COMP 540 - Homework 4

Guangyuan Yu(gy12),I still don't have a teammate

6 March 2017

1 Kernelizing the k-nearest neighbors

The distance we use in k-nn is

$$dis(x^{(i)}, x^{(j)}) = (x^{(i)}, x^{(j)})^2 = x^{(i)} * x^{(i)} - 2 * x^{(i)} * x^{(j)} + x^{(j)} * x^{(j)} \qquad (1)$$

We can kernelize the distance using Gaussian kernel K, which satisfies the Mercer Condition

$$\begin{aligned} dis(x^{(i)}, x^{(j)}) =& exp(-\frac{||x^{(i)} - x^{(i)}||}{2\sigma^2}) - 2 * exp(-\frac{||x^{(i)} - x^{(j)}||}{2\sigma^2}) + exp(-\frac{||x^{(j)} - x^{(j)}||}{2\sigma^2}) \\ =& 2 - 2 - exp(-\frac{||x^{(i)} - x^{(j)}||}{2\sigma^2}) \end{aligned} \qquad (2)$$

2 Constructing kernels

2.1

For arbitrary $\{x_1, x_2 ... x_n\}$ and $g \in R^n$, we show that

$$g^{'} K g = \sum_{i,j} g_i K(x_i, x_j) g_j = \sum_{i,j} g_i c * k_1(x_i, x_j) g_j = c * \sum_{i,j} g_i k_1(x_i, x_j) g_j \geqslant 0 \qquad (3)$$

2.2

$$g^{'} K g = \sum_{i,j} g_i f(x_i) k_1(x_i, x_j) f(x_j) g_j = \sum_{i,j} h_i k_1(x_i, x_j) h_j \geqslant 0 \qquad (4)$$

The last line follows because k_1 is a kernel function.

2.3

$$g^{'}Kg = \sum_{i,j} g_i K(x_i, x_j) g_j = \sum_{i,j} g_i (k_1(x_i, x_j) + k_2(x_i, x_j)) g_j = \sum_{i,j} g_i k_1(x_i, x_j) g_j + \sum_{i,j} g_i k_2(x_i, x_j) g_j \geqslant 0 \tag{5}$$

3 Fitting an SVM classifier by hand

3.1

we have $x = (0, -1) \rightarrow x = (1, 0, 0)$ and $x = (\sqrt{2}, 1) \rightarrow x = (1, 2, 2)$. $\theta^{'} = (0, 2, 2)$ is parallel to the optimal vector.

3.2

The margin is $\sqrt{2}$. The midpoint is (1,1,1). Margin is also the distance between SV to the midpoint.

3.3

Since $\frac{1}{|\theta|} = \sqrt{2}$, let assume the$\theta = (0, b, b)$, so that $\frac{1}{\sqrt{2 * b^2}} = \sqrt{2}$ we get $b = 0.5$. $\theta = (0, 0.5, 0.5)$

3.4

$$\begin{cases} -1((1,0,0) * (0, 0.5, 0.5) + \theta_0) & \geqslant 1 \\ 1 * ((1,2,2) * (0, 0.5, 0.5) + \theta_0) & \geqslant 1 \end{cases} \tag{6}$$

$\theta_0 = -1$

3.5

$$(0, 1/2, 1/2) * (1, \sqrt{2}x, x^2) - 1 = 0 \tag{7}$$

$$\frac{\sqrt{2}x}{2} + \frac{x^2}{2} - 1 = 0 \tag{8}$$

4 Support vector machines for binary classification

4.1 The hinge loss function and gradient

From Jupyter, I get $J = 1.0 grad = [-0.12956186 - 0.00167647]$
The code is

```
correctness = y * X.dot(theta)
J = 1. / (2 * m) * sum(theta ** 2) + 1. * C / m * sum(1 - correctness[co
grad = 1. / m * theta + 1. * C / m * (-y[correctness < 1].dot(X[correctn
return J,grad
```

4.2 impact of varying C

Figure 1: Example dataset 1

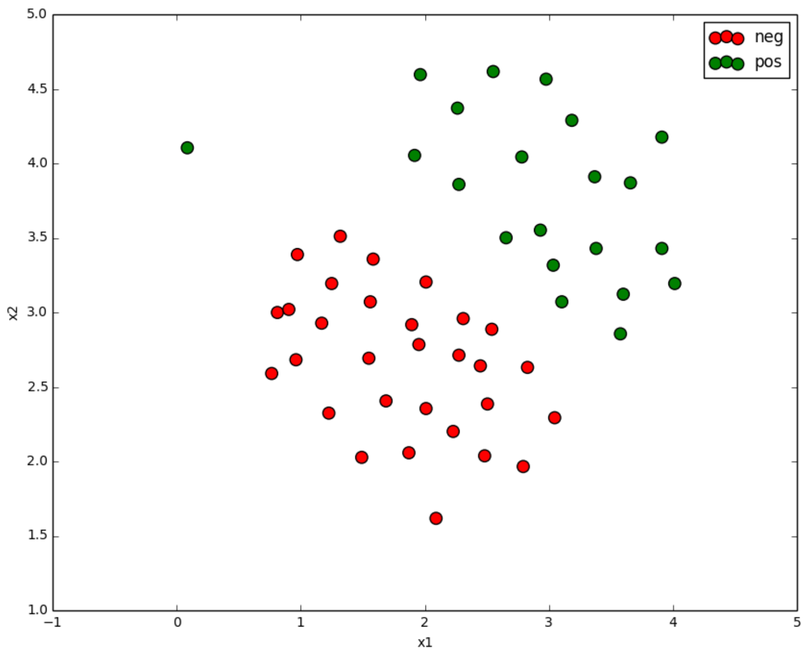


Figure 2: C=100

