



Natural Language Processing

Hugging Face Tutorial 2024/10/22



Outline

- Introduction to Hugging Face
- BERT
- Hugging Face Trainer



Introduction to Hugging Face

- An **open-source** library for state-of-the-art AI models (Natural Language Processing / Computer Vision / Audio ...) with various deep learning frameworks (PyTorch / TensorFlow / JAX ...)
- Links
 - Documentation: <https://huggingface.co/docs>
 - Source code: <https://github.com/huggingface>

Hugging Face made the first PyTorch Implementation of BERT

[Releases](#) / v0.1.2

First release

thomwolf released this Nov 17, 2018 v0.1.2 ce37b8e

This is the first release of `pytorch_pretrained_bert`.

Assets 4

pytorch_pretrained_bert-0.1.2-py3-none-any.whl	35.6 KB	Nov 26, 2018
pytorch_pretrained_bert-0.1.2.tar.gz	45.2 KB	Nov 26, 2018
Source code (zip)		Nov 26, 2018
Source code (tar.gz)		Nov 26, 2018

3 people reacted

Commits

master

Commits on Nov 1, 2018

Merge pull request #7 from stefan-it/typo-fixes

jacobdevlin-google authored on Nov 1, 2018

tree-wide: minor typo fixes

stefan-it committed on Nov 1, 2018

Merge pull request #6 from cbockman/patch-1

jacobdevlin-google authored on Nov 1, 2018

minor README spelling

cbockman authored on Nov 1, 2018

Commits on Oct 31, 2018

Initial BERT release

jacobdevlin-google committed on Oct 31, 2018

Commits on Oct 26, 2018

Adding placeholder README.md file

jacobdevlin-google committed on Oct 26, 2018

<https://github.com/huggingface/transformers/releases/tag/v0.1.2>

<https://github.com/google-research/bert/commits>



 Documentations<https://huggingface.co/docs>**• Hub**

Host Git-based models, datasets and Spaces on the Hugging Face Hub.

• Hub Python Library

Client library for the HF Hub: manage repositories from your Python runtime.

• Inference API (serverless)

Experiment with over 200k models easily using the serverless tier of Inference Endpoints.

• Transformers

State-of-the-art ML for Pytorch, TensorFlow, and JAX.

• Datasets

Access and share datasets for computer vision, audio, and NLP tasks.

• Huggingface.js

A collection of JS libraries to interact with Hugging Face, with TS types included.

• Inference Endpoints (dedicated)

Easily deploy models to production on dedicated, fully managed infrastructure.

• Diffusers

State-of-the-art diffusion models for image and audio generation in PyTorch.

• Gradio

Build machine learning demos and other web apps, in just a few lines of Python.

• Transformers.js

Community library to run pretrained models from Transformers in your browser.

• PEFT

Parameter efficient finetuning methods for large models.

Installation

Basically, you need to install PyTorch first before you install transformers.

(Recommended)

```
pip install transformers
```

(If you want to try new things)

```
git clone  
https://github.com/huggingface/transformers.git  
cd transformers  
pip install -e .
```



Packages required in this tutorial

We need the following packages for this tutorial. On Colab, you may not need to re-install these packages for they may have already been installed.

```
!pip install torch==2.4.0
!pip install transformers==4.37.0
!pip install datasets==3.0.1
!pip install accelerate==0.21.0
!pip install scikit-learn==1.5.2
!pip install wget
!pip install tarfile
```

Documentations

<https://huggingface.co/docs>

• Hub

Host Git-based models, datasets and Spaces on the Hugging Face Hub.

• Transformers

State-of-the-art ML for Pytorch, TensorFlow, and JAX.

• Diffusers

State-of-the-art diffusion models for image and audio generation in PyTorch.

• Datasets

Access and share datasets for computer vision, audio, and NLP tasks.

• Gradio

Build machine learning demos and other web apps, in just a few lines of Python.

• Hub Python Library

Client library for the HF Hub: manage repositories from your Python runtime.

• Huggingface.js

A collection of JS libraries to interact with Hugging Face, with TS types included.

• Transformers.js

Community library to run pretrained models from Transformers in your browser.

• Inference API (serverless)

Experiment with over 200k models easily using the serverless tier of Inference Endpoints.

• Inference Endpoints (dedicated)

Easily deploy models to production on dedicated, fully managed infrastructure.

• PEFT

Parameter efficient finetuning methods for large models.

