Introduction to Hugging Face

Ikram Ullah, Staff Scientist

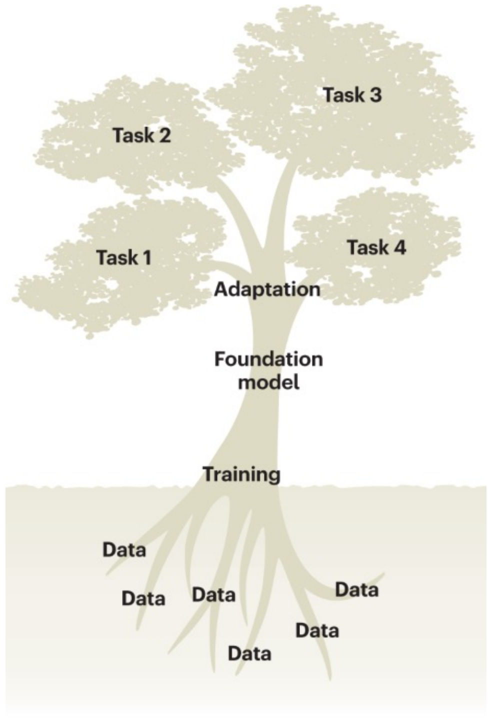


Image taken from – Tang, Lin. "Large models for genomics." *Nature Methods* 20.12 (2023): 1868-1868.

Agenda

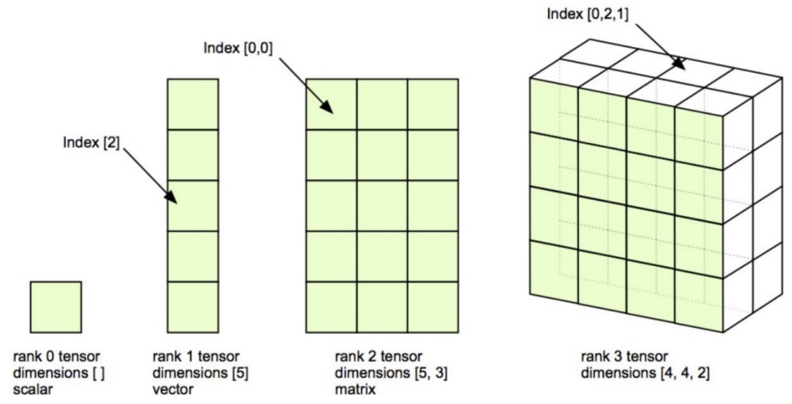
- Tensors in PyTorch
- Navigating Hugging Face Platform
- Core components of Hugging Face Platform
- Hugging Face API examples
- DNA Sequence Analysis: Minimalistic code walkthrough

UMAP visualization of the pretrained scGPT cell embeddings (emb; a random 10% subset), colored by major cell types

Cui, Haotian, et al. "scGPT: toward building a foundation model for single-cell multi-omics using generative AI." *Nature Methods* 21.8 (2024): 1470-1480.

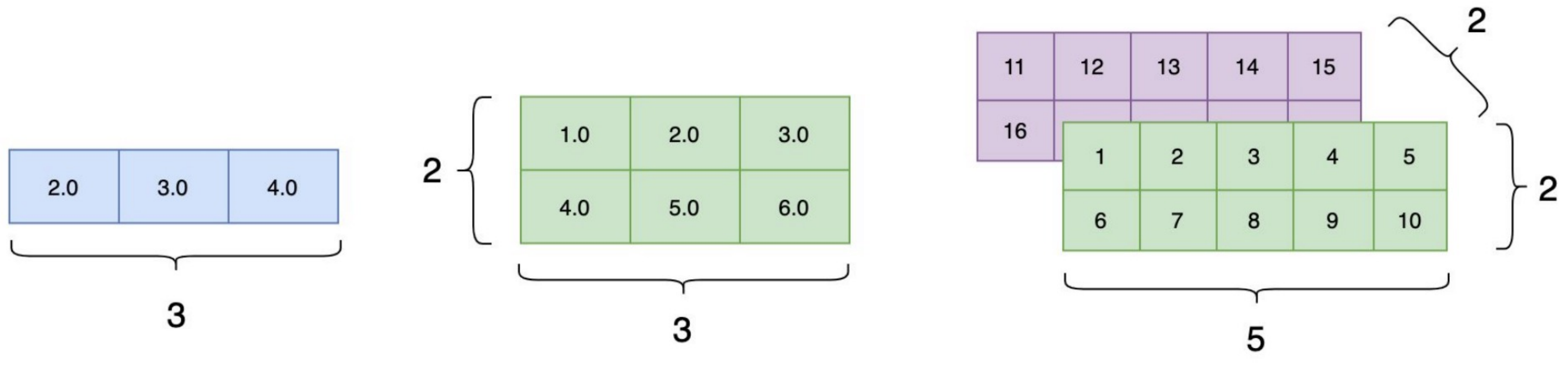
Tensors

- A tensor is the fundamental data structure in PyTorch
  - Multi-dimensional array, similar to NumPy's ndarray but with **GPU acceleration**
- Used to represent
  - DNA sequences (one-hot encoded)
  - Expression profiles
  - Model weights and activations



Dim	**Type**	**Example**
0D	Scalar	torch.tensor(3.14)
1D	Vector	DNA sequence → [1,4,3,3,1,5,4]
2D	Matrix	Batch of sequences
3D+	High-rank Tensors	batch_size × vocabulary × seq length

Tensor shapes



1-D Tensor, shape[3] 2-D Tensor, shape[2, 3] 3-D Tensor, shape[2, 2, 5]

Tensor multiplication: Quick recap

$$\mathbf{A[2,3]} \qquad \mathbf{B[3,4]} \qquad \mathbf{C[2,4]}$$

$$\begin{array}{c|ccc} & \mathbf{3} & & \\ \hline \mathbf{2} & 1 & 2 & 3 \\ & 4 & 5 & 6 \end{array} \times \begin{array}{c|cccc} & \mathbf{4} & & & \\ \hline \mathbf{3} & 7 & 8 & 9 & 10 \\ & 11 & 12 & 13 & 14 \\ & 15 & 16 & 17 & 18 \end{array} = \begin{array}{c|cccc} & \mathbf{4} & & & \\ \hline \mathbf{2} & 74 & 80 & 86 & 92 \\ & 173 & 188 & 203 & 218 \end{array}$$

Calculation Example: C[0,0] = 74

C[0,0] = A[0,0]×B[0,0] + A[0,1]×B[1,0] + A[0,2]×B[2,0]

C[0,0] = 1×7 + 2×11 + 3×15 = 7 + 22 + 45 = 74

Each element in C is the dot product of a row from A and a column from B

Inner dimensions must match and will be contracted

Outer dimensions remain

Data Types

Using different data types for model and data will cause errors

Data Type	dtype	
16-bit floating point	torch.float16	1 sign + 5 exponent + **10** mantissa
16-bit brain floating point	torch.bfloat16	1 sign + 8 exponent + **7** mantissa
32-bit floating point	torch.float32	1 sign + 8 exponent + 23 mantissa
8-bit signed integer	torch.int8	8 bit number

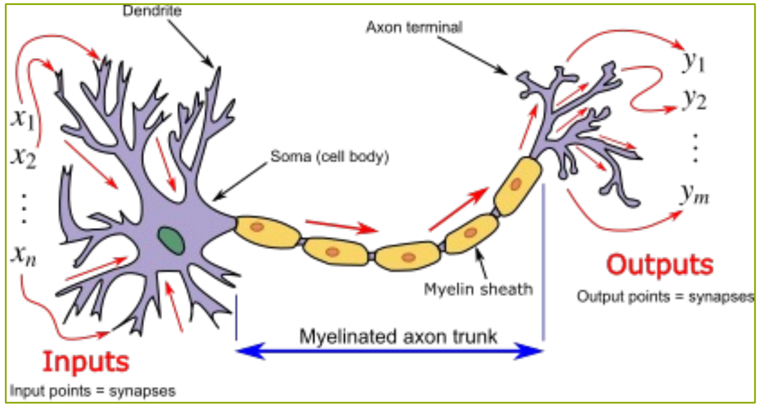
PyTorch Device Management

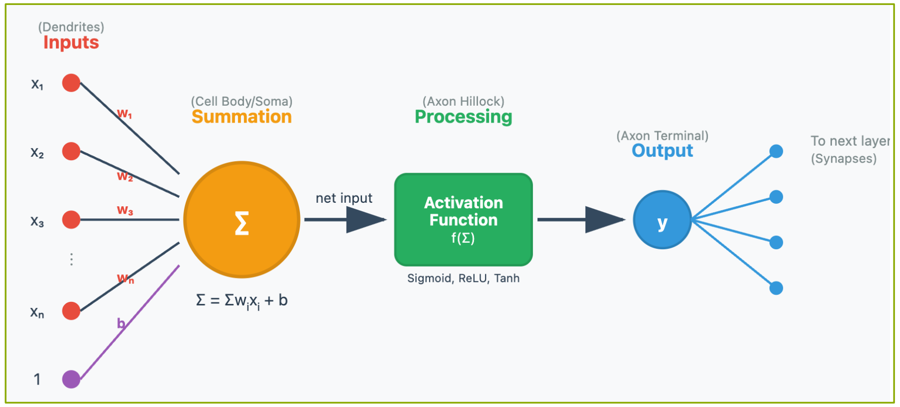
- By default, tensors are on the CPU
- However, you can change this by using the `.to()` operation
  - `x = x.to(‘cpu’)`
  - `x = x.to(‘cuda’)`
- Training/inference can be slow or infeasible on CPU alone
- No need to rewrite model logic, just move the data and model

```
import torch

# Move tensor to GPU
tensor = tensor.to("cuda")

# Move model to same device
model = model.to(tensor.device)

# Back to CPU
tensor = tensor.to("cpu")
```

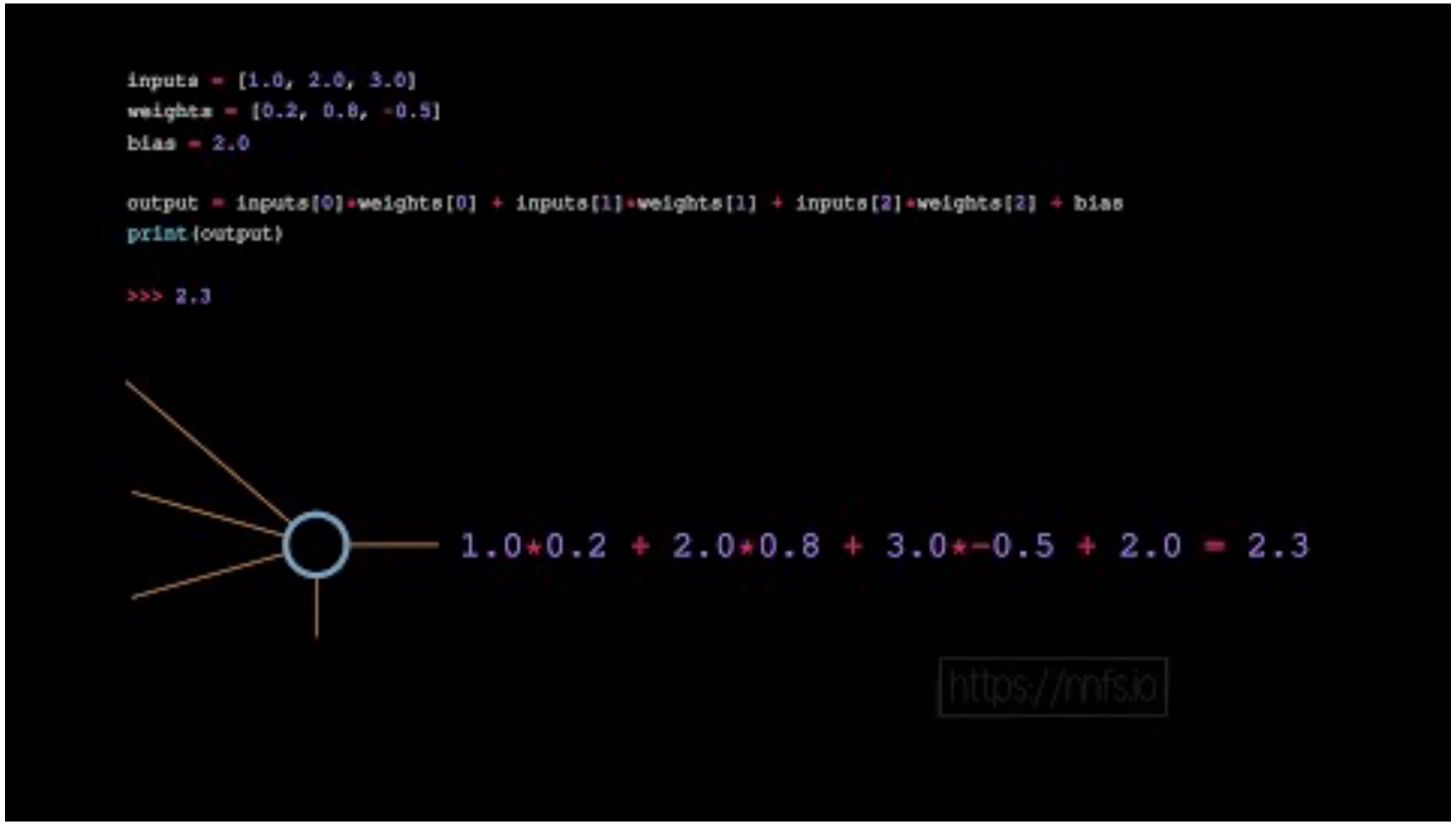
Neural Networks: A Quick recap

Biological vs Artificial Neuron



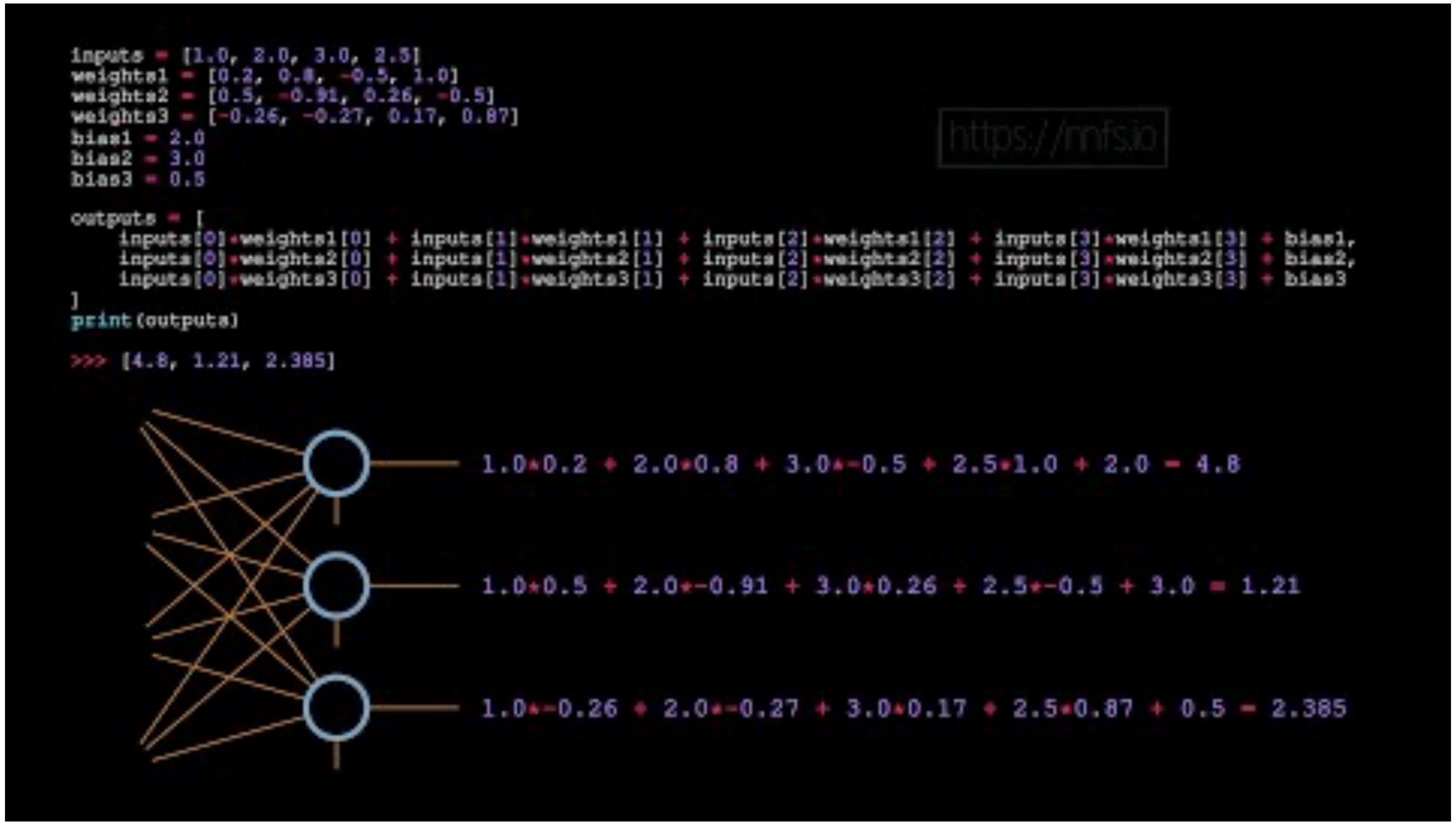


Computation on a single neuron



```
inputs = [1.0, 2.0, 3.0]
weights = [0.2, 0.8, -0.5]
bias = 2.0

output = inputs[0]*weights[0] + inputs[1]*weights[1] + inputs[2]*weights[2] + bias
print(output)

>>> 2.3
```

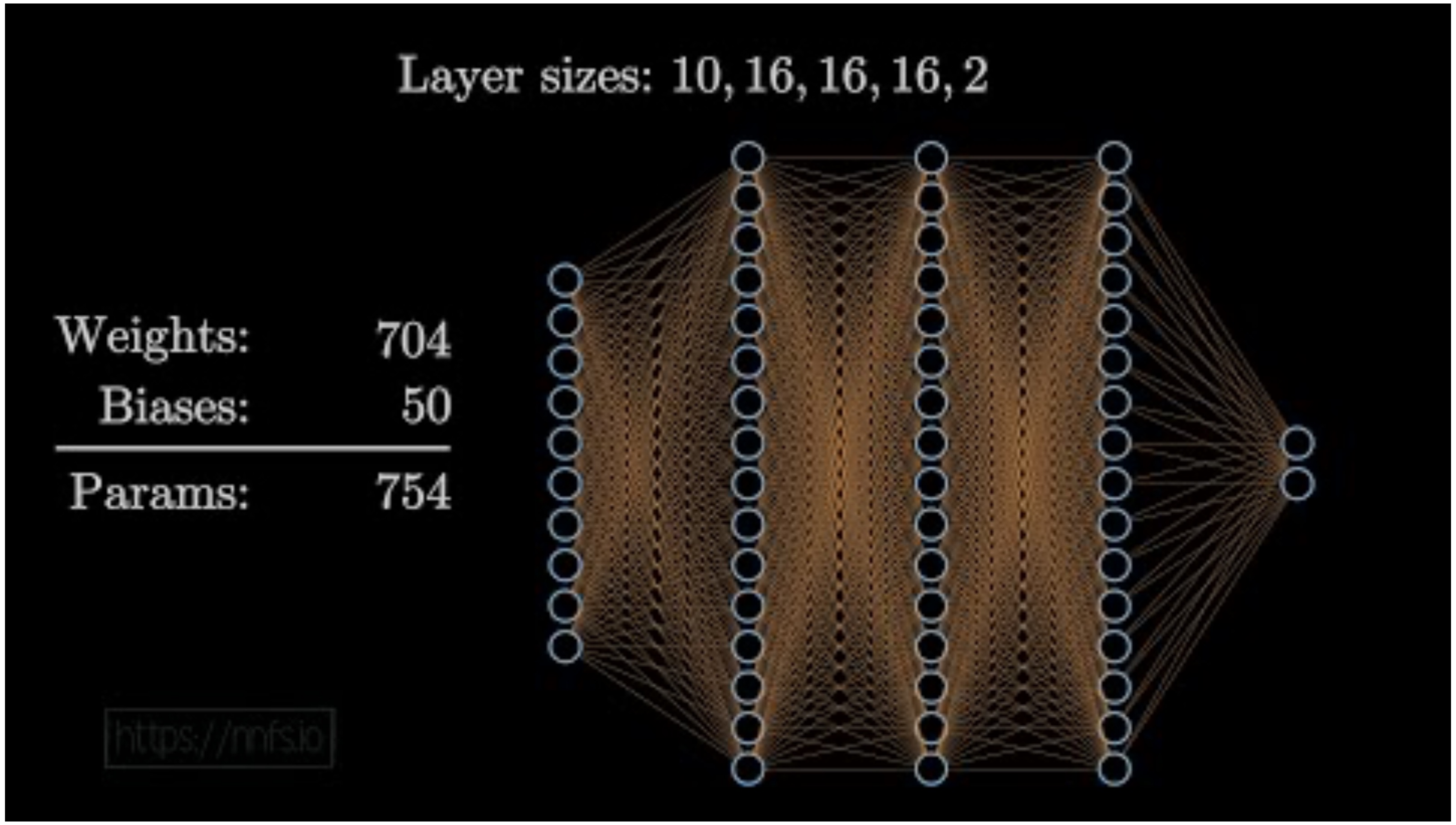
Computation on 3-neuron layer with 4 inputs



```
inputs = [1.0, 2.0, 3.0, 2.5]
weights1 = [0.2, 0.8, -0.5, 1.0]
weights2 = [0.5, -0.91, 0.26, -0.5]
weights3 = [-0.26, -0.27, 0.17, 0.87]
bias1 = 2.0
bias2 = 3.0
bias3 = 0.5

outputs = [
    inputs[0]*weights1[0] + inputs[1]*weights1[1] + inputs[2]*weights1[2] + inputs[3]*weights1[3] + bias1,
    inputs[0]*weights2[0] + inputs[1]*weights2[1] + inputs[2]*weights2[2] + inputs[3]*weights2[3] + bias2,
    inputs[0]*weights3[0] + inputs[1]*weights3[1] + inputs[2]*weights3[2] + inputs[3]*weights3[3] + bias3
]
print(outputs)

>>> [4.8, 1.21, 2.385]
```

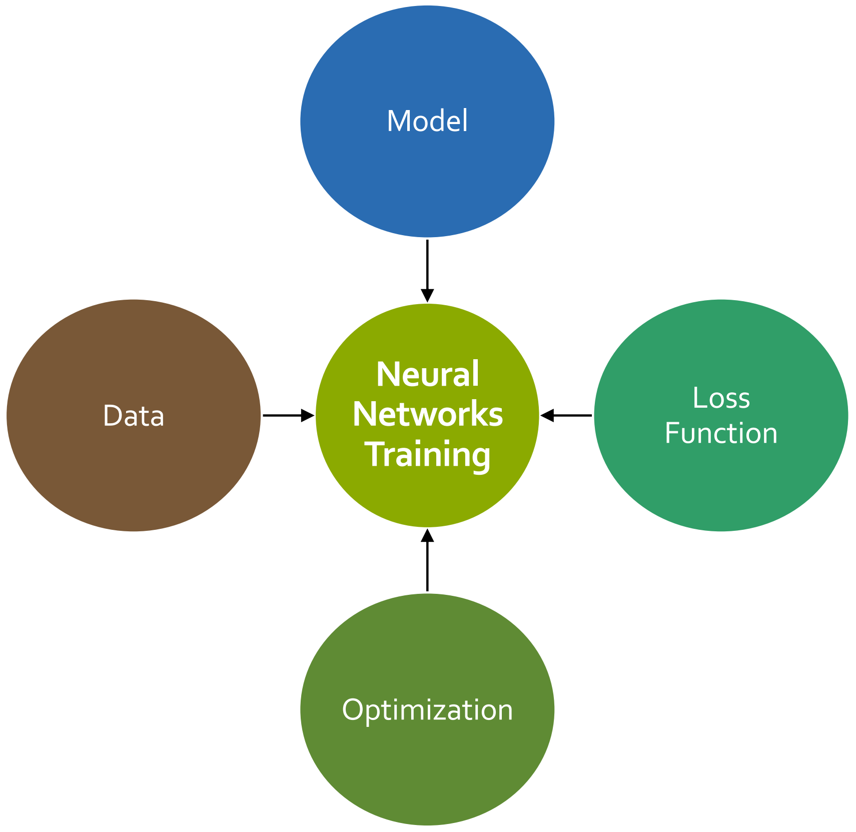
```
1.0*0.2 + 2.0*0.8 + 3.0*-0.5 + 2.5*1.0 + 2.0 = 4.8

1.0*0.5 + 2.0*-0.91 + 3.0*0.26 + 2.5*-0.5 + 3.0 = 1.21

1.0*-0.26 + 2.0*-0.27 + 3.0*0.17 + 2.5*0.87 + 0.5 = 2.385
```

Visualizing Neural Network Sizes



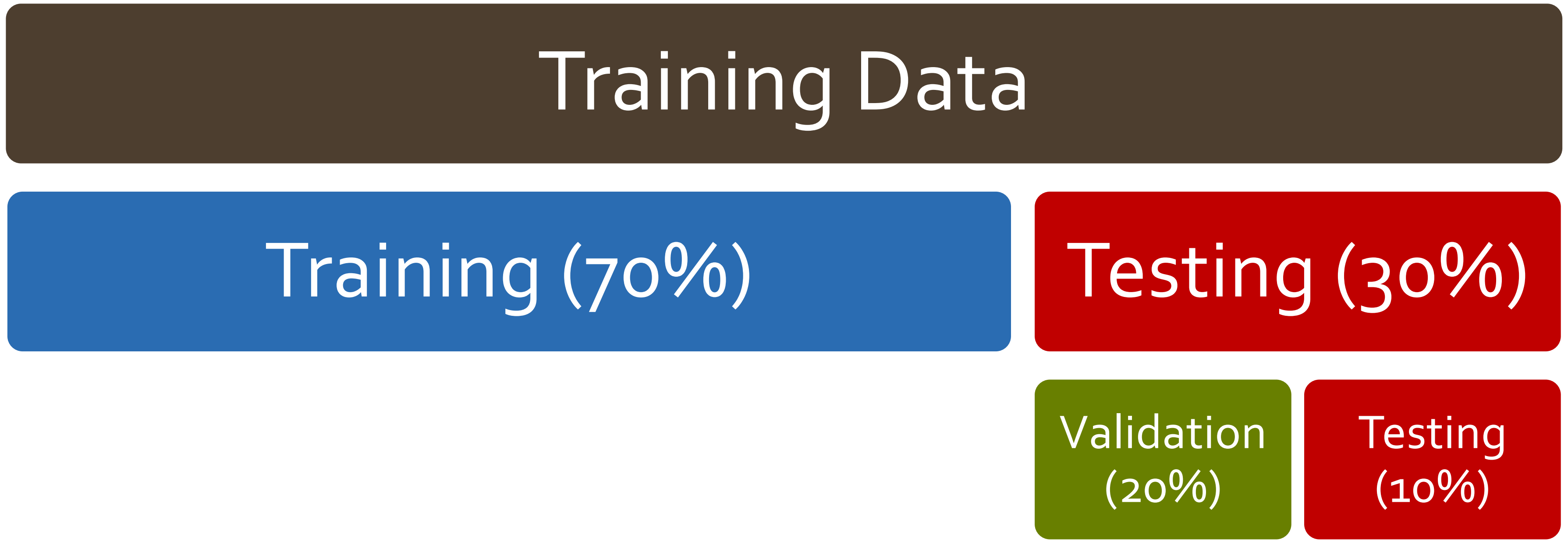
Neural Network Components

Model	• Defining the Neural Network
Training Data	• Data with Labels
Loss Function	• Definition of loss
Optimization	• Optimum path to minimize the loss



Data split

The ratio can be use-case specific but most of the time it is a 7:2:1 split



Hugging Face

What is HuggingFace

- Hugging Face is an open-source platform for building and sharing AI models including genomic foundation and fine-tuned models
  - Foundation model: zhihan1996/DNABERT-2-117M
  - Fine-tuned version: zhangtaolab/dnabert2-promoter
- It provides **open-source tools** and a **central hub** for pre-trained models, supporting key applications such as sequence classification, variant effect prediction, and regulatory region modeling.
- Hugging Face's Transformers library has been extended to support genomic models like DNABERT and Nucleotide Transformer, with adaptations for domain-specific tokenization (e.g., k-mers) and DNA-based datasets

Platform Components

The Hub

- Central repository for models, datasets, and applications

Libraries

- Transformers, Datasets, Tokenizer, Accelerate etc

Inference API

- Serverless access to models

Spaces

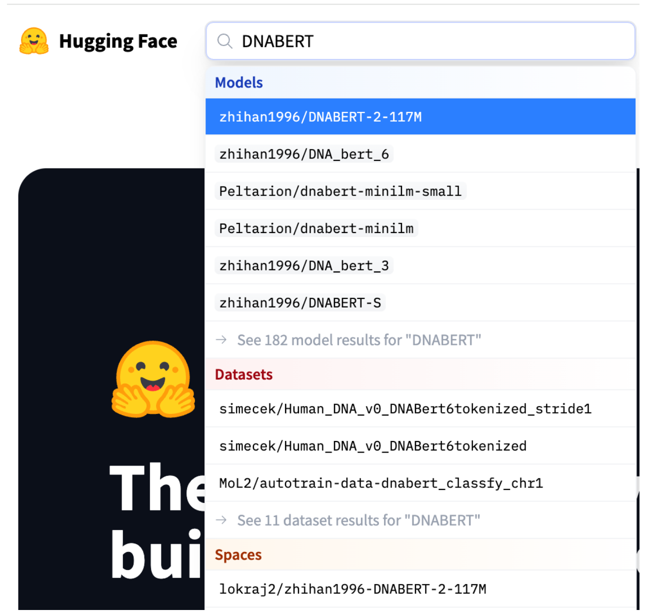
- Platform to build and host interactive demos

Community Features

- Discussions, collaboration tools

Navigating the Hugging Face Ecosystem

- Model Hub
  - Search for genomic models using tags like "genomics," "biology," or specific model names (e.g., "DNABERT"). Explore model cards for details on training data, intended use, and evaluation metrics.
- Datasets
  - Discover genomic datasets for pre-training or fine-tuning.
- Spaces
  - Find and interact with demos of genomic models.
- Documentation
  - Guides and tutorials accompanying models and other features
- Community Forums
  - Engage with other users, ask questions, and share insights.



Models

zhihan1996/DNABERT-2-117M
zhihan1996/DNA_bert_6
Peltarion/dnabert-minilm-small
Peltarion/dnabert-minilm
zhihan1996/DNA_bert_3
zhihan1996/DNABERT-S
→ See 182 model results for "DNABERT"

Datasets

simecek/Human_DNA_v0_DNABert6tokenized_stride1
simecek/Human_DNA_v0_DNABert6tokenized
MoL2/autotrain-data-dnabert_classfy_chr1
→ See 11 dataset results for "DNABERT"

Spaces

lokraj2/zhihan1996-DNABERT-2-117M

Core Components of the Hugging Face Hub

Models

- AI models for NLP, computer vision, audio processing, genomics
- Model architecture & weights
- Documentation describing usecases & limitations
- Example code & version history

Datasets

- Diverse data collections across domains and modalities
- Raw & processed formats optimized for ML
- Documentation on data collection & pre-processing
- Licensing and version control

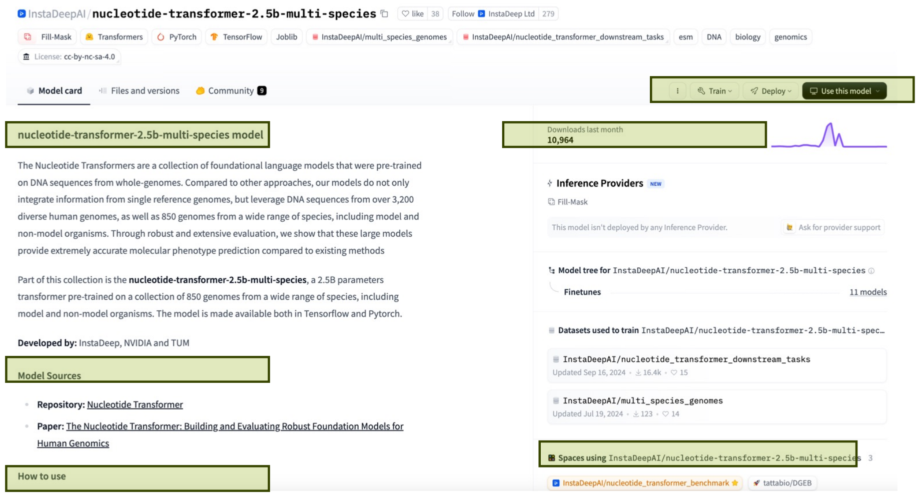
Spaces

- Interactive environments for demonstrating models directly in web browsers
- No local setup required
- Customizable User Interfaces for Model Interaction using front-end frameworks like Gradio and Streamlit

Hugging Face Model Card

- A Hugging Face Model Card is a standardized documentation format that provides important details about an AI model's
  - Intended use
  - Training data
  - Evaluation methods
  - Ethical considerations
  - Limitations

- Let's check [model card for Nucleotide Transformer model]()



nucleotide-transformer-2.5b-multi-species model

The Nucleotide Transformers are a collection of foundational language models that were pre-trained on DNA sequences from whole-genomes. Compared to other approaches, our models do not only integrate information from single reference genomes, but leverage DNA sequences from over 3,200 diverse human genomes, as well as 850 genomes from a wide range of species, including model and non-model organisms. Through robust and extensive evaluation, we show that these large models provide extremely accurate molecular phenotype prediction compared to existing methods

Part of this collection is the **nucleotide-transformer-2.5b-multi-species**, a 2.5B parameters transformer pre-trained on a collection of 850 genomes from a wide range of species, including model and non-model organisms. The model is made available both in Tensorflow and Pytorch.

Developed by: InstaDeep, NVIDIA and TUM

Model Sources

- **Repository:** Nucleotide Transformer
- **Paper:** The Nucleotide Transformer: Building and Evaluating Robust Foundation Models for Human Genomics

How to use

Downloads last month
10,964

Inference Providers NEW
Fill-Mask
This model isn't deployed by any Inference Provider. Ask for provider support

Model tree for InstaDeepAI/nucleotide-transformer-2.5b-multi-species
Finetunes 11 models

Datasets used to train InstaDeepAI/nucleotide-transformer-2.5b-multi-spec…
InstaDeepAI/nucleotide_transformer_downstream_tasks
Updated Sep 16, 2024 • 16.4k • 15
InstaDeepAI/multi_species_genomes
Updated Jul 19, 2024 • 123 • 14

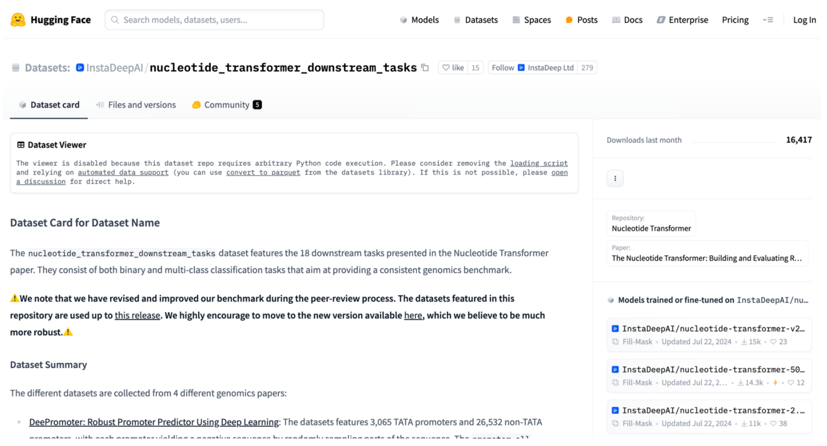
Spaces using InstaDeepAI/nucleotide-transformer-2.5b-multi-species 3
InstaDeepAI/nucleotide_transformer_benchmark
tattabio/DGEB

python

```python
from huggingface_hub import hf_hub_download

# Specify the model ID and the filename of the model card
model_id = "bert-base-uncased"  # Replace with your desired model ID
filename = "README.md"  # Model cards are typically stored in README.md

# Download the model card
model_card_path = hf_hub_download(repo_id=model_id, filename=filename)

# Read and print the content of the model card
with open(model_card_path, "r", encoding="utf-8") as f:
    model_card_content = f.read()

print(model_card_content)
```

Hugging Face Dataset Card

- A Hugging Face Model Card is a standardized documentation format that provides important details about a HuggingFace dataset's
  - Purpose, contents, and structure
  - Data formats, splits, and features
  - Data collection methods, sources, and processing steps
  - Intended applications and potential limitations
  - Known biases or risks associated with the dataset
  - References for citing the dataset
- Let's check dataset card for genomic downstream task dataset



Datasets: InstaDeepAI / nucleotide_transformer_downstream_tasks

Dataset Viewer

The viewer is disabled because this dataset repo requires arbitrary Python code execution. Please consider removing the loading script and relying on automated data support (you can use convert to parquet from the datasets library). If this is not possible, please open a discussion for direct help.

Dataset Card for Dataset Name

The nucleotide_transformer_downstream_tasks dataset features the 18 downstream tasks presented in the Nucleotide Transformer paper. They consist of both binary and multi-class classification tasks that aim at providing a consistent genomics benchmark.

⚠️**We note that we have revised and improved our benchmark during the peer-review process. The datasets featured in this repository are used up to this release. We highly encourage to move to the new version available here, which we believe to be much more robust.**⚠️

Dataset Summary

The different datasets are collected from 4 different genomics papers:

- DeePromoter: Robust Promoter Predictor Using Deep Learning: The datasets features 3,065 TATA promoters and 26,532 non-TATA

Downloads last month 16,417

Repository: Nucleotide Transformer

Paper: The Nucleotide Transformer: Building and Evaluating R...

Models trained or fine-tuned on InstaDeepAI/nu...

InstaDeepAI/nucleotide-transformer-v2...
Fill-Mask • Updated Jul 22, 2024 • 15k • 23

InstaDeepAI/nucleotide-transformer-50...
Fill-Mask • Updated Jul 22, 2... • 14.3k • 12

InstaDeepAI/nucleotide-transformer-2...
Fill-Mask • Updated Jul 22, 2024 • 11k • 38

python

```python
from datasets import load_dataset

# Load the dataset
dataset = load_dataset("InstaDeepAI/multi_species_genomes")

# Access the train split
train_dataset = dataset["train"]

# Display the first example
print(train_dataset[0])
```

Hugging Face API

- A programmatic interface to access Hugging Face's models and resources

- Key Components:
  - Inference API - For model predictions
  - Hub API - For interacting with the model repository
  - Datasets API - For working with datasets
  - Spaces API - For managing demo applications

- Common Use Cases:

1. Research Prototyping
   - Quickly test models without setup overhead
   - Experiment with different model architectures
   - Compare results across multiple models

2. Pipeline Integration
   - Embed predictions in larger genomic workflows
   - Chain models for complex analysis tasks
   - Integrate with existing bioinformatics tools

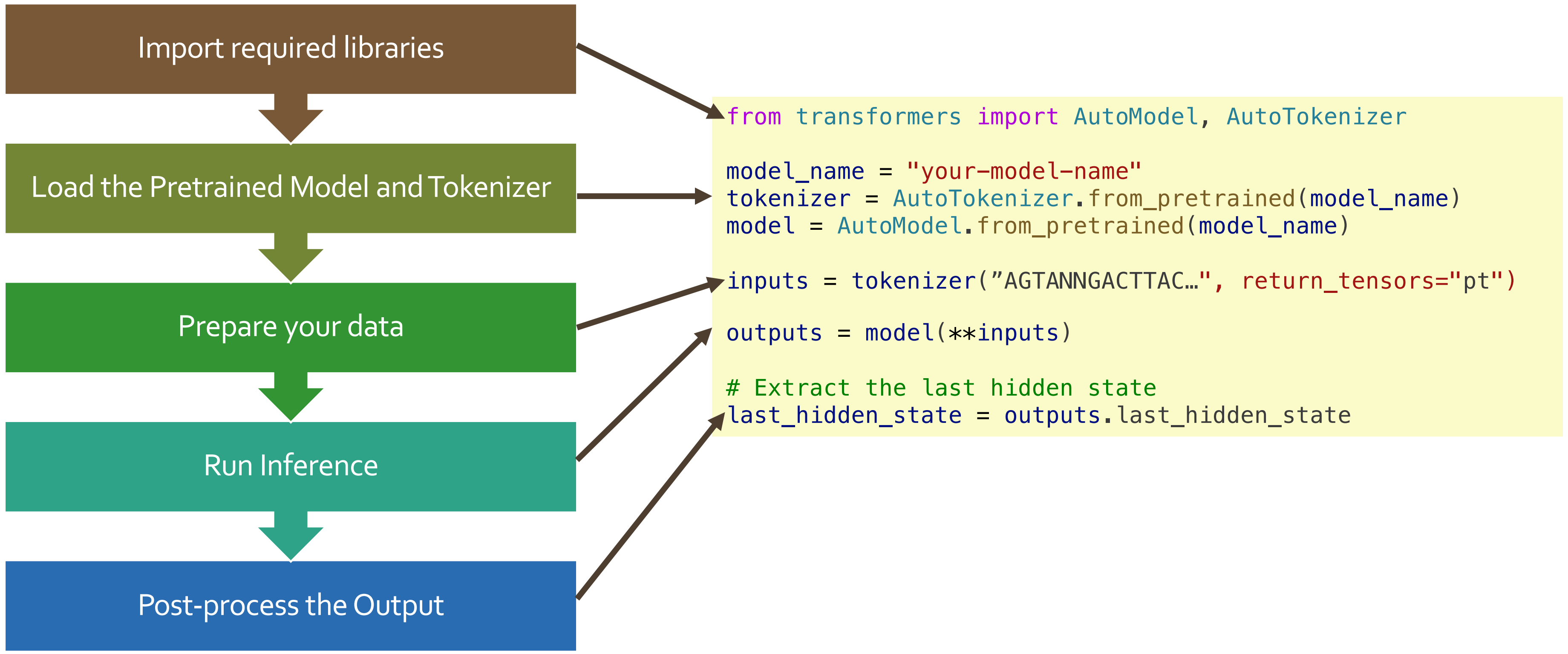
Inference API Example

```
import requests

# Step 1: Set your API token (get this from
huggingface.co/settings/tokens)
API_TOKEN = "hf_..." # Replace with your token # # Step

2: Define the model endpoint - using Nucleotide Transformer
API_URL = "https://api-
inference.huggingface.co/models/InstaDeepAI/nucleotide-
transformer-500m-human-ref"
headers = {"Authorization": f"Bearer {API_TOKEN}"}

# Step 3: Define a sample DNA sequence with unknown nucleotides
(N) # This is a partial promoter sequence with masked positions
dna_sequence = "ACGTGCACGGACTCAGCANNN" print(f"Original
sequence: {dna_sequence}")

# Step 4: Create the request payload - replace 'N' with '[MASK]'
masked_sequence = dna_sequence.replace("N", "[MASK]")
payload = {"inputs": masked_sequence}

# Step 5: Make the API call and
response = requests.post(API_URL, headers=headers, json=payload)
results = response.json()
print(results)
```

Hub API Example

```python
from huggingface_hub import HfApi, hf_hub_download, list_models
from transformers import AutoTokenizer, AutoModelForMaskedLM
import torch
import pandas as pd
import os

# Step 1: Initialize the Hub API with your token
api = HfApi(token="hf_...") # Replace with your token

# Step 2: Search for genomic models on the Hub
print("Searching for genomic models...")
genomic_models = list(list_models(filter="genomics"))

# Display some basic information about the first 5 models
print(f"Found {len(genomic_models)} genomic models. Here are the first 5:")

for i, model in enumerate(genomic_models[:5]):
        print(f"{i+1}. {model.modelId} - {model.downloads:,} downloads")
        print(f" Tags: {', '.join(model.tags)}")
        print(f" Last modified: {model.lastModified}")
        print("-" * 50)
```

Main steps in Hugging Face model usage



Import required libraries
Load the Pretrained Model and Tokenizer
Prepare your data
Run Inference
Post-process the Output

```
from transformers import AutoModel, AutoTokenizer

model_name = "your-model-name"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

inputs = tokenizer(”AGTANNGACTTAC…", return_tensors="pt")

outputs = model(**inputs)

# Extract the last hidden state
last_hidden_state = outputs.last_hidden_state
```

DNA Sequence Analysis Using HuggingFace

A Code Walkthrough

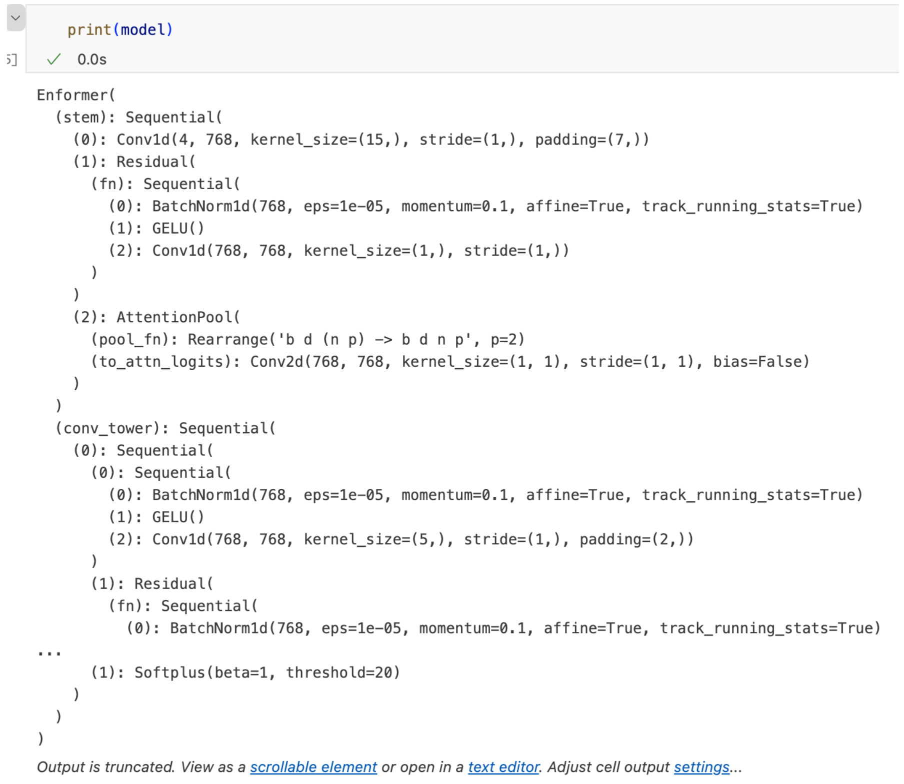
Open Jupyter notebook

DNA Sequence Analysis with Transformer Models: Code Walkthrough

```
from transformers import AutoTokenizer,
AutoModelForMaskedLM
import torch
import pandas as pd

# 1. Load a popular genomic foundation model:
Nucleotide Transformer
print("Loading Nucleotide Transformer model...")
model_name = "InstaDeepAI/nucleotide-
transformer-500m-human-ref"

# 500M parameter version # Load tokenizer and
model tokenizer =
AutoTokenizer.from_pretrained(model_name)
model =
AutoModelForMaskedLM.from_pretrained(model_name)
```

```
# 2. Display basic model information

print("\nBasic Model Information:")
print(f"Model name: {model_name}")
print(f"Model type: {model.__class__.__name__}")
print(f"Number of parameters: {sum(p.numel() for p in
model.parameters()) / 1_000_000:.2f} million")
print(f"Number of layers:
{model.config.num_hidden_layers}")
print(f"Hidden size: {model.config.hidden_size}")
print(f"Vocabulary size: {model.config.vocab_size}")
print(f"Maximum sequence length:
{model.config.max_position_embeddings}")

# 3. Explore the tokenizer vocabulary
print("\nTokenizer Information:")
print(f"Tokenizer type: {tokenizer.__class__.__name__}")
print(f"Vocabulary size: {len(tokenizer.vocab)}")
print(f"Mask token: {tokenizer.mask_token}")
print(f"Special tokens: {tokenizer.all_special_tokens}")
```

Questions & Comments



```
print(model)
```

✓ 0.0s

```
Enformer(
  (stem): Sequential(
    (0): Conv1d(4, 768, kernel_size=(15,), stride=(1,), padding=(7,))
    (1): Residual(
      (fn): Sequential(
        (0): BatchNorm1d(768, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): GELU()
        (2): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
      )
    )
    (2): AttentionPool(
      (pool_fn): Rearrange('b d (n p) -> b d n p', p=2)
      (to_attn_logits): Conv2d(768, 768, kernel_size=(1, 1), stride=(1, 1), bias=False)
    )
  )
  (conv_tower): Sequential(
    (0): Sequential(
      (0): Sequential(
        (0): BatchNorm1d(768, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (1): GELU()
        (2): Conv1d(768, 768, kernel_size=(5,), stride=(1,), padding=(2,))
      )
      (1): Residual(
        (fn): Sequential(
          (0): BatchNorm1d(768, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
...
      (1): Softplus(beta=1, threshold=20)
    )
  )
)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...