Nama : Muhammad Thoriq Zam
NIM : 1103210008

1. Pendahuluan

Model Transformer merevolusi Natural Language Processing (NLP) dengan memperkenalkan arsitektur yang unggul dalam menangani data berurutan. Tidak seperti model tradisional, Transformer secara efektif menangkap hubungan jangka panjang dengan menggunakan mekanisme seperti self-attention dan positional encoding.

2. Pemrosesan Bahasa Alami (NLP)

NLP berfokus pada mengajarkan mesin untuk memahami dan menghasilkan bahasa manusia. Transformer banyak digunakan dalam tugas seperti:

- Analisis Sentimen
- Penjawaban Pertanyaan
- Ringkasan Teks
- Penerjemahan Bahasa

3. Apa yang Bisa Dilakukan Transformer?

Transformer dapat menangani berbagai tugas dalam NLP, termasuk:

- Klasifikasi Teks
- Named Entity Recognition (NER)
- Generasi Teks
- Penerjemahan Bahasa
- Ringkasan Teks

4. Bagaimana Cara Kerja Transformer?

Transformer mengandalkan komponen kunci berikut:

1. **Self-Attention**: Menentukan pentingnya setiap kata dalam sebuah urutan terhadap kata lainnya.
2. **Positional Encoding**: Mengodekan urutan kata ke dalam model.
3. **Feedforward Neural Network**: Memproses output dari attention menjadi representasi yang bermakna.

```
# Contoh Kode untuk Memahami Cara Kerja Transformer
from transformers import AutoModel, AutoTokenizer

# Memuat model dan tokenizer pra-latih
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

# Contoh teks
text = "Transformers are powerful for NLP tasks!"
inputs = tokenizer(text, return_tensors="pt")
outputs = model(**inputs)

# Cetak bentuk tensor output
print("Output shape:", outputs.last_hidden_state.shape)
```

Kode tersebut memanfaatkan pustaka Hugging Face Transformers untuk memproses teks menggunakan model pra-latih BERT (Bidirectional Encoder Representations from Transformers). Pertama, model dan tokenizer BERT dimuat melalui AutoModel dan AutoTokenizer dengan nama model "bert-base-uncased", yang merupakan versi BERT tanpa pembedaan huruf besar/kecil. Teks input "Transformers are powerful for NLP tasks!" ditokenisasi menggunakan tokenizer untuk diubah menjadi representasi numerik berupa tensor PyTorch. Tensor ini kemudian diproses oleh model BERT, menghasilkan output berupa representasi vektor (hidden states) dari setiap token dalam teks, dengan dimensi mencakup ukuran batch, jumlah token, dan panjang vektor (768 untuk model BERT). Output ini mencerminkan representasi kaya konteks dari tiap token, yang dapat digunakan untuk tugas NLP lanjutan seperti klasifikasi teks, ekstraksi entitas, atau penilaian kemiripan teks.

```
# Contoh Kode BERT
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")

# Encode dan proses sebuah kalimat
sentence = "Hugging Face makes NLP easier!"
encoded_input = tokenizer(sentence, return_tensors='pt')
output = model(**encoded_input)
print("BERT Output Shape:", output.last_hidden_state.shape)
```

Kode tersebut menggunakan model pra-latih BERT (Bidirectional Encoder Representations from Transformers) dan tokenizer dari pustaka Hugging Face Transformers untuk memproses teks. Tokenizer BERT (BertTokenizer) dan model (BertModel) dimuat menggunakan model pra-latih "bert-base-uncased", yang mengabaikan perbedaan huruf besar dan kecil. Kalimat contoh "Hugging Face makes NLP easier!" ditokenisasi dengan fungsi tokenizer, yang mengonversi teks menjadi representasi numerik berupa tensor PyTorch. Tensor ini kemudian diproses oleh model BERT, menghasilkan representasi kaya konteks (hidden states) untuk setiap token dalam teks. Hasil akhirnya berupa tensor dengan dimensi yang mencerminkan jumlah token dalam teks, panjang vektor representasi (768 untuk BERT), dan ukuran batch. Proses ini merupakan dasar untuk memanfaatkan BERT dalam tugas-tugas NLP seperti klasifikasi teks atau ekstraksi fitur.

6. Model Decoder

Model decoder seperti GPT digunakan untuk tugas generatif seperti pelengkapan teks dan generasi teks.

```
# Contoh Kode GPT
from transformers import GPT2Tokenizer, GPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

prompt = "Hugging Face is"
inputs = tokenizer.encode(prompt, return_tensors="pt")
outputs = model.generate(inputs, max_length=50, num_return_sequences=1)
```

```
print("Generated Text:", tokenizer.decode(outputs[0]))
```

7. Model Sequence-to-Sequence

Model sequence-to-sequence seperti T5 atau BART digunakan untuk tugas seperti penerjemahan dan ringkasan teks.

```
# Contoh Kode T5
from transformers import T5Tokenizer, T5ForConditionalGeneration

tokenizer = T5Tokenizer.from_pretrained("t5-small")
model = T5ForConditionalGeneration.from_pretrained("t5-small")

# Menerjemahkan Bahasa Inggris ke Bahasa Prancis
text = "translate English to French: Hugging Face is amazing."
inputs = tokenizer.encode(text, return_tensors="pt")
outputs = model.generate(inputs)

print("Translated Text:", tokenizer.decode(outputs[0]))
```

Kode tersebut menggunakan model T5 (Text-to-Text Transfer Transformer) dari pustaka Hugging Face Transformers untuk melakukan tugas terjemahan dari Bahasa Inggris ke Bahasa Prancis. Model dan tokenizer T5 dimuat menggunakan varian pra-latih "t5-small", yang dirancang untuk menangani berbagai tugas NLP dalam format input-output berbasis teks. Kalimat input "translate English to French: Hugging Face is amazing." dikodekan oleh tokenizer menjadi representasi numerik berupa tensor PyTorch menggunakan fungsi tokenizer.encode. Tensor ini kemudian diproses oleh model T5 melalui fungsi model.generate, yang menghasilkan output berupa token-token terjemahan dalam Bahasa Prancis. Terjemahan ini akhirnya didekodekan kembali menjadi teks dengan fungsi tokenizer.decode, menghasilkan teks terjemahan yang ditampilkan melalui perintah print. Pendekatan ini menunjukkan fleksibilitas T5 dalam mengubah tugas NLP apa pun menjadi masalah transformasi teks-ke-teks.

8. Bias dan Keterbatasan

Transformer sangat kuat tetapi memiliki tantangan:

- Biaya komputasi yang tinggi
- Membutuhkan dataset yang besar
- Dapat mewarisi bias dari data pelatihan

9. Ringkasan

- Transformer telah merevolusi NLP dengan kemampuan mereka menangani data berurutan secara efektif.
- Komponen utama termasuk self-attention, positional encoding, dan feedforward networks.
- Jenis populer termasuk model encoder (misalnya, BERT), model decoder (misalnya, GPT), dan model sequence-to-sequence (misalnya, T5).

Unit 2: Menggunakan Transformers

1. Pendahuluan

Bagian ini membahas bagaimana menggunakan pustaka Transformers untuk tugas NLP dengan berbagai fitur bawaan. Anda akan belajar cara menggunakan pipeline untuk menyederhanakan tugas-tugas seperti klasifikasi teks, tokenisasi, dan memanfaatkan model Transformer secara praktis.

2. Di Balik Pipeline

Pipeline adalah abstraksi tingkat tinggi yang ditawarkan oleh pustaka Transformers untuk menyederhanakan penggunaan model. Di balik layar, pipeline:

1. Memuat model dan tokenizer yang relevan.
2. Memproses input (tokenisasi).
3. Melakukan inferensi menggunakan model.
4. Mengembalikan hasil dengan format yang mudah dipahami.

```
# Contoh Kode: Menggunakan Pipeline untuk Klasifikasi Teks
from transformers import pipeline

# Inisialisasi pipeline untuk klasifikasi sentimen
classifier = pipeline("sentiment-analysis")
```

```
# Teks untuk analisis
text = "Hugging Face is an amazing library!"
result = classifier(text)
print(result)
```

Kode ini menggunakan fungsi **pipeline** dari pustaka **Hugging Face Transformers** untuk melakukan klasifikasi sentimen secara sederhana dan cepat. Pipeline adalah antarmuka tingkat tinggi yang memungkinkan pengguna untuk langsung menjalankan berbagai tugas NLP, seperti klasifikasi sentimen, tanpa perlu mengatur model atau tokenizer secara manual. Dalam kode ini, pipeline diinisialisasi dengan parameter "sentiment-analysis", yang secara otomatis memuat model dan tokenizer yang telah dilatih untuk klasifikasi sentimen. Kalimat "Hugging Face is an amazing library!"dianalisis oleh pipeline, dan hasil klasifikasi berupa label sentimen (misalnya, "POSITIVE" atau "NEGATIVE") beserta skor kepercayaan dikembalikan sebagai output. Hasil tersebut kemudian ditampilkan menggunakan perintah print. Pendekatan ini sangat berguna untuk menerapkan tugas NLP tanpa memerlukan konfigurasi yang kompleks.

3. Model

Model Transformer adalah inti dari pipeline. Pustaka Transformers menyediakan berbagai model seperti BERT, GPT-2, dan T5. Anda dapat memuat model spesifik untuk tugas tertentu.

```
# Contoh Kode: Memuat Model Secara Manual
from transformers import AutoModelForSequenceClassification, AutoTokenizer

# Memuat model dan tokenizer
model_name = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModelForSequenceClassification.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Tokenisasi teks
inputs = tokenizer("I love using Hugging Face!", return_tensors="pt")
outputs = model(**inputs)
print(outputs)
```

Kode ini memuat model Transformer untuk tugas klasifikasi teks secara manual menggunakan pustaka Hugging Face Transformers. Model dan tokenizer dimuat dengan nama pra-latih "distilbert-base-uncased-finetuned-sst-2-english", yaitu versi DistilBERT yang telah di-finetune untuk analisis sentimen pada dataset SST-2. Tokenizer (AutoTokenizer) mengonversi teks input,

"I love using Hugging Face!", menjadi tensor PyTorch melalui proses tokenisasi dengan parameter return_tensors="pt". Tensor ini berisi informasi yang diperlukan, seperti ID token dan perhatian (attention masks). Tensor hasil tokenisasi kemudian diproses oleh model (AutoModelForSequenceClassification), yang menghasilkan output berupa skor logit untuk setiap label kelas (misalnya, "POSITIVE" atau "NEGATIVE"). Output tersebut ditampilkan melalui perintah print. Kode ini memungkinkan pengguna untuk menjalankan model klasifikasi teks dengan kontrol manual atas model dan tokenizer yang digunakan.

