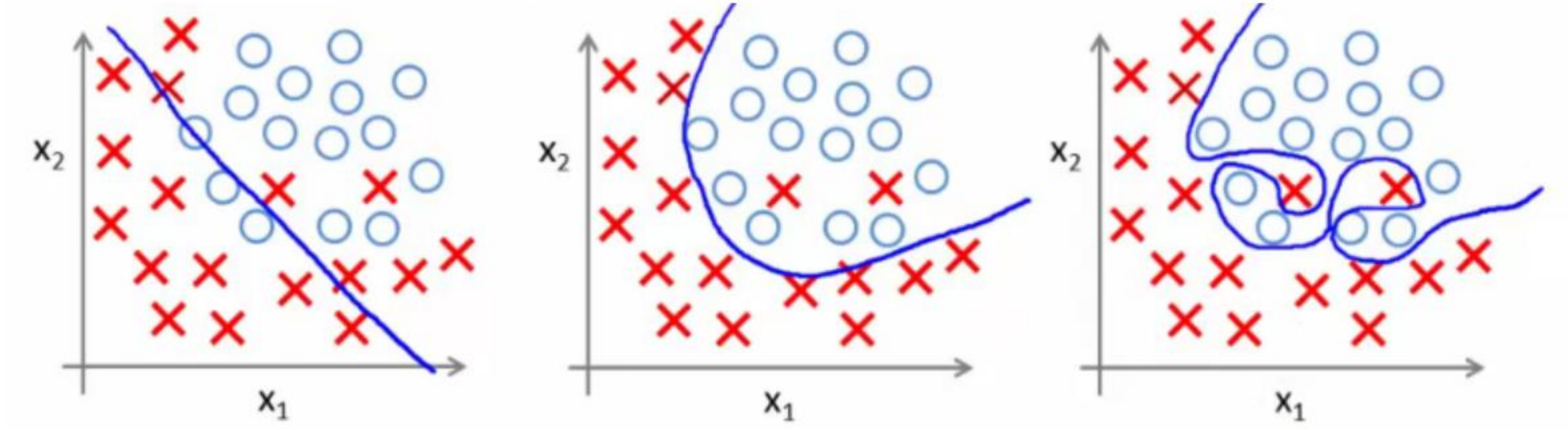
Lecture 5
Machine Learning Adv.

Jaeyun Kang

Overfitting



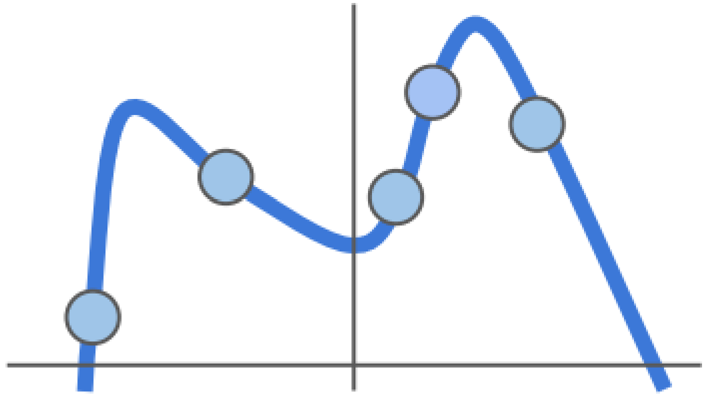
Underfit **Just Right** **Overfit**

Overfitting

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

If loss goes to 0,

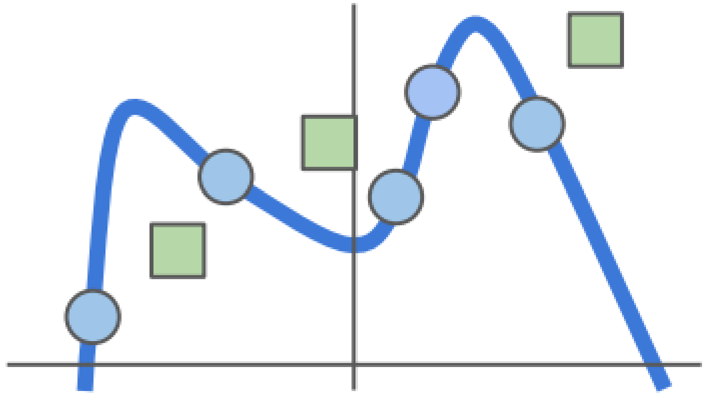
trained model captures
all training data's pattern



Overfitting

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

How about **new data**?

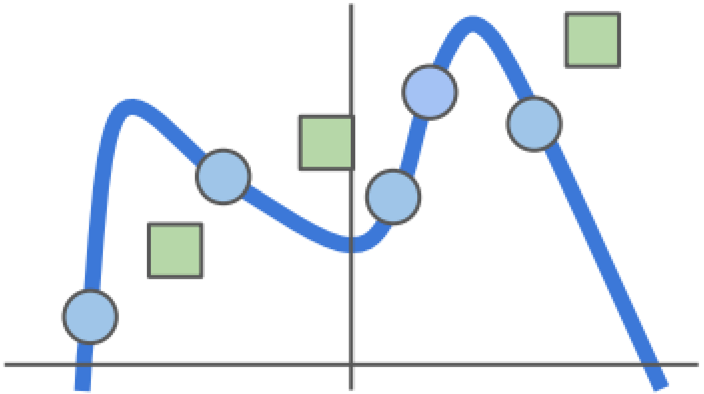


Overfitting

$$L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

If the model is over-fitting the pattern of training data,

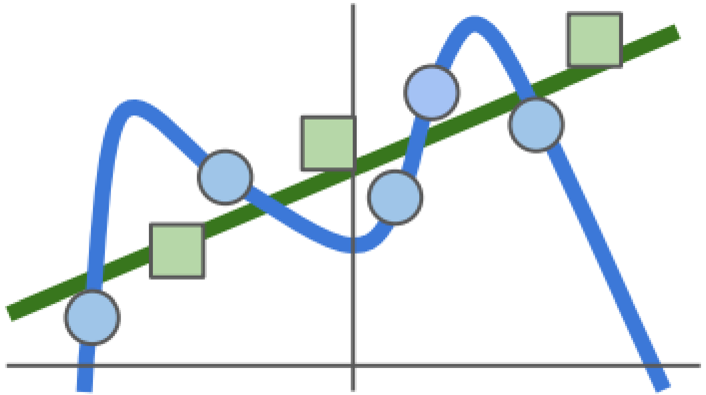
it can't find common data pattern



Overfitting

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

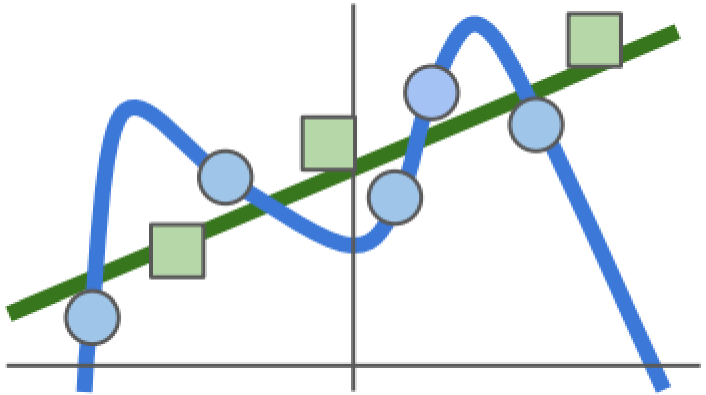
common data pattern



Overfitting

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

How can we train the model to find **common data pattern**?

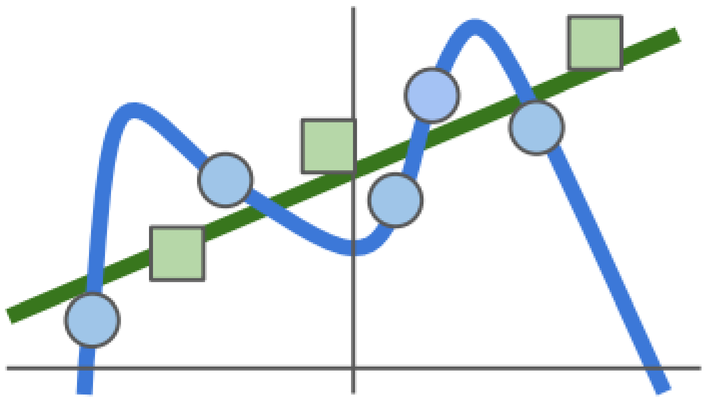


Overfitting – Regularization

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

How can we train the model to find common data pattern?

