Artificial Intelligence Final Report Assignment 問題 3 (Problem 3)
レポート解答用紙 (Report Answer Sheet)

Group Leader
学生証番号 (Student ID): 18521359
名前(Name): レ・ミン・タイ (Le Minh Tai)

Group Members
学生証番号 (Student ID): 18521127
名前(Name): グエン・フォン・ナム (Nguyen Phuong Nam)

学生証番号 (Student ID): 18521490
名前(Name): ファン・ヴー・キエウ・チエン (Pham Vu Kieu Tien)

問題 3 (Problem 3)のレポート

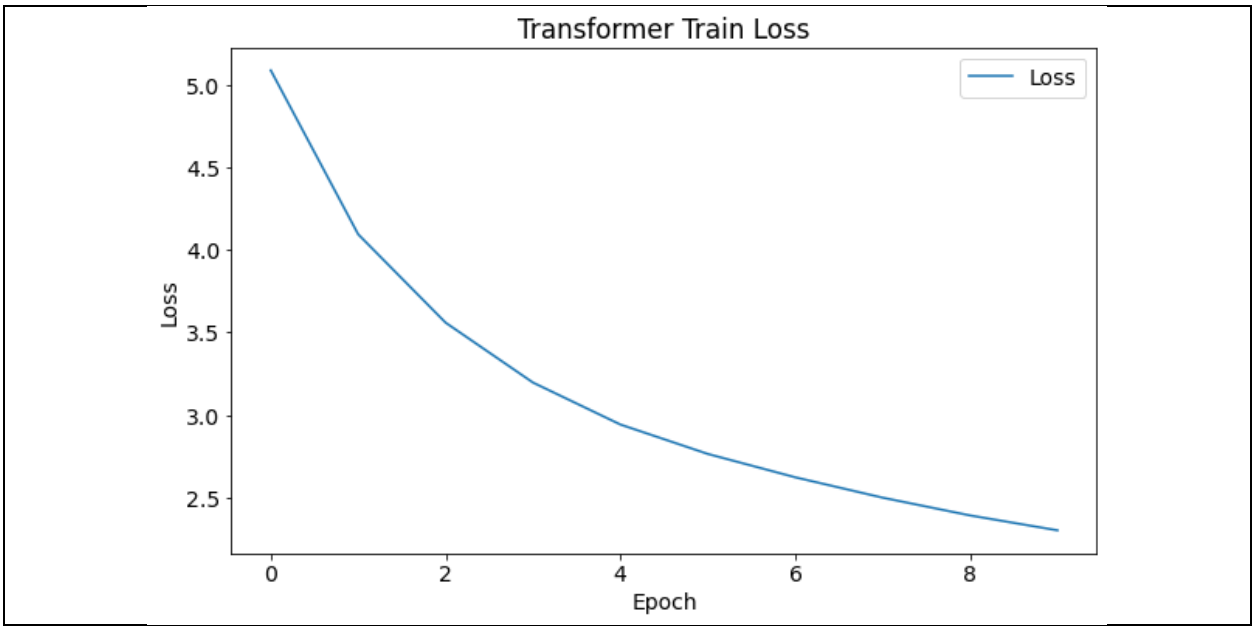
IWSLT15(en-vi)のデータセットに対してより高い精度(BLEU)を実現するプログラム(PyTorch)を作成せよ。ただし、プログラムは第 13 回の講義資料のプログラム(もしくは Lab Work (6)で作成したプログラム)を改良して作成せよ。そのプログラムと実行結果およびそれらに関する解説を word ファイルにまとめて提出せよ。

(プログラム)

リンク: 問題 3 -プログラム

(実行結果)

- GPU で約 1 時 14 分
- BLEU: 19.21%
- Loss: 1.852



（解説）

❑ システム要件

- Windows10 でオペレーティングシステム
- CPU: Intel Core i5 8300H
- RAM: 8GB
- GPU: NVIDIA GTX 1050 4G Python 3.8
- Pytorch 1.9
- CUDA 11+

❑ EPOCH = 10、BATCHSIZE = 2500、学習率(LR) = 0.001

❑ 最適化パラメータ

- torch.optim.Adam(model.parameters(), lr=LR, betas=(0.9, 0.98), eps = 1e-9)

❑ 誤差計算

- torch.nn.functional.cross_entropy()

❑ **モデル： Transformer (Multi-Head Attention)**

❖ Encoder のクラス：

- エンコーダーからまずみてみましょう。入力はまず単語埋め込み層によって、埋め込み表現に変換されます。
    - Embedding のクラス：
        - 非表示層 512 次元、英語文におけるトークン 21891 数量
            - torch.nn.Embedding(21891, 512)
- その後、 positional encoding によって、位置情報が単語埋め込みに付加されます。ここの計算は 単純に単語埋め込みと位置エンコーディングの足し算を計算します。
    - 非表示層 512 次元、続編の最大長=800、dropout 率=0.1、現在の非表示層 **i**、文のパディング **pos**
        - math.sin(pos / (10000 ** ((2 * i) / 512)))
    - 非表示層 512 次元、続編の最大長=800、dropout 率=0.1、次の非表示層 **i+1**
        - math.cos(pos / (10000 ** ((2 * (i+1)) / 512)))
    - unsqueeze(0)で positional encoding(上記の結果)を追加する。
        - torch.unsqueeze(PositionalEncoding, 0)
    - Dropout のライブラリ: torch.nn.Dropout
        - dropout 率=0.1
            - torch.nn.Dropout(0.1)
- 続いて、MultiHeadAttention の層を通過し、Norm 層で再合流していま す。Norm は残差接続(residual connection)とレイヤー正規化(layer normalization)を行っています。
    - Norm のクラス：
        - 非表示層 512 次元
        - Parameter のライブラリ: torch.nn.Parameter
            - torch.nn.Parameter(torch.ones(512))
            - torch.nn.Parameter(torch.zeros(512))
    - MultiHeadAttention のクラス：
        - Attention のクラス：
            - Q: エンコーダーの内部状態
            - K: エンコーダーの内部状態
            - V: エンコーダーの内部状態(Q と K と V は同じ)
            - score: QK^Tを計算し、各行ごとに softmax の計算
            - Dropout のライブラリ: torch.nn.Dropout

