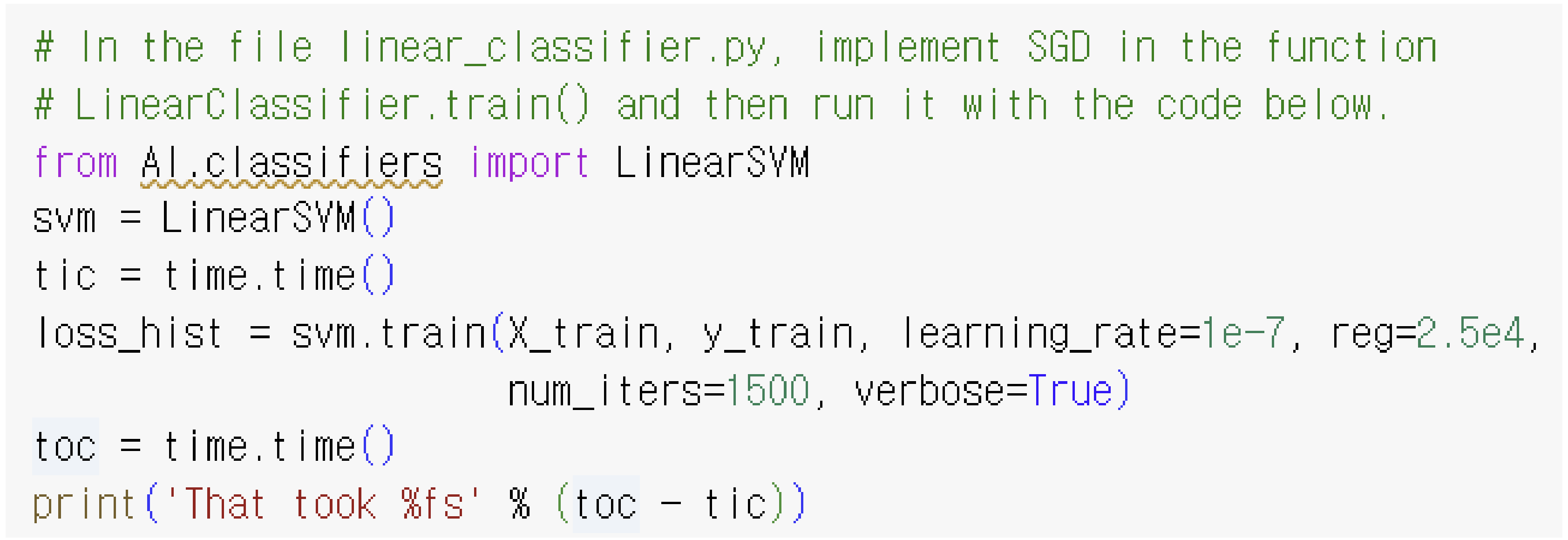
실습2: svm 후반부

확률적 경사 하강법

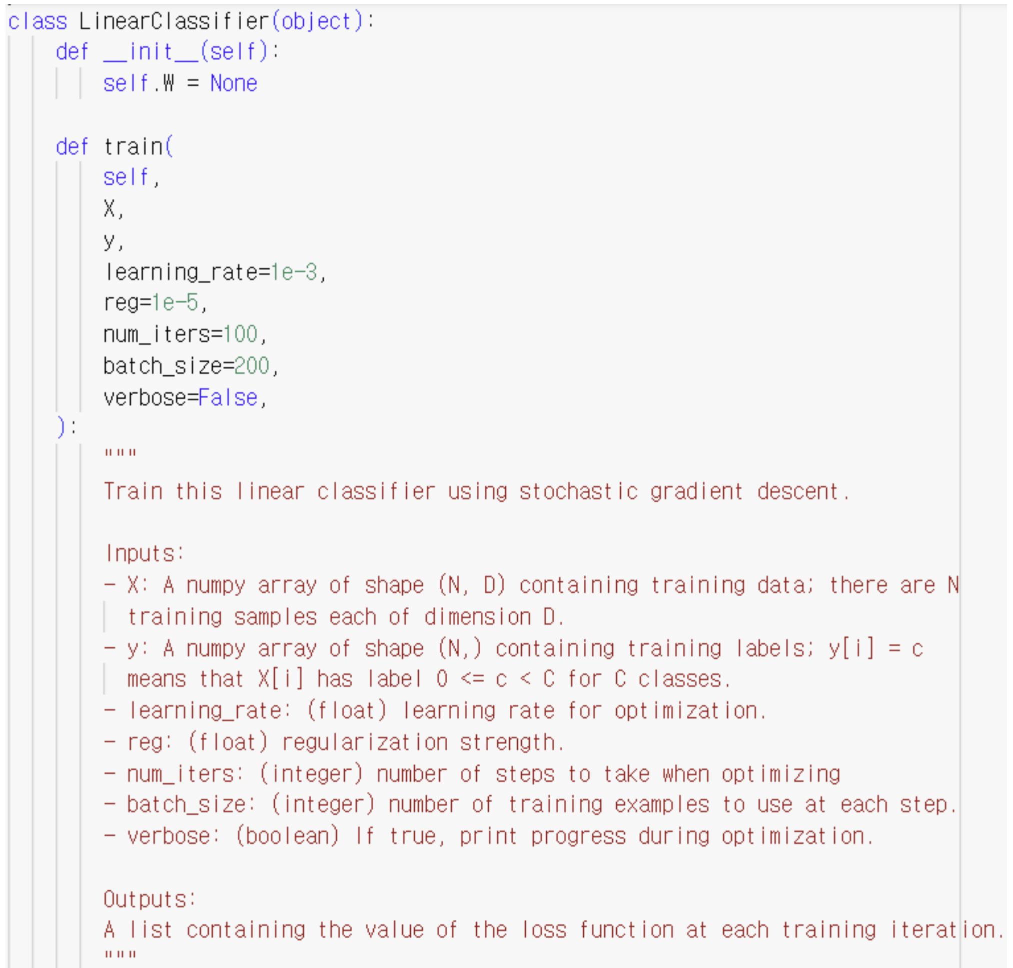
- SVM에서 손실과 그래디언트에 대한 벡터화 사용 계산 구현 완료
- 확률적 경사 하강법을 사용하여 손실을 최소화할 예정



