ML\_final

TV conversation

**How to build good sentence vectors?**

Preprocessing & Feature engineering

利用jeiba將句子妥善的斷詞，並訓練一個word2vec的model，我這裡使用的是內建的斷詞辭典，並沒有採用繁體中文版的辭典。

Models

Building Sentence Vectors:

先將句子中的各個word透過word2vec的方法對應到一把word vector，再從word vector拼組出適當的sentence vector，這樣做的好處是可以同時考慮到一個句子中，所有word的資訊，在直觀上，我們判斷兩個句子的相似程度並不能只用幾個詞就斷定，因此sentence vector學到的是句子整體的concept或是說semantic。以下會介紹我們如何組成適當的sentence vector。

1. 加總所有的word vectors來組成sentence vector

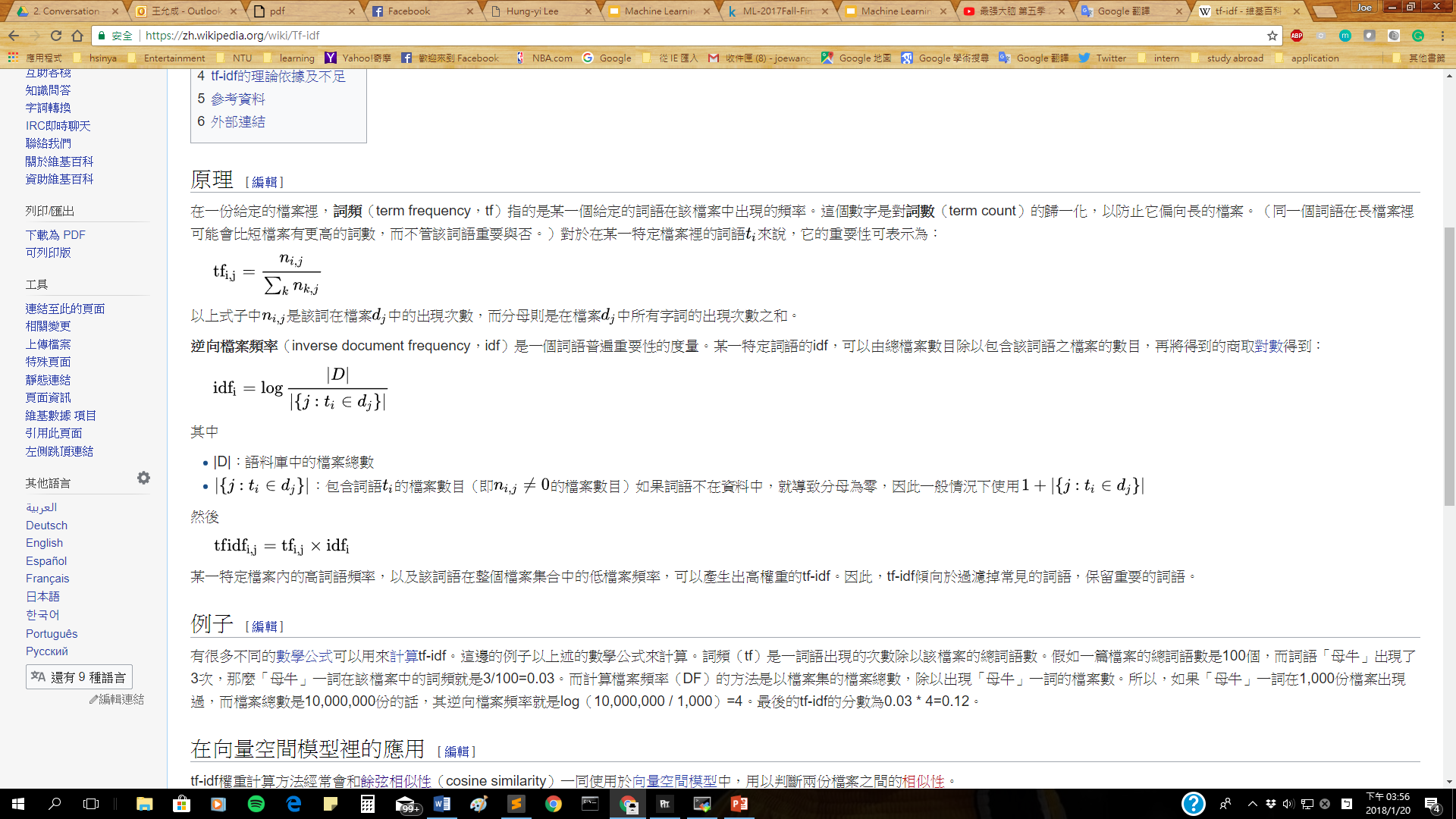
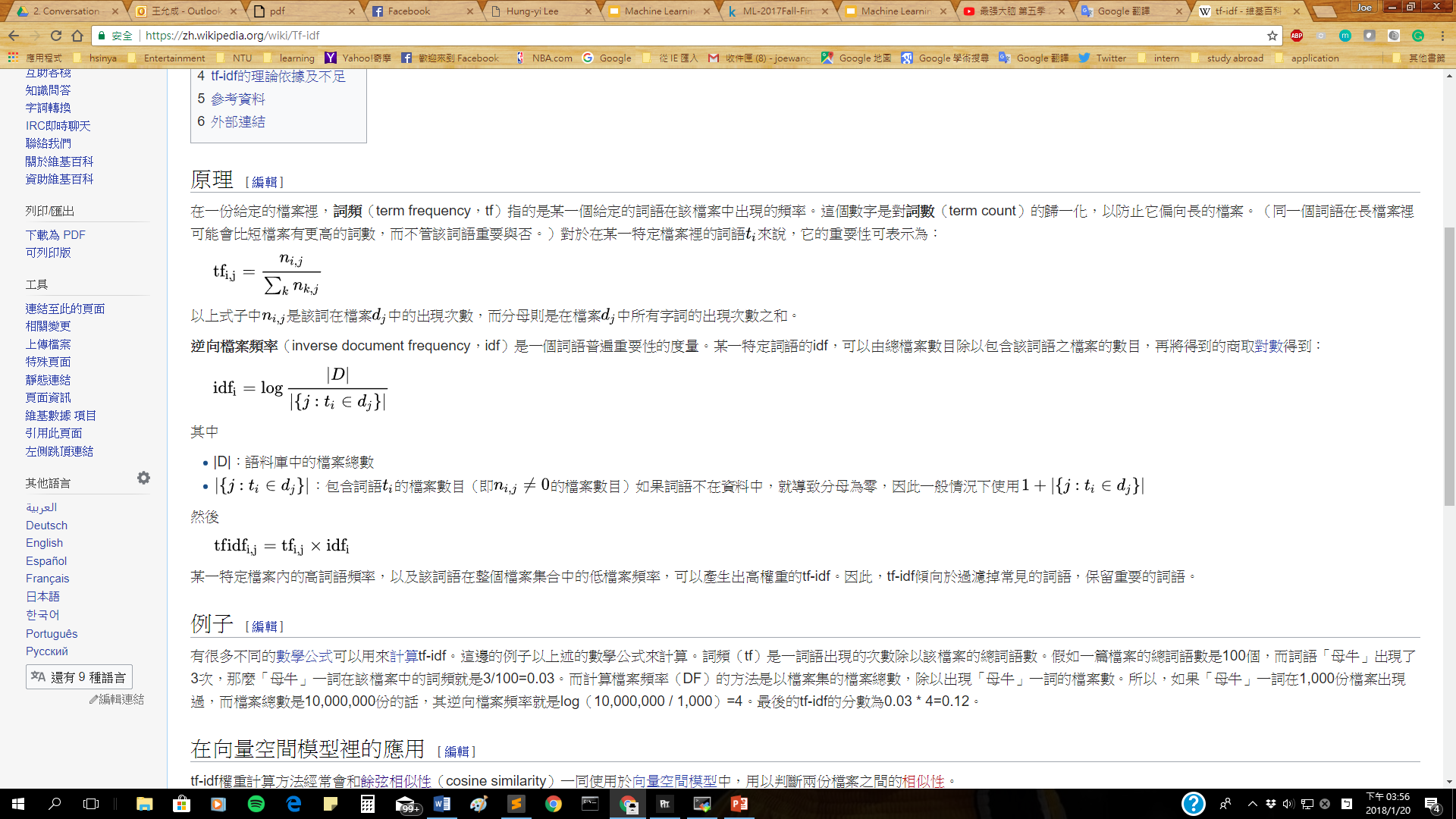
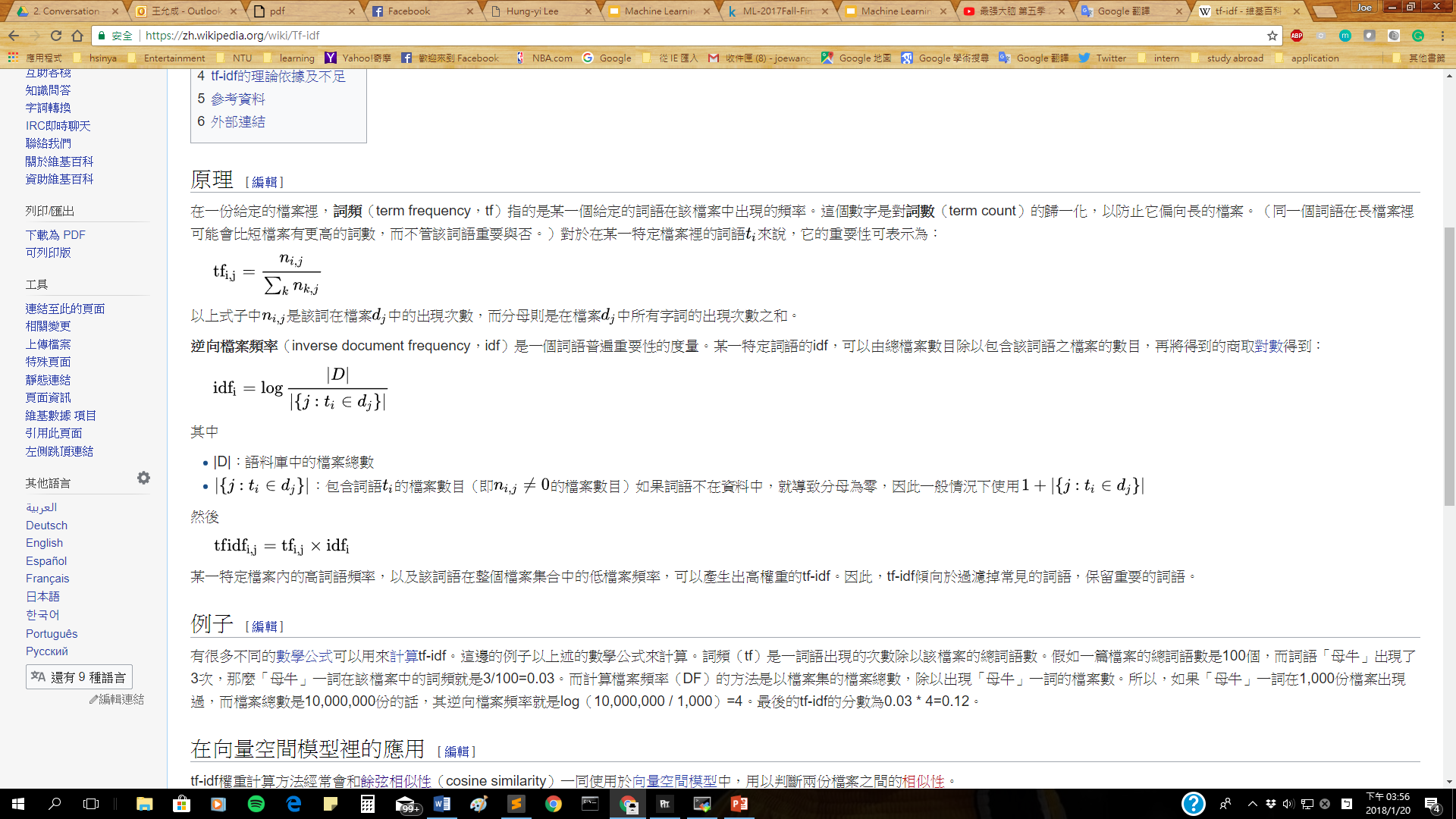
一種產生sentence vector最直觀的方式就是直接加總句子中所有的word vectors。

