Hugging Face Transformers Tutorial

CS224N: Natural Language Processing with Deep Learning

Winter 2026

Slides By: Minsik Oh

Session Agenda

Interactive hands-on tutorial

Setup & Introduction

Install packages, overview of HF ecosystem

Tokenizers Deep Dive

Tokenization pipeline, special tokens, padding, batch encoding

Models & Inference

Loading models, AutoModel classes, forward pass, attention visualization

Fine-tuning on IMDB

Dataset loading, training loop, HF Trainer, callbacks, evaluation

Generation & Wrap-up

Text generation with GPT-2, custom datasets, pipelines, Q&A

What is Hugging Face Transformers?

Models

Pretrained weights + code (Llama 3, DBRX, BERT, GPT-2, etc.)

Tokenizers

Model-specific text preprocessing (Python & Rust)

Pipelines

One-line inference for common NLP tasks

Datasets

Standard dataset loading & preprocessing

Trainer

Training loop abstraction with logging & checkpointing

Supported Frameworks

- PyTorch (used today)
- TensorFlow
- Flax / JAX

Project Types this helps with:

1. Applying pretrained models to new tasks
3. Analyzing model behavior & representations

The Common Pattern

HF workflow follows this 3-step pattern:

1

Find a Model

Browse the Hugging Face Hub
Anyone can upload models!

2

Initialize

Load the Tokenizer
Load the Model

3

Predict

Tokenize input → Forward pass
→ Get prediction

```
tokenizer = AutoTokenizer.from_pretrained("siebert/sentiment-roberta-large-english")
model     = AutoModelForSequenceClassification.from_pretrained(...)
outputs   = model(**tokenizer(text, return_tensors="pt"))
```

Hands-On: Setup & First Prediction

1. Open the Colab notebook linked on the course page
2. Run the install cells: pip install transformers datasets accelerate
3. Run Part 0 — load the sentiment model and tokenizer
4. Try changing the input string and re-running the prediction

💡 Expected output: "The prediction is POSITIVE" for the default input

Tokenizers: From Text to Numbers

Three Ways to Load a Tokenizer

`DistilBertTokenizer`	Model-specific, Python	
`DistilBertTokenizerFast`	Model-specific, Rust (faster)	
`AutoTokenizer`	Auto-detects model type	✦ Recommended

Tokenizer: Step by Step

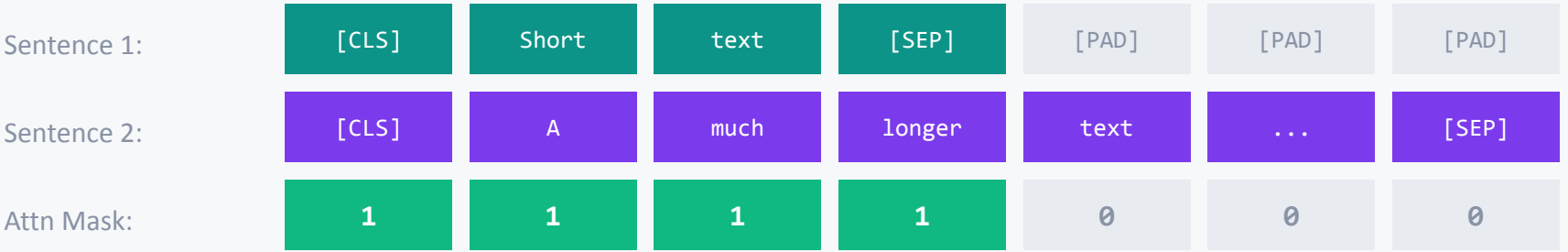
```
input_tokens = tokenizer.tokenize(input_str)
# → ["Hu", "##gging", "Face", "Transformers", "is", "great", "!"]

input_ids = tokenizer.convert_tokens_to_ids(input_tokens)
# → [20164, 10932, 10289, 25267, 1110, 1632, 106]

input_ids_special = cls + input_ids + sep
# → [101, 20164, ..., 106, 102]   ([CLS]...[SEP])

decoded = tokenizer.decode(input_ids_special)
# → "[CLS] Hugging Face Transformers is great! [SEP]"
```

- **Special Tokens** — [CLS] at start, [SEP] at end — model-dependent
- **Subword Tokenization** — "Hugging" → "Hu" + "##gging" — handles unknown words
- **Attention Mask** — 1 for real tokens, 0 for padding — tells model what to attend to

Padding, Truncation & Batching

```
model_inputs = tokenizer(
    ["Short text", "A much longer text..."],
    return_tensors="pt", padding=True, truncation=True
)
```

What happens:



← *Sentence 1's attention mask (padding = 0)*

Hands-On: Explore Tokenizers

1. Run Section 0.1 — compare DistilBertTokenizer vs Fast vs Auto
2. Try tokenizing your own sentences, decode them back
3. Experiment with padding=True on batches of different lengths
4. Use char_to_token() to find which wordpiece a character belongs to
5. Try batch_decode with and without skip_special_tokens=True

💡 Tip: Use return_tensors="pt" to get PyTorch tensors directly

Models: Architecture & Heads

Three Types of Transformer Models

Encoders

`BERT, DistilBERT`

Classification, NER, analysis

Decoders

`GPT-2, LLaMA`

Text generation

Enc-Dec

`BART, T5`

Summarization, translation

Task-Specific Model Heads

`*Model`	Base representations
`*ForMaskedLM`	Fill in the blank
`*ForSequenceClassification`	Sentiment, topic classification
`*ForTokenClassification`	NER, POS tagging
`*ForQuestionAnswering`	Extractive QA
`*ForCausalLM`	Text generation

Model Inference: It's Just PyTorch!

```
# Pass inputs via keyword args or dict unpacking
model_inputs = tokenizer(text, return_tensors="pt")
model_outputs = model(**model_inputs)

# Calculate loss normally
loss = F.cross_entropy(outputs.logits, labels)
loss.backward()  # Standard PyTorch!
```

Standard nn.Module	Use any PyTorch optimizer, scheduler, or loss function
Built-in Loss	Pass 'labels' kwarg and the model computes loss automatically
Attention & Hidden States	Set output_attentions=True and output_hidden_states=True

Hands-On: Models & Inference

1. Run Section 0.2 — load DistilBERT for classification
2. Compare base model vs classification model outputs
3. Try passing 'labels' to the model and inspect the loss
4. Run the attention visualization code — examine the heatmaps
5. Which attention heads attend to [SEP]? Which attend locally?

💡 See "What does BERT look at?" (Clark et al., 2019) for attention analysis

Fine-tuning: Loading Datasets

IMDB Dataset

25,000 train / 25,000 test reviews
Binary sentiment: POSITIVE / NEGATIVE
We use 128 train + 32 val (for speed)
Truncated to 50 tokens each

```
from datasets import load_dataset

ds = load_dataset("stanfordnlp/imdb")

# Tokenize the dataset
tokenized = ds.map(
  lambda ex: tokenizer(
    ex['text'], padding=True,
    truncation=True),
  batched=True)
```

Dataset Preprocessing Pipeline

1. Tokenize

Apply tokenizer to text with padding & truncation

2. Remove text

remove_columns(["text"]) — model only needs IDs

3. Rename label

rename_column("label", "labels") — HF convention

4. Set format

set_format("torch") — convert to PyTorch tensors

Fine-tuning: The Training Loop

Option A: PyTorch Training Loop

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

for epoch in range(num_epochs):
  model.train()
  for batch in train_dataloader:
    output = model(**batch)
    output.loss.backward()
    optimizer.step()
    optimizer.zero_grad()

  model.eval()  # validation
  for batch in eval_dataloader:
    with torch.no_grad():
      output = model(**batch)
```

Option B: HF Trainer (recommended)

```
args = TrainingArguments(
  output_dir="checkpoints",
  per_device_train_batch_size=16,
  num_train_epochs=2,
  eval_strategy="epoch",
  learning_rate=2e-5)

trainer = Trainer(
  model=model,
  args=args,
  train_dataset=train_ds,
  eval_dataset=val_ds,
  compute_metrics=fn)

trainer.train()
```

HF Trainer: Callbacks & Evaluation

Callbacks

`EarlyStoppingCallback`
Stop training when validation loss plateaus

`LoggingCallback (custom)`
Log metrics to JSONL file on each logging step

Custom callbacks can hook into:
`on_log, on_epoch_end, on_evaluate, on_save, ...`

Evaluation

`trainer.evaluate()`
Returns evaluation metrics

`trainer.predict(test_ds)`
Returns predictions + metrics

compute_metrics function:
Custom metric calculation at eval time (accuracy, F1, etc.)

Recommended Starting Hyperparameters

Epochs	Learning Rate	Batch Size	Weight Decay	Warmup Steps
2–4	**2e-5 or 5e-5**	**As large as fits GPU**	**0, 0.01, or 0.1**	**0, 100, or 500**

Hands-On: Fine-tune on IMDB

1. Run Section 2.1 — load & preprocess the IMDB dataset
2. Run the PyTorch training loop (Section 2.2) — watch the loss decrease
3. Run the HF Trainer version — compare the two approaches
4. Add EarlyStoppingCallback and LoggingCallback
5. Use trainer.predict() to evaluate — check accuracy
6. Load the saved checkpoint and test on your own review!

Text Generation with GPT-2

```
from transformers import AutoModelForCausalLM, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('gpt2')
model = AutoModelForCausalLM.from_pretrained('distilgpt2')

prompt = "Once upon a time"
tokenized = tokenizer(prompt, return_tensors="pt")

output = model.generate(**tokenized,
    max_length=50, do_sample=True, top_p=0.9)
```

Key Generation Parameters

`max_length`	Maximum number of tokens to generate
`do_sample`	Enable sampling (vs greedy decoding)
`top_p`	Nucleus sampling — only consider top-p probability mass
`temperature`	Controls randomness: lower = more deterministic
`top_k`	Only sample from top-k most likely tokens

Pipelines & Masked Language Modeling

Pipelines: One-Line Inference

```
sa = pipeline("sentiment-analysis",
  model="siebert/...")
sa("This movie is great!")
```

Available pipelines: sentiment-analysis, fill-mask, text-generation, question-answering, summarization, translation, NER, ...

Masked Language Modeling

```
mlm =
pipeline("fill-mask","bert-base-cased")
mlm("I am [MASK] to learn!")
```

Top predictions:
→ excited (35.5%)
→ going (15.6%)
→ eager (7.9%)
→ here (3.6%)

Custom Datasets (Appendix 1 in notebook)

You can create custom PyTorch Datasets for HF models — just return tokenized dicts from __getitem__. See the E2E Dataset example in the notebook for encoder-decoder tasks.

Hands-On: Generation, Pipelines & Wrap-up

1. Run Appendix 0 — generate text with DistilGPT-2
2. Try different prompts and play with top_p / temperature
3. Run Appendix 2 — use the sentiment pipeline on your text
4. Run Appendix 4 — try fill-mask with BERT, craft your own [MASK] prompts

🎯 Challenge: Can you get GPT-2 to generate a coherent story? What temperature works best?

Resources & Next Steps

Hugging Face Docs
API reference, tutorials, model list

huggingface.co/docs

HF Course
Free NLP course with interactive exercises

huggingface.co/course

HF Examples
Task-specific code templates

github.com/huggingface/transformers/examples

HF O'Reilly Book
In-depth guide with code

Natural Language Processing with Transformers

For your CS224N projects: start with a pretrained model + HF Trainer, iterate on hyperparameters, evaluate different prompts, and monitor validation loss. Good luck!