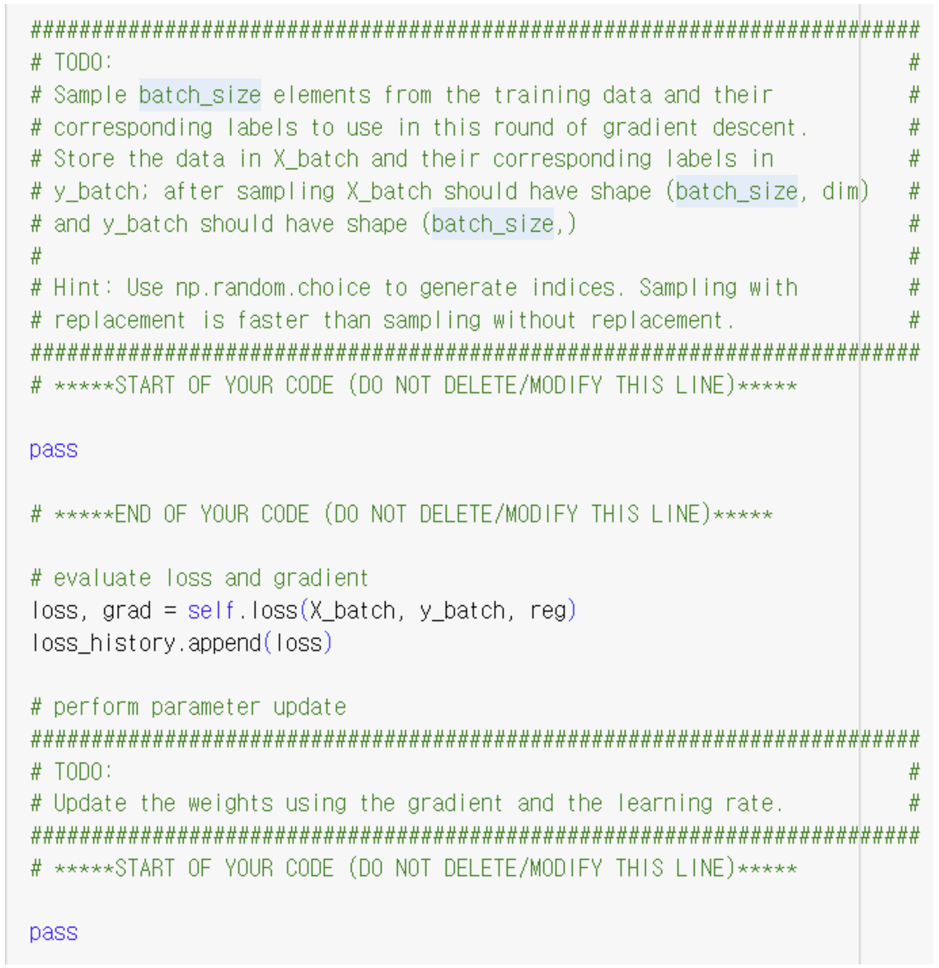
```
# In the file linear_classifier.py, implement SGD in the function
# LinearClassifier.train() and then run it with the code below.
from AI.classifiers import LinearSYM
svm = LinearSYM()
tic = time.time()
loss_hist = svm.train(X_train, y_train, learning_rate=1e-7, reg=2.5e4,
                      num_iters=1500, verbose=True)
toc = time.time()
print('That took %fs' % (toc - tic))
```

구현1: LinearClassifier 클래스의 train 메서드

- AI1/AI/classifiers/linear_classifier.py 파일 안의 LinearClassifier 클래스의 train 메서드 구현 필요



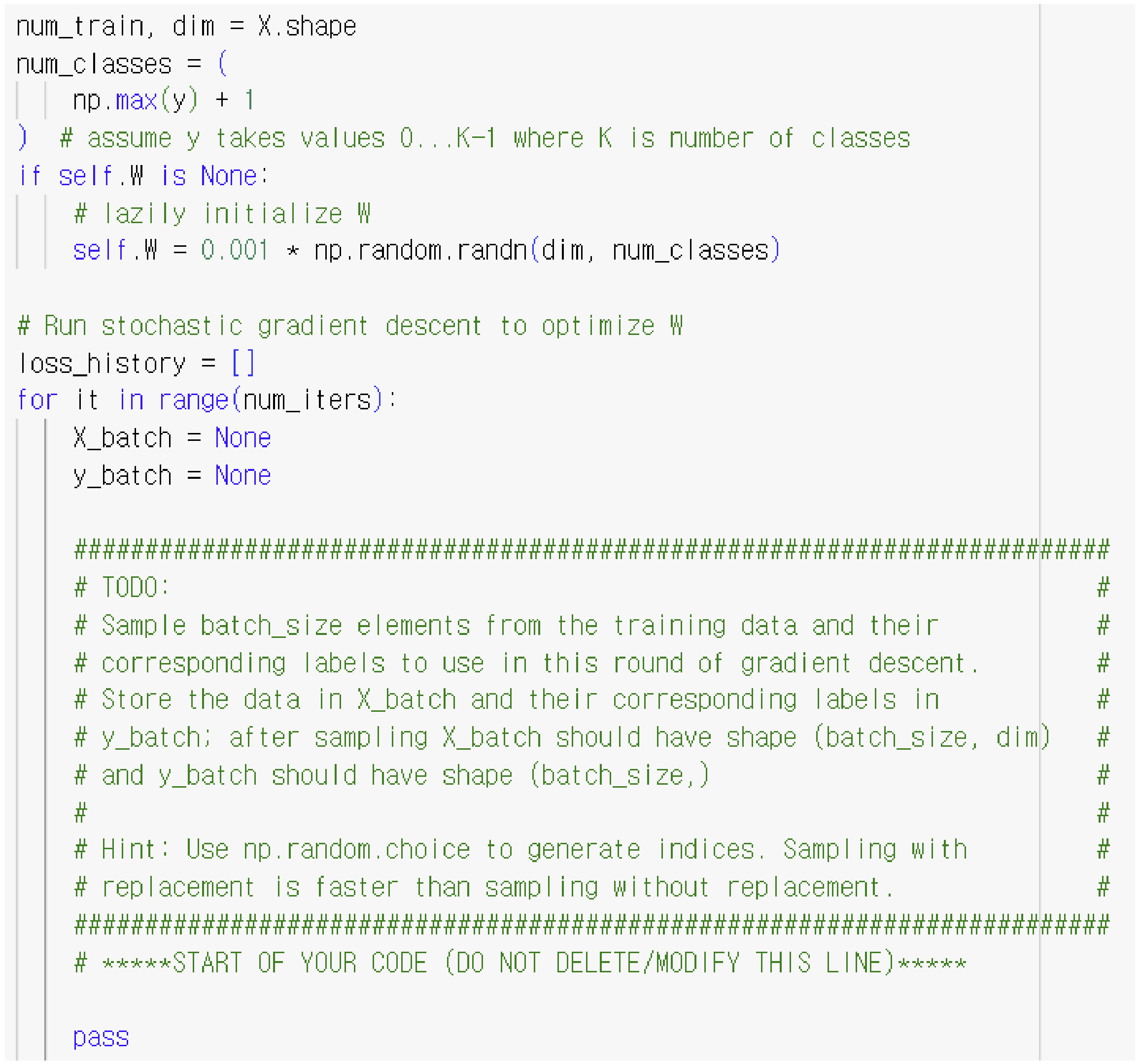
```python
class LinearClassifier(object):
    def __init__(self):
        self.W = None

    def train(
        self,
        X,
        y,
        learning_rate=1e-3,
        reg=1e-5,
        num_iters=100,
        batch_size=200,
        verbose=False,
    ):
        """
        Train this linear classifier using stochastic gradient descent.

        Inputs:
        - X: A numpy array of shape (N, D) containing training data; there are N
          training samples each of dimension D.
        - y: A numpy array of shape (N,) containing training labels; y[i] = c
          means that X[i] has label 0 <= c < C for C classes.
        - learning_rate: (float) learning rate for optimization.
        - reg: (float) regularization strength.
        - num_iters: (integer) number of steps to take when optimizing
        - batch_size: (integer) number of training examples to use at each step.
        - verbose: (boolean) If true, print progress during optimization.

        Outputs:
        A list containing the value of the loss function at each training iteration.
        """
```



```python
#########################################################################
# 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)*****

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)*****

pass
```

구현1: LinearClassifier 클래스의 train 메서드

- 훈련세트에서 batch_size 크기의 배치를 선택



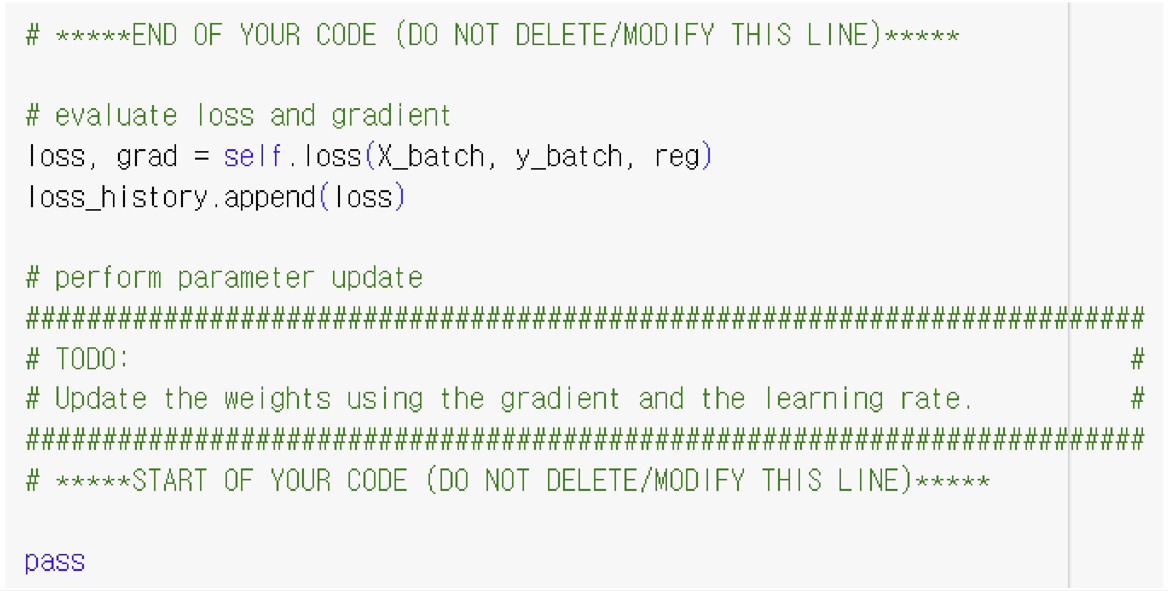
```
num_train, dim = X.shape
num_classes = (
    np.max(y) + 1
)  # assume y takes values 0...K-1 where K is number of classes
if self.W is None:
    # lazily initialize W
    self.W = 0.001 * np.random.randn(dim, num_classes)

# Run stochastic gradient descent to optimize W
loss_history = []
for it in range(num_iters):
    X_batch = None
    y_batch = None

    #########################################################################
    # 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)*****

    pass
```

구현1: LinearClassifier 클래스의 train 메서드

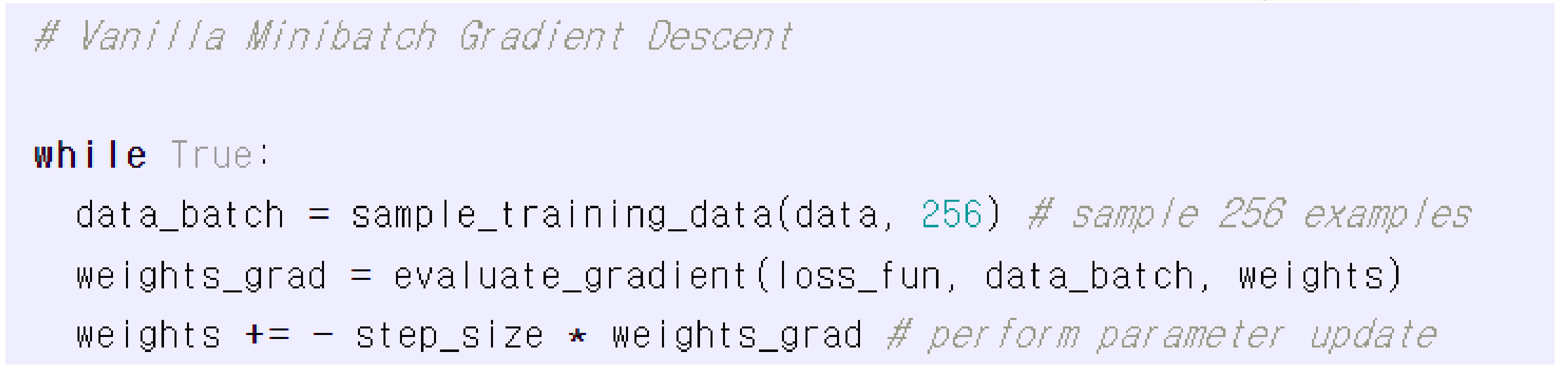
- 순차적 선택이 아닌 랜덤 선택 사용
- 중복을 허용하는 복원 추출 사용
- batch_size x D 크기의 배열 X_batch 및 batch_size 크기 배열 y_batch 생성

구현1: LinearClassifier 클래스의 train 메서드

- loss 메서드는 이미 구현이 되어있음
- 가중치 W 업데이트 부분을 구현해야함



```
# *****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)*****

pass
```



```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

확률적 경사 하강법

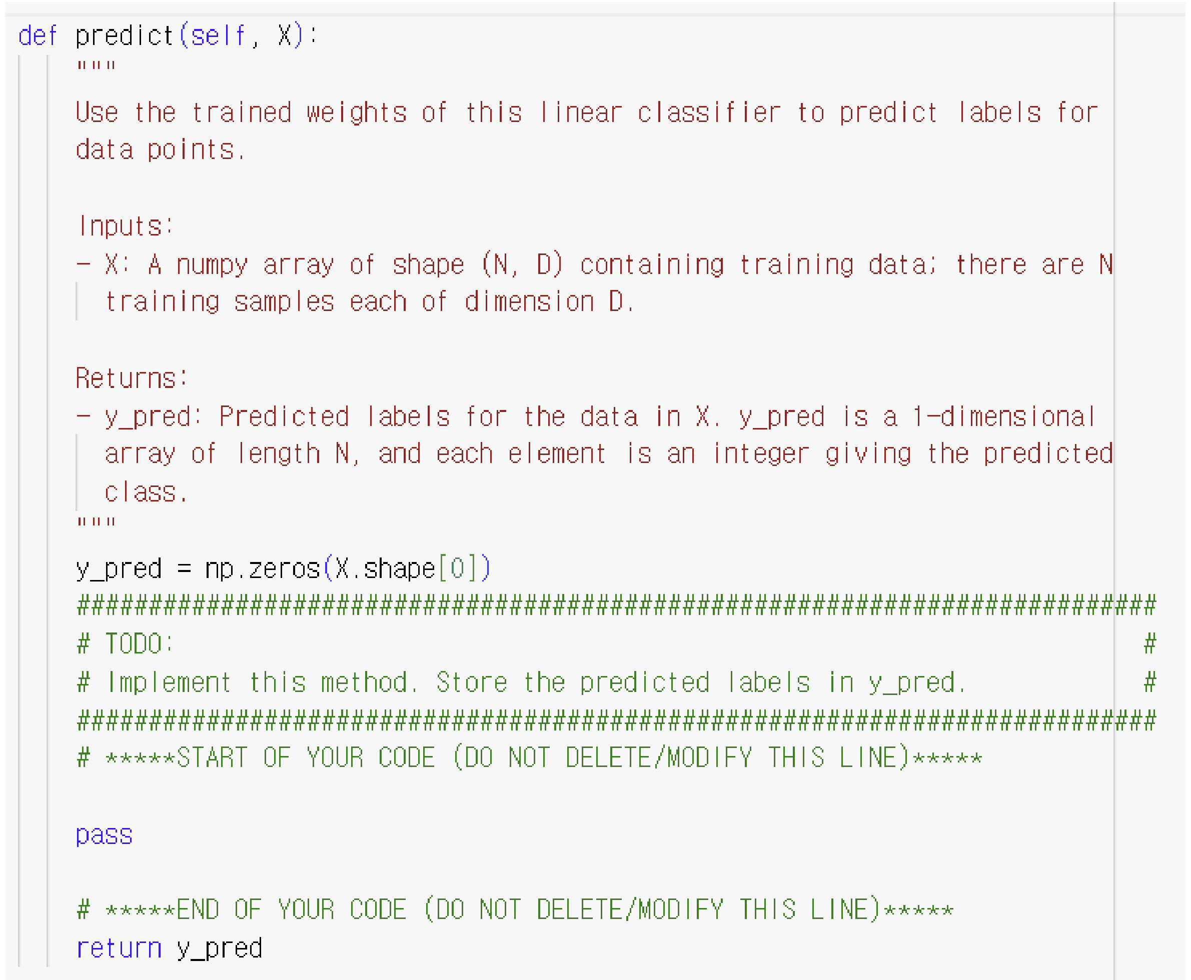
```
# In the file linear_classifier.py, implement SGD in the function
# LinearClassifier.train() and then run it with the code below.
from AI.classifiers import LinearSYM
svm = LinearSYM()
tic = time.time()
loss_hist = svm.train(X_train, y_train, learning_rate=1e-7, reg=2.5e4,
                      num_iters=1500, verbose=True)
toc = time.time()
print('That took %fs' % (toc - tic))
```

손실 그래프 그리기

```
# A useful debugging strategy is to plot the loss as a function of
# iteration number:
plt.plot(loss_hist)
plt.xlabel('Iteration number')
plt.ylabel('Loss value')
plt.show()
```

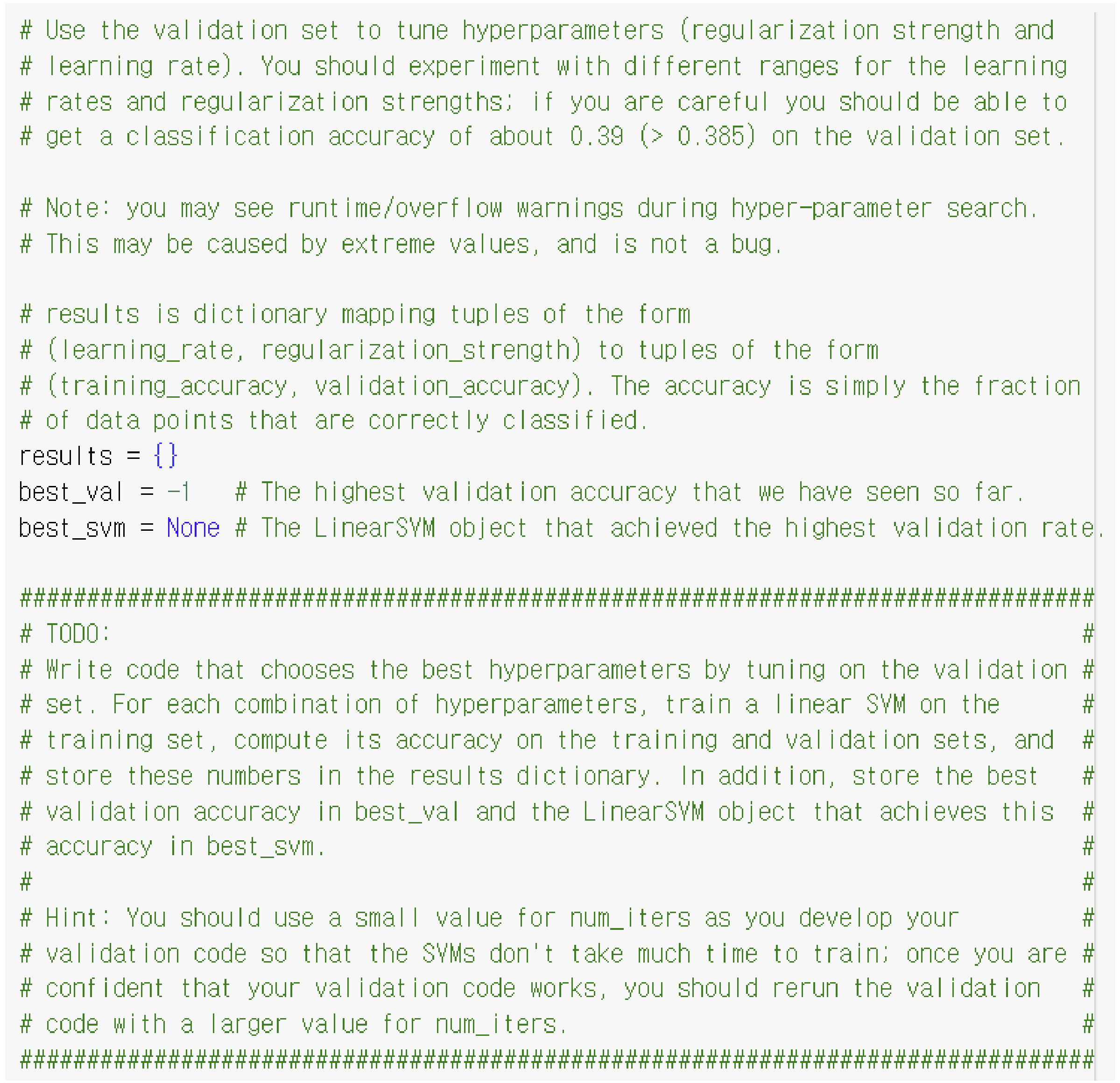
구현2: LinearClassifier 클래스의 predict 메서드

- 점수 행렬 계산: $S = X \cdot W$
- 점수 행렬의 각 행에서 가장 큰 점수를 가진 것의 인덱스(label)을 구하면 됨
- np.argmax 함수 활용 가능

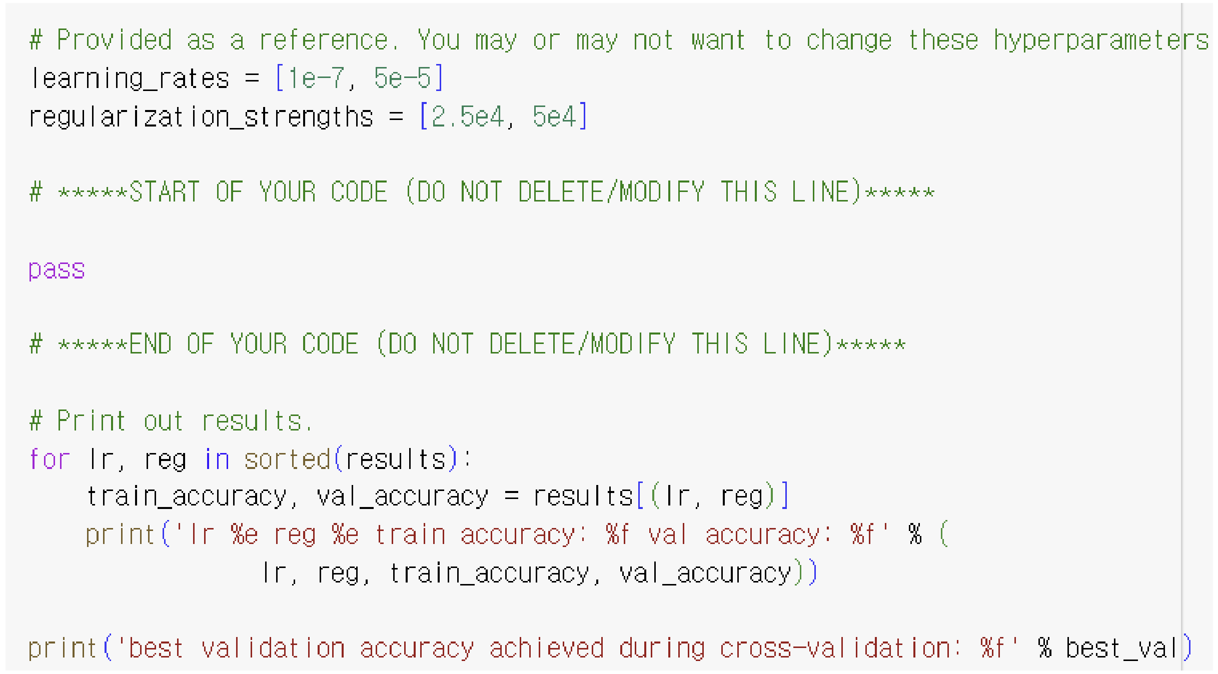


```python
def predict(self, X):
    """
    Use the trained weights of this linear classifier to predict labels for
    data points.

    Inputs:
    - X: A numpy array of shape (N, D) containing training data; there are N
      training samples each of dimension D.

    Returns:
    - y_pred: Predicted labels for the data in X. y_pred is a 1-dimensional
      array of length N, and each element is an integer giving the predicted
      class.
    """
    y_pred = np.zeros(X.shape[0])
    ###########################################################################
    # TODO:                                                                   #
    # Implement this method. Store the predicted labels in y_pred.            #
    ###########################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    pass

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

구현3: 검증 과정 구현



```python
# Use the validation set to tune hyperparameters (regularization strength and
# learning rate). You should experiment with different ranges for the learning
# rates and regularization strengths; if you are careful you should be able to
# get a classification accuracy of about 0.39 (> 0.385) on the validation set.

# Note: you may see runtime/overflow warnings during hyper-parameter search.
# This may be caused by extreme values, and is not a bug.

# results is dictionary mapping tuples of the form
# (learning_rate, regularization_strength) to tuples of the form
# (training_accuracy, validation_accuracy). The accuracy is simply the fraction
# of data points that are correctly classified.
results = {}
best_val = -1   # The highest validation accuracy that we have seen so far.
best_svm = None # The LinearSVM object that achieved the highest validation rate.

################################################################################
# TODO:                                                                        #
# Write code that chooses the best hyperparameters by tuning on the validation #
# set. For each combination of hyperparameters, train a linear SVM on the      #
# training set, compute its accuracy on the training and validation sets, and  #
# store these numbers in the results dictionary. In addition, store the best   #
# validation accuracy in best_val and the LinearSVM object that achieves this  #
# accuracy in best_svm.                                                        #
#                                                                              #
# Hint: You should use a small value for num_iters as you develop your         #
# validation code so that the SVMs don't take much time to train; once you are #
# confident that your validation code works, you should rerun the validation   #
# code with a larger value for num_iters.                                      #
################################################################################
```

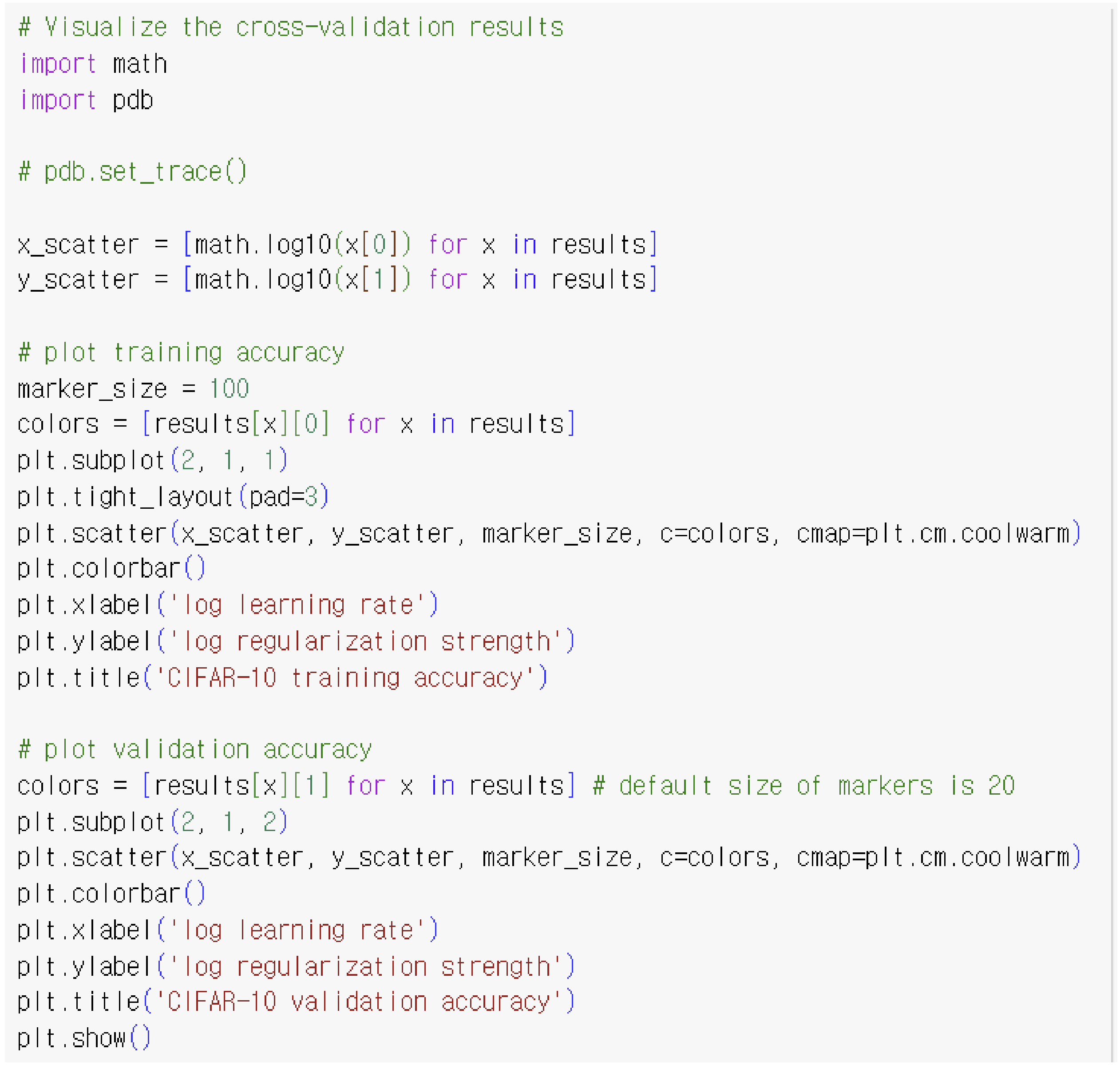


```python
# Provided as a reference. You may or may not want to change these hyperparameters
learning_rates = [1e-7, 5e-5]
regularization_strengths = [2.5e4, 5e4]

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

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)
```

구현3: 검증 과정 구현

- 학습률, 정규화 강도에 대한 이중 loop 사용
- train 및 predict 메서드를 사용하여 훈련 및 레이블 예측
- 훈련세트 및 검증세트 각각에 대한 정확도를 모두 구해야함

정확도 시각화



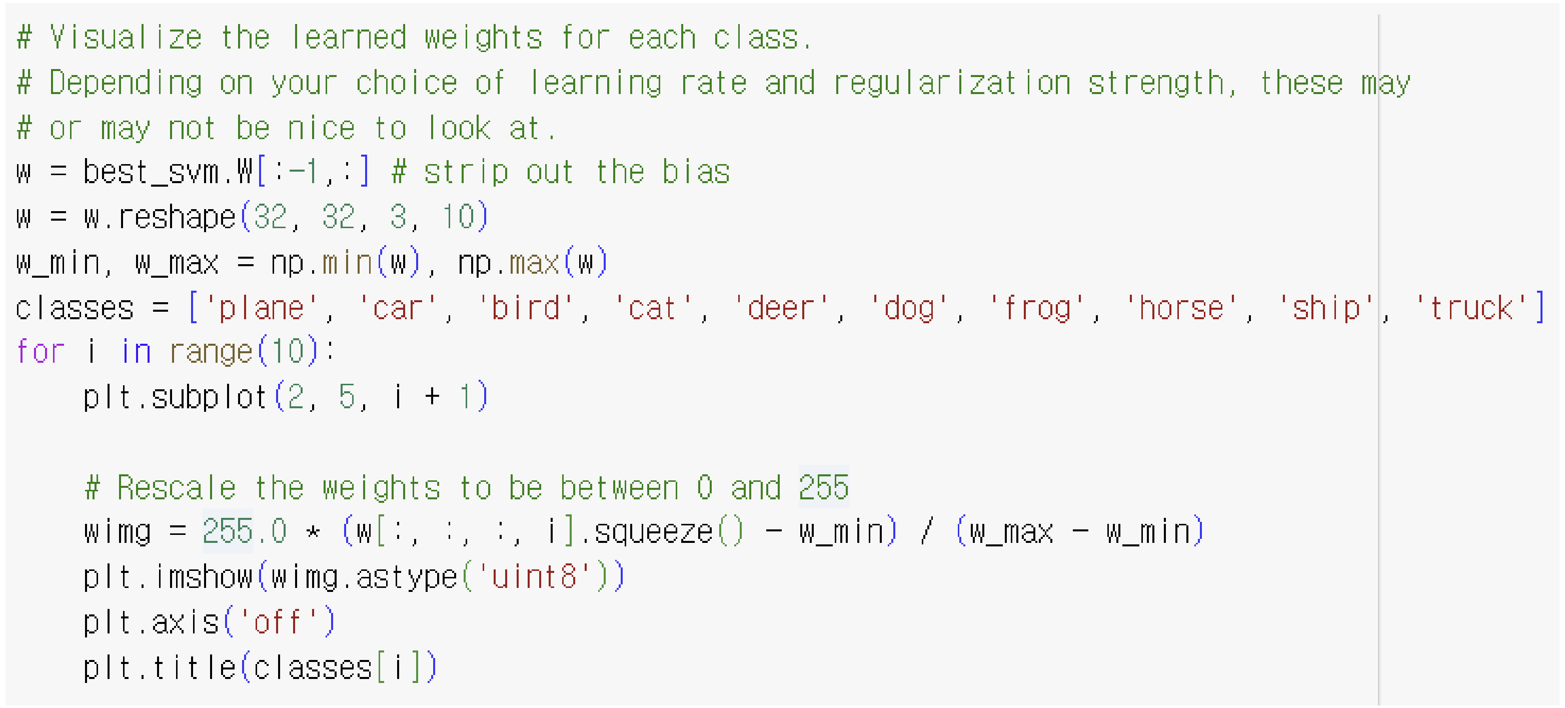
```python
# Visualize the cross-validation results
import math
import pdb

# pdb.set_trace()

x_scatter = [math.log10(x[0]) for x in results]
y_scatter = [math.log10(x[1]) for x in results]

# plot training accuracy
marker_size = 100
colors = [results[x][0] for x in results]
plt.subplot(2, 1, 1)
plt.tight_layout(pad=3)
plt.scatter(x_scatter, y_scatter, marker_size, c=colors, cmap=plt.cm.coolwarm)
plt.colorbar()
plt.xlabel('log learning rate')
plt.ylabel('log regularization strength')
plt.title('CIFAR-10 training accuracy')

# plot validation accuracy
colors = [results[x][1] for x in results] # default size of markers is 20
plt.subplot(2, 1, 2)
plt.scatter(x_scatter, y_scatter, marker_size, c=colors, cmap=plt.cm.coolwarm)
plt.colorbar()
plt.xlabel('log learning rate')
plt.ylabel('log regularization strength')
plt.title('CIFAR-10 validation accuracy')
plt.show()
```

테스트 세트 분류

```python
# Evaluate the best svm on test set
y_test_pred = best_svm.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print('linear SVM on raw pixels final test set accuracy: %f' % test_accuracy)
```

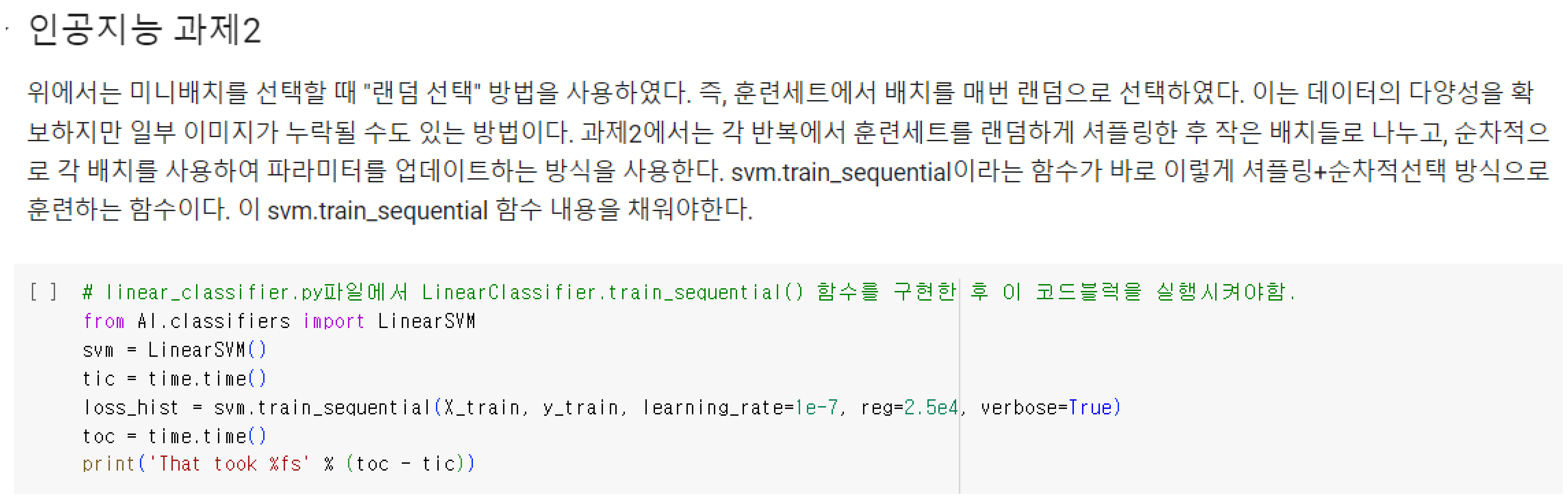
템플릿 그리기

- 훈련된 가중치 행렬을 이미지로 그림 (템플릿)



```
# Visualize the learned weights for each class.
# Depending on your choice of learning rate and regularization strength, these may
# or may not be nice to look at.
w = best_svm.W[:-1,:] # strip out the bias
w = w.reshape(32, 32, 3, 10)
w_min, w_max = np.min(w), np.max(w)
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
for i in range(10):
    plt.subplot(2, 5, i + 1)

    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[:, :, :, i].squeeze() - w_min) / (w_max - w_min)
    plt.imshow(wimg.astype('uint8'))
    plt.axis('off')
    plt.title(classes[i])
```

과제2

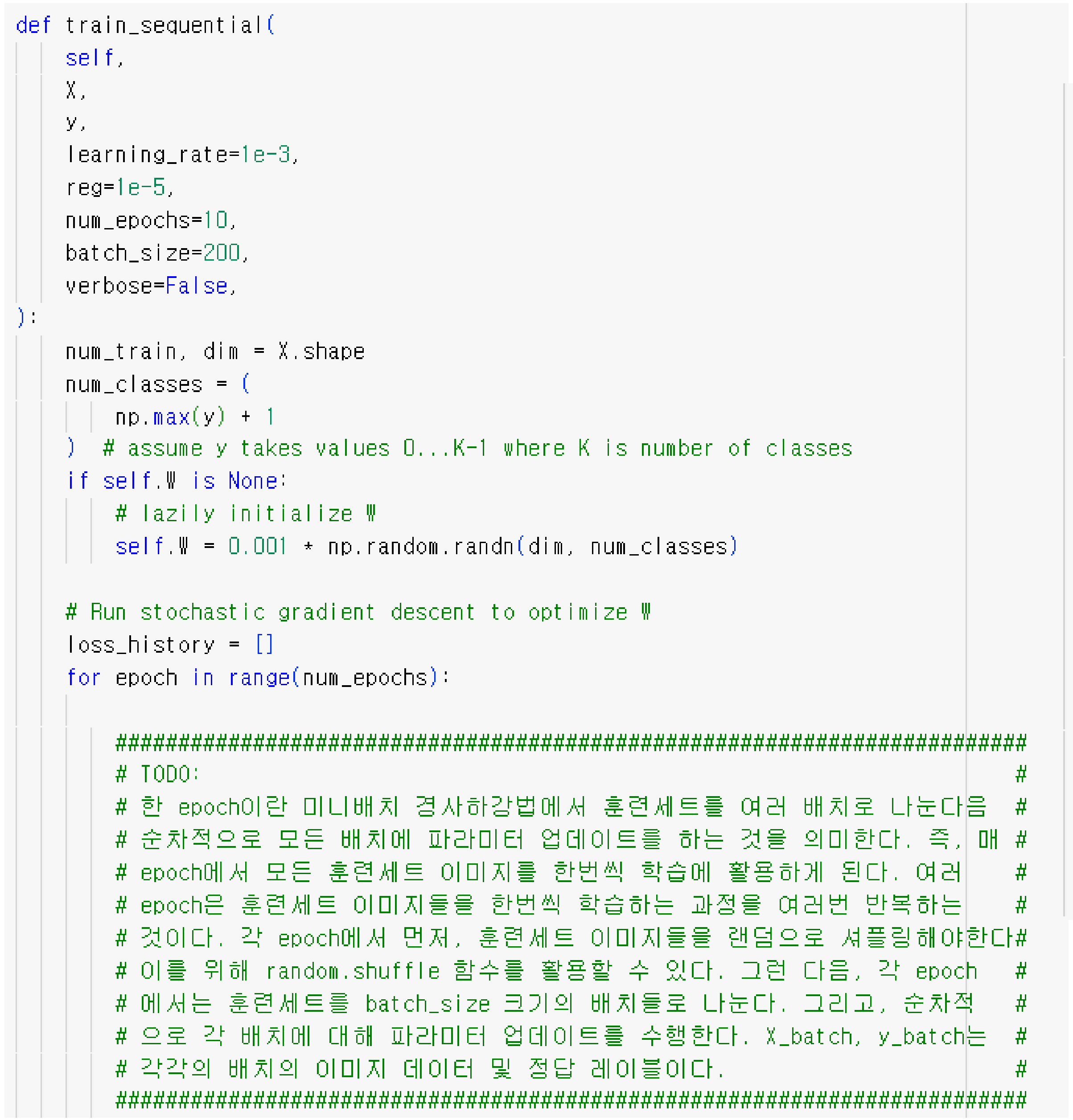


인공지능 과제2

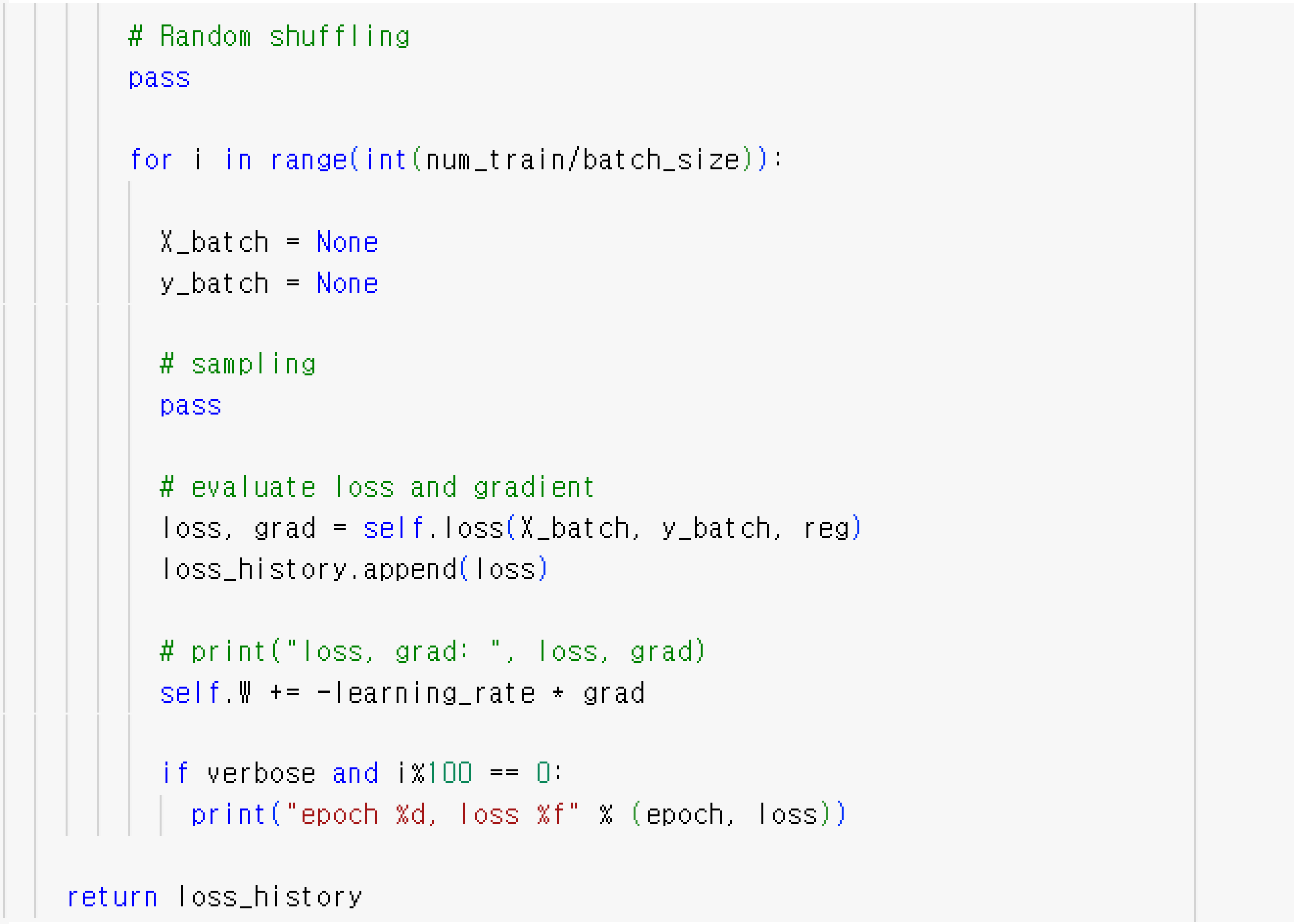
위에서는 미니배치를 선택할 때 "랜덤 선택" 방법을 사용하였다. 즉, 훈련세트에서 배치를 매번 랜덤으로 선택하였다. 이는 데이터의 다양성을 확보하지만 일부 이미지가 누락될 수도 있는 방법이다. 과제2에서는 각 반복에서 훈련세트를 랜덤하게 셔플링한 후 작은 배치들로 나누고, 순차적으로 각 배치를 사용하여 파라미터를 업데이트하는 방식을 사용한다. svm.train_sequential이라는 함수가 바로 이렇게 셔플링+순차적선택 방식으로 훈련하는 함수이다. 이 svm.train_sequential 함수 내용을 채워야한다.

```
[ ]  # linear_classifier.py파일에서 LinearClassifier.train_sequential() 함수를 구현한 후 이 코드블럭을 실행시켜야함.
     from AI.classifiers import LinearSVM
     svm = LinearSVM()
     tic = time.time()
     loss_hist = svm.train_sequential(X_train, y_train, learning_rate=1e-7, reg=2.5e4, verbose=True)
     toc = time.time()
     print('That took %fs' % (toc - tic))
```

과제2



```python
def train_sequential(
    self,
    X,
    y,
    learning_rate=1e-3,
    reg=1e-5,
    num_epochs=10,
    batch_size=200,
    verbose=False,
):
    num_train, dim = X.shape
    num_classes = (
        np.max(y) + 1
    )  # assume y takes values 0...K-1 where K is number of classes
    if self.W is None:
        # lazily initialize W
        self.W = 0.001 * np.random.randn(dim, num_classes)

    # Run stochastic gradient descent to optimize W
    loss_history = []
    for epoch in range(num_epochs):

        #########################################################################
        # TODO:                                                                 #
        # 한 epoch이란 미니배치 경사하강법에서 훈련세트를 여러 배치로 나눈다음  #
        # 순차적으로 모든 배치에 파라미터 업데이트를 하는 것을 의미한다. 즉, 매 #
        # epoch에서 모든 훈련세트 이미지를 한번씩 학습에 활용하게 된다. 여러    #
        # epoch은 훈련세트 이미지들을 한번씩 학습하는 과정을 여러번 반복하는    #
        # 것이다. 각 epoch에서 먼저, 훈련세트 이미지들을 랜덤으로 셔플링해야한다#
        # 이를 위해 random.shuffle 함수를 활용할 수 있다. 그런 다음, 각 epoch   #
        # 에서는 훈련세트를 batch_size 크기의 배치들로 나눈다. 그리고, 순차적   #
        # 으로 각 배치에 대해 파라미터 업데이트를 수행한다. X_batch, y_batch는  #
        # 각각의 배치의 이미지 데이터 및 정답 레이블이다.                       #
        #########################################################################
```



```python
        # Random shuffling
        pass

        for i in range(int(num_train/batch_size)):

            X_batch = None
            y_batch = None

            # sampling
            pass

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

            # print("loss, grad: ", loss, grad)
            self.W += -learning_rate * grad

            if verbose and i%100 == 0:
                print("epoch %d, loss %f" % (epoch, loss))

    return loss_history
```

과제2

Question 1

위에서는 학습률(learning rate)로 1e-7, 1e-6를 사용하였다. 만약 이보다 더 큰 학습률 (e.g., e-5, 5e-5)를 사용하면 성능이 어떤지 쓰시오. 직접 학습률을 바꿔서 돌려보되, 그 상태로 과제제출을 하지 말고 이후에 다시 작은 학습률(1e-7, 1e-6)에 대한 실험을 하여 그 결과가 있는 채로 과제를 제출해야한다.

답: