exam 개인학습

Lec 1.2

AI summary Try on this page

Type something...

01) SVM

01_Data Preprocessing

- 데이터 각각을 표준화시켜서 전처리를 진행하였다.

```
[8]  # Preprocessing: reshape the image data into rows
     X_train = np.reshape(X_train, (X_train.shape[0], -1))
     X_val = np.reshape(X_val, (X_val.shape[0], -1))
     X_test = np.reshape(X_test, (X_test.shape[0], -1))
     X_dev = np.reshape(X_dev, (X_dev.shape[0], -1))

     # As a sanity check, print out the shapes of the data
     print('Training data shape: ', X_train.shape)
     print('Validation data shape: ', X_val.shape)
     print('Test data shape: ', X_test.shape)
     print('dev data shape: ', X_dev.shape)

     Training data shape:  (49000, 3072)
     Validation data shape:  (1000, 3072)
     Test data shape:  (1000, 3072)
     dev data shape:  (500, 3072)
```

02_svm_loss_naive(W, x, y, reg)

- 함수는 총 2개를 구현하는 것이 목적이었다. 먼저 svm_loss_naive를 구현해보자.
- reg는 규제 정도로 loss를 구한 후 W의 L2 norm과 곱해서 loss에 더할 것이다.
- 먼저 loss를 구하는 것이 목적이었는데 각각의 loss를 구하기 위해서 먼저 score를 계산해주었다.
- train data와 W를 (행렬곱)시켜줘서 값을 계산해준 후 y[i]에 해당하는 correct score를 따로 선언하였다.
- 그 후 score - correct_ score + 1과 0의 max 값을 할당해주었다. 이와 동시에 dW를 계산해주어야 했는데 이는 ppt의 나와있는 식을 참고하여 계산하였다.
- 평균을 구해줘야하기 때문에 모든 열에다가 다 더해준 후 `num_train`으로 나눠줬다.

Computing the gradient analytically with Calculus

The numerical gradient is very simple to compute using the finite difference approximation, but the downside is that it is approximate (since we have to pick a small value of *h*, while the true gradient is defined as the limit as *h* goes to zero), and that it is very computationally expensive to compute. The second way to compute the gradient is analytically using Calculus, which allows us to derive a direct formula for the gradient (no approximations) that is also very fast to compute. However, unlike the numerical gradient it can be more error prone to implement, which is why in practice it is very common to compute the analytic gradient and compare it to the numerical gradient to check the correctness of your implementation. This is called a **gradient check**.

Lets use the example of the SVM loss function for a single datapoint:

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

We can differentiate the function with respect to the weights. For example, taking the gradient with respect to w_{y_i} we obtain:

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

where $\mathbf{1}$ is the indicator function that is one if the condition inside is true or zero otherwise. While the expression may look scary when it is written out, when you're implementing this in code you'd simply count the number of classes that didn't meet the desired margin (and hence contributed to the loss function) and then the data vector x_i scaled by this number is the gradient. Notice that this is the gradient only with respect to the row of W that corresponds to the correct class. For the other rows where $j \neq y_i$ the gradient is:

$$\nabla_{w_j} L_i = \mathbf{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

```python
def svm_loss_naive(W, X, y, reg):
    """
    Structured SVM loss function, naive implementation (with loops).

    Inputs have dimension D, there are C classes, and we operate on minibatches
    of N examples.

    Inputs:
    - W: A numpy array of shape (D, C) containing weights.
    - X: A numpy array of shape (N, D) containing a minibatch of data.
    - y: A numpy array of shape (N,) containing training labels; y[i] = c means
      that X[i] has label c, where 0 <= c < C.
    - reg: (float) regularization strength

    Returns a tuple of:
    - loss as single float
    - gradient with respect to weights W; an array of same shape as W
    """
    dW = np.zeros(W.shape)  # initialize the gradient as zero

    # compute the loss and the gradient
    num_classes = W.shape[1] # 10개
    num_train = X.shape[0] # X의 개수 -> 500, 49000....
    loss = 0.0
    #print(X[0].shape) # 1x3073
    #print(W.shape)    # 3073 x 10
    #print(num_train) # 500
    for i in range(num_train):
        scores = X[i].dot(W) #  X가 전치행렬로 바뀌어서 계산
        correct_class_score = scores[y[i]] # np.dot와 같음
        for j in range(num_classes):
            if j == y[i]:
                continue
            margin = scores[j] - correct_class_score + 1  # note delta = 1
            if margin > 0:
                loss += margin
                W_der1 = X[i]
                dW[:, j] += W_der1 # j번째 열에 모두 더해주기
                dW[:, y[i]] -= W_der1

    # dW = dL/dW
```

```python
    # Right now the loss is a sum over all training examples, but we want it
    # to be an average instead so we divide by num_train.
    loss /= num_train
    dW /= num_train
    # Add regularization to the loss.
    loss += reg * np.sum(W * W) # reg: regularization strength

    #############################################################################
    # TODO:                                                                     #
    # Compute the gradient of the loss function and store it dW.                #
    # Rather that first computing the loss and then computing the derivative,   #
    # it may be simpler to compute the derivative at the same time that the     #
    # loss is being computed. As a result you may need to modify some of the    #
    # code above to compute the gradient.                                       #
    #############################################################################
    # +++++START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)+++++

    # loss function의 기울기 구하고 dW를 저장, loss는 숫자
    pass
    dW += 2 * reg * W # 뒷부분 더하기
    # +++++END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)+++++

    return loss, dW
```

03_svm_loss_vectorized(W, x, y, reg)

- 이 함수는 1번 함수에서 vectorized를 이용하였다.
- 똑같이 계산해주는데 `X.dot(W)`로 한꺼번에 계산해주고 correct_score도 `scores[range(num_train), y)`를 이용해서 한번에 슬라이싱 해준다.
- 그리고 maximum를 이용해서 배열에서 최대값을 한꺼번에 연산해준다.
- correct_score에 해당하는 부분도 한꺼번에 0으로 만들어준다.
- dW에 correct_score에 해당하는 부분을 0으로 지정해주었다.
- 그리고 dW는 xi들의 합으로 이루어지기 때문에 이를 이용하기 위해 새로운 temp이라는 배열을 선언해주었다.
- temp[loss > 0] = 1로 0보다 큰 부분에는 1을 할당해주었다.
- 그 후 loss가 0보다 큰 횟수만큼 correct score를 빼주어야 하므로 1의 개수를 새어주기 위해서 np.sum(temp, axis = 1)을 사용하였다./co

```
def svm_loss_vectorized(W, X, y, reg):
    """
    Structured SVM loss function, vectorized implementation.

    Inputs and outputs are the same as svm_loss_naive.
    """
    loss = 0.0
    dW = np.zeros(W.shape)  # initialize the gradient as zero

    #############################################################################
    # TODO:                                                                     #
    # Implement a vectorized version of the structured SVM loss, storing the    #
    # result in loss.                                                           #
    #############################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    num_train = X.shape[0]
    scores = X.dot(W)
    # 슬라이싱
    correct_class_score = scores[range(num_train), y].reshape(-1, 1)
    #print(correct_class_score.shape) # 500 x 1
    loss = np.maximum(0, scores - correct_class_score + 1) # 배열에서 최대값
    loss[range(num_train), y] = 0
    # dW: 3073 x 10
    temp = np.zeros(loss.shape) # 500 x 10
    temp[loss > 0] = 1
    temp[range(num_train), y] -= np.sum(temp, axis = 1)
    dW = np.dot(X.T, temp) # 3073 x 10, X에 있는 값들 모두 더해주면됨



    dW = dW / num_train
    loss = np.sum(loss)/num_train
    loss += reg * np.sum(W * W)
    pass

```

04_train

```
            ###########################################################################
            # TODO:                                                                   #
            # Sample batch_size elements from the training data and their             #
            # corresponding labels to use in this round of gradient descent.          #
            # Store the data in X_batch and their corresponding labels in             #
            # y_batch; after sampling X_batch should have shape (batch_size, dim)     #
            # and y_batch should have shape (batch_size,)                             #
            #                                                                         #
            # Hint: Use np.random.choice to generate indices. Sampling with           #
            # replacement is faster than sampling without replacement.                #
            ###########################################################################
            # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
            x_idx = np.random.choice(num_train, batch_size, False)
            X_batch = X[x_idx]
            y_batch = y[x_idx] # index일치
            #print(X_batch.shape) # 200 x 3073
            pass

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

            # evaluate loss and gradient
            loss, grad = self.loss(X_batch, y_batch, reg)
            loss_history.append(loss)

            # perform parameter update
            ###########################################################################
            # TODO:                                                                   #
            # Update the weights using the gradient and the learning rate.            #
            ###########################################################################
            # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
            self.W -= learning_rate * grad
            pass

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

            if verbose and it % 100 == 0:
                print("iteration %d / %d: loss %f" % (it, num_iters, loss))

        return loss_history
```

- 간단하게 W를 update해주는 부분만 추가되었다.

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
for lr in learning_rates:
  for rs in regularization_strengths:
    svm = LinearSVM()
    result = (lr, rs)
    loss_hist = svm.train(X_train, y_train, learning_rate=lr, reg=rs,
                          num_iters=1500, verbose=False)
    y_train_pred = svm.predict(X_train)
    training_accuracy = np.mean(y_train == y_train_pred)
    y_val_pred = svm.predict(X_val)
    validation_accuracy = np.mean(y_val == y_val_pred)

    value = (training_accuracy, validation_accuracy)
    results[result] = value
    # 최적인지 판단
    if (best_val < validation_accuracy):
      best_val = validation_accuracy
      best_svm = svm

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)
```

```
/content/drive/My Drive/cs231n/assignments/assignment1/cs231n/classifiers/linear_svm.py:110: RuntimeWarning: overflow encountered in double_scalars
  loss += reg * np.sum(W * W)
/usr/local/lib/python3.8/dist-packages/numpy/core/fromnumeric.py:86: RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
/content/drive/My Drive/cs231n/assignments/assignment1/cs231n/classifiers/linear_svm.py:110: RuntimeWarning: overflow encountered in multiply
  loss += reg * np.sum(W * W)
/content/drive/My Drive/cs231n/assignments/assignment1/cs231n/classifiers/linear_svm.py:125: RuntimeWarning: overflow encountered in multiply
  dW += 2 * reg * W # 윗부분 더하기
/content/drive/My Drive/cs231n/assignments/assignment1/cs231n/classifiers/linear_classifier.py:86: RuntimeWarning: invalid value encountered in subtract
  self.W -= learning_rate * grad
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.369939 val accuracy: 0.386000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.354327 val accuracy: 0.361000
lr 5.000000e-05 reg 2.500000e+04 train accuracy: 0.054531 val accuracy: 0.050000
lr 5.000000e-05 reg 5.000000e+04 train accuracy: 0.100265 val accuracy: 0.087000
best validation accuracy achieved during cross-validation: 0.386000
```

- 모델을 만들어서 최적화를 시켜줬다. 각각의 parameter을 통해서 언제 최적의 값을 갖는지를 구했다.

1. Inline Question 2

Inline question 2

Describe what your visualized SVM weights look like, and offer a brief explanation for why they look the way they do.

SVM weight의 이미지를 보면 원래의 이미지와 비슷한 것을 확인할 수 있다. 이는 내적했을 때 값이 크게 나와야 하므로 평행한 값으로 할당되었기 때문이다.