Udacity - Generative AI

[Blog ideas 2](#_Toc184619806)

[Course 2 2](#_Toc184619807)

[Technical Terms Explained: 2](#_Toc184619808)

[1.8 Training Generative AI Models 4](#_Toc184619809)

[1.9 Generation Algorithms 5](#_Toc184619810)

[1.12 More generative AI Architectures 6](#_Toc184619811)

[2.2 What is Perceptron 7](#_Toc184619812)

[2.3 The Multi-Layer Perceptron 7](#_Toc184619813)

[2.4 Training Deep Learning Neural Networks 8](#_Toc184619814)

[2.10 Pytorch Loss Function 8](#_Toc184619815)

[2.11 PyTorch Datasets and Data Loaders 9](#_Toc184619816)

[Code Examples 9](#_Toc184619817)

[Datasets 9](#_Toc184619818)

[Data Loaders 10](#_Toc184619819)

[2.17 Hugging Face Trainers 11](#_Toc184619820)

[Resources Web 11](#_Toc184619821)

[Resources 11](#_Toc184619822)

[3.7 The GLUE Benchmark 12](#_Toc184619823)

[GLUE Tasks / Benchmarks 12](#_Toc184619824)

[4.3 Retrieval-Augmented Generation (RAG) 14](#_Toc184619825)

[4.8 In-Context Learning 14](#_Toc184619826)

[4.12 Using Probing to train a Classifier 14](#_Toc184619827)

[4.16 Parameter-Efficient Fine-tuning 15](#_Toc184619828)

[Course 3 15](#_Toc184619829)

[1.13 Chain of Thought Prompting 15](#_Toc184619830)

[1.14 Encoder vs. Decoder Models 16](#_Toc184619831)

[1.8 LLM Inference and Decoding Parameters 16](#_Toc184619832)

[Temperature 17](#_Toc184619833)

[Top P 18](#_Toc184619834)

[Frequency penalty + Presence penalty 18](#_Toc184619835)

[1.8 NLP Applications and Tasks 19](#_Toc184619836)

# Blog ideas

* Course 3
  + Vicious and Virtuous Cycles

# Course 2

pre-training dataset, focusing on four key sources:

* CommonCrawl,
* Github,
* Wikipedia, and
* the Gutenberg project.<https://www.gutenberg.org/> a library of over 70,000 free eBooks

## Technical Terms Explained:

**Selection Bias:** When the data used to train an AI model does not accurately represent the whole population or situation by virtue of the selection process, e.g. those choosing the data will tend to choose dataset their are aware of

**Historical Bias:** Prejudices and societal inequalities of the past that are reflected in the data, influencing the AI in a way that perpetuates these outdated beliefs.

**Confirmation Bias:** The tendency to favor information that confirms pre-existing beliefs, which can affect what data is selected for AI training.

**Discriminatory Outcomes:** Unfair results produced by AI that disadvantage certain groups, often due to biases in the training data or malicious actors.

**Echo Chambers:** Situations where biased AI reinforces and amplifies existing biases, leading to a narrow and distorted sphere of information.

**Bias Detection and Correction:** Processes and algorithms designed to identify and remove biases from data before it's used to train AI models.

**Transparency and Accountability:** Openness about how AI models are trained and the nature of their data, ensuring that developers are answerable for their AI's performance and impact.

**Semantic-embedding:** A representation of text in a high-dimensional space where distances between points correspond to semantic similarity. Phrases with similar meanings are closer together. = encode context into a vector representation.

**Cosine similarity:** A metric used to measure how similar two vectors are, typically used in the context of semantic embeddings to assess similarity of meanings.

**Prompt:** In AI, a prompt is an input given to the model to generate a specific response or output.

**Prompt Tuning:** This is a method to improve AI models by optimizing prompts so that the model produces better results for specific tasks.

**Hard Prompt:** A manually created template used to guide an AI model's predictions. It requires human ingenuity to craft effective prompts.

