Project 1 – Two-layer MLP Implementation

2021320117 Bae Minseong

In this project, we have to implement two-layer MLP with SoftMax classifier. First, in the neural_net.py, we have to implement the forward/backward propagation and calculating the loss. It is pretty simple to implement forward propagation, just using some numpy functions like np.dot, np.sum, np.exp and np.tile to implement SoftMax function. The key point is that we have to store the value that SoftMax is not applied to the score variable. After the forward propagation, we have to calculate the data loss and regularization loss and sum them, we already know that the loss for SoftMax classifier is log likelihood, as we learned in the lecture. So, I also used some numpy functions to implement loss calculation like np.log or np.float_power for L2 loss (we have to square each element in weight matrices). Finally, to calculate the gradients in backpropagation, we have to calculate the one-hot version vector of y. Then, we can implement the backpropagation code using the derivation of gradients in the propagation stream similarly we learned in the class. The difference is now we are calculating in multiple instances, so we have to mean the gradient of each data, and the activation function is changed to ReLU, which has original upstream value if it is positive or zero value of gradients because of its definition. And also, we have to add the gradient of regularization loss in the gradients of W1 and W2. In the training implementation, if the size of the batch is smaller than the size of training set, then we do not have to do random sampling, else we randomly choose the indices of training data and use it in SGD. Then, updating gradient is so simple and after the training, we can predict the value of new inputs using updated parameters just using similar code to the forward propagation, just the difference is you have to apply the SoftMax and find the most probable class with np.argmax. In the hyperparameter tuning, I decide to adjust three hyperparameters, size of hidden layer, learning rate and regularization constant.

Using the hyperparameters with the best validation accuracy, we can classify the test dataset in the accuracy of 47.5% just using simple MLP. In the hyperparameter tuning process, we can find out too small learning rate can make a worse result in the training process, maybe because the process is too slow so it cannot reach to the global minimum, and also, we have to wait too much time to complete the training. But we can also just find out there are no unconditional tendency in validation accuracy with hyperparameters, it depends on the dataset and the combination with the other hyperparameters. So, it is very important to do the hyperparameter tuning because like in this project, we can easily and greatly improve the accuracy of the learning without a new idea for model implementation or constructing new features.