Question 1:

1. Understanding and comparing various Optimizer methods in training machine learning models.
2. Exploring "Continual Learning and Test Production" in machine learning.

Optimizer methods in training machine learning models:

-In machine learning we have types of optimizers to create a model that has optimal performance and can make accurate predictions. Optimization in machine learning is the process of adjusting parameters, attributes to minimize a cost function. The goal of minimizing the cost function is to ensure that the difference between the predicted value and the actual value is not significant.

-There are many types of optimizer methods in machine learning. Each will have their own use case and benefits.

1. **Feature scaling.**
    a. **Normalization (min-max scaling or min-max normalization):** This method involves scaling the range values of features to range of [0,1]. The general formula for normalization is $x' = \frac{x - min(x)}{max(x) - min(x)}$. Where $max(x)$ is the largest value and $min(x)$ is the smallest value. We can also normalize over different intervals between [a,b] with the formula $x' = a + \frac{(x - min(x))(b-a)}{max(x) - min(x)}$.
    b. **Standardization**: This method is a technique that transforms the data so that each feature has a mean of zero and a standard deviation of one. This helps to normalize the data and reduce the effect of different scales and ranges. The formula for this method is $x' = \frac{x - \bar{x}}{\sigma}$. This will determine the distribution mean and standard deviation for each feature and calculate the new data point.
    c. **Scaling to unit length:** This method involves dividing each feature vector by the Euclidean length of the vector so that the resulting

vector has a length of one. This is the formula of the technique: $x' = \frac{x}{||x||}$.

d. **Log scaling:** This method is helpful when there are many values that have many points and some others have few points. This data distribution type is called power law distribution. Log scaling will help compute the log of the values to compress a wide range to a narrow range. The formula for log scaling is $x' = log(x)$

e. **Z-score:** This method is a variation of scaling that scales the values of a feature to have a mean of 0 and a standard deviation of 1. The formula for calculating the z-score of a point: $x' = \frac{x-\mu}{\sigma}$

Pros:

- Scaling help prevent the model from giving a higher weight to certain attribute compared to others
- Feature scaling helps to make Gradient Descent converge much faster.

Cons:

- It does not always guarantee better results.

2. **Batch Normalization**

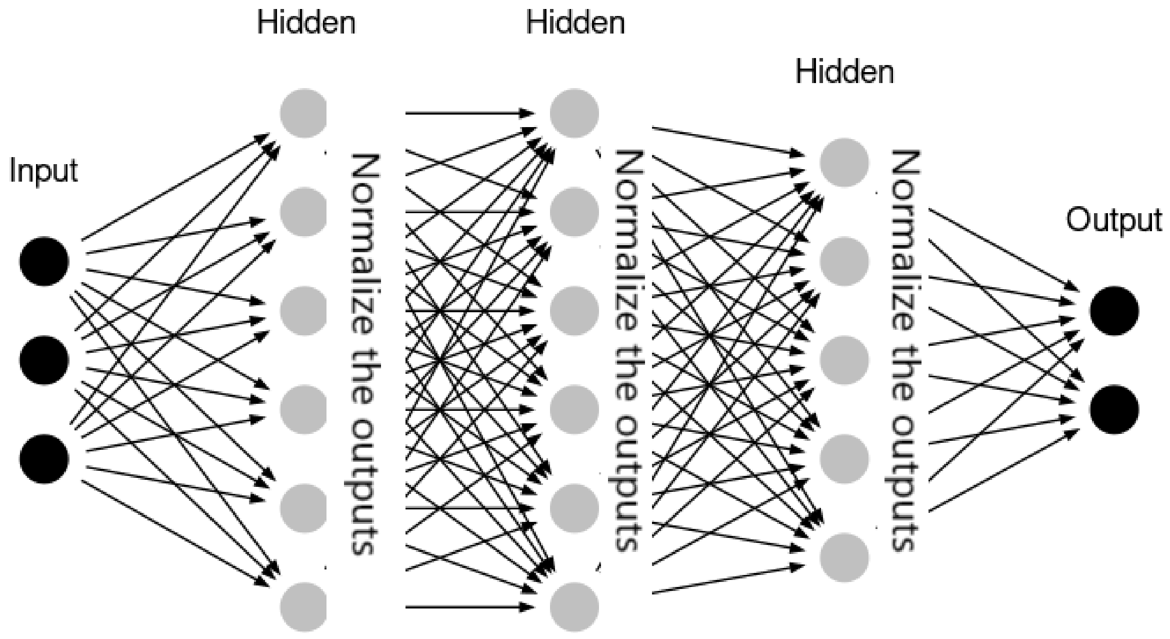
- This is a method that has been proposed to address the vanishing/ exploding gradient problem. This learns two parameters to find the optimal scale and mean of the inputs for each layer. It's placed just before the activation function of each layer. It zero-center and normalizes the inputs over the current mini-batch then scales the result for each layer. The model is then trained to find the optimal values for the scaling and shifting parameters.

Pros:

- Reduce the vanishing gradients problem
- Less sensitive to the weight initialization
- Able to use much larger learning rates to speed up the learning process
- Acts like a regularizer

Cons:

- Slower predictions due to the extra computations at each layer
- Not good for RNN, LSTM, online learning



Location of batch normalization in a model

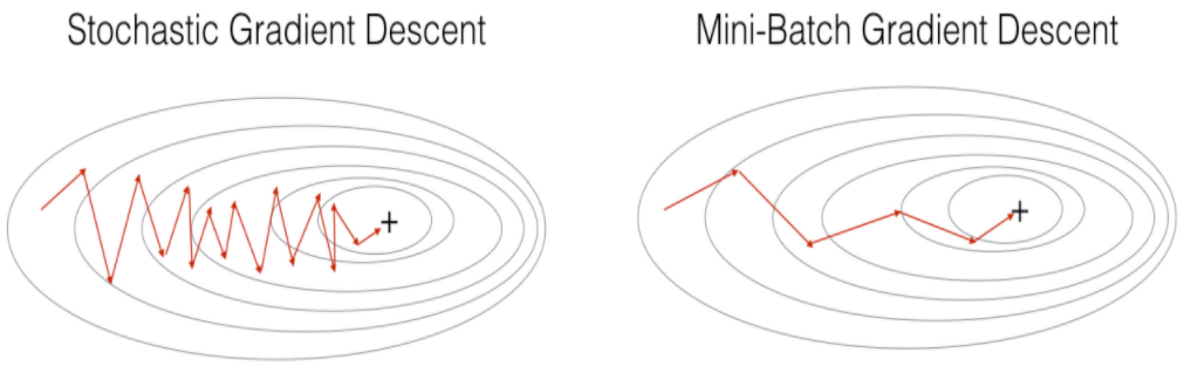
3. **Mini-Batch Gradient Descent**
   - This method is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients. This method seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

Pros:

- Update frequency is higher than batch gradient descent which allows for a robust convergence, avoiding local minimum.
- The batched updates provide a computationally more efficient process than stochastic gradient descent
- Batching allows both efficiency of not having all training data in memory and algorithm implementation

Cons:

- Mini-batch requires the configuration of an additional “mini-batch size” hyperparameter for the learning algorithm
- Error information must be accumulated across mini-batches of training examples like batch gradient descent.



Compare the efficiency between stochastic gradient descent and mini-batch gradient descent

4. **Gradient Descent with Momentum:**
   - This is a method that helps accelerate the optimization process, decrease the number of function evaluations required to reach the optima, and improve the capability of the optimization algorithm which leads to better final results. We have the formula for gradient descent with momentum:

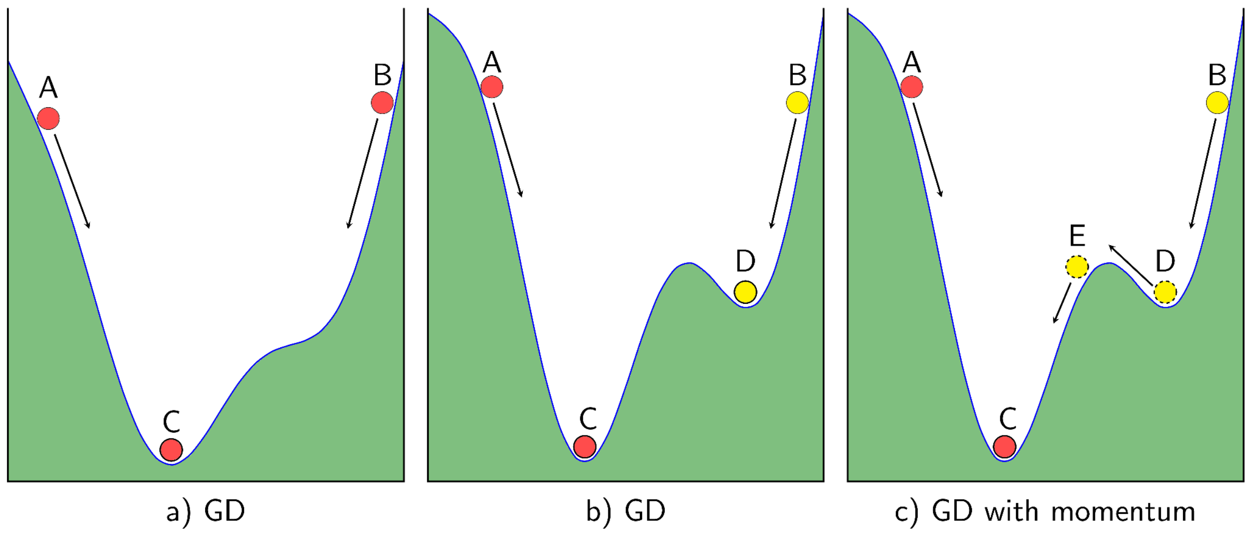
$$v_t = \gamma * v_{t-1} + n\Delta w_t$$
$$w_{t+1} = w_t - v_t$$

Pros:

- Faster convergence than traditional other gradient descent techniques

Cons:

- Takes larger steps even in the regions of the gentle slopes, as the momentum of the history gradients is carried with it every step
- If the momentum is too much, this technique miss the local minima



Gradient descent with momentum

5. **RMSProp Optimization:**
   - This is a gradient-based optimization method used in training neural networks and was developed as a stochastic technique for mini-batch learning. RMSProp deals with the problem of vanishing or exploding gradients by using a moving average of squared gradients to normalize the gradient. This will balance the step size (momentum), decreasing the step for large gradients to avoid exploding and increasing the step for small gradients to avoid vanishing. This technique also uses adaptive learning rate instead of treating the learning rate as a hyperparameter. The update formula is given as:

$$v_{dw} = \beta . v_{dw} + (1 - \beta). dw^2$$

$$v_{db} = \beta . v_{dw} + (1 - \beta). db^2$$

$$W = W - \alpha . \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

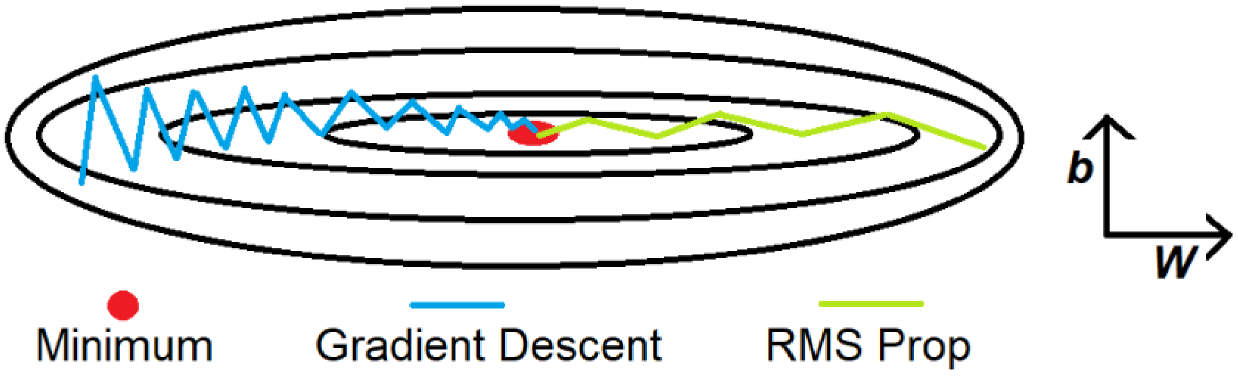
$$b = b - \alpha . \frac{db}{\sqrt{v_{db}} + \epsilon}$$

