Task 1 (3 points): Individual work

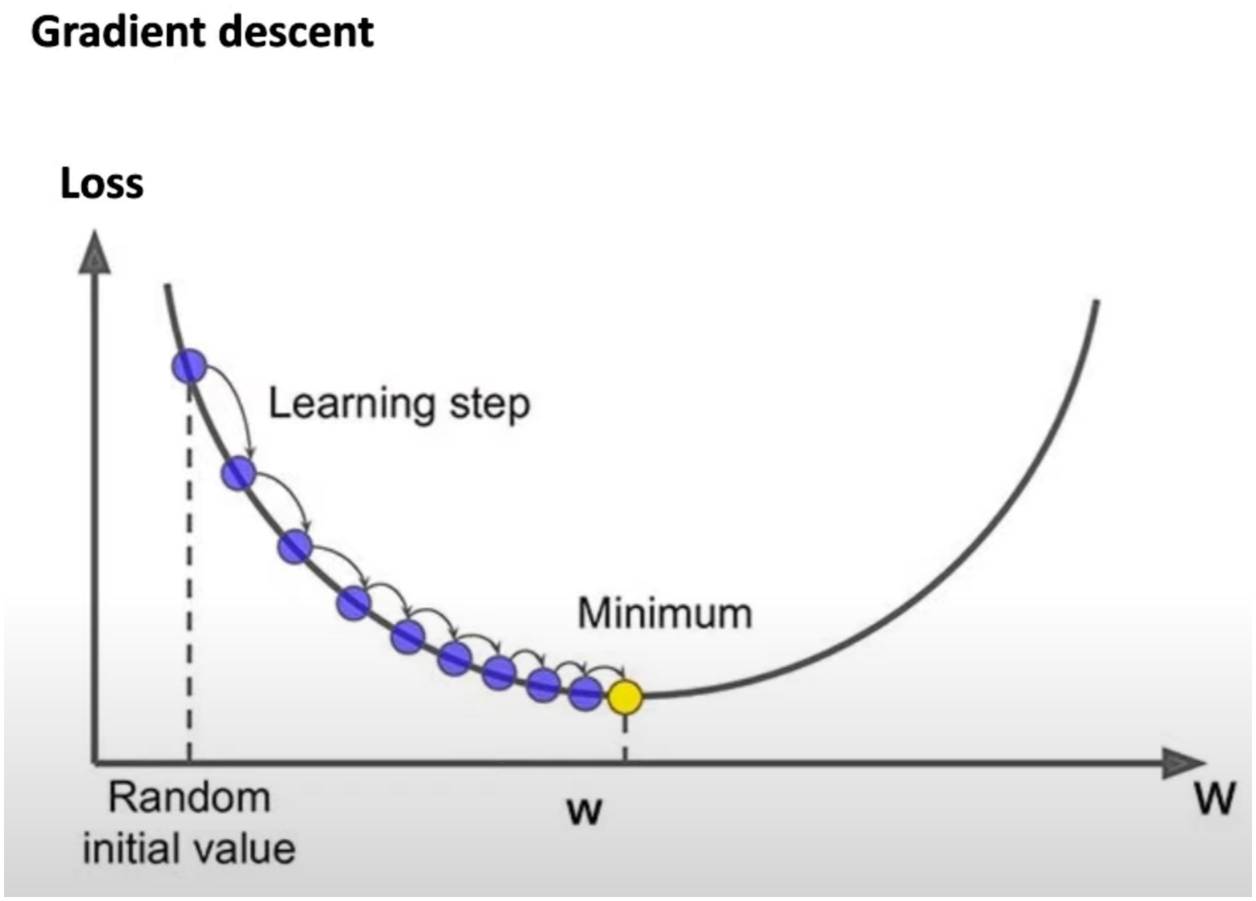
Study your self and present the following issues:

1. Understanding and comparing various Optimizer methods in training machine learning models.

In machine learning, optimizers are algorithms that adjust the model's parameters during training to minimize a loss function. They enable neural networks to learn from data by iteratively updating weights and biases.

