Sentiment (Emotion) classification Using BERT

情緒分類使用BERT神經網路

Setimental (Emotional) score:

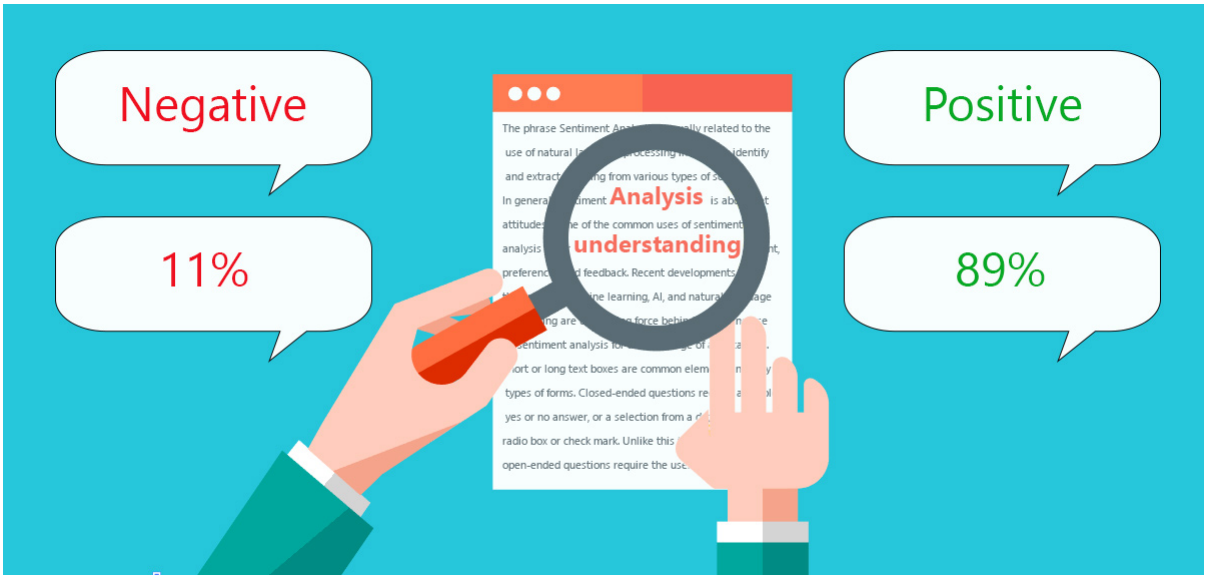
Negagive, Positive, Neutral (Objetive客觀):

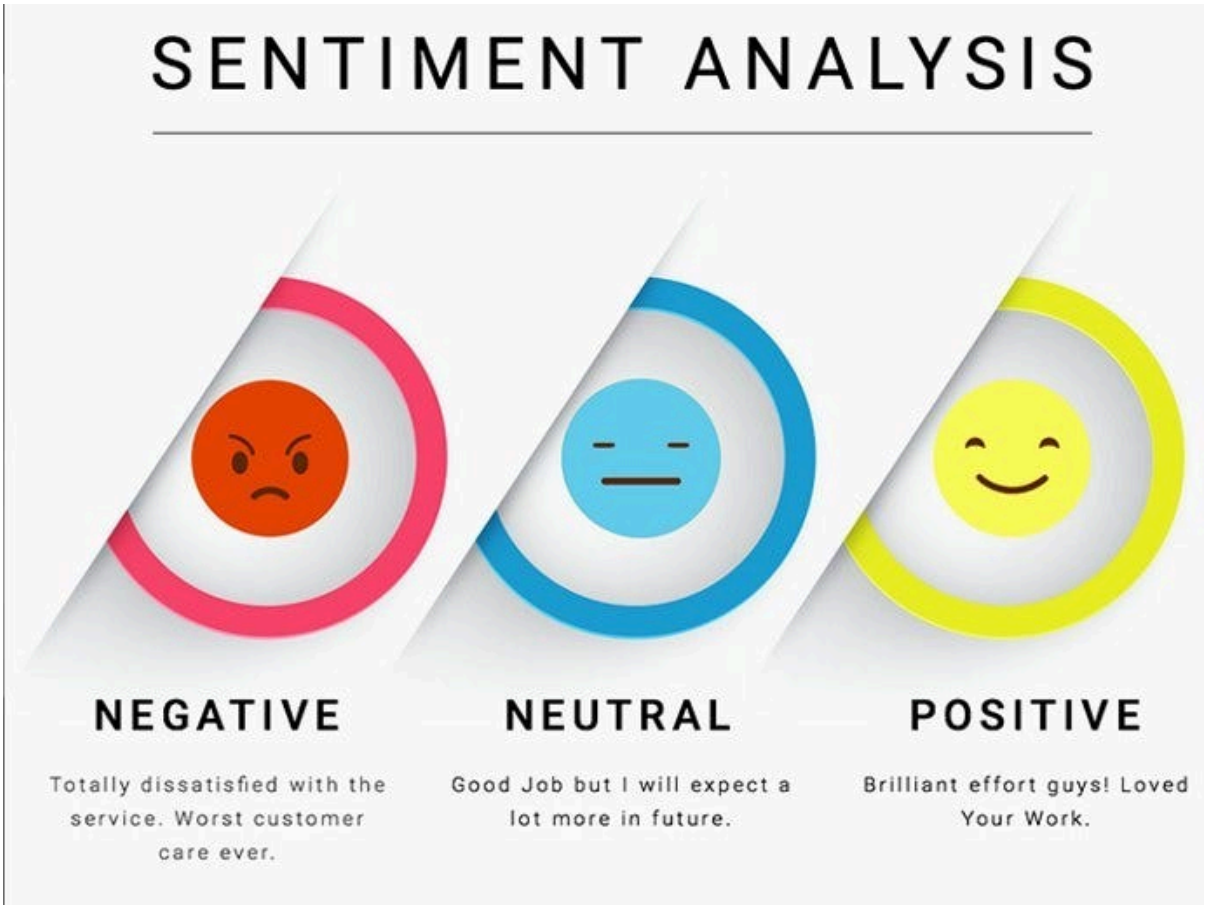
負面:0 正面:1

負面:0 正面:1 中立:2

負面:0 正面:1 中立:2 無情緒:3

Sentiment Analysis 情緒分析、情緒分類





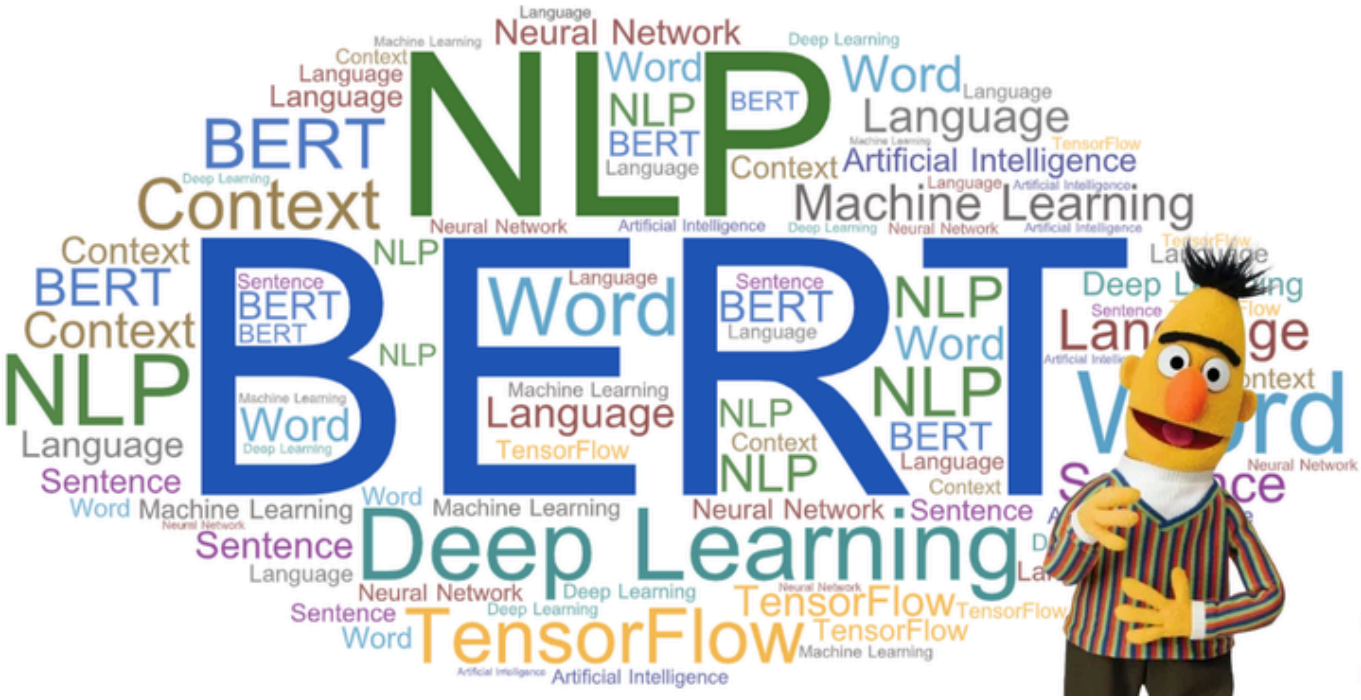
What is Sentiment Analysis?

不支援的儲存格類型。按兩下即可檢查/編輯內容。

Where can we find the dataset?

BERT

Google BERT Algorithm



What is BERT?

Bidirectional Encoder Representations from Transformers (BERT) is a Natural Language P

* Masked Language Modeling (MLM): 15% of the words in each sequence are replaced w
* Next Sentence Prediction (NSP): the model receives pairs of sentences as input an

BERT is deeply bidirectional, which means that it can learn the context of a word

For further details, you might want to read the original BERT paper.

什麼是 BERT?

模型來自 Transformers 的雙向編碼器表示 (BERT)是 Google Research 在 2018 年提出的自然語言

大量的訓練數據

33 億字的海量數據集

BERT 專門針對 Wikipedia（約 2.5B 字）和 Google 的 BooksCorpus（約 8 億字）進行了

在這麼大的數據集上訓練需要很長時間。BERT 的訓練得益於新穎的 Transformer 架構，並通過使

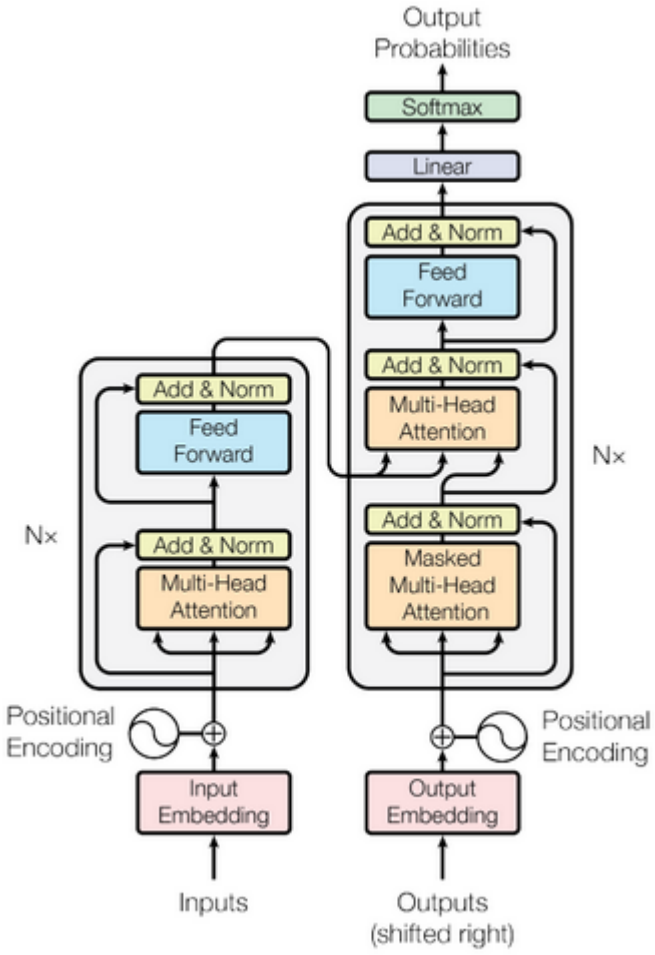
* 掩碼語言建模 (MLM)：每個序列中 15% 的單詞被替換為一個[MASK]標記。然後，該模型嘗試根據非蒙面
* Next Sentence Prediction (NSP)：模型接收成對的句子作為輸入，並學習預測第二個句子是否是原始

BERT 是深度雙向的，這意味著它可以根據輸入序列中包含的所有信息來學習單詞的上下文，同時考慮前後標

有關更多詳細信息，您可能需要閱讀原始BERT 論文。

source: https://riccardo-cantini.netlify.app/post/bert_text_classification/

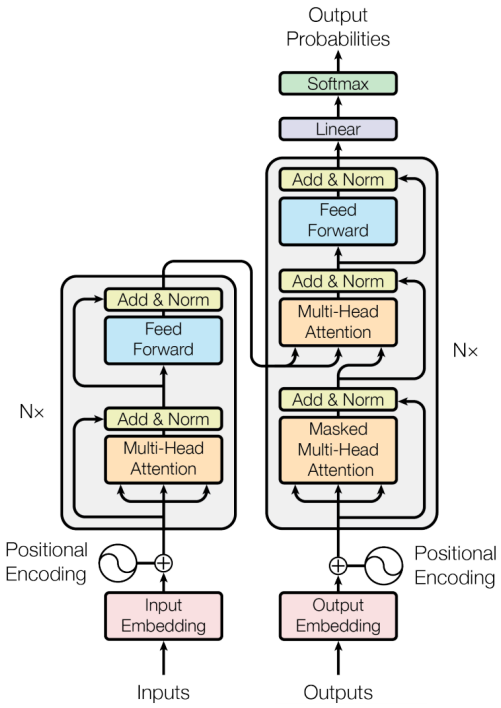
Transformers變形金剛(變壓器)模型



Bert and GPT are parts of Transformer

BERT

Encoder



GPT

Decoder

Encoders and Decoders As mentioned, there are encoders and decoders. BERT uses encoders only, GTP uses decoders only. Both options understand language including syntax and semantics. Especially the next generation of large language models like GPT with billions of parameters do this very well.

The two models focus on different scenarios. However, since the field of foundation models is evolving, the differentiation is often fuzzier.

BERT (encoder): classification (e.g., sentiment), questions and answers, summarization, named entity recognition GPT (decoder): translation, generation (e.g., stories) The outputs of the core models are different:

BERT (encoder): Embeddings representing words with attention information in a certain context GPT (decoder): Next words with probabilities Both models are pretrained and can be reused without intensive training. Some of them are available as open source and can be downloaded from communities like Hugging Face, others are commercial. Reuse is important, since trainings are often very resource intensive and expensive which few companies can afford.

The pretrained models can be extended and customized for different domains and specific tasks. Layers can sometimes be reused without modifications and more layers are added on top. If layers need to be modified, the new training is more expensive. The technique to customize these models is called Transfer Learning, since the same generic model can easily be transferred to other domains.

Source

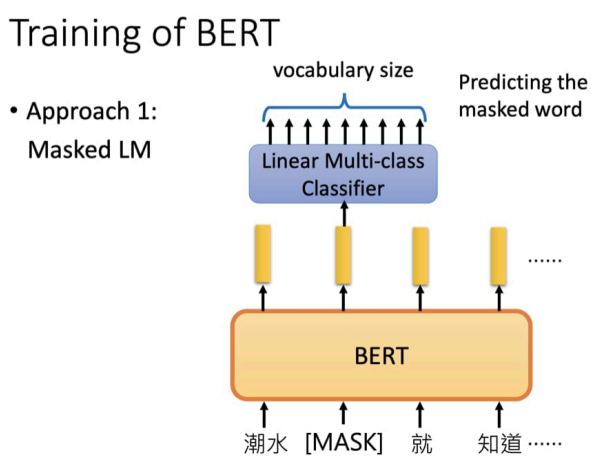
Transfor Learning (遷移學習), Finetune from the pretrained model (微調一個預訓練模型)

預訓練

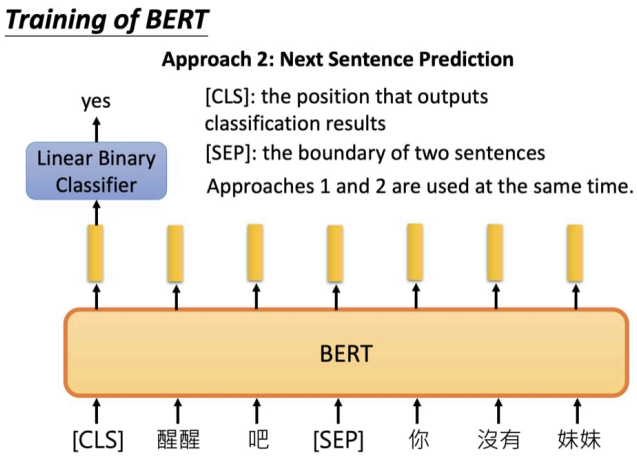
克漏字填空（Mask Language Model） Masked Language Model（MLM） 句子丟進去模型進行訓練，會把一些字詞遮蓋(Mask)起來，讓模型去預測被遮蓋起來的字詞是什麼。

下文預測（Next Sentence Prediction） Next Sentence Prediciton (NSP)。預測下一個可能的句子是什麼。(為了要解決文本生成，或是 QA 的任務)

預訓練任務 1：克漏字填空



預訓練任務 2：下個句子預測



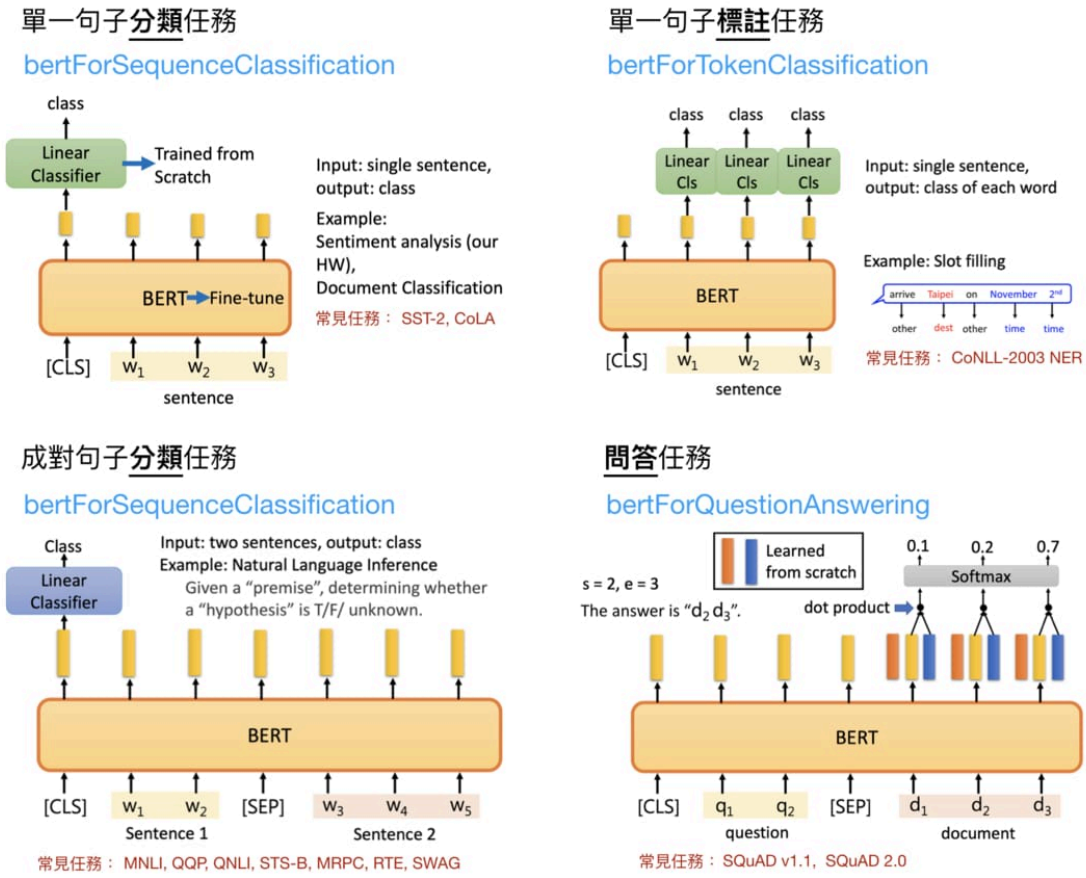
微調

將一個大型的預訓練模型拿來加以運用在解決其他領域的問題上，加上自己的資料再加以微調預訓練模型，變成一個可以解決你的問題的模型，可以節省成本。

例如:將Bert模型施以在訓練，使用自己準備的情緒語料，變成一個情緒分類器，這就是在做微調的工作。這樣就是一種遷移學習。

單一句子分類、單一句子標註、成對句子預測，以及問答任務。

來源:台大李宏毅老師講義

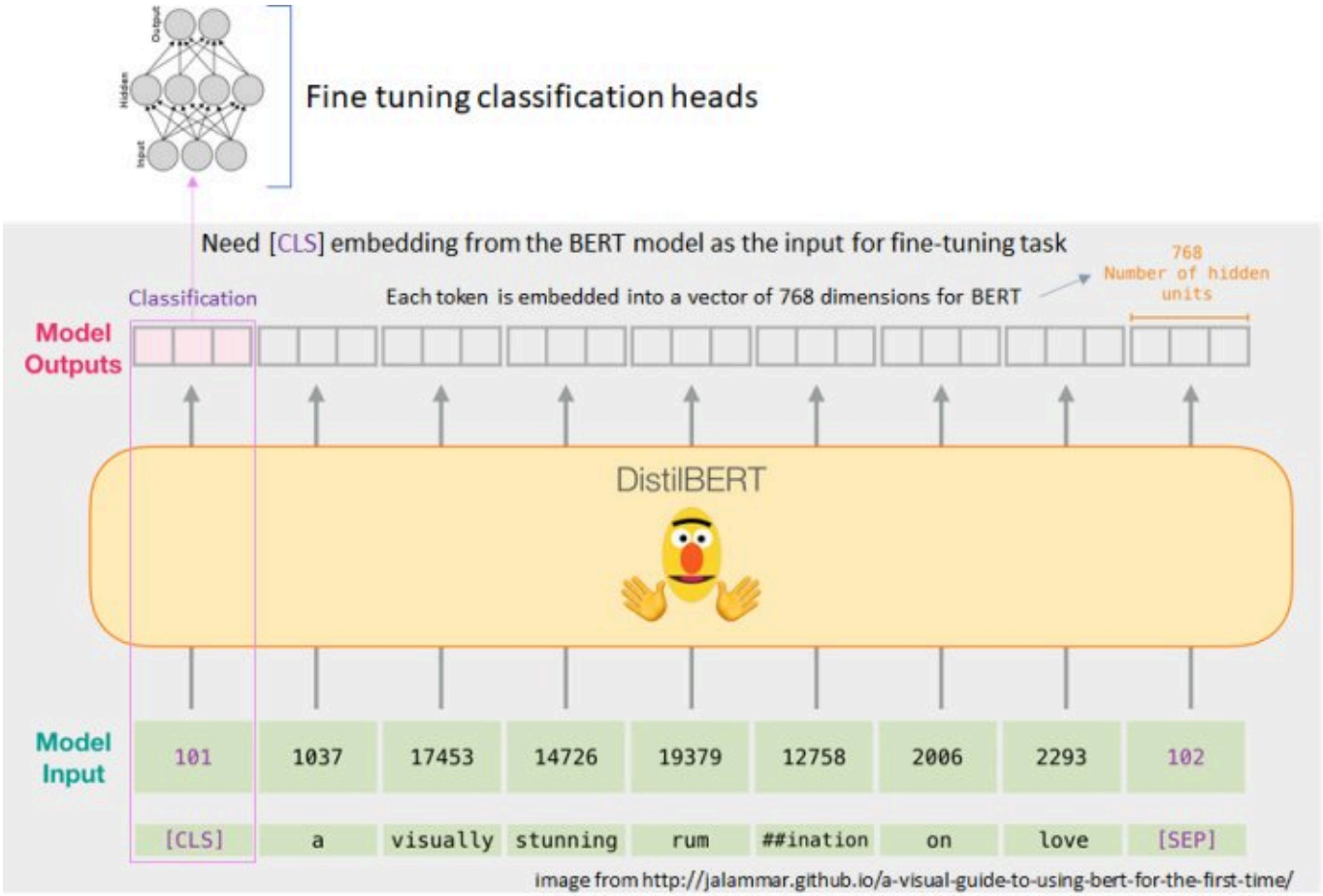


BERT Input and Output (輸入與輸出)

BERT的輸出有兩項:last_hidden_state, pooler_output

Bert input and output

Bert input and output is as follows:



Supervised Learning for text classification

A text classification procedure is composed of following components:

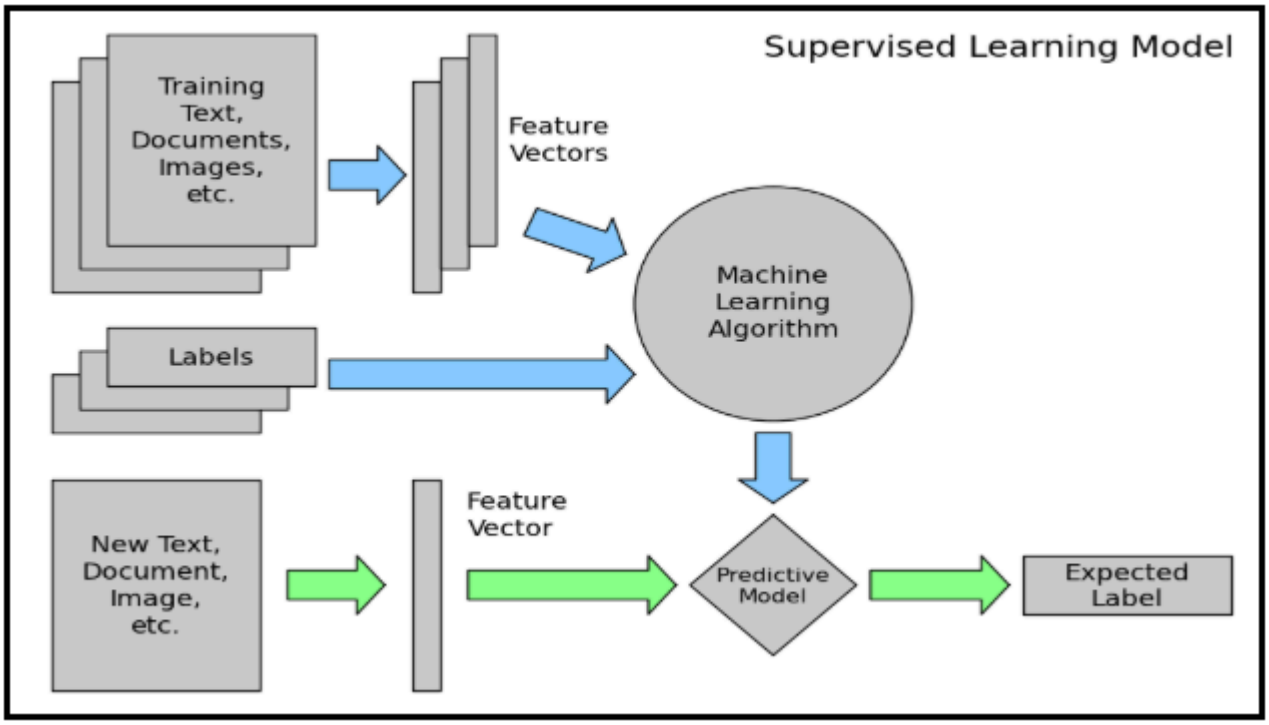
Training text: It is the input text through which our supervised learning model is able to learn and predict the required class.

Feature Vector: A feature vector is a vector that contains information describing the characteristics of the input data.

Labels: These are the predefined categories/classes that our model will predict.

ML Algorithm: It is the algorithm through which our model is able to deal with text classification (In our case : CNN, RNN)

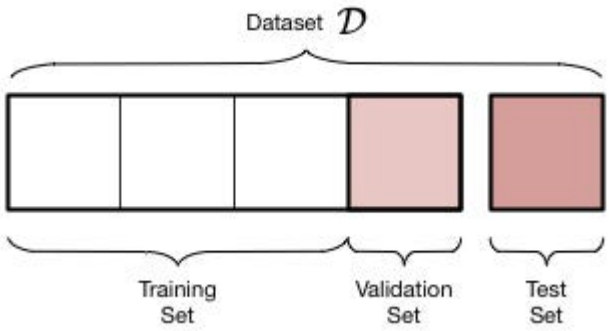
Predictive Model: A model trained on the dataset can perform label predictions.



Dataset for Model Training, Validation and Testing

不支援的儲存格類型。按兩下即可檢查/編輯內容。

Slicing a single data set into three subsets.



