Table of Contents

Introduction to Machine Learning Interviews Book

You can read the web-friendly version of the book here. You can find the source code on GitHub.

As a candidate, I’ve interviewed at a dozen big companies and startups. I’ve got offers for machine learning roles at companies including Google, NVIDIA, Snap, Netflix, Primer AI, and Snorkel AI. I’ve also been rejected at many other companies.

As an interviewer, I’ve been involved in designing and executing the hiring process at NVIDIA and Snorkel AI, having taken steps from cold emailing candidates whose work I love, screening resumes, doing exploratory and technical interviews, debating whether or not to hire a candidate, to trying to convince candidates to choose us over competitive offers.

As a friend and teacher, I’ve helped many friends and students prepare for their machine learning interviews at big companies and startups. I give them mock interviews and take notes of the process they went through as well as the questions they were asked.

I’ve also consulted several startups on their machine learning hiring pipelines. Hiring for machine learning roles turned out to be pretty difficult when you don’t already have a strong in-house machine learning team and process to help you evaluate candidates. As the use of machine learning in the industry is still pretty new, a lot of companies are still making it up as they go along, which doesn’t make it easier for candidates.

This book is the result of the collective wisdom of many people who have sat on both sides of the table and who have spent a lot of time thinking about the hiring process. It was written with candidates in mind, but hiring managers who saw the early drafts told me that they found it helpful to learn how other companies are hiring, and to rethink their own process.

The book consists of two parts. The first part provides an overview of the machine learning interview process, what types of machine learning roles are available, what skills each role requires, what kinds of questions are often asked, and how to prepare for them. This part also explains the interviewers’ mindset and what kind of signals they look for.

The second part consists of over 200 knowledge questions, each noted with its level of difficulty -- interviews for more senior roles should expect harder questions -- that cover important concepts and common misconceptions in machine learning.

After you've finished this book, you might want to checkout the 30 open-ended questions to test your ability to put together what you know to solve practical challenges. These questions test your problem-solving skills as well as the extent of your experiences in implementing and deploying machine learning models.

Some companies call them **machine learning systems design** questions. Almost all companies I've talked to ask at least a question of this type in their interview process, and they are the questions that candidates often find to be the hardest.

"Machine learning systems design" is an intricate topic that merits its own book. To learn more about it, check out my course **CS 329S: Machine learning systems design** at Stanford.

This book is not a replacement to machine learning textbooks nor a shortcut to game the interviews. It's a tool to consolidate your existing theoretical and practical knowledge in machine learning. The questions in this book can also help identify your blind/weak spots. Each topic is accompanied by resources that should help you strengthen your understanding of that topic.

Target audience

If you’ve picked up this book because you’re interested in working with one of the key emerging technologies of the 2020s but not sure where to start, you’re in the right place. Whether you want to become an ML engineer, a platform engineer, a research scientist, or you want to do ML but don’t yet know the differences among those titles, I hope that this book will give you some useful pointers.

This book focuses more on roles involving machine learning production than research, not because I believe production is more important. As fewer and fewer companies can afford to pursue pure research whereas more and more companies want to adopt machine learning, there will be, and already are, vastly more roles involving production than research.

This book was written with two main groups of candidates in mind:

1. Recent graduates looking for their first full-time jobs.
2. Software engineers and data scientists who want to transition into machine learning.

I imagine the majority of readers of this book come from a computer science background. The second part of the book, where the questions are, is fairly technical. However, as machine learning finds its use in more industries -- healthcare, farming, trucking, fashion, you name it -- the field needs more people with diverse interests. If you’re interested in machine learning but hesitant to pursue it because you don’t have an engineering degree, I strongly encourage you to explore it. This book, especially the first part, might address some of your needs. After all, I only took an interest in matrix manipulation after working as a writer for almost a decade.

About the questions

The questions in this book were selected out of thousands of questions, most have been asked in actual interviews for machine learning roles. You will find several questions that are technically incorrect or ambiguous. This is on purpose. Sometimes, interviewers ask these questions to see whether candidates will correct them, point out the edge cases, or ask for clarification. For these questions, the accompanying hints should help clarify the ambiguity or technical incorrectness.

Machine learning is a tool, and to effectively use any tool, we should know how, why, or when to use it on top of knowing what it is. Because the “what” questions can be easily found online, and if something can be easily acquired, it isn’t worth testing for. This book focuses on the “how”, “why”, and “when” questions. For example, instead of asking for the exact algorithm for K-means clustering, the question asks in what scenarios K-means doesn’t work. You don’t need to understand K-means to cite its definition, but you do to know when not to use it.

Still, this book contains a small number of “what” questions. While they aren’t good interview questions, they are good for interview preparation.

About the answers

I started the book with the naive optimism that I'd write the answers for every question in this book. It turned out writing detailed answers to 300+ technical questions -- while juggling a full-time job, a teaching gig, a raging pandemic with a couple of family members in the hospital -- is a lot.

Given the slow progress I've been making, I've decided that publishing the draft then continuing writing/crowdsourcing the answers might be more productive. The first draft of the book has the answers for about 10% of all the questions. These existing answers, while I believe to be sufficiently mathematically rigorous, prioritize intuitive explanations over mathematical equations. The answers for the questions will be continually updated. Answer contribution via the book's GitHub repository is much appreciated.

Hiring is a process, and questions aren't evaluated in isolation. Your answer to each question is evaluated as part of your performance during the entire process. A candidate who claims to work with computer vision and fails to answer a question about techniques typically used for computer vision tasks is going to be evaluated differently from a candidate who doesn't work with computer vision at all. Interviewers often care more about your approach than the actual objective correctness of your answers.

Gaming the interview process

People often ask me: “Don’t you worry that candidates will just memorize the answers in this book and game the system?”

First, I don’t encourage interviewers to ask the exact questions in this book, but I hope this book provides a framework for interviewers to distinguish good questions from bad ones.

Second, there’s nothing wrong with memorizing something as long as that memorization is useful. The problem begins when memorization is impractical -- candidates memorize something to pass the interviews and never use that knowledge again, or don’t know how to use it in real situations.

For this book, I aimed to include only concepts that I and many of my helpful colleagues deemed practical. For every concept, I ask: “Where in the real world is it used?” If I can’t find a good answer after extensive research, the concept is discarded. For example, while I chose to include questions about inner product and outer product, I left out cross product. You can see the list of discarded questions in the list of “Bad questions” on the book's GitHub repository. This is far from a foolproof process. As the field expands, concepts that aren’t applicable now might be all that AI researchers ever talk about in 2030.

Interviews are stressful, even more so when they are for your dream job. As someone who has been in your shoes, and might again be in your shoes in the future, I just want to tell you that it doesn’t have to be so bad. Each interview is a learning experience. An offer is great, but a rejection isn’t necessarily a bad thing and is never the end of the world.

There are many random variables that influence the outcome of an interview: the questions asked, other candidates the interviewer has seen before you, after you, the interviewer’s expectation, even the interviewer’s mood. It is, in no way, a reflection of your ability or your self-worth.

I was pretty much rejected for every job I applied to when I began this process. Now I just get rejected at a less frequent rate. Keep on learning and improving. You’ve got this!

Acknowledgments

This work wouldn't be possible without the generous help of many friends and colleagues, listed in no particular order. If I've forgotten to include your name, it's not because your contribution didn't matter, but because my memory is faulty. If that's the case, please let me know and I'll try to rectify that!

All the glory belongs to those who made it better. All the mistakes belong to me.

Ben Krause (a hero I don't deserve), Joshua Levy (whose invaluable insights helped me restructure the entire first part), Luke Metz (my brave first reader), Boris Ginsburg, Oleksii Kuchaiev, Raine Hoover, Akshay Agrawal, Hugo Valdivia, Dat Tran, Nick Pawlowski, David Bourgin, Kyle Kranen, Mike Houston, Yang Zhang, Adeh DeSandies, Marko Vasic, Karson Elmgren, Andrej Karpathy.

About the author

Chip Huyen (https://huyenchip.com) is a best-selling author and engineer who develops tools and best practices for bringing AI research into production. Through her work at Netflix, NVIDIA, and Snorkel AI, she has helped some of the world's largest organizations develop and deploy machine learning systems.

In 2017, she created and taught the Stanford course TensorFlow for Deep Learning Research. In 2021, she created and taught Machine Learning Systems Design, also at Stanford.

She is also the author of four Vietnamese books that have sold more than 100,000 copies. The first two books belong to the series *Xách ba lô lên và Đi* (Quảng Văn 2012, 2013). The first book in the series was the #1 best-selling book of 2012 on Tiki.vn. The series was among FAHASA's Top 10 Readers Choice Books in 2014.

Chip's expertise is in the intersection of software engineering and machine learning. LinkedIn included her among the 10 Top Voices in Software Development in 2019, and Top Voices in Data Science & AI in 2020.

Disclaimer

This book doesn't reflect the view of any of my previous or current employers, or any of the organizations that I'm associated with.

Part I. Overview

Chapter 1. Machine learning jobs

Before embarking on a journey to find a job, it might be helpful to know what types of jobs there are. These jobs vary wildly from company to company based on their focus, customer profile, and stage, so in the second part of this chapter, we will go over different types of companies.

1.1 Different machine learning roles

Some of the roles we'll look into in this chapter:

- Machine learning engineer
- Data scientist
- ML/AI platform engineer
- ML/AI infrastructure engineer
- Framework engineer
- Solution architect
- Developer advocate
- Solutions engineer
- Applications engineer
- Applied research scientist
- Research engineer
- Research scientist

1.1.1 Working in research vs. working in production

*I use research vs. production instead of academia vs. industry because even though academia is mostly concerned with research, research isn't mostly done in academia. In fact, ML research nowadays is spearheaded by big corporations. See **1.1.2 Research** for more details.*

The first question you might want to figure out is whether you want to work in research or in production. They have very different job descriptions, requirements, hiring processes, and compensations.

The goal of research is to find the answers to fundamental questions and expand the body of theoretical knowledge. A research project usually involves using scientific methods to validate whether a hypothesis or a theory is true, without worrying about the practicality of the results.

The goal of production is to create or enhance a product. A product can be a good (e.g. a car), a service (e.g. ride-sharing service), a process (e.g. detecting whether a transaction is fraudulent), or a business insight (e.g. "to maximize profit we should increase our price 10%").

A research project doesn't need users, but a product does. For a product to be useful, it has many more requirements other than just performance, such as inference latency, interpretability (both to users and to developers), fairness (to all subgroups of users), adaptability to changing environment. The majority of a production team's job might be to ensure these other requirements.

The given definitions above are, of course, handwavy at best. What's research and what's production in machine learning remain a heated topic of debate as of 2021[3]. One reason for the ambiguity is that novel ideas with obvious usefulness tend to attract more researchers, and solving practical problems often requires coming up with novel ideas.

For more differences between machine learning in research and in production, see Stanford's CS 329S, lecture 1: Understanding machine learning production.

As a candidate, if you're unfamiliar with both and not sure whether you want to find roles in research or in production, the latter might be the smoother path. There are many more roles involving production than roles involving research.

[3]: One example is the argument whether GPT-3 is research. Many researchers were upset when Language Models are Few-Shot Learners (OpenAI, 2020) was awarded the best paper at NeurIPS because they didn't consider it research.

1.1.2 Research

As the research community takes the “bigger, better” approach, new models often require a massive amount of data and tens of millions of dollars in computing. The estimated market cost to train DeepMind’s AlphaStar and OpenAI’s GPT-3 is in the tens of millions each[4][5]. Most companies and academic institutions can’t afford to pursue pure research.

Outside academic institutions, there are only a handful of machine learning research labs in the world. Most of these labs are funded by corporations with deep pockets such as Alphabet (Google Brain, DeepMind), Microsoft, Facebook, Tencent[6]. You can find these labs by browsing the affiliations of published papers at major academic conferences including NeurIPS, ICLR, ICML, CVPR, ACL). In 2019 and 2020, Alphabet accounts for over 10% of all papers at NeurIPS[1, 2].

> 🌳 **Tip** 🌳
> Not all these industry labs publish papers -- companies like Apple and Tesla are notoriously secretive. Even if an industry lab publishes, it might only publish a portion if its research. Before joining an industry lab, you might want to consider its publishing policy. Joining a secretive lab might mean that you won’t be able to explain to other people what you’ve been working on or what you’re capable of doing.

[4]: State of AI Report 2019 by Nathan Benaich and Ian Hogarth.

[5]: OpenAI’s massive GPT-3 model is impressive, but size isn’t everything by VentureBeat.

[6]: In an earlier draft of this book, I included Uber AI and Element AI. However, Uber AI research lab was laid off in April 2020, and Element AI was sold for cheap in November 2020.

1.1.2.1 Research vs. applied research

At some companies, you might encounter roles involving applied research. Applied research is somewhere between research and production, but much closer to research than production. Applied research involves finding solutions to practical problems, but doesn’t involve implementing those solutions in actual production environments.

Applied researchers are researchers. They come up with novel hypotheses and theses as well as validate them. However, since their hypotheses and theses deal with practical problems, they need to understand these problems as well. In industry lingo, they need to have **subject matter expertise**.

In machine learning, an example of a research project would be to develop an unsupervised transfer learning method for computer vision, experiment on a standard academic dataset. An example of an applied research project would be to develop techniques to make that new method work on a real-world problem in a specific industry, e.g. healthcare. People working on this applied research project will, therefore, need to have expertise in both machine learning and healthcare.

1.1.2.2 Research scientist vs. research engineer

There’s much confusion about the role of a research engineer. This is a rare role, often seen at major research labs in the industry. Loosely speaking, if the role of a research scientist is to come up with original ideas, the role of a research engineer is to use their engineering skills to set up and run experiments for these ideas. The research scientist role typically requires a Ph.D. and/or first author papers at top-tier conferences. The research engineer role doesn’t, though publishing papers always helps.

For some teams, there’s no difference between a research scientist and a research engineer. Research scientists should, first and foremost, be engineers. Both research scientists and engineers come up with ideas and implement those ideas. A researcher might also act as an advisor guiding research engineers in their own research. It’s not uncommon to see research scientists and research engineers be equal contributors to papers[7]. The different job titles are mainly a product of bureaucracy -- research scientists are supposed to have bigger academic clout and are often better paid than research engineers.

Startups, to attract talents, might be more generous with the job titles. A candidate told me he chose a startup over a FAAAM company because the startup gave him the title of a research scientist, while that big company gave him the title of a research engineer.

Akihiro Matsukawa gave an interesting perspective on the difference between the research scientist and the research engineer with his post: Research Engineering FAQs.

[7]: Notable examples include “*Attention Is All You Need*” from Google and “*Language Models are Unsupervised Multitask Learners*” from OpenAI.

1.1.3 Production

As machine learning finds increasing use in virtually every industry, there's a growing need for people to bring machine learning models into production. In this section, we will first cover the production cycle for machine learning, the skills needed for each step, and the distinctions of several roles that often confuse the candidates I've talked to.

1.1.3.1 Production cycle

To understand different roles involving machine learning in production, let’s first explore different steps in a production cycle. There are six major steps in a production cycle.

> **⚠ On the main skills listed at each step ⚠**
> The main skills listed at each step below will upset many people, as any attempt to simplify a complex, nuanced topic into a few sentences would. This portion should only be used as a reference to get a sense of the skill sets needed for different ML-related jobs.

1. **Project scoping**

   A project starts with scoping the project, laying out goals & objectives, constraints, and evaluation criteria. Stakeholders should be identified and involved. Resources should be estimated and allocated.

   Main skills needed: product management, subject matter expertise to understand problems, some ML knowledge to know what ML can and can’t solve.

2. **Data management**

   Data used and generated by ML systems can be large and diverse, which requires scalable infrastructure to process and access it fast and reliably. Data management covers data sources, data formats, data processing, data control, data storage, etc.

   Main skills needed: databases/query engines to know how to store/retrieve/process data, systems engineering to implement distributed systems to process large amounts of data, minimal ML knowledge to optimize the organization data for ML access patterns would be helpful, but not required.

3. **ML model development**

   From raw data, you need to create training datasets and possibly label them, then generate features, train models, optimize models, and evaluate them. This is the stage that requires the most ML knowledge and is most often covered in ML courses.

   Main skills needed: This is the part of the process that requires the most amount of ML knowledge, statistics and probability to understand the data and evaluate models. Since feature engineering and model development require writing code, this part needs coding skills, especially in algorithms and data structures.

4. **Deployment**

   After a model is developed, it needs to be made accessible to users.

Main skills needed: Bringing an ML model to users is largely an infrastructure problem: how to set up your infrastructures or help your customers set up their infrastructures to run your ML application. These applications are often data-, memory-, and compute-intensive. It might also require ML to compress ML models and optimize inference latency unless you can push these to the previous step of the process.

5. **Monitoring and maintenance**

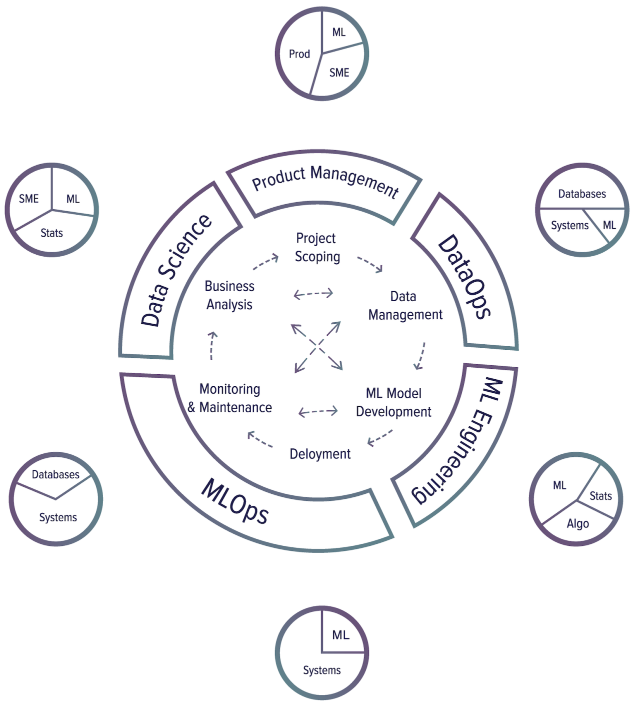
   Once in production, models need to be monitored for performance decay and maintained/updated to be adaptive to changing environments and changing requirements.

   Main skills needed: Monitoring and maintenance is also an infrastructure problem that requires computer systems knowledge. Monitoring often requires generating and tracking a large amount of system-generated data (e.g. logs), and managing this data requires an understanding of the data pipeline.

6. **Business analysis**

   Model performance needs to be evaluated against business goals and analyzed to generate business insights. These insights can then be used to eliminate unproductive projects or scope out new projects.

   Main skills needed: This part of the process requires ML knowledge to interpret ML model's outputs and behavior, in-depth statistics and probability knowledge to extract insights from data, as well as subject matter expertise to map these insights to the practical problems the ML models are supposed to solve.



Skill annotation

- **Systems**: system engineering e.g. to building distributed systems, container deployment.
- **Databases**: data management, storage, processing, databases, query engines. This is closely related to **Systems** since you might need to build distributed systems to process large amounts of data.
- **ML**: linear algebras, ML algorithms, etc.
- **Algo**: algorithmic coding
- **Stats**: probability, statistics
- **SME**: subjective matter expertise
- **Prod**: product management

The most successful approach to ML production I've seen in the industry is **iterative and incremental development**. It means that you can't really be done with a step, move to the next, and never come back to it again. There's a lot of back and forth among various steps.

Here is one common workflow that you might encounter when building an ML model to predict whether an ad should be shown when users enter a search query[8].

[8]: Praying and crying not featured but present through the entire process.

1. Choose a metric to optimize. For example, you might want to optimize for impressions -- the number of times an ad is shown.
2. Collect data and obtain labels.
3. Engineer features.
4. Train models.
5. During error analysis, you realize that errors are caused by wrong labels, so you relabel data.
6. Train model again.
7. During error analysis, you realize that your model always predicts that an ad shouldn't be shown, and the reason is that 99.99% of the data you have is a no-show (an ad shouldn't be shown for most queries). So you have to collect more data on ads that should be shown.
8. Train model again.
9. The model performs well on your existing test data, which is by now two months ago. But it performs poorly on the test data from yesterday. Your model has degraded, so you need to collect more recent data.
10. Train model again.
11. Deploy model.
12. The model seems to be performing well but then the business people come knocking on your door asking why the revenue is decreasing. It turns out the ads are being shown but few people click on them. So you want to change your model to optimize for clickthrough rate instead.
13. Start over.

There are many people who will work on an ML project in production -- ML engineers, data scientists, DevOps engineers, subject matter experts (SMEs). They might come from very different backgrounds, with very different languages and tools, and they should all be able to work on the system productively. Cross-functional communication and collaboration are crucial.

> 🌳 **Tip**🌳
> As a candidate, understanding this production cycle is important. First, it gives you an idea of what work needs to be done to bring a model to the real world and the possible roles available. Second, it helps you avoid ML projects that are bound to fail when the organizations behind them don't set them up in a way that allows iterative development and cross-functional communication.

1.1.3.2 Machine learning engineer vs. software engineer

ML engineering is considered a subfield of software engineering. In most organizations, the hiring process for MLEs is spun out of their existing SWE hiring process. Some organizations might swap out a few SWE questions for ML-specific questions. Some just add an interview specially focused on ML on top of their existing interview process for ML, making their MLE process a bit longer than their SWE process.

Overall, MLE candidates are expected to know how to code and be familiar with software engineering tools. Many traditional SWE tools can be used to develop and deploy ML applications.

In the early days of ML adoption, when companies had little understanding of what ML production entailed, many used to expect MLE candidates to be both stellar software engineers and stellar ML researchers. However, finding a candidate fitting that profile turned out to be difficult, and many companies had relaxed their ML criteria. In fact, several hiring managers have told me that they'd rather hire people who are great engineers but don't know much ML because it's easier for great engineers to pick up ML than for ML experts to pick up good engineering practices.

> 🌳 **Tip** 🌳
> If you're a candidate trying to decide between software engineering and ML, choose engineering.

1.1.3.3 Machine learning engineer vs. data scientist

ML engineers might spend most of their time wrangling and understanding data. This leads to the question: how is a data scientist different from an ML engineer?

There are three reasons for much overlap between the role of a data scientist and the role of an ML engineer.

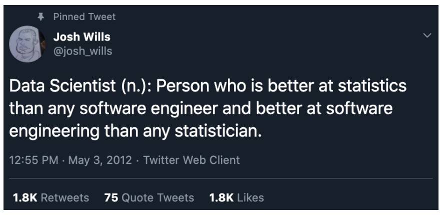
First, according to Wikipedia, "*data science is a multidisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data*." Since machine learning models learn from data, **machine learning is part of data science**.

Second, traditionally, companies have data science teams to generate business insights from data. When interests in ML were revived in the early 2010s, companies started looking into using ML. Before making a significant investment in starting full-fledged ML teams, companies might want to start with small ML projects to see if ML can add value. A natural candidate for this exploration is the team that is already working with the data: the data science team.

Third, many tasks of the data science teams, including demand forecasting, can be done using ML models. This is also how most data scientists transition into ML roles.

However, there are many differences between ML engineering and data science. The goal of data science is to **generate business insights**, whereas the goal of ML engineering is to **turn data into products**. This means that data scientists tend to be better statisticians, and ML engineers tend to be better engineers. ML engineers definitely need to know ML algorithms, whereas many data scientists can do their jobs without ever touching ML.

As a company's adoption of ML matures, it might want to have a specialized ML engineering team. However, with an increasing number of prebuilt and pretrained models that can work off-the-shelf, it's possible that developing ML models will require less ML knowledge, and ML engineering and data science will be even more unified.



Pinned Tweet

Josh Wills
@josh_wills

Data Scientist (n.): Person who is better at statistics than any software engineer and better at software engineering than any statistician.

12:55 PM · May 3, 2012 · Twitter Web Client

1.8K Retweets 75 Quote Tweets 1.8K Likes

1.1.3.4 Other technical roles in ML production

There are many other technical roles in the ML ecosystem. Many of them don’t require ML knowledge at all, e.g. you can build tools and infrastructures for ML without having to know what a neural network is (though knowing might help with your job). Examples include framework engineers at NVIDIA who work on CUDA optimization, people who work on TensorFlow at Google, or AWS platform engineers at Amazon.

ML infrastructure engineer, ML platform engineer

Because ML is resource-intensive, it relies on infrastructures that scale. Companies with mature ML pipelines often have infrastructure teams to help them build out the infrastructure for ML. Valuable skills for ML infrastructure/platform engineers include familiarity with parallelism, distributed computing, and low-level optimization.

These skills are hard to learn and take time to master, so companies prefer hiring engineers who are already skillful in this and train them in ML. If you are a rare breed that knows both systems and ML, you’ll be in demand.

ML accelerator/hardware engineer

Hardware is a major bottleneck for ML. Many ML algorithms are constrained by processors not being able to do computation fast enough, not having enough memory to store data/models and load them into memory, not being cheap enough to run experiments at scale, not having enough power to run applications *on device*.

There has been an explosion of companies that focus on building hardware both for training and serving ML models, both for cloud computing and edge computing. These hardware companies need people with ML expertise to guide their processor development process: to decide what ML models to focus on, then implement, optimize, and benchmark these models on their hardware. More and more hardware companies are also looking into using ML algorithms to improve their chip design process[11][12].

ML solutions architect

This role is often seen at companies that provide services and/or products to other companies that use ML. Because each company has its own use cases and unique requirements, this role involves working with existing or potential customers to figure out whether your service and/or product can help with their use case and if yes, how.

Developer advocate, developer programs engineer

You might have seen developer relationship (devrel) engineer roles such as **developer advocate** and **developer programs engineer** for ML. These roles bridge communication between people who build ML products and developers who use these products. The exact responsibilities vary from company to company, role to role, but you can expect them to be some combination of being the first users of ML products, writing tutorials, giving talks, collecting and addressing feedback from the community. Products like TensorFlow and AWS owe part of their popularity to the tireless work of their excellent devrel engineers.

Previously, these roles were only seen at major companies. However, as many machine learning startups now follow the open-core business model -- open-sourcing the core or a feature-limited version of their products while offering commercial versions as proprietary software -- these startups need to build and maintain good relationships with the developer community, devrel roles are crucial to their success. These roles are usually very hard to fill, as it's rare to find great engineers who also have great communication skills. If you're an engineer interested in more interaction with the community, you might want to consider this option.

[11]: Chip Design with Deep Reinforcement Learning (Google AI blog, 2020)

[12]: Accelerating Chip Design with Machine Learning (NVIDIA research blog, 2020)

1.1.3.5 Understanding roles and titles

While role definitions are useful for career orientation, a role definition poorly reflects what you do on the job. Two people with the same title on the same team might do very different things. Two people doing similar things at different companies might have different titles. In 2018, Lyft decided to rename their role of “Data Analyst” to “Data Scientist”, and “Data Scientist” to “Research Scientist”[13], a move likely motivated by the job market’s demands, which shows how interchangeable roles are.

> 🌳 **Tip** 🌳
> When unsure what the job entails, ask. Here are some questions that might help you understand the scope of a role you're applying for.
>
> - How much of the job involves developing ML models?
> - How much of the job involves data exploration and data wrangling? What are the characteristics of the data you'd have to work with, e.g. size, format?
> - How much of the job involves DevOps?
> - Does the job involve working with clients/customers? If yes, what kind of clients/customers? How many would you need to talk to? How often?
> - Does the job involve reading and/or writing research papers?
> - What are some of the tools that the team can't work without?

In this book, I use the term “machine learning engineer” as an umbrella term to include research engineer, devrel engineer, framework engineer, data scientist, and the generic ML engineer.

> 🌊 **Resources** 🌊
>
> - What machine learning role is right for you? by Josh Tobin, Full Stack Deep Learning Bootcamp 2019.
> - Data science is different now by Vicki Boykis, 2019.
> - The two sides of Getting a Job as a Data Scientist by Favio Vázquez, 2018.
> - Goals and different roles in the Data Science platform at Netflix by Julie Pitt, live doc.
> - Unpopular Opinion - Data Scientists Should Be More End-to-End by Eugene Yan, 2020.

[13]: What’s in a name? The semantics of Science at Lyft by Nicholas Chamandy (Lyft Engineering blog, 2018)

1.2 Types of companies

Different types of companies offer different roles, require different skills, and need to be evaluated using different criteria.

1.2.1 Applications companies vs. tooling companies

👦 **Personal story** 👦

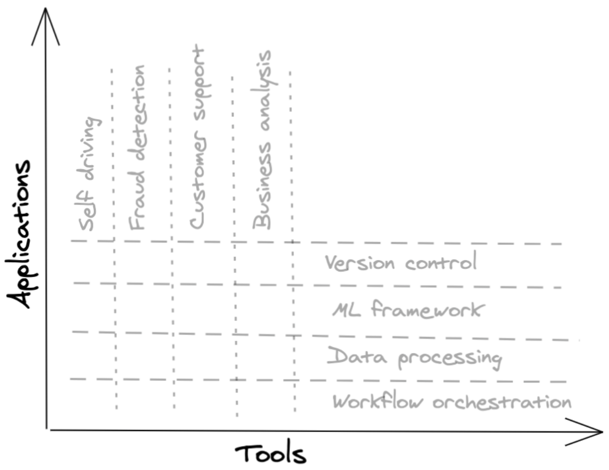
After NVIDIA, I wanted to join an early-stage startup. I was considering two AI startups that were similar on the surface. Both had just raised seed rounds. Both had about 10 employees, most of these were engineers, and were ready for hypergrowth.

Startup A already had three customers, had just hired its first salesperson, and was ready to hire more salespeople to aggressively sell. Startup B had two customers, who they called design partners, and had no plan yet of hiring salespeople. I liked the work at startup B more but thought that startup A had a better sales prospect than startup B, and for early-stage startups, sales are essential for survival.

When I told this dilemma to a friend, who had invested in companies similar to both A and B, he pointed out to me that I forgot to consider the key difference between these two startups: A was an applications company while B was a tooling company.

Applications companies provide applications to solve a specific business problem such as business analytics or fraud detection. Tooling companies create tools to help companies build their own applications. Examples of tools are TensorFlow, Sparks, Airflow, Kubernetes.

In industry lingo, an application is said to target a vertical, whereas a tool targets a horizontal.



Applications are likely made to be used by subject matter experts who don't want to be cumbered with engineering aspects (e.g. bankers who want applications for fraud detection or customer representatives who want applications to classify customer tickets). Tools are likely made to be used by developers. Some are explicitly known as devtools.

In the beginning, it's easier to sell an application because it's easier to see the immediate impact of an application and there's less overhead for adopting an application. For example, you can tell a company that you can detect fraudulent transactions with 10% higher accuracy and the company can just input their data into your application and get out results whether a transaction is fraudulent or not.

For a company to adopt a tool, there’s a lot of engineering overhead. They might have to swap out their existing tool, integrate the new tool with the rest of their infrastructure, retrain their staff or replace their staff. Many companies want to wait until a tool proves its usefulness and stability to a large number of companies first before adopting it.

However, for tooling companies, selling becomes a lot easier later on. Once your tool has reached a critical mass with a sufficient number of engineers who are proficient in it and prefer it, other companies might just become a user without you having to sell it. However, it’s really, really hard to get to that critical mass for new tools, and therefore, in general, tooling companies have higher risks than applications.

After talking to my friend, I realized that it’s normal for a company like A to have more customers than a company like B early on. But it doesn’t mean that A has a better sales prospect. In fact, having two large companies as design partners is a really good sign for A.

A year later, both companies acquired a similar number of customers and grew to have around 30 employees, but more than half of company A are in sales whereas 80% of company B are in engineering.

This new understanding helped me narrow down my choices. Because I preferred building tools for developers and wanted to work for an engineering-first instead of a sales-first organization, the decision became much easier.

> **⚠ Ambiguity ⚠**
> Whether a company is an application company or a tooling company might just be a go-to-market strategy. For example, you have a new tool that can address use cases that companies aren’t aware of yet, and you know it’d be hard to convince companies to make significant changes to their existing infrastructures for uncertain use cases. So you come up with a compelling use case that can’t be done without your tool, build an application around that use case, and sell that application instead. Once customers are aware of the usefulness of your tool, you switch to selling the tool directly.

> **🌳 Tip 🌳**
> If you're unsure whether the role involves working with an application or a tool, here are a few questions you may ask.
>
> - Who are the main users of your product?
> - What are the use cases you're targeting?
> - How many people does your company have? How many are engineers? How many are in sales?

1.2.2 Enterprise vs. consumer products

Another important distinction is between companies that build enterprise products (B2B - business to business) and companies that build customer products (B2C - business to consumer).

B2B companies build products for organizations. Examples of enterprise products are Customer relationship management (CRM) software, project management tools, database management systems, cloud hosting services, etc.

B2C companies build products for individuals. Examples of consumer products are social networks, search engines, ride-sharing services, health trackers, etc.

Many companies do both -- their products can be used by individuals but they also offer plans for enterprise users. For example, Google Drive can be used by anyone but they also have Google Drive for Enterprise.

Even if a B2C company doesn't create products for enterprises directly, they might still need to sell to enterprises. For example, Facebook's main product is used by individuals but they sell ads to enterprises. Some might argue that this makes Facebook users products, as famously quipped: "If you're not paying for it, you're not the customer; you're the product being sold.[14]"

These two types of companies have different sales strategies and engineering requirements. Consumer products tend to rely on viral marketing (e.g. invite your friends and get your next order for free) to reach a large number of users. Selling enterprise products tends to require selling to each user separately.

Enterprise companies usually have the role of **solutions architect** and its variances (**solutions engineer**, **enterprise architect**) to work with enterprise customers to figure out how to use the tool for their use cases.

> 🌳 **Tip** 🌳
> Since these two types of companies have different business models, they need to be evaluated differently when you consider joining them.
>
> For enterprise products, you might want to ask:
>
> - How many customers do they have? What's the customer growth rate (e.g. do they sign on a customer every month)?
> - How long is their sales cycle (e.g. how long it usually takes them from talking to a potential customer to closing the contract)?
> - How does their pricing structure work?
> - How hard is it to integrate their product with their customers' systems?
>
> For consumer products, you might want to ask:
>
> - How many active users do they have? What's their user growth rate?
> - How much does it cost to acquire a user? This is extremely important since the cost of user acquisition has been hailed as a startup killer[15].
> - Do users pay to use the product? If not, how are they going to make money?
> - What privacy measures do they take when handling users' data? E.g. you don’t want to work for the next Cambridge Analytica.

---

[14]: You’re Not the Customer; You’re the Product (Quote Investigator).

[15]: Startup Killer: the Cost of Customer Acquisition by David Skok.

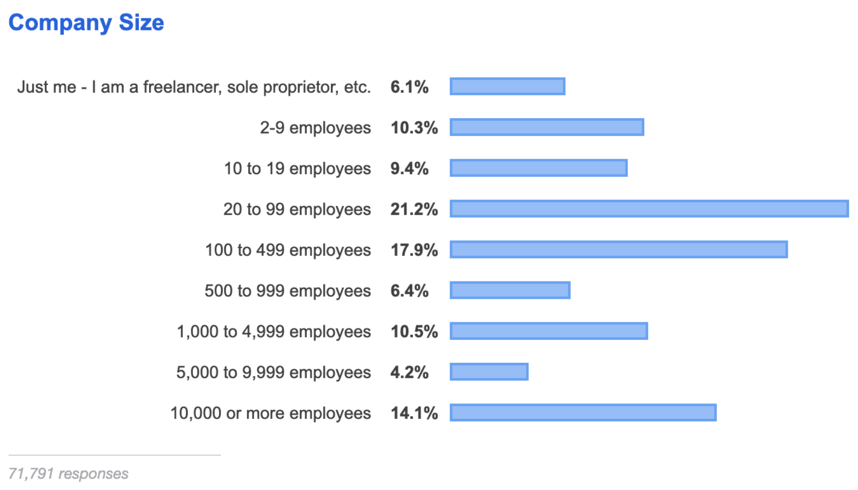
This book was created by Chip Huyen with the help of wonderful friends. For feedback, errata, and suggestions, the author can be reached here.

1.2.3 Startups or big companies

This is a question that I often hear from people early in their careers, and a question that can prompt heated discussions. I've worked at both big companies and startups, and my impressions were pretty much aligned with what is often said of the trade-off between big company stability and startup high impact (and high risk).

	Big companies	**Startups**
Pros	• Brand name recognition. Good for your resume. • Stability. FAAAM's stock is unlikely to be worthless anytime soon. • Well-defined roles. You won't have to work as much as at a startup. • A standard procedure to go up the ranks. Just do your job reasonably and you're set. • (Hopefully) good code review to help you become a better engineer. • Plenty of smart people to work with. Mentoring and guidance. • Perks: free food, free massage, free car rentals, free Lyft codes, generous 401(k) matching, ...	• Can contribute significantly to the product. • More autonomy and more decision making power. • Can get to know everyone. • Can do multiple things at once, making your job interesting. • Big picture, both technically and non-technically. You can see the entire software stack and see how a company runs as a whole. • Can grow with the company and get promoted much faster than at a big company. • Will learn A LOT. • No complacency to settle into. • Might end up with a lot of money.
Cons	• Easy to settle into complacency. • You'll be just another cog in the system. Your effort or lack of it won't change anything. • You'll probably work with a small piece of code. Your work becomes boring fast. • Unlikely that the management will ask for your opinion about where the company is headed. • Too many meetings.	• When you tell people about the company, they go: "What?". • The startup will probably fail and your 0.1% equity will become useless. • Have to work on a lot of things, even things that you hate. • Terrible code review. After a year, you might become an even worse coder. • Less mentoring. • The startup doesn't have as many world-class engineers as Google for you to work with. • Fewer perks.

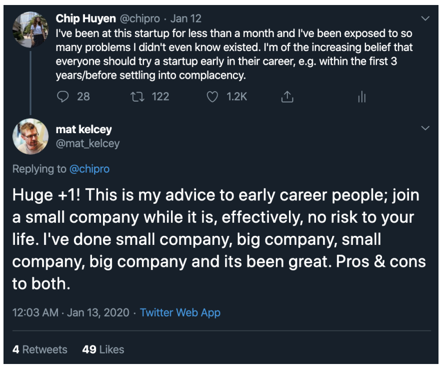
Statistically speaking, software engineers are more likely to work for a big company than a small startup. Even though there are more small companies than large corporations, large corporations employ more people. According to StackOver Developer Survey 2019, more than half of the 71K respondents worked for a company of at least 100 employees.



I couldn't find a survey for ML specific roles, so I asked on Twitter and found similar results. This means that an average MLE mostly likely works for a company of at least 100 employees.

> 🌳 **Tip** 🌳
> A piece of advice I often hear and read in career books is that one should join a big company after graduation. The reasons given are:
>
> - Big companies give you brand names you can put on your resume and benefit from for the rest of your life.
> - Big companies often have standardized tech stack and good engineering practices.
> - Major tech companies offer good compensation packages. Working for them even briefly will allow you to save money to pursue riskier ventures in the future.
> - It's good to learn how a big company operates since the small company that you later join or start might eventually grow big.
> - You'll know what's like to work for a big company so you'll never again have to wonder.
> - Most startups are bad startups. Working at a big company for a while will better equip you with technical skills and experience to differentiate a good startup from a bad one.
> - For immigrants, big companies might be the only option since small companies can't afford to sponsor visas.
>
> If you want to maximize your future career options, spending a year, or even just an internship, at a big company, is not a bad strategy.
>
> Whether you choose to join a startup or a big company, I hope that you get a chance to experience both environments and learn the very different sets of skills that they teach. You don't know what you like until you've tried it. You might join a big company and realize you never want to join another big company again, or you might join a startup and realize you can't live without the stability big companies give you.
>
> And if you believe that you're offered a once-in-a-lifetime opportunity, take it, whether it's at a big company or a startup.

> 👦 **Personal story** 👦
> After graduation, I joined NVIDIA, not because it was a big company, but because I was excited about the opportunity to be part of a brand new team working on challenging projects.
>
> Looking back, I realized the brand name of NVIDIA helped my work to be taken seriously. Being an unknown employee at an unknown company would have thrown me even further into obscurity.
>
> I stayed at NVIDIA for a year and a half then joined a startup. I wanted a fast-moving environment with a steep learning curve, and I wasn't disappointed.


Chip Huyen @chipro · Jan 12
I've been at this startup for less than a month and I've been exposed to so many problems I didn't even know existed. I'm of the increasing belief that everyone should try a startup early in their career, e.g. within the first 3 years/before settling into complacency.
28 122 1.2K

mat kelcey
@mat_kelcey
Replying to @chipro
Huge +1! This is my advice to early career people; join a small company while it is, effectively, no risk to your life. I've done small company, big company, small company, big company and its been great. Pros & cons to both.
12:03 AM · Jan 13, 2020 · Twitter Web App
4 Retweets 49 Likes

> 🌳 **Tip for engineers early in your careers: Know what you're optimizing for** 🌳
> With each career decision, be mindful of what you're optimizing for so that you can get closer to your eventual goal. Some of the goals you can optimize for are:
>
> - **Money now**: some people need or want immediate money, e.g. to pay off debts or to prepare for an economic downturn that they believe will happen in the near future. They might interview with multiple companies and go for the highest bidder. There's nothing wrong with that.
> - **Money in the future**: some are more concerned with being able to make a lot of money in the future. They might choose to pursue a Ph.D. that pays next to nothing but will help them get a highly paid job later.
> - **Impact**: some focus on making an impact. You might work for a startup that allows you to make decisions that affect millions of users or work for a non-profit organization that changes people's lives.
> - **Experience diversity**: the most interesting people I've met optimize for new experiences. They choose jobs that allow them to do things that they've never done before.
> - **Brand name recognition**: it's not a bad strategy to choose to work for the most well-known company or person in your field. This brand can open many doors for you down the line.
> - **Personal growth**: those who optimize for this choose the job that allows them to learn the most, which in turn maximizes their career option. They might choose a job because it offers mentorship or allows them to work on new, challenging tasks.
>
> You can optimize for different things at different stages in your life, but you can only optimize for one thing at a time. You might optimize for new experiences when you're younger, money and recognition when you start having more responsibilities, then impact when you've had enough money to not have to worry about it.
>
> If you don't know what you're optimizing for, optimize for personal growth. Get a skill set that maximizes your options in the future[14].

🌊 **Resources** 🌊

Twitter thread: Advice for people who want to leave a big company to join a startup by Jensen Harris.

How to get rich in tech, guaranteed (Startups and Shit, 2016).

Twitter thread: Joining a startup is not a get-rich-quick scheme by me (shameless plug)

---

[14]: This is a corollary of the principle of maximum entropy: the probability distribution which best represents the current state of knowledge is the one with largest entropy.

Chapter 2. Machine learning interview process

2.1 Understanding the interviewers' mindset

To understand the interview process, it's important to see it from the employers' perspective. Not only candidates hate the hiring process. Employers hate it too. It's expensive for companies, stressful for hiring managers, and boring for interviewers.

While a handful of well-known organizations are swamped with resumes, lesser-known companies struggle to attract talent. I keep hearing from small companies that it's near impossible to compete with offers made by tech giants. After weeks of pulling out all the stops to court a candidate, the company makes an offer only to find out that FAAAM has outbid them. Companies often contract talent agencies that might charge 20-30% first year's salary for a hire, which easily translates to $50K for an entry-level engineering hire.

The competition for talent is especially brutal in Silicon Valley where the high number of companies per capita makes the odds in candidates' favor. Recruiters, even those from companies that receive millions of resumes every year like Google[17], aggressively court potential candidates even if these candidates aren't looking. The majority of people who took a new job in 2018 weren't searching for one[18].

Some candidates express a mild annoyance at recruiters' unsolicited contact. This attitude is often misguided because recruiters are your biggest ally: they work to get you hired. As a candidate, you want to have enough visibility so that recruiters reach out to you.

Every company says that they want to hire the best people. That's not true. **Companies want to hire the best people who can do a reasonable job within time and monetary constraints**. If it takes a month and $10K to find candidate A who can do 93% of the job but six months and $50K to find candidate B who can do 96% of the job, most companies will go with candidate A. The best engineers can still be overlooked if they don't highlight the skills that recruiters value.

You'd think that when companies hire, they know exactly what they want their new hires to do. Unless it's an established team with routine tasks, hiring managers can seldom predict with perfect clarity what tasks need to get done or what skills are needed. Sometimes, companies can't even be sure that they'll need that person. Sam Altman, chairman of the startup accelerator Y Combinator and co-chairman of OpenAI, advises companies that, in the beginning, *"you should only hire when you desperately need to."*

However, because hiring is so competitive and time-consuming, companies can't afford to wait until they're desperate. A desperate hire is likely to be a bad one. Sarah Catanzaro, a partner focusing on AI at Amplify Partners, advises her portfolio companies to start hiring when they're 50% sure that they need someone.

Imagine a startup that has just raised several million dollars and decided that they want to turn their logs into useful features. They think ML can help them, but don't know how it'd be done. When the recruiter asks them for a job description, they whip up a list of generic ML-related skills and experiences they think might be necessary. Requirements such as "5 years of experience" or "degrees in related fields" are arbitrary and might prevent them from hiring the right candidate.

> 🌳 **Tip** 🌳
> Job descriptions are for reference. Apply for jobs you like even if you don't have all the skills and experiences in the job descriptions. Chances are you don't need them for those jobs.

Engineers start interviewing candidates after one to six months at a new company. New hires begin by shadowing more senior interviewers for a few interviews before doing it on their own, and that's often all the training they get. Interviewers might have been in your shoes just months ago, and like you, they don't know everything. Even after years of conducting interviews, I still worry that I'll make a fool of myself in front of candidates and give them a bad impression of my company.

This lack of training means that even within the same company, interviewers may have different interviewing techniques and different ideas of what a good interview looks like. Rubrics to grade candidates -- if in existence at all -- are qualitative instead of quantitative (e.g. "candidates show good debugging skills'").

Hiring managers also aggregate feedback from interviewers differently. Some hiring managers rely on the average feedback from all interviewers. Some rely on the best feedback -- they'd prefer a candidate that at least one interviewer is really excited to work with to someone whose general feedback is good but no one is crazy about. Google is an example of a company that values enthusiastic endorsements over uniformly lukewarm reviews.

Some companies, in their aggressive expansion, might hire anyone as long as there's no reason not to hire them. Other companies might only hire someone if there's a great reason to hire them.

If you think one interview goes poorly, don't despair. There are many random variables other than your actual performance that influence the outcome of an interview: the questions asked, other candidates the interviewer has seen before you, after you, the interviewer's expectation, even the interviewer's mood. It is, in no way, a reflection of your ability or your self-worth. Companies know that too, and it's a common practice for companies to invite rejected candidates to interview again after a year or so.

---

[17]: Google Automatically Rejects Most Resumes for Common Mistakes You've Probably Made Too (Inc., 2018).

[18]: Your Approach to Hiring Is All Wrong (Peter Cappelli, Harvard Business Review, 2019)

2.1.1 What companies want from candidates

The goal of the interviewing process is for a company to assess:

1. whether you have the necessary skills and knowledge to do the job
2. whether they can provide you with a suitable environment to carry out that task.

Companies will be looking at both your technical and non-technical skills.

2.1.1.1 Technical skills

1. **Software engineering**. As ML models often require extensive engineering to train and deploy, it's important to have a good understanding of engineering principles. Aspects of computer science that are more relevant to ML include algorithms, data structures, time/space complexity, and scalability. You should be comfortable with the usual suspects: Python, Jupyter Notebook or Google Colab, NumPy, scikit-learn[19], and a deep learning framework. Knowing at least one performance-oriented language such as C++ or Go can come in handy. BestPracticer has an interesting list of engineering skills needed for skills at different levels.

1. **Data cleaning, analytics, and visualization**. Data handling is important yet often overlooked in ML education. It's a huge bonus when a candidate knows how to collect, explore, clean data as well as knowing how to create training datasets. You should be comfortable with dataframe manipulation (pandas, dask) and data visualization (seaborn, altair, matplotlib, etc.). SQL is popular for relational databases and R for data analysis. Familiarity with distributed toolkits like Spark and Hadoop is also very useful.
2. **Machine learning knowledge**. You should understand ML beyond citing buzzwords. Ideally, you should be able to explain every architectural choice you make. You might not need this understanding if all you do is clone an existing open-source implementation and it runs flawlessly on your data. But models seldom run flawlessly, so you'd need this understanding to evaluate potential solutions and debug your models.
3. **Domain-specific knowledge**. You should have knowledge relevant to the products of the company you're interviewing for. If it's in the autonomous vehicle space, you're probably expected to know computer vision techniques as well as computer vision tasks such as object detection, image segmentation, and motion analysis. If the company builds speech recognition systems, you should know about mel-filterbank features, CTC loss, and common benchmark datasets for the task of speech recognition.

[19]: As of 2019, scikit-learn is as popular as TensorFlow.

2.1.1.2 Non-technical skills

1. **Analytical thinking**, or the ability to solve problems effectively. This involves a step-by-step approach to break down complex problems into manageable components. You might not immediately know how to solve a problem, especially if it's something you've never encountered before, but you should know how to systematically approach it. When hiring for junior roles, employers might value this skill more than anything else. You can teach someone Python in a few weeks, but it takes years to teach someone how to think.
2. **Communication skills**. Real-world ML projects involve many different stakeholders from different backgrounds: ML engineers, DevOps engineers, subject matter experts (e.g. doctors, bankers, lawyers), product managers, business leaders. It's important to communicate technical aspects of your ML models to people who are also involved in the developmental process but don't necessarily have technical backgrounds.

   It's hard to work with someone who can't explain what they are doing. If you have a brilliant idea but nobody understands it, it's not brilliant. Keep in mind that there's a huge difference between fundamentally complex ideas and ideas made complicated by the author's inability to articulate them.

3. **Experience**. Whether you have completed similar tasks in the past and whether you can generalize from those experiences to future tasks. The tech industry is notorious for downplaying experience in favor of ready-to-burn-out twentysomethings. However, in ML where improvements are often made from empirical observations, experience makes all the difference between having a model that performs well on a benchmark dataset and making it work in real-time on real-world data. Experience is different from seniority. There are complacent engineers who've worked for decades with less experience than an inquisitive college student.

4. **Leadership**. In this context, leadership means the ability to take initiative and complete tasks. If you're assigned a task, will you be able to do it from start to finish without someone holding your hand? You don't need to know how to do all components on your own, but you should know what help you need and be proactive in seeking it. This quality can be evaluated based on your past projects. In school or in your previous jobs, did you only do what you were told or did you seize opportunities and take initiative?

The skillset required varies from role to role. See section **1.1 Different machine learning roles** for differences among roles.

2.1.1.3 What exactly is culture fit?

There’s one thing that companies look for that isn’t a skill and sometimes a source of contention: **culture fit**. Some even argue that this is a proxy to create an exclusive culture -- managers hire only people who look like them, talk like them, and come from the same background as them.

Some companies have switched the term “culture fit” for terms like “value alignment”. A fit should be alignment to values, not lifestyle, e.g. whether you value constructive criticism, not whether you go out to drink every Sunday afternoon.

For most big companies, because their culture is already established, one new employee is unlikely to change the office dynamic and culture fit boils down to whether you’re someone people would like to work with (e.g. you’re not an asshole, you’re not defensive, you’re a team player). For small organizations, culture fit is more important as companies want people who share their mission, drive, ethics, and work habits.

Value alignment is also about you evaluating whether this is a company you want to work for. Do you believe in their mission and vision? Are their current employees the type you want to be around? Will the company provide a good environment for you to grow? One candidate told me he turned down an offer after being invited to the company’s push-up competition. He didn’t feel like a culture that places so much importance on testosterone-filled activities would be a good fit for him.

2.1.1.4 Junior vs senior roles

Companies might put junior and senior roles on different hiring processes. Junior candidates, who haven't proved themselves through previous working experience, might have to go through more screenings. Amazon, for example, only requires coding challenges for junior candidates. Senior candidates, however, might be asked more difficult questions. For example, a few companies have told me that they only give the machine learning systems design questions[21] to more senior candidates since they don't think junior candidates would have the context to do those questions well.

A hiring manager at NVIDIA told me: "*When you hire senior engineers, you hire for skills. When you hire junior engineers, you hire for attitude*." I've heard this sentiment echo in multiple places.

