Homework 1

Yuanteng Chen (3039725444)

1. Least Squares and the Min-norm problem from the Perspective of SVD

(a) How can we solve $\min_w \|Xw-y\|^2$

Solution: it's an ordinary Least Squares problem to solve $\min \|Xw-y\|^2$:

$$\vec{w} = (X^T X)^{-1} X^T \vec{y}$$

(b) plug in the SVD $X = U\Sigma V^T$ and simplify:

Solution: $\vec{w} = (X^T X)^{-1} X^T \vec{y}$

as U and V are orthonormal square matrics,

$$U^T = U^{-1},\ V^T = V^{-1},\ U^T U = U^{-1} U = E$$

$$\therefore \vec{w} = (V\Sigma^T U^T U \Sigma V^T)^{-1} V \Sigma^T U^T \vec{y}$$

$$= (V\Sigma^T E \Sigma V^T)^{-1} V \Sigma^T U^T \vec{y}$$

$$= (V \Sigma^T \Sigma V^T)^{-1} V \Sigma^T U^T \vec{y}$$

$$(V\Sigma^T \Sigma V^T)^{-1} = V \Sigma^{-1} (\Sigma^T)^{-1} V^{-1}$$

$$\therefore \vec{w} = V\Sigma^{-1} (\Sigma^T)^{-1} V^{-1} V \Sigma^T U^T \vec{y}$$

$$= V \Sigma^{-1} (\Sigma^T)^{-1} \Sigma^T U^T \vec{y}$$

$$= V \Sigma^{-1} U^T \vec{y}$$

$\Sigma^{+} = \Sigma^{-1} \to$ an $n \times m$ matrix with the reciprocals of the single value ($\frac{1}{\sigma_i}$) along the diagonal.

(c) $w^{*} = Ay$. What happens if we left-multipy by our matrix A?

Solution: $Ax = V\Sigma^{+}U^{T}U\Sigma V^{T}$

as $U^{T}U = U^{-1}U = E$

$Ax = V\Sigma^{T}\Sigma V^{T}$

assume: $\Sigma = \begin{pmatrix} \sigma_1 & \cdots & 0 \\ 0 & \sigma_2 & \vdots \\ \vdots & \ddots & \sigma_n \\ 0 & \cdots & 0 \end{pmatrix}$ $\Sigma^{+} = \begin{pmatrix} \frac{1}{\sigma_1} & \cdots & \cdots & 0 \\ \vdots & \frac{1}{\sigma_2} & & \vdots \\ 0 & \cdots & \frac{1}{\sigma_n} & 0 \end{pmatrix}$

$n \times m \quad m \times n$

$\Sigma^{+}\Sigma = E \ (n \times n)$

$\therefore Ax = VV^{T} = VV^{-1} = E$

(d) in the case $m < n$. We want to solve $\min \|w\|^2$, $Xw = y$. What is the minimum norm solution?

Solution: $Xw = y$

$X^{T}Xw = X^{T}y$

$$(X^T X)^{-1} X^T X w = (X^T X)^{-1} X^T y$$
$$w = (X^T X)^{+} X^T y$$

But I don't know how to solve $\min \|w\|^2$

(e) Plug in the SVD $X = U \Sigma V^T$ and simplify.

Solution: same as (b).

$$w = V \Sigma^{+} U^T \vec{y}$$

(f) min-norm solution is in the form $w^* = By$

What happens if we right-multify X by matrix B?

Solution:

Same as (c): $XB = U \Sigma V^T V \Sigma^{+} U^T$
$$= U \Sigma \Sigma^{+} U^T$$
$$= U U^T$$
$$= E$$

2. The 5 Interpretations of Ridge Regression

(a) P1: Optimization problem

$$\underset{w}{\operatorname{argmin}} \|y - Xw\|_2^2 + \lambda \|w\|_2^2$$

$X \in R^{n \times d}$, $y \in R^n$ is the target vector of values.

Solution: $\|y - Xw\|_2^2 + \lambda\|w\|_2^2$

$$= y^T y + (Xw)^T (Xw) - 2y \cdot Xw + \lambda w^T w$$

$$= y^T y + w^T X^T X w - 2y \cdot Xw + \lambda w^T w$$

in order to find min

$$\frac{d}{dw}(y^T y + w^T x^T x w - 2y \cdot Xw + \lambda w^T w)$$

$$= 2X^T X w - 2X^T y + 2\lambda w = 0$$

$$X^T X w - X^T y + \lambda w = 0$$

$$(X^T X + \lambda) w = X^T y$$

$$w = (X^T X + \lambda)^{-1} X^T y$$

(b) P2: "Hack of shifting the singular values,

$X = U \Sigma V^T$ be the full SVD of the X

Plug this into the Ridge Regression solution and simplify. What happens to the singular values of $(X^T X + \lambda E)^{-1} X^T$ when $\sigma_i \ll \lambda$ or $\sigma_i \gg \lambda$

Solution: as U and V are both square orthonormal matrices $U^TU = V^TV = E$

$$\therefore W = (X^TX + \lambda E)^{-1} X^T y$$

$$= (V\Sigma^T U^T U \Sigma V^T + \lambda E)^{-1} V \Sigma^T U^T \vec{y}$$

$$= (V\Sigma^T \Sigma V^T + \lambda E)^{-1} V \Sigma^T U^T \vec{y}$$

as $\lambda E = \lambda V E V^T$

$$W = (V\Sigma^T \Sigma V^T + \lambda V E V^T)^{-1} V \Sigma^T U^T \vec{y}$$

$$= (V(\Sigma^T\Sigma + \lambda E)V^T)^{-1} V \Sigma^T U^T \vec{y}$$

$$= (V^T)^{-1} (\Sigma^T\Sigma + \lambda E)^{-1} V^{-1} V \Sigma^T U^T \vec{y}$$

$$= V (\Sigma^T\Sigma + \lambda E) \Sigma^T U^T \vec{y}$$

$$\downarrow$$

$$(d\times n)\cdot(n\times d)$$

$$\downarrow$$

$$d\times d$$

$$\lambda E + \Sigma^T \Sigma = \mathrm{diag}(\sigma_i^2 + \lambda)$$

$$\therefore W = V\, \mathrm{diag}\left(\frac{1}{\sigma_i^2+\lambda}\right) \cdot \Sigma^T U^T \vec{y}$$

$$\Sigma^T = \begin{bmatrix} \sigma_1 & \cdots & & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \sigma_d & \cdots & 0 \end{bmatrix} (d\times n)$$

$$\therefore W = V \left[\mathrm{diag}\left(\frac{\sigma_i}{\sigma_i^2+\lambda}\right), O_{(d, n-d)} \right] U^T \vec{y}$$

in $W = V \left[\mathrm{diag}\left(\frac{\sigma_i}{\sigma_i^2+\lambda}\right), O_{\in(d\times n-d)} \right] U^T \vec{y}$

$\frac{\sigma_i}{\sigma_i^2+\lambda}$ $(i=1,\cdots,d)$ are singular values and λ prevents denominator of any singular value to be 0

$$\begin{cases} \text{When } \sigma_i << \lambda, & \frac{\sigma_i}{\sigma_i^2+\lambda} \approx \frac{\sigma_i}{\lambda} \\ \text{when } \sigma_i >> \lambda, & \frac{\sigma_i}{\sigma_i^2+\lambda} \approx \frac{1}{\sigma_i} \end{cases}$$

(c): P3: Maximum A posteriori (MAP) estimation. Ridge Regression can be viewed as finding the MAP estimate when we apply a prior on the W, we can think of the prior for W as being $N(0, E)$ and view the random Y as $Y = \vec{x}^T W + \sqrt{\lambda} N$,

(noise N is distributed iid as $N(0,1)$)

Vector → $Y = XW + \sqrt{\lambda} N$ (rows of $X = n$)

Show that (1) is the MAP estimate for W given an observation $Y=y$.

Solution: MAP estimate for W is same as

$$\underset{w}{\operatorname{argmax}}\ P(w \mid Y=y)$$

$$= \underset{w}{\operatorname{argmax}} \frac{P(w, y)}{P(y)}$$

$$= \underset{w}{\operatorname{argmax}} \frac{P(y \mid w) \cdot P(w)}{P(y)} = \underset{W}{\operatorname{argmax}} \frac{\prod_{i=1}^{n} P(y_i \mid w) P(w)}{P(y)}$$

$$Y = XW + \sqrt{\lambda}\, N$$

$$y_i = x_i^T W + \sqrt{\lambda}\, N$$

$$N = \frac{y_i - x_i W}{\sqrt{\lambda}} \sim N(0,1)$$

for $X \sim N(0,1) \rightarrow p(z) = \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}}$

$$\therefore p(y_i|w) = \frac{e^{-\frac{1}{2}\left(\frac{y_i - x_i w}{\sqrt{\lambda}}\right)^2}}{\sqrt{2\pi}}$$

$$\therefore \underset{w}{\operatorname{argmax}} \frac{\prod_{i=1}^{n} p(y_i|w)\, p(w)}{p(y)} = \underset{w}{\operatorname{argmax}} \frac{\frac{e^{-\frac{\|w\|^2}{2}}}{\sqrt{2\pi}} \prod_{i=1}^{n} \frac{e^{-\frac{\left(\frac{y_i - x_i w}{\sqrt{\lambda}}\right)^2}{2}}}{\sqrt{2\pi}}}{p(y)}$$