= Regularization

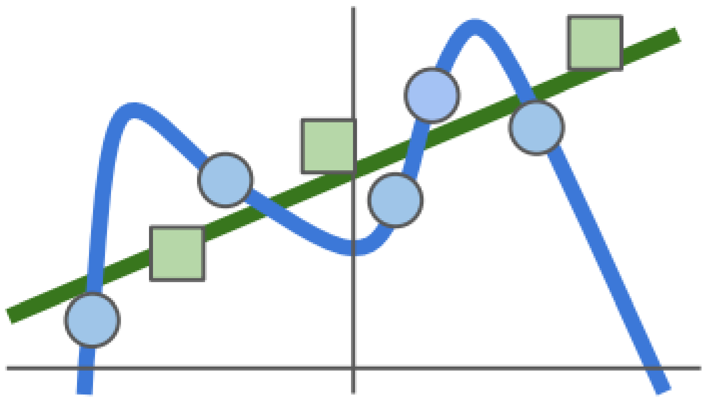


Overfitting - Regularization

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

For **Regularization**,

The **variance** of **W** matrix must be small



Interpreting a Linear Classifier (Review)

$f(x, W) = Wx$

Example trained weights of a linear classifier trained on CIFAR-10:

Interpreting a Linear Classifier (Review)

$$f(x, W) = Wx$$

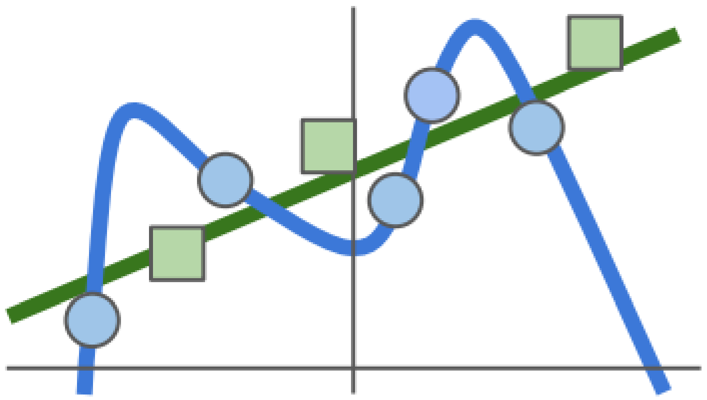
W matrix with small variance
can represent the
common data pattern

Overfitting - Regularization

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

For Regularization,

The variance of W matrix must be small



Overfitting - Regularization

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

For Regularization.

The variance of W matrix must be small

$$x = [1, 1, 1, 1]$$

$$w_1^T x = w_2^T x = 4$$

$$w_1 = [4, 0, 0, 0] \quad ?$$

$$w_2 = [1, 1, 1, 1] \quad ?$$

Overfitting - Regularization

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

For Regularization.

The variance of W matrix must be small

$$x = [1, 1, 1, 1]$$

$$w_1^T x = w_2^T x = 4$$

$$w_1 = [4, 0, 0, 0]$$

$$\boxed{w_2 = [1, 1, 1, 1]}$$

Overfitting - Regularization

$$L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

For Regularization.

The variance of W matrix must be small

How can we adjust

variance of W matrix ?

$$w_1 = [4, 0, 0, 0]$$

$$\boxed{w_2 = [1, 1, 1, 1]}$$

Overfitting - Regularization

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

For **Regularization.**

The **variance** of **W** matrix must be small

How can we adjust variance of W matrix ?

minimize **square sum** of w components!

$$w_1 = [4, 0, 0, 0]$$

$$\boxed{w_2 = [1, 1, 1, 1]}$$

Overfitting - Regularization

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) + \lambda \sum_k \sum_l W_{k,l}^2$$

How can we adjust

variance of W matrix ?

minimize **square sum** of
w components!

$$w_1 = [4, 0, 0, 0]$$

$$\boxed{w_2 = [1, 1, 1, 1]}$$

Overfitting - Regularization

Regularization Strength

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) + \lambda \sum_k \sum_l W_{k,l}^2$$

How can we adjust

variance of W matrix ?

minimize square sum of w components!

$$w_1 = [4, 0, 0, 0]$$

$$\boxed{w_2 = [1, 1, 1, 1]}$$

Evaluation

How we evaluate the model's performance?
(How well trained?)

Evaluation

How we evaluate the model's performance?
(How well trained?)

num(predicted_label == true_label) / total_num * 100 (%)

Evaluation

How we evaluate the model's performance?
(How well trained?)

num(predicted_label == true_label) / total_num * 100 (%)

But, not for training dataset

Evaluation

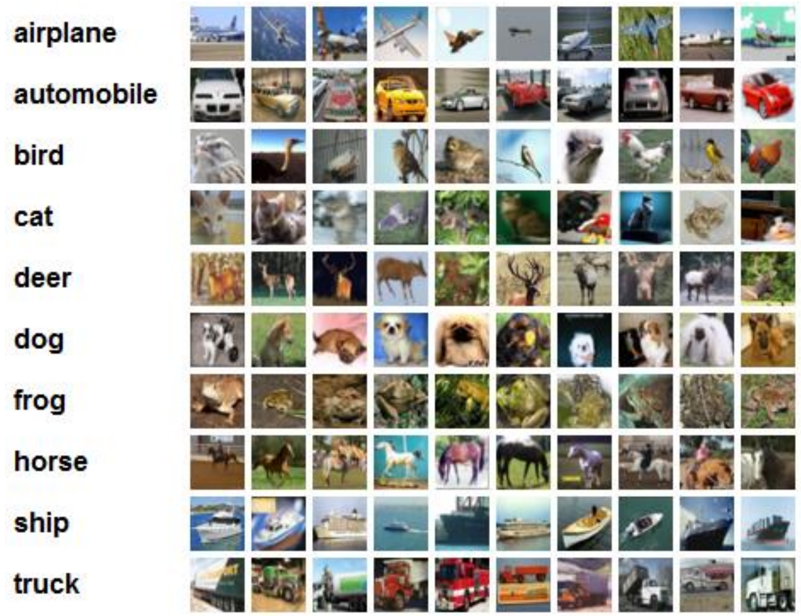
How we evaluate the model's performance?
(How well trained?)

num(predicted_label == true_label) / total_num * 100 (%)

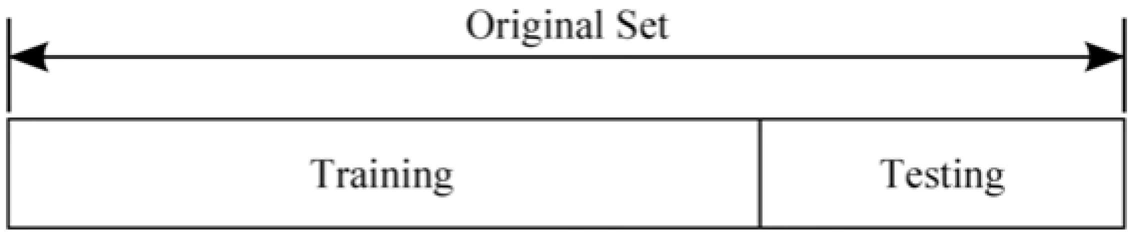
But, not for training dataset

Overfitting may have occurred !

