Text Classification Using Transformer Networks (BERT)

Some initialization:

In [1]:

```
import random
import torch
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm

# enable tqdm in pandas
tqdm.pandas()

# set to True to use the gpu (if there is one available)
use_gpu = True

# select device
device = torch.device('cuda' if use_gpu and torch.cuda.is_available() else '
print(f'device: {device.type}')

# random seed
seed = 1122

# set random seed
if seed is not None:
    print(f'random seed: {seed}')
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
```

```
device: cuda
random seed: 1122
```

Read the train/dev/test datasets and create a HuggingFace `Dataset` object:

In [2]:

```
def read_data(filename):
    # read csv file
    df = pd.read_csv(filename)
    # add column names
    df.columns = ['label', 'title', 'description']
    # make labels zero-based
    df['label'] -= 1
    # concatenate title and description, and remove backslashes
    df['text'] = df['title'] + " " + df['description']
    df['text'] = df['text'].str.replace('\\', ' ', regex=False)
    return df
```

In [3]:

```
labels = ['World', 'Sports', 'Business', 'Sci/Tech']
train_df = read_data('/kaggle/input/ag-news-classification-dataset/train.csv
test_df = read_data('/kaggle/input/ag-news-classification-dataset/test.csv')
train_df
```

Out[3]:

	label	title	description	text
0	2	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...	Wall St. Bears Claw Back Into the Black (Reute...
1	2	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...	Carlyle Looks Toward Commercial Aerospace (Reu...
2	2	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...	Oil and Economy Cloud Stocks' Outlook (Reuters...
3	2	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...	Iraq Halts Oil Exports from Main Southern Pipe...
4	2	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...	Oil prices soar to all-time record, posing new...
...	...	...	...	...
119995	0	Pakistan's Musharraf Says Won't Quit as Army C...	KARACHI (Reuters) - Pakistani President Perve...	Pakistan's Musharraf Says Won't Quit as Army C...
119996	1	Renteria signing a top-shelf deal	Red Sox general manager Theo Epstein acknowled...	Renteria signing a top-shelf deal Red Sox gene...
119997	1	Saban not going to Dolphins yet	The Miami Dolphins will put their courtship of...	Saban not going to Dolphins yet The Miami Dolp...
119998	1	Today's NFL games	PITTSBURGH at NY GIANTS Time: 1:30 p.m. Line: ...	Today's NFL games PITTSBURGH at NY GIANTS Time...
119999	1	Nets get Carter from Raptors	INDIANAPOLIS -- All-Star Vince Carter was trad...	Nets get Carter from Raptors INDIANAPOLIS -- A...

120000 rows × 4 columns

In [4]:

```
from sklearn.model_selection import train_test_split

train_df, eval_df = train_test_split(train_df, train_size=0.9)
train_df.reset_index(inplace=True, drop=True)
eval_df.reset_index(inplace=True, drop=True)

print(f'train rows: {len(train_df.index):,}')
print(f'eval rows: {len(eval_df.index):,}')
print(f'test rows: {len(test_df.index):,}')
```

```
train rows: 108,000
eval rows: 12,000
test rows: 7,600
```

In [5]:

```
from datasets import Dataset, DatasetDict

ds = DatasetDict()
ds['train'] = Dataset.from_pandas(train_df)
ds['validation'] = Dataset.from_pandas(eval_df)
ds['test'] = Dataset.from_pandas(test_df)
ds
```

Out[5]:

```
DatasetDict({
    train: Dataset({
        features: ['label', 'title', 'description', 'text'],
        num_rows: 108000
    })
    validation: Dataset({
        features: ['label', 'title', 'description', 'text'],
        num_rows: 12000
    })
    test: Dataset({
        features: ['label', 'title', 'description', 'text'],
        num_rows: 7600
    })
})
```

Tokenize the texts:

In [6]:

```
from transformers import AutoTokenizer

transformer_name = 'bert-base-cased'
tokenizer = AutoTokenizer.from_pretrained(transformer_name)
```

```
/opt/conda/lib/python3.10/site-packages/transformers/tokenization_utils_bas
e.py:1617: FutureWarning: `clean_up_tokenization_spaces` was not set. It wil
l be set to `True` by default. This behavior will be deprecated in transform
ers v4.45, and will be then set to `False` by default. For more details chec
k this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
```

In [7]:

```
def tokenize(examples):
    return tokenizer(examples['text'], truncation=True)

train_ds = ds['train'].map(
    tokenize, batched=True,
    remove_columns=['title', 'description', 'text'],
)
eval_ds = ds['validation'].map(
    tokenize,
    batched=True,
    remove_columns=['title', 'description', 'text'],
)
train_ds.to_pandas()
```

```
Map:   0%|          | 0/108000 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/12000 [00:00<?, ? examples/s]
```

Out[7]:

	label	input_ids	token_type_ids	attention_mask
0	2	[101, 16752, 13335, 1186, 2101, 6690, 9717, 11...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
1	1	[101, 145, 11680, 17308, 9741, 2428, 150, 1469...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
2	2	[101, 1418, 14099, 27086, 1494, 1114, 4031, 11...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
3	1	[101, 2404, 117, 6734, 1996, 118, 1565, 5465, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4	3	[101, 142, 10044, 27302, 4317, 1584, 3273, 111...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
...	...	...	...	...
107995	1	[101, 4922, 2274, 1654, 1112, 10503, 1505, 112...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107996	3	[101, 10605, 24632, 11252, 21285, 10221, 118, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107997	2	[101, 13832, 3484, 11300, 4060, 5058, 112, 188...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107998	3	[101, 142, 13675, 3756, 5795, 2445, 1104, 109,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107999	2	[101, 157, 16450, 1658, 5302, 185, 7776, 11006...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

108000 rows × 4 columns

Create the transformer model:

In [8]:

```
from torch import nn
from transformers.modeling_outputs import SequenceClassifierOutput
from transformers.models.bert.modeling_bert import BertModel, BertPreTrainec

# https://github.com/huggingface/transformers/blob/65659a29cf5a079842e61a63c

class BertForSequenceClassification(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels
        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
        self.init_weights()

    def forward(self, input_ids=None, attention_mask=None, token_type_ids=No
        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            **kwargs,
        )
        cls_outputs = outputs.last_hidden_state[:, 0, :]
        cls_outputs = self.dropout(cls_outputs)
        logits = self.classifier(cls_outputs)
        loss = None
        if labels is not None:
            loss_fn = nn.CrossEntropyLoss()
            loss = loss_fn(logits, labels)
        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

In [9]:

```
from transformers import AutoConfig

config = AutoConfig.from_pretrained(
    transformer_name,
    num_labels=len(labels),
)

model = (
    BertForSequenceClassification
    .from_pretrained(transformer_name, config=config)
)
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized: ['classifier.
bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
```

Create the trainer object and train:

In [15]:

```
from transformers import TrainingArguments

num_epochs = 2
batch_size = 24
weight_decay = 0.01
model_name = f'{transformer_name}-sequence-classification'

training_args = TrainingArguments(
    output_dir=model_name,
    log_level='error',
    num_train_epochs=num_epochs,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    evaluation_strategy='epoch',
    weight_decay=weight_decay,
)
```

```
/opt/conda/lib/python3.10/site-packages/transformers/training_args.py:1545: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
```

In [16]:

```
from sklearn.metrics import accuracy_score

def compute_metrics(eval_pred):
    y_true = eval_pred.label_ids
    y_pred = np.argmax(eval_pred.predictions, axis=-1)
    return {'accuracy': accuracy_score(y_true, y_pred)}
```

In [17]:

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_ds,
    eval_dataset=eval_ds,
    tokenizer=tokenizer,
)
```

In [18]:

```
import os
os.environ["WANDB_DISABLED"] = "true"
```

In [ ]:

```
trainer.train()
```

Evaluate on the test partition:

In [21]:

```
test_ds = ds['test'].map(
    tokenize,
    batched=True,
    remove_columns=['title', 'description', 'text'],
)
test_ds.to_pandas()
```

```
Map:   0%|          | 0/7600 [00:00<?, ? examples/s]
```

Out[21]:

	label	input_ids	token_type_ids	attention_mask
0	2	[101, 11284, 1116, 1111, 157, 151, 12966, 1170...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
1	3	[101, 1109, 6398, 1110, 1212, 131, 2307, 7219,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
2	3	[101, 148, 1183, 119, 1881, 16387, 1116, 4468,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
3	3	[101, 11689, 15906, 6115, 12056, 1116, 1370, 2...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4	3	[101, 11917, 8914, 119, 19294, 4206, 1106, 215...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
...	...	...	...	...
7595	0	[101, 5596, 1103, 1362, 5284, 5200, 3234, 1384...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7596	1	[101, 159, 7874, 1110, 2709, 1114, 13875, 1556...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7597	1	[101, 16247, 2972, 9178, 2409, 4271, 140, 1418...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7598	2	[101, 126, 1104, 1893, 8167, 10721, 4420, 1107...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7599	2	[101, 142, 2064, 4164, 3370, 1154, 13519, 1116...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

7600 rows × 4 columns

In [22]:

```
output = trainer.predict(test_ds)
output
```

Out[22]:

```
PredictionOutput(predictions=array([[-0.7238231, -4.379002 ,  4.65339  , -
1.0163295],
       [-0.3337259, -3.4288719, -3.2322338,  5.696346 ],
       [-1.1298318, -3.0482962, -2.9136894,  5.489656 ],
       ...,
       [-0.8102388,  6.9141192, -2.1829498, -3.3619492],
       [-1.679037 , -3.1050372,  5.469485 , -2.0497522],
       [-3.1804867, -4.8155856,  2.9170318,  3.15535  ]], dtype=float32), l
abel_ids=array([2, 3, 3, ..., 1, 2, 2]), metrics={'test_loss': 0.1871038675
3082275, 'test_accuracy': 0.9443421052631579, 'test_runtime': 24.5588, 'tes
t_samples_per_second': 309.462, 'test_steps_per_second': 15.473})
```

```
In [23]: from sklearn.metrics import classification_report

         y_true = output.label_ids
         y_pred = np.argmax(output.predictions, axis=-1)
         target_names = labels
         print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

       World       0.96      0.96      0.96      1900
      Sports       0.99      0.99      0.99      1900
    Business       0.93      0.90      0.91      1900
    Sci/Tech       0.91      0.93      0.92      1900

    accuracy                           0.94      7600
   macro avg       0.94      0.94      0.94      7600
weighted avg       0.94      0.94      0.94      7600
```

Descripción de los pasos del código

El modelo obtuvo métricas sobresalientes, un accuracy del 94%, el flujo de trabajo para poder obtener estos resultados fue el siguiente:

1. **Inicialización**: Importación de módulos necesarios y configuración del dispositivo (CPU o GPU) y la semilla aleatoria.
2. **Lectura de los conjuntos de datos**: Lectura de los archivos CSV de entrenamiento y prueba, y creación de un objeto `Dataset` de HuggingFace.
3. **División del conjunto de datos**: División del conjunto de datos de entrenamiento en entrenamiento y validación.
4. **Tokenización de los textos**: Uso de `AutoTokenizer` de HuggingFace para tokenizar los textos.
5. **Creación del modelo Transformer**: Definición de la clase `BertForSequenceClassification` para la clasificación de secuencias.
6. **Configuración del modelo**: Carga de la configuración y el modelo preentrenado de BERT.
7. **Creación del objeto de entrenamiento**: Configuración de los argumentos de entrenamiento y creación del objeto `Trainer`.
8. **Entrenamiento del modelo**: Entrenamiento del modelo usando el conjunto de datos de entrenamiento.
9. **Evaluación en el conjunto de prueba**: Evaluación del modelo en el conjunto de datos de prueba y generación de un informe de clasificación.