        - dropout 率=0.1
            - torch.nn.Dropout(0.1)
    - 内部状態を表すベクトル(***h***)をヘッド数(H)に分割して、それぞれの分割さ れたベクトルに対してアテンションの計算を行う
        - *d*: 内部状態の次元数 (非表示層次元)
        - *H*: ヘッド数
        - *d_k*: 各ヘッドの次元数
        - $d = 512, H = 8$ なら、$d\_k = 512/8 = 64$ 次元
    - 各ヘッドのアテンションの計算
        - scores = torch.matmul(q,k.transpose(-2, -1))/math.sqrt(d_k)
    - アテンションの計算 (*Attention*)の計算結果は、マルチヘッドでそれぞれ計算したアテンションの計算結果 を連結して、連結されたベクトルに対し絵 t 線形変換をして得られます。
        - scores = torch.nn.functional.softmax(scores, dim = -1)
        - *Attention* = torch.matmul(scores, V)
    - Linear のライブラリ: torch.nn.Linear
        - 表示層 512 次元
            - torch.nn.Linear (512,512)
    - 最終的なアテンションの計算結果は、マルチヘッドでそれぞれ計算したアテンションの計算結果 を連結して、連結されたベクトルに対し絵 t 線形変換をして得られます。
        - Dropout のライブラリ: torch.nn.Dropout
            - dropout 率=0.1
                - torch.nn.Dropout(0.1)
        - *Attention*(Q, K, V, 0.1)

❖ その後、feed forward 層では単純な線形変換と ReLU の計算を行っています。

- Feed Forward 層の 2048 次元、表示層 512 次元
- **__init__メソッド**
    - Dropout のライブラリ: torch.nn.Dropout
        - dropout 率=0.1
            - torch.nn.Dropout(0.1)
    - Linear のライブラリ: torch.nn.Linear

- torch.nn.Linear の第一引数: 全結合層の入力次元数
- torch.nn.Linear の第二引数: 全結合層の出力次元数

- **forward メソッド**
  - l1 が 512 次元(入力層)を Feed Forward 層次元(2048 次元)に圧縮
    - l1 = torch.nn.Linear(512, 2048)
  - l1 層に対しては、活性化関数の relu 関数
    - l1 = torch.nn.functional.relu(l1)
  - l1 の後に Dropout を適用し: torch.nn.Dropout
    - dropout 率=0.1
      - l1 = torch.nn.Dropout(l1)
  - l2 は l1 の出力を受け取り、512 次元のベクトル(出力層)に変換
    - l2 = torch.nn.Linear(2048, 512)

- Decoder のクラス：
  - ❖ エンコーダーと同じ様に、出力に対して、単語畳み 込みと位置エンコーディングを行った。
    - Embedding のクラス：
      - 非表示層 512 次元、英語文におけるトークン 15708 数量
        - torch.nn.Embedding(15708, 512)
  - ❖ (マスク付き)自己アテンションの計算を行い、Norm 層の処理を行います。
    - MultiHeadAttention のクラス：
      - Attention のクラス：
        - Q: エンコーダーの内部状態
        - K: エンコーダーの内部状態
        - V: エンコーダーの内部状態(Q と K と V は同じ)
        - score: QK^Tを計算し、各行ごとに softmax の計算
        - Dropout のライブラリ: torch.nn.Dropout
          - dropout 率=0.1
            - torch.nn.Dropout(0.1)
        - unsqueeze(0)で マスクを追加する。
          - torch.unsqueeze(マスク, 1)

- 内部状態を表すベクトル($\boldsymbol{h}$)をヘッド数(H)に分割して、それぞれの分割さ れたベクトルに対してアテンションの計算を行う
  - d: 内部状態の次元数 (非表示層次元)
  - H: ヘッド数
  - d_k: 各ヘッドの次元数
  - $d = 512, H = 8$ なら、$d\_k = 512/8 = 64$ 次元
- 各ヘッドのアテンションの計算
  - scores = torch.matmul(q,k.transpose(-2, -1))/math.sqrt(d_k)
- アテンションの計算 ($Attention$)の計算結果は、マルチヘッドでそれぞれ計算したアテンションの計算結果 を連結して、連結されたベクトルに対し絵 t 線形変換をして得られます。
  - scores = torch.nn.functional.softmax(scores, dim = -1)
  - $Attention$ = torch.matmul(scores, V)
- Linear のライブラリ: torch.nn.Linear
  - 表示層 512 次元
    - torch.nn.Linear (512,512)
- 最終的なアテンションの計算結果は、マルチヘッドでそれぞれ計算したアテンションの計算結果 を連結して、連結されたベクトルに対し絵 t 線形変換をして得られます。
  - Dropout のライブラリ: torch.nn.Dropout
    - dropout 率=0.1
      - torch.nn.Dropout(0.1)
  - $Attention$(Q, K, V, マスク, 0.1)

❖ その後、エンコーダーから内部状態を受け取り、言語間アテンションの計算を行います。Norm 層の処理を行ます。

❖ feed forward 層の計算を行い、最後に線形変換と softmax により単語出力確率を計算します。