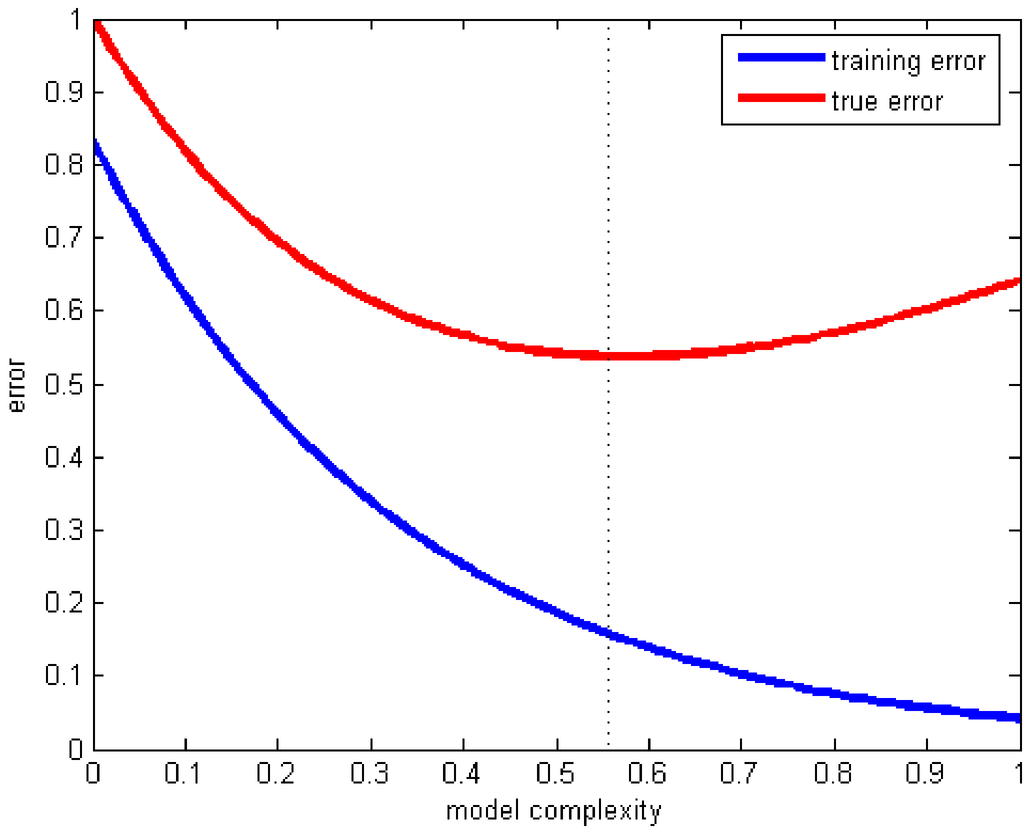
Training Set and Test Set



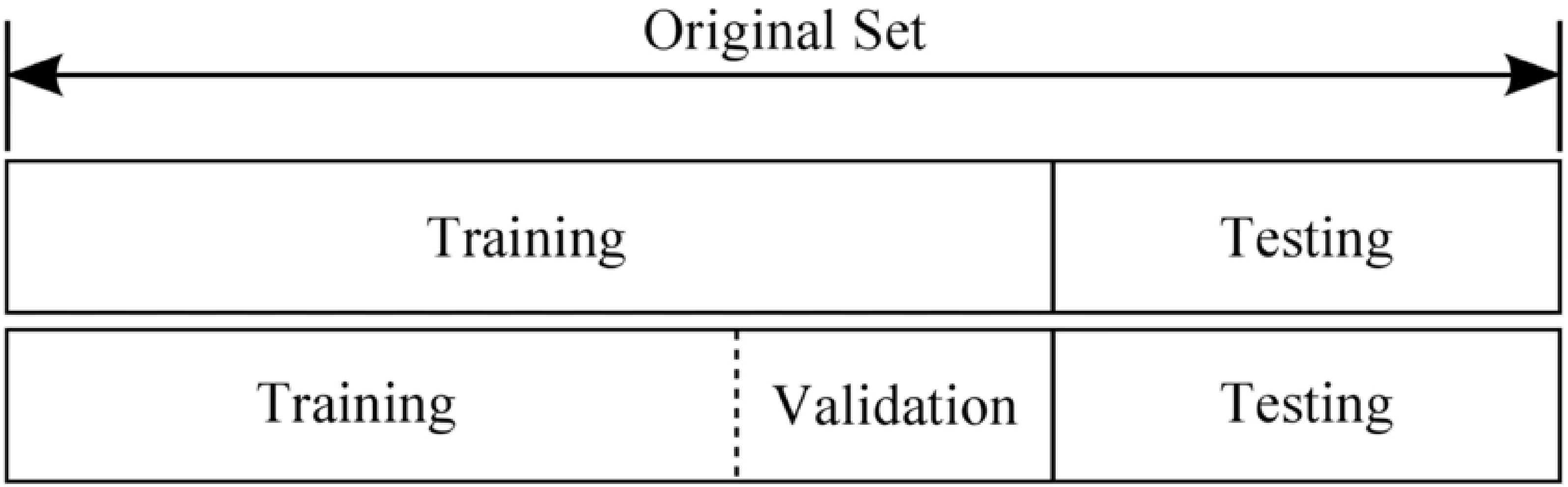
Split Dataset into
Training Set for Training
Test Set for Testing (Predicting)



Overfitting and Training-Test Error



Validation Set



$$\alpha, \lambda$$

learning rate, regularization strength

Hyperparameter Tuning

Tensorflow Practice - MNIST

```
r = randint(0, mnist.test.num_examples - 1)
plt.imshow(mnist.test.images[r:r+1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
print("Prediction: ", sess.run(tf.argmax(scores, 1), feed_dict={x: mnist.test.images[r:r+1]}))
```

```
correct_prediction = tf.equal(tf.argmax(prob, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

Tensorflow Practice - MNIST

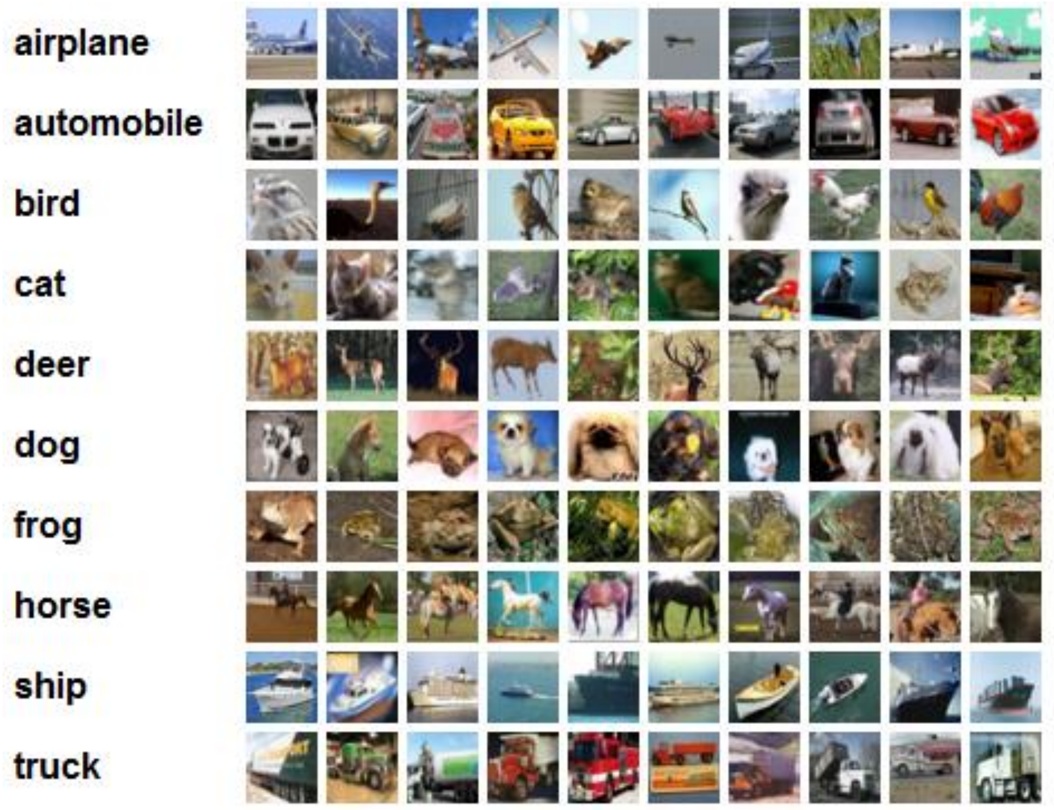
```
r = randint(0, mnist.test.num_examples - 1)
plt.imshow(mnist.test.images[r:r+1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
print("Prediction: ", sess.run(tf.argmax(scores, 1), feed_dict={x: mnist.test.images[r:r+1]}))
```

```
correct_prediction = tf.equal(tf.argmax(prob, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

```
loss:  0.280254434008
loss:  0.277931706214
loss:  0.277339565713
Prediction:  [6]
0.9189
```

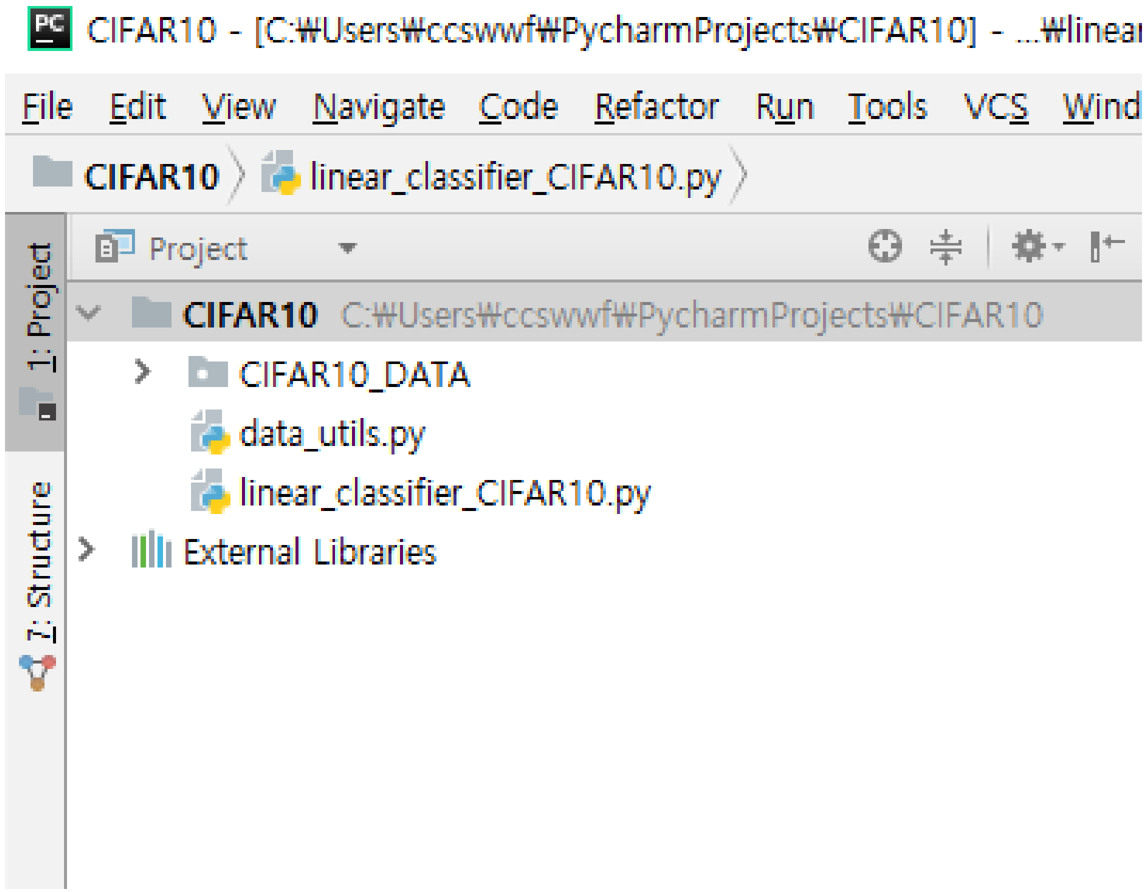
92%

Tensorflow Practice - CIFAR10



60000 32x32 colour images
in 10 classes

Tensorflow Practice - CIFAR10



Tensorflow Practice - CIFAR10

data_utils.py   linear_classifier_CIFAR10.py

```python
from data_utils import load_CIFAR10
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

# Load the raw CIFAR-10 data.
cifar10_dir = 'CIFAR10_DATA/cifar-10-batches-py'
X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

# As a sanity check, we print out the size of the training and test data.
print('Training data shape: ', X_train.shape)
print('Training labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
print('')
```

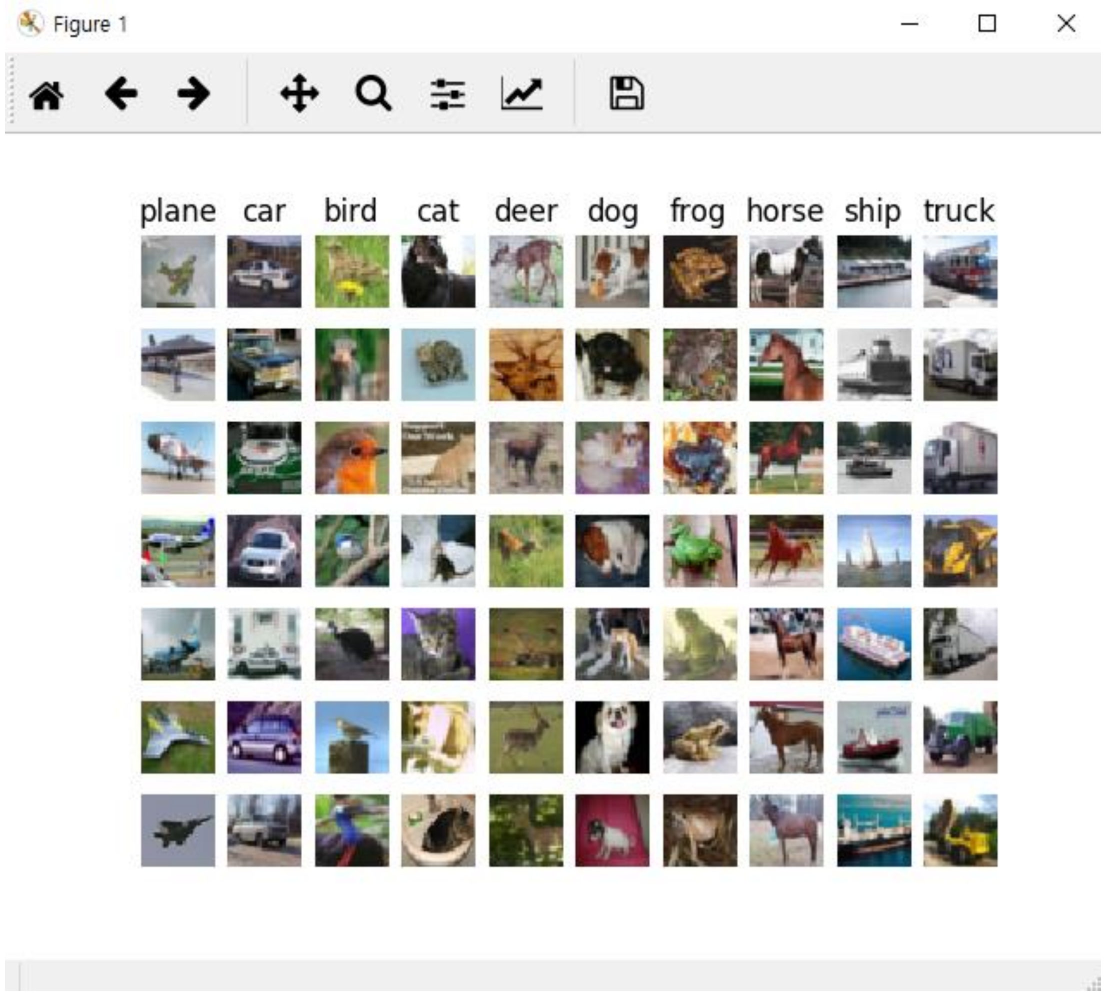
linear_classifier_CIFAR10

```
C:₩Users₩ccswwf₩Anaconda3₩python.exe C:/Use
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

Tensorflow Practice - CIFAR10

```python
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
num_classes = len(classes)
samples_per_class = 7
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```

Tensorflow Practice - CIFAR10



Tensorflow Practice - CIFAR10

```
num_training = 49000
num_test = 1000

# Make training set
mask = range(num_training)
X_train = X_train[mask]
y_train = y_train[mask]

# Make test set
mask = range(num_test)
X_test = X_test[mask]
y_test = y_test[mask]
```

Tensorflow Practice - CIFAR10

```
num_training = 49000
num_test = 1000

# Make training set
mask = range(num_training)
X_train = X_train[mask]
y_train = y_train[mask]

# Make test set
mask = range(num_test)
X_test = X_test[mask]
y_test = y_test[mask]
```

```
# Vectorize X matrix
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
```

Tensorflow Practice - CIFAR10

```python
num_training = 49000
num_test = 1000

# Make training set
mask = range(num_training)
X_train = X_train[mask]
y_train = y_train[mask]

# Make test set
mask = range(num_test)
X_test = X_test[mask]
y_test = y_test[mask]
```

```python
# Vectorize X matrix
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))

# One-hot encoding for train labels
Y_train = np.zeros((y_train.shape[0], 10))
Y_train[np.arange(y_train.shape[0]), y_train] = 1
y_train = Y_train

# One-hot encoding for test labels
Y_test = np.zeros((y_test.shape[0], 10))
Y_test[np.arange(y_test.shape[0]), y_test] = 1
y_test = Y_test
```

Tensorflow Practice - CIFAR10

```
num_training = 49000
num_test = 1000

# Make training set
mask = range(num_training)
X_train = X_train[mask]
y_train = y_train[mask]

# Make test set
mask = range(num_test)
X_test = X_test[mask]
y_test = y_test[mask]
```

```
# Vectorize X matrix
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))

# One-hot encoding for train labels
Y_train = np.zeros((y_train.shape[0], 10))
Y_train[np.arange(y_train.shape[0]), y_train] = 1
y_train = Y_train

# One-hot encoding for test labels
Y_test = np.zeros((y_test.shape[0], 10))
Y_test[np.arange(y_test.shape[0]), y_test] = 1
y_test = Y_test
```

Tensorflow Practice - CIFAR10

```
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

Tensorflow Practice - CIFAR10

```
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

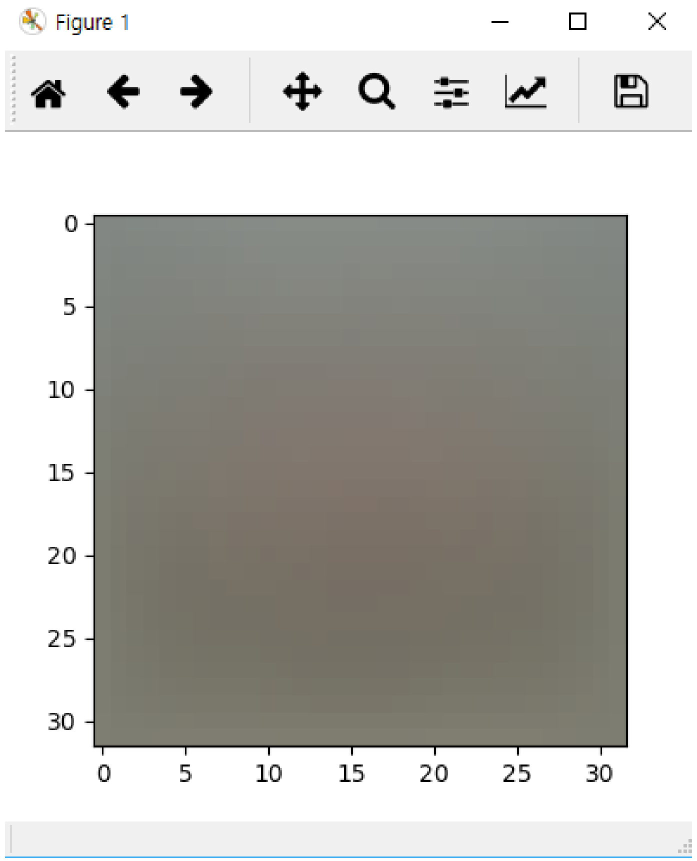
```
Train data shape:  (49000, 3072)
Train labels shape:  (49000, 10)
Test data shape:  (1000, 3072)
Test labels shape:  (1000, 10)
```

Tensorflow Practice - CIFAR10

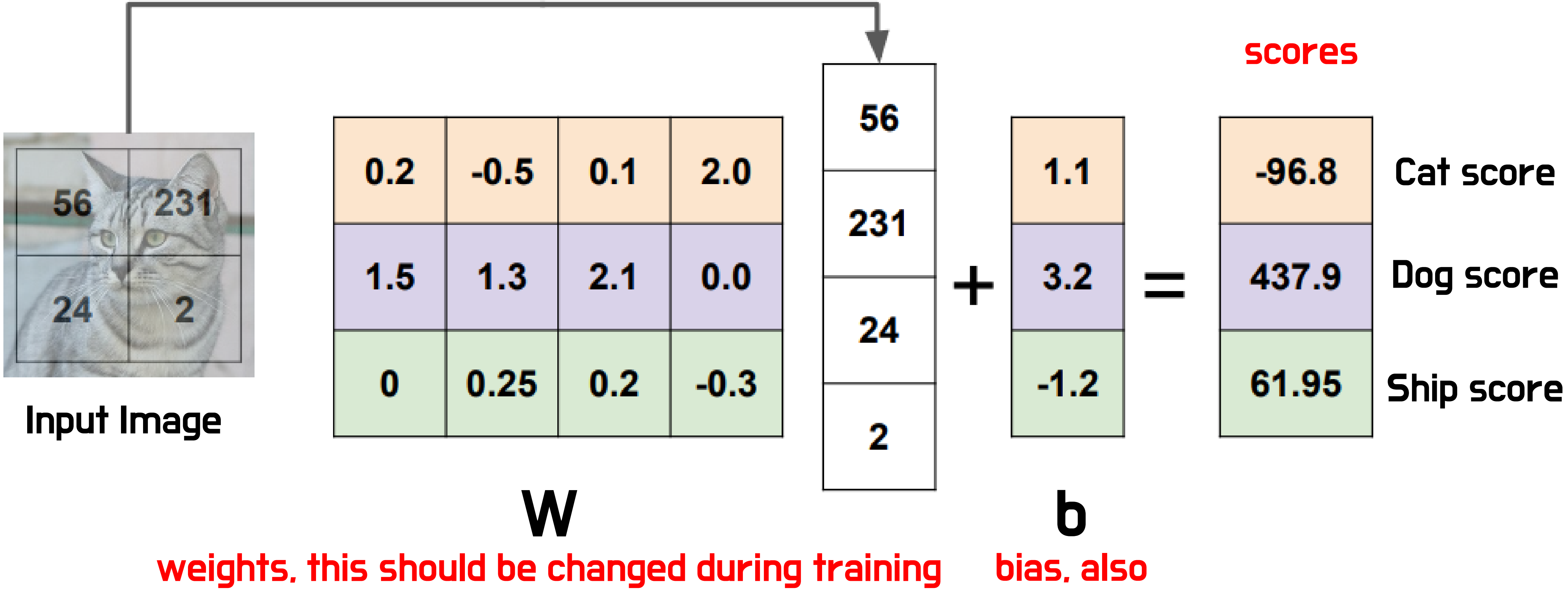
```
# Preprocessing: subtract the mean image (for zero-centered data)
# first: compute the image mean based on the training data
mean_image = np.mean(X_train, axis=0)
plt.figure(figsize=(4,4))
plt.imshow(mean_image.reshape((32,32,3)).astype('uint8')) # visualize the mean image
plt.show()

# second: subtract the mean image from train and test data
X_train -= mean_image
X_test -= mean_image
```