$\because$ $p(y)$, $\sqrt{2\pi}$ will not change with w

$$\therefore \text{MAP} \Leftrightarrow \underset{w}{\operatorname{argmax}}\ e^{-\frac{\|w\|^2}{2}} \prod_{i=1}^{n} e^{-\frac{(y_i - x_i w)^2}{2\lambda}}$$

$$\ln(\text{MAP}) \Leftrightarrow \underset{w}{\operatorname{argmax}} -\frac{\|w\|^2}{2} + \sum_{i=1}^{n}\left(-\frac{(y_i - x_i w)^2}{2\lambda}\right)$$

$$\Leftrightarrow \underset{w}{\operatorname{argmax}} -\frac{\|w\|^2}{2} - \frac{1}{2\lambda} \sum_{i=1}^{n} (y_i - x_i w)^2$$

$$\Leftrightarrow \underset{w}{\operatorname{argmin}}\ \|w\|^2 + \frac{1}{\lambda} \sum_{i=1}^{n} (y_i - x_i w)^2$$

$$\Leftrightarrow \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - x_i w)^2 + \lambda \|w\|^2$$

(d) P4: Fake data

$$\vec{y} = \begin{bmatrix} \vec{y} \\ 0_d \end{bmatrix}, \quad \vec{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I_d \end{bmatrix}$$

where 0_d is the zero vector in R^d and $I_d \in R^{d \times d}$ is the identity matrix:

Solution:

$$\vec{w} = (X^T X)^{-1} X^T \vec{y}$$

$$= \left(\begin{bmatrix} X \\ \sqrt{\lambda} E \end{bmatrix}^T \begin{bmatrix} X \\ \sqrt{\lambda} E \end{bmatrix} \right)^{-1} \begin{bmatrix} X \\ \sqrt{\lambda} E \end{bmatrix}^T \begin{bmatrix} \vec{y} \\ \vec{0} \end{bmatrix}$$

$$= ([X^T \ \sqrt{\lambda} E] \cdot \begin{bmatrix} X \\ \sqrt{\lambda} E \end{bmatrix})^{-1} \begin{bmatrix} X \\ \sqrt{\lambda} E \end{bmatrix}^T \cdot \begin{bmatrix} \vec{y} \\ \vec{0} \end{bmatrix}$$

$$= (X^T X + \lambda E)^{-1} X^T \vec{y}$$

(e) P5: Fake Features

$$\vec{X} = [X, \sqrt{\lambda} E_n]$$

We are interested in the min-norm solution:

$$\underset{\eta}{\operatorname{argmin}} \|\eta\|_2^2 \quad \vec{X} \eta = \vec{y}$$

$$[X, \sqrt{\lambda} E_n] \begin{bmatrix} \vec{w} \\ \vec{J} \end{bmatrix} = \vec{y}$$

$X: n \times d \quad I: n \times n \quad \vec{w}: d \times 1 \quad \vec{J}: n \times 1$

to solve the min-norm:

$$\vec{w} = X^T (X X^T)^{-1} \vec{y}$$

$$\begin{bmatrix} \vec{w} \\ \vec{f} \end{bmatrix} = \begin{bmatrix} X^T \\ \sqrt{\lambda}E \end{bmatrix} \left([X, \sqrt{\lambda}E_n] \begin{bmatrix} X^T \\ \sqrt{\lambda}E \end{bmatrix} \right)^{-1} \vec{y}$$

$$\begin{bmatrix} \vec{w} \\ \vec{f} \end{bmatrix} = \begin{bmatrix} X^T \\ \sqrt{\lambda}E \end{bmatrix} \cdot (XX^T + \lambda E)^{-1} \vec{y}$$

$$\therefore \begin{cases} \vec{w} = X^T (XX^T + \lambda E)^{-1} \vec{y} \\ \vec{f} = \sqrt{\lambda}E \cdot (XX^T + \lambda E)^{-1} \vec{y} \end{cases}$$

(g) $\vec{W_r} = (X^T X + \lambda E)^{-1} X^T \vec{y}$

what happens when $\lambda \to \infty$?

