Dimensional Sentiment Analysis - Implementation

簡報製作：陽明交通大學 林孜彌 博士生
E-mail: ltmdegf4.ii12@nycu.edu.tw

目錄

Colab 平台介紹

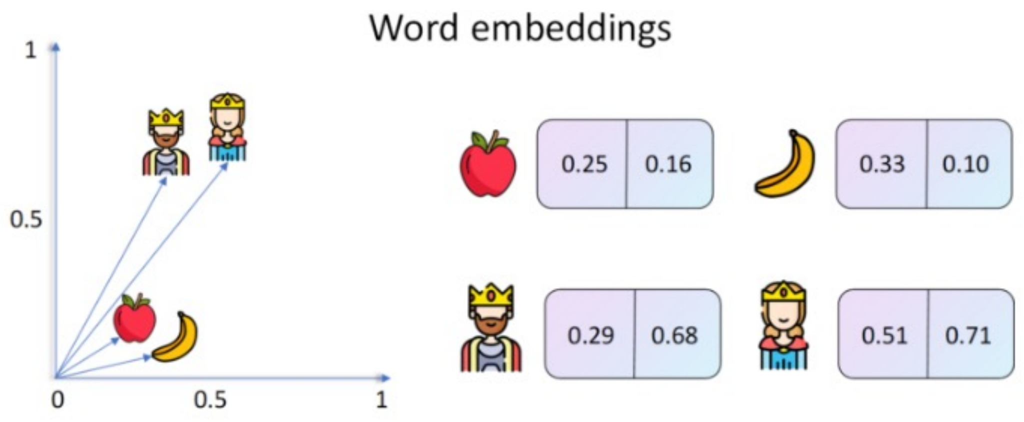
Colab

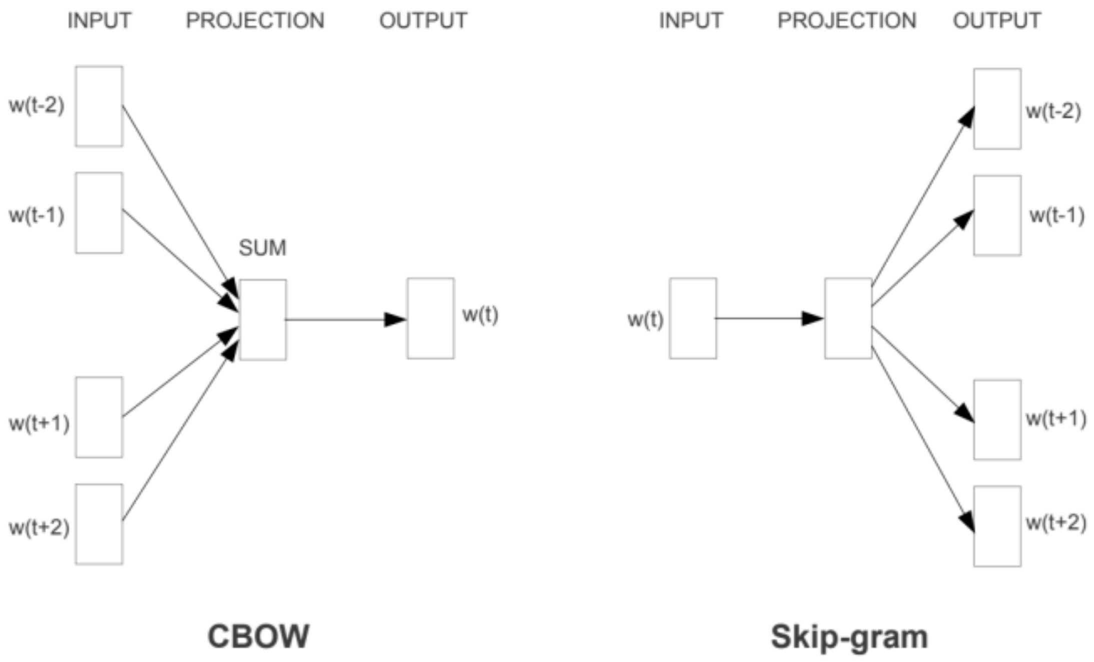
- Google Colaboratory (Colab) 是 Google 提供的免費服務
- 基於 Jupyter Notebook 的開發環境
- 提供免費的運算資源 (GPU、TPU)
- 有記憶體與執行時間的限制

Dimensional VA
Regional CNN-LSTM 模型實作

Word2Vec 介紹

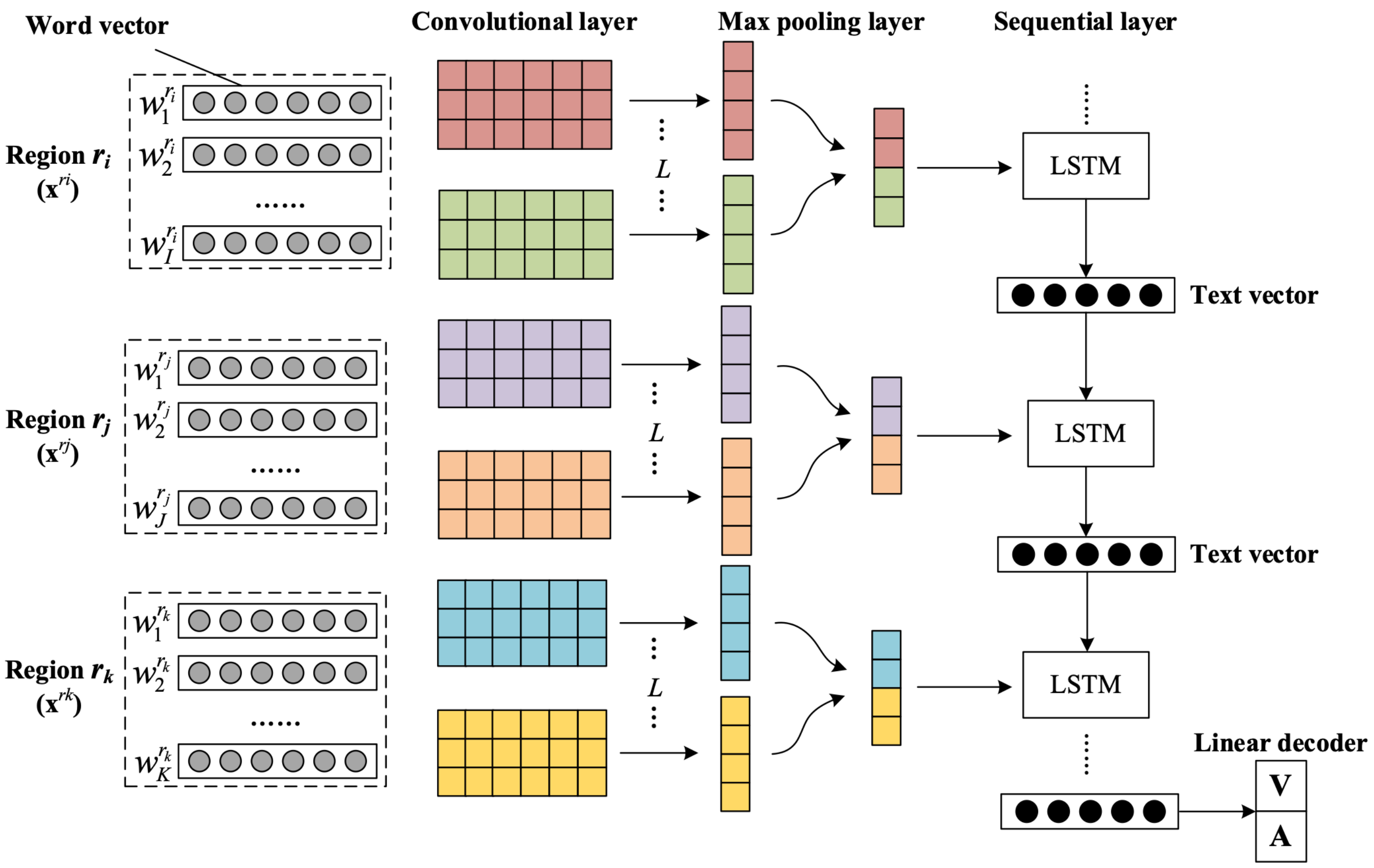
- Word2Vec 是將文字轉換成高維向量的技術
- 利用上下文來學習單詞之間的語意關係，將有相似語意的單詞映射到相近的向量空間





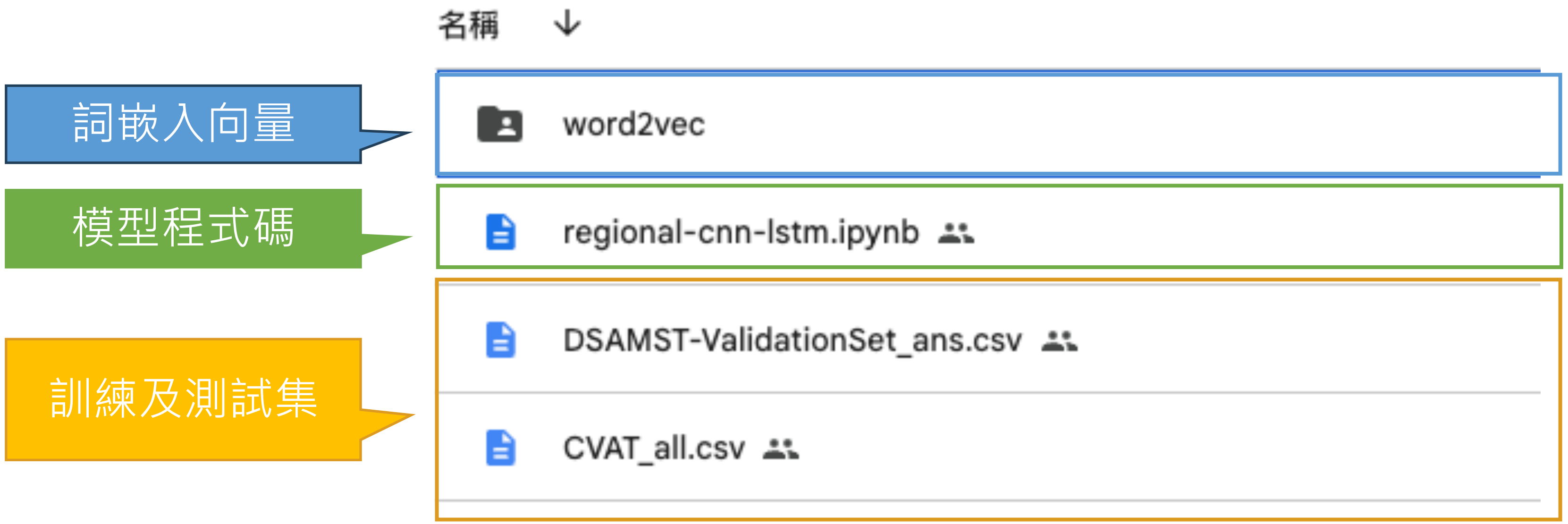
https://towardsdatascience.com/deep-learning-for-nlp-word-embeddings-4f5c90bcdab5/
Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*

Regional CNN-LSTM 架構介紹



Jin Wang, Liang-Chih Yu, K. Robert Lai, and Xuejie Zhang. 2016. [Dimensional Sentiment Analysis Using a Regional CNN-LSTM Model](). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 225–230, Berlin, Germany. Association for Computational Linguistics.

Regional CNN-LSTM 資料夾介紹



Regional CNN-LSTM實作

- Step 1: 掛載Google雲端硬碟至 Colab，並且設置模型及詞嵌入向量存放的位置

```
[1] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[2] import os
    current_dir = '/content/drive/MyDrive/tutorial/cnn-bilstm'
    print(current_dir)
    model_path = os.path.join(current_dir,'model')
    if os.path.exists(model_path) == False:
      os.makedirs(model_path)

    embedding_path = os.path.join(current_dir,'word2vec')
    if os.path.exists(embedding_path) == False:
      os.makedirs(embedding_path)
```

Regional CNN-LSTM實作

- Step 2: 安裝jieba套件進行斷詞，並使用標點符號進行斷句

```
[3] # 載入繁體中文jieba斷詞
    !pip install git+https://github.com/APCLab/jieba-tw.git
```

```
[4] import pandas as pd
    import jieba
    from tqdm import tqdm
    import numpy
    import re

    training_data = pd.read_csv(os.path.join(current_dir,'CVAT_all.csv'),sep=',')
    validation_data = pd.read_csv(os.path.join(current_dir,'DSAMST-ValidationSet_ans.csv'),sep=',')

    def data_load(data):
      x, y_valence, y_arousal = [], [], []
      with open(os.path.join(embedding_path,"segmented_corpus.txt"), "w", encoding="utf-8") as f:
        for i in tqdm(range(len(data))):
            segmented_line = " ".join(jieba.cut(data.iloc[i]['Text']))  # 使用空格分隔詞
            f.write(segmented_line + "\n")
            y_valence.append(float(data.iloc[i]['Valence']))
            y_arousal.append(float(data.iloc[i]['Arousal']))

            sentences = re.split(r'[。！？；;]', segmented_line)  # 用標點符號拆分
            sentences = [s.strip() for s in sentences if s.strip()]
            x.append([segmented_line,sentences])
      return x, y_valence, y_arousal

    x_train, y_train_valence, y_train_arousal = data_load(training_data)
    x_valid, y_valid_valence, y_valid_arousal = data_load(validation_data)
```

Regional CNN-LSTM實作

- Step 3: 安裝 gensim 套件以訓練word2vec模型，用來初始化詞向量

詞嵌入向量訓練 -> 近期版本更新，需處理相容問題

```
!pip install gensim

from gensim.models import word2vec

sentences = word2vec.LineSentence(os.path.join(embedding_path,"segmented_corpus.txt"))
model = word2vec.Word2Vec(sentences, vector_size=300,min_count=1,epochs=10)

#保存模型，供日後使用
model.wv.save_word2vec_format(os.path.join(embedding_path,'word2vec_embedding.txt'), binary=False)
```

Regional CNN-LSTM實作

- Step 4: 將資料轉換成模型所需要的格式

```python
from collections import defaultdict
import ast

def build_data_train_valid(x_train, y_train_valence, y_train_arousal, x_valid, y_valid_valence, y_valid_arousal):
    revs = []
    vocab = defaultdict(float)

    for i in range(len(x_train)):
        orig_rev, region_rev = x_train[i][0],x_train[i][1]
        words = set(orig_rev.split())
        for word in words:
            vocab[word] += 1
        datum = {
            'valence': y_train_valence[i],
            'arousal': y_train_arousal[i],
            'text': orig_rev,
            'region_text': region_rev,
            'num_regions': len(region_rev),
            'num_words': len(orig_rev.split()),
            'split': 'train'
        }
        revs.append(datum)
    print('Train Done')

    for i in range(len(x_valid)):
        orig_rev, region_rev = x_valid[i][0],x_valid[i][1]

        words = set(orig_rev.split())
        for word in words:
            vocab[word] += 1
        datum = {
            'valence': y_valid_valence[i],
            'arousal': y_valid_arousal[i],
            'text': orig_rev,
            'region_text': region_rev,
            'num_regions': len(region_rev),
            'num_words': len(orig_rev.split()),
            'split': 'valid'
        }
        revs.append(datum)
    print('Validation Done')

    return revs, vocab
```

Regional CNN-LSTM實作

- Step 5: 轉換資料並進行簡單的統計

```
[8] revs, vocab = build_data_train_valid(x_train, y_train_valence, y_train_arousal, \
                                         x_valid, y_valid_valence, y_valid_arousal)
```

```
[9] import numpy as np
    import pandas as pd
    max_l = np.max(pd.DataFrame(revs)['num_words'])
    mean_l = np.mean(pd.DataFrame(revs)['num_words'])
    print('data loaded!')
    print('number of sentences: ' + str(len(revs)))
    print('vocab size: ' + str(len(vocab)))
    print('max sentence length: ' + str(max_l))
    print('mean sentence length: ' + str(mean_l))
```

```
data loaded!
number of sentences: 3963
vocab size: 20316
max sentence length: 226
mean sentence length: 38.40903356043401
```

Regional CNN-LSTM實作

- Step 6: 將訓練好的詞向量建構成詞向量矩陣以及對應的索引

```
[10] def get_W(word_vecs, embedding_dim=300):
         """
         Get word matrix. W[i] is the vector for word indexed by i
         """
         vocab_size = len(word_vecs)
         word_idx_map = dict()
         # position 0 was not used
         W = np.zeros(shape=(vocab_size+2, embedding_dim), dtype=np.float32)
         W[0] = np.zeros((embedding_dim, ))
         W[1] = np.random.uniform(-0.25, 0.25, embedding_dim)

         i = 2
         for word in word_vecs:
             try:
                 W[i] = word_vecs[word]
                 word_idx_map[word] = i
                 i = i + 1
             except:
                 continue
         return W, word_idx_map
```

Regional CNN-LSTM實作

- Step 7: 將詞嵌入向量進行轉換並將訓練資料以及詞嵌入矩陣等等存成.pickle檔

```
[11] import pickle
     import os

     embeddig_path = os.path.join(embedding_path,'word2vec_embedding.txt')
     word2vec = pd.read_csv(embeddig_path, sep=" ", quoting=3, header=None, index_col=0, skiprows=1)
     embeddings_index = {key: val.values for key, val in word2vec.T.items()}
     w2v = embeddings_index

     print('word embeddings loaded!')
     print('num words in embeddings: ' + str(len(w2v)))

     W, word_idx_map = get_W(w2v)
     print('extracted index from embeddings! ')

     pickle_file = os.path.join(embedding_path, f'tutorial.pickle3')
     pickle.dump([revs, W, word_idx_map, vocab, max_l], open(pickle_file, 'wb'))
```

```
word embeddings loaded!
num words in embeddings: 5122
extracted index from embeddings!
```

Regional CNN-LSTM實作

- Step 8: 載入訓練模型所需要的套件
  - 橘色: 建立及訓練模型套件
  - 藍色: 評估指標套件

```
[ ] #載入所需要的函式
    import numpy as np
    import tensorflow as tf
    from keras.models import Model
    from keras.layers import Convolution1D, MaxPooling1D, Dense, Flatten, TimeDistributed
    from keras.layers import Embedding, LSTM, Bidirectional, Input
    from keras.preprocessing.sequence import pad_sequences
    from keras.callbacks import ModelCheckpoint
    from keras.models import load_model

    from sklearn.metrics import mean_absolute_error
    from scipy.stats import pearsonr
```

Regional CNN-LSTM實作

- Step 9: 讀取.pickle檔案，並確認GPU的使用以及模型訓練參數

```
[13] pickle_file = os.path.join(embedding_path,'tutorial.pickle3')
     revs, W, word_idx_map, vocab, max_len = pickle.load(open(pickle_file, 'rb'))
     id2emb = W

     print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
     print(tf.test.is_built_with_cuda())

     option = 'arousal'

     batch_size = 32
     epochs = 10
     hidden_dim = 256

     kernel_size = 3
     nb_filter = 256

     max_l = 512
     max_r = 50
```

```
Num GPUs Available:  1
True
```

Regional CNN-LSTM實作

- Step 8: 將文字資料轉換成詞嵌入向量

```
[14] def make_idx_data(revs, word_idx_map, maxlen, max_region):
         """
         Transforms sentences into a 2-d matrix.
         """
         X_train, X_valid, y_train, y_valid = [], [], [], []
         for rev in revs:
             sent = np.zeros((max_region, maxlen))
             region_text = rev['region_text']
             for i, region in enumerate(region_text):
                 if i >= max_region:
                     continue
                 for j, word in enumerate(region.split()):
                     if j == maxlen:
                         break
                     if word in word_idx_map:
                         sent[i, j] = word_idx_map[word]
                     else:
                         sent[i, j] = 1
             y = rev[option]

             if rev['split'] == 'train':
                 X_train.append(sent)
                 y_train.append(y)
             elif rev['split'] == 'valid':
                 X_valid.append(sent)
                 y_valid.append(y)

         X_train = np.array(X_train, dtype='int')
         X_valid = np.array(X_valid, dtype='int')
         y_train = np.array(y_train)
         y_valid = np.array(y_valid)

         return [X_train, X_valid, y_train, y_valid]
```

Regional CNN-LSTM實作

- Step 9: 資料轉換並進行統計

```
[15] X_train, X_valid, y_train, y_valid = make_idx_data(
                        revs, word_idx_map, maxlen=max_l, max_region=max_r)

     n_train_sample = X_train.shape[0]
     print("n_train_sample [n_train_sample]: %d" % n_train_sample)

     n_valid_sample = X_valid.shape[0]
     print("n_valid_sample [n_valid_sample]: %d" % n_valid_sample)

     len_region = X_train.shape[1]     # 200
     print("len_region [len_region]: %d" % len_region)

     len_sentence = X_train.shape[2]     # 200
     print("len_sentence [len_sentence]: %d" % len_sentence)

     max_features = W.shape[0]
     print("num of word vector [max_features]: %d" % max_features)

     num_features = W.shape[1]               # 400
     print("dimension num of word vector [num_features]: %d" % num_features)
```

```
n_train_sample [n_train_sample]: 2969
n_valid_sample [n_valid_sample]: 994
len_region [len_region]: 50
len_sentence [len_sentence]: 512
num of word vector [max_features]: 5124
dimension num of word vector [num_features]: 300
```

Regional CNN-LSTM實作

- Step 10: 建立模型

```
[16] # 建立 Keras Model
sentence_input = Input(shape=(max_l,), dtype='int32')
embedded_sequences = Embedding(input_dim=max_features, output_dim=num_features, input_length=max_l, weights=[W], trainable=False) (sentence_input)
l_convolution = Convolution1D(filters=nb_filter,
                        kernel_size=kernel_size,
                        padding='valid',
                        activation='relu',
                        strides=1
                        ) (embedded_sequences)

l_maxpooling = MaxPooling1D(pool_size=2) (l_convolution)
l_cnn = Flatten() (l_maxpooling)

sentEncoder = Model(sentence_input, l_cnn)

review_input = Input(shape=(max_r, max_l), dtype='int32')
review_encoder = TimeDistributed(sentEncoder)(review_input)
l_lstm_sent = Bidirectional(LSTM(hidden_dim))(review_encoder)
preds = Dense(1, activation='linear') (l_lstm_sent)
model = Model(review_input, preds)
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 50, 512)	0
time_distributed (TimeDistributed)	(None, 50, 65280)	1,767,856
bidirectional (Bidirectional)	(None, 512)	134,219,776
dense (Dense)	(None, 1)	513

Total params: 135,988,145 (518.75 MB)
Trainable params: 134,450,945 (512.89 MB)
Non-trainable params: 1,537,200 (5.86 MB)

Regional CNN-LSTM實作

- Step 11: 模型訓練，並且儲存模型在測試集上的預測

