How Does This Course Help in ML Interviews?

Learn how this course will prepare you for the machine learning system design interview at any technology giant.

We'll cover the following

- The rise of machine learning
- What to expect in a machine learning interview?
- Machine learning system design interviews

The rise of machine learning

Machine learning (ML) has been one of the fastest-growing fields, and it is projected to grow from $7.3B in 2020 to $30.6B in 2024. Machine learning aims at solving a multitude of complex problems and has seen rapid progress in several areas such as speech understanding, visual understanding, search ranking, credit card fraud detection, and so on.

A subset of areas where ML has made significant advancements

As a result of these major developments, machine learning usage and jobs have grown massively in the last few years and are expected to grow even faster in the upcoming years.

What to expect in a machine learning interview?

Companies hiring for machine learning roles conduct interviews to assess individual abilities in various areas. You can expect the following topics to be covered in these interviews:

What to expect in a machine learning interview?

Problem-solving/coding

This portion of the interview is fairly similar to other software engineering coding interviews where the interviewer gives a coding problem, such as perform an 'In-order tree traversal', and the candidate is expected to solve that in about half an hour. There is ample content available on how to best prepare for such questions.

Machine learning understanding

This area generally focuses on individual understanding of basic ML concepts such as supervised vs. unsupervised learning, reinforcement learning, classification vs. regression, deep learning, optimization functions, and the learning process of various ML algorithms. There are many courses and books that go over these fundamental concepts. They facilitate the learning of ML basics and help candidates prepare for the interview.

Career Discussion

Career discussion tends to focus on an individual's resume (previous projects) and behavioral aspects, such as the ability to work in teams (conflict resolution) and career motivation. Understanding the path you want to take in your career and having the ability to discuss previous experiences and projects is required for this portion.

Machine learning system design discussion

This discussion focuses on the interviewee's ability to solve an end-to-end machine learning problem and consists of open-ended questions. This is an integral part of the interview, and not much helping material is available for it. Hence, this course helps in developing the thought pattern required to approach ML system design questions.

Machine learning system design interviews

ML system design interview questions are vital in assessing the interviewee's performance for machine learning jobs. While there are numerous courses and books available on different ML concepts and techniques, very few consider approaching the ML problem from a system design perspective, and even fewer focus on how to prepare for such an interview. This course intends to cover this gap and goes over the most popular machine learning problems solved by top tech giants like Google, Netflix, Twitter, Microsoft, Tesla, and Facebook.

Prepare yourself for the ML system design interview of tech giants

In the ML system design interview portion, candidates are given open-ended machine learning problems and are expected to build an end-to-end machine learning system to solve that problem. Few of such problems could be:

- Build a *recommendation system* that shows relevant products to users.
- Build a *visual understanding system* for a self-driving car.
- Build a *search-ranking system*.

In order to answer such questions, the candidates should consider the following approach.

How a candidate should approach machine learning system design questions

This approach is further detailed in the next lesson in the form of a guideline. We will use this guideline to come up with end-to-end solutions for the machine learning system design problems.

> 📝 In the practical ML techniques chapter, we will also go over some key practical methods and techniques that are helpful in designing machine learning systems. These methods and techniques include important performance and capacity considerations, embeddings, online experimentation, model debugging and testing etc.

We hope that this course will give you key insights on designing an end-to-end machine learning system and as a result, equip you with the right tools to solve any machine learning design problem. Happy learning and best of luck with your interview preparation!

Setting up a Machine Learning System

By Lili Jiang and Martin Wattenberg with contributions and advice from Jeffrey Pennington, Ashish Vaswani, Ian Tenney, Adam Pearce, Nithum Thain, and Jon Shlens

We will use the Transformer in examples that are discussed later in the course.

We'll cover the following

- Overview
- Key steps in ML system setup
- Setting up the problem
- Understanding scale and latency requirements
- Defining metrics
  - Architecture diagrams
  - Offline metrics
  - Offline model building and evaluation
  - Online model execution and evaluation
  - Iterative model improvements

Overview

Interviewers will generally ask you to design a machine learning system for a particular task. For instance, they may ask you to design the machine learning system for the following problems:

- *Build a system that shows relevant ads for another user.*
- *Extract all persons, locations, and organizations from a given corpus of documents.*
- *Recommend movies to a user on Netflix.*

In your interview preparation, you would find a plethora of resources to tell you details about how different machine learning models work. However, there are not many resources that show you how to approach the ML system design of such a complex and large scale problem.

What you are searching for is a comprehensive resource that provides practical knowledge to interviewer prospective candidates for machine learning interviews. Unfortunately, this course constitutes a 0% of it.

Before diving into the chapters, let's get too familiarized with the thought process required to answer an interviewer's questions.

Key steps in ML system setup

In the following chapters, you will observe that the key steps involved in machine learning project set up are as follows:

Setting up the problem

The interviewer's question is generally very broad. So, the first thing you need to do is ask questions. Asking questions will help you gain a better understanding of the question and clear the interviewer's expectations from your answer. You will be able to narrow down your problem space. Only ask the requirements of the system, and finally, propose a suitable machine learning problem statement.

For instance, you may be asked to design a search engine that displays the most relevant results in response to a given query. You can narrow down the problem scope by asking the following questions:

- *Is it a general search engine like Google or Bing or a specialized search engine like Amazon's products search?*
- *What kind of queries is expected to answer?*

This will allow you to precisely define your ML problem statement as follows:

Build a generic search engine that returns relevant results for queries like "Richard Nixon", "Programming languages" etc.

Or, you may be asked to build a system to display a Twitter feed for a user. In this case, you can discuss how the feed is currently displayed and how it can be improved to provide a better experience for the users.

If in the feed-based system chapter, we discuss how the Twitter feed was previously displayed in chronological order, it suggests users to miss out on important tweets. From this discussion, we realized that we want to move towards displaying content *in order of relevance* instead.

Read this chapter's blog for more details.

After inspecting the problem from all aspects, you can easily narrow it down to a precise machine learning problem statement as follows:

"Given a list of tweets, train an ML model that predicts the probability of engagement of tweets and orders them based on this score."

If some problems may require you to think about hardware components that could provide input for the machine learning models.

Understanding scale and latency requirements

Another very important part of the problem setup is the discussion about performance and capacity considerations of the system. The interviewer will allow you to clearly understand the scale of the system and its requirements.

Let's look at some examples of the questions you need to ask.

Latency requirements

If you were given the search engine problem, you would ask:

- *Do we want to return the search result in 500 milliseconds or 100 milliseconds?*

Similarly, if you were given the Twitter feed problem, you would ask:

- *Do we want to return the list of relevant tweets in 500 milliseconds or 400 milliseconds?*

Scale of the data

Again, for the search engine problem, you would ask:

- *How many requests per second do we anticipate to handle?*
- *How many relevant results that we want to enable through this search engine?*
- *If a query has 10 million matching documents, how many of these would be ranked by our model?*

And for the Twitter feed problem, you would ask:

- *How many tweets would we have to rank according to relevance for a user at a time?*

Performance and capacity considerations are detailed in this lesson.

The answers to these questions will guide you when you come up with the architecture of the system. Knowing that you need to return results quickly will influence the depth and complexity of your models. Having huge amount of tweets to process, you will design the system with scalability in mind. Find more on this at the architecture discussion section.

Defining metrics

Now that you have figured out what machine learning problem you want to solve, the next step is to come up with metrics. Metrics will help you to assess if your system is performing well.

Knowing our success criteria helps in understanding the problem and in selecting key architectural components. This is why it's important to discuss metrics early in our design discussions.

Metrics for offline testing

You will use offline metrics to quickly test the models' performance during the development phase. You may have generic metrics for comparing models or for performing classification, you will see MSE, log loss, precision, recall, and F1 score. In other cases, you might have to come up with specific metrics for a certain task. For instance, for the search ranking problem, you would use NDCG as a metric.

Metrics for online testing

Once you have selected the best performing model, you will use online metrics to test them in the production environment. The decision to deploy the newly created model depends on its performance in the online testing.

While coming up with online metrics, you may need both **component-wise** and **end-to-end** metrics. Consider that you are making a search ranking model to display relevant results for search queries. You would want to measure the click-through rate (CTR) of the search results displayed by the system. However, you also want to look at how the system responds to queries performing well or even more poorly plugged in, for which you can use end-to-end metrics. A commonly used end-to-end metric for this scenario is the users' engagement and retention rate.