Main steps for training machine learning models

不支援的儲存格類型。按兩下即可檢查/編輯內容。

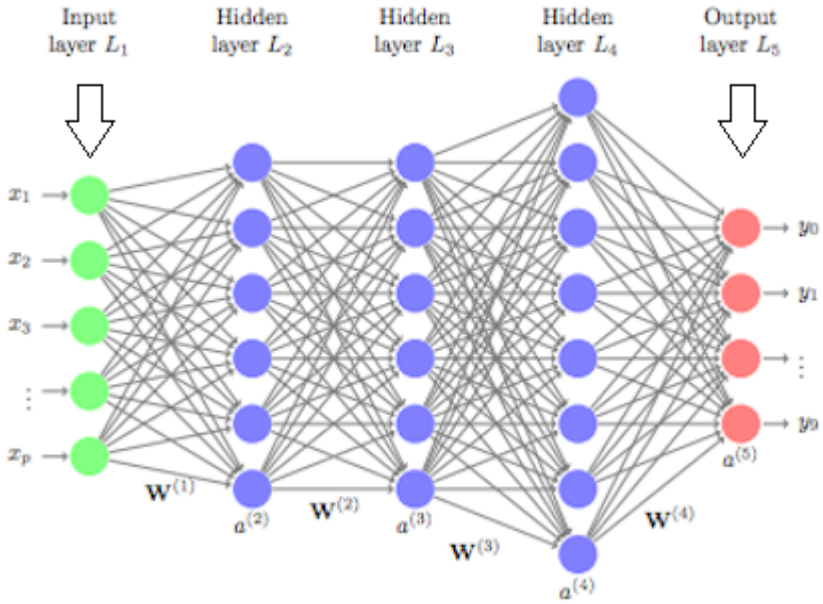
Word Embedding

Pretrained Embedding:
https://keras-cn.readthedocs.io/en/latest/blog/word_embedding/
http://ben.bolte.cc/blog/2016/gensim.html

https://gist.github.com/codekansas/15b3c2a2e9bc7a3c345138a32e029969
https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html