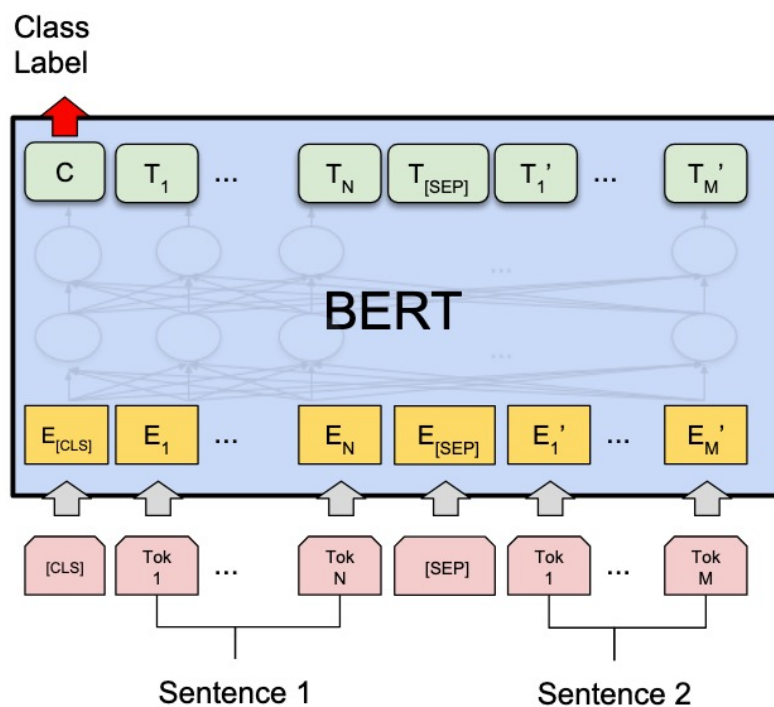
Check the versions of your packages

```
import torch
print(f"PyTorch version: {torch.__version__}")

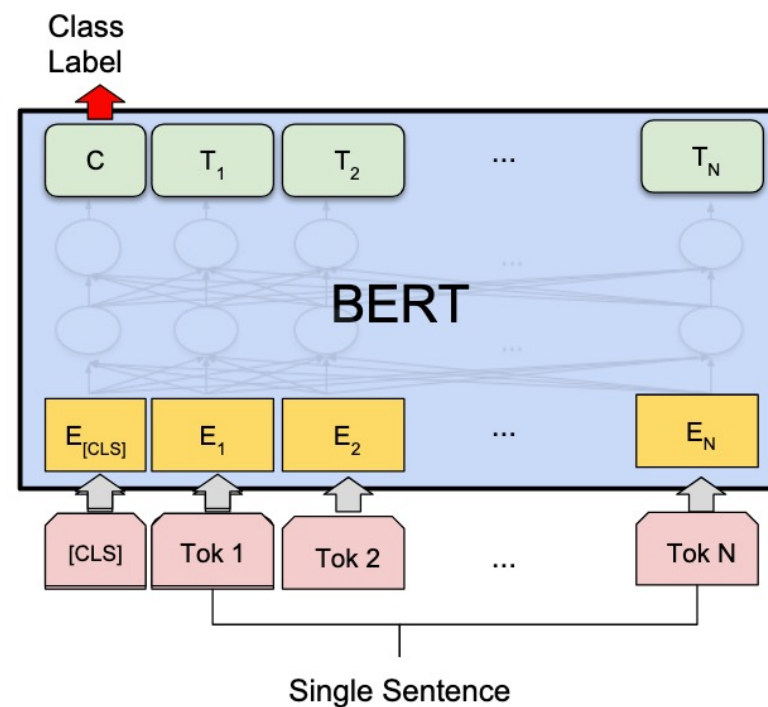
import transformers
print(f"Hugging Face Transformers version: {transformers.__version__}")

import datasets
print(f"Hugging Face Datasets version: {datasets.__version__}")
```

Sentence Classification with BERT



(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



(b) Single Sentence Classification Tasks:
SST-2, CoLA

Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." NAACL 2019.

Transformers

Search documentation

V4.45.2

EN

133,612

Feature Extractor

Image Processor

MODELS

TEXT MODELS

ALBERT

BART

BARThez

BARTpho

BERT

BertGeneration

BertJapanese

Bertweet

BigBird

BigBirdPegasus

BioGpt

Blenderbot

Blenderbot Small

BLOOM

BertForSequenceClassification

`class transformers.BertForSequenceClassification`

<source>

(config)

Parameters

- config** (`BertConfig`) — Model configuration class with all the parameters of the model. Initializing with a config file does not load the weights associated with the model, only the configuration. Check out the `from_pretrained()` method to load the model weights.

Bert Model transformer with a sequence classification/regression head on top (a linear layer on top of the pooled output) e.g. for GLUE tasks.

This model inherits from `PreTrainedModel`. Check the superclass documentation for the generic methods the library implements for all its model (such as downloading or saving, resizing the input embeddings, pruning heads etc.)

This model is also a PyTorch `torch.nn.Module` subclass. Use it as a regular PyTorch Module and refer to the PyTorch documentation for all matter related to general usage and behavior.

`forward`

<source>

Usage tips

Using Scaled Dot Product Attention (SDPA)

Training

Inference

Resources

BertConfig

BertTokenizer

BertTokenizerFast

TFBertTokenizer

Bert specific outputs

BertModel

BertForPreTraining

BertLMHeadModel

BertForMaskedLM

BertForNextSentence Prediction

BertForSequence Classification

BertForMultipleChoice

BertForTokenClassification

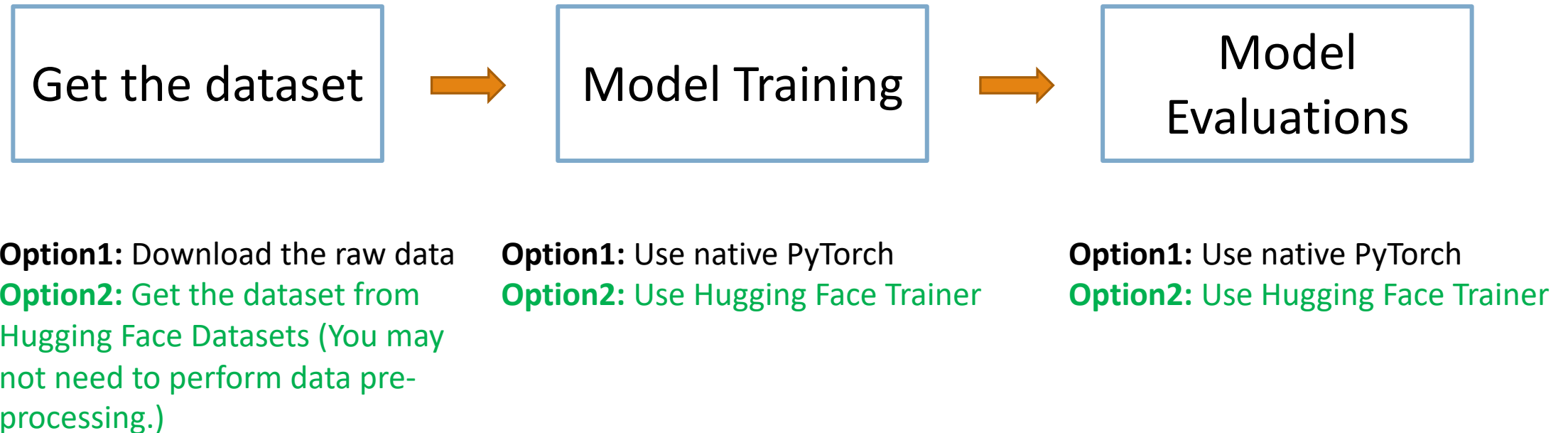
BertForQuestionAnswering

TFBertModel

https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertForSequenceClassification



Workflow for training a model using Hugging Face



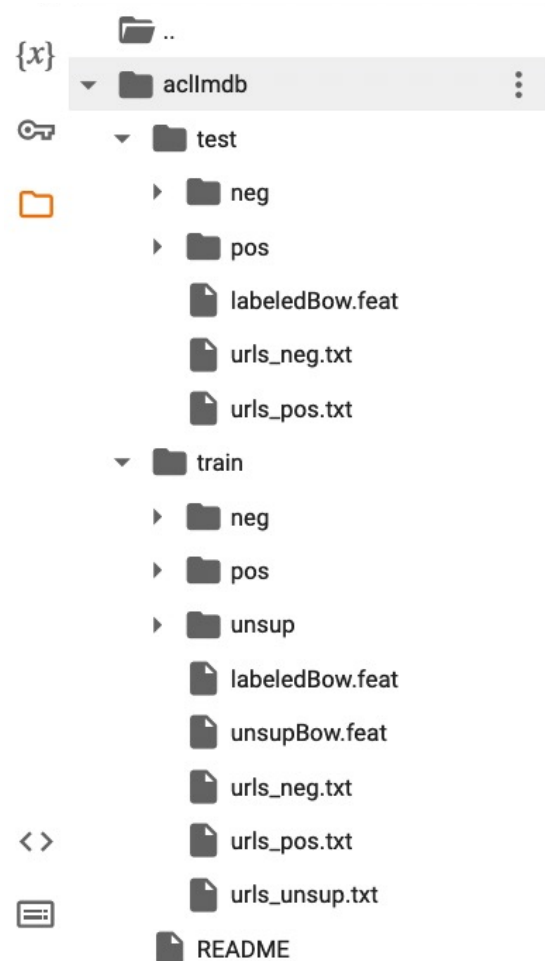
Dataset Preparation (IMDb reviews)

```
import wget
import tarfile

# Download the IMDb dataset
url = 'http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz'
filename = wget.download(url, out='./')

# Unzip the `aclImdb_v1.tar.gz` file
tar = tarfile.open('aclImdb_v1.tar.gz', 'r:gz')
tar.extractall()
```

The folder structure of IMDb after unzipping



Sentiment Analysis

pos

Positive movie reviews

neg

Negative movie reviews

Create a function to pre-process the IMDb dataset

```
1 # 5. Create a function to pre-process the IMDb dataset
2 def read_imdb_split(split_dir):
3     split_dir = Path(split_dir)
4     texts, labels = [], []
5     for label_dir in ["pos", "neg"]:
6         # Use glob() to get files with the extension ".txt"
7         for text_file in (split_dir/label_dir).glob("*.txt"):
8             # read_text() returns the decoded contents of the pointed-to file as a string
9             tmp_text = text_file.read_text()
10
11         # Append the read text to the list we defined in advance
12         texts.append(tmp_text)
13
14         # Build labels based on the folder name
15         labels.append(0 if label_dir == "neg" else 1)
16
17     return texts, labels
```

[Doc of Pathlib] https://docs.python.org/3/library/pathlib.html#pathlib.Path.read_text



Pre-process the IMDb dataset

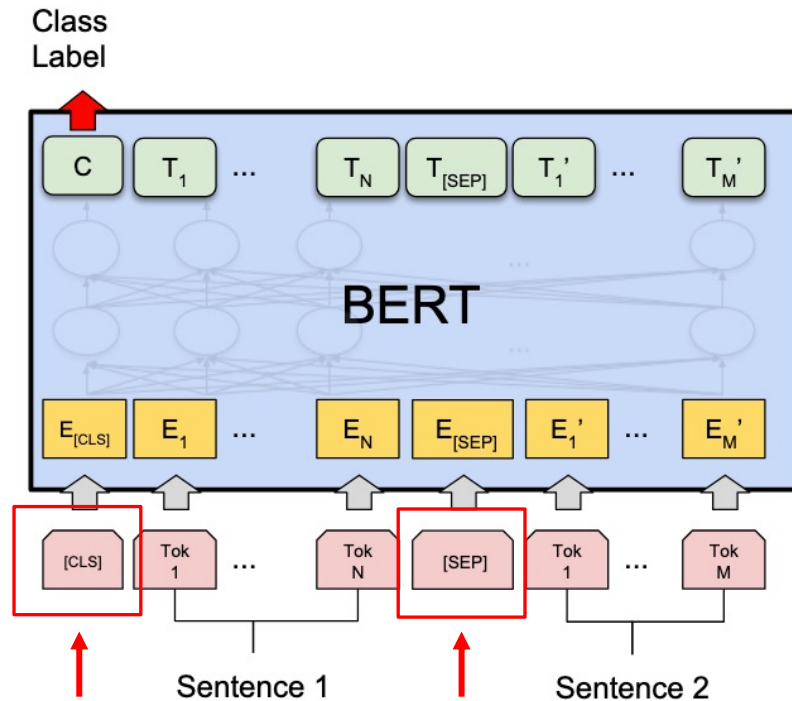
```
# 6. Pre-process the IMDb dataset (run the cell)
```

```
train_texts, train_labels = read_imdb_split('aclImdb/train')  
test_texts, test_labels = read_imdb_split('aclImdb/test')
```

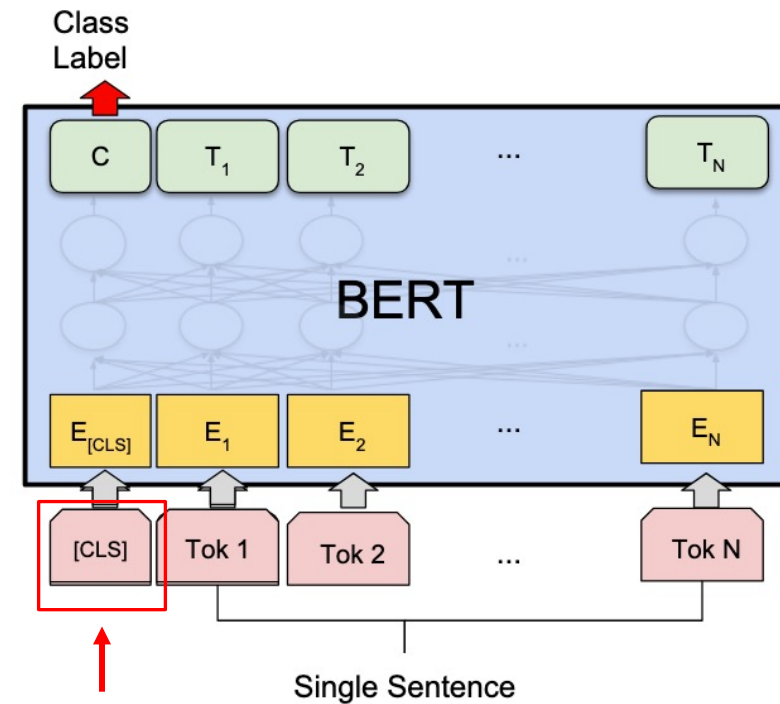
Dataset splitting

```
1 # 7. Use train_test_split to split the training data into training and validation data
2 from sklearn.model_selection import train_test_split
3
4 # Set a random seed for reproducibility
5 random_seed = 42
6
7 # Set the ratio of the validation set to the training set
8 valid_ratio = 0.2
9
10 train_texts, val_texts, train_labels, val_labels = train_test_split(
11     train_texts,
12     train_labels,
13     test_size=valid_ratio,
14     random_state=random_seed
15 )
```

Pre-process the input sentences for BERT



(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



(b) Single Sentence Classification Tasks:
SST-2, CoLA

Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." NAACL 2019.

Hugging Face Tokenizer

```
# Load the Hugging Face tokenizer

model_name = "bert-base-uncased"
# Use .from_pretrained() for a pre-trained model
tokenizer = transformers.AutoTokenizer.from_pretrained(model_name)
```

uncased: it does not make a difference between english and English.

<https://huggingface.co/google-bert/bert-base-uncased>

<https://huggingface.co/google-bert/bert-base-cased>



Tasks Libraries Datasets Languages Licenses
Other

Multimodal

 Image-Text-to-Text
 Visual Question Answering
 Document Question Answering
 Video-Text-to-Text  Any-to-Any

Computer Vision

 Depth Estimation  Image Classification
 Object Detection  Image Segmentation
 Text-to-Image  Image-to-Text
 Image-to-Image  Image-to-Video
 Unconditional Image Generation
 Video Classification  Text-to-Video
 Zero-Shot Image Classification
 Mask Generation

Models 1,061,004

↑↓ Sort: Trending

 **nvidia/Llama-3.1-Nemotron-70B-Instruct-HF**

 Text Generation • Updated 3 days ago • ⬇ 48.7k • ❤ 733

 **SWivid/F5-TTS**

 Text-to-Speech • Updated 5 days ago • ⬇ 60.5k • ❤ 511

 **nvidia/Llama-3.1-Nemotron-70B-Instruct**

Updated 3 days ago • ⬇ 1.26k • ❤ 306

 **black-forest-labs/FLUX.1-dev**

 Text-to-Image • Updated Aug 16 • ⬇ 1.08M • ⚡ • ❤ 5.64k

 **rain1011/pyramid-flow-sd3**

 Text-to-Video • Updated 9 days ago • ❤ 672

 **deepseek-ai/Janus-1.3B**

 Any-to-Any • Updated 1 day ago • ⬇ 1.62k • ❤ 232

 **mistralai/Ministral-8B-Instruct-2410**

Updated 2 days ago • ⬇ 28 • ❤ 207

<https://huggingface.co/models>



AutoClasses (models, tokenizers)

- BertTokenizer. GPT2Tokenizer
- In many cases, the architecture you want to use **can be guessed** from the name or the path of the pretrained model you are supplying to the `from_pretrained` method.
- AutoClasses are here to do this job for you so that you automatically retrieve the relevant model given the name/path to the pretrained weights/config/vocabulary:
- Ex: `model = AutoModel.from_pretrained('bert-base-cased')` will create an instance of `BertModel`).

https://huggingface.co/transformers/v3.0.2/model_doc/auto.html



Outputs of the Tokenizer (Case1)

Do truncation and padding to the model_max_length

8. Load the Hugging Face tokenizer

```
train_encodings = tokenizer(train_texts, truncation=True, padding=True)
val_encodings = tokenizer(val_texts, truncation=True, padding=True)
test_encodings = tokenizer(test_texts, truncation=True, padding=True)
```

	Type_1	Type_2
truncation	True / False	longest_first / only_second / only_first / do_not_truncate
padding	True / False	longest / max_length / do_not_pad
max_length	None (model_max_length)	Your desired maximum length (integer).

https://huggingface.co/docs/transformers/pad_truncation



Outputs of the Tokenizer (Case2)

Do truncation and padding to a length of 300 tokens

```
encodings = tokenizer(  
    texts,  
    truncation=True,  
    padding=True,  
    max_length=300  
)
```

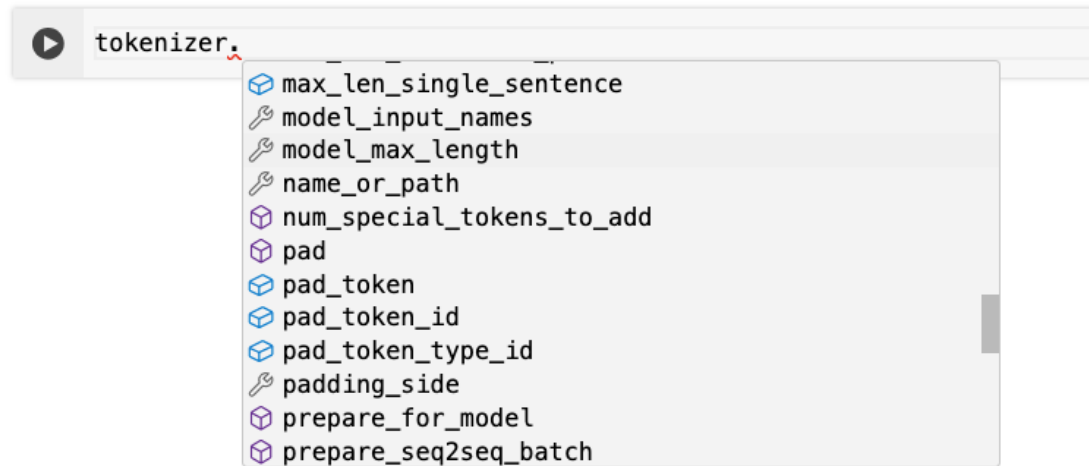
	Type_1	Type_2
truncation	True / False	longest_first / only_second / only_first / do_not_truncate
padding	True / False	longest / max_length / do_not_pad
max_length	None (model_max_length)	Your desired maximum length (integer).

https://huggingface.co/docs/transformers/pad_truncation



Methods inside `tokenizer`

- Type the dot and wait for a few seconds (on Colab)



- Or you can use `dir` to check all the methods of the object

```
tokenizer_methods = dir(tokenizer)

# Group them into rows for better readability
for i in range(0, len(tokenizer_methods), 5):
    print(", ".join(tokenizer_methods[i:i+5]))
```

Some useful methods of `tokenizer`

Example using `"bert-base-uncased"`

(From this slide, the output will be highlighted in blue.)

```
# Get the maximum length of the pre-trained model
print(tokenizer.model_max_length)
```

512

```
# Get the ID of the special tokens
print(tokenizer.vocab["[CLS]"])
print(tokenizer.vocab["[SEP]"])
print(tokenizer.vocab["[PAD]"])
```

101

102

0



Check the tokenized results

The output of `tokenizer` is a dictionary wrapper.

```
# See the keys of the tokenizer output
print(val_encodings.keys())
```

```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
```

```
# See which type the output encodings belong to
print(type(val_encodings))
```

[transformers.tokenization_utils_base.BatchEncoding](https://huggingface.co/transformers/v3.5.1/_modules/transformers/tokenization_utils_base.html)

https://huggingface.co/transformers/v3.5.1/_modules/transformers/tokenization_utils_base.html



Check the first data of the validation set.

[illegible]

[PAD]

Check the tokenized results (token_type_ids)

Check the first data of the validation set.

```
print(val_encodings.token_type_ids[0])
```

[illegible]

All zeros show that this is a single-sentence task.
There will be additional ones for the token places of a second sentence.

Check the tokenized results (attention_mask)

Check the first data of the validation set.

```
print(val_encodings.attention_mask[0])
```

[illegible]

Padding (telling the model not to focus on these regions)

Building the Dataset class using PyTorch

```
1 class IMDbDataset(torch.utils.data.Dataset):
2     def __init__(self, encodings, labels):
3         self.encodings = encodings
4         self.labels = labels
5
6     def __getitem__(self, idx):
7         # Note that the tokenizer output is a dict wrapper
8         # Convert data and labels into PyTorch tensors
9         item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
10        item['labels'] = torch.tensor(self.labels[idx])
11
12        return item
13
14    def __len__(self):
15        # Number of a dataset
16        return len(self.labels)
17
18 train_dataset = IMDbDataset(train_encodings, train_labels)
19 val_dataset = IMDbDataset(val_encodings, val_labels)
20 test_dataset = IMDbDataset(test_encodings, test_labels)
```



Or you can use Hugging Face Datasets

Using Hugging Face Dataset

```
# Load the IMDb training set
train = datasets.load_dataset("imdb", split="train")

# Split the validation set
random_seed = 42
splits = train.train_test_split(
    test_size=0.2,
    seed=random_seed
)
train, valid = splits['train'], splits['test']

# Load the IMDb test set
test = datasets.load_dataset("imdb", split="test")
```

Datasets: [stanfordnlp/imdb](#)
 like 236

Tasks: [Text Classification](#)

Modalities: [Text](#)

Formats: [parquet](#)

Sub-tasks: [sentiment-classification](#)

Languages: [English](#)

Size: [100K - 1M](#)

Libraries: [Datasets](#)
[pandas](#)
[Croissant](#)

+1

License: [other](#)
[Dataset card](#)
[Viewer](#)
[Files and versions](#)
[Community](#) **6**
Dataset Viewer
[Auto-converted to Parquet](#)
[API](#)
[Embed](#)
[Full Screen Viewer](#)

Split (3)
train · 25k rows

[SQL](#) [Console](#)
text

string · lengths


label

class label

2 classes

I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in 1967. I also heard that at...

0 neg

"I Am Curious: Yellow" is a risible and pretentious steaming pile. It doesn't matter what one's political views are because this film can hardly be taken...

0 neg

If only to avoid making this type of film in the future. This film is interesting as an experiment but tells no cogent story.

One might...

0 neg

This film was probably inspired by Godard's Masculin, féminin and I urge you to

0 neg

Downloads last month

43,632
[Use this dataset](#)
[Edit dataset card](#)
[Papers with Code](#)


Homepage:
[ai.stanford.edu](#)

Size of downloaded dataset files:
83.4 MB

Size of the auto-converted Parquet files:
83.4 MB

Number of rows:
100,000

Models trained or fine-tuned on stanfordnlp/im...
[sileod/deberta-v3-base-tasksource-nli](#)
<https://huggingface.co/datasets/stanfordnlp/imdb>

You still need to tokenize the dataset

Using Hugging Face Dataset

```
1 def to_torch_data(hug_dataset):
2     """Transform Hugging Face Datasets into PyTorch Dataset
3     Args:
4         - hug_dataset: data loaded from HF Datasets
5     Return:
6         - dataset: PyTorch Dataset
7     """
8     dataset = hug_dataset.map(
9         lambda batch: tokenizer(
10             batch["text"],
11             truncation=True,
12             padding=True
13         ),
14         batched=True
15     )
16     dataset.set_format(
17         type='torch',
18         columns=[
19             'input_ids',
20             'attention_mask',
21             'label'
22         ]
23     )
24     return dataset
```

Speed up processing ([ref](#))

Set the format of the dataset to return PyTorch tensors instead of lists ([ref](#))

```
1 train_dataset = to_torch_data(train)
2 val_dataset = to_torch_data(valid)
3 test_dataset = to_torch_data(test)
```



Load the BERT classification model

```
from transformers import AutoModelForSequenceClassification

model_name = "bert-base-uncased"
# Use .from_pretrained() for a pre-trained model
model = AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=2
)
```

Different models for different tasks

Question Answering Tasks (SQuAD v1.1), [ref](#)

```
model = AutoModelForQuestionAnswering.from_pretrained(model_name)
```

Single Sentence Tagging Tasks (CoNLL-2003 NER; Named Entity Recognition), [ref](#)

```
model = AutoModelForTokenClassification.from_pretrained(model_name)
```

Semantic Textual Similarity Tasks (STS-B), [ref](#)

```
model = AutoModelForSequenceClassification.from_pretrained(  
    model_name, num_labels=1  
)
```

What happens for down-stream tasks?

https://github.com/huggingface/transformers/blob/v4.45.2/src/transformers/models/bert/modeling_bert.py#L1651-L1665

```
1651  ✓ class BertForSequenceClassification(BertPreTrainedModel):
1652  ✓     def __init__(self, config):
1653         super().__init__(config)
1654         self.num_labels = config.num_labels
1655         self.config = config
1656
1657         self.bert = BertModel(config)
1658         classifier_dropout = (
1659             config.classifier_dropout if config.classifier_dropout is not None else config.hidden_dropout_prob
1660         )
1661         self.dropout = nn.Dropout(classifier_dropout)
1662         self.classifier = nn.Linear(config.hidden_size, config.num_labels)
1663
1664         # Initialize weights and apply final processing
1665         self.post_init()
```



What happens for down-stream tasks?

https://github.com/huggingface/transformers/blob/v4.45.2/src/transformers/models/bert/modeling_bert.py#L1715-L1734

```
1651     class BertForSequenceClassification(BertPreTrainedModel):
1675         def forward(
1715             if self.config.problem_type is None:
1716                 if self.num_labels == 1:
1717                     self.config.problem_type = "regression"
1718                 elif self.num_labels > 1 and (labels.dtype == torch.long or labels.dtype == torch.int):
1719                     self.config.problem_type = "single_label_classification"
1720                 else:
1721                     self.config.problem_type = "multi_label_classification"
1722
1723             if self.config.problem_type == "regression":
1724                 loss_fct = MSELoss()
1725                 if self.num_labels == 1:
1726                     loss = loss_fct(logits.squeeze(), labels.squeeze())
1727             else:
1728                 loss = loss_fct(logits, labels)
1729             elif self.config.problem_type == "single_label_classification":
1730                 loss_fct = CrossEntropyLoss()
1731                 loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
1732             elif self.config.problem_type == "multi_label_classification":
1733                 loss_fct = BCEWithLogitsLoss()
1734                 loss = loss_fct(logits, labels)
```



Model Training with Hugging Face Trainer

Setting up TrainingArguments



Setting up Trainer / metric



`trainer.train()`

Trainer can perform training, automatic evaluations, and so on.



`trainer.predict()`

Trainer can also help you perform predictions with a trained model.

Why Trainer?

This is a training script with native PyTorch ([ref](#)).

```
72 model.train()
73 for epoch in tqdm(range(args.num_epoch)):
74     for batch in tqdm(train_loader):
75         optimizer.zero_grad()
76         input_ids = batch['input_ids'].to(device)
77         attention_mask = batch['attention_mask'].to(device)
78         labels = batch['labels'].to(device)
79         outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
80         loss = outputs[0]
81         loss.backward()
82         optimizer.step()
83
84 model.eval()
85 with torch.no_grad():
86     y_preds = []
87     for batch in tqdm(test_loader):
88         input_ids = batch['input_ids'].to(device)
89         attention_mask = batch['attention_mask'].to(device)
90         labels = batch['labels'].to(device)
91         outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
92         _, predicted = torch.max(outputs[1], 1)
93         y_preds.extend(predicted.tolist())
```

➡ `trainer.train()`

Setting up TrainingArguments

```
1 training_args = transformers.TrainingArguments(  
2     output_dir='./results',           # output folder  
3     num_train_epochs=3,               # number of epochs for training  
4     learning_rate=2e-5,              # learning rate  
5     per_device_train_batch_size=16,   # batch size used for training  
6     per_device_eval_batch_size=64,    # batch size used for test time  
7     gradient_accumulation_steps=2,    # training models on bigger batch size  
8     warmup_steps=500,                # hyperparameter for learning rate scheduler  
9     weight_decay=0.01,               # hyperparameter for optimizer  
10    evaluation_strategy='steps',       # time unit to perform evaluation  
11    save_strategy='steps',            # time unit to save checkpoints  
12    save_steps=500,                  # how often to save checkpoints  
13    eval_steps=500,                  # how often to perform evaluation  
14    load_best_model_at_end=True,      # if loading the best checkpoint at the end of training  
15    metric_for_best_model='eval_loss', # how to judge the best model  
16    report_to='tensorboard',         # if saving TensorBoard records  
17    save_total_limit=10,              # maximum number of saved checkpoints  
18    logging_dir='./logs',            # folder for logs  
19    logging_steps=10,                # how often to save logs  
20    seed=random_seed                 # for reproducibility control  
21 )
```

https://huggingface.co/docs/transformers/main_classes/trainer#transformers.TrainingArguments



Build a function for evaluations

```
1 from sklearn.metrics import accuracy_score, precision_recall_fscore_support
2
3 def compute_metrics(pred):
4     labels = pred.label_ids
5     preds = np.argmax(pred.predictions, axis=1)
6     precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
7     acc = accuracy_score(labels, preds)
8     return {
9         'accuracy': acc,
10        'f1': f1,
11        'precision': precision,
12        'recall': recall
13    }
```

Setting up Trainer

```
trainer = transformers.Trainer(  
    model=model, # 🤗 model  
    args=training_args, # the `TrainingArguments` you set  
    train_dataset=train_dataset, # the training dataset  
    eval_dataset=val_dataset, # the evaluation dataset  
    compute_metrics=compute_metrics # evaluation metric  
)  
# Use 1 GPU for training  
trainer.args._n_gpu=1  
  
# start training  
trainer.train()
```

https://huggingface.co/docs/transformers/main_classes/trainer#transformers.Trainer



Perform predictions after training

```
trainer.predict(test_dataset)
```

- This line will call `compute_metrics` using the `test_dataset`.
- Because we are now using `predict()` with `trainer`, it will not perform model updates.