Question 1: Learning Rate Scheduling in Gradient Descent

Learning rate is one of the most important hyperparameters in the training of neural networks, impacting the speed and effectiveness of the learning process. A learning rate that is too high can cause the model to oscillate around the minimum, while a learning rate that is too low can cause the training process to be very slow or even stall. This article provides a visual introduction to learning rate schedulers, which are techniques used to adapt the learning rate during training.

What is a Learning Rate?

In the context of machine learning, the learning rate is a hyperparameter that determines the step size at which an optimization algorithm (like gradient descent) proceeds while attempting to minimize the loss function.

When it comes to training deep neural networks, one of the crucial factors that significantly influences model performance is the learning rate.

The learning rate determines the size of the steps taken during the optimization process and plays a pivotal role in determining how quickly or slowly a model converges to the optimal solution.

In recent years, adaptive learning rate scheduling techniques have gained prominence for their effectiveness in optimizing the training process and improving model performance.

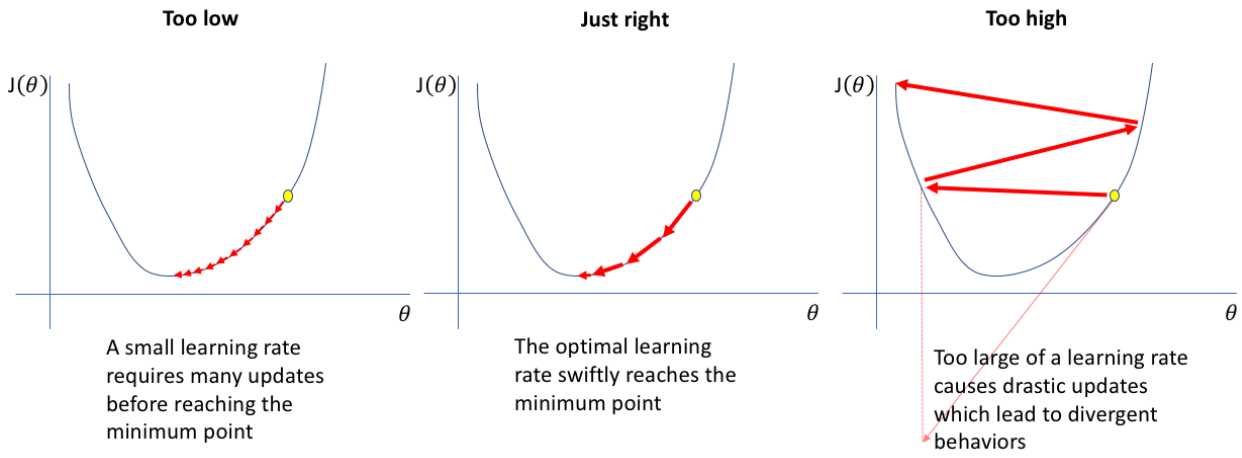
Understanding the Importance of Learning Rate

let's first understand why the learning rate is so important in training deep neural networks.

In essence, the learning rate controls the amount by which we update the parameters of the model during each iteration of the optimization algorithm, such as stochastic gradient descent (SGD) or its variants.

A learning rate that is too high can lead to overshooting the optimal solution, causing oscillations or divergence in the optimization process.

On the other hand, a learning rate that is too low can result in slow convergence and potentially get stuck in local minima.



The Need for Adaptive Learning Rate Scheduling

Traditional approaches to setting the learning rate often involve manually tuning hyperparameters based on heuristics or trial and error.

However, in practice, finding the optimal learning rate can be challenging, especially when dealing with large and complex datasets or architectures.

This is where adaptive learning rate scheduling techniques come into play.

Instead of relying on a fixed learning rate throughout the training process, these techniques dynamically adjust the learning rate based on the feedback received during training.

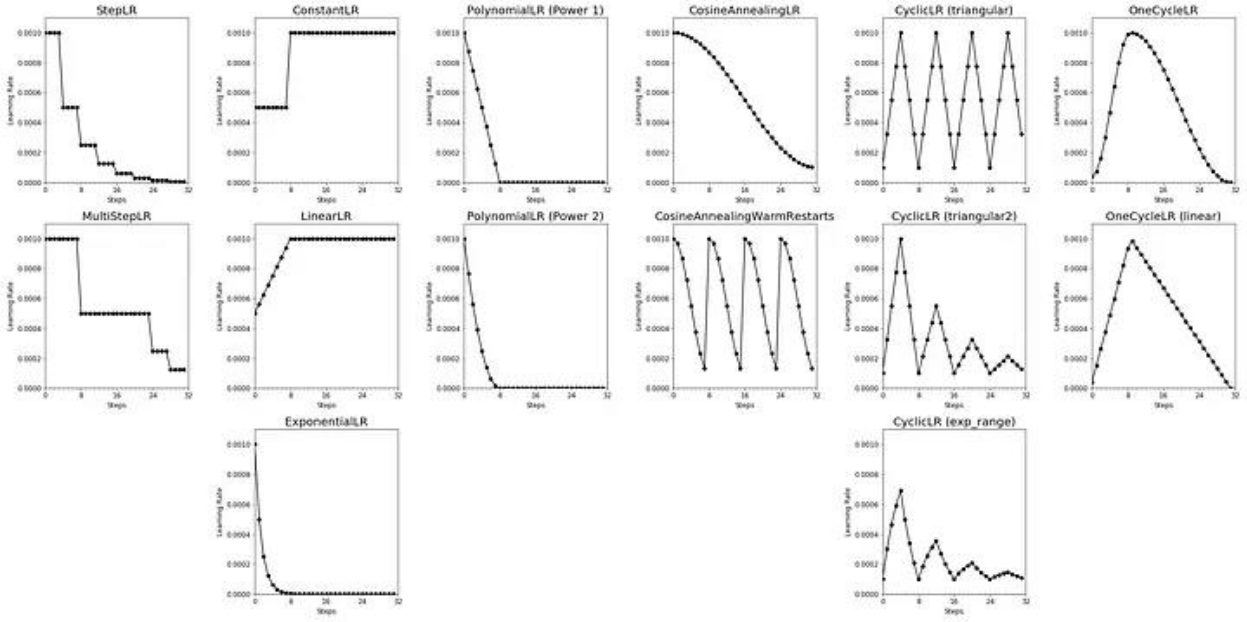
By adapting the learning rate according to the characteristics of the optimization landscape, adaptive scheduling methods aim to achieve faster convergence and improved generalization performance.

Reference: Adaptive Learning Rate Scheduling: Optimizing Training in Deep Networks

What is a Learning Rate Scheduler?

A learning rate scheduler is a method that adjusts the learning rate during the training process, often lowering it as the training progresses. This helps the model to make large updates at the beginning of training when the parameters are far from their optimal values, and smaller

updates later when the parameters are closer to their optimal values, allowing for more fine-tuning.



Below, we discuss two widely used techniques: **Step Decay** and **Cyclical Learning Rate (CLR)**.

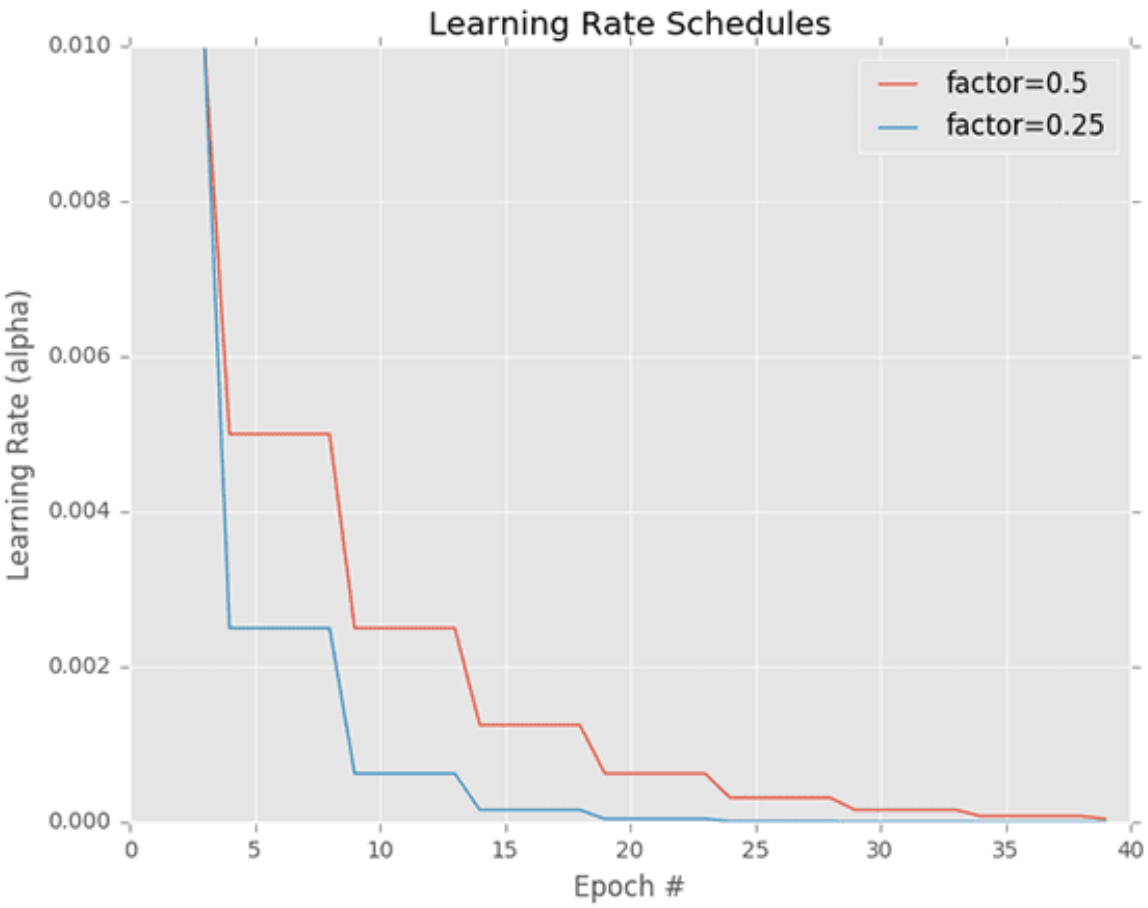
1. Step Decay

The learning rate is reduced by a factor after a specific number of epochs. For example, you might start with a learning rate of 0.1 and reduce it by a factor of 0.5 every 10 epochs. This method is simple yet effective, allowing for initial rapid learning that slows down over time.

$$lr = lr_0 \cdot d^{(\text{floor}(1+\text{epoch})/s)}$$

where:

- lr_0 is the initial learning rate,
- d is the decay rate,
- s is the step size, and
- epoch is the index of the epoch



Reference: A (Very Short) Visual Introduction to Learning Rate Schedulers (With Code)

Pros of Step Decay:

- Simple to implement
- Predictable learning rate reduction
- Works well for many standard training scenarios

Cons:

- Rigid schedule that might not adapt to all model behaviors
- Can be less effective for complex datasets

Benefits:

✅ **Faster convergence** in the early training phase.
✅ Helps the model settle into a more optimal **local minimum**.
✅ **Prevents overshooting** by reducing LR at key stages.

Real-World Application:

🔷 Used in **image classification tasks (CNNs)** where rapid learning is required early, but fine-tuning is needed later.

Implementation (TensorFlow Example):

```
import tensorflow as tf

def step_decay(epoch):
    initial_lr = 0.01
    drop = 0.5
    epochs_drop = 10
    return initial_lr * (drop ** (epoch // epochs_drop))

lr_scheduler = tf.keras.callbacks.LearningRateScheduler(step_decay)
```

Reference: ChatGPT

2. Cyclical learning rate

Introduced by Leslie Smith, cyclical learning rates oscillate between a lower and upper bound, allowing the model to explore different learning rates during training. This approach can help models escape poor local minima and potentially find better solutions.

Similar to adaptive learning rate, the learning rate changes are cyclic, always returning to the learning rate's initial value. A very high learning rate causes the model to fluctuate more or to diverge from the minima, while a lower learning rate can cause the model to converge very slowly or to converge to the local minima. Cyclical learning rate(CLR) allows keeping the learning rate high and low, causing the model not to diverge along with jumping from the local minima.

In CLR learning rate oscillates between base learning rate and max learning rate. The oscillation of learning rate can be based on various function-triangular (linear), Welch window (parabolic), or Hann window (sinusoidal). The triangular window is a simpler way of changing the learning rate.

Reference: Cyclical Learning Rate

$$\eta t = \eta min + (\eta max - \eta min) \times f(t)$$

Benefits:

✅ Allows the optimizer to **escape poor local minima** by increasing LR periodically.

✅ Reduces the need for manually tuning the learning rate.
✅ Works well for **complex loss landscapes** (e.g., NLP, vision models).

Real-World Application:

🔷 Often used in **NLP models (Transformers, LSTMs)** and **transfer learning**, where adjusting the learning rate dynamically improves fine-tuning performance.

Research & Insights

- A study by **Leslie Smith (2017)** introduced CLR and demonstrated that **cyclical learning rates can outperform standard decay methods** (Source).
- Step Decay is widely used in frameworks like **ResNet**, while CLR is popular in **NLP models like BERT**.
- **From personal experience**, Step Decay works well in CNN-based **image classification**, whereas CLR is more effective in **transfer learning tasks**.

> - **Reference:** A (Very Short) Visual Introduction to Learning Rate Schedulers (With Code)

Conclusion

Both **Step Decay** and **Cyclical Learning Rate (CLR)** help optimize neural networks by adjusting the learning rate dynamically:

- **Step Decay** is best for **CNNs and large datasets** where gradual fine-tuning is needed.
- **Cyclical Learning Rate** is ideal for **NLP and deep architectures** where exploring different LR values improves results.

Using the right learning rate scheduler **can significantly boost model performance** and prevent training inefficiencies. 🚀

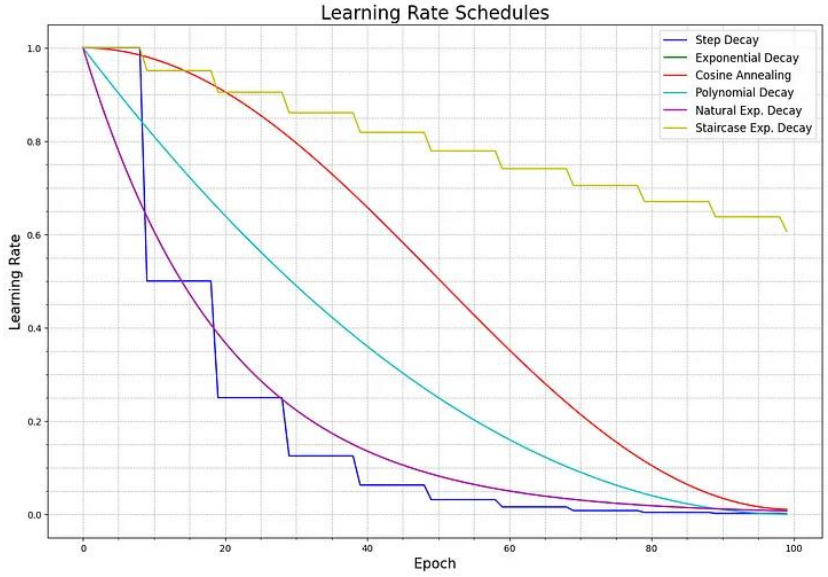
Reference: ChatGPT

Personal Reflection

In one of my earlier projects on image classification, I experimented with both fixed and dynamic learning rates. Switching to a **cyclical learning rate** improved convergence speed and model accuracy noticeably, as the learning rate oscillations helped the model escape local minima.

Bonus

Some more learning rates in one plot.



Question 2: The Role of Computation Graphs in Backpropagation

Computational Graphs

A computational graph is a way to represent a math function in the language of graph theory. Recall the premise of graph theory: nodes are connected by edges, and everything in the graph is either a node or an edge.

In a computational graph node are either input values or functions for combining values. Edges receive their weights as the data flows through the graph. Outbound edges from an input node are weighted with that input value; outbound nodes from a function node are weighted by combining the weights of the inbound edges using the specified function.

Reference: Deep Neural Networks As Computational Graphs

Importance in Backpropagation

Backpropagation leverages computation graphs to compute gradients efficiently:

- **Forward Pass**: Calculates the output of the network by passing inputs through the computation graph.
- **Backward Pass**: Traverses the graph in reverse to compute gradients using the chain rule, allowing for efficient parameter updates.

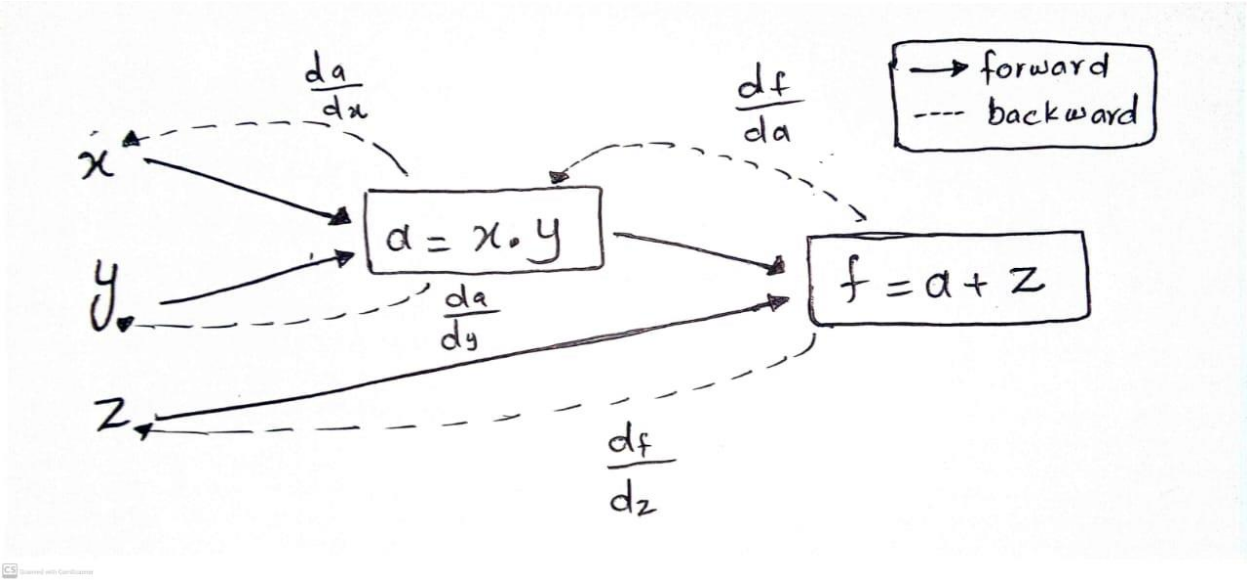
This method ensures that gradients are computed systematically, even for complex networks, by breaking down the operations into manageable parts.

Reference: Calculus on Computational Graphs: Backpropagation

Computing Gradients Using a Computation Graph

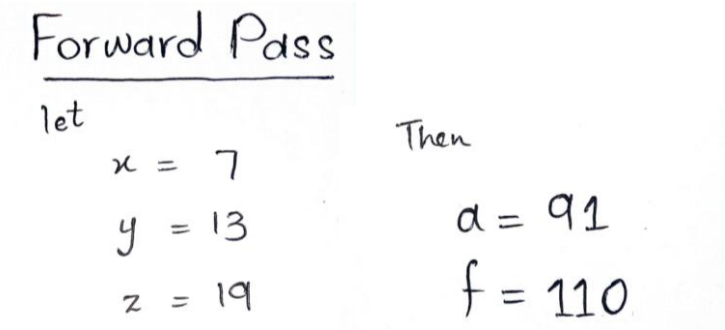
Let's consider the simple function:

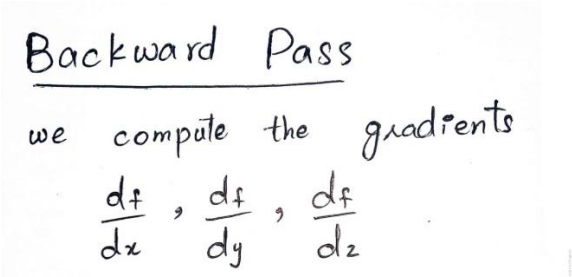
$$f(x, y, z) = x \cdot y + z$$



We can decompose it into intermediate operations:

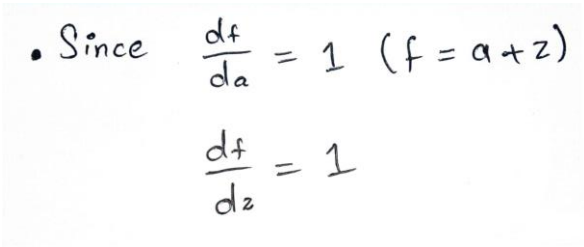
1. a = x \cdot y
2. f = a + z

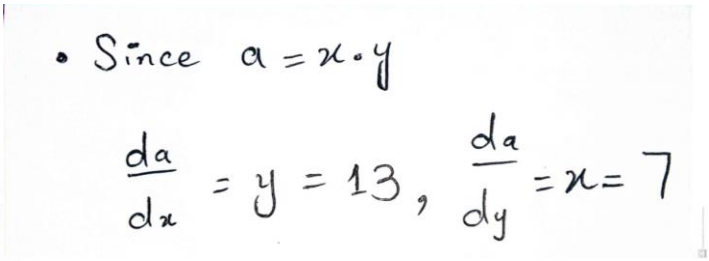


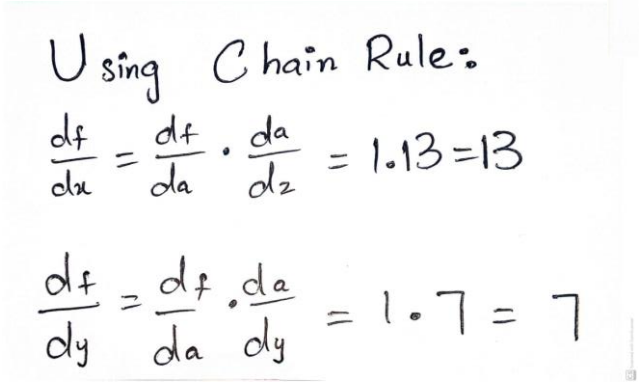


Understanding Partial Derivatives and Chain Rule

Partial derivatives are used to measure how a multivariable function changes when one variable changes, while all other variables are held constant. This concept is essential in neural networks, where we calculate the effect of each weight or input on the final output. The **chain rule**, on the other hand, is a fundamental tool in calculus that helps us compute derivatives of composite functions. It allows us to "chain" together gradients through multiple layers or steps in a computation graph. During backpropagation, we use the chain rule to systematically compute gradients layer-by-layer, starting from the output and moving backward — making it possible to train deep learning models efficiently.







Real-World Applications of Computation Graphs

Computation graphs play a crucial role in many real-world applications involving deep learning and AI. Here are a few examples:

1. Training Deep Neural Networks (DNNs):

In frameworks like TensorFlow and PyTorch, every operation (like matrix multiplication, activation functions, etc.) is added as a node in a computation graph. This allows automatic differentiation to efficiently compute gradients using backpropagation — making it possible to train large models like GPT, BERT, or image classifiers.

2. Natural Language Processing (NLP):

In NLP tasks such as machine translation or sentiment analysis, computation graphs help track operations from embeddings to outputs. They allow dynamic and efficient computation of gradients across sequences of variable length.

3. Computer Vision:

In tasks like object detection or facial recognition, computation graphs map the flow from input images through convolutional layers, ReLU, pooling, and fully connected layers — making it easy to train and update weights during learning.

4. Reinforcement Learning (RL):

In RL models (e.g., used in robotics or game AI), gradients need to be propagated through multiple steps of decisions. Computation graphs help in modeling this sequence of operations and in optimizing the policies.

5. Healthcare & Medical Imaging:

Neural networks powered by computation graphs are used for tumor detection, x-ray classification, and even predictive diagnosis based on patient history.

Reference: CS231n: Deep Learning for Computer Vision

Personal Reflection

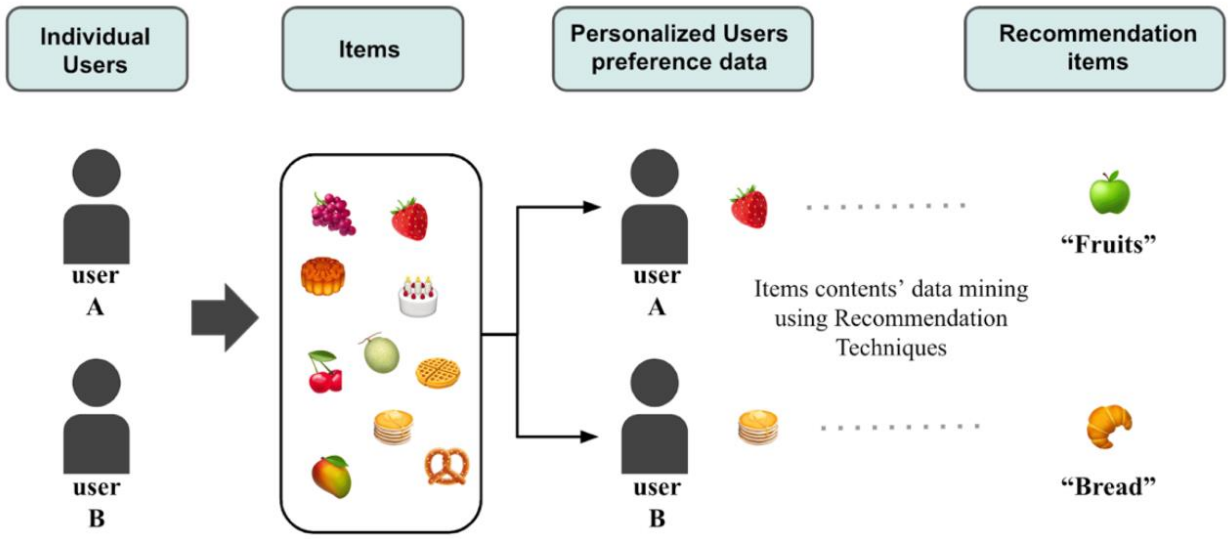
As a student of deep learning, understanding computation graphs has been a game-changer for me. Initially, I saw backpropagation as just a series of formulas, but computation graphs helped me **visualize** and **intuitively grasp** how information flows through a model. Now, I can confidently say that every time I train a model, I can picture the graph being built behind the scenes — from input to output — and gradients flowing backward like a chain reaction.

This concept has not only helped me in assignments but also in **debugging neural networks** and understanding how libraries like PyTorch work internally. I believe this fundamental knowledge will support my journey as I build more complex AI systems, and it's something I'll continue to explore deeply.

Question 3: Applications of Fully Connected Neural Networks

Fully Connected Neural Networks (FCNNs), also known as **dense neural networks**, are fundamental components in deep learning architectures. Their ability to model complex, non-linear relationships makes them suitable for a variety of advanced real-world applications. This discussion focuses on two such applications: medical diagnosis using Electronic Health Records (EHR) and personalized recommendation systems.

Personalized Recommendation Systems



Application Overview

Recommendation systems are integral to various online platforms, including e-commerce, streaming services, and social media. They aim to predict user preferences and suggest items or content that align with individual tastes. FCNNs have been employed to enhance the accuracy and personalization of these systems.

1. **Adaptation of FCNNs to Recommendation Systems**

- **Embedding Layers**: In recommendation systems, both users and items can be represented as embeddings—dense vector representations that capture latent features. FCNNs utilize embedding layers to transform sparse categorical data (e.g., user IDs, item IDs) into continuous vectors, facilitating the learning of user-item interactions.

- **Reference:** Using Neural Networks for Your Recommender System

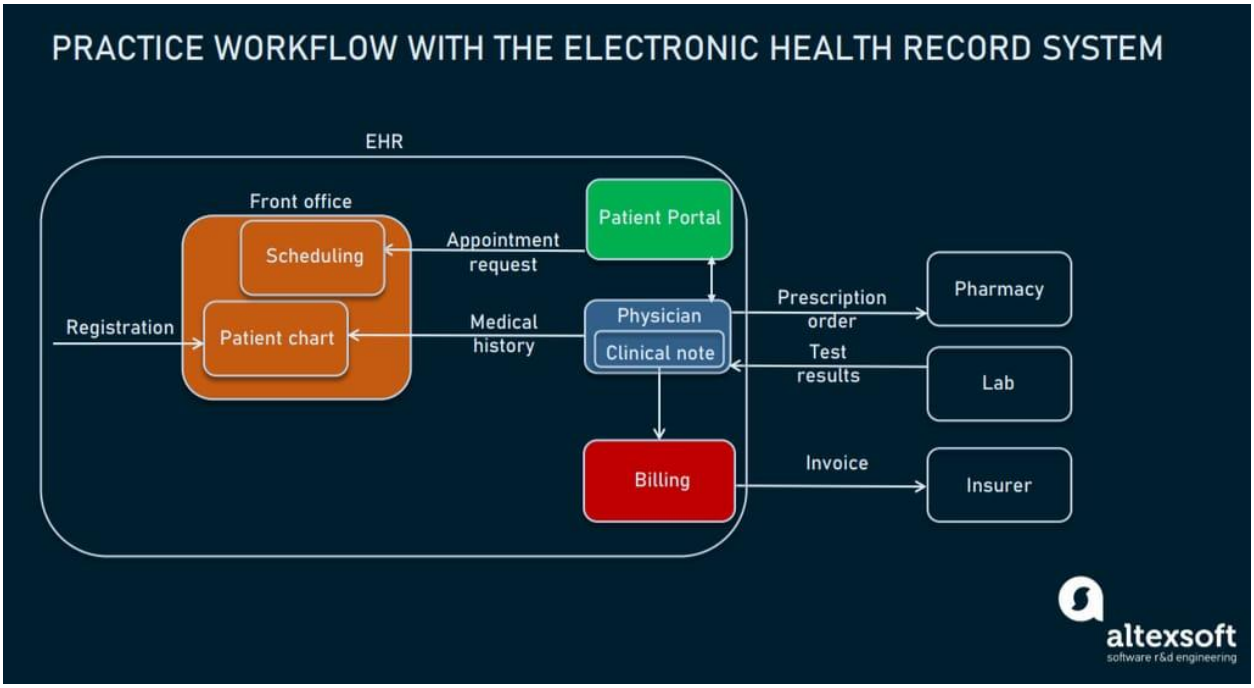
- **Modeling Non-Linear Relationships:** User preferences are often influenced by complex, non-linear relationships among various factors, such as item attributes and contextual information. FCNNs are capable of modeling these non-linearities, enabling more accurate predictions of user preferences.
- **Hybrid Models:** FCNNs can be integrated with other neural network architectures, such as **convolutional neural networks (CNNs)** for image data or **recurrent neural networks (RNNs)** for sequential data, to create hybrid models that leverage multiple data modalities. For example, incorporating product images alongside textual descriptions can enhance recommendation quality. NVIDIA Developer

2. **Suitability of FCNNs for Recommendation Systems**

- **Enhanced Predictive Performance:** The flexibility of FCNNs allows for capturing intricate patterns in user behavior, leading to improved recommendation accuracy compared to traditional collaborative filtering methods.

- **Scalability to Large User Bases:** FCNNs can handle large-scale data, accommodating millions of users and items, which is essential for commercial recommendation systems.
- **Adaptability to Dynamic Data**: FCNNs can be retrained or fine-tuned with new data, allowing recommendation systems to adapt to evolving user preferences and emerging trends.

Medical Diagnosis Using Electronic Health Records (EHR)



- **Application Overview**

  Electronic Health Records (EHR) are comprehensive digital compilations of patient information, including demographics, medical history, laboratory results, and treatment plans. Analyzing this multifaceted data is crucial for early disease detection, prognosis, and personalized treatment strategies. FCNNs have been

effectively employed to process EHR data for predictive modeling and diagnostic support.

1. Adaptation of FCNNs to EHR Data

- **Handling High-Dimensional Data**: EHR datasets are typically high-dimensional, encompassing numerous patient attributes. FCNNs are adept at managing such data due to their fully connected structure, which allows each neuron in one layer to connect to every neuron in the subsequent layer, facilitating the modeling of intricate relationships among variables.
- **Integration of Diverse Data Types**: EHRs contain heterogeneous data types, such as categorical (e.g., gender), continuous (e.g., blood pressure), and temporal data (e.g., medication history). FCNNs can integrate these diverse data types by transforming them into a unified numerical format suitable for input into the network.
- **Feature Learning and Representation**: FCNNs can automatically learn relevant features from raw EHR data without the need for manual feature engineering. This capability is particularly beneficial in uncovering hidden patterns and interactions that may not be apparent to human analysts.

2. Suitability of FCNNs for EHR Analysis

- **Predictive Accuracy:** Studies have demonstrated that FCNNs achieve high predictive accuracy in clinical tasks, such as disease onset prediction and patient outcome forecasting. For instance, research indicates that deep learning models, including FCNNs, outperform traditional statistical methods in predicting clinical diagnoses from EHR data. arXiv
- **Scalability**: FCNNs are scalable to large datasets, making them suitable for healthcare systems with extensive patient records.

- **Personalization:** The ability of FCNNs to model complex interactions enables the development of personalized treatment plans based on individual patient data.

- **Reference:** Predicting Clinical Diagnosis from Patients Electronic Health Records Using BERT-based Neural Networks

Personal Reflection

The application of FCNNs in medical diagnosis and recommendation systems underscores their versatility and efficacy in handling complex, high-dimensional data across diverse domains. In healthcare, FCNNs contribute to more accurate and personalized patient care by effectively analyzing EHR data. In the realm of personalized recommendations, they enhance user experiences by providing tailored content suggestions. These applications highlight the transformative potential of FCNNs in leveraging data-driven insights to inform decision-making and improve outcomes.