```
[ ] model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1)
    y_pred = model.predict(X_valid, batch_size=batch_size).flatten()

    predict_file = open(os.path.join(model_path,f'{option}_predict.txt'),'w')
    for y,pred in zip(y_valid, y_pred):
      print(y,pred,file=predict_file)
    predict_file.close()
```

Regional CNN-LSTM實作

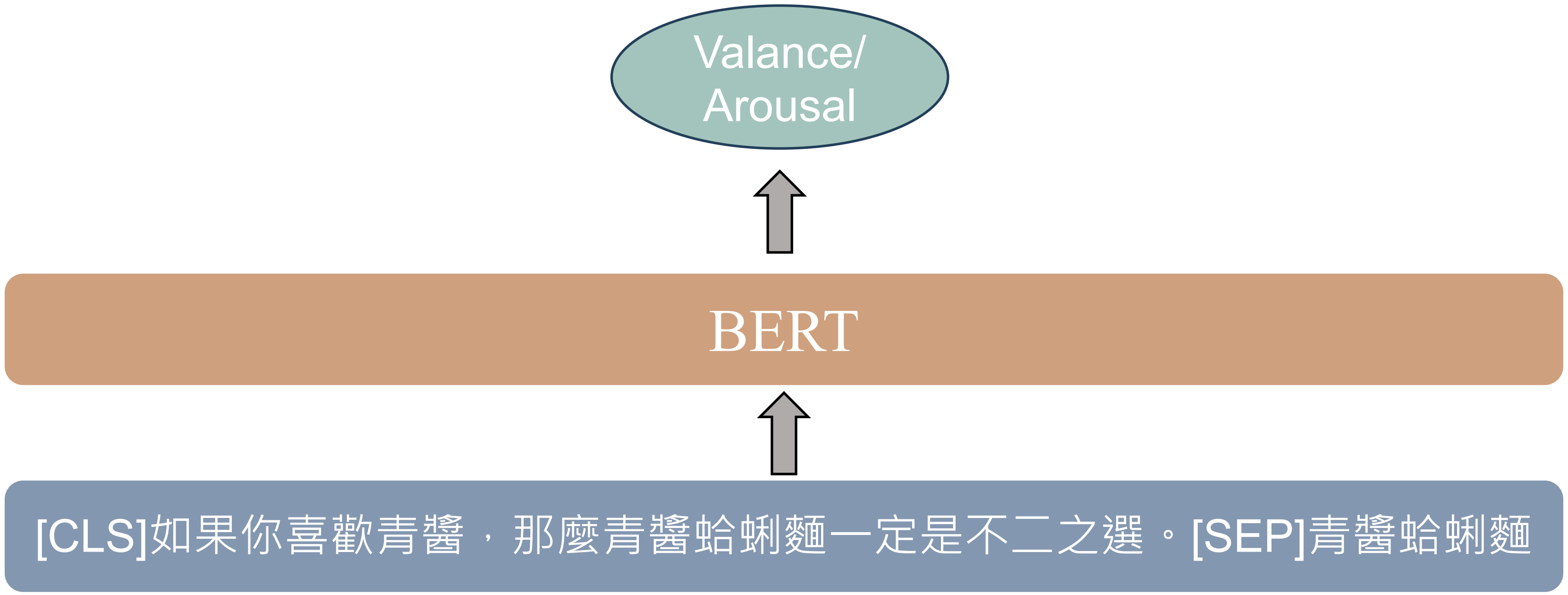
- Step 12: 定義評估程式並評估結果

```
[17] def evaluate(y_true, y_pred):
         mae = mean_absolute_error(y_true, y_pred)
         pr = pearsonr(y_true, y_pred)[0]

         print('Evaluation length: %d' % len(y_pred))
         print('MAE: %.3f' % (mae))
         print('Pearsonr: %.3f' % (pr))
         return mae,pr
```