Solution, when $\lambda \to \infty$

$(X^T X + \lambda E) \simeq \mathrm{diag}_n(\lambda)$

$(X^T X + \lambda E)^{-1} \simeq \mathrm{diag}_n(0)$

$\therefore \vec{W_r} \approx \vec{0}$

(h) what happens when $\lambda \to 0$

Solution:

when $\lambda \to 0$.

$\vec{w} = (X^T X)^{-1} X^T \vec{y}$

3. General Case Tikhonov Regularization

Consider the optimization problem:

$$\min_x \|W_1(A\vec{x}-\vec{b})\|_2^2 + \|W_2(\vec{x}-c)\|_2^2$$

W_1 can be viewed as a generic weighting of the residuals and W_2 along with c can be viewed as a general weighting of the parameters.

(a) $$f(x) = \|W_1(A\vec{x}-\vec{b})\|_2^2 + \|W_2(\vec{x}-c)\|_2^2$$

$$= [W_1(A\vec{x}-\vec{b})]^T W_1(A\vec{x}-\vec{b}) + [W_2(\vec{x}-c)]^T W_2(\vec{x}-c)$$

$$= (A\vec{x}-\vec{b})^T W_1^T W_1 (A\vec{x}-\vec{b}) + (\vec{x}-c)^T W_2^T W_2 (\vec{x}-c)$$

$$= x^T A^T W_1^T W_1 A x - 2b^T W_1^T W_1 A x - b^T W_1^T W_1 \vec{b}$$
$$+ x^T W_2^T W_2 x - 2c^T W_2^T W_2 x + c^T W_2^T W_2 c$$

$$\frac{df}{dx} = 2A^T W_1^T W_1 A\vec{x} - 2b^T W_1^T W_1 A + 2W_2^T W_2 \vec{x}$$
$$- 2c^T W_2^T W_2$$

$$\frac{df}{dx} = 0 \Rightarrow (2A^T W_1^T W_1 A + 2W_2^T W_1)\vec{x} = 2b^T W_1^T W_1 A$$
$$+ 2c^T W_2^T W_2$$

$$\therefore (A^T W_1^T W_1 A + W_2^T W_1)\vec{x}$$
$$= (b^T W_1^T W_1 A + c^T W_2^T W_2)$$

$$\vec{x} = (A^T W_1^T W_1 A + W_2^T W_1)^{-1}(b^T W_1^T W_1 A + c^T W_2^T W_2)$$

(b) construct an appropriate matrix C and vector d that allows to rewrite this problem as

$$\min_x \|Cx-d\|^2$$

and use the OLS solution $(x^* = (C^TC)^{-1}C^Td)$

Solution,

$$\min_x \|W_1(Ax-b)\|_2^2 + \|W_2(x-c)\|_2^2$$

① the first part:

$\|W_1(Ax-b)\|_2^2$

$\rightarrow C_1 = [W_1A]$, $d_1 = [W_1b]$

$\|C_1x-d_1\|^2 = \|W_1(Ax-b)\|_2^2$

② the second part:

$\|W_2(x-c)\|_2^2$

$\rightarrow C_2 = [W_2]$ $d_2 = [W_2c]$

$\|C_2x-d_2\|_2^2 = \|W_2(x-c)\|_2^2$

$$\therefore C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} W_1A \\ W_2 \end{bmatrix}$$

$$d = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} W_1b \\ W_2c \end{bmatrix}$$

$$\therefore x^* = (C^TC)^{-1}C^Td$$

$$= ([A^TW_1^T, W_2^T]\begin{bmatrix} W_1A \\ W_2 \end{bmatrix})^{-1}[A^TW_1^T, W_2^T]\begin{bmatrix} W_1b \\ W_2c \end{bmatrix}$$

$$= (A^TW_1^TW_1A + W_2^TW_2)^{-1}[A^TW_1^TW_1b + W_2^TW_2c]$$

(c) choose a W_1, W_2 and c such that this reduces to the simple case of ridge regression that you've seen in the previous problem,

$$x^* = (A^T A + \lambda E)^{-1} A^T b$$

Solution:

$$x = (A^T W_1^T W_1 A + W_2^T W_2)^{-1} (A^T W_1^T W_1 b + W_2^T W_2 c)$$

$$\Downarrow$$

$$x^* = (A^T A + \lambda E)^{-1} A^T b$$

$$\therefore \begin{cases} W_1^T W_1 = E \\ W_2^T W_2 = \lambda E \\ W_2^T W_2 c = 0 \end{cases} \Rightarrow \begin{cases} W_1 = E \\ W_2 = \sqrt{\lambda} E \\ c = \vec{0} \end{cases}$$

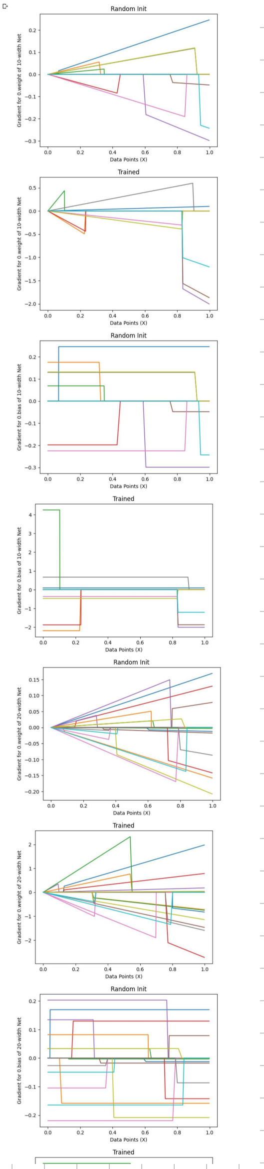
4. Coding Fully Connected Networks.

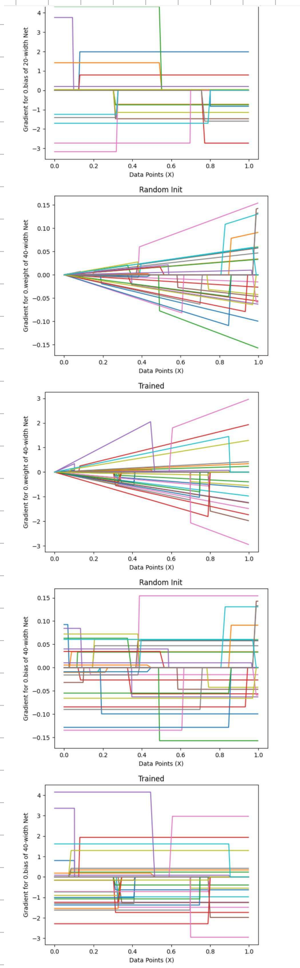
(a) ① higher learning rate is more suitable for three-layer network and we need to low down the learning-rate when training a five layers network.

(b) of course training five layers network costs more time.

5. Visualizing features from local linearization of neutral nets.

(a)



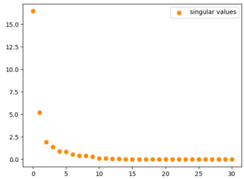


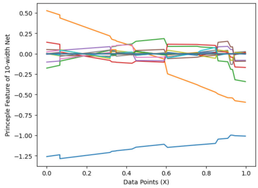
(b) SVD for feature matrix

During training, we can imagine that we have a generalized linear model with a feature matrix corre
corresponding to each learnable parameter. We know from our analysis of gradient descent, that th
corresponding to this feature matrix are important.

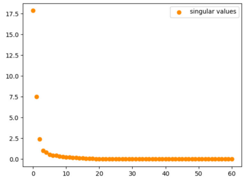
(b).

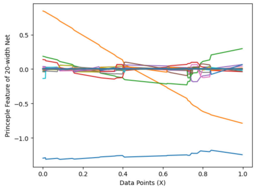
Width 10



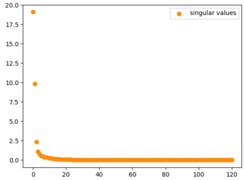


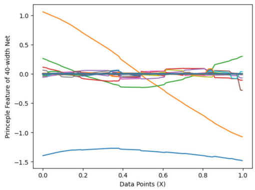
Width 20





Width 40





(c) Two-layer Network

Augment the jupyter notebook to add a second hidden layer of the same size as the first hidden layer, ful

6. Homework Process and Study Group.

(b) Yujie Zhao 3039725470

(c) 15 hours

Setup Environment

If you are working on this assignment using Google Colab, please execute the codes below.

Alternatively, you can also do this assignment using a local anaconda environment (or a Python virtualenv). Please clone the GitHub repo by running `git clone https://github.com/gonglinyuan/cs182hw1.git` and refer to `README.md` for further details.

