自然語言處理
詞嵌入語言模型

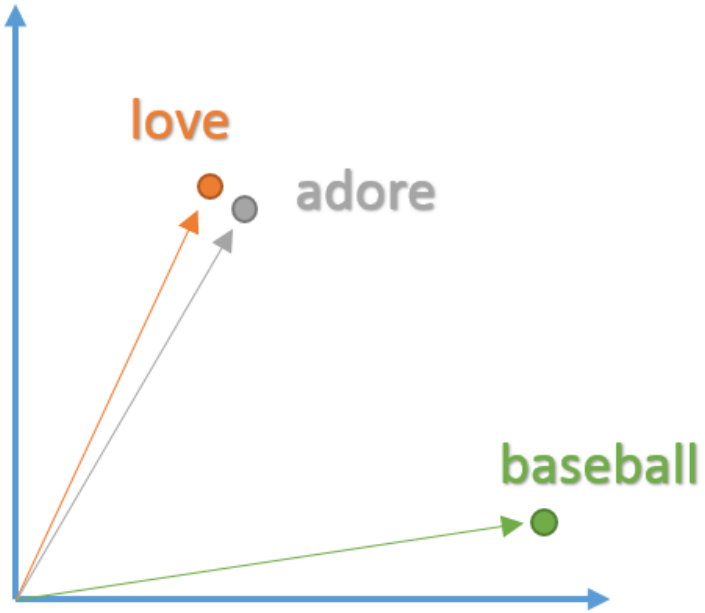
Instructor: 馬豪尚

詞嵌入Word Embedding

- 一般進行NLP(自然語言處理)時，資料最小的單位是詞(word)，由詞組成句子，句子再組成段落、篇章、文檔。
- 很多處理NLP的問題的前處理程序，首先就要思考詞(word)的處理。
- 經過分詞和清理處理過後，仍然是人類的語言抽象表示，通常以符號形式來表現的（例如中文、英文、拉丁文等等），所以需要把他們轉換成數值形式，讓機器能夠理解和計算
- 轉換成數值形式，或者說"嵌入"到一個數學空間裡，這種嵌入方式，就叫詞嵌入(word embedding)

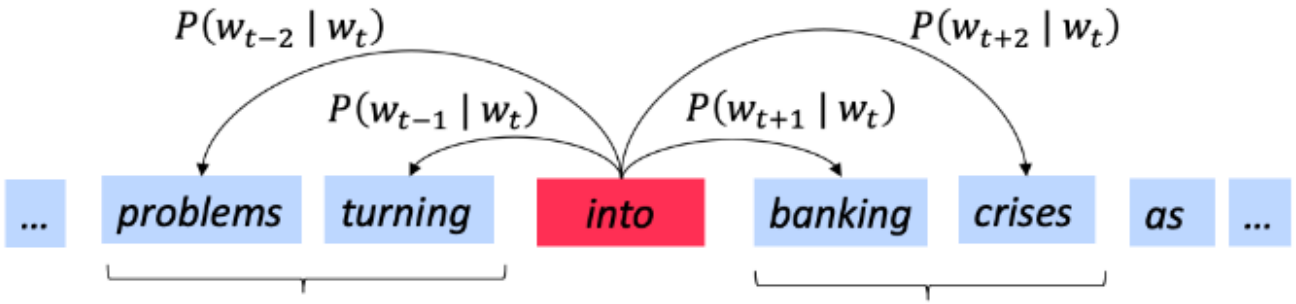
詞嵌入Word Embedding

- 語意相似度資訊
  - 「物以類聚」是我們耳熟能詳的一句諺語
  - 「分布假說」是在語言學的脈絡裡，語言學家認為在相同上下文中一起出現的兩個單詞會有相似的意義
- 詞嵌入的優點
  - 維度縮減（ diemension reduction ）
  - 上下文相似性（ context similarity ）



Word2vec

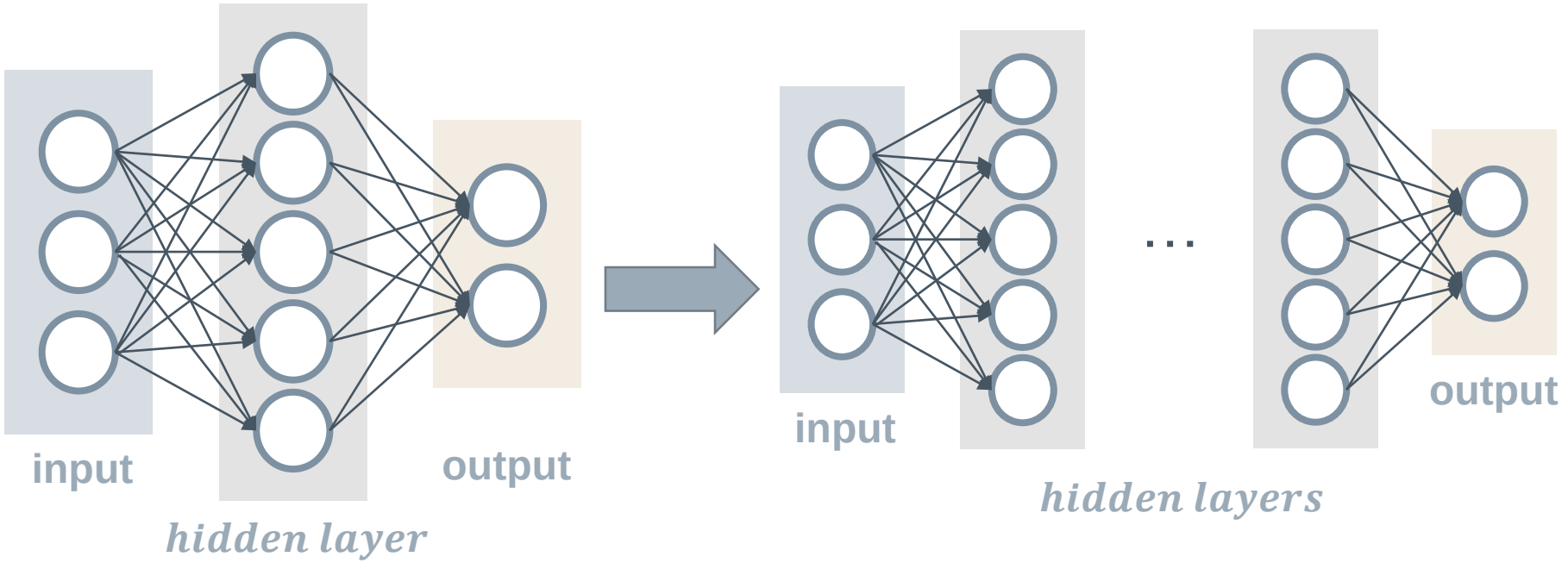
- 在NLP中，把x看做一個句子裡的一個詞，y是這個語詞的上下文(context)詞
- 這個模型的目的，就是判斷(x, y)這個樣本，是否符合自然語言的規則
  - 在自然語言中，將這兩個詞放在一起是否會成立
  - 當這兩個詞一起出現是否像是人寫的句子
  - 是不是這兩個詞會常常一起被使用成為上下文關係
- Word2vec正是來自於這個想法



