國立清華大學 電機工程學系
實作專題研究成果報告

Locality Sensitive Hashing on ALBERT

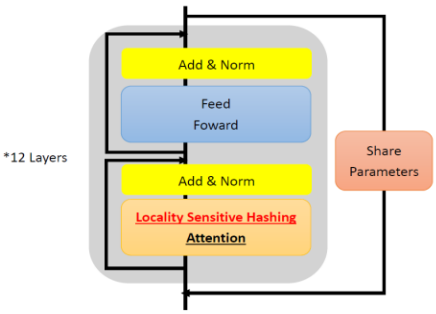
使用位置敏感雜湊的 ALBERT 語言模型

專題領域：資工系
組 別：A47
指導教授：孫民
組員姓名：陳兆廷、沈永聖
研究期間：109 年 3 月 1 日至 109 年 5 月底止,計 3 個月

Abstract

1. Introduction

Space computational costs and time complexities are more and more important in the era of large language models. ALBERT greatly reduces the time and space needed for training a language model. Furthermore, Locality Sensitive Hashing attention mechanism also improves on language model's space complexity.



ALBERT-LSH Structure

Fig. 1

2. Purpose

We would like to experiment whether it will achieve a better result if the attention layer for ALBERT is replaced by Locality Sensitive Hashing.

3. Model selection

3.1. Locality Sensitive Hashing

Locality Sensitive Hashing is an attention mechanism that replaces the original dot-product attention and reduces the former space complexity of $O(N^2)$ to $O(N\ lg\ N)$. It randomly rotates Q vectors several rounds and hashes each q_i into several buckets:

$$\Pr_{A^{(1)},\dots A^{(l)}}[h_i(p) = r_{v_i}^{(i)}\ for\ all\ i \in [k]|A^{(i)}q = x^{(i)}]$$

This process finds all related q_is and computes them into attention-matrix with lower computational cost:

$$o_i = \sum_{j \in P_i} exp(q_i * k_i - z(i_i P_i))v_i/\sqrt{d_k}$$

3.2. Model building

We connected locality sensitive hashing attention layer on ALBERT structure (fig 1). For training process, we use standard ALBERT pre-train model provided by Google for pre-training, then fine-tune it to specific tasks.

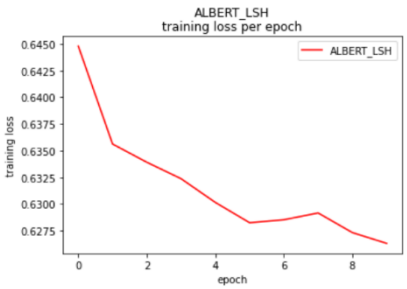


Fig. 2

4. Result

4.1. Model Correctness

We can see the Loss is decreasing through epochs of training. (fig. 2)

4.2. Evaluate on Test Benches

	MRPC	MNLI	SST-2	SQuAD1.1	SQuAD2.0
ALBERT_LSH	0.78	0.35	0.51	13.93	50.07
ALBERT	0.88	0.84	0.93	78.45	72.63

5. Result

Implementing Locality Sensitive Hashing on natural language processing tasks is workable. It reduces space needed for module to compute therefore we can construct a larger module. It is a trade-off between space and time. Our ALBERT_LSH uses less space, but hashing loses some text feature and decreases accuracies. We need further experiments to determine whether our module needs more time to train, or it is already the limit of LSH attention mechanism.

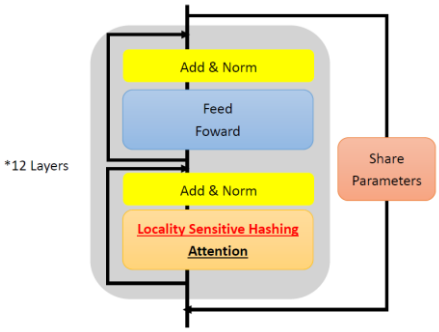
摘要

一、 前言

在模型越趨龐大的時代，如何降低運算空間或時間是極其重要的。有 ALBERT 大幅減少了語言模型運算時間及空間的需求；Locality Sensitive Hashing(位置敏感雜湊），在空間複雜度上進步許多，但並沒有在下流工作中實作。

二、 實驗目的

實驗若是將 ALBERT 的注意力機制替換至 Locality Sensitive Hashing 注意力，是否也能達到相當不錯的效果。



ALBERT-LSH Structure

圖一

三、 模型介紹

3.1. Locality Sensitive Hashing

Locality Sensitive Hashing 是一種雜湊方式，取代了原本矩陣相乘的注意力機制，將空間複雜度從$O(N^2)$降為$O(N\ lg\ N)$。其將Q向量依照多次隨機旋轉後記錄其分配之區分桶（bucket）中如下式，

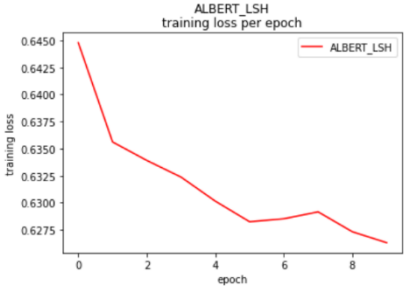
$$\Pr_{A^{(1)},\ldots A^{(l)}}[h_i(p) = r_{v_i}^{(i)}\ for\ all\ i \in [k]|A^{(i)}q = x^{(i)}]$$

確認每個q_i向量之關聯度，進而依照關聯度，如下式，進行注意力機制運算上的節省。

$$o_i = \sum_{j \in P_i} exp(q_i * k_i - z(i_i P_i))v_i/\sqrt{d_k}$$

3. 2. 模型架設

我們將 Locality Sensitive Hashing 架接至 ALBERT 模型之注意力層上（圖一），使用 Google 提供之大型預訓練模型，再以我們所實驗的模型進行各種語言工作的微調訓練。



圖二

四、 結果

4. 1. 模型正確性

經過多個 epoch 的訓練後，明顯看出其 Loss 值在進行收斂（圖二）。

4. 2. 測試分數

	MRPC	MNLI	SST-2	SQuAD1.1	SQuAD2.0
ALBERT_LSH	0.78	0.35	0.51	13.93	50.07
ALBERT	0.88	0.84	0.93	78.45	72.63

五、 結論

Locality Sensitive Hashing 在自然語言處裡工作上是可行的，並且可以減少模型運算所需空間，使訓練架構更龐大，是以訓練時間換取運算空間的方法。我們的 ALBERT_LSH 也就是減少了硬體使用的空間，但雜湊的注意力機制會丟失或簡化文字特徵使模型的準確率下降，但或許其需要更長的訓練時間，又抑或是此雜湊方法的極限就是如此，有待往後再進行研究。

一、 前言

在 Transformer 提出將擁有編碼器及解碼器的語言模型加上自注意力機制（Self-attention）後，不僅在自然語言處理上達到更好更有效的成果，更能將此注意力機制運用在電腦視覺捕捉感受野等方面。而 Transformer 的提出也讓更多人運用注意力機制創造出效能更好的語言模型，如 BERT，從 Transformer 提出之雙向編碼器，在各個資料集上均得出極高的結果；而 ALBERT 是將 BERT 所花的計算量及空間做縮減，更能達到相同的效果，其高效率的運算方法是本篇專題的主要採用方式，使我們更好實際運行計算；最後 Reformer 是本專題的主軸之一，引進了尚未使用過的位置敏感雜湊（Locality Sensitive Hashing）注意力機制，使得 Transformer 的注意力層得到更好的發揮。

二、 動機

在各研究機構均對語言模型進行微調及更精細的研究時，或許能針對各個創新的元素進行整合，進而達到更高的效果。如 ALBERT 及 Reformer 皆針對現有的模型，BERT 及最原始的 Transformer，提出一些創新的想法及元素。其中 Locality Sensitive Hashing 在 Reformer 上對空間的進步以及對 Transformer 效果的提昇令我們相當有興趣，但 Reformer 並沒有在各項 dataset 上進行檢測，更令我們感到好奇。

此篇專題主要在研究及實做，將 Locality Sensitive Hashing 應用在 BERT 的雙向編碼器上，並以 ALBERT 之輕量運算模式運行至各個不同的語言工作上，如問題回答、語意生成等，檢測運用 Locality Sensitive Hashing 的注意力機制是否能在 ALBERT 之雙向編碼器上得到更好的效果，又抑或是運用此種 Hashing 方式對文字進行編碼時，反而是犧牲了準確度來而換取空間上的運算。

三、 相關研究

3.1. Transformer（Attention is all you need）[1]

3.1.1. 架構

Transformer 由一個編碼器及一個解碼器所組成，其各有 6 層，如圖（三）所示。

編碼器每層有兩個子層：多自注意力層（Multi-head self-attention layer），也就是 Transformer 的創新點，及全連接層（Fully connected feed forward layer；而每個子層均加上殘餘連接（Residual Connection）及常規化（Layer Normalization）。每層的輸出如下式所示：

$$Output = LayerNorm(x + SubLayer(x))$$

注意力層之目的在於使更需要被模型關注的語句及序列得到更多模型的關注，也使不重要的語句在模型中更不備註意。

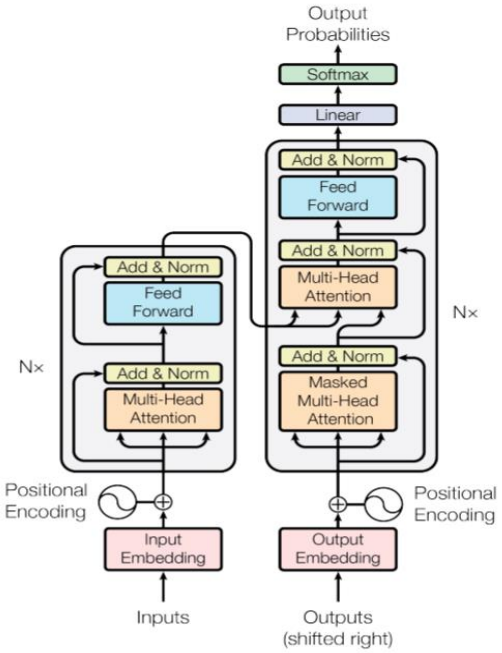
多注意力層在使用多個不同的線性變化對一般計算注意力值的Q（query）、K（key）、V（value）向量進行運算，而得到多個不同的注

意力值，進而產生比一般注意力機制所無法學習到的特徵；再用不同權重將多個注意力進行學習，而產生出在重要的語句序列有更重要注意比重，使語言模型有更好的翻譯結果。在注意力的計算上使用 scale dot-product 的計算方式，原因為此方式在高維度的計算上相較於累加式更有效率及節省空間，最後再經過 softmax 得出注意力值。

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$
$$head_1 = Attention(QW_i^Q, KW_i^K, VW_i^V)$$
$$Multihead(Q,K,V) = Concat(head_1,\dots,head_h)W^O$$

自注意力層是在編碼器及解碼器間，此層對前一層之注意力的關係。在編碼器及解碼器交接之注意力層中，K值為前一層解碼器注意力層之輸出，Q值為前一層編碼器注意力層之輸出；在編碼器及解碼器的注意力層則Q、K皆為前一注意力層的輸出。如此有兩個優點，一為使用自注意力只會關注語句與鄰近語句之關係，在翻譯工作上能達到高效果，也符合語言學之性質；二為如此能夠減輕運算效率，相比 RNN 或 CNN 的空間、時間複雜度均降低許多。

而 Transformer，及 3.2 BERT、3.3 ALBERT、3.4 Reformer 皆使用此多自注意力機制。



圖（三）

3.1.2. 結論

Transformer 在 BLEU（英德/英法翻譯測試）上均在當時得到最好的效果，並且使用相對於其他模型在運算時間上有非常卓越之進步。並實驗得出只使用多自注意力機制而非 RNN、CNN，並不會使得效能降低。

Transformer 使用多自注意力層取代循環層，而在時間上及效能上均達到卓越的效果，在翻譯的工作上達到當時最佳的效能，更啟發往後研究語言模型的研究者對注意力層新的認知。

3.2. BERT（Pre-training of Deep Bidirectional Transformers）[2]

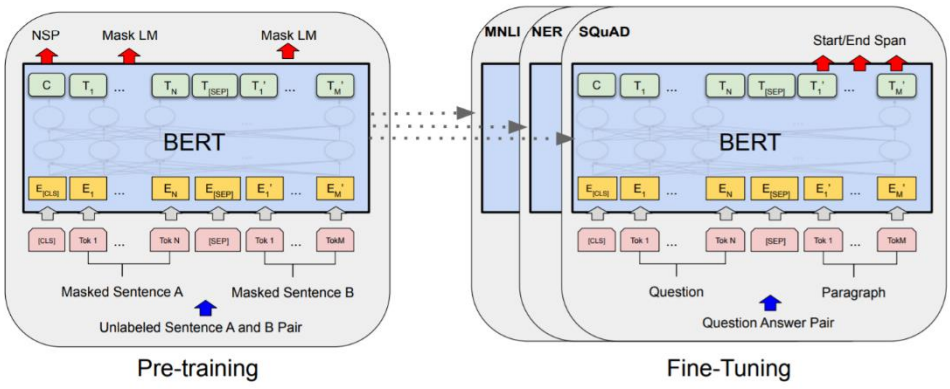
3.2.1. 架構

BERT 是取用 Transformer 編碼器的部分，並使用遮罩語言模型使得模型能學會不只單向的文字特徵，最重要的一點是提出預訓練步驟對語言工作的重要性，使用預訓練模型再微調至所要執行的工作上。其流程如圖（四）。

預訓練中，遮罩語言模型取代以往語言模型預測下一個字的訓練，而是隨機將一定比例之字詞進行遮蔽，並訓練編碼器預測遮蔽值。此步驟的目的是得到更完善的自然語言學習，因語言並非只有由前一個字或前句來得到。Google 以龐大的記憶體及運算量，使用英文維基百科及 BooksCorpus 資料庫，預訓練出對英文有一定理解之 BERT，讓大家在微調時可以使用一般人無法得到之龐大模型。

在微調的過程中，再將 BERT 訓練至目標工作上，如問題回答、語句生成、判斷語意、語法正確等等。

最後，BERT 新增下句預測的判斷，對於模型有些微的改善。



圖（四）

3.2.2. 結論

BERT 在各個測試集，如 GLUE，有多種語言工作的、SQuADs，問題回答、SWAG，語句生成，中均得出當時最佳之結果；並證明使用預訓練而不需要針對每個工作重新訓練即可得到最好的效果。

BERT 提出雙向編碼之 Transformer 模型，並使同一個預訓練模型能非常好的實作在各個自然語言處理的工作上。

3.3. ALBERT (a lite BERT) [3]

3.3.1. 架構

在訓練效果好的語言模型時，通常需要大量的 GPU/TPU 來運行。ALBERT 使用新的文字嵌入方式及參數跨層分享來降低模型訓練所需要的空間及時間，因此稱作 A Lite BERT。

ALBERT 將原本的複雜度 $O(V * H)$ 的文字嵌入層，V為單字的長度，H為隱藏層數，運用矩陣分解的方式，將嵌入層的複雜度降低為 $O(V * E + E * H)$，E為嵌入變數，如此可以省下許多計算時間及空間，因為當模型非常巨大時，隱藏層數及文字長度皆非常巨大，而使用此矩陣分解嵌入能大幅減少參數的數量以節省時間及空間。

接著，與其每層都計算所有參數，不如將層與層之間的參數共享，若此方法並不會影響模型的效能，參數跨層分享可以使模型訓練及計算時降低許多，進而改善時間及空間的運算效率。

最後，ALBERT 將 BERT 的下句判斷改為語句順序判斷，目的是讓模型能學習到更完善的義，而語句順序判斷明顯比下句判斷還要更能在各個工作上得到好的效果。

3.3.2. 結論

ALBERT 實驗得出，使用上述兩方法來使 BERT 運行更快速及空間更少，並不會讓模型品質降低，而若將省下的時間及空間來進行比 BERT 更長、更大量的訓練，會擁有比 BERT 更好的結果。

