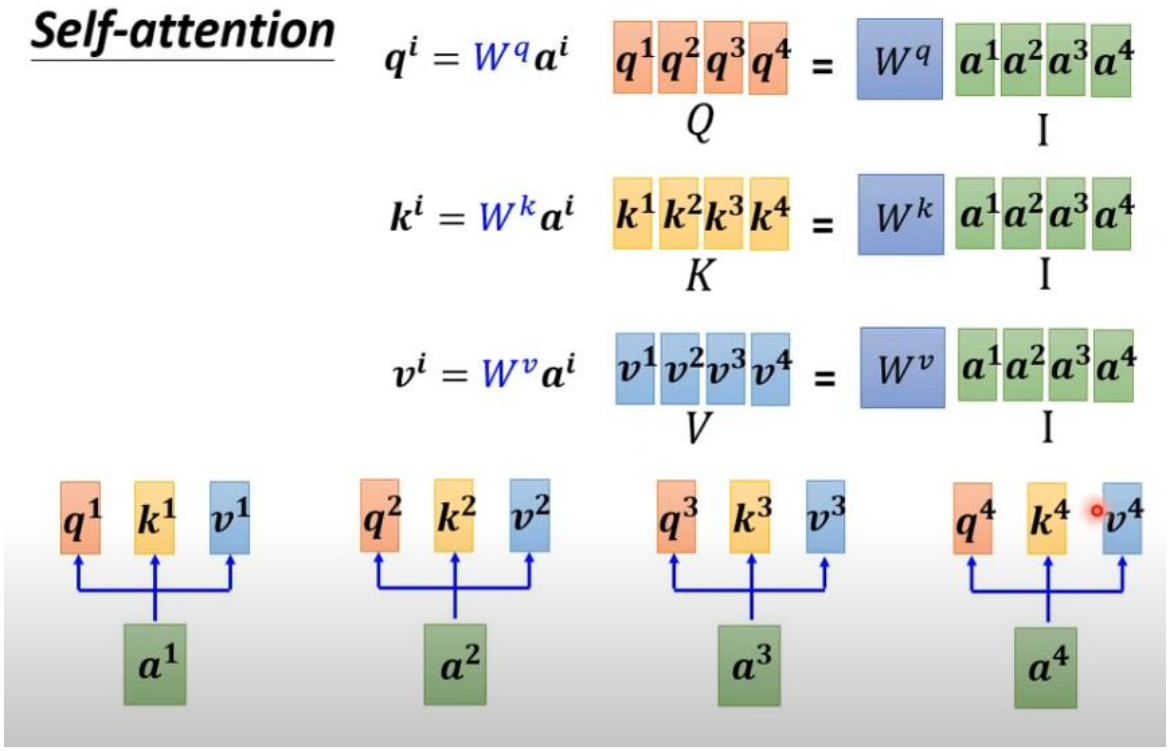
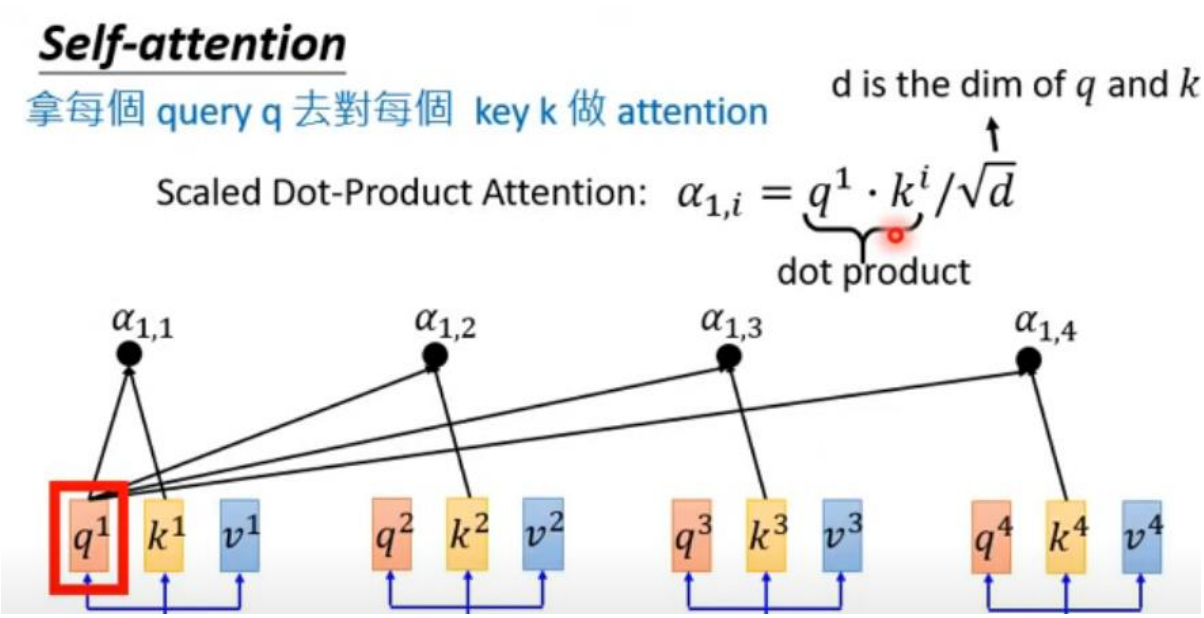
I. Attention mechanism:

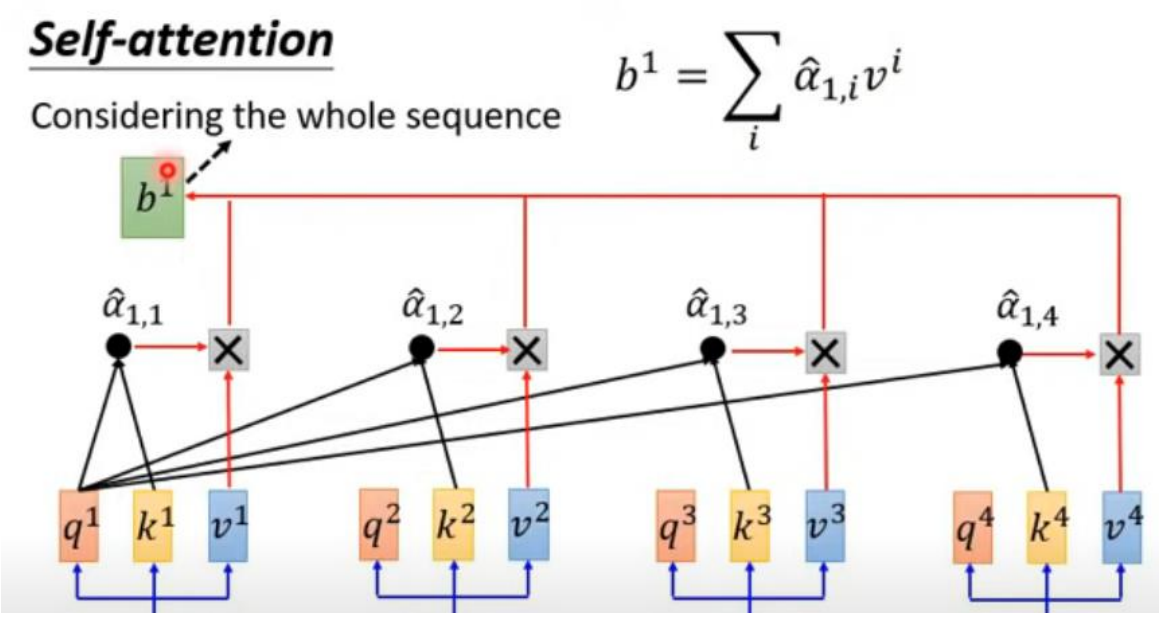
假如輸入是一段句子，在 self attention 中，將輸入進行 input embedding 後，分別乘上三個不同矩陣，成為 q, k, v 三種向量(如下圖)。



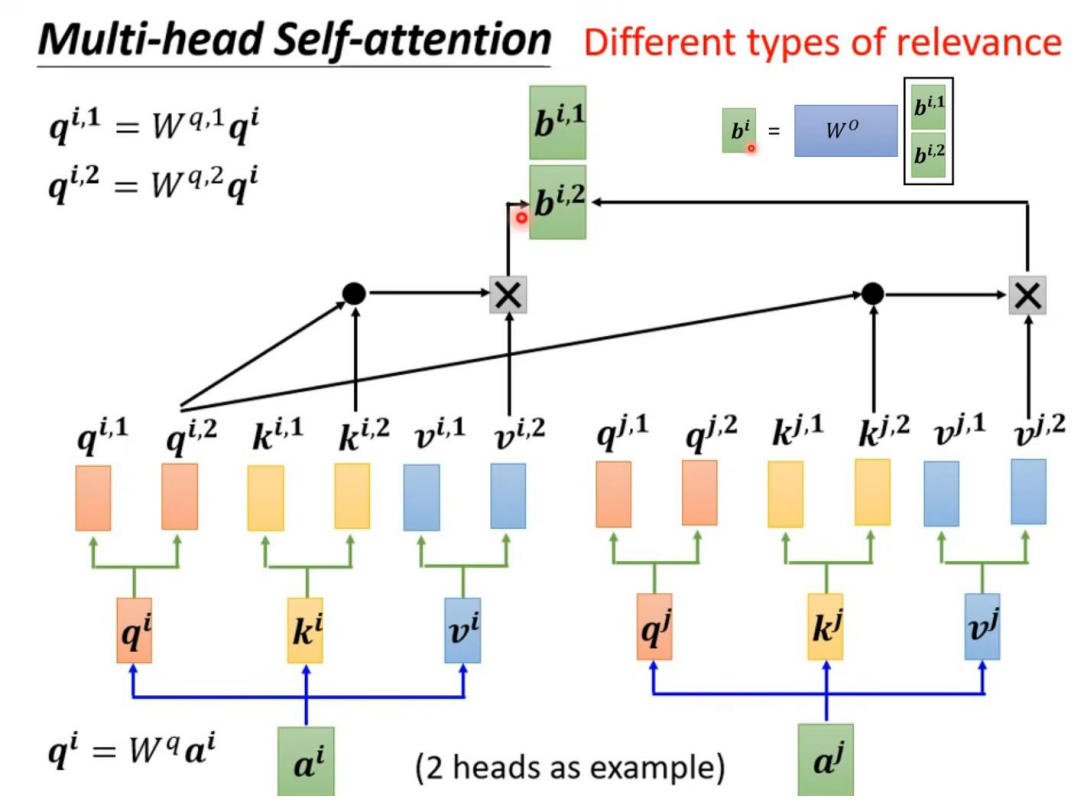
接著用 q, k 向量計算出 alpha (如下圖)。



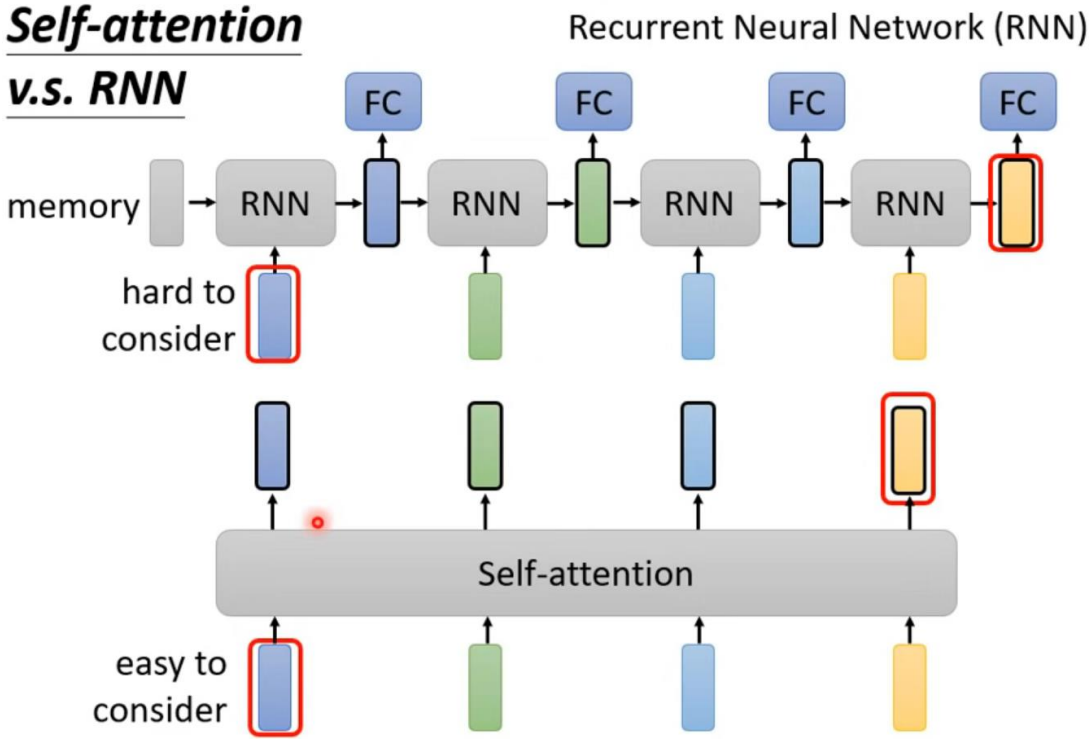
用 softmax 將 alpha 轉為 alpha-hat 後，利用 alpha-hat 與向量 v 計算出輸出(如下圖)。



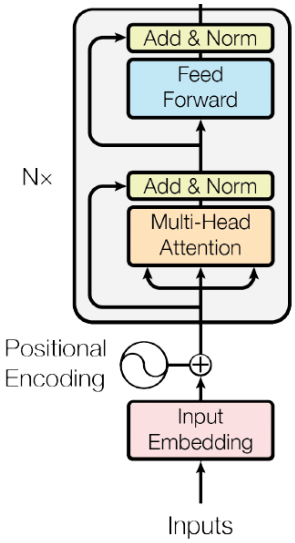
但有時因為需要取得不只一種相關性，所以需要 multihead(如下圖)。



而跟 RNN 比較時，會發現 self attention 有不少優勢，包括它能輕鬆考慮到距離較遠的輸入，畢竟在 self attention 中，距離遠近不會有差別，但在 RNN 中，距離越遠的輸入就越難考慮到；此外，self attention 中，可以運用矩陣進行資料平行運算，但在 RNN 中不行，因此耗時較久。

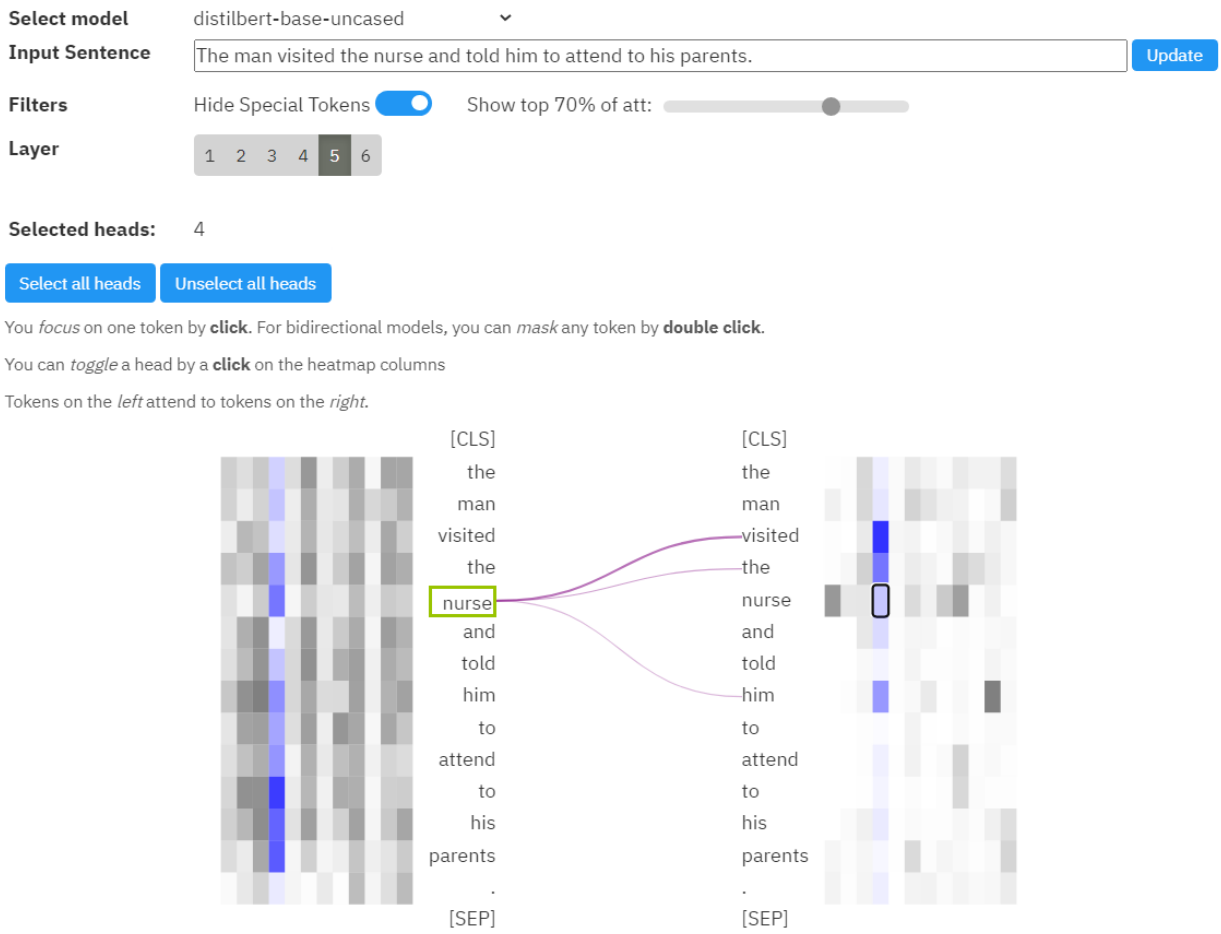


在 Transformer Encoder 中就是使用了 self attention(如下圖)。



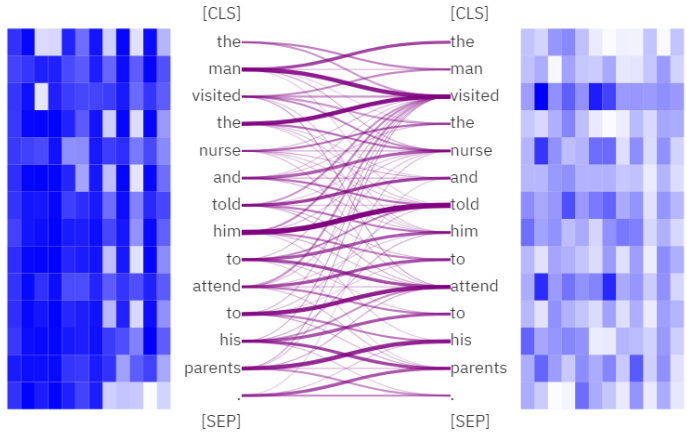
而 DistilBERT 是較為簡易的 BERT，保留了 97%的語言理解能力，但同時減少空間大小 40%且加速 60%。它是一種深度雙向、無監督、且僅使用純文字進行 Pretrain 的模型，它疊加多層 Transformer Encoder，可以更好地理解語意，並在各 NLP 項目中被不斷使用。

在 exBERT 中，就能很輕鬆地看到各個 attention 的關聯，此外，它能夠讓使用者自由選擇 layer, heads, tokens 等等(如下圖)。

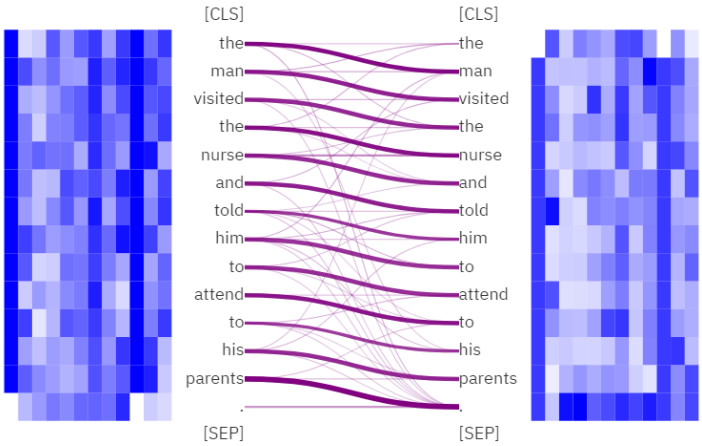


譬如選擇不同 layer 可以看到各 layer 間的差異(如下)。

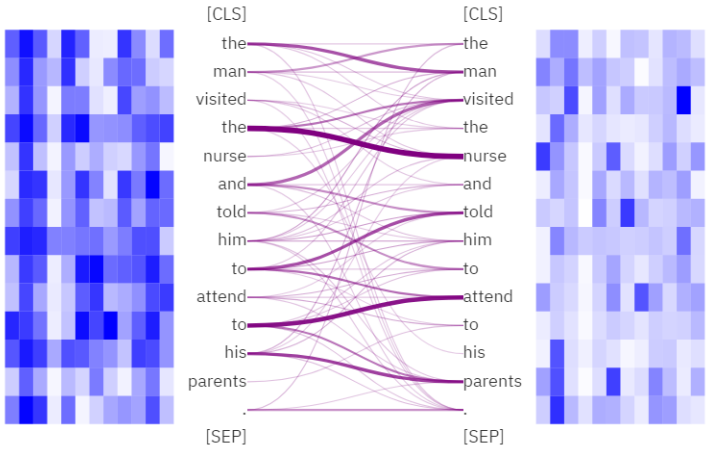
Layer1:



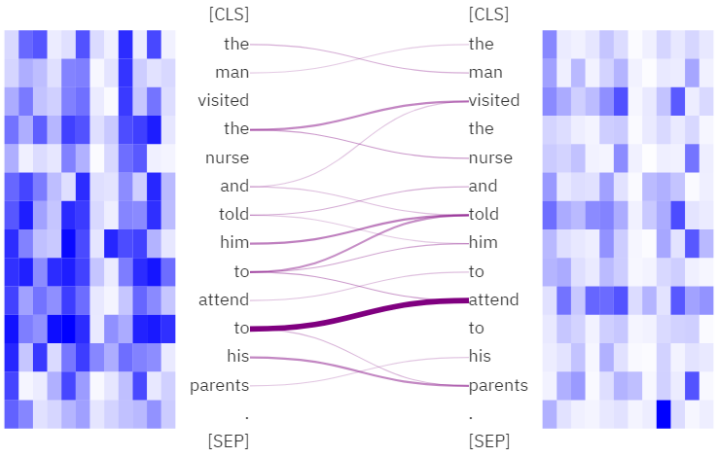
Layer2:



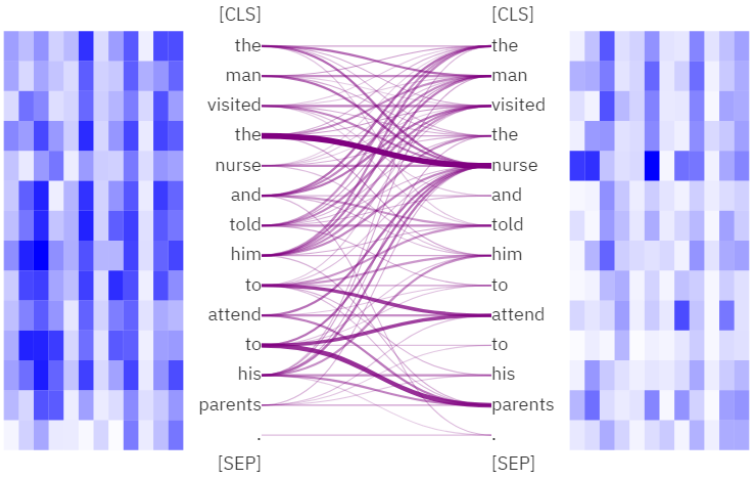
Layer3:



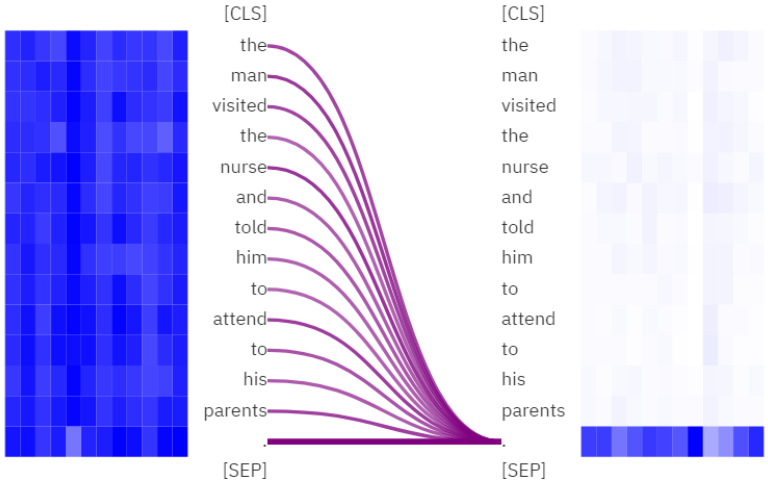
Layer4:



Layer5:



Layer6:



II. Compare 2 sentiment classification models

A. TA_model1(distilbert-base-uncased):

DistilBERT 是較為簡易的 BERT，保留了 97%的語言理解能力，但同時減少空間大小 40%且加速 60%，使用 distill technique，從原本的 BERT 中蒸餾成較小的模型。

B. TA_model2(prajjwal1/bert-small):

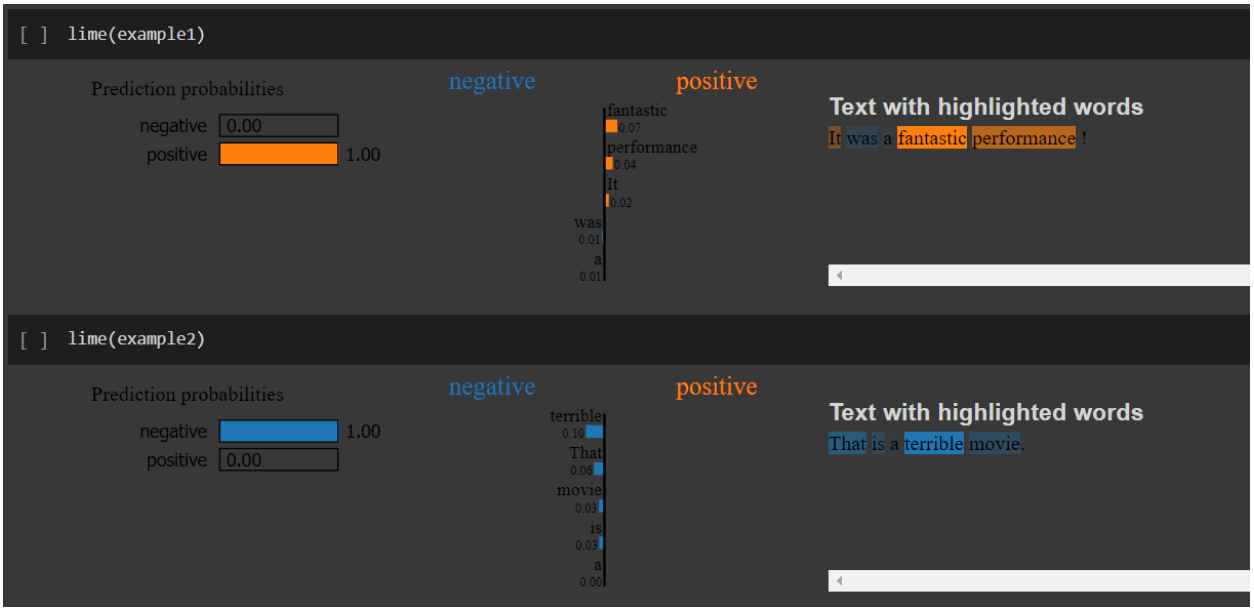
BERT-small 與 BERT-base 的差距是他們的 model 大小差異， BERT-base 有 12 個 attention header、768 個 hidden layer，而 BERT-small 只有 4 個 attention header、512 個 hidden layer(如下圖)；在 GLUE score 方面，BERT-small 比 BERT-base 差了一點，不過這些較小的 model 被訓練出來的用意就是為了在較少運算資源時也能使用，而模型的表現只是一個 tradeoff 而已。

	H=128	H=256	H=512	H=768
L=2	2/128 (BERT-Tiny)	2/256	2/512	2/768
L=4	4/128	4/256 (BERT-Mini)	4/512 (BERT-Small)	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512 (BERT-Medium)	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	12/768 (BERT-Base)

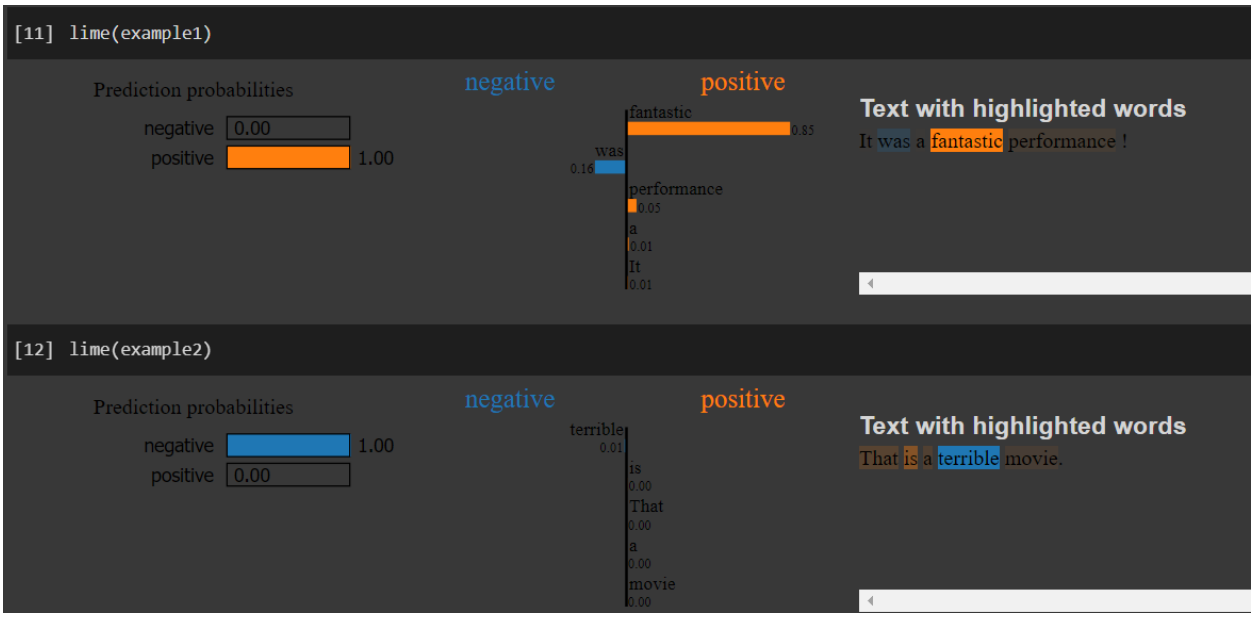
C. 使用 LIME 比較兩個 model 表現差異

1. 使用例句('It was a fantastic performance !', 'That is a terrible movie.')

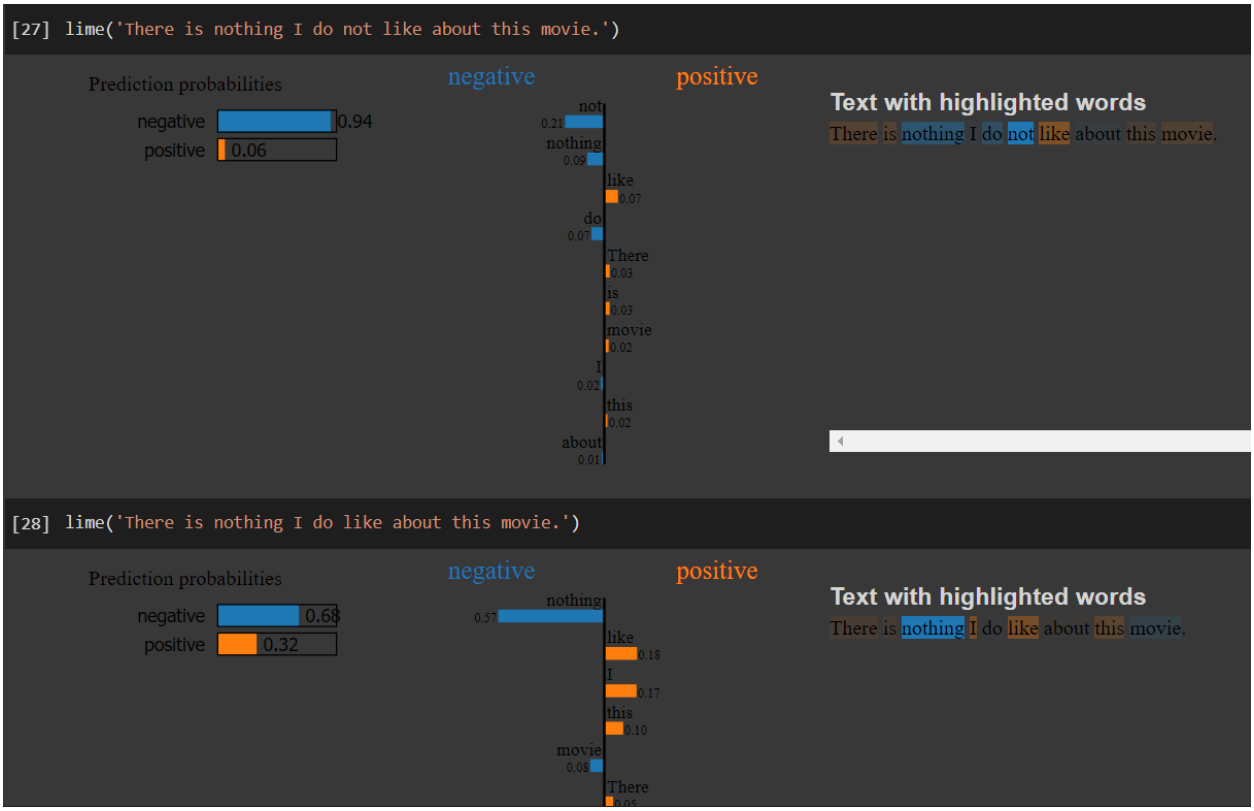
distilbert-base-uncased:



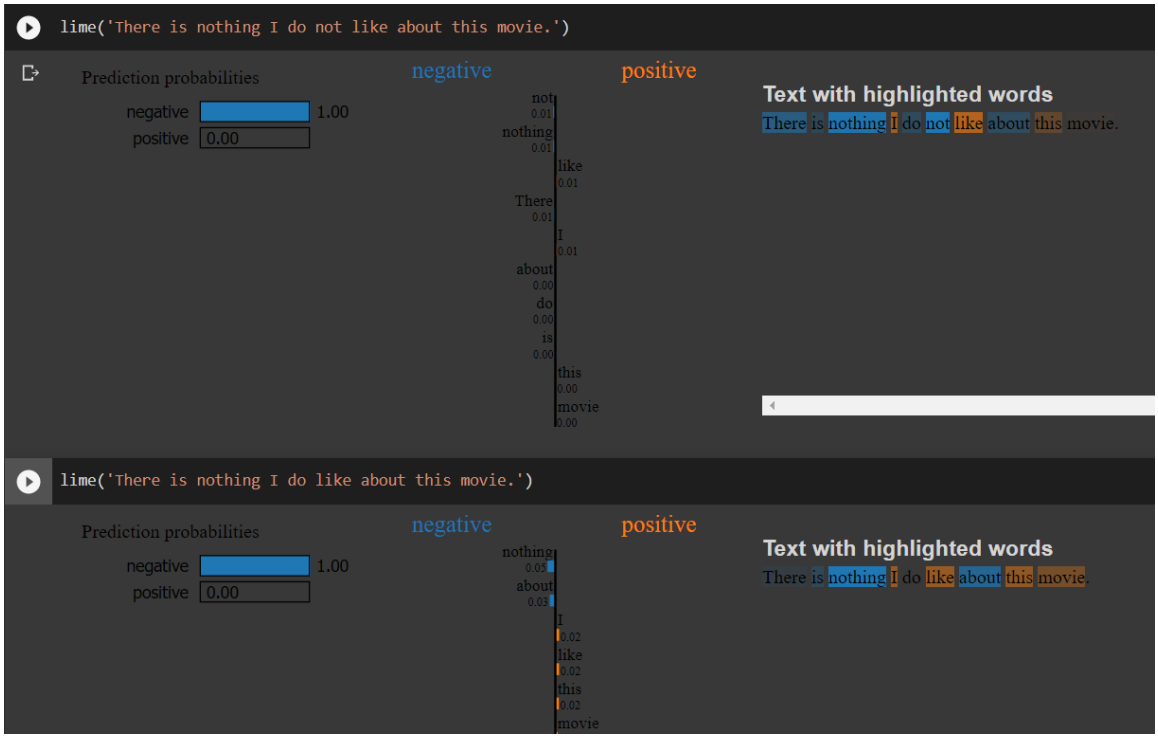
prajjwal1/bert-small:



2. 使用句子('There is nothing I do not like about this movie.', 'There is nothing I do like about this movie.')
   distilbert-base-uncased:



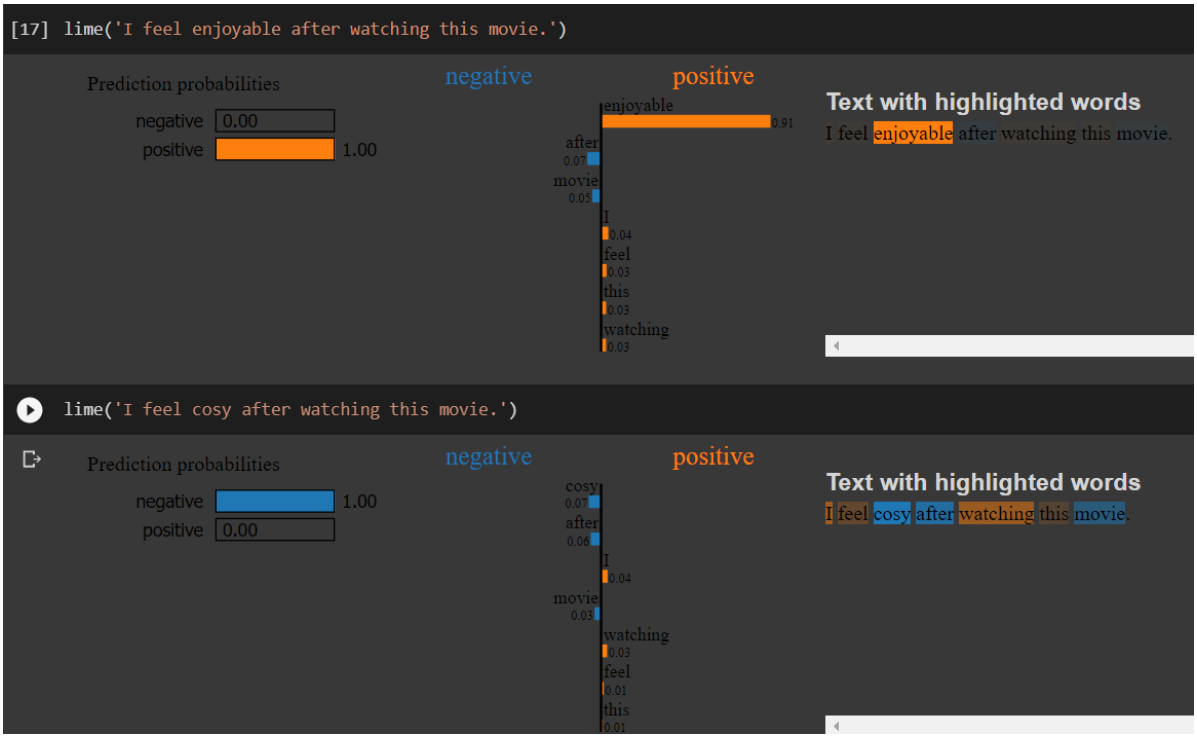
prajjwal1/bert-small:



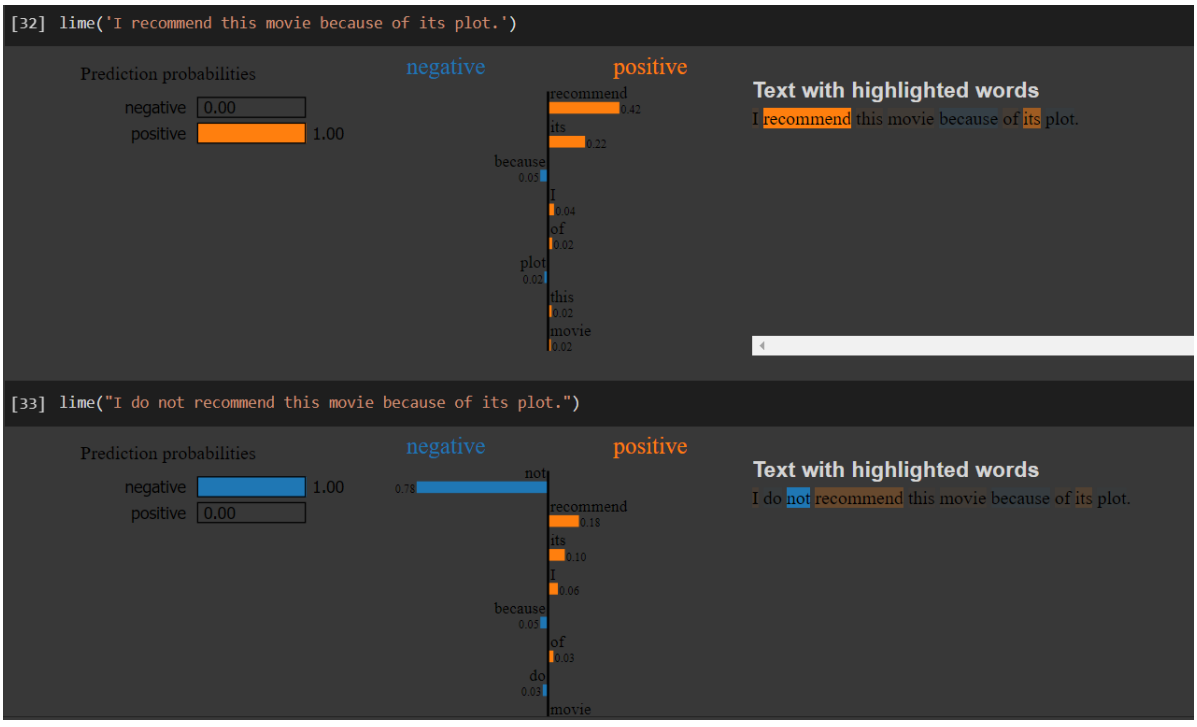
3. 使用句子('I feel enjoyable after watching this movie.', 'I feel cosy after watching this movie.')

   distilbert-base-uncased:

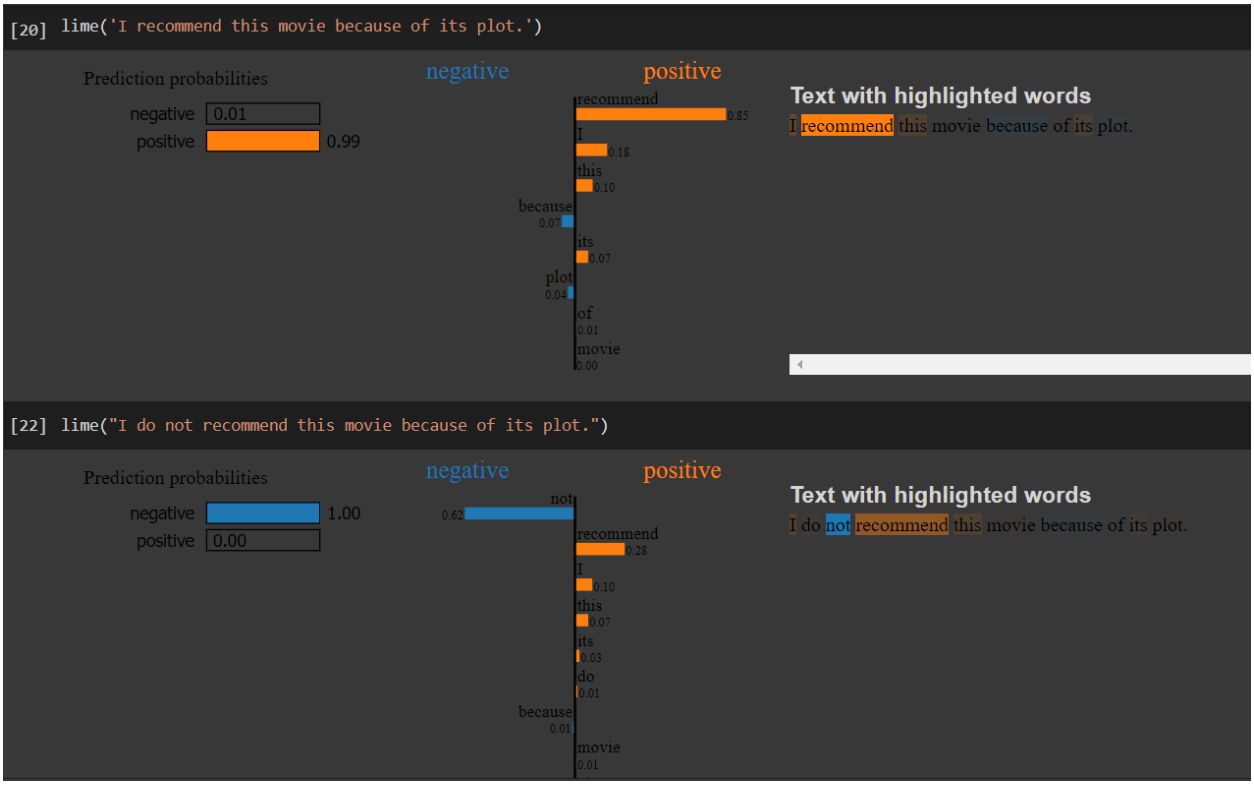
prajjwal1/bert-small:



4. 使用句子("I recommend this movie because of its plot.", "I do not recommend this movie because of its plot.")
   distilbert-base-uncased:



prajjwal1/bert-small:

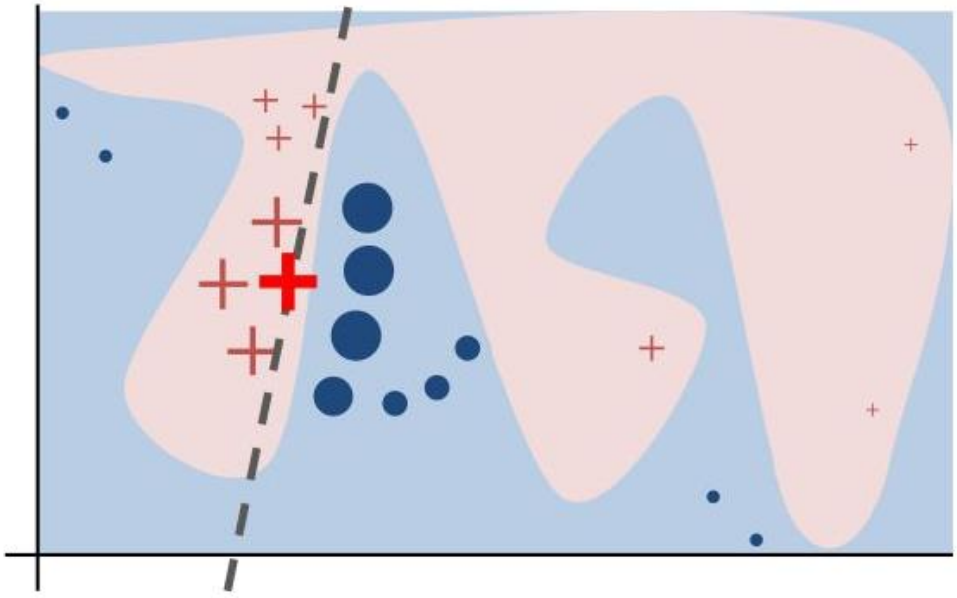


可以看到，這兩個 model 在判斷句子是 negative 或 positive 方面上差異不大，不過在句子中的單一 token 上，兩個 model 顯示出的 prediction probability 不太相同，平均上來說，BERT-small 的 probability 較 DistilBERT 的小，不過也不排除是觀察例子過少所造成。

III. LIME & SHAP

A. LIME（Local Interpretable Model-agnostic Explanations）

LIME 是在"Why Should I Trust You?"論文中第一次被提出，他有著 Model agnosticism 的特性，代表著能夠被使用於任何視為 black box 的 supervised learning model 中；也有著 Local explanations 的特性，代表他局部的忠實性。假如利用兩個特徵 x1, x2 建立了一個複雜但準確的模型，要解釋個體為何屬於哪一區域是很困難的，因此 LIME 就在該個體的附近建立一個線性或是簡單的預測模型，在觀察某個體時，在該個體附近的預測準確度與原來的模型一樣，但在離該個體較遠的區域預測準確度就會大幅下降，就是所謂的局部忠實性(如下圖)。



Step1. 在個體附近隨機產生 pertubations，當作一組新的 input data(x_lime)

Step2. 用原本的預測模型預估上一步中的 x_lime，產生新的預測資料(y_lime)

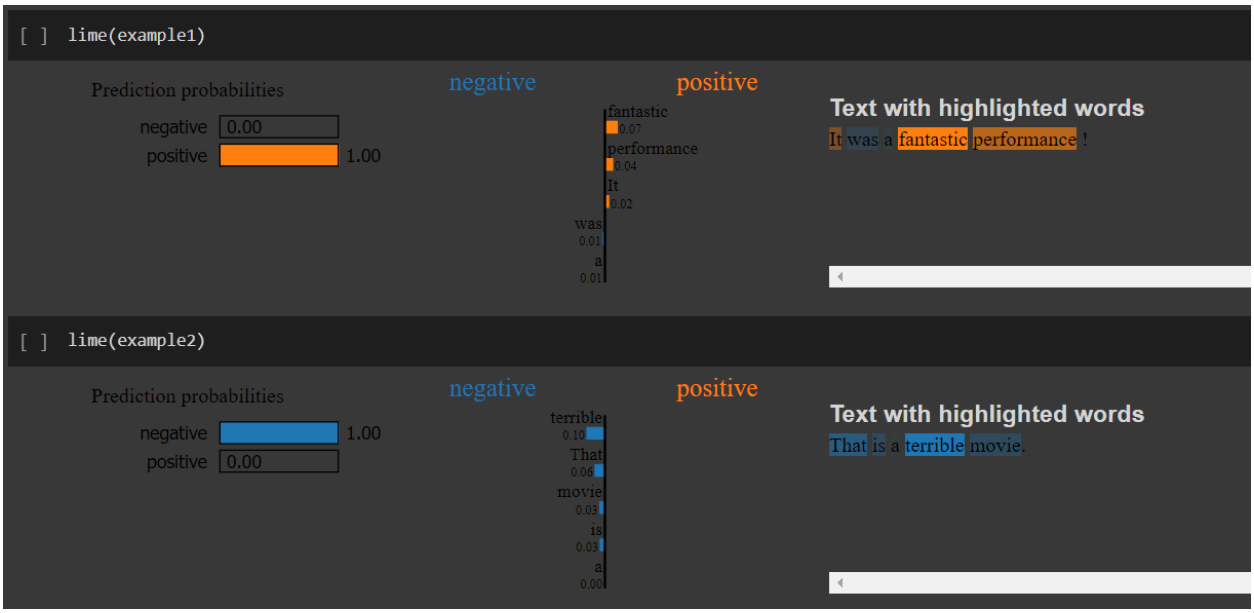
Step3. 計算個體與各個 pertubation 的距離(Euclidean distance or cosine distance)，並將距離用 kernel function 轉為 0~1 之間的 weight

Step4. 用 x_lime, y_lime, weight 訓練一個新的簡易模型，它的預估準確度只會在個體附近與原本的模型一樣，距離過遠的準確度會逐漸下降。

而我們也可以使用 lime package 輕易使用 lime 來解釋 model，這次作業中就用 lime 解釋了 sentiment analysis model(程式碼如下圖)。

```
def lime(text):
    def predict(data):
        '''
        Prediction probability function that Lime needs.
        The input is a list of d strings and output is a (d, k) numpy array with prediction probabilities, where k is the number of classes (i.e. two).
        '''
        input_text = tokenizer(data, padding=True, truncation=True, max_length=512, return_tensors='pt')
        with torch.no_grad():
          outputs = model(**input_text)
          outputs = nn.Softmax(dim=1)(outputs)

        return outputs.detach().numpy()

    explainer = LimeTextExplainer(class_names=['negative', 'positive'])

    ## Generate explanations for a prediction
    exp = explainer.explain_instance(text, predict, num_features=10, num_samples=500)

    ## Visualize explanations
    exp.show_in_notebook()
```

使用 lime 後，可以看到每個 token 所佔的影響比例大小，下圖中就可看到 fantastic 在 positive 中佔了很大部分的影響，而 terrible 在 negative 中佔了很大部分的影響。



B. SHAP (Shapley Additive Explanations)

在 SHAP 中會用到 Shapley value，它是基於合作賽局理論提出來的解決方案，求出每人一個 value 來代表每個人所做的貢獻。本次作業中的講義就很清楚地以例子求出 Shapley value(如下圖)。

Shapley Value - An Example

$$\phi(x_i) = \sum_{S \subseteq N \setminus \{i\}} \frac{1}{n\binom{n-1}{|S|}} \left(\nu(S \cup \{i\}) - \nu(S)\right)$$

- Assume 3 engineers need to do a project with 100 lines of codes, what is the Shapley values of the first engineer (x1) ?

