Lab02 Report – Deep Learning

學生：梁皓鈞，學號：314580042，系所：前瞻半導體研究所晶片組

一、實作 Transformer 架構

最初的 Baseline 為助教模板：Multi Head Attention、Transformer Encoder Layer、Transformer Decoder Layer、Transformer Encode/Decode、Positional Encoding、Token Embedding，甚至推論用的 Translate 都還在 TO-DO 階段，沒有可用的 Attention、Mask，連參數量也無從計算。最終的提交版本從零手刻出符合 Attention Is All You Need 論文的 Vanilla Transformer 架構，並遵守課堂限制，不倚賴被禁止的 PyTorch 清單 APIs。

完整模型採四層 Encoder Layers 搭配四層 Decoder Layers，其中，Word Embedding Dimension（d_{model}）被設置為 256，Attention Heads 數量則設置為 8，因此每個 Head 的d_k即為d_{model}除以 Attention Heads 數量得到 32。

每個 Encoder Layer 先做 Multi-Head Self-Attention，再加上 Residual Add & Layer Normalization，接著以兩層前饋網路進行處理（Fead-Forward Network：256 → 1024 → 256，中間各別有使用到 ReLU 以及 Dropout 功能），最後再一次 Residual Add & Layer Normalization。

每個 Decoder Layer 則與 Encoder Layer 類似，但在 Self-Attention 之外，還包含了對 Encoder 輸出的 Cross-Attention，且 Self-Attention 部分會套用因果遮罩（Causal Mask）避免洩漏未來的 Token。

全域 Dropout 設置為 0.15，能在小型資料規模下提供足夠 Regularization。輸入端與輸出端各有獨立的 Token Embedding（會乘上$\sqrt{d_{model}}$來穩定訓練），再加上 Sinusoidal Positional Encoding；位置編碼依原論文的正弦/餘弦封閉形式實作，支援到長度 5000。詞彙表大小方面，中英兩端各約 13,000。最後透過線性投影到目標語言詞彙表，得到各時間步的機率分佈。

Multi-Head Attention 方面完全以矩陣乘法手刻，並沒有使用到 nn.functional 提供的 scaled_dot_product_attention()：先用獨立線性層投影出 Q、K、V，重排為(Batch Size, #Heads, Sequence, d_k)後，計算$\text{Scores} = QK^T/\sqrt{d_k}$，在送入 Softmax 運算前，會以 Mask 把不允許注意的位置填入極大負值，因此經過 Softmax 後便等效於分數歸零，再與注意權重加權 V，合併各 Head，經輸出端做投影並搭配 Dropout。

Mask 方面，Encoder/Decoder 都處理 Padding Mask（形狀會 Broadcast 到(Batch Size, 1, 1, Length of Sequence)），Decoder 的 Self-Attention 另有下三角 Causal Mask；兩者 Bitwise-AND 後套用在 Attention Scores 上，確保自回歸訓練與變長序列都被正確處理而不致於洩漏資訊。

每個 Layer 皆採 Residual Add 後搭配 Layer Normalization，符合 Add-Norm 的標準做法。整體模型參數量約 27.66M，遠低於 100M 的作業上限。之所以將原本助教模板中的 6 層、前饋網路 1024 維、64 個 Head 等設定下修，是因為資料規模只有 5 萬對句子：以 256 維、8 個 Head、前饋網路 1024 維的 4+4 層設計，更能避免過度參數化、過擬合，也能把 Batch 做大，使訓練更有效率。

二、訓練策略（資料處理、最佳化與調度）

Dataset 的部分包含五萬組中英對句，拆成四萬五千組訓練集與五千組驗證集，另有兩百組測試集。為提供穩定梯度與適當的訓練 GPU Throughput，Batch Size 被設置為 48，而 Data Loader 開啟 Shuffle 打亂資料順序，避免對語料的原始排序產生擬合。損失函數採 Cross Entropy，忽略<PAD>位置，並加入設置為 0.08 的 Label Smoothing 來降低模型可能的過度自信情況，來改善泛化、校準機率。