雖然 ALBERT 在進行與 BERT 相同時間的學習後，由於效率提升而得到當時最好的結果，但由於其過於巨大的架構及空間上的需求，注意力層仍有改善空間。另外，雖然語句順序判斷能有效增加模型的準確度，但是否還有更好的學習方式，有待發現。

3.4. Reformer (Reformer, The Efficient Transformer) [4]

3.4.1. 架構

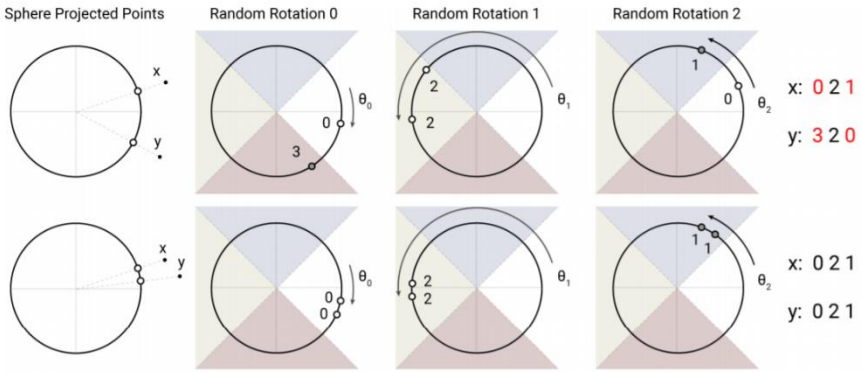
大型的 Transformer 雖然能夠得到非常好的結果，但訓練過程非常昂貴、消耗資源，特別是在長序列上。Reformer 使用 Locality Sensitive Hashing 之注意力層使得空間複雜度大幅降低，並使用可逆的殘餘層讓激活函數（activation）只儲存一次，使得 Reformer 成為空間效率更好的 Transformer。

Locality Sensitive Hashing（位置敏感雜湊）是一種以雜湊方式來計算注意力值的方法，它將原本 Transformer 的 Dot-prodoct 注意力層進行修改，原先之計算方法為$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$，需要的複雜度為$QK^T$也就是$N^2$的複雜度，每個$Q$向量之計算如下式所示，其中$o_i$代表其中一個注意力向量，$P$代表所有向量的子集，$z$為原先 softmax 對於$o_i$之影響。

$$o_i = \sum_{j \in P_i} exp(q_i * k_i - z(i_i P_i)) v_i / \sqrt{d_k}$$

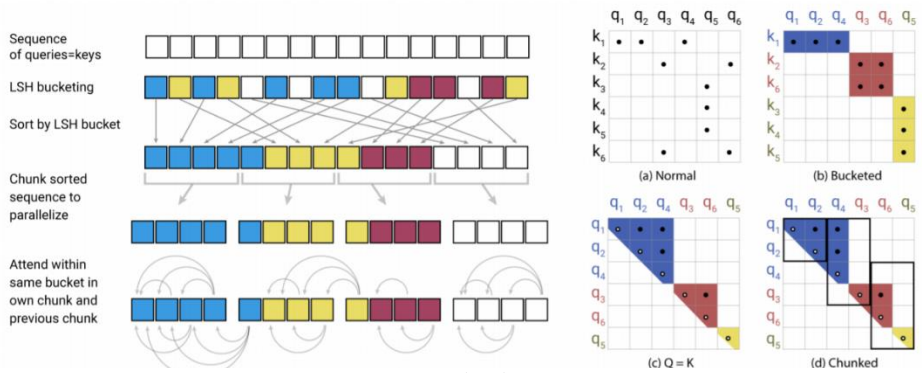
Reformer 提出將文字向量Q依照其相似程度分區（Bucket）一起運算，如此分區便能只計算每個區塊重要之處，而非完整的矩陣計算，而分區則是運用雜湊之技巧，如圖（五）所示，若兩者的Q向量極為相似，在經過多種隨機轉換後，兩個Q向量依然會在同一分區中，便可以歸納此兩個詞的相似度極高，因此稱此注意力機制為位置敏感雜湊。因此上式中的P_i即會變成只有計算相似度較高之Q向量及K向量。如下式所示，h為雜湊之函式：

$$P_i = \{j : h(q_i) = h(k_i)\}$$



圖（五）

經過雜湊後，由於Q向量及K向量均為從學習文字編碼而來，由序列長度決定，因此若是直接使得$Q \equiv K$，便能使此注意力層在計算時儲存更少之空間，因此讓$k_j = \frac{q_j}{||q_j||}$，進行排序後，依照 Bucket 數量及序列長度決定 Chunk 數量，$chunk = \frac{2 * seq\ length}{n_{bucket}}$，如圖（六），進行運算。



圖（六）

可逆式殘餘層儲存激活式之方法，即儲存最上層之參數，再經由運算回推各層，如此即不需要空間儲存所有參數。如下式所示：

$$Y_1 = X_1 + Attention(X_2) \qquad Y_2 = X_2 + FFd(Y_1)$$

其中再將所有計算的矩陣經由分解來降低儲存之空間，如下式：

$$Y_2 = \left[Y_2^{(1)}; \ldots; Y_2^{(c)}\right] = \left[X_2^{(1)} + FFd(Y_1^{(2)}) + \ldots + X_2^{(c)} + FFd(Y_1^{(c)})\right]$$

