CSC 4315: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Week 14, Lecture 28

Hugging Face Transformers Library

Lecture Overview

This lecture covers the Hugging Face ecosystem including pipelines, tokenizers, models, and the Trainer API for fine-tuning.

Topics Covered:

- Hugging Face ecosystem
- Pipelines for quick inference
- Tokenizers and models
- AutoModel classes
- Trainer API for fine-tuning

1. Introduction to Hugging Face

1.1 The Hugging Face Ecosystem

```
print("Hugging Face Ecosystem")
print("="*70)

print("Hugging Face: The GitHub of ML")

print("\nKey components:")
print("  1. Transformers library: Pre-trained models")
print("  2. Datasets library: Standard datasets")
print("  3. Tokenizers library: Fast tokenization")
print("  4. Hub: Model and dataset repository")

print("\nWhy Hugging Face?")
print("  - 200,000+ pre-trained models")
print("  - Consistent API across models")
print("  - Easy to use pipelines")
print("  - Active community")

print("\nInstallation:")
print("  pip install transformers")
print("  pip install datasets")

print("\nSupported frameworks:")
print("  - PyTorch")
print("  - TensorFlow")
print("  - JAX/Flax")
```

```
Hugging Face Ecosystem
======================================================================
Hugging Face: The GitHub of ML

Key components:
  1. Transformers library: Pre-trained models
  2. Datasets library: Standard datasets
  3. Tokenizers library: Fast tokenization
  4. Hub: Model and dataset repository

Why Hugging Face?
  - 200,000+ pre-trained models
  - Consistent API across models
  - Easy to use pipelines
  - Active community

Installation:
  pip install transformers
  pip install datasets

Supported frameworks:
  - PyTorch
  - TensorFlow
  - JAX/Flax
```

2. Pipelines

2.1 Using Pipelines

```
print("Hugging Face Pipelines")
print("="*70)

print("Pipelines: Easiest way to use models")
print("""
from transformers import pipeline

# Sentiment analysis
classifier = pipeline("sentiment-analysis")
result = classifier("I love this movie!")
# [{'label': 'POSITIVE', 'score': 0.9998}]

# Text generation
generator = pipeline("text-generation", model="gpt2")
result = generator("Once upon a time", max_length=50)

# Question answering
qa = pipeline("question-answering")
result = qa(question="What is AI?",
            context="AI is artificial intelligence...")

# Named entity recognition
ner = pipeline("ner")
result = ner("Microsoft was founded by Bill Gates")

# Summarization
summarizer = pipeline("summarization")
result = summarizer(long_text, max_length=100)
""")

print("Available pipelines:")
print("  - sentiment-analysis, text-classification")
print("  - text-generation, text2text-generation")
print("  - question-answering, fill-mask")
print("  - ner, translation, summarization")
print("  - zero-shot-classification")
```

```
Hugging Face Pipelines
======================================================================
Pipelines: Easiest way to use models

from transformers import pipeline

# Sentiment analysis
classifier = pipeline("sentiment-analysis")
result = classifier("I love this movie!")
# [{'label': 'POSITIVE', 'score': 0.9998}]

# Text generation
generator = pipeline("text-generation", model="gpt2")
result = generator("Once upon a time", max_length=50)

# Question answering
qa = pipeline("question-answering")
result = qa(question="What is AI?",
            context="AI is artificial intelligence...")

# Named entity recognition
ner = pipeline("ner")
result = ner("Microsoft was founded by Bill Gates")

# Summarization
summarizer = pipeline("summarization")
result = summarizer(long_text, max_length=100)

Available pipelines:
  - sentiment-analysis, text-classification
  - text-generation, text2text-generation
  - question-answering, fill-mask
  - ner, translation, summarization
- zero-shot-classification
```

3. Tokenizers and Models

3.1 Tokenizers

```
print("Tokenizers")
print("="*70)

print("Loading tokenizers:")
print("""
from transformers import AutoTokenizer

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

# Tokenize text
text = "Hello, how are you?"
tokens = tokenizer(text)
print(tokens)
# {'input_ids': [101, 7592, 1010, 2129, 2024, 2017, 1029, 102],
#  'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0],
#  'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1]}

# Decode back
decoded = tokenizer.decode(tokens['input_ids'])
print(decoded)  # '[CLS] hello, how are you? [SEP]'

# Batch tokenization with padding
texts = ["Hello", "How are you doing today?"]
batch = tokenizer(texts, padding=True, return_tensors="pt")
""")

print("\nKey tokenizer methods:")
print("  tokenizer(text) - Tokenize")
print("  tokenizer.decode(ids) - Decode")
print("  tokenizer.encode(text) - Get input_ids only")
print("  tokenizer.vocab_size - Vocabulary size")
```

```
Tokenizers
======================================================================
Loading tokenizers:

from transformers import AutoTokenizer

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

# Tokenize text
text = "Hello, how are you?"
tokens = tokenizer(text)
print(tokens)
# {'input_ids': [101, 7592, 1010, 2129, 2024, 2017, 1029, 102],
#  'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0],
#  'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1]}

# Decode back
decoded = tokenizer.decode(tokens['input_ids'])
print(decoded)  # '[CLS] hello, how are you? [SEP]'

# Batch tokenization with padding
texts = ["Hello", "How are you doing today?"]
batch = tokenizer(texts, padding=True, return_tensors="pt")

Key tokenizer methods:
  tokenizer(text) - Tokenize
  tokenizer.decode(ids) - Decode
  tokenizer.encode(text) - Get input_ids only
  tokenizer.vocab_size - Vocabulary size
```

3.2 Loading Models

```
print("Loading Models")
print("="*70)

print("AutoModel classes:")
print("""
from transformers import (
    AutoModel,
    AutoModelForSequenceClassification,
    AutoModelForQuestionAnswering,
    AutoModelForCausalLM,
)

# Base model (no head)
model = AutoModel.from_pretrained("bert-base-uncased")

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

# Text generation
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Get model outputs
outputs = model(**tokenizer(text, return_tensors="pt"))
# outputs.last_hidden_state, outputs.logits, etc.
""")

print("\nPopular models:")
print("  BERT: bert-base-uncased, bert-large-uncased")
print("  RoBERTa: roberta-base, roberta-large")
print("  DistilBERT: distilbert-base-uncased")
print("  GPT-2: gpt2, gpt2-medium, gpt2-large")
print("  T5: t5-small, t5-base, t5-large")
```

```
Loading Models
======================================================================
AutoModel classes:

from transformers import (
    AutoModel,
    AutoModelForSequenceClassification,
    AutoModelForQuestionAnswering,
    AutoModelForCausalLM,
)

# Base model (no head)
model = AutoModel.from_pretrained("bert-base-uncased")

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

# Text generation
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Get model outputs
outputs = model(**tokenizer(text, return_tensors="pt"))
# outputs.last_hidden_state, outputs.logits, etc.

Popular models:
  BERT: bert-base-uncased, bert-large-uncased
  RoBERTa: roberta-base, roberta-large
  DistilBERT: distilbert-base-uncased
  GPT-2: gpt2, gpt2-medium, gpt2-large
  T5: t5-small, t5-base, t5-large
```

4. Training with Trainer

4.1 Trainer API

```
print("Hugging Face Trainer")
print("="*70)

print("Complete training workflow:")
print("""
from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    Trainer,
    TrainingArguments,
)
from datasets import load_dataset

# Load dataset
dataset = load_dataset("imdb")

# Load model and tokenizer
model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=2
)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Tokenize dataset
def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)

dataset = dataset.map(tokenize, batched=True)

# Training arguments
args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    evaluation_strategy="epoch",
)

# Create trainer
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
)

# Train!
trainer.train()
""")
```

```
Hugging Face Trainer
======================================================================
Complete training workflow:

from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    Trainer,
    TrainingArguments,
)
from datasets import load_dataset

# Load dataset
dataset = load_dataset("imdb")

# Load model and tokenizer
model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=2
)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

```
# Tokenize dataset
def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)

dataset = dataset.map(tokenize, batched=True)

# Training arguments
args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    evaluation_strategy="epoch",
)

# Create trainer
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
)

# Train!
trainer.train()
```

Summary

Key Takeaways:

- Hugging Face provides unified API for transformers
- Pipelines are easiest way to use models
- AutoTokenizer/AutoModel load correct classes
- Always use padding and truncation
- Trainer simplifies training workflow
- Hub has 200,000+ pre-trained models

Practice Exercises:

1. Use pipeline for sentiment analysis
2. Tokenize text with BERT tokenizer
3. Fine-tune BERT on custom dataset
4. Compare different pre-trained models
5. Push trained model to Hub