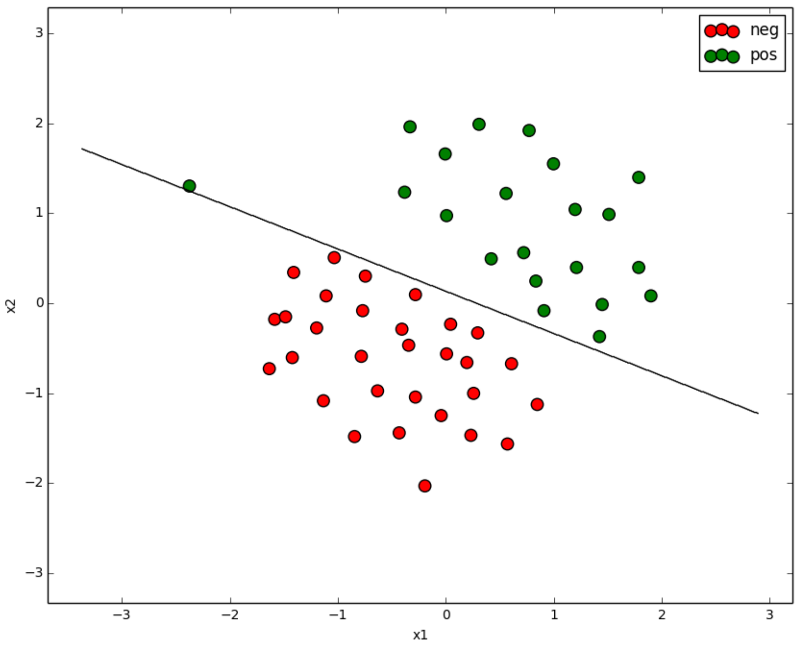


Figure 3: C=10

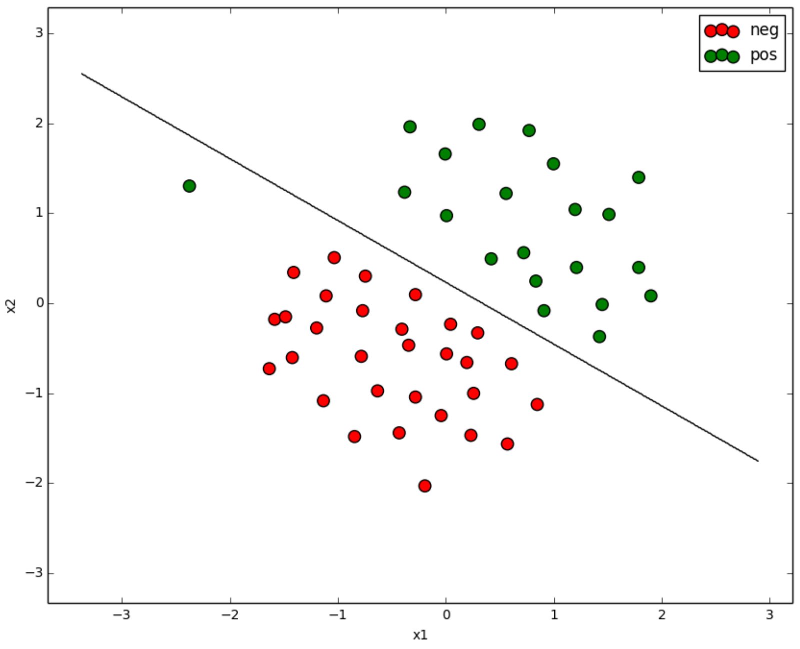
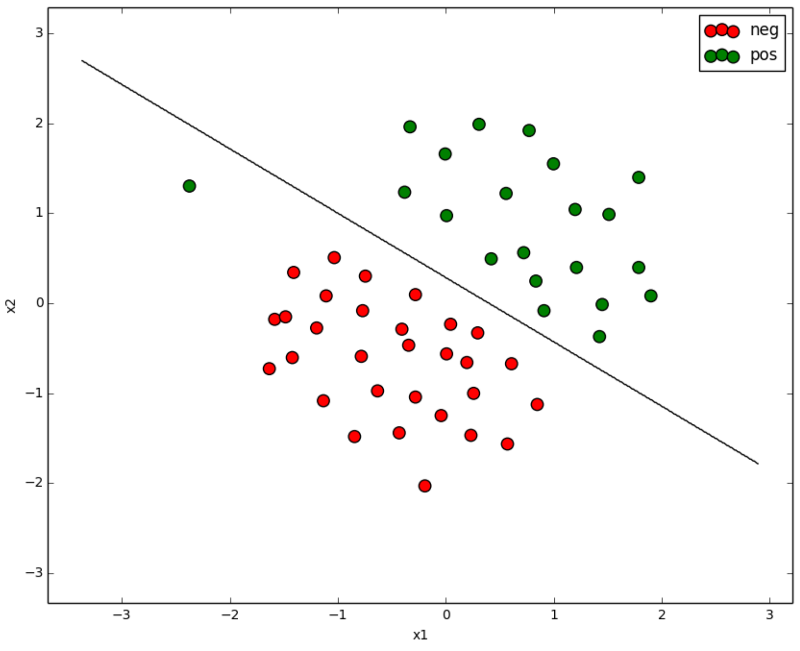


Figure 4: C=5



4.3 Gaussian kernel

```
k = 0
k = np.exp(-np.linalg.norm(x1 - x2) ** 2 / (2 * sigma ** 2))
return k
```

And we get $Guassiankernelvalue(shouldbearound0.324652) = 0.324652467358$

Figure 5: dataset2

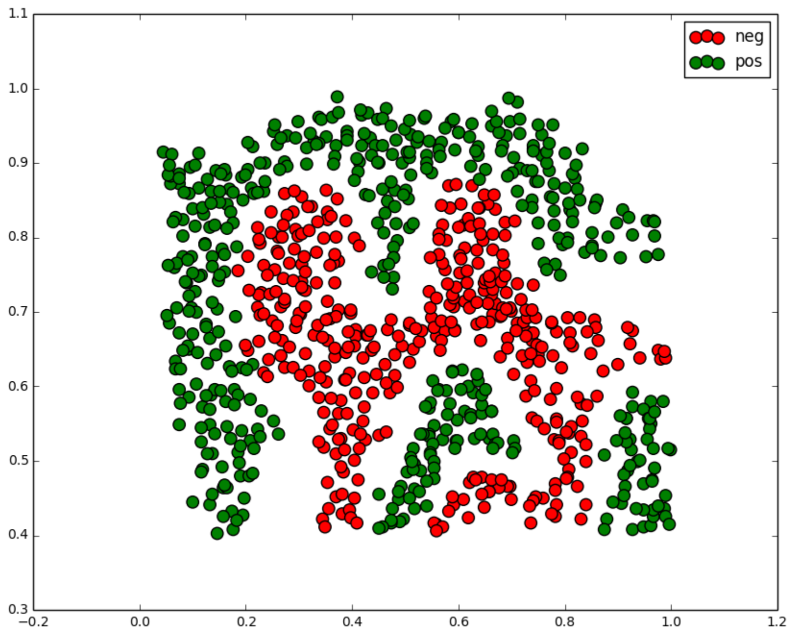
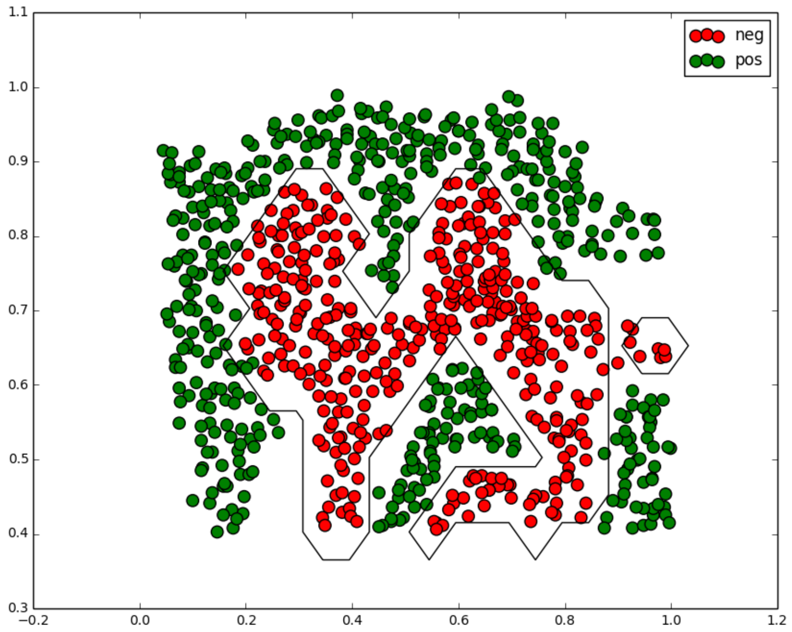


Figure 6: sigma=0.01



4.4 selecting hyper parameters for SVM

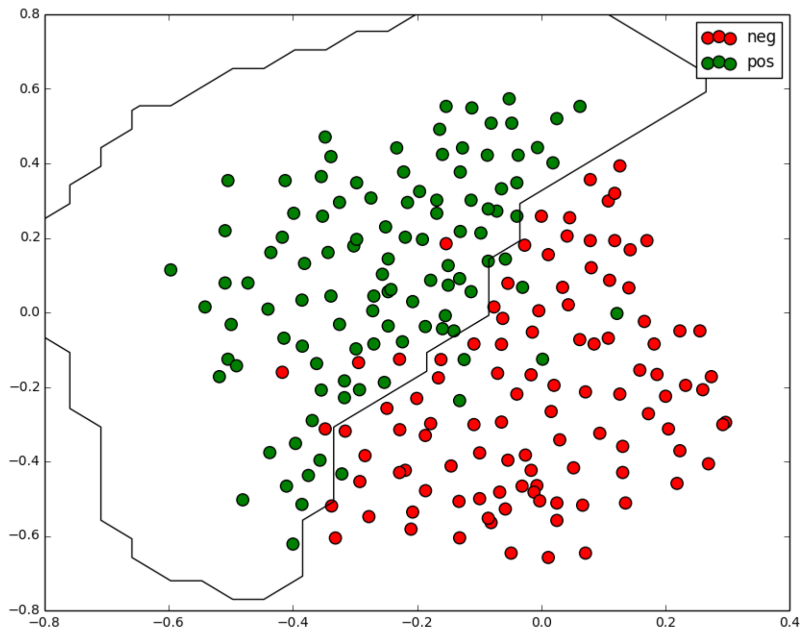
```
best_acc = 0
svm = LinearSVM_twoclass()
for sigma in sigma_vals:
    K = np.array([utils.gaussian_kernel(x1, x2, sigma) for x1 in X for x2
    scaler = preprocessing.StandardScaler().fit(K)
    scaleK = scaler.transform(K)
    KK = np.vstack([np.ones((scaleK.shape[0],)), scaleK]).T

    Kval = np.array([utils.gaussian_kernel(x1, x2, sigma) for x1 in Xval fo
    scalerval = preprocessing.StandardScaler().fit(Kval)
    scaleKval = scalerval.transform(Kval)
    KKval = np.vstack([np.ones((scaleKval.shape[0],)), scaleKval.T]).T

    for C in Cvals:
        svm.theta = np.zeros((KK.shape[1],))
        svm.train(KK, yy, learning_rate=1e-4, reg=C, num_iters=20000, verbo
        yp = svm.predict(KKval)
        acc = metrics.accuracy_score(yyval, yp)
        print C, sigma, acc
        if acc > best_acc:
            best_acc = acc
            best_C = C
            best_sigma = sigma
print 'best', best_C, best_sigma, best_acc
```

I get the best c=0.1, best sigma=0.1

Figure 7: best parameter,c=0.1,sigma=0.1



4.5 Spam Classification with SVM

4.5.1 Training SVM for spam classfication

Most values in Figure 10 is around 1000, except one value is 71,so I didn't scale the numbers. We chose no kernel, penalty parameter $C = 0.1$, learning rate $\alpha = 0.1$, and 10000 iterations. Here we add a cell to calculate the training accuracy. We had a 99.3% training accuracy and a 99.1% test accuracy. Our top 15 predictors of spam are listed below. At a cursory glance, most of these words do look like they belong in a spam email.
`click, remov, our, basenumb, guarante, pleas, you, free, visit, here, nbsp, will, hour, most,dollar`
When choosing a kernel, we found that no kernel resulted in an average of 10% increase in accuracy over a gaussian kernel. Because no kernel had an accuracy of over 98%, we decided to simply use the raw features. We felt that there was no need for any kernelizing of the features due to this high accuracy.
In addition, we also saw an increase in accuracy if we didn't scale the data. We suspect this is because using the raw values put more weight onto words that appeared more frequently. Scaling was not really needed since counting the words caused them to be evaluated on the same scale already.
We ran our parameter sweep on three combinations of kernels/scaling. We found that (no kernel, no scaling) ¿ (no kernel, scaling) ¿ (kernel, scaling). This supported our decision to use no kernels and no scaling.
In sweeping our parameter space, we already knew that we were not going to use a kernel. So the only three parameters we had to sweep over was the penalty parameter, the learning rate, and the number of iterations. We ran across a number of combinations

$$C = [0.1, 0.3, 1, 3, 10, 30]$$

$$\alpha = [1e-2, 3e-2, 1e-1, 3e-1, 1, 3]$$

and found the best accuracy on our 20% validation split was for $C = 0.1$ and $\alpha = 0.1$. The loss plateaued around 7500 iterations, so we found that 10000 iterations was more than enough. Given our 99%+ accuracy on both training and test data, we are satisfied with our results.

5 Support vector machines for multi-class classification

5.1 5A naive version

```
K = theta.shape[1]
m = X.shape[0]
dtheta = np.zeros(theta.shape)
J = 0.0
for i in xrange(m):
  scores = X[i,:].dot(theta)
  correct_class_score = scores[y[i]]
  for j in xrange(K):
    if j == y[i]:
      continue
    margin = max(0,scores[j] - correct_class_score + delta)
    #J += margin
    if margin > 0:
      J += margin
      dtheta[:,j] += X[i]
      dtheta[:,y[i]] -= X[i]

        J /= m
dtheta /=m
# Add regularization to the loss.
J += 0.5 * reg * np.sum(theta * theta)/m

dtheta +=reg*theta/m
```

5.2 5B, vector version

```
J = 0.0
  dtheta = np.zeros(theta.shape) # initialize the gradient as zero
  delta = 1.0
  num_classes = theta.shape[1]
  num_train = X.shape[0]
  scores = X.dot(theta)
  Y = np.zeros(scores.shape)
  for i,row in enumerate(Y):
    row[y[i]] = 1
  Correct_class_scores = np.array( [ [ scores[i][y[i]] ]*num_classes for i
  Margin = scores - Correct_class_scores + ((scores - Correct_class_scores
  X_with_margin_count = np.multiply(X.T , ( Margin > 0).sum(1) ).T

  J += np.sum((Margin>0)*Margin)/num_train
  J += 0.5 * reg * np.sum(theta * theta)/num_train
  dtheta += ( Margin > 0 ).T.dot(X).T/num_train
  dtheta -= (Margin == 0).T.dot(X_with_margin_count).T/num_train
  dtheta += reg*theta
```

It takes 1206.436000s.

5.3 5c Prediction function

```
        Scores = X.dot(self.theta)
        y_pred = np.argmax(Scores, axis = 1)
```

we get training accuracy 0.361245 and validation accuracy 0.362000

5.4 5D tning hyper parameters

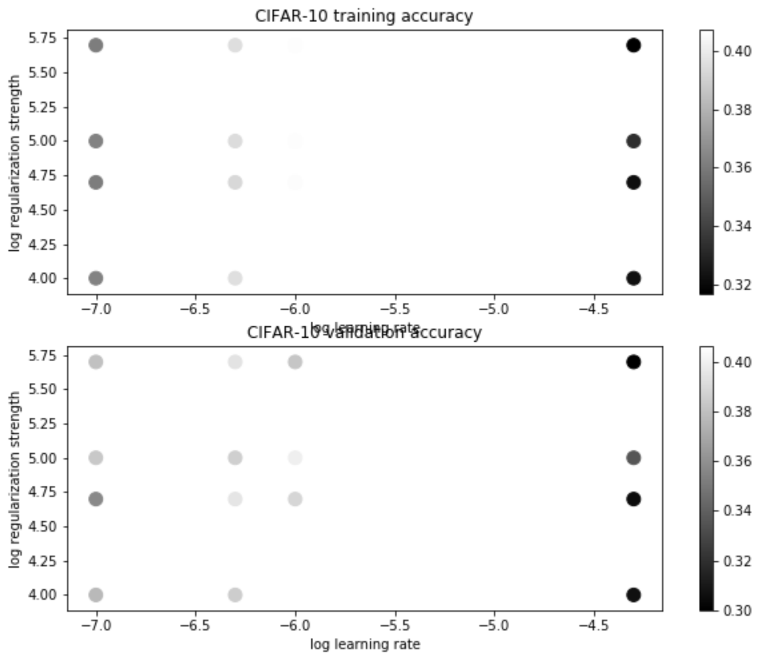
I get an validation accuracy of 0.406000 with learning rate of 1e-6 and reg 1e4.The code is here:

```
for learning_rate in learning_rates:
    for reg in regularization_strengths:
        print("LR",learning_rate,"reg",reg)
        svm = LinearSVM()
        loss_hist = svm.train(X_train, y_train, learning_rate, reg,
                              num_iters=1500, verbose=True)
        y_train_pred = svm.predict(X_train)
        y_val_pred = svm.predict(X_val)
        results[(learning_rate,reg)] = (np.mean(y_train == y_train_pred),n

        if best_val < np.mean(y_val == y_val_pred):
            best_val = np.mean(y_val == y_val_pred)
            best_parameters = { 'LR':learning_rate, 'reg': reg}
```