3.4.2. 結論

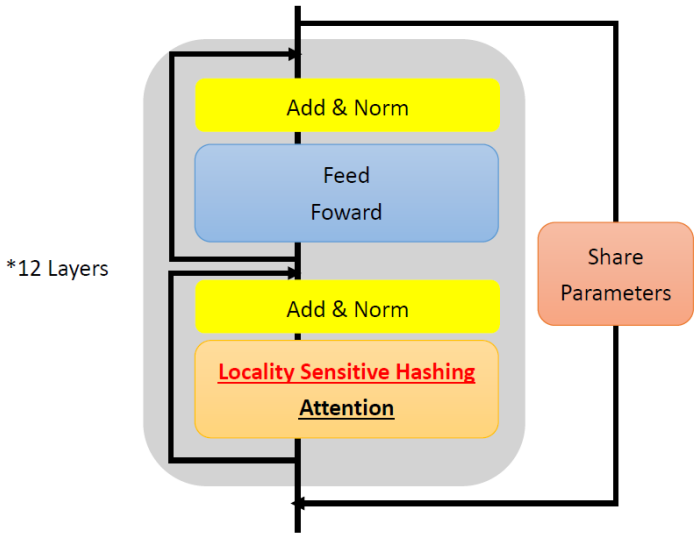
Reformer 成功使用 Locality Sensitive Hashing 使注意力層之空間複雜度由原本的$O(N^2)$降為$O(N\ lg\ N)$，並用可逆式殘餘層運算方法，兩者結合後 Reformer 在空間複雜度達到新低，且並不會對 Transformer 之結果產生巨大的影響。

Reformer 彌補了 Transformer 無法在較長語句上進行有效率運算的缺點，並可以使得更大、更多參數的基於 Transformer 架構之語言模型有更好的運算能力，生成出更長序列的語句，空間的節省讓更多人能使用到大型 Transformer 架構之語言模型。

四、 實驗設計

4.1. 架構介紹

4.1.1. 注意力層



ALBERT-LSH Structure

圖（七）

將 ALBERT 原先之 Dot-product 注意力層進行更改，原先使用到Q、K、V向量進行$softmax(\frac{QK^T}{\sqrt{d_k}})V$之注意力計算方式，依照 Reformer 中之介紹，變更為 Locality Sensitive Hashing 注意力機制，如圖（七），有三個重點。

i. $softmax(\frac{QK^T}{\sqrt{d_k}})$得到的結果，由於實際上針對每個$q_i$其$QK^T$運算時最主要的貢獻最主要來自於與其相似對應的幾個 key，因此實際上我們並不需要計算整個QK^T而只需要與相似的 key 計算即可。

ii. 更進一步，由於q_i與k皆由文字遷入時產生，因此可以假設q與k相等簡化整個運算。

iii. 基於上述兩個推論，藉由 Locality Sensitive Hashing 找出相似的q做運算。

我們選擇使用 Multiprobe scheme for the cross-polytope LSH 作為雜湊的方法，透過一個隨機高斯旋轉矩陣與$k_j = \frac{q_j}{||q_j||}$相乘，相當於將$k_j$在圓上做隨機旋轉，如圖（五），再透過找到相乘後最大值所在的區域即為所對應的 bucket，如下式：

$$\Pr_{A^{(1)},\dots A^{(l)}}[h_i(p) = r_{v_i}^{(i)} \; for \; all \; i \in [k] | A^{(i)}q = x^{(i)}]$$

其中A代表隨機高斯旋轉矩陣、x為q旋轉後的 hash 值、r_v代表x在各個 dim 絕對值大到小的第 v 個值。

藉由多次旋轉可以降低將相似的q分配到不同的 bucket 之中。根據各個 bucket 將各個q做排序並定義，只針對同一個 bucket 內的q做 attention，如下式：

$$o_i = \sum_{j \in \widetilde{P_i}} \exp\left(q_i * k_j - m(j, P_i) - z(i, P_i)\right) v_j,$$

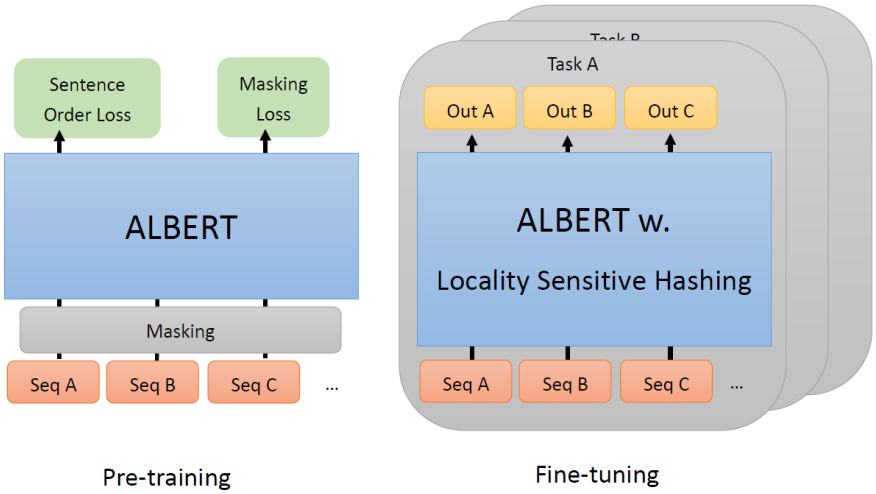
$$where \; m(j, P_i) = \begin{cases} \infty \; if \;\; j \notin P_i \\ 0 \; otherwise \end{cases}$$