A senior director at a public tech company told me that when he interviews junior candidates, including interns and recent college graduates, he cares more about how they think, how they respond, and how they adapt. A junior candidate with weaker technical skills but is willing to learn might be preferred over another junior candidate with stronger technical skills but a horrible attitude.

---

[21]: See examples of machine learning systems design questions in Machine Learning Systems Design (Chip Huyen, 2019).

2.1.1.5 Do I need a Ph.D. to work in machine learning?

No, you don’t need a Ph.D. to work in machine learning.

Those who think that a Ph.D. is needed often cite job posts for research scientists that list “Ph.D.” as a requirement. First, research scientist roles make up a very small portion of the ML ecosystem. No other roles, including the popular ML engineer, require a Ph.D.

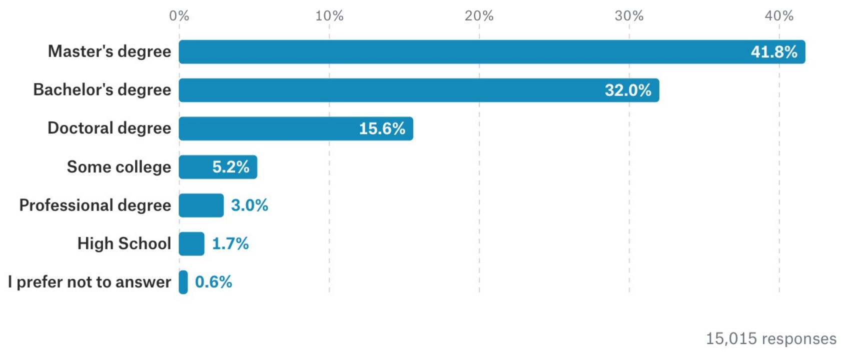
Even for research scientists, there are plenty of false negatives. For example, OpenAI, one of the world’s top AI research labs, lists only two requirements for their research scientist position:

1. track record of coming up with new ideas in machine learning
2. past experience in creating high-performance implementations of deep learning algorithms (optional)[22]. The long list of people who have done amazing work in machine learning but don’t have a Ph.D. includes the current OpenAI CTO, IBM Watson master inventor, PyTorch creator, Keras creator, etc.

Companies know that you don’t need a Ph.D. to do ML research, but still require a Ph.D. because it’s a signal that you’re serious about research. At many companies, the people who screen your resumes aren’t technical and therefore rely on weak signals like Ph.D. to decide whether to pass your resume to the hiring managers.

Engineering roles that require PhDs are the exceptions, not the norm. Some candidates complain that they get rejected by big companies because they don't have Ph.D.’s. Unless the rejections explicitly say so, don't confuse correlation with causation. People with PhDs get rejected too.

In November 2017, Kaggle surveyed 16,000 of their users and found that 15.6% of those working in data science have a Ph.D., 41.8% have only master’s degrees, and 32% have only bachelor’s. However, among those with the highest income, 37.3% (>$200k/year) and 41.% ($150-200k/year) have a doctoral degree.



If you’re serious about research, a Ph.D. is encouraged. However, you shouldn’t let not having a Ph.D. stop you from applying for a job. If you're interested in a company, build up your portfolio, and apply.

[22]: When Google AI opened their office in Tokyo, their requirements for research scientists included a PhD, but the founding research scientist of Google AI Tokyo, David Ha, doesn’t have a PhD.

2.1.2 How companies source candidates

To get hired, it might be helpful to put yourself where employers are looking. Out of all possible channels for sourcing candidates, referrals are, by far, the best channel. Recruiters have, for a long time, unanimously agreed on the effectiveness of referrals. Here are some numbers:

1. 82% of employers rated employee referrals above all other sources for generating the best return on investment.
2. Referred candidates are 55% faster to hire compared to candidates sourced through career sites.
3. The retention rate of referred employees is 45% after two years, compared to 20% for those from job boards.
4. Referred candidates are 2.6 to 6.6% more likely to accept an offer] (https://www.glassdoor.com/research/studies/interview-sources/.
5. Across all jobs, referrals account for 7% of applications but 40% of all hires. If you're referred, you're almost 6x more likely to get your dream job.

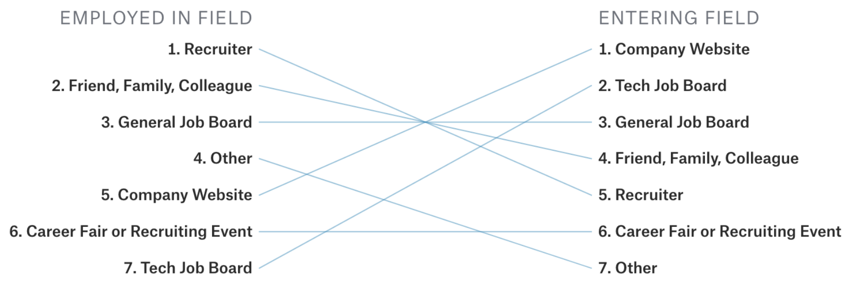
Sam Altman, CEO of OpenAI and the former president of Y Combinator, wrote that: "*By at least a 10x margin, the best candidate sources I've ever seen are friends and friends of friends.*"

Lukas Biewald, founder of two machine learning startups Figure Eight and Weights & Biases, analyzed the performance of 129 hires and concluded that:

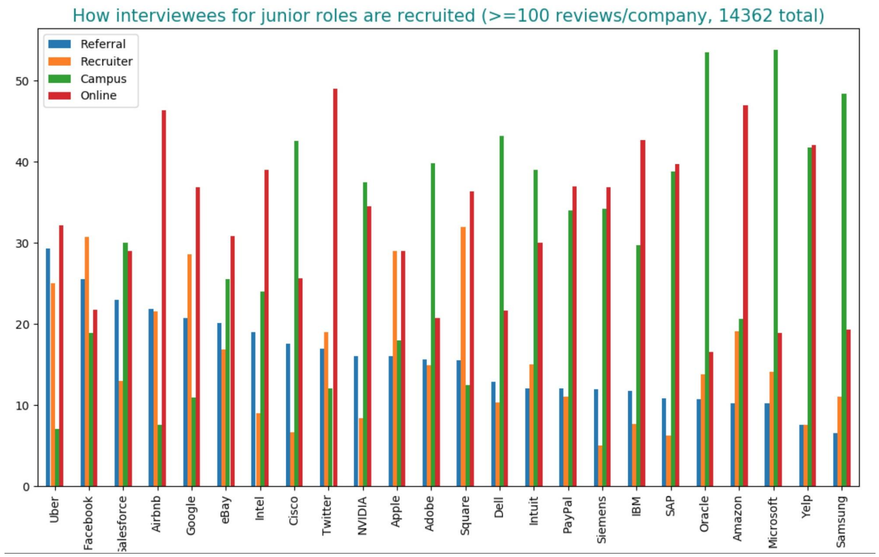
"*Referral quality was incredibly important – the eight worst hires that I've been involved in were all unknown to me and everyone at the company at the time of hiring … The average hire was a 'might hire again' but the average employee or personal referral was a 'definitely hire again'.*"

An analysis of 15,897 Glassdoor interview reviews for software engineering related roles at 27 major tech companies showed that: "*For junior roles, about 10 - 20% of candidates that get to onsites are referred, with Uber leading the chart with almost 30%. For senior roles, that number is higher. Salesforce, Uber, and Cisco all have approximately 30% of their senior onsite candidates referred.*"

The State of Data Science & Machine Learning survey in 2017 by Kaggle shows that while most people seeking to enter the field look for jobs through company websites and tech job boards, most people already employed in the field got their jobs through recruiters' outreach or referrals.



For junior roles, the biggest source for onsite candidates is campus recruiting. Microsoft and Oracle have more than half of their interviewees recruited through campus events such as career fairs and tech talks. Internet giants like Google, Facebook, and Airbnb rely less on campus recruiting, but it still accounts for between 20 and 30% of their onsites. Big tech companies concentrate their recruiting efforts on a handful of popular engineering schools: Stanford, UC Berkeley, MIT, Caltech, CMU, University of Toronto, and the University of Waterloo. The students recruited from those schools then refer their classmates. The circle goes on, turning those major tech companies into a Tech Ivy alumni mixer.



From the employers' perspective, targeting their most promising sources can reduce the hiring cost as well as the risk of disastrous hires. It is, therefore, not surprising that the default message to most candidates who submit their resumes through less promising sources like online applications is "Thank you, next." This process is far from ideal as it creates an exclusive, anti-meritocratic environment. Many qualified people are rejected simply because they don't go to the right school or don't have the right network.

If you're one of these statistically unlucky candidates, one thing you can hope for is that you have a set of skills and/or portfolio that attract recruiters. Around 15 to 25% of onsite candidates for junior roles at each company are contacted by recruiters. For senior roles, this number doubles.

If all else fails, submit your applications and hope for the best. Companies that are the friendliest to online applicants are Twitter, Amazon, and Airbnb with roughly half of their onsite candidates being online applicants. Companies among the most likely to pass on hopeful online applicants are Facebook, Microsoft, and Oracle.

2.1.3 What signals companies look for in candidates

Accurately evaluating candidates is very challenging. First, you can only evaluate something as well as your evaluators allow. Companies can only evaluate a candidate to the extent of the interviewers' knowledge. If your interviewer has a shallow understanding of X, they won't be able to evaluate your in-depth understanding of X. Many companies, including those who claim to be ML companies, don't already have a strong in-house ML team to act as good evaluators[35]. Second, even strong in-house teams don't always mean strong evaluators. Therefore, companies have to rely on signals to help them predict whether a candidate would be a good fit.

As you might have already suspected, pedigrees make for strong signals. It's not a coincidence that companies like to advertise how many ex-Googlers or ex-Facebookers they have on the payroll. If you've worked as a full-time ML engineer at Google, you must have passed its ML interviews and learned good engineering practices from Google.

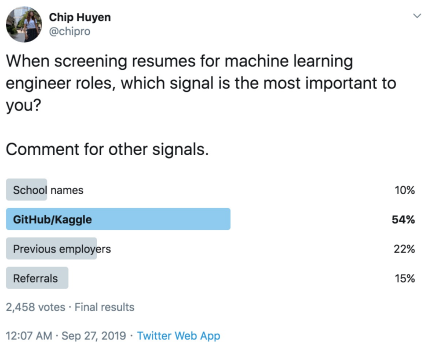
On resumes, college names matter but not much. Their importance is inversely proportional to seniority. If someone, with all the privileges of an elite education, still has no interesting past projects to put on their resume, the fancy college name might even hurt.

However, going to a popular engineering school has several benefits. First, given two equally mediocre resumes, one from MIT, the other from a college nobody has ever heard of, the recruiters might be more inclined to give the one from MIT a call. Second, popular engineering colleges give you access to recruiters who hire from campus events. Third, you'll likely have classmates at big companies who can refer you.

If you're a recent graduate, your college name might matter less than your GPA, which shows your dedication during your studies. Still, your GPA doesn't matter as much if you have other things to show. I've had only one employer asking for my GPA, and it was after I'd got the offer so that they could put it in their database.

The strongest signal is past experience, especially experience similar to the job you're applying for. The experience can be work done at your previous jobs, projects you do independently, or competitions you enter. If you've placed highly in Kaggle competitions, made significant contributions to open-source projects, presented papers at top-tier conferences, written in-depth technical blog posts, self-published books, or done any interesting side projects, you should put them online and highlight them in your resume. There are so many things you can do to signal to people that you're proactive, capable, and willing to work hard.

When I asked on Twitter which signal is most important when screening for ML engineering roles, more than 50% of the 2458 respondents picked GitHub/Kaggle. 22% chose previous employers, 15% picked referrals, and only 10% thought school names were the most important signal. There's a lot of noise in this survey because it's possible that people picked the signal they wanted to be important instead of signals that were actually important.



Chip Huyen
@chipro

When screening resumes for machine learning engineer roles, which signal is the most important to you?

Comment for other signals.

School names	10%
GitHub/Kaggle	54%
Previous employers	22%
Referrals	15%

2,458 votes · Final results

12:07 AM · Sep 27, 2019 · Twitter Web App

⚠ The free project bias ⚠

In 2013, Chris Anderson, the author of *The Long Tail*, tweeted about the advice he received about hiring software developers: "*reject anyone who doesn't have a GitHub profile (the more active the better)*."

Even though GitHub/Kaggle in particular and past projects, in general, seem meritocratic, we have to be mindful of the candidates' circumstances when looking at them. Not everyone can afford to contribute to open-source projects or enter Kaggle competitions. If we place too much importance on voluntary activities, we accidentally punish candidates from less privileged backgrounds -- those who work long hours, have too many responsibilities at home or face online harassment for who they are.

One group that suffers if hiring decisions are made based on open-source contributions is women. According to a 2016 research by the National Center for Women & Information Technology, the percentages of women in various software engineering occupations are 21% (Computer Programmers), 18% (Software Developers), and 34% (Web Developers).[36] Yet, according to reports on Toptal[37] and Wired[38], only 3-5% of open-source contributors are women.

Some hiring managers are aware of this privilege bias. Jeremy Howard, an ex-president of Kaggle and co-founder of fast.ai, responded to my survey on Twitter that he evaluates candidates' achievements with respect to their backgrounds: "*I look for people that have achieved an unusually high level of capability despite limited opportunities or significant constraints. It's been the best hiring signal over many years and companies for me.*[39]"

> 🌳 **Tip: Sell yourself. Highlight your qualities** 🌳
> The hiring process of most tech companies, including and especially the biggest ones, is far from perfect. It's riddled with biases and loopholes. Yet, it's still being used because of legacy and bureaucracy. Until a better process comes along, the best that candidates can do is to understand the signals employers look for and maximize our visibility. On average, recruiters spend only 7.4 seconds on a resume. If you're a great ML engineer but can't signal to recruiters that you're amazing in those 7.4 seconds, you're out.

---

[35]: 40% of AI startups in Europe don't really use AI according to "The State of AI: Divergence 2018" report by MMC Ventures.

[36]: https://www.ncwit.org/sites/default/files/resources/womenintech_facts_fullreport_05132016.pdf

[37]: https://www.toptal.com/open-source/is-open-source-open-to-women

[38]: https://www.wired.com/2017/06/diversity-open-source-even-worse-tech-overall/

2.2 Interview pipeline

The interview pipeline for ML roles evolved out of the tried-and-true software engineering interview pipeline and includes the same components that one can see in a traditional technical interviewing process.

There are many people involved in the interviewing process. For hiring managers, it’s crucial to assign each interviewer a set of skills to evaluate, so that different interviewers ask different questions and that collectively, they get a holistic picture of where you’re at with all the skills they care about. One interviewer might ask you about theories, a couple about coding, one about ML systems design. Ideally, the interviewer tells you the skills they want to focus on so that you can tailor your answers to highlight those skills. If they don’t, ask.

Recruiters are often encouraged to share with candidates the names of their interviewers. If they don’t, ask. You can look up your interviewers beforehand to learn what they do. It’ll give you a sense of not only what you’ll be doing if you join the company, but also your interviewers’ areas of interest.

You should also ask about the team you’re being considered for. At most companies, you interview for a specific team. However, if you apply to companies such as Google and Facebook, you’re matched with a team after you’ve passed. This means that it’s possible to pass their interview process without getting an offer if no team takes you, though it’s rare[40].

The following list of interview components is long and intimidating, but companies usually use only a subset of them. The number of interviews in each component also varies. Companies might skip any step if they’re confident about your ability. Strong candidates might even be invited directly to onsites without all the previous steps. On the flip side, candidates that need to travel for onsites might be vetted more rigorously beforehand.

The entire process can be long and tiring, but it’s long and tiring for every candidate. You don’t have to answer all questions flawlessly. You only need to do better than other candidates. No company expects you to know everything. The field is moving so fast it’s unrealistic to expect any candidate to know all the latest papers and techniques.

---

[40]: How many people failed at the team matching phase when interviewing with Google? (Quora)

2.2.1 Common interview formats

1. **Resume screen**. If you apply by yourself, your resume is likely screened by a recruiter first then passed onto the relevant hiring manager. If you're referred, it might go to the hiring manager directly. For companies inundated with resumes, passing recruiter screening is very difficult.
2. **Phone screen**. The phone screen serves two purposes:

    - to gauge your interest and expectations
    - sell you on the company. Most companies will try to convince you how great the company is regardless of whether they'll make an offer.

      During the phone screen, if the interviewer is an engineer, they might ask technical non-coding questions too.

      Typically, there's only one phone screen. However, in some cases, you might have two phone screens, either because the team isn't convinced by your performance on the first, you're being considered for another team, or it's standard for that company to have two phone screens.

3. **Coding challenge**. Coding challenges are often given to new graduates and interns. Some teams have their challenge before their phone screen, but most have it after. "Challenge before screen" means they don't want to talk to you before they know you can code. A challenge typically consists of 1-3 questions that you have between 30 mins and 3 hours to solve. You can prepare for it by practicing on sites like HackerRank or Leetcode. This step might be skipped for candidates with high-quality code online.

4. **Offsite technical interviews**. Done remotely, these interviews might require coding. You should ask recruiters before each interview for its scope. If there's coding involved, ask what platform will be used so you can familiarize yourself with it beforehand. With the rise of remote work, more and more traditional onsite technical interviews are becoming offsite.
5. **Presentation**. This is often required for research scientist roles to evaluate your insights and for developer advocate roles for your communication skills. For research scientists, you have to present your own work, but for other roles, it may be about any technical topic. Some interviewers like this format as it allows them to not only see candidates in action but also learn something new. Make sure you discuss with your recruiter about acceptable topics so that you don't violate Non-disclosure agreements (NDAs) with your current or previous employers.
6. **Onsite interviews**. The last and most intimidating aspect of the hiring process can consist of anywhere between 4 and 8 (yes, 8) technical/whiteboard interviews which might include a lunch interview and/or social hangout with some team members. Your lunch host might or might not submit an evaluation. Traditionally, this is done onsite. However, with the rise of remote work, more and more teams are experimenting with remote onsites via conference calls.

Before your onsite, companies should send you an interview schedule that looks like this. If they don't, you can ask to get a sense of what your day will be like, and get the names of your interviewers.

10.00 - 11.30	Joan Rivers	Software Engineer	Coding question
11.30 - 12.15	Stephen Chow	Research Scientist	ML theory
12.15 - 13.30	Ali Wong	Engineering Manager	Lunch with the team
13.30 - 15.00	Kumail Nanjiani	Research Engineer	ML implementation
15.00 - 15.30	Dave Chappelle	VP of Engineering	Behavioral questions

2.2.2 Alternative interview formats

Given how expensive, time-consuming, and inaccurate the traditional interview process is, companies are experimenting with new interviewing formats.

1. **Multiple-choice online quiz**. In the last few years, there have been several startups providing online multiple-choice quizzes as a service to companies who want to use this format as part of their hiring process. The quizzes are timed so that you won't have time to google the answers. Some companies are using it as the first screen in replacement of the coding challenge.
2. **Quiz**. One or more interviewers might rapidly throw questions at you to test your knowledge and experiences. DeepMind was the company best known for this. Their quiz lasted two to three hours. At some point, their list of questions was leaked and passed among candidates, which might be why DeepMind abolished this format.
3. **Take-home assignment**. Much of ML is still empirical -- knowing the concepts doesn't necessarily translate to getting things done -- and companies want to see if you can get things done in your comfortable environment. They might give you a paper to reproduce or a problem related to their products to solve and ask you to present your solution. You can look up the solution online, but be ready to defend every architectural choice you make.
4. **Code debugging**. The hard part of coding isn't writing code, but building upon other people's code. In this interview, candidates are asked to fix a buggy program, which might be a buggy ML model. This interview allows companies to evaluate your code comprehension, your reactions to other people's code, and your ability to debug. Stanford uses code debugging when recruiting section leaders[41]. OpenAI and Snorkel AI are two companies that also use this format.
5. **Pair programming**. This is very similar to the traditional coding interview, except that when you get stuck, the interviewer is willing to write code with you, allowing you to continue to the next part.
6. **Good cop, bad cop**. Most companies have one-on-one interviews, but some companies, including Apple and Element AI[42], do two-on-one interviews. Candidates have described this format similar to "good cop, bad cop", with one interviewer asking difficult questions and another helping you out.

[41]: A role similar to teaching assistants.

[42]: Yoshua Bengio's Montreal-based AI startup that suffered a rather bleak outcome. See Element AI sold for $230-million as founders saw value mostly wiped out, document reveals (Globe and Mail, 2020).

2.2.3 Interviews at big companies vs. at small companies

Hiring at a small company is very different from hiring at a big company. One reason is that big companies make a lot of hires, so they need to standardize their process. Small companies refine the process as they go. Another reason is that big companies can afford to occasionally make bad hires, whereas a few bad hires can run a small company into the ground.

As a result, processes at smaller companies can adapt to each role and each candidate. Processes at big companies can be rigid and bureaucratic and might involve questions irrelevant to the role or the candidate. The standardization at big companies also means that the process is more hackable -- thorough preparation can substantially increase your odds.

Another important difference is that big companies can afford to hire specialists, who are great in only a small area. Startups are unpredictable and ever-changing, so they might care less about what you do best and more about whether you can address their most urgent needs.

2.2.4 Interviews for internships vs. for full-time positions

It’s typically much easier to interview for an internship and then get a return offer as a full-time employee than to interview for a full-time position directly. The interview process is a proxy to predict how well you will perform on the job. If you’ve already interned with a team, they know your ability and fit, and therefore might prefer you to an unknown candidate.

At major tech companies, intern programs exist to provide a steady, reliable source for full-time talent. An average intern at Facebook makes $8,000/month, almost twice as much as an average American full-time worker[43], and Facebook isn’t even the highest bidder[44]. It’s hard to justify this salary unless it can offset the recruiting cost later.

The interview process for interns is less complicated because an internship is less of a commitment. If a company doesn’t like an intern, that intern will be gone in three months. But if they don’t like a full-time employee, firing that person is expensive. The number of full-time positions is subjected to a strict headcount, but the number of interns often isn’t. Even when a company freezes the hiring process, such as to cut costs, they might still hire interns.

If your internship doesn’t go terribly wrong and the company is hiring, you’ll likely get an intern-to-full-time offer. At NVIDIA, the majority of full-time offers for new graduates go to their interns. Rachelle Gupta, an ex-recruiter for Google and GitHub, wrote in one of her answers on Quora that:

“*Ranges [of the intern conversion rates] are between the high 60’s% and can be as high as the low 90’s% at top tech companies. I am defining conversion to mean ‘eligible to convert to full time employment’ and not just eligible for an internship next year. At Google, when a team gets an internship it comes with a full-time role that the intern can convert to.*[45]”

If you’re a student, it’s never too early to start looking for internships. It’s not uncommon to intern as high school students. However, if you’ve already passed that phase, don’t fret. This book is to help you with the process.

---

[43]: Some college internships pay twice what regular workers earn (CNBC, 2019).

[44]: “Snapchat interns earn $10,000 per month, plus $1,500 for housing, while Pinterest interns earn $9,000 per month, plus $1,000 towards relocation and another $3,000 for housing.”

[45]: What are intern conversion rates at top tech companies?

2.3 Types of questions

In this section, we’ll go over the main types of questions in an interview process for ML roles.

2.3.1 Behavioral questions

Each interviewer is likely to start with a short introduction and one or two questions about your background. These questions are to get a sense of who you are and maybe to get you more comfortable. For companies that have a dedicated behavioral round, this part should be short during the technical interviews. The behavioral round may or may not be during lunch, which is usually hosted by the hiring manager or a designated behavioral interviewer.

Behavioral questions are to assess whether your values align with those of the company and whether the company can provide an environment for you to thrive. They can be grouped into four main topics: your background/resume, your interests, your communication style, and your personality.

2.3.1.1 Background and resume

One common question that a lot of interviewers start with is “Tell me about yourself”. You should come up with an under-a-minute answer and practice it beforehand. Your answer should be both personal (you’re you and nobody else) and professional (why you’re a good fit for the role). Your answer might guide the rest of the interview.

Interviewers might also test your general level of expertise with the question: “What level of involvement have you had with [ML|computer vision|natural language processing|etc.]?” Don’t bluff. If you pretend that you’re more experienced than you actually are, the interviewer will find out.

Companies will want to know about your career history. They will probably ask about your educational background, your previous jobs, and your past projects. You should highlight two things. O**ne is the decisions you made at each position that had an impact, both positive and negative. Another is why you decided to leave one company to join the next.**

The interviewer will try to see if you’ve done what you claim on your resume. If you have, how much ownership you took. It’s pretty easy to spot the talkers from the doers just by asking for details. They might pick up one project from your resume and make you explain every choice you made in it. If you claim to be familiar with TensorFlow, be prepared to talk about eager execution, tf.estimator, and distributed training in TensorFlow.

Employers will of course want to know about your current job search: why are you looking and what are you looking for. A question that every company asks is whether you have any impending offer/deadline that they should be aware of. Some candidates feel awkward to admit that they don’t have any offer yet for fear that the company will think less of them. Some resort to lying, telling recruiters that they have impending onsites or offers when they don’t.

You might be able to get away with it -- some even argue that since companies regularly take advantage of their users and employees, we have no ethical obligation to be honest with them. However, it’s a slippery slope. I wouldn’t want to be the person who lies to get ahead, but it’s a personal decision each of us has to make for ourselves.

2.3.1.2 Interests

It’s in companies’ financial interest to have their employees passionate about their jobs. They want to make sure that your interests align. Some example questions are:

- Why are you interested in ML?
- What interests you about this position?
- What do you hope to learn in your next job?
- In your current role, for what types of problems are you the go-to person? If I asked others on your team this question, what would they say?

Joshua Levy, a founding engineer at the AI-power SaaS company BloomReach, said his go-to interest question is: “*Think of a period when you were thrilled and excited to go to work. What made you excited?*” Was it the problem you were trying to solve, the impact you could make, the technologies you had access to, the people you worked with, your learning curve, or something else?

One pet question for many interviewers is “*explain a paper that you really like*.” This answer shows not only the kind of problems you’re interested in, but also how far you’re willing to go to understand a problem, and how well you communicate technical concepts. It doesn’t matter what paper you pick, as long as you understand it enough to talk about it. Before your interviews, choose any paper, read it inside out and implement it yourself to understand all the subtleties.

My personal favorite is: “*What have you done that you’re most proud of?*” The answer to this question doesn’t even have to be technical. I once told an interviewer that I’m most proud of the non-profit organization I started in high-school to encourage young people to explore outside their comfort zone. I got an offer.

Interviewers want to get a sense of who you are: what motivates you to do your best work, how you handle obstacles. Employers want you to overcome obstacles and do your best work at their companies. Companies want passionate people, so talk about things that can bring out the fire in your eyes.

2.3.1.3 Communication

Communication is arguably the most important soft skill for a job. Teamwork is impossible if team members don’t or can’t communicate with each other. Each company will want to evaluate not only how well you communicate but also whether you can adapt to the communication style of the company. For example, if a company is heavy on daily status updates and you prefer to disappear for days to focus on a project and only emerge when you’re done, it might not be a good fit.

Your communication style and skills are evaluated through the entire process, from your cover letter, how you respond to their emails, to the way you answer each interview question. Companies might also ask explicit questions about how you communicate. Here are some of the questions to think about:

- Tell me about a time you had to give constructive criticism.
- Tell me about a time you received critical feedback.
- Have you ever disagreed with your manager? How did you address that?
- How would your coworkers describe your communication style?
- What was the most ambiguous project you’ve been part of? How did you handle it?
- What kind of managerial style would bring out the best in you?

2.3.1.4 Personality

Companies want to know about you, your personality, your grit, your strengths, and your weaknesses. Directly asking a candidate for their strengths and weaknesses often yields poor results -- most people are reluctant to admit their flaws and too shy to boast about their strengths. Interviewers, therefore, might frame the questions differently.

Here are some of the questions an interviewer might ask.

- How would your colleagues describe you?
- If you were interviewing a potential teammate, what traits would you look for?
- If you needed a partner to solve a programming problem, how would you describe your ideal partner?
- If you could go back to five years ago, what skill would you learn?
- Tell me about a time when you faced a seemingly insurmountable challenge and how you overcame it.

Senior interviewers tend to have a go-to question that they believe reveals the most about a person. In his book *Zero to One*, Peter Thiel revealed that he asked everyone he interviewed: “*What important truth do very few people agree with you on?*” This question is hard to answer as it first requires you to know what most people agree on, be able to think independently and be confident enough to express it.

Be yourself, but don’t be a jerk. Silicon Valley has been known for tolerating brilliant jerks -- high performers who are rude and unpleasant to work with -- which contributes to the alleged toxic culture of the tech industry. However, in recent years, many tech companies including Netflix have made the commitment to not hire jerks, no matter how brilliant they are.

2.3.2 Questions to ask your interviewers

You usually have 5-10 minutes at the end of each interview, and a lot more during the behavioral round, to ask questions. Sometimes, you can learn more about someone from their questions than their answers.

The interview process is a two-way street -- not only a company evaluates whether you're a fit but you also evaluate whether you want to work for that company. You should use every opportunity you have to ask questions to learn about the company -- their mission, vision, values, competitors, future plans, challenges they're facing, possible career path, policies that you should know about, internal hierarchy, and existing corporate politics.

You should learn about the team you're interviewing for: team composition and dynamics, team events, managerial style, the kind of people they want to bring onto the team. You should also try to get a sense of the projects you're expected to work on and how your performance will be evaluated.

If you care about the visibility of your work -- which is especially important for those early in their career -- you should ask about the company's publishing policy: do they publish their papers and open-source their code? If a company doesn't publish at all, you join and disappear -- the outside world has no idea what you work on.

Recently, there have been employee walkouts at major tech companies to protest their employers' involvement with certain branches of the government. If that's what you care about, you should definitely ask your potential employer for their stand on working with the government.

Your interviewer's career perspective is a good indication of what yours will be like if you join the company, so you should try to understand what they do and why/how they do it. Why did they choose this company? How is it compared to their previous employers? What do they find challenging about their job? How much freedom do they have in choosing what to work on?

If you need more questions to ask, Julia Evans has a pretty great list.

> **⚠ Never ask your interviewers about compensation ⚠**
> Unless the topic is explicitly brought up by the interviewers. Some hiring managers consider this a red flag as it signals that the candidate only cares about money and will jump ship as soon as a better offer comes along.

2.3.3 Bad interview questions

Most interviewers you meet will be bad interviewers. Few companies have proper interview training programs. Junior interviewers lack the experience to know what signals to look for and lack the technical depth to evaluate your expertise. Senior interviewers might be set in their way with their list of pet questions and might defend to the death the merit of their techniques even in light of contradicting evidence.

Bad interviewers ask bad questions. Even good interviewers sometimes ask bad questions. Here are some examples of bad interview questions.

1. **Questions that ask for the retention of knowledge that can be easily looked up**

   Example: “Write down the equation for Adam optimizer.”

   These questions reward those who happen to review this piece of knowledge before the interview or those who prioritize rote learning. Good candidates might be able to work out an equation eventually, but it’s probably not the best use of interview time. The knowledge that can be easily acquired isn’t worth testing for.

2. **Questions that evaluate irrelevant skills**

   Example: “Write a linked list.”

   Interviewers who ask these questions defend their decision by saying that it’s important for ML engineers to have good coding skills. This is based on the assumption that the ability to write linked lists signals coding ability, but it often only signals that a candidate has practiced for a standard software engineering interview. A better approach is to ask candidates to write solutions for practical tasks such as remove duplicated samples in a dataset that doesn’t fit in memory.

3. **Questions whose solutions rely on one single insight**

   Example: “Take derivative of x^x."

   Solving these questions relies on having seen them before or one moment of brilliant insight. It says nothing about one’s ability.

4. **Questions that try to evaluate multiple skills at once**

   Example: “Explain PCA to your grandma.”

   If a candidate does poorly, you have no idea if it’s because they don’t understand PCA or because they don’t know how to talk to grandmas. A good interviewer would isolate the skills they want to evaluate. For example, interviewers can first give candidates a problem that can be solved with PCA (e.g. a dataset has a lot more features than samples). If candidates bring up PCA, interviewers ask them to explain PCA intuitively.

5. **Questions that use specific hard-to-remember names**.

Example: questions about “Moore–Penrose inverse” or “Frobenius norm.”

Candidates might know pseudoinverse or matrix norm without knowing that they are called Moore–Penrose and Frobenius. This can be especially hard for non-native English speakers who study in another language. If you encounter a term you’re not familiar with during interviews, ask for an example.

6. **Open-ended questions with one expected answer**

   Some interviewers ask open-ended questions that have multiple possible answers. However, because they know only one answer, they expect candidates to come up with that exact solution. This can be frustrating for candidates.

7. **Easy questions during later interview rounds**

   Example: “Find the longest common subsequence.”

   Once these questions are solved, there isn’t much room for improvement. Answering them poorly means that a candidate is unqualified, but answering them well doesn’t mean that a candidate is good. These questions are okay during screening to make sure that a candidate can pass a bar, but bad for later interview rounds when you want to know how high a candidate can jump.

When asked a question that you think is bad, should you tell your interviewer that it’s bad? There’s a non-zero chance that your interviewer might appreciate your candid feedback, but statistically speaking, they might get offended and write you off as someone they wouldn’t want to work with.

Instead, you should ask for clarification. If stuck, explain why you’re stuck and what information you’d need to overcome it. If unsure about the interviewer’s intention, ask. “To better answer your question, is it to evaluate my understanding of X?” should be sufficient.

2.4 Red flags

You might want to avoid companies that have displayed the following red flags.

1. The company is less interested in your expertise and more interested in whether you can fulfill their most urgent needs -- they might ask whether you can do this or that instead of digging deep into your knowledge and experience. If they only want you for their urgent needs, they might let you go when they've outgrown those needs.
2. The company doesn't respect your time, e.g. canceling your interviews last minute. Either they aren't that interested in hiring you[46], or they are an organizational mess.
3. The majority of questions are bad. It means the company hasn't spent a lot of time thinking about their hiring pipeline, which leads to poor hiring decisions, which in turn can ruin the company.
4. The company pressures or threatens you into taking their offers. For example, they give you only a day to consider their offer[47]. Companies should give you at least a week, and if you need more time to wait for interviewing results at other companies, they usually extend their deadline. Companies should hire only the people who want to join them after having considered all options, otherwise, they might keep wondering and leave at the first chance they get.
5. The company displays a culture incompatible with your values. If you have small children, you might want to avoid companies that expect employees to respond to emails at 8pm. If you're on the more mellow side, you might not want to join a company whose team building activities involve beer-pong or ax-throwing competitions.

---

[46]: DeepMind is also known for cancelling interviews last minute. They did that to me and also to a friend of mine who eventually joined them. It could just be that they weren't terribly excited about us joining.

[47]: A candidate told me he was once threatened by a well-known ML researcher that if he doesn't take his offer, he won't be able to find jobs at any of the various organizations that this researcher is involved in.

2.5 Timeline

Hiring speed varies wildly from company to company. On the long end of the spectrum, Google and DeepMind processes typically take between six weeks and three months. The long process is partly because of their recurring influx of candidates, and partly because they make hiring decisions at the company-level. Every two weeks, the hiring committee at Google meets to look at all the candidates that have passed the interview process and decide on who to hire. If you've made it to the hiring committee, your odds look good.

On the short end of the spectrum, we have startups and big companies with flat organizational structures. The entire process at a startup can take days if they really want you. Big companies like NVIDIA and Netflix are fast, as they make hiring decisions at the team-level. A team manager can make hiring decisions on the spot. The whole process for my internship at NVIDIA took less than a week. I talked to my manager about converting to full-time in the afternoon and got my offer the next morning.

The rest of the companies fall somewhere in the middle. The timeline depends a great deal on your availability for interviews, companies' availability for interviewing spots, how much they want you, and how much they need you. You should ask your recruiter about the expected timeframe if they haven't told you already, and inform them of any time constraints you have.

If it's been at least a week after your interviews and you haven't heard from your recruiter, it's okay to send a short and respectful check-in. Let them know that you're excited about the company and would love to hear any updates as you have to consider other opportunities.

2.6 Understanding your odds

When applying for a role, you might wonder what your odds for that role are. If hiring decisions followed a uniformly random distribution, the odds at major tech companies would be abysmal. Each year, Google receives several million resumes and hires several thousand, which makes the odds around 0.2%.

However, the odds are not uniformly distributed for people applying for the same role at the same company. It depends on your profile, whether you're referred and who referred you, how much the company needs that role, who screens your resume, who they already have in their pipeline, how serious other applicants are.

Companies have very different screening philosophies -- some give every not-obviously-disqualified candidate a phone screen whereas some only respond to the top applicants.

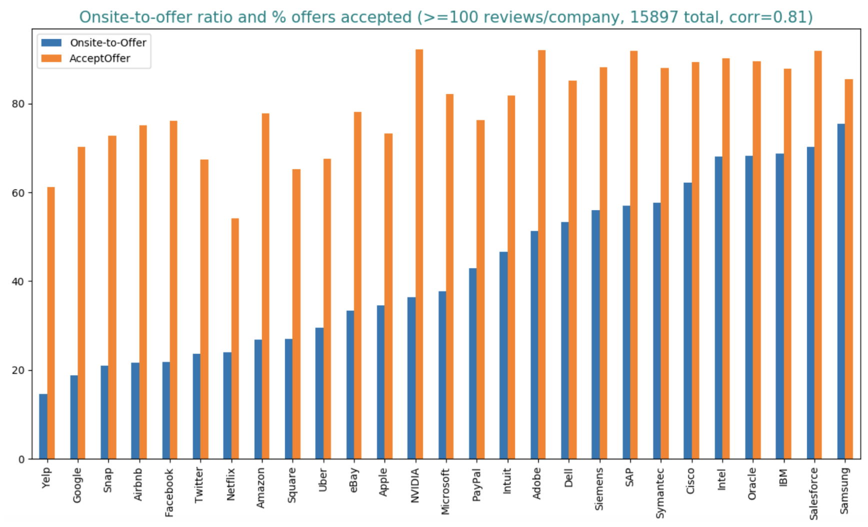
All of these factors, coupled with the fact that few companies publicize the number of resumes they receive or the number of hires each year, make it impossible to estimate the odds from submitting an application to getting an offer.

However, it's possible to estimate the onsite-to-offer ratio, the percentage of onsites that lead to offers, using the 15,897 interview reviews for software engineering related roles at 27 major tech companies on Glassdoor as of August 2019. This ratio correlates to the yield rate -- the percentage of candidates who accept their offers at a company. Even though the estimation is for software engineering roles, it serves as an indication for ML roles. There are many biases in this data, but hopefully, a large number of reviews smoothes out some noise[49]. If all reviews suffer from the same biases, they are still useful for comparison across companies.

The data shows that the onsite-to-offer ratio ranges from a low of 15% to a high of 70%, whereas the yield rate goes from 50% to 90%. For example, at Google, 18.83% of onsites lead to offers, and 70% accept their offers.

Due to the biases of online reviews, the actual numbers should be lower. After talking to recruiters and doing extensive research, I found that the onsite-to-offer ratios here are a few percentage points higher than the actual numbers. For example, this and this claim that the onsite-to-offer ratio for Google is 10-20% and Amazon 20%.

The offer yield rate of near 90% is unheard of. Recruiters from companies with high yield rates told me that they aim to get those numbers up to 80%. The two companies leading the chart are NVIDIA, Adobe, SAP, and Salesforce. However, companies like Salesforce encourage candidates who accept their offers to leave reviews on Glassdoor, which inflates their actual yield rates.



The 5 companies with the lowest onsite-to-offer ratios are all Internet giants -- Yelp, Google, Snap, Airbnb, and Facebook -- who are known to be highly selective. Companies with high onsite-to-offer ratios aren't necessarily unselective. They might be more selective during the screening process and only interview candidates that they really like. Onsites are costly, so the higher the onsite-to-offer ratio, the more financially sound the process.

There's a strong correlation (0.81) between the onsite-to-offer ratio and the yield rate -- the higher the onsite-to-offer ratio, the higher the yield rate. A candidate that gets an offer from Google is more likely to turn it down than a candidate that gets an offer from a less selective company.

There are several reasons. First, if a candidate passes the interviews at selective companies like Google or Facebook, they probably have other attractive offers to choose from. Second, selective companies tend to make competitive offers, which incentivizes candidates to get offers from them to negotiate with companies that they really want to work for. Third, the process at those companies usually takes a long time. By the time a candidate receives the offer, they might have already settled at another company. Last but not least, since candidates at Google and Facebook only get matched with a team after they've received their offers, they might reject the offers if they don't like the team.

---

[49]: Some of the biases in this data:

```
*   Few people actually leave reviews for anything online
*   Those who do are likely compelled by either a really good or really bad ex
*   Those who receive offers are more likely to give reviews than those who do
*   Those who accept offers are likely to give reviews than those who don't
*   Junior candidates are more likely to give reviews than senior candidates
```

This book was created by Chip Huyen with the help of wonderful friends. For feedback, errata, and suggestions, the author can be reached here. Copyright ©2021 Chip Huyen.

Chapter 3. After an offer

Life doesn’t end after a rejection. It doesn’t end after getting an offer either. You might want to negotiate your offer, consider other opportunities, contemplate your career perspective at the company or after you leave.

It seems silly to think about leaving a company before joining, but at 13.2%, the tech industry has the highest turnover rate out of all business sections. Median tenure at Amazon is one year, and Google 1.1 years[50]. You might find yourself being a candidate again sooner than you would think.

In this chapter, we’ll discuss compensation details and career progression that might be useful when you consider your options.

[50]: The Real Problem With Tech Professionals: High Turnover. Forbes, 2020.

3.1 Compensation package

There are two types of compensation: direct compensations and indirect compensations. For most tech companies, direct benefits include three main components:

- base salary (cash)
- equity grants (shares, options)
- bonuses

Indirect benefits are more diverse and can be very generous but often at major tech companies. Below are some excamples.

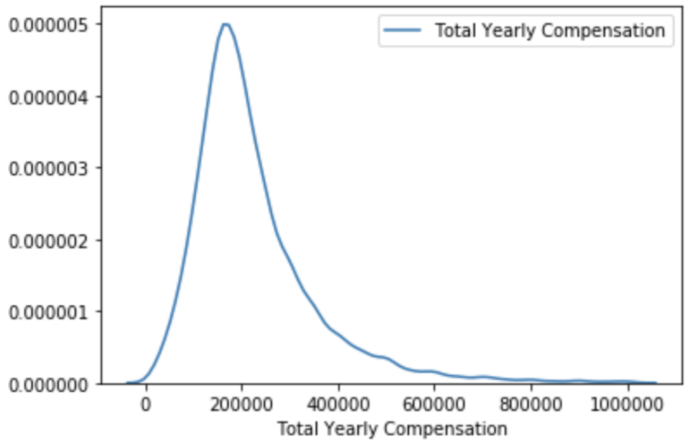
- health insurance
- 401(k) matching
- catered meals, snacks and drinks
- wellness stipend for gym membership, workout gear reimbursement, massages, physical therapy, meditation
- subsidy for commuting, phone, phone plan
- corporate discounts for car rentals, flights, electronics, events
- student loan repayment
- education assistance programs such as tuition reimbursement
- conference attendance assistance
- paid time off

3.1 Base salary

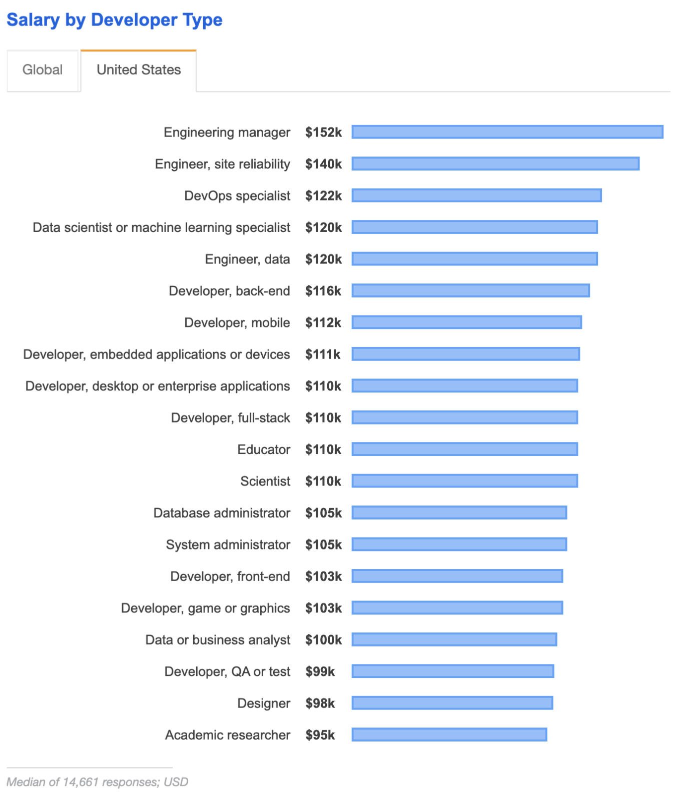
In early 2020, levels.fyi shared their data with me, which includes self-reported direct compensation details of 18.8k tech workers. This data is US-focused with 80% of the entries are US-based (40% from California, 24% from Washington State) and 40% are from FAAAM companies. 65% of the entries are software engineers. The 7 most popular jobs account for 88% of all the entries.

The reported compensation packages follow a skew-normal distribution with a small percentage making outlier amounts of money -- tech workers in India or Russia can make under $20k, while top engineers at Google, Facebook, OpenAI can make millions a year.

The median yearly compensation is $195,000, while the mean is $225,000.



These numbers are higher than the results of the StackOverflow survey in 2019, which states that in the US, the median salary for an Engineering Manager -- the highest paying engineering position in the survey -- is $152,000. This might be because levels.fyi data is FAAAM-focused and includes the whole package -- base salary, equity, and bonus -- while StackOverflow surveyed only salary.



Engineering manager	$152k
Engineer, site reliability	$140k
DevOps specialist	$122k
Data scientist or machine learning specialist	$120k
Engineer, data	$120k
Developer, back-end	$116k
Developer, mobile	$112k
Developer, embedded applications or devices	$111k
Developer, desktop or enterprise applications	$110k
Developer, full-stack	$110k
Educator	$110k
Scientist	$110k
Database administrator	$105k
System administrator	$105k
Developer, front-end	$103k
Developer, game or graphics	$103k
Data or business analyst	$100k
Developer, QA or test	$99k
Designer	$98k
Academic researcher	$95k

Median of 14,661 responses; USD

3.2 Equity grants

Equity grants can be understood as promises -- a company promises you a certain amount of equity if you stay with that company for a certain period of time. When that time is up and you receive your equity, that equity is said to have been vested.

Equity grants are vested on schedule which is the same for everyone in the company. The two most common schedules are:

- equally over four years. e.g. if you're granted 100 shares, 25 shares are vested at the end of each year.
- a quarter of your equity grants is vested after the first year and the rest is vested equally at the end of each month for the next 3 years.

The more you stay at a company, the more equity you'll get. Your new equity grants will also follow a vesting schedule. For example, you're granted 100 shares when you join. After the first year, you have access to 25 shares. Because you're performing well, you're also awarded 80 extra shares over the next 4 years. So, at the end of your second year, you have: 25 old shares + 20 new shares = 45 shares. The longer you stay at a company, the more grants you have left to vest. The schedule is to incentivize employees to stay longer at a company.

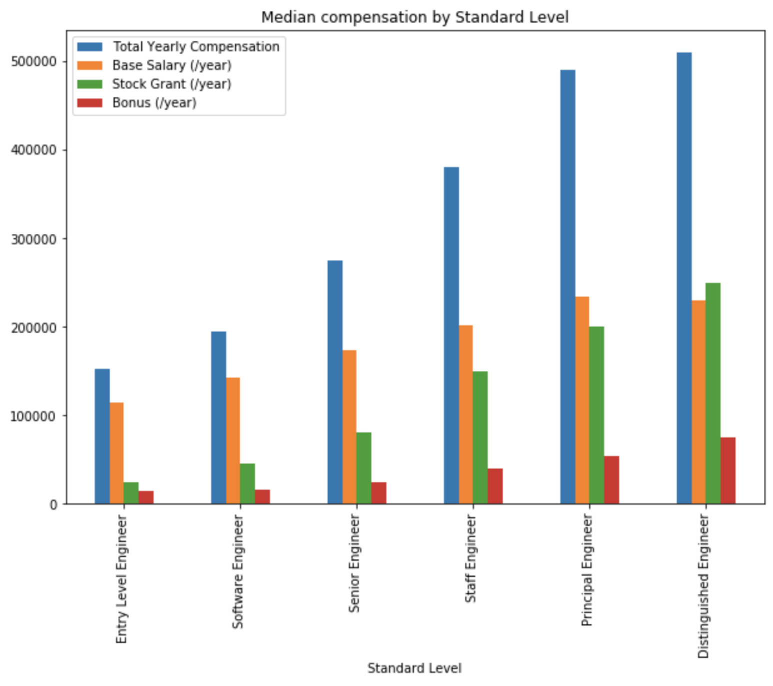
3.3 Bonuses

There are many types of bonuses, but the two most common are a sign-on bonus that you get when you join and an annual bonus. Sign-on bonuses for junior roles can go anywhere from $0 to over $100K. For more senior roles, they can be unbounded. I've seen sign-on bonuses in the neighborhood of half a million.

Annual bonuses are often between 10% and 20% of your base salary. However, companies like Netflix don't believe in bonuses, but prefer to include bonuses to the base salary because they believe that "$200K base salary with 15% performance bonus" is less attractive to candidates than "$230K base salary."

3.4 Compensation packages at different levels

As level increases, base salary and bonus increase slowly, but equity increases the most. For junior levels, base salary makes up the largest chunk of your direct compensations. As you level up, the proportion of equity goes up -- higher equity reflects more responsibilities you have towards the company's success.



3.2 Negotiation

Companies expect you to negotiate. The initial offers are designed to make room for that. They offer the smallest amount you'd accept, not what is fair. I've seen candidates who had their offers doubled through negotiation. I've also seen people getting paid $100K/year less than their peers because they didn't.

Negotiation is stressful. "How much should I ask for?" "Am I being lowballed?" "What if they rescind the offer because they think I'm being greedy?" In case you're worried about the last question, relax. If you're respectful during your negotiation, companies won't rescind their offers. If they do, it's not a company you want to work for.

When negotiating, it's important to know how much other people are getting paid and how much you should get paid. The employer has this information since they've negotiated with many candidates before, but most of us only negotiate with a handful of employers during our entire life. This imbalance of information gives employers more leverage. It especially hurts candidates of underrepresented groups who don't have access to people who can guide them through the process.

There are several ways you can gather more information:

1. Ask your peers, especially those who have worked/received offers for similar positions at similar companies. In many societies, there's a stigma against discussing money. If you ask your friends, don't ask to satisfy your curiosity. Explain how their answers can help you make important decisions. You can also ask for their opinions without asking them how much they make. For example: ask them for level X at company Y, what's the range of compensation you should expect, or tell them your offer and ask them if they think it's fair.
2. Check what other people are getting paid online. There have been several efforts to democratize the compensation information, most notable is levels.fyi where people can share their compensation details anonymously and you can see the range of compensation for certain levels at certain companies. There are also apps like Blind that allow you to anonymously share and get feedback on your offers.
3. See how much other companies offer you. One reason to start your job search early is that you'll have more time to shop for offers. If you can get one offer, you can get another offer. After three offers, you should have a rough idea. Having competing offers helps a lot with negotiation. Companies always match offers, even if they say they don't.

When negotiating with big companies, it's easier to negotiate equity grants and bonuses than base salary, since base salary.

Look beyond direct compensations. You can negotiate for more paid days off, more flexibility at work, a better title, more conferences they'll sponsor you to attend.

3.2.1 Compensation expectations

Candidates dread the question: “What are your compensation expectations?” One way to answer is to get the recruiter to give you a number first. Here’s an example of how you can answer that question.

“I’m excited about this opportunity and I believe that if it’s a good match, we’ll work out an agreement on compensation details. I’d like to learn more about what this position entails first to have a realistic expectation. It’d also be helpful to me to know the range of compensation one can expect at this position if you can share it with me.”

This answer works because it shows that you’re ready to negotiate but you also want to make it work.

One problem with asking the recruiter to give a number is that the negotiation will anchor around it. If they give you a number that’s too low, you might feel awkward raising it significantly.

Another way to answer is to give them a range without committing to it. Come up with a number you want, add ~20% to it, and give it as the lower range. For example, if you’re aiming for $200k/year, you can say something like this.

“My range is flexible. However, I’d like to be compensated fairly for my experience, my unique set of skills, and what the job entails. My understanding is that for this position in this area, you can expect between $220k and $240k annually.”

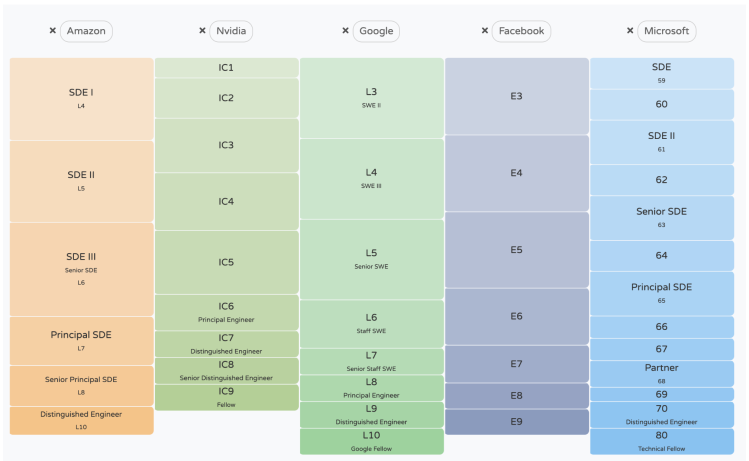
Once you’ve given a number, it can only go down, not up, so make sure to give a number you won’t regret later.

For more negotiation tips, check out 15 Rules for Negotiating a Job Offer (Harvard Business Review, 2014) and Ten Rules for Negotiating a Job Offer (Haseeb Qureshi, 2016).

3.3 Career progression

> **⚠ This section only applies to big companies. At startups, hierarchies are flat and levels are not well-defined. ⚠**

For each offer, you should also look at the level. Most companies have well-defined levels for their software engineering roles that encompass ML roles. Higher levels suggest higher compensations, more decision-making power, and more responsibilities.



Major tech companies follow very different ladders. For example, engineering roles at Google have levels from L3 to L10, while Facebook E3 to E9, Microsoft from 59 to 80. These levels also map to a more standardized ladder which includes:

1. entry level (associate/junior) engineer
2. engineer
3. senior engineer
4. staff engineer
5. principal engineer
6. distinguished engineer

Usually, recent college graduates start at the lowest level, master graduates at the level above, and Ph.D. graduates at the next level. There's not much variance in the base salary for the same level at the same company, as the base salary is usually capped for each level. But there's a lot more variance in equity grants which can change significantly through negotiation.

It's not uncommon to see strong candidates being offered levels higher than their peers. Sometimes, a company might up your level to give you a higher base salary to match another offer you might have. However, companies might be reluctant to level up a new hire, as the higher level means higher expectations, which, many argue, can negatively affect their ability to succeed at the company.

Few companies put levels in their offer letters. If you ask, recruiters should tell you, since you'll find out if you join anyway. It's fashionable for people in tech to say things like "titles don't matter", but they do. In the absence of a perfect method to evaluate someone's actual professional ability, society responds well to titles. Having a higher level means not only higher compensation but also more freedom in deciding what to work on and more negotiating power if you want to change jobs.

In theory, companies should put a lot of thought into designing their levels, and employees should be able to find out what's expected at each level and what they can do to go up the ladder. If you don't know, talk to your manager. You can put it as simple as: "*I want to build my career here and take on more responsibilities at the company. What do you need to see from me to consider me for the next level?*"

For more information on engineering levels, checkout Things to know about engineering levels (Charity Majors, 2020).

Chapter 4. Where to start

In a better world, we shouldn't have to prepare for interviews. In an ideal world, there shouldn't be interviews at all. Unfortunately, in our world, for our ability to be favorably assessed, we need to prepare.

This preparation is a lifelong process and should begin way before you start applying for any job. It includes building up your portfolio, experience, skill sets, and network. You can only write a good resume if you have good things to put on that resume.

If you're only thinking about it now that you want a job, it's not too late. Regardless of what job you do now, keep on preparing so that you can put your best foot forward for your next job search.

This section covers the resources to strengthen your application. They include free online courses, books, articles, tools to familiarize yourself with, etc. The level of preparation you can commit to depends on your timeline.

4.1 How long do I need for my job search?

The job search is a long, stressful, and occasionally demoralizing process. The timeline varies based on where you are and where you want to work. In the US, your job search should start three to six months before you want to start at your new job. For students, it may start at the beginning of your last year in school. If you're required to give notice before leaving -- for example, in Germany, leaving employees must give a three-month notice -- this process should start much earlier. If you need a visa, you need to take into account the time it takes to obtain a visa.

If you plan on getting a job in a year, you might be able to, in your free time, go over all those resources and build up your online presence. When going over an online course, make sure to do assignments on top of watching lectures. You can try to reimplement the papers that you find interesting, improve on them, and put them on GitHub. Try to enter at least a couple of Kaggle competitions.

Three months before your interviews, you might have time to do two to three courses and read three books. For courses, I'd recommend one hands-on course like fast.ai's *Practical Deep Learning for Coders* and one theoretical course like *Machine Learning* by Coursera. You might also want to check out my course at Stanford: *Machine Learning Systems Design* since the course covers practical challenges and solutions for ML in production. For books, I'd recommend *Deep Learning* by Goodfellow et al., M*achine Learning: A Probabilistic Perspective* by Kevin P. Murphy.

A week before your interviews, review the notes of *CS231N: Convolutional Neural Networks for Visual Recognition*, especially the parts about gradient descent, activations, and optimizations as well as rewatch *Full Stack Deep Learning* lectures, especially the ones on *Setting up Machine Learning Projects* and *Infrastructure and Tooling*. You might want to skim the questions in part 2 and part 3 of this book again. You should also review your previous projects in case your interviewers want to know all about them.

A day before your interviews, make sure that you get enough sleep. Don't repeat my mistake of staying up late cramming and showing up to my interviews half-asleep. Arrive 10 minutes before so you have time to settle in.

If you get the job but want a better job in the future, prepare early this time. If you don't get the job, rinse and repeat.

> **⚠ How to become a machine learning expert in 3 months ⚠**
> You can't. Becoming an expert in anything takes years, if not decades. Stay clear of anyone who claims that they can give you a shortcut to becoming a machine learning expert. At best, they teach you bad machine learning. At worst, it's a scam.
>
> Peter Norvig, director of search at Google, wrote a wonderful blog post on how long it takes to learn programming: Teach Yourself Programming in Ten Years. His advice is applicable to ML.

4.2 How other people did it

I find it helpful to follow people whose careers I admire and learn how they got there. There's no one path to any job -- not all ML researchers did their PhDs and not all ML engineers studied computer science in college or went to college at all. Often, candidates with more unconventional backgrounds are more desirable as they can bring fresh perspectives to the team.

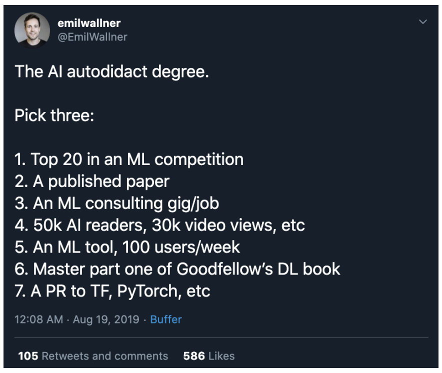
Many people have written about their career paths. Here are some of the stories that I found inspiring.

1. Shifting Careers to Autonomous Vehicles: How I moved from debt collection to self-driving cars

   Vladimir Iglovikov, Senior Computer Vision Engineer at Lyft on how he got rejected at Google and NVIDIA and had his Tesla offer rescinded because he violated his NDA. He talked about the immigration frustrations that everyone wishing to work in another country can relate to.

2. John Washam on how he studied full-time for 8 months for a Google interview and got rejected before he got an offer from Amazon.

3. After getting laid off, Emma Ding analyzed data science job listings, bucketed them into three groups based on their requirements: "Product Analytics", "Modeling, and "Data Engineering". She chose to enhance her resume and practice for Product Analytics roles. By the end of two frenzied months, she received a total of 10 interviews, 4 onsite interviews, and 4 job offers at Twitter, Lyft, Airbnb, and a healthcare startup.
4. Emil Wallner, a resident at Google Arts&Culture on how he self-taught himself to become a machine learning researcher. His ideal curriculum is: "*Knowing how to code is a prerequisite. Then, I'd spend 1-2 months completing Fast.ai course V3, and spend another 4-5 months completing personal projects or participating in machine learning competitions.*"



emilwallner
@EmilWallner

The AI autodidact degree.

Pick three:

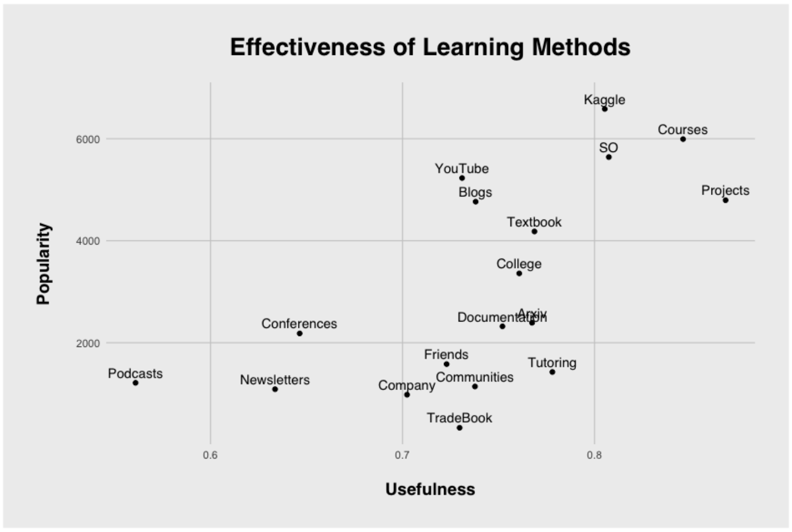
1. Top 20 in an ML competition
2. A published paper
3. An ML consulting gig/job
4. 50k AI readers, 30k video views, etc
5. An ML tool, 100 users/week
6. Master part one of Goodfellow's DL book
7. A PR to TF, PyTorch, etc

12:08 AM · Aug 19, 2019 · Buffer

105 Retweets and comments 586 Likes

4.3 Resources

Given the plethora of available resources online, it can be disorienting trying to figure out which resources to focus on. The Kaggle's state of data science and machine learning survey 2017 asked respondents about the learning methods that they found the most helpful. Here is a visualization of the responses, created by my previous colleague Jack Cook.



The most effective learning methods are doing projects, taking courses, and just spending a ton of time on StackOverFlow (SO). Kaggle competitions rank high on the list but since the respondents are Kaggle users, their answers are biased. A college education is perceived as slightly more useful than watching YouTube tutorials and reading blogs. The least useful methods in this survey are podcasts, newsletters, and conferences.

Attending conferences might not be useful for building your skill sets, but very useful for building up your network. Getting published at conferences is a great way to put your name out there and signal that your knowledge of the field is in-depth enough to come up with original research ideas.

4.3.1 Courses

The courses are listed in the order they should be taken. It was combined in August 2019, so some of the links might have become outdated, but the curriculum can still be useful to have a sense of what areas of knowledge you should acquire and find other ways to acquire them, e.g. other courses or books[51].

1. Probability and Statistics by Stanford Online

See course materials (free online course)

This self-paced course covers basic concepts in probability and statistics spanning over four fundamental aspects of machine learning: exploratory data analysis, producing data, probability, and inference.

Alternatively, you might want to check out this excellent course in statistical learning: An Introduction to Statistical Learning with Applications in R.

2. 18.06: Linear Algebra by MIT

Textbook: *Introduction to Linear Algebra* (5th ed.) by Gilbert Strang

See course materials (videos available)

The best linear algebra course I've seen, taught by the legendary professor Gilbert Strang. I've heard students describe this as "life-changing".

3. CS231N: Convolutional Neural Networks for Visual Recognition by Stanford

See video lectures (2017)

See course materials

CS231N is hands down the best deep learning course I've come across. It balances theories with practices. The lecture notes are well written with visualizations and examples that explain difficult concepts such as backpropagation, gradient descent, losses, regularizations, dropouts, batchnorm, etc.

4. Practical Deep Learning for Coders by fast.ai

See course materials (free online course)

With the ex-president of Kaggle as one of its co-founders, this hands-on course focuses on getting things up and running. It has a forum with helpful discussions about the current best practices in ML.

5. CS224N: Natural Language Processing with Deep Learning by Stanford[52]

See video lectures (2017)

See course materials

Taught by one of the most influential (and most down-to-earth) researchers, Christopher Manning, this is a must-take course for anyone interested in NLP. The course is well organized, well taught, and up-to-date with the latest NLP research. The assignments, while useful, can sometimes be frustrating as training NLP models takes time.

6. Machine Learning by Coursera

See course materials (free online course)

Originally taught at Stanford, Andrew Ng's course is probably the most popular ML course. As of writing, its Coursera version has been enrolled by more 2.5M people. This course is theoretical, so students would benefit more from it after more practical courses such as CS231N, CS224N, and Practical Deep Learning for Coders.

7. Probabilistic Graphical Models Specialization by Coursera

Textbook: *Probabilistic Graphical Models: Principles and Techniques* by Daphne Koller and Nir Friedman

See course materials (free online courses)

Unlike most AI courses that introduce small concepts one by one or add one layer on top of another, this specialization tackles AI top down as it asks you to think about the relationships between different variables, how you represent those relationships, what independence you're assuming, what exactly you're trying to learn when you say machine learning. This specialization isn't easy, but it'll change the way you approach ML. You can also consult detailed notes written by Stanford CS228's TAs here.

8. Introduction to Reinforcement Learning by DeepMind

See lecture videos

Reinforcement learning is hard. This course provides a great introduction to RL with intuitive explanations and fun examples, taught by one of the world's leading RL experts, David Silver.

9. Full Stack Deep Learning Bootcamp[53]

See lecture videos

Most courses only teach you how to train and tune your models. This is the first one I've seen that shows you how to design, train, and deploy models from A to Z. This is also a great resource for those struggling with the machine learning system design questions in interviews.

10. How to Win a Data Science Competition: Learn from Top Kagglers by Coursera

See course materials (free online course)

With all the knowledge we've learned, it's time to head over to Kaggle to build some machine learning models to gain experience and win some money. Warning: Kaggle grandmasters might not necessarily be good instructors.

For even more online sources, kmario23 compiled a list of available online courses. David Venturi also aggregated reviews for popular courses. Emil Wallner posted his 12-month curriculum on How to learn Deep Learning.

---

[51]: The list was originally shared on Twitter. It's since then been retweeted more than 2,000 times, including by MIT CSAIL (Computer Science and Artificial Intelligence Laboratory) and Stanford NLP (Natural Language Processing) groups.

[52]: Disclaimer: I gave a guest lecture in a version of this course in 2018, unpaid.

[53]: Disclaimer: I gave a guest lecture in a version of this course in 2019, unpaid.

4.3.2 Books & articles

1. *Data Structures and Algorithms in Python* by Michael T. Goodrich or *Introduction to Algorithms* by Thomas Cormen et al..
2. *A First Course in Probability* by Sheldon Ross.
3. *Machine Learning: A Probabilistic Perspective* by Kevin P. Murphy
4. *Information Theory, Inference, and Learning Algorithms* by David MacKay. Free online version here.
5. *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Free online version here.
6. *Introduction to Information Retrieval* by Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Essential for anyone interested in Natural Language Processing. Free online version here.
7. *Reinforcement Learning: An Introduction* by Richard S. Sutton and Andrew G. Barto. Essential for reinforcement learning. Free online version here.
8. OpenAI Spinning up in Deep Reinforcement Learning: A collection of articles that give great intuition for many RL algorithms. Highly recommended for anyone interested in RL.
9. *Convex Optimization* by Stephen Boyd and Lieven Vandenberghe. Super helpful but also super hard -- Stephen Boyd is basically a god. Free online version here.
10. *Mining of Massive Datasets* by Jure Leskovec, Anand Rajaraman, and Jeff Ullman. This book is extremely relevant as machine learning is moving towards bigger models that use massive amounts of compute and data. Free online version here.
11. Deep Learning with Hadoop & Apache Spark

    No book on the subject that I can find, but there are a few helpful articles here.

4.3.3 Other resources

1. Official tutorials from TensorFlow, PyTorch, or Keras are all great.
2. Kaggle kernels.
3. StackOverflow and StackExchange.
4. Wikipedia.
5. distill.pub.
6. Google Colab[54] seedbank.

1. Rooftopslushie.com: detailed description of interview processes at big companies.
2. Acing AI Interviews: collections of questions asked at AI interviews at big companies. No answers though.
3. Interview.io: take anonymous technical interviews and watch anonymous technical interviews.
4. Sebastian Ruder's NLP newsletters. It's mind-boggling how much information Sebastian manages to distill into each newsletter. Useful to keep track of how the field progresses.
5. Advice for Applying to Data Science Jobs by Emily Robinson, 2018.

---

[54]: As of 2021, you can access free GPU on Colab.

4.4 Do’s and don’ts for ML interviews

In summary, below are some dos and don’ts that you should keep in mind during your job search.

4.4.1 Do’s

1. Start your preparation early, even way before you start looking for your first job.
2. Get someone to refer you. The world runs on social capital. Many companies have a huge backlog of resumes and the chance of a random resume getting through the crack is slim. Having a referral won’t help you get a job that you aren’t qualified for, but will fast track you into the pipeline. For tips on how to get people to refer you, see the *Appendix: Building your network*.
3. Reach out to people in your network to let them know of your job search and discover opportunities.
4. Find a group of people who’re also interviewing and keep each other updated with your progress. It’s helpful to know what is being asked to get a sense of what the industry cares about. It’s also reassuring to know that you aren’t alone.
5. Have friends give you mock interviews. You can pick questions, both technical and non-technical, from this book for those sessions.
6. Invest time in open-source projects and publish your code on GitHub. This strengthens your resume and helps other people find you.
7. Update your LinkedIn.
8. Keep up to date with new technologies and best practices. Take courses. Read books and in-depth technical blog posts.
9. Write in-depth technical blog posts.
10. Join Kaggle competitions and read Kaggle kernels to understand how the winning teams do it. It can give you a rough understanding of what tools and techniques are being used to solve practical problems. Here’s a useful guide on how to get started with Kaggle competitions by Will Koehrsen (2018).
11. Ask questions on StackExchange and StackOverflow. These websites are amazing. You can post the most random questions and some good Samaritan on the Internet will probably spend an evening writing a detailed 3000-word answer to it.
12. Read Glassdoor reviews for each company you apply to.
13. Review all your previous work, at least what you’ve put on your resume. Interviewers might want to know all the details.
14. Before your interviews, do a few coding exercises to get your brain into the problem-solving mode.
15. Ask your recruiter for information on your interviewers so you can look up their areas of interest. Not all companies let you know in advance, but few would refuse if you ask.
16. Don’t mention anything you can’t be accountable for. Everything you mention in an interview will be fair-game for the interviewer to ask. For example, if you say that you’ve taken a course that covers LDA, the interviewer might ask you to explain LDA.
17. Think out loud. Employers are interested in not only your solutions but also how you approach a problem.
18. If you encounter a term you’re not familiar with during interviews, ask for an example.

19. Ask questions. The interview process is a two-way street -- not only a company evaluates whether you're a fit but you also evaluate whether you want to work for that company. If you don't ask questions, companies might think you're not interested.
20. Listen to the interviewer's feedback and accept their help. Interviewers want to know how well you work with others (in this case, the interviewer).
21. Ask your interviewer what skills they are trying to assess so you can tailor your answers.
22. Check your references[55].
23. Interview at a lot of companies to get practice before interviewing for your dream job.
24. The best time to interview is when you don't need a job. Even if you don't have any immediate plan to leave your current job, it's useful to schedule interviews with a couple of companies to see what is out there and brush up on your interviewing skills.
25. Have competing offers.
26. Think of interviews as part of the learning process. A rejection isn't much different from getting a question wrong. You get it wrong now doesn't mean you'll always get it wrong.
27. Thank the interviewers for their time and solicit feedback.

---

[55]: A candidate excelled in interviews but got rejected because all his references mentioned that he was a bad team player.

4.4.2 Don’ts

1. Don’t pretend that you know something. Don’t give hand-wavy explanations or dismiss a question as unimportant. If an interviewer asks you a question, it’s important to them.
2. Don’t give canned responses. If an interviewer asks why you’re interested in Artificial Intelligence, refrain from saying “AI is the new electricity.” This is your chance for your own journey and personality to shine.
3. Don’t aggressively steer the conversation to topics that you know well. It’s okay to do it subtly, but don’t ignore the interviewer’s questions. Don’t change the subject. A good interviewer will eventually guide the interview to areas that you know about.[56]
4. Don’t criticize your previous or current employers. In fact, don’t criticize anyone during your interviews. You can talk about the challenges you face when working with them, but you don’t want to be the person who finds faults in everyone. There’s this saying: if everyone around you seems to be a problem, you’re the problem.
5. Don’t be intimidated by your interviewer.
6. Don’t be afraid to disagree with your interviewers -- some might say something wrong on purpose to see if you catch it.
7. Don’t look down on your interviewer either. It’s not uncommon for the interviewer to be more junior than the candidate. They might not have your credentials or experiences, but they still know things that you don’t.
8. Don’t talk about your age, marital status, religion, political affiliation. Interviewers aren’t allowed to ask for this kind of information because it’s irrelevant to the job and has the potential for bias.
9. Don’t discuss salary with your interviewers, unless explicitly asked. In most organizations, you don’t negotiate offers with your interviewers but with your recruiter or the company’s operation staff.
10. Don’t be happy when you get easy questions. You should be concerned that you get questions below your ability because the company might be pegging you down for a position without many learning opportunities.
11. Don’t be upset when you get difficult questions. If it’s difficult for you, it’s difficult for everyone.
12. Don’t sweat it. If you flunk an interview, move on. There are plenty of other companies. If that’s your dream company, check in with them a year or so later.

---

[56] I’ve had a few candidates do that to me and it’s annoying. For example, I’d try to gauge their knowledge in dropouts for RNNs and they’d say: “Dropouts aren’t a big problem for RNNs, vanishing gradients are” and talk about vanishing gradients instead.

Part II: Questions

This part contains over 200 questions that have more or less deterministic answers. This type of question is to test your understanding of machine learning concepts. In an hour-long interview, you can cover 10 - 15 of those questions. I rank the questions by three levels of difficulty:

- **[E]**asy - fundamental, everyone should know.
- **[M]**edium - people who understand a concept beyond just the definition should know.
- **[H]**ard - people with experience should know.

Some of the knowledge questions are considered bad interview questions, especially those about definitions that can be easily looked up. For example, asking someone to explain PCA is good for evaluating their memorization of PCA, not their understanding of PCA. However, some bad interview questions can still make good questions when practicing for interviews, so I include some definition questions to remind readers that certain concepts are important.

Techniques go in and out of fashion, but fundamental challenges stay the same. Instead of asking candidates to write out complex equations for certain techniques, this book focuses on the challenges that gave rise to those techniques in the first place. Most of the questions in this section are about why something matters and how it works.

> 🌳 **Tip: Strategy for `definition` questions** 🌳
> When asked to give the definition of or explain a technique, always start with the motivation for that technique. For example, if asked to explain LSTM for recurrent neural networks, you should first bring up the problems that arise in normal RNNs and how LSTMs address those problems.

Chapter 5. Math

If the extent of your ML work will only ever consist of running `keras.fit` or cloning existing implementations, you probably don't need math. There are many courses and books that promise you machine learning mastery with little or no math at all. If that's what you're looking for, feel free to skip this chapter.

Machine learning be like

However, some mathematical background will be helpful to the following.

1. evaluate the tradeoffs of different algorithms and choose the ones that work best for your problem
2. debug your models if something goes wrong during training
3. make changes to improve your models, either in performance or efficiency, even if nothing goes wrong
4. explain certain aspects of your model performance
5. create new models.

This section covers the following branches of math that are important in ML: algebra, probability and statistics, dimensionality reduction, and very little calculus and convex optimization. This list is far from exhaustive. For example, graph theory, logic, topology, and other mathematical branches occur frequently in ML but aren't included here.

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

Notation

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

- All vectors in this book are column vectors of dimension $n \times 1$, with n being the number of elements in that vector.
- A superscript is used for a sample's index. The i^{th} element in the dataset is denoted $x^{(i)}$.
- A subscript is used for dimension. x_1 of an $n \times n$ matrix denotes its first row.

5.1 Algebra and (little) calculus

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

5.1.1 Vectors

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

1. Dot product
    i. [E] What's the geometric interpretation of the dot product of two vectors?
    ii. [E] Given a vector u, find vector v of unit length such that the dot product of u and v is maximum.
2. Outer product
    i. [E] Given two vectors $a = [3, 2, 1]$ and $b = [-1, 0, 1]$. Calculate the outer product a^Tb?
    ii. [M] Give an example of how the outer product can be useful in ML.
3. [E] What does it mean for two vectors to be linearly independent?
4. [M] Given two sets of vectors $A = a_1, a_2, a_3, \dots, a_n$ and $B = b_1, b_2, b_3, \dots, b_m$. How do you check that they share the same basis?
5. [M] Given n vectors, each of d dimensions. What is their dimensionality span?
6. Norms and metrics
    i. [E] What's a norm? What is L_0, L_1, L_2, L_{norm}?
    ii. [M] How do norm and metric differ? Given a norm, make a metric. Given a metric, can we make a norm?

5.1.2 Matrices

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

1. [E] Why do we say that matrices are linear transformations?
2. [E] What’s the inverse of a matrix? Do all matrices have an inverse? Is the inverse of a matrix always unique?
3. [E] What does the determinant of a matrix represent?
4. [E] What happens to the determinant of a matrix if we multiply one of its rows by a scalar $t \times R$?
5. [M] A 4×4 matrix has four eigenvalues $3, 3, 2, -1$. What can we say about the trace and the determinant of this matrix?
6. [M] Given the following matrix:

   $$\begin{bmatrix} 1 & 4 & -2 \\ -1 & 3 & 2 \\ 3 & 5 & -6 \end{bmatrix}$$

   Without explicitly using the equation for calculating determinants, what can we say about this matrix’s determinant?

   Hint: rely on a property of this matrix to determine its determinant.

7. [M] What’s the difference between the covariance matrix A^TA and the Gram matrix AA^T?
8. Given $A \in R^{n\times m}$ and $b \in R^n$

   i. [M] Find x such that: $Ax = b$.
   ii. [E] When does this have a unique solution?
   iii. [M] Why is it when A has more columns than rows, $Ax = b$ has multiple solutions?
   iv. [M] Given a matrix A with no inverse. How would you solve the equation $Ax = b$? What is the pseudoinverse and how to calculate it?
9. Derivative is the backbone of gradient descent.

   i. [E] What does derivative represent?
   ii. [M] What’s the difference between derivative, gradient, and Jacobian?
10. [H] Say we have the weights $w \in R^{d\times m}$ and a mini-batch x of n elements, each element is of the shape $1 \times d$ so that $x \in R^{n\times d}$. We have the output $y = f(x; w) = xw$. What’s the dimension of the Jacobian $\frac{\delta y}{\delta x}$?
11. [H] Given a very large symmetric matrix A that doesn’t fit in memory, say $A \in R^{1M\times 1M}$ and a function f that can quickly compute $f(x) = Ax$ for $x \in R^{1M}$. Find the unit vector x so that x^TAx is minimal.

    Hint: Can you frame it as an optimization problem and use gradient descent to find an approximate solution?

5.1.3 Dimensionality reduction

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

1. [E] Why do we need dimensionality reduction?
2. [E] Eigendecomposition is a common factorization technique used for dimensionality reduction. Is the eigendecomposition of a matrix always unique?
3. [M] Name some applications of eigenvalues and eigenvectors.
4. [M] We want to do PCA on a dataset of multiple features in different ranges. For example, one is in the range 0-1 and one is in the range 10 - 1000. Will PCA work on this dataset?
5. [H] Under what conditions can one apply eigendecomposition? What about SVD?
    i. What is the relationship between SVD and eigendecomposition?
    ii. What’s the relationship between PCA and SVD?
6. [H] How does t-SNE (T-distributed Stochastic Neighbor Embedding) work? Why do we need it?

> **In case you need a refresh on PCA, here's an explanation without any math.**
>
> Assume that your grandma likes wine and would like to find characteristics that best describe wine bottles sitting in her cellar. There are many characteristics we can use to describe a bottle of wine including age, price, color, alcoholic content, sweetness, acidity, etc. Many of these characteristics are related and therefore redundant. Is there a way we can choose fewer characteristics to describe our wine and answer questions such as: which two bottles of wine differ the most?
>
> PCA is a technique to construct new characteristics out of the existing characteristics. For example, a new characteristic might be computed as `age - acidity + price` or something like that, which we call a linear combination.
>
> To differentiate our wines, we'd like to find characteristics that strongly differ across wines. If we find a new characteristic that is the same for most of the wines, then it wouldn't be very useful. PCA looks for characteristics that show as much variation across wines as possible, out of all linear combinations of existing characteristics. These constructed characteristics are principal components of our wines.
>
> If you want to see a more detailed, intuitive explanation of PCA with visualization, check out amoeba's answer on StackOverflow. This is possibly the best PCA explanation I've ever read.

5.1.4 Calculus and convex optimization

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

1. Differentiable functions
    i. [E] What does it mean when a function is differentiable?
    ii. [E] Give an example of when a function doesn't have a derivative at a point.
    iii. [M] Give an example of non-differentiable functions that are frequently used in machine learning. How do we do backpropagation if those functions aren't differentiable?
2. Convexity
    i. [E] What does it mean for a function to be convex or concave? Draw it.
    ii. [E] Why is convexity desirable in an optimization problem?
    iii. [M] Show that the cross-entropy loss function is convex.
3. Given a logistic discriminant classifier:

   $$p(y=1|x) = \sigma(w^T x)$$

   where the sigmoid function is given by:

   $$\sigma(z) = (1 + \exp(-z))^{-1}$$

   The logistic loss for a training sample x_i with class label y_i is given by:

   $$L(y_i, x_i; w) = -\log p(y_i|x_i)$$

    i. Show that $p(y=-1|x) = \sigma(-w^T x)$.
    ii. Show that $\Delta_w L(y_i, x_i; w) = -y_i(1 - p(y_i|x_i))x_i$.
    iii. Show that $\Delta_w L(y_i, x_i; w)$ is convex.
4. Most ML algorithms we use nowadays use first-order derivatives (gradients) to construct the next training iteration.

    i. [E] How can we use second-order derivatives for training models?
    ii. [M] Pros and cons of second-order optimization.
    iii. [M] Why don't we see more second-order optimization in practice?
5. [M] How can we use the Hessian (second derivative matrix) to test for critical points?
6. [E] Jensen's inequality forms the basis for many algorithms for probabilistic inference, including Expectation-Maximization and variational inference.. Explain what Jensen's inequality is.
7. [E] Explain the chain rule.
8. [M] Let $x \in R_n$, $L = crossentropy(softmax(x), y)$ in which y is a one-hot vector. Take the derivative of L with respect to x.
9. [M] Given the function $f(x, y) = 4x^2 - y$ with the constraint $x^2 + y^2 = 1$. Find the function's maximum and minimum values.

> On convex optimization

Convex optimization is important because it's the only type of optimization that we more or less understand. Some might argue that since many of the common objective functions in deep learning aren't convex, we don't need to know about convex optimization. However, even when the functions aren't convex, analyzing them as if they were convex often gives us meaningful bounds. If an algorithm doesn't work assuming that a loss function is convex, it definitely doesn't work when the loss function is non-convex.

Convexity is the exception, not the rule. If you're asked whether a function is convex and it isn't already in the list of commonly known convex functions, there's a good chance that it isn't convex. If you want to learn about convex optimization, check out Stephen Boyd's textbook.

> On Hessian matrix

The Hessian matrix or Hessian is a square matrix of second-order partial derivatives of a scalar-valued function.

Given a function $f : \mathbb{R}n \to \mathbb{R}$. If all second partial derivatives of f exist and are continuous over the domain of the function, then the Hessian matrix H of f is a square nn matrix such that: $H_{ij} = \frac{\delta f}{\delta x_i \delta x_j}$.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

The Hessian is used for large-scale optimization problems within Newton-type methods and quasi-Newton methods. It is also commonly used for expressing image processing operators in image processing and computer vision for tasks such as blob detection and multi-scale signal representation.

5.2 Probability and statistics

A Reddit user once said: “Data science is just doing statistics on a Mac.” Knowledge of probability and statistics is extremely important in ML and data science. If you don’t understand cross-entropy, KL divergence, or just general probability distribution, most of the objective functions in ML will statistically make little sense.

5.2.1 Probability

A likely question would be to explain any of the common distributions and write out its equation, draw its probability mass function (PMF) if it’s discrete and the probability density function (PDF) if it’s continuous. It’d be useful to review all the common distributions.

5.2.1.1 Basic concepts to review

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

Random variable

Formally, a random variable is a measurable function $X : \Omega \to E$ from a set of possible outcomes Ω to a measurable space E. The probability that X takes on a value in a measurable set $S \subseteq E$ is written as:

$P(X \in S) = P(\omega \in \Omega | X(\omega) \in S)$, where P is the probability measure on (Ω, F).

The randomness comes from the randomness of the outcomes in Ω.

Informally, a random variable is a variable that probabilistically takes on different values. You can think of a random variable as being like a variable in a programming language. They take on values, have types, and have domains over which they are applicable.

Random variable is a general concept. Almost everything in life can be described using a random variable. The time it takes you to commute to work is a normal random variable. The number of people you date before finding your life partner is a geometric random variable.

Probability distribution

A probability distribution is a function that describes possible outcomes of a random variable along with its corresponding probabilities.

Normal random variable

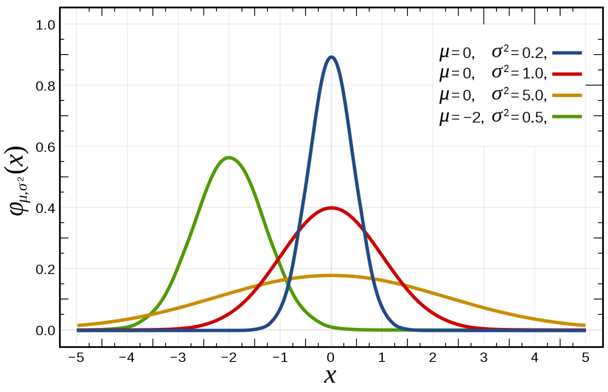
Also known as the Gaussian random variable, this is the single most important random variable. It's parameterized by a random variable, parametrized by a mean μ and variance σ^2.

$$X \sim N(\mu, \sigma^2)$$
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

The term we want to go over in this function is $-\frac{(x-\mu)^2}{2\sigma^2}$. Intuitively, it punishes the value far away from the mean, but the punishment is less when the variance is high. The term $\frac{1}{\sigma\sqrt{2\pi}}$ is normalization so that it integrates to 1.

$$E[X] = \mu$$
$$Var(X) = \sigma^2$$

Here is the PDF of the normal distribution with different parameters.



Categorical distribution

Also known as the multinoulli distribution, the categorical distribution is a generalization of the Bernoulli distribution. It describes the possible results of a random variable that can take on one of k possible categories, with the probability of each category separately specified.

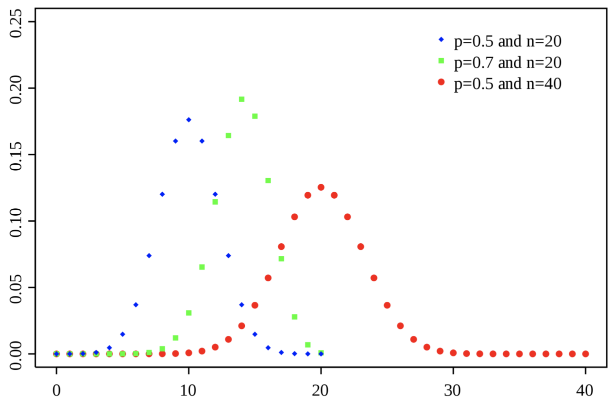
$$X \in \text{Cat}(\phi); \phi = (p_1, p_2, \ldots, p_k) \text{ and } \sum_{i=1}^{k} p_i = 1$$

Binomial random variable

A binomial random variable represents the number of successes in n successive independent trials, each succeeding with probability p and failing with probability $1 - p$. One example is the number of heads in n coin flips, each with a 0.5 probability of landing head. The binomial distribution is the basis for the binomial test for statistical significance. When there's only 1 trial, it's known as the Bernoulli distribution.

$$X \sim \text{Bin}(n, p)$$
$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$
$$E[X] = np$$
$$Var(X) = np(1-p)$$

Below is the PMF of the binomial distribution with different parameters.



Multinomial random variable

The multinomial random variable is a generalization of the binomial distribution. Instead of having only two outcomes like with a coin flip, it can have multiple outcomes like with a k-sided die. When the number of trials is 1, it's the categorical distribution.

$$X \sim \text{Multi}(n, \pi) \text{ with } \pi = (p_1, p_2, \ldots, p_k) \text{ and } \sum_{i=1}^{k} p_i = 1$$
$$P(X = (x_1, x_2, \ldots, x_k)) = \frac{n!}{x_1!x_2!\ldots x_k!} \prod_{i=1}^{k} p_i x_i \text{ with } n = \sum_{i}^{k} x_i$$
$$E[X_i] = np_i$$
$$Var(X_i) = np_i(1 - p_i)$$

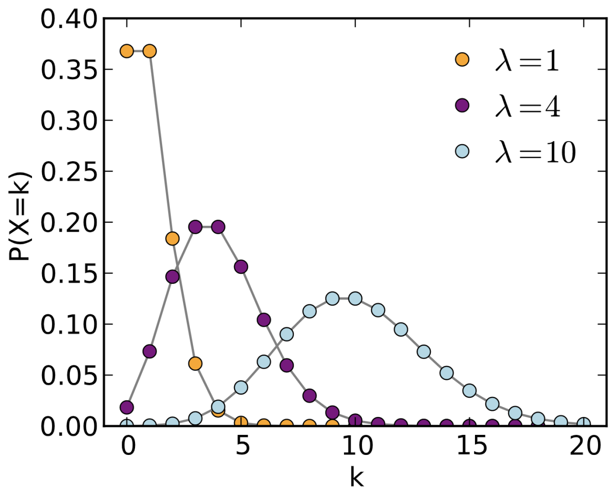
Poisson random variable

The Poisson distribution is, in my opinion, among the more interesting distributions. It expresses the probability of a given number of events occurring in a fixed interval if these events occur with a known constant rate. This rate is denoted as λ. Note that the Poisson distribution is *memoryless*, which means the probability that an event occurs is independent of the time since the last event.

One pretty neat perspective is to see the Poisson distribution as an approximation of the Binomial where n is large, p is small, and $\lambda = np$. For example, a Binomial random variable of 10000 trials with the success rate of 0.01 can be seen as a Poisson random variable of events happening every 10000 * 0.01 = 100 trials.

$$X \sim \text{Poi}(\lambda)$$
$$P(X = i) = \frac{\lambda^i}{i!} e^{-\lambda}$$
$$E[X] = \lambda$$
$$Var(X) = \lambda$$

Below is the PMF of the Poisson distribution with different values of λ, made by Skbkekas.



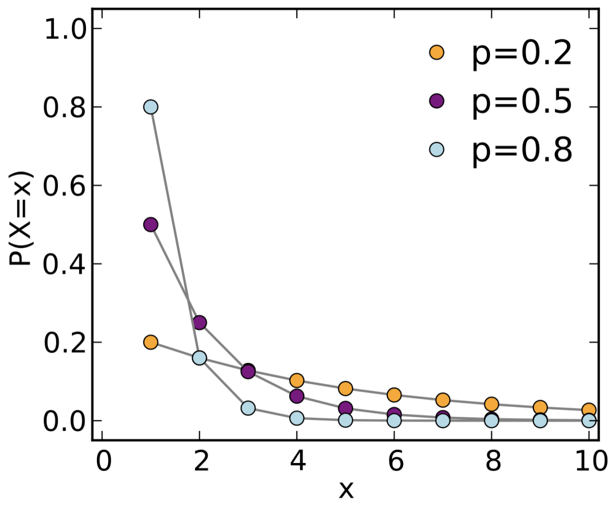
> **Poisson vs binomial according to Data Science Central**:
>
> If your question has an average probability of an event happening per unit (i.e. per unit of time, cycle, event) and you want to find the probability of a certain number of events happening in a period of time (or a number of events), then use the Poisson Distribution.
>
> If you are given an exact probability and you want to find the probability of the event happening a certain number of times out of x (i.e. 10 times out of 100, or 99 times out of 1000), use the Binomial Distribution formula.

Geometric random variable

If each trial has the probability of success of p, then the geometric random variable represents the number of independent trials until the first success. One example is the number of candidates you have to interview until you hire someone.

$$\begin{aligned} X &\sim \text{Geo}(p) \\ P(X=n) &= (1-p)^{n-1}p \\ E[X] &= \frac{1}{p} \\ Var(X) &= \frac{1-p}{p^2} \end{aligned}$$

Below is the PMF of the geometric distribution with different values of p, made by Skbkekas.



Beta random variable

Beta is my favorite distribution (what do you mean you don't have a favorite distribution?). It's a random variable that estimates another random variable.

Say, we have a coin with an unknown probability of turning heads. Let p represent this probability. After $n+m$ flips, we get n heads and m tails. We might want to estimate that $p = \frac{n}{n+m}$. However, this is unreliable, especially if $n+m$ is small. We'd like to say something like this: p can also be more than, less than, or equal to $\frac{n}{n+m}$, the values further away from $\frac{n}{n+m}$ having a smaller probability. And the higher the value of $n+m$, the higher the probability of p being $\frac{n}{n+m}$. The beta distribution allows you to do that.

The beta random variable is represented using two variables: α to represent the number of successes and β to represent the number of failures. The beta distribution can represent beyond coin flips. In fact, α and β can rerepsent continuous value (though they can't be non-positive).

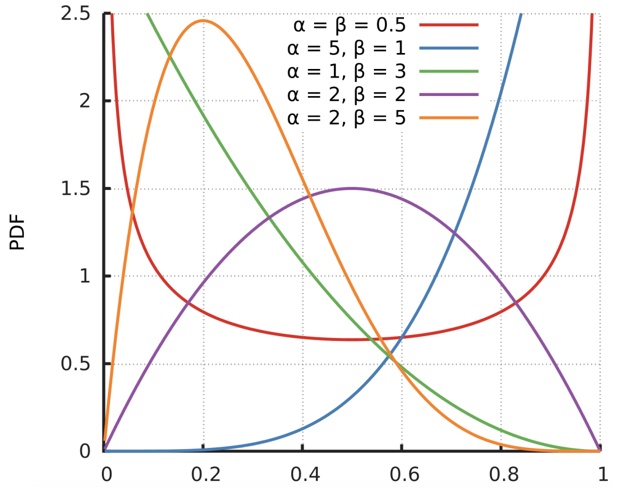
$$x \sim \text{Beta}(\alpha, \beta) \text{ with } 0 < \alpha, \beta$$
$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \text{ with } 0 \le x \le 1$$

Γ is the Gamma function: $\Gamma(n) = (n-1)!$. The term $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$ is the normalization constant so that the expression integrates to 1.

$$E[X] = \frac{\alpha}{\alpha+\beta}$$
$$Var(X) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

We can also incorporate a priori belief in the beta distribution. For example, if before even flipping the coin, we believe that the coin has a moderate chance of being biased towards heads, we can set the a priori to be $\text{Beta}(4, 2)$. Then after 10 coin flips of which 7 are heads and 3 are tails, we can update the distribution to be $\text{Beta}(4+7, 2+3) = \text{Beta}(11, 5)$. In fact, in Bayesian inference, the beta distribution is the conjugate prior probability distribution for Bernoulli, binomial, negative binomial, and geometric distributions.

Below is the PDF of the geometric distribution with different parameters.



The multivariate generalization of the beta random variable is called Dirichlet.

Exponential family of distributions

A class of distributions is in the exponential family if it can be written in the form:

$$p(x; \eta) = h(x) \exp(B(\eta)T(x) - a(\eta))$$

where:

1. η is the parameter of the distribution, e.g. if it's the normal distribution then $\eta = (\mu, \sigma)$.
2. $T(x)$ is the sufficient statistic. For most distributions, $T(x) = x$. For the beta distribution, $T(x) = (\log x, \log(1-x))$.
3. $a(\eta)$ is the normalization constant so that the expression integrates to 1.

The exponential family of distribution is important because it provides a general framework for working with many of the most common distributions, including the Bernoulli, binomial, Poisson, normal, and more.

You can write the PMF and PDF of those distributions to match the form defined above. For example, given the Poisson random variable $\mathrm{Poi}(\lambda)$, with $f(x) = \frac{\lambda^x}{x!}\exp(-\lambda)$, belongs to the exponential family because $f(x)$ can be written in the form $h(x)\exp(B(\eta)T(x) - a(\eta))$.

$$
\begin{gathered}
f(x) = \frac{\lambda^x}{x!}\exp(-\lambda) = \frac{1}{x!}\exp(\log(\lambda)x - \lambda) \\
\eta = \lambda \\
h(x) = \frac{1}{x!} \\
B(\eta) = \log(\eta) \\
T(x) = x \\
a(\eta) = \lambda
\end{gathered}
$$

More examples that show that other distributions belong to the exponential family can be found here and here.

Marginal distribution, joint distribution, conditional distribution

A joint probability distribution gives the probability of two or more events happening at the same time. For example, given two discrete random variables X and Y, the joint probability distribution of X and Y gives the probability of $P(X = x, Y = y)$ for any combination of value $x \in X$ and value $y \in Y$.

A marginal distribution gives the probabilities of various values of a subset of variables without reference to the values of the other variables. For example, given the joint distribution of X and Y, we want to have a marginal probability distribution of X without reference to Y.

$$P(X = x) = \sum_y P(X = x, Y = y)$$

A conditional probability distribution gives the probability of a subset of events occurring assuming that other events also occur. One example is $P(X|Y)$.

5.2.1.2 Questions

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

1. [E] Given a uniform random variable X in the range of $[0, 1]$ inclusively. What's the probability that $X = 0.5$?
2. [E] Can the values of PDF be greater than 1? If so, how do we interpret PDF?
3. [E] What's the difference between multivariate distribution and multimodal distribution?
4. [E] What does it mean for two variables to be independent?
5. [E] It's a common practice to assume an unknown variable to be of the normal distribution. Why is that?
6. [E] How would you turn a probabilistic model into a deterministic model?
7. [H] Is it possible to transform non-normal variables into normal variables? How?
8. [M] When is the t-distribution useful?
9. Assume you manage an unreliable file storage system that crashed 5 times in the last year, each crash happens independently.
   i. [M] What's the probability that it will crash in the next month?
   ii. [M] What's the probability that it will crash at any given moment?
10. [M] Say you built a classifier to predict the outcome of football matches. In the past, it's made 10 wrong predictions out of 100. Assume all predictions are made independently., what's the probability that the next 20 predictions are all correct?
11. [M] Given two random variables X and Y. We have the values $P(X|Y)$ and $P(Y)$ for all values of X and Y. How would you calculate $P(X)$?
12. [M] You know that your colleague Jason has two children and one of them is a boy. What's the probability that Jason has two sons? Hint: it's not $\frac{1}{2}$.
13. There are only two electronic chip manufacturers: A and B, both manufacture the same amount of chips. A makes defective chips with a probability of 30%, while B makes defective chips with a probability of 70%.
    i. [E] If you randomly pick a chip from the store, what is the probability that it is defective?
    ii. [M] Suppose you now get two chips coming from the same company, but you don't know which one. When you test the first chip, it appears to be functioning. What is the probability that the second electronic chip is also good?
14. There's a rare disease that only 1 in 10000 people get. Scientists have developed a test to diagnose the disease with the false positive rate and false negative rate of 1%.
    i. [E] Given a person is diagnosed positive, what's the probability that this person actually has the disease?
    ii. [M] What's the probability that a person has the disease if two independent tests both come back positive?
15. [M] A dating site allows users to select 10 out of 50 adjectives to describe themselves. Two users are said to match if they share at least 5 adjectives. If

Jack and Jin randomly pick adjectives, what is the probability that they match?

16. [M] Consider a person A whose sex we don't know. We know that for the general human height, there are two distributions: the height of males follows $h_m = N(\mu_m, \sigma_m^2)$ and the height of females follows $h_j = N(\mu_j, \sigma_j^2)$. Derive a probability density function to describe A's height.
17. [H] There are three weather apps, each the probability of being wrong ⅓ of the time. What's the probability that it will be foggy in San Francisco tomorrow if all the apps predict that it's going to be foggy in San Francisco tomorrow and during this time of the year, San Francisco is foggy 50% of the time?

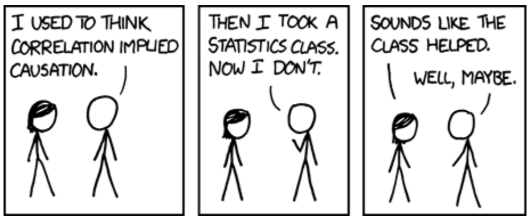
    Hint: you'd need to consider both the cases where all the apps are independent and where they are dependent.

18. [M] Given n samples from a uniform distribution $[0, d]$. How do you estimate d? (Also known as the German tank problem)
19. [M] You're drawing from a random variable that is normally distributed, $X \sim N(0, 1)$, once per day. What is the expected number of days that it takes to draw a value that's higher than 0.5?
20. [M] You're part of a class. How big the class has to be for the probability of at least a person sharing the same birthday with you is greater than 50%?
21. [H] You decide to fly to Vegas for a weekend. You pick a table that doesn't have a bet limit, and for each game, you have the probability p of winning, which doubles your bet, and $1 - p$ of losing your bet. Assume that you have unlimited money (e.g. you bought Bitcoin when it was 10 cents), is there a betting strategy that has a guaranteed positive payout, regardless of the value of p?
22. [H] Given a fair coin, what's the number of flips you have to do to get two consecutive heads?
23. [H] In national health research in the US, the results show that the top 3 cities with the lowest rate of kidney failure are cities with populations under 5,000. Doctors originally thought that there must be something special about small town diets, but when they looked at the top 3 cities with the highest rate of kidney failure, they are also very small cities. What might be a probabilistic explanation for this phenomenon?

    Hint: The law of small numbers.

24. [M] Derive the maximum likelihood estimator of an exponential distribution.

5.2.2 Stats


I USED TO THINK CORRELATION IMPLIED CAUSATION.
THEN I TOOK A STATISTICS CLASS. NOW I DON'T.
SOUNDS LIKE THE CLASS HELPED.
WELL, MAYBE.

From xkcd. To distract yourself from interviewing stress, here are more statistics jokes.

1. [E] Explain frequentist vs. Bayesian statistics.
2. [E] Given the array $[1, 5, 3, 2, 4, 4]$, find its mean, median, variance, and standard deviation.
3. [M] When should we use median instead of mean? When should we use mean instead of median?
4. [M] What is a moment of function? Explain the meanings of the zeroth to fourth moments.
5. [M] Are independence and zero covariance the same? Give a counterexample if not.
6. [E] Suppose that you take 100 random newborn puppies and determine that the average weight is 1 pound with the population standard deviation of 0.12 pounds. Assuming the weight of newborn puppies follows a normal distribution, calculate the 95% confidence interval for the average weight of all newborn puppies.
7. [M] Suppose that we examine 100 newborn puppies and the 95% confidence interval for their average weight is $[0.9, 1.1]$ pounds. Which of the following statements is true?

   i. Given a random newborn puppy, its weight has a 95% chance of being between 0.9 and 1.1 pounds.
   ii. If we examine another 100 newborn puppies, their mean has a 95% chance of being in that interval.
   iii. We're 95% confident that this interval captured the true mean weight.

   Hint: This is a subtle point that many people misunderstand. If you struggle with the answer, Khan Academy has a great article on it.

8. [H] Suppose we have a random variable X supported on $[0, 1]$ from which we can draw samples. How can we come up with an unbiased estimate of the median of X?
9. [H] Can correlation be greater than 1? Why or why not? How to interpret a correlation value of 0.3?
10. The weight of newborn puppies is roughly symmetric with a mean of 1 pound and a standard deviation of 0.12. Your favorite newborn puppy weighs 1.1 pounds.
    i. [E] Calculate your puppy's z-score (standard score).
    ii. [E] How much does your newborn puppy have to weigh to be in the top 10% in terms of weight?

iii. [M] Suppose the weight of newborn puppies followed a skew distribution. Would it still make sense to calculate z-scores?

11. [H] Tossing a coin ten times resulted in 10 heads and 5 tails. How would you analyze whether a coin is fair?
12. Statistical significance.
    i. [E] How do you assess the statistical significance of a pattern whether it is a meaningful pattern or just by chance?
    ii. [E] What's the distribution of p-values?
    iii. [H] Recently, a lot of scientists started a war against statistical significance. What do we need to keep in mind when using p-value and statistical significance?
13. Variable correlation.
    i. [M] What happens to a regression model if two of their supposedly independent variables are strongly correlated?
    ii. [M] How do we test for independence between two categorical variables?
    iii. [H] How do we test for independence between two continuous variables?
14. [E] A/B testing is a method of comparing two versions of a solution against each other to determine which one performs better. What are some of the pros and cons of A/B testing?
15. [M] You want to test which of the two ad placements on your website is better. How many visitors and/or how many times each ad is clicked do we need so that we can be 95% sure that one placement is better?
16. [M] Your company runs a social network whose revenue comes from showing ads in newsfeed. To double revenue, your coworker suggests that you should just double the number of ads shown. Is that a good idea? How do you find out?
17. Imagine that you have the prices of 10,000 stocks over the last 24 month period and you only have the price at the end of each month, which means you have 24 price points for each stock. After calculating the correlations of 10,000 * 9,9992 pairs of stock, you found a pair that has the correlation to be above 0.8.

    i. [E] What's the probability that this happens by chance?
    ii. [M] How to avoid this kind of accidental patterns?

    Hint: Check out the curse of big data.

18. [H] How are sufficient statistics and Information Bottleneck Principle used in machine learning?

This book was created by Chip Huyen with the help of wonderful friends. For feedback, errata, and suggestions, the author can be reached here.

Chapter 6. Computer Science

For coding questions, the best way to prepare is to write code every day. Programming mastery requires consistent practice.

However, the type of coding you do every day is different from the type of coding asked during interviews, so it’s a good idea to get some practice. Before your interviews, do a few typical software engineering questions to get yourself into the problem-solving, whiteboard coding mode.

If you haven’t seen them in a while, pick up a good book on algorithms and data structures and skim it. We recommend the book *Data Structures and Algorithms in Python* by Michael T. Goodrich and the classic *Introduction to Algorithms* by Thomas Cormen et al.

We also recommend practicing websites such as LeetCode, CodeSignal, and HackerRank. Those sites rank problems by difficulty -- you should try to solve medium and hard problems. Most of them have solutions available in case you want to compare your solutions to more optimal ones[1]

This chapter addresses the three major aspects of computer science that are covered in machine learning interviews: algorithms, complexity and numerical analysis, and data, which includes data structures.

> 🌳 **What programming language to use during interviews** 🌳
>
> If you're comfortable with only one language, feel no qualms using it during interviews. If you're comfortable with multiple languages, listen to the question first before choosing a language. It shows that you understand that different languages are built for different purposes: a language suitable for one task might not be optimal for another.
>
> Put all the languages you know on your resume but don't feel the need to show them off during interviewers. Employers would rather hire someone really good at one language -- it means that you can learn to be good at other languages -- than hiring someone mediocre at multiple languages.
>
> Based on the language you choose, interviewers might infer what you're interested in. For example, a hiring manager told me that if a candidate chooses to implement data structures in Python, he knows that this candidate doesn't focus on performance.
>
> Python has become the de facto lingua franca of machine learning -- most frameworks have Python APIs and most open-source projects are written in Python. It's a useful language to know and most interviewers probably expect it, but don't feel like you have to use it during interviews. If you're more comfortable with another language, use it. Writing a complex model in another language, say Swift, is a lot more impressive and helps you stand out.
>
> It also helps if you know at least one performance-oriented language such as C++ or Go. C++ is more popular with more support, but Go is easier to learn and manage. Since more and more machine learning models are being served as web applications, many startups look for machine learning engineers with front-end skills. Fluency in TypeScript or React is a huge plus.

6.1 Algorithms

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

Examples of classic algorithms you should know include various sorting algorithms (quicksort, radix sort), shortest path algorithms (Dijkstra’s, A*), tree algorithms (pre-, in-, post-order traversal), and solutions to popular problems such as the stable marriage problem and traveling salesman problem. You will probably never have to implement them since there already exist many efficient implementations, but it’s important to understand their underlying design principles and implementation tradeoffs in case you have to make similar decisions in your job.

There are also programming techniques that you should be comfortable with, such as dynamic programming, recursions, string manipulation, matrix multiplication, regular expression, and memory allocation. Below are some of the questions that you might want to go over to refresh your memory.

1. Write a Python function to recursively read a JSON file.
2. Implement an $O(N \log N)$ sorting algorithm, preferably quick sort or merge sort.
3. Find the longest increasing subsequence in a string.
4. Find the longest common subsequence between two strings.
5. Traverse a tree in pre-order, in-order, and post-order.
6. Given an array of integers and an integer k, find the total number of continuous subarrays whose sum equals k. The solution should have $O(N)$ runtime.
7. There are two sorted arrays $nums1$ and $nums2$ with m and n elements respectively. Find the median of the two sorted arrays. The solution should have $O(\log(m+n))$ runtime.
8. Write a program to solve a Sudoku puzzle by filling the empty cells. The board is of the size 9×9. It contains only 1-9 numbers. Empty cells are denoted with *. Each board has one unique solution.
9. Given a memory block represented by an empty array, write a program to manage the dynamic allocation of that memory block. The program should support two methods: `malloc()` to allocate memory and `free()` to free a memory block.
10. Given a string of mathematical expression, such as `10 * 4 + (4 + 3) / (2 - 1)`, calculate it. It should support four operators `+`, `-`, `:`, `/`, and the brackets `()`.
11. Given a directory path, descend into that directory and find all the files with duplicated content.
12. In Google Docs, you have the `Justify alignment` option that spaces your text to align with both left and right margins. Write a function to print out a given text line-by-line (except the last line) in Justify alignment format. The length of a line should be configurable.

13. You have 1 million text files, each is a news article scraped from various news sites. Since news sites often report the same news, even the same articles, many of the files have content very similar to each other. Write a program to filter out these files so that the end result contains only files that are sufficiently different from each other in the language of your choice. You’re free to choose a metric to define the “similarity” of content between files.

6.2 Complexity and numerical analysis

If some characters seem to be missing, it's because MathJax is not loaded correctly. Refreshing the page should fix it.

Given that most of the recent breakthroughs in machine learning come from bigger models that require massive memory and computational power, it's important to not only know how to implement a model but also how to scale it. To scale a model, we'd need to be able to estimate memory requirement and computational cost, as well as mitigate numerical instability when training and serving machine learning models. Here are some of the questions that can be asked to evaluate your understanding of numerical stability and scalability.

1. Matrix multiplication
    i. [E] You have three matrices: $A \in R^{100 \times 5}, B \in R^{5 \times 200}, C \in R^{200 \times 20}$ and you need to calculate the product ABC. In what order would you perform your multiplication and why?
    ii. [M] Now you need to calculate the product of N matrices $A_1 A_2 \dots A_n$. How would you determine the order in which to perform the multiplication?
2. [E] What are some of the causes for numerical instability in deep learning?
3. [E] In many machine learning techniques (e.g. batch norm), we often see a small term ϵ added to the calculation. What's the purpose of that term?
4. [E] What made GPUs popular for deep learning? How are they compared to TPUs?
5. [M] What does it mean when we say a problem is intractable?
6. [H] What are the time and space complexity for doing backpropagation on a recurrent neural network?
7. [H] Is knowing a model's architecture and its hyperparameters enough to calculate the memory requirements for that model?
8. [H] Your model works fine on a single GPU but gives poor results when you train it on 8 GPUs. What might be the cause of this? What would you do to address it?
9. [H] What benefits do we get from reducing the precision of our model? What problems might we run into? How to solve these problems?
10. [H] How to calculate the average of 1M floating-point numbers with minimal loss of precision?
11. [H] How should we implement batch normalization if a batch is spread out over multiple GPUs?
12. [M] Given the following code snippet. What might be a problem with it? How would you improve it? **Hint**: this is an actual question asked on StackOverflow.

```
import numpy as np

def within_radius(a, b, radius):
    if np.linalg.norm(a - b) < radius:
        return 1
    return 0

def make_mask(volume, roi, radius):
    mask = np.zeros(volume.shape)
    for x in range(volume.shape[0]):
        for y in range(volume.shape[1]):
            for z in range(volume.shape[2]):
                mask[x, y, z] = within_radius((x, y, z), roi, radius)
    return mask
```

6.3 Data

In an academic or research setting, you likely only work with clean, readily available datasets and therefore can afford to spend most of your time on modeling. In production, it’s likely that you will spend most of your time on the data pipeline. The ability to manage, process, and monitor data will make you attractive to potential employers.

In your interviews, you might be asked questions that evaluate how comfortable you are with working with data. At a high level, you should be familiar with reading, writing, and serializing different types of data. You should have your go-to library for dataframe manipulation: `pandas` is popular for general data applications, and `dask` is a good option if you want something GPU-compatible. You should be comfortable with at least a visualization library such as `seaborn`, `matplotlib`, `Tableau`, or `ggplot`.

If you want to work with big data, it doesn’t hurt to familiarize yourself with distributed data management systems such as Spark and Hadoop.

Beyond Python, SQL is still ubiquitous for all applications that require persistent databases, and while R isn’t the sexiest language, it’s handy for quick data analytics.

6.3.1 Data structures

If our world is run by data, then data structures are what keep us from descending into chaos. From the dawn of the digital age, the best minds of computer science have kept themselves up at night thinking of efficient ways to store and manipulate data. Data structures are even more important in machine learning as the field is fueled by big data.

While there are classical data structures that have stood the test of time, developing new data structures and improving on the existing ones are never-ending battles as new formats are introduced and new data are generated at a scale never seen before. Your familiarity with existing data structures, understanding of how they are implemented, and intuition on what data structures to use and when to use them will be highly valuable.

Some of the data structures whose runtime complexities you should know and that you should be able to implement in at least one language:

- Trees: binary search tree, heap, trie (prefix and suffix tree)
- Queues, stacks, priority queues
- Linked lists
- HashMap and HashTable

We don’t have questions about those data structures here, but you should try to implement them yourself, either on a coding exercise website or locally, and compare with known implementation.

You should be comfortable with manipulating popular data formats such as the ubiquitous CSV format and web- and serialization-friendly JSON format. Both CSV and JSON are examples of the traditional row-based file formats: data is stored and often indexed row-by-row.

In recent years, the column-based format has become more and more common, as it allows big data applications to quickly extract one feature from all the data points by calling the column corresponding to that feature. Popular data frameworks for machine learning include `pandas` and `dask` are optimized for column-based operations. The two common column-based file formats are Parquet, championed by Apache Hadoop, and ORC, championed by Apache Hive.

Row-based data formats are more efficient for writing while column-based formats are more efficient for reading. If your data is write-once-read-many, use column-based. If it requires regular rewriting, opt for row-based.

For more detail on data engineering for machine learning, check out the lecture note on Data Engineering for the course Machine Learning Systems Design.

Chapter 7. Machine learning workflows

Even though deep learning seems to be all that people in the research community is talking about, most real-world problems are still being solved by classical machine learning algorithms including k-nearest neighbor and XGBoost. In this chapter, we will cover fundamentals that are essential for understanding machine learning algorithms, as well as non-deep learning algorithms that you might find useful in both your day-to-day jobs and interviews.

7.1 Basics

1. [E] Explain supervised, unsupervised, weakly supervised, semi-supervised, and active learning.
2. Empirical risk minimization.
    i. [E] What's the risk in empirical risk minimization?
    ii. [E] Why is it empirical?
    iii. [E] How do we minimize that risk?
3. [E] Occam's razor states that when the simple explanation and complex explanation both work equally well, the simple explanation is usually correct. How do we apply this principle in ML?
4. [E] What are the conditions that allowed deep learning to gain popularity in the last decade?
5. [M] If we have a wide NN and a deep NN with the same number of parameters, which one is more expressive and why?
6. [H] The Universal Approximation Theorem states that a neural network with 1 hidden layer can approximate any continuous function for inputs within a specific range. Then why can't a simple neural network reach an arbitrarily small positive error?
7. [E] What are saddle points and local minima? Which are thought to cause more problems for training large NNs?
8. Hyperparameters.
    i. [E] What are the differences between parameters and hyperparameters?
    ii. [E] Why is hyperparameter tuning important?
    iii. [M] Explain algorithm for tuning hyperparameters.
9. Classification vs. regression.
    i. [E] What makes a classification problem different from a regression problem?
    ii. [E] Can a classification problem be turned into a regression problem and vice versa?
10. Parametric vs. non-parametric methods.
    i. [E] What's the difference between parametric methods and non-parametric methods? Give an example of each method.
    ii. [H] When should we use one and when should we use the other?
11. [M] Why does ensembling independently trained models generally improve performance?
12. [M] Why does L1 regularization tend to lead to sparsity while L2 regularization pushes weights closer to 0?
13. [E] Why does an ML model's performance degrade in production?
14. [M] What problems might we run into when deploying large machine learning models?
15. Your model performs really well on the test set but poorly in production.
    i. [M] What are your hypotheses about the causes?
    ii. [H] How do you validate whether your hypotheses are correct?
    iii. [M] Imagine your hypotheses about the causes are correct. What would you do to address them?

7.2 Sampling and creating training data

1. [E] If you have 6 shirts and 4 pairs of pants, how many ways are there to choose 2 shirts and 1 pair of pants?
2. [M] What is the difference between sampling with vs. without replacement? Name an example of when you would use one rather than the other?
3. [M] Explain Markov chain Monte Carlo sampling.
4. [M] If you need to sample from high-dimensional data, which sampling method would you choose?
5. [H] Suppose we have a classification task with many classes. An example is when you have to predict the next word in a sentence -- the next word can be one of many, many possible words. If we have to calculate the probabilities for all classes, it'll be prohibitively expensive. Instead, we can calculate the probabilities for a small set of candidate classes. This method is called candidate sampling. Name and explain some of the candidate sampling algorithms.

   Hint: check out this great article on candidate sampling by the TensorFlow team.

6. Suppose you want to build a model to classify whether a Reddit comment violates the website's rule. You have 10 million unlabeled comments from 10K users over the last 24 months and you want to label 100K of them.

   i. [M] How would you sample 100K comments to label?
   ii. [M] Suppose you get back 100K labeled comments from 20 annotators and you want to look at some labels to estimate the quality of the labels. How many labels would you look at? How would you sample them?

      Hint: This article on different sampling methods and their use cases might help.

7. [M] Suppose you work for a news site that historically has translated only 1% of all its articles. Your coworker argues that we should translate more articles into Chinese because translations help with the readership. On average, your translated articles have twice as many views as your non-translated articles. What might be wrong with this argument?

   Hint: think about selection bias.

8. [M] How to determine whether two sets of samples (e.g. train and test splits) come from the same distribution?

9. [H] How do you know you've collected enough samples to train your ML model?
10. [M] How to determine outliers in your data samples? What to do with them?
11. Sample duplication
    i. [M] When should you remove duplicate training samples? When shouldn't you?
    ii. [M] What happens if we accidentally duplicate every data point in your train set or in your test set?

12. Missing data
    i. [H] In your dataset, two out of 20 variables have more than 30% missing values. What would you do?
    ii. [M] How might techniques that handle missing data make selection bias worse? How do you handle this bias?
13. [M] Why is randomization important when designing experiments (experimental design)?
14. Class imbalance.
    i. [E] How would class imbalance affect your model?
    ii. [E] Why is it hard for ML models to perform well on data with class imbalance?
    iii. [M] What do you do when your training data has class imbalance?
    iv. [M] Imagine you're training a model to classify skin legion. Only 1 ...
15. Training data leakage.
    i. [M] Why would oversampling before splitting can cause data leakage?
    ii. [M] You want to build a model to classify whether a comment is
16. [M] How does data sparsity affect your models?

    Hint: sparse data is different from missing data.

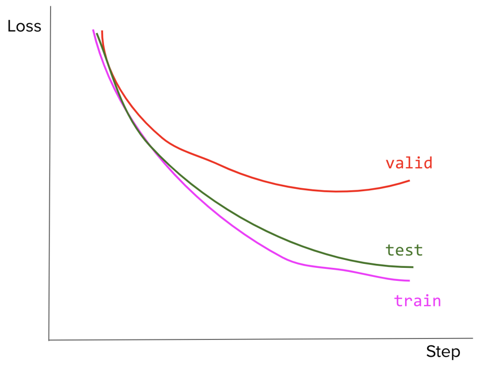
17. Feature leakage

    i. [E] What are some causes of feature leakage?
    ii. [E] Why does normalization help prevent feature leakage?
    iii. [M] How do you detect feature leakage?
18. [M] Suppose you want to build a model to classify whether a tweet spreads misinformation. You have 100K labeled tweets over the last 24 months. You decide to randomly shuffle on your data and pick 80% to be the train split, 10% to be the valid split, and 10% to be the test split. What might be the problem with this way of partitioning?
19. [M] You're building a neural network and you want to use both numerical and textual features. How would you process those different features?
20. [H] Your model has been performing fairly well using just a subset of features available in your data. Your boss decided that you should use all the features available instead. What might happen to the training error? What might happen to the test error?

    Hint: Think about the curse of dimensionality: as we use more dimensions to describe our data, the more sparse space becomes, and the further are data points from each other.

7.3 Objective functions, metrics, and evaluation

1. Convergence.
   i. [E] When we say an algorithm converges, what does convergence mean?
   ii. [E] How do we know when a model has converged?
2. [E] Draw the loss curves for overfitting and underfitting.
3. Bias-variance trade-off
   i. [E] What's the bias-variance trade-off?
   ii. [M] How's this tradeoff related to overfitting and underfitting?
   iii. [M] How do you know that your model is high variance, low bias? What would you do in this case?
   iv. [M] How do you know that your model is low variance, high bias? What would you do in this case?
4. Cross-validation.
   i. [E] Explain different methods for cross-validation.
   ii. [M] Why don't we see more cross-validation in deep learning?
5. Train, valid, test splits.
   i. [E] What's wrong with training and testing a model on the same data?
   ii. [E] Why do we need a validation set on top of a train set and a test set?
   iii. [M] Your model's loss curves on the train, valid, and test sets look like this. What might have been the cause of this? What would you do?



6. [E] Your team is building a system to aid doctors in predicting whether a patient has cancer or not from their X-ray scan. Your colleague announces that the problem is solved now that they've built a system that can predict with 99.99% accuracy. How would you respond to that claim?
7. F1 score.
   i. [E] What's the benefit of F1 over the accuracy?
   ii. [M] Can we still use F1 for a problem with more than two classes. How?
8. Given a binary classifier that outputs the following confusion matrix.

	Predicted True	Predicted False
Actual True	30	20
Actual False	5	40

   i. [E] Calculate the model's precision, recall, and F1.

    ii. [M] What can we do to improve the model's performance?

9. Consider a classification where 99% of data belongs to class A and 1% of data belongs to class B.
    i. [M] If your model predicts A 100% of the time, what would the F1 score be? **Hint**: The F1 score when A is mapped to 0 and B to 1 is different from the F1 score when A is mapped to 1 and B to 0.
    ii. [M] If we have a model that predicts A and B at a random (uniformly), what would the expected F1 be?
10. [M] For logistic regression, why is log loss recommended over MSE (mean squared error)?
11. [M] When should we use RMSE (Root Mean Squared Error) over MAE (Mean Absolute Error) and vice versa?
12. [M] Show that the negative log-likelihood and cross-entropy are the same for binary classification tasks.
13. [M] For classification tasks with more than two labels (e.g. MNIST with 10 labels), why is cross-entropy a better loss function than MSE?
14. [E] Consider a language with an alphabet of 27 characters. What would be the maximal entropy of this language?
15. [E] A lot of machine learning models aim to approximate probability distributions. Let's say P is the distribution of the data and Q is the distribution learned by our model. How do measure how close Q is to P?
16. MPE (Most Probable Explanation) vs. MAP (Maximum A Posteriori)
    i. [E] How do MPE and MAP differ?
    ii. [H] Give an example of when they would produce different results.
17. [E] Suppose you want to build a model to predict the price of a stock in the next 8 hours and that the predicted price should never be off more than 10% from the actual price. Which metric would you use?

    Hint: check out MAPE.

> In case you need a refresh on information entropy, here's an explanation without any math.
>
> Your parents are finally letting you adopt a pet! They spend the entire weekend taking you to various pet shelters to find a pet.
>
> The first shelter has only dogs. Your mom covers your eyes when your dad picks out an animal for you. You don't need to open your eyes to know that this animal is a dog. It isn't hard to guess.
>
> The second shelter has both dogs and cats. Again your mom covers your eyes and your dad picks out an email. This time, you have to think harder to guess which animal is that. You make a guess that it's a dog, and your dad says no. So you guess it's a cat and you're right. It takes you two guesses to know for sure what animal it is.
>
> The next shelter is the biggest one of them all. They have so many different kinds of animals: dogs, cats, hamsters, fish, parrots, cute little pigs, bunnies, ferrets, hedgehogs, chickens, even the exotic bearded dragons! There must be close to a hundred different types of pets. Now it's really hard for you to guess which one your dad brings you. It takes you a dozen guesses to guess the right animal.
>
> Entropy is a measure of the "spread out" in diversity. The more spread out the diversity, the header it is to guess an item correctly. The first shelter has very low entropy. The second shelter has a little bit higher entropy. The third shelter has the highest entropy.

Chapter 8. Machine learning algorithms

> 🌳 **Tip** 🌳
> To refresh your knowledge of different ML algorithms, it's a good idea to look at winning solutions for recent Kaggle competitions. For a list of machine learning algorithms and how they are used in winning solutions on Kaggle, check out Data Science Glossary on Kaggle.

8.1 Classical machine learning

8.1.1 Overview: Basic algorithms

8.1.1.1 k-nearest neighbor (k-NN)

k-NN is a non-parametric method used for classification and regression. Given an object, the algorithm's output is computed from its k closest training examples in the feature space.

- In k-NN classification, each object is classified into the class most common among its k nearest neighbors.
- In k-NN regression, each object's value is calculated as the average of the values of its k nearest neighbors.

Applications: anomaly detection, search, recommender system

8.1.1.2 k-means clustering

k-means clustering aims to partition observations into k clusters in which each observation belongs to the cluster with the nearest mean. k-means minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances.

The algorithm doesn't guarantee convergence to the global optimum. The result may depend on the initial clusters. As the algorithm is usually fast, it is common to run it multiple times with different starting conditions.

The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum.

The algorithm has a loose relationship to the k-nearest neighbor classifier. After obtaining clusters using k-means clustering, we can classify new data into those clusters by applying the 1-nearest neighbor classifier to the cluster centers.

Applications: Vector quantization for signal processing (where k-means clustering was originally developed), cluster analysis, feature learning, topic modeling.

8.1.1.3 EM (expectation-maximization) algorithm

EM algorithm is an iterative method to find maximum likelihood (MLE) or maximum a posteriori (MAP) estimates of parameters. It's useful when the model depends on unobserved latent variables and equations can't be solved directly.

The iteration alternates between performing:

- an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters
- a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

EM algorithm is guaranteed to return a local optimum of the sample likelihood function.

Example: Gaussian mixture models (GMM)

Applications: Data clustering, collaborative filtering.

8.1.1.4 Tree-based methods

Decision tree is a tree-based method that goes from observations about an object (represented in the branches) to conclusions about its target value (represented in the leaves). At its core, decision trees are nest if-else conditions.

In **classification trees**, the target value is discrete and each leaf represents a class. In **regression trees**, the target value is continuous and each leaf represents the mean of the target values of all objects that end up with that leaf.

Decision trees are easy to interpret and can be used to visualize decisions. However, they are overfit to the data they are trained on -- small changes to the training set can result in significantly different tree structures, which lead to significantly different outputs.

8.1.1.5 Bagging and boosting

Bagging and boosting are two popular ensembling methods commonly used with tree-based algorithms that can also be used for other algorithms.

8.1.1.5.1 Bagging

Bagging, shortened for **b**ootstrap **agg**regat**ing**, is designed to improve the stability and accuracy of ML algorithms. It reduces variance and helps to avoid overfitting.

Given a dataset, instead of training one classifier on the entire dataset, you sample **with** replacement to create different datasets, called bootstraps, and train a classification or regression model on each of these bootstraps. Sampling with replacement ensures each bootstrap is independent of its peers.

If the problem is classification, the final prediction is decided by the majority vote of all models. For example, if 10 classifiers vote SPAM and 6 models vote NOT SPAM, the final prediction is SPAM.

If the problem is regression, the final prediction is the average of all models' predictions.

Bagging generally improves unstable methods, such as neural networks, classification and regression trees, and subset selection in linear regression. However, it can mildly degrade the performance of stable methods such as k-nearest neighbors[2].

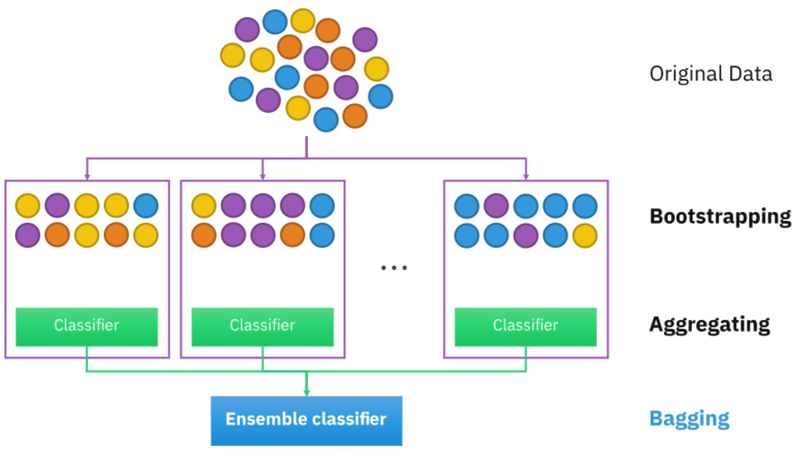


Illustration by Sirakorn

A **random forest** is an example of bagging. A random forest is a collection of decision trees constructed by both **bagging** and **feature randomness**, each tree can pick only from a random subset of features to use.

