TL08 Trainer

Índice

1 Introducción

Documentación de transformers:

- **Get started:** TL02
- **Clases base:** TL03
- **Inferencia:**
  - **API Pipeline:** TL03
  - **LLMs:** TL06
  - **Chat with models:** TL07
- **Entrenamiento:**
  - **API Trainer:** secciones siguientes

Otras librerías y recursos:

- **pytorch** TL01
- **datasets:** TL04
- **evaluate:** TL04
- Benchmarking: TL06
- **accelerate:** secciones siguientes

Objetivo: introducir trainer y accelerate

2 Trainer

Trainer: bucle de entrenamiento y evaluación para modelos PyTorch basado en accelerate; pasos:

1. Calcular la pérdida a partir de un paso de entrenamiento
2. Calcular los gradientes con el método backward
3. Actualizar los pesos con base en los gradientes
4. Repetir los pasos anteriores hasta alcanzar un número de épocas predeterminado

Fine-tuning: adaptación (con Trainer) de un modelo preentrenado a una tarea específica con datos especializados

Optimzadores: `AdamW`, `AdaFactor` o especializados

Búsqueda de hiperparámetros: con `hyperparameter_search()` y un backend apropiado

3 Fine-tuning de un ViT para food101

Fine-tuning for Image Classification with 🤗 Transformers: ejemplo en el que se basa esta sección

Librerías:

In [ ]:
```
import numpy as np; import torch; import transformers; import evaluate
```

Modelo:

In [ ]:
```
model_checkpoint = "google/vit-base-patch16-224-in21k"
```

Dataset: food101

In [ ]:
```
from datasets import load_dataset
full_dataset = load_dataset("food101", split="train")
dataset = full_dataset.shuffle(seed=7).select(range(50000)); del full_dataset # mucho menos para pruebas
```

In [ ]:
```
dataset[0]['image'].reduce(2)
```

Out[ ]:

Dataset: diccionarios `label2id` y `id2label`

In [ ]:
```
labels = dataset.features["label"].names; label2id, id2label = dict(), dict()
for i, label in enumerate(labels):
    label2id[label] = i; id2label[i] = label
id2label[0]
```

Out[ ]:
```
'apple_pie'
```

Dataset: procesador de imagen del modelo a adaptar

In [ ]:
```
from transformers import AutoImageProcessor
image_processor = AutoImageProcessor.from_pretrained(model_checkpoint)
image_processor
```

Out[ ]:
```
ViTImageProcessor {
  "do_convert_rgb": null,
  "do_normalize": true,
  "do_rescale": true,
  "do_resize": true,
  "image_mean": [
    0.5,
    0.5,
    0.5
  ],
  "image_processor_type": "ViTImageProcessor",
  "image_std": [
    0.5,
    0.5,
    0.5
  ],
  "resample": 2,
  "rescale_factor": 0.00392156862745098,
  "size": {
    "height": 224,
    "width": 224
  }
}
```

Dataset: partición train+valid y preproceso

In [ ]:
```
from torchvision.transforms import (CenterCrop, Compose, Normalize, RandomHorizontalFlip, RandomResizedCrop,
    Resize, ToTensor)
normalize = Normalize(mean=image_processor.image_mean, std=image_processor.image_std)
train_transforms = Compose([RandomResizedCrop(image_processor.size["height"]),
    RandomHorizontalFlip(), ToTensor(), normalize])
val_transforms = Compose([Resize(image_processor.size["height"]), CenterCrop(image_processor.size["height"]),
    ToTensor(), normalize])
def preprocess_train(example_batch):
    """Apply train_transforms across a batch."""
    example_batch["pixel_values"] = [train_transforms(image.convert("RGB")) for image in example_batch["image"]]
    return example_batch
def preprocess_val(example_batch):
    """Apply val_transforms across a batch."""
    example_batch["pixel_values"] = [val_transforms(image.convert("RGB")) for image in example_batch["image"]]
    return example_batch
splits = dataset.train_test_split(test_size=0.1)
train_ds = splits["train"]; val_ds = splits["test"]
train_ds.set_transform(preprocess_train); val_ds.set_transform(preprocess_val)
```

In [ ]:
```
train_ds[0].keys()
```

Out[ ]:
```
dict_keys(['image', 'label', 'pixel_values'])
```

Modelo:

In [ ]:
```
from transformers import AutoModelForImageClassification, TrainingArguments, Trainer
model = AutoModelForImageClassification.from_pretrained(model_checkpoint, label2id=label2id, id2label=id2label)
print(str(model)[:1400]+" ...")
```

```
ViTForImageClassification(
  (vit): ViTModel(
    (embeddings): ViTEmbeddings(
      (patch_embeddings): ViTPatchEmbeddings(
        (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
      )
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): ViTEncoder(
      (layer): ModuleList(
        (0-11): 12 x ViTLayer(
          (attention): ViTAttention(
            (attention): ViTSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
            )
            (output): ViTSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
          )
          (intermediate): ViTIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): ViTOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
          (layernorm_before): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (layernorm_after): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
  ...
```

