exam 개인학습

Lec 1.3

Type something...

01) Implement a Softmax classifier

01_softmax_loss_navie(W, X, y, reg)

```
num_train = X.shape[0]
num_classes = W.shape[1]
scores = X.dot(W)
for i in range(num_train):
  f = scores[i] - np.max(scores[i]) # avoid numerical instability
  softmax = np.exp(f)/np.sum(np.exp(f))
  loss += -np.log(softmax[y[i]])
  for j in range(num_classes):
    dW[:, j] += X[i] * softmax[j]
  dW[:, y[i]] -= X[i]


loss /= num_train
dW /= num_train

loss += reg * np.sum(W * W)
dW += 2 * reg * W

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

return loss, dW
```

- numerical instability를 피하기 위해 최대값을 먼저 빼주었다.
- 아래의 수식을 이용하여 gradient를 구해주었다.

$$\frac{\partial L_i}{\partial W_j} = x_i * prob(j)(i \neq label)$$

$$\frac{\partial L_i}{\partial W_j} = -x_i + x_i * prob(j)(i = label)$$

1. Inline Question 1

Inline Question 1

Why do we expect our loss to be close to -log(0.1)? Explain briefly.**

A: 처음에 행렬을 거의 0으로 초기화해주기 때문에 모든 score들의 값이 동등하게 나올 것이다. 따라서 확률 값이 1/10이 나오게 되어 loss가 -log(0.1)에 가까워질 것이다.

02_softmax_loss_vectorized(W, X, y, reg)

```python
def softmax_loss_vectorized(W, X, y, reg):
    """
    Softmax loss function, vectorized version.

    Inputs and outputs are the same as softmax_loss_naive.
    """
    # Initialize the loss and gradient to zero.
    loss = 0.0
    dW = np.zeros_like(W)
    num_train = X.shape[0]
    num_classes = W.shape[1]
    scores = X.dot(W)
    scores = np.exp(scores) # 500 * 10
    correct_scores = scores[range(num_train), y]
    loss = -np.log(correct_scores / np.sum(scores, axis = 1))

    dS = scores / np.reshape(np.sum(scores, axis = 1), [-1, 1])
    dS[range(num_train), y] = scores[range(num_train), y] / np.sum(scores, axis = 1) - 1
    dW = np.dot(X.T, dS)
    #-----------------------------#
    loss /= num_train
    dW /= num_train
    loss = np.sum(loss)

    loss += reg * np.sum(W * W)
    dW += 2 * reg * W

    return loss, dW
```

- 벡터를 빼는 것은 -1로 넣어서 나중에 한번에 행렬곱셈으로 계산한다.
- `reshape` 를 통해서 사이즈를 맞춰준다.
- 공통되는 분모들은 한번에 나눠준다.

```python
for lr in learning_rates:
  for reg in regularization_strengths:
    softmax = Softmax()
    softmax.train(X_train, y_train, learning_rate = lr, reg = reg,
                  num_iters = 1500, verbose = False)
    y_train_pred = softmax.predict(X_train)
    training_accuracy = np.mean(y_train == y_train_pred)
    y_val_pred = softmax.predict(X_val)
    validation_accuracy = np.mean(y_val == y_val_pred)

    results[(lr, reg)] = (training_accuracy, validation_accuracy)

    if best_val < validation_accuracy:
      best_val = validation_accuracy
      best_softmax = softmax

pass

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

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print('lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy))

print('best validation accuracy achieved during cross-validation: %f' % best_val)
```

```
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.334551 val accuracy: 0.345000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.313612 val accuracy: 0.329000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.317898 val accuracy: 0.327000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.306327 val accuracy: 0.321000
```

- validation은 코드가 같으니 생략하겠다.

2. Inline Question 2

Inline Question 2 - *True or False*

Suppose the overall training loss is defined as the sum of the per-datapoint loss over all training examples. It is possible to add a new datapoint to a training set that would leave the SVM loss unchanged, but this is not the case with the Softmax classifier loss.

A: False

SVM의 loss가 변하지 않는 data는 score값이 클 것으로 예상된다. 하지만 Softmax의 loss가 변하지 않기 위해서는 확률 값이 1이 되어야 하는데 그러면 다른 데이터 값의 score가 -inf가 되어야 exp취했을 때 0이된다. 이는 불가능하므로 Softmax의 loss는 변하게 된다.