由於各個被分配到各個 bucket 的數量不平均，因此藉由下式的方式讓排序後的 bucket 可以前後連接，讓相似的q在相鄰的 bucket 中仍然能夠做運算。

$$\widetilde{P_i} = \{j : \left\lfloor \frac{s_i}{m} \right\rfloor - 1 \leq \left\lfloor \frac{s_j}{m} \right\rfloor \leq \left\lfloor \frac{s_i}{m} \right\rfloor\}$$

4.2. 模型架設

4.2.1. 預訓練



圖（八）

預訓練之用意在於：使得模型先學習大部分之自然語言的特徵，假設語言模型需要做的工作為回答問題，他必須先學習英文的文法、結構等特徵，因此必須做預訓練的工作。預訓練通常需要極大的時間及資源

進行運算，縱使我們使用已經加上 Locality Sensitive Hashing 的 ALBERT 語言模型在本機進行預訓練，也無法達到 Google 使用多 TPU 進行長時間訓練的效果。因此我們使用 Google 在網路上公布之標準 ALBERT 預訓練模型，擁有 12 層的編碼器，隱藏層大小 H 為 768，嵌入大小 E 為 128，並使用英文維基百科、BookCorpus 進行預訓練，序列長度為 512，Batch Size 4096，以及 Learning Rate 0.00176。

4.2.2. 微調（Fine-tune）

我們使用加上 Locality Sensitive Hashing 的 ALBERT 進行微調至各個不同工作的步驟。由於硬體的限制，在微調的步驟無法進行如預訓練般大量之訓練步驟，我們使用同樣 12 層之編碼器，隱藏層大小為 768，嵌入大小 128，序列長度為 128 至 64，依照不同的工作及 GPU 能夠負荷之大小，Batch Size 32，以及 Learning Rate 1e-5 這兩個變數會稍作調整。

4.3. 實驗流程

4.3.1. 測試集介紹

不同的測試集能觀察出 Locality Sensitive Hashing 在不同自然語言處理工作上的成效，我們主要使用 GLUE，包含大量自然語言理解的工作，另外測試 ALBERT 達到非常好之結果的 SQuADs，測試問題回答等自然語言生成的工作。

資料集	簡介
MRPC	判斷二語句是否同義
MNLI	分類不同風格之語句
SST-2	對電影評論情感分析
SQuAD1.1	依照文章回答問題
SQuAD2.0	新增無法回答之問題

4.3.2. 模型運作正確性測試

目的：

檢測模型是否有進行學習，是否成功替換注意力層。

方法：

針對訓練小型資料集時的 Loss 進行觀察，是否有逐漸收斂的趨勢。因硬體設備及效能限制，選擇 GLUE 的 MRPC 進行測試。訓練 10 個 epoch 後，觀察 Loss 是否有收斂的趨勢。

4.3.3. 模型準確度

目的：

檢測模型是否能達到目標工作之要求。針對同一個資料集，對 ALBERT 及 ALBERT-LSH 進行相同設定之測試，探討 ALBERT 在經過 Locality Sensitive Hashing 之注意力機制後，是否能維持相同之準確度，又或者會因空間上的節省而讓些許特徵被忽略，導致準確度降低。

方法：

針對不同資料集進行學習後預測準確度。與 ALBERT 互相比較。

五、 實驗結果

5.1. 模型運作正確性測試

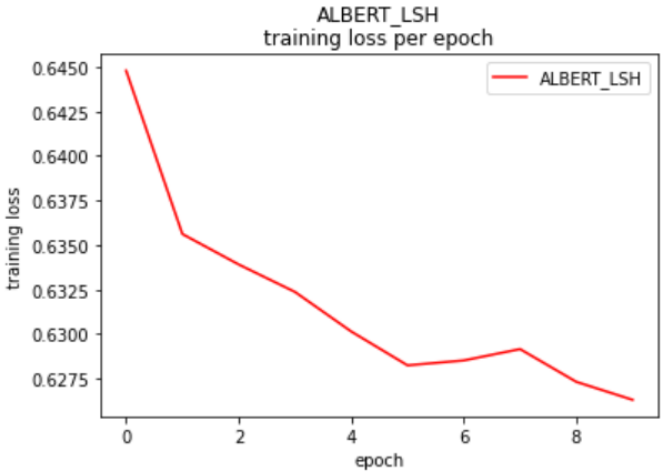
為了確認接好的 ALBERT_LSH 是否有正確的訓練，我們將 ALBERT_LSH 在 MRPC 資料集上訓練多個 epoch，檢測其是否有成功收斂每次的 Training Loss。

參數設定：

Module	Seq length	Batch size	Learning rate	epochs
ALBERT_LSH	64	32	1.00E-04	10

結果：

每個 epoch 的 Loss 值穩定的下降，如圖（九），雖然下降頻率不高，但當訓練越久，語言模型的準確值會越來越高。



圖（九）

5.2. 模型準確度

結果：

	MRPC	MNLI	SST-2	SQuAD1.1	SQuAD2.0
ALBERT_LSH	0.78	0.35	0.51	13.93	50.07
ALBERT	0.88	0.84	0.93	78.45	72.63

分析：

MNLI、SQuAD1.1、SQuAD2.0、SST-2 與單純文意的分析的 MRPC 不同，除了要理解基礎語意之外可以分類為「產生」感情、風格、回答等，明顯需要更加細緻的訊息才能完整表達。

因此可以歸納出一個重點，自注意力層因為使用 LSH 而造成的訊息丟失或許都是來自於關於語言的「細節」部分，而高度相似的文意訊息則會被保留，所以 ALBERT_LSH 在做有關於需要對語言的細節部分有高度可能會表現較差。

5.2.1. MRPC：

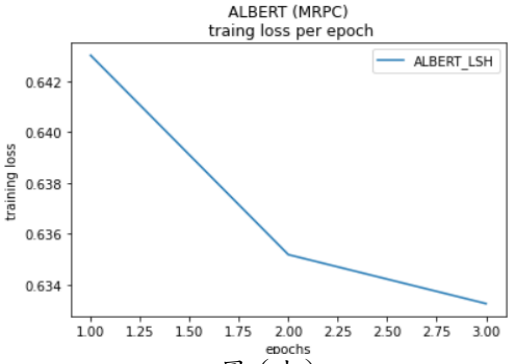
參數設定：

Module	Seq length	Batch size	Learning rate	epochs
ALBERT_LSH	64	32	5.00E-06	3
ALBERT	64	32	1.00E-05	3

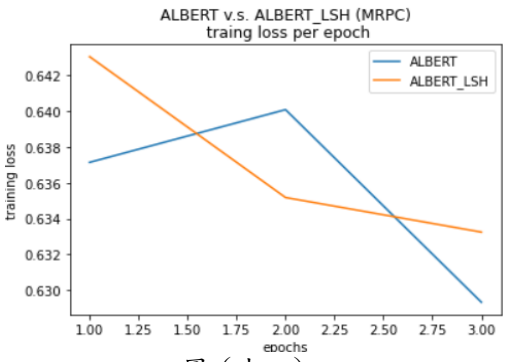
結果：

Module	eval_f1	eval_acc_and_f1
ALBERT_LSH	0.823	0.777
ALBERT	0.899	0.878

分析：



圖（十）



圖（十一）

MRPC 的任務是屬於文意了解的類型，經由輸入兩個不同的句子，判斷兩個句子之間的意思是否相等，相較於其他的任務類型，所需要學習的資訊最為簡單。

由於 MRPC 這個資料集訓練較快，我們可以找出合適的 learning rate 調整我們的 ALBERT_LSH。從結果與 Loss 的圖可以觀察出兩點，如下

i. 跟 ALBERT 的正確率相差其實不是過於巨大，可以說明 ALBERT_LSH 在即使因為使用雜湊而使資訊量降低，仍然可以充分的學習正確的語意資訊。

ii. 從圖(十一)可以觀察到 ALBERT_LSH loss 降低的趨勢比 ALBERT 慢上許多，因此能夠知道 ALBERT_LSH 即使在簡單的語意判斷上就必須花上更多的時間才能趕上 ALBERT 效果。

5.2.2. MNLI：

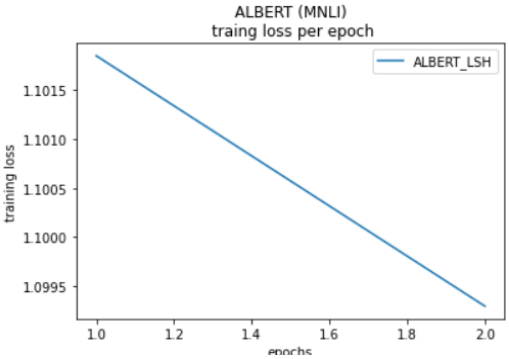
參數設定：

Module	Seq length	Batch size	Learning rate	epochs
ALBERT_LSH	64	16	3.00E-05	2
ALBERT	64	16	3.00E-05	2

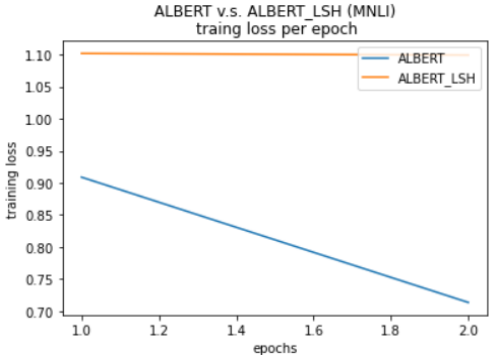
結果：

Module	eval_acc
ALBERT_LSH	0.352
ALBERT	0.840

分析：



圖（十二）



圖（十三）

MNLI 的任務是在於分類兩個不同「風格」的語句，要在基於理解文意的基礎之上，更進一步的總結歸納出較為隱晦的風格，比起 MRPC 所需學習的資訊量更多。

從結果上可以明顯的看出比 ALBERT 的效果差上不少，說明 ALBERT_LSH 即使可以理解文意，但是無法有效的學習隱含在文章中的風格，造成這種結果的因素可以推測可能由於自注意力層使用 LSH 造成訊息丟失，因此使得剩下的訊息不足夠讓 ALBERT_LSH 有效率的學習風格。此外，礙於設備上的因素，使得我們無法跑出訓練足夠多的 epochs 而導致這樣的結果。