In [ ]:
```
#@title Mount your Google Drive

import os
from google.colab import drive
drive.mount('/content/gdrive')
```

In [ ]:
```
#@title Set up mount symlink

DRIVE_PATH = '/content/gdrive/My\ Drive/cs182hw1_sp23'
DRIVE_PYTHON_PATH = DRIVE_PATH.replace('\\', '')
if not os.path.exists(DRIVE_PYTHON_PATH):
  %mkdir $DRIVE_PATH

## the space in `My Drive` causes some issues,
## make a symlink to avoid this
SYM_PATH = '/content/cs182hw1'
if not os.path.exists(SYM_PATH):
  !ln -s $DRIVE_PATH $SYM_PATH
```

In [ ]:
```
#@title Install dependencies

!pip install numpy==1.21.6 imageio==2.9.0 matplotlib==3.2.2
```

In [ ]:
```
#@title Clone homework repo

%cd $SYM_PATH
if not os.path.exists("cs182hw1"):
  !git clone https://github.com/gonglinyuan/cs182hw1.git
%cd cs182hw1
```

In [ ]:
```
#@title Download datasets

%cd deeplearning/datasets/
!bash ./get_datasets.sh
%cd ../..
```

In [13]:
```
#@title Configure Jupyter Notebook

import matplotlib
%matplotlib inline
%load_ext autoreload
%autoreload 2
```

executed in 161ms, finished 17:43:28 2023-08-31

Fully-Connected Neural Nets

In this notebook we will implement fully-connected networks using a modular approach. For each layer we will implement a `forward` and a `backward` function. The `forward` function will receive inputs, weights, and other parameters and will return both an output and a `cache` object storing data needed for the backward pass, like this:

```python
def layer_forward(x, w):
  """ Receive inputs x and weights w """
  # Do some computations ...
  z = # ... some intermediate value
  # Do some more computations ...
  out = # the output

  cache = (x, w, z, out) # Values we need to compute gradients

  return out, cache
```

The backward pass will receive upstream derivatives and the `cache` object, and will return gradients with respect to the inputs and weights, like this:

```python
def layer_backward(dout, cache):
  """
  Receive derivative of loss with respect to outputs and cache,
  and compute derivative with respect to inputs.
  """
  # Unpack cache values
  x, w, z, out = cache

  # Use values in cache to compute derivatives
  dx = # Derivative of loss with respect to x
  dw = # Derivative of loss with respect to w

  return dx, dw
```

After implementing a bunch of layers this way, we will be able to easily combine them to build classifiers with different architectures.

In [14]:
```
# As usual, a bit of setup

import os
import time
import numpy as np
import matplotlib.pyplot as plt
from deeplearning.classifiers.fc_net import *
from deeplearning.data_utils import get_CIFAR10_data
from deeplearning.gradient_check import eval_numerical_gradient, eval_numerical_gradient_array
from deeplearning.solver import Solver

plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython

def rel_error(x, y):
    """ returns relative error """
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

executed in 105ms, finished 17:43:33 2023-08-31

In [15]:
```
# Lddoad the (preprocessed) CIFAR10 data.

data = get_CIFAR10_data()
for k, v in data.items():
    print('%s: ' % k, v.shape)
```

executed in 3.54s, finished 17:43:38 2023-08-31

```
deeplearning/datasets/cifar-10-batches-py\data_batch_1
deeplearning/datasets/cifar-10-batches-py\data_batch_2
deeplearning/datasets/cifar-10-batches-py\data_batch_3
deeplearning/datasets/cifar-10-batches-py\data_batch_4
deeplearning/datasets/cifar-10-batches-py\data_batch_5
deeplearning/datasets/cifar-10-batches-py\test_batch
X_train:  (49000, 3, 32, 32)
y_train:  (49000,)
X_val:  (1000, 3, 32, 32)
y_val:  (1000,)
X_test:  (1000, 3, 32, 32)
y_test:  (1000,)
```

Affine layer: forward

Open the file `deeplearning/layers.py` and implement the `affine_forward` function.

Once you are done you can test your implementaion by running the following:

In [24]:
```python
# Test the affine_forward function

num_inputs = 2
input_shape = (4, 5, 6)
output_dim = 3

input_size = num_inputs * np.prod(input_shape)
weight_size = output_dim * np.prod(input_shape)

x = np.linspace(-0.1, 0.5, num=input_size).reshape(num_inputs, *input_shape)
w = np.linspace(-0.2, 0.3, num=weight_size).reshape(np.prod(input_shape), output_dim)
b = np.linspace(-0.3, 0.1, num=output_dim)

out, _ = affine_forward(x, w, b)
correct_out = np.array([[ 1.49834967,  1.70660132,  1.91485297],
                        [ 3.25553199,  3.5141327,   3.77273342]])