這樣的方法就可以將所有word的概念都包含在了一個vector之中。

2. 以tf-idf score來做為權重，以weighted sum的方式以word vectors組成sentence vector。

然而在一個句子裡面，每個word的重要性很明顯地會不同，例如：”我很快樂”，這個句子的重點會是在傳達”快樂”這個概念，因此”我”跟”很”明顯的相對不重要，因此以單純直接相加的結果也許對句子的concept會產生偏差。

而tf-idf score是用來判斷一個word在整個corpus之中的重要性的一個指標，這個word越常出現，且出現在越少的document之中，代表這個word在整個corpus之中是更加獨特而且重要的，tf-idf score分成了兩個部分，第一個部分tf score代表的是在這個document中，某個詞語出現的頻率，而idf score代表了在所有的document中，這個word總共出現在了幾個document之中，最終的tf-idf score = tf-score \* idf-score (idf-score是一個反向的分數)。

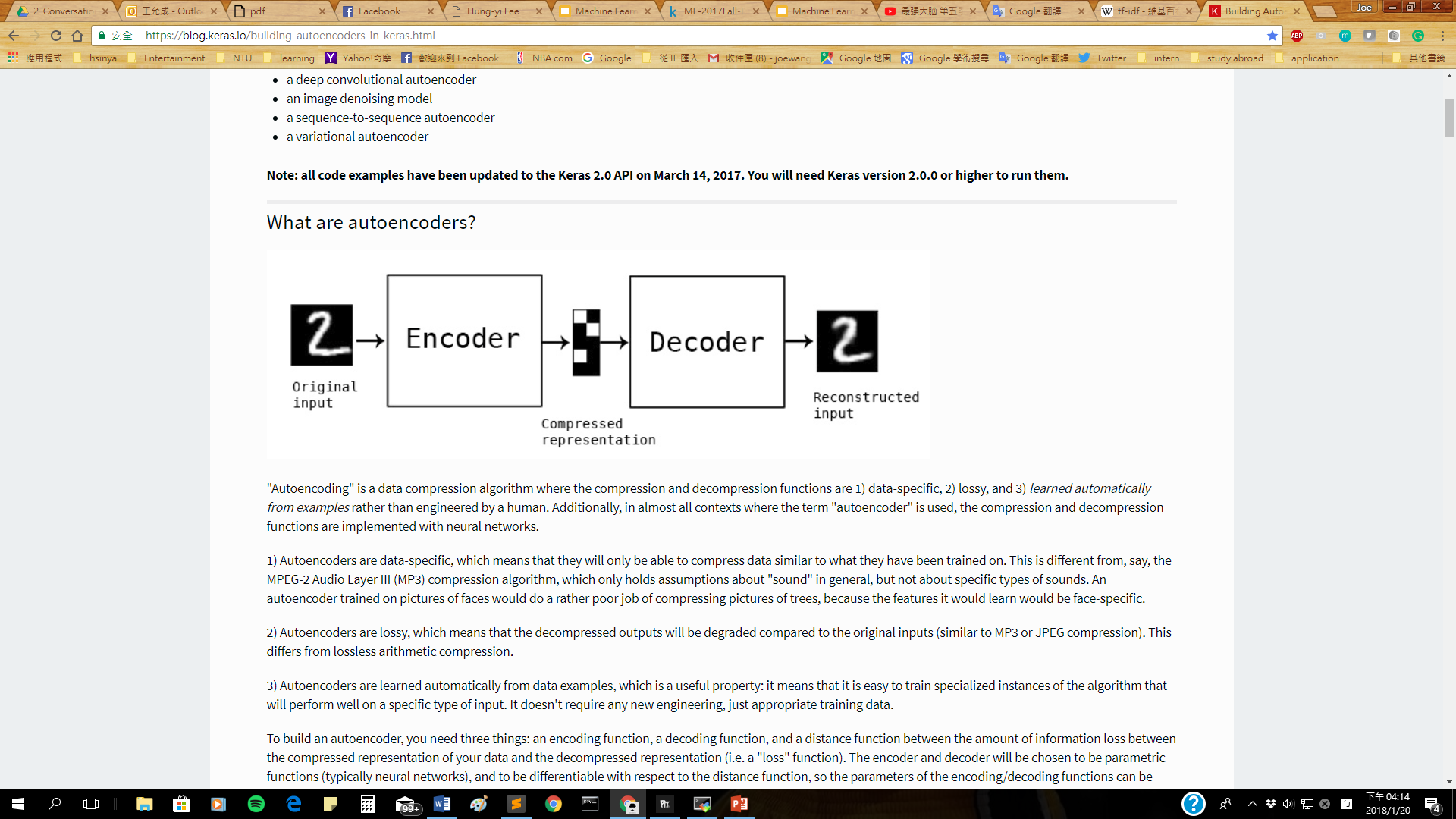


我將idf-score train在了所有所有的provide data之上，而tf-score則是計算在了當句題目或是解答的句子上，這樣的作法其實不盡理想，而結果也確實十分差勁，會在之後的單元討論到。