**Soft Prompt:** A series of tokens or embeddings optimized through deep learning to help guide model predictions, without necessarily making sense to humans.

**One-shot prompting:** Giving an AI model a single example to learn from before it attempts a similar task.

**Few-shot prompting:** Providing an AI model with a small set of examples, such as five or fewer, from which it can learn to generalize and perform tasks.

**Zero-shot prompting:** This refers to the capability of an AI model to correctly respond to a prompt or question it hasn't explicitly been trained to answer, relying solely on its prior knowledge and training.

**Chain-of-Thought Prompting:** A method of guiding a language model through a step-by-step reasoning process to help it solve complex tasks by explicitly detailing the logic needed to reach a conclusion.

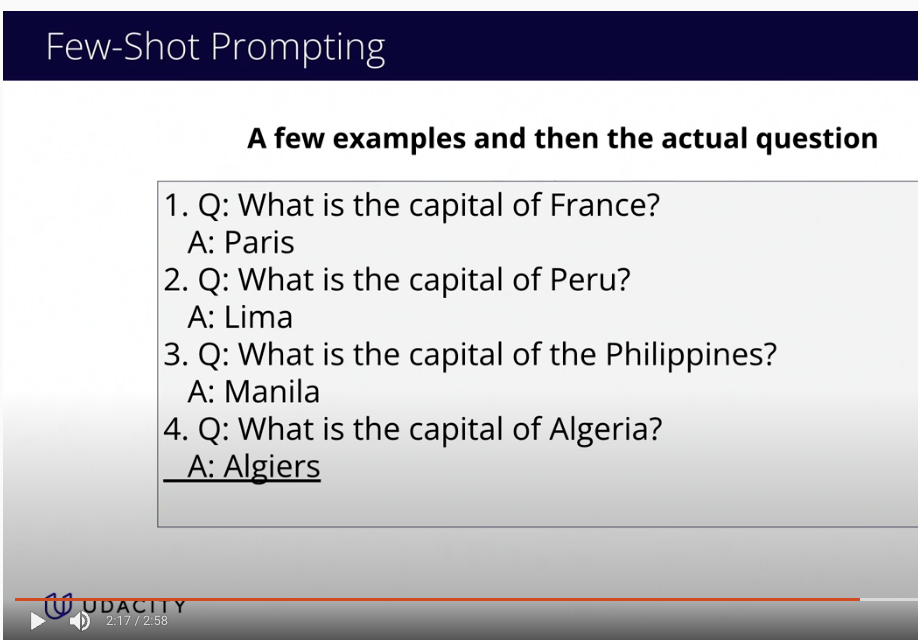
**Retrieval-Augmented Generation (RAG):** is a powerful approach for keeping Generative AI models informed with the most recent data, particularly when dealing with domain-specific questions.

**Parameter-efficient fine-tuning:** A method of updating a predefined subset of a model's parameters to tailor it to specific tasks, without the need to modify the entire model, thus saving computational resources.

**Frozen Parameters:** In the context of machine learning, this refers to model parameters that are not changed or updated during the process of training or fine-tuning.

**Low-Rank Adaptation (LoRA):** A technique where a large matrix is approximated using two smaller matrices, greatly reducing the number of parameters that need to be trained during fine-tuning.

**Adapters:** Additional model components inserted at various layers; only the parameters of these adapters are trained, not of the entire model.



**Probing:** This is a method of examining what information is contained in different parts of a machine learning model.

**Linear Probing:** A simple form of probing that involves attaching a linear classifier to a pre-trained model to adapt it to a new task without modifying the original model.

**Classification Head:** It is the part of a neural network that is tailored to classify input data into defined categories.

### 1.8 Training Generative AI Models

Technical Terms Explained:

**Large Language Models (LLMs):** These are AI models specifically designed to understand and generate human language by being trained on a vast amount of text data.

**Variational Autoencoders (VAEs)**: A type of AI model that can be used to create new images. It has two main parts: the encoder reduces data to a simpler form, and the decoder expands it back to generate new content.

**Latent Space:** A compressed representation of data that the autoencoder creates in a simpler, smaller form, which captures the most important features needed to reconstruct or generate new data.

**Parameters:** Parameters are the variables that the model learns during training. They are internal to the model and are adjusted through the learning process. In the context of neural networks, parameters typically include weights and biases.

**Weights:** Weights are coefficients for the input data. They are used in calculations to determine the importance or influence of input variables on the model's output. In a neural network, each connection between neurons has an associated weight.

**Biases (not mentioned in the video):** Biases are additional constants attached to neurons and are added to the weighted input before the activation function is applied. Biases ensure that even when all the inputs are zero, there can still be a non-zero output.

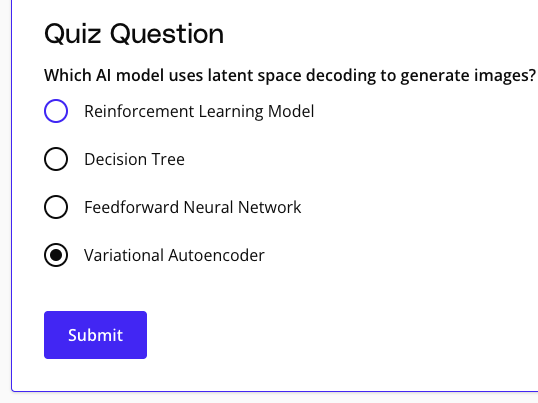
**Hyperparameters:** Hyperparameters, unlike parameters, are not learned from the data. They are more like settings or configurations for the learning process. They are set prior to the training process and remain constant during training. They are external to the model and are used to control the learning process.

### 1.9 Generation Algorithms

**Autoregressive text generation:** Autoregressive text generation is like a game where the computer guesses the next word in a sentence based on the words that came before it. It keeps doing this to make full sentences.

**Latent space decoding:** Imagine if you had a map of all the possible images you could create, with each point on the map being a different image. Latent space decoding is like picking a point on that map and bringing the image at that point to life.

**Diffusion models:** Diffusion models start with a picture that's full of random dots like TV static, and then they slowly clean it up, adding bits of the actual picture until it looks just like a real photo or painting.



### 1.12 More generative AI Architectures

There are various Generative AI Architectures for creating new content by mimicking patterns. These architectures, like GANs, RNNs, and Transformers, excel at producing novel images, text, and sounds by understanding and repurposing what they've learned. They enable us to push the boundaries of creativity and innovation, opening up a world of new possibilities.

Technical Terms Explained:

**Generative Adversarial Networks (GANs):** A system where two neural networks, one to generate data and one to judge it, work against each other. This competition helps improve the quality of the generated results.

**Recurrent Neural Networks (RNNs):** A network that's really good at handling sequences, like sentences or melodies, because it processes one piece at a time and remembers what it saw before.

**Transformer-based models:** A more advanced type that looks at whole sequences at once, not one piece at a time, making it faster and smarter at tasks like writing sentences or translating languages.

**Sequential Data:** Data that is connected in a specific order, like words in a sentence or steps in a dance routine.

### 2.2 What is Perceptron

A perceptron is an essential component in the world of AI, acting as a **binary classifier** capable of deciding whether data, like an image, belongs to one class or another. It works by adjusting its weighted inputs—think of these like dials fine-tuning a radio signal—until it becomes better at predicting the right class for the data. This process is known as learning, and it shows us that even complex tasks start with small, simple steps.

Technical Terms Explained:

**Perceptron:** A basic computational model in machine learning that makes decisions by weighing input data. It's like a mini-decision maker that labels data as one thing or another.

**Binary Classifier:** A type of system that categorizes data into one of two groups. Picture a light switch that can be flipped to either on or off.

**Vector of Numbers:** A sequence of numbers arranged in order, which together represent one piece of data.

**Activation Function:** (Like ReLU) A mathematical equation that decides whether the perceptron's calculated sum from the inputs is enough to trigger a positive or negative output.

### 2.3 The Multi-Layer Perceptron

The multi-layer perceptron is a powerful tool in the world of machine learning, capable of making smart decisions by mimicking the way our brain's neurons work. This amazing system can learn from its experiences, growing smarter over time as it processes information through layers, and eventually, it can predict answers with astonishing accuracy!

Technical Terms Explained:

**Multi-Layer Perceptron (MLP):** A type of artificial neural network that has multiple layers of nodes, each layer learning to recognize increasingly complex features of the input data.

**Input Layer:** The first layer in an MLP where the raw data is initially received.

**Output Layer:** The last layer in an MLP that produces the final result or prediction of the network.

**Hidden Layers:** Layers between the input and output that perform complex data transformations.

### 2.4 Training Deep Learning Neural Networks

We learned that training deep neural networks involves guided adjustments to improve their performance on tasks like image recognition. By gradually refining the network's parameters and learning from mistakes, these networks become smarter and more skilled at predicting outcomes. The marvel of this technology is its ability to turn raw data into meaningful insights.

Technical Terms Explained:

**Labeled Dataset:** This is a collection of data where each piece of information comes with a correct answer or label. It's like a quiz with the questions and answers already provided.

**Gradient Descent:** This method helps find the best settings for a neural network by slowly tweaking them to reduce errors, similar to finding the lowest point in a valley.

**Cost Function:** Imagine it as a score that tells you how wrong your network's predictions are. The goal is to make this score as low as possible.

**Learning Rate:** This hyperparameter specifies how big the steps are when adjusting the neural network's settings during training. Too big, and you might skip over the best setting; too small, and it'll take a very long time to get there.

**Backpropagation:** Short for backward propagation of errors. This is like a feedback system that tells each part of the neural network how much it contributed to any mistakes, so it can learn and do better next time.

### 2.10 Pytorch Loss Function

**what does a lower cross-entropy loss value indicate**

Based on the search results, a lower cross-entropy loss value indicates better performance of a classification model. This is because cross-entropy loss measures the difference between the predicted probabilities and the true labels, and a lower value indicates that the model is accurately assigning high probabilities to the correct class and low probabilities to incorrect classes.

In mathematical terms, a lower cross-entropy loss value means that the model is minimizing the error between the predicted probabilities and the true labels, which is the goal of training a classification model. This is reflected in the formula for binary cross-entropy loss:

l = - (y\_true \* log(p) + (1 - y\_true) \* log(1 - p))

where y\_true is the true label (0 or 1), p is the predicted probability, and log is the natural logarithm.

A lower loss value indicates that the model is producing predicted probabilities that are close to the true class probabilities, and the logarithmic component of the BCE function is penalizing wrong predictions more heavily than correct ones. This drives the model to improve its predictions during training.

In summary, a lower cross-entropy loss value is an indication of a well-performing classification model, and it can be used as a metric to evaluate the model’s accuracy during training and testing.

### 2.11 PyTorch Datasets and Data Loaders

PyTorch makes accessing data for your model a breeze! These tools ensure that the flow of information to our AI is just right, making its learning experience effective and fun.

Technical Terms:

**PyTorch Dataset class**: This is like a recipe that tells your computer how to get the data it needs to learn from, including where to find it and how to parse it, if necessary.

**PyTorch Data Loader:** Think of this as a delivery truck that brings the data to your AI in small, manageable loads called batches; this makes it easier for the AI to process and learn from the data.

**Batches**: Batches are small, evenly divided parts of data that the AI looks at and learns from each step of the way.

**Shuffle**: It means mixing up the data so that it's not in the same order every time, which helps the AI learn better.

## Code Examples

### Datasets

**from** torch.utils.data **import** Dataset

*# Create a toy dataset*

**class** **NumberProductDataset**(Dataset):

**def** **\_\_init\_\_**(self, data\_range=(1, 10)):

self.numbers = list(range(data\_range[0], data\_range[1]))

**def** **\_\_getitem\_\_**(self, index):

number1 = self.numbers[index]

number2 = self.numbers[index] + 1

**return** (number1, number2), number1 \* number2

**def** **\_\_len\_\_**(self):

**return** len(self.numbers)

*# Instantiate the dataset*

dataset = NumberProductDataset(

data\_range=(0, 11)

)

*# Access a data sample*

data\_sample = dataset[3]

**print**(data\_sample)

*# ((3, 4), 12)*

### Data Loaders

**from** torch.utils.data **import** DataLoader

*# Instantiate the dataset*

dataset = NumberProductDataset(data\_range=(0, 5))

*# Create a DataLoader instance*

dataloader = DataLoader(dataset, batch\_size=3, shuffle=True)

*# Iterating over batches*

**for** (num\_pairs, products) **in** dataloader:

**print**(num\_pairs, products)

*# [tensor([4, 3, 1]), tensor([5, 4, 2])] tensor([20, 12, 2])*

*# [tensor([2, 0]), tensor([3, 1])] tensor([6, 0])*

### 2.17 Hugging Face Trainers

## Resources Web

<https://ourworldindata.org/> Public Data

## Resources

**[Hugging Face Trainers documentation index](https://huggingface.co/docs/transformers/main_classes/trainer)**

**[(opens in a new tab)](https://huggingface.co/docs/transformers/main_classes/trainer)**

**[Hugging Face DistilBertForSequenceClassification documentation](https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertForSequenceClassification)**

**[(opens in a new tab)](https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertForSequenceClassification)**

**[Hugging Face DistilBertTokenizer documentation](https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertTokenizer)**

**[(opens in a new tab)](https://huggingface.co/docs/transformers/model_doc/distilbert#transformers.DistilBertTokenizer)**

**[distilbert-base-uncased Model documentation on Hugging Face](https://huggingface.co/distilbert-base-uncased)**

**[(opens in a new tab)](https://huggingface.co/distilbert-base-uncased)**

**[Hugging Face transformers.TrainingArguments documentation](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments)**

**[(opens in a new tab)](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments)**

[**Hugging Face transformers.Trainer documentation**](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)

**from** transformers **import** (DistilBertForSequenceClassification,

DistilBertTokenizer,

TrainingArguments,

Trainer

)

**from** datasets **import** load\_dataset

model = DistilBertForSequenceClassification.from\_pretrained(

"distilbert-base-uncased", num\_labels=2

)

tokenizer = DistilBertTokenizer.from\_pretrained("distilbert-base-uncased")

**def** **tokenize\_function**(examples):

**return** tokenizer(examples["text"], padding="max\_length", truncation=True)

dataset = load\_dataset("imdb")

tokenized\_datasets = dataset.map(tokenize\_function, batched=True)

training\_args = TrainingArguments(

per\_device\_train\_batch\_size=64,

output\_dir="./results",

learning\_rate=2e-5,

num\_train\_epochs=3,

)

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=tokenized\_datasets["train"],

eval\_dataset=tokenized\_datasets["test"],

)

trainer.train()

### 3.7 The GLUE Benchmark

The GLUE benchmarks serve as an essential tool to assess an AI's grasp of human language, covering diverse tasks, from grammar checking to complex sentence relationship analysis. By putting AI models through these varied linguistic challenges, we can gauge their readiness for real-world tasks and uncover any potential weaknesses.

**Technical Terms Explained:**

**Semantic Equivalence**: When different phrases or sentences convey the same meaning or idea.

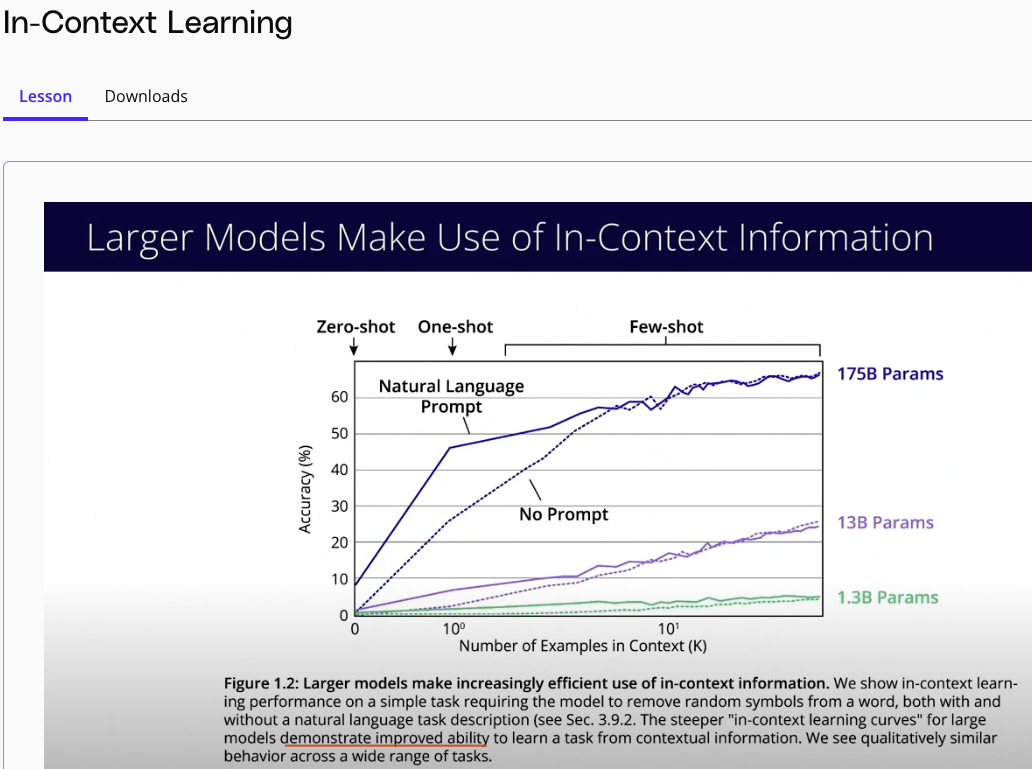
**Textual Entailment:** The relationship between text fragments where one fragment follows logically from the other.

#### GLUE Tasks / Benchmarks

|  |  |  |
| --- | --- | --- |
| **Short Name** | **Full Name** | **Description** |
| CoLA | Corpus of Linguistic Acceptability | Measures the ability to determine if an English sentence is linguistically acceptable. |
| SST-2 | Stanford Sentiment Treebank | Consists of sentences from movie reviews and human annotations about their sentiment. |
| MRPC | Microsoft Research Paraphrase Corpus | Focuses on identifying whether two sentences are paraphrases of each other. |
| STS-B | Semantic Textual Similarity Benchmark | Involves determining how similar two sentences are in terms of semantic content. |
| QQP | Quora Question Pairs | Aims to identify whether two questions asked on Quora are semantically equivalent. |
| MNLI | Multi-Genre Natural Language Inference | Consists of sentence pairs labeled for textual entailment across multiple genres of text. |
| QNLI | Question Natural Language Inference | Involves determining whether the content of a paragraph contains the answer to a question. |
| RTE | Recognizing Textual Entailment | Requires understanding whether one sentence entails another. |
| WNLI | Winograd Natural Language Inference | Tests a system's reading comprehension by having it determine the correct referent of a pronoun in a sentence, where understanding depends on contextual information provided by specific words or phrases. |

### 4.3 Retrieval-Augmented Generation (RAG)

### 4.8 In-Context Learning



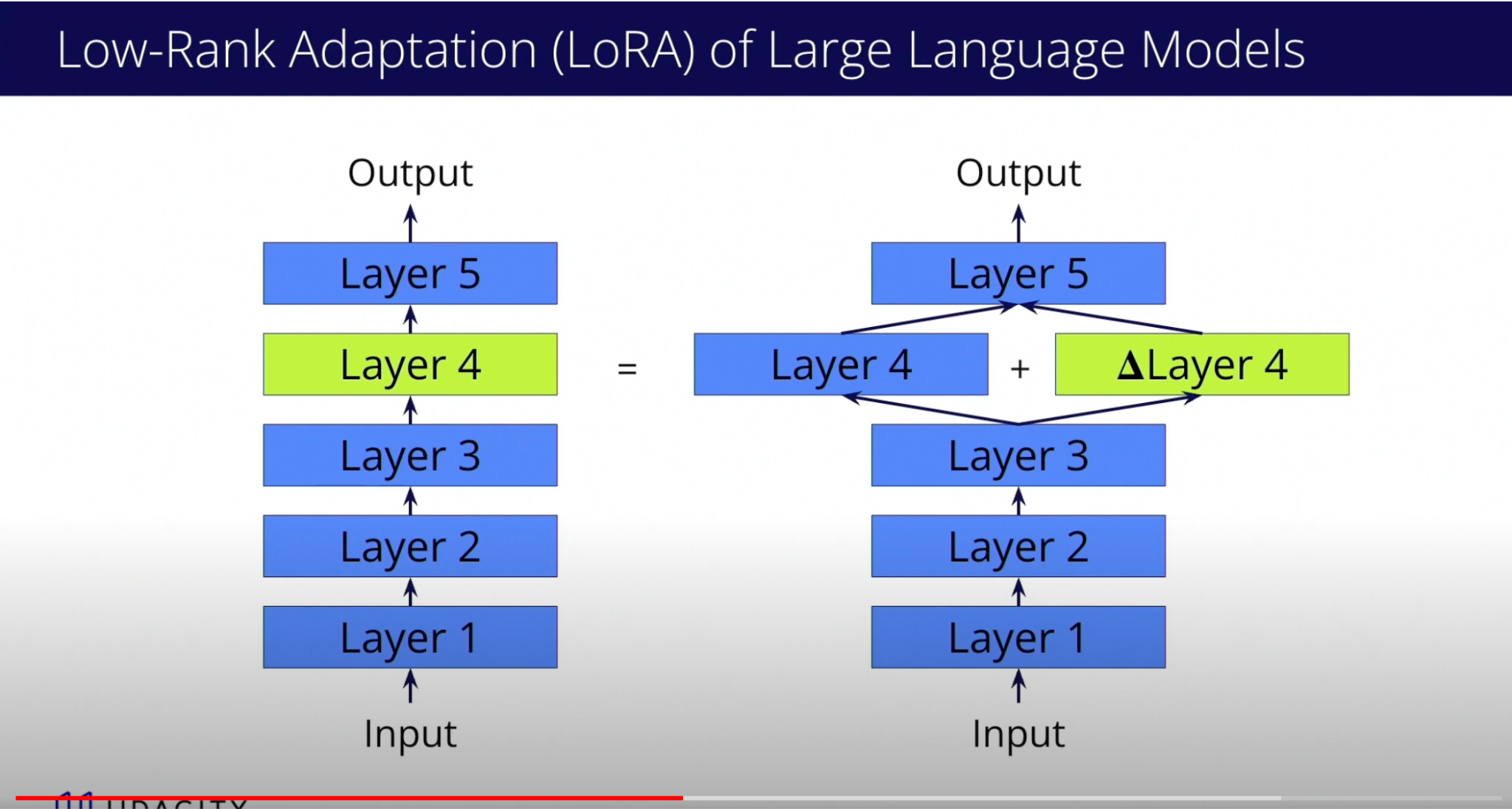
### 4.12 Using Probing to train a Classifier

**Probing:** This is a method of examining what information is contained in different parts of a machine learning model.

**Linear Probing:** A simple form of probing that involves attaching a linear classifier to a pre-trained model to adapt it to a new task without modifying the original model.

**Classification Head:** It is the part of a neural network that is tailored to classify input data into defined categories.

### 4.16 Parameter-Efficient Fine-tuning



**Parameter-efficient fine-tuning:** A method of updating a predefined subset of a model's parameters to tailor it to specific tasks, without the need to modify the entire model, thus saving computational resources. (VS fine-tuning to train the pretrained model from scratch, not efficient since it requires same amount of CPU/GPU + Labeled data used to train the original model)

**Frozen Parameters:** In the context of machine learning, this refers to model parameters that are not changed or updated during the process of training or fine-tuning.

**Low-Rank Adaptation (LoRA):** A technique where a large matrix is approximated using two smaller matrices, greatly reducing the number of parameters that need to be trained during fine-tuning.

**Adapters:** Additional model components inserted at various layers; only the parameters of these adapters are trained, not of the entire model.

# Course 3

## 1.13 Chain of Thought Prompting

As we explore in this demo, “Chain of Thought” (COT), prompting effectively amounts to asking the LLM to provide step-by-step reasoning for its answer **before** providing the final answer to the question.

COT was first introduced in the paper [**"Large Language Models are Zero-Shot Reasoners"(opens in a new tab)**](https://arxiv.org/abs/2205.11916), and later simplified in a subsequent paper, [**"Chain-of-Thought Prompting Elicits Reasoning in Large Language Models"(opens in a new tab)**](https://arxiv.org/abs/2201.11903), and is still in use today. A variation of the COT method was used in Google's [**Gemini technical report(opens in a new tab)**](https://arxiv.org/abs/2312.11805).

## 1.14 Encoder vs. Decoder Models

**A Transformer is Born**

In mid-2017, Google published [**"Attention Is All You Need"(opens in a new tab)**](https://arxiv.org/abs/1706.03762) and introduced the Transformer model to the world. There are many wonderful works explaining the Transformer model, in particular the [**Illustrated Transformer(opens in a new tab)**](https://jalammar.github.io/illustrated-transformer/) (a wonderful introduction) and the [**Annotated Transformer(opens in a new tab)**](https://nlp.seas.harvard.edu/annotated-transformer/) (with a line-by-line implementation in Python).

**Encoder-only Models**

The first LLM to gain broad adoption was [**BERT(opens in a new tab)**](https://arxiv.org/abs/1810.04805) (Bidirectional Encoder Representations from Transformers), an encoder-only model. Encoder-only models are most commonly used as base models for subsequent fine-tuning with a distinct objective, e.g. for the inference-time task of binary classification of movie reviews.

**Decoder-only Models**

However, before BERT was released, the first [**GPT(opens in a new tab)**](https://openai.com/research/language-unsupervised) (Generative Pre-Trained Transformer) model, a decoder-only model, was released by OpenAI. Decoder-only models are most commonly used for the inference-time task of text generation. In distinction to encoder-only models, the Transformer's pre-training objective of next token prediction is very similar to the decoder-only model's inference-time task of text generation.

## 1.8 LLM Inference and Decoding Parameters

A screenshot of a computer

Description automatically generated

### Temperature

Temperature=0 => Selecting Greedy decoding

A screenshot of a computer

Description automatically generated

Higher Temperature flatten the distribution probability, Thus using token with lower probability => increase creativity of the model

### Top P

A screen shot of a computer

Description automatically generated

Top P shop the bottom of the probability. Greedy Token is impacted by the Top P value selected

### Frequency penalty + Presence penalty

A screenshot of a computer

Description automatically generated

These two reduce the likelihood of sampling specific tokens that have already appeared in the generated sequence => reduce the likelihood of repeated sequences **which impact particularly small models**

## 1.8 NLP Applications and Tasks

Extractive vs. Abstractive Summarization

* **Extractive**: Directly quotes main points from the source
* **Abstractive**: Summarizes with novel words and phrases

## 2.18 Sampling Methods for tokens

* **Temperature**: adjusts the randomness in choosing the next token
* **Top-**k **sampling**: samples f10rom only the k most likely tokens
* **Nucleus or top-**p **sampling**: uses a dynamic cutoff for sampling the most likely tokens (cumulative probability is under *p*)
* **Beam search**: considers the likelihood of strings of multiple tokens instead of just a single next token