Fine-tuning

Fine-tuning adapts a pretrained model to a specific task with a smaller specialized dataset. This approach requires far less data and compute compared to training a model from scratch, which makes it a more accessible option for many users.

Transformers provides the Trainer API, which offers a comprehensive set of training features, for fine-tuning any of the models on the Hub.

> Learn how to fine-tune models for other tasks in our Task Recipes section in Resources!

This guide will show you how to fine-tune a model with Trainer to classify Yelp reviews.

Log in to your Hugging Face account with your user token to ensure you can access gated models and share your models on the Hub.

```python
from huggingface_hub import login

login()
```

Start by loading the Yelp Reviews dataset and preprocess (tokenize, pad, and truncate) it for training. Use map to preprocess the entire dataset in one step.

```python
from datasets import load_dataset
from transformers import AutoTokenizer

dataset = load_dataset("yelp_review_full")
tokenizer = AutoTokenizer.from_pretrained("google-bert/bert-base-cased")

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

```
dataset = dataset.map(tokenize, batched=True)
```

Fine-tune on a smaller subset of the full dataset to reduce the time it takes. The results won't be as good compared to fine-tuning on the full dataset, but it is useful to make sure everything works as expected first before committing to training on the full dataset.

```
small_train = dataset["train"].shuffle(seed=42).select(range(1000))
small_eval = dataset["test"].shuffle(seed=42).select(range(1000))
```

Trainer

Trainer is an optimized training loop for Transformers models, making it easy to start training right away without manually writing your own training code. Pick and choose from a wide range of training features in TrainingArguments such as gradient accumulation, mixed precision, and options for reporting and logging training metrics.

Load a model and provide the number of expected labels (you can find this information on the Yelp Review dataset card).

```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained("google-bert/bert-base-case
"Some weights of BertForSequenceClassification were not initialized from the model che
"You should probably TRAIN this model on a down-stream task to be able to use it for p
```

> The message above is a reminder that the models pretrained head is discarded and replaced with a randomly initialized classification head. The randomly initialized head needs to be fine-tuned on your specific task to output meaningful predictions.

With the model loaded, set up your training hyperparameters in TrainingArguments. Hyperparameters are variables that control the training process - such as the learning rate, batch size, number of epochs - which in turn impacts model performance. Selecting the correct hyperparameters is important and you should experiment with them to find the best configuration for your task.

For this guide, you can use the default hyperparameters which provide a good baseline to begin with. The only settings to configure in this guide are where to save the checkpoint, how to evaluate model performance during training, and pushing the model to the Hub.

Trainer requires a function to compute and report your metric. For a classification task, you'll use evaluate.load to load the accuracy function from the Evaluate library. Gather the predictions and labels in compute to calculate the accuracy.

```
import numpy as np
import evaluate

metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    # convert the logits to their predicted class
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

Set up TrainingArguments with where to save the model and when to compute accuracy during training. The example below sets it to `"epoch"`, which reports the accuracy at the end of each epoch. Add `push_to_hub=True` to upload the model to the Hub after training.

```
from transformers import TrainingArguments
```

```
training_args = TrainingArguments(
    output_dir="yelp_review_classifier",
    eval_strategy="epoch",
    push_to_hub=True,
)
```

Create a Trainer instance and pass it the model, training arguments, training and test datasets, and evaluation function. Call train() to start training.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    compute_metrics=compute_metrics,
)
trainer.train()
```

Finally, use push_to_hub() to upload your model and tokenizer to the Hub.

```
trainer.push_to_hub()
```

Resources

Refer to the Transformers examples for more detailed training scripts on various tasks. You can also check out the notebooks for interactive examples.

</> Update on GitHub

← Trainer Optimizers →