5.2.3. SST-2：

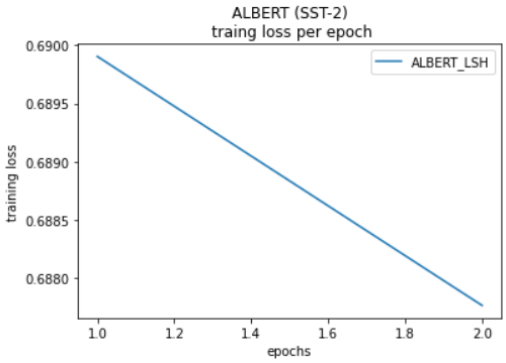
參數設定：

Module	Seq length	Batch size	Learning rate	epochs
ALBERT_LSH	64	16	5.00E-06	2
ALBERT	64	16	1.00E-05	2

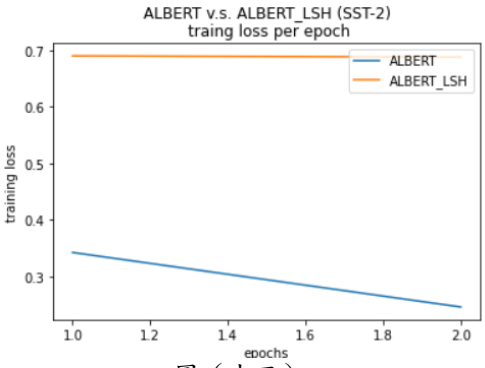
結果：

Module	eval_acc
ALBERT_LSH	0.509
ALBERT	0.931

分析：



圖（十四）



圖（十五）

SST-2 的任務為分類區分電影評論的「感情」，基於文意的理解從中歸納生成所謂的感情，比起 MRPC 僅理解文意需要更多的資訊。

ALBERT_LSH 的表現比 ALBERT 差上非常多，從中再次說明了因為 ALBERT_LSH 的自注意力層使用 LSH 而造成的資訊缺失，使得 ALBERT_LSH 無法有效率的學習藏在文意中的「感情」。但是從圖(十四)可以看出 ALBERT_LSH 的 loss 有著明顯的下降趨勢，若再將訓練的 epochs 數量增加，應該能夠改善 ALBERT_LSH 的表現。

5.2.4. SQuADs：

參數設定：

Module	Seq length	Batch size	Learning rate	epochs
ALBERT_LSH	64	32	1.00E-04	2
ALBERT	64	32	1.00E-05	2

結果：

Module	SQuAD1.1		SQuAD2.0	
	Accuracy	f1 score	Accuracy	f1 score
ALBERT_LSH	8.42	13.93	49.90	50.07
ALBERT	70.07	78.45	70.23	72.63

分析：

從結果可以發現 ALBERT_LSH 在 v1.1 與 v2.0 上與 ALBERT 都有著非常大的落差，甚至可以說上做得非常不好。推測是因為 SQuADs 屬於問題回答的自然語言處理工作，模型除了要能夠理解文意的上下文關係之外，它必須產生出一串文字，需要理解的資訊量明顯大上許多。推測可能的原因有兩種可能。

i. 由於 LSH 造成的 Loss 傳遞不易的問題上，沒辦法在兩個 epochs 上得到一個最佳結果，礙於 SQuADs 訓練需要花費許多時間，我們無法讓它跑到足夠多的 epochs 做比較。

ii. 在一開始使用 LSH 注意力機制的 Reformer 並沒有在 SQuADs 上進行測試，僅在翻譯的工作上做訓練，因此有可能是 LSH 注意

力機制所造成的資訊減小問題使得它並不適合問題回答的自然語言處理工作。

六、 結論

Locality Sensitive Hashing 在自然語言處裡工作上是可行的，並且可以減少模型運算所需空間，使訓練架構更龐大，是以訓練時間換取運算空間的方法。我們的 ALBERT_LSH 也就是減少了硬體使用的空間，但雜湊的注意力機制會丟失或簡化文字特徵使模型的準確率下降，使得有以下兩種狀況

i. ALBERT_LSH 能夠充分理解文意間關係。
ii. 使用 LSH 而造成的訊息丟失大多屬於「細節」訊息，因此額外創造出文字、情感等需要藉由理解文句細節的任務(SQuADs、SST-2 等)正確率會下降非常多。

又或許需要更長的訓練時間能夠改善第二點，又抑或是此雜湊方法的極限就是如此。

基於這次的專題，要是有足夠的資源的話，對於未來可以持續改進的方向有

i. 對 ALBERT_LSH 做完整的預訓練：從預訓練開始就讓整個模型是用 ALBERT_LSH 去學習文法、結構等，讓模型能夠以最適合 LSH 的方式理解文章，而並非是基於 ALBERT 的理解移花接木到 LSH 上。
ii. 訓練更多的 epochs：從各個資料集的實驗中可以看出 ALBERT_LSH 的 loss 都有著下降的趨勢，因此要是讓訓練的 epochs 增加，能夠改善表現。
iii. 更精細的調整各個參數：在 MRPC 這個資料集的訓練我們有調整參數並且得到不錯的結果，因此若是時間與硬體充足，妥善地對每個資料集調整參數，勢必能改善不少模型的表現。
iv. 硬體的改善：即使我們使用 Google Colab Pro 進行模型的訓練，仍然有許多參數是超出硬體的負荷範圍而無法調整，若是改善硬體條件，除了能更快調整參數之外，更能有更多參數的彈性範圍，對模型的改善能夠更快完成。

七、 參考文獻

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In NAACL 2019

[3] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In ICLR 2020

[4] Nikita Kitaev, Łukasz Kaiser, Anselm Levskaya : REFORMER: THE EFFICIENT TRANSFORMER. In ICLR 2020.
[5] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. CoRR, abs/1509.02897, 2015.