Part II

Deep Generative Models for Trading and Asset Management

Chapter 4

Understanding Generative AI

Generative models represent a fascinating frontier in the advance of artificial intelligence. These models have the ability to understand and generate complex, high-dimensional data. This capability not only opens the door to numerous practical applications but also offers deeper insight into the underlying structure of data. This chapter explores the essence of generative models, their applications, and the differences with discriminative models that many quant finance professionals have been familiar with. We will also provide a taxonomy to navigate the landscape of generative models (Figure 4.1).

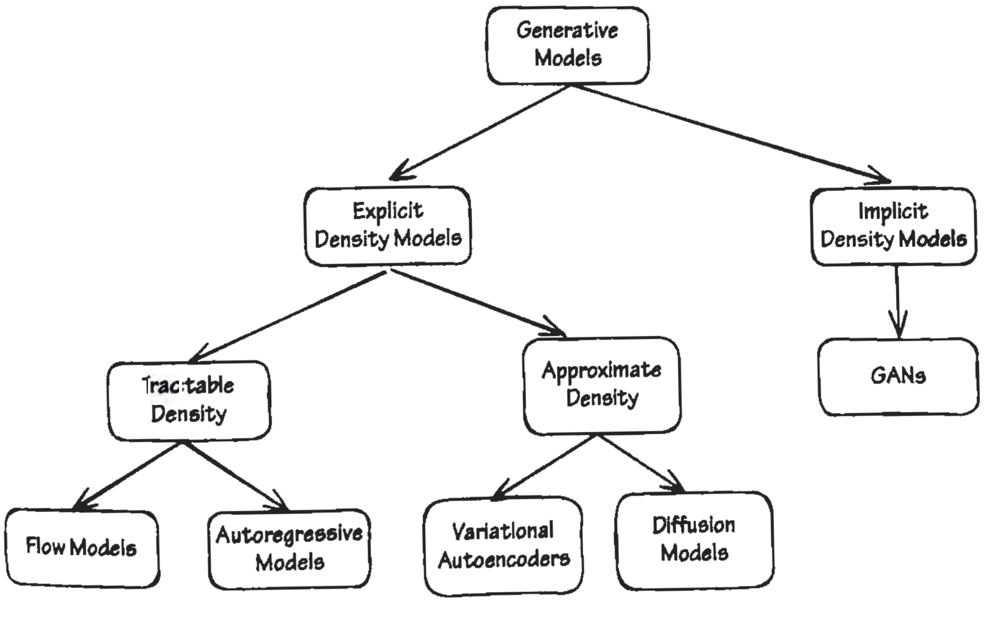

Figure 4.1 Model taxonomy.

4.1 Why Generative Models

Generative models are pivotal for several reasons. Firstly, they enable machines to generate new data instances that resemble the data they have encountered during training, effectively learning the distribution of the dataset. This ability goes beyond merely replicating observed data; it's about understanding the underlying structure of the data to generate new instances. Secondly, generative models facilitate unsupervised learning tasks, where the machine learns common patterns in the data without needing labeled data. This aspect is crucial for exploiting the vast amounts of unlabeled data available today and avoiding the costly and time-consuming process of manual data labeling. Lastly, they aid in understanding data distribution, enabling applications for anomaly detection, data compression, and more.

We want to disabuse the reader of the notion that Generative AI is only useful for understanding unstructured data such as text or images, or that it is only useful for generating synthetic data for simulation purposes. As the following section illustrates, it can be crucial for the usual supervised learning tasks using discriminative models that most quant finance professionals are primarily interested in.

4.2 Difference with Discriminative Models

At the heart of machine learning lies the quest to model and make predictions from data. Discriminative and generative models represent two approaches to this quest, differentiated by the type of probability distribution they aim to learn from the data.

Discriminative models focus on learning the probability of a label or target y given an input x, denoted as $p(y|x)$. These models excel in classification or regression tasks, where the goal is to correctly predict the target. They are closely associated with supervised learning tasks, where the models are trained on a labeled dataset, learning to map inputs to targets. Despite their powerful ability to determine the most probable target for a given input, a critical limitation of discriminative models is their inability to assess the likelihood $p(x)$ of the input data itself. This design limitation makes them susceptible to adversarial attacks or misinterpretations of novel or outlier data.

Generative models, on the other hand, aim to model the distribution of data itself, learning either the joint probability distribution $p(y,x)$ or the unconditional probability distribution $p(x)$. Unconditional generative models attempt to learn the probability distribution of the input $p(x)$, enabling them to generate new data instances that resemble the training data. As an example, let's suppose we want to model the SPY ETF return data and decide to approximate its distribution using a Gaussian Mixture Model (GMM), one of the simplest generative models. This model not only allows us to generate new data instances but, as we will see in Chapter 6, it also captures some underlying structure of the data that can be used for applications like regime detection. In general, generative models provide a deeper understanding of the data's structure, enabling applications like data generation and outlier detection.