Pros:

- Very robust optimizer which has pseudo-curvature information
- Can deal with stochastic objectives very nicely, making it applicable to mini batch learning
- Faster convergence than momentum

Cons:

- Learning rate is still manual, because the suggested value is not always appropriate for every task
- Implementation of RMSProp Descent with Employee Attrition



Root Mean Squared Propagation

6. **Adam Optimization:**
   - This is a first-order-gradient-descent algorithm of stochastic objective function, based on adaptive estimates of lower-order moments. Adam is one of the latest state-of-the-art optimizations being used by many practitioners of machine learning. The first moment normalized by the second moment gives the direction of the update. The update rules is as follows:

$$\theta_{n+1} = \theta_n - \frac{\alpha}{\sqrt{\hat{v}_n + \epsilon}} \hat{m}_n$$

Pros:

- Easy to implement
- Quite computationally efficient
- Requires little memory space
- Good for non-stationary objectives
- Works well on problems with noisy or sparse gradients
- Works well with large data sets and large parameters

Cons:

- Adam optimizer tends to converge faster, but other algorithms like stochastic gradient descent focus on the data points and generalize in a better manner

- The performance depends on the type of data being provided and the speed/generalization trade-off.

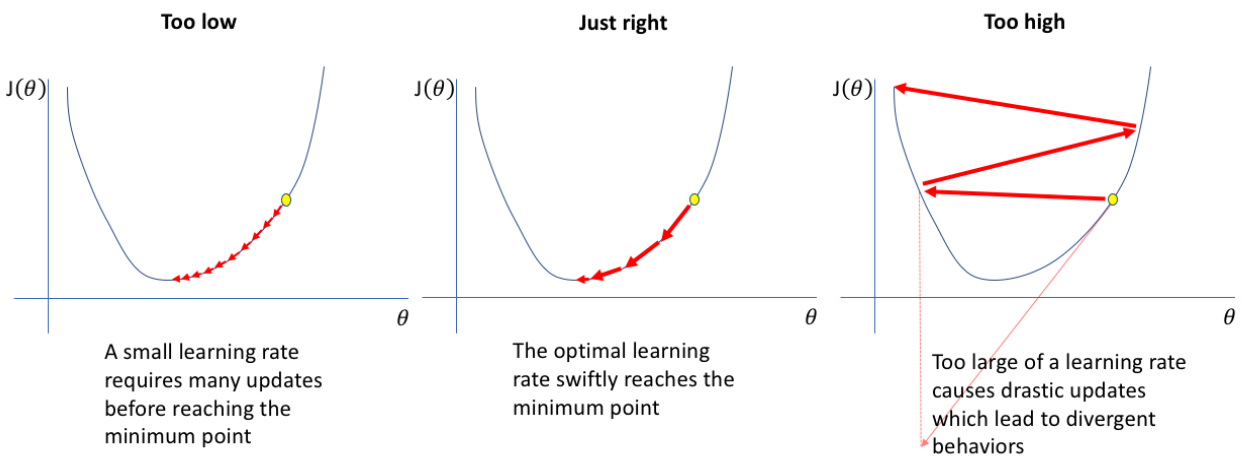
7. **Learning Rate Decay**
   - The learning rate in machine learning is a hyperparameter which determines to what extent newly acquired information overrides old information. It is the most important hyperparameter to tune for training deep neural networks. Learning rate will control both the speed of convergence and the ultimate performance of the network. During earlier iterations, faster learning rates lead to faster convergence while during later epochs, slower learning rate produces better accuracy. Changing the learning rate over time can overcome this tradeoff.

Pros:

- Set the learning in a nice place
- Avoid oscillation

Cons:

- Problem with selecting the correct learning rate
- Too small fixed learning rate causes slow convergence.


Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Use learning rate decay to find the optimal learning rate

8. **Stochastic Gradient Descent**
   - This is an extension of the Gradient Descent algorithm and it overcomes some of the disadvantages of the gradient descent. In gradient descent it requires a lot of memory to load the entire dataset at a time to compute the derivative of the loss function. In the SGD

algorithm derivative is computed taking one point at a time. The derivative can be computed with:

$$\theta = \theta - \alpha . \frac{\partial(J(\theta; x(i), y(i)))}{\partial\theta}$$

Pros:

- Less memory needed

Cons:

- Time to complete 1 epoch is larger than Gradient Descent
- Takes a long time to converge
- May stuck in local minima

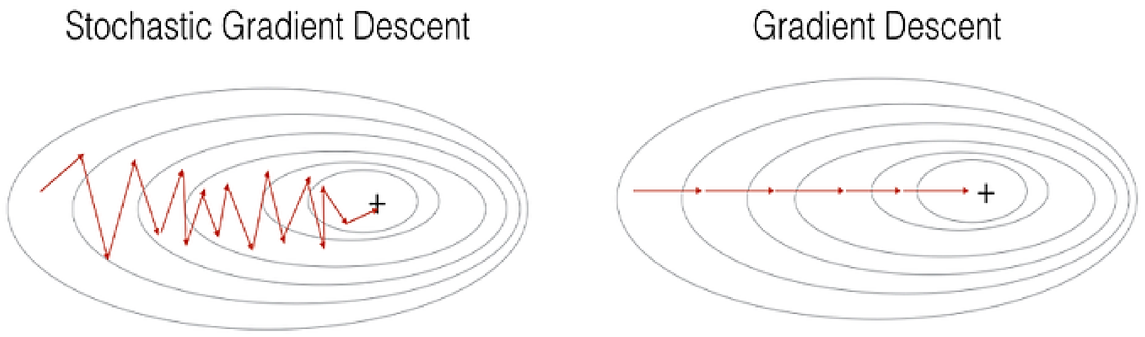


Figure 1 : SGD vs GD
"+" denotes a minimum of the cost. SGD leads to many oscillations to reach convergence. But each step is a lot faster to compute for SGD than for GD, as it uses only one training example (vs. the whole batch for GD).

SGD vs Gradient Descent

9. Nesterov Accelerated Gradient (NAG)

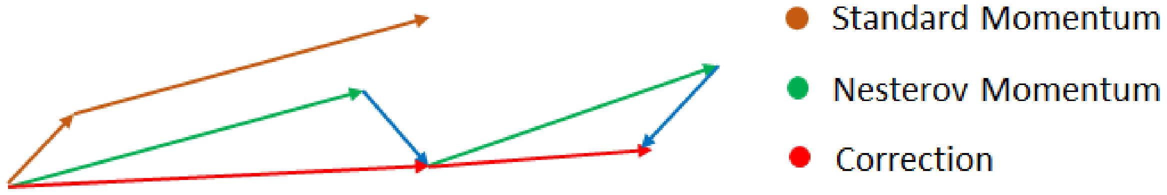
- This technique is an extension of momentum that involves calculating the decaying moving average of the gradients of projected positions in the search space rather than the actual position. The updating formula for NAG is: $v_t = \gamma v_{t-1} + \alpha . \frac{\partial(J(\theta - \gamma V_{t-1})}{\partial\theta}$

Pros:

- Less memory needed
- Converge faster

Cons:

- Need to compute one more variable before each update



NAG convergence

10. Adaptive Gradient Descent (AdaGrad)

- The key idea for this technique is to have an adaptive learning rate for each of the weights. It will perform smaller updates for parameters associated with infrequently occurring features. Its update rule is:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

Pros:

- No need to update the learning rate manually as it changes adaptively with iterations

Cons:

- As the number of iterations becomes very large, the learning rate decreases to a very small number which leads to slow convergence.

11. AdaDelta

- AdaDelta algorithm has an idea to take an exponentially decaying average. Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. The formula for AdaDelta:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Pros:

- No need to update the learning rate manually as it changes adaptively with iterations

Cons:

- As the number of iterations becomes very large, the learning rate decreases to a very small number which leads to slow convergence.

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

1. **Continual learning:**

   What is continual learning?

   - Continual Learning is a concept of updating a model whenever new data becomes available, this will help the model relevant and accurate with the current data distribution. This allows the model to adapt to the changing data distribution and avoid forgetting previous knowledge.
   - Continual learning is often mistaken with a special class of ML algorithm that can update the model incrementally with each new datapoint. These algorithms are sequential bayesian updating and KNN classifiers. This class of algorithm is small and is sometimes referred to as “online learning algorithms”. Continual learning can be applied to any supervised ML algorithm.
   - Continual learning is not the same as retraining a model whenever there is a new datapoint. This is a risky approach, because it can cause neural networks to forget. Continual learning is about updating a model in a way that preserves its previous knowledge.

+ Why continual learning?

  Continual learning is essential for models that need to adapt to data distribution shifts. Some scenarios require fast and flexible responses to changing situations. For example: Scenarios where unforeseen and rapid changes can occur, scenarios where training data for a specific event is unavailable, scenarios where the cold start problem is a challenge.

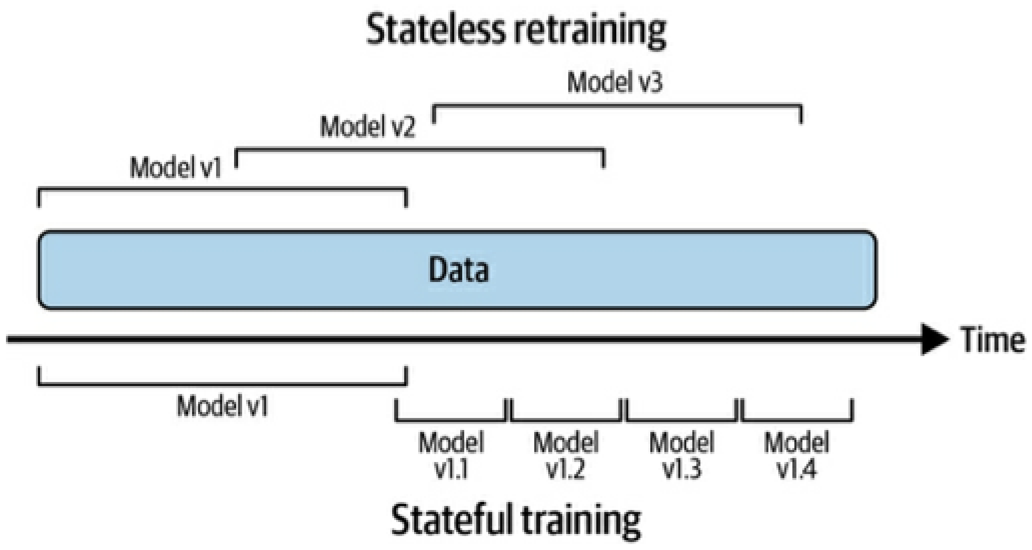


Figure 9-2. Stateless retraining versus stateful training

There are 2 concepts for continual learning: stateless retraining and stateful training.

1. Stateless retraining: This concept involves retraining a model from scratch each time, using randomly initialized weights and fresher data. There might be some overlap with data that had been used for training previous model versions.
2. Stateful training: Initialized a model with the weights from the previous training round and continue the training using new unseen data. This will allow that model to update with significantly less data, also the model will be able to converge faster and use less compute power. Occasionally, a stateless retraining is conducted with large training data to re-calibrate the model.

+ Model iteration vs data iteration: Stateful training is most used to incorporate new data into an existing and fixed model architecture. If a change to the model's features and architecture is needed, you will need to do a first-pass of stateless retraining.

Feature reuse through log and wait: Inference requires feature calculation for each data sample. Some companies keep the calculated features in a storage system, so they can use them later for continual learning training. This way they can reduce the computation cost.

+ Continual learning challenges

  Continual learning has been applied in industry with great success. However it has three major challenges that companies need to overcome:

  1. Fresh data access challenge:

     To keep the model updated with hourly data, reliable and accurate data labels are required. This challenge becomes more crucial as the update frequency increases.

     a. Speed of data deposit into data warehouses: Most data is stored in a data warehouse like Snowflake or BigQuery. This storing method has a downside of each data coming from different sources is deposited into the warehouse using different mechanisms and at different speeds. A common approach to solve this is to pull data directly from real-time transport for training before it is deposited into the warehouse. There are also some problem with this approach such as you won't have all the data pumping through events, batched ETLs (Extract-Transform-Load) do a lot of heavy processing and joining data so if we use real-time transport approach we will need to figure out how to do that in data stream.

     b. Speed of labeling: The speed of labeling new data is often the bottleneck. The best candidates for continual learning are tasks that have natural labels with short feedback loops. If natural labels are not easy to obtain in the timeframe needed, we can also try weak-supervision or semi-supervision techniques to get them in time.

  2. Evaluation challenge:

     Adopting continual learning as a practice comes at the risk of model failures. The more frequent the model update, the more chances that the model has to fail. This means that testing the model is crucial step before deploying them

  3. Data scaling challenge:

To calculate features, we often need to scale the data. Scaling depends on global data statistics such as min, max, average and variance. When we train a model with stateful training, we have to use global statistics from both the previous and the new data. This can be difficult to do.

4. Algorithm challenge:

   Some algorithms require the entire dataset for training, such as matrix-based, dimensionality reduction-based and tree-based models. These algorithms cannot be updated incrementally with new data, unlike neural networks or other weight-based models.

+ Stages of Continual Learning.

  Stage 1: Manual, stateless training
  Stage 2: Fixed schedule automated stateless retraining
  Stage 3: Fixed schedule automated stateful training
  Stage 4: Continual Learning

2. **Testing models in Production**

   To sufficiently test models before deployment, both pre-deployment offline evaluation and testing in production is needed. Offline evaluations alone are not sufficient. It is best if these evaluation pipelines are automated and kicked-off when there is a new model update. The stage promotions should be reviewed similar to how CI/CD is evaluated in software engineering.

   Pre-deployment offline evaluations

   The most common two are test split and backtests:

   - Test splits are usually static so that we have a trusted benchmark to compare multiple models. This also means that if the model performs well in test split it doesn't mean that it will perform well in the data distribution condition in production.
   - Backtesting is the idea of using the freshest labeled data that hasn’t been seen by the model during training to test performance. Production performance is much more label-related performance.

   Testing in Production Strategies

   + **Shadow Deployment:** Deploy the challenger model in parallel with the existing champion model. Send every incoming request to both models but only serve the inference of the champion model. Log the prediction for the both models to compare them.

Pros:

-This is the safest way to deploy the models. Even if the new model is buggy, prediction will not be served.

-It is conceptually simple.

-Your experiment will gather enough data to achieve statistical accuracy faster than all other strategies as all models receive full traffic.

Cons:

- This technique can't be used when measuring model performance depends on observing how the user interacts with the predictions.

-This technique is expensive to run

+ **A/B Testing:** Deploy the challenger model alongside the champion model (model A) and route a percentage of traffic to the challenger (model B). Predictions from the challenger are shown to the users. Use monitoring and prediction analysis on both models to determine if the performance of the challenger is better than the champion.

  Pros:
  - This technique captures how users react to different models.
  - Simple to understand and there are a lot of libraries and documentation
  - It is cheap to run

  Cons:
  - Less safe than shadow deployments
  - Choice to make between more risk vs gaining enough samples to make an analysis faster

+ **Canary Release:** Deploy challenger and champion side by side but start with the challenger taking no traffic. Slowly move traffic from the champion to the challenger (aka the canary). Monitor the performance metrics of the challenger, if they look good, keep going until all traffic is going to the challenger.

  Pros:

        - Easy to understand
        - Simplest of all strategies to implement
        - Use this with models that require user interaction to capture performance.
        - Cheaper to run compared to shadow deployment

    Cons:

        - Open the possibility to not be rigorous in determining performance differences
        - If not supervised carefully, accidents can happen

+ **Interleaving Experiments:** In A/B testing a single user either gets the predictions from model A or model B. In interleaving, a single user gets interleaved predictions from both model A and model B. We then track how each model is doing by measuring the user preference with each model's predictions.

    Pros:

    - Best model with smaller sample size
    - Capture how users behave against predictions

    Cons:

    - More complex than A/B testing
    - Double the compute power
    - Cannot be used for all types of tasks
    - Not easily scale

+ **Bandits:** Bandits are an algorithm that keeps track of the current performance of each model variant and makes a dynamic decision on every request on whether to use the model that has the most performance or try out any of the other models to gain more information about them.

    Pros:

    - Need a lot less data than A/B testing
    - More data efficient
    - Safer because if a model is really bad.

    Cons:

    - Much harder to implement
    - Can only be used on certain use cases
    - Not safe as shadow deployments