So in short, an optimizer updates the model in response to the output of the loss function. They assist in minimizing the loss function

For your information, Gradient Descent is one of the very first optimization algorithms. Louis Cauchy is the first one to suggest it in 1847 and only until 1950 that Gradient Descent was popularized by the machine learning community by the first work being done by Arthur Samuel in the 1950.



Here's a simple description of how Gradient Descent works:

1. We start with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.
2. The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.
3. The derivative of the cost is calculated. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.
4. Now that we know from the derivative which direction is downhill, we can now update the coefficient values.

This process is repeated until the cost function approaches zero, at which point we have found the values of the coefficients that result in the minimum cost.

As a learning student with limited knowledge, I won't be able to present deeply into every single optimization algorithm. Here are brief descriptions included with link to research paper and article to each optimizers to help we understand and compare them:

1. **Gradient Descent:** It is one of the most commonly used optimization algorithms to train machine learning models by minimizing errors between actual and expected results. The main objective of gradient descent is to minimize the convex function using iteration of parameter updates.

   Repeat until convergence {

   $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \; (for \; j = 1 \; and \; j = 0)$$

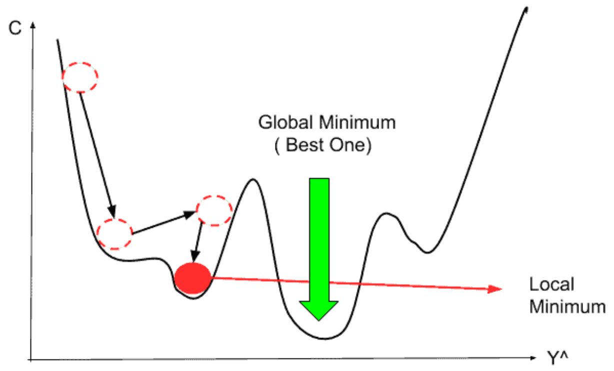
   }

   Advantages: It’s computationally very fast and memory efficient. Commonly used for linear regression.

2. **Stochastic Gradient Descent (SGD):** SGD is a variant of the Gradient Descent algorithm that is used for optimizing machine learning models. It helps in finding the local minimum of a function.

   Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x (i) and label y (i) :

   $$\theta = \theta - \eta . \nabla_{\theta} J(\theta; \, x^{(i)}; y^{(i)})$$



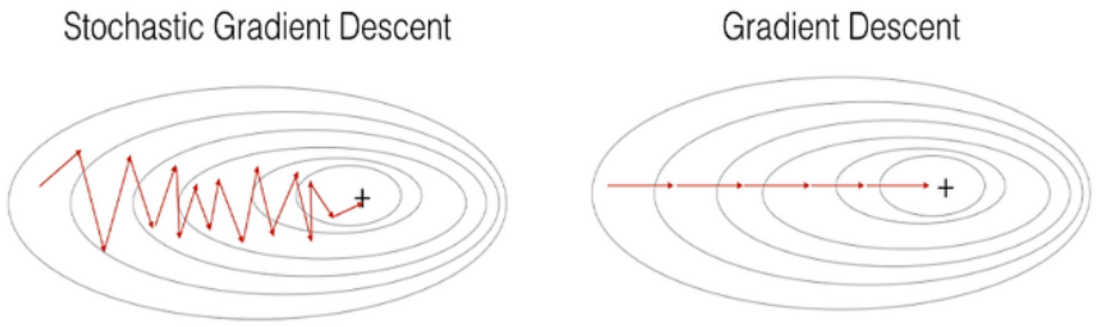


Figure 1 : SGD vs GD
"+" denotes a minimum of the cost. SGD leads to many oscillations to reach convergence. But each step is a lot faster to compute for SGD than for GD, as it uses only one training example (vs. the whole batch for GD).

