L03 Assignment: AWS MLU Lab Reflection

Martin Demel

Department of Science, Technology, Engineering & Math, Houston Community College

ITAI 2376 Deep Learning in Artificial Intelligence

Patricia McManus

February 5th, 2025.

**Introduction**

In Module 1, we had three laboratory works. To be precise, Lab 1 had two notebooks. I primarily focused on understanding neural networks and the PyTorch framework without writing code from scratch. Instead, I studied, modified, and experimented with the provided code as part of the assignment. Even though I have previously worked with data preprocessing and model architecture, these labs pushed me to think more critically about how tensors, neural networks, and data flows come together within PyTorch. By performing the three labs, I have deepened my knowledge of how neural networks learn and how they can be trained and regularized. I especially appreciated Lab 3, which showed how to build an end-to-end solution.

**Body**

**Lab 01 Getting Started with PyTorch**

**In Notebook 1**, I focused on tensors and automatic differentiation. One part that caught my eye was how essential it is for the data shapes to match when performing matrix multiplications. This is something I have explored and faced in my previous assignments, so I paid close attention to it. For example, I tried reshaping the tensor of size (12) into different 2D shapes to observe how that impacted subsequent operations. This helped me further clarify why PyTorch emphasizes the importance of aligned dimensions.

I also discovered that when using *requires\_grad=True*, PyTorch creates an internal computational graph. Tracing through the partial derivatives reminded me of calculus fundamentals but in a practical, code-driven way.

I encountered a small challenge when I experimented with in-place operations on tensors that require gradients. This disrupted the computational graph. This was caused by in-place operations that can be overwritten by variables. PyTorch needs to compute derivatives, to my understanding.

**Notebook 2 of the Lab 1** focused further on implementing a basic neural network. A simple logic regression guided me through a complete training loop, including data ingestion and defining the model, applying the loss function, and running optimization steps.

Something new I observed and learned was the nn.Sequential container with a linear layer (nn.Linear) including sigmoid activation function. Seeing how MLP or logic regression is trained in code gave me a clearer sense of how gradients move through the network layers. I still have to learn a lot, and I need to definitely dive deeper, but I really appreciate it.

Configuring the data was a small hurdle. Ensuring that the input tensor dimension matched the model’s expected input size was critical. If. Fed a 2D data that didn’t match—the batch size and a number of features, the matrix multiplication would fail.

Even though the logistic regression is to some degree self-explanatory, it’s the foundation for more advanced networks like for example, CNNs. It was very clear that as long as I ensure that I understand how to define layers, choose an optimizer, and compute a loss, it would be easy for me to understand how to scale up to more complex architectures.

**Lab 02 Creating Multilayer Perceptron and Using Dropout Layers**

In laboratory work 2, I have learned more about training a multilayer perceptron (MLP) on the fashion MNIST dataset. By reviewing the lab code, I have come across the loss function – cross-entropy loss and using a torch. optimal for stochastic gradient descent. By observing the training lop and validation steps, I saw clearly how neural networks learn through repeated forward passes, loss calculation, backward passes and weight updates.

I really enjoyed the part about trying it yourself. Activity where I was prompted to modify hyperparameters such as epoch count and learning rate. Knowing from the previous course—computer vision, this can be very tricky to fine-tune. By changing the epochs and learning rate, I have observed how the training can improve but also can be ultimately affected by overfitting—meaning that the validation accuracy did not improve. I could see how they positively impacted the model generalization by testing the different dropout rates, such as 0.2 and 0.3. Very important, the overfitting or underfitting I could notice. When performing the test with a learning rate and changing it to 0.01 from 0.1, I could observe how important it is to carefully select the right parameters and finetune them.

In addition to the suggestion in the lab, I swapped the optimizer from SGD to Adam to see if it would converge faster. Interestingly enough, with Adam, I need to adjust another parameter, as I have learned by reading the explanations from the geek for geek and ChatGPT explanation. As explored, the selection of the right optimizer can play a pivotal role in training dynamics.

To conclude, in Lab 2, I had the opportunity to work with it myself and experiment with different model configurations. That is something that is very important also in real-life application, and it helped me to refresh my practical experience from the fall semester. The approach of changing the hyperparameter at a time, observing the effect on training/validation losses, and interacting accordingly is crucial for tasks ranging from image recognition to tabular classification, as I have also learned thanks to ChatGPT.

**Lab 03 Building an End-to-End Neural Network Solution**

By performing lab 3, I observed an end-to-end workflow of the Neural Network by using text data from the Austin Animal Center dataset. In comparison, this lab integrated numeric, categorical, and text data into a single pipeline. I reviewed the code for data splitting, feature engineering, and building a multilayer neural network to predict adoption outcomes.

Again, the try-it-yourself part was very engaging and got a little more interesting as I experimented with the text vectorizer parameters, such as max\_features in countVectorizer.I quickly realized how changing feature dimension can significantly affect it if I set max\_features to a high value. Even though it can help capture more nuanced text signals, the neural network can overfit or become very large. Another test I performed was to change the dropout layer positions and rates in the neural network. I tested placing dropout layers after the hidden layer vs. only after the first hidden layer. Seeing it work in real-time and how the changes in validation accuracy gave me a sense of how dropout mitigates overfitting, particularly when dealing with high-dimensional text inputs. It was very interesting that when I tried it a second time, I was unable to achieve the same exact output again.

When I tried adjusting the text preprocessing steps for the try-it-yourself activity, I faced some challenges. Removing certain stop words or applying more aggressive stemming sometimes resulted in improved accuracy but also risked losing domain-specific words crucial for predicting adoption status. It was very important to ensure that balancing these transformations requires a very thought and balanced approach and rechecking the lab instructions as I got lost at one point and had to revert to the original version. Ultimately, I have ensured consistency and ensured that transformations were applied for both training and validation data.

Being able to combine text features with numeric and categorical data in a single PyTorch model is a highly practical skill. Many industry datasets include textual descriptions like product reviews, notes, or simply text or logs, along with structured attributes. Lab 3 taught me a lesson on how to build a neural network and ensure to have the right approach. By testing different pipeline configurations and neural network architectures, I can approach similar tasks in the future with greater confidence.

One last thing I have experimented with is changing the hidden layer size from 64 to 128. This gave me a higher capacity network, but the validation loss started to fluctuate. I tried different tokenization approaches and also incorporated a dropout rate, as I have learned from ChatGPT that it could help or benefit the neural network. Each experiment reinforced my knowledge, and I could see how small pipeline changes can lead to significant shifts in the model performance.

**Conclusion**

Overall, I really enjoyed this laboratory work, and it was very engaging. It has refreshed my memories from the fall 2024 computer vision class, and I have reinforced my learning by trying some of the features that I already knew. I liked the try it yourself, and it also encouraged me to change the code beyond just that. Playing with the code, changing the parameters, and trying error tests deepened my knowledge, reinforced some of the knowledge I already had, and helped me better understand the mechanism.

I now feel more confident in applying these techniques in future labs, and whether for image recognition or text classification, I can definitely see all the benefits that neural networks can offer as a powerful solution.

**Resources:**

GeeksforGeeks. (2024, September 10). Deep Neural net with forward and back propagation from scratch Python. GeeksforGeeks. <https://www.geeksforgeeks.org/deep-neural-net-with-forward-and-back-propagation-from-scratch-python/>

Pipeline. (2025). Scikit-learn. <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>

LogisticRegression. (2024). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html

GeeksforGeeks. (2024, August 7). Artificial Neural Networks and its Applications. GeeksforGeeks. <https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/>

GeeksforGeeks. (2021, October 16). *Understanding MultiLayer Feed Forward networks*. GeeksforGeeks. <https://www.geeksforgeeks.org/understanding-multi-layer-feed-forward-networks/>