國立臺灣大學電機資訊學院電信工程學研究所

碩士論文

Graduate Institute of Communication Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

輕量化微調的預訓練語言模型引導之系統性分析

Systematic Analysis of Pre-trained Language Model Priming for Parameter-efficient Fine-tuning

黃世丞

Shih-Cheng Huang

指導教授: 李宏毅 教授

Advisor: Hung-yi Lee, PhD.

中華民國 112 年 6 月

June, 2023

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

致謝

晃眼之間，三年的碩士生涯也即將迎來尾聲。

我自認不是個擅長做研究的碩士生，在大學時也對未來感到十分迷茫，但卻十分幸運地在大三時修習了宏毅老師的專題課程，讓當時連 PyTorch 都不會寫的我得以一窺機器學習的堂奧。從讀論文、參與 meeting 開始，逐漸熟悉這個領域，在研究碰壁時、寫論文遇到困難時、投稿國際會議被拒時，宏毅老師總是能從更高的角度提出有效的建議，並鼓勵我們繼續努力。無論是學術上抑或是研究態度上，老師都給了我非常大的幫助，真的非常感謝老師當初願意接納我當專題生以及進入語音實驗室。另外，也很感謝琳山老師主持著語音實驗室這個大家庭，無論是實驗室的風氣或是擁有的資源都使我獲益良多。

研究的日子並非總是一帆風順，有許多個日夜都無可避免地在與焦慮和降不下去的訓練損失奮戰，幸好我在實驗室有著兩位知心好友，軒哥和子晴姊。不管是研究遇到瓶頸、或是臨時需要有人幫忙看論文、或是遇到不順心的事需要人大吐苦水，他們總是義不容辭地幫忙。文字難以充分表達我對他們的謝意，但我還是必須要說，能夠在研究所遇到這樣的朋友真的是三生有幸，少了他們的支持我絕對無法完成我的碩士學業。

我也要感謝我的家庭提供我一個無後顧之憂的環境，讓我的求學之路不需要耗費心力為了柴米油鹽而煩惱。從小看著父親寫程式、接電路也啟發了我對於電機領域的興趣，也才有今天就讀電信所的我。

能夠寫出這一篇論文，實屬有著太多的幸運，十分感謝一路走來幫助過我的各位。

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

摘要

隨著預訓練語言模型 (Pre-trained Language Model) 的參數量變得越來越大，輕量化微調 (Parameter-Efficient Fine-tuning) 顯得更為重要，但在少樣本學習 (Few-Shot Learning) 的情境下進行輕量化微調的效果卻遠遠不及微調整個預訓練模型。

為了解決這個問題，本研究提出在進行輕量化微調前加入一個稱為「引導」(Priming) 的訓練過程來強化預訓練語言模型的量化微調效果，並且在一個包含 160 個不同自然語言處理任務的少樣本資料集上驗證了本方法的有效性。相較於直接進行輕量化微調，經過引導的模型在 ARG(Average Relative Gain) 分數上達到了近 30% 的進步量，其表現也超越了其他的輕量化微調基石模型。

除此之外，我們針對引導模型的方法進行了系統性的實驗，分析了在引導階段使用不同訓練演算法和訓練不同參數對於引導效果的影響，並找出最有效的引導方法。本研究的結果將能有效增強輕量化微調在少樣本學習上的表現，並使得大型預訓練語言模型的微調和使用更加有效率。

關鍵字：自然語言處理、附加器、輕量化微調、元學習、多任務學習、少樣本學習

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

Abstract

As the parameter size of pre-trained language models (PLMs) continues to grow, parameter-efficient fine-tuning becomes more important. However, the effectiveness of parameter-efficient fine-tuning is far inferior to that of fine-tuning the entire pre-trained model in the context of few-shot learning.

To address this issue, this study proposed a training process called ”priming” to enhance the effectiveness of parameter-efficient fine-tuning by strengthening the pre-trained language model before performing the downstream fine-tuning. The effectiveness of this method was verified on a few-shot dataset consisting of 160 different NLP tasks. Compared to directly performing parameter-efficient fine-tuning, the primed model achieved an improvement of nearly 30% in ARG (Average Relative Gain) score and outperformed other parameter-efficient fine-tuning baselines.

In addition, we conducted systematic experiments to analyze the impact of different training algorithms and different upstream trainable parameters and identify the most effective priming method. The results of this study will effectively enhance the performance of parameter-efficient fine-tuning in few-shot learning and make fine-tuning and usage of large-scale pre-trained language models more efficient.

Keywords: natural language processing, adapter, parameter-efficient fine-tuning, meta-learning, multi-task learning, few-shot learning

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

目錄

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

圖目錄

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

表目錄

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

第一章　導論

1.1　研究動機

隨著科技的進步，人類不斷地往人工智慧的目標邁進，嘗試著讓機器也擁有如同人類般的認知、思考與推理能力，而其中一個十分重要的領域就是自然語言處理 (Natural Language Processing, NLP)。自然語言處理是一門運用語言學與機器學習的技術去了解文字的結構與意義的領域，其中的子領域又包含了文本情感分析、自然語言推理、句義相似度等不同任務，這些任務一直都是機器學習的一大挑戰。DeepMind 於 2018 年發表了 GLUE (General Language Understanding Evaluation) 基準，GLUE 基準集合了數個不同的自然語言理解 (Natural Language Understanding, NLU) 任務，這些任務囊括了自然語言理解的各種面向，同時也包含了單個句子分類、句子對分類等不同的輸入形式，有的任務具備充分的訓練資料，但其中也有些任務的訓練資料較為有限或是和測試資料的類型不匹配。這些任務的多樣性鼓勵模型學習跨任務的語言學知識，並盡量善用有限的訓練樣本。

在 2018 年，Google 提出了 BERT (Bidirectional Encoders Representations for Transformers) 模型，在包含 GLUE 基準的 11 項任務上都超越了當時的最佳紀錄，更超越了人類的分數，是自然語言處理的又一里程碑。有別於以往訓練模型時大量使用標註資料的傳統，在 BERT 的訓練過程中使用了大量的無標註資料，由於無標註資料不需要進行人工標註，一般來說收集資料的成本較低，也因此較容易大量取得。除此之外，BERT 採用了先預訓練 (Pre-train) 再進行微調 (Finetune) 的兩階段式訓練，先利用豐富的無標註資料設計自監督 (Self-supervised) 任務進行預訓練，再針對不同的下游任務 (Downstream Task) 使用有標註資料進行微調。這樣的訓練方法為模型的表現帶來了極大的提升，也開啟了大型預訓練模型的時代，在 BERT 之後，有越來越多的大型預訓練模型被提出，並在諸多自然語言處理的任務上都取得了相當出色的表現。

doi:10.6342/NTU202302478

但是支撐這樣亮眼表現的是越來越巨大的儲存空間以及龐大的運算資源，數量高達億萬等級的參數量導致一般人難以自行訓練和儲存巨大的語言模型，尤其單一模型的大小已經相當龐大的情況下，對於不同的下游任務又要分別進行微調成了艱鉅的任務。若要對多個下游任務進行微調，傳統的做法上需要複製多份預訓練模型的參數，並且獨立對這些不同的任務進行微調，這樣的做法相當耗費運算資源以及儲存空間。因此，學者們陸續提出了將模型輕量化的方法，其中霍氏 (Houlsby) 所提出的附加器 (Adapter) 模型大大降低了模型所需的儲存空間。附加器模型是一組輕量化的神經網路模塊，可以被添加於原來的轉換器模塊當中，其結構中的瓶頸 (bottleneck) 設計讓附加器模型的參數量相當小，僅有大約原模型的 3.6% 而已。霍氏的論文指出，對於不同的下游任務，我們可以使用一組共享的預訓練模型參數，只對每個任務額外添加附加器模型，並且只需要訓練附加器模型即可達到和微調整個模型幾乎相同的表現，因此我們並不需要對每個下游任務分別儲存一個獨立的模型，而是儲存一個預訓練的語言模型參數以及許多任務專用 (task-specific) 的附加器即可。

在實際應用上，有的任務並沒有足夠豐富的訓練資料，如某些少數民族語言的機器翻譯任務，對於這些任務我們只會擁有少量的有標註樣本，所以必須有效利用僅有的少數樣本，此種情境稱為少樣本學習 (Few-Shot Learning)。由於該任務本身具有的樣本數極少，因此除了從該任務的樣本學習以外，我們亦可以設計演算法使模型能夠從其他相關或類似的任務中學習某些跨任務 (Cross-task) 的特徵，使模型即使在少樣本學習的情境下依舊能夠獲取較好的表現。

雖然文獻 [12] 指出附加器模型能夠在微調少量參數的情況下就達到跟微調整個預訓練模型差不多的表現，但也有些研究 [35, 36] 指出，傳統微調附加器的方法在少樣本情境下的表現和微調整個模型存在著一定的落差。因此，本論文提出了在進行下游微調之前加入一個稱為引導 (priming) 的訓練階段，利用多個訓練任務學習跨任務的特徵，以加強模型在下游任務上的表現。

要使模型學習跨任務的特徵，其中一個主流的作法為多任務學習 (Multi-task Learning)，多任務學習不同於傳統在單一任務上訓練的作法，而是在訓練時讓同一個模型一次進行多個任務，這些任務可能有相同或不同的損失函數 (Loss Function)，而使用梯度下降法更新參數時，則會讓這些不同任務的損失函數相加或平均後再進行反向傳播。這樣作法的好處是讓模型不只專注在某個特定的任務上，而是讓模型學習到跨任務的共同特徵，從而讓單一模型就能夠處理多個任

doi:10.6342/NTU202302478

務。

在少樣本學習的情境之下，另一種演算法元學習 (Meta Learning) 應運而生。元學習利用多個訓練任務來讓模型學習「如何學習」，每個訓練任務僅需要少量訓練樣本，在經過多個任務的訓練後，能夠在遇到新的任務時只依賴少量樣本就讓模型達到不錯的表現。

本論文提出了一個一般化的訓練框架 (framework)，可用於訓練一個預訓練自監督模型，使其在加上附加器或任何其他的輕量化模組之後，能夠在下游的少樣本任務上取得更好的表現。

1.2 論文研究方向

本論文欲探討在少樣本學習的情境下，如何利用多個少樣本的上游任務，來讓使用了附加器模型的轉換器模型在下游任務上達到更好的表現。我們除了著重模型在各任務上的表現外，也著眼於分析模型在跨任務上的一般化能力，並且探討了使用何種演算法，及在上游學習階段中訓練模型的哪些部分，對於下游微調更有幫助。

1.3 主要貢獻

本論文提出了在進行輕量化微調之前加入額外的「引導」階段來強化模型的表現，實驗證明該方法能夠大幅增加模型的表現，且本研究之方法不僅可用於附加器，亦適用於其他種類的輕量化微調方法，因此未來亦可應用於其他輕量化模組。

另外本研究對引導階段進行了系統性的實驗，並分析在引導階段採用的訓練方法及可訓練參數對於輕量化微調表現的影響，最後找出了對下游微調最有效的引導方法。

1.4 章節安排

本論文之章節安排如下：

doi:10.6342/NTU202302478

- 第二章：介紹本論文相關背景知識。
- 第三章：介紹轉換器模型加上附加器之二階段訓練框架
- 第四章：在 GLUE 上初步實驗之結果
- 第五章：主實驗結果分析討論
- 第六章：結論與展望

doi:10.6342/NTU202302478

第二章　背景知識

2.1　深層類神經網路

2.1.1　基本原理

類神經網路 (Neural Network) 最早的雛形可以追溯至西元 1943 年由麥氏 (McCulloch) 等人所提出的神經模型 [22]，他們運用電路建立了一個模擬生物大腦神經元運作的計算模型，雖然其運作方式與現代的類神經網路有所差異，但卻為類神經網路的發展打下了基礎。在西元 1958 年，羅氏 (Rosenblatt) 提出了感知器 (Perceptron) 模型 [27]，此時的感知器模型已經非常接近現代的感知器模型了，而將多個感知器模型堆疊在一起則構成了所謂的多層感知器 (Multi-Layer Perceptron) [14]，也就是最基本的類神經網路結構。類神經網路模型的參數可以透過反向傳播演算法 (Backpropagation) 來訓練，在反向傳播演算法中，模型的參數會透過梯度下降法 (Gradient Descent) 更新，經過訓練以後的參數能夠使類神經網路近似成適合解決特定任務的函數。

經過一段時間的發展與演變，再加上現代電腦越來越強的計算能力，便衍生了深層類神經網路這個分支。一般來說，深層類神經網路指的是不同於以往較淺、只有一層的類神經網路，而是疊加兩層以上、甚至可以深達數十層至數百層的類神經網路，而研究深層類神經網路的領域則被稱為深度學習。深度學習在許多領域如物體偵測、語音辨識、機器翻譯等任務上都帶來了大幅度的進步 [17]，而從類神經網路演變到深層類神經網路，不僅增加了模型的參數，其深層的結構設計更為模型帶來了更強的模擬函數的能力，使其能夠處理更多更複雜的任務 [6]。

doi:10.6342/NTU202302478

2.1.2 轉換器類神經網路

轉換器類神經網路 (Transformer) [32] 是西元 2017 年由谷哥 (Google) 的團隊所提出的模型架構，最初被應用在機器翻譯的任務，隨後被廣泛使用在各種自然語言處理、語音辨識、甚至電腦視覺的任務上。圖2.1中可以看到轉換器模型的結構主要由兩個部分所組成，編碼器 (Encoder) 和解碼器 (Decoder)。如圖所示，編碼器和解碼器內部分別由 N 層相同的編碼層 (Encoder Layer) 和解碼層 (Decoder Layer) 所組成。

圖2.2顯示每個編碼層包含兩個主要結構：自注意力 (Self-attention) 層以及全連接 (Fully Connected) 層；每個解碼層包含三個主要結構：自注意力層、跨注意力 (Cross-attention) 層以及全連接層。

轉換器模型運作的方式可以參考圖2.2，左下角的紅色長方形向量是編碼器的輸入句子，編碼器會把輸入句子編碼為隱藏向量 (Latent Vector)，接著由編碼層所編碼的隱藏向量會輸入解碼層，解碼器可以透過其中的跨注意力層來獲得輸入句子的資訊，最後輸出右上角的藍色向量。

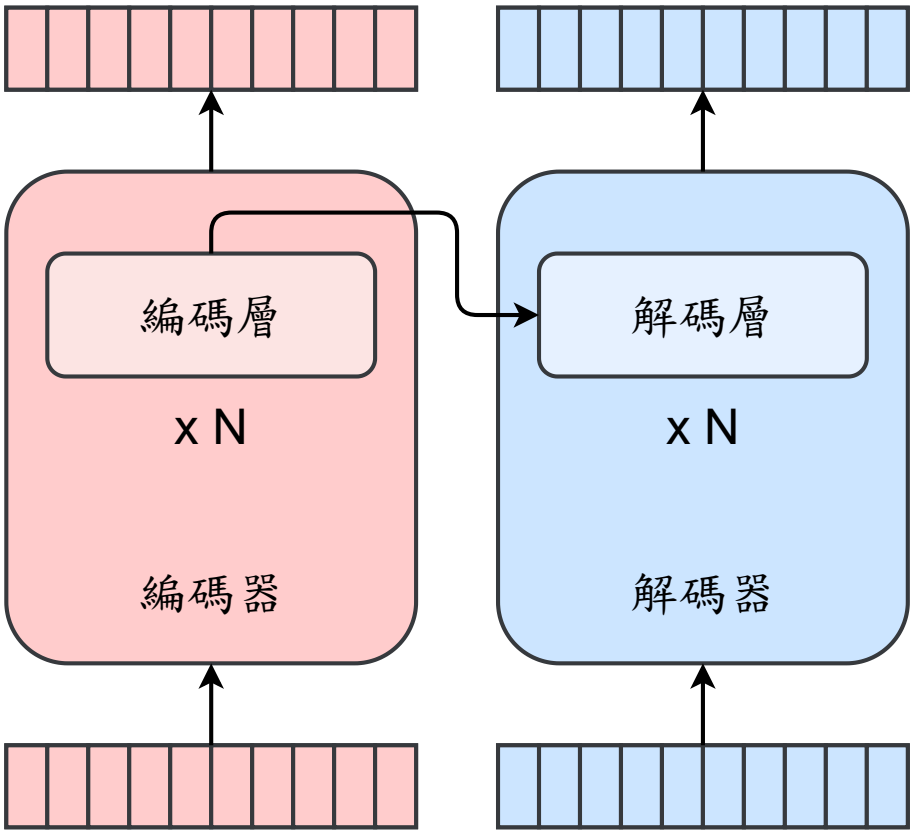


圖 2.1: 轉換器基本概念圖

doi:10.6342/NTU202302478

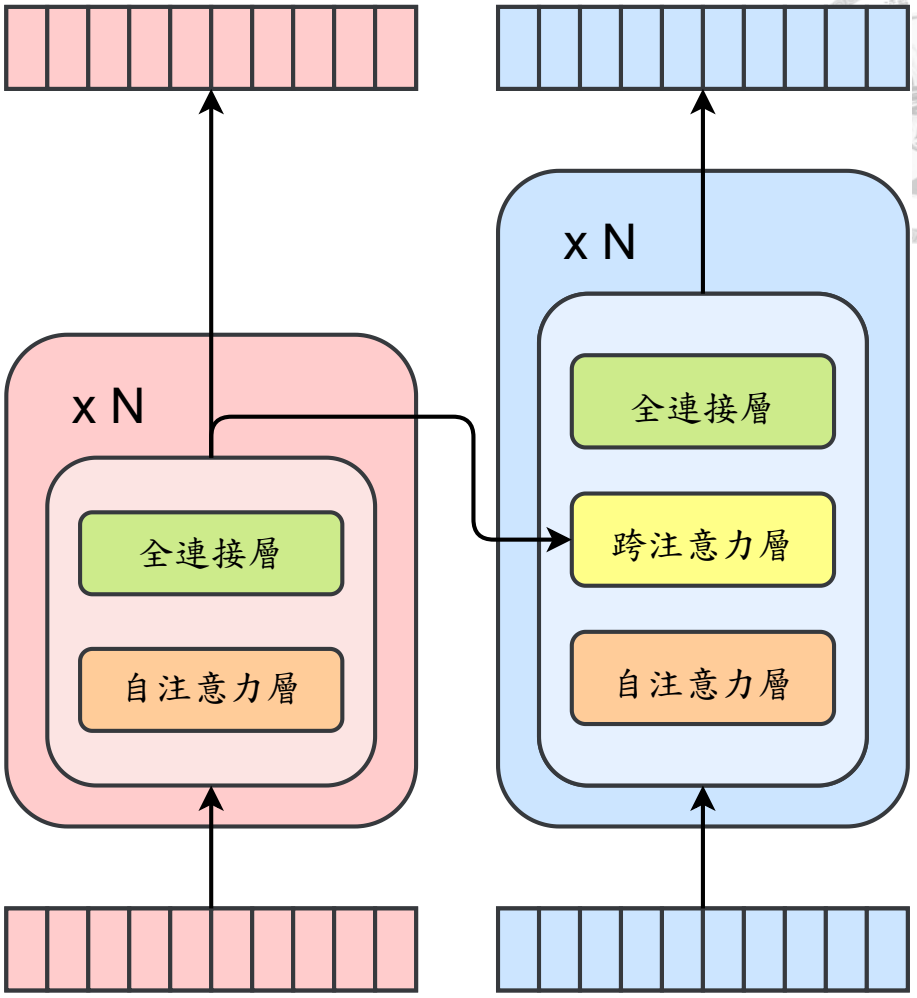


圖 2.2: 轉換器模型結構示意圖

注意力機制

注意力機制 (Attention Mechanism) 是一種類似於人類的「注意力」的機制，能幫助模型專注於較為重要的資訊上，最早由巴氏 (Bandanau) 所提出 [4]，用於機器翻譯上。以機器翻譯為例，模型在進行翻譯的過程中，會需要動態地參考輸入句子中不同部分的內容以便翻譯，而注意力機制就能幫助模型專注於較為重要的資訊上。

當時機器翻譯的傳統作法是利用遞迴式類神經網路 (Recurrent Neural Network) 所組成的序列對序列 (Sequence To Sequence) 模型 [31]，模型由一個編碼器和解碼器所構成，編碼器會將輸入句子的資訊編碼成一個固定維度的向量，接著將這個向量輸入到解碼器中，解碼器能夠透過這個向量來獲取關於輸入的資訊。而巴氏所提出的注意力機制則是運用類神經網路來計算不同的輸入文字向量所對應的權重，接著將此權重和對應的文字向量相乘，並將相乘後的向量加起來得到一個

doi:10.6342/NTU202302478

動態的句子向量。這個句子向量能夠幫助解碼器在解碼當下輸出的文字時，著重在較為重要的資訊上。以機器翻譯為例，假如我們想把"I like to eat apples." 翻譯成「我喜歡吃蘋果。」，當解碼器在解碼「喜歡」這個詞彙時，在原輸入句子中的"like" 會是比較重要的字，此時注意力機制的向量就會把權重集中在這個字上。

轉換器模型中所使用的注意力機制為「尺度點乘注意力操作」(Scaled Dot-Product Attention) [32]，其做法源自於蘇氏 (Sukhbaatar) 等人於西元 2015 年所發表的的端對端記憶網路 (End-to-end Memory Networks) [30]。記憶網路的概念是將資訊儲存在一個記憶資料庫內，記憶資料庫的內容實際上就是許多的向量，這些向量編碼了模型在執行任務時所需要的資訊，供模型查詢，而尺度點乘注意力操作即改良自記憶網路的作法。首先模型會將輸入編碼為三種不同的向量：詢向量 Q (Query Vector)、鑰向量 K (Key Vector)、值向量 V (Value Vector)，向量的名稱亦參考了記憶網路中的稱呼，Q、K、V 皆為輸入序列編碼後的向量排列而成的矩陣：

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_N \end{bmatrix}^\top \tag{2.1}$$

$$K = \begin{bmatrix} k_1 & k_2 & \dots & k_N \end{bmatrix}^\top \tag{2.2}$$

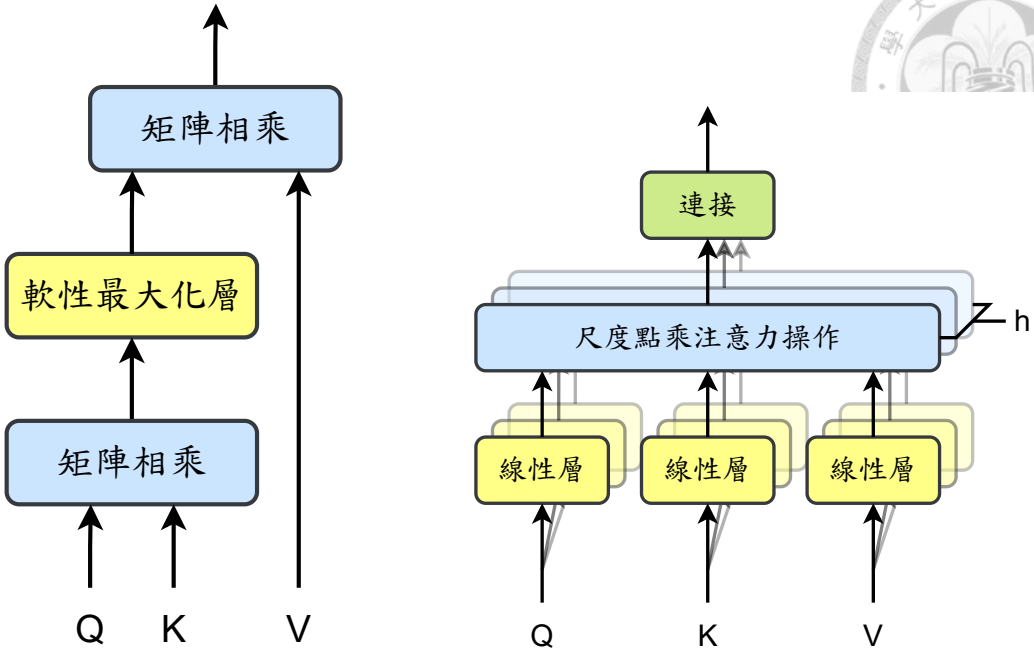
$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_N \end{bmatrix}^\top \tag{2.3}$$

其中 $q_i, k_i \in \mathbb{R}^{d_k}, v_i \in \mathbb{R}^{d_v}$ ，d_k, d_v 是編碼後向量的維度，N 是輸入序列的長度。接著模型會計算詢向量和鑰向量的內積，內積的數值代表了每一個詢向量和鑰向量的相似程度，同時也代表了該位置的輸入資訊對輸出的重要程度。所有詢向量和鑰向量的一對一內積可以寫成矩陣相乘的形式，相乘後的矩陣會再除以 $\sqrt{d}$ 再經過一個軟性最大化層 (Softmax Layer) 得到值向量的權重，最後值向量分別乘以對應的權重之後相加得到最終注意力層的輸出，完整的計算公式如下：

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V \tag{2.4}$$

尺度點乘注意力操作的運算可參考圖2.3a，相乘後的矩陣 $QK^\top \in \mathbb{R}^{N \times N}$ 代表了不同輸出字符 (Token) 對應到不同輸入字符的重要程度，而除以 $\sqrt{d_k}$ 是因為當編碼的維度增大時，內積的數值大小也會隨之增加，容易在訓練時產生梯度消失 (Gradient Vanishing) 的問題。在經過軟性最大化層後我們就得到了注意力權重，最後注意力權重和由值向量所構成的矩陣 V 相乘，以不同權重混合值向量後得到注意力層的輸出。

doi:10.6342/NTU202302478



(a) 尺度點乘注意力操作　　(b) 多頭注意力

圖 2.3: 注意力機制中的運算操作

除了尺度點乘注意力操作以外，轉換器模型中的注意力層還運用了另一種機制來強化模型的表現，那便是多頭注意力機制 (Multi-Head Attention)。多頭注意力機制設計了多個注意力頭 (Attention Head) 來捕捉輸入序列中不同的重要資訊，以改善只使用單一注意力頭時資訊過多而來不及捕捉的問題，其運作方式可以參考圖 2.3b。模型會先將維度為 d_{model} 的詢向量、鑰向量和值向量透過矩陣 W_i^Q, W_i^K, W_i^V 投影到 h 組維度為 d_k, d_k, d_v 的向量，而每一組詢向量、鑰向量和值向量會各自進行注意力機制的計算，得到 h 個注意力頭。最後再把這 h 個注意力頭所產生的向量連接在一起，並再經過矩陣 W^O 的投影，得到多頭注意力層的輸出，如以下式子所示：

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.5}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.6}$$

轉換器的注意力層結合了尺度點乘注意力操作以及多頭注意力機制，讓轉換器模型能夠準確捕捉輸入序列中的重要資訊。

doi:10.6342/NTU202302478

位置編碼

由於轉換器模型的設計是對輸入序列的每一個字符 (token) 平行化地進行運算，並不像傳統遞迴式類神經網路或是卷積運算那樣能夠將輸入的位置資訊隱含在輸入中，因此我們需要透過位置編碼 (Positional Encoding) 將輸入字符的位置資訊傳遞給模型。在轉換器模型的原始論文 [32] 中，作者採用了以正弦和餘弦函數編碼的位置向量作為位置編碼：

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{2.7}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{2.8}$$

其中 pos 是輸入字符的位置而 i 是輸入向量的維度。這樣的設計能夠保證對於某個偏移量 k 來說，位置編碼 PE_{pos+k} 都能夠被表示成 PE_{pos} 的線性組合，因此有利於模型學習不同位置之間的相對關係。在得到輸入向量的位置編碼後，我們就能直接把輸入向量和位置編碼相加，藉此將位置資訊融入到輸入序列中。

而在後續應用轉換器模型的研究 [8, 19, 24] 裡，則多採用可訓練的位置嵌入 (Position Embedding) 向量，亦有些論文 [25, 28] 採用相對位置編碼等不同的位置編碼方法。

編碼層

轉換器模型的解碼器是由 N 個相同結構的編碼層疊在一起所構成，每一層編碼層的輸入都是前一個編碼層的輸入，而單一編碼層的結構可以參考圖2.2，主要分為三個部分，自注意力層、全連接層以及層正規化。自注意力層即為小節2.1.2中所介紹的注意力機制之應用，當輸入的詢向量、鑰向量、值向量皆來自同樣的輸入向量時，這個注意力層我們稱之為自注意力層。輸入向量會先經過自注意力層，自注意力層的輸出會和原本的輸入相加，這樣的設計稱為殘差連接 (Residual Connection) [11]，相加後的輸出會經過一個層正規化 (Layer Normalization) [3]；通過層正規化後的向量會再輸入全連接層，此處同樣設計了殘差連接，因此全連接層的輸出會先和輸入向量相加，接著再經過第二個層正規化，得到此一編碼層輸出的隱藏向量。

轉換器模型中自注意力層的設計使得模型從輸入向量中萃取資訊時，能夠把

doi:10.6342/NTU202302478

注意力集中在那些較為重要的資訊上；殘差連接的設計使得參數的梯度在透過反向傳播演算法傳遞時，除了經過較為複雜的模塊以外，多了一條路徑可以從模塊的輸出直接傳回輸入端，過往的實驗表明殘差連接的設計在較深層的類神經網路模型中可以有效減緩模型底層梯度消失的問題。而層正規化則是對模型輸出的隱藏向量進行正規化，能夠起到穩定模型訓練的效果。

解碼層

解碼層的結構大體而言和編碼層類似，但稍有不同。在圖2.2中，我們可以很清楚地看出解碼層的設計和編碼層相同與相異的地方。解碼層的輸入向量同樣會先經過一個自注意力層，和殘差連接相加後經過層正規化，得到隱藏向量。此時在進入全連接層之前，向量會再經過一個跨注意力層。跨注意力層和自注意力層的差別為，自注意力層的輸入向量皆來自同樣的隱藏向量，如解碼層中的自注意力層，其詢向量、鑰向量和值向量都是由解碼層的輸入向量所計算而來。而解碼層中的跨注意力層，其詢向量是由解碼層的輸入向量計算而來，鑰向量和值向量則是來自編碼層所輸出的隱藏向量，這樣的設計是為了能讓解碼層的輸出除了本身輸入向量的資訊以外，還能夠參考到編碼層的資訊，進而使輸出的內容與輸入更加吻合。