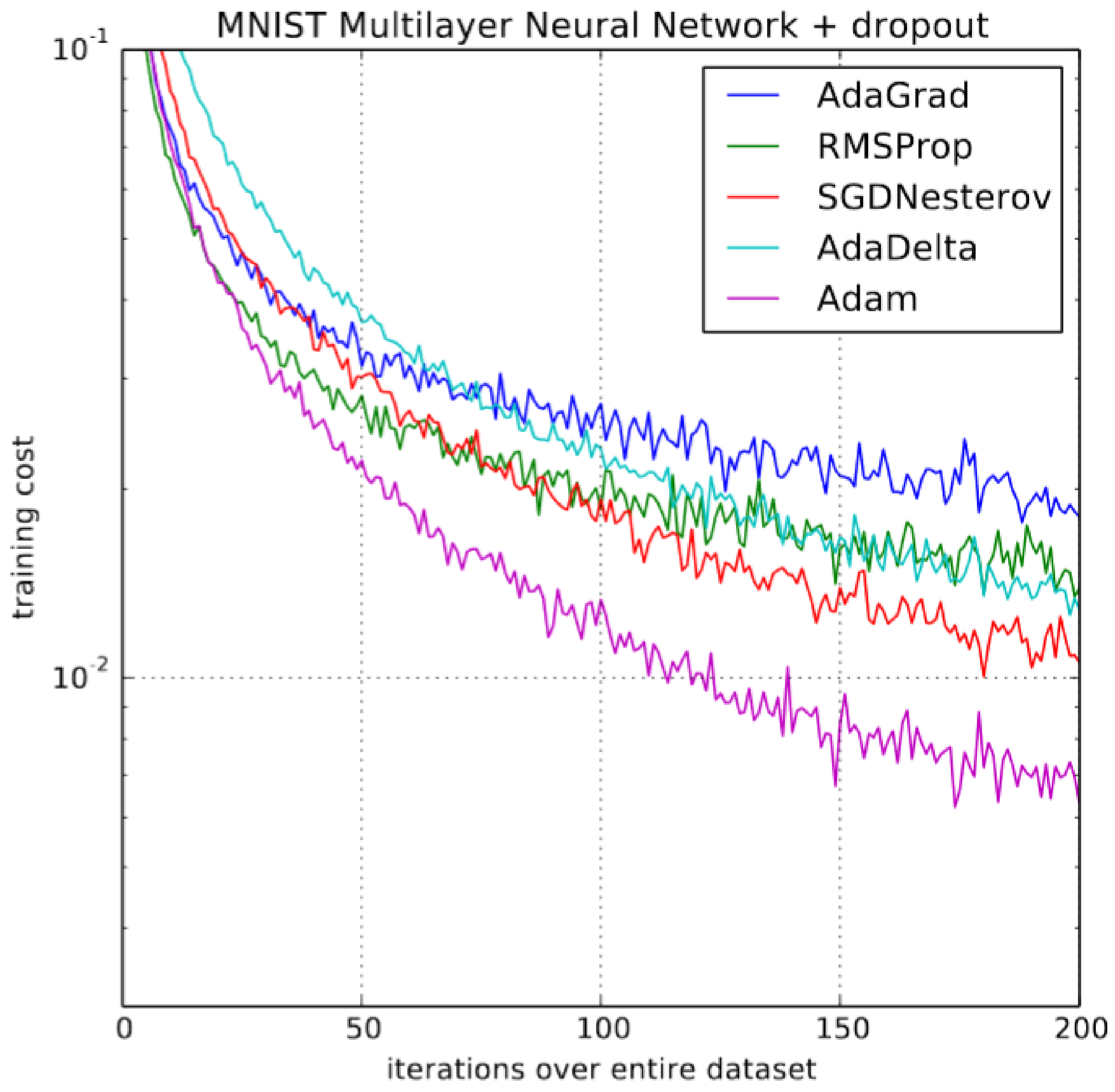
Advantages: Suitable for optimizing non-convex functions.