5.5 5E

Figure 8: accuracy



The SVM takes longer time. And for the accuracy, Softmatrix is slightly better . It is 0.403 better than 0.369.As for the theta I still think Softmatrix is better.As for the running time, I think softmatrix is better. The SVM has a test accuracy of 0.369.

Figure 9: hw3

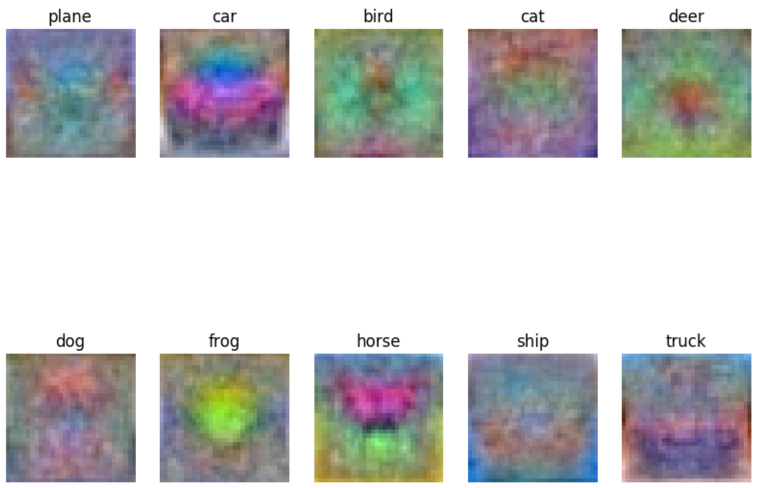
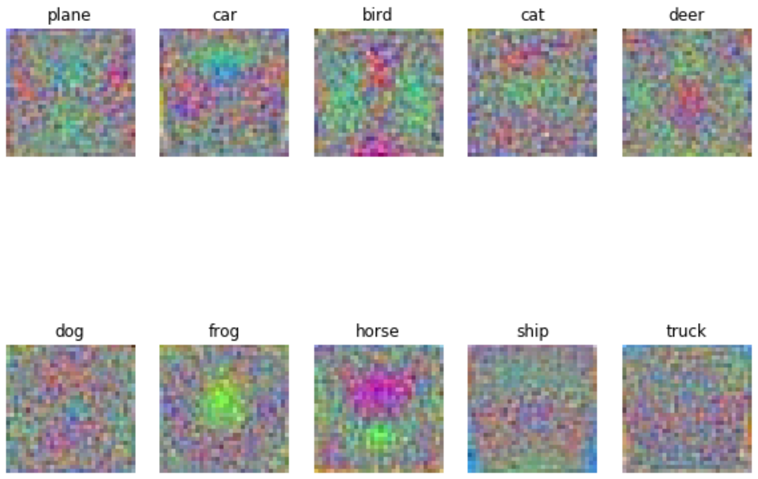


Figure 10: hw4


plane
car
bird
cat
deer
dog
frog
horse
ship
truck