Due to its ensembling nature, random forests correct for decision trees' overfitting to their training set.

Applications: Random forests are among the most widely used machine learning algorithms in the real world. They are used in banking for fraud detection, medicine for disease prediction, stock market analysis, etc.

For more information on random forests, see Understanding Random Forest by Tony Yiu.

8.1.1.5.2 Boosting

Boosting is a family of iterative ensemble algorithms that convert weak learners to strong ones. Each learner in this ensemble is trained on the same set of samples but the samples are weighted differently among iterations. Thus, future weak learners focus more on the examples that previous weak learners misclassified.

1. You start by training the first weak classifier on the original dataset.
2. Samples are reweighted based on how well the first classifier classifies them, e.g. misclassified samples are given higher weight.
3. Train the second classifier on this reweighted dataset. Your ensemble now consists of the first and the second classifiers.
4. Samples are weighted based on how well the ensemble classifies them.
5. Train the third classifier on this reweighted dataset. Add the third classifier to the ensemble.
6. Repeat for as many iterations as needed.
7. Form the final strong classifier as a weighted combination of the existing classifiers -- classifiers with smaller training errors have higher weights.

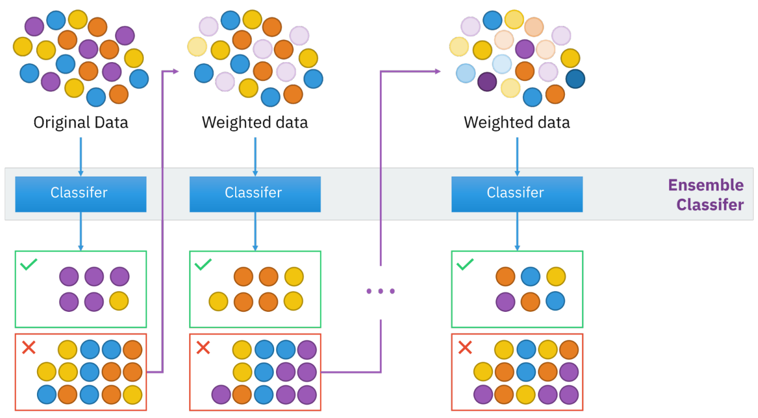


Illustration by Sirakorn

An example of a boosting algorithm is Gradient Boosting Machine which produces a prediction model typically from weak decision trees. It builds the model in a stage-wise fashion as other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

XGBoost, a variant of GBM, used to be the algorithm of choice for many winning teams of machine learning competitions. It's been used in a wide range of tasks from classification, ranking, to the discovery of the Higgs Boson[3]. However, many teams have been opting for LightGBM, a distributed gradient boosting framework that allows parallel learning which generally allows faster training on large datasets.

8.1.1.6 Kernel methods

In machine learning, kernel methods are a class of algorithms for pattern analysis, whose best-known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified feature map: in contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation.

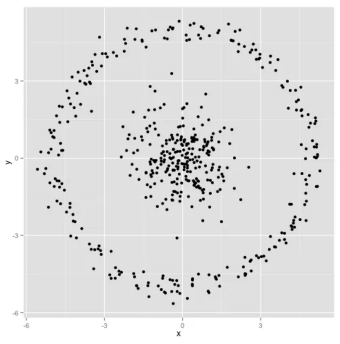
Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick".[1] Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

Algorithms capable of operating with kernels include the kernel perceptron, support vector machines (SVM), Gaussian processes, principal components analysis (PCA), canonical correlation analysis, ridge regression, spectral clustering, linear adaptive filters, and many others. Any linear model can be turned into a non-linear model by applying the kernel trick to the model: replacing its features (predictors) with a kernel function.

8.1.2 Questions

1. [E] What are the basic assumptions to be made for linear regression?
2. [E] What happens if we don't apply feature scaling to logistic regression?
3. [E] What are the algorithms you'd use when developing the prototype of a fraud detection model?
4. Feature selection.
   i. [E] Why do we use feature selection?
   ii. [M] What are some of the algorithms for feature selection? Pros and cons of each.
5. k-means clustering.

   i. [E] How would you choose the value of k?
   ii. [E] If the labels are known, how would you evaluate the performance of your k-means clustering algorithm?
   iii. [M] How would you do it if the labels aren't known?
   iv. [H] Given the following dataset, can you predict how K-means clustering works on it? Explain.



6. k-nearest neighbor classification.
   i. [E] How would you choose the value of k?
   ii. [E] What happens when you increase or decrease the value of k?
   iii. [M] How does the value of k impact the bias and variance?
7. k-means and GMM are both powerful clustering algorithms.

   i. [M] Compare the two.
   ii. [M] When would you choose one over another?

   Hint: Here's an example of how K-means and GMM algorithms perform on the artificial mouse dataset.

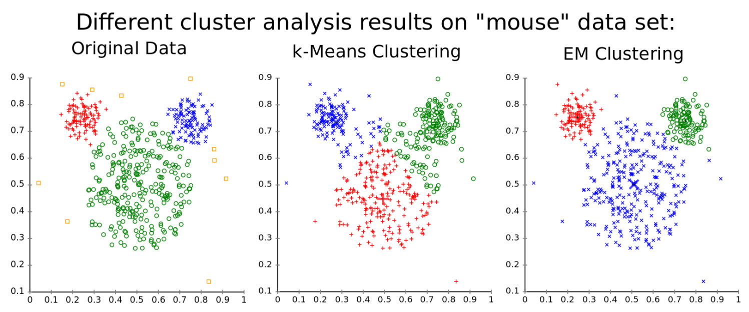
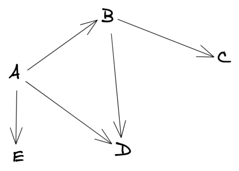


Image from Mohamad Ghassany's course on Machine Learning

8. Bagging and boosting are two popular ensembling methods. Random forest is a bagging example while XGBoost is a boosting example.
   i. [M] What are some of the fundamental differences between bagging and boosting algorithms?
   ii. [M] How are they used in deep learning?
9. Given this directed graph.

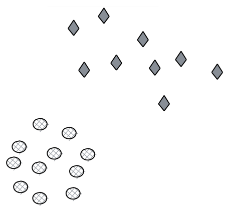


   i. [E] Construct its adjacency matrix.
   ii. [E] How would this matrix change if the graph is now undirected?
   iii. [M] What can you say about the adjacency matrices of two isomorphic graphs?
10. Imagine we build a user-item collaborative filtering system to recommend to each user items similar to the items they've bought before.
   i. [M] You can build either a user-item matrix or an item-item matrix. What are the pros and cons of each approach?
   ii. [E] How would you handle a new user who hasn't made any purchases in the past?
11. [E] Is feature scaling necessary for kernel methods?
12. Naive Bayes classifier.
   i. [E] How is Naive Bayes classifier naive?
   ii. [M] Let's try to construct a Naive Bayes classifier to classify whether a tweet has a positive or negative sentiment. We have four training samples:

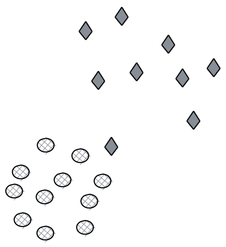
Tweet	**Label**
This makes me so upset	Negative
This puppy makes me happy	Positive
Look at this happy hamster	Positive
No hamsters allowed in my house	Negative

   According to your classifier, what's sentiment of the sentence `The hamster is upset with the puppy`?

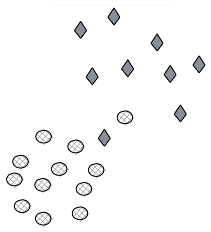
13. Two popular algorithms for winning Kaggle solutions are Light GBM and XGBoost. They are both gradient boosting algorithms.
   i. [E] What is gradient boosting?
   ii. [M] What problems is gradient boosting good for?
14. SVM.
   i. [E] What's linear separation? Why is it desirable when we use SVM?
   ii. [M] How well would vanilla SVM work on this dataset?



iii. [M] How well would vanilla SVM work on this dataset?



iv. [M] How well would vanilla SVM work on this dataset?



8.2 Deep learning architectures and applications

There are three main subfields in machine learning: speech and natural language processing (NLP), computer vision, and reinforcement learning. NLP has been successfully applied in business intelligence, voice assistant, machine translation, autocompletion and autocorrection, automated customer services, etc.

Computer vision is the core technology in self-driving cars, security (surveillance cameras, facial recognition), photo/video generation (which are terrifyingly good), and other entertainment services such as photo editing, photo filters, face swap.

Reinforcement learning is harder to deploy as the real-world environment is so much more complex than simulation, but we’ve seen the use of RL in ads bidding optimization, unmanned aerial vehicles (such as drones), and various robotic applications such as warehouse and production robots.

A company or a team might focus on a subfield. For example, the Siri team at Apple might focus on speech and natural language understanding, and the Autopilot team at Tesla might be more interested in computer vision. However, techniques in one subfield can be used for another, and there are tasks that have components from different subfields. There’s undoubtedly value in being a world-class expert in your niche subfield, but to get there, you might need to have knowledge of other subfields.

8.2.1 Natural language processing

1. RNNs
    i. [E] What's the motivation for RNN?
    ii. [E] What's the motivation for LSTM?
    iii. [M] How would you do dropouts in an RNN?
2. [E] What's density estimation? Why do we say a language model is a density estimator?
3. [M] Language models are often referred to as unsupervised learning, but some say its mechanism isn't that different from supervised learning. What are your thoughts?
4. Word embeddings.
    i. [M] Why do we need word embeddings?
    ii. [M] What's the difference between count-based and prediction-based word embeddings?
    iii. [H] Most word embedding algorithms are based on the assumption that words that appear in similar contexts have similar meanings. What are some of the problems with context-based word embeddings?
5. Given 5 documents:

```
D1: The duck loves to eat the worm
D2: The worm doesn’t like the early bird
D3: The bird loves to get up early to get the worm
D4: The bird gets the worm from the early duck
D5: The duck and the birds are so different from each other but one thing
```

    i. [M] Given a query Q: “The early bird gets the worm”, find the two top-ranked documents according to the TF/IDF rank using the cosine similarity measure and the term set {bird, duck, worm, early, get, love}. Are the top-ranked documents relevant to the query?
    ii. [M] Assume that document D5 goes on to tell more about the duck and the bird and mentions “bird” three times, instead of just once. What happens to the rank of D5? Is this change in the ranking of D5 a desirable property of TF/IDF? Why?
6. [E] Your client wants you to train a language model on their dataset but their dataset is very small with only about 10,000 tokens. Would you use an n-gram or a neural language model?
7. [E] For n-gram language models, does increasing the context length (n) improve the model's performance? Why or why not?
8. [M] What problems might we encounter when using softmax as the last layer for word-level language models? How do we fix it?
9. [E] What's the Levenshtein distance of the two words “doctor” and “bottle”?
10. [M] BLEU is a popular metric for machine translation. What are the pros and cons of BLEU?
11. [H] On the same test set, LM model A has a character-level entropy of 2 while LM model A has a word-level entropy of 6. Which model would you choose to deploy?
12. [M] Imagine you have to train a NER model on the text corpus A. Would you make A case-sensitive or case-insensitive?

13. [M] Why does removing stop words sometimes hurt a sentiment analysis model?
14. [M] Many models use relative position embedding instead of absolute position embedding. Why is that?
15. [H] Some NLP models use the same weights for both the embedding layer and the layer just before softmax. What’s the purpose of this?

8.2.2 Computer vision

1. [M[For neural networks that work with images like VGG-19, InceptionNet, you often see a visualization of what type of features each filter captures. How are these visualizations created?

   Hint: check out this Distill post on Feature Visualization.

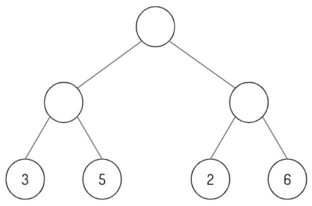
2. Filter size.
   i. [M] How are your model's accuracy and computational efficiency affected when you decrease or increase its filter size?
   ii. [E] How do you choose the ideal filter size?
3. [M] Convolutional layers are also known as "locally connected." Explain what it means.
4. [M] When we use CNNs for text data, what would the number of channels be for the first conv layer?
5. [E] What is the role of zero padding?
6. [E] Why do we need upsampling? How to do it?
7. [M] What does a 1x1 convolutional layer do?
8. Pooling.
   i. [E] What happens when you use max-pooling instead of average pooling?
   ii. [E] When should we use one instead of the other?
   iii. [E] What happens when pooling is removed completely?
   iv. [M] What happens if we replace a 2 x 2 max pool layer with a conv layer of stride 2?
9. [M] When we replace a normal convolutional layer with a depthwise separable convolutional layer, the number of parameters can go down. How does this happen? Give an example to illustrate this.
10. [M] Can you use a base model trained on ImageNet (image size 256 x 256) for an object classification task on images of size 320 x 360? How?
11. [H] How can a fully-connected layer be converted to a convolutional layer?
12. [H] Pros and cons of FFT-based convolution and Winograd-based convolution.

    Hint: Read Fast Algorithms for Convolutional Neural Networks (Andrew Lavin and Scott Gray, 2015)

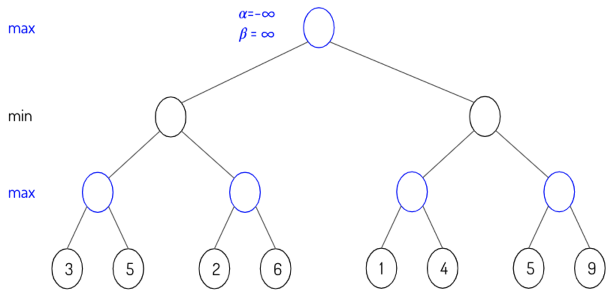
8.2.3 Reinforcement learning

> 🌳 **Tip** 🌳
> To refresh your knowledge on deep RL, checkout Spinning Up in Deep RL (OpenAI)

1. [E] Explain the explore vs exploit tradeoff with examples.
2. [E] How would a finite or infinite horizon affect our algorithms?
3. [E] Why do we need the discount term for objective functions?
4. [E] Fill in the empty circles using the minimax algorithm.



5. [M] Fill in the alpha and beta values as you traverse the minimax tree from left to right.



6. [E] Given a policy, derive the reward function.
7. [M] Pros and cons of on-policy vs. off-policy.
8. [M] What’s the difference between model-based and model-free? Which one is more data-efficient?

8.2.4 Other

1. [M] An autoencoder is a neural network that learns to copy its input to its output. When would this be useful?
2. Self-attention.
    i. [E] What's the motivation for self-attention?
    ii. [E] Why would you choose a self-attention architecture over RNNs or CNNs?
    iii. [M] Why would you need multi-headed attention instead of just one head for attention?
    iv. [M] How would changing the number of heads in multi-headed attention affect the model's performance?
3. Transfer learning
    i. [E] You want to build a classifier to predict sentiment in tweets but you have very little labeled data (say 1000). What do you do?
    ii. [M] What's gradual unfreezing? How might it help with transfer learning?
4. Bayesian methods.
    i. [M] How do Bayesian methods differ from the mainstream deep learning approach?
    ii. [M] How are the pros and cons of Bayesian neural networks compared to the mainstream neural networks?
    iii. [M] Why do we say that Bayesian neural networks are natural ensembles?
5. GANs.
    i. [E] What do GANs converge to?
    ii. [M] Why are GANs so hard to train?

8.3 Training neural networks

> 🌳 **Tip** 🌳
> For more tips on training neural networks, check out:
>
> - A Recipe for Training Neural Networks (Karpathy 2019)
> - NLP's Clever Hans Moment has Arrived (Heinzerling 2019): an excellent writeup on trying to understand what exactly your neural network learns, and techniques to ensure that your model works correctly with textual data.
> - An overview of gradient descent optimization algorithms (Ruder 2016)

1. [E] When building a neural network, should you overfit or underfit it first?
2. [E] Write the vanilla gradient update.
3. Neural network in simple Numpy.
   i. [E] Write in plain NumPy the forward and backward pass for a two-layer feed-forward neural network with a ReLU layer in between.
   ii. [M] Implement vanilla dropout for the forward and backward pass in NumPy.
4. Activation functions.
   i. [E] Draw the graphs for sigmoid, tanh, ReLU, and leaky ReLU.
   ii. [E] Pros and cons of each activation function.
   iii. [E] Is ReLU differentiable? What to do when it's not differentiable?
   iv. [M] Derive derivatives for sigmoid function $\sigma(x)$ when x is a vector.
5. [E] What's the motivation for skip connection in neural works?
6. Vanishing and exploding gradients.
   i. [E] How do we know that gradients are exploding? How do we prevent it?
   ii. [E] Why are RNNs especially susceptible to vanishing and exploding gradients?
7. [M] Weight normalization separates a weight vector's norm from its gradient. How would it help with training?
8. [M] When training a large neural network, say a language model with a billion parameters, you evaluate your model on a validation set at the end of every epoch. You realize that your validation loss is often lower than your train loss. What might be happening?
9. [E] What criteria would you use for early stopping?
10. [E] Gradient descent vs SGD vs mini-batch SGD.
11. [H] It's a common practice to train deep learning models using epochs: we sample batches from data **without** replacement. Why would we use epochs instead of just sampling data **with** replacement?
12. [M] Your model' weights fluctuate a lot during training. How does that affect your model's performance? What to do about it?
13. Learning rate.
    i. [E] Draw a graph number of training epochs vs training error for when the learning rate is:
       i. too high
       ii. too low

        iii. acceptable.
    ii. [E] What’s learning rate warmup? Why do we need it?

14. [E] Compare batch norm and layer norm.
15. [M] Why is squared L2 norm sometimes preferred to L2 norm for regularizing neural networks?
16. [E] Some models use weight decay: after each gradient update, the weights are multiplied by a factor slightly less than 1. What is this useful for?
17. It’s a common practice for the learning rate to be reduced throughout the training.
    i. [E] What’s the motivation?
    ii. [M] What might be the exceptions?
18. Batch size.
    i. [E] What happens to your model training when you decrease the batch size to 1?
    ii. [E] What happens when you use the entire training data in a batch?
    iii. [M] How should we adjust the learning rate as we increase or decrease the batch size?
19. [M] Why is Adagrad sometimes favored in problems with sparse gradients?
20. Adam vs. SGD.
    i. [M] What can you say about the ability to converge and generalize of Adam vs. SGD?
    ii. [M] What else can you say about the difference between these two optimizers?
21. [M] With model parallelism, you might update your model weights using the gradients from each machine asynchronously or synchronously. What are the pros and cons of asynchronous SGD vs. synchronous SGD?
22. [M] Why shouldn’t we have two consecutive linear layers in a neural network?
23. [M] Can a neural network with only RELU (non-linearity) act as a linear classifier?
24. [M] Design the smallest neural network that can function as an XOR gate.
25. [E] Why don’t we just initialize all weights in a neural network to zero?
26. Stochasticity.
    i. [M] What are some sources of randomness in a neural network?
    ii. [M] Sometimes stochasticity is desirable when training neural networks. Why is that?
27. Dead neuron.
    i. [E] What’s a dead neuron?
    ii. [E] How do we detect them in our neural network?
    iii. [M] How to prevent them?
28. Pruning.
    i. [M] Pruning is a popular technique where certain weights of a neural network are set to 0. Why is it desirable?
    ii. [M] How do you choose what to prune from a neural network?
29. [H] Under what conditions would it be possible to recover training data from the weight checkpoints?
30. [H] Why do we try to reduce the size of a big trained model through techniques such as knowledge distillation instead of just training a small model from the beginning?

Appendix

A. For interviewers

Candidates are frustrated with the hiring process -- it's beyond their control. But hiring managers also seem frustrated with their own processes because they can't or don't know how to change them. Every hiring manager I've talked to expressed interest in reading this book to learn from other companies. I don't hope to change a system with one book. I just hope it provides a scaffold on which we can start the conversation and establish good interviewing practices.

Hiring for ML roles is competitive. Strong candidates always have options, but strong candidates in ML have a lot more options. A hiring process with the candidates' experience in mind will give candidates a better impression of your company, making them more likely to accept your offer. A better process can help you attract more suitable candidates, make better hires, which saves you time down the line. Even if it's a no-hire, you still want to win their allegiance. You might want to reinterview them in the future, they might recommend you to their network, and they can become advocates for your products.

ML interviews are still a black box. For a field so obsessed with qualitative methodologies, it's surprising that there has been so little qualitative research done on the hiring process for ML roles. Few companies can state with any certainty the correlation between their interview techniques and the quality of hires. Some of the challenges include:

1. People who interview aren't the same people who evaluate performance on the job.
2. No established metrics for interview outcomes and performance outcomes, or the correlation between them[57].
3. No control group. Companies don't hire people who fail their interviews to compare their job performances to the performances of those who pass.

A low hanging fruit is testing your pipeline on existing people within your organization. See how they like the process and whether their performances on the interviews correlate with their performances on the jobs. Another is actively soliciting feedback from candidates. Instead of sending out survey links that nobody clicks on, after each onsite, have someone not involved in the hiring process spend 15 minutes talking to the candidate. Make sure the candidate knows that their feedback won't affect the hiring decision in any way.

If you don't have a training program for interviewers already, please set up one. Interviewers are both gatekeepers and faces of your organization. Letting new hires shadow for a few interviews isn't enough. New hires have no idea what they are doing, and companies have no way of standardizing feedback across teams, interviewers, and candidates.

The zen of interviews

1. The goal of the hiring process is to hire people, not to eliminate them.
2. Interview questions should be tailored for each candidate. Companies who ask different candidates for the same role the same questions are process-driven, not people-driven.
3. Standardized interview questions lead to standardized answers which, in turn, leads to standardized people.
4. If a piece of knowledge is easy to acquire, it's not worth testing for.
5. Ask questions with multiple hurdles. After giving the candidate a hint, the question becomes slightly easier, but there's still room for the candidate to show off their skills.
6. Ask hard questions. Extraordinary candidates can write up a simple solution within an hour, and there's no upper bound on how much a candidate can improve the solution. Asking hard questions shows that you think about and work on hard problems, which attracts people who want to solve hard problems.
7. A good interview is a question that turns into a conversation.
8. For long-term hires, interviews should focus on critical thinking and problem-solving skills instead of techniques. Techniques change over time, but critical thinking and problem-solving skills will always be relevant.
9. Both interviewers' and candidates' time should be respected.
10. Interviews should be transparent. For each interview, not only the final evaluation but also the log of the questions and answers should be submitted.

B. Building your network

If you didn't think that having a network mattered much before, I hope that this book has changed your mind. Having the right network can open you up to new opportunities and bring you closer to your dream jobs.

The best contacts are people you've studied or worked with. The main value of attending a selective school or joining a selective company is for the network. Your friends from MIT or Stanford might eventually join FAAAM, and your friends from FAAAM might eventually start their own companies.

If a network isn't a built-in feature of your education, you'll have to make an effort. I get it, you hate networking. The reward function for talking to strangers is stochastic. Once in a while, you meet someone you hit it off with. But most of the time, it's boring. I haven't been to a networking event since my first year in college. Why spend an evening standing in a corner at a networking event while you can sit on your couch, eating pizza and browsing memes?

But networking doesn't have to be about meeting random strangers and forcing a connection. It can be about finding people who share your interests and learning new things from them. Seek out people who've done things that you enjoy, and try to have a constructive discussion about their work.

When I just started out in ML, I tried to reimplement papers. When I came across something I didn't understand or something I thought could have been done differently, I emailed the authors. Most people are receptive to discussions about their work -- everyone wants their work to be more popular. I got to know a lot of people in the process.

Sometimes, back-and-forth emailing isn't effective and I ask people to meet for coffee or hop on a call. They sometimes don't respond or say no. I receive messages like that too. As long as the senders are respectful, their messages make me happy (somebody acknowledges my work!) and I try to make time for them. If I don't respond or say no, it's because I'm busy or forgetful.

Publish your work so that people interested in it can reach out to you. I enjoy doing side projects to satisfy my own curiosity, but they also help put my name out there. I often receive emails along the lines of: "Your analysis of X is really cool. Do you think it'd work on Y?" or "Can I use it in my project Z?" Sometimes, we find ways to work together, and sometimes, they become my friends. I've more than once received emails from people I admire but have no courage to reach out to.

Social networks help a lot if you're mindful of how you use them. ML people are surprisingly active on Twitter. I follow people whose opinions I value and contribute to their public conversations when I have things to contribute. I post what I care deeply about so that I attract only people who share my interests. If someone posts something I find helpful or thoughtfully responds to my post, I send them a short message to thank them for their effort.

Top-tier conferences, such as NeurIPS, ICLR, ICML, CVPR, ACL, SIGGRAPH, are also great for meeting people. Many might disagree with me on this, but I believe that since most top-tier conferences publish their talks and papers online, the main value of going to a conference is to meet other attendees. You should check out the conference schedule and browse the list of accepted papers in advance to see who is going and reach out to those you want to meet. While at the conference, try to talk to as many people as possible about their work (and yours, too).

You'll make a lot more out of a conference if you present your work, either at the main conference or at one of the workshops. There are, of course, exceptions but in general, it's easier getting a paper accepted to a workshop than to the main conference. Major conferences and workshops often have travel grants for accepted authors who need financial assistance.

Building a network is a lifelong process, not something that you can "hack" over a weekend. Be open to meeting new people, discovering what they know that you don't. Don't approach someone because you want to be friends with them -- you can't force friendships and it's borderline creepy. Approach someone because there's something you want to talk with them about. Friendships will naturally occur if there's mutual interest.

When you're ready to look for opportunities, don't be afraid of letting relevant people in your network know. Most people are happy to put in referrals for strong candidates. They want to work with smart people and they also get a referral bonus if their candidates are hired.

You can reach out to people outside your network for referrals too. I've more than once referred candidates who cold-emailed me asking for referrals. Engineers are usually better than recruiters at judging someone's technical skills. If you lack signals that recruiters look for, you might have better luck impressing other engineers.

The best job search strategy is to build an impossible-to-ignore portfolio. Write in-depth technical blog posts or papers, contribute to open-source projects, do hackathons, compete in coding challenges, and most importantly, publish your work so that the world knows what an amazing engineer you are.