COSE474 Deep Learning

Project #1: MLP Implementation

2017320122 김정규

Code1 – Forward Pass

```
a = np.dot(X, W1) + b1
b = np.maximum(a, 0)
scores = np.dot(b, W2) + b2
```

The code above is the part that implements the forward pass. The score of each class is calculated for input data. The result is stored in the score variable.

Code2 – Calculating Loss

```
softmax = np.exp(scores)
softmax /= np.sum(softmax, axis=1).reshape(N, 1)

loss = np.sum(-np.log(softmax[np.arange(N), y]))
loss /= N
loss += reg * (np.sum(W2 * W2) + np.sum( W1 * W1 ))
```

This is the part where the forward pass is completed and the loss is finally calculated. That means the data loss is calculated, and L2 regularization is calculated for W1 and W2. Softmax is used for loss function.

Code3 – Back Propagation

```
out = np.copy(softmax)
out[np.arange(N), y] -= 1
aa = np.dot(out, W2.T) * (a > 0)
bb = np.dot(out, W2.T)

grads['W1'] = np.dot(X.T, aa) / N
grads['b1'] = np.sum(aa, axis=0) / N
grads['W2'] = np.dot(b.T, out) / N
grads['b2'] = np.sum(out, axis=0) / N

grads['W1'] += reg * W1
grads['W2'] += reg * W2
```

This is the part that implements backpropagation. The learning rate is subtracted from each weight value.

Code4 – Random Indices

```
Randomindices = np.random.choice(num_train, batch_size)
X_batch = X[Randomindices]
y_batch = y[Randomindices]
```

The batch_size number of valid integers are extracted randomly from 0 to num_train. By using these integers as indices, put these data in X_batch and y_batch respectively.

Code5 - Update

```
self.params['W1'] -= learning_rate * grads['W1']
self.params['b1'] -= learning_rate * grads['b1']
self.params['W2'] -= learning_rate * grads['W2']
self.params['b2'] -= learning_rate * grads['b2']
```

Multiply the grades calculated from the loss function by the learning_rate and subtract them from weight and bias, which means to update the value.

Code 6 – Prediction

```
y_pred = np.argmax(self.loss(X), axis=1)
```

This is a code for testing using learned parameters. we can get the value which was predicted to be the largest among those other elements using argmax function

Code 7 - Tuning

```
best_val = -1
best_stats = []
input_size = 32 * 32 * 3
num_classes = 10
np.random.seed(0)

def Randomsearch(hs_value, lr_value, reg_value):
    hid = hs_value[np.random.randint(0,len(hs_value))]
    lr = lr_value[np.random.randint(0,len(lr_value))]
    reg = reg_value[np.random.randint(0,len(reg_value))]
    return hid, lr, reg

for i in range(20):
    hidden_size, lr, reg = Randomsearch([100, 300, 500], [0.001, 0.0001, 0.00001], [0.05, 0.15, 0.25])
    net = TwoLayerNet(input_size, hidden_size, num_classes)

    stats = net.train(X_train, y_train, X_val, y_val, num_iters=1500, batch_size=300,
                      learning_rate=lr, learning_rate_decay=0.9, reg=reg, verbose=False)
    val_acc = (net.predict(X_val) == y_val).mean()

    if best_val < val_acc:
        best_val = val_acc
        best_net = net
        best_stats = stats

    print('Validation accuracy for hidden size %d, lr %e and reg %e: %f' % (hidden_size, lr, reg, val_acc))

print('best validation accuracy: %f' % best_val)
```

Based on the functions completed, set the environments and parameters to be applied . Hidden_size, lr, and reg were set to follow the results in the skeleton file, but they also follow the random order. Additionally, the accuracies obtained in each case are compared with each other. Then the largest value is stored and printed.

Results

```
Validation accuracy for hidden size 100, lr 1.000000e-04 and reg 5.000000e-02: 0.347000
Validation accuracy for hidden size 500, lr 1.000000e-04 and reg 1.500000e-01: 0.361000
Validation accuracy for hidden size 300, lr 1.000000e-03 and reg 2.500000e-01: 0.486000
Validation accuracy for hidden size 300, lr 1.000000e-04 and reg 2.500000e-01: 0.358000
Validation accuracy for hidden size 300, lr 1.000000e-04 and reg 1.500000e-01: 0.362000
Validation accuracy for hidden size 300, lr 1.000000e-04 and reg 5.000000e-02: 0.366000
Validation accuracy for hidden size 500, lr 1.000000e-05 and reg 2.500000e-01: 0.175000
Validation accuracy for hidden size 500, lr 1.000000e-05 and reg 2.500000e-01: 0.165000
Validation accuracy for hidden size 300, lr 1.000000e-03 and reg 1.500000e-01: 0.491000
Validation accuracy for hidden size 500, lr 1.000000e-05 and reg 2.500000e-01: 0.172000
Validation accuracy for hidden size 100, lr 1.000000e-04 and reg 5.000000e-02: 0.354000
Validation accuracy for hidden size 500, lr 1.000000e-05 and reg 1.500000e-01: 0.162000
Validation accuracy for hidden size 300, lr 1.000000e-03 and reg 2.500000e-01: 0.463000
Validation accuracy for hidden size 500, lr 1.000000e-05 and reg 5.000000e-02: 0.172000
Validation accuracy for hidden size 500, lr 1.000000e-03 and reg 1.500000e-01: 0.462000
Validation accuracy for hidden size 100, lr 1.000000e-03 and reg 2.500000e-01: 0.455000
Validation accuracy for hidden size 500, lr 1.000000e-03 and reg 2.500000e-01: 0.468000
Validation accuracy for hidden size 100, lr 1.000000e-05 and reg 1.500000e-01: 0.156000
Validation accuracy for hidden size 100, lr 1.000000e-03 and reg 5.000000e-02: 0.458000
Validation accuracy for hidden size 100, lr 1.000000e-03 and reg 5.000000e-02: 0.439000
best validation accuracy: 0.491000
```

This the data obtained by ruunning two_layer_net.ipynb. Accuracies obtained through various parameters can be identified. Among them, the best value is 0.491, and the parameters are [hidden size 300], [lr 0.001] and [reg 0.15]