Optimizer 的部分使用 AdamW，學習率採用5×10^{-4}，$\beta = (0.9, 0.98)$、$\varepsilon = 1 \times 10^{-9}$，Weight Decay 設置為 0.01。選 AdamW 而非 Adam，是因為 Decoupled Weight Decay 在 Transformer 上的 Regularization 效果更佳，亦為 Natural Language Processing 常見的 Default Setting。

Learning Rate Scheduler 的部分採 One Cycle Learning Rate，總 Epoch 數被設置為 100；依每個 Epoch 推導出約 937 個 Step(約為 45,000 組訓練資料除以 48 大小 Batch Size 的近似值)，前 10%的步數做 Warming Up，慢慢把學習率從約2×10^{-5}平滑拉升到5×10^{-4}後，以 Cosine 退火一路降到約5×10^{-8}。此策略初期可避免梯度爆炸與發散，中期可用較高學習率來搜索較尖銳的極小值，末期再以極低學習率做細緻收斂；實務上讓模型在 30–40 個 Epoch 其實就達到不錯的 Loss 水準。其中，Scheduler 更新是在訓練迴圈的每個 Step 進行，而非每個 Epoch 進行。

梯度爆炸的部分，在 Loss 後傳之後、Optimizer 更新前做 Clipping，把整個模型梯度的范數裁切到 1.0 來避免。Early Stopping 的部分以驗證集準確率為指標，耐心值為 10 個 Epoch，若連續 10 次無進步則停止，並以最佳驗證集準確率當做權重的 Checkpoint。

三、效能與品質的改進

最大改進從「把模型寫到能動」開始：把助教所提供之僅有骨架的模板程式碼，補齊所有核心組件—從手刻的 Scaled Dot-Product Attention、完整的 Encoder/Decoder Layer、正確的 Masking，到 Inference Pipeline，並在結構上將 6 層 Encoder/Decoder、前饋網路 1024 維、64 個 Head，進行謹慎實驗重新設計為 4 層 Encoder/Decoder、前饋網路 1024 維、8 個 Head。如

此一來，結論是把參數量控制在約 27.66M，既保住足夠的模型表達能力，又能把 Batch Size 拉到 48，帶來更穩定的梯度、更快速的訓練迭代；對五萬對句子的資料集規模也較為匹配，降低模型過擬合風險。Regularization 除了設置為 0.15 的 Dropout，也使用設置為 0.08 的 Label Smoothing 與設置為 0.01 的 Weight Decay 強化模型泛化能力。

訓練程序上，OneCycleLR 算是蠻關鍵的角色：前 10% warmup 讓模型穩穩起步，之後的 Cosine 退火比傳統 StepLR 相比較為平滑，最終能在更少 Epoch 內收斂到相近甚至更佳的解。同時加入梯度裁切到最大范數 1 避免早期發散，並用早停避免模型在驗證集準確率表現停滯時浪費算力。資料面把 Batch Size 從 8 提到 48 並開啟 Shuffle，帶來訓練穩定度與速度提升。

推論算法方面，我將 Greedy Search 升級為帶長度正規化的 Beam Search，並把原本固定的 Length Penalty 改為自適應長度懲罰，依來源句長自動決定，公式如下：

$$\text{Length Penalty} = 0.5 + 0.01 \times \text{Min}(\text{Source Length}, 30)$$

並將 Length Penalty 的值域控制在 0.5 到 0.8 的範圍內。約五個單詞的短句會落在 0.55 附近，允許較長譯文；約 15 個單詞的中句會落在 0.65 附近，較為平衡；長於 30 個單詞的長句會落在 0.8 附近，抑制過度生成。其餘設定維持 Beam Width 長度為 3、Max Length 長度為 128。優化的部分則包含：Encoder 只前向一次、已完成的 Beam 不再解碼、且當所有 Beam 皆產生<EOS>即提前停止。

Mask 的實作對模型訓練品質非常關鍵。Padding Mask 會在注意力中屏蔽<PAD>的位置；Decoder 的 Causal Mask 以下三角矩陣確保每一步只看得到現在與過去，並把未來給屏蔽住；兩者相乘後於 Softmax 前填入極小值，讓對應到的注意力分數為零；既避免資料外洩，也讓變長序列的訓練、推論更穩定。

配合上述設定，最終模型在作業需求下可達 BLEU-1 約 0.4771、BLEU-2 約 0.337，200 句測試集推論時間 9.3 秒，參數量 27.66M，全面滿足 BLEU 門檻、推論時間上限與參數量上限。測試結果如下，並搭配當時測試環境的 GPU 資訊。

```
The parameter size of model is 27656.516 k
====================  PASS parameter size requirement  ====================
BLEU score (1-gram) =  0.47708785090595485
BLEU score (2-gram) =  0.33660926871001723
BLEU score (3-gram) =  0.24592221073806286
BLEU score (4-gram) =  0.1708295253664255
====================  PASS BLEU score requirement      ====================
execution time = 9.343s
====================  PASS execution time requirement  ====================
```

```
Wed Oct 15 15:20:46 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.65.06              Driver Version: 580.65.06      CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4070 ...    Off |   00000000:01:00.0 Off |                  N/A |
|  0%   51C    P8             24W /  285W |   15121MiB /  16376MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A            1872      G   /usr/lib/xorg/Xorg                        63MiB |
|    0   N/A  N/A            2101      G   /usr/bin/gnome-shell                       7MiB |
|    0   N/A  N/A          254148      C   ...orge3/envs/dl2025f/bin/python        9050MiB |
|    0   N/A  N/A          254890      C   ...orge3/envs/dl2025f/bin/python        5960MiB |
+-----------------------------------------------------------------------------------------+
```

四、遭遇的困難與解法（調參與除錯經驗）

起步時最大門檻是閱讀論文：Attention is all you need。Multi-Head Attention、Positional Embedding、Mask、Residual Add 與 Layer Normalization 等模組如何組出一個 Pipeline，需要來回閱讀並交叉對照範例實作。我分段閱讀、手繪資料流圖，再把各模組獨立跑通；前後花約 10-15 小時才建立足夠直覺。

從零實作 Multi-Head Attention 時，最常見的錯誤來自於張量維度的匹配與轉置：例如忘記對 K 做轉置矩陣就去算QK^T、把(Batch Size, Sequence, Sequence)的 Mask 直接拿去 Broadcast 到 Multi-Head 而導致維度形狀不符、或漏了除以$\sqrt{d_k}$導致注意力過尖。整體開發流程為：先寫對單頭，再擴展到多頭，並在每步印出 Shape 檢查，用極小的 Batch / Length 做手算比對；必要時以 PyTorch 官方 Multi-Head Attention 在 Toy Data 上交叉驗證邏輯。

Mask 的實作上也踩了幾個坑：一開始把布林條件方向弄反、在 Softmax 之後才套遮罩、或把 Mask 留在 CPU 導致 Device Mismatch。最後將 Padding Mask 寫成Sequence ! = PAD_IDX，取 Unsqueeze(1)，並取 Unsqueeze(2)；Causal Mask 用 Tril 產生下三角，再把兩者相乘後於 Softmax 前以 Masked Fill $(\text{Mask} == 0, -1 \times 10^9)$套用；並在單元測試裡直接列印小矩陣，肉眼確認三角形結構與 Broadcast 形狀。這段調整約花 4-5 小時。

訓練早期也遇到發散、損失 NaN 與高方差等問題；主因是沒有梯度裁切、學習率太高、缺少 Warming Up 機制、Dropout 太低、Batch Size 太小。依序加入 Clip Grad Norm (…, 1.0)、改用 One Cycle Learning Rate 引入 Warming Up，把 Dropout 從 0.1 調到 0.15，Batch Size 大小從 8 提到 48，並加上設置為 0.8 的 Label Smoothing，訓練不穩定現象才受到控制。這段試錯約略花費 8-10 小時，過程中也對比不同學習率組合（如1×10^{-3}在第 7 個 Epoch 就爆掉），最