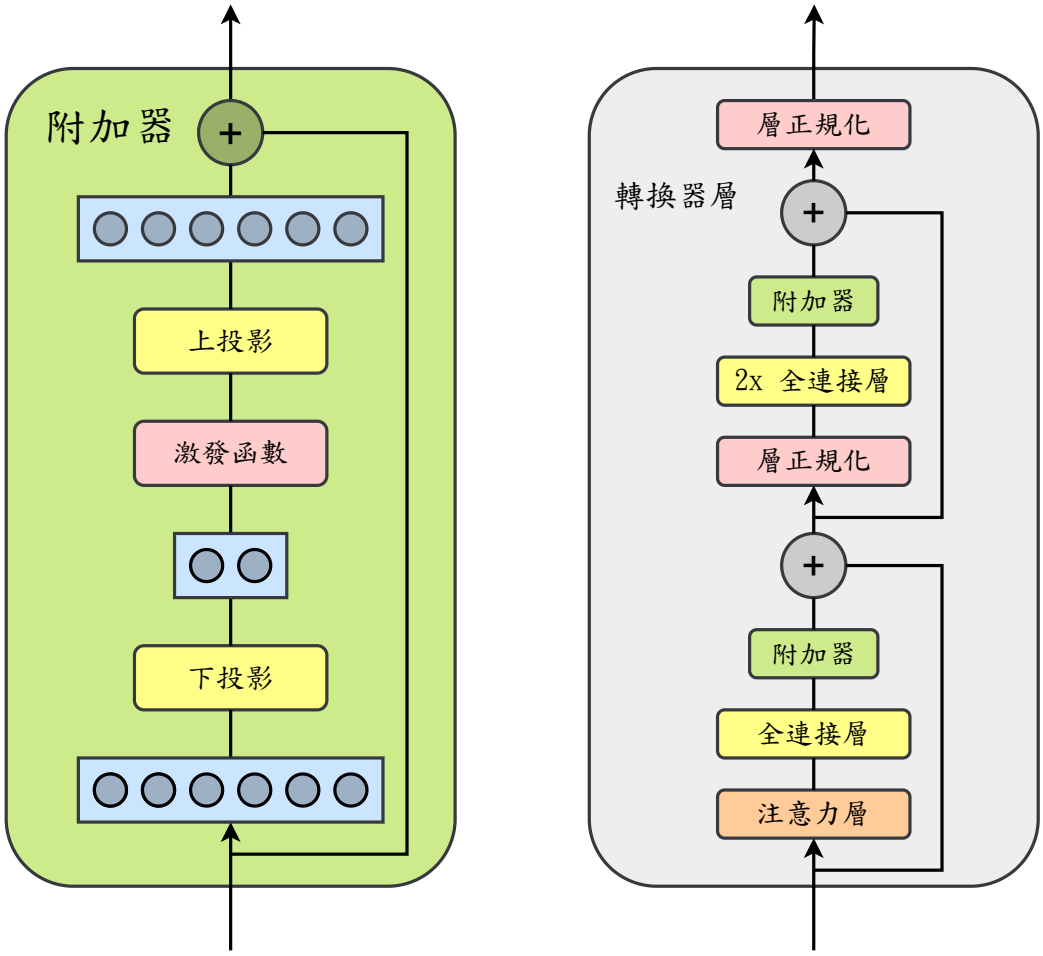
2.1.3 附加器模塊

在深度學習的領域中，一種常見的訓練模型方法為預訓練 (Pre-train) 後微調 (Fine-tune)，讓模型先在大量的標註或未標註資料上進行預訓練，再將預訓練模型在下游任務 (Downstream Task) 上進行微調，通常以這樣方法訓練出的模型在微調時，不但訓練所需要的時間較少，通常也能取得較好的表現。在自然語言處理中，人們常使用的模型是大型的預訓練轉換器模型，如 BERT(Bidirectional Encoder Representations from Transformers) [8]、GPT-2(Generative Pre-trained Transformer 2) [24] 等，在許多下游任務上也取得非常好的效果。但由於轉換器模型的參數量龐大，動輒數百萬以上，針對每一個不同的下游任務都進行微調其實是相對缺乏效率的做法，我們每對一個下游任務進行微調都需要儲存一組參數完全不同的模型參數。而為了解決這個問題，霍氏 (Houlsby) 提出了附加器模塊 (Adapter Module) [12] 的概念，附加器模塊在原始的轉換器模型中加入了少量的參數，並且在下游任務上進行微調時，僅需要微調附加器的參數，原始的轉換器模型參數

doi:10.6342/NTU202302478

可以維持不動。

圖2.4b展示了霍氏的附加器模塊是如何加入到轉換器模型中的，最外層的綠色矩形代表一個轉換器塊 (Transformer Block)，每一個轉換器塊中包含兩個附加器模塊，分別位於轉換器塊的兩個殘差連接之前，而整個模型在微調時可訓練的參數只有圖中的附加器模塊以及層正規化的參數。



(a) 附加器結構示意圖　　(b) 附加器添加於轉換器層示意圖

圖 2.4: 霍氏附加器應用於轉換器模型示意圖

這樣的設計解決了原來微調大型預訓練模型時的問題，原始論文中附加器模塊的參數量僅占原始轉換器模型參數量的 3.6%，因此對於多個下游任務，我們僅需儲存一個原始的轉換器模型，接著對每個下游任務微調各自附加器的參數，每個任務僅需儲存額外的 3.6% 參數，比起原始的微調方法省下了大量的儲存空間，卻能夠獲得幾乎一樣的表現。霍氏所提出的附加器模塊結構如圖2.4a，由兩個全連接層所構成，第一個全連接層會將隱藏向量投影至低維空間中，經過非線性的激發函數後，再透過第二個全連接層投影回原來的維度，最後再和附加

doi:10.6342/NTU202302478

器的輸入向量的殘差連接相加。假設原本轉換器模型的隱藏向量維度為 d_{model}，而我們在附加器模型中將其投影至較低的維度 $d_{adapter}$ 後再投影回 d_{model}，則每個附加器所額外添加的參數量為 $2d_{model}d_{adapter} + d_{model} + d_{adapter}$，只要我們限制 $d_{adapter} \ll d_{model}$，就能很好的限制附加器額外增加的參數量。而殘差連接的設計使得一個參數隨機初始化的附加器模塊會表現得像是一個恆等函數 (Identity Function)，使得附加器模塊的訓練更加穩定。

由於霍氏附加器的成功，後續亦有研究者提出了各式各樣不同結構的附加器模塊，如融合了不同任務資訊的 AdapterFusion [23]、結構比起霍氏附加器更為簡單的 BitFit [39] 等，而本論文也採用了更為簡單的附加器結構 AdapterBias [10]，其結構如圖2.5所示。AdapterBias 的可訓練參數僅包含層正規化、一個可訓練的向量、以及一個全連接層，比起原始的附加器結構參數更少，也更好訓練，但在下游任務上仍然有很好的表現。AdapterBias 的目標是為下游任務訓練一個位移向量 v 加到模型的隱藏向量中，而這個位移向量的權重 α 則透過全連接層 L_α 決定，每個字符所對應的隱藏向量都會計算出一個權重 α，接著這個權重會和向量 v 相乘後加到輸出的隱藏向量中。

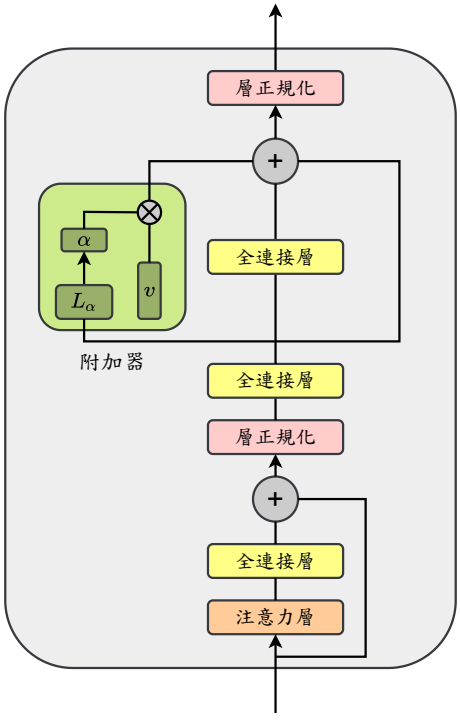


圖 2.5: AdapterBias 結構示意圖

doi:10.6342/NTU202302478

2.2 多任務學習

多任務學習 (Multi-Task Learning)[7] 是一種利用多個任務訓練單一模型的方法，在訓練階段我們會選取多個不同但彼此相關的任務，並使模型同時在這些任務上最佳化其參數。這樣的作法有幾個好處，首先模型能夠從不同的任務汲取資訊，從而學習到某些跨任務的特徵，使模型在不同任務上的一般化能力更強。另外，多任務學習的作法也能有效地節省模型所需要的參數量，一般而言我們需要對每個任務都分別訓練一組獨立的模型參數，但由於多任務學習模型在不同任務上的參數是共享的，因此不需要額外儲存多餘的參數，從而大大的減少了所需的參數量。

在設計多任務學習的模型時，我們也可以針對每個任務添加一些額外的參數，這些參數稱為任務特定 (Task-Specific) 的參數，僅用於他們所負責的任務上，而整個模型除了這些任務特定的參數以外，其餘的參數都是跨任務共享的。這樣的作法保證了模型既能從不同的任務學習到跨任務的知識，也大大減少了模型的參數量，如 [29] 的作法。但亦有研究摒棄了任務特定的參數，使用單一的語言模型去處理所有的任務 [21]。

2.3 元學習

2.3.1 基本原理

深度學習在許多領域的應用上都取得了極大的成功，但訓練深度學習模型時常需要大量的資料，很大程度的限制了深度學習在某些領域的應用，而元學習便應運而生。元學習不同於傳統認知上的讓模型在單一任務上學習，而是讓模型透過多個不同的任務來「學習如何學習」，舉例來說，我們可以讓模型學習一個更好的初始化參數，使其能透過少量訓練樣本就快速地解決一個新的任務 [9]、或是學習更好的最佳化策略 [1, 26]、學習好的模型架構 [40] 等。

doi:10.6342/NTU202302478

2.3.2 模型無關元學習

模型無關元學習 (Model Agnostic Meta Learning, MAML) [9]，是一種與模型種類無關的元學習方法，可以被使用在分類 (Classification)、迴歸 (Regression) 或強化學習 (Reinforcement Learning) 等各類問題上。模型無關元學習的目標是利用多個不同的訓練任務來讓模型學習一個好的初始化參數，使得模型在面對一個新的任務時，能夠僅靠少量的訓練樣本就達到好的表現。模型無關元學習的直觀理解可以參考圖2.6，模型在不同的任務中計算出的梯度方向不一定相同，但會讓模型學習到一個好的初始化參數，從而快速應用到其他下游任務。

模型無關元學習的訓練可以分為兩個階段，元訓練 (Meta Train) 以及元測試 (Meta Test)，元訓練階段就像是傳統監督式學習 (Supervised Learning) 的訓練階段，只不過元學習把「如何學習」本身當成是一項任務，因此在元訓練階段所用來訓練的「樣本」是一項項任務，這也是為什麼我們會把元學習稱為「學習如何學習」。

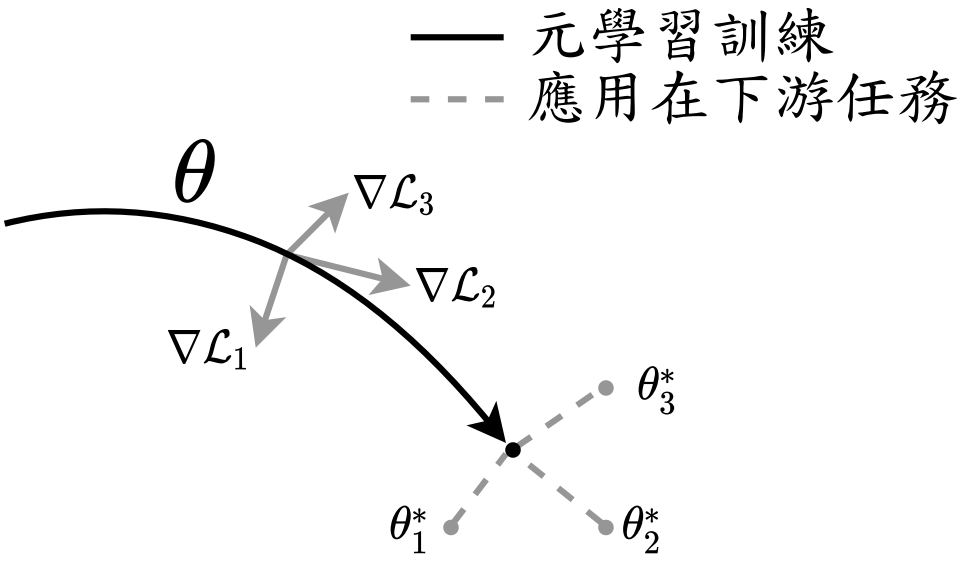


圖 2.6: 元學習概念示意圖

在元訓練階段，考慮一個參數為 θ 模型 f_θ ，在訓練階段時我們會隨機選取一個批次的訓練任務 $\mathcal{T}_i \sim p(\mathcal{T})$，並對每個任務都計算損失 $\mathcal{L}_{\mathcal{T}_i}(f_\theta)$ 以及梯度 $\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$ ，得到每個任務更新過後的模型參數：

$$\theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{2.9}$$

doi:10.6342/NTU202302478

α 是超參數，代表學習率。接著再使用更新過的模型計算在該任務上的損失，最後再透過反向傳播把梯度傳播回原模型，以計算原模型上參數的更新：

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i} \left(f_{\theta_i'} \right) \tag{2.10}$$

β 同樣是超參數，代表學習率。透過這樣的模型訓練方法，當模型遇到一個訓練時沒有見過的任務時，也能夠快速地應用到該任務上。從式 2.10可以看出模型無關元學習的演算法中包含了二階導數的計算，由於計算二階導數較為耗費空間，原論文中也提出對原始演算法的一階近似 (First Order Model Agnostic Meta Learning, FOMAML)，即直接把更新後的模型參數的梯度更新到原模型上：

$$\theta \leftarrow \theta - \beta \nabla_{\theta'} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i} \left(f_{\theta_i'} \right) \tag{2.11}$$

而經過以上的訓練之後，模型在元測試階段會遇到一個或多個元訓練階段沒有看過的新任務，接著利用新任務所提供的少量訓練樣本對模型進行訓練並更新模型參數，接著才會在該任務上進行測試或驗證。

doi:10.6342/NTU202302478

第三章　轉換器模型加上附加器之訓練框架

3.1　任務簡介

3.1.1　資料集

GLUE 基準

我們使用的其中一個資料集為由紐約大學、華盛頓大學、DeepMind 等機構所推出的 GLUE 基準 (General Language Understanding Evaluation)[34]，其中包含了九項自然語言理解任務，分別是 CoLA、SST-2、MRPC、STS-B、QQP、MNLI、QNLI、RTE 以及 WNLI。其中 CoLA 和 SST-2 屬於單個句子的分類任務，餘下的則為句子對的分類任務。這些任務包含了問答、情感分析、文本蘊含等各種不同的任務，目的是讓模型不只有學會某項特定任務中輸入和輸出之間的關係，而是學習人類語言的共同特性。關於這些任務的簡介如下：

- CoLA：句子是否合乎文法，二分類任務
- SST-2：判斷電影評論句子中蘊含的情感為正面或負面情感，二分類任務
- MRPC：句子來源為新聞，標籤為兩個句子的語意是否一樣，二分類任務
- STS-B：判斷兩個句子的相似性，標籤為句子對的相似性評分，從 0 到 5，本質上是一個迴歸任務
- QQP：資料取自問答網站 Quora[1]，判斷兩個問題句子在語意上是否一致，二分類任務

[1] https://www.quora.com/

doi:10.6342/NTU202302478

- MNLI：給定前提 (premise) 和假設 (hypothesis) 語句，預測前提語句和假設語句的關係為蘊含 (entailment)、矛盾 (contradiction) 或中立 (neutral)，三分類任務
- QNLI：資料包含一段來自維基百科的文字和一個問題，標籤為該段落是否蘊含問題的答案，二分類任務
- RTE：判斷兩個句子之間的關係是蘊含還是矛盾，二分類任務
- WNLI：給定兩個句子，第一個句子中會包含一個代名詞，而第二個句子則會將代名詞替換為第一個句子中的某個對象，該任務的標籤為代名詞所替換的對象正確與否，屬於二分類任務

GLUE 基準是自然語言處理中一個相當具指標性的資料集，其任務的多樣性使得該資料集相當適合用於衡量模型在跨任務的一般化能力。

CrossFit

另一個我們使用的資料集為 CrossFit 資料集 [38]，該資料集是由南加州大學的學者所發表，旨在提出一個自然語言處理領域的少樣本學習基準。CrossFit 收集了多達 160 個不同的自然語言處理任務，其中包含了問答 (Question Answering)、分類 (Classification)、條件生成 (Conditional Generation)、迴歸 (Regression) 等各種不同類型的任務，並把所有的任務都統一整理成文字對文字 (Text-to-text) 的格式，此種作法參考自谷哥團隊所提出的 T5 模型 [25]。舉例來說，文本蘊含任務含有一個前提 (Premise) 和假設 (Hypothesis)，兩者都是一段文字敘述，如果從前提的文字敘述中可以推出假設的內容，則這個樣本的標籤是蘊含 (Entailment)；如果假設的內容和前提違背，則標籤是矛盾 (Contradiction)；如果前提的內容無法用於判斷假設的真偽，則標籤是中立 (Neutral)。傳統的作法 [8, 19] 通常是將前提和假設加上起始和分隔句子的特殊字符後當作輸入序列，標籤則是蘊含、矛盾和中立三者其一，對應到數字 0,1,2。而文字對文字的格式則會把輸入改成以下的格式：「前提：＜前提＞假設：＜假設＞」，其中＜前提＞和＜假設＞的部分填入原始資料中的內容，輸出的部分則同樣是文字，蘊含、矛盾、中立其中之一。注意此處的標籤是文字的形式，和傳統上轉換為索引的做法不同。表3.1展示了文本蘊含任務上的一個範例，可以看到不管是樣本還是標籤，其格式都略有不同。

doi:10.6342/NTU202302478

前提	有一隻小狗在雪地裡玩飛盤。	
假設	有隻動物在寒冷的天氣裡玩塑膠玩具。	
	原始資料	文字對文字格式
樣本	[CLS] 有一隻小狗在雪地裡玩飛盤。[SEP] 有隻動物在寒冷的天氣裡玩塑膠玩具。	前提：有一隻小狗在雪地裡玩飛盤。[SEP] 假設：有隻動物在寒冷的天氣裡玩塑膠玩具。
標籤	0(蘊含)	蘊含

表 3.1: 文本蘊含任務轉換為文字對文字格式之範例

因為採用了統一的文字對文字格式，不管是什麼樣的任務都可以用同樣的語言模型去生成答案，進而確保模型能在所有的任務上以統一的方法進行訓練。由於是著重於在少樣本學習在跨任務的一般化能力，CrossFit 所收集的每個任務樣本數都不多，以分類問題來說，每個類別只有 16 筆樣本。

由於此處「任務」一詞常具有多重意涵，既可以指涉任務的不同種類，如：問答、文本分類、閱讀理解，亦可以代表不同資料集上同樣種類的任務，如：SciTail[15]、QNLI[34]、CommitmentBank[33] 都是屬於自然語言推理 (NLI, Natural Language Inference) 的任務，但資料來源分別來自不同的資料集。因此這裡我們給任務一詞較為精確的定義，每個任務 T 都是一個 $(\mathcal{D}_{train}, \mathcal{D}_{dev}, \mathcal{D}_{test})$ 的元組，下標 $train$ 代表訓練集、下標 dev 代表驗證集、下標 $test$ 代表測試集。每個集合 $\mathcal{D}$ 都是由一系列的已標註資料 $\{(x_i, y_i)\}$ 所構成，其中下標 i 代表第幾筆資料，x_i 代表該筆資料的輸入，y_i 代表對應的正確輸出。在少樣本學習的情境下，$\mathcal{D}_{train}$ 和 $\mathcal{D}_{dev}$ 的大小都將被限制在每個類別 16 筆樣本，而 $\mathcal{D}_{test}$ 的大小則沒有限制。

3.2 模型架構

本論文中使用了兩種基於轉換器的語言模型，分別為谷哥所提出的 BERT 模型以及由 Meta 公司所提出的 BART 模型。

3.2.1 BERT

BERT[8] 的英文全名為"Bidirectional Encoder Representations from Transformers"，翻譯成中文的意思是「基於轉換器的雙向編碼器表示」，正如其名所示，BERT 是被設計用於產生未標註文本的雙向隱藏表示向量。由於 BERT 在大量的

doi:10.6342/NTU202302478

資料上進行了無監督式學習的預訓練，能夠有效地擷取和濃縮整個輸入序列的資訊，我們在使用 BERT 進行下游任務訓練時只要添加一個額外的輸出層，將隱藏向量映射至輸出的維度即可。

將 BERT 模型應用於下游任務上的作法如圖3.1所示，圖中粉紅色的矩形是一個預訓練的 BERT 模型，任務的輸入則是一個句子或是句子對。我們會在輸入句子前方加上一個特殊字符 (Token)[CLS]，如果輸入有兩個句子的話，在第一個句子後方也加入一個特殊字符 [SEP]，以讓模型區分不同的句子。在經過 BERT 模型之後，每個字符都會被編碼成一個隱藏表示向量，而 [CLS] 這個字符的位置對應到的向量則代表了整個輸入文本被編碼而成的資訊。這樣的做法來自於預訓練時所採用的 NSP(Next Sentence Prediction) 任務，NSP 任務的內容為分辨一個句子對中的兩個輸入句子在原始資料中是否真的為相鄰的句子，由於預訓練時模型使用 [CLS] 字符所對應的隱藏向量來進行下一句的預測，因此能讓模型隱性地將輸入句子的全局資訊濃縮在 [CLS] 字符對應的隱藏向量中。

圖3.1中綠色的矩形代表了 [CLS] 字符的隱藏向量，在經過 BERT 的編碼後，該隱藏向量會再經過一個輸出層，通常由一個線性層和軟性最大化層所構成，接著得到輸出不同選項的機率。

3.2.2 BART

BART[19] 是由 Meta AI(發表論文時名為 Facebook AI) 的團隊所發表的序列對序列轉換器模型，其架構包含了一個雙向編碼器 (Bidirectional Encoder) 和一個自迴歸解碼器 (Autoregressive Decoder)，可用於進行文本生成，也可以用於問答、語意理解等任務上。

BART 的模型架構如圖3.2所示，圖中粉紅色的矩形為雙向編碼器，負責將輸入序列的資訊編碼為隱藏向量，而藍色的矩形則是解碼器，負責產生最後的輸出。圖3.2a展示了 BART 如何應用在進行文本生成類的任務，如條件文本生成、文本大綱生成等，自迴歸解碼器能夠同時參考輸入的資訊以及已經解碼的輸出資訊，每一輪解碼出的輸出字符會成為下一輪解碼器的輸入，一直重複該過程直到解碼器輸出終止的特殊字符。

在 BART 的原論文中處理分類問題的做法和文本生成略有不同，如圖3.2b所示，編碼器和解碼器的輸入是同樣的序列，而最後一個字符在解碼器最後一層的

doi:10.6342/NTU202302478

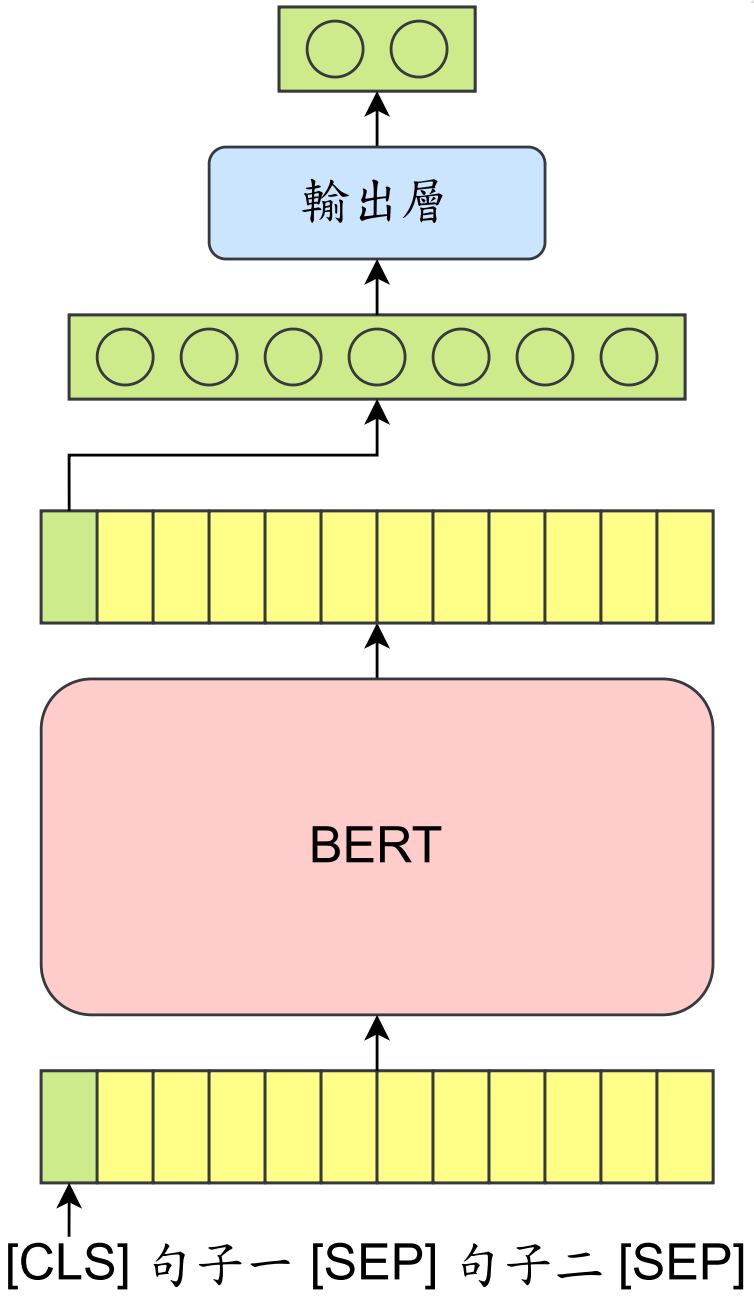
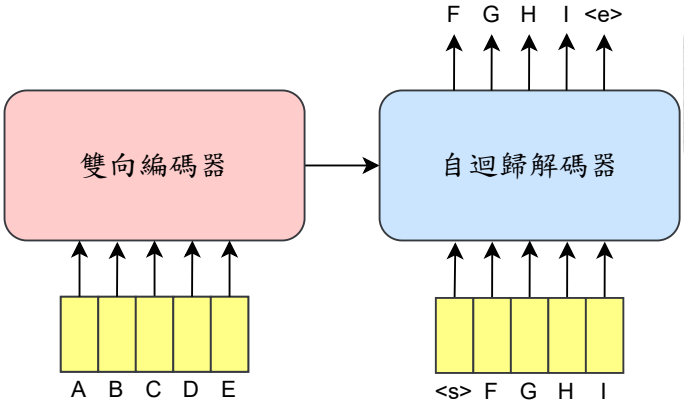


圖 3.1: BERT 應用於下游任務示意圖

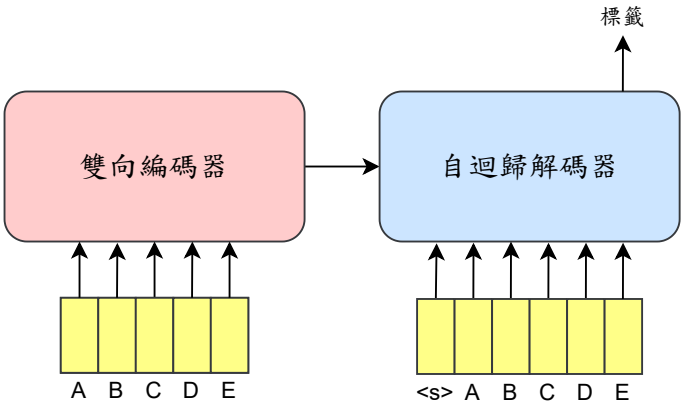
隱藏向量則會被用來代表整個輸入序列濃縮的資訊，並用來進行分類。

在本論文中，我們參考 CrossFit[38] 原論文中的作法，將所有的任務都轉換為文字對文字的格式，並讓 BART 模型以文本生成的方式直接生成任務的回答，和 BART 原始論文中的做法不同。

doi:10.6342/NTU202302478



(a) BART 模型用於文本生成



(b) BART 模型用於分類問題

圖 3.2: BART 模型應用於下游任務

3.3 模型訓練框架

3.3.1 簡介

由於傳統使用附加器模型加上預訓練轉換器模型的做法在少樣本學習的情境下表現較差，因此我們提出在進行下游微調附加器模型之前，加入一個額外的「引導」階段，來讓整個模型學會如何更好的進行下游微調。其基本概念可參考圖3.3，左邊是傳統的輕量化微調，我們直接將預訓練語言模型加上一小塊下游可訓練參數，並在下游任務上只微調該部分參數。右邊則是本研究的方法，我們在進行下游微調之前會加入一個我們稱為「引導」的階段，經過該階段的訓練後，使得預訓練語言模型和下游可訓練參數能夠契合得更好 (如示意圖右下，預訓練語言模型的缺口更貼合下游可訓練參數)。