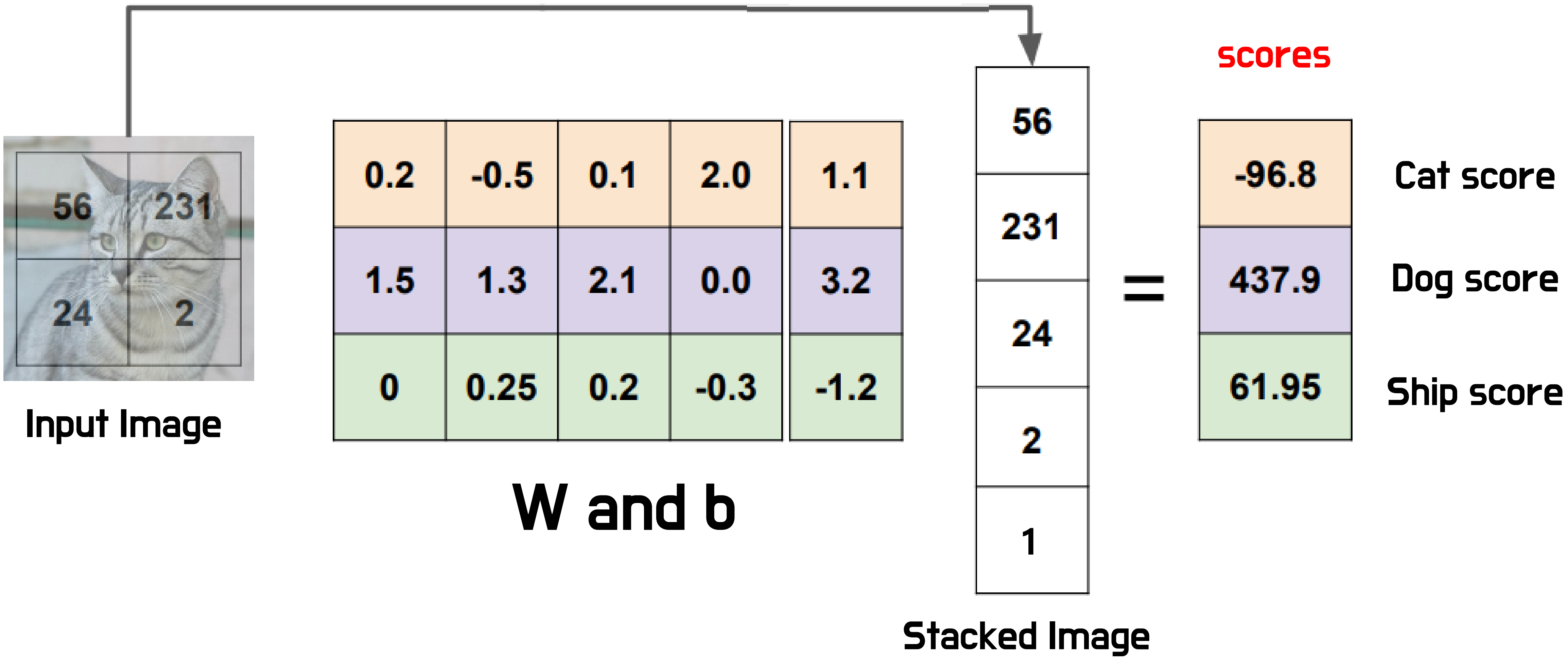
Tensorflow Practice - CIFAR10



Tensorflow Practice - CIFAR10



Tensorflow Practice - CIFAR10



Tensorflow Practice - CIFAR10

```
# third: append the bias dimension of ones (i.e. bias trick) so that our SVM
# only has to worry about optimizing a single weight matrix W.
X_train = np.hstack([X_train, np.ones((X_train.shape[0], 1))])
X_test = np.hstack([X_test, np.ones((X_test.shape[0], 1))])

print(X_train.shape, X_test.shape)
```

```
(49000, 3073) (1000, 3073)
```

Tensorflow Practice - CIFAR10

```
learning_rate = 1e-7
reg_strength = 5e4

x = tf.placeholder(tf.float32, [None, 3073])
y = tf.placeholder(tf.float32, [None, 10])

# Make weight matrix (includes 'b'(bias) vector)
W = tf.Variable(tf.random_normal(shape=(3073, 10), mean=0.0, stddev=0.001, dtype=tf.float32))

scores = tf.matmul(x, W)
prob = tf.nn.softmax(scores)

cross_entrophy_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y, logits = scores))
#cross_entrophy_loss += reg_strength * tf.reduce_sum(tf.square(W))

train = tf.train.GradientDescentOptimizer(learning_rate).minimize(cross_entrophy_loss)
```

Tensorflow Practice - CIFAR10

```python
with tf.Session() as sess:
    sess.run(init)
    loss_hist = []
    for iter in range(1500):
        batch_size = 200
        rand_idx = np.random.choice(X_train.shape[0], batch_size)

        batch_x = X_train[rand_idx]
        batch_y = y_train[rand_idx]

        loss, _ = sess.run([cross_entrophy_loss, train], feed_dict={x:batch_x, y:batch_y})
        loss_hist.append(loss)

        if iter % 100 == 0:
            print("iteration: ", iter, " loss: ", loss)
```

Tensorflow Practice - CIFAR10

```
iteration:  0  loss:  5.59391
iteration:  100  loss:  3.89708
iteration:  200  loss:  3.26234
iteration:  300  loss:  3.50787
iteration:  400  loss:  3.03151
iteration:  500  loss:  2.97595
iteration:  600  loss:  3.18339
iteration:  700  loss:  2.99337
```

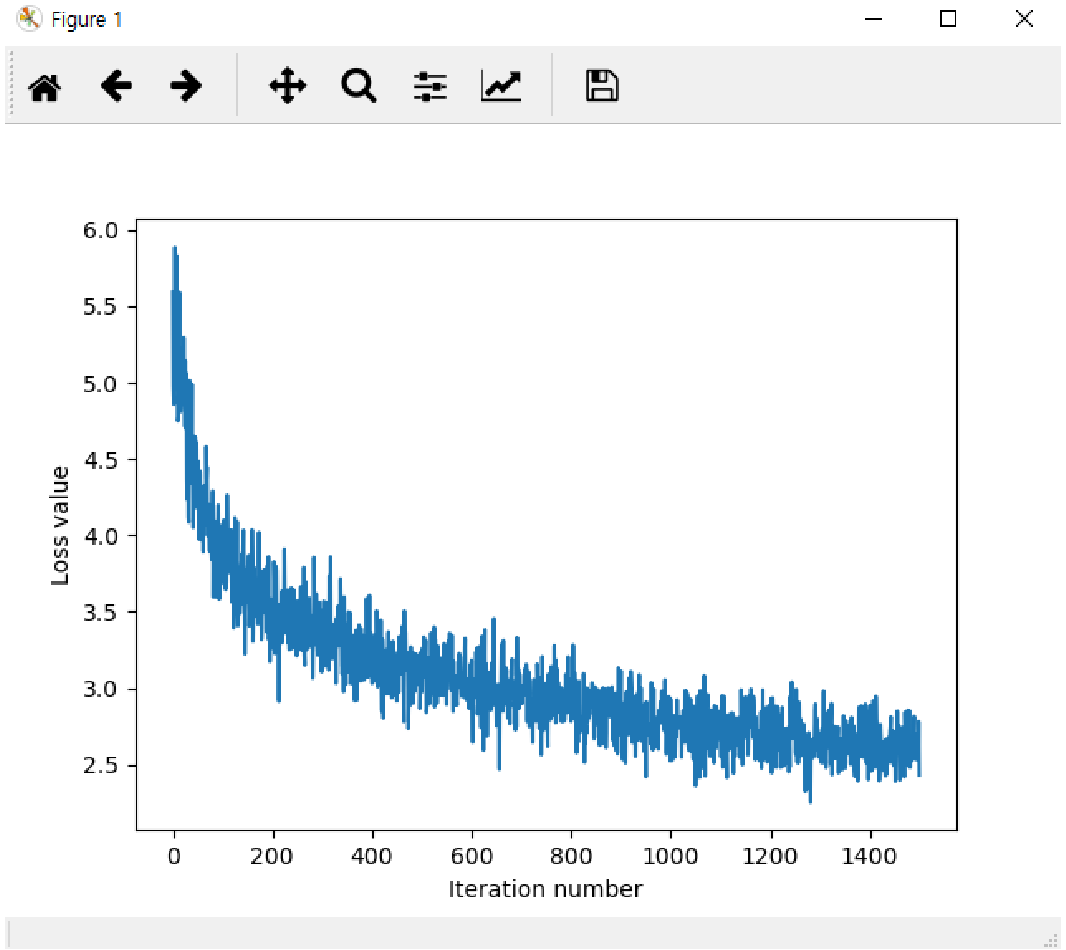
Tensorflow Practice - CIFAR10

```python
    loss, _ = sess.run([cross_entrophy_loss, train], feed_dict={x:batch_x, y:batch_y})
    loss_hist.append(loss)

    if iter % 100 == 0:
        print("iteration: ", iter, " loss: ", loss)

plt.plot(loss_hist)
plt.xlabel('Iteration number')
plt.ylabel('Loss value')
plt.show()
```

Tensorflow Practice - CIFAR10



Tensorflow Practice - CIFAR10

```python
plt.show()

# Evaluate the model with test set
correct_prediction = tf.equal(tf.argmax(prob, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: X_test, y: y_test}))
```

```
iteration:  1200  loss:  2.63387
iteration:  1300  loss:  2.62079
iteration:  1400  loss:  2.88722
0.267
```

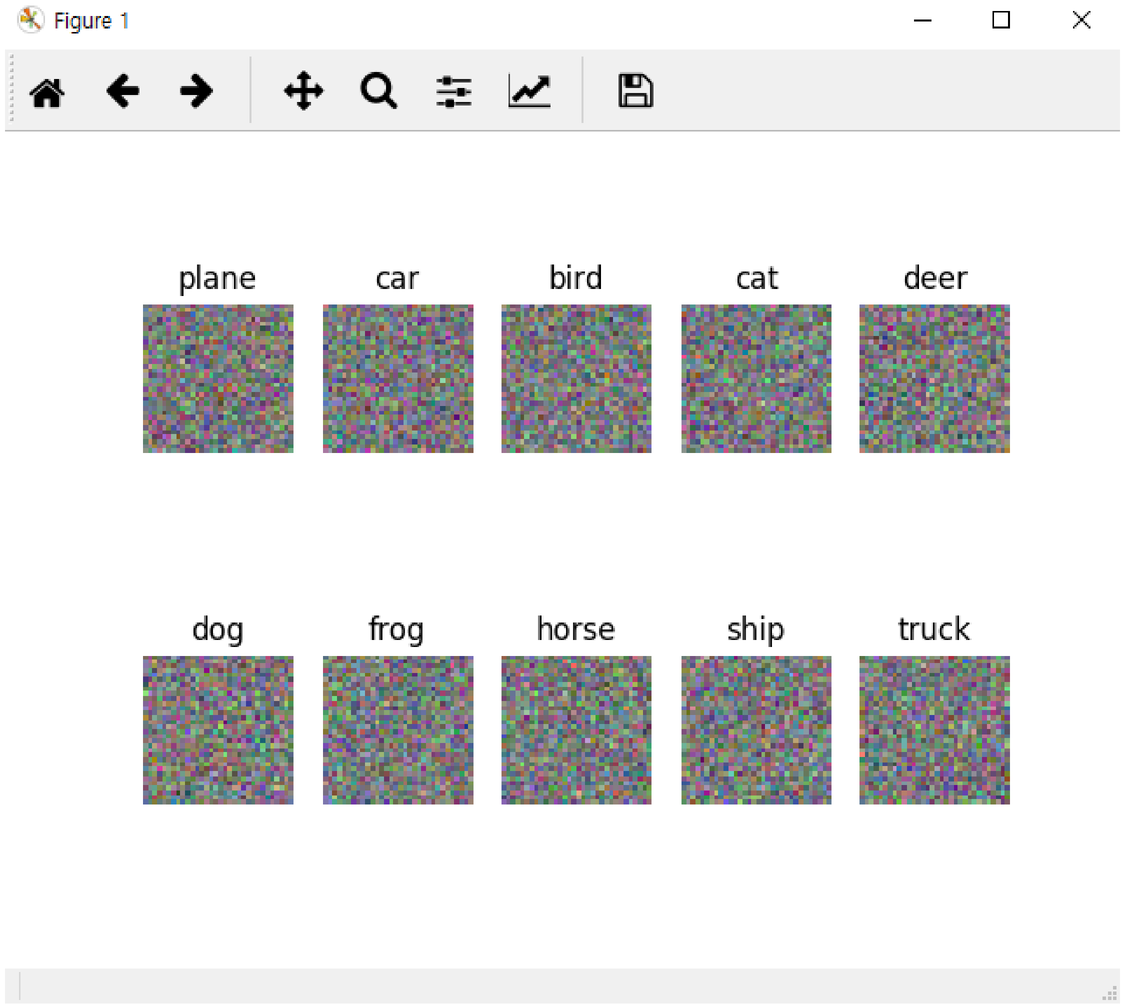
26.7%

Tensorflow Practice - CIFAR10

```
print(sess.run(accuracy, feed_dict={x: X_test, y: y_test}))

# Visualize the trained weight
w = sess.run(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])
plt.show()
```

Tensorflow Practice - CIFAR10



Tensorflow Practice - CIFAR10

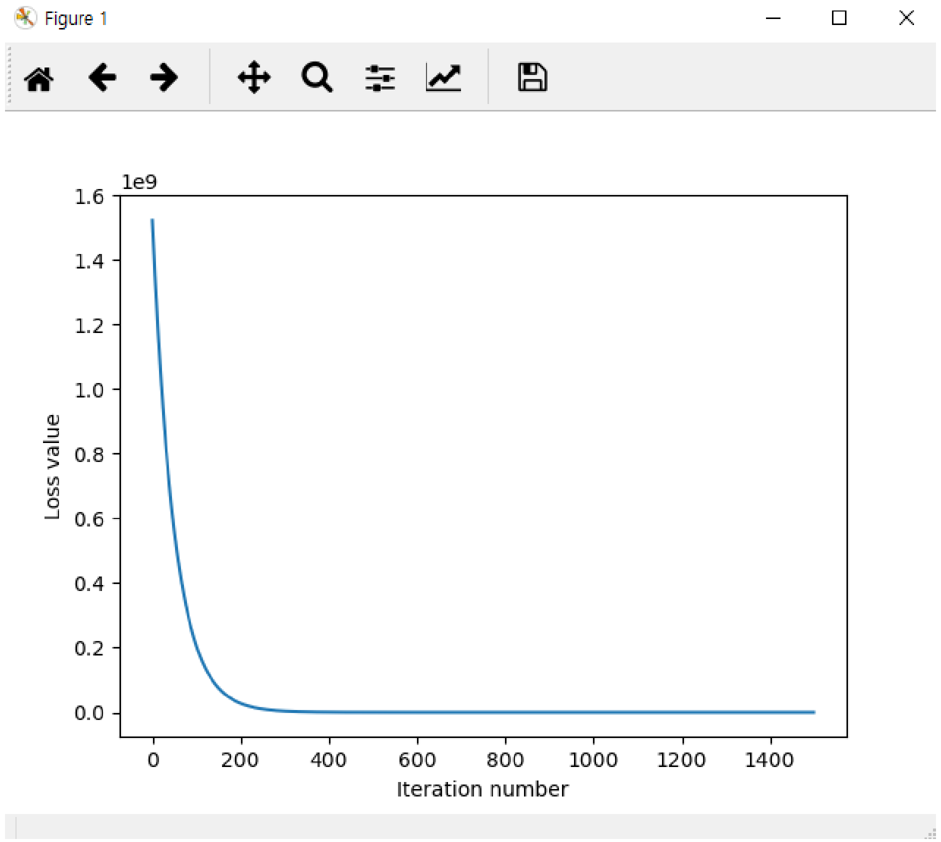
```
cross_entrophy_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y, logits = scores))
cross_entrophy_loss += reg_strength * tf.reduce_sum(tf.square(W))
```