3. **RMSprop:** RMSprop (Root Mean Squared Propagation) is an optimization algorithm used in deep learning and other Machine Learning techniques. It is a variant of the gradient descent algorithm that helps to improve the convergence speed and stability of the model training process.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Advantages: Has fewer hyperparameters than other optimization algorithms that make it easier to tune and use in practice. Good for recurrent neural networks.

4. **Adam:** Adam, short for Adaptive Moment Estimation, is an optimization algorithm that is used in the field of machine learning to update network weights iteratively based on training data. It adapts the learning rate of each parameter based on its historical gradients and momentum.



Adam vs others

$mt = \beta1 m_{t-1} + (1 - \beta1) gt$

$$vt = \beta 2 v_{t-1} + (1 - \beta 2) g^2 t$$

mt and vt are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As mt and vt are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β1 and β2 are close to 1).

They counteract these biases by computing bias-corrected first and second moment estimates:

$$m\hat{}\, t = \frac{m_t}{1-\beta_1^t}$$

$$v\hat{}\, t = \frac{v_t}{1-\beta_2^t}$$

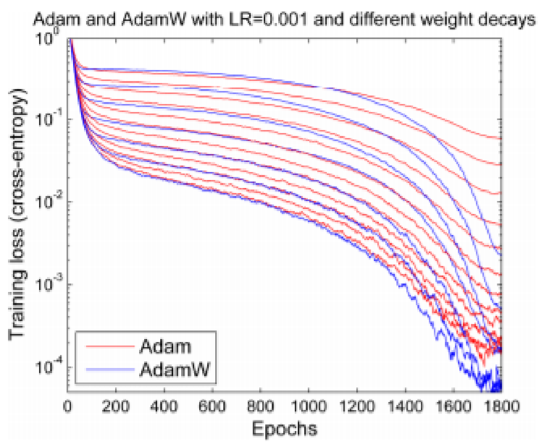
They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule:

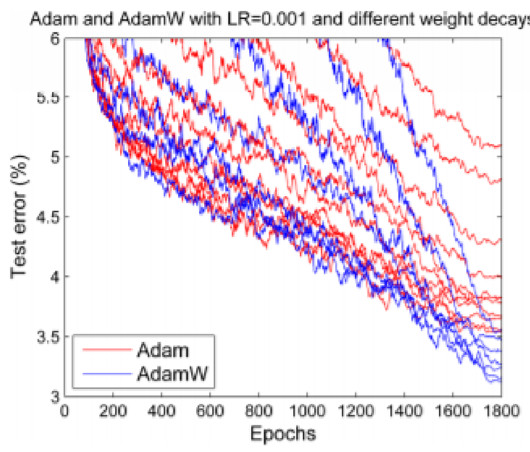
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v\hat{}\, t} + 10^{-8}} m\hat{}\, t$$

The authors propose default values of 0.9 for β1, 0.999 for β2, and 10^-8 . They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

Advantages: Has the ability to handle noisy and sparse datasets, which are common. Can also handle problems where the optimal solution lies in wide range of parameter values

5. **AdamW:** AdamW is an extension of the Adam algorithm that incorporates weight decay for regularization. It is used in PyTorch for training machine learning models.





Comparison between Adam and AdamW

Advantages: Computationally efficient, requiring minimal memory. It calculates individual adaptive learning rates for different parameters, which helps in faster convergence.

6. **Adadelta:** Adadelta is a powerful adaptive learning rate optimization algorithm that addresses the challenge of tuning learning rates in traditional optimization techniques. By leveraging the historical information of gradients, Adadelta adapts the learning rate for each parameter, making the optimization process more efficient and robust.