Engineer (S)	Coding ability (val(S))
x_1	10
x_2	30
x_3	5
x_1, x_2	50
x_2, x_3	35
x_1, x_3	40
x_1, x_2, x_3	100

Order	x_1 Contribution	value
x_1, x_2, x_3	$val(x_1)$	10
x_1, x_3, x_2	$val(x_1)$	10
x_2, x_1, x_3	$val(x_2, x_1) - val(x_2)$	$50 - 30 = 20$
x_2, x_3, x_1	$val(x_2, x_3, x_1) - val(x_2, x_3)$	$100 - 35 = 65$
x_3, x_1, x_2	$val(x_3, x_1) - val(x_3)$	$40 - 5 = 35$
x_3, x_2, x_1	$val(x_3, x_2, x_1) - val(x_3, x_2)$	$100 - 35 = 65$
	$\frac{1}{6}(10 + 10 + 20 + 65 + 35 + 65) = 34.17$	

10

從 Shapley value 可以看出每個特徵的相對貢獻，因此也能用來解釋 sentiment analysis model(程式碼如下圖)。

```
def shapely(text):
    def predict(data_all):
        '''
        Prediction probability function that SHAP needs.
        The input is a list of d strings and output is a (d, k) numpy array with prediction probabilities, where k is the number of classes (i.e. two).
        '''
        prob = []

        for data in data_all:
            input_text = tokenizer(data, padding=True, truncation=True, max_length=512, return_tensors='pt')
            with torch.no_grad():
                outputs = model(**input_text).detach().numpy()[0]

            scores = (np.exp(outputs).T / np.exp(outputs).sum(-1)).T
            prob.append(sp.special.logit(scores))

        return prob

    explainer = shap.Explainer(predict, tokenizer, output_names=['negative', 'positive'])
    shap_values = explainer(pd.Series(text))

    plt.figure()
    shap.text_plot(shap_values)
    plt.show()
```

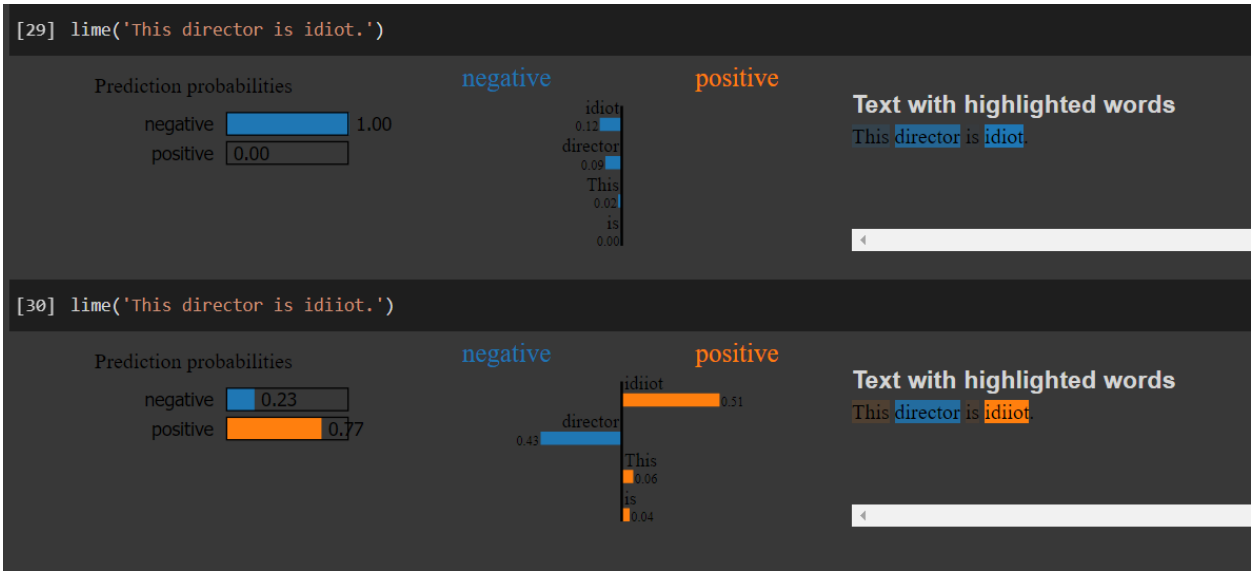
使用 SHAP 後，可以看到每個 token 所佔的影響比例大小，下圖中就可看到 fantastic 在 positive 中佔了很大部分的影響，而 terrible 在 negative 中佔了很大部分的影響。

IV. Attacks in NLP

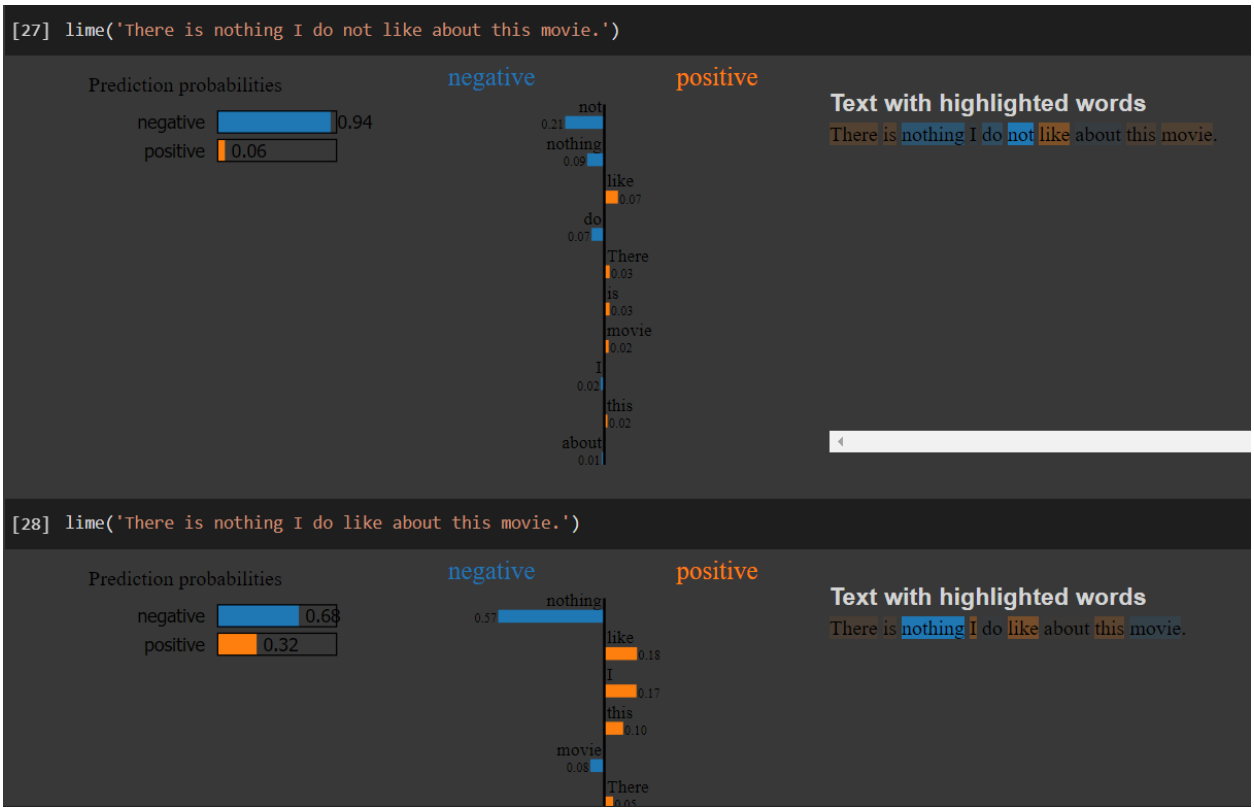
Adversarial attack 在 AI 許多方面都有出現及研究，無論是 Speech processing、Image classification 或 NLP 中都有，攻擊方與防守方的技巧也都在不斷精進。在 Speech processing 中，加入一些雜訊就能讓 model 認為一段合成語音其實事人說話的聲音；Image classification 中，加入肉眼不可見的雜訊就能讓 model 辨識錯誤，甚至是 100% 確定那個 image 就是他所錯誤辨認的 label；在 Question answering 中，讓 model 無論是甚麼 input，都只輸出"to kill American people"；在這次作業的 sentiment analysis 中也能看到這些 model 正常運作時 performance 都不錯，但面對 adversarial attack 都太脆弱了。

在使用助教提供的 distilbert-base-uncased 配合 LIME 解釋後可以看到，下面四個例子中都顯示了 model 的判斷錯誤。

A. idiot 的拼音錯誤變成 idiiot，導致句子判斷成 positive。



B. model 不太能判斷英文雙重否定句，第一句”There is nothing I do not like about movie”，其實是我喜歡這個電影的每個部份，但 model 判斷卻是 negative。



C. 將 good 拆開成 g o o d 後判斷成 negative。

D. Enjoyable 與 cosy 算是同義詞，兩者皆是正面詞彙，但 model 確判斷 cosy 為 negative。