Tugas Course NLP

Chapter 3

Nama : muhammad makhlufi makbullah

Kelas : TK 45 01

NIM : 1103210171

Chapter 3 Fine-Tuning a Pretrained Model

Fine-tuning adalah proses menyesuaikan model pre-trained pada dataset spesifik untuk suatu tugas tertentu, seperti klasifikasi teks, analisis sentimen, atau pengenalan entitas.Model pre-trained sudah dilatih pada data umum (misalnya, corpus teks besar) dan hanya perlu sedikit penyesuaian untuk tugas spesifik.

Processing the data

1. Menginstal Dependensi

!pip install datasets evaluate transformers[sentencepiece]

Menginstal pustaka datasets, evaluate, dan transformers untuk memproses dataset, mengevaluasi model, dan menggunakan model dari Hugging Face Transformers.

2. Mengimpor Modul yang Diperlukan

import torch

from transformers import AdamW, AutoTokenizer, AutoModelForSequenceClassification

Mengimpor:

- torch untuk tensor dan operasi pelatihan model.
- AdamW sebagai optimizer.
- AutoTokenizer dan AutoModelForSequenceClassification untuk tokenisasi dan model klasifikasi pre-trained.

3. Memuat Tokenizer dan Model

checkpoint = "bert-base-uncased"

tokenizer=AutoTokenizer.from_pretrained(checkpoint)

model = AutoModelForSequenceClassification.from_pretrained(checkpoint)

- tokenizer = AutoTokenizer.from_pretrained(checkpoint)
- model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
- bert-base-uncased adalah model BERT pre-trained.
- AutoTokenizer memuat tokenizer pre-trained untuk model.
- AutoModelForSequenceClassification memuat model BERT dengan lapisan klasifikasi tambahan.

4. Tokenisasi Input

sequences = [

"I've been waiting for a HuggingFace course my whole life.",

"This course is amazing!",

]

batch = tokenizer(sequences, padding=True, truncation=True, return_tensors="pt")

- sequences: List kalimat yang akan diklasifikasikan.
- tokenizer memproses teks dengan:
- padding=True: Menambahkan padding agar semua input memiliki panjang yang sama.
- truncation=True: Memotong teks jika melebihi panjang maksimum.
- return_tensors="pt": Mengembalikan tensor PyTorch.

5. Menambahkan Label

batch["labels"] = torch.tensor([1, 1])

- Menambahkan tensor label (1 menunjukkan sentimen positif) ke batch untuk digunakan dalam pelatihan.

6. Optimizer dan Backpropagation

optimizer = AdamW(model.parameters())

*loss = model(**batch).loss*

loss.backward()

optimizer.step()

- AdamW: Optimizer untuk memperbarui bobot model.
- model(**batch).loss: Menghitung loss dari input dan label.
- loss.backward(): Melakukan backpropagation untuk menghitung gradien.
- optimizer.step(): Memperbarui bobot model berdasarkan gradien.

7. Memuat Dataset

from datasets import load_dataset

raw_datasets = load_dataset("glue", "mrpc")

- Memuat dataset GLUE (tugas Microsoft Research Paraphrase Corpus (MRPC)).

8. Menampilkan Dataset

raw_datasets

raw_train_dataset = raw_datasets["train"]

raw_train_dataset[0]

raw_train_dataset.features

- raw_datasets: Menampilkan keseluruhan dataset (train, validation, test).
- raw_train_dataset: Dataset pelatihan.
- raw_train_dataset[0]: Contoh pertama dalam dataset.
- raw_train_dataset.features: Fitur dataset seperti sentence1, sentence2, dan label.

9. Tokenisasi Dataset

tokenized_sentences_1 = tokenizer(raw_datasets["train"]["sentence1"])

tokenized_sentences_2 = tokenizer(raw_datasets["train"]["sentence2"])

inputs = tokenizer("This is the first sentence.", "This is the second one.")

inputs

tokenizer.convert_ids_to_tokens(inputs["input_ids"])

- tokenized_sentences_1 dan tokenized_sentences_2: Tokenisasi setiap kolom kalimat.
- tokenizer(sentence1, sentence2): Menggabungkan dua kalimat untuk tokenisasi pasangan.
- convert_ids_to_tokens: Mengubah token ID menjadi token aslinya.

10. Tokenisasi Seluruh Dataset

```
tokenized_dataset = tokenizer(
    raw_datasets["train"]["sentence1"],
    raw_datasets["train"]["sentence2"],
    padding=True,
    truncation=True,
)
```

- Tokenisasi seluruh dataset, termasuk padding dan truncation, untuk mendapatkan tensor siap model.

11. Fungsi Tokenisasi

```
def tokenize_function(example):
    return tokenizer(example["sentence1"], example["sentence2"], truncation=True)
```

- Fungsi untuk tokenisasi dataset dengan menggabungkan dua kolom (sentence1 dan sentence2) sambil menerapkan truncation.

12. DatasetDict

```
DatasetDict({
    train: Dataset({
        features: ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2', 'token_type_ids'],
        num_rows: 3668
    })
    ...
})
```

- DatasetDict adalah format dataset terstruktur yang memuat:
- Features: Kolom data seperti input_ids, attention_mask, dan label.
- num_rows: Jumlah sampel dalam dataset.

13. Data Collator

```
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
samples = tokenized_datasets["train"][:8]
samples = {k: v for k, v in samples.items() if k not in ["idx", "sentence1", "sentence2"]}
[len(x) for x in samples["input_ids"]]
batch = data_collator(samples)
{k: v.shape for k, v in batch.items()}
```

- DataCollatorWithPadding: Alat untuk membuat batch data dengan padding otomatis.

- samples: Mengambil contoh dari dataset tokenized untuk batching.
- data_collator(samples): Membuat batch dari contoh tokenized dengan padding.
- v.shape: Menampilkan bentuk tensor untuk memverifikasi batch.

Fine-tuning a model with the Trainer API

1. Menginstal Dependensi

!pip install datasets evaluate transformers[sentencepiece]

- Menginstal pustaka datasets untuk memuat dataset, evaluate untuk evaluasi metrik, dan transformers[sentencepiece] untuk dukungan tokenisasi dengan SentencePiece (digunakan oleh beberapa model seperti T5 dan XLM-R).

2. Memuat Dataset

from datasets import load_dataset

raw_datasets = load_dataset("glue", "mrpc")

- load_dataset memuat dataset dari Hugging Face glue untuk tugas MRPC (Microsoft Research Paraphrase Corpus), yang digunakan untuk klasifikasi pasangan kalimat (apakah kedua kalimat tersebut memiliki arti yang sama).

3. Memuat Tokenizer

from transformers import AutoTokenizer

checkpoint = "bert-base-uncased"

tokenizer = AutoTokenizer.from_pretrained(checkpoint)

- bert-base-uncased adalah model pre-trained BERT.
- AutoTokenizer.from_pretrained memuat tokenizer yang sesuai untuk model BERT tersebut.

4. Fungsi Tokenisasi

def tokenize_function(example):

return tokenizer(example["sentence1"], example["sentence2"], truncation=True)

- Fungsi tokenize_function menerima data example yang berisi pasangan kalimat (sentence1 dan sentence2).
- Tokenizer digunakan untuk memproses kedua kalimat, dengan truncation=True untuk memotong kalimat yang lebih panjang dari panjang maksimum.

5. Tokenisasi Dataset

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)

- map digunakan untuk menerapkan fungsi tokenize_function pada seluruh dataset, memproses kalimat dalam batch (secara bersamaan), dan menghasilkan dataset yang sudah ter-tokenisasi.

6. Menyiapkan Data Collator

from transformers import DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

- DataCollatorWithPadding memastikan bahwa input batch dipadukan ke panjang yang sama, dan padding dilakukan agar semua input dalam batch memiliki panjang yang seragam.

7. Menyiapkan Pengaturan Pelatihan

from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")

- TrainingArguments adalah kelas untuk mendefinisikan argumen atau parameter pelatihan, seperti nama folder output untuk model yang dilatih.
- "test-trainer" adalah nama folder tempat model yang dilatih akan disimpan.

8. Memuat Model

from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)

- AutoModelForSequenceClassification memuat model BERT pre-trained dengan lapisan tambahan untuk tugas klasifikasi sekuens.
- num_labels=2 menandakan bahwa ini adalah tugas klasifikasi dua kelas (apakah pasangan kalimat tersebut memiliki arti yang sama atau tidak).

9. Menyiapkan Trainer

from transformers import Trainer

trainer = Trainer(

model,

training_args,

train_dataset=tokenized_datasets["train"],

eval_dataset=tokenized_datasets["validation"],

data_collator=data_collator,

tokenizer=tokenizer,

)

- Trainer adalah kelas dari Hugging Face yang digunakan untuk menangani pelatihan dan evaluasi model.
- Argumen yang diberikan meliputi:
- model: Model yang akan dilatih.
- training_args: Pengaturan pelatihan yang sudah disiapkan.
- train_dataset dan eval_dataset: Dataset untuk pelatihan dan validasi.
- data_collator: Mengatur padding untuk batch.
- tokenizer: Tokenizer yang digunakan untuk pre-processing input.

10. Melatih Model

trainer.train()

- Memulai proses pelatihan model dengan dataset yang sudah disiapkan dan argumen yang telah ditentukan.

11. Memprediksi dengan Model

predictions = trainer.predict(tokenized_datasets["validation"])

print(predictions.predictions.shape, predictions.label_ids.shape)

- trainer.predict digunakan untuk menghasilkan prediksi pada dataset validasi.
- predictions.predictions.shape dan predictions.label_ids.shape menampilkan dimensi dari prediksi dan label yang digunakan untuk evaluasi.

12. Menghitung Prediksi

import numpy as np

preds = np.argmax(predictions.predictions, axis=-1)

- np.argmax mengambil prediksi dengan nilai tertinggi untuk setiap pasangan kalimat (karena ini adalah klasifikasi dua kelas, model akan memberikan dua nilai logits untuk setiap pasangan).

13. Menghitung Metrik Evaluasi

import evaluate

metric = evaluate.load("glue", "mrpc")

metric.compute(predictions=preds, references=predictions.label_ids)

- evaluate.load("glue", "mrpc") memuat metrik evaluasi untuk tugas GLUE MRPC.

- metric.compute menghitung metrik evaluasi (misalnya, akurasi) berdasarkan prediksi dan label yang sebenarnya.

14. Menghitung Metrik dalam Fungsi

```
def compute_metrics(eval_preds):
    metric = evaluate.load("glue", "mrpc")
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

- compute_metrics adalah fungsi untuk menghitung metrik evaluasi berdasarkan prediksi dan label dari model.
- Metrik ini akan digunakan dalam Trainer untuk evaluasi otomatis selama pelatihan.

15. Mengubah Pengaturan Pelatihan dan Melatih Model

```
training_args = TrainingArguments("test-trainer", evaluation_strategy="epoch")
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)


trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
trainer.train()
```

- evaluation_strategy="epoch": Menetapkan strategi evaluasi setiap akhir epoch (saat pelatihan selesai pada setiap iterasi penuh atas dataset).
- Model, dataset, data collator, dan tokenizer yang telah disiapkan digunakan lagi untuk memulai pelatihan ulang dengan evaluasi metrik.

A full training

1. Menginstal Dependensi

!pip install datasets evaluate transformers[sentencepiece]

!pip install accelerate

- datasets: Memuat dataset dari Hugging Face.
- evaluate: Digunakan untuk mengevaluasi hasil model menggunakan metrik standar.
- transformers[sentencepiece]: Menambahkan dukungan untuk tokenizer SentencePiece.
- accelerate: Menyediakan utilitas untuk memanfaatkan multi-GPU atau perangkat keras lainnya secara lebih efisien.

2. Memuat Dataset

from datasets import load_dataset

raw_datasets = load_dataset("glue", "mrpc")

- load_dataset: Memuat dataset GLUE untuk tugas MRPC (Microsoft Research Paraphrase Corpus).
- raw_datasets berisi data mentah dari dataset MRPC, yang meliputi kolom kalimat dan label.

3. Memuat Tokenizer

from transformers import AutoTokenizer

checkpoint = "bert-base-uncased"

tokenizer = AutoTokenizer.from_pretrained(checkpoint)

- AutoTokenizer.from_pretrained(checkpoint): Memuat tokenizer untuk model BERT bert-base-uncased, yang telah dilatih sebelumnya.

4. Fungsi Tokenisasi

def tokenize_function(example):

return tokenizer(example["sentence1"], example["sentence2"], truncation=True)

- tokenize_function adalah fungsi yang mengambil dua kalimat (sentence1 dan sentence2) dan mengubahnya menjadi token dengan memotong (truncation) jika panjangnya lebih dari batas input model.

5. Tokenisasi Dataset

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)

- map: Menerapkan tokenize_function ke seluruh dataset.
- batched=True: Memproses dataset dalam batch untuk efisiensi.

6. Menyiapkan Data Collator

from transformers import DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

- DataCollatorWithPadding: Menggabungkan batch tokenized sequences, memastikan bahwa semua input memiliki panjang yang sama dengan padding jika perlu.

7. Menghapus dan Mengubah Nama Kolom Dataset

tokenized_datasets = tokenized_datasets.remove_columns(["sentence1", "sentence2", "idx"])

tokenized_datasets = tokenized_datasets.rename_column("label", "labels")

tokenized_datasets.set_format("torch")

- remove_columns: Menghapus kolom yang tidak diperlukan seperti sentence1, sentence2, dan idx.
- rename_column: Mengganti nama kolom label menjadi labels sesuai dengan format yang diharapkan oleh model.
- set_format("torch"): Mengatur format dataset menjadi tensor PyTorch.

8. Menyiapkan DataLoader

from torch.utils.data import DataLoader

train_dataloader = DataLoader(

tokenized_datasets["train"], shuffle=True, batch_size=8, collate_fn=data_collator

)

eval_dataloader = DataLoader(

tokenized_datasets["validation"], batch_size=8, collate_fn=data_collator

)

- DataLoader: Membuat objek DataLoader untuk training dan evaluation, yang memungkinkan pembacaan data dalam batch.
- shuffle=True: Mengacak urutan data pada training set untuk meningkatkan keberagaman saat pelatihan.

9. Mengecek Batch Data

for batch in train_dataloader:

break

{k: v.shape for k, v in batch.items()}

- Melakukan iterasi melalui train_dataloader dan menampilkan dimensi (shape) dari masing-masing elemen dalam batch.

10. Memuat Model

```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

- AutoModelForSequenceClassification: Memuat model BERT yang telah dilatih untuk tugas klasifikasi sekuens.
- num_labels=2: Menandakan bahwa tugas ini adalah klasifikasi dua kelas (paraphrase atau tidak).

11. Mengecek Output Model

```
outputs = model(**batch)

print(outputs.loss, outputs.logits.shape)
```

- Memberikan batch ke model dan mencetak loss dan logits (output dari model).

12. Menyiapkan Optimizer

```
from transformers import AdamW

optimizer = AdamW(model.parameters(), lr=5e-5)
```

- AdamW: Menggunakan optimizer Adam dengan Weight Decay, yang sering digunakan untuk pelatihan model transformer.
- lr=5e-5: Menetapkan learning rate.

13. Menyiapkan Learning Rate Scheduler

```
from transformers import get_scheduler

num_epochs = 3

num_training_steps = num_epochs * len(train_dataloader)

lr_scheduler = get_scheduler(

    "linear",

    optimizer=optimizer,

    num_warmup_steps=0,

    num_training_steps=num_training_steps,

)

print(num_training_steps)
```

- get_scheduler: Membuat learning rate scheduler dengan strategi penurunan linier selama pelatihan.
- num_training_steps adalah total langkah pelatihan berdasarkan jumlah epoch dan ukuran batch.

14. Menyiapkan Perangkat untuk Pelatihan

```
import torch

device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")

model.to(device)

device
```

- Menyiapkan perangkat (GPU atau CPU) untuk pelatihan dan memindahkan model ke perangkat tersebut.

15. Pelatihan dengan Loop Epoch

```
from tqdm.auto import tqdm

progress_bar = tqdm(range(num_training_steps))

model.train()

for epoch in range(num_epochs):
    for batch in train_dataloader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()


        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar.update(1)
```

- tqdm: Digunakan untuk menampilkan progress bar pelatihan.
- Loop: Melakukan pelatihan selama beberapa epoch, memindahkan batch ke perangkat (GPU/CPU), melakukan backward pass, dan memperbarui optimizer.

16. Evaluasi Model

```
import evaluate
metric = evaluate.load("glue", "mrpc")
model.eval()
for batch in eval_dataloader:
    batch = {k: v.to(device) for k, v in batch.items()}
    with torch.no_grad():
        outputs = model(**batch)

    logits = outputs.logits
    predictions = torch.argmax(logits, dim=-1)
    metric.add_batch(predictions=predictions, references=batch["labels"])
metric.compute()
```

- Evaluasi: Menghitung metrik GLUE (MRPC) untuk validasi.
- model.eval(): Menandakan bahwa model dalam mode evaluasi.
- torch.no_grad(): Menonaktifkan perhitungan gradien saat evaluasi untuk menghemat memori.
- metric.add_batch: Menambahkan prediksi dan referensi ke metrik evaluasi.

17. Melatih dengan Accelerate untuk Multi-GPU

```
from accelerate import Accelerator
accelerator = Accelerator()
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
optimizer = AdamW(model.parameters(), lr=3e-5)
train_dl, eval_dl, model, optimizer = accelerator.prepare(
    train_dataloader, eval_dataloader, model, optimizer
)
```

- Accelerator: Menyediakan API untuk melakukan pelatihan multi-GPU secara otomatis.
- accelerator.prepare: Menyiapkan dataloader, model, dan optimizer untuk multi-GPU (atau perangkat keras lainnya).

18. Melakukan Pelatihan dengan Accelerator

```
num_epochs = 3
num_training_steps = num_epochs * len(train_dl)
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps,
)
progress_bar = tqdm(range(num_training_steps))
model.train()
for epoch in range(num_epochs):
    for batch in train_dl:
        outputs = model(**batch)
        loss = outputs.loss
        accelerator.backward(loss)
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar.update(1)
```

- Melakukan pelatihan dengan Accelerator yang mendukung pelatihan di banyak GPU atau perangkat keras.
- accelerator.backward(loss) digunakan untuk melakukan backward pass.

19. Meluncurkan Pelatihan di Notebook

```
from accelerate import notebook_launcher
notebook_launcher(training_function)
```

- notebook_launcher digunakan untuk menjalankan pelatihan di lingkungan notebook dengan Accelerator.

apa yang telah dipelajari mengenai **fine-tuning** model pre-trained pada bab ini:

1. **Mengenal Dataset di Hugging Face Hub**:
    - Kita telah mempelajari cara memuat dan menggunakan dataset yang ada di Hugging Face Hub, seperti dataset GLUE untuk tugas klasifikasi teks.
2. **Memuat dan Memproses Dataset**:
    - Kita belajar cara memuat dataset dan melakukan preprocessing dengan tokenizer, termasuk menggunakan padding dinamis dan collators untuk menyusun data agar dapat digunakan dalam model.
3. **Melakukan Fine-Tuning dan Evaluasi Model**:
    - Kita telah mengimplementasikan fine-tuning model pre-trained seperti BERT untuk menyesuaikan dengan data kita.
    - Selama proses ini, kita juga melakukan evaluasi untuk memeriksa performa model yang telah dilatih menggunakan metrik yang sesuai.
4. **Membangun Loop Pelatihan Secara Manual**:
    - Kita belajar bagaimana membuat loop pelatihan manual dengan menggunakan optimizer, loss function, dan learning rate scheduler.
5. **Menggunakan 🤗 Accelerate untuk Pelatihan Multi-GPU/TPU**:
    - Kita mempelajari bagaimana **Accelerate** dapat digunakan untuk mempercepat pelatihan dengan mendistribusikan tugas di beberapa GPU atau TPU, tanpa perlu menulis kode yang rumit untuk paralelisasi.