Conditional generative models represent a hybrid approach, learning to generate data x conditioned on specific values of y, such as class labels, denoted as $p(x \mid y)$. These models combine the generative capability to produce new instances with the discriminative power to condition these instances on particular attributes or classes, making them more robust to adversarial attacks or misinterpretations of novel or outlier data. As an example, commonly encountered, consider models like ChatGPT, which can generate new data like text, images, or source code conditioned on input prompts provided by the user.

To illustrate the difference between discriminative and generative models, let's consider the classic dog vs. cat classifier example. A discriminative model learns $p(y \mid x)$—the probability of a label y (dog or cat) given an image x. At inference time, it takes an input image and predicts the probability that it belongs to each class. A conditional generative model, on the other hand, learns $p(x \mid y)$, modeling the probability of an image x given a specific class y.

This model can also be used as a classifier, known as a generative classifier, by comparing the probability of an image x under $p(x \mid y = \text{dog})$ versus $p(x \mid y = \text{cat})$. However, unlike discriminative models, it can also generate new images conditioned on a label. For example, when conditioned on *dog* ($y = \text{dog}$), the model can generate an image of a dog by sampling from $p(x \mid y = \text{dog})$.

In this sense, a discriminative model maps the input image x to the output class label y, while a generative model works in reverse, generating x from a given y. Additionally, in generative modeling, y is not restricted to class labels—it can also represent attributes or features.

The distinction between generative and discriminative models, which may initially seem somewhat disconnected, is rooted in Bayes' rule of probability. This relationship is illustrated by the equation

$$p(x \mid y) = \frac{p(x)p(y \mid x)}{p(Y)} \tag{4.1}$$

where $p(x)$ represents the unconditional generative model, $p(y \mid x)$ corresponds to the discriminative model, $p(Y)$ is the marginal probability over the labels, and $p(x \mid y)$ is a conditional model of x given y. By combining the strengths of both unconditional generative models and discriminative models, we can construct conditional generative models, which can be used not only for conditional data generation but also for supervised learning tasks, such as classification.

4.3 How Can We Use Them?

These models are designed with some key core abilities or objectives in mind, which open a world of possibilities for innovation, creativity, and efficiency across various domains. Following is a summary of these characteristics, which we will explore further in next chapters.

4.3.1 Probability Density Estimation

These models can approximate *high-dimensional*, multimodal distributions of data. Once we have access to this versatile object, we can develop and solve a wide range of applications. Notable examples include multivariate time series forecasting, as illustrated in Chapter 3, Value at Risk (VaR) applications for risk management; anomaly detection, and classification tasks.

Finance professionals are already familiar with high dimensional probability distributions of returns, but they are typically modeled as the multivariate versions of simple parametric models such as the multivariate Gaussian or t-distribution. More advanced models such as copulas attempt to capture the nonlinear nature of the co-dependency between variables, but at the heart of them are still simple parametric models. Generative AI, on the other hand, allow much more complex, nonlinear, distributions that cannot be expressed in simple parametric forms.

4.3.2 Generating New Data

At the heart of generative models' capabilities is their power to create. They can produce highly complex, high-dimensional objects, like realistic images, videos, text, computer code, multivariate time series, etc. This makes them invaluable across applications like entertainment, marketing, productivity tools, and risk management. Their ability to generate data is often indistinguishable from humans. This ability goes beyond simply crafting content that is often indistinguishable from human-created work, as seen in widely available examples in the media at the time of writing.

New generated data can be used for the following:

- Improving data quality and availability: Generated data can address challenges related to data scarcity and quality. These models can augment existing datasets, especially in fields where data collection is expensive or impractical, by creating new data instances and enhancing datasets for downstream learning tasks.
- Facilitating discovery and decision-making: one of the most promising applications of generative models lies in their ability to simulate and predict complex, high-dimensional structures and phenomena. In the pharmaceutical industry, for example, these models can predict molecular structures, potentially leading to groundbreaking drugs. Additionally, they can simulate environments for AI training, such as for reinforcement learning applications with are known to be very data hungry, therefore reducing the need for costly and time-consuming real-world data collection, accelerating research and development across fields as diverse as autonomous driving and urban planning.

4.3.3 Learning New Data Representations

The goal of finding new data representations, or representation learning, is to discover an effective way to represent data. Ideally, this new representation should possess desirable properties, such as lower dimensionality compared to the data space or "independent" axes of variation. This new representation serves as a compressed form of the data that can be used for solving downstream tasks such as prediction, anomaly detection, or data generation (see Torralba et al., [2024]).

