AWS MLU Lab Reflection

Lab 01: Getting Started with PyTorch

Summary of Key Learnings

This lab introduced the fundamentals of PyTorch, including tensors, tensor operations, and basic tensor manipulation. I learned how to create tensors, perform mathematical operations, and utilize PyTorch's capabilities for automatic differentiation.

Insights & Understanding

One key takeaway was how PyTorch handles computational graphs dynamically, which allows for efficient optimization in deep learning models. Understanding tensors as the fundamental data structure in PyTorch provided clarity on how neural networks process and store information.

Challenges Encountered

Initially, I found it challenging to grasp the differences between NumPy arrays and PyTorch tensors. However, experimenting with the conversion functions (`torch.from_numpy()`) helped solidify my understanding.

Application & Relevance

The ability to manipulate tensors is crucial in AI and machine learning. This lab provided a foundation for future deep learning tasks, such as training neural networks and optimizing models.

Code and Experimentation

I experimented with modifying tensor sizes and operations. By increasing the tensor dimensions, I observed how PyTorch adjusted memory allocation dynamically, demonstrating its flexibility in handling large datasets.

Lab 02: How Neural Networks Learn

Summary of Key Learnings

This lab explored the training process of neural networks, including forward propagation, loss computation, and backpropagation. I learned how neural networks adjust weights using gradient descent to minimize errors.

Insights & Understanding

A major insight was the role of the loss function in guiding the learning process. Understanding how gradients flow backward through the network reinforced why PyTorch's `autograd` feature is powerful for training models.

Challenges Encountered

One challenge was visualizing the effect of different learning rates. Initially, I didn't notice much change, but after tweaking the rate, I saw how smaller values led to stable but slow learning, while larger values caused instability.

Application & Relevance

This lab is relevant to AI applications such as fraud detection and personalized financial advice. Efficient training ensures models can identify patterns in data and make accurate predictions.

Code and Experimentation

I experimented with different activation functions and noticed how ReLU improved training speed compared to Sigmoid, confirming its effectiveness in deep learning.

Lab 03: First Example of Neural Networks

Summary of Key Learnings

This lab provided a hands-on example of building and training a simple neural network. I learned about defining layers, choosing activation functions, and optimizing models.

Insights & Understanding

The biggest realization was how hyperparameters like batch size and epochs significantly influence training performance. Fine-tuning these values can drastically improve accuracy.

Challenges Encountered

Understanding the impact of batch size was initially confusing, but running multiple training iterations with different values helped clarify its effect on gradient updates.

Application & Relevance

Neural networks are widely used in AI applications, from image recognition to predictive modeling. This lab provided practical knowledge applicable to real-world problems like stock market predictions.

Code and Experimentation

I adjusted the number of hidden layers and neurons, observing how deeper networks learned more complex patterns but also risked overfitting. This highlighted the importance of model architecture tuning.

Conclusion & References

These labs provided a solid foundation in PyTorch and neural networks. The hands-on approach helped bridge theoretical concepts with practical implementation.

References:

1. AWS Machine Learning University. (2025). *Getting Started with PyTorch*. Retrieved from AWS Documentation
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.