BE201 – HW2 Report

202311131 이상민

** Please download tqdm and matplotlib to run the code!! **

1. Explanation of the code

The code for the model is made up of 5 classes : model, convolution, max-pooling, affine, and CNN. The model class manages the layers of the CNN model – it can add layers, calculate the cross-entropy loss and accuracy, perform forward and backward calculations, and train the model. The classes convolution performs the calculation for the convolutional layer, which includes the activation function with it. The max-pooling class performs max pooling, and the affine class performs calculations for the fully connected layer. The CNN class provides im2col and col2im functions for the convolution and max-pooling class, which inherits from CNN. Here is the pseudocode for the design of the model:

```
class CNN:
    def im2col(self, X)
    def col2im(self, col, N)

class Convolution(CNN):
    def set_parameters(self, input_shape)
    def relu(Z)
    def forward(self, X)
    def backward(self, grad, lr)

class MaxPooling(CNN):
    def __init__(self, filter_shape, stride)
    def set_parameters(self, input_shape)
    def forward(self, X)
    def backward(self, grad, lr)

class Affine:
    def set_parameters(self, input_shape)
    def softmax(self, X)
    def forward(self, X)
    def backward(self, grad, lr)

class Model
    def add_layers(self, *models)
    def set_parameters(self)
    def cross_entropy_loss(pred, ans)
    def forward(self, X)
    def backward(self, Y_pred, Y_ans)
    def predict(self, X)
    def score(self, X, Y)
    def fit(self, X_train, y_train, batch_size, learning_rate)
```

by using these classes, a CNN model can be made and be trained like this:

```
def train(epochs, batch_size, learning_rate, early_stopping=False):
    X_train, X_test, y_train, y_test = load_data("./dataset")
    model = Model()
    model.add_layers(layers)
    model.set_parameters()
    for epoch in range(1, epochs+1):
        loss = model.fit(X_train, y_train, batch_size, learning_rate)

    with open('../ckpt/ckpt.pkl', 'wb') as f:
        pickle.dump(model, f)

if __name__ == "__main__":
    train(epochs, batch_size, learning_rate)
```

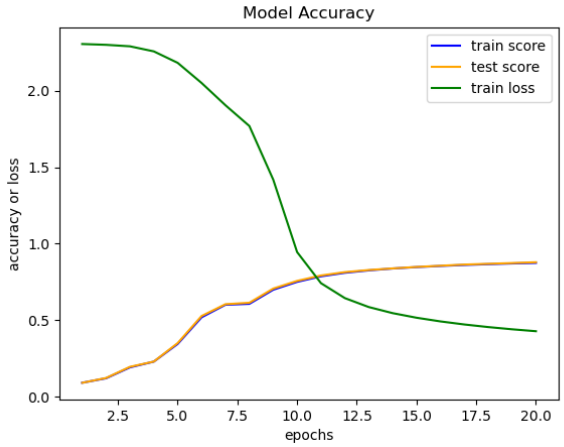
When layers are added to the model by using add_layers(), then the function set_parameters() initialize parameters for each layers. It will initialize the input shape, output shape, and weights.

When the model performs forward(), it goes through each layer sequently. The result of the forward() of each layer is passed to the next layer. similarly backward() also passes the gradient of each layer to the previous layer. In each layer, it will update its weights according to the learning rate and the gradient. In the convolution and pooling layers, it will use im2col or col2im respectively for forward and backward processes. It transforms the input matrix into another form which allows matrix products. By using the transformations, it makes the entire process faster since matrix products are very fast and efficient when using numpy.

2. Experimental Results

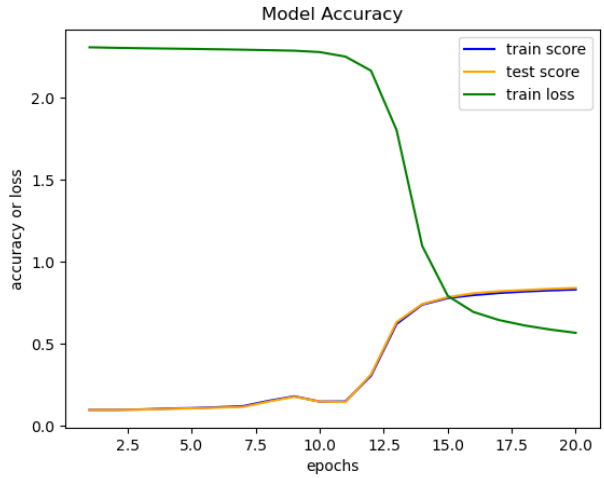
(1) Using 3 building blocks

- Layer settings
  Convolution(filter=(5, 1, 3, 3), padding=1, stride=1) -> relu
  -> MaxPooling((2, 2), stride=2)
  -> Convolution(filter=(1, 5, 3, 3), padding=1, stride=1) -> relu
  -> MaxPooling((2, 2), stride=2) -> Affine()
- Train settings : epochs 20, batch size 64, learning rate 0.002
- Train Result : train accuracy 0.8748, test accuracy 0.8793



(2) Adding a convolution layer with filter = (FH=1, FW=1)

- Layer settings
  Convolution(filter=(3, 1, 3, 3), padding=1, stride=1) -> relu
  -> MaxPooling((2, 2), stride=2)
  -> Convolution(filter=(5, 3, 3, 3), padding=1, stride=1) -> relu
  -> Convolution(filter=(1, 5, 1, 1)
  -> MaxPooling((2, 2), stride=2) -> Affine()
- Train settings : epochs 20, batch size 64, learning rate 0.002
- Result : train accuracy 0.8017, test accuracy 0.8361



(3) Using learning rate decay for faster training

By decreasing the learning rate as epochs increase, we can optimize the model faster. As the learning rates will be large in the beginning and small afterwards, we can approach near to the optimal weights faster in the beginning then carefully optimize the model using small learning rates.

- Layer settings
  Convolution(filter=(5, 1, 3, 3), padding=1, stride=1) -> relu
  -> MaxPooling((2, 2), stride=2)
  -> Convolution(filter=(1, 5, 3, 3), padding=1, stride=1) -> relu
  -> MaxPooling((2, 2), stride=2) -> Affine()
- Train settings : epochs 10, batch size 64, learning rate 1/(epoch^0.75)
- Result : train accuracy 0.8549, test accuracy 0.8575