# Compare your output with ours. The error should be around 1e-9.
print('Testing affine_forward function:')
print('difference: ', rel_error(out, correct_out))
```

executed in 116ms, finished 00:01:17 2023-09-01

```
Testing affine_forward function:
difference:  9.769848888397517e-10
```

Affine layer: backward

Now implement the `affine_backward` function and test your implementation using numeric gradient checking.

In [26]:
```
# Test the affine_backward function

x = np.random.randn(10, 2, 3)
w = np.random.randn(6, 5)
b = np.random.randn(5)
dout = np.random.randn(10, 5)

dx_num = eval_numerical_gradient_array(lambda x: affine_forward(x, w, b)[0], x, dout)
dw_num = eval_numerical_gradient_array(lambda w: affine_forward(x, w, b)[0], w, dout)
db_num = eval_numerical_gradient_array(lambda b: affine_forward(x, w, b)[0], b, dout)

_, cache = affine_forward(x, w, b)
dx, dw, db = affine_backward(dout, cache)
# The error should be around 1e-10
print('Testing affine_backward function:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

executed in 108ms, finished 00:02:31 2023-09-01

```
Testing affine_backward function:
dx error:  1.0073197343977312e-09
dw error:  5.284631764367503e-11
db error:  1.714610111690866e-10
```

ReLU layer: forward

Implement the forward pass for the ReLU activation function in the `relu_forward` function and test your implementation using the following:

In [34]:
```
# Test the relu_forward function

x = np.linspace(-0.5, 0.5, num=12).reshape(3, 4)

out, _ = relu_forward(x)
correct_out = np.array([[ 0.,          0.,          0.,          0.,         ],
                        [ 0.,          0.,          0.04545455,  0.13636364,],
                        [ 0.22727273,  0.31818182,  0.40909091,  0.5,        ]])

# Compare your output with ours. The error should be around 1e-8
print('Testing relu_forward function:')
print('difference: ', rel_error(out, correct_out))
```

executed in 98ms, finished 00:27:03 2023-09-01

```
Testing relu_forward function:
difference:  4.9999997980221 58e-08
```

ReLU layer: backward

Now implement the backward pass for the ReLU activation function in the `relu_backward` function and test your implementation using numeric gradient checking. Note that the ReLU activation is not differentiable at 0, but typically we don't worry about this and simply assign either 0 or 1 as the derivative by convention.

In [41]:
```
x = np.random.randn(10, 10)
dout = np.random.randn(*x.shape)

dx_num = eval_numerical_gradient_array(lambda x: relu_forward(x)[0], x, dout)

_, cache = relu_forward(x)
dx = relu_backward(dout, cache)

# The error should be around 1e-12
print('Testing relu_backward function:')
print('dx error: ', rel_error(dx_num, dx))
```

executed in 108ms, finished 00:41:51 2023-09-01

```
Testing relu_backward function:
dx error:  3.2755955148731643e-12
```

"Sandwich" layers

There are some common patterns of layers that are frequently used in neural nets. For example, affine layers are frequently followed by a ReLU nonlinearity. To make these common patterns easy, we define several convenience layers in the file `deeplearning/layer_utils.py` .

For now take a look at the `affine_relu_forward` and `affine_relu_backward` functions, and run the following to numerically gradient check the backward pass:

In [42]:
```
from deeplearning.layer_utils import affine_relu_forward, affine_relu_backward

x = np.random.randn(2, 3, 4)
w = np.random.randn(12, 10)
b = np.random.randn(10)
dout = np.random.randn(2, 10)

out, cache = affine_relu_forward(x, w, b)
dx, dw, db = affine_relu_backward(dout, cache)

dx_num = eval_numerical_gradient_array(lambda x: affine_relu_forward(x, w, b)[0], x, dout)
dw_num = eval_numerical_gradient_array(lambda w: affine_relu_forward(x, w, b)[0], w, dout)
db_num = eval_numerical_gradient_array(lambda b: affine_relu_forward(x, w, b)[0], b, dout)

print('Testing affine_relu_forward:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

executed in 150ms, finished 00:52:37 2023-09-01

```
Testing affine_relu_forward:
dx error:  1.6189981732725837e-11
dw error:  6.634449034685112e-11
db error:  1.8928932101850778e-11
```

Loss layers: Softmax and SVM

Here we provide two loss functions that we will use to train our deep neural networks. You should understand how they work by looking at the implementations in `deeplearning/layers.py`.

You can make sure that the implementations are correct by running the following:

In [51]:
```
num_classes, num_inputs = 10, 50
x = 0.001 * np.random.randn(num_inputs, num_classes)
print(x.shape)
y = np.random.randint(num_classes, size=num_inputs)
print(y.shape)

dx_num = eval_numerical_gradient(lambda x: svm_loss(x, y)[0], x, verbose=False)
loss, dx = svm_loss(x, y)

