BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Causes of ML System failure

Pravin Y Pawar

Adapted from “Designing Machine Learning Systems”
By Chip Huyen

ML system failure

- A failure happens when one or more expectations of the system is violated

- In traditional software, mostly care about a system’s operational expectations:
  - whether the system executes its logic within the expected operational metrics,
  - e.g., latency and throughput

- For an ML system, care about both its operational metrics and its ML performance metrics
- For example, consider an English-French machine translation system
  - Operational expectation might be that, given an English sentence, the system returns a French translation within a one-second latency
  - ML performance expectation is that the returned translation is an accurate translation of the original English sentence 99% of the time

ML system failure types

- Operational expectation violations are easier to detect,
  - as usually accompanied by an operational breakage
  - such as a timeout, a 404 error on a webpage, an out-of-memory error, or a segmentation fault

- However, ML performance expectation violations are harder to detect
  - as doing so requires measuring and monitoring the performance of ML models in production

- In the English-French machine translation system, detecting whether the returned translations are correct 99% of the time is difficult
  - if don't know what the correct translations are supposed to be

- To effectively detect and fix ML system failures in production,
  - it's useful to understand why a model, after proving to work well during development, would fail in production

- Two types of failures: software system failures and ML-specific failures

Software System Failures

failures that would have happened to non-ML systems

- Dependency failure
  - A software package or a codebase that system depends on breaks, which leads system to break
  - common when the dependency is maintained by a third party

- Deployment failure
  - failures caused by deployment errors,
  - such as when accidentally deploy the binaries of an older version of model instead of the current version,
  - or when systems don’t have the right permissions to read or write certain files

- Hardware failures
  - When the hardware that is used to deploy model, such as CPUs or GPUs, doesn’t behave the way it should
  - For example, the CPUs used might overheat and break down

- Downtime or crashing
  - If a component of system runs from a server somewhere, such as AWS or a hosted service, and that server is down, system will also be down

Software System Failures(2)

- Addressing software system failures requires not ML skills, but traditional software engineering skills!

- Because of the importance of traditional software engineering skills in deploying ML systems,
  - ML engineering is mostly engineering, not ML!

- The reasons for the prevalence of software system failures:
  - ML adoption in the industry is still nascent,
  - tooling around ML production is limited
  - and best practices are not yet well developed or standardized

- However, as tooling's and best practices for ML production mature,
  - there are reasons to believe that the proportion of software system failures will decrease
  - and the proportion of ML-specific failures will increase

ML-Specific Failures

failures specific to ML systems

- Examples include
  - data collection and processing problems,
  - poor hyper parameters,
  - changes in the training pipeline not correctly replicated in the inference pipeline and vice versa,
  - data distribution shifts that cause a model's performance to deteriorate over time,
  - edge cases,
  - and degenerate feedback loops

- Even though they account for a small portion of failures, they can be more dangerous than non-ML failures
  - as they're hard to detect and fix, and they can prevent ML systems from being used altogether

Production data differing from training data

- ML model learns from the training data, means that the model learns the underlying distribution of the training data
  - with the goal of leveraging this learned distribution to generate accurate predictions for unseen data
  - when the model is able to generate accurate predictions for unseen data - model “generalizes to unseen data”

- The assumption
  - unseen data comes from a stationary distribution that is the same as the training data distribution

- This assumption is incorrect in most cases for two reasons!
- First, the underlying distribution of the real-world data is unlikely to be the same as the underlying distribution of the training data
  - Real-world data is multifaceted and, in many cases, virtually infinite,
  - whereas training data is finite and constrained by the time, compute, and human resources available during the dataset creation and processing
  - divergence leads to a common failure mode known as the train-serving skew: a model that does great in development but performs poorly when deployed
- Second, the real world isn’t stationary. Things change. Data distributions shift.

Edge cases

- Edge cases are the data samples so extreme that they cause the model to make catastrophic mistakes

- An ML model that performs well on most cases but fails on a small number of cases
  - might not be usable if these failures cause catastrophic consequences
  - major self-driving car companies are focusing on making their systems work on edge cases

- Also true for any safety-critical application such as medical diagnosis, traffic control, e-discovery, etc.
- Can also be true for non-safety-critical applications
  - Imagine a customer service chatbot that gives reasonable responses to most of the requests,
  - but sometimes, it spits out outrageously racist or sexist content
  - will be a brand risk for any company that wants to use it, thus rendering it unusable

Degenerate feedback loops

- Feedback loop as the time it takes from when a prediction is shown until the time feedback on the prediction is provided
  - feedback can be used to
    - extract natural labels to evaluate the model's performance
    - train the next iteration of the model

- A degenerate feedback loop can happen when the predictions themselves influence the feedback,
  - which, in turn, influences the next iteration of the model
- A degenerate feedback loop is created when a system's outputs are used to generate the system's future inputs,
  - which, in turn, influence the system's future outputs

- Degenerate feedback loops are especially common in tasks with natural labels from users,
  - such as recommender systems and ads click-through-rate prediction

- Imagine building a system to recommend to users songs that they might like
  - songs that are ranked high by the system are shown first to users
  - because they are shown first, users click on them more,
  - which makes the system more confident that these recommendations are good

Thank You!

In our next session: