# Text Classification Using Transformer Networks (BERT)

Some initialization:

In [1]:

```python
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]:

```python
def read_data(filename):
    # read csv file
    df = pd.read_csv(filename, header=None)
    # 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 = open('/kaggle/input/ag-news/ag_news_csv/classes.txt').read().splitl
train_df = read_data('/kaggle/input/ag-news/ag_news_csv/train.csv')
train_df=train_df.sample(frac=0.8, random_state=42)
test_df = read_data('/kaggle/input/ag-news/ag_news_csv/test.csv')
test_df=test_df.sample(frac=0.7, random_state=42)
train_df
```

Out[3]:

| | label | title | description | text |
|---|---|---|---|---|
| **71787** | 2 | BBC set for major shake-up, claims newspaper | London - The British Broadcasting Corporation,... | BBC set for major shake-up, claims newspaper L... |
| **67218** | 2 | Marsh averts cash crunch | Embattled insurance broker #39;s banks agree t... | Marsh averts cash crunch Embattled insurance b... |
| **54066** | 1 | Jeter, Yankees Look to Take Control (AP) | AP - Derek Jeter turned a season that started ... | Jeter, Yankees Look to Take Control (AP) AP - ... |
| **7168** | 3 | Flying the Sun to Safety | When the Genesis capsule comes back to Earth w... | Flying the Sun to Safety When the Genesis caps... |
| **29618** | 2 | Stocks Seen Flat as Nortel and Oil Weigh | NEW YORK (Reuters) - U.S. stocks were set to ... | Stocks Seen Flat as Nortel and Oil Weigh NEW ... |
| **...** | ... | ... | ... | ... |
| **59228** | 3 | Investors Flock to Web Networking Sites | Internet whiz kids Marc Andreessen, Josh Kopel... | Investors Flock to Web Networking Sites Intern... |
| **61417** | 2 | Samsung Electric Quarterly Profit Up | Samsung Electronics Co. Ltd. #39;s (005930.KS:... | Samsung Electric Quarterly Profit Up Samsung E... |
| **20703** | 2 | Coeur Still Committed to Wheaton Deal | Coeur d #39;Alene Mines Corp. said Tuesday tha... | Coeur Still Committed to Wheaton Deal Coeur d ... |
| **40626** | 2 | Clouds on horizon for low-cost airlines | NEW YORK -- As larger US airlines suffer growi... | Clouds on horizon for low-cost airlines NEW YO... |
| **25059** | 1 | Furcal issues apology for DUI arrest, returns ... | NAMES Atlanta Braves shortstop Rafael Furcal r... | Furcal issues apology for DUI arrest, returns ... |

96000 rows × 4 columns

In [4]:

```python
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: 86,400
eval rows: 9,600
test rows: 5,320
```

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: 86400
    })
    validation: Dataset({
        features: ['label', 'title', 'description', 'text'],
        num_rows: 9600
    })
    test: Dataset({
        features: ['label', 'title', 'description', 'text', '__index_level_
0__'],
        num_rows: 5320
    })
})
```

Tokenize the texts:

In [6]:
```
from transformers import AutoTokenizer

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

```
tokenizer_config.json:   0%|          | 0.00/49.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]
```

```
/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%|          | 0/86400 [00:00<?, ? examples/s]
Map:   0%|          | 0/9600 [00:00<?, ? examples/s]
```

Out[7]:

| | label | input_ids | token_type_ids | attention_mask |
|---|---|---|---|---|
| **0** | 1 | [101, 1370, 5676, 117, 1122, 108, 3614, 132, 1... | [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** | 2 | [101, 9105, 9800, 10704, 5596, 109, 4389, 170,... | [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** | 2 | [101, 9105, 7352, 2303, 1170, 1646, 2592, 9105... | [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** | 0 | [101, 14812, 11819, 3814, 7988, 1116, 7277, 23... | [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** | 1 | [101, 13626, 22171, 1895, 1556, 1457, 11098, 1... | [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, ... |
| **...** | ... | ... | ... | ... |
| **86395** | 3 | [101, 13020, 157, 21697, 1106, 8060, 1200, 195... | [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, ... |
| **86396** | 0 | [101, 19569, 5480, 10582, 2087, 117, 5329, 110... | [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, ... |
| **86397** | 2 | [101, 16393, 6603, 15104, 1116, 8211, 139, 202... | [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, ... |
| **86398** | 2 | [101, 9105, 12814, 1106, 2612, 108, 3614, 132,... | [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, ... |
| **86399** | 2 | [101, 157, 11811, 6385, 3377, 4785, 11415, 715... | [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, ... |

86400 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=Nc
        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)
)
```

```
model.safetensors:   0%|          | 0.00/436M [00:00<?, ?B/s]
```

```
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 [10]:
```
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 ve
rsion 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
```

In [11]:
```
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 [12]:
```
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 [13]:
```
trainer.train()
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run name by setting the `TrainingArguments.run_name` parameter.
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
  ········
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
VBox(children=(Label(value='Waiting for wandb.init()...\r'), FloatProgress(value=0.011113777144444441, max=1.0)…
```

Tracking run with wandb version 0.18.3

Run data is saved locally in `/kaggle/working/wandb/run-20241125_023125-5853c6t1`

Syncing run **bert-base-cased-sequence-classification** to Weights & Biases (docs)

View project at https://wandb.ai/a01742342-tecnol-gico-de-monterrey/huggingface

View run at https://wandb.ai/a01742342-tecnol-gico-de-monterrey/huggingface/runs/5853c6t1

[7200/7200 36:16, Epoch 2/2]

| Epoch | Training Loss | Validation Loss | Accuracy |
|---|---|---|---|
| 1 | 0.184600 | 0.205049 | 0.935729 |
| 2 | 0.112600 | 0.194179 | 0.945208 |

```
Out[13]: TrainOutput(global_step=7200, training_loss=0.1760442320505778, metrics={'train_runtime': 2197.8061, 'train_samples_per_second': 78.624, 'train_steps_per_second': 3.276, 'total_flos': 1.047667424173344e+16, 'train_loss': 0.1760442320505778, 'epoch': 2.0})
```

Evaluate on the test partition:

```
In [14]: test_ds = ds['test'].map(
             tokenize,
             batched=True,
             remove_columns=['title', 'description', 'text'],
         )
         test_ds.to_pandas()
```

```
Map:   0%|          | 0/5320 [00:00<?, ? examples/s]
```

Out[14]:

| | label | __index_level_0__ | input_ids | token_type_ids | attention_mask |
|---|---|---|---|---|---|
| **0** | 1 | 7094 | [101, 16061, 191, 16061, 131, 4280, 1392, 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, 1, ... |
| **1** | 0 | 1017 | [101, 2123, 3124, 3681, 12646, 1111, 7443, 110... | [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 | 2850 | [101, 20820, 9780, 131, 1790, 112, 189, 3641, ... | [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 | 1452 | [101, 3177, 11741, 11951, 1530, 3187, 16137, 1... | [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 | 457 | [101, 142, 13675, 2181, 11171, 3844, 1104, 141... | [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, ... |
| **...** | ... | ... | ... | ... | ... |
| **5315** | 3 | 6306 | [101, 1109, 139, 13791, 16752, 17149, 146, 210... | [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, ... |
| **5316** | 2 | 1799 | [101, 152, 2101, 8231, 3561, 15691, 1116, 9105... | [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, ... |
| **5317** | 3 | 1014 | [101, 19265, 15729, 1116, 1706, 17300, 26941, ... | [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, ... |
| **5318** | 1 | 128 | [101, 19753, 2240, 3128, 1942, 21580, 3692, 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, ... |
| **5319** | 3 | 1364 | [101, 145, 14935, 14117, 5300, 2239, 1111, 197... | [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, ... |

5320 rows × 5 columns

In [15]:

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

Out[15]:

```
PredictionOutput(predictions=array([[-0.75240576,  7.3236136 , -2.7632585 ,
-2.7384746 ],
       [ 2.2885764 , -4.5289006 , -3.147025  ,  4.9397216 ],
       [-2.7778184 , -4.5765634 ,  0.9413744 ,  4.9241147 ],
       ...,
       [-3.047365  , -4.506488  , -0.16100709,  5.95451   ],
       [-1.8705817 ,  7.34204   , -2.504134  , -2.517787  ],
       [-0.51633906, -3.2232592 ,  5.384857  , -2.0062768 ]],
      dtype=float32), label_ids=array([1, 0, 3, ..., 3, 1, 3]), metrics={'t
est_loss': 0.17778262495994568, 'test_accuracy': 0.9464285714285714, 'test_
runtime': 18.5976, 'test_samples_per_second': 286.058, 'test_steps_per_seco
nd': 11.937})
```

In [16]:

```
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      0.95      0.96      1330
      Sports       0.99      0.99      0.99      1334
    Business       0.92      0.92      0.92      1314
    Sci/Tech       0.92      0.93      0.92      1342

    accuracy                           0.95      5320
   macro avg       0.95      0.95      0.95      5320
weighted avg       0.95      0.95      0.95      5320
```

El código implementa un proceso de clasificación de texto utilizando BERT. Primero, importa las bibliotecas necesarias como torch, pandas y las herramientas de Hugging Face. Luego, carga los datos de texto desde archivos CSV y los organiza en conjuntos de entrenamiento, validación y prueba, utilizando Dataset de Hugging Face para estructurarlos.

A continuación, aplica tokenización a los textos mediante un tokenizador preentrenado (bert-base-cased), convirtiendo las frases en secuencias de números que el modelo BERT puede interpretar. Divide los datos en subconjuntos para entrenamiento y evaluación, asegurando una validación adecuada del modelo.

El modelo BERT, preentrenado en grandes corpus de texto, se ajusta para la tarea específica de clasificación utilizando las etiquetas del conjunto de datos. Finalmente, el modelo se entrena en los datos procesados y se evalúa en el conjunto de prueba para medir su precisión y rendimiento en la tarea de clasificación.