BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Continual Learning

Pravin Y Pawar

Adapted from “Designing Machine Learning Systems”
By Chip Huyen

Continual Learning

- How do adapt models to data distribution shifts?
  - The answer is by continually updating our ML models
- Continual learning is largely an infrastructural problem
  - Setting up infrastructure allows to update models as frequently as required
- The goal of continual learning is to safely and efficiently automate the update
  - allows to design an ML system that is maintainable and adaptable to changing environments

What is continual learning?

- Many people think of the training paradigm where a model updates itself with every incoming sample in production
  - people imagine updating models very frequently, such as every 5 or 10 minutes

- Very few companies actually do that!
  - makes model susceptible to catastrophic forgetting - the tendency of a neural network to completely and abruptly forget

    previously learned information upon learning new information
  - make training more expensive - most hardware backends today were designed for batch processing, so processing only

    one sample at a time causes a huge waste of compute power and is unable to exploit data parallelism

- Most companies don't need to update their models that frequently because of two reasons
  - First, they don't have enough traffic (i.e., enough new data) for that retraining schedule to make sense
  - Second, their models don't decay that fast
    - If changing retraining schedule from a week to a day gives no return and causes more overhead, there's no need to do it.

A simplification of how continual learning work in production

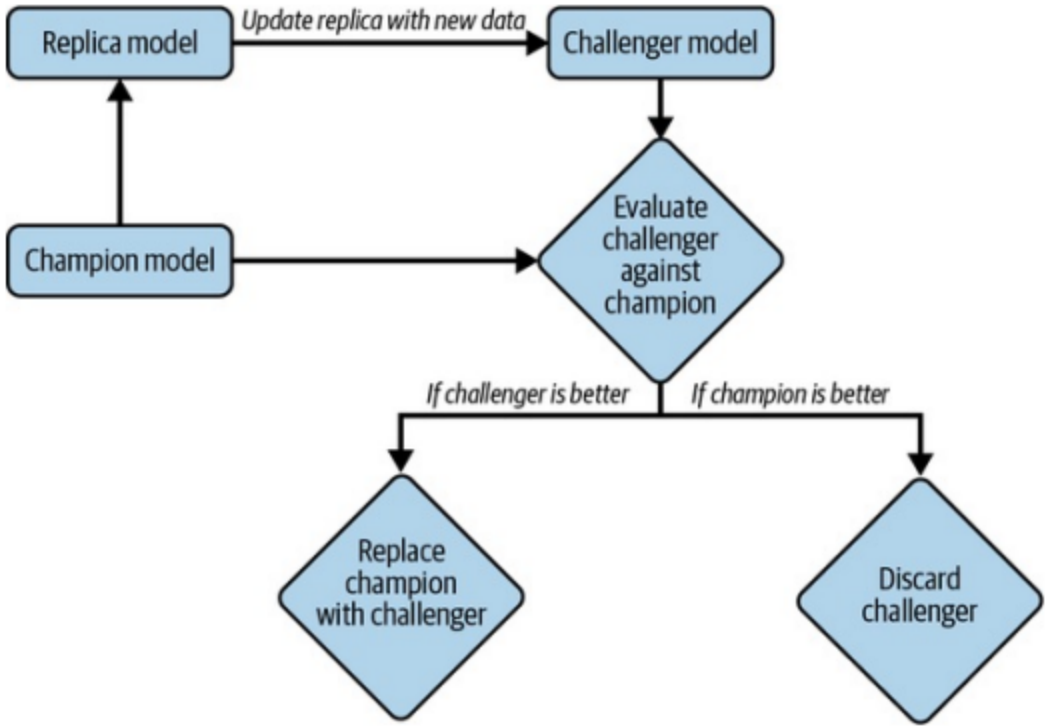
Champion-Challenger

- The updated model shouldn't be deployed until it's been evaluated
  - means that shouldn't make changes to the existing model directly

- Instead,
  - should create a replica of the existing model
  - update this replica on new data
  - only replace the existing model with the updated replica if
  - the updated replica proves to be better

- The existing model - champion model
- The updated replica - the challenger

- An oversimplification of the process for the sake of understanding!
  - In reality, a company might have multiple challengers at the same time



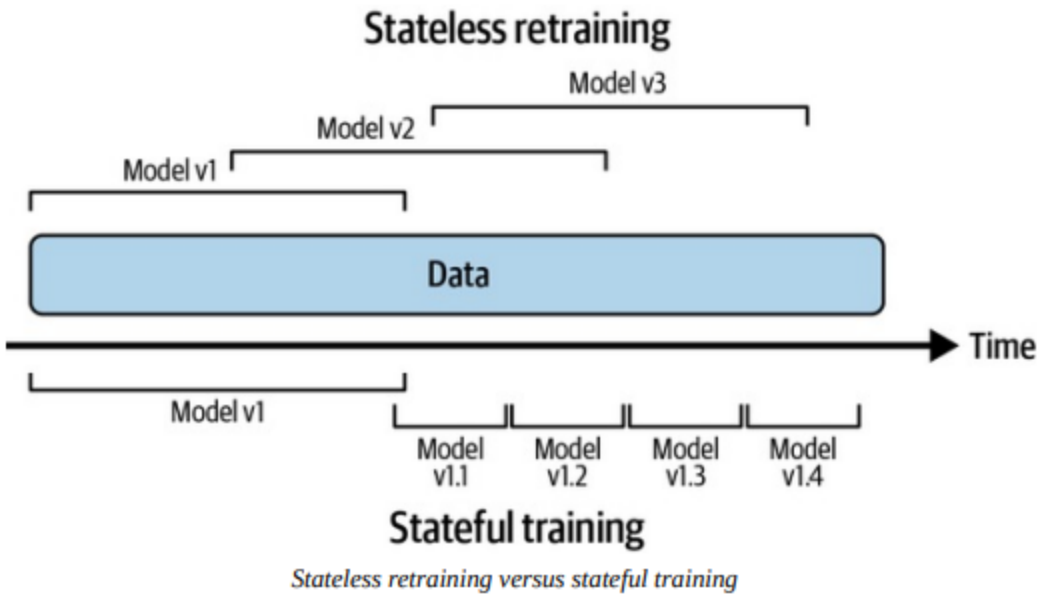
Stateless Retraining Versus Stateful Training

- Continual learning isn't about the retraining frequency,
  - but the manner in which the model is retrained

- Most companies do stateless retraining — the model is trained from scratch each time
- Some follows stateful training — the model continues training on new data
  - also known as fine-tuning or incremental learning

- Stateful training allows to update model with less data
- Training a model from scratch tends to require a lot more data

- One beautiful property that is often overlooked is that with stateful training,
  - it might be possible to avoid storing data altogether



Stateless retraining versus stateful training

Stateless Retraining Versus Stateful Training(2)

- The companies that have most successfully used stateful training
  - also occasionally train their model from scratch on a large amount of data to calibrate it

- Once infrastructure is set up to allow both stateless retraining and stateful training,
  - the training frequency is just a knob to twist
  - can update models once an hour, once a day, or whenever a distribution shift is detected

- Continual learning is about setting up infrastructure in a way that
  - allows a data scientist or ML engineer,
  - to update models whenever it is needed,
  - whether from scratch or fine-tuning,
  - to deploy this update quickly

Model iteration vs Data iteration

- Stateful training sounds cool, but how does this work if needs to add a new feature or another layer to model?

- Must differentiate two types of model updates:
  - Model iteration - A new feature is added to an existing model or the model architecture is changed
  - Data iteration - Model architecture and features remain the same, but refresh this model with new data.

- As of today, stateful training is mostly applied for data iteration!

- Changing model architecture or adding a new feature still requires training the resulting model from scratch
  - research shows that it might be possible to bypass training from scratch for model iteration
  - by using techniques such as knowledge transfer (Google, 2015) and model surgery (OpenAI, 2019)

Why Continual Learning?

Why would you need the ability to update your models as fast as you want?

- The first use case of continual learning is to combat data distribution shifts,
  - especially when the shifts happen suddenly
- Another use case of continual learning is to adapt to rare events
- Can help overcome is the continuous cold start problem

- "Why continual learning?" should be rephrased as "why not continual learning?"
  - Continual learning is a superset of batch learning - allows to do everything traditional batch learning can do
  - If continual learning takes the same effort to set up and costs the same to do as batch learning,
    - there's no reason not to do continual learning!

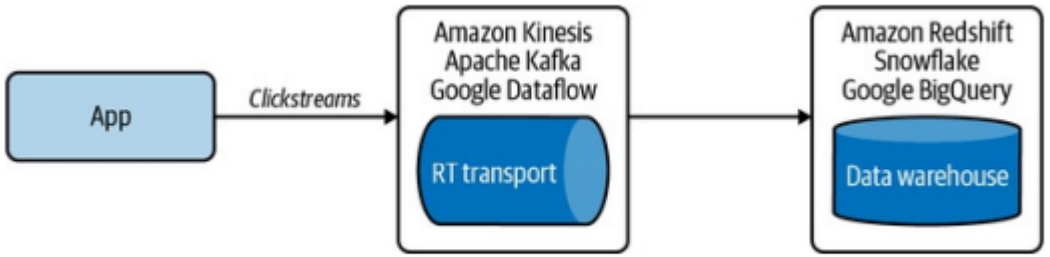
- MLOps tooling for continual learning is maturing, which means, one day not too far in the future,
  - it might be as easy to set up continual learning as batch learning

Continual Learning Challenges

Fresh data access challenge - Data Sources

- If want to update model every hour, need new data every hour!

- Data Sources
  - Currently, many companies pull new training data from their data warehouses
    - The speed at which data can be pulled from data warehouses depends on the speed at which this data is deposited into data warehouses
    - The speed can be slow, especially if data comes from multiple sources
  - An alternative is to allow pull data before it's deposited into data warehouses,
    - e.g., directly from real-time transports such as Kafka and Kinesis that transport data from applications to data warehouses



Pulling data directly from real-time transports, before it's deposited into data warehouses, can allow you to access fresher data

Continual Learning Challenges(2)

Fresh data access challenge - Labelling

- If model needs labeled data to update, data will need to be labeled as well!
  - In many applications, the speed at which a model can be updated is bottlenecked by the speed at which data is labeled

- The best candidates for continual learning are tasks where can get natural labels with short feedback loops
  - dynamic pricing, estimating time of arrival, stock price prediction,
  - ads click-through prediction, and recommender systems

- If model's speed iteration is bottlenecked by labeling speed,
  - possible to speed up the labeling process by leveraging programmatic labeling tools like Snorkel to generate fast labels
  - possible to leverage crowdsourced labels to quickly annotate fresh data

- Given that tooling around streaming is still nascent,
  - architecting an efficient streaming-first infrastructure for accessing fresh data and extracting fast labels from real-time transports can be engineering-intensive and costly

Continual Learning Challenges(3)

Evaluation challenge

- ML systems make catastrophic failures in production,
  - from millions of minorities being unjustly denied loans,
  - to drivers who trust autopilot too much being involved in fatal crashes

- The biggest challenge is in making sure that this update is good enough to be deployed!

- The risks for catastrophic failures amplify with continual learning
  - More frequently models are updated, the more opportunities there are for updates to fail
  - Makes models more susceptible to coordinated manipulation and adversarial attack

- To avoid similar or worse incidents, it's crucial to thoroughly test each of model updates to ensure its performance and safety before deploying the updates to a wider audience.

- When designing the evaluation pipeline for continual learning, keep in mind that evaluation takes time,
  - which can be another bottleneck for model update frequency.

Continual Learning Challenges(4)

Algorithm challenge - “softer” challenge

- Challenge as it only affects certain algorithms and certain training frequencies.
  - only affects matrix-based and tree-based models that want to be updated very fast (e.g., hourly)

- Consider two different models:
  - a neural network and a matrix-based model, such as a collaborative filtering model
  - The collaborative filtering model uses a user-item matrix and a dimension reduction technique

- A neural network model
  - Can update model with a data batch of any size
  - Can even perform the update step with just one data sample

- Collaborative filtering model
  - For updating model, first need to use the entire dataset to build the user-item matrix before performing dimensionality reduction on it
  - if matrix is large, the dimensionality reduction step would be too slow and expensive to perform frequently
  - less suitable for learning with a partial dataset than the preceding neural network model

Four Stages of Continual Learning

- How to overcome these challenges and make continual learning happen?

- The move toward continual learning happens in four stages
  - Stage 1: Manual, stateless retraining
  - Stage 2: Automated retraining
  - Stage 3: Automated, stateful training
  - Stage 4: Continual learning

Four Stages of Continual Learning(2)

Stage 1: Manual, stateless retraining

- In the beginning, the ML team often focuses on developing ML models to solve as many business problems as possible
  - Because team is focusing on developing new models, updating existing models takes a backseat
  - No fixed frequency to update the models

- Update an existing model only when the following two conditions are met:
  - the model's performance has degraded to the point that it's doing more harm than good
  - team has time to update it

- The process of updating a model is manual and ad hoc
  - Someone, usually a data engineer, has to query the data warehouse for new data.
  - Someone else cleans this new data, extracts features from it, retrains that model from scratch on both the old and new data, and then exports the updated model into a binary format.
  - Someone else takes that binary format and deploys the updated model.

- A vast majority of companies outside the tech industry—
  - e.g., any company that adopted ML less than three years ago and doesn't have an ML platform team—are in this stage

Four Stages of Continual Learning(3)

Stage 2: Automated retraining

- After a few years, team has managed to deploy models to solve most of the obvious problems
  - Priority is no longer to develop new models, but to maintain and improve existing models
  - The ad hoc, manual process of updating models mentioned from the previous stage has grown into a pain point too big to be ignored

- Team decides to write a script to automatically execute all the retraining steps
  - run periodically using a batch process such as Spark

- Status
  - Most companies with somewhat mature ML infrastructure are in this stage
  - Some sophisticated companies run experiments to determine the optimal retraining frequency
  - For most companies in this stage, the retraining frequency is set based on gut feeling
    - e.g., “once a day seems about right” or “let’s kick off the retraining process each night when we have idle compute"

Four Stages of Continual Learning(4)

Stage 2: Automated retraining - Requirements for setup

- If company has ML models in production,
  - likely that company already has most of the infrastructure pieces needed for automated retraining

- The feasibility of this stage revolves around the feasibility of writing a script to automate workflow and configure infrastructure to automatically:
  1. Pull data.
  2. Downsample or upsample this data if necessary.
  3. Extract features.
  4. Process and/or annotate labels to create training data.
  5. Kick off the training process.
  6. Evaluate the newly trained model.
  7. Deploy it.

- In general, the three major factors that will affect the feasibility of this script are:
  - scheduler,
  - data,
  - and model store

Four Stages of Continual Learning(4)

Stage 3: Automated, stateful training

- Need to reconfigure automatic updating script so that, when the model update is kicked off
  - it first locates the previous checkpoint and loads it into memory before continuing training on this checkpoint

- Requirements
  - The main thing needed at this stage is a way to track data and model lineage

- Imagine you first upload model version 1.0
  - is updated with new data to create model version 1.1, and so on to create model 1.2
  - Another model is uploaded and called model version 2.0
  - is updated with new data to create model version 2.1
  - After a while, you might have model version 3.32, model version 2.11, model version 1.64

- Might want to know
  - how these models evolve over time,
  - which model was used as its base model,
  - which data was used to update it so that you can reproduce and debug it

- Need model store that has this model lineage capacity

Four Stages of Continual Learning(5)

Stage 4: Continual learning

- At stage 3, models are still updated based on a fixed schedule set out by developers
  - Finding the optimal schedule isn't straightforward and can be situation-dependent
    - might want models to be automatically updated whenever data distributions shift and the model's performance plummets
  - The move from stage 3 to stage 4 is steep!

- Requirements
  - First need a mechanism to trigger model updates
    - Time-based - every five minutes
    - Performance-based - whenever model performance plummets
    - Volume-based - whenever the total amount of labeled data increases by 5%
    - Drift-based - whenever a major data distribution shift is detected
  - For this trigger mechanism to work, need a solid monitoring solution
    - Hard part is not to detect the changes, but to determine which of these changes matter
    - If monitoring solution gives a lot of false alerts, model will end up being updated much more frequently than it needs to be
  - Need a solid pipeline to continually evaluate model updates.
    - Writing a function to update models isn't much different from stage 3
    - Hard part is to ensure that the updated model is working properly

How Often to Update Your Models?

- Depends on how much gain model will get from being updated with fresh data
  - The more gain model can get from fresher data, the more frequently it should be retrained

- The question on how often to update your model is a difficult one to answer!
  - In the beginning, when infrastructure is nascent and the process of updating a model is manual and slow, the answer is: as often as you can.
  - As infrastructure matures and the process of updating a model is partially automated and can be done in a matter of hours, if not minutes,
    - the answer to this question is contingent on the answer to the following question:
    - "How much performance gain would I get from fresher data?"

- Points to be considered
  - Value of data freshness
  - Model iteration versus data iteration

Value of data freshness

- The question of how often to update a model becomes a lot easier
  - if knows how much the model performance will improve with updating

- For example,
  - If switch from retraining model every month to every week, how much performance gain can we get?
  - What if we switch to daily retraining?

- People keep saying that data distributions shift, so fresher data is better,
  - but how much better is fresher data?
    - One way to figure out the gain is by training model on the data from different time windows in the past and evaluating it on the data from today to see how the performance changes.

Model iteration versus data iteration

- Not all model updates are the same - can be model iteration or data iteration!

- Might wonder not only how often to update model, but also what kind of model updates to perform
  - In theory, can do both types of updates
  - In practice, should do both from time to time

- The more resources spend in one approach, the fewer resources can be spent in another
  - if iterating on data doesn't give much performance gain, then should spend resources on finding a better model
  - if finding a better model architecture requires 100X compute for training and gives 1% performance
  - whereas updating the same model on data from the last three hours requires only 1X compute and also gives 1% performance gain, will be better off iterating on data

- Currently no book can give the answer on which approach will work better for specific model on specific task
  - have to do experiments to find out.

Thank You!

In our next session: