ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID

Fine Tuning

Hugging Face

Eduardo Fernández Carrión, PhD

Datasets

- Librería de HuggingFace que nos permite **descargar datasets** de su Hub y también subir nuestros propios datasets.
- Está optimizada para ser **eficiente en memoria y rendimiento**.
- Está diseñada para manejar **grandes conjuntos de datos** que podrían no caber en la memoria RAM de manera eficiente.
- Contiene funciones integradas para el **preprocesamiento** y la **transformación** de variables.
- No solo para textos, también **Audios** e **Imágenes**.

```python
from datasets import load_dataset

dataset = load_dataset("rotten_tomatoes")
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 8530
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 1066
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 1066
    })
})
```

Fine tuning

Es la técnica de tomar un **modelo que ya ha sido entrenado** en un conjunto de datos grande y **ajustarlo posteriormente** con un conjunto de datos más pequeño y específico.

Esto nos permite al modelo adaptarse a tareas específicas manteniendo la riqueza de conocimientos adquiridos durante el pre-entrenamiento.

- En **transfer learning** se suelen entrenar **solo las capas finales del modelo** (generalmente, de hidden states en adelante).
- En **fine tuning** se entrenan **todas las capas.**

TrainingArguments

La clase `TrainingArguments` contendrá todos los **hiperparámetros** que luego la clase `Trainer` usará para el entrenamiento y la evaluación.

El único argumento es un directorio donde se guardará el modelo entrenado, así como los puntos de control (`checkpoints`) a lo largo del proceso.

```python
from transformers import TrainingArguments

training_args = TrainingArguments("model-trainer-checkpoint")
```

```
TrainingArguments(
_n_gpu=0,
adafactor=False,
adam_beta1=0.9,
adam_beta2=0.999,
adam_epsilon=1e-08,
auto_find_batch_size=False,
bf16=False,
bf16_full_eval=False,
data_seed=None,
dataloader_drop_last=False,
dataloader_num_workers=0,
dataloader_pin_memory=True,
ddp_backend=None,
ddp_broadcast_buffers=None,
ddp_bucket_cap_mb=None,
ddp_find_unused_parameters=None,
ddp_timeout=1800,
debug=[],
deepspeed=None,
disable_tqdm=False,
dispatch_batches=None,
do_eval=False,
do_predict=False,
do_train=False,
eval_accumulation_steps=None,
eval_delay=0,
eval_steps=None,
evaluation_strategy=no,
fp16=False,
fp16_backend=auto,
fp16_full_eval=False,
fp16_opt_level=O1,
```

DataCollators

Los `DataCollators` son objetos que se utilizan para **agrupar** de forma eficiente muestras individuales de datos **en lotes** para el entrenamiento o la evaluación de modelos de procesamiento de NLP.

- **DataCollatorWithPadding**: Agrega automáticamente padding a los datos para que todos los elementos de un lote tengan la misma longitud. Es útil para modelos de secuencia que necesitan entradas de longitud uniforme.
- **DataCollatorForTokenClassification**: Ideal para tareas de clasificación de tokens, como el reconocimiento de entidades nombradas (Named Entity Recognition, NER), donde el padding debe ser aplicado a los tokens.
- **DataCollatorForSeq2Seq**: Específico para modelos de secuencia a secuencia, como T5 o BART, donde se requiere padding tanto para la entrada como para las etiquetas de salida.

DataCollatorWithPadding

DataCollatorForTokenClassification

DataCollatorForSeq2Seq

DataCollatorForLanguageModeling

DataCollatorForWholeWordMask

DataCollatorForPermutationLanguage Modeling

https://huggingface.co/docs/transformers/en/main_classes/data_collator

Trainer

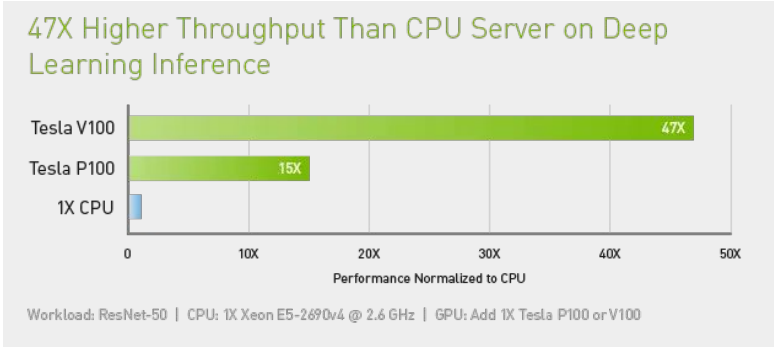
`Trainer` es una clase de HF que abstrae y simplifica el proceso de entrenamiento de modelos.

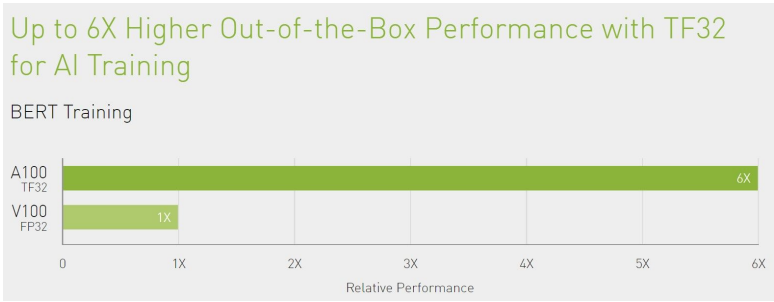
```python
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

```python
trainer.train()
```

[1377/1377 1:50:14, Epoch 3/3]

Step	Training Loss
500	0.533500
1000	0.319400





Evaluation

Con nuestro modelo ya entrenado, la métrica personalizarla a nuestro problema.

```
predictions = trainer.predict(tokenized_datasets["test"])
predictions.predictions
```

```
array([[-3.1999178,  2.8812287],
       [-1.6312883,  1.7061045],
       [-3.2638092,  2.9540458],
       ...,
       [-3.0577457,  2.7345617],
       [-3.2176082,  2.917534 ],
       [-3.1510766,  2.9679265]], dtype=float32)
```

```
import numpy as np

preds = np.argmax(predictions.predictions, axis=-1)
```

```
from sklearn.metrics import accuracy_score

accuracy_score(y_pred=preds, y_true=ds['test']['label'])
```

```
0.8243478260869566
```

```
from sklearn.metrics import f1_score

f1_score(y_pred=preds, y_true=ds['test']['label'])
```

```
0.8718816067653277
```

ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID