HuggingFace Genomics Hands-On Exercise

DNA Sequence Analysis with Foundation Models

Duration and Level

Duration: 2–2.5 hours

Level: Intermediate

Format: Jupyter Notebook / Google Colab

Learning Objectives

1. Master PyTorch tensors for genomic data
2. Navigate the Hugging Face ecosystem
3. Load pre-trained genomic models
4. Generate DNA embeddings
5. Visualize embeddings with PCA / UMAP
6. Use Hugging Face Inference API

Setup (Slide 1/2)

```python
# Install packages
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
!pip install matplotlib scikit-learn ipython
!pip install transformers umap-learn

# Import libraries
import torch
import numpy as np
import matplotlib.pyplot as plt
from transformers import AutoTokenizer, AutoModel
from sklearn.decomposition import PCA
import umap

print(f"PyTorch: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")
```

Setup (Slide 2/2)

Check installation:

✓ PyTorch installed

✓ Transformers loaded

✓ Visualization libraries ready

Device Selection

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using: {device}")
```

💡 *Recommended: Google Colab (GPU runtime)*

Part 1: PyTorch Tensors (1/3)

Exercise 1.1: DNA Sequence Tensors

```
dna = "ATCGATCGATCG"
mapping = {'A': 0, 'T': 1, 'C': 2, 'G': 3}

encoded = [mapping[b] for b in dna]
tensor = torch.tensor(encoded)

print(f"Original: {dna}")
print(f"Tensor: {tensor}")
print(f"Shape: {tensor.shape}")
```

Output

- Tensor shape → `torch.Size([12])`
- Data type → `torch.int64`

Part 1: PyTorch Tensors (2/3)

Exercise 1.2: One-Hot Encoding

```python
def one_hot_encode(seq):
    mapping = {'A': 0, 'T': 1, 'C': 2, 'G': 3}
    one_hot = torch.zeros(len(seq), 4)
    for i, b in enumerate(seq):
        one_hot[i, mapping[b]] = 1
    return one_hot

encoded = one_hot_encode("ATCG")
print(encoded.shape)  # (4, 4)
```

Batch Encoding

```python
seqs = ["ATCG", "GCTA"]
batch = torch.stack([one_hot_encode(s) for s in seqs])
print(batch.shape)  # (2, 4, 4)
```

Part 1: PyTorch Tensors (3/3)

Exercise 1.3: Device Management

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

t = torch.randn(100, 512)
t_gpu = t.to(device)
print("Device:", t_gpu.device)

t_cpu = t_gpu.cpu()
print("Back to:", t_cpu.device)
```

Tips

- Move both model and data to the same device
- `.to(device)` moves tensors
- `.cpu()` moves back

Part 2: Hugging Face Platform (1/2)

Exercise 2.1: Browse the Model Hub

Visit → https://huggingface.co/models

Search examples:

- "DNABERT" → `zhihan1996/DNABERT-2-117M`
- "nucleotide transformer" → InstaDeepAI variants
- "genomic" → multiple models

Explore

- Model ID
- Download count
- Tokenization method

Part 2: Hugging Face Platform (2/2)

Exercise 2.2: Model Card Analysis

Visit → DNABERT-2 Model Card

Find

- Tokenization → BPE
- Max seq length → 512
- Architecture → BERT + ALiBi
- Parameters → 117M

Model Card Sections

1. Overview
2. Model Details

Part 3: Load Models (1/3)

Exercise 3.1: Load DNABERT-2

```python
from transformers import AutoTokenizer, AutoModel, BertConfig
model_name = "zhihan1996/DNABERT-2-117M"
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)

# Explicitly load the config from the repo so we use the correct config class
config = BertConfig.from_pretrained(model_name)
model = AutoModel.from_pretrained(
    model_name,
    trust_remote_code=True,
    config=config
)

params = sum(p.numel() for p in model.parameters())
print(f"Parameters: {params:,}")
```

Part 3: Load Models (2/3)

Exercise 3.2: Load Nucleotide Transformer

```python
nt_name = "InstaDeepAI/nucleotide-transformer-500m-human-ref"
nt_tokenizer = AutoTokenizer.from_pretrained(nt_name, trust_remote_code=True)
nt_model = AutoModel.from_pretrained(nt_name, trust_remote_code=True)
nt_model = nt_model.to(device).eval()

nt_params = sum(p.numel() for p in nt_model.parameters())
print(f"NT Parameters: {nt_params:,}")
```

Part 3: Load Models (3/3)

Exercise 3.3: Tokenize and Embed

```
seqs = ["ATCGATCGATCGATCG", "GCTAGCTAGCTAGCTA"]

tokens = tokenizer(
    seqs, padding=True, truncation=True, max_length=512,
    return_tensors="pt"
)

with torch.no_grad():
    out = model(**{k: v.to(device) for k, v in tokens.items()})
    emb = out.last_hidden_state.mean(dim=1)

print(emb.shape)
```

Part 4: Classification (1/4)

Prepare sequences

```python
promoters = ["TATAAA", "CAAT", "GGGCGG"]
non_prom = ["ACTGACTG", "GTCAGTCA", "TTGGCCAA"]

def extend(seq, size=50):
    return (seq * ((size // len(seq)) + 1))[:size]

prom_ext = [extend(s) for s in promoters]
nprom_ext = [extend(s) for s in non_prom]
```

Part 4: Classification (2/4)

Generate embeddings

```
all_seqs = prom_ext + nprom_ext
labels = [1]*len(prom_ext) + [0]*len(nprom_ext)

tokens = tokenizer(
    all_seqs, padding=True, truncation=True, max_length=512,
    return_tensors="pt"
).to(device)

with torch.no_grad():
    emb = model(**tokens).last_hidden_state.mean(dim=1).cpu().numpy()
```

Part 4: Classification (3/4)

PCA Visualization

```
pca = PCA(n_components=2)
emb_pca = pca.fit_transform(emb)

plt.figure(figsize=(8,6))
colors = ['blue' if l==1 else 'red' for l in labels]
plt.scatter(emb_pca[:,0], emb_pca[:,1], c=colors, s=100, alpha=0.7)
plt.xlabel('PC1'); plt.ylabel('PC2')
plt.title('DNA Embeddings (PCA)')
plt.legend(['Promoter','Non-promoter'])
plt.tight_layout()
plt.show()
```

Part 4: Classification (4/4)

UMAP Visualization

```
reducer = umap.UMAP(n_neighbors=5, min_dist=0.3, metric='cosine')
emb_umap = reducer.fit_transform(emb)

plt.figure(figsize=(8,6))
scatter = plt.scatter(
    emb_umap[:,0], emb_umap[:,1], c=labels,
    cmap='RdBu', s=100, alpha=0.6
)
plt.colorbar(scatter, label='Class')
plt.xlabel('UMAP1'); plt.ylabel('UMAP2')
plt.title('DNA Embeddings (UMAP)')
plt.tight_layout()
plt.show()
```

Part 5: Hugging Face Inference API (1/3)

Setup Authentication

1. Go to → huggingface.co/settings/tokens
2. Create a new token → *Read access*
3. Copy the token

```
HF_TOKEN = "your_token_here"
```

⚠️ *Do not commit tokens to GitHub!*

Part 5: Hugging Face Inference API (2/3)

Call Inference API

```python
import requests

def query_api(seq, model_id, token):
    url = f"https://api-inference.huggingface.co/models/{model_id}"
    headers = {"Authorization": f"Bearer {token}"}
    response = requests.post(url, headers=headers, json={"inputs": seq})
    return response.json()

result = query_api(
    "ATCGATCGATCG",
    "zhihan1996/DNABERT-2-117M",
    HF_TOKEN
)
print(result)
```

Part 5: Hugging Face Inference API (3/3)

Compare Local vs API Speed

```python
import time

# Local
start = time.time()
tokens = tokenizer(["ATCGATCG"], return_tensors="pt").to(device)
with torch.no_grad(): _ = model(**tokens)
print("Local:", time.time()-start)

# API
start = time.time()
_ = query_api("ATCGATCG", "zhihan1996/DNABERT-2-117M", HF_TOKEN)
print("API:", time.time()-start)
```

✅ Local: Faster, flexible

☁️ API: Simpler, scalable

Bonus: scGPT (Optional)

```
!pip install scgpt scanpy
import scgpt
print("scGPT ready!")
```

Resources:

- Quickstart
- GitHub

Reflection Questions (1/2)

1. What advantages do pre-trained models offer?
2. Compare DNABERT-2 vs Nucleotide Transformer.
3. When to use API vs local inference?

Reflection Questions (2/2)

4. Applications of DNA embeddings?
5. Common challenges?
    - GPU memory issues
    - Tensor shapes
    - Model selection
    - Visualization interpretation

Resources

- **Docs:** Transformers
- **DNABERT-2:** GitHub
- **Nucleotide Transformer:** BioRxiv
- **Galaxy Genomic LLM:** Galaxy Training
- **Datasets:** Hugging Face Datasets

Submission Guidelines

Deliverables

1. Completed Jupyter notebook
2. Answered reflection questions
3. At least one visualization (PCA / UMAP)
4. Short 1–2 page report

Grading

- Code correctness: 40%
- Reflection depth: 30%
- Visualization quality: 20%
- Exploration: 10%

Quick Reference

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
tokenizer = AutoTokenizer.from_pretrained(name, trust_remote_code=True)
model = AutoModel.from_pretrained(name, trust_remote_code=True)

tokens = tokenizer(seqs, padding=True, truncation=True, return_tensors="pt")
with torch.no_grad():
    emb = model(**tokens).last_hidden_state.mean(dim=1)
```

Summary

✓ PyTorch tensors for genomic data

✓ Hugging Face platform navigation

✓ Model loading & embedding

✓ Visualization (PCA / UMAP)

✓ API usage

Next Steps: Apply to your research 🚀

End

Thank you!

Questions? Check troubleshooting or docs.