4. Tokenizers

Tokenizer mengonversi teks mentah menjadi token numerik yang dapat diproses oleh model. Ada berbagai metode tokenisasi:

- Word-based: Memisahkan teks berdasarkan kata.
- Subword-based: Memecah kata menjadi sub-unit (digunakan oleh BERT, GPT-2).
- Character-based: Tokenisasi pada tingkat karakter.

```
# Contoh Kode: Menggunakan Tokenizer
from transformers import AutoTokenizer

# Memuat tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Tokenisasi teks
text = "Transformers are incredible!"
tokens = tokenizer.tokenize(text)
ids = tokenizer.convert_tokens_to_ids(tokens)
print("Tokens:", tokens)
print("Token IDs:", ids)
```

Kode ini menunjukkan cara menggunakan tokenizer dari pustaka Hugging Face Transformers untuk memproses teks secara manual. Tokenizer dimuat dengan model pra-latih "bert-base-uncased", yang merupakan versi BERT tanpa pembedaan huruf besar dan kecil. Teks input "Transformers are incredible!" diproses menggunakan metode tokenizer.tokenize, yang memecah teks menjadi token-token individu berdasarkan aturan tokenisasi model. Token-token ini kemudian diubah menjadi ID numerik yang sesuai dengan kosakata model menggunakan metode tokenizer.convert_tokens_to_ids. Hasilnya adalah daftar token dan ID token yang merepresentasikan teks dalam format numerik yang dapat dipahami oleh model Transformer.

Output ini ditampilkan melalui perintah print, memberikan wawasan tentang bagaimana teks diubah menjadi input untuk model.

5. Menangani Banyak Sekuens

Transformer dirancang untuk menangani beberapa sekuens sekaligus, yang dikenal sebagai pemrosesan batch. Hal ini memungkinkan model untuk memproses kumpulan data yang lebih besar secara paralel, meningkatkan efisiensi.

Pustaka Transformers menyediakan opsi seperti padding dan truncation untuk memastikan bahwa input dalam batch memiliki panjang yang seragam:

- **Padding**: Menambahkan token khusus (seperti [PAD]) ke teks yang lebih pendek untuk mencocokkan panjang teks terpanjang dalam batch.
- **Truncation**: Memotong teks yang melebihi panjang maksimum yang diperbolehkan oleh model.

```
# Contoh Kode: Memproses Banyak Sekuens
texts = ["I love NLP!", "Transformers are amazing.", "Batch processing is efficient."]
# Tokenisasi teks dengan padding dan truncation
inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt")

# Hapus token_type_ids untuk kompatibilitas dengan DistilBERT
inputs.pop("token_type_ids", None)

# Inferensi dengan model
outputs = model(**inputs)

# Menampilkan hasil
print("Input IDs:", inputs["input_ids"])
print("Attention Mask:", inputs["attention_mask"])
print("Logits:", outputs.logits)
```

Kode ini menunjukkan cara memproses banyak teks sekaligus menggunakan tokenizer dan model Transformer dari pustaka **Hugging Face Transformers**. Daftar teks (texts) berisi beberapa kalimat diproses oleh tokenizer dengan parameter padding=True untuk memastikan semua input memiliki panjang yang sama, dan truncation=True untuk memotong teks yang terlalu panjang agar sesuai dengan batas panjang model. Output tokenisasi dikembalikan sebagai tensor PyTorch melalui parameter return_tensors="pt". Karena model yang digunakan adalah DistilBERT (tidak

memerlukan token_type_ids), atribut ini dihapus dari input menggunakan inputs.pop. Tensor hasil tokenisasi kemudian diberikan ke model untuk inferensi, menghasilkan output berupa **logits** yang merepresentasikan skor untuk setiap label kelas. Hasil tokenisasi seperti ID token (input_ids), perhatian (attention_mask), dan **logits** dari model ditampilkan melalui perintah print. Pendekatan ini mendemonstrasikan pemrosesan batch yang efisien untuk tugas NLP dengan model Transformer.

6. Menggabungkan Semua

Pada bagian ini, kita akan menggabungkan semua langkah yang telah dijelaskan sebelumnya untuk membangun alur kerja NLP lengkap menggunakan 🤗 Transformers. Dalam implementasi produksi, langkah-langkah ini dapat diintegrasikan ke dalam pipeline otomatis dengan fitur seperti logging, pengelolaan kesalahan, dan integrasi API.

Penerapan dalam Pipeline Produksi

1. **Preprocessing**:
    - Langkah tokenisasi dan normalisasi teks dilakukan secara otomatis untuk memastikan data konsisten.
    - Data diproses dalam batch untuk efisiensi.
2. **Model Serving**:
    - Model dilayani menggunakan framework seperti TensorFlow Serving atau FastAPI untuk inferensi real-time.
    - Endpoint API menyediakan akses ke model dengan latensi rendah.
3. **Postprocessing**:
    - Hasil prediksi (logits atau probabilitas) diterjemahkan ke dalam format yang mudah dipahami oleh pengguna akhir.
4. **Monitoring**:
    - Performa model dipantau secara berkala untuk mendeteksi degradasi kinerja.
    - Logging digunakan untuk melacak permintaan dan menganalisis anomali.

