CSC 8980 Deep Learning

Assignment 2

1. KNN

Inline question: best_k = 10

TODO1:
```
dists[i,j] = np.sqrt(np.sum((X[i,:]-self.X_train[j,:])**2))
```

TODO2:
```
dists[i,:] = np.sqrt(np.sum((X[i,:]-self.X_train)**2,axis=1))
```

TODO3:
```
test_square = np.sum(X**2, axis=1, keepdims=True)
train_square = np.sum(self.X_train**2, axis=1)
dists = np.sqrt(test_square + train_square - 2*np.dot(X, self.X_train.T))
```

TODO4:
```
idx = np.argsort(dists[i,:])[0:k]
closest_y = self.y_train[idx]
```

TODO5:
```
y_pred[i] = np.argmax(np.bincount(closest_y))
```

2. SVM

Inline question: Each of them looks like a template image for that category. But due to limited capacity of linear classifier, the template is very blurry.

In linear_classifier.py
TODO1:
```
indexs = np.random.choice(num_train, batch_size)
X_batch = X[indexs]
y_batch = y[indexs]
```

TODO2:
```
self.W = self.W - learning_rate * grad
```

TODO3:
```
y_pred = np.argmax(np.dot(X, self.W), axis=1)
```

In linear_svm.py
TODO1:
```
if margin > 0:
  loss += margin
  dW[:,y[i]] -= X[i]
  dW[:,j] += X[i]
```

3. Softmax

Inline question: due to random initialization of W, the initial 10 scores are roughly equal. After applying softmax, p_k=0.1, for k=0…9, and therefore the expected loss is -log(0.1).

TODO1:

```
    # Use stable version of softmax
    exp_scores = np.exp(scores - np.max(scores))
    p = exp_scores / np.sum(exp_scores)
    loss += -np.log(p [y[i]])

    dp = probs
    dp [y[i]] -= 1
    dW += np.outer(X[i],dp)

  # Divide the loss by the number of training examples
  loss /= num_train
  dW /= num_train
  # Add regularization
  loss += reg * np.sum(W * W)
  dW += 2 * reg * W
```

TODO2:

```
  # Use stable version of softmax
  exp_scores = np.exp(scores - np.max(scores, axis=1, keepdims=True))
  p = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
  loss = -np.sum(np.log(p[np.arange(num_train),y]))

  # Divide the loss by the number of training examples and add regularization term
  loss /= num_train
  loss += reg * np.sum(W * W)

  dp = p
  dp[np.arange(num_train),y] -= 1
  dW = X.T.dot(dp)
  dW /= num_train
  dW += 2 * reg * W
```