**COSE474 Deep Learning Project #1:**

**MLP Implementation**

**1. Statement, Objective**

Recently, artificial intelligence has made a lot of progress. numerous models and methodologies are improving the perf ormance.However, in order to implement and understand the deep learning model well, it is necessary to implement and understand MLP, which is the foundation of Deep Learning. As a task for this purpose, this MLP Implementation task does not simply import a model from a library or github, but implements a Multi Layer Perceptron directly to understand the structure and essence.

**2. Model Architecture**

The MLP we want to implement is a 2-Layer Neural Network (1-Hidden Layer). There are weights W1 and W2, and biases b1 and b2. The structure to be created is Multi-Class Classification and has a total of 3 Labels. The input\_size is 4, the hidden\_size is 10, and the ouput\_size is 3. In the middle, the ReLU passes through the activation layer. Finally, the final probability value is calculated while passing through the softmax function.

For the loss function, Softmax + Log Likelihood Loss and L2 Regularization were used. L2 regularization penalizes complex models to make them simple. This can prevent overfitting.

**3. Dataset**

The dataset used CIFAR-10. CIFAR-10 is a popular collection of images used in machine learning and computer vision algorithms. it has a size of 32 \* 32. Since the image is color, the number of channels is 3.

**4. Experiments**

The Main experiments that I had to implement are Forward, Loss, Backprop agation, Predict, and Hyperparameter Tuning.

Forward is relatively simple, y(z1) = W1\*x + b1 -> Activation Layer(ReLU) -> y(z2) = W1\*x + b2 -> Softmax through the work. Through this, when the y value does not exist, the difference between the Score and the y value was calculated.

Next, the loss could be obtained by Softmax\_loss + L2 Regularization. Softmax can be obtained by calculating the Log Likelihood and then summing it. L2 regularization uses square(^2) differently from L1.

It was to find the gradient value through backpropagation, but it was the most diffi cult task. I have used an existing deep learning model before, and it was very Easy to implement it directly like this because it is usually calculated using the backward() function based on pytorch. I proceeded with Transpose's Dot Product while rotating the layers from the differential output.

Training was given through the values ​​obtained in this way. First, as a test, the train was conducted with 1e-3, decay=0.95, reg=5e-6, and iteration=100, but the desired results were not obtained.

To find the best parameter, with learnin g\_rates = [1e-2, 5e-3, 1e-3], learning \_rate\_decay=[0.75, 0.85, 0.9, 0.95], reg = [0.4, 0.5, 0.6] as variables, the best was obtained.

Accuracy of Validation: 0.445

Train History:

[0.185, 0.385, 0.36, 0.445, 0.45, 0.51, 0.545, 0.525, 0.58, 0.595, 0.545]

Validation History:

[0.141, 0.26, 0.355, 0.396, 0.402, 0.408, 0.433, 0.422, 0.431, 0.445, 0.44]

Hyperparameter at Best Score:

(0.001, 0.95, 0.6)

**5. Conclusion**

Through this, the Test Accuracy obtained 0.461. Changing the hyperparameter greatly affects the accuracy, and I was able to actually feel the No Free Lunch Theory. It was a very meaningful experiment that allowed me to study the structure and transfer method of the model once again while implementing the model myself.