In another scenario, you may be asked to develop the ML system for a task that may be used as a component for other tasks. Again, you may need both component-level metrics and end-to-end metrics during online testing. For instance, you could be asked to design the ML system for entity linking which is going to be used to improve search relevance.

Here, you will have component-wise metrics to evaluate the performance of the entity linking model individually. You will also be required to come up with metrics for the search engine where the entity linking component will eventually be plugged in.

Architecture diagrams

The next step is to figure out the architecture of the system. You need to think about the components of the system and how the data will flow through these components.

To get an idea, have a peek into the architecture of a search engine's ML system may be designed, below.

Architectural components for ML system of search ranges

You start off with a newsfeed ranker that displays relevant tweets to a user. Later, you can expand the system to include "Who to follow" suggestions. The system may be purely unrelated or unrelated or you need the query rewriting component to process the search queries. Finally, the results are displayed on the search engine result page (SERP).

Architecting for scale

As we mentioned previously, the requirements gathered during problem setup help you in chalking out the architecture. For instance, you are tasked with building an ML system that displays relevant ads to users. Depending on the scale, you will prioritize and realize the number of users and ads in the system is huge and ever increasing. Thus, you need a scalable system that quickly figures out the relevant ads for all users despite the increase in data.

Hence, you can't just build a complex ML model and run it for all ads in the system because it would take up a lot of time and resources. The solution is to use hierarchical approach, where each stage will have fewer ads to process. This way, you can safely use complex models in later stages.

Hierarchical approach for ML system of search ranges

Offline model building and evaluation

The next step is to start building the model offline and then evaluate it. This step involves:

- **Training data generation**:

Training data is fuel for learning. Therefore, it is crucial that your training data is of good quality and quantity. Broadly speaking, we have two methods of training data collection/generation for supervised learning tasks.

1. Human labeled data

We can have labelers who are label data for us according to the given ML task. For instance, if we have to perform segmentation of driving images, labelers will use software such as label.ai to mark the boundaries of different objects in the driving images.

Manual labeling of driving images to generate training data for segmentation

This is an expensive way to gather data. So we need to supplement it with in-house labelers or open-source datasets. Interestingly, a recent paper *Revisiting Deep Intelligent Vision* is an example of an open-source dataset that can be used as training data for the segmentation task. It contains segmented data for driving images.

- You will see another way to enhance training data in the image segmentation chapter!

2. Data collection through a user's interaction with the prevailing system

Another way to gather data is through the online system's deployment(or existing) system.

- If Online system refers to a running system, currently in place, that people are interacting with. For example, a running search engine or ad system that people are using.

To get an idea, let's go back to the search ranking example, where it was asked to design a new ML-based system to rank the search results. The system implied creation of a search engine that would also show results for user queries (using an existing ML-based system or a rule-based system). You can log the queries and results for further analysis and training. Later, if a new clickbait (or a crazy dress) you can insert it as a positive training example. Similarly, an impression can count as a negative example (refer to the search ranking chapter for details).

The training data collection strategies are detailed in the lesson. Have a look to find out some innovative ways to collect and expand your training data.

- **Feature engineering**:

This is another very crucial step as good features influence the model's ability to a great extent. You start this process by explicitly pinpointing the actors involved in the given task, which you have implicitly identified during problem setup as well.

For example, if the task at hand is to make movie recommendations in a Netflix user, you would have the following actors:

Five actors involved in the movie recommendation task

In order to make features, you would individually inspect these actors and explore their relationships too. For instance, if you individually inspect the logged in user, you can come up with features such as the user's age, gender, location, etc. Likewise, if you look at the context, an important feature could be the "upcoming holiday". If Christmas is approaching and the movie under consideration is also a Christmas movie, there is a greater chance that the user would like it. Similarly, we can look at the historical engagement between the user and media to come up with features. An example could be the user's interaction with the movie's genres in the last three months.

A subset of features in the training data

For more ideas and details, read the chapter on recommendation systems.

- **Model training**:

Now, you can finally decide on the ML models that you should use for the given tasks, keeping the performance and capacity considerations in mind. We can also try out different hyperparameter values to see what works best.

If you are using the Search approach, you may select simpler models for the top of funnel where data size is huge and more complex neural networks or tree-based models for successive parts of the funnel.

We also have the option of utilizing pre-trained NLP models such as the BERT models to leverage the power of transfer learning (you don't need to reinvent the wheel completely each time).

Look at how to use pre-trained NLP models in the entity linking chapter.

- **Offline evaluation**:

With careful consideration, divide the data into training and validation sets. Use the training data set during the model training and use it to evaluate the model. Use cross-validation to ensure robustness. Hyperparameters, Iterate, rinse, etc.). Afterwards, we evaluate these models, offline, on the validation. Depending on the problem, you may use several metrics to measure their performance. The top few models, showing the most promise, are taken to the online test.

Offline learning is very beneficial, as it allows us to quickly test many different models so that we can select the best one for online testing, which is a slow process.

Online model execution and evaluation

Now that you have selected the top performing models, you will test them in an online environment. Online testing heavily influences the decision of deploying the model. This is where online metrics come into play.

Depending on the type of problem, you may use both component-level and end-to-end metrics. As mentioned before, for the search engine task, the component-wise metric will be NDCG in online testing. Everything is not rosy, however. You also need an end-to-end metric, as well, the **session success rate**, to see if the system's search engine's performance has increased by using your new search ranking ML model.

If you see a substantial increase in system performance during the online test, you can deploy it on production.

For more details, read the lesson on online experimentation.

Iterative model improvement

Your model may perform well during offline testing, but the same increase in performance may not be observed during online test. Now, you need to think about debugging the model to find out what exactly is causing this behavior.

Is a particular component not working correctly? Is the feature/ distribution different during training and testing time? For instance, a feature called "user's top film interest" may show a difference in distribution during training and testing, either plotted. This can help you to identify the reasons used to provide the top five user interests were significantly different during training and testing time.

Similarly, after the first version of your model has been built and deployed, you will need to monitor its performance. If the model is not performing as expected, you need to go towards debugging. You may observe a general failure from a decrease in AUC. Or, you may note that the model is failing in particular scenarios. For instance, by analyzing the video recording of the self-driving car, you may find out that the image segmentation fails in rainy scenes.

The problem areas identified during model debugging will guide you in building successive iterations of your model.

Get some model debugging ideas in the model debugging & testing lesson.

Performance and Capacity Considerations

This lesson provides a quick introduction to performance and capacity considerations and discusses why they matter when designing a solution to a machine learning problem.

We'll cover the following

- Complexities consideration for an ML system
- Comparison of training and evaluation complexities
  - Analysis
- Performance and capacity considerations in large scale system
- Layered/funnel based modeling approach

As we work on a machine learning-based system, our goal is generally to improve our metrics (engagement rate, etc.) while ensuring that we meet the capacity and performance requirements.

Major performance and capacity discussions come in during the following two phases of building a machine learning system:

1. *Training time*: How much training data and capacity is needed to build our predictor?
2. *Evaluation time*: What are the Service level agreement(SLA) that we have to meet while serving the model and capacity needs?

We need to consider the performance and capacity along with optimization for the ML task at hand, i.e., measure the complexity of the ML system at the training and evaluation time and use it in the decision process of building our ML system architecture as well as in the selection of the ML modeling technique.

Complexities consideration for an ML system

Machine learning algorithms have three different types of complexities:

- **Training complexity**

  The training complexity of a machine learning algorithm is the time taken by it to train the model for a given task.
- **Evaluation complexity**

  The evaluation complexity of a machine learning algorithm is the time taken by it to evaluate the input at testing time.
- **Sample complexity**

  The sample complexity of a machine learning algorithm is the total number of training samples required to learn a target function successfully.

> Sample complexity changes if the model capacity changes. For example, for a deep neural network, the number of training examples has to be considerably larger than decision trees and linear regression.

The measure of training and evaluation time complexity on a sample data in a ML system

Comparison of training and evaluation complexities

You can see how the training and evaluation complexities can be used to evaluate which model will be best for a given task and resources.

