

特徵值分解 (Eigen-Decomposition)

- 由特徵向量以及特徵值組合而成
- 假設我們有一個向量 \mathbf{A} 及一個滿足於下列式子的特徵向量 \mathbf{u}
- $\mathbf{A}\mathbf{u} = \lambda\mathbf{u} \quad (1)$
- 我們可以將所有特徵向量放到同一矩陣並且將特徵值放到一個對角矩陣(Λ)
- 因此，可以將公式(1)改寫成下列公式：
- $\mathbf{A}\mathbf{U} = \mathbf{U}\Lambda$
- a.k.a. $\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^{-1}$

Positive Semi-Definite Matrix A

- 如果A是一個Semi-Definite Matrix ，
 - $A = XX^T$
 - 特徵值永遠為正
 - 如果特徵值都不同，代表矩陣為正交。
- 如果一個矩陣為正交，
- $MM^T = I$
- 因此，特徵值公式改寫如下：
- $A = U\Lambda U^T$

降維後及降維後的向量

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\IanSu\Desktop\Program\co_matrix.py =====
{'I': 0, 'go': 1, 'to': 2, 'school': 3, 'and': 4, 'think': 5}
Co-matrix before dimension reduction:
[[0. 1. 0. 0. 1. 1.]
 [1. 0. 1. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0.]
 [0. 0. 1. 0. 1. 0.]
 [1. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0.]]

U matrix after dimension reduction:
[[-5.22966014e-01 -6.63504341e-01 -1.33248950e-15 -5.00000000e-01
  -8.91708026e-17  1.90443004e-01]
 [-4.29374351e-01  4.39042241e-01  3.71748034e-01  2.22044605e-16
   6.01500955e-01  3.50541834e-01]
 [-3.85121039e-01 -1.53468031e-01 -6.01500955e-01  5.00000000e-01
   3.71748034e-01 -2.79516276e-01]
 [-3.85121039e-01 -1.53468031e-01  6.01500955e-01  5.00000000e-01
   -3.71748034e-01 -2.79516276e-01]
 [-4.29374351e-01  4.39042241e-01 -3.71748034e-01 -6.10622664e-16
   -6.01500955e-01  3.50541834e-01]
 [-2.47276065e-01  3.56568279e-01  7.25133404e-15 -5.00000000e-01
  -2.30632210e-16 -7.49475557e-01]]
>>>
```

References

- 齋藤康毅, Deep Learning 2: 用Python進行自然語言處理的基礎理論實作.
- Manning et al., CS224n Natural Language Processing with Deep Learning, Stanford University.
- Hervé Abdi, The Eigen-Decomposition: Eigenvalues and Eigenvectors
- D.Manning, Introduction of Information Retrieve.