Tensorflow Practice - CIFAR10

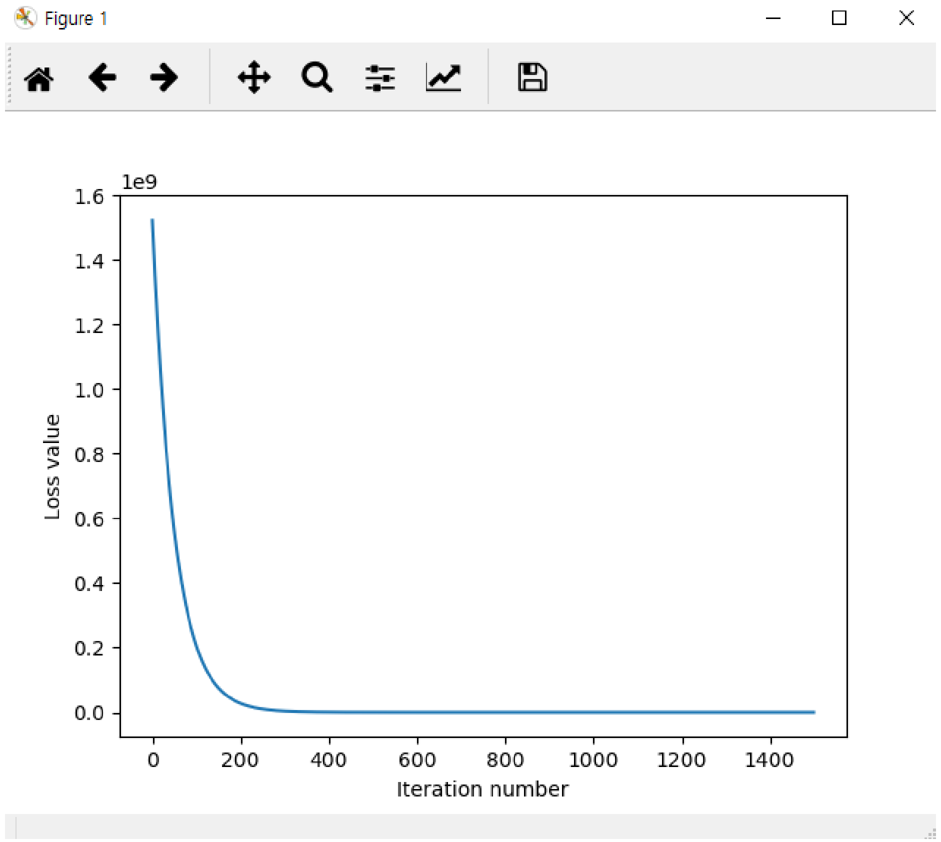
```
cross_entrophy_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y, logits = scores))
cross_entrophy_loss += reg_strength * tf.reduce_sum(tf.square(W))
```

```
iteration:  700  loss:  1179.57
iteration:  800  loss:  159.498
iteration:  900  loss:  23.1094
iteration:  1000  loss:  4.93028
iteration:  1100  loss:  2.53405
iteration:  1200  loss:  2.19926
iteration:  1300  loss:  2.15682
iteration:  1400  loss:  2.16726
```

Tensorflow Practice - CIFAR10



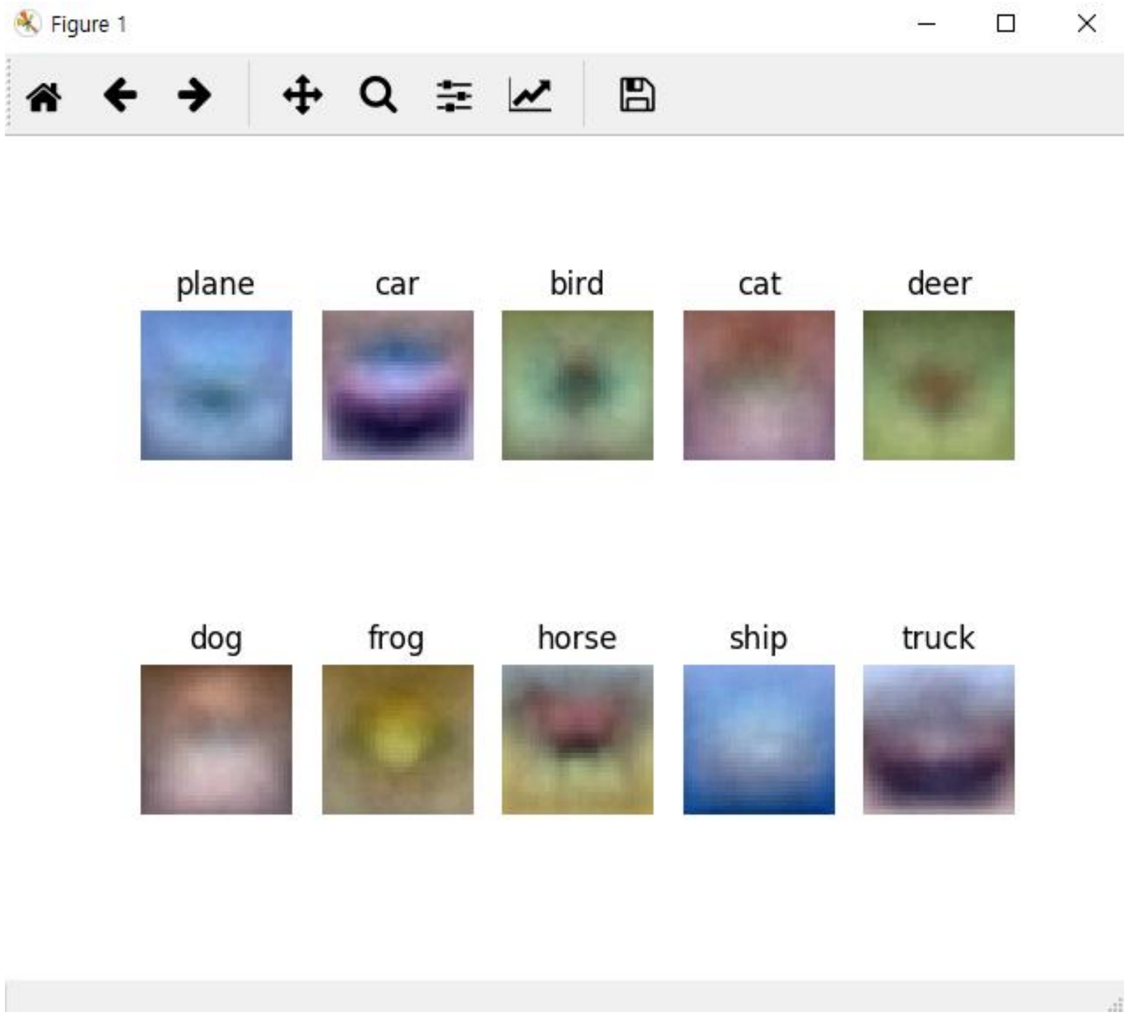
Tensorflow Practice - CIFAR10




```
iteration: 1300 loss: 2.15682
iteration: 1400 loss: 2.16726
0.32
```

32%

Tensorflow Practice - CIFAR10



Next

model is mathematical function!

Image

Array of 32x32x3 numbers
(3072 numbers total)

model

f(x, W)

W

parameters or weights

10 numbers giving
class scores

Next

Linear Classifer

$$f(x, W) = Wx$$

Image

Array of 32x32x3 numbers
(3072 numbers total)

model

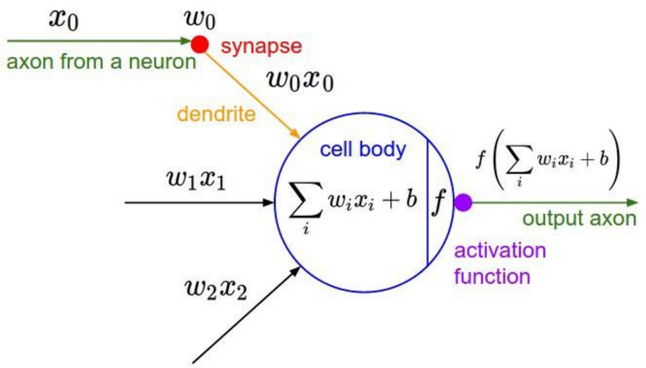
$$f(x, W)$$

10 numbers giving
class scores

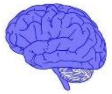
W

parameters or weights

Next - Deep Learning



Neural Network

f(x, W) = 

Image

model

f(x, W)

10 numbers giving
class scores

Array of 32x32x3 numbers
(3072 numbers total)

W
parameters or weights

Question