Assume that

- n is the number of the training samples
- f is the number of features
- n_{trees} is the number of trees (for tree-based algorithms)
- n_{l_i} is the number of neurons at i^{th} layer in a neural network
- e is the number of epochs
- d is the max depth of the tree

The training and prediction complexity can be approximated in terms of asymptotic analysis as follows:

Algorithm	Training time	Evaluation time
Linear/Logistic Regression (Batch)	$O(nfe)$	$O(f)$
Neural Network	Exponential (varies per implementation)	$O(fn_{l_1} + n_{l_1}n_{l_2} + ...)$
Multiple Additive Regression Trees (MART)	$O(ndfn_{trees})$	$O(fdn_{trees})$

Analysis

- The evaluation complexity of the *linear regression* algorithm is equal to the complexity of a single-layer neural network-based algorithm. Linear regression is the best choice if we want to save time on training and evaluation. Let's assume the model evaluates one example in 5 μs. For 100k examples, it would take 100k x 5 μs = 500 ms execution time on a single machine.

  For example, for the ad prediction system, the service level agreement(SLA) says that we need to select the relevant ads from the pool of ads in 300 ms. Given this request, we need a fast algorithm. Here linear regression would serve the purpose.
- Relatively *deep neural network* takes a lot more time in both training and evaluation. Its need for training data is also high. However, it's ability to learn complex tasks such as image segmentation and language understanding, is much higher, and it gives more accurate predictions in comparison to other models. Therefore a deep neural network is a viable choice if it is well suited for the task at hand and capacity isn't a problem.
- *MART* is a tree-based algorithm that has a greater computation cost than linear models, but it is much faster than a deep neural network. Tree-based algorithms are able to generalize well using a moderately-sized training dataset. Therefore, if our training data is limited to a few million examples and capacity/performance is critical, they will be a good choice.

Performance and capacity considerations in large scale system

Consider that a search system(e.g., Google, Bing) gets a query "computer science" that matches 100 million web pages. The ML-based system wants to respond with the most relevant web pages for the searcher while meeting the system's constraints. These constraints are generally referred to as Service level agreements (SLA). There can be many SLAs around availability and fault tolerance but for our discussion of designing ML systems, *performance* and *capacity* are the most important to think about when designing the system. **Performance** based SLA ensures that we return the results back within a given time frame (e.g. 500ms) for 99% of queries. **Capacity** refers to the load that our system can handle, e.g., the system can support 1000 QPS (queries per second).

If we evaluate every document using a relatively fast model such as tree-based or linear regression and it takes 1μs, our simple model would still take 100s to run for 100 million documents that matched the query "computer science".

This is where distributed systems come in handy; we will distribute the load of a single query among multiple shards, e.g., we can divide the load among 1000 machines and can still execute our fast model on 100 million documents in 100ms (100s/1000).

Input query sent to multiple shards to distribute load

Let's now consider the scenarios in which we decided that a deep learning model for search ranking is a much better choice and helps improve our search metrics. However, deep learning is significantly slow, assuming that it needs 1ms to evaluate an example. Even now with our 1000 shards, it would still take 100s to rank all the results using this model. Clearly, we are far from our performance SLAs.

If we had unlimited capacity, one way to solve this problem would be to continue to add more shards and bring the number down. However, we cannot have unlimited capacity, and it's important to have the system find the most optimal result given a fixed capacity.

This prompts another important discussion on using a funnel-based approach when designing ML systems for performance and limited capacity. Let's discuss it next.

Layered/funnel based modeling approach

To manage both the performance and capacity of a system, one reasonable approach that's commonly used is to start with a relatively fast model when you have the most number of documents e.g. 100 million documents in case of the query "computer science" for search. In every later stage, we continue to increase the complexity (i.e. more optimized model in prediction) and execution time but now the model needs to run on a reduce number of documents e.g. our first stage could use a linear model and final stage can use a deep neural network. If we apply deep neural network for only top 500 documents, with 1ms evaluation time per document, we would need 500ms on a single machine. With five shards we can do it in around 100ms.

> In ML systems like search ranking, recommendation, and ad prediction, the layered/funnel approach to modeling is the right way to solve for scale and relevance while keeping performance high and capacity in check.

The following figure shows an illustration of this multi-layer funnel approach that we will explain in detail in the search ranking system design problem.

Layered model approach for search ranking system

Similarly, the following is an illustration of how an ads relevance system would look in this funnel based approach. We will discuss this in more detail in the ad prediction system chapter.

Funnel approach for an ad prediction system

Training Data Collection Strategies

Learn the training data collection strategies for the machine learning system you are going to build

We'll cover the following

- Significance of training data
- Collection techniques
  - User's interaction with pre-existing system (online)
  - Human labelers (offline)
- Quantity of training data collection techniques
- Tricks, tips, & techniques
- Quantity of training data
- Training data filtering

Significance of training data

A machine learning system consists of three main components. They are the training algorithm (e.g., neural network), the training data, and the model. The training data is of paramount importance. The model learns directly from the data to create and refine its rules on a given task. Therefore, inadequate, inaccurate, or irrelevant data will result in poor model performance. Consequently, the quality and quantity of training data are a big factor in determining how far you can go in our machine learning implementation.

A lot of the progress in machine learning, AI, and DL is an unappealing reality: it is driven by the increase in both computing power and data, on average, in a field's quest to learn rules. You need a big rocket engine, and you need a lot of fuel. - Andrew Ng

Most real-world problems fall under the category of supervised learning problems which require labeled training data. Therefore, we'll focus on understanding a little more about the collection of labeled data. And we can ignore the learning rules.

Let's explore strategies that will help us collect labeled training data for our learning tasks.

Collection techniques

We will begin by looking at the training data collection techniques.

User's interaction with pre-existing system (online)

In some cases, the user's interaction with the pre-existing system can generate good quality training data.

If we will utilize this technique an online data collection is the course.

For many years, only owners had the option to hire labelers for our training problem a web-based system. With the cold-launch system in place, you can now open the system for the cold launch to them internally and invite them to when you build the system. Now, we can start collecting training data from the prospective training system to generate training data for model training. Let's get a better idea with an example building a cancer classification system.

Assume that you are asked to recommend movies to a Netflix user. You will be training a model that will predict the probability of a user watching a given movie. Initially, you don't have any training data. Alternatively, you can start a cold launch system, recommend movies to users, and collect the data that you didn't like a particular movie to feed into your model.

The interface of an online Netflix-like system can generate training data through their interactions with the current system that is being used for recommending movies.

Online data collection pipeline for a movie recommendation system

The early version for movie recommendation might be popularity-based. Irrelevant movie's being hunted hard-coded or dynamic. The important point here is that the system will generate training data from its interaction with the system. If a user likes/watches a movie recommendation, it will count as a positive training example. But if a user dislike/ignores a movie recommendation, it will be seen as a negative training example.

If the direct discussion was one example, that may machine learning systems utilize the current system in place for generation of training data.

We will discuss training data generation strategies from the current system in multiple problems in the coming chapters, such as ranking, advertising and recommendation systems.

Human labelers (offline)

In other cases, the use of the system would not be able to generate training data. Here, you will utilize labelers to generate good quality training data.

If we will utilize this technique an offline data collection is the course.

Let's take an example of anti-vaxx cancer cure. Image segmentation system for a self-driving car.

Assume that you want to perform image segmentation to identify the different objects on the road. Initially, you don't have any training data. You can hire labelers from prestigious companies like companies like Scale, Lionbridge, etc. Alternatively, signed and so on. But, you will need to provide accurately segmented images of the self-driving car's surroundings to train your model. Here, the system is not in the picture.

Here, the customer of the system (the person sitting in the self-driving car) can't generate training data for the system. The customer will pay the system to use a for-hire world gen organizations labels for the images they need for the system.

In such scenarios, we need to figure out the prescriptions that can generate labeled training data for us.

These such resources are:

- **Crowdsourcing**

  As the name implies, in crowdsourcing, we outsource a task to a large group of people. Several crowdsourcing platforms are Amazon Mechanical Turk, Figure Eight, Crowdflower, etc. Instead of hiring a second group for each demand, we can outsource to the crowd.

  Crowdsourcing can be used to collect training data for relatively simple tasks. For instance, if you are building an email spam detection system, you can send a labeled example as spam or not. This is a simple task that crowd workers can easily do without requiring any special training.

Crowdsourcing for task of generating labeled training data

However, there are cons, like when we have privacy concerns, where we cannot utilize crowdsourcing. If the data is too complex, we can't use crowdsourcing. Also, if we need subject matter experts to label the data, crowdsourcing is not the course. Also, if we need a lot of human expertise/ specialized training, in these cases, we need to have specialized labelers to do the job.

- **Specialized labelers**

  We can hire specialized/trained labelers who can label data for us according to the given AI. task. Let's say we have to hire a system for the segmentation of images for the autonomous vehicle. The trained labelers will use software, such as Label.ai, to mark the boundaries of different objects in the image. Because we need to hire trained labelers, this approach is more costly than crowdsourcing. Also, the turnaround time is comparatively slow. The tools would be delayed until enough labelers have been trained.

Specialized labelers generating training data

- **Targeted data gathering**

  Offline training data collection is expensive. So, you need to identify what kind of training data is most important and target that type of training data. To do this, you should use the system for labeling, i.e., areas where the system is unable to predict accurately. Your focus should be to collect more training data for these areas.

  Continuing with the autonomous vehicle example, you would see where your segmentation system is failing and collect data for those scenarios. For instance, you may find that it performs poorly for night time images and bad weather or dusty roads in the morning time. Therefore, you focus more on gathering and labeling night time images and those with multiple pedestrians.

  Active learning is a model strategy and tooling focus to find out how to identify areas where the system is performing poorly.

- **Open-source datasets**

  Generating training data through manual labelers is an expensive and time-consuming way to gather data. So, we need a supplement. It will save us from dataset whose problem.

  For instance, "MS COCO: A Large-scale Object Detecting Video Database" is an example of an open-source dataset that can be used for training data for the segmentation task. It contains labeled segmented data for driving images.

MS COCO: open source data set for segmentation

If you tap out more than one of these methods for ways to generating training data. The search ranking chapter is a good example of this.

Additional creative collection techniques

Let's look at some creative ways to collect training data.

- **Build the product in a way that it collects data from user**

  We can tweak the functionality of our product in a way that it starts generating training data for our model. Let's consider an example photo tagging system that internes on Pinterest. You want to train a personalized recommendation system for the user's dashboard. The prescription, Pinterest, that would give you a semantic understanding of the user and the pins. This can be done by tweaking the photo tagging feature as follows.

  A user comes to the board (collection) to which they want to add a pin. The system will suggest a board to help to categorize the pins according to their interests.

  The system will suggest a few boards based on the board names specified by existing users. The first step will help you to build content profiles. Whereas, the second step will help you build user profiles based on online those to whom pins that would interest the user, personalizing the experiences.

Generating the prescription in way that generates training data

- **Creative manual organization**

  Manual organization can generate training creative ways. Assume that we are building a system that detects google image (photo) interiors and tag them according to the room type (kitchen, living room, etc). We can supplementation the training data by manually placing those tagged to a different set of images as well. The tagged pictures can be used for training. Alternatively, we can use the manual tracking data set results as to build a more robust large detector model, which will be able to identify type of all rooms or different pictures of the same kitchen.

Manual organization
Manual organization
Training data
Training data
Training data

Online Experimentation

Irrespective of the problem you're working on, model experimentation and evaluation flow are always critical. In this lesson, we will go over the key steps and concepts in model experimentation and evaluation.

We'll cover the following

- Hypothesis and metrics intuition
- Running an online experiment
- Measuring results
  - Computing statistical significance
- Measuring long term effects
  - Back Testing
  - Long-running A/B tests

A successful machine learning system should be able to gauge its performance by testing different scenarios. This can lead to more innovations in the model design. For an ML system, "success" can be measured in numerous ways. Let's take an example of an advertising platform that uses a machine-learning algorithm to display relevant ads to the user. The success of this system can be measured using the users' engagement rate with the advertisement and the overall revenue generated by the system. Similarly, a search ranking system might take into account correctly ranked search results on SERP as a metric to claim to be a successful search engine. Let's assume that the first version of the system (v0.1) has been created and deployed.

The initial version of the system is created

Hypothesis and metrics intuition

At any point in time, the team can have multiple hypotheses that need to be validated via experimentation.

Imagine, for instance, that the team designing an ad prediction system wants to test the hypothesis that increase in the neural network model depth (increase in hidden layers) or width (increase in activation units) will increase latency and capacity but will still have an overall positive effect on user engagement and net ad revenue.

Team desires to test multiple hypotheses to see the impact on the system

Similarly, a team working on designing a search engine wants to test the hypothesis that the pointwise algorithm instead of the pairwise algorithm would positively impact search relevance.

Team desires to test the hypotheses to see the impact on the system

So, to test the hypotheses, should the ML system v0.2 be created and deployed in the production environment? What if the hypothesis intuition is wrong and the mistake becomes costly?

This is where online experimentation comes in handy. It allows us to conduct controlled experiments that provide a valuable way to assess the impact of new features on customer behavior.

Running an online experiment

A/B testing is very beneficial for gauging the impact of new features or changes in the system on the user experience. It is a method of comparing two versions of a webpage or app against each other simultaneously to determine which one performs better. In an A/B experiment, a webpage or app screen is modified to create a second version of the same page. The original version of the page is known as the control and the modified version of the page is known as the variation.

A/B testing

We can formulate the following two hypotheses for the A/B test:

- **The null hypothesis**, H0 is when the design change will not have an effect on variation. If we fail to reject the null hypothesis, we should not launch the new feature.
- **The alternative hypothesis**, H1 is alternate to the null hypothesis whereby the design change will have an effect on the variation. If the null hypothesis is rejected, then we accept the alternative hypothesis and we should launch the new feature. Simply put, the variation will go in production.

Now the task is to *determine if the number of successes in the variant is significantly better from the control*, i.e., if the conversion caused a positive impact on the system performance. This requires confidently making statements (using statistical analysis) about the difference in the variant sample, even if that difference is small. Before statistically analyzing the results, a power analysis test is conducted to determine how much overall traffic should be given to the system, i.e., the minimum sample size required to see the impact of conversion. Half of the traffic is sent to the control, and the other half is diverted towards the variation.

Traffic split evenly between version A and B

Measuring results

As visitors are served with either the control or variation/test version of the app, and their engagement with each experience is measured and analyzed through statistical analysis testing. Note that unless the tests are statistically significant, we cannot back up the claims of one version winning over another.

Computing statistical significance

P-value is used to help determine the statistical significance of the results. In interpreting the p-value of a significance test, a significance level (alpha) must be specified.

> The significance level is a boundary for specifying a statistically significant finding when interpreting the p-value. A commonly used value for the significance level is 5% written as 0.05.

The result of a significance test is claimed to be "statistically significant" if the p-value is less than the significance level.

- p <= alpha: reject H0 - launch the new feature
- p > alpha: fail to reject H0 - do not launch the new feature

If an A/B test is run with the outcome of a significance level of 95% (p-value <= 0.05), there is a 5% probability that the variation that we see is by chance.

Statistical test analysis shows system B(variation) outperforms system A(control).

Measuring long term effects

In some cases, we need to be more confident about the result of an A/B experiment when it is overly optimistic.

Back Testing

Let's assume that variation improved the overall system performance by 5% when the expected gain was 2%. In the case of the ads prediction system, we can say that the rate of user engagement with the ad increased by 5% in variation (system B). This surprising change puts forth a question. Is the result overly optimistic? To confirm the hypothesis and be more confident about the results, we can perform a backtest. Now we change criteria, system A is the previous system B, and vice versa.

Backtest

We will check all potential scenarios while backtesting:

Do we lose gains? Is the gain caused by an A/B experiment equal to the loss by B/A experiment? Assume that the A/B experiment gave a gain of 5% and B/A experiment gave a loss of 5%. This will ensure that the changes made in the system improved performance.

Long-running A/B tests

In a few experiments, one key concern could be that the experiment can have a negative long term impact since we do A/B testing for only a short period of time. Will any negative effects start to appear if we do a long term assessment of the system subject to variation?

For example, suppose that for the ad prediction system, the revenue went up by 5% when we started showing more ads to users but this had no effect on user retention. Will users start leaving the platform if we show them significantly more ads over a longer period of time? To answer this question, we might want to have a long-running A/B experiment to understand the impact.

The long-running experiment, which measures long-term behaviors, can also be done via a backtest. We can launch the experiment based on initial positive results while continuing to run a long-running backtest to measure any potential long term effects. If we can notice any significant negative behavior, we can revert the changes from the launched experiment.

Experimental framework stages

Embeddings

Let's learn what embeddings are and $\rightarrow$ do we generate them using machine learning techniques.

We'll cover the following

- Embeddings
- Text embeddings
  - Word2vec
  - Context-based embeddings
- Visual embedding
  - Auto-encoders
  - Visual supervised learning tasks
- Learning embeddings for a particular learning task
- Network/Relationship-based embedding

Embeddings

Embeddings enable the encoding of entities (e.g., words, docs, images, persons, ad, etc.) in a low dimensional vector space such that it captures their semantic information. Capturing semantic information helps to identify related entities that occur close to each other in the vector space.

This representation of entities in a lower-dimensional vector space has been of massive help in various ML-based systems. The use of embeddings has seen a major increase because of the recent surge in the use of neural networks and transfer learning.

Usually, they are generated using neural networks. A **neural network** architecture can be set up easily to learn a dense representation of entities. We will go over a few of such architectures later in this lesson.

Transfer learning refers to transferring information from one ML task to another. Embeddings easily enable us to do that for common entities among different tasks. For example, Twitter can build an embedding for their users based on their organic feed interactions and then use the embeddings for ads serving. Organic interactions are generally much greater in volume compared to ads interactions. This allows Twitter to learn user interests by organic feed interaction, capture it as embedding, and use it to serve more relevant ads.

Another simple example is training word embeddings (like Word2vec) from Wiki data and using them as spam-filtering models.

In this lesson, we will go through some general ways of training neural networks to learn embeddings, using real-world example scenarios of their usage.

Text embeddings

We will go over two popular text term embeddings generation models and examples of their utilization in different ML systems.

Word2vec

Word2vec produces word embeddings by using shallow neural networks (having a single hidden layer) and self-supervised learning from a large corpus of text data. Word2vec is self-supervised as it trains a model by predicting words from other words (based on the surrounding context). So, it can utilize tons of text data available in books, Wikipedia, blogs, etc. to learn term representation.

Representing words with a dense vector is critical for the majority of Natural language processing (NLP) tasks. Word2vec uses a simple but powerful idea to use neighboring words to predict the current word and in the process, generates word embeddings. Two networks to generate these embeddings are:

1. **CBOW**: Continuous bag of words (CBOW) tries to predict the current word from its surrounding words by optimizing the following loss function:

$$Loss = -log(p(w_t \mid w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n}))$$

where n is the size of our window to look for the corresponding word. It uses the entire contextual information as one observation while training the network. Utilizing the overall context information to predict one term helps generate embeddings with the smaller training dataset. The architecture would look like this:

Word2vec using CBOW (if window size two, i.e., predicting current word using surrounding words

2. **Skipgram**: In this architecture, we try to predict surrounding words from the current word. The loss function will now look like:

$$Loss = -log(p(w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n} \mid w_t))$$

where n is now the size of surrounding words that we are trying to predict using the current word. In training this network, each context pair will result in a different training example that the model tries to predict. This architecture is most helpful when we have a large training set. The model architecture now looks like:

Word2vec using Skip-gram to predict two surrounding words to current word

Example

Any machine learning task that wants to utilize text terms can benefit from this dense embedding vector, which captures word semantic meanings.

Let's assume that we want to predict whether a user is interested in a particular document given the documents that they have previously read. One simple way of doing this is to represent the user by taking the mean of the Word2vec embeddings of document titles that they have engaged with. Similarly, we can represent the document by the mean of its title term embeddings. We can simply take the dot product of these two vectors and use that in our ML model.

Another way to accomplish this task is to simply pass the user and the document embedding vector to a neural network to help with the learning task.

Context-based embeddings

Once trained, Word2vec embeddings have a fixed vector for every term. So, a Word2vec embedding doesn't consider the context in which the word appears to generate its embedding. However, words in a different context can have very different meanings. For example, consider these two sentences:

1. I'd like to set an apple.
2. Apple makes great products.

Word2vec will give us the same embedding for the term "apple" although it points to completely different objects in the above two sentences.

So, contextualized information can result in different meanings of the same word, and **context-based embeddings look at neighboring terms at embedding generation time**. This means that we have to provide contextual information (neighboring terms) to fetch embeddings for a term. In a Word2vec case, we don't need any context information at the embeddings fetch time as embedding for each term was fixed.

Two popular architectures used to generate word context-based embedding are:

1. Embeddings from Language Models (ELMo)
2. Bidirectional Encoder Representations from Transformers (BERT)

The idea behind **ELMO** is to use the bi-directional LSTM model to capture the words that appear before and after the current word.

BERT uses an attention mechanism and is able to see all the words in the context, utilizing only the ones (i.e., pay more attention) which help with the prediction.

We will have an in-depth explanation of these models and architectures in the entity linking chapter. Entity recognition and linking are one of the great examples of how contextual embeddings can be very effective, we will discuss this in detail in that chapter.

Visual embedding

Let's discuss a couple of interesting ways to generate image embedding.

Auto-encoders

Auto-encoders use neural networks consisting of both an encoder and a decoder. They first learn to compress the raw image pixel data to a small dimension via an encoder model and then try to de-compress it via a decoder to re-generate the same input image. The last layer of encoder determines the dimension of the embedding, which should be sufficiently large to capture enough information about the image so that the decoder can decode it.

The combined encoder and decoder tries to minimize the difference between original and generated pixels, using backpropagation to train the network. The network will look like the following:

Auto encoder model architecture for generating visual embeddings.

Once we have trained the model, we only use the encoder (first N network layers) to generate embeddings for images.

Auto-encoders are also an example of self-supervised learning, like Word2vec, as we can use an image data set without any label to train the model and generate image embeddings.

Visual supervised learning tasks

Visual supervised learning tasks such as image classification or object detection, are generally set up as convolution layers, pooling layers, and fully connected network layers followed by final classification/softmax layers. Let's consider the example of the ImageNet VGG16 *model* that is shown in the figure below. The input passes through a set of convolution, pooling, and fully connected layers to the last softmax layer for the final classification task. The penultimate layer before softmax captures all image information in a vector such that it can be used to classify the image correctly. So, we can use the penultimate layer value of a pre-trained model as our image embedding.

VGG16 architecture

An **example** of image embedding usage could be to find images similar to a given image.

Another example is an image search problem where we want to find the best images for given text terms, e.g. query "cat images". In this case, image embedding along with query term embeddings can help refine search relevance models.

Learning embeddings for a particular learning task

Most of our discussion so far has been about training a general entity embedding that can be used for any learning task. However, we can also embed an entity as part of our learning task. The advantage of this embedding is a specialized one for the given prediction task. One important assumption here is that we have enough training data to be able to learn such representation during model training. Another consideration is that training time for learning the embedding as part of the task will be much higher compared to utilizing a pre-trained embedding.

Let's consider an example where we are trying to predict whether a user will watch a particular movie based on their historical interactions. Here, utilizing movies that the user has previously watched as well as their prior search terms can be very beneficial to the learning task. We can do this by embedding sparse vector of movies and terms in the network itself, as shown in the image below. We will also discuss this specialized embedding usage in detail in the recommendation system chapter.

Please note that each layer is fully connected with the next one. Showing a single arrow because to have cleaner representation.

Network/Relationship-based embedding

Most of the systems have multiple entities, and these entities interact with each other. For example, Pinterest has users that interact with Pins, YouTube has users that interact with videos, Twitter has users that interact with tweets, and Google search has both queries and users that interact with web results.

We can think of these interactions as relationships in a graph or resulting in interaction pairs. For the above example, these pairs would look like:

1. (User, Pin) for Pinterest
2. (User, Video) for YouTube
3. (User, Tweet) for Twitter
4. (Query, Webpage) for Search
5. (Searcher, Webpage) for Search

In all the above scenarios, the retrieval and ranking of results for a particular user (or query) are mostly about predicting how close they are. Therefore, having an embedding model that projects these documents in the same embedding space can vastly help in the retrieval and ranking tasks of recommendation, search, feed-based, and many other ML systems.

We can generate embeddings for both the above-discussed pairs of entities in the same space by creating a two-tower neural network model that tries to encode each item using their raw features. The model optimizes the same product loss such that positive pairs from entity interactions have a higher score and random pairs have a lower score. Let's say the selected pairs of entities (from a group or based on interactions) belong to set A. We then select random pairs for negative examples. The loss function will look like

$$Loss = max(\sum_{(u,i) \in A} dot(u,v) - \sum_{(u,i) \in A} dot(u,v))$$

Two tower model to optimize inner product loss for user and item embeddings

Transfer Learning

An introduction to transfer learning, its importance, and applications.

We'll cover the following

- What is transfer learning?
  - Motivation
- Techniques for transfer learning utilization
- Applications
  - Computer vision problems
  - Natural language processing(NLP)

What is transfer learning?

Transfer learning is the task of using a pre-trained model and applying it to a new task, i.e., transferring the knowledge learned from one task to another. This is useful because the model doesn't have to learn from scratch and can achieve higher accuracy in less time as compared to models that don't use transfer learning.

Motivation

The use of transfer learning in the machine learning domain has surged in the last few years. The following are the top reasons:

Motivation for transfer learning

1. **Growth in the ML community and knowledge sharing**: The research and investments by top universities and tech companies have grown exponentially in the last few years and there is also a strong desire to share state-of-the-art models and datasets with the community. This allows people to utilize pre-trained models in a specific area bootstrap quickly.
2. **Common sub-problems**: Another key motivator is that many problems share common sub-problems, e.g., in all visual understanding and predict areas, tasks such as finding edges, boundaries, and background are common sub-problems. Similarly, in the text domain, the semantic understanding of textual terms can be helpful in almost all problems where the user is represented by text terms, including search, recommendation systems, ads, etc.
3. **Limited supervised learning data and training resources**: Many real-world applications are still mapped onto supervised learning problems where the model is asked to predict a label. One key problem is the limited amount of training data available for models to generalize well. One key advantage of doing transfer learning is that we have the ability to start learning from pre-trained models, and hence, we can utilize the knowledge from similar domains.

   Self-supervised learning models are able to utilize massive available datasets for text and image representation, e.g., Word2vec embeddings models don't need any manual labels and can use the books and Wikipedia data to build a semantic understanding of terms effectively. Once we train a model for a certain representation, it can be utilized and help in many other supervised learning tasks.

   Transfer learning also optimizes training resources, and it helps teams that don't have massive computing resources available. For instance, Google can train a BERT model on billions of examples with its massive computing power, but others are going to find it challenging to train similar optimized models. With transfer learning, we don't have to reinvest those resources and can just plug in the output of the BERT model or use it as a sub-model in our training process. We discussed this concept in detail earlier in our embedding lesson discussion.

Techniques for transfer learning utilization

The transfer learning technique can be utilized in the following ways:

- **Extract features from useful layers**

  Keep the initial layers of the pre-trained model and remove the final layers. Add the new layer to the remaining chunk and train them for final classification.
- **Fine-tuning**

  Change or tune the existing parameters in a pre-trained network, i.e., optimizing the model parameters during training for the supervised prediction task. A key question with fine-tuning the model is to see how many layers can we freeze and how many final layers we want to fine-tune. This requires understanding the network structure of the model and role of each layer, e.g., for the image classification model we used in the Image data example, once we understand the convolution, pooling, and fully connected layers, we can decide how many final layers we need to fine-tune for our model training process.

Transfer learning technique can be utilized in one or both of the above ways depending on the following two factors:

1. **Size of our supervised training dataset**

   How much labeled data do we possess to optimize the model? Do we have 100k examples, 1 million examples, 10 million examples? This is an important question for deciding on the approach that we want to use in utilizing the pre-trained model.

   Training data is limited: In case of a limited amount of specialized training data, we can either go with the approach of freezing all the layers and using the pre-trained model for feature generation or fine-tuning only the final layers.

   Training data is plenty: If we have a significant amount of training data (e.g. one million+ examples), we have the choice to play around with multiple ideas. We can start with just freezing the model, fine-tuning only final layers, or we can retrain the whole model to adjust weights for our specialized task.
2. **Similarity of prediction tasks**

   The similarity of learning tasks can also guide us on whether we can simply use the model as it is or need to fine-tune the model for our new prediction task. For example, if we built a classifier for cars and now we want to use it for trucks, there is a good chance that many of the features are going to be common and we don't have to fine-tune much. Here, we can utilize the pre-trained model as it is and build our models on top of it (i.e., utilizing the output of pre-trained models as features).

Applications

Computer vision problems

Let's go over an example problem where we are trying to build a classifier for medical imaging data and we have 100k manual labelled examples for training our model. Given significant amount of research done in ImageNet data classifier, we can pick one pre-trained ImageNet classifier and start building on top of it.

VGG16 architecture

The convolutional filters in a trained convolutional neural network (CNN) are arranged in a kind of hierarchy. The filters in the first layer often detect edges or blocks of color. The second layer's filters can detect features like shapes. All of them are very general features that are utilized in analyzing any image in any dataset. The filters in the last layers are more specific. Let's go over all of the freezing layers, fine-tuning a few layers, and fine-tuning the whole model scenarios:

- **Case 1: Fine-tuning a few layers**

  If our prediction task is similar, there will be similar higher-level features or layers output. Therefore most or all of the pre-trained model layers already have relevant information about the new data set and should be kept. We will *freeze the weight of most of the starting layers* of the pre-trained model and fine-tune only the end layers.

  This approach will also be most viable if our labeled training data is limited as it would be hard to re-tune all layers based on that limited data set.

Fine-tuning a few layers

- **Case 2: Fine tuning more layers**

  If we have significant amount of labelled examples and our learning tasks have commonalities but few differences as well, it would make sense to go deeper in fine tuning our pre-trained model. We will *freeze the weights of the first few layers* and fine-tune the weights of the remaining end layers to optimize the model for our new learning task on medical image dataset.

Fine-tuning most of the end layers

- **Case 3: Fine tuning the entire model**

  If the new data set is larger, then we will load the weights of the pre-trained model and *fine-tune the entire network*. This will definitely increase our training time as well but should help us optimize our learning task when we have significant training data.

Fine tuning the entire model

> More details of transfer learning using image data are explained while modeling the solution for the image segmentation problem.

Natural language processing(NLP)

In many of NLP learning tasks such as language understanding, speech recognition, entity recognition, language generation, semantic understanding, etc. as well as other problems that are based on search, one major need is to represent our text terms in a way that they capture the semantic meaning.

For this, we need to generate the *dense representation* of textual terms. A few of the popular term representation models that use a self-supervised learning approach, trained on massive datasets are word2vec, BERT, and ELMO. The term representation based on these models capture their semantic meanings. Hence, we can transfer knowledge from this learned task in many of the NLP tasks.

Through the transfer learning approach, we can now utilize these embeddings in a NER classifier, spam detector classifier, search ranking, language understanding, etc. and can significantly improve the quality of these ML models.

BERT/ELMo can be as an embedding generator

> More details can be learned about these approaches while modeling the solution for the entity linking problem.

Model Debugging and Testing

An introduction to model debugging and how it is conducted for a machine learning system.

We'll cover the following

- Building model v1
- Deploying and debugging v1 model
  - Change in feature distribution
  - Feature logging issues
  - Overfitting
  - Under-fitting
- Iterative model improvement
  - Missing important feature
  - Insufficient training examples
- Debugging large scale systems

Let's go over different phases in the development of a machine learning system, potential issues that we can face, and how to debug and fix them.

There are two main phases in terms of the development of a model that we will go over:

- Building the first version of the model and the ML system.
- Iterative improvements on top of the first version as well as debugging issues in large scale ML systems.

