LLMs for **JDM**

AND THE HUGGING FACE
ECOSYSTEM

WHY THE HUGGING FACE?

WHY THE HUGGING FACE?

Traditional fine-tuning pipeline:

WHY THE HUGGING FACE?

Traditional fine-tuning pipeline:

1. Find out the model architecture

1. FINDING ARCHITECTURES DETAILS CAN BE HARD

Is Google word2vec pertrained model CBOW or skipgram

Asked 4 years ago Modified 1 year, 11 months ago Viewed 1k times Part of NLP Collective

1. FINDING ARCHITECTURES DETAILS CAN BE HARD

Is Google word2vec pertrained model CBOW or skipgram

Asked 4 years ago Modified 1 year, 11 months ago Viewed 1k times Part of NLP Collective

2 Answers Sorted by: Highest score (default)

The `GoogleNews` word-vectors were trained by Google, using a proprietary corpus, but they're never explicitly described all the training-parameters used. (It's not encoded in the file.)

It's been asked a number of times on the Google Group devoted to the `word2vec-toolkit` code, without a definitive answer. For example, there's a response from word2vec author Mikolov that he doesn't remember the training parameters. Elsewhere, another poster thinks one of the word2vec papers implies skip-gram was used – but as that passage doesn't precisely match other aspects (like vocabulary-size) of the released `GoogleNews` vectors, I wouldn't be completely confident of that.

WHY THE HUGGING FACE?

Traditional fine-tuning pipeline:

1. Find out the model architecture

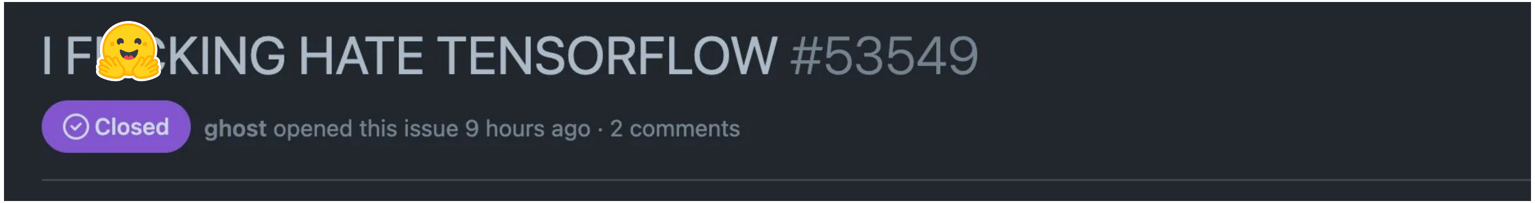
WHY THE HUGGING FACE?

Traditional fine-tuning pipeline:

1. Find out the model architecture
2. Implement the model architecture in code with deep learning frameworks (e.g PyTorch/Tensorflow).

1. DEEP LEARNING LIBRARIES CAN BE … DIFFICULT

1. DEEP LEARNING LIBRARIES CAN BE … DIFFICULT



WHY HUGGING FACE?

Traditional fine-tuning pipeline:

