exam 개인학습

Lec 1.4

AI summary Try on this page

Type something...

01) Two-Layer Neural Network

01_affine_forward(x, w, b):

```
out = None
###########################################################################
# TODO: Implement the affine forward pass. Store the result in out. You   #
# will need to reshape the input into rows.                               #
###########################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

x = x.reshape(x.shape[0], -1)
out = x.dot(w) + b
```

- `wx +b` 를 구현하는거기 때문에 매우 간단하게 구현할 수 있다.
- 여기서 `x size`를 `w`와 맞춰주기 위해 `reshape`를 진행하였다.

02_affine_backward(dout, cache):

- `dout * gradient` 를 이용해서 계산해주었다.
- 각각의 사이즈를 잘 보고 계산해주어야 한다.

03_relu_forward(x):

```
def relu_forward(x):
    """
    Computes the forward pass for a layer of rectified linear units (ReLUs).

    Input:
    - x: Inputs, of any shape

    Returns a tuple of:
    - out: Output, of the same shape as x
    - cache: x
    """
    out = None
    ###########################################################################
    # TODO: Implement the ReLU forward pass.                                  #
    ###########################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    out = np.maximum(0, x)
```

- `Relu = max(0, x)` 를 이용하였다. `maximum` 을 사용해서 배열로 반환해주었다.

04_relu_backward(dout, cache):

- `ReLU` 를 이용해서 0보다 작으면 0, 0보다 크면 1을 반환해주자.

1. **Inline Question 1**

Inline Question 1:

We've only asked you to implement ReLU, but there are a number of different activation functions that one could use in neural networks, each with its pros and cons. In particular, an issue commonly seen with activation functions is getting zero (or close to zero) gradient flow during backpropagation. Which of the following activation functions have this problem? If you consider these functions in the one dimensional case, what types of input would lead to this behaviour?

1. Sigmoid
2. ReLU
3. Leaky ReLU

A: ReLu를 사용하다보면 기울기 소실문제가 발생한다. 또한 sigmoid함수도 0에서 멀어질수록 기울기가 0에 가까워져 소실된다. 하지만 Leaky ReLU는 기울기가 0이 되지는 않아 기울기 소실문제는 발생하지 않는다.

05_svm_loss(x, y):

- score에 대해서 미분하기 때문에 loss > 0 이면 1을 주고 1을 준 횟수만큼 `yi` 에 빼줘야 함.

06_softmax_loss(x, y):

```python
def softmax_loss(x, y):
    """
    Computes the loss and gradient for softmax classification.

    Inputs:
    - x: Input data, of shape (N, C) where x[i, j] is the score for the jth
      class for the ith input.
    - y: Vector of labels, of shape (N,) where y[i] is the label for x[i] and
      0 <= y[i] < C

    Returns a tuple of:
    - loss: Scalar giving the loss
    - dx: Gradient of the loss with respect to x
    """
    loss, dx = None, None
    num_train = x.shape[0]
    x = np.exp(x)
    correct_scores = x[range(num_train), y]
    loss = correct_scores / np.sum(x, axis = 1) # (50, )
    loss = np.sum(-np.log(loss)) / num_train

    dx = x / np.sum(x, axis = 1).reshape(- 1, 1) # (50, 1)
    dx[range(num_train), y] -= 1
    dx /= num_train

    return loss, dx
```

- 기본적인 것은 다 같지만 score에 대해 미분하기 때문에 label과 같은 부분은 **-1을 해주어야 한다.**

07___init__(input_dim, hidden_dim, num_classes, weight_scale, reg):

```python
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    W1 = np.random.normal(0, weight_scale, (input_dim, hidden_dim))
    W2 = np.random.normal(0, weight_scale, (hidden_dim, num_classes))
    b1 = 0
    b2 = 0
    self.params['W1'] = W1
    self.params['W2'] = W2
    self.params['b1'] = b1
    self.params['b2'] = b2
    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

- 정규분포 `normal` 을 이용해서 `W1` 과 `W2` 를 초기화 해준다.
- `params` 에 딕셔너리 형태로 저장한다.

08_loss(self, X, y=None):

- 먼저 forward를 통해 scores를 구해준다.

```
loss, dloss = softmax_loss(scores, y)
loss += 0.5 * self.reg * (np.sum(self.params['W1'] ** 2) + np.sum(self.params['W2'] ** 2))

dout3, dW2, db2 = affine_backward(dloss, cache3)
dout2 = relu_backward(dout3, cache2)
dout1, dW1, db1 = affine_backward(dout2, cache1)

dW1 += self.reg * self.params['W1']
dW2 += self.reg * self.params['W2']

grads['W1'] = dW1
grads['b1'] = db1
grads['W2'] = dW2
grads['b2'] = db2
```

- softmax를 이용하여 loss를 구하고 backpropagation을 진행한다.
- `weight` 와 `loss` 에 각각 regularization을 더해주었다.

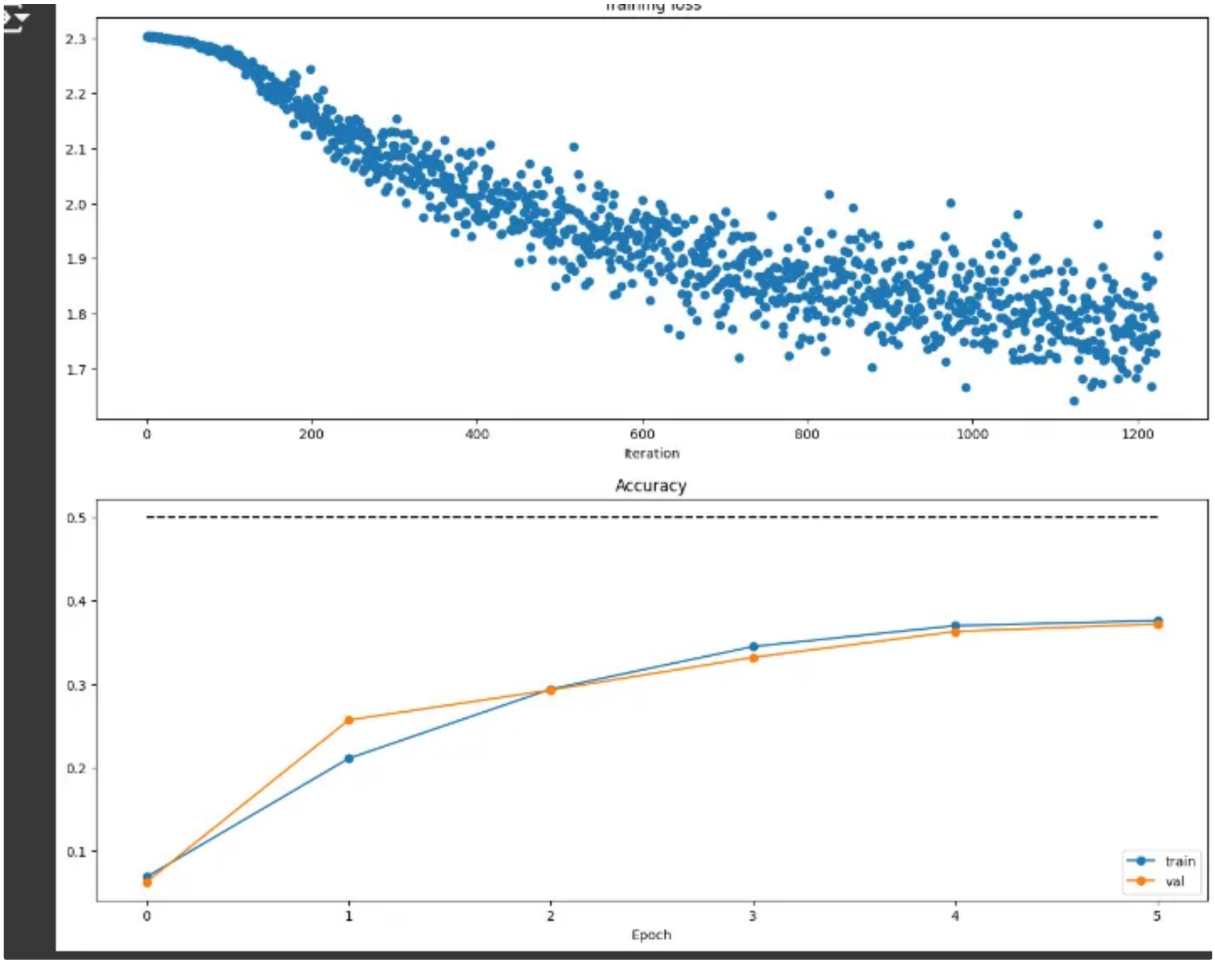
09_Solver:

```
solver = Solver(model, data, update_rule = 'sgd',
                optim_config = {'learning_rate':1e-4}, lr_decay = 0.95,
                num_epochs = 5, batch_size = 200, print_every = 300)
solver.train()

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                              END OF YOUR CODE                                #
################################################################################
```

```
(Iteration 1 / 1225) loss: 2.303922
(Epoch 0 / 5) train acc: 0.069000; val_acc: 0.063000
(Iteration 101 / 1225) loss: 2.280164
(Iteration 201 / 1225) loss: 2.176490
(Epoch 1 / 5) train acc: 0.211000; val_acc: 0.257000
(Iteration 301 / 1225) loss: 2.104870
(Iteration 401 / 1225) loss: 1.997602
(Epoch 2 / 5) train acc: 0.294000; val_acc: 0.293000
(Iteration 501 / 1225) loss: 1.929971
(Iteration 601 / 1225) loss: 2.006990
(Iteration 701 / 1225) loss: 1.811825
(Epoch 3 / 5) train acc: 0.345000; val_acc: 0.332000
(Iteration 801 / 1225) loss: 1.792892
(Iteration 901 / 1225) loss: 1.808718
(Epoch 4 / 5) train acc: 0.370000; val_acc: 0.363000
(Iteration 1001 / 1225) loss: 1.892665
(Iteration 1101 / 1225) loss: 1.767647
(Iteration 1201 / 1225) loss: 1.700919
(Epoch 5 / 5) train acc: 0.376000; val_acc: 0.372000
```

- Solver를 이용하여 epoch마다 loss와 accuracy를 출력해주었다.
- `lr_decay` 는 학습을 반복할 때마다 `learning_rate` 을 줄여서 학습을 해준다.



- 정확도가 높아지고 loss가 낮아지는 것을 확인할 수 있다.

10_Cross_validation

```
learning_rates = [1e-3, 1e-4]
regularization_strenghts = [0, 1e1]
hidden_dims= [64, 128]
epochs = [10]
for lr in learning_rates:
  for reg in regularization_strenghts:
    for hd in hidden_dims:
      for epoch in epochs:
        model = TwoLayerNet(hidden_dim = hd, reg = reg)
        solver = Solver(model, data, optim_config = {'learning_rate': lr},
                        num_epochs = epoch, verbose = False)
        solver.train()
        accuracy = solver.check_accuracy(data['X_val'], data['y_val'])
        if accuracy > best_accuracy:
          best_accuracy = accuracy
          best_model = model
        results[(lr, reg, hd, epoch)] = accuracy


for lr, reg, hd, epoch in sorted(results):
   print('learning rates:',lr, 'regularization:',reg, 'hidden_dims:',hd, 'epochs:',epoch,'accuracy:

print('best accuracy is:', best_accuracy)
# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
################################################################################
#                              END OF YOUR CODE                                #
################################################################################
```

- learing_rate, regularization, hidden_dim, epoch등을 변경시켜가며 가장 좋은 모델을 찾아보았다.

- 앞서서 했던 교차검증 모델 코드와 똑같다.

```
learning rates: 0.0001 regularization: 0 hidden_dims: 64 epochs: 10 accuracy: 0.464
learning rates: 0.0001 regularization: 0 hidden_dims: 128 epochs: 10 accuracy: 0.48
learning rates: 0.0001 regularization: 10.0 hidden_dims: 64 epochs: 10 accuracy: 0.406
learning rates: 0.0001 regularization: 10.0 hidden_dims: 128 epochs: 10 accuracy: 0.411
learning rates: 0.001 regularization: 0 hidden_dims: 64 epochs: 10 accuracy: 0.48
learning rates: 0.001 regularization: 0 hidden_dims: 128 epochs: 10 accuracy: 0.521
learning rates: 0.001 regularization: 10.0 hidden_dims: 64 epochs: 10 accuracy: 0.399
learning rates: 0.001 regularization: 10.0 hidden_dims: 128 epochs: 10 accuracy: 0.405
best accuracy is: 0.521
```

- `hidden_dim` 이 깊을수록 모델의 성능이 좋아지는 것을 확인할 수 있다.
- 그렇다고 해서 무조건 깊으면 `underfitting` 에 빠질 수 있기 때문에 깊이를 잘 조절해야 한다.

2. Inline Question 2

Inline Question 2:

Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

A: training accuracy와 testing accuracy가 크게 차이나는 문제는 overfitting의 문제이다. 데이터 셋을 큰 것을 사용하면 그 일부로 학습을 진행하기 때문에 overfitting이 덜어진다. 또한 regularization 강도를 높이면 overfitting을 해결할 수 있다. 하지만 hidden unit과는 관련이 없다. 따라서 답은 1, 3번이다.