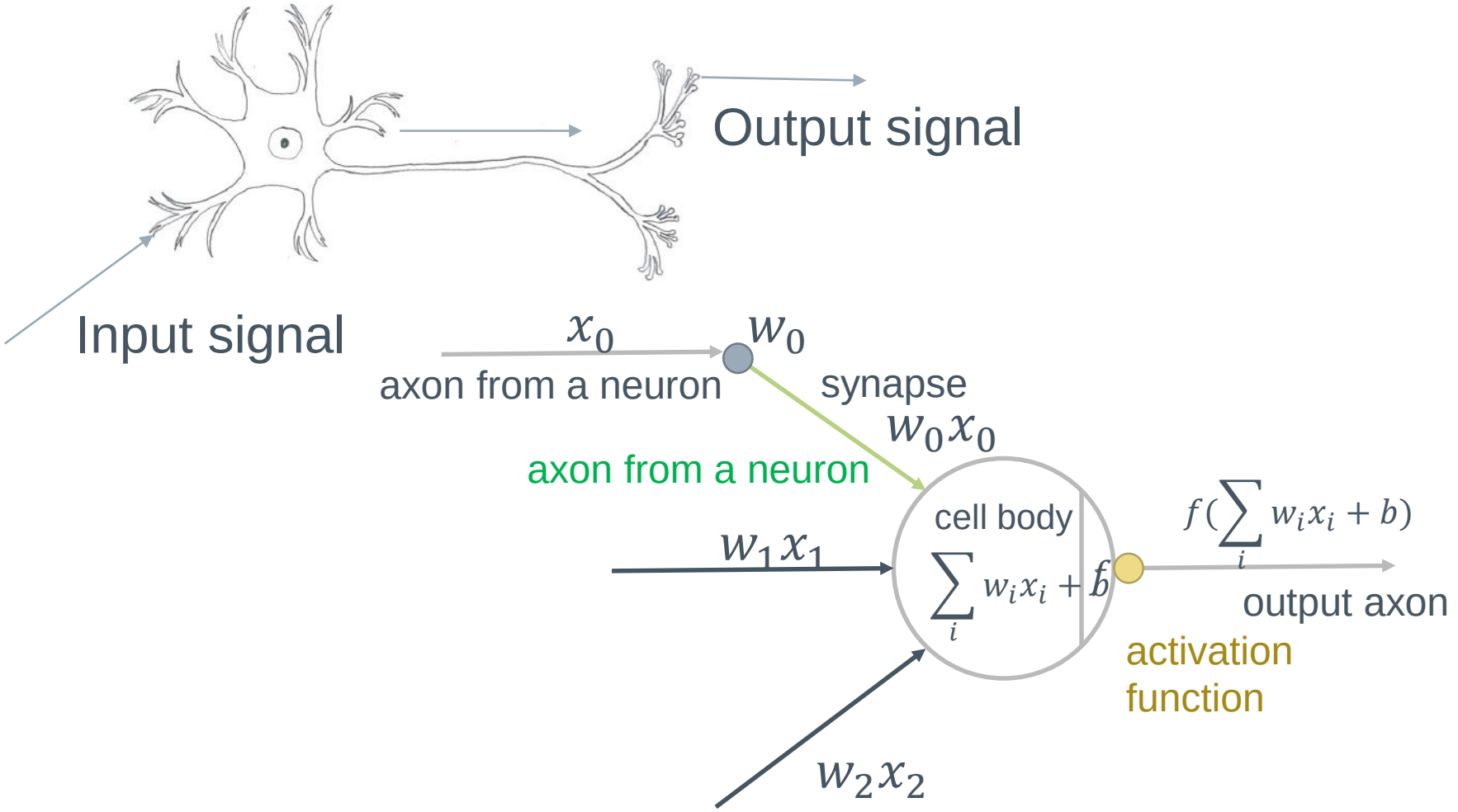
Word2vec

- 透過上下文關係的輸入方式來訓練一個自然語言模型
- 儲存模型訓練完後的副產物-"模型參數"
  - 這裡指的是類神經網絡的權重，並將這些參數，作為某個輸入詞的向量化表示
  - 這個向量便叫做詞嵌入向量(word embedding)
- 整個過程就是將詞(word)轉成向量(vector)

基本類神經網路架構

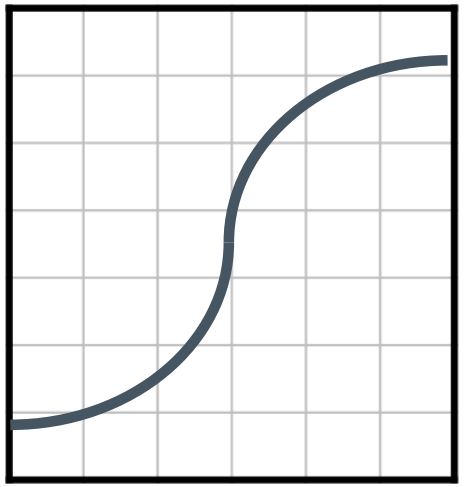


神經元的設計

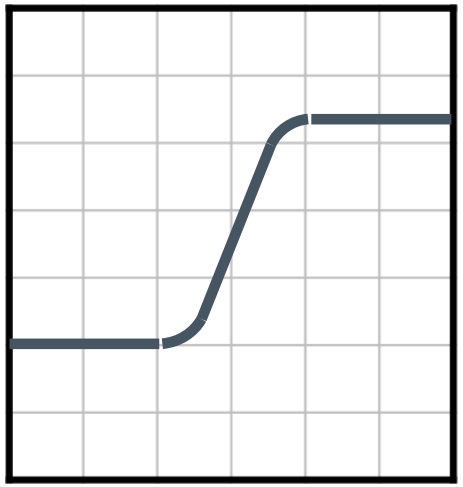

Output signal
Input signal
x_0
w_0
axon from a neuron
synapse
w_0x_0
axon from a neuron
cell body
$\sum_i w_i x_i + b$
$f(\sum_i w_i x_i + b)$
w_1x_1
output axon
activation function
w_2x_2

Activation Function

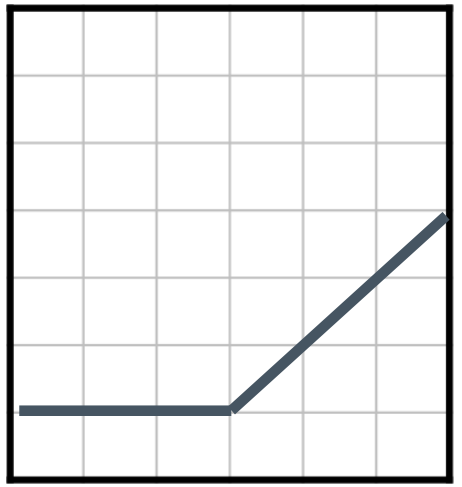
graphs



Sigmoid



tanh



ReLU

equations

$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = 2\sigma(2x) - 1$$

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

ranges

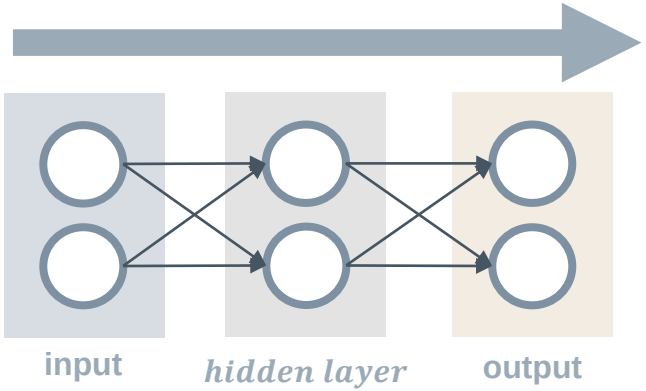
$$0 < \sigma(x) < 1$$

$$-1 < \tanh(x) < 1$$

$$0 < ReLU(x) < 1$$

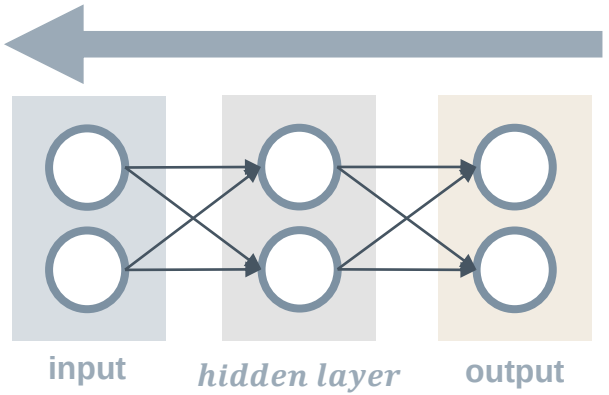
如何訓練一個類神經網路

- 輸入資料(向量表示)
- 計算出每個神經元的輸出
  - Forward propagation

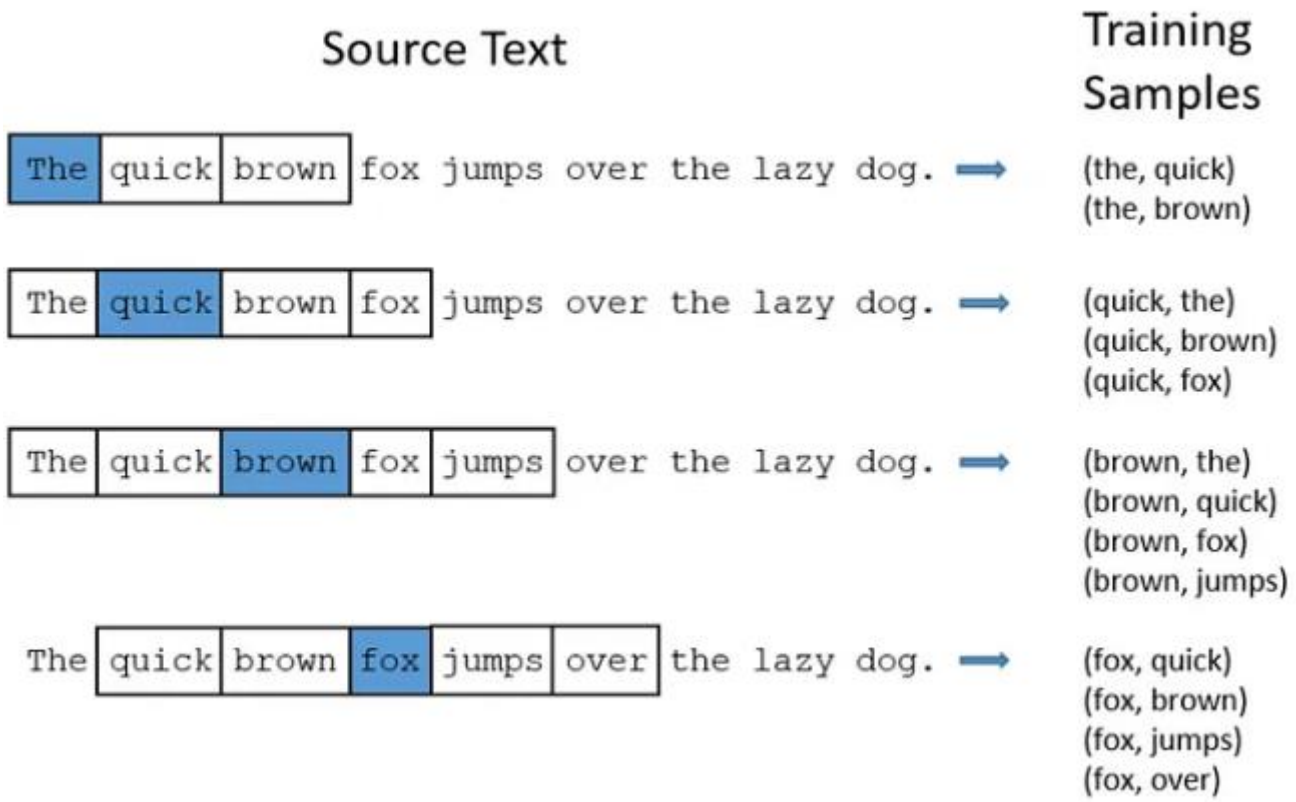


- 計算輸出結果和正確答案之間的錯誤程度
  - Loss Function

- 根據錯誤去重新調整神經網路權重
  - Back propagation

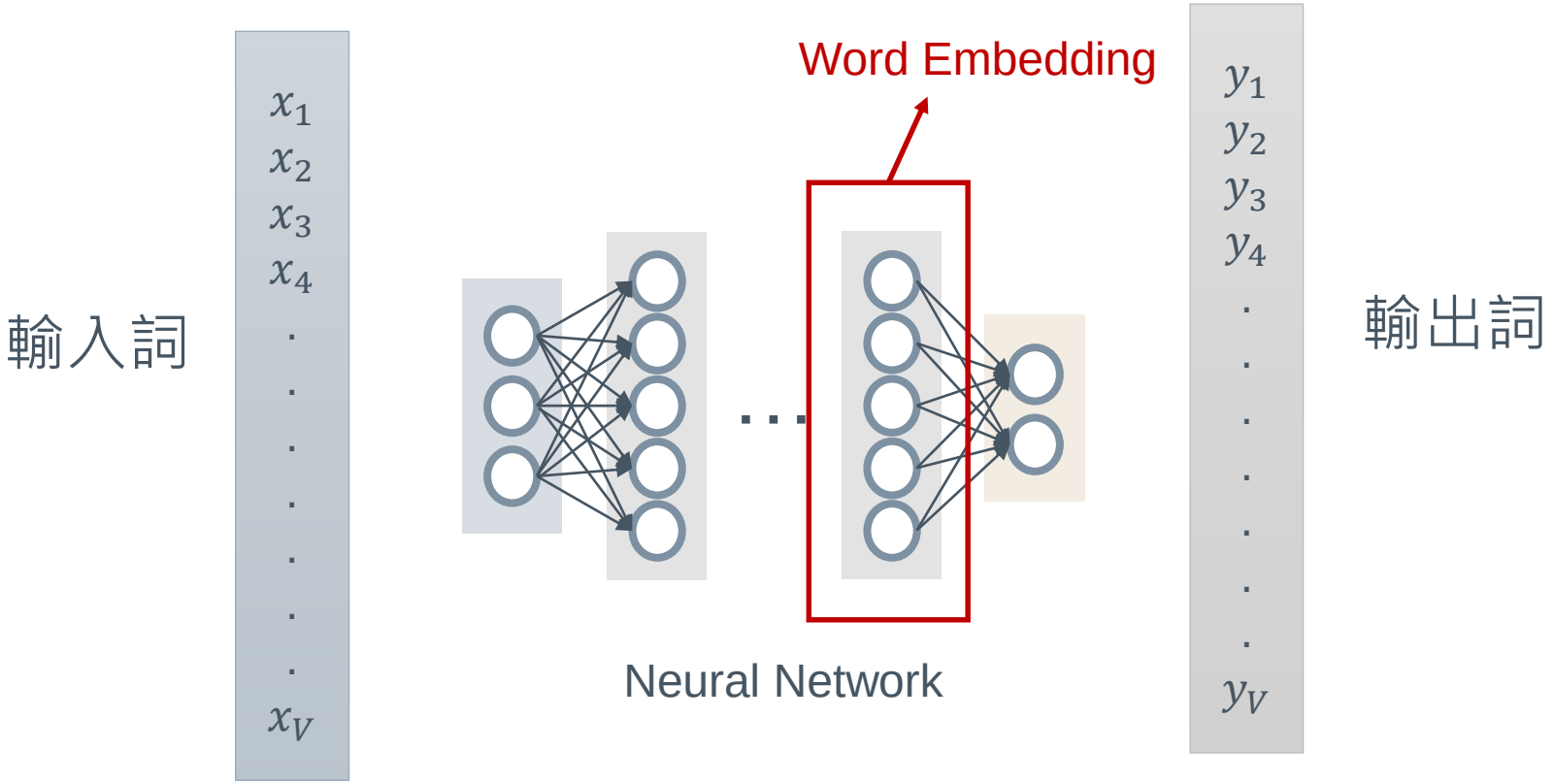


詞嵌入訓練資料產生



圖片來源: Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, “Efficient Estimation of Word Representations in Vector Space,” 2013

訓練詞嵌入的基本範例


輸入詞
x_1
x_2
x_3
x_4
x_V
Word Embedding
Neural Network
y_1
y_2
y_3
y_4
y_V
輸出詞

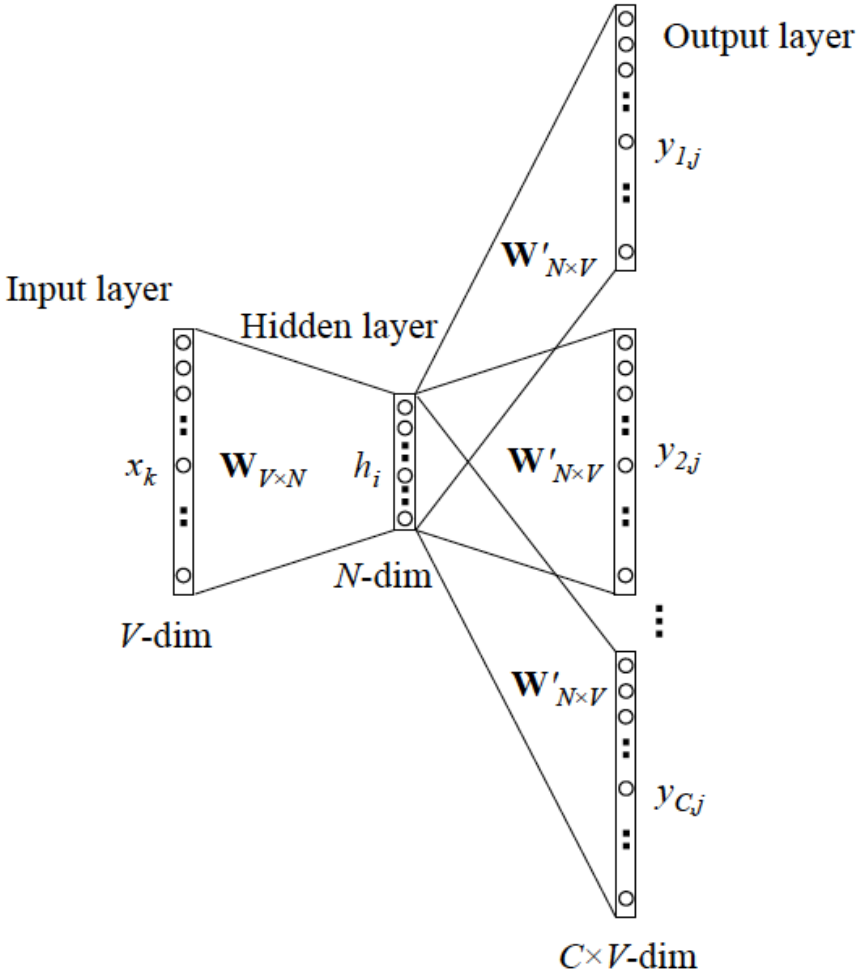
Word2vec

- Word2Vec的模型根據訓練方式的不同有不同的模型
  - Skip-Gram模型
  - CBOW(Continuous Bag of Words)模型

- Skip-Gram為用一個詞作為輸入，來預測它周圍的上下文詞
- CBOW 則是輸入上下文的詞去預測目標詞

Skip-Gram模型

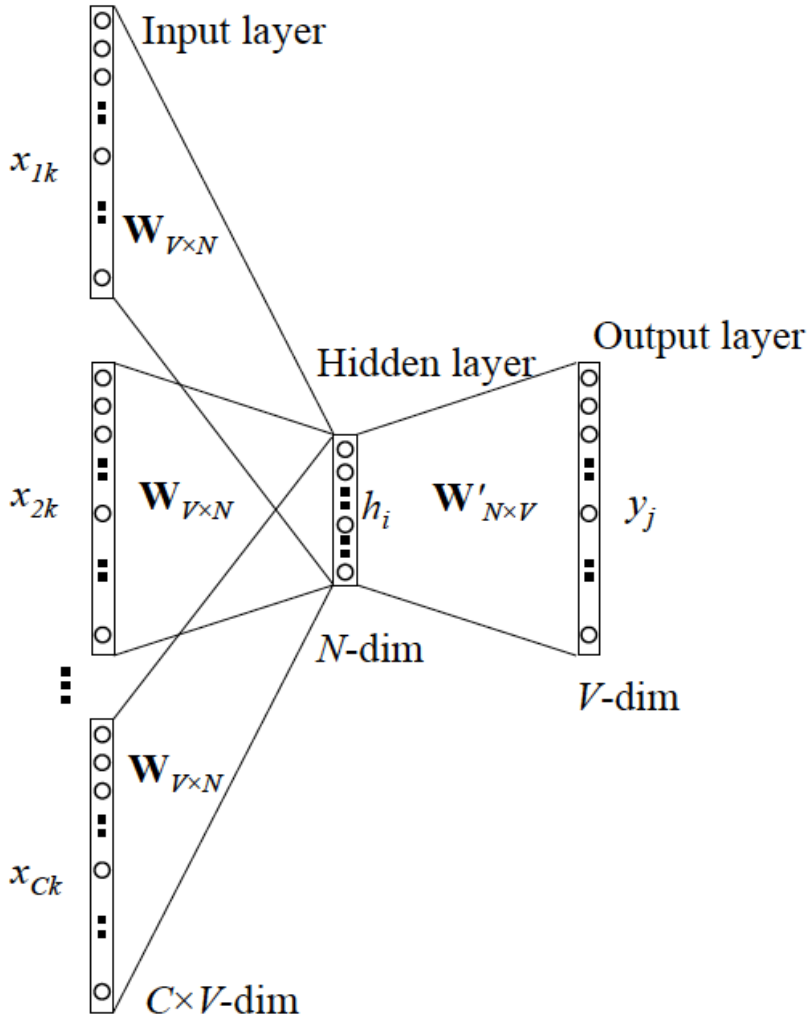
用一個詞
作為輸入



預測它周圍
的上下文詞

CBOW模型

用上下文詞
作為輸入



預測目標一個詞

Global Vectors for Word Representation(GloVe)

- 基於共同出現頻率統計的概念
  - 兩個字是否常常一起出現
- 共現矩陣(Co-occurrence Matrix)

輸入句子 He is not lazy. He is intelligent. He is smart.
Context window size = 2

	He	Is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
Is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

GloVe的共現矩陣

- 考慮到共同出現時兩個詞的距離衰減
  - 距離越遠的兩個單詞所佔總計數越小
- 提出了一個衰減函數（decreasing weighting）
  - 根據兩個單字在上下文視窗的距離，用於計算權重
  - 乘上距離的倒數(1/d)
- 訓練出的詞嵌入向量想要符合這個共現矩陣

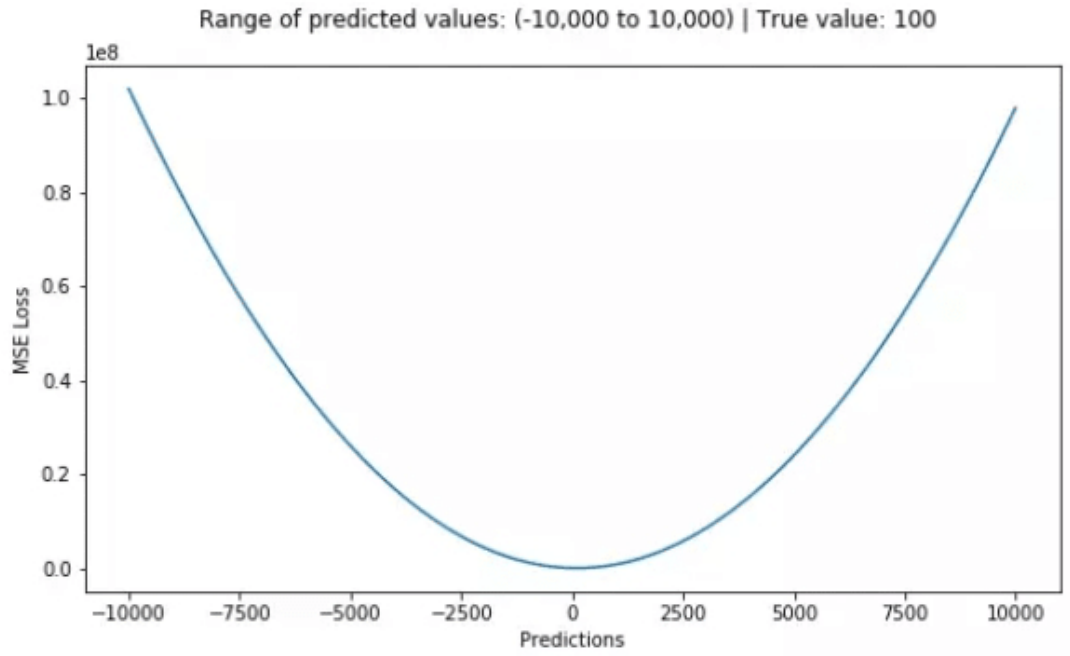
$$w_i^T \tilde{w}_j + b_i + \tilde{b}_j = \log(X_{ij})$$

w_i和w_j為訓練出的詞嵌入向量，X_{ij}為共現矩陣

GloVe的損失函數(Loss Function)

› Mean Square Error 均方誤差

$$MSE = \sum_{i=1}^{n} (y_i - y_i^p)^2$$



MSE損失（Y軸）-預測值（X軸）

GloVe的損失函數(Loss Function)

› Loss Function

$$J = \sum_{i,j=1}^{V} f(X_{ij})\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij})\right)^2$$

