Exploring_Generative_AI_Libraries

March 31, 2025

1 Exploring Generative AI Libraries

Estimated time needed: **60** minutes

1.1 Table of Contents

Objectives

What is generative AI?

Real-world impact of generative AI

Art and creativity

Natural language processing (NLP)

Computer vision

Virtual avatars

Text generation before transformers

N-gram language models

Recurrent neural networks (RNN)

Long short-term memory (LSTM) and gated recurrent units (GRUs)

Seq2seq models with attention

```
</li>
<li><a href="#Transformers">Transformers</a>
    <ol>
        <li><a href="#Implementation:-Building-a-simple-chatbot-with-transformers">Implementati
            <ol>
                <li><a href="#Step-1:-Installing-libraries">Step 1: Installing libraries</a>
                <li><a href="#Step-2:-Importing-the-required-tools-from-the-transformers-libra
                    Step 2: Importing the required tools from the transformers library</a>
        </li>
    </ol>
</li>
```

1.2 Objectives

After completing this lab, you will be able to:

- Gain an understanding of generative AI and its impact across various domains.
- Familiarize yourself with different types of models in generative AI.
- Acquire the skills to build and interact with a chatbot using transformers.

1.3 What is generative AI?

Imagine presenting a computer with a vast array of paintings. After analyzing them, it tries to craft a unique painting of its own. This capability is termed generative AI. Essentially, the computer derives inspiration from the provided content and uses it to create something new.

1.4 Real-world impact of generative AI

Generative AI is transforming multiple industries. Its applications span:

1.4.1 1. Art and creativity

- Generative art: Artists employing generative AI algorithms can create stunning artworks by learning from existing masterpieces and producing unique pieces inspired by them. These AI-generated artworks have gained recognition in the art world.
- Music Composition: Projects in the realm of generative AI have been employed to compose music. They learn from a vast data set of musical compositions and can generate original pieces in various styles, from classical to jazz, revolutionizing the music industry.

1.4.2 2. Natural language processing (NLP)

- Content generation: Tools like generative pre-trained transformer (GPT) have demonstrated their ability to generate coherent and context-aware text. They can assist content creators by generating articles, stories, or marketing copy, making them valuable tools in content creation.
- Chatbots and virtual assistants: Generative AI powers many of today's chatbots and virtual assistants. These AI-driven conversational agents understand and generate human-like responses, enhancing user experiences.
- Code Writing: Generative AI models can also produce code snippets based on descriptions or requirements, streamlining software development.

1.4.3 3. Computer vision

- Image synthesis: Models like data analysis learning with language model for generation and exploration, frequencly known as DALL-E, can generate images from textual descriptions. This technology finds applications in graphic design, advertising, and creating visual content for marketing.
- Deepfake detection: With the advancement in generative AI techniques, the generation of deep fake content is also on the rise. Consequently, generative AI now plays a role in developing tools and techniques to detect and combat the spread of misinformation through manipulated videos.

1.4.4 4. Virtual avatars

- Entertainment: Generative AI is utilized to craft virtual avatars for gaming and entertainment. These avatars mimic human expressions and emotions, bolstering user engagement in virtual environments.
- Marketing: Virtual influencers, propelled by generative AI, are on the rise in digital marketing. Brands are harnessing these virtual personas to endorse their products and services.

1.5 Neural structures behind generative AI

Before we had the powerful transformers, which are like super-fast readers and understand lots of words at once, there were other methods used for making computers generate text. These methods were like the building blocks that led to the amazing capabilities we have today.

1.6 Large language models (LLMs)

Large language models are like supercharged brains. They are massive computer programs with lots of "neurons" that learn from huge amounts of text. These models are trained to do tasks like understanding and generating text, and they're used in many applications. However, there's a limitation: these models are not very good at understanding the bigger context or the meaning of words. They work well for simple predictions but struggle with more complex text.

1.7 Text generation before transformers

1.7.1 1. N-gram language models

N-gram models are like language detectives. They predict what words come next in a sentence based on the words that came before. For example, if you say "The sky is," these models guess that the next word might be "blue."

1.7.2 2. Recurrent neural networks (RNN)

Recurrent neural networks (RNNs) are specially designed to handle sequential data, making them a powerful tool for applications like language modeling and time series forecasting. The essence of their design lies in maintaining a 'memory' or 'hidden state' throughout the sequence by employing loops. This enables RNNs to recognize and capture the temporal dependencies inherent in sequential data. - Hidden state: Often referred to as the network's 'memory', the hidden state is a dynamic storage of information about previous sequence inputs. With each new input, this hidden state is updated, factoring in both the new input and its previous value. - Temporal dependency: Loops in RNNs enable information transfer across sequence steps.

Image Source

Illustration of RNN's operation: Consider a simple sequence, such as the sentence: "I love RNNs". The RNN interprets this sentence word by word. Beginning with the word "I", the RNN ingests it, generates an output, and updates its hidden state. Moving on to "love", the RNN processes it alongside the updated hidden state which already holds insights about the word "I". The hidden state is updated again post this. This pattern of processing and updating continues until the last word. By the end of the sequence, the hidden state ideally encapsulates insights from the entire sentence.

1.7.3 3. Long short-term memory (LSTM) and gated recurrent units (GRUs)

Long short-term memory (LSTM) and gated recurrent units (GRUs) are advanced variations of recurrent neural networks (RNNs), designed to address the limitations of traditional RNNs and enhance their ability to model sequential data effectively. They processed sequences one element at a time and maintained an internal state to remember past elements. While they were effective for a variety of tasks, they struggled with long sequences and long-term dependencies.

1.7.4 4. Seq2seq models with attention

- Sequence-to-sequence (seq2seq) models, often built with RNNs or LSTMs, were designed to handle tasks like translation where an input sequence is transformed into an output sequence.
- The attention mechanism was introduced to allow the model to "focus" on relevant parts of the input sequence when generating the output, significantly improving performance on tasks like machine translation.

While these methods provided significant advancements in text generation tasks, the introduction of transformers led to a paradigm shift. Transformers, with their self-attention mechanism, proved to be highly efficient at capturing contextual information across long sequences, setting new benchmarks in various NLP tasks.

1.8 Transformers

Proposed in a paper titled "Attention Is All You Need" by Vaswani et al. in 2017, the transformer architecture replaced sequential processing with parallel processing. The key component behind its success? The attention mechanism, more precisely, self-attention.

Key steps include: - Tokenization: The first step is breaking down a sentence into tokens (words or subwords). - Embedding: Each token is represented as a vector, capturing its meaning. - Self-attention: The model computes scores determining the importance of every other word for a particular word in the sequence. These scores are used to weight the input tokens and produce a new representation of the sequence. For instance, in the sentence "He gave her a gift because she'd helped him", understanding who "her" refers to requires the model to pay attention to other words in the sentence. The transformer does this for every word, considering the entire context, which is particularly powerful for understanding meaning. - Feed-forward neural networks: After attention, each position is passed through a feed-forward network separately. - Output sequence: The model produces an output sequence, which can be used for various tasks, like classification, translation, or text generation. - Layering: Importantly, transformers are deep models with multiple layers of attention and feed-forward networks, allowing them to learn complex patterns.

The architecture's flexibility has allowed transformers to be used beyond NLP, finding applications in image and video processing too. In NLP, transformer-based models like BERT, GPT, and their variants have set state-of-the-art results in various tasks, from text classification to translation.

1.8.1 Implementation: Building a simple chatbot with transformers

Now, you will build a simple chatbot using `transformers` library from Hugging Face, which is an open-source natural language processing (NLP) toolkit with many useful features. #### Step 1: Installing libraries

[1]:
```
!pip install -qq tensorflow
!pip install transformers==4.42.1 -U
!pip install sentencepiece
!pip install torch==2.2.2
!pip install torchtext==0.17.2
!pip install numpy==1.26
#!pip install --upgrade numpy transformers torch
```

```
Requirement already satisfied: transformers==4.42.1 in
/opt/conda/lib/python3.12/site-packages (4.42.1)
Requirement already satisfied: filelock in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (0.29.3)
Requirement already satisfied: numpy<2.0,>=1.17 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (1.26.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (2024.11.6)
Requirement already satisfied: requests in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (2.32.3)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (0.19.1)
Requirement already satisfied: safetensors>=0.4.1 in
/opt/conda/lib/python3.12/site-packages (from transformers==4.42.1) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /opt/conda/lib/python3.12/site-
packages (from transformers==4.42.1) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/opt/conda/lib/python3.12/site-packages (from huggingface-
hub<1.0,>=0.23.2->transformers==4.42.1) (2025.3.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/opt/conda/lib/python3.12/site-packages (from huggingface-
hub<1.0,>=0.23.2->transformers==4.42.1) (4.12.2)
Requirement already satisfied: charset_normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from requests->transformers==4.42.1)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from requests->transformers==4.42.1) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from requests->transformers==4.42.1)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.12/site-packages (from requests->transformers==4.42.1)
(2024.12.14)
Requirement already satisfied: sentencepiece in /opt/conda/lib/python3.12/site-
```

```
packages (0.2.0)
Requirement already satisfied: torch==2.2.2 in /opt/conda/lib/python3.12/site-
packages (2.2.2)
Requirement already satisfied: filelock in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2) (3.18.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (4.12.2)
Requirement already satisfied: sympy in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2) (1.13.3)
Requirement already satisfied: networkx in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2) (3.4.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2) (3.1.5)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2) (2025.3.1)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2) (12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
/opt/conda/lib/python3.12/site-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch==2.2.2) (12.8.93)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.12/site-packages (from jinja2->torch==2.2.2) (3.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/opt/conda/lib/python3.12/site-packages (from sympy->torch==2.2.2) (1.3.0)
Requirement already satisfied: torchtext==0.17.2 in
/opt/conda/lib/python3.12/site-packages (0.17.2)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.12/site-packages
(from torchtext==0.17.2) (4.67.1)
```

```
Requirement already satisfied: requests in /opt/conda/lib/python3.12/site-
packages (from torchtext==0.17.2) (2.32.3)
Requirement already satisfied: torch==2.2.2 in /opt/conda/lib/python3.12/site-
packages (from torchtext==0.17.2) (2.2.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.12/site-packages
(from torchtext==0.17.2) (1.26.0)
Requirement already satisfied: filelock in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2->torchtext==0.17.2) (3.18.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(4.12.2)
Requirement already satisfied: sympy in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2->torchtext==0.17.2) (1.13.3)
Requirement already satisfied: networkx in /opt/conda/lib/python3.12/site-
packages (from torch==2.2.2->torchtext==0.17.2) (3.4.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2->torchtext==0.17.2) (3.1.5)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.12/site-packages
(from torch==2.2.2->torchtext==0.17.2) (2025.3.1)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
```

```
(2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
/opt/conda/lib/python3.12/site-packages (from torch==2.2.2->torchtext==0.17.2)
(12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
/opt/conda/lib/python3.12/site-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch==2.2.2->torchtext==0.17.2) (12.8.93)
Requirement already satisfied: charset_normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from requests->torchtext==0.17.2)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from requests->torchtext==0.17.2) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from requests->torchtext==0.17.2)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.12/site-packages (from requests->torchtext==0.17.2)
(2024.12.14)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.12/site-packages (from
jinja2->torch==2.2.2->torchtext==0.17.2) (3.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/opt/conda/lib/python3.12/site-packages (from
sympy->torch==2.2.2->torchtext==0.17.2) (1.3.0)
Requirement already satisfied: numpy==1.26 in /opt/conda/lib/python3.12/site-
packages (1.26.0)
```

Step 2: Importing the required tools from the transformers library In the upcoming script, you initiate variables using two invaluable classes from the transformers library: - `model` is an instance of the class `AutoModelForSeq2SeqLM`. This class lets you interact with your chosen language model. - `tokenizer` is an instance of the class `AutoTokenizer`. This class streamlines your input and presents it to the language model in the most efficient manner. It achieves this by converting your text input into "tokens", which is the model's preferred way of interpreting text. You choose "facebook/blenderbot-400M-distill" for this example model because it is freely available under an open-source license and operates at a relatively brisk pace. For a diverse range of models and their capabilities, you can explore the Hugging Face website: Hugging Face Models.

[2]:
```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

# Selecting the model. You will be using "facebook/blenderbot-400M-distill" in␣
 ↪this example.
model_name = "facebook/blenderbot-400M-distill"

# Load the model and tokenizer
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Following the initialization, let's set up the chat function to enable real-time interaction with the chatbot.

[ ]:
```
# Define the chat function
def chat_with_bot():
    while True:
        # Get user input
        input_text = input("You: ")

        # Exit conditions
        if input_text.lower() in ["quit", "exit", "bye"]:
            print("Chatbot: Goodbye!")
            break

        # Tokenize input and generate response
        inputs = tokenizer.encode(input_text, return_tensors="pt")
        outputs = model.generate(inputs, max_new_tokens=150)
        response = tokenizer.decode(outputs[0], skip_special_tokens=True).
 ↪strip()

        # Display bot's response
        print("Chatbot:", response)

# Start chatting
chat_with_bot()
```

Alright! You have successfully interacted with your chatbot. By providing it with a prompt, the chatbot used the power of the transformers library and the underlying model to generate a response. This exemplifies the prowess of transformer-based models in comprehending and generating human-like text based on a given context. As you continue to engage with it, you will observe its capacity to simulate a wide range of conversational topics and styles.

Step 3: Trying another language model and comparing the output You can use a different language model, for example the "flan-t5-base" model from Google, to create a similar chatbot. You can use a chat function similar to the one defined in Step 2 and compare the outputs of both models.

[ ]:
```
import sentencepiece
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

model_name = "google/flan-t5-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

[ ]:
```
### Let's chat with another bot
def chat_with_another_bot():
    while True:
        # Get user input
        input_text = input("You: ")
        # Exit conditions
        if input_text.lower() in ["quit", "exit", "bye"]:
            print("Chatbot: Goodbye!")
            break

        # Tokenize input and generate response
        inputs = tokenizer.encode(input_text, return_tensors="pt")
        outputs = model.generate(inputs, max_new_tokens=150)
        response = tokenizer.decode(outputs[0], skip_special_tokens=True).
 ↪strip()

        # Display bot's response
        print("Chatbot:", response)

# Start chatting
chat_with_another_bot()
```

There are many language models available in Hugging Face. In the following exercise, you will compare the output for the same input using two different models.

2 Exercise

2.0.1 Create a chatbot using different models from Hugging Face

Create a simple chatbot using the transformers library from Hugging Face(https://huggingface.co/models). Run the code using the following models and compare the output. The models are “google/flan-t5-small”, “facebook/bart-base”. (Note: Based on the selected model, you may notice differences in the chatbot output. Multiple factors, such as model training and fine-tuning, influence the output.)

```
[ ]: # Add code for the exercise here:
```

Click here for Solution

```
import sentencepiece
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

model_name = "google/flan-t5-small" #here the model name can be changed as you like.
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

---

3 Congratulations! You have completed the lab.

3.1 Authors

Vicky Kuo is completing her Master’s degree in IT at York University with scholarships. Her master's thesis explores the optimization of deep learning algorithms, employing an innovative approach to scrutinize and enhance neural network structures and performance.

```
{## Change Log}

{|Date (YYYY-MM-DD)|Version|Changed By|Change Description|}           {|-|-|-|-|}
{|2023-09-10|0.1|Vicky Kuo|Initial Lab Created|}
```