UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

6CS012/HJ1: Artificial Intelligence and Machine Learning

(Q&A)

Full Name : Diparshan Baral

University ID : 2358244

Module Leader : Siman Giri

University email : D.Baral@wlv.ac.uk

Date of submission: 2025-05-20

Table of Contents

1. Long Question: 1
   1.1. Data Drift 1
   1.2. Imbalanced Data 2
   1.3. Latency and Slow Predictions 2
2. Short Question 3
   2.1. Overfitting 3
   2.2. Neural Network Architecture 4

1. Long Question:

You are a Machine Learning Engineer at a growing e-commerce company preparing to implement and scale machine learning systems.

• List and explain at least three real-world challenges you expect during ML model development, deployment, or maintenance (e.g., data drift, imbalanced data, system latency).

• For each challenge:

– Discuss potential consequences if not properly addressed.

– Propose technical or organizational solutions you would implement (e.g., retraining pipelines, feature monitoring, distributed serving, MLOps practices).

• Finally, reflect on how cross-functional collaboration (between data scientists, engineers, product teams) can help mitigate these challenges more effectively.

Answer:

As a Machine Learning Engineer at a fast-growing e-commerce company, here are three real-world challenges I'd expect when building and scaling ML systems:

1.1. Data Drift

What it is: Over time, the type of data coming in might change. For example, users may start behaving differently or we might add new products that weren't there when the model was trained.

Why it's a problem: If the model was trained in old patterns, it might start making bad predictions.

How to fix it: We can set up monitoring tools to keep an eye on incoming data and compare it to training data. If we notice big changes, we can retrain the model regularly to keep it updated.

1.2. Imbalanced Data

What it is: Sometimes one class is way more common than the other — like 95% of users don't return items, but 5% do.

Why it's a problem: The model might just learn to ignore the rare cases and get lazy — it could predict "no return" all the time.

How to fix it: Use techniques like oversampling, undersampling, or special algorithms that can handle imbalanced data better (like XGBoost with class weights).

1.3. Latency and Slow Predictions

What it is: Models that are too slow can delay responses, especially during peak shopping times.

Why it's a problem: Nobody wants to wait — slow systems can mean lost sales or bad user experience.

How to fix it: Use lighter models for real-time tasks or deploy models with optimized serving solutions like TensorFlow Serving or use caching.

2. Short Question

2.1. Overfitting

Overfitting is a common challenge in deep learning models.

• Describe at least two techniques commonly used to prevent or reduce overfitting (e.g., dropout, early stopping, data augmentation).

• For each method:

– Briefly explain how it works.

– Provide a practical example of how it would be applied in a real-world deep learning project (e.g., image classification, sentiment analysis).

Answer:

Overfitting is when your model does great on the training data but messes up on new, unseen data. Two simple ways to handle this are:

Dropout:

How it works: Dropout randomly turns off some neurons during training, so the model doesn't rely too much on any one path. This makes it more flexible and better at handling new data.

Example: In an image recognition project, like recognizing digits, dropout helps make sure the model doesn't just memorize the training images but can also work on new ones.

Early Stopping:

How it works: While training, we keep checking how well the model does on test data. If the performance starts to get worse, we stop training early before it overfits.

Example: In a project like classifying positive or negative reviews, early stopping helps make sure the model doesn't memorize training texts and stays general.

2.2. Neural Network Architecture

What are the main differences between normal Neural Network architecture and autoencoders? Explain in brief with example, how autoencoders can be applied in various contexts (problems at least one)? Explain how they help to address these problems.

Answer:

Main difference: A regular neural network is built to predict or classify something — like whether a message is spam or not. An autoencoder, on the other hand, tries to copy the input. It first compresses the input (encoder), then try to recreate it (decoder) and in between encoder and decoder it has a bottle neck.

Example: Let's say we give it an image. The autoencoder learns to compress the image into fewer numbers, then rebuilds it. If it does this well, it means it learned the important features.

Where it helps: Autoencoders are super useful in finding weird or unexpected data like fraud. If we train one on normal transaction data, it'll do a good job recreating it. But if a new transaction looks very different (maybe it's a fraud), it won't be able to recreate it well and that tells us something's wrong.

They're also great for cleaning data or reducing size — especially when we have large and messy info like transaction logs.