Entrenamiento: `TrainingArguments`

In [ ]:
```
model_name = model_checkpoint.split("/")[-1]; batch_size = 32
args = TrainingArguments(
    f"{model_name}-finetuned-food101",
    remove_unused_columns=False,
    eval_strategy = "epoch",
    save_strategy = "epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=batch_size,
    gradient_accumulation_steps=4,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=10,
    warmup_ratio=0.1,
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    push_to_hub=False)
```

Entrenamiento: métrica para evaluar un batch

In [ ]:
```
metric = evaluate.load("accuracy")
def compute_metrics(eval_pred):
    """Computes accuracy on a batch of predictions"""
    predictions = np.argmax(eval_pred.predictions, axis=1)
    return metric.compute(predictions=predictions, references=eval_pred.label_ids)
```

Entrenamiento: función `collate` para homogeneizar un batch

In [ ]:
```
def collate_fn(examples):
    pixel_values = torch.stack([example["pixel_values"] for example in examples])
    labels = torch.tensor([example["label"] for example in examples])
    return {"pixel_values": pixel_values, "labels": labels}
```

Entrenamiento: `Trainer`

In [ ]:
```
trainer = Trainer(
    model,
    args,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    processing_class=image_processor,
    compute_metrics=compute_metrics,
    data_collator=collate_fn)
```

In [ ]:
```
train_results = trainer.train()
```

[3520/3520 2:05:29, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	3.285400	3.201704	0.687400
2	1.927300	1.849738	0.771000
3	1.303000	1.234237	0.812000
4	1.057800	0.970029	0.825800
5	0.865500	0.847530	0.831600
6	0.665200	0.771722	0.836600
7	0.635000	0.726028	0.839600
8	0.534300	0.700680	0.840600
9	0.604700	0.691146	0.841600
10	0.552300	0.681789	0.843800

Entrenamiento: estadísticas

In [ ]:
```
trainer.log_metrics("train", train_results.metrics)
```

```
***** train metrics *****
  epoch                    =          10.0
  total_flos               = 32505334274GF
  train_loss               =        1.3284
  train_runtime            =    2:05:33.32
  train_samples_per_second =        59.735
  train_steps_per_second   =         0.467
```

Entrenamiento: comprobamos que se escoge el mejor modelo (quizás no el último)

In [ ]:
```
metrics = trainer.evaluate()
trainer.log_metrics("eval", metrics)
```

 [157/157 00:48]

```
***** eval metrics *****
  epoch                   =       10.0
  eval_accuracy           =     0.8438
  eval_loss               =     0.6818
  eval_runtime            = 0:00:49.19
  eval_samples_per_second =    101.642
  eval_steps_per_second   =      3.192
```

4 Accelerate

Documentación de accelerate:

- **Get started:** accelerate, instalación y quicktour
- **Tutoriales:** Overview; Add Accelerate to your code; Execution process; TPU training; Launching Accelerate scripts; Launching distributed training from Jupyter Notebooks
- **How to guides:**
  - **Accelerate:** Start Here!; Model memory estimator; Model quantization; Experiment trackers; Profiler; Checkpointing; Troubleshoot; Example Zoo
  - **Training:** Gradient accumulation; Local SGD; Low precision (FP8) training; DeepSpeed; ...
  - **Inference:** Big Model Inference; Distributed inference
- **Concepts and fundamentals:** ...
- **Referencia:** ...

Ejemplos de uso: tras clonar el repo de accelerate, se puede ejecutar el Simple NLP example y el Simple vision example (con `timm`, `torchvision` y el Ofxord-IIT Pet Dataset)

```
(APR2526) ajuan@songo:examples$ python nlp_example.py
...
epoch 0: {'accuracy': 0.7916666666666666, 'f1': 0.8380952380952381}
epoch 1: {'accuracy': 0.8455882352941176, 'f1': 0.8908145580589255}
epoch 2: {'accuracy': 0.8602941176470589, 'f1': 0.900523560209424}
```

```
(APR2526) ajuan@songo:examples$ python cv_example.py --data_dir images
...
epoch 0: 79.91
epoch 1: 90.05
epoch 2: 91.00
```

Official Accelerate Examples: incluye ejemplos básicos, ejemplos sobre características específicas y ejemplos completos

Image classification fine-tuning example: ejemplo completo de fine-tuning de un modelo preentrenado para clasificación de imágenes

Ejemplo: tras consultar el README, se puede reproducir el siguiente experimento de fine-tuning de `google/vit-base-patch16-224-in21k` con `cifar10`

```
(APR2526) ajuan@songo:image-classification$ accelerate launch run_image_classification_no_trainer.py --
image_column_name img
...
***** Running training *****
Num examples = 42500
Num Epochs = 3
Instantaneous batch size per device = 8
Total train batch size (w. parallel, distributed & accumulation) = 8
Gradient Accumulation steps = 1
Total optimization steps = 15939
epoch 0: {'accuracy': 0.9782666666666666}
epoch 1: {'accuracy': 0.984}
epoch 2: {'accuracy': 0.9889333333333333}
```

accelerate vs Trainer: `Trainer` puede verse como una interfaz de alto nivel para `accelerate`