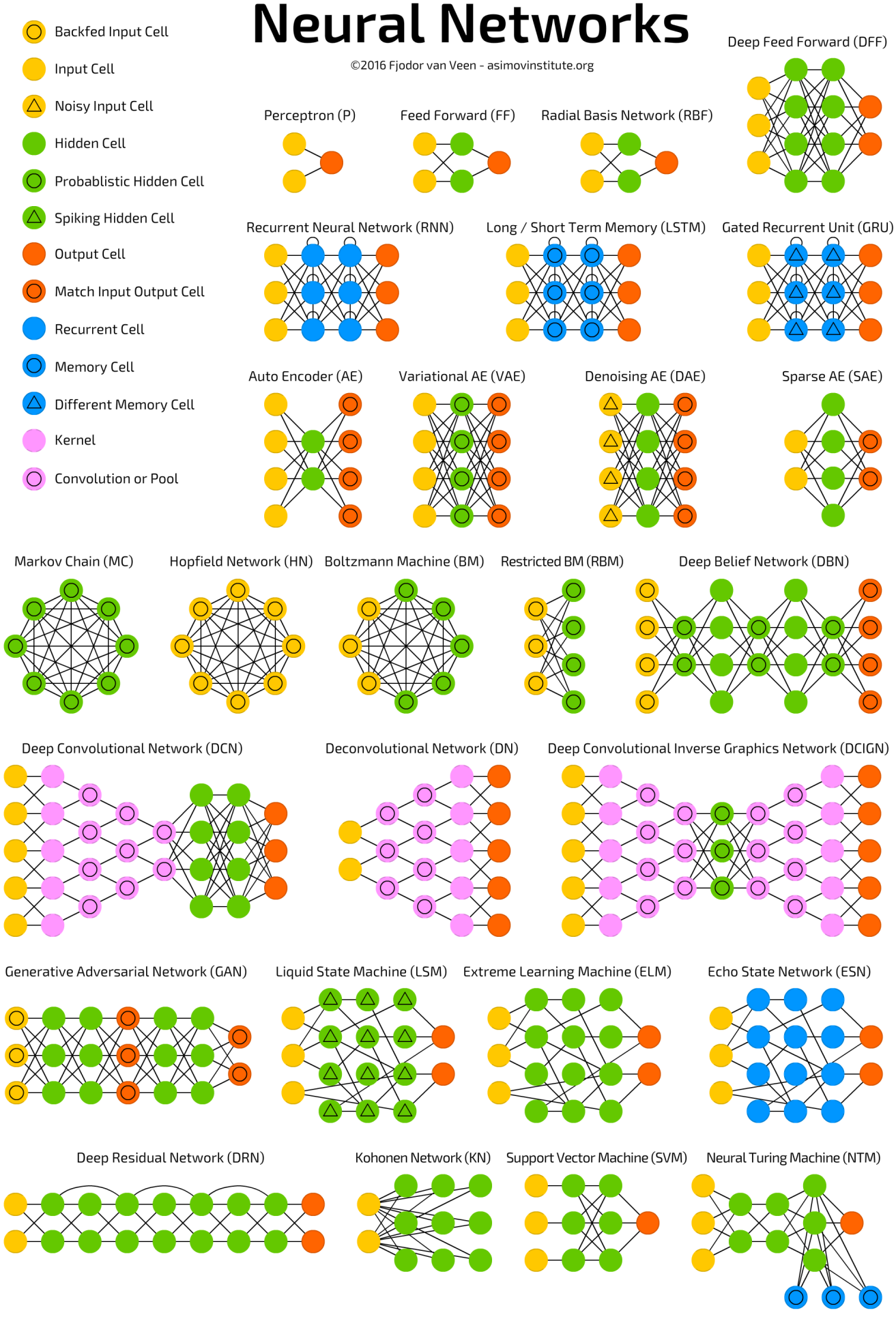
開始使用 AI 編寫或生成程式碼。

Various Deep Learning Model

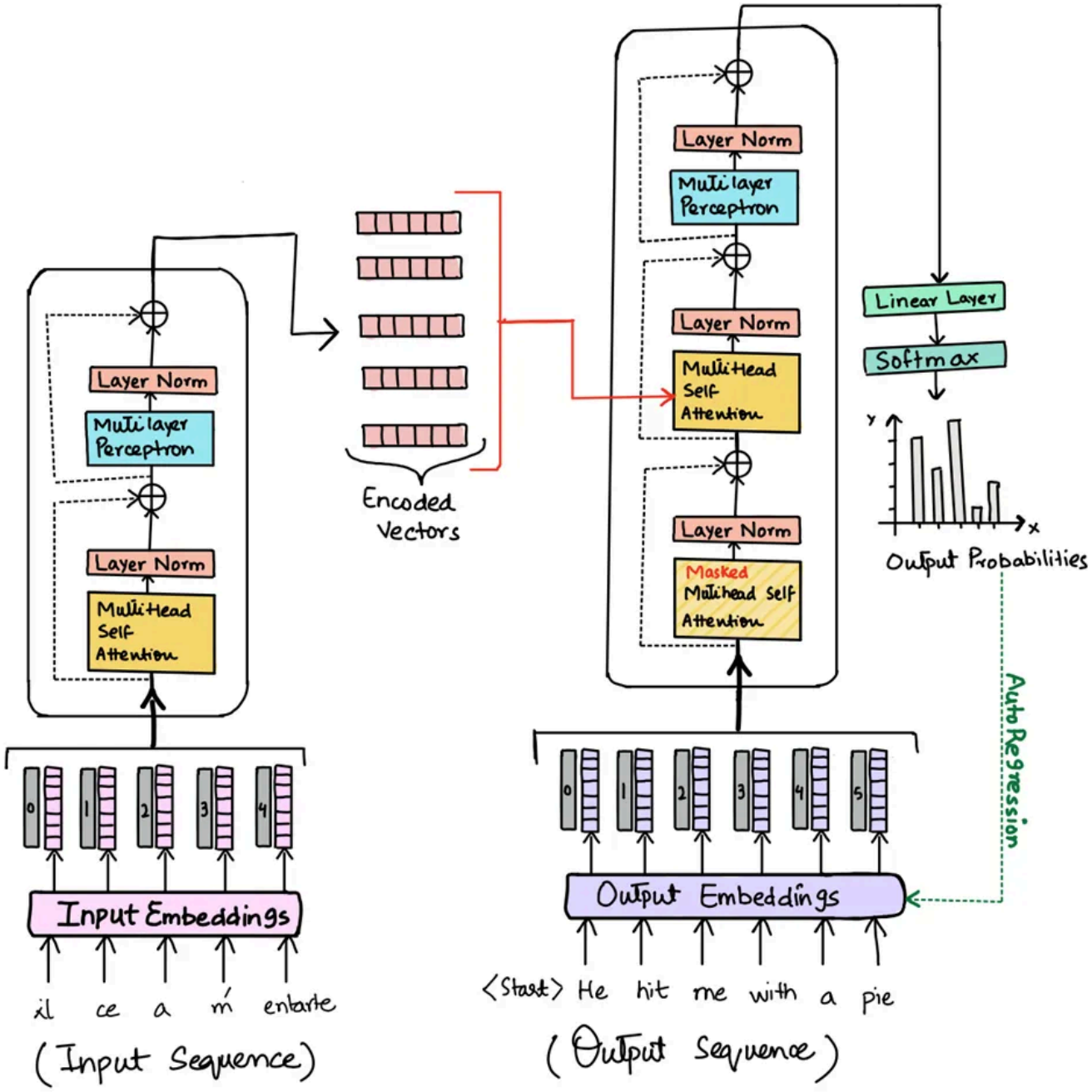


A mostly complete chart of

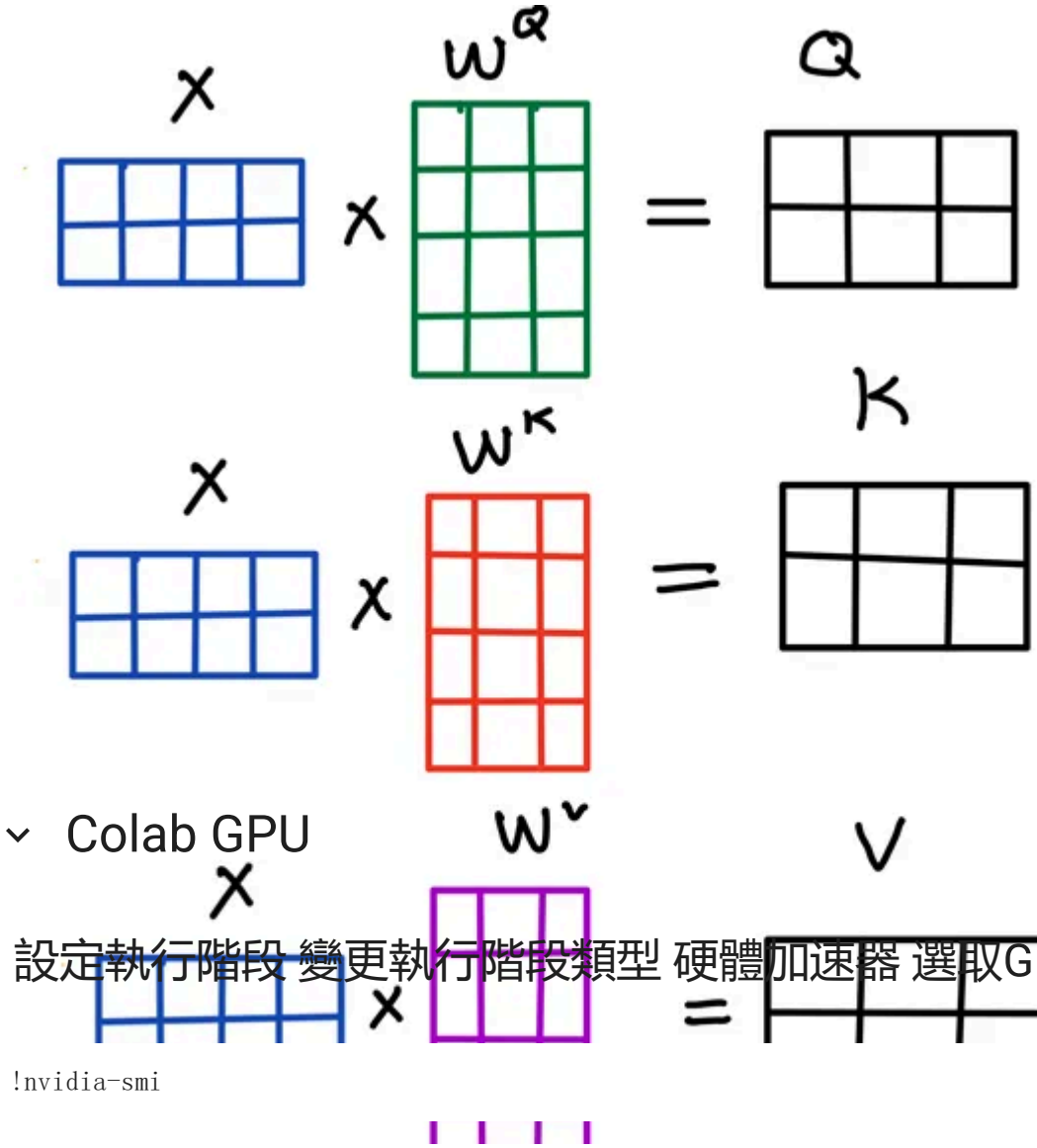
Neural Networks



Encoder-Decoder Transformers



Attension Layer



Colab GPU

設定執行階段 變更執行階段類型 硬體加速器 選取GPU

```
!nvidia-smi
```

```
#  列出GPU與CPU資源
from  tensorflow.python.client  import  device_lib
print(device_lib.list_local_devices())
```

```
#import  os
#os.environ["CUDA_VISIBLE_DEVICES"]  =  "0"
```

Ubuntu版本資訊

```
!lsb_release  -a
```

掛載雲端硬碟

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

切換工作目錄到雲端硬碟目錄下(取用資料比較方便，可以用相對路徑)

```
#你要改成你自己雲端硬碟目錄的路徑，有複製路徑的功能可以使用，不要用鍵盤輸入!
#  cd  前面一行不要加上說明文字，否則colab的cd會認不得指令
```

```
#  cd  to_your_folder
```

```
cd  /content/drive/MyDrive/bigdata/w13-2-Bert自接情緒分類器Training  Bert  based  sentiment  class
```

```
pwd
```

```
ls  -l
```

Insatll packages

```
        transformers:  For  model  handling.
        peft:  For  LoRA  integration.
        datasets:  For  dataset  handling.
        accelerate:  For  distributed  training.
        bitsandbytes:  For  memory-efficient  training  (optional  but  useful  for  Qwen).
```

```
#!pip  install  transformers  peft  datasets  accelerate  bitsandbytes
```

```
pip  install  evaluate
```

```
import  numpy  as  np
import  pandas  as  pd

#  Load  Huggingface  transformers
from  transformers  import  TrainingArguments,  Trainer
from  transformers  import  BertTokenizer,  BertTokenizerFast,  BertForSequenceClassification
import  torch
import  evaluate
import  datasets

#  Setting  device  on  GPU  if  available,  else  CPU
device  =  torch.device('cuda'  if  torch.cuda.is_available()  else  'cpu')
print('Using  device:',  device)
```

```
Using device: cuda
```

Read data

簡體中文資料，資料集來自於網路，轉換成繁體中文

```
df  =  pd.read_csv('./dataset_reviews.csv',  sep='|')
```

```
df
```

```
df.shape
```

```
df.groupby('label').size()
```

```
df.dtypes
```

Read data

簡體中文資料，資料集來自於網路，轉換成繁體中文

```
df  =  pd.read_csv('./dataset_reviews.csv',  sep='|')
```

```
df
```

	text	label
0	做父母一定要有劉墉這樣的心態，不斷地學習，不斷地進步，不斷地給自己補充新鮮血液，讓自己保持一...	1
1	作者真有英國人嚴謹的風格，提出觀點、進行論述論證，儘管本人對物理學瞭解不深，但是仍然能感受到...	1
2	作者長篇大論借用詳細報告數據處理工作和計算結果支持其新觀點。為什麼荷蘭曾經縣有歐洲最高的生產...	1
3	作者在戰幾時之前用了〝擁抱〞令人叫絕・日本如果沒有戰敗，就有會有美軍的佔領，沒胡官僚主義的延...	1
4	作者在少年時即喜閱讀，能看出他精讀了無數經典，因而他有一個龐大的內心世界。他的作品最難能可貴...	1
...	...	...
80437	以前幾乎天天吃，現在調料什麼都不放，	0
80438	昨天訂涼皮兩份，什麼調料都沒有放，就放了點麻油，特別難吃，丟了一份，再也不想吃了	0
80439	涼皮太辣,吃不下都	0

```
df.dtypes
```

80442 rows × 2 columns

Convert the format of y

Convert the format of y from int to LongTensor

Convert label using one-hot representation 輸出資料格式one-hot轉換

```
轉成用2個節點表達兩類
類別0: [1  0]
類別1: [0  1]


如果是3個類別用3個節點表之:
類別0: [1  0  0]
類別1: [0  1  0]
類別2: [0  0  1]
```

負面情緒Negative 0 --> [1 0]

正面情緒Positive 1 --> [0 1]

Easy Represention 0,1,2... (內部會自動轉換為one-hot)

```
負面情緒Negative  0  -->  [0]

正面情緒Positive  1  -->  [1]

如果是3個類別
中立情緒  Neutral  2  -->  [2]
```

label <--> id

```
#  Map  labels  to  integers
categories=['負面','正面']


label_to_id  =  {  cate  :  i  for  i,  cate  in  enumerate(categories)}
```

```
label_to_id

id_to_label = { i : cate for i, cate in enumerate(categories)}

id_to_label
```

開始使用 AI 編寫或生成程式碼。

Sample some examples for demonstration

```
#df = df.sample(20000)
```

Conver pandas dataframe to Huggingface Dataset

```
dataset = datasets.Dataset.from_pandas(df, preserve_index=False)
# eval_data = Dataset.from_pandas(X_eval)

dataset

dataset[0]

dataset.to_pandas().head(5)
```

開始使用 AI 編寫或生成程式碼。

開始使用 AI 編寫或生成程式碼。

Load tokenizer

Purpose: 對輸入文字進行數字代號編碼

```
[('的', 1),
('了', 2),
('是', 3),
('不', 4),
('很', 5),
('我', 6),
('一', 7),
('個', 8),
```

```
  ('好',  9),
  ('就',  10),
  ('也',  11),
  ('這',  12),
  ('有',  13),
  ('還',  14),
  ('都',  15),
  ('酒店',  16),
  ('不錯',  17),
  ('在',  18),
  ('買',  19),
  ('次',  20)]
```

```
# Name of the BERT model
model_name = 'bert-base-chinese'
tokenizer = BertTokenizerFast.from_pretrained(model_name)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarnin
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mo
  warnings.warn(
```

```
tokenizer_config.json: 100%  49.0/49.0 [00:00<00:00, 3.59kB/s]
vocab.txt: 100%  110k/110k [00:00<00:00, 7.25MB/s]
tokenizer.json: 100%  269k/269k [00:00<00:00, 1.64MB/s]
config.json: 100%  624/624 [00:00<00:00, 73.4kB/s]
```

```
tokenizer('你好嗎?')
```

```
tokenizer.decode([101,  872,  1962,  1621,  136,  102])
```

```
tokenizer.get_vocab()['[CLS]']
```

```
tokenizer.get_vocab()['你']
```

```
tokenizer.get_vocab()['好']
```

```
tokenizer.get_vocab()['?']
```

```
tokenizer.get_vocab()
```

Tokenzie text

```
def  tokenize_function(example):
        return  tokenizer(
                example["text"],
                padding="max_length",
                max_length=512,
                truncation=True,
        )

#tokenized_dataset  =  dataset
tokenized_dataset  =  dataset.map(tokenize_function,  batched=True)
```

```
Map:  100%  80442/80442  [00:33<00:00,  3227.80  examples/s]
```

```
tokenized_dataset


#  tokenized_dataset[0]
```

Split dataset for training and testing

Split dataset: Train, Test (Val)

Training set: 訓練資料集 -->給模型讀進去訓練

Test set: 測試資料集 -->驗證或測試模型的準確度

Split dataset into 90% for training and 10% for testing

```
tokenized_dataset  =  tokenized_dataset.train_test_split(test_size=0.05,  seed=1234)
train_data  =  tokenized_dataset["train"]
test_data  =  tokenized_dataset["test"]


train_data


print(test_data)


train_data[0]
```

Encode (Tokenize) input sentences X

```
        tokenizer(
        text=list(input_sentences),
        add_special_tokens=True,
        max_length=250,     #  文件若較長，必須設定數字大一些，最多512字
        truncation=True,
        #padding=True,
        padding='max_length',
        return_tensors='pt',
        return_token_type_ids  =  False,
        return_attention_mask  =  True,
        verbose  =  True)

        Make  the  length  of  every  document  equal  to  250
        Padding(文章填塞變成長度一樣):  文章必須長度一致  最多為250個字詞
```

Load model

Load pretrained Bert model

模型載入方式

(1)可以自己拼接模型，但需要一些深度學習基礎
(2)使用BertForSequenceClassification類別，自動拼接一個具有多類別輸出層的分類器classifier

```
model  =  BertForSequenceClassification.from_pretrained('ckiplab/albert-base-chinese',num_labels=
```

```
device  =  torch.device('cuda')  if  torch.cuda.is_available()  else  torch.device('cpu')

#  Ggoogle標準模型
#model  =  BertForSequenceClassification.from_pretrained('bert-base-chinese',  num_labels=2).to("c

#  中研院資訊所的繁體中文BERT模型
#model  =  BertForSequenceClassification.from_pretrained('ckiplab/albert-base-chinese',  num_label
model  =  BertForSequenceClassification.from_pretrained('ckiplab/albert-tiny-chinese',  num_labels
```