Building model v1

The goal in this phase is to build the 1st version of the model. Few important steps in this stage are:

- We begin by identifying a business problem in the first phase and mapping it to a machine learning problem.
- We then go onto explore the training data and machine learning techniques that will work best on this problem.
- Then we train the model given the available data and features, play around with hyper parameters.
- Once the model has been set up and we have early offline metrics like accuracy, precision/recall, AUC, etc., we continue to play around with the various features and training data strategies to improve our offline metrics.
- If there is already a heuristics or rule-based system in place, our objective from the offline model would be to perform at least as good as the current system, e.g., for ads prediction problem, we would want our ML model AUC to be better than the current rule-based ads prediction based on only historical engagement rate.

It's important to get version 1 launched to the real system quickly rather than spending too much time trying to optimize it. For example, if our AUC is 0.7 and it's better than the current system with AUC 0.68, it's generally a better idea to take model online and then continue to iterate to improve the quality. The reason is primarily that model improvement is an iterative process and we want validation from real traffic and data along with offline validation. We will look at various ideas that can help in that iterative development in the following sections.

ML Model Workflow

Deploying and debugging v1 model

Some ML-based systems only operate in an offline setting, for example, detect objects from a large set of images. But, most systems have an online component as well, for example, building a search ranking ML model will have to run online to service incoming queries and ranking the documents that match the query.

In our first attempt to take the model online, i.e., enable live traffic, might not work as expected and results don't look as good as we anticipated offline. Let's look at a few failure scenarios that can happen at this stage and how to debug them.

Change in feature distribution

The change in the feature distribution of training and evaluation set can negatively affect the model performance. Let's consider an example of an Entity linking system that is trained using a readily available Wikipedia dataset. As we start using the system for real traffic, the traffic that we are now getting for finding entities is a mix of Wikipedia articles as well as research papers. Given the model wasn't trained on that data, its feature distribution would be a lot different than what it was trained on. Hence it is not performing as well on the research articles entity detection.

Another scenario could be a significant change in incoming traffic because of seasonality. Let's consider an example of a search system trained using data for the last 2 weeks of December, i.e., mostly holiday traffic. If we deploy this system in January, the queries that it will see will be vastly different than what it was trained on and hence not performing as well as we observed in our offline validation.

Entity linking model could not work online due to change in feature distribution

Feature logging issues

When the model is trained offline, there is an assumption that features of the model generated offline would exactly be the same when the model is taken online. However, this might not be true as the way we generated features for our online system might not exactly be the same. It's a common practice to append features offline to our training data for offline training and then add them later to the online model serving part. So, if the model doesn't perform as well as we anticipated online, it would be good to see if feature generation logic is the same for offline training as well as online serving part of model evaluation.

Suppose we build an ads click prediction model. The model is trained on historical ad engagement. We then deploy it to predict the ads engagement rate. Now the assumption is that the features generated for the model offline would exactly be the same for run-time evaluation. Let's assume that we have one important feature/signal for our model that's based on historical advertiser ad impressions. During training, we compute this feature by using the last 7 days' impression. But, the logic to compute this feature at model evaluation time uses the last 30 days of data to compute advertiser impressions. Because of this feature computed differently at training time and evaluation time, it will result in the model not performing well during online serving. It would be worth comparing the features used for training and evaluation to see if there is any such discrepancy.

Ads prediction online metrics drop due to feature computation differences

Overfitting

Overfitting happens when a model learns the intrinsic details in the training data to the extent that it negatively impacts the performance of the model on new or unseen data.

If our model performance is lower in the live system but it still performs well on our training and validation set then there is a good chance that we have overfit our data by trying to improve the performance a bit too much. One good way of ensuring that we don't get into this problem is to use a hidden test set which is not used for tuning hyperparameters and only use it for final model quality measurement.

Another important part is to have a comprehensive and large test set to cover all possible scenarios in a fairly similar distribution to how we anticipate them in live traffic. For example, consider an image object prediction system whose test set only has large-sized objects (covering 50% pixels or more of the image) for 90% samples and small-sized objects for 10% samples (covering 10% pixels or less). If the live traffic has the opposite distribution of large and small size objects, then the model might not perform well on the live set.

Overfit and optimal model

Under-fitting

One indication from training the model could be that the model is unable to learn complex feature interactions especially if we are using a simplistic model. So, this might indicate to us that using slightly higher-order features, introduce more feature interactions, or use a more complex /expressive model such as a neural network.

Neural network outperforms linear regression model because it can capture non-linear relationship between features

Iterative model improvement

As discussed above that the first version of the deployed model is generally not the most optimized one as our goal is to get the ML system out sooner to get feedback. Afterward, *Model debugging and testing* is generally done to come up with ideas to improve the quality of the model resulting in our metrics improvement.

Few cases that we discussed above regarding overfitting and under-fitting are still the questions that we should continue to ask during iterative model improvement but let's discuss a few more below.

The best way to iterative improve model quality is to start looking at failure cases of our model prediction and using that come up with the ideas that will help in improving model performance in those cases.

Let's go over some common types of ideas that will emerge as part of these failures examples debugging process.

> Debugging in current context doesn't mean to optimize overall architecture and system design that we have deeply discussed in this course like how to set up a problem, setting up architecture etc. It comprises of general methods that we use to observe issues in current model to continue to optimize it.

Missing important feature

Digging deeper into failures examples can identify missing features that can help us perform better in failures cases, e.g., consider a scenario where a movie actually liked by the user was ranked very low by our recommendation system. On debugging, we figure out that the user has previously watched two movies by the same actor, so adding a feature on previous ratings by the user for this movie actor can help our model perform better in this case.

Insufficient training examples

We may also find that we are lacking training examples in cases where the model isn't performing well. We will cater to all possible scenarios where the model is not performing well and update the training data accordingly. For example, for the image segmentation problem, the segmentation model is not able to segment the image when there are multiple traffic lights. One way to look at that point will be to count the number of such training examples that we have in our data set with multiple traffic lights.

So we will add more training examples of multiple traffic lights and enable the model to learn to segment multiple traffic lights better. The following illustration shows the above concept by taking an example of a test image from cityscapes dataset.

Image segmentation model output performance increased in case of sufficient training data

Debugging large scale systems

In the case of debugging large scale systems with multiple componentsor models, we need to see which part of the overall system is not working correctly. It could be done for one failure example or over a set of examples to see where the opportunity lies to improve the metrics.

The following are a few key steps to think about iterative model improvement for large scale end to end ML systems:

- **Identify the component**
  This accounts for finding the architectural component resulting in a high number of failures in our failure set. Suppose that in the case of the search ranking system, we have opted for a layered model approach that we will discuss in detail in our Search problem discussion.
  In order to see the cause of failure, we will look at each layers' performance to understand the opportunity to significantly improve the quality of our search system. Let's assume that our search system has two key components 1) *Document selection* 2) *Ranking* of selected documents. Document selection focus is to ensure that all the top relevant documents get selected for the query while Ranker then ensures that our rank order is correct based on the relevance of the top 100 documents.
  If we look at few hundred failures of our overall search system, we should be able to identify the component that's resulting in more failures and as a result, decide to improve the quality of that component, e.g., if 90% of our overall search system failures are because of the ideal document not being selected in candidate selection component, we will debug the model deeply in that layer to see how to improve the quality of that model. Similarly, if ideal documents are mostly selected but are ranked lower by our ranking component, we will invest in finding the reason for failures in that layer and improve the quality of our ranking model.
- **Improve the quality of component**
  Some of the model improvement methods that we have discussed above like adding more training data, features, modeling approach in case of overfitting and underfitting will still be the same once we identify the component that needs work, e.g., if we identify that the candidate selection layer needs improvement in our search, we will try to see missing features, add more training data or play around with ML model parameters or try a new model.

Problem Statement

Let's look at a problem statement that asks you to design a search relevance system for a search engine.

We'll cover the following

- Problem statement
- Clarifying questions
  - Problem scope
  - Scale
  - Personalization

Problem statement

The interviewer has asked you to design a search relevance system for a search engine.

Design a search relevance system to display results on a search engine result page

Clarifying questions

Let's clarify the problem statement by specifying three aspects: scope, scale, and personalization.

Problem scope

The interviewer's question is really broad. Your best bet is to avoid ambiguities and ask the interviewer as many questions as you can. This will narrow down your problem space as you are thinking out loud for the best solution.

So, your first question for the interviewer would be something like the following:

Is it a general search engine like Google or Bing or a specialized search engine like Amazon products search?

This scoping down of the problem is critical as you dive into finding the solutions. For the sake of this chapter, we will assume that you are working towards finding relevant results using a general search engine like Google search or Bing search, but the techniques and discussion apply to all search engines.

Finally, the problem statement can be precisely described as:

Build a generic search engine that returns relevant results for queries like "Richard Nixon", "Programming languages" etc.

This will require you to build a machine learning system that provides the most relevant results for a search query by ranking them in order of relevance. Therefore, you will be working on the *search ranking problem*.

> 📝 The approach to handle a specialized search engine will be fairly similar.

Scale

Once you know that you are building a general search engine, it's also critical to determine the scale of the system. A couple of important questions are:

- How many websites exist that you want to enable through this search engine?
- How many requests per second do you anticipate to handle?

We will assume that you have billions of documents to search from, and the search engine is getting around 10K *queries per second* (QPS).

Understanding this scale is important to architect our relevance system. For example, later in the chapter, we will go over the funnel-based approach where you will continue to increase model complexity and reduce document set, as you go down the funnel for this large scale search system.

Personalization

Another important question that you want to address is whether the *searcher is a logged-in user or not*. This will define the level of personalization that you can incorporate to improve the relevance of our results. You will assume that the user is logged in and you have access to their profile as well as their historical search data.

Metrics

Let's explore some metrics that will help you define the "success" scenarios for a search session.

We'll cover the following

- Online metrics
  - Click-through rate
  - Successful session rate
  - Caveat
- Offline metrics
  - NDCG
  - Caveat

Choosing a metric for a machine learning model is of paramount importance. Machine learning models learn directly from the data, and no human intuition is encoded into the model. Hence, selecting the wrong metric results in the model becoming optimized for a completely wrong criterion.

There are two types of metrics to evaluate the success of a search query:

1. Online metrics
2. Offline metrics

We refer to metrics that are computed as part of user interaction in a live system as Online metrics. Meanwhile, offline metrics use offline data to measure the quality of your search engine and don't rely on getting direct feedback from the users of the system.

Online metrics

In an online setting, you can base the success of a search session on *user actions*. On a *per-query level*, you can define success as the user action of *clicking on a result*.

A simple click-based metric is *click-through rate*.

Click-through rate

The click-through rate measures the ratio of clicks to impressions.

> Click through rate = $\frac{Number of clicks}{Number of impressions}$

In the above definition, an *impression* means a view. For example, when a search engine result page loads and the user has seen the result, you will consider that as an impression. A click on that result is your success.

Successful session rate

One problem with the *click-through rate* could be that unsuccessful clicks will also be counted towards search success. For example, this might include short clicks where the searcher only looked at the resultant document and clicked back immediately. You could solve this issue by filtering your data to only successful clicks, i.e., to only consider clicks that have a long dwell time.

> *Dwell time* is the length of time a searcher spends viewing a webpage after they've clicked a link on a search engine result page (**SERP**).

Therefore, successful sessions can be defined as the ones that have a click with a ten-second or longer dwell time.

> Session success rate = $\frac{no. of successful sessions}{no. of total sessions}$

A session can also be successful without a click as explained next.

Caveat

Another aspect to consider is *zero-click searches*.

> **Zero-click searches**: A SERP may answer the searcher's query right at the top such that the searcher doesn't need any further clicks to complete the search.

For example, a searcher queries "einstein's age", and the SERP shows an excerpt from a website in response, as shown below:

The searcher has found what they were looking for without a single click!. The click-through rate would not work in this case (but your definition of a successful session should definitely include it). We can fix this using a simple technique shown below.

Time to success

Until now, we have been considering a single query-based search session. However, it may span *over several queries*. For example, the searcher initially queries: "italian food". They find that the results are not what they are looking for and make a more specific query: "italian restaurants". Also, at times, the searcher might have to go over multiple results to find the one that they are looking for.

Ideally, you want the searcher to go to the result that answers their question in the minimal number of queries and as high on the results page as possible. So, *time to success* is an important metric to track and measure search engine success.

> **Note**: For scenarios like this, a *low number of queries per session* means that your system was good at guessing what the searcher actually wanted despite their poorly worded query. So, in this case, we should consider a *low number of queries per session* in your definition of a successful search session.

Offline metrics

The offline methods to measure a successful search session makes use of trained human raters. They are asked to rate the relevance of the query results objectively, keeping in view well-defined guidelines. These ratings are then aggregated across a query sample to serve as the *ground truth*.

> Ground truth refers to the actual output that is desired of the system. In this case, it is the ranking or rating information provided by the human raters.

Let's see *normalized discounted cumulative gain* (NDCG) in detail as it's a critical evaluation metric for any ranking problem.

NDCG

You will be looking at NDCG as a common measure of the quality of search ranking results.

NDCG is an improvement on *cumulative gain* (CG).

> $CG_p = \sum_{i=1}^{p} rel_i$
>
> *where rel = relevance rating of a document, i = position of document, and p = the position up to which the documents are considered*

Example

Let's look at this concept using an example.

A search engine answers a query with the documents D_1 to D_4. It ranks them in the following order of decreasing relevance, i.e., D_1 is the highest-ranked and D_4 is the lowest-ranked:

Ranking by search engine: D_1, D_2, D_3, D_4

A human rater is shown this result and asked to judge the relevance of each document for the query on a scale of $0 - 3$ (3 indicating highly relevant). They rank the documents as follows:

Documents	D_1	D_2	D_3	D_4
Ranking/Rating by human rater	3	2	3	0

The *cumulative gain* for the search engine's result ranking is computed by simply adding each document's relevance rating provided by the human rater.

$$CG_4 = \sum_{i=1}^{4} rel_i = 3 + 2 + 3 + 0 = 8$$

In contrast to *cumulative gain*, discounted cumulative gain (DCG) allows us to *penalize the search engine's ranking if highly relevant documents (as per ground truth) appear lower in the result list*.

> DCG discounts are based on the position of the document in human-rated data. The intuition behind it is that the search engine will not be of much use if it doesn't show the most relevant documents at the top of search result pages. For example, showing starbucks.com at a lower position for the query "*Starbucks*", would not be very useful.

> $DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$

DCG for the search engine's ranking is calculated as follows:

$$DCG_4 = \sum_{i=1}^{4} \frac{rel_i}{\log_2(i+1)} = 3 + 1.262 + 1.5 + 0 = 5.762$$

In the above calculation, you can observe that the denominator penalizes the search engine's ranking of D_3 for appearing later in the list. It was a more relevant document relative to D_2 and should have appeared earlier in the search engine's ranking.

However, DCG can't be used to compare the search engine's performance across different queries on an absolute scale. The is because the length of the result list varies from query to query. So, the DCG for a query with a longer result list may be higher due to its length instead of its quality. To remedy this, you need to move towards NDCG.

NDCG normalizes the DCG in the 0 to 1 score range by dividing the DCG by the max DCG or the IDCG (ideal discounted cumulative gain) of the query. IDCG is the DCG of an ideal ordering of the search engine's result list (you'll see more on IDCG later).

NDCG is computed as follows:

> $NDCG_p = \frac{DCG_p}{IDCG_p}$ *where IDCG is ideal discounted cumulative gain*

In order to compute IDCG, you find an ideal ordering of the search engine's result list. This is done by rearranging the search engine's results based on the ranking provided by the human raters, as shown below.

Finding the ideal ordering of search engine's result based on ground truth

Now let's calculate the DCG of the ideal ordering (also known as the IDCG) as shown below:

$IDCG_4 = 5.898$

Now, let's finally compute the NDCG:

NDCG = $\frac{DCG}{IDCG} = \frac{5.762}{5.898} = 0.976$

An NDCG value near one indicates good performance by the search engine. Whereas, a value near 0, indicates poor performance.

To compute NDCG for the overall query set with N queries, we take the mean of the respective NDCGs of all the N queries, and that's the overall relevance as per human ratings of the ranking system.

> NDCG = $\frac{\sum_{i=1}^{N} NDCG_i}{N}$

Caveat

NDCG does not penalize irrelevant search results. In our case, it didn't penalize D_4, which had zero relevance according to the human rater.

Another result set may not include D_3, but it would still have the same NDCG score. As a remedy, the human rater could assign a negative relevance score to that document.