5 Ejercicio: fine-tuning de un ViT para CIFAR10

In [ ]:
```
import numpy as np; import torch; import transformers; import evaluate
```

In [ ]:
```
model_checkpoint = "google/vit-base-patch16-224-in21k"
```

In [ ]:
```
from datasets import load_dataset
train_ds, test_ds = load_dataset('cifar10', split=['train[:50000]', 'test[:10000]']) # mucho menos para pruebas
```

In [ ]:
```
train_ds
```

Out[ ]:
```
Dataset({
    features: ['img', 'label'],
    num_rows: 50000
})
```

In [ ]:
```
train_ds.features
```

Out[ ]:
```
{'img': Image(mode=None, decode=True),
 'label': ClassLabel(names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truc
k'])}
```

In [ ]:
```
train_ds[0]['img']
```

Out[ ]:


In [ ]:
```
train_ds[0]['label']
```

Out[ ]:
```
0
```

In [ ]:
```
id2label = {id:label for id, label in enumerate(train_ds.features['label'].names)}
label2id = {label:id for id,label in id2label.items()}
str(id2label)
```

Out[ ]:
```
"{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'tr
uck'}"
```

Ejercicio: completa el experimento para hallar la precisión en test

Solución:

In [ ]:
```
from transformers import AutoImageProcessor
image_processor = AutoImageProcessor.from_pretrained(model_checkpoint)
```

In [ ]:
```
from torchvision.transforms import (CenterCrop, Compose, Normalize, RandomHorizontalFlip, RandomResizedCrop,
    Resize, ToTensor)
normalize = Normalize(mean=image_processor.image_mean, std=image_processor.image_std)
train_transforms = Compose([RandomResizedCrop(image_processor.size["height"]),
    RandomHorizontalFlip(), ToTensor(), normalize])
val_transforms = Compose([Resize(image_processor.size["height"]), CenterCrop(image_processor.size["height"]),
    ToTensor(), normalize])
def preprocess_train(example_batch):
    """Apply train_transforms across a batch."""
    example_batch["pixel_values"] = [train_transforms(image.convert("RGB")) for image in example_batch["img"]]
    return example_batch
def preprocess_val(example_batch):
    """Apply val_transforms across a batch."""
    example_batch["pixel_values"] = [val_transforms(image.convert("RGB")) for image in example_batch["img"]]
    return example_batch
train_ds.set_transform(preprocess_train)
test_ds.set_transform(preprocess_val)
```

In [ ]:
```
from transformers import AutoModelForImageClassification, TrainingArguments, Trainer
model = AutoModelForImageClassification.from_pretrained(model_checkpoint, label2id=label2id, id2label=id2label)
```

In [ ]:
```
model_name = model_checkpoint.split("/")[-1]; batch_size = 32
args = TrainingArguments(
    f"{model_name}-finetuned-cifar10",
    remove_unused_columns=False,
    eval_strategy = "epoch",
    save_strategy = "epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=batch_size,
    gradient_accumulation_steps=4,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=3,
    warmup_ratio=0.1,
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    push_to_hub=False)
```

In [ ]:
```
metric = evaluate.load("accuracy")
def compute_metrics(eval_pred):
    """Computes accuracy on a batch of predictions"""
    predictions = np.argmax(eval_pred.predictions, axis=1)
    return metric.compute(predictions=predictions, references=eval_pred.label_ids)
```

In [ ]:
```
def collate_fn(examples):
    pixel_values = torch.stack([example["pixel_values"] for example in examples])
    labels = torch.tensor([example["label"] for example in examples])
    return {"pixel_values": pixel_values, "labels": labels}
```

In [ ]:
```
trainer = Trainer(
    model,
    args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
    processing_class=image_processor,
    compute_metrics=compute_metrics,
    data_collator=collate_fn)
```

In [ ]:
```
train_results = trainer.train()
```

[1173/1173 42:18, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.382300	0.205862	0.976900
2	0.322600	0.122392	0.983100
3	0.209100	0.099222	0.986400

In [ ]:
```
trainer.log_metrics("train", train_results.metrics)
```

```
***** train metrics *****
  epoch                    =           3.0
  total_flos               = 10826282096GF
  train_loss               =        0.5027
  train_runtime            =    0:42:20.66
  train_samples_per_second =         59.04
  train_steps_per_second   =         0.462
```

In [ ]:
```
metrics = trainer.evaluate()
trainer.log_metrics("eval", metrics)
```

[313/313 01:27]

```
***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =     0.9864
  eval_loss               =     0.0992
  eval_runtime            = 0:01:30.31
  eval_samples_per_second =    110.723
  eval_steps_per_second   =      3.466
```