   Advantages: Doesn't need a default learning rate and doesn't decrease the learning rate as aggressively and monotonically as Adagrad

7. **Adagrad (Adaptive Gradient Descent):** Adagrad is an optimization algorithm used in the field of machine learning and deep learning. The algorithm adapts the learning rate for each weight in the model individually, based on the history of gradients.

   Advantages: Eliminates the need to manually tune the learning rate. Convergence is faster and more reliable.

8. **Adamax:** Adamax is a variant of Adam based on the infinity norm. It adjusts the learning rate based on data characteristics, making it suited to learn time-variant processes.

The vt factor in the Adam update rule scales the gradient inversely proportionally to the `2 norm of the past gradients (via the vt−1 term) and current gradient |gt|^2 :

Advantages: Is an extension of Adam version of gradient descent designed to accelerate the optimization process.

9. **Nadam:** Nadam is a powerful optimization algorithm that combines the best features of Adam and Nesterov Momentum to achieve faster convergence, better handling of noisy gradients, robustness to poor initializations, and ease of implementation.

   Advantages: Combines Nesterov momentum and Adam benefits. Faster convergence and better handling of noisy gradients. (may require careful tuning)

10. **Ftrl (Follow The Regularized Leader):** FTRL is an optimization algorithm developed at Google for click-through rate prediction in the early 2010s. It is most suitable for shallow models with large and sparse feature spaces.

    Advantages: Most suitable for shallow models with large and sparse feature spaces.

So depending on the choice of the optimizer you choose, it will greatly affect the speed of training and performance of your machine learning model. And that is why we should experiment with different optimizers and learning rates to see which works best for specific problems.

2. Exploring “Continual Learning and Test Production” in machine learning.

Continual Learning:

What is continual learning?

Continual learning refers to a model's capacity to learn continuously from an ongoing stream of data. In practical terms, it entails supporting a model's autonomous learning and adaptation in real-time as fresh data is introduced. It's also known as auto-adaptive learning or continual AutoML. The concept behind continual learning is to emulate the human ability to consistently acquire, fine-tune, and transfer knowledge and skills over a lifetime.

In the context of machine learning, the primary objective is to deploy models in a production environment. With continual learning, our aim is to leverage the incoming data in the production environment and retrain the model based on new activities.

For instance, consider Netflix's highly effective recommender system for "Up Next." This system suggests a show immediately after the conclusion of the last episode, creating a compelling user experience that is hard to resist. However, models like this one in production require periodic retraining due to the introduction of new movies, evolving tastes, and emerging market trends. Continual learning addresses this need by using incoming data to automatically retrain the model, ensuring sustained high accuracy and performance.

Why do we need continual learning?

The reason is straightforward—data is dynamic. Changes may occur due to trends or user actions. For example, in 2017, the value of bitcoin was $19K, but within a month and a half, it plummeted to $6K. And now at the end of 2023 its value reached a whopping $42K . Continual learning becomes crucial as it ultimately fine-tunes models for accuracy, enhances overall model performance, and saves time by enabling models to adapt autonomously.

Challenges in Continual Learning

Continual learning, while successful in various industries, confronts some significant challenges that we must address:

Fresh Data Access Challenge:

- Issue: Updating a model every hour demands high-quality labeled training data at the same frequency, making this challenge more critical with shorter update intervals.
- Problem: Data pulled from diverse sources into data warehouses (e.g., Snowflake or BigQuery) varies in mechanisms and speed.
- Solution: To overcome, one approach is direct data extraction from real-time transport before warehouse deposition, especially effective when integrated with a feature store. Challenges include dealing with data residing in external vendor systems and adapting heavy processing from batched ETLs to real-time transport.

Speed of Labeling:

- Issue: Labeling new data quickly is often a bottleneck. Ideal tasks for continual learning involve natural labels with short feedback loops.
- Problem: Obtaining natural labels within the required timeframe is challenging. Weak-supervision, semi-supervision, and fast crowdsourcing are potential solutions, considering trade-offs such as potentially noisier labels.
- Solution: Label computation strategies, whether batched or computed directly from real-time transport (events), impact labeling speed. Streaming computation introduces its own set of challenges.

Evaluation Challenge:

- Issue: Frequent model updates increase the risk of catastrophic failures and open avenues for coordinated adversarial attacks.
- Problem: Testing models before wider deployment is critical but time-consuming, posing limits on the fastest update frequency.
- Solution: Rigorous testing becomes crucial, balancing the need for frequent updates with thorough evaluation. Mitigating risks from adversarial attacks is paramount.

Data Scaling Challenge:

- Issue: Feature calculation often requires scaling, which, in turn, requires global data statistics.
- Problem: Stateful training necessitates considering both previous and new data for global statistics, posing tracking challenges.
- Solution: Incremental calculation or approximation of global statistics as new data is observed helps manage the challenge. Techniques like "Optimal Quantile Approximation in Streams" and partial-fit functions, such as Sklearn's StandardScaler, aid in running statistics.

Algorithm Challenge:

- Issue: Certain algorithms, like matrix-based, dimensionality reduction-based, and tree-based models, face challenges in fast updates due to their reliance on the full dataset for training.
- Problem: Models like PCA dimensionality reduction cannot be incrementally trained quickly.
- Solution: While some variants like Hoeffding Trees support incremental training, their widespread adoption is limited. Addressing this challenge involves exploring alternative algorithms designed for rapid updates, such as Hoeffding Trees and sub-variants.

The Four Stages of Continual Learning

Companies typically progress through four stages when adopting continual learning:

Stage 1: Manual, Stateless Retraining

- Models are retrained only when two conditions are met: severe degradation in performance and availability of time for the update.

Stage 2: Fixed Schedule Automated Stateless Retraining

- This stage is reached when maintaining and improving existing models becomes a priority.
- Retraining frequency is based on a "gut feeling."
- Inflection point: Introduction of a periodic script for stateless retraining.
- High-level steps of the script:
    - Pull data.
    - Adjust data sampling if needed.
    - Extract features.
    - Process and annotate labels for training data.
    - Initiate the training process.
    - Evaluate the new model.
    - Deploy the model.
- Infrastructure needed: Scheduler and a model store (e.g., AWS SageMaker or Databrick's MLFlow).

Stage 3: Fixed Schedule Automated Stateful Training

- Transition involves script reconfiguration and data and model lineage tracking.
- Model lineage versioning example:
    - V1 vs V2: Different model architectures for the same problem.
    - V1.2 vs V2.3: Iterations of stateless retraining.
    - V1.2.12 vs V2.3.43: Number of stateful trainings.
- Use of versioning techniques (e.g., data versioning) to track model evolution.
- Model stores often lack this lineage capability, leading companies to build in-house solutions.
- Multiple models may run concurrently in production through testing models in production.

Stage 4: Continual Learning

- Fixed schedule is replaced by trigger-based retraining mechanisms.
- Triggers can be time-based, performance-based (e.g., performance dropping below x%), volume-based (e.g., a 5% increase in labeled data), or drift-based (detecting a "major" data distribution shift).
- Challenges with drift-based triggers include determining when a data distribution shift adversely affects model performance.

Test Production:

What is Test production?

Testing models in production is a very important part of the Machine Learning lifecycle. It is the act of evaluating the performance of your ML models in a real world environment as the model's performance during training and testing might not accurately reflect how it will perform when exposed to real-world data.

How to sufficiently test your models

To ensure the robustness of your models prior to widespread deployment, it is crucial to conduct both pre-deployment offline evaluations and testing in a production environment. Relying solely on offline evaluations may prove inadequate.

For optimal results, each team should establish a transparent pipeline outlining the model evaluation process. This pipeline should detail the tests to be executed, identify responsible parties for conducting the tests, and establish thresholds that determine when a model is ready for promotion to the next stage. Automation of these evaluation pipelines is highly recommended, with triggers set to initiate assessments whenever a new model update is introduced. Similar to the evaluation of Continuous Integration/Continuous Deployment (CI/CD) in software engineering, the review of stage promotions is paramount.

Testing Strategies in Production

Shadow Deployment

Intuition: Deploy the challenger model alongside the existing champion model, directing all requests to both but serving only the champion's inferences. Compare logged predictions for analysis.

Pros:

- Safest deployment method, preventing buggy new models from serving predictions.
- Conceptually simple.
- Faster attainment of statistical significance due to full traffic to all models.

Cons:

- Unsuitable for tasks requiring user interaction with predictions.
- Expensive as it doubles prediction volume and computational requirements.
- Challenges with potential delays and failures in online prediction modes.

A/B Testing

Intuition: Deploy challenger and champion models simultaneously, directing a portion of traffic to the challenger (model B). Users receive predictions from the challenger, enabling analysis for statistical performance comparison.

Pros:

- Captures user reactions to different models in real-time.
- Simple to understand and implement.
- Cost-effective with one prediction per request.
- Flexibility for A/B/C/D tests.

Cons:

- Less safe than shadow deployments; requires stronger offline evaluation.
- Balancing risk versus sample size for analysis.
- No protection from edge cases as in shadow deployments.
-

Canary Release

Intuition: Deploy challenger and champion models side by side, gradually shifting traffic from the champion to the challenger. Monitor challenger performance; if satisfactory, transition all traffic.

Pros:

- Easy to understand and implement, especially with existing feature flagging infrastructure.
- Suitable for models requiring user interaction.

- Cost-effective with one inference per request.
- Dynamic traffic adjustment with A/B testing.

Cons:

- Potential for less rigorous performance determination.
- Supervision needed to prevent accidents; less safe but easy to rollback.

Interleaving Experiments

Intuition: Users receive interleaved predictions from both models, and user preference is measured. Suitable for recommendation tasks, particularly with Netflix's team-drafting method.

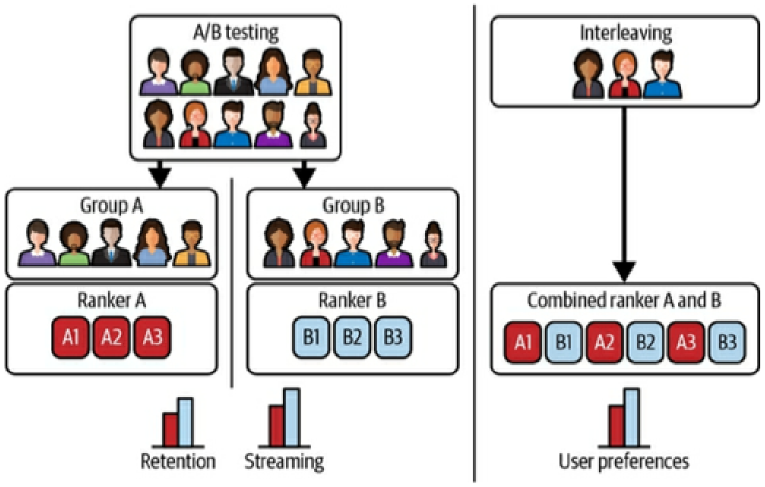


Figure 9-6. An illustration of interleaving versus A/B testing. Source: Adapted from an image by Parks et al.

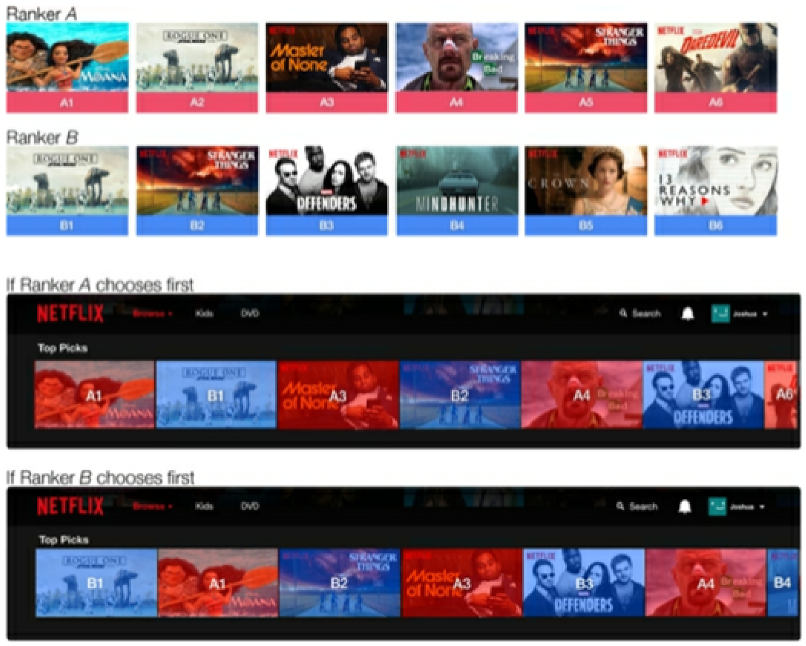


Figure 9-7. Interleaving video recommendations from two ranking algorithms using team draft. Source: Parks et al.[32]

Pros:

- Identifies the best model with smaller sample sizes compared to traditional A/B testing.
- Captures user behavior against predictions.
- Full traffic for both models.

Cons:

- More complex implementation than A/B testing.
- Requires handling delays or failures in interleaved models.
- Doubles computational requirements.
- Limited applicability; not suitable for all tasks or large numbers of challenger models.

Bandit Strategies

Intuition:
Bandits are algorithms that dynamically decide, on each request, whether to use the current best-performing model (exploit) or explore other models to gain more information. They introduce the concept of opportunity cost to the decision-making process.
Requirements:

- Online predictions are necessary; batched offline predictions are incompatible.

- Short feedback loops are essential to update the bandit algorithm with performance feedback promptly.

Algorithms:

- Various bandit algorithms exist, with epsilon-greedy being the simplest, and Thompson Sampling and Upper Confidence Bound (UCB) being widely used and powerful.

Pros:

- Data Efficiency: Bandits require significantly less data than A/B testing to determine the better model. For instance, 630K samples for 95% confidence in A/B testing versus 12K with bandits.
- Optimality: Bandits are often considered optimal due to their data efficiency and minimal opportunity cost.
- Safety: Safer than A/B testing as the algorithm selects bad models less frequently. Convergence is faster, enabling quicker elimination of underperforming challengers.

Cons:

- Implementation Complexity: Bandits are challenging to implement due to continuous feedback propagation into the algorithm.
- Use Case Limitations: Applicable only to certain use cases (online predictions with short feedback loops).
- Safety Concerns: Not as safe as Shadow Deployments since challengers take live traffic.

Link for references:
Part 1:

https://keras.io/api/optimizers/

https://arxiv.org/abs/1412.6980

https://www.youtube.com/watch?v=JhQqquVeCE0

Gradient Descent For Machine Learning - MachineLearningMastery.com

Gradient descent - Wikipedia
Adam Definition | DeepAI
Part 2:
https://arxiv.org/abs/2112.08654

https://blog.research.google/2022/04/learning-to-prompt-for-continual.html
https://www.cell.com/action/showPdf?pii=S1364-6613%2820%2930219-9
https://blog.cnvrg.io/5-steps-to-maximize-business-impact-with-machine-learning
https://github.com/serodriguez68/designing-ml-systems-summary/blob/main/09-continual-learning-and-test-in-production.md?fbclid=IwAR0lmLjldD9fhuJLA-R8myJ8SYD1i0m1gat5aqKxBeNnoc5swaEbWL2Pj94#continual-learning-challenges

-End-