Pipeline ini cocok untuk apl

```
# Contoh Kode: Klasifikasi Sentimen Lengkap
text = "Hugging Face simplifies NLP!"
```

```
# Tokenisasi teks
inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
inputs.pop("token_type_ids", None)  # Hapus token_type_ids jika ada

# Inferensi dengan model
outputs = model(**inputs)
logits = outputs.logits

# Menghitung prediksi
predictions = torch.nn.functional.softmax(logits, dim=-1)
predicted_class = torch.argmax(predictions, dim=-1).item()

# Menampilkan hasil
print("Logits:", logits)
print("Probabilities:", predictions)
print("Predicted Class:", predicted_class)
```

7. Penggunaan Dasar Selesai!

Bagian ini merangkum langkah-langkah yang telah dibahas dan menjelaskan bagaimana mereka membentuk fondasi untuk menggunakan 🤗 Transformers dalam tugas-tugas NLP. Berikut adalah poin-poin utama yang perlu diingat:

1. **Pipeline Abstraksi Tingkat Tinggi**:
    - Pipeline mempermudah implementasi tugas umum seperti klasifikasi sentimen, penerjemahan, atau analisis teks.
    - Ini sangat cocok untuk memulai tanpa perlu memahami detail teknis model atau tokenisasi.
2. **Pemahaman Komponen Utama**:
    - Memahami cara kerja model Transformer, seperti BERT, GPT, dan T5.
    - Tokenizer memainkan peran penting dalam mempersiapkan data untuk model dengan memastikan format input sesuai.
3. **Pemrosesan Batch untuk Efisiensi**:
    - Pemrosesan batch memungkinkan model untuk menangani banyak input secara paralel, yang berguna dalam aplikasi skala besar.

4. **Produksi dan Integrasi**:
    - Dalam aplikasi produksi, model dapat dilayani melalui API untuk inferensi real-time.
    - Pipeline ini dapat diintegrasikan dengan framework lain seperti FastAPI atau Flask untuk membuat layanan prediksi.
5. **Monitoring dan Pemeliharaan**:
    - Sangat penting untuk memantau performa model secara berkala, termasuk logging dan analisis anomali untuk mendeteksi degradasi performa.

Dengan menguasai langkah-langkah ini, Anda memiliki dasar yang kuat untuk mengembangkan solusi NLP menggunakan Transformers.

Dengan memanfaatkan pipeline, model, tokenizer, dan teknik pemrosesan batch, Anda sekarang dapat dengan mudah mengimplementasika

Unit 3: Fine-Tuning a Pretrained Model

1. Introduction

Fine-tuning adalah proses menyesuaikan model pra-latih (pretrained model) untuk tugas spesifik tertentu dengan menggunakan dataset yang lebih kecil dan domain-spesifik. Model pra-latih seperti BERT, GPT, atau T5 telah dilatih pada dataset besar untuk memahami pola umum dalam bahasa alami. Dengan fine-tuning, kita mengambil pengetahuan ini dan menyesuaikannya dengan tugas seperti:

1. **Klasifikasi Teks**: Memilah teks ke dalam kategori tertentu, seperti analisis sentimen atau deteksi spam.
2. **Pengenalan Entitas Bernama (NER)**: Mengenali nama orang, lokasi, organisasi, atau informasi lainnya dalam teks.
3. **Terjemahan Bahasa**: Mengubah teks dari satu bahasa ke bahasa lain.
4. **Ringkasan Teks**: Menghasilkan ringkasan pendek dari dokumen panjang.

Fine-tuning membantu meningkatkan performa pada tugas spesifik tanpa perlu melatih model dari awal, yang dapat menghemat waktu dan sumber daya komputasi secara signifikan. Dalam bab ini, Anda akan mempelajari:

- Bagaimana memproses data untuk fine-tuning.
- Menggunakan API seperti Trainer dari Hugging Face untuk melatih model.
- Mengevaluasi dan menyimpan model yang telah dilatih.

Bagian ini membahas konsep fine-tuning model pra-latih. Model pra-latih, seperti BERT, GPT, atau T5, telah dilatih pada dataset besar, dan fine-tuning memungkinkan kita untuk menyesuaikan model ini dengan tugas spesifik, seperti analisis sentimen atau klasifikasi teks.

2. Processing the Data

Langkah pemrosesan data adalah fondasi penting dalam fine-tuning model. Data harus dipersiapkan agar sesuai dengan format model yang digunakan. Proses ini melibatkan:

1. **Memuat Dataset**:
    - Dataset dapat berupa data publik yang tersedia di pustaka seperti Hugging Face Datasets atau data custom yang disiapkan pengguna.
2. **Tokenisasi Data**:
    - Menggunakan tokenizer model pra-latih untuk mengonversi teks menjadi token numerik yang dimengerti oleh model.
    - Memastikan teks memiliki panjang yang seragam dengan padding (menambahkan token hingga panjang tertentu) atau truncation (memotong teks yang terlalu panjang).
3. **Membuat Dataset untuk Pelatihan dan Validasi**:
    - Memisahkan data ke dalam set pelatihan, validasi, dan test untuk mengevaluasi performa model secara obyektif.

Langkah-langkah ini memastikan bahwa model dapat belajar dari data dengan cara yang optimal.

```
# Contoh Kode: Memproses Data
```

```
from datasets import load_dataset
from transformers import AutoTokenizer

# Memuat dataset
raw_datasets = load_dataset("imdb")

# Memuat tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Tokenisasi data
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, padding=True)

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

Penjelasan:

1. **load_dataset**: Memuat dataset IMDb, dataset umum untuk analisis sentimen yang berisi ulasan film.
2. **tokenizer**: Menggunakan tokenizer BERT untuk mengubah teks menjadi token numerik.
3. **tokenize_function**: Fungsi ini secara otomatis memotong teks yang terlalu panjang dan menambahkan padding ke teks yang lebih pendek.
4. **map**: Menerapkan fungsi tokenisasi pada setiap sampel dataset secara batch untuk mempercepat proses.

Hasil dari proses ini adalah dataset yang siap digunakan untuk pelatihan model.

Sebelum melatih model, data perlu diproses agar sesuai dengan format yang diperlukan. Langkah-langkah ini meliputi:

1. Memuat dataset.
2. Melakukan tokenisasi menggunakan tokenizer model.
3. Membuat dataset untuk pelatihan dan validasi.

```
# Contoh Kode: Memproses Data
from datasets import load_dataset
from transformers import AutoTokenizer

# Memuat dataset
```

```
raw_datasets = load_dataset("imdb")

# Memuat tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Tokenisasi data
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, padding=True)

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

Penjelasan:

1. load_dataset: Memuat dataset IMDb untuk analisis sentimen.
2. tokenize_function: Melakukan tokenisasi dengan memotong teks panjang dan menambahkan padding.
3. map: Menerapkan fungsi tokenisasi pada seluruh dataset.

3. Fine-Tuning a Model with the Trainer API or Keras

Hugging Face menyediakan API bernama Trainer untuk melatih model dengan mudah. Jika menggunakan TensorFlow, kita juga bisa menggunakan Keras untuk fine-tuning model.

```
# Contoh Kode: Fine-Tuning dengan Trainer API
from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer

# Memuat model
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)

# Menyiapkan argumen pelatihan
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
)
```

```
# Menyiapkan Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["test"],
)

# Melatih model
trainer.train()
```

Penjelasan:

1. AutoModelForSequenceClassification: Model BERT disesuaikan untuk klasifikasi dengan 2 label.
2. TrainingArguments: Mengatur parameter pelatihan seperti learning rate, batch size, dan jumlah epoch.
3. Trainer: API untuk melatih dan mengevaluasi model secara otomatis.

4. A Full Training

Langkah pelatihan penuh mencakup semua tahapan dari awal hingga akhir untuk menyesuaikan model pra-latih dengan dataset spesifik. Berikut adalah langkah-langkah utama yang perlu dilakukan:

1. **Persiapan Dataset**:
    - Dataset dibagi menjadi tiga bagian: train (pelatihan), validation (validasi), dan test (pengujian).
    - Pembagian dataset ini memastikan model dilatih pada data pelatihan, disesuaikan menggunakan data validasi, dan diuji performanya pada data test.
2. **Tokenisasi dan Preprocessing**:
    - Data teks mentah diubah menjadi token yang dapat diproses oleh model.
    - Padding dan truncation diterapkan untuk memastikan keseragaman panjang input.
3. **Pelatihan Model**:

- Model dilatih menggunakan Trainer API yang menangani proses seperti penjadwalan learning rate, logging, dan evaluasi otomatis.

4. **Evaluasi Model**:
    - Model dievaluasi pada dataset validasi untuk mengukur performanya selama pelatihan.
    - Setelah selesai, model diuji pada dataset test untuk memastikan kemampuan generalisasi.

Fine-tuning yang baik memerlukan pemantauan yang cermat terhadap metrik seperti loss dan akurasi pada setiap epoch untuk mencegah overfitting atau underfitting.

```
# Contoh Kode: Evaluasi Model
# Evaluasi model pada dataset test
eval_results = trainer.evaluate()
print("Evaluation Results:", eval_results)
```

Penjelasan:

1. **evaluate**: Fungsi ini digunakan untuk mengukur performa model pada dataset test.
2. **Metrik Evaluasi**: Hasil evaluasi meliputi metrik penting seperti:
    - Akurasi: Persentase prediksi benar dibandingkan total data.
    - Loss: Indikator error model pada dataset test.
3. **Output Evaluasi**: Hasil evaluasi ini dapat digunakan untuk membandingkan performa model dengan baseline atau model lainnya.

Hasil evaluasi memastikan model telah dilatih dengan baik dan dapat diandalkan untuk tugas spesifik.

Langkah pelatihan penuh mencakup:

1. Menyiapkan dataset (train/test split).
2. Melakukan tokenisasi.
3. Melatih model dengan Trainer API.
4. Mengevaluasi model pada data validasi atau test set.

```
# Contoh Kode: Evaluasi Model
# Evaluasi model pada dataset test
```

```
eval_results = trainer.evaluate()
print("Evaluation Results:", eval_results)
```

Penjelasan:

1. evaluate: Mengevaluasi performa model pada dataset test.
2. Hasil evaluasi meliputi metrik seperti akurasi dan loss.

5. Fine-Tuning, Check!

Setelah fine-tuning, model yang telah disesuaikan siap digunakan untuk tugas spesifik. Berikut adalah langkah-langkah utama yang perlu dilakukan setelah pelatihan selesai:

1. **Menyimpan Model yang Telah Dilatih**:
    - Model yang telah dilatih harus disimpan ke dalam direktori untuk digunakan di masa mendatang.
    - Folder ini akan berisi file model, tokenizer, dan konfigurasi.
2. **Memuat Model untuk Inferensi**:
    - Model yang telah disimpan dapat dimuat kembali menggunakan pipeline untuk prediksi.
    - Pipeline menyediakan cara mudah untuk menggunakan model tanpa harus menulis ulang kode tokenisasi atau prediksi.
3. **Pengujian pada Data Baru**:
    - Model harus diuji dengan data yang belum pernah dilihat sebelumnya untuk memastikan performanya tetap baik.
    - Ini penting untuk memastikan bahwa model tidak overfitting pada dataset pelatihan.
4. **Integrasi dengan Aplikasi**:
    - Model yang telah dilatih dapat diintegrasikan ke dalam aplikasi atau sistem, seperti API berbasis FastAPI atau Flask.

Langkah-langkah ini memastikan bahwa model siap untuk digunakan dalam produksi atau pengujian lebih lanjut.

```
# Contoh Kode: Menggunakan Model yang Dilatih
from transformers import pipeline

# Memuat pipeline klasifikasi sentimen
sentiment_model = pipeline("sentiment-analysis", model="./results")

# Prediksi pada teks baru
result = sentiment_model("I love using Hugging Face for NLP tasks!")
print(result)
```

Penjelasan:

1. **Memuat Pipeline**:
    - Pipeline memuat model, tokenizer, dan konfigurasi dari folder yang telah disimpan.
2. **Inferensi**:
    - Model digunakan untuk membuat prediksi pada teks baru dengan cara yang sederhana.
3. **Hasil Prediksi**:
    - Output berupa prediksi label (misalnya, "POSITIVE" atau "NEGATIVE") dan skor kepercayaan untuk label tersebut.

Hasil ini dapat digunakan untuk evaluasi lebih lanjut atau diintegrasikan ke aplikasi produksi.

Setelah fine-tuning, model dapat digunakan untuk tugas spesifik. Pastikan model telah disimpan dan diuji dengan benar.

Unit 4: Sharing Models and Tokenizers

1. The Hugging Face Hub

Hugging Face Hub adalah platform untuk berbagi model, tokenizer, dan dataset. Platform ini memfasilitasi kolaborasi antar peneliti dan pengembang dengan menyediakan repositori berbasis cloud untuk menyimpan dan mengakses model.

Fitur utama:

- Menyimpan dan berbagi model.
- Dokumentasi model melalui "Model Cards".
- Akses API untuk mengunduh model secara langsung.

```
# Contoh Kode: Mengunggah Model ke Hugging Face Hub
from huggingface_hub import notebook_login
from transformers import AutoModelForSequenceClassification, AutoTokenizer

# Login ke akun Hugging Face
notebook_login()

# Menyimpan model lokal
model_name = "bert-base-uncased"
model = AutoModelForSequenceClassification.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

model.save_pretrained("./my_model")
tokenizer.save_pretrained("./my_model")

# Mengunggah model ke Hub
from huggingface_hub import HfApi

api = HfApi()
api.upload_folder(
    folder_path="./my_model",  # Folder berisi model dan tokenizer
    path_in_repo="bert-base-custom",  # Nama repositori di Hub
    repo_id="username/bert-base-custom",  # ID repositori Anda
    repo_type="model"  # Tipe repositori (model/dataset/tokenizer)
)
```

1. notebook_login: Autentikasi dengan akun Hugging Face.
2. save_pretrained: Menyimpan model dan tokenizer ke folder lokal.
3. upload_folder: Mengunggah model ke Hugging Face Hub.

2. Using Pretrained Models

Menggunakan model pra-latih dari Hugging Face Hub memungkinkan Anda untuk memanfaatkan kekuatan model yang telah dilatih tanpa perlu mengunduh atau menyimpan model secara lokal. Fitur ini sangat berguna untuk eksperimen cepat atau saat bekerja dengan model besar yang membutuhkan banyak ruang penyimpanan. Berikut adalah langkah-langkah utama:

1. **Mengakses Model dari Hub**:
    - Anda dapat memuat model langsung menggunakan nama repositori di Hugging Face Hub.
    - Model dapat berupa model standar (pra-latih) atau model yang telah di-fine-tune.
2. **Inisialisasi Pipeline**:
    - Pipeline adalah abstraksi tingkat tinggi untuk tugas-tugas NLP seperti klasifikasi teks, analisis sentimen, atau penerjemahan bahasa.
    - Pipeline secara otomatis memuat model, tokenizer, dan konfigurasi yang diperlukan.
3. **Penggunaan Model dalam Aplikasi**:
    - Setelah model dimuat, Anda dapat langsung menggunakannya untuk prediksi pada data baru.
    - Tidak diperlukan pengaturan tambahan untuk tokenisasi atau preprocessing.

```
# Contoh Kode: Menggunakan Model Pra-Latih dari Hub
from transformers import pipeline

# Inisialisasi pipeline
classifier = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

# Prediksi dengan model pra-latih
result = classifier("I love Hugging Face!")
print(result)
```

1. **Pipeline Initialization**:
    - pipeline("sentiment-analysis") secara otomatis memuat model yang sesuai untuk analisis sentimen.
    - Model yang digunakan adalah "distilbert-base-uncased-finetuned-sst-2-english," yang telah dilatih untuk tugas klasifikasi sentimen.
2. **Input dan Prediksi**:
    - Input berupa teks mentah seperti "I love Hugging Face!".

- Model memproses teks melalui tokenisasi, inferensi, dan menghasilkan prediksi dalam format yang mudah dibaca.

3. **Output**:
    - Output adalah label prediksi (misalnya, "POSITIVE" atau "NEGATIVE") beserta skor kepercayaan (confidence score) untuk setiap label.

Kelebihan dari pendekatan ini adalah kemudahannya dalam penggunaan, karena semua proses internal, seperti tokenisasi dan decoding, diatur secara otomatis oleh pipeline.

Hugging Face Hub memungkinkan Anda untuk menggunakan model pra-latih tanpa harus menyimpan model secara lokal. Model dapat diakses langsung melalui repositori Hub.

```
# Contoh Kode: Menggunakan Model Pra-Latih dari Hub
from transformers import pipeline

# Inisialisasi pipeline
classifier = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

# Prediksi dengan model pra-latih
result = classifier("I love Hugging Face!")
print(result)
```

1. pipeline: Memuat model pra-latih langsung dari Hugging Face Hub.
2. Model digunakan untuk tugas spesifik tanpa perlu pelatihan tambahan.

3. Sharing Pretrained Models

Setelah melatih atau menyesuaikan model, Anda dapat membagikannya melalui Hugging Face Hub. Berbagi model memungkinkan pengembang lain untuk mengakses dan menggunakan model Anda tanpa harus melatihnya ulang. Model yang dibagikan dapat mencakup:

1. **Parameter Model**:
    - File yang menyimpan bobot dan bias yang telah dilatih.
    - Dapat dimuat ulang dengan cepat untuk inferensi atau pelatihan lanjutan.
2. **Tokenizer**:

- Tokenizer yang digunakan untuk memproses data input sehingga konsisten dengan model.

3. **Dokumentasi Tambahan (Model Cards)**:
    - Penjelasan tentang tujuan, dataset, performa, dan batasan model.

Langkah-langkah berbagi model:

1. Login ke Hugging Face Hub menggunakan notebook_login untuk autentikasi.
2. Gunakan metode push_to_hub untuk mengunggah model dan tokenizer.
3. Tambahkan dokumentasi yang relevan di Model Cards untuk membantu pengguna memahami penggunaan model.

```
# Contoh Kode: Mengunggah Model yang Telah Dilatih
from huggingface_hub import notebook_login

# Login ke Hugging Face Hub
notebook_login()

# Mengunggah model yang telah dilatih
model.push_to_hub("fine-tuned-bert")
tokenizer.push_to_hub("fine-tuned-bert")
```

1. **notebook_login**:
    - Fungsi ini membuka autentikasi interaktif untuk masuk ke akun Hugging Face.
2. **push_to_hub**:
    - Metode ini digunakan untuk mengunggah model dan tokenizer ke repositori Hub.
    - Nama repositori dapat disesuaikan (misalnya, "fine-tuned-bert").
3. **Akses Publik atau Privat**:
    - Model yang diunggah dapat diatur sebagai publik (terlihat oleh semua pengguna) atau privat (hanya Anda yang dapat mengakses).
4. **Manfaat Berbagi Model**:
    - Menghemat waktu bagi pengguna lain dengan menyediakan model yang siap digunakan.

- Meningkatkan kolaborasi dalam komunitas NLP dengan berbagi hasil pelatihan.

Setelah melatih atau menyesuaikan model, Anda dapat membagikannya melalui Hub. Model yang dibagikan dapat mencakup:

- Parameter model.
- Tokenizer.
- Dokumentasi tambahan dalam bentuk Model Cards.

```
# Contoh Kode: Mengunggah Model yang Telah Dilatih
# Login ke Hugging Face Hub
notebook_login()

# Mengunggah model yang telah dilatih
model.push_to_hub("fine-tuned-bert")
tokenizer.push_to_hub("fine-tuned-bert")
```

1. push_to_hub: Mengunggah model dan tokenizer langsung ke Hub dengan nama repositori tertentu.
2. Model ini dapat diakses publik atau privat berdasarkan pengaturan repositori.

4. Building a Model Card

Model Cards adalah dokumentasi yang bertujuan untuk memberikan informasi detail tentang model yang dibagikan di Hugging Face Hub. File ini memberikan transparansi dan membantu pengguna memahami model secara mendalam, termasuk performa, batasan, dan cara penggunaannya. Model Cards biasanya dibuat dalam format Markdown (README.md) dan disertakan di repositori model.

Elemen Utama Model Cards:

1. **Deskripsi Model**:
    - Informasi tentang arsitektur model (misalnya, BERT, GPT-2).
    - Tugas yang didukung oleh model, seperti klasifikasi teks atau analisis sentimen.
    - Dataset yang digunakan untuk pelatihan dan fine-tuning.

2. **Metrik Performa**:
    - Hasil evaluasi model pada dataset validasi atau test (misalnya, akurasi, F1-score).
    - Informasi tentang loss selama pelatihan.
3. **Batasan dan Bias**:
    - Keterbatasan model, seperti hanya mendukung teks dalam Bahasa Inggris.
    - Bias yang mungkin muncul karena dataset pelatihan.
4. **Instruksi Penggunaan**:
    - Panduan tentang bagaimana model dapat digunakan melalui API atau pipeline Hugging Face.

Model Cards sangat penting untuk memastikan bahwa model digunakan secara etis dan sesuai dengan batasannya.

```
# Contoh Kode: Membuat dan Mengunggah Model Card
from huggingface_hub import Repository

# Konten Model Card
model_card_content = """
# Model Name

## Model Description
- **Architecture**: BERT-base
- **Dataset**: IMDb Sentiment Analysis
- **Fine-tuned for**: Sentiment Classification

## Metrics
- **Accuracy**: 95%
- **Loss**: 0.23

## Limitations
- Hanya cocok untuk teks dalam Bahasa Inggris.
- Mungkin bias terhadap dataset pelatihan.

## How to Use
```
from transformers import pipeline
classifier = pipeline("sentiment-analysis", model="username/fine-tuned-bert")
result = classifier("I love Hugging Face!")
print(result)
```
```

```
```

Acknowledgments
- Terima kasih kepada Hugging Face untuk platform dan model dasar.
"""

# Buat file README.md di folder model
repo = Repository(local_dir="./my_model", clone_from="username/fine-tuned-bert")
with open("./my_model/README.md", "w") as f:
    f.write(model_card_content)

# Commit dan push perubahan ke repositori Hugging Face
repo.git_add()
repo.git_commit("Add Model Card")
repo.git_push()
```

Penjelasan:

1. **Konten Model Card**: Menyediakan detail tentang model, metrik, batasan, dan cara penggunaannya.
2. **Repository API**: Mengelola repositori model lokal dan sinkronisasinya ke Hugging Face Hub.
3. **Pengunggahan**: Mengunggah Model Card sebagai file README.md ke repositori Hub.

Dengan kode ini, Model Card Anda akan tersedia di repositori Hugging Face bersama dengan model.

Model Cards adalah dokumentasi yang bertujuan untuk memberikan informasi detail tentang model yang dibagikan di Hugging Face Hub. File ini memberikan transparansi dan membantu pengguna memahami model secara mendalam, termasuk performa, batasan, dan cara penggunaannya. Model Cards biasanya dibuat dalam format Markdown (README.md) dan disertakan di repositori model.

Elemen Utama Model Cards:

1. **Deskripsi Model**:
    - Informasi tentang arsitektur model (misalnya, BERT, GPT-2).
    - Tugas yang didukung oleh model, seperti klasifikasi teks atau analisis sentimen.
```

- Dataset yang digunakan untuk pelatihan dan fine-tuning.

2. **Metrik Performa**:
   - Hasil evaluasi model pada dataset validasi atau test (misalnya, akurasi, F1-score).
   - Informasi tentang loss selama pelatihan.
3. **Batasan dan Bias**:
   - Keterbatasan model, seperti hanya mendukung teks dalam Bahasa Inggris.
   - Bias yang mungkin muncul karena dataset pelatihan.
4. **Instruksi Penggunaan**:
   - Panduan tentang bagaimana model dapat digunakan melalui API atau pipeline Hugging Face.

Model Cards sangat penting untuk memastikan bahwa model digunakan secara etis dan sesuai dengan batasannya.

```
# Contoh: Model Card Template
# Model Name

## Model Description
- **Architecture**: BERT-base
- **Dataset**: IMDb Sentiment Analysis
- **Fine-tuned for**: Sentiment Classification

## Metrics
- **Accuracy**: 95%
- **Loss**: 0.23

## Limitations
- Hanya cocok untuk teks dalam Bahasa Inggris.
- Mungkin bias terhadap dataset pelatihan.

## How to Use
```
from transformers import pipeline
classifier = pipeline("sentiment-analysis", model="username/fine-tuned-bert")
result = classifier("I love Hugging Face!")
print(result)
```
```

```
## Acknowledgments
- Terima kasih kepada Hugging Face untuk platform dan model dasar.
```

Penjelasan:

1. **Deskripsi Model**: Memberikan detail tentang arsitektur, dataset, dan tujuan model.
2. **Metrik**: Menunjukkan performa model dengan angka-angka evaluasi yang relevan.
3. **Batasan**: Membantu pengguna memahami di mana model mungkin tidak bekerja dengan baik.
4. **Cara Penggunaan**: Menyediakan contoh kode langsung untuk mempermudah pengguna dalam menggunakan model.

Model Cards tidak hanya membantu pengguna memahami model, tetapi juga meningkatkan transparansi dan kolaborasi di komunitas NLP.

```
# Contoh: Model Card Template
"""
# Model Name

## Model Description
- **Architecture**: BERT-base
- **Dataset**: IMDb Sentiment Analysis
- **Fine-tuned for**: Sentiment Classification

## Metrics
- **Accuracy**: 95%
- **Loss**: 0.23

## Limitations
- Hanya cocok untuk teks dalam Bahasa Inggris.
- Mungkin bias terhadap dataset pelatihan.

## How to Use
```
from transformers import pipeline
classifier = pipeline("sentiment-analysis", model="username/fine-tuned-bert")
result = classifier("I love Hugging Face!")
print(result)
```
```

Template ini adalah contoh **Model Card**, yaitu dokumen standar untuk mendeskripsikan model yang dibuat atau di-finetune menggunakan pustaka **Hugging Face Transformers**. Model yang dimaksud adalah model BERT yang di-finetune untuk klasifikasi sentimen pada dataset IMDb.

Model Card mencakup informasi penting, seperti arsitektur model (BERT-base), dataset yang digunakan (IMDb Sentiment Analysis), dan metrik evaluasi, termasuk akurasi (95%) dan loss (0.23). Model ini hanya mendukung teks dalam Bahasa Inggris dan mungkin memiliki bias berdasarkan dataset pelatihan. Bagian "How to Use" menyediakan contoh kode untuk menggunakan model ini dengan pipeline klasifikasi sentimen, memungkinkan pengguna untuk langsung menganalisis teks menggunakan model. Template ini memberikan dokumentasi ringkas dan jelas bagi pengguna untuk memahami performa, batasan, dan cara penggunaan model.