MSE

› $f(X_{ij})$為一個權重函數其目的如下

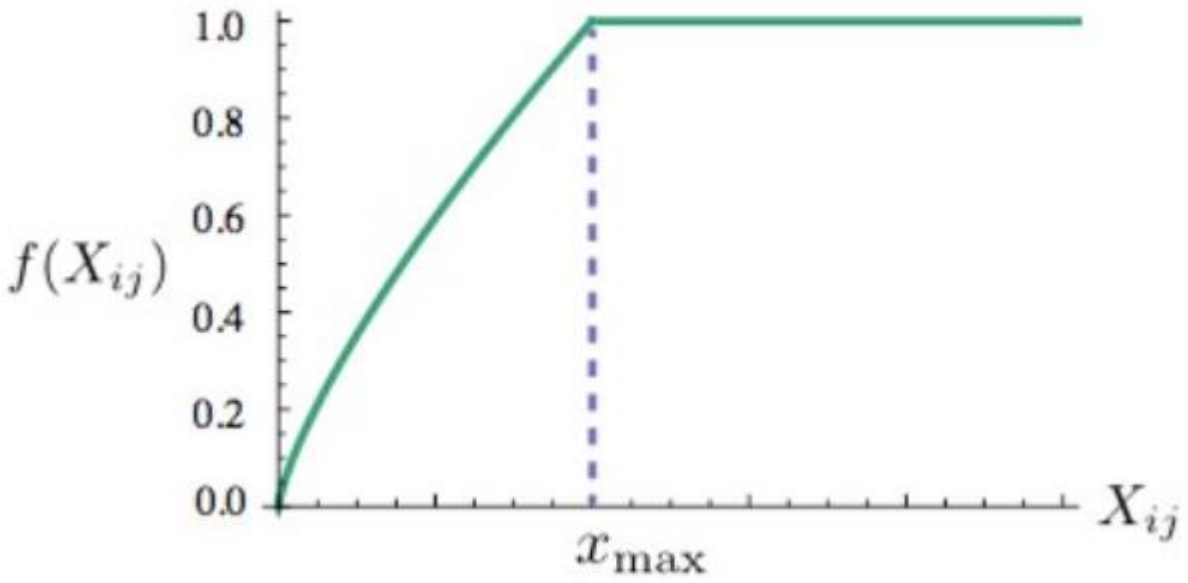
– 這些單字的權重要大於那些很少在一起出現的單字（rare co-occurrences），所以這個函數要是非遞減函數（non-decreasing）

– 但我們也不希望這個權重過大（overweighted），當到達一定程度之後應該不再增加

– 如果兩個單字沒有在一起出現，也就是，那他們應該不參與到loss function的計算當中去

GloVe的損失函數(Loss Function)

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$



Gensim套件

- Gensim套件
  - 使用unsupervised machine learning處理原始的、非結構化的文本（text），藉由統計training documents在語料庫中字與字之間組合或共同出現(co-occurrence)的模式
  - 安裝:
    - pip install --upgrade genism

Gensim核心概念

- Document(文章)
  - 一群文字。可以是 140 個字的簡短推文、單個段落（即期刊文章摘要）、新聞文章或書籍中的任何內容。
- Corpus(語料庫)它在Gensim中扮演兩個角色
  - 當做 model 的輸入。Model 藉由 training corpus 來初始化模型內部的參數。
  - 可以被處理後並組織成document。當一個關鍵字提取模型被訓練後，可以從新的 document (在training corpus中未看到的 document) 中提取 新的關鍵字。

Gensim核心概念

- Vector為具有數學意義的 document
  - 帶有 features 的 vector 正好可以滿足這個需求，而一個 single feature 可以被認為是一個 question-answer 對，questions 通常以整數的方式標示，answer 只被允許是浮點數 (single floating point number)。
    - Document 中單詞splonge出現幾次? 0次 (bag of words詞頻統計)
      - (1, 0.0)
    - Document 中由多少段落組成? 2個
      - (2, 2.0)
    - Document 使用了多少種字體？ 5種
      - (3, 5.0)
  - 可以將事前已知的所有 questions 隱藏起來，簡單地將 document 表示成 (0, 2, 5)

Gensim核心概念

- Model
  - 一種將 vector 從一種表示轉換為另一種表示的演算法
  - 透過訓練的過程學習不同的特徵來達到
  - 一旦你創建了 model，你就可以用它做各種很酷的事情。 例如，通過 Tf-Idf model轉換整個語料庫並對其進行索引，為相似性查詢 (similarity queries) 做準備

詞嵌入向量

- 常見的word embedding模型
  - Word2vec
    - 是用來產生詞向量的相關模型。用來訓練以重新建構語言學之詞文本，輸入以詞為單位，並且需猜測相鄰位置的輸入詞
  - GloVe (2014)
    - 預測字跟字同時出現的次數（co-occurrence count）來訓練。也是滿早期的model，在小一點的 dataset 也能有效訓練。
- 下載預先訓練好的詞嵌入向量(Glove)
  - https://github.com/stanfordnlp/GloVe

Gensim 載入預訓練好的模型

- 載入genism 的word2vec模組
  - from gensim.models import word2vec
- 把 GloVe word vectors 儲存格式轉為Gensim可以用的word2vec 格式
  - 指定glove文件輸入位置
    - glove_file = './data/glove.6B.100d.txt'
  - 指定word2vec格式輸出文件位置
    - word2vec_glove_file = './data/glove.6B.100d.word2vec.txt'
  - 用gensim提供的函數轉換格式
    - glove2word2vec(glove_file, word2vec_glove_file)

Gensim 載入預訓練好的模型

- 載入預訓練好的詞嵌入向量
  - WordVector = KeyedVectors.load_word2vec_format(word2vec_glove_file)
    - 預訓練好的模型是用KeyedVectors載入
- 使用模型
  - WordVector.函數名稱()

KeyedVectors的函數

- 查詢指定詞的詞嵌入向量
  - wv = WordVector[word]
  - .get_vector('word', norm=True)
    - norm:是否做normalize
- 查詢模型所有的詞
  - .vectors
    - 回傳一個numpy.ndarray，包含所有的詞嵌入向量
  - .index_to_key
    - 回傳一個列表包含所有的詞，依照詞的index位置排序
  - .key_to_index
    - 回傳一個字典包含所有的詞和詞的index位置

KeyedVectors的函數

- 找到與指定詞top-K最相似的詞和 similarity score
  - .most_similar(positive[word])
- 計算詞跟詞的相對關係，依照給定的一組兩個詞的關係，找出跟指定詞有相對類似關係的詞
  - .most_similar(positive=[y_1, x_2], negative=[x_1])
    - 如果美國相對於漢堡，那加拿大相對於什麼?
    - 日本相對於日文，等於法國相對於什麼?

us-ham=can-?
us-ham-can = -?
-us +ham +can = ?
han+can -us = ?

使用Gensim自己訓練Word2vec

- 訓練模型
  - model = word2vec.Word2Vec(參數)
- 設定模型參數
  - train_data, (訓練資料)
  - min_count=1, (詞頻少於 min_count 的詞不會參與訓練)
  - vector_size=100, (詞嵌入向量的維度)
  - workers=8, (訓練並行的數量)
  - epochs=10, (訓練的迭代次數)
  - window=10, (選取周圍詞的數量)
  - sg=0, (0或1兩種，0是CBOW，1是Skip-Gram)
  - seed=546, (亂數種子，模型隨機選取起始點)
  - batch_words=1000, (每次給予多少詞來訓練)

已訓練模型儲存和調用

- 模型儲存
  - model.save(“model_name")
- 模型讀取
  - from gensim import models
  - model = models.Word2Vec.load(" model_name ")
- 模型接續訓練新詞
  - model.train([[“你好", "world"]], total_examples=1, epochs=1)
    - 第一個參數為要訓練的句子的串列
    - total_examples為要訓練的句子數量
    - Epochs為迭代次數

使用已訓練的模型

- 查詢指定詞的詞嵌入向量
  - wv = model.wv[word]
  - model.wv.get_vector('word', norm=True)
- 查詢模型所有的詞
  - model.wv.vectors
    - 回傳一個numpy.ndarray，包含所有的詞嵌入向量
  - model.wv.index_to_key
    - 回傳一個列表包含所有的詞，依照詞的index位置排序
  - model.wv.key_to_index
    - 回傳一個字典包含所有的詞和詞的index位置
- 儲存詞嵌入向量
  - model.wv.save("檔案名稱")

使用已訓練模型

- 找到與指定詞top-K最相似的詞和 similarity score
  - model.wv.most_similar(positive[word])
- 計算詞跟詞的相對關係，依照給定的一組兩個詞的關係，找出跟指定詞有相對類似關係的詞
  - model.wv.most_similar(positive=[y_1, x_2], negative=[x_1])
- 計算兩個字詞的相似度
  - model.wv.similarity(word1, word2)

維基百科中文文本

- 下載文本
  - https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-articles.xml.bz2
  - wget https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-articles.xml.bz2
- 文本是一個xml檔案
  - 可延伸標記式語言(Extensible Markup Language, XML)
  - XML設計是用來傳送和攜帶資料資訊
  - 可以自定義結構化標籤

WikiExtractor套件

- 下載
  - pip install wikiextractor
- 使用Extractor將載來的維基文件做資料清理並存成檔案
  - python -m wikiextractor.WikiExtractor -b 1024M -o extracted zhwiki-latest-pages-articles.xml.bz2
  - -b 設定使用的記憶體
  - -o 輸出檔案的路徑
  - 輸入處理的檔案

WikiExtractor套件

- 儲存的檔案格式
  - <doc id=“文章id” url=“文章網址” title=“文章標題">
    文章內容
  - </doc>

Open CC 開放中文轉換套件

- 安裝套件
  - pip install opencc
- 載入套件
  - from opencc import OpenCC
- 簡體轉繁體
  - opencc = OpenCC('s2twp')
  - raw_data_changed = opencc.convert(raw_data)
  - raw_data為要轉的資料

Open CC轉換模式

- hk2s: 繁體中文 (香港) -> 簡體中文
- s2hk: 簡體中文 -> 繁體中文 (香港)
- s2t: 簡體中文 -> 繁體中文
- s2tw: 簡體中文 -> 繁體中文 (台灣)
- s2twp: 簡體中文 -> 繁體中文 (台灣, 包含慣用詞轉換)
- t2hk: 繁體中文 -> 繁體中文 (香港)
- t2s: 繁體中文 -> 簡體中文
- t2tw: 繁體中文 -> 繁體中文 (台灣)
- tw2s: 繁體中文 (台灣) -> 簡體中文
- tw2sp: 繁體中文 (台灣) -> 簡體中文 (包含慣用詞轉換)

練習

- 用gensim載入glove預先訓練好的word embedding
  - 讀取bbcnews文件，做好英文斷詞之後，建立一個詞典包含文件內的詞 → {word: word embedding}
  - 在glove裡取不到的詞就忽略
- 用gensim自己訓練一個中文word embedding
  - 下載中文維基文本，做中文斷詞，訓練模型
  - 建立一個詞典包含文件內的詞 → {word: word embedding}