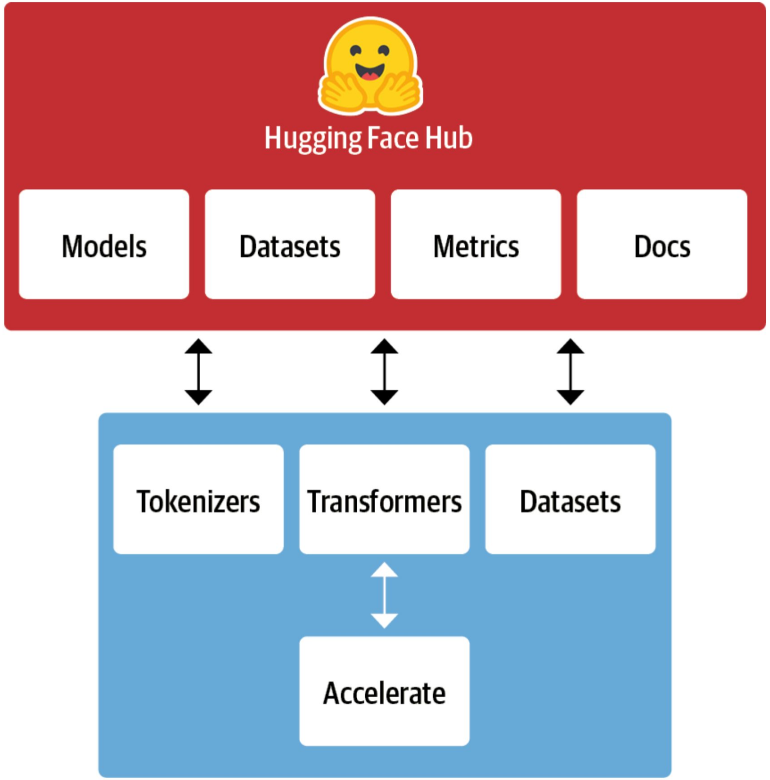
1. Find out the model architecture
2. Implement the model architecture in code with deep learning libraries (e.g PyTorch/Tensorflow).

WHY HUGGING FACE?

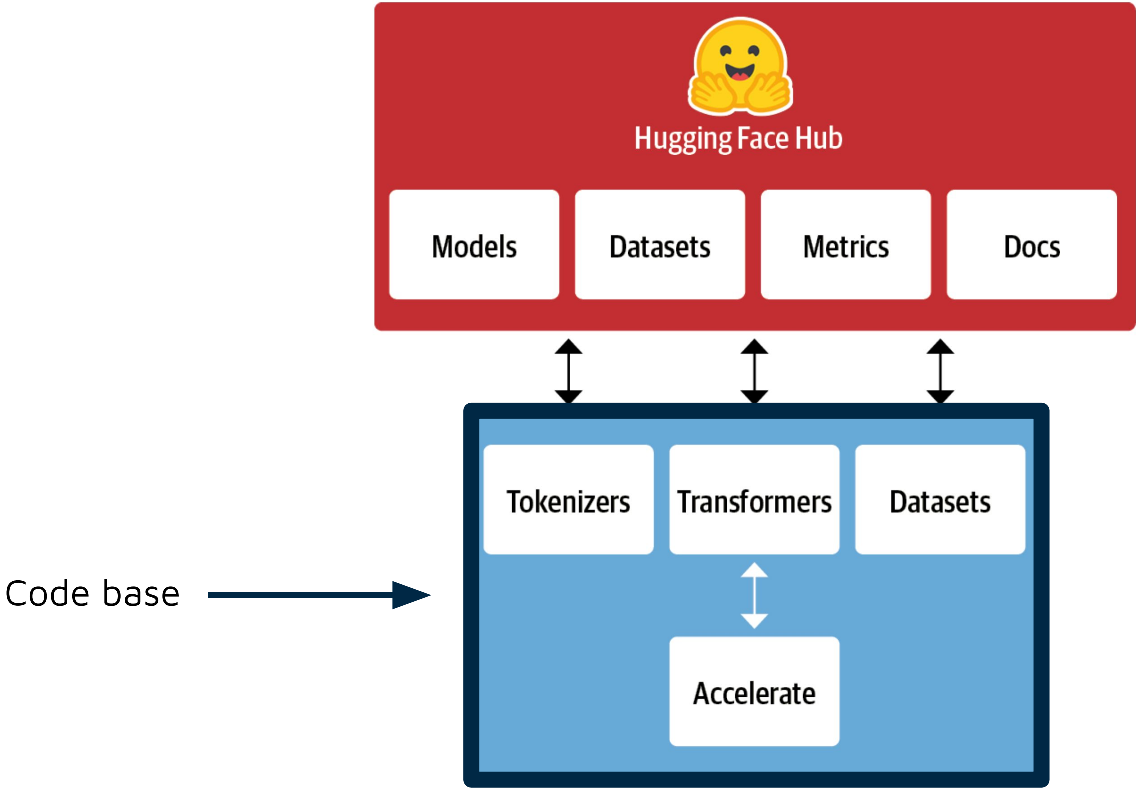
Traditional fine-tuning pipeline:

1. Find out the model architecture
2. Implement the model architecture in code with deep learning libraries (e.g PyTorch/Tensorflow).
3. Load the pretrained weights (if available) from a server.
4. Process the inputs (using the correct tokenizer for the model)
5. Implement data loaders
6. Define a loss function
7. Stick a task-specific “head” on the model

THE HUGGING FACE ECOSYSTEM



THE HUGGING FACE ECOSYSTEM



WHAT NEXT?

1. Feature extraction (exercises 1 and 2)

2. Fine-tuning (exercise 1)

3. Generation (exercise 3)

FEATURE EXTRACTION & PREDICTION

tokenize

```
dat = pd.read_csv('dat.csv')
dat = Dataset.from_pandas(dat)
model_ckpt = 'distilbert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
batch_tokenize = lambda batch: tokenizer(batch['text'])
dat = dat.map(batch_tokenize, batched=True)
```

GPU

```
dat.set_format('torch', columns=['input_ids', 'attention_mask', 'labels'])
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')
model = AutoModel.from_pretrained(model_ckpt).to(device)
```

feature extract

```
def extract_features(batch):
    inputs = {k:v.to(device) for k, v in batch.items() if k in tokenizer.model_input_names}
    with torch.no_grad():
        last_hidden_state = model(**inputs).last_hidden_state
    return {"hidden_state": last_hidden_state[:,0].cpu().numpy()}

dat = dat.map(extract_features, batched=True, batch_size=8)
```

regress

```
embeds = pd.DataFrame(dat['hidden_state'])
X_train, X_test, y_train, y_test = train_test_split(embeds, dat['labels'], random_state=42)
regr = RidgeCV()
regr.fit(X_train, y_train)
```

1. TOKENIZATION

```python
import pandas as pd
from datasets import Dataset
from transformers import AutoTokenizer

# read data from csv file and convert to dataset
dat = pd.read_csv('dat.csv')
dat = Dataset.from_pandas(dat)

# Defining the tokenizer and tokenizing the text
model_ckpt = 'distilbert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
batch_tokenize = lambda batch: tokenizer(
    batch['text'], padding="max_length", truncation=True
)
dat = dat.map(batch_tokenize, batched=True)
```

2. GPU

```python
import torch
torch.manual_seed(0) # for reproducibility
from transformers import AutoModel

# Loading the model and moving it to the GPU if available
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

# Loading distilbert-base-uncased and moving it to the GPU if available
model = AutoModel.from_pretrained(model_ckpt).to(device)
```

3. FEATURE EXTRACTION

```python
# Convert the dataset to PyTorch tensors
dat.set_format('torch', columns=['input_ids', 'attention_mask', 'labels'])

def extract_features(batch):
    # Each batch is a dictionary with keys corresponding to the feature names.
    inputs = {k:v.to(device) for k, v in batch.items() if k in tokenizer.model_input_names}

    # Tell torch not to build the computation graph during inference with `torch.no_grad()`
    with torch.no_grad():
        last_hidden_state = model(**inputs).last_hidden_state # Extract last hidden states

    # Return vector for [CLS] token
    return {"hidden_state": last_hidden_state[:,0].cpu().numpy()}

# Extracting features
dat = dat.map(extract_features, batched=True, batch_size=8)
```

4. REGRESS

```
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import train_test_split

# Converting features to a pandas dataframe for compatibility with sklearn
embeds = pd.DataFrame(dat['hidden_state'])

# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(embeds, dat['labels'], random_state=42)

# Instantiating the RidgeCV model
regr = RidgeCV()

# Fitting the model and evaluating performance
regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
```

FINE-TUNING AND PREDICTION

tokenize

```
dat = Dataset.from_pandas(dat)
model_ckpt = 'distilbert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
batch_tokenize = lambda batch: tokenizer(batch['text'])
dat = dat.map(batch_tokenize, batched=True)
```

GPU

```
dat.set_format('torch', columns=['input_ids', 'attention_mask', 'labels'])
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')
model = AutoModel.from_pretrained(model_ckpt).to(device)
```

feature extract

```
def extract_features(batch):
    inputs = {k:v.to(device) for k, v in batch.items() if k in tokenizer.model_input_names}
    with torch.no_grad():
        last_hidden_state = model(**inputs).last_hidden_state
    return {"hidden_state": last_hidden_state[:,0].cpu().numpy()}

dat = dat.map(extract_features, batched=True, batch_size=8)
```

regress

```
embeds = pd.DataFrame(dat['hidden_state'])
X_train, X_test, y_train, y_test = train_test_split(embeds, dat['labels'], random_state=42)
regr = RidgeCV()
regr.fit(X_train, y_train)
```

FINE-TUNING AND PREDICTION

tokenize

```
dat = Dataset.from_pandas(dat)
model_ckpt = 'distilbert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
batch_tokenize = lambda batch: tokenizer(batch['text'])
dat = dat.map(batch_tokenize, batched=True)
```

GPU

```
dat.set_format('torch', columns=['input_ids', 'attention_mask', 'labels'])
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')
model = AutoModel.from_pretrained(model_ckpt).to(device)
```

fine-tune

```
dat = dat.train_test_split(test_size=0.2)
model = AutoModelForSequenceClassification.from_pretrained(model_ckpt, num_labels=1).to(device)
training_args = TrainingArguments(
    output_dir=finetuned-health, evaluation_strategy="epoch", num_train_epochs=10
)
def compute_metrics(eval_preds):
    metric = evaluate.load("r_squared")
    preds, labels = eval_preds
    return {"r_squared": metric.compute(predictions=preds, references=labels)}

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

4. FINE-TUNING

```
from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer
import evaluate

# Splitting the data into train and test sets
dat = dat.train_test_split(test_size=0.2)

# Loading distilbert-base-uncased and moving it to the GPU if available
model = AutoModelForSequenceClassification.from_pretrained(model_ckpt, num_labels=1).to(device)

# Setting up training arguments for the trainer
model_name = f"{model_ckpt}-finetuned-health"
training_args = TrainingArguments(output_dir=model_name, evaluation_strategy="epoch", num_train_epochs=10)

def compute_metrics(eval_preds):
    """Computes the coefficient of determination (R2) on the test set"""
    metric = evaluate.load("r_squared")
    preds, labels = eval_preds
    return {"r_squared": metric.compute(predictions=preds, references=labels)}

# Instantiating the trainer
trainer = Trainer(
    model=model, args=training_args, train_dataset=dat['train'],
    eval_dataset=dat['test'], compute_metrics=compute_metrics,
)
# Training the model
trainer.train()
```

GENERATION (AND PIPELINES)

```python
from transformers import pipeline

# Initialising generator via pipeline
generator = pipeline("text-generation", model='gpt2')

prompts = [
    'A bat and a ball cost $1.10 in total. The bat costs $1.00 more than the ball. How much does the ball cost?',
    'If it takes 5 machines 5 minutes to make 5 widgets, how long would it take 100 machines to make 100 widgets?',
    'In a lake, there is a patch of lily pads. Every day, the patch doubles in size. If it takes 48 days for the patch
     to cover the entire lake, how long would it take for the patch to cover half of the lake?'
]

outputs = generator(prompts, max_length=100)
```

HUGGING FACE DOCUMENTATION

Documentations

Search across all 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**
  Use more than 50k models through our public inference API, with scalability built-in.
- **Inference Endpoints**
  Easily deploy your model to production on dedicated, fully managed infrastructure.
- **PEFT**
  Parameter efficient finetuning methods for large models

SCHEDULE

09:45 -10:15	Intro to large language models
10:15 - 10:45	Intro to Hugging Face
10:45 - 11:15	Break
11:15 - 12:00	Exercise 1 - Predicting health perception
12:00 - 1:00	Lunch
01:00 - 2:00	Exercise 2 - Predicting personality structure
02:00 - 2:30	Exercise 3 - Predicting cognitive reflection
02:30 - 3:00	Discussion