```
mae,pr = evaluate(y_valid, y_pred)

score = open(os.path.join(model_path,f'{option}_score.txt'),'w')
score.write('MAE: %.3f' % (mae)+'\n')
score.write('Pearsonr: %.3f' % (pr)+'\n')
score.close()
```

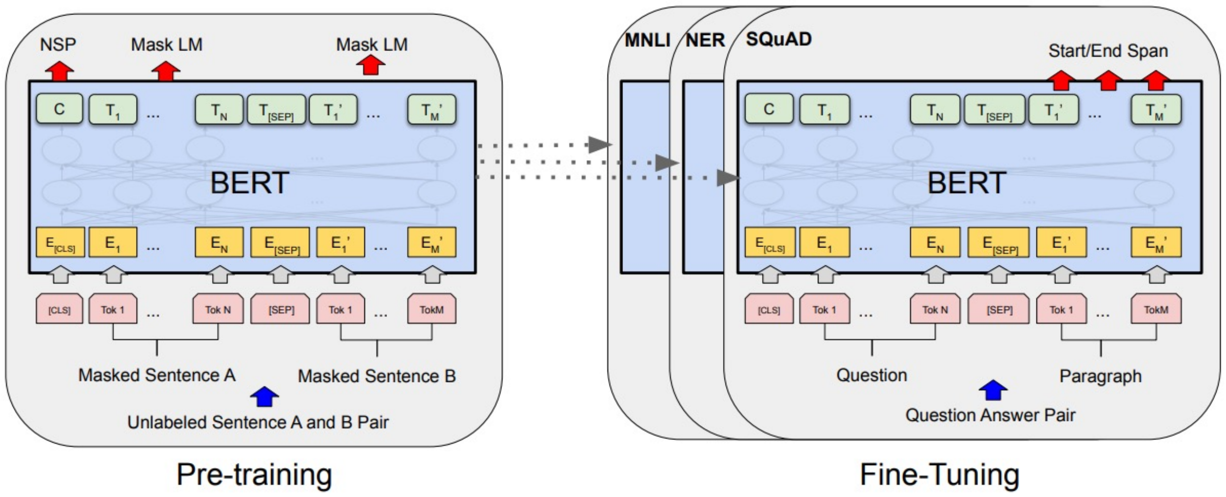
Aspect-Based Dimensional VA
BERT 實作

BERT 架構介紹


Valance/
Arousal
BERT
[CLS]如果你喜歡青醬，那麼青醬蛤蜊麵一定是不二之選。[SEP]青醬蛤蜊麵

BERT

- BERT 的全名為 Bidirectional Encoder Representations from Transformers，是一種基於Transformer架構的雙向編碼器
- BERT 的預訓練分為兩個任務:
  - 遮罩語言模型（Masking Language Modeling）： 理解詞語在上下文中的語義
  - 下一句預測（Next Sentence Prediction）： 理解句子間的邏輯關係



Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

BERT 資料夾介紹

名稱 ↑

模型程式碼

bert.ipynb

訓練及測試集

SIGHAN2024_dimABSA_Testing_Task1_Traditional.json

SIGHAN2024_dimABSA_TrainingSet1_Traditional.json

BERT實作

- Step 1: 掛載Google雲端硬碟至 Colab

```
[1] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[2] import os
    current_dir = '/content/drive/MyDrive/tutorial/bert'
    print(current_dir)
    model_path = os.path.join(current_dir,'model')
    if os.path.exists(model_path) == False:
          os.makedirs(model_path)

    /content/drive/MyDrive/tutorial/bert
```

BERT實作

- Step 2: 載入模型所需套件，分別為
  - 橘色: 建立及訓練模型套件
  - 藍色: 進度條及評估指標套件
  - 綠色： 資料處理套件

載入模型所需套件

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertModel

from tqdm import tqdm
from sklearn.metrics import mean_absolute_error
from scipy.stats import pearsonr

import pandas as pd
import json
```

BERT實作

- Step 3: 讀取資料， 並利用pandas整理成 Dataframe以利後面使用

讀取訓練資料及測試資料

```
def data_load(data_path):
  with open(data_path) as f:
    data = json.load(f)
  id_list, valence_list, arousal_list, sentence_list, aspect_list = [], [], [], [], []
  for sentence in data:
      if len(set(sentence['Aspect'])) == len(sentence['Aspect']):
          for num,aspect in enumerate(sentence['Aspect']):
              id_list.append(sentence['ID'])
              valence_list.append(float(sentence['Intensity'][num].split('#')[0]))
              arousal_list.append(float(sentence['Intensity'][num].split('#')[1]))
              aspect_list.append(sentence['Aspect'][num])
              sentence_list.append(sentence['Sentence'])
  data_dict = {'ID':id_list,'Text':sentence_list,'Aspect':aspect_list,'Valence':valence_list,'Arousal':arousal_list}
  return pd.DataFrame(data_dict)
```

```
[5] train_data = data_load(os.path.join(current_dir,'SIGHAN2024_dimABSA_TrainingSet1_Traditional.json'))
    test_data = data_load(os.path.join(current_dir,'SIGHAN2024_dimABSA_Testing_Task1_Traditional.json'))
```

