深度學習

HW12

學號 : B103012002

姓名 : 林凡皓

1. Load the toy data and Preprocessing data

Transformers對於seq-to-seq的資料有很好的表現，這一次作業中我們將它用來做數學運算。這次作業會採用transformer模型來算數學的加減法。

首先查看這次要訓練的資料，前四筆資料如下

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

以第一個資料為例作說明。

第一筆資料的輸入為BOS NEGATIVE 30 subtract NEGATIVE 34 EOS，輸出為BOS POSITIVE 04 EOS。將它翻譯成我們習慣的數學表示式就會變成輸入(-30) – (-34)，得到輸出+4，而BOS和EOS分別代表序列的開始與結束。

接著對這些數據做tokenization。Tokenization為一種資料預處理方式，我們需要先將人類閱讀的文字轉換為一個序列的token。首先我們需要創建一個詞彙表，在這次作業中，詞彙表會由16個元素組成，分別為數字0~9，正負號標記(POSITIVE、NEGATIVE)、運算標記(add、subtract)和序列開始與結束的標記(BOS、EOS)。

通常我們以list來實現詞彙表，其中vocab[i] = s代表說字串符號s被分配到索引i。此外，我們會建立一個dictionary convert\_str\_to\_token來幫助我們將字串映射到其相應的整數索引。

建立好vocab之後我們實現兩個函數分別為generate\_token\_dict和preprocess\_input\_sequence。

* generate\_token\_dict :

1. 實現方法 :

此函數的功能為接收一個list vocab並回傳一個dictionary，此dictionary為用來convert\_str\_to\_token的字典。

透過enumerate(vocab)來抓取vocab list中的索引與存放的單字，定將抓取到的索引設定為dictionary的數值，單字設定為dictionary的key。

1. 執行結果 :



* preprocess\_input\_sequence :

1. 實現方法 :

此函數用來將一個輸入字串轉換為vocab list中的token。

首先要利用split將輸入句子中每個字都分開，並對每一個字做迭代。每一次迭代先去判斷目前的字為特殊字元或是數字，如果是特殊字元就利用dictionary將token抓出來，如果是數字就要針對每一個數字再做一次迭代(因為最初的數字可能為二位數)，再去dictionary中抓取token。

1. 執行結果 :

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

1. Implementing transformer building blocks

完成資料預處理後，接下來要實作transformer使用的building block，而這些building block會在後續建構encoder與decoder時用到。

這邊先實作MultiHeadAttention block、FeedForward block、Layer Normalization和Positional Encoding block。

在實作MultiHeadAttentin block之前，由於我們想透過對Self Attention block做擴充的方式來實現MultiHeadAttention block，因此先去實作Self Attention block。

* Self Attention block :

Self attention block的架構如下

一張含有 文字, 圖表, 數字, 筆跡 的圖片

自動產生的描述

根據此架構，實現scaled\_dot\_product\_two\_loop\_single、scaled\_dot\_product\_two\_loop\_batch和scaled\_dot\_product\_no\_loop\_batch。

1. scaled\_dot\_product\_two\_loop\_single :

此函數為最基本的self attention block。

利用兩個for loop計算query和key的乘積，每一次迭代都去計算query與key的內積。接著透過來計算attention。再將attention送入softmax後，與value相乘得到最後的輸出。

檢查結果如下



1. scaled\_dot\_product\_two\_loop\_batch :

根據剛才實現的scaled\_dot\_product\_two\_loop\_simple做延伸，將它改成對batch N的self attention。與上一個函數的主要差別在於剛才每次迭代的內積修改為torch.einsum來實現對每個batch做內積。

檢查結果如下



1. scaled\_dot\_product\_no\_loop\_batch :

此函數有加入一個mask的選項，用來選擇是否要採用masked self attention。在mask = False時，此函數做的事與剛才的scaled\_dot\_product\_two\_loop\_batch相同，差別在於這邊直接使用torch.bmm來實現query和key在每個batch的相乘，而非透過迴圈。

當mask = True時，主要的差別在於attention的部分，詳細masked self attention的架構如下

一張含有 文字, 圖表, 方案, 行 的圖片

自動產生的描述

由上圖可以注意到，masked self attention的attention會有部分被設定為負無窮大的數值(紅色三角形)，這主要是因為我們希望忽略掉這些位置的影響。在實作的部分，我們則利用-1e-9來取代負無窮大。

檢查結果如下



實作完成後，討論transformer的時間複雜度。分別對sequence length = 256與sequence length = 512的序列做測試，測試結果如下

sequence length = 256 :sequence length = 512 :

由結果可以看到時間差了四倍，因此transformer的時間複雜度約為。

接著將SelfAttention block完成。

1. \_\_init\_\_ :

初始化部分有q、k和v要做初始化，channel shape分別為(dim\_in, dim\_q)、(dim\_in, dim\_q)、(dim\_in, dim\_v)。這變將他們初始化為一個linear layer，因為它們主要用來將輸入映射到query、key和value。關於linear layer中權重的初始化，weight都是初始化為大小介於到之間的uniform distribution，而bias則初始化為0。

1. forward :

forward path的部分即剛才定義好的scaled\_dot\_product\_no\_loop\_batch。

1. 執行結果 :



* MultiHeadAttention:

MultiHeadAttention block將輸入分為多個小塊，每一塊會有一個self attention block去處理，以達到平行計算的效果，最後再將每一個self attention的輸出結果拼接起來，形成最終輸出。

1. 實現方法 :

初始化部分，由於MultiHeadAttention block有多個SelfAttention block，因此可以透過nn.ModuleList來實現多個SelfAttention block的架構，而SelfAttention block的數量取決於heads的數量。此外我們還需要一層linear layer將每一個SelfAttention block的輸出結果轉換為輸入維度。關於linear layer權重初始化部分，weight採用xavier的初始化方式，而bias初始化為0。

Forward path的部分，將query、key、value送進每一個SelfAttention block計算出輸出，再將所有輸出拼起來，最後再透過linear layer將維度映射到輸入的維度。

1. 執行結果 :



* LayerNormalization :

在做CNN相關作業時有時做過Batch normalization，但是batch normalization會因為批次的大小而影響到表現，特別是在處理序列資料時，因為序列長度不同導致批次大小不穩定，batch normalization的表現也會跟著不穩定，因此這次改為使用layer normalization。Layer normalization不依賴批次大小，而是對每個timestep單獨做歸一化，其優點在於它可以做平行化處理，而且在測試時的運作方式與在訓練時相同。

1. 實現方法 :

首先是初始化部分，這邊要初始化scale和shift，scale初始化為1，shift初始化為0。

Forward path部分即根據layer normalization的公式做實現，公式如下

一張含有 字型, 行, 白色, 數字 的圖片

自動產生的描述

公式中為x的mean、為x的標準差、惟為了防止分母為零而加入的一個微小常數、為scale、為shift。

1. 執行結果 :



* FeedForwardBlock :

Feed forward block會在transformer的encoder和decoder中用到，通常由堆疊多層的MLP與ReLU所構成。

1. 實現方法 :

初始化部分，初始化兩個MLP，及linear layer 🡪 ReLU 🡪 linear layer。Shape的部分輸入和輸出的形狀相同。參數初始化部分，weight都利用xavier initialization，bias都初始化為0。

Forward path部分只需將輸入送進linear layer 1 🡪 ReLU 🡪 linear layer 2即可。

1. 執行結果 :



* EncoderBlock :

Encoder block由MultiHeadAttention、FeedForward、LayerNormalization組成。在transformer的encoder中，每個encoder會接收三個輸入，分別為query、key、value。Encoder架構如下

一張含有 文字, 螢幕擷取畫面, 字型, 圖表 的圖片

自動產生的描述

關於positional encoding的部分，由於他是不可學習的，因此這邊將他視作data loader中預處理的步驟。

接著根據上面架構建立encoder。

1. 實現方法 :

利用先前建立好的MultiHeadAttention、LayerNormalization、FeedForwardBlock來建構encoder。初始化部分只要設定好這些layer即可。

Forward path部分，要特別注意的是residual的部分。第一層LayerNormalization的輸入為做一開始的輸入x，加上經過MultiHeadAttention後的結果。第二層LayerNormalization的輸入也是residual後的結果。

1. 執行結果 :



* DecoderBlock :

在比較複雜的任務中，我們會需要decoder來將encoder的輸出轉換為我們想要的序列樣式。Decoder會接收encoder的輸出以及前一個產生的結果來產生下一個結果。在訓練過程中，我們會對輸入使用mask，以防止decoder提前看到未來的輸入，在預測的時候則是循序的處理資料。

1. Get\_subseqeunt\_mask :

Mask主要是為了避免decoder提前得知未來資訊。根據masked self attention的架構可以得知，mask應該為一個上三角矩陣，因此這邊先去產生一個全部為1的矩陣，再產生出一個全部元素為1的下三角矩陣，將兩個相減得到全部為1的上三角矩，再將矩陣中數值轉為布林值。測試結果如下



1. Decoder實現方法 :

Decoder架構如下

一張含有 文字, 螢幕擷取畫面, 圖表, 行 的圖片

自動產生的描述

初始化部分需要初始化兩個MultiHeadAttention、三個LayerNormalization和一個FeedForwardBlock。

Forward path部分根據架構圖實現，要特別注意的是第一個MultiHeadAttention為masked self attention。

1. Decoder檢查結果 :



1. Data loader

* Simple positional encoding :

由於transformer中沒有內建的位置資訊，因此需要透過positional encoding來未輸入數據加上位置資訊。這些位置資訊通常會與輸入做相加，因此形狀應該與輸入相同。這邊時做最簡單的positional encoding，即將序列中第n個元素分配給n/k的值。

1. 實現方法 :

透過torch.linspace產生出0 ~ 1 – 1/k，等間距的張量，再透過repeat的方式將其擴展以及permute讓形狀符合輸入的形狀。

1. 檢查結果 :



* Sinusoid positional encoding :

Simple positional encoding的缺點在於當序列長度增加時，兩個連續位置編碼的差異會變很小，導致編碼的作用下降。因此改為利用sin、cos來做positional encoding。

Sinusoid positional encoding的公式如下

一張含有 文字, 字型, 筆跡, 印刷術 的圖片

自動產生的描述

其中，M為transformer的embedding dimension。

1. 實現方法 :

根據上述公式完成即可。

1. 檢查結果 :



1. Using transformer on the toy dataset

* Implement the transformer model :

初始化部分我們只需要實作embedding layer的部分。Embedding layer可以透過呼叫torch.nn內建的embedding layer來實現，其主要功能為將vocab\_len映射到emb\_dim。

Forward path的部分，首先將q\_emb\_inp送進encoder，接著透過先前建立好的get\_subsequent\_mask來建構mask，並將encoder的ouput、a\_emb\_inp和mask一起送進decoder中。

* Overfitting the model using small data :

使用小型資料嘗試讓模型overfit，以確認模型能夠正常作訓練。訓練200個epochs後結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 印刷術 的圖片

自動產生的描述



最後準確度為1，表示模型有順利overfit。

* Fit the model using complete data :

利用完整資料及來訓練模型。這部分由於作業建構的模型表現不是很好，因此我嘗試修改一些參數來提升模型表現。

首先我修改模型架構，將embedding dimension與feedforward dimension提升到256、512，number of head提升至8，number of encoder layer與number of decoder layer設定為4。

此外我也將positional encoding改為使用sinusoid positional encoding。

訓練300個epochs後結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 印刷術 的圖片

自動產生的描述



最終模型有85.16 %的準確度。

* Visualize and inference : model in action

將剛才訓練完成的模型拿來作預測，部分結果如下

 

 

 

可以看到模型是能夠正確計算出結果。

1. 額外嘗試

* Attention is all you need :

這次作業基本上都圍繞在”Attention is all you need”這篇論文，因此這邊將這篇論文內容做簡單的整理。

Transformer架構主要由encoder、decoder堆疊而成，在論文中，作著使用六層的encoder與decoder來實現，encoder-decoder架構如下。

一張含有 文字, 圖表, 螢幕擷取畫面, 方案 的圖片

自動產生的描述

首先看到encoder的部分，輸入會先經過embedding轉換為向量，接著再與positional encoding做相加以涵蓋位置資訊。Positional encoding部分，論文採用sinusoid encoding的方式做編碼，公式部分已在實作sinusoid positional encoding時說明過，這邊主要討論為甚麼要用sin、cos來編碼。選用sin、cos來編碼的主要原因在於sin、cos可以表示兩個向量之間的線性關係，並呈現不同詞語間的相堆位置，此外，sin、cos比較不會受限於序列長度的限制。將input embedding與positional encoding相加後，會進入multi-head attention，關於self attention部分，論文採用scaled dot product attention，其公式與架構如下

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述 一張含有 字型, 文字, 白色, 行 的圖片

自動產生的描述

Multi-head attention與self attention差異只有在multi-head attention會先將query、key、value拆分成多個低維度向量，架構與公式如下

一張含有 文字, 螢幕擷取畫面, 圖表 的圖片

自動產生的描述 一張含有 文字, 字型, 收據, 白色 的圖片

自動產生的描述

論文中採用h = 8，而。經過multi-head attention後會進入residual connection和layer normalization，接著再進入Feed forward，Feed forward包含兩層的線性轉換層與一層ReLU夾在中間，forward path為 **。

Decoder部分主要有加入masked multi-head attention，實現方式與作用已經在實作部分說明。

訓練部分，optimizer採用Adam，參數配置為。learning rate部分，採用下列公式在訓練過程中調整learning rate大小

Normalization部分，在與sub-layer相加和normalization之前，以及在positional encoding與input embedding相加時，都有使用dropout。此外在訓練過程中也有使用label smoothing。

* Layer Normalization :

在transformer架構中所使用的normalization為layer normalization。Layer normalization會獨立對每層的所有特徵進行正規化，代表說每層的activation的mean和variance會被獨立計算，然後activation output會被縮放和平移，以達到normal distribution。Layer normalization的公式如下



1. 優點 :

* 減少內部covariance偏移的影響，提高訓練的穩定性。
* 緩解梯度消失的問題。
* 使網路對權重與bias的初始值不那麼敏感。

1. 缺點 :

* 增加計算成本。
* Layer normalization中兩個可學習的超參數scale和shift對網路表現有很大的影響。
* 在某些情況下，layer normalization會放大數劇中的noise。
* 與batch normalization相比，靈活性比較低。

1. 與batch normalization比較 :

* Layer normalization是獨立對每層的所有特徵進行正規化，而batch normalization則是對mini-batch正規化。
* Layer normalization需要的記憶體空間較少，因為他不需要對整個mini-batch儲存mean和variance。
* 對於RNN或是transformer來說，layer normalization會有比較好的表現，因為layer normalization不會受batch size的影響。

1. Reference

[1] Google “ Attention Is All You Need” <https://arxiv.org/pdf/1706.03762.pdf>

[2] Sujatha Mudadla “Layer Normalization” <https://medium.com/@sujathamudadla1213/layer-normalization-48ee115a14a4>

[3] Hunter Phillips “Positional Encoding” <https://medium.com/@hunter-j-phillips/positional-encoding-7a93db4109e6>

[4] Minhajul Hoque “A comprehensive overview of transformer-based models : encoders, decoders, and more” <https://medium.com/@minh.hoque/a-comprehensive-overview-of-transformer-based-models-encoders-decoders-and-more-e9bc0644a4e5>

[5] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. <https://chat.openai.com/>