Search models, datasets, users...

Transformers documentation

Installation

Quickstart

Open in Colab Open Studio Lab Copy page

Transformers is designed to be fast and easy to use so that everyone can start learning or building with transformer models.

The number of user-facing abstractions is limited to only three classes for instantiating a model, and two APIs for inference or training. This quickstart introduces you to Transformers' key features and shows you how to:

- load a pretrained model
- run inference with Pipeline
- fine-tune a model with Trainer

Set up

To start, we recommend creating a Hugging Face account. An account lets you host and access version controlled models, datasets, and Spaces on the Hugging Face Hub, a collaborative platform for discovery and building.

Create a User Access Token and log in to your account.

notebook CLI

Paste your User Access Token into notebook_login when prompted to log in.

```
from huggingface_hub import notebook_login

notebook_login()
```

Install Pytorch.

```
!pip install torch
```

Then install an up-to-date version of Transformers and some additional libraries from the Hugging Face ecosystem for accessing datasets and vision models, evaluating training, and optimizing training for large models.

```
!pip install -U transformers datasets evaluate accelerate timm
```

Pretrained models

Each pretrained model inherits from three base classes.

Class	Description
PretrainedConfig	A file that specifies a models attributes such as the number of attention heads or vocabulary size.
PreTrainedModel	A model (or architecture) defined by the model attributes from the configuration file. A pretrained model only returns the raw hidden states. For a specific task, use the appropriate model head to convert the raw hidden states into a meaningful result (for example, LlamaModel versus LlamaForCausalLM).
Preprocessor	A class for converting raw inputs (text, images, audio, multimodal) into numerical inputs to the model. For example, PreTrainedTokenizer converts text into tensors and ImageProcessingMixin converts pixels into tensors.

We recommend using the AutoClass API to load models and preprocessors because it automatically infers the appropriate architecture for each task and machine learning framework based on the name or path to the pretrained weights and configuration file.

Use from_pretrained() to load the weights and configuration file from the Hub into the model and preprocessor class.

When you load a model, configure the following parameters to ensure the model is optimally loaded.

- `device_map="auto"` automatically allocates the model weights to your fastest device first.

- `dtype="auto"` directly initializes the model weights in the data type they're stored in, which can help avoid loading the weights twice (PyTorch loads weights in `torch.float32` by default).

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b-hf", dtype="auto",
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-hf")
```

Tokenize the text and return PyTorch tensors with the tokenizer. Move the model to an accelerator if it's available to accelerate inference.

```
model_inputs = tokenizer(["The secret to baking a good cake is "], return_tensors="pt"
```

The model is now ready for inference or training.

For inference, pass the tokenized inputs to generate() to generate text. Decode the token ids back into text with batch_decode().

```
generated_ids = model.generate(**model_inputs, max_length=30)
tokenizer.batch_decode(generated_ids)[0]
'<s> The secret to baking a good cake is 100% in the preparation. There are so many re
```

> Skip ahead to the Trainer section to learn how to fine-tune a model.

Pipeline

The Pipeline class is the most convenient way to inference with a pretrained model. It supports many tasks such as text generation, image segmentation, automatic speech recognition, document question answering, and more.

> Refer to the Pipeline API reference for a complete list of available tasks.

Create a Pipeline object and select a task. By default, Pipeline downloads and caches a default pretrained model for a given task. Pass the model name to the `model` parameter to choose a specific model.

text generation | image segmentation | automatic speech recognition

Use infer_device() to automatically detect an available accelerator for inference.

```
from transformers import pipeline, infer_device

device = infer_device()

pipeline = pipeline("text-generation", model="meta-llama/Llama-2-7b-hf", device=device
```

Prompt Pipeline with some initial text to generate more text.

```
pipeline("The secret to baking a good cake is ", max_length=50)
[{'generated_text': 'The secret to baking a good cake is 100% in the batter. The secre
```

Trainer

Trainer is a complete training and evaluation loop for PyTorch models. It abstracts away a lot of the boilerplate usually involved in manually writing a training loop, so you can start training faster and focus on training design choices. You only need a model, dataset, a preprocessor, and a data collator to build batches of data from the dataset.

Use the TrainingArguments class to customize the training process. It provides many options for training, evaluation, and more. Experiment with training hyperparameters and features like batch size, learning rate, mixed precision, torch.compile, and more to meet your training needs. You could also use the default training parameters to quickly produce a baseline.

Load a model, tokenizer, and dataset for training.

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from datasets import load_dataset

model = AutoModelForSequenceClassification.from_pretrained("distilbert/distilbert-base
tokenizer = AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased")
dataset = load_dataset("rotten_tomatoes")
```

Create a function to tokenize the text and convert it into PyTorch tensors. Apply this function to the whole dataset with the map method.

```
def tokenize_dataset(dataset):
    return tokenizer(dataset["text"])
dataset = dataset.map(tokenize_dataset, batched=True)
```

Load a data collator to create batches of data and pass the tokenizer to it.

```
from transformers import DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Next, set up TrainingArguments with the training features and hyperparameters.

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="distilbert-rotten-tomatoes",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=2,
    push_to_hub=True,
)
```

Finally, pass all these separate components to Trainer and call train() to start.

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()
```

Share your model and tokenizer to the Hub with push_to_hub().

```
trainer.push_to_hub()
```

Congratulations, you just trained your first model with Transformers!

Next steps

Now that you have a better understanding of Transformers and what it offers, it's time to keep exploring and learning what interests you the most.

- **Base classes**: Learn more about the configuration, model and processor classes. This will help you understand how to create and customize models, preprocess different types of inputs (audio, images, multimodal), and how to share your model.
- **Inference**: Explore the Pipeline further, inference and chatting with LLMs, agents, and how to optimize inference with your machine learning framework and hardware.
- **Training**: Study the Trainer in more detail, as well as distributed training and optimizing training on specific hardware.
- **Quantization**: Reduce memory and storage requirements with quantization and speed up inference by representing weights with fewer bits.
- **Resources**: Looking for end-to-end recipes for how to train and inference with a model for a specific task? Check out the task recipes!

</> Update on GitHub

← Transformers Quickstart →