Many finance professionals are already familiar with representation learning disguised in other names: factor models, PCA, etc. Some of which we already discussed in Chapter 3, but we will reframe them within the Generative AI framework. This allows us to potentially improve on them at a fundamental level.

The power of representation learning allows you to combine multiple models as building blocks to develop complex applications. We will explore examples of using them, particularly in Chapters 9 and 10, where they are applied to train a model specialized in sentiment analysis for financial data.

4.4 Illustrating Generative Models with ChatGPT

To better understand how we can use generative models for computing probability densities and generate new data, let's examine a concrete example in Natural Language Processing (NLP). For simplicity, we'll set aside specific model details and instead focus on a pre-built model, ChatGPT, examining its density estimation, sampling, and representation learning capabilities.

Let's consider a sequence of random variables where each variable represents a word or token in a sentence. (For this example, we'll ignore the distinction between words and tokens, which will be covered in Chapter 5.) Each element in the sequence takes values from a finite vocabulary—a discrete set. In ChatGPT-4o, for example, the vocabulary consists of 16,384 unique tokens (See OpenAI documentation at https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo), which include common words, word fragments, and characters.

A probabilistic model for a sequence is defined by its joint probability distribution: $p(x_1, \ldots, x_N)$

4.4.1 *Language Modeling*

Modeling this joint probability over a sequence of words/tokens is what is known as the Language Modeling (LM) task in NLP. One of the most straightforward ways to express this joint distribution is by applying the product rule of probability:

$$\begin{aligned} p(x_1, \ldots, x_N) = {} & p(x_1) \\ & \times p(x_2 \mid x_1) \\ & \times p(x_3 \mid x_1, x_2) \\ & \ldots \\ & \times p(x_N \mid \mathbf{x}_{1:N-1}) \end{aligned} \tag{4.2}$$

Models that exploits the right-hand side of Equation 4.2 to represent joint distributions as a sequence of conditional probabilities like this are known as autoregressive models. As it turns out, ChatGPT is an instance of an autoregressive model, specifically a transformer-based model, which we will explore in Chapter 5.

From Equation 4.2, we see that each conditional probability models the likelihood of the next word/token given the previous ones. In other words, ChatGPT performs probabilistic "one-step-ahead" forecasts, where each prediction is based on past values, or past context.

A key question is: How many previous words can ChatGPT consider when making predictions?

This is defined by what is known in NLP as the context window. As of the time of writing, the context window of ChatGPT-4o is 128,000 tokens, meaning it can analyze up to 128,000 previous tokens to estimate the probability of the next one (https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo). This significantly enhances its ability to capture long-range dependencies in text.

4.4.2 *Sampling: How Generative Models Create New Data*

Language models are generative models, meaning they can generate new sequences by sampling from the learned probability distribution. The process follows these steps:

- Generate the first token, $x_1 \sim p(x_1)$
- Generate the second token, $x_2 \sim p(x_2 \mid x_1)$
- Continue generating tokens, $x_n \sim p(x_n \mid x_1, \ldots, x_{n-1})$

Since sampling is sequential, each newly generated token depends on previously generated ones, making the process relatively slow for autoregressive models.

4.4.3 *Conditional Language Generation: Asking ChatGPT a Question*

Just as we can model a sequence of words/tokens with a language model, we can also generate text conditionally. That is, we can condition the generated output on some given context, such as a question or a prompt, which we will denote in the following example as y. This process is used in question answering, where the model generates responses conditioned on user input.

Mathematically, the conditional version of the product rule is given by:

$$\begin{aligned} p(x_1, \ldots, x_N \mid y) = \; & p(x_1 \mid y) \\ & \times p(x_2 \mid x_1, y) \\ & \times p(x_3 \mid x_1, x_2, y) \\ & \ldots \\ & \times p(x_N \mid \mathbf{x}_{1:N-1}, y) \end{aligned} \tag{4.3}$$

Sampling follows the same sequential process described in Equation 4.3:

- Generate the first token $x_1 \sim p(x_1 \mid y)$
- Generate the second token $x_2 \sim p(x_2 \mid x_1, y)$
- Continue generating tokens $x_n \sim p(x_n \mid x_1, \ldots, x_{n-1}, y)$

To illustrate this, we first ask ChatGPT to generate humorous responses to our prompts. Now, let's consider the following user prompt:

User Prompt

What do Mark Zuckerberg and Elon Musk have in common?

When ChatGPT generates a response to given prompt, it follows the process outlined above, predicting one word at a time based on the probabilities it assigns to possible next tokens conditioned on previous tokens and the prompt (y).

Here's an example response generated by ChatGPT-4o:

ChatGPT Response

They both have enough money to buy a small country ... or at least a really nice pizza place! 🍕 💰

Now, how did ChatGPT-4o generate this response?

To understand its output, we used the OpenAI API to access both the sequence of tokens generated by ChatGPT-4o and the probabilities associated with each token in the vocabulary. Since ChatGPT-4o has a vocabulary size of 16,384 unique tokens, visualizing the entire probability distribution over the next word would be impractical. Instead, we display only the top 20 most probable tokens based on their probabilities.

Step-by-Step Breakdown: How the AR Model Generates a Response

1. Computing the first word's conditional probabilities
   - Conditioned on the prompt, the model calculates the probabilities of the first word.
   - As shown in Figure 4.2, the top choices include "They", "They're", and "Both".
   - The model samples "They" (which happens to be the most probable word in this case).

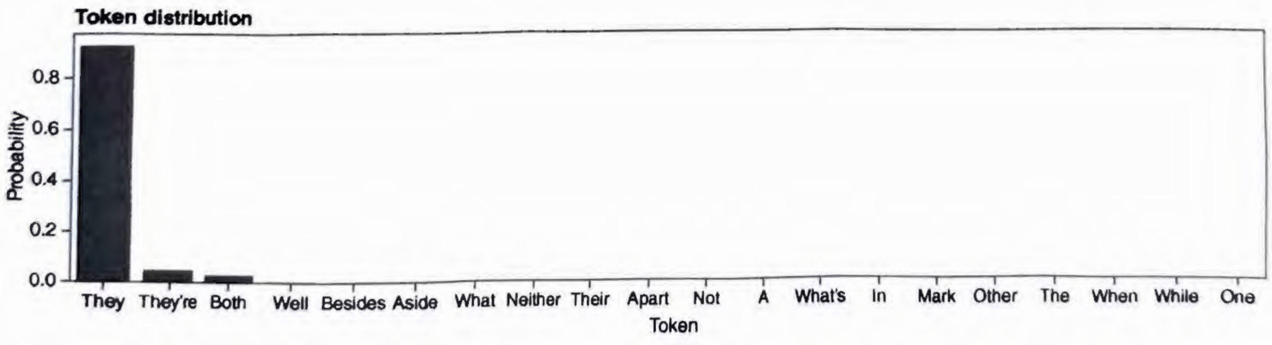

Figure 4.2 Conditional probability of the first token given the prompt y: $p(x_1 \mid y)$.

2. Computing the second word's conditional probabilities
    - Next, the model predicts the probabilities of the second word, conditioned on the prompt and the first word ("They").
    - As shown in Figure 4.3, the top choices include is "both".
    - The model samples "both."

3. Computing the third word's conditional probabilities
    - Now, the model predicts the next word, conditioned on the prompt, "They" and "both".
    - As shown in Figure 4.4, the most probable words are "have" "now" and "think."
    - The model samples "have".

4. Computing the fourth word's conditional probabilities
    - The model then predicts the next word, given the prompt, "They", "both", and "have".
    - As shown in Figure 4.5, the top choices include "enough", "more", "a", and "billions."
    - The model samples "enough."

5. Computing the fifth word's conditional probabilities
    - The model continues, now predicting the next word given the prompt, "They", "both", "have", and "enough".
    - As shown in Figure 4.6, the most probable words include "money", "wealth", and "cash".
    - The model samples "money."

6. The process continues
    - The model sequentially predicts each word based on the previous ones and the original prompt until it forms a coherent response.

Each word generated depends on the probability distribution estimated by the model at that step. In this case, data generation is performed using the probability density

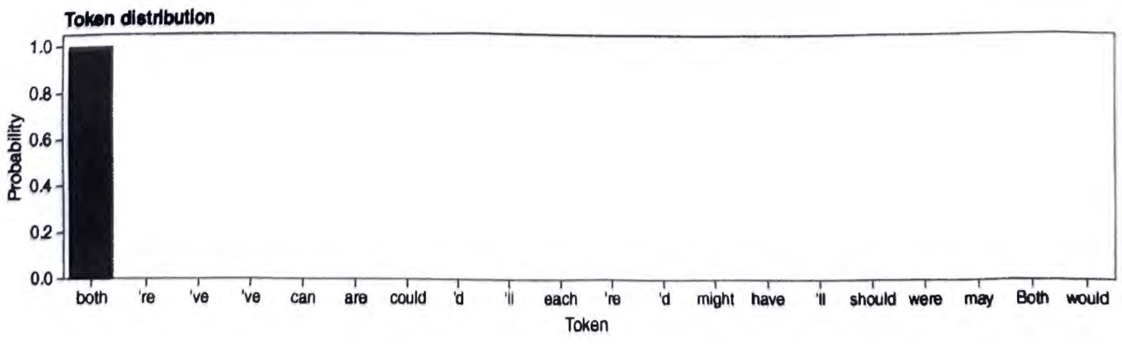

Figure 4.3 Conditional probability of the second token given the prompt y and previous token $x_1 : p(x_2 | x_1, y)$.

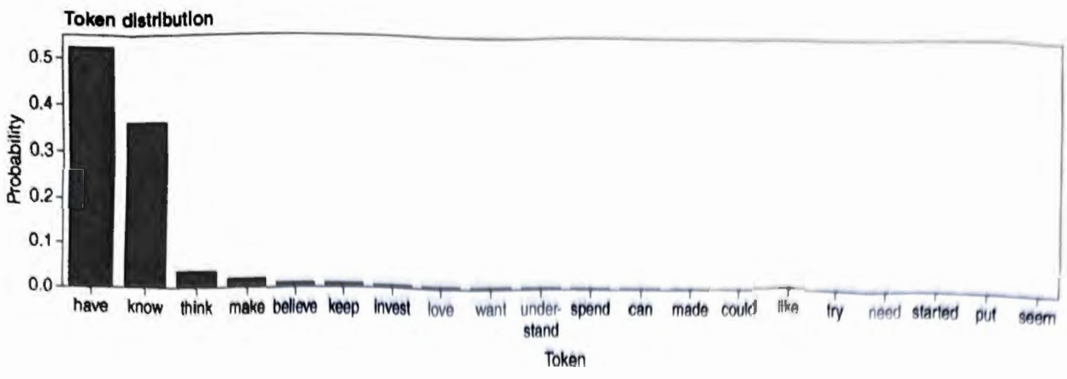

Figure 4.4 Conditional probability of the third token given the prompt y and previous tokens x_1, x_2: $p(x_3 \mid x_1, x_2, y)$.

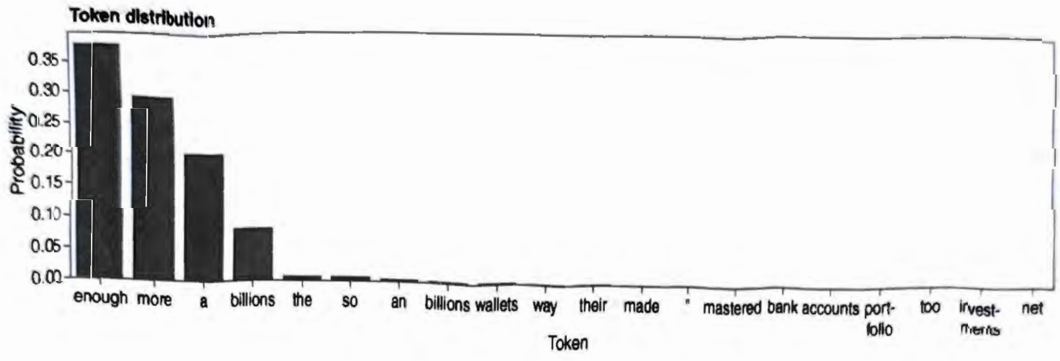

Figure 4.5 Conditional probability of the fourth token given the prompt y and previous tokens x_1, x_2, x_3: $p(x_4 \mid x_1, x_2, x_3, y)$.

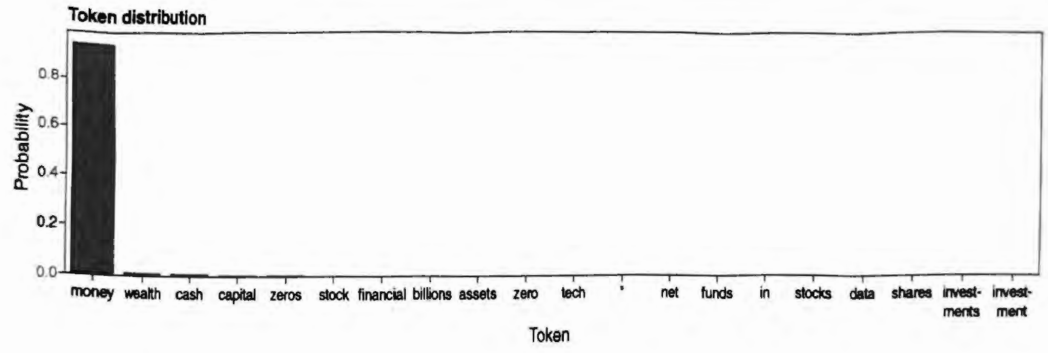

Figure 4.6 Conditional probability of the fourth token given the prompt y and previous tokens x_1, x_2, x_3, x_4: $p(x_5 \mid x_1, x_2, x_3, x_4, y)$.

function. This process illustrates how autoregressive generative models perform sampling in a real-world scenario.

Beyond data generation, probability density estimation can also be applied to forecasting the next most probable token or answering questions about the probability of rare words in a text. This becomes even more relevant when shifting from modeling a sequence of words to modeling a sequence of financial returns, a topic we will explore in Chapter 5. These techniques can be used for applications such as anomaly detection or for computing key financial metrics like Value at Risk (VaR).

As you can see, generative models can function as simulators, which are widely used in finance. A well-known example is option pricing, especially for complex option payoffs using Monte Carlo techniques.

In the example above, the sampling process selects the word/token from the vocabulary with the highest probability at each step. However, in practice, ChatGPT may use more complex sampling strategies (such as temperature scaling or top-k/top-p sampling) to generate more diverse and contextually relevant responses.

If you prefer video lectures, a tutorial based on this section, presented by the authors of this book, can be found on YouTube. You can search for: Trading Using LLM | Generative AI & Sentiment Analysis for Finance | Webinar on the QuantInsti Quantitative Learning YouTube channel.

As you might have noticed with this example, even though these models are highly effective at generating new data, a key question arises: how can we evaluate the quality of the generated data if every run produces different results? As we saw in Chapter 3, the answer is clear for applications like forecasting, where you can compare the true realized values against your predictions using a chosen metric, such as mean squared error.

In our experience developing applications, one of the best ways to evaluate the quality of generated data is to use it for training a downstream model to solve a specific task. By combining the real data with the newly generated data, you can train the model and assess whether its performance improves. However, many metrics for evaluating the quality of generated samples are available in the literature, and we will use several of them throughout the book to assess the performance of these models in data generation.

In the accompanying notebook [*Synthetic Data Evaluation—Notebook*], we will walk through an example to explain the different metrics we will use in our experiments in the book.

This example highlights how ChatGPT, as an autoregressive model, generates text by modeling probability distributions over sequences of tokens.

4.4.4 A Few Words on Representation Learning with ChatGPT

As mentioned earlier, representation learning is about discovering effective ways to encode data that facilitates solving downstream tasks such as prediction, anomaly detection, data generation, among others. Surprisingly, ChatGPT also has the capability to compute new representations of input data in the form of vectors, also known as embeddings or vector embeddings in the NLP literature. These embeddings capture semantic information about words and concepts, enabling us to manipulate and "understand language" by performing mathematical operations with these objects. We will explore embeddings in more details in Chapter 5.

OpenAI provides access to ChatGPT embeddings through its API, allowing users to use these representations for solving applications.

As a use case, imagine you are an investor or a discretionary trader tracking a set of companies where you either hold positions in your portfolio or are considering an allocation. Let's say that you frequently analyze 10-K and 10-Q filings, earnings transcripts, and financial analysis reports to make informed decisions. However, retrieving relevant information across multiple documents can be very time consuming.

Wouldn't it be useful to have a way to extract specific insights from these sources? For example, imagine asking a question like "What are the risks associated with Company XYZ?", and having a system that retrieve relevant documents where these risks are mentioned. To solve this kind of problem, text representations might come to rescue.

To approach this, the first step is to build a knowledge database using vector embeddings. Here's how:

1. Chunking the Documents: Break each financial report or document into manageable chunks (e.g., sentences, paragraphs, or pages).
2. Generating Embeddings: Use OpenAI's Embeddings API to generate vector representations for each document chunk. Store these embeddings in a database.
3. Performing Semantic Search: When a user submits a query (e.g., "What are the risks associated with Company XYZ?"), converts the query into an embedding. This embedding is then compared against stored embeddings in the database using a similarity measure, such as cosine similarity. Then, you can retrieve the top K most relevant text chunks, such that you get the most contextually relevant information for your query.

This embedding-based retrieval approach is a core building block for more advanced techniques like Retrieval-Augmented Generation (RAG), see https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts. RAG enables Large Language Models (LLMs) to generate responses based on external knowledge by enriching the input query, allowing you to customize your LLM for your specific domain without fine-tuning the model. More details on this topic can be found in Chapter 10.

4.5 Hybrid Modeling: Combining Generative and Discriminative Models

To illustrate the benefits of combining both modeling approaches, let's consider an example from commodity trading.

Commodity trading involves buying and selling raw materials such as agricultural products, oil, and natural gas, etc. Success in this field often depends on our ability to accurately predicting supply and demand. For instance, in energy trading, investors and traders would like to track global oil and gas production, which is distributed across different regions of the world.

A particularly useful application of machine learning in this field is leveraging satellite imagery to monitor fracking operations—specifically, detecting well pads (prepared sites where the drilling process occurs). This use case is mentioned in SkyFi's blog, at https://skyfi.com/en/blog/synmax-case-study. SkyFi is a company that provides satellite imagery data and analytics.

Let's say we have access to this data, and we are considering how it can be used to make informed trading decisions. One idea could be to build a classifier that takes satellite images as input and outputs the probability of a well pad being present. This model could be used to estimate the number of active well pads, providing insights into natural gas production and enabling investors and hedge funds to improve their decision making. In the case of fully systematic strategies, for example, the output of such a model could be incorporated into more complex factor-based trading strategies.

Now, let's imagine the process of actually building this model. We'll denote the input image as x and the binary label as y, indicating whether a well pad is present. A common approach is to use a discriminative model that directly estimates $p(y|x)$—the probability of a well pad being present given the satellite image.

We can assume we've collected a high-quality dataset of satellite images, all labeled with precision—either by human annotators or another reliable method. Let's say we use a Deep Neural Network model to solve this problem. The model takes an input image x and directly outputs the probability of y, indicating the presence of a well pad. If the model performs well on test data—measured using common classification metrics like AUC, precision, or recall—we might be tempted to think the problem is solved.

However, real-world deployment presents challenges. One key issue is that unexpected noise in the inputs caused by multiple factors might affect performance. Let's consider what happens when our model encounters an image that is significantly different from anything it has seen before. What will $p(y|x)$ output?

What if an input image has been affected by severe weather and environmental factors? For example, clouds, which cover around 70% of the Earth's surface at any given time, according to NASA's Earth Observatory (https://earthobservatory.nasa.gov/features/CloudsGallery), can obscure key features. What happens if an input image is completely covered by clouds?. Or, instead of clouds, what if geopolitical crises destroy major pipelines or natural disasters drastically alter the landscape? In real-time production environments, the input images may differ drastically from those in our training set. What will $p(y|x)$ output in such cases?

Ideally, in situations of extreme uncertainty, we'd want our model to simply say, "I don't know," and return $p(y = \text{well pad presence}\,|\,x) = 0.5$. But in reality, that's often not the case. Instead, the model may still produce an overconfident prediction.

This problem is well-documented in deep learning, and uncertainty in deep learning remains an active area of research. These models often achieve impressive performance, but they are unable to recognize "when they don't know". For example, in the paper "Deep Neural Networks are Easily Fooled:High Confidence Predictions for Unrecognizable Images" by Nguyen, Yosinski, and Clune (2015), the authors conducted experiments where state-of-the-art classifiers were presented with input images consisting of random noise that were unrecognizable for humans. Despite of the input been random noise, deep neural networks (DNNs) trained on ImageNet classified these inputs with $\geq 99.6\%$ confidence as belonging to familiar object categories, see Nguyen, Yosinski, and Clune (2015). This experiment provides empirical evidence on how modern deep learning models can be overconfident in their predictions, even when the input is entirely noise. Ideally, we would want our model to be robust to such inputs.

This issue arises when the distribution of the training data differs significantly from the distribution encountered during at inference time, a problem known in the machine learning literature as the out-of-distribution (OOD) problem or dataset shift. In the time series and finance literature, this is commonly referred to as the non-stationarity problem. Without going into too much detail, two common scenarios encountered in practice are covariate shift, where the distribution of input features $p(x)$ changes at inference time, and open set recognition, which occurs when new, unseen classes appear at inference time that were not present during training.

One fundamental limitation of modeling $p(y|x)$ alone is that, since it represents a probability distribution over y, the model is forced to assign probability mass over of the known classes, regardless of the input. In a binary classification scenario, for instance, the model must satisfy:

$$p(y = \text{well pad presence}\,|\,x) + p(y = \text{no well pad presence}\,|\,x) = 1$$

As a side note regarding satellite imagery and geopolitical crisis, SkyFi mentions in its blog how satellite imagery was used in 2022 to track energy infrastructure in Ukraine, including power plants, pipelines, and fuel storage facilities.

Well, let's get back to our problem. The question is: how can we solve or mitigate issues like this?

A common solution used in practice is to introduce a new class, typically called an unknown class, where images that do not belong to any of the target categories are labeled as unknown. However, this introduces another challenge: how many classes that do not belong to the target categories should we label as "unknown"? How many examples should we include for each of these classes to achieve good coverage of all possible out-of-distribution (OOD) inputs? If we rely on this method alone, we might find ourselves endlessly adding new categories without truly solving the problem. So, can we tackle this issue using a different approach?

One approach to solving this problem could be combining generative and discriminative models. The main issue with the previous approach is that, at training time, the new image was not present in the training set. But what if we had a way to quantify the probability that x comes from the training distribution? What if, instead of modeling $p(y|x)$, we modeled $p(x, y)$ instead?

By applying the product rule, we know that: $p(x, y) = p(x)p(y|x)$.

This means that if we can build a model for $p(x)$ (a generative model for the input features) and a separate discriminative model for $p(y|x)$, we could potentially provide a solution to our problem. Since our $p(x)$ model aims to approximate the probability distribution of the data, it should assign high probability density to familiar images (inputs) and low probability density to those that are out of distribution. Consequently, inputs that are out of distribution relative to the training set should have low probability density under $p(x)$.

For an OOD input x during training, $p(x)$ should be low, which in turn attenuates the product $p(y|x)p(x)$, making it also low. We can then use a probability threshold to decide whether to trust the prediction. For example, if $p(y|x)p(x)$ falls below a certain value, it indicates too much uncertainty, and the model should avoid making a confident prediction.

Thus, this hybrid modeling approach allows us to combine our preferred discriminative model with a generative model, improving robustness to OOD inputs and uncertainty estimation, leading to better decision-making.

Another use case, distinct from the satellite imagery example, is the well-known application in finance for predicting market regimes. Suppose we have collected a dataset consisting of pairs of inputs x and targets y, where y is a binary variable indicating whether the next market state corresponds to a bull or bear market. The inputs x represent features or characteristics of the current market state that we believe provide valuable information for predicting the next market regime state.

Similar to the previous case on satellite imagery, if we collect data only relevant for bull and bear markets—each with specific characteristics encoded in x—what happens when we encounter a new input that is very different from those observed in either regime? As before, if we model $p(y|x)$, the system will still provide a high-confidence prediction for either bull or bear, even if the input does not fit either category well.

However, modeling $p(x)$ could offer additional insights into the novelty of a given input. This can be useful in two ways:

1. As a standalone anomaly detection system, identifying inputs that deviate significantly from the training distribution.
2. In combination with a discriminative model, improving forecasts by accounting for out-of-distribution (OOD) inputs and uncertainty.

Now, the question is: if we take this approach, how can we build $p(x)$ and $p(y|x)$? There are multiple ways to do this, but two main approaches are described below.

One straightforward solution is to build them independently and then attempt to combine them as $p(y|x)p(x)$. However, there is no guarantee that this will work. In practice, sometimes it performs very well, and other times it does not. This approach may also require numerous iterations over both models, as they are trained independently and decoupled from each other by design.

We can consider an example in finance. You may be building a ML-based trading model to predict the returns of some volatility futures. But there is a great risk of overfitting due to the limited historical data. It may therefore be advantageous to learn from the vast amount of historical data from across all financial series, futures, stocks, FX, or otherwise, and use that as prior probability of the volatility futures returns. We may also be using certain technical indicators as input to this supervised learning model. Computing the marginal probability of these indicators may inform us if certain input are outliers and may not generalize.

Another way to approach this problem is by making the models dependent on each other. This idea is explored in the paper "Hybrid Models with Deep and Invertible Features" by Nalisnick et al. (2019). The key idea is to use a latent variable model that not only captures the distribution of the data but also serves as an input to the discriminative model. This way, both models are tightly coupled and depend on the same latent representation. The training of $p(x)$ and $p(y|x)$ is now coupled.

In the paper, they use a flow model for $p(x)$ (a type of generative modeling you'll learn more about in Chapter 7) and a Generalized Linear Model (GLM) for $p(y|x)$. This setup makes it possible to apply the approach to both general classification and regression problems.

4.6 Taxonomy of Generative Models

Generative models can be broadly categorized into two main types based on how they model data distribution: Explicit Density Models and Implicit Density Models. Explicit Density Models explicitly define and calculate the probability distribution of the data. Examples of Explicit Density Models include the following:

- Autoregressive Models: These models generate data sequentially, predicting each piece of the data based on what has been generated so far.
- Variational Autoencoders (VAEs): VAEs learn to encode input data into a lower-dimensional space and then decode it back, allowing for efficient data generation.
- Flow Models: This family of models transforms simple distributions into complex ones, maintaining the ability to calculate the exact likelihood of the data.
- Diffusion Models: These models gradually add noise to data and learn to reverse this process, generating data by denoising. (We don't have space to discuss diffusion models in this book, but we will cite references to it.)

On the other hand, Implicit Density Models aim to generate data by capturing the data distribution without explicitly defining it. A well-known example of these models is:

- Generative Adversarial Networks (GANs): GANs involve a dual-network architecture where one network generates data and the other evaluates it, leading to highly realistic outputs.

See Figure 4.1 for a categorization of the main types of generative models.

Each model family, as specified in the model taxonomy, has its own advantages and disadvantages for abilities mentioned in the previous section. In the accompanying notebook [*Model Family Comparison—Notebook*], we present a list of the main model families and compare their efficiency. This table is something we hope to keep updating as new models are released or better techniques are developed.

4.7 Conclusion

Generative models represent a profound shift in AI's capabilities. By learning to understand and manipulate data, they enable us to categorize the world as it is and to imagine and create what it could be. From creating art to advancing scientific discovery, Generative AI is not just a tool for innovation but a foundation for future advancements in the field. In the following chapter, we will dive deeper into each of the key model families for generative models.