模型每個參數層都是Trainable

```
def  print_model_details(model):
        print("\n"  +  "="*80)
        print("MODEL  ARCHITECTURE  WITH  TRAINABILITY  STATUS")
```

```
        print("="*80)

        trainable_params = 0
        non_trainable_params = 0

        # Print each layer with details
        for idx, (name, layer) in enumerate(model.named_modules(), 1):
                if not list(layer.named_children()):  # Only print leaf nodes (actual la
                        param_count = sum(p.numel() for p in layer.parameters())
                        status = "TRAINABLE" if any(p.requires_grad for p in layer.paramet

                        print(f"\nLayer #{idx:02d}")
                        print(f"├── Name: {name}")
                        print(f"├── Type: {layer.__class__.__name__}")
                        print(f"├── Details: {layer}")
                        print(f"├── Parameters: {param_count:,}")
                        print(f"└── Status: {status}")

                        if status == "TRAINABLE":
                                trainable_params += param_count
                        else:
                                non_trainable_params += param_count

        total_params = trainable_params + non_trainable_params
        trainable_percentage = (trainable_params / total_params) * 100 if total_params >

        print("\n" + "="*80)
        print(f"Total Trainable Parameters: {trainable_params:,}")
        print(f"Total Non-Trainable Parameters: {non_trainable_params:,}")
        print(f"Total Parameters: {total_params:,}")
        print(f"Trainable Parameters Percentage: {trainable_percentage:.2f}%")
        print("="*80)


print_model_details(model)
```

Evaluation metrics using accuracy

```
def compute_metrics(eval_pred):
        predictions, labels = eval_pred
        predictions = np.argmax(predictions, axis=1)  # Convert probabilities to predicte
        metric = evaluate.load('accuracy')
        return metric.compute(predictions=predictions, references=labels)
```

Training arguments

save_strategy (str or [~trainer_utils.IntervalStrategy], *optional*, defaults to "steps")
- `"no"`: No save is done during training.
- `"epoch"`: Save is done at the end of each epoch.

```
- `"steps"`: Save is done every `save_steps`.
save_steps (int, *optional*, defaults to 500): Number of updates steps before two ch
save_total_limit (int, *optional*): If a value is passed, will limit the total amoun
no_cuda (bool, *optional*, defaults to False): Whether to not use CUDA even when it


training_args = TrainingArguments(
        output_dir=repo_name,
        group_by_length=True,
        length_column_name='input_length',
        per_device_train_batch_size=24,
        gradient_accumulation_steps=2,
        evaluation_strategy="steps",
        num_train_epochs=20,
        fp16=True,
        save_steps=1000,
        save_strategy='steps', # we cannot set it to "no". Otherwise, the model canno
        eval_steps=1000,
        logging_steps=1000,
        learning_rate=5e-5,
        warmup_steps=500,
        save_total_limit=3,
        load_best_model_at_end = True # this will let the model save the best checkp
)

https://stackoverflow.com/questions/62525680/save-only-best-weights-with-huggingface-transformer
https://discuss.huggingface.co/t/save-only-best-model-in-trainer/8442



# requires several GB of GPU memory
training_args = TrainingArguments(
       # 訓練多少回合與批量
       num_train_epochs=5,    # Number of training epochs
       output_dir="checkpoints_v1",    # Output directory for checkpoints

       learning_rate=5e-5,    # Learning rate for the optimizer
       weight_decay=0.01,    # Weight decay for regularization
       warmup_steps=500,
       seed=101,

       per_device_train_batch_size=64,    # Batch size per device
       per_device_eval_batch_size=32,    # Batch size per device for evaluation

       eval_strategy='steps',    # Evaluate after each epoch
       save_strategy="steps",    # Save model checkpoints after each epoch

       #load_best_model_at_end=True,    # Load the best model based on the chosen metric
       save_total_limit=2,
       push_to_hub=False,    # Disable pushing the model to the Hugging Face Hub
```

```
        report_to="none",    # Disable logging to Weight&Bias
        #fp16=True,  # 是否用此精度訓練Whether to use fp16 16-bit (mixed) precision traini
        #bf16=True,  # for 新型GPU 才能設定bf16
        logging_steps=1000,
)


trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_data,
        eval_dataset=test_data,
        #  callbacks=[EarlyStoppingCallback(early_stopping_pa)],
        compute_metrics=compute_metrics,
)
```

Resume training from the last checkpoint if available

接續訓練

訓練後，手動載入之前訓練的分類器權重

```
#model_path='checkpoints'  #  訓練後的模型存放路徑
#model  =  BertForSequenceClassification.from_pretrained(model_path,  num_labels=2)
```

Let's train the model

```
%%time
trainer.train()
```

[5975/5975 1:04:26, Epoch 5/5]

Step	Training Loss	Validation Loss	Accuracy
1000	0.403600	0.278437	0.889635
2000	0.267800	0.272809	0.888889
3000	0.240100	0.237229	0.910017
4000	0.227200	0.237954	0.904052
5000	0.212700	0.235466	0.905792

Downloading builder script: 100% 4.20k/4.20k [00:00<00:00, 291kB/s]

```
CPU times: user 1h 3min 32s, sys: 11.4 s, total: 1h 3min 44s
Wall time: 1h 4min 28s
TrainOutput(global_step=5975, training_loss=0.25934356561764516, metrics=
{'train_runtime': 3867.8258, 'train_samples_per_second': 98.788,
'train_steps_per_second': 1.545, 'total_flos': 5619714263009280.0, 'train_loss':
0.25934356561764516, 'epoch': 5.0})
```

開始使用 AI 編寫或生成程式碼。

Evaluation

```
trainer.evaluate()
```

[126/126 00:13]

```
{'eval_loss': 0.23371019959449768,
 'eval_accuracy': 0.906785980611484,
 'eval_runtime': 14.5858,
 'eval_samples_per_second': 275.816,
 'eval_steps_per_second': 8.639,
 'epoch': 5.0}
```

模型之存檔與讀取檔案

評估後會存檔，並刪除前一次較差(val_loss)的模型，保留較佳的模型，一直會有兩個模型儲存著。

Save model

trainer.save_model()

```
# First way to save model
# Three files are saved: config.json pytorch_model.bin training_args.bin
model_path='best-model-v1'
trainer.save_model(model_path)


# Reload from model
model = BertForSequenceClassification.from_pretrained(model_path, num_labels=2)
```

model.save_pretrained()

```
# Second way to save the model
# Two files are saved: config.json pytorch_model.bin
# model_path = "best-model-v1"
# model.save_pretrained(model_path)


# Reload from model
model = BertForSequenceClassification.from_pretrained(model_path, num_labels=2)
```

Save tokenizer

```
model_path = "best-model-v1"
tokenizer.save_pretrained(model_path)
```

Load model and tokenizer

```
model_path = "best-model-v1"
model = BertForSequenceClassification.from_pretrained(model_path, num_labels=2)
model.to(device)

# reload our model/tokenizer. Optional, only usable when in Python files instead of r
tokenizer = BertTokenizerFast.from_pretrained(model_path)
```

Pediction模型使用

```
# Function to make predictions
def predict_sentiment(text, model, tokenizer, device):
    # Tokenize the input text
    inputs = tokenizer(
        text,
        max_length=512,
        truncation=True,
        return_tensors="pt"
    ).to(device)

    # Get model predictions
    with torch.no_grad():
        outputs = model(**inputs)

    # Extract logits and apply softmax to get probabilities
    # logits = outputs.logits
    logits = outputs["logits"]    # 取出 logits


    probabilities = torch.nn.functional.softmax(logits, dim=-1)

    # Get the predicted class (0: negative, 1: positive)
    predicted_class = torch.argmax(probabilities, dim=-1).item()

    # Get the class name using id_to_label
    predicted_label = id_to_label[predicted_class]

    # Get the confidence score
    confidence = probabilities[0][predicted_class].item()
```

```
    return {
        "text": text,
```