在進行跨任務一般化模型的訓練時，整個模型的訓練可以被分為兩個階

doi:10.6342/NTU202302478

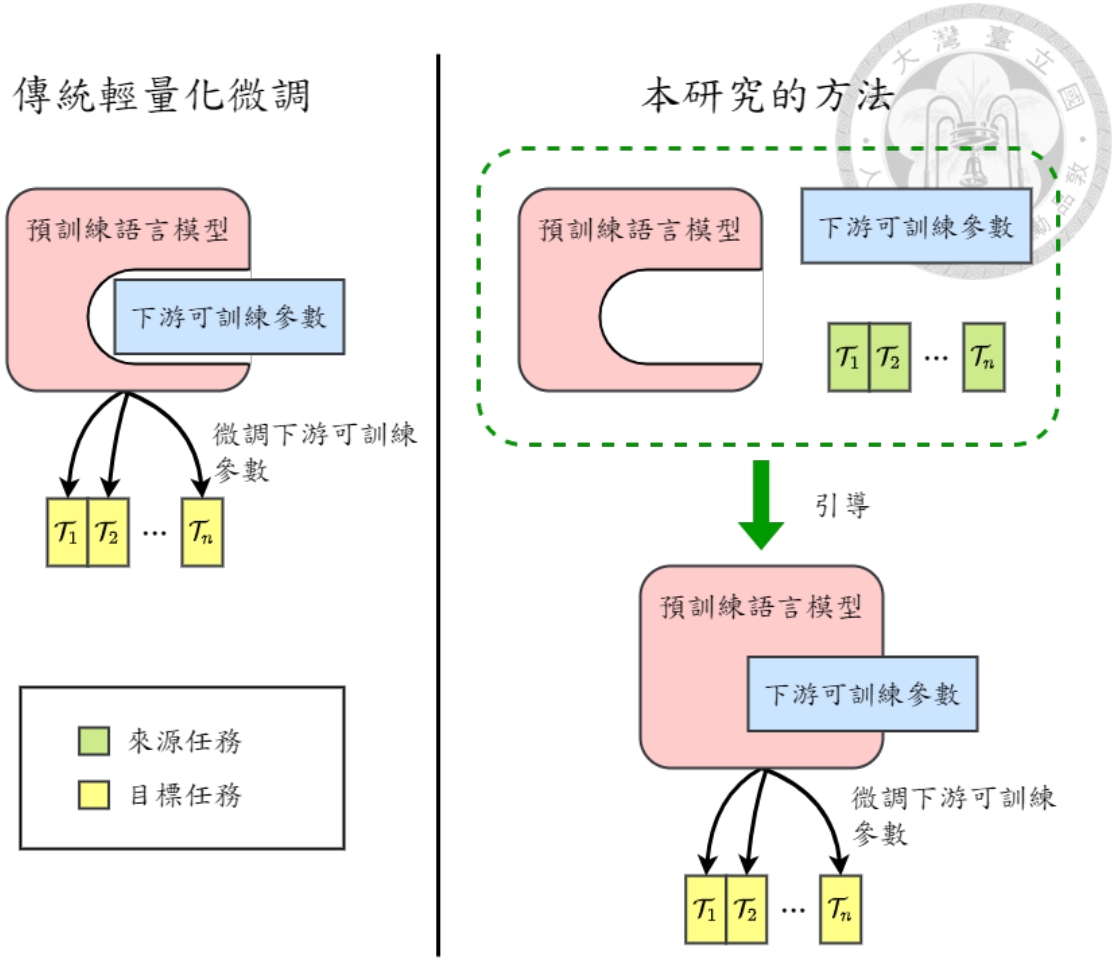


圖 3.3: 預訓練語言模型引導概念圖

段，第一個階段是上游學習 (Upstream Learning) 階段，第二個階段則是下游微調 (Downstream Finetuning) 階段。在說明兩個階段的訓練方法前，我們必須先正式地定義跨任務學習的訓練、驗證以及任務和任務集。同小節3.1.1所述，每個任務 T 都是一個 $(\mathcal{D}_{train}, \mathcal{D}_{dev}, \mathcal{D}_{test})$ 的元組，$T \in \mathcal{T}$，$\mathcal{T}$ 為所有任務的集合。我們把 $\mathcal{T}$ 中的任務分為三個互不重疊的部分，分別是 $\mathcal{T}_{train}$、$\mathcal{T}_{dev}$ 和 $\mathcal{T}_{test}$，其中 $\mathcal{T}_{train}$ 和 $\mathcal{T}_{dev}$ 用於第一階段的上游學習，而 $\mathcal{T}_{test}$ 則用於第二階段的下游微調。

在上游學習的階段，我們將會從任務集 $\mathcal{T}_{train}$ 中隨機採樣任務進行訓練，並更新模型的參數，而 $\mathcal{T}_{dev}$ 中的任務僅用於參考模型訓練的效果，不能直接用於更新模型的參數。此階段我們將採用多任務學習以及元學習的方法訓練模型，使模型獲得一個較好的初始化參數，進而使模型在下游微調的階段中能有較好的表現。

在下游微調的階段，我們會使用在上游學習中得到的模型作為初始化的參數，並將模型對所有在 $\mathcal{T}_{test}$ 中的任務進行微調。對每個任務 $T = (\mathcal{D}_{train}, \mathcal{D}_{dev}, \mathcal{D}_{test}) \in \mathcal{T}_{test}$，我們僅使用 $\mathcal{D}_{train}$ 中的樣本來訓練模型和更新參

doi:10.6342/NTU202302478

數，$\mathcal{D}_{dev}$ 中的樣本並不會直接被用來訓練模型的參數，而是用來計算正確率，以挑選在微調過程中表現較好的參數。模型在經過少樣本的訓練之後，再以 $\mathcal{D}_{test}$ 中的樣本來衡量模型的表現。

為了要找出何種訓練方法以及訓練的模型參數能帶給模型表現最大的提升，我們將整個訓練分為三個部分分別進行調整：訓練方法、上游可訓練參數、下游可訓練參數。其中訓練方法代表的是我們在上游學習階段使用何種演算法來訓練模型的參數，在本論文中我們實驗了兩種方法，分別是多任務學習 (Multi-task Learning) 以及模型無關元學習 (MAML)。多任務學習是一種傳統但有效的方法，其做法為將多個訓練任務的損失加起來或做平均後再反向傳播計算梯度，藉此來讓模型學習跨任務的特徵。模型無關元學習則是一種可以利用多個訓練任務來進行少樣本學習的演算法，相當適用於本論文所欲探討的情境。而關於多任務學習和模型無關元學習和附加器結合使用的細節，將會在下一個小節中詳述。

上游可訓練參數代表的則是在上游學習階段時所可以訓練的參數，我們在上游學習階段並不會訓練整個模型中所有的參數，而是將參數分為不同部分，如分成「轉換器、附加器」這兩個部分，假設我們在上游學習階段只會更新轉換器的參數，則上游可訓練參數就只有轉換器。假設在上游學習階段我們選擇訓練整個模型，則轉換器和附加器都屬於上游可訓練參數。

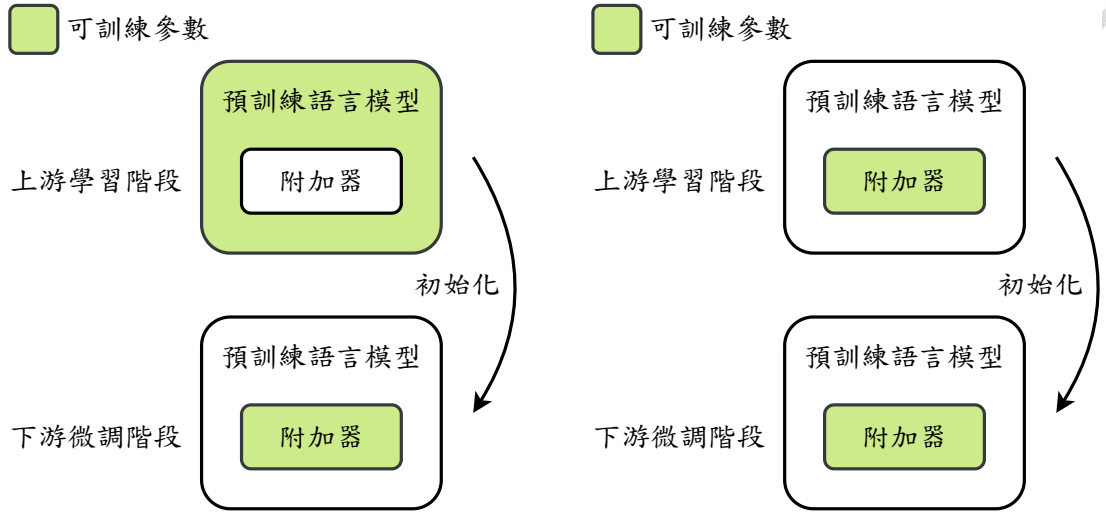
下游可訓練參數則是代表在下游學習階段所可以訓練的參數，由於本論文聚焦於減少下游微調時的計算資源，故將下游學習階段的可訓練參數限制在附加器的部分。

3.3.2 上游訓練階段

在本論文所提出的框架中，在上游階段和下游階段所訓練的參數不一定相同，我們可以藉由在上游階段訓練不同部分的參數，來找出上游訓練中真正重要的參數是什麼。舉例來說，一個預訓練的語言模型加上附加器之後，我們可以選擇在上游階段訓練「預訓練語言模型、附加器」，也就是訓練所有的參數，也可以選擇只訓練「預訓練語言模型」的部分，或者也可以訓練「附加器」的部分，讓預訓練語言模型的參數保持固定。以圖3.4為例，圖3.4a代表上游可訓練參數為轉換器，下游可訓練參數為附加器，而圖3.4b則代表上游可訓練參數為附加器，下游可訓練參數也是附加器。

doi:10.6342/NTU202302478

在上游學習階段裡的被訓練過的上游可訓練參數，將會被用來初始化接下來下游微調階段所使用的模型參數，而不屬於上游可訓練參數的部分，則會保持固定或是在下游微調階段前隨機初始化。



(a) 上游可訓練參數：轉換器
下游可訓練參數：附加器

(b) 上游可訓練參數：附加器
下游可訓練參數：附加器

圖 3.4: 不同上下游可訓練參數的組合比較

元學習

在上游學習階段，我們採用的其中一種演算法為模型無關元學習 (Model Agnostic Meta Learning, MAML)，模型的訓練分成內迴圈 (Inner Loop) 和外迴圈 (Outer Loop)，其中內迴圈和外迴圈的訓練參數不一定相同。在外迴圈中，我們訓練的是上游可訓練參數，而在內迴圈中，我們則是訓練下游可訓練參數，如圖3.5所示。首先在外迴圈中先從訓練任務集 $\mathcal{T}_{train}$ 中採樣一批訓練任務 $T_i \in \mathcal{T}_{train}$，任務的數量 N 取決於批次的大小 (Batch Size)，進入內迴圈之前，整個模型的參數 ψ 會先被複製 N 份用來進行內迴圈的訓練，注意在內迴圈中我們只會更新下游可訓練參數 ψ_d，也就是附加器的參數，其他部分的參數將會保持固定。這樣做的原因是，模型無關元學習的內迴圈訓練方式必須要和下游微調的訓練方式保持一致。在內迴圈裡，我們使用訓練任務 T_i 中的 $\mathcal{D}_{train}$ 作為支持集 (Support Set) 來訓練模型，接著計算更新後的模型 ψ_i' 在詢問集 (Query Set) 上的損失，最後透過反向傳播計算在最初模型中的上游可訓練參數 ψ_u 的梯度。於是，上游可訓練參數 ψ_u 會在外迴圈中被更新，而下游可訓練參數 ψ_d 只會在內迴圈中被更新，因此在上游訓練階段結束後，會被訓練並且用來初始化下游微調階段模型的參數只有上游可訓練參數而已。

doi:10.6342/NTU202302478

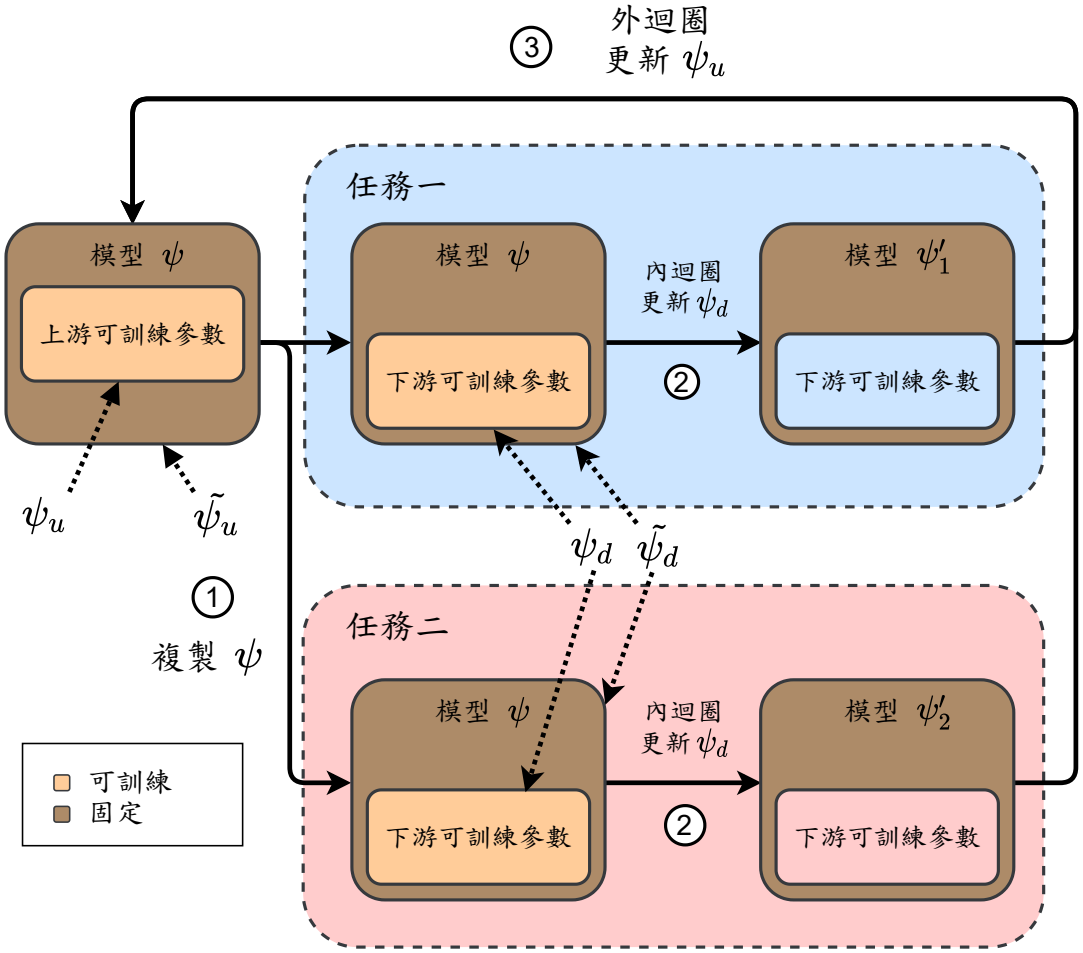


圖 3.5: 使用模型無關元學習作為上游訓練階段的訓練方法示意圖。(1) 複製整個模型的參數 ψ 作為內迴圈的初始化參數。(2) 在內迴圈中模型的參數被分為兩個部分，ψ_d 為下游可訓練參數，其餘固定的參數為 $\tilde{\psi}_d$，內迴圈中我們僅更新 ψ_d。(3) 在不同訓練任務的內迴圈中得到更新後的模型參數 $\psi'_1, \psi'_2, \ldots$，並用這些模型參數在各自任務上的詢問集計算損失，接著計算該損失在最初模型的上游可訓練參數 ψ_u 上的梯度，並在外迴圈更新 ψ_u。

doi:10.6342/NTU202302478

多任務學習

多任務學習是另一個我們在上游學習階段使用的演算法，作法相較於元學習較為簡單直接，訓練過程見圖 3.6，我們從訓練任務集 $\mathcal{T}_{train}$ 採樣一批訓練任務後，計算模型在訓練任務的 $\mathcal{D}_{train}$ 和 $\mathcal{D}_{val}$ 中的損失，接著把每個任務的損失平均後計算上游可訓練參數的梯度並更新。

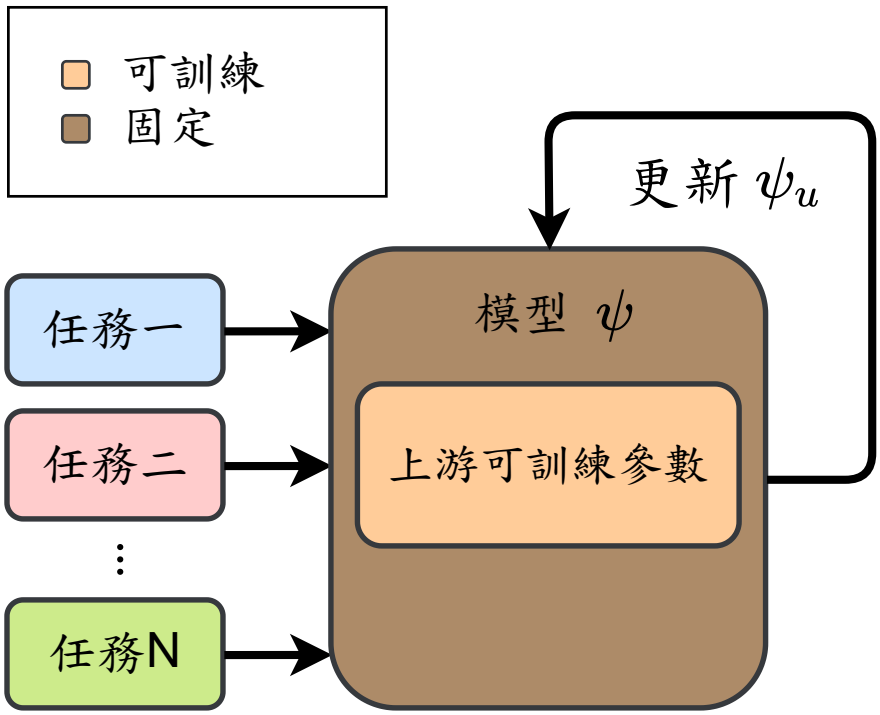


圖 3.6: 使用多任務學習作為上游訓練階段的訓練方法示意圖。注意如果下游可訓練參數並未被包含在上游可訓練參數內，則在上游訓練過程中皆保持固定。

3.3.3 下游微調階段

為了節省下游微調時所使用的運算資源和模型儲存所需的空間，進行下游微調時我們會讓預訓練語言模型的參數保持固定，僅微調附加器的參數。換句話說，下游可訓練參數僅包含附加器。

doi:10.6342/NTU202302478

3.3.4 特定的可訓練參數組合

訓練初始化參數

假設我們固定預訓練語言模型的參數，使得上游可訓練參數只有附加器，而下游可訓練參數也同樣是附加器的情況下，這個組合可以被視為是單純在訓練附加器的初始化參數。

元附加器

為了減少過往微調整個模型的龐大計算開銷，班氏 (Trapit Bansal) 提出了元附加器 (Meta-Adapters)[5] 的做法來降低微調模型時所需要訓練的參數量。元附加器的結構可以參考圖 (待補)，以霍氏附加器 [12] 為基礎，在原本轉換器插入附加器的位置額外加入了兩個附加器，這兩個新的附加器把原本的附加器夾在中間，被稱為元附加器。[5] 使用模型無關元學習來訓練模型，其中外迴圈同時訓練元附加器和附加器，而內迴圈則只訓練附加器。如果我們把元附加器、附加器都視為上游可訓練參數，而附加器同時也是下游可訓練參數，其實元附加器的訓練方法亦可視為本論文所提出的框架的其中一個組合。

3.4 實驗設置

3.4.1 超參數

在使用元學習進行訓練時，外迴圈的學習率為 1×10^{-5}，內迴圈的學習率為 5×10^{-4}，採用的優化器為 Adam[16]。

BERT

在 GLUE 資料集上我們使用 BERT-base 模型進行初步的實驗驗證該訓練方法的有效性，接著在 CrossFit 資料集上進行完整的實驗及分析。在 GLUE 資料集上，少樣本訓練的樣本數為每個任務每類別 20 筆樣本，訓練 2000 個週期 (Epoch)，在內迴圈中更新參數的次數為 10 次。

doi:10.6342/NTU202302478

BART

在 CrossFit 資料集上我們使用 Meta 公司推出的 BART-base 模型，每個任務每類別 16 筆樣本，訓練 80 個週期，內迴圈更新參數的次數為 1 次。

3.4.2 附加器

霍氏附加器

霍氏 (Houlsby) 所提出的附加器 [12]，由兩個線性層和一層非線性的激勵函數所組成，其中的線性層具有瓶頸 (Bottleneck) 的結構，也就是利用線性層將隱藏向量的維度縮小至原來的二分之一或更小，在下一個線性層再把維度投影回原來的大小，這樣的設計可以大幅減少模型的參數量。在本論文中所採用的瓶頸隱藏維度大小皆為 8 。

AdapterBias

在霍氏附加器被提出之後，有許多的研究致力於探討如何進一步減少附加器所使用的參數量，AdapterBias [10] 就是其中之一。AdapterBias 在每個轉換器模塊中的可訓練參數僅有一個線性層和一個向量，與瓶頸維度為 8 的霍氏附加器相比，AdapterBias 僅需要霍氏附加器的約 15% 參數，微調時卻能達到差不多或是更好的表現。

元附加器

班氏 (Trapit Bansal) 提出了元附加器，在原本加入附加器的位置額外插入兩個附加器，並在模型無關元學習的內外迴圈分別訓練。本論文中元附加器的實作皆使用霍氏所提出的附加器結構 [12]，元附加器和附加器的瓶頸維度皆為 8 。

3.4.3 評量標準

由於本論文所使用的 CrossFit 資料集 [38] 包含了多達 160 種不一樣的自然語言處理任務，其中包含了迴歸、問答、閱讀理解、分類、條件生成等各種不同

doi:10.6342/NTU202302478

的任務，而不同的任務類型又會有各自不同的評量標準，如分類 F1 分數、準確率、問答 F1 分數、EM 分數 (Exact Match Score)、Rouge-L 分數、馬修斯相關係數 (Matthew Correlation)、皮爾森相關係數 (Pearson Correlation) 等，每項評量標準的數字分布和意義都不相同，因此無法簡單的以平均值去比較不同模型的好壞。

在 CrossFit 的原始論文中，他們提出了 ARG(Average Relative Gain) 分數用來衡量在經過上游訓練以後的模型與原始模型相比的進步幅度。ARG 分數的算法如下，假設我們使用的資料集 $\mathcal{T}_{test} = \{T_A, T_b\}$，其中 T_A 和 T_B 代表不同的任務，經過了上游訓練後的模型在任務 T_A 上的少樣本 F1 分數從 50% 進步到了 70%，也就是有了 $\frac{70\%-50\%}{50\%} = 40\%$ 的相對進步幅度 (RG, Relative Gain)；在 T_B 上的準確率從 40% 退步到 30%，也就是有了 $\frac{30\%-40\%}{40\%} = -25\%$ 的相對進步幅度，注意此處的相對進步幅度可以是正的也可以是負的。最後的 ARG 分數就是把所有任務的相對進步幅度進行平均，也就是 $\frac{40\%+(-20\%)}{2} = 7.5\%$。

比起原始分數，ARG 分數更能夠反映出在所有任務上「整體的」表現好壞，無論這些任務所使用的評量標準和內容是什麼，ARG 都能給出一個在這些任務上整體而言進步的幅度大小。但除了 ARG 分數以外，我們也會針對個別任務的原始分數進行額外的分析。

而除了 ARG 分數以外，我們也考慮模型在不同任務上 RG 分數的標準差，也就是所謂的 RGSTD(Relative Gain Standard Deviation)，RGSTD 的數值反映了模型在不同任務上表現的平均程度，可以用來衡量模型跨任務一般化的能力。

doi:10.6342/NTU202302478

第四章　在 GLUE 上初步實驗之結果

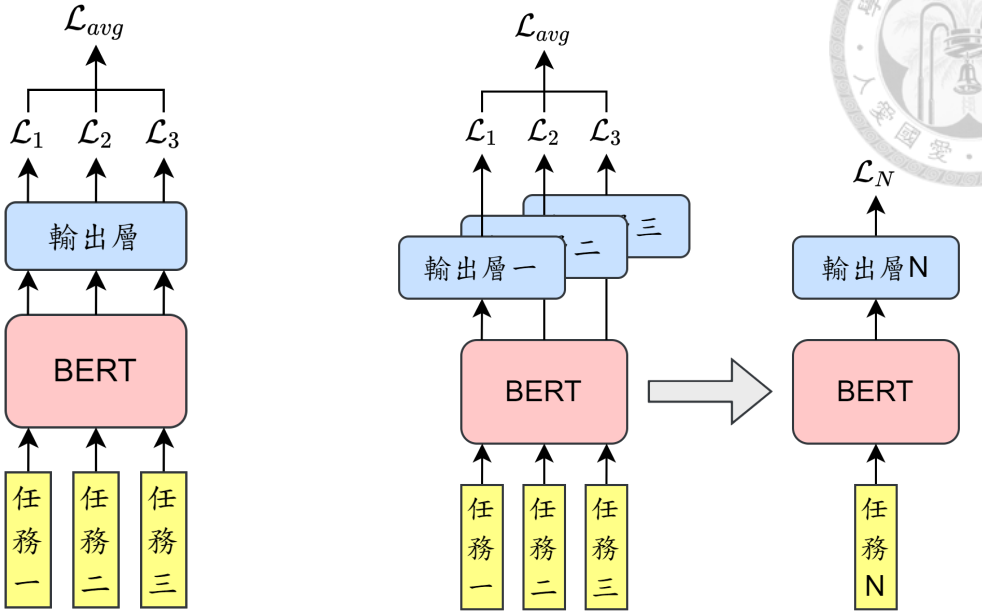
4.1　模型訓練方法

4.1.1　多任務學習

BERT 應用在下游分類任務時，通常輸出層的部分我們會使用一個簡單的全連接層，輸入維度和 BERT 模型的隱藏維度一致，輸出維度則會跟下游分類任務的類別數相等。在一般情況下，如果每個任務的類別數都相等，那我們可以直接使用同樣的 BERT 模型來處理所有的任務，如圖4.1a所示，我們從訓練任務中採樣一批任務，並讓模型分別計算這些任務的損失 $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, ...$，接著將這些損失平均後得到 $\mathcal{L}_{avg}$ 並使用反向傳播更新模型參數。

但如果多任務學習的訓練任務中包含了所需輸出維度不同的任務，如二元分類任務與三元分類任務，前者所需的輸出維度是二，而後者則是三，我們就沒有辦法使用前述的方法直接訓練模型。因此此處我們引入任務特定參數 (Task-specific Parameter) 的概念，對每個不同的訓練任務我們都給模型一組獨立的輸出層參數，但所有的任務都共享同一個 BERT 模型的參數，如圖4.1b所示。在上游學習的時候，模型的運作方式如圖4.1b左邊所示，每個任務都使用各自的輸出層進行運算，最後將得到的損失平均後再反向傳播更新參數，注意這裡雖然不同任務的損失 $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, ...$ 是使用不同的輸出層參數進行計算，但由於 BERT 模型的參數是共享的，因此每個任務都會對 BERT 模型的參數訓練產生影響。圖4.1b右邊則是在經過上游訓練之後的 BERT 模型，當我們進行下游微調時，一樣會給新的任務一組新的輸出層參數，並在新的輸出層上進行微調。

doi:10.6342/NTU202302478



(a) BERT 模型用於相同輸出維度的多任務學習

(b) BERT 模型用於不同輸出維度的多任務學習

圖 4.1: BERT 模型的多任務學習

4.1.2 模型無關元學習

模型無關元學習的訓練過程分為外迴圈和內迴圈，而模型在外迴圈中會複製多組參數以初始化不同任務進行內迴圈的模型參數，但該訓練流程同樣假設了單一模型可以同時處理所有的任務，因此對於不同輸出維度的任務，我們依然要稍微修改 BERT 模型訓練的流程。修改過後的訓練流程如圖4.2所示，可以看到模型的參數分成兩個部分，粉紅色的是 BERT 的參數，黃色的則是輸出層的參數。對每個訓練任務，在內迴圈時會複製一份 BERT 的參數，並新增一組輸出層的參數用於該任務，接著便利用支持集的資料更新模型的輸出層。更新完輸出層的參數後，再利用詢問集中的資料計算損失，最後再把梯度傳播回 BERT 模型上。對於每個訓練任務來說，新增的輸出層參數都是隨機初始化的，因此就算在下游微調時面對新的任務，也可以直接給模型新增一組新的隨機初始化的輸出層參數並進行訓練。

doi:10.6342/NTU202302478

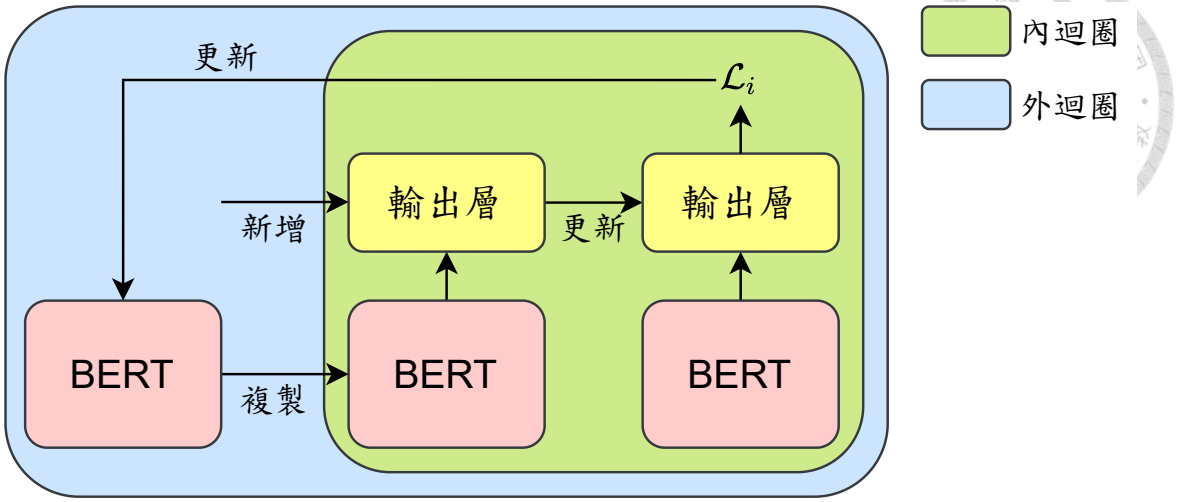


圖 4.2: BERT 模型無關元學習流程示意圖

4.2 實驗結果

4.2.1 MNLI 做為測試任務的表現

首先我們保留 GLUE 資料集中的 MNLI 任務做為測試用，使用其他的 6 個任務 (CoLA, MRPC, QNLI, QQP, RTE, SST2) 進行訓練，並觀察在 MNLI 上少樣本學習的表現。表4.1列出了不同組合的表現：

方法	附加器	準確率
預訓練	無	0.3354
元學習	無	0.4531
元學習	有	**0.4952**
多任務學習	無	0.3793

表 4.1: BERT 模型在 MNLI 上少樣本學習的表現

其中預訓練指的是預訓練的 BERT 模型未經過上游訓練，直接在 MNLI 上進行少樣本學習。MNLI 是一個三元分類問題，也就是說在完全隨機猜測的情況下準確率約為 33% 左右，可以發現預訓練模型在少樣本學習的情況下表現並不好，其準確率幾乎和隨機猜測無異，顯示傳統的預訓練後微調輸出層之訓練流程在少樣本學習的情境下較為受限。而透過多任務學習則能夠帶來些微的進步，但效果不及元學習。

在模型的上游訓練過程中，我們也發現元學習的訓練過程極端不穩定，有的時候在同樣的超參數組合下，單純更換隨機種子 (Random Seed，影響訓練過程隨機因素的一個數字) 就會讓模型的損失無法收斂，過往也有研究 [2] 指出元學習的

doi:10.6342/NTU202302478

訓練對於超參數的設置，甚至是隨機種子都十分敏感。我們經過嘗試後發現，比起過去訓練 BERT 模型常用的學習率 1×10^{-4}，改用較小的學習率能夠使訓練更加穩定，但同時會讓損失的收斂較慢，進而需要訓練更多週期。而除了外迴圈的學習率以外，內迴圈的學習率雖然沒有對訓練造成那麼大的影響，卻也會使最終的表現有些微差異，內迴圈的學習率比外迴圈大的情況下能夠有較好的表現，在後續實驗中我們統一使用 1×10^{-5} 作為外迴圈的學習率，5×10^{-4} 作為內迴圈的學習率，少樣本學習的樣本數為 20，元學習內迴圈的參數更新次數為 10 次。

4.2.2 移除訓練任務的影響

在保持超參數和測試任務 (MNLI) 不變的情況下，我們嘗試從原來的 6 個訓練任務中移除其中一個訓練任務，並觀察模型在測試任務上的表現變化，結果如表4.2所示。

移除訓練任務	準確率	準確率變化
無	0.4952	0.0
QNLI	0.4955	0.0003
CoLA	0.4928	-0.0024
SST2	0.4785	-0.0167
QQP	0.4747	-0.0203
MRPC	0.4691	-0.0259
RTE	0.4581	-0.0369

表 4.2: 移除某個訓練任務後模型在 MNLI 上的表現

可以觀察到在移除 QNLI 或 CoLA 的情況下，對模型在 MNLI 上的表現幾乎沒有影響，而移除 RTE、MRPC 等任務則會使模型的準確率較明顯的退步，但也沒有造成非常嚴重的影響。實驗結果表明多個訓練任務的上游學習確實對模型在 MNLI 的少樣本學習表現有所助益，但並非每個任務都會產生幫助，如前述的 QNLI 和 CoLA 就和模型的表現關聯較小，而訓練任務是否會讓對模型下游的表現有所助益則和上游任務與下游任務的選擇有關。

4.2.3 其他任務作為下游測試任務的表現

接下來我們嘗試在 GLUE 資料集中的其他任務上進行相同的實驗，以驗證該方法的普適性，結果如表4.3所示，表中所列之結果皆使用 GLUE 資料集的任務進行訓練，惟將測試用的任務從訓練任務中去除。舉例來說，如果測試任務的

doi:10.6342/NTU202302478

欄位是 RTE，則其對應的訓練任務為 GLUE 資料集中去掉 RTE 的其餘 6 個任務 (MNLI, QNLI, CoLA, SST2, QQP, MRPC)。

測試任務	準確率	隨機猜測準確率
MNLI	**0.4952**	0.33
RTE	**0.6861**	0.5
QNLI	0.4958	0.5
CoLA	0.4589	0.5

表 4.3: 模型在其他下游任務上少樣本學習的表現

可以看到在四個測試任務中，有兩個任務取得了顯著的效果，但在其餘兩個任務上卻效果不彰。可能的原因為我們所使用的上游任務數量較少，對於某些性質差異較大、或是資料的分布差異過大的任務，上游的訓練任務無法有效的讓模型學得能應用在下游任務上的資訊，如表中的 QNLI 和 CoLA，而這兩個任務在表4.2中做為上游訓練任務時，亦屬於移除後對模型表現較無影響的任務，顯示此二任務的性質和其他任務的差異較大。

4.3 小結

本章節介紹了如何利用 BERT 模型加上附加器並以元學習訓練，使得模型在下游任務上少樣本學習的表現有著顯著的進步，同時我們也發現超參數以及訓練任務的選擇將會大幅影響模型的表現。在下個章節我們將會採用能夠直接生成文本的 BART 模型，避免在每個任務上都重新訓練一個輸出層，並在涵蓋任務更多更廣的 CrossFit 資料集上進行更全面的實驗，並系統性的探討在上游訓練採用的演算法、訓練的參數對下游表現的影響。

doi:10.6342/NTU202302478

doi:10.6342/NTU202302478

第五章　主實驗結果分析討論

5.1　概述

本章節將系統性的比較不同的訓練方法與可訓練參數將會對模型的表現產生什麼樣的影響，並列出在不同訓練條件下，經過上游訓練後與未經上游訓練的模型在下游任務上 ARG 之差值。本章節實驗所使用之模型為 BART-base，訓練所使用之資料集為 CrossFit 資料集。

5.2　實驗結果

圖5.1列出了，為了方便閱讀，此處將訓練的相關參數都以縮寫來表示，並以 {訓練方法}_{上游可訓練參數}_{下游可訓練參數} 的格式來表示一個組合。其中縮寫的對應表如下所示：

- Meta: 元學習 (MAML)
- Multi: 多任務學習 (Multi-task Learning)
- M: 預訓練語言模型 (PLM)
- A: 附加器 (Adapter)
- MA: 元附加器 (Meta-Adapters)
- FT: 直接微調 (Fine-tuning)

舉例來說，Meta_M+A_A 代表使用上游使用元學習訓練，上游可訓練參數為預訓練語言模型及附加器，下游可訓練參數為附加器模型。FT_A 則代表不經過上游學習，直接微調附加器的參數。

doi:10.6342/NTU202302478

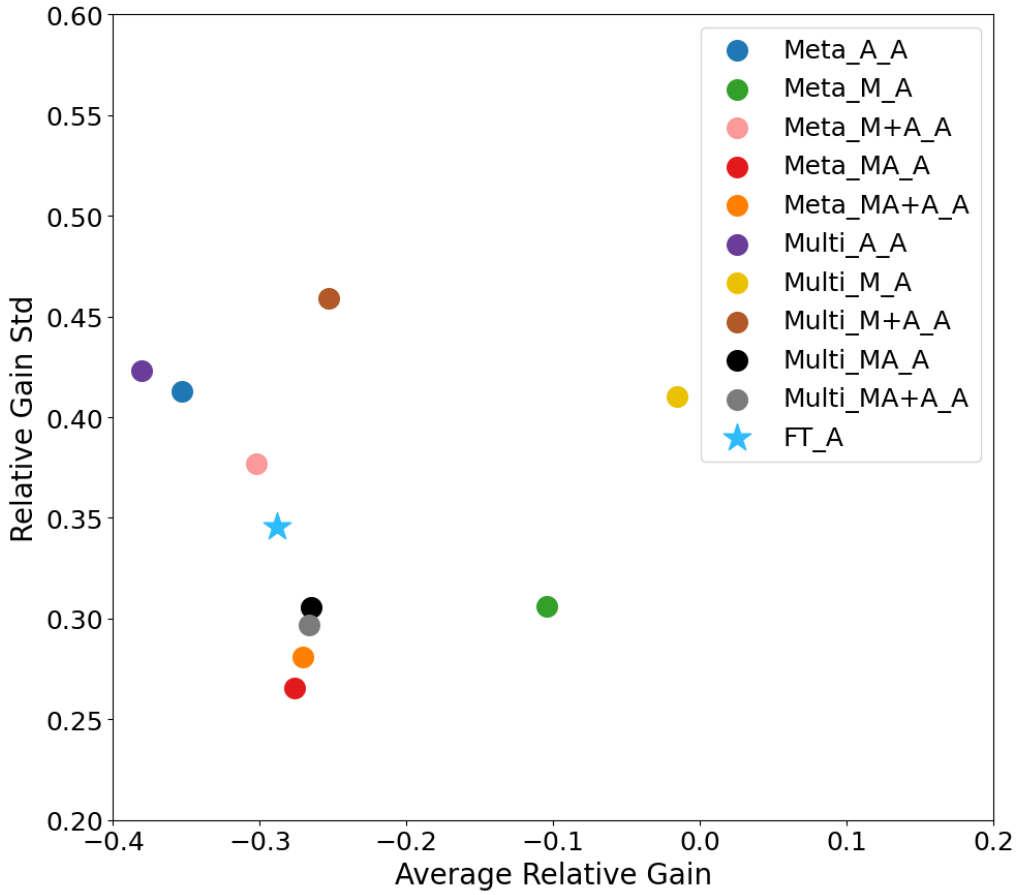


圖 5.1: 所有組合的 ARG 和 RGSTD 比較圖，其中橫軸為 ARG，縱軸為 RGSTD

在圖5.1中可以觀察到表現最好的是黃色的點 (Multi_M_A)，另外一個表現較好的點是綠色的點 (Meta_M_A)，相較於圖中淺藍色的星星 (FT_A)，也就是我們要比較的基準，有著相當大幅度的進步，證實了上游學習的有效性。

5.3 上游訓練方法

在本論文中，我們在上游學習階段採用了兩種訓練方法，多任務學習和元學習，圖5.2比較了在不同的上下游可訓練參數組合中使用兩種訓練方法的效果差異。可以看到除了 A_A，也就是在上下游皆只訓練附加器的組合以外，其餘組合相較於直接微調附加器皆有進步。而在這些組合中，可以發現使用多任務學習的組合的表現皆優於使用元學習的組合，意味著在本論文所提出的訓練框架中，多任務學習是較為適合的方法。

doi:10.6342/NTU202302478

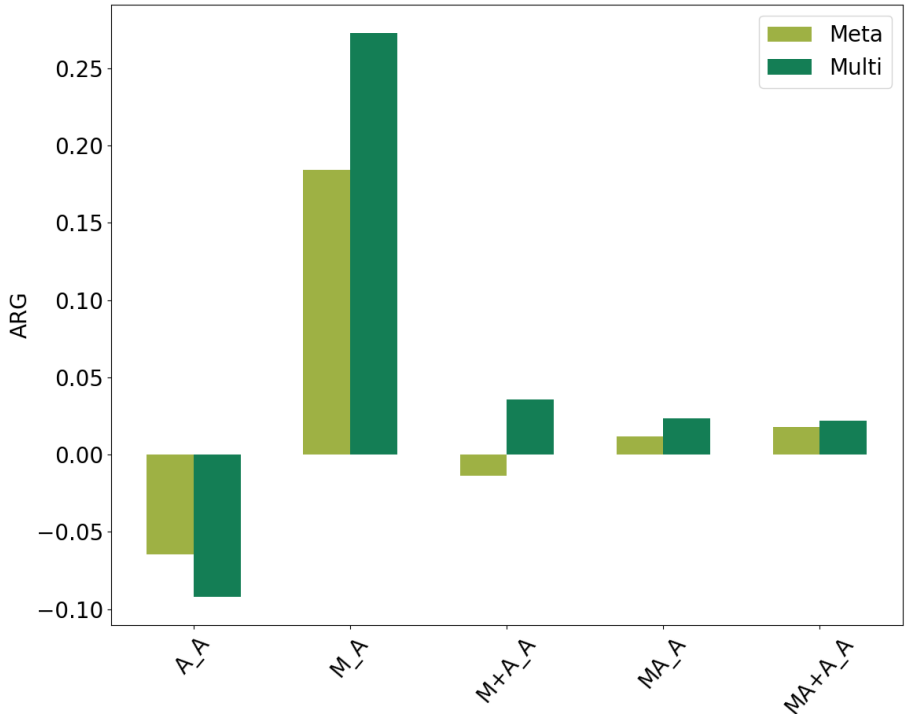


圖 5.2: 上游學習階段使用元學習與多任務學習的 ARG 之比較

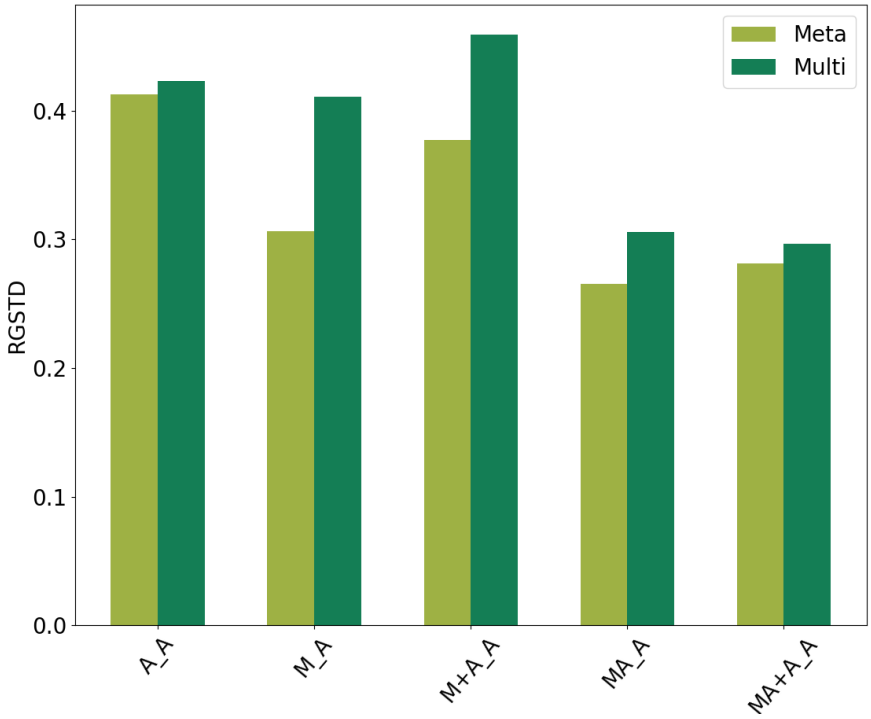


圖 5.3: 上游學習階段使用元學習與多任務學習的 RGSTD 之比較

doi:10.6342/NTU202302478

另外，在圖5.3中比較了使用多任務學習以及元學習進行訓練的模型之RGSTD的差異，RGSTD反映了經過上游學習階段的訓練後模型在各任務上進步的平均程度，RGSTD越高就代表進步的幅度越不平均，也就是模型可能在某些任務上進步特別多，在另外一些任務上則較差。可以注意到雖然在圖5.2中顯示多任務學習的效果較好，但是圖5.3指出元學習訓練的模型在不同任務上進步的幅度較為平均。