3. 以RNN auto-encoder來產生sentence vector

然而以上的幾種方法都沒有考慮到句子中詞語的順序性，例如：”我愛台妹”跟”台妹愛我”包含了完全相同的word，因此在前面兩個方法之中，會產生出完全相同的sentence vector，然而很明顯的兩句想表達的概念卻是大相逕庭，因此一個RNN auto-encoder可以幫助我們解決這個問題。

auto-encoder是一種unsupervised learning，可以透過encoder將原本高維度的raw data轉化成為context vector，而在透過decoder還原為原來高維度的raw data並使distortion越少越好，而RNN auto-encoder尤其適合用於NLP方面的加密，因為詞語的順序對於句子十分重要。



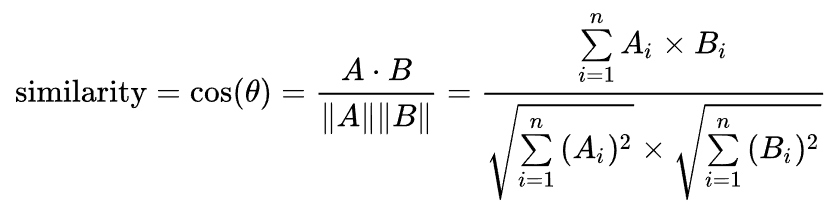
在train RNN的auto-encoder時，我們會把句子中所有的word vector疊起來，並且padding到50個字，而OOV則直接忽略，使用auto-encoder其實還有一個好處是能夠assign一個<unknown>的tag給他，讓他能夠自己處理OOV的詞語，然而這部分因為我們覺得我們OOV的詞語的詞義相差太大，可能並不適合使用這樣的方法。

而因為

我們使用了200000句的句子當作training data，input shape = (50, 400)產生出256維的sentence vector。

Calculate Similarity:

使用cosine similarity來作為判斷兩句句子相似程度的metric。



cosine similarity代表的是兩個vector之間夾角的大小，夾角越小代表相似程度越大，然而在這樣的一個標準下，和Euclidean distance不同的是，vector本身的長度並不重要，因此使用了這個評斷方式也很有效的幫助我們稍微的處理了OOV的問題，因為在有OOV的問題時，我們不好判斷該怎麼處理句子的長度，因此sentence vector的長度是不具有任何物理意義的，而cosine similarity可以幫助我們避開這個問題。

Experiments & Discussions

我們利用了三種不同產生sentence vector的方式來predict我們的結果。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sum | Weighted Sum with Tf-Idf score | RNN auto-encoder |
| Kaggle Public Accuracy | 0.44466 | 0.28893 | 0.32648 |

出乎意料的，竟然是最直觀的方法結果最好，以下討論一下tf-idf跟auto-encoder失敗的原因。

why tf-idf fails?

tf-idf其實很在意的是corpus的大小，corpus夠大才有辦法展現出tf-idf的統計特性，雖然訓練idf時的corpus夠大，但是在計算tf-score時只用了單一句子去算明顯有問題，應該將同一個劇本裡的所有台詞當作corpus來計算tf-score，而且加入了tf-idf vectorizor又大大增加了OOV的問題。

why RNN auto-encoder fails?

下面這張圖是所有training data embedded到256維之後再降維到2維平面上的圖，可以看到，sentence vector之間的cosine similarity確實會不一樣，然而實際上差距卻是很小，可以推斷我們input的raw data之間可能本身就太像了，因為我們將字數padding到了50字，然而有很大一不分的句子只有3~4個字，因此造成了這些raw data embedded之後十分接近，然而又很難縮短padding的長度，因為仍然有許多長度較長的句子，因此像對話這種有長度不一的句子，可能較不適合使用RNN訓練。

