Gen AI and LLMs : Architecture and Data Preparation

Course Overview

Welcome to this short course on generative AI and data preparation for large language models (LLMs). This is the first of the series of courses in the Generative AI Engineering Essentials with LLMs Professional Certificate.

This course is your first step toward understanding how you can use generative AI architectures and LLMs, such as generative pre-trained transformers (GPT), for natural language processing (NLP). It is an ideal course to learn how text is preprocessed and loaded for effective training and analysis by LLMs.

Let's get into the details.

Pre-requisites: For this course, a basic knowledge of Python and PyTorch and an awareness of machine learning and neural networks would be an advantage, though not strictly required.

In this course, you will learn about the significance of generative AI in various domains. You will differentiate between generative AI models and learn how to apply LLMs for NLP tasks.

You will get hands-on opportunities through labs to explore generative AI libraries, implement tokenization, and create an NLP data loader.

After completing this course, you will be able to:

- Explain the significance of generative AI in various domains.
- Differentiate between various generative AI architectures and models.
- Describe the use of LLMs for NLP tasks.
- Describe the key features and significance of the libraries and tools used in generative AI for language processing.
- Use Hugging Face libraries in a Jupyter environment to explore generative AI techniques and build a simple chatbot using the Transformers library.
- Explain the tokenization process, tokenization methods, and the use of tokenizers.
- Implement tokenization.
- Explain how data loaders are used for training generative AI models.
- Create an NLP data loader.

Who should take this course?

This course is suitable for those interested in AI engineering, including training, developing, fine-tuning, and deploying LLMs. It is specifically designed for those who want to learn about generative AI and LLMs and text data preprocessing using tokenizers and data loaders.

Recommended background

As this is an intermediate-level course, it assumes that you have a basic knowledge of Python and PyTorch and a familiarity with machine learning and neural network concepts.

Course content

This course is approximately four hours and is divided into two modules. You can complete one module a week or at a pace that suits you.

Week 1 - Module 1: Generative AI Architecture

This module will teach you the basics of generative AI and using LLMs for NLP tasks. You will be able to describe the types of generative AI and its real-world applications. You will learn to differentiate between various generative AI architectures and models, such as transformers, recurrent neural networks (RNNs), diffusion models, generative adversarial networks (GANs) and variational autoencoders (VAEs). You will learn the differences in the training approaches used for each model. You will be able to explain the use of various generative AI architectures for NLP tasks and the key features and significance of the libraries and tools used in NLP.

The knowledge acquired in this module will help you use the generative AI libraries in Hugging Face. You will be able to explore generative AI libraries and build a simple chatbot using the Transformers library from Hugging Face.

Week 2 - Module 2: Data Preparation for LLMs

In Module 2, you will learn to prepare data for training LLMs. You will learn about tokenization methods and the use of tokenizers for word-based, character-based, and subword-based tokenization. You will be able to explain how you can use data loaders for training generative AI models and list the PyTorch libraries for preparing and handling data within data loaders.

You will implement tokenization using various libraries such as nltk, spaCy, BertTokenizer, and XLNetTokenizer. You will also create a data loader and use the collate function to process batches of text.

The module includes a cheat sheet with quick reference content, such as code snippets. A glossary will help you review the technical terms used in the course. The module concludes with a final graded quiz.

Please note: This course does not cover data acquisition techniques such as web crawling, initial data cleaning procedures or the use of regular expressions. When you practice the lab exercises, you may want to ensure that you have well-prepared data by removing unwanted characters, symbols, etc.

Learning resources

The course offers a variety of learning assets: videos, readings, hands-on labs, a cheat sheet, a glossary, and quizzes.

The **videos** and **readings** present the instruction, supported by **labs** that provide hands-on learning experiences.

The **cheat sheet** provides quick reference material, such as code snippets.

The **glossary** provides a reference list for all the technical terms used in the course, along with their definitions.

Practice quizzes at the end of each lesson test your understanding of what you learned, and the **graded quizzes** will assess your conceptual understanding of the course.

We wish you good luck completing the course and getting the most out of it!

Module 1: Gen AI Architecture.

Summary and Highlights

Congratulations! You have completed this lesson. At this point in the course, you know that:

- Generative AI refers to deep-learning models that can generate content, such as text, images, audio, 3D objects, and music, based on the training data.
- These models understand the relationship between words and phrases and generate contextually relevant text. An example of a generative AI model for text generation is a generative pre-trained transformer (GPT).
- These models can generate images from text input and seed images. Examples of generative AI models for image generation are data analysis learning with language model for generation and exploration (DALL-E) and generative adversarial networks (GANs).
- Generative AI models can generate natural-sounding speech. An example of this type of generative AI model is WaveNet.
- Generative AI architectures and models include recurrent neural networks (RNNs), transformers, GANs, variational autoencoders (VAEs), and diffusion models.
    - RNNs use sequential or time series data and a loop-based design for training.
    - Transformers utilize the self-attention mechanism to focus on the most important parts of the information.
    - GANs consist of a generator and a discriminator, which work in a competitive mode.
    - VAEs operate on an encoder-decoder framework and create samples based on similar characteristics.
    - Diffusion models generate creative images by learning to remove noise and reconstruct distorted examples, relying on statistical properties.
- Generative AI started with rule-based systems that use predefined linguistic rules, followed by machine-learning approaches focusing on statistical methods. It later moved to deep learning, which uses neural networks trained on extensive data sets. Transformers represent the latest in this evolution.
- The evolution of generative AI has led to advancements in machine translation, chatbot conversations, sentiment analysis, and text summarization.
- Large language models (LLMs) are foundation models that use AI and deep learning with vast data sets. LLMs have training data sets that run into petabytes and contain billions of parameters. Examples of LLMs are GPT, Bidirectional Encoder Representations from Transformers (BERT), Bidirectional and Auto-Regressive Transformers (BART), and Text-to-Text Transfer Transformer (T5).
- Some libraries and tools you can use to implement generative AI for NLP are PyTorch, TensorFlow, Hugging Face, LangChain, and Pydantic.

Basics of AI Hallucinations (Reading 1a)

Objectives

After completing this reading, you will be able to:

- Define AI hallucinations.
- List the problems caused by AI hallucinations and the ways of preventing these.

Estimated reading time: 10 minutes

Introduction

You can utilize large language models (LLMs) to generate authoritative text across domains. However, they may generate information that sounds right but is inaccurate. They may also produce biased content. These problems can be a result of AI hallucinations. In this reading, you will learn about AI hallucinations.

AI hallucinations

In AI hallucinations, the model generates output that it presents as accurate but is seen as unrealistic, inaccurate, irrelevant, or nonsensical by humans. It is similar to the way humans experience hallucinations.

For example, there was an incident where ChatGPT falsely claimed that a mayor in Australia was found guilty and imprisoned in a bribery case. In reality, the mayor notified the authorities about a bribery issue. (Reference: Australian mayor readies world's first defamation lawsuit over ChatGPT content | Reuters)

The example shows that there can be a significant implication of AI hallucinations. However, note that such incidents are rare and isolated.

AI hallucinations are strongly associated with LLMs. Factors such as biases in the training data, limited training, complexity of the model, and lack of human oversight can cause AI hallucinations. Also, the outputs generated by the AI models might not be based on the patterns the models learned from the training data.

Problems caused by AI hallucinations

AI hallucinations can have serious implications. For example, if an LLM summarizes pages of a legal document incorrectly, it can lead to legal disputes and litigation.

Some of the problems caused are:

- Generation of inaccurate information
- Creation of biased views or misleading information
- Wrong input provided to sensitive applications, such as those used in autonomous vehicles or medical domain

Methods for mitigating hallucinations

- Eliminating any bias in the training data and performing extensive training of the models on high-quality data
- Avoiding manipulation of the inputs that are fed into the models
- Ongoing evaluation and improvement of the models
- Fine-tuning a pre-trained LLM on domain-specific data

Preventing the problems caused by AI hallucinations

It is inevitable for hallucinations to occur within LLMs. What can be frustrating is that the generated text often contains subtle mistakes that are challenging to identify. There are a couple of best practices that you can follow. These include:

- Being vigilant and understanding that these models do not understand the actual meaning of the words but are focused on predicting the next word in a sequence based on patterns. These models are trained on vast amounts of data and learn statistical patterns, but they lack semantic understanding or comprehension **like human beings**.
- Ensuring human oversight regularly for fact-checking and continuous testing
- Providing additional context in the prompt or input. This will enable LLMs to understand the desired output better and generate more accurate and contextually relevant responses.

Summary

In this reading, you learned that:

- AI hallucinations refer to an AI model generating output presented as accurate but seen as unrealistic, inaccurate, irrelevant, or nonsensical by humans.
- AI hallucination can result in the generation of inaccurate information, the creation of biased views, and wrong input provided to sensitive applications.
- You can prevent the problems caused by AI hallucinations through:
  - Extensive training with high-quality data,
  - Avoiding manipulation,
  - Ongoing evaluation and improvement of the models,
  - Fine-tuning on domain-specific data,
  - Being vigilant,
  - Ensuring human oversight, and
  - Providing additional context in the prompt.

Overview of Libraries and Tools (Reading 1b)

Objective(s)

After completing this reading, you will be able to:

- Describe the key features and the significance of the libraries and tools used in generative AI for natural language processing (NLP).

Estimated reading time: 20 minutes

Introduction

As an AI engineer or NLP engineer, you will design, develop, and deploy generative AI applications for NLP. However, you may face challenges as these applications require a deep understanding of linguistic nuances.

The good news is that due to the rapid advancement and evolution of generative AI for NLP, you now have increased accessibility to various tools and libraries. The unique capabilities of these tools and libraries enable you to explore the full potential of what you can achieve using generative AI.

Some key libraries and tools are PyTorch, TensorFlow, Hugging Face, LangChain, and Pydantic.

Let's explore these libraries and tools.

PyTorch

PyTorch is an open source deep learning framework originally developed by Facebook's AI Research lab (now Meta). It is a Python-based library well-known for its ease of use, flexibility, and dynamic computation graphs. A dynamic computation graph means that the network's structure can change on the fly during execution, allowing for more intuitive and flexible model development. This feature is helpful in advanced NLP applications where the neural network architecture needs to adapt dynamically to varying inputs.

PyTorch is highly customizable and utilized in many companies such as OpenAI and Tesla.

Let's look at the key features and the significance of PyTorch.

Dynamic computation graphs (Autograd)

PyTorch's Autograd system allows dynamic changes to the network during training, enhancing flexibility and easing the development process. This adaptability is particularly beneficial for research and experimentation.

Rich ecosystem and community

PyTorch has a comprehensive ecosystem, offering tools for various machine learning domains such as computer vision and NLP. The vibrant community contributes extensive resources, including tutorials and third-party extensions. For example, torchtext is a library within the PyTorch ecosystem. It provides resources such as data sets and pretrained models for loading, preprocessing, and handling text data.

Application in NLP

PyTorch is used to develop and train neural network models that can understand and generate human language. It is particularly favored for research and development as it provides an adaptable environment for model experimentation and prototyping.

You can learn more by visiting PyTorch. (Please note: If the link does not launch the page, right-click and open the link in a new tab.)

TensorFlow

TensorFlow, developed by Google, is an open-source framework for machine learning and deep learning. It provides tools and libraries to facilitate the development and deployment of machine learning models.

TensorFlow stands out for its robust architecture, suitable for both research and production environments.

Let's look at the key features and the significance of TensorFlow.

Scalability

TensorFlow is designed with a scalable architecture that facilitates a seamless transition from research prototypes to production. This feature streamlines the training process and large-scale deployment of machine learning models.

TensorFlow Extended (TFX)

TFX is a platform for deploying production-ready machine learning (ML) pipelines. The platform is built on the TensorFlow foundation. It provides functionality for integrating the various phases of a machine learning system deployment, such as defining, launching, and monitoring.

Keras integration

Keras and TensorFlow are tightly integrated. Keras provides a user-friendly high-level neural networks API, facilitating rapid prototyping and building and training deep learning models in TensorFlow.

Users can use TensorFlow's `tf.keras` module to define and train neural networks.

Application in NLP

You can use TensorFlow for typical NLP tasks such as sentiment analysis, text classification, and machine translation. It contains the latest AI models and libraries to help you work with raw text. Its capacity for handling large-scale deployments makes TensorFlow a preferred choice for enterprise-level NLP applications.

You can learn more by visiting TensorFlow. (Please note: If the link does not launch the page, right-click and open the link in a new tab.)

Hugging Face

Hugging Face is a platform that offers an open-source library with pretrained models and tools to streamline the process of training and fine-tuning generative AI models. Hugging Face has significantly impacted the field of NLP, making state-of-the-art technologies more accessible.

Let's look at the key features and the significance of Hugging Face.

Extensive model repository

The Hugging Face Model Hub is a comprehensive online platform that hosts a vast collection of pretrained machine learning models, primarily focusing on NLP. It allows users to easily share, discover, and use models across a wide range of applications, including text classification, translation, and question-answering. The Hub supports models from popular frameworks like PyTorch and TensorFlow, allowing developers to integrate models into their existing workflows and applications. It also provides detailed information about each model, including its architecture, training data, and usage examples. This community-driven platform is a key resource for researchers, developers, and AI enthusiasts, facilitating easy access and collaboration in the field of machine learning.

Simplicity

The platform simplifies the deployment of complex models, making cutting-edge NLP techniques accessible to beginners and experts. The Transformers library, a key component for NLP, provides a user-friendly interface to work with various pretrained models.

Community-driven development

With a strong focus on collaboration, Hugging Face fosters a vibrant community. The community enables a collaborative approach where developers and contributors share resources and expertise. They participate in creating and sharing NLP models and tools, increasing accessibility to models that you can use in a wide range of applications.

Application in NLP

Hugging Face is extensively used in NLP for tasks such as named entity recognition, sentiment analysis, and text summarization. Named entity recognition involves classifying named entities into predefined categories such as persons, locations, and so on. The readily available resources it provides streamline building and deploying NLP applications.

Some useful libraries in Hugging Face

1. **Transformers**: This is perhaps the most famous library from Hugging Face. It offers many pretrained models for working with text. Using these models, you can perform tasks such as text generation, summarization, translation, classification, and question-answering. The library is designed for PyTorch and TensorFlow.
2. **Datasets**: This library is designed to easily access and share large-scale data sets and evaluation metrics for NLP. It includes a wide array of data sets in different languages and for various tasks, making it easier for researchers and developers to benchmark and evaluate their models.
3. **Tokenizers**: This library is optimized for performance and versatility in tokenization, which is a critical step in preparing data for NLP models. It can handle all pre-tokenization requirements needed for models like BERT and GPT.

You can learn more by visiting Hugging Face. (Please note: If the link does not launch the page, right-click and open the link in a new tab.)

LangChain

LangChain is an open-source framework that helps streamline AI application development using large language models (LLMs), significantly improving LLM accessibility and functionality for diverse applications.

Let's look at the key features and the significance of LangChain.

Advanced prompt engineering

LangChain provides sophisticated tools for designing effective prompts, unlocking the full potential of language models. Prompts refer to specific inputs used to guide the model's behavior.

Prompt engineering is crucial for tailoring responses and guiding model outputs towards desired outcomes.

Seamless integration with leading models

LangChain offers compatibility with major models such as generative pre-trained transformers (GPT). This integration simplifies the development of applications built on these advanced models, enabling a smoother transition from concept to deployment.

Application in NLP

LangChain is an essential tool for developers aiming to leverage LLMs. Its versatility makes it ideal for creating tools such as interactive chatbots and intricate analytical tools.

LangChain's ability to harmonize model integration, prompt engineering, and application-specific customization makes it a pivotal tool in language model utilization.

You can learn more by visiting LangChain. (Please note: If the link does not launch the page, right-click and open the link in a new tab.)

Pydantic

Pydantic is a Python library that helps you streamline data handling. You can use it to parse and validate your data. It uses Python-type annotations.

Let's look at the key features and the significance of Pydantic.

Robust data validation

Pydantic ensures the accuracy of data types and formats before an application processes them. This feature enhances the reliability of applications. In Pydantic, you can use the `BaseModel` class to define data models and their validation rules.

Efficient settings management

Pydantic manages application settings and environment variables. This functionality is vital for the scalability of larger projects.

Application in NLP

Pydantic has an important role In NLP pipelines for validating and managing data. It ensures data integrity and consistency, especially when dealing with diverse and large data sets.

You can learn more by visiting Pydantic. (Please note: If the link does not launch the page, right-click and open the link in a new tab.)

Summary

In this reading, you learned that:

- There are various libraries and tools that you can use to develop NLP applications using generative AI. Some tools are PyTorch, TensorFlow, Hugging Face, LangChain, and Pydantic.
- PyTorch is an open source deep learning framework. It is a Python-based library well-known for its ease of use, flexibility, and dynamic computation graphs.
- TensorFlow is an open-source framework for machine learning and deep learning. It provides tools and libraries to facilitate the development and deployment of machine learning models.
- The tight integration of TensorFlow with Keras provides a user-friendly high-level neural networks API, facilitating rapid prototyping and building and training deep learning models.
- Hugging Face is a platform that offers an open-source library with pretrained models and tools to streamline the process of training and fine-tuning generative AI models. It offers libraries such as Transformers, Datasets, and Tokenizers.
- LangChain is an open-source framework that helps streamline AI application development using LLMs. It provides tools for designing effective prompts.
- Pydantic is a Python library that helps you streamline data handling. It ensures the accuracy of data types and formats before an application processes them.

Module 2 : Data Preparation for LLMs

Summary and Highlights

Congratulations! You have completed this lesson. At this point in the course, you know that:

- Tokenization and data loading are part of the data preparation activities for natural language processing (NLP).
- Tokenization breaks a sentence into smaller pieces or tokens.
- Tokenizers are essential tools that break down text into tokens. These tokens can be words, characters, or subwords, making complex text understandable to computers. Examples of tokenizers are natural language toolkit (NLTK) and spaCy.
- Word-based tokenization preserves the semantic meaning, though it increases the model's overall vocabulary.
- Character-based tokenization has smaller vocabularies but may not convey the same information as entire words.
- Subword-based tokenization allows frequently used words to stay unsplit while breaking down infrequent words.
    - Using the WordPiece, Unigram, and SentencePiece algorithms, you can implement subword-based tokenization.
- You can add special tokens such as <bos> at the beginning and <eos> at the end of a tokenized sentence.
- A data set in PyTorch is an object that represents a collection of data samples. Each data sample typically consists of one or more input features and their corresponding target labels.
- A data loader helps you prepare and load data to train generative AI models. Using data loaders, you can output data in batches instead of one sample at a time.
- Data loaders have several key parameters, including the data set to load from, batch size (determining how many samples per batch), shuffle (whether to shuffle the data for each epoch), and more. Data loaders also provide an iterator interface, making it easy to iterate over batches of data during training.
- PyTorch has a dedicated DataLoader class.
- Data loaders seamlessly integrate with the PyTorch training pipeline and simplify data augmentation and preprocessing.
- A collate function is employed in the context of data loading and batching in machine learning, particularly when dealing with variable-length data, such as sequences (e.g., text, time series, and sequences of events). Its primary purpose is to prepare and format individual data samples (examples) into batches that machine learning models can efficiently process.

Data Quality and Diversity for Effective LLM Training (Reading 2a)

Data quality and diversity for effective LLM training

The quality and diversity of data are foundational to developing robust and inclusive large language models (LLMs). As models advance in sophistication, well-curated data is more critical than ever. Below are key aspects of preparing high-quality data and why they are crucial for effective LLM training.

Data quality

Data quality refers to the accuracy, consistency, and completeness of the dataset used for training. Poor-quality data introduces noise that can lower a model's accuracy and reliability. Here are practices to ensure high data quality:

- **Noise reduction**: Remove irrelevant or repetitive data to help the model focus on significant patterns and linguistic structures. For example, clean datasets by removing typos and irrelevant information, such as forum tags, to enhance quality.
- **Consistency checks**: Regularly verify consistency to prevent conflicting or outdated information from confusing the model. Consistency is essential for entities, like public figure names or technical terms, ensuring uniform usage throughout the dataset.
- **Labeling quality**: For labeled datasets, accurate labeling is crucial to avoid misleading the model. Clear guidelines for human annotators can reduce subjective errors and improve labeling quality.

Diverse representation

A diverse dataset enhances a model's inclusivity, ensuring it responds accurately to varied cultural, demographic, and regional inputs. Without diverse representation, models may reflect narrow views, leading to unintentional biases. Here's how to achieve meaningful diversity:

- **Inclusion of varied demographics**: Incorporate text from various demographic groups to avoid over-representing a single perspective. Include sources in multiple languages or dialects and represent diverse cultural norms to improve global applicability.
- **Balanced data sources**: Draw a balanced dataset from sources such as news, social media, literature, and technical documents. This broadens the model's knowledge base and reduces dependence on any single source.
- **Regional and linguistic variety**: Including datasets from diverse regions and languages expands the model's linguistic and cultural context, enhancing accuracy in multilingual contexts and better supporting translation tasks.

Regular updates

Language is constantly evolving, with new terminologies and shifts in usage patterns emerging frequently. Regular updates to datasets are essential to keep a model relevant and accurate. Here's why updates matter:

- **New vocabulary and trends**: Capture evolving language trends, such as "selfie" or "cryptocurrency," through regular data updates to reflect current terminology.
- **Cultural and social norms**: As societal perspectives shift, language models should adapt accordingly. An LLM trained on outdated data may inadvertently reinforce outdated norms or stereotypes.
- **Model retraining**: Periodically updating the model with fresh data helps maintain alignment with contemporary knowledge and societal standards.

Ethical considerations in data collection

Ethics in data collection are essential to protect user privacy and ensure fair representation. Ethical data practices are fundamental to building trust and reducing biases:

- **Data privacy**: Use anonymized data to protect personal information, especially in datasets containing sensitive or identifiable information.
- **Fair representation**: Ensure the inclusion of marginalized voices to avoid bias that can reinforce societal inequalities.
- **Transparency in data sources**: Disclose data sources used for model training. This transparency fosters user trust and allows for understanding the foundation of the model's knowledge.

Conclusion

Focusing on data quality, diversity, and ethical practices contributes to developing LLMs that are accurate, inclusive, and socially responsible. With a well-prepared dataset, your LLM will perform more effectively while helping bridge gaps in AI fairness and representation.

Cheat Sheet

Package/Method	Description	Code example

NLTK	NLTK is a Python library used in natural language processing (NLP) for tasks such as tokenization and text processing. The code example shows how you can tokenize text using the NLTK word-based tokenizer.	`import nltk`
		`nltk.download("punkt")`
		`from nltk.tokenize import word_tokenize`
		`text = "Unicorns are real. I saw a unicorn yesterday. I couldn't see it today."`
		`token = word_tokenize(text)`
		`print(token)`

spaCy	spaCy is an open-source library used in NLP. It provides tools for tasks such as tokenization and word embeddings. The code example shows how you can tokenize text using spaCy word-based tokenizer.	`import spacy`
		`text = "Unicorns are real. I saw a unicorn yesterday. I couldn't see it today."`
		`nlp = spacy.load("en_core_web_sm")`
		`doc = nlp(text)`
		`token_list = [token.text for token in doc]`
		`print("Tokens:", token_list)`

BertTokenizer	BertTokenizer is a subword-based tokenizer that uses the WordPiece algorithm. The code example shows how you can tokenize text using BertTokenizer.	`from transformers import BertTokenizer`
		`tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")`
		`tokenizer.tokenize("IBM taught me tokenization.")`

XLNetTokenizer	XLNetTokenizer tokenizes text using Unigram and SentencePiece algorithms. The code example shows how you can tokenize text using XLNetTokenizer.	`from transformers import XLNetTokenizer`
		`tokenizer = XLNetTokenizer.from_pretrained("xlnet-base-cased")`
		`tokenizer.tokenize("IBM taught me tokenization.")`

torchtext	The torchtext library is part of the PyTorch ecosystem and provides the tools and functionalities required for NLP. The code example shows how you can use torchtext to generate tokens and convert them to indices.	`from torchtext.vocab import build_vocab_from_iterator`
		`# Defines a dataset`
		`dataset = [`
		`(1,"Introduction to NLP"),`
		`(2,"Basics of PyTorch"),`
		`(1,"NLP Techniques for Text Classification"),`
		`(3,"Named Entity Recognition with PyTorch"),`
		`(3,"Sentiment Analysis using PyTorch"),`
		`(3,"Machine Translation with PyTorch"),`
		`(1,"NLP Named Entity,Sentiment Analysis, Machine Translation"),`
		`(1,"Machine Translation with NLP"),`
		`(1,"Named Entity vs Sentiment Analysis NLP")]`
		`# Applies the tokenizer to the text to get the tokens as a list`
		`from torchtext.data.utils import get_tokenizer`
		`tokenizer = get_tokenizer("basic_english")`
		`tokenizer(dataset[0][1])`
		`# Takes a data iterator as input, processes text from the iterator,`
		`# and yields the tokenized output individually`
		`def yield_tokens(data_iter):`
		`for _,text in data_iter:`
		`yield tokenizer(text)`
		`# Creates an iterator`
		`my_iterator = yield_tokens(dataset)`
		`# Fetches the next set of tokens from the data set`
		`next(my_iterator)`
		`# Converts tokens to indices and sets <unk> as the`
		`# default word if a word is not found in the vocabulary`
		`vocab = build_vocab_from_iterator(yield_tokens(dataset), specials=["<unk>"])`
		`vocab.set_default_index(vocab["<unk>"])`
		`# Gives a dictionary that maps words to their corresponding numerical indices`
		`vocab.get_stoi()`

Package/Method	Description	Code example

vocab	The vocab object is part of the PyTorch torchtext library. It maps tokens to indices. The code example shows how you can apply the vocab object to tokens directly.	`# Takes an iterator as input and extracts the next tokenized sentence.`
		`# Creates a list of token indices using the vocab dictionary for each token.`
		`def get_tokenized_sentence_and_indices(iterator):`
		`  tokenized_sentence = next(iterator)`
		`  token_indices = [vocab[token] for token in tokenized_sentence]`
		`  return tokenized_sentence, token_indices`
		`# Returns the tokenized sentences and the corresponding token indices.`
		`# Repeats the process.`
		`tokenized_sentence, token_indices = \`
		`get_tokenized_sentence_and_indices(my_iterator)`
		`next(my_iterator)`
		`# Prints the tokenized sentence and its corresponding token indices.`
		`print("Tokenized Sentence:", tokenized_sentence)`
		`print("Token Indices:", token_indices)`

Special tokens in PyTorch: <eos> and <bos>	Special tokens are tokens introduced to input sequences to convey specific information or serve a particular purpose during training. The code example shows the use of <bos> and <eos> during tokenization. The <bos> token denotes the beginning of the input sequence, and the <eos> token denotes the end.	`# Appends <bos> at the beginning and <eos> at the end of the tokenized sentences`
		`# using a loop that iterates over the sentences in the input data`
		`tokenizer_en = get_tokenizer('spacy', language='en_core_web_sm')`
		`tokens = []`
		`max_length = 0`
		`for line in lines:`
		`  tokenized_line = tokenizer_en(line)`
		`  tokenized_line = ['<bos>'] + tokenized_line + ['<eos>']`
		`  tokens.append(tokenized_line)`
		`  max_length = max(max_length, len(tokenized_line))`

Special tokens in PyTorch: <pad>	The code example shows the use of <pad> token to ensure all sentences have the same length.	`# Pads the tokenized lines`
		`for i in range(len(tokens)):`
		`  tokens[i] = tokens[i] + ['<pad>'] * (max_length - len(tokens[i]))`

Dataset class in PyTorch	The Dataset class enables accessing and retrieving individual samples from a data set. The code example shows how you can create a custom data set and access samples.	`# Imports the Dataset class and defines a list of sentences`
		`from torch.utils.data import Dataset`
		`sentences = ["If you want to know what a man's like, take a`
		`good look at how he treats his inferiors, not his equals.",`
		`"Fae's a fickle friend, Harry."]`
		`# Downloads and reads data`
		`class CustomDataset(Dataset):`
		`  def __init__(self, sentences):`
		`  self.sentences = sentences`
		`  # Returns the data length`
		`  def __len__(self):`
		`  return len(self.sentences)`
		`  # Returns one item on the index`
		`  def __getitem__(self, idx):`
		`  return self.sentences[idx]`
		`# Creates a dataset object`
		`dataset=CustomDataset(sentences)`
		`# Accesses samples like in a list`
		`E.g., dataset[0]`

DataLoader class in PyTorch	A DataLoader class enables efficient loading and iteration over data sets for training deep learning models. The code example shows how you can use the DataLoader class to generate	`# Creates an iterator object`
		`data_iter = iter(dataloader)`
		`# Calls the next function to return new batches of samples`
		`next(data_iter)`
		`# Creates an instance of the custom data set`

Package/Method	Description	Code example

	batches of sentences for further processing, such as training a neural network model	`from torch.utils.data import DataLoader`
		`custom_dataset = CustomDataset(sentences)`
		`# Specifies a batch size`
		`batch_size = 2`
		`# Creates a data loader`
		`dataloader = DataLoader(custom_dataset, batch_size=batch_size, shuffle=True)`
		`# Prints the sentences in each batch`
		`for batch in dataloader:`
		`  print(batch)`

Custom collate function in PyTorch	The custom collate function is a user-defined function that defines how individual samples are collated or batched together. You can utilize the collate function for tasks such as tokenization, converting tokenized indices, and transforming the result into a tensor. The code example shows how you can use a custom collate function in a data loader.	`# Defines a custom collate function`
		`def collate_fn(batch):`
		`  tensor_batch = []`
		`# Tokenizes each sample in the batch`
		`  for sample in batch:`
		`  tokens = tokenizer(sample)`
		`# Maps tokens to numbers using the vocab`
		`  tensor_batch.append(torch.tensor([vocab[token] for token in tokens]))`
		`# Pads the sequences within the batch to have equal lengths`
		`  padded_batch = pad_sequence(tensor_batch,batch_first=True)`
		`  return padded_batch`
		`# Creates a data loader using the collate function and the custom dataset`
		`dataloader = DataLoader(custom_dataset, batch_size=batch_size, shuffle=True, collate_fn=collate_fn)`

Skills Network

Generative AI

Course Glossary

Welcome! This alphabetized glossary contains many of the terms in this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are essential for you to recognize when working in the industry, participating in user groups, and in other certificate programs.

Estimated reading time: 4 minutes

Term	Definition
Bidirectional and Auto-Regressive Transformers (BART)	Sequence-to-sequence large language model (LLM), which follows an encoder-decoder architecture. It leverages encoding for contextual understanding and decoding to generate text.
Bidirectional Encoder Representations from Transformers (BERT)	Large language model (LLM), which utilizes an encoder-only transformer architecture. It is exceptional at understanding the context of a word within a sentence, which is crucial for nuanced tasks like sentiment analysis.
Data analysis learning with language model for generation and exploration (DALL-E)	AI model developed by OpenAI, known for generating complex and creative images from textual descriptions using deep learning techniques
Data loader	Application component that enables efficient batching and shuffling of data, which is essential for training neural networks. It allows for on-the-fly preprocessing, which optimizes memory usage.
Data set	Collection of data samples and their labels
Diffusion model	Probabilistic generative AI model commonly used for image generation. A diffusion model is trained to generate images by learning to remove noise or reconstruct examples from its training data that have been distorted beyond recognition.
Fine-tuning	Adjusting a pretrained model to improve performance for a specific task or data set. This makes the model generate more accurate and contextually relevant content.
Generative adversarial network (GAN)	Generative AI model that can generate images from random input vectors or seed images. It consists of a generator and a discriminator, which work in a competitive mode.
Generative AI	Deep-learning models that can generate high-quality text, images, and other content based on the data they were trained on. These models are developed and trained to understand patterns and structures within existing data and apply the understanding to produce new and relevant data.
Generative pre-trained transformers (GPT)	Generative AI model based on transformer architecture. It has been pretrained on large amounts of text data and can predict and generate text sequences based on the patterns learned from its training data.
Hugging Face	Platform that offers an open-source library with pretrained models and tools to streamline the process of training and fine-tuning generative AI models.
Iterator	An object that can be looped over. It contains elements that can be iterated through and typically includes two methods: iter and next.
LangChain	Open-source framework that helps in streamlining AI application development using large language models (LLMs)
Large language models (LLMs)	Foundation models that use AI and deep learning with vast data sets to generate text, translate languages, and create various types of content. They are called large language models due to the size of the training data set and the number of parameters.
Natural language processing (NLP)	Subfield of artificial intelligence (AI) that deals with the interaction of computers and humans in human language. It involves creating algorithms and models that will help computers understand and comprehend human language and generate contextually relevant text in human language.
NLTK	Python library used in natural language processing (NLP) for tasks such as tokenization and text processing
Pydantic	Python library that helps streamline data handling. It can be used to parse and validate your data.
PyTorch	Dynamic deep learning framework developed by Facebook's AI Research lab. It is a Python-based library well-known for its ease of use, flexibility, and dynamic computation graphs.
Recurrent neural networks (or RNNs)	Artificial neural networks that use sequential or time series data. RNNs are used to solve data-related problems with a natural order or time-based dependencies.
SentencePiece	Subword-based tokenization algorithm that segments text into manageable parts and assigns unique IDs
spaCy	Open-source library used in natural language processing. It provides tools for tasks such as tokenization and word embeddings.
TensorFlow	Open-source machine learning framework. It provides a set of tools and libraries to facilitate the development and deployment of machine learning models.
Text-to-Text Transfer Transformer (T5)	Transformer-based large language model, which uses a text-to-text framework. It leverages encoding for contextual understanding and decoding to generate text.
Tokenization	Breaking text into smaller pieces or tokens. The tokens help a generative AI model understand the text better.

Term	Definition
Tokenizer	Program that breaks down text into individual tokens
Transformers	Deep learning models that can translate text and speech in near-real-time. They take data, such as words or numbers, and pass it through different layers, with information flowing in one direction.
Unigram	Subword-based tokenization algorithm that breaks text into smaller pieces. It begins with a large list of possibilities and gradually narrows down based on how frequently they appear in the text.
variational autoencoders (VAEs)	Generative AI model that operates on an encoder-decoder framework. The encoder network first compresses input data into a simplified, abstract space that captures essential characteristics. The decoder network then uses this condensed information to recreate the original data.
WaveNet	Generative AI model designed for generating audio content. It can be used for tasks such as speech synthesis.
WordPiece	Subword-based tokenization algorithm that evaluates the benefits and drawbacks of splitting and merging two symbols to ensure its decisions are valuable.

Skills Network

Course Conclusion

Congratulations! You've successfully completed the course and are now equipped with valuable insights into generative AI architecture and data preparation for large language models (LLMs). At this point, you know that:

- Generative AI refers to deep-learning models that can generate content, such as text, images, audio, 3D objects, and music, based on the training data.
- These models understand the relationship between words and phrases and generate contextually relevant text. An example of a generative AI model for text generation is a generative pre-trained transformer (GPT).
- These models can generate images from text input and seed images. Examples of generative AI models for image generation are data analysis learning with language model for generation and exploration (DALL-E) and generative adversarial networks (GANs).
- Generative AI models can generate natural-sounding speech. An example of this type of generative AI model is WaveNet.
- Generative AI architectures and models include recurrent neural networks (RNNs), transformers, GANs, variational autoencoders (VAEs), and diffusion models.
    - RNNs use sequential or time series data and a loop-based design for training.
    - Transformers utilize the self-attention mechanism to focus on the most important parts of the information.
    - GANs consist of a generator and a discriminator, which work in a competitive mode.
    - VAEs operate on an encoder-decoder framework and create samples based on similar characteristics.
    - Diffusion models generate creative images by learning to remove noise and reconstruct distorted examples, relying on statistical properties.
- Generative AI started with rule-based systems that use predefined linguistic rules, followed by machine-learning approaches focusing on statistical methods. It later moved to deep learning, which uses neural networks trained on extensive data sets. Transformers represent the latest in this evolution.
- The evolution of generative AI has led to advancements in machine translation, chatbot conversations, sentiment analysis, and text summarization.
- Large language models (LLMs) are foundation models that use AI and deep learning with vast data sets. LLMs have training data sets that run into petabytes and contain billions of parameters. Examples of LLMs are GPT, Bidirectional Encoder Representations from Transformers (BERT), Bidirectional and Auto-Regressive Transformers (BART), and Text-to-Text Transfer Transformer (T5).
- Some libraries and tools you can use to implement generative AI for NLP are PyTorch, TensorFlow, Hugging Face, LangChain, and Pydantic.

- Tokenization and data loading are part of the data preparation activities for natural language processing (NLP).
- Tokenization breaks a sentence into smaller pieces or tokens. These tokens can be words, characters, or subwords, making complex text understandable to computers. Examples of tokenizers are natural language toolkit (NLTK) and spaCy.
- Word-based tokenization preserves the semantic meaning, though it increases the model's overall vocabulary.
- Character-based tokenization has smaller vocabularies but may not convey the same information as entire words.
- Subword-based tokenization allows frequently used words to stay unsplit while breaking down infrequent words.
    - Using the WordPiece, Unigram, and SentencePiece algorithms, you can implement subword-based tokenization.
- You can add special tokens such as <bos> at the beginning and <eos> at the end of a tokenized sentence.
- A data set in PyTorch is an object that represents a collection of data samples. Each data sample typically consists of one or more input features and their corresponding target labels.
- A data loader helps you prepare and load data to train generative AI models. Using data loaders, you can output data in batches instead of one sample at a time.
- Data loaders seamlessly integrate with the PyTorch training pipeline and simplify data augmentation and preprocessing.