5.4 上游可訓練參數的組合

為了找出在上游訓練階段訓練哪些模型參數對下游微調最有利，表5.1列出不同可訓練參數組合的ARG平均進步幅度，可以看到其中表現最好的組合是M_A(上游只訓練PLM)，平均達到0.228的ARG進步幅度，遠高於其他種類的組合。而在所有的組合中，僅有A_A的組合不但沒有進步，反而比單純微調附加器的基準還差。

除此之外，如果將可訓練參數的組合分為「上游可訓練參數包含附加器」以及「上游可訓練參數不包含附加器」這兩種組合的話，可以發現在其他條件不變的情況下，在上游訓練的參數裡額外加入附加器並不會帶來提升。比如說在上游可訓練參數已經包含「預訓練語言模型」的情況下，在上游可訓練參數中再加入附加器，並不會使下游微調的表現變好。這告訴我們在進行上游學習階段時，並不是一味的訓練越多參數就會帶來越好的結果。

	MAML	Multi-task	Average
A_A	-0.065	-0.092	-0.079
M_A	**0.184**	**0.273**	**0.228**
M+A_A	-0.014	0.035	0.011
MA_A	0.012	0.023	0.018
MA+A_A	0.018	0.022	0.020

表 5.1: 不同可訓練參數組合的比較，表中列出的是該組合的ARG分數與直接微調附加器的ARG分數的差值

doi:10.6342/NTU202302478

5.5 任務

在前面的小節中我們以 ARG 和 RGSTD 為評量標準分析了不同訓練方法和訓練參數組合的表現，這兩個評量標準分別反映了模型在下游任務上整體的進步程度和平均程度，但卻缺乏模型實際上在各任務的表現好壞的資訊，因此圖5.4列出了模型在各個下游任務上的表現。

圖5.4所呈現的是 FT_A、Meta_M_A、Multi_M_A 三種組合的比較，X 軸是不同的下游任務，Y 軸則是模型的表現，注意不同的任務可能會有不同的評量標準，有的評量標準亦有可能出現負值 (如 glue-cola 所使用的馬修斯相關係數)。

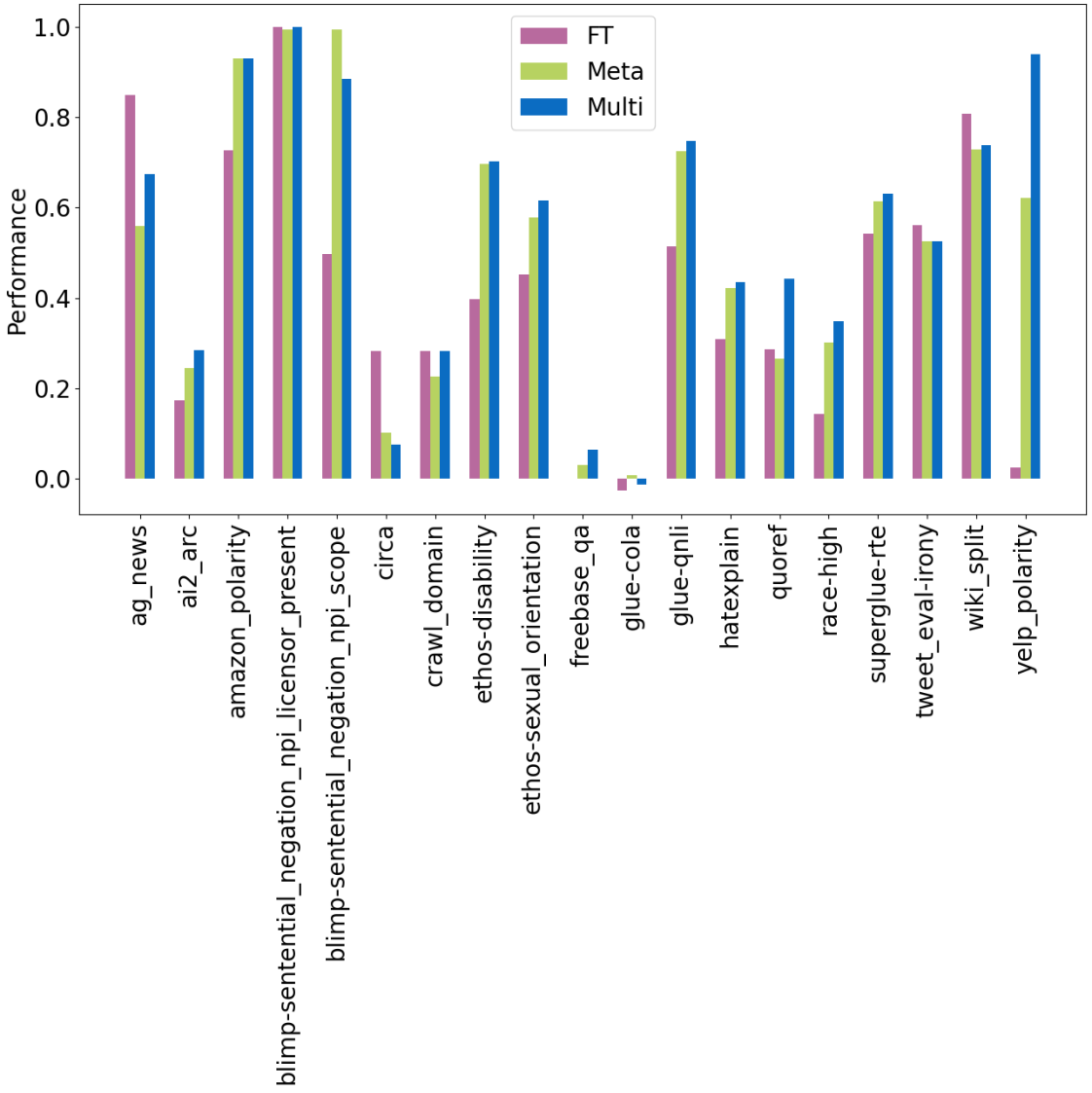


圖 5.4: FT_A、Meta_M_A、Multi_M_A 在下游任務上原始分數的比較

從圖中可以看到雖然經過上游訓練的模型 ARG 基本上都高於未經過上游訓

doi:10.6342/NTU202302478

練的 FT_A，但是在不同任務上的表現並不相同。舉例來說，在 ag_news、circa、tweet_eval-irony、wiki_split 等四個任務中，經過上游訓練後的模型表現反而較差，這樣的現象在遷移學習 (Transfer Learning) 當中稱為負遷移 (Negative Transfer)[37]，其原因可能是用來進行訓練的來源任務 (Source Task) 與進行評量的目標任務 (Target Task) 之間存在資料不匹配 (Misalignment) 的問題，在本研究所使用的資料集原論文中也有觀察到類似的現象 [38]。而除了以上所提到的任務以外，在大部分的下游任務上經過上游訓練的模型表現都有一定幅度的提升。

注意圖5.4中央的 freebase_qa，此處雖然沒有列出用於計算 ARG 分數的 Bart 基準分數，但該數值和 FT 的表現幾乎一樣，非常接近 0。由於 ARG 的計算公式會將直接微調整個 Bart 模型的基準分數放在分母，當該基準分數相當接近 0 時，會導致該任務的 RG 變得很大，進而主導了該模型整體的 ARG 分數，故在比較不同組合的 ARG 分數時將會移除 freebase_qa。

5.6 小結

本章節中我們採用 BART-base 模型加上附加器，在經過上游學習的訓練後，在下游任務上的表現相較於傳統微調附加器的做法有了大幅的進步。除此之外，本章節亦探討了在上游學習階段使用不同的訓練方法及訓練不同的參數會對下游微調的表現產生什麼影響，進而發現在上游學習階段只訓練 PLM 能為下游任務的表現帶來最大的幫助。

doi:10.6342/NTU202302478

第六章　結論與展望

6.1　研究貢獻與討論

轉換器模型在自然語言處理的領域上是一個非常重要且被廣泛使用的模型，而隨著人類科技與知識的不斷發展，預訓練的轉換器模型使用的資料量越來越多，模型本身的大小也越來越龐大，使得在下游任務上微調整個模型幾乎成了不可能的任務。因此有研究提出了在龐大的轉換器模型中插入參數量較小的模塊，這些模塊稱為附加器，當我們在下游任務上進行微調時，可以讓預訓練語言模型的參數保持固定，只訓練附加器的參數。

但並不是所有的任務都有豐富的訓練資料，舉例來說，某些少數民族的語言可能就相當缺乏訓練資料，在這種情況下模型可能只能使用極少數的訓練樣本來訓練模型，這樣的情境稱為少樣本學習。雖然在資料充足的狀況下傳統微調附加器的方式能夠取得和微調整個模型差不多的表現，但在少樣本學習的情境下表現卻遠不如微調整個模型。

本論文的貢獻在於提出了二階段的訓練框架，在下游微調階段之前加入了上游學習，利用多個少樣本的任務讓預訓練語言模型更適合微調附加器。同時進一步探討在上游學習階段使用不同訓練方法及訓練不同參數對於下游微調的影響，並成功的讓預訓練語言模型在經過上游學習後，在多個下游任務上的表現都有顯著的進步。

6.2　未來展望

本論文嘗試了多任務學習以及元學習中的模型無關元學習方法作為上游階段的訓練方法，並優化了預訓練轉換器模型微調附加器的表現，而事實上轉換器模

doi:10.6342/NTU202302478

型來說有多種輕量化模組 (Lightweight Module) 可以選擇，不一定要選擇附加器。理論上任何適用於轉換器模型的輕量化模組都適用於本論文所提出的框架，如在輸入句子中加入提示的的訓練方法 (Prompt Tuning, Prefix Tuning)[18, 20]，或是 LoRA[13] 等，都是可以選擇的方法。而多個輕量化模組的結合，在上游階段中選擇訓練不同的輕量化模組等，都是在本框架的基礎上相當值得研究的應用。

另外一方面，本論文用於衡量模型表現的評量標準 (ARG)，雖然一定程度上能夠衡量模型平均的進步幅度，但如果我們細看每個任務上的表現就會發現不同任務上的表現差距不小，且有受到單一任務 RG 過大進而主導 ARG 分數的問題。如何設計一個更公平的評量標準，用以評估模型在眾多下游任務上的表現，是一個未來可以探討的方向。

文章中及眾多研究文獻都提到過，在遷移學習中，來源任務 (Target Task) 的選擇對目標任務 (Target Task) 的表現將會有很大的影響，但對於來源任務的選擇方法卻是本文未深入探討的一項因素。未來如能找出快速有效的訓練任務選擇方法，將能夠使模型在下游任務上的表現更上一層樓。

doi:10.6342/NTU202302478

參考文獻

[1] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. Advances in neural information processing systems, 29, 2016.

[2] A. Antoniou, H. Edwards, and A. Storkey. How to train your maml. arXiv preprint arXiv:1810.09502, 2018.

[3] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[4] D. Bahdanau, K. H. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In 3rd International Conference on Learning Representations, ICLR 2015, 2015.

[5] T. Bansal, R. Jha, and A. McCallum. Learning to few-shot learn across diverse natural language classification tasks. arXiv preprint arXiv:1911.03863, 2019.

[6] Y. Bengio. Learning deep architectures for AI. Now Publishers Inc, 2009.

[7] R. Caruana. Multitask learning. Machine learning, 28(1):41–75, 1997.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[9] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning, pages 1126–1135. PMLR, 2017.

doi:10.6342/NTU202302478

[10] C.-L. Fu, Z.-C. Chen, Y.-R. Lee, and H.-y. Lee. Adapterbias: Parameter-efficient token-dependent representation shift for adapters in nlp tasks. arXiv preprint arXiv:2205.00305, 2022.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[12] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In International Conference on Machine Learning, pages 2790–2799. PMLR, 2019.

[13] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

[14] K. Kawaguchi. A multithreaded software model for backpropagation neural network applications. The University of Texas at El Paso, 2000.

[15] T. Khot, A. Sabharwal, and P. Clark. Scitail: A textual entailment dataset from science question answering. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[17] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. nature, 521(7553):436–444, 2015.

[18] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691, 2021.

[19] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461, 2019.

[20] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. arXiv preprint arXiv:2101.00190, 2021.

doi:10.6342/NTU202302478

[21] B. McCann, N. S. Keskar, C. Xiong, and R. Socher. The natural language decathlon: Multitask learning as question answering. arXiv preprint arXiv:1806.08730, 2018.

[22] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

[23] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. arXiv preprint arXiv:2005.00247, 2020.

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

[25] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21(140):1–67, 2020.

[26] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. 2016.

[27] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386, 1958.

[28] P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations. arXiv preprint arXiv:1803.02155, 2018.

[29] A. C. Stickland and I. Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In International Conference on Machine Learning, pages 5986–5995. PMLR, 2019.

[30] S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. Advances in neural information processing systems, 28, 2015.

[31] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27, 2014.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

doi:10.6342/NTU202302478

[33] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. Advances in neural information processing systems, 32, 2019.

[34] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461, 2018.

[35] Y. Wang, S. Mukherjee, X. Liu, J. Gao, A. Awadallah, and J. Gao. List: Lite prompted self-training makes parameter-efficient few-shot learners. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 2262–2281, 2022.

[36] Y. Wang, S. Mukherjee, X. Liu, J. Gao, A. H. Awadallah, and J. Gao. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. arXiv preprint arXiv:2205.12410, 2022.

[37] S. Wu, H. R. Zhang, and C. Ré. Understanding and improving information transfer in multi-task learning. arXiv preprint arXiv:2005.00944, 2020.

[38] Q. Ye, B. Y. Lin, and X. Ren. Crossfit: A few-shot learning challenge for cross-task generalization in nlp. arXiv preprint arXiv:2104.08835, 2021.

[39] E. B. Zaken, S. Ravfogel, and Y. Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. arXiv preprint arXiv:2106.10199, 2021.

[40] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578, 2016.

doi:10.6342/NTU202302478