BERT實作

- Step 4: 選定欲使用的 tokenizer 模型以及要針對 Valence 或是 Arousal做訓練， 並使用上一步定義好的 preprocess_data 及 DataLoader 將資料的格式作轉換

```
[7] tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
    option = 'Valence'

    input_ids, attention_mask, intensity = preprocess_data(
        data=train_data,
        tokenizer=tokenizer,
        option = option
        )

    train_dataset = TensorDataset(input_ids, attention_mask, intensity)
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

    input_ids, attention_mask, test_intensity = preprocess_data(
        data=test_data,
        tokenizer=tokenizer,
        option = option
        )
    test_dataset = TensorDataset(input_ids, attention_mask, test_intensity)
    test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

BERT實作

- Step 5: 建立BERT模型，並設置 GPU、optimizer、loss function 等等

建立 BERT 模型

```
[ ] # 定義模型
    class BERT_MLP(nn.Module):
      def __init__(self, bert_model_name='bert-base-chinese', hidden_size=128):
          super(BERT_MLP, self).__init__()
          self.bert = BertModel.from_pretrained(bert_model_name)
          self.mlp = nn.Sequential(
              nn.Linear(self.bert.config.hidden_size, hidden_size),
              nn.ReLU(),
              nn.Linear(hidden_size, 1)
          )

      def forward(self, input_ids, attention_mask):
          outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
          cls_output = outputs.last_hidden_state[:, 0, :]
          intensity = self.mlp(cls_output)
          return intensity
```

```
[ ]  #建立模型
    if torch.cuda.is_available():
        device = torch.device("cuda")
        print('There are %d GPU(s) available.' % torch.cuda.device_count())
        print('We will use the GPU:', torch.cuda.get_device_name(0))
    else:
        print('No GPU available, using the CPU instead.')
        device = torch.device("cpu")
    model = BERT_MLP().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
    loss_fn = nn.MSELoss()
    model_save_path = os.path.join(model_path,f"{option}_best_model.pth")
```

```
There are 1 GPU(s) available.
We will use the GPU: Tesla T4
model.safetensors: 100%  412M/412M [00:01<00:00, 242MB/s]
```

BERT實作

- Step 6: 訓練模型， 並將 training loss 最低的模型存下來

訓練模型

```
[ ] best_val_loss = float('inf')
    epochs = 1
    for epoch in range(epochs):
        model.train()
        train_loss = 0.0
        for input_ids, attention_mask, labels in tqdm(train_loader):
            input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(input_ids, attention_mask)
            loss = loss_fn(outputs, labels)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss /= len(train_loader)
        print(f"Epoch {epoch + 1}/{epochs}, Train Loss: {train_loss:.4f}")

        # 儲存最佳模型
        if train_loss < best_val_loss:
            best_val_loss = train_loss
            torch.save(model.state_dict(), model_save_path)
            print(f"Best model saved at epoch {epoch + 1}")
```

```
100%|██████████| 94/94 [00:26<00:00,  3.56it/s]
Epoch 1/1, Train Loss: 3.9264
Best model saved at epoch 1
```

BERT實作

- Step 7: 讀取最佳的模型並進行測試，最後計算 MAE 及 Pearson 相關係數

測試模型

```
model.load_state_dict(torch.load(model_save_path))
model.eval()
outputs = []
with torch.no_grad():
    for input_ids, attention_mask, labels in test_loader:
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)
        output = model(input_ids, attention_mask)
        outputs+=(output)

outputs = torch.cat(outputs)

predict_file = open(os.path.join(model_path,f'{option}_predict.txt'),'w')
for y,pred in zip(test_intensity, outputs):
  print(y.item(),pred.item(),file=predict_file)
predict_file.close()
```

```
<ipython-input-11-1a0abb758313>:1: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which use
  model.load_state_dict(torch.load(model_save_path))
```

```
[ ] mae = mean_absolute_error(test_intensity.squeeze().cpu().numpy(), outputs.cpu().numpy())
    pr = pearsonr(test_intensity.squeeze().cpu().numpy() , outputs.cpu().numpy())[0]

    print('MAE: %.3f' % (mae))
    print('Pearsonr: %.3f' % (pr))

    score = open(os.path.join(model_path,f'{option}_score.txt'),'w')
    score.write('MAE: %.3f' % (mae)+'\n')
    score.write('Pearsonr: %.3f' % (pr)+'\n')
    score.close()
```


Q
&
A