後收斂速度與泛化皆明顯改善。

推論速度方面，首個實作為 Greedy Algorithm，在 200 句測試集上推論約 3.64 秒；爾後經過大量實驗後，真正奏效的是把 Beam Search 做好：對 Encode 結果 Cache、對已完成的 Beam 停止展開、全部完成則提前結束，並配合自適應長度懲罰的長度正規化，避免固定 Penalty 造成偏短或重複。於是平均句長更合理、收斂更穩定（例如：「你好嗎？」會直接收斂成「How are you? 」而非重複拖長）。最終在 Beam 為 3 時，長度自適應的 Beam Search 約 9.404 秒，雖較 Greedy 慢，但 BLEU Score 同步上升。這部分的分析與優化耗時約 6 到 7 小時。以下是我全部推論算法所做過的實驗記錄：

```
METHOD: greedy
BLEU score (1-gram) =  0.4607506287842989
execution time = 3.638s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================


PART 1: LENGTH PENALTY TUNING (beam3 variants)
------------------------------------------------

METHOD: beam3_p50
BLEU score (1-gram) =  0.4741560998931527
execution time = 9.411s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: beam3_p55
BLEU score (1-gram) =  0.4761213091388345
execution time = 9.234s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: beam3_p60
BLEU score (1-gram) =  0.4766863848641515
execution time = 9.275s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: beam3_p65
BLEU score (1-gram) =  0.47493975449353454
execution time = 9.438s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: beam3_p70
BLEU score (1-gram) =  0.47493975449353454
execution time = 9.496s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: beam3_p75
BLEU score (1-gram) =  0.4716410121694207
execution time = 9.443s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: beam3_p80
BLEU score (1-gram) =  0.46907156784087417
execution time = 9.410s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================
```

```
PART 2: ADVANCED METHODS
------------------------

METHOD: length_adaptive
BLEU score (1-gram) =  0.47708785090595485
execution time = 9.404s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: diverse_beam
BLEU score (1-gram) =  0.47630170088261364
execution time = 17.774s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: mbr_fast

METHOD: mbr


PART 3: SAMPLING METHODS (Reference)
------------------------------------

METHOD: sample_t70
BLEU score (1-gram) =  0.4365963597781956
execution time = 4.636s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================

METHOD: sample_t80
BLEU score (1-gram) =  0.43386183857917787
execution time = 4.019s
====================  PASS execution time requirement  ====================
====================  PASS BLEU score requirement      ====================
```

為了把 Batch 做大與嘗試更大的模型，過程中也碰到數次 CUDA Out-Of-Memory：如 Batch Size 在 64 於 16GB 的 GPU 會噴錯、d_{model}設置為 512 搭配 Batch Size 大小 32 也會 CUDA Out-Of-Memory。最後把 Batch Size 固定在 48，模型維持 27M 級別，必要時清理快取並密切觀察 *nvidia-smi*，作為訓練速度、容量與記憶體之間的妥協。除此之外，形狀不一致帶來的 Runtime Error 時常出現；為此，我在關鍵張量旁加詳細 Shape 註解、使用較容易懂的變數名稱（如 src_seq_len/tgt_seq_len），並在前傳的關鍵輸出口加上 Assert 檢查；像推論裡忘記加 Batch 維度這種小失誤，就靠這些東西快速定位跟修正了。

評估指標的穩定性也需留意。早期因未固定 Random Seed、驗證前忘了 model.eval()，導致 BLEU 分數在不同 Run 波動很大（例如 0.45～0.48）。把種子固定、正確切換到 Eval 模式後，分數方能可重現。Hyperparameter 採逐步式調整：先用文獻常見配置把模型跑起來，再分階段掃學習率（1×10^{-4}～7×10^{-4}）、Batch Size（16～64）、Dropout（0.1/0.15/0.2）、Label Smoothing（0/0.05/0.08/0.1），最後在推論端微調 Beam Width（1/3/5/7）與 Length Penalty（0.4/0.6/0.8/1.0）。整體調參歷時約 15-20 小時。從進度來看，四部份節奏分為：第一部分建構與驗證模組，第二部分調整訓練、Regularization，第三部分把推論做快做穩，第四部分收斂。

綜合上述解法與迭代，最終模型以 4 Encoder + 4 Decoder、8 個 Heads、d_{model}設置為 256、Dropout 設置為 0.15、Label Smoothing 設置為 0.08 的配置，在五萬對句子的資料集規模上取得 BLEU-1 約 0.47-0.48、BLEU-2 約 0.33-0.34 的表現，200 句推論約 9.4 秒，參數量 27.66M，

完整符合作業三大門檻，也把我在閱讀、實作、除錯與調參上的收穫，確實反映在模型品質與系統效率上。