# Test svm_loss function. Loss should be around 9 and dx error should be 1e-9
print('Testing svm_loss:')
print('loss: ', loss)
print('dx error: ', rel_error(dx_num, dx))

dx_num = eval_numerical_gradient(lambda x: softmax_loss(x, y)[0], x, verbose=False)
loss, dx = softmax_loss(x, y)

# Test softmax_loss function. Loss should be 2.3 and dx error should be 1e-8
print('\nTesting softmax_loss:')
print('loss: ', loss)
print('dx error: ', rel_error(dx_num, dx))
```

executed in 316ms, finished 01:43:41 2023-09-01

```
(50, 10)
(50,)
Testing svm_loss:
loss:  9.000977026840296
dx error:  1.4021566006651672e-09

Testing softmax_loss:
loss:  2.302683249916458
dx error:  8.766618947477147e-09
```

Two-layer network

Open the file `deeplearning/classifiers/fc_net.py` and complete the implementation of the `TwoLayerNet` class. This class will serve as a model for the other networks you will implement in this assignment, so read through it to make sure you understand the API. Run the cell below to test your implementation.

In [53]:
```
N, D, H, C = 3, 5, 50, 7
X = np.random.randn(N, D)
y = np.random.randint(C, size=N)

std = 1e-2
model = TwoLayerNet(input_dim=D, hidden_dim=H, num_classes=C, weight_scale=std)

print ('Testing initialization ... ')
W1_std = abs(model.params['W1'].std() - std)
b1 = model.params['b1']
W2_std = abs(model.params['W2'].std() - std)
b2 = model.params['b2']
assert W1_std < std / 10, 'First layer weights do not seem right'
assert np.all(b1 == 0), 'First layer biases do not seem right'
assert W2_std < std / 10, 'Second layer weights do not seem right'
assert np.all(b2 == 0), 'Second layer biases do not seem right'

print ('Testing test-time forward pass ... ')
model.params['W1'] = np.linspace(-0.7, 0.3, num=D*H).reshape(D, H)
model.params['b1'] = np.linspace(-0.1, 0.9, num=H)
model.params['W2'] = np.linspace(-0.3, 0.4, num=H*C).reshape(H, C)
model.params['b2'] = np.linspace(-0.9, 0.1, num=C)
X = np.linspace(-5.5, 4.5, num=N*D).reshape(D, N).T
scores = model.loss(X)
correct_scores = np.asarray(
  [[11.53165108,  12.2917344,   13.05181771,  13.81190102,  14.57198434, 15.33206765,  16.09215096],
   [12.05769098,  12.74614105,  13.43459113,  14.1230412,   14.81149128, 15.49994135,  16.18839143],
   [12.58373087,  13.20054771,  13.81736455,  14.43418138,  15.05099822, 15.66781506,  16.2846319 ]])
scores_diff = np.abs(scores - correct_scores).sum()
assert scores_diff < 1e-6, 'Problem with test-time forward pass'

print ('Testing training loss (no regularization)')
y = np.asarray([0, 5, 1])
loss, grads = model.loss(X, y)
correct_loss = 3.4702243556
assert abs(loss - correct_loss) < 1e-10, 'Problem with training-time loss'

model.reg = 1.0
loss, grads = model.loss(X, y)
correct_loss = 26.5948426952
assert abs(loss - correct_loss) < 1e-10, 'Problem with regularization loss'
```

```
for reg in [0.0, 0.7]:
    print ('Running numeric gradient check with reg = ', reg)
    model.reg = reg
    loss, grads = model.loss(X, y)

    for name in sorted(grads):
        f = lambda _: model.loss(X, y)[0]
        grad_num = eval_numerical_gradient(f, model.params[name], verbose=False)
        print ('%s relative error: %.2e' % (name, rel_error(grad_num, grads[name])))
```

executed in 712ms, finished 10:54:12 2023-09-01

```
Testing initialization ...
Testing test-time forward pass ...
Testing training loss (no regularization)
Running numeric gradient check with reg =  0.0
W1 relative error: 1.83e-08
W2 relative error: 3.37e-10
b1 relative error: 8.01e-09
b2 relative error: 2.53e-10
Running numeric gradient check with reg =  0.7
W1 relative error: 2.53e-07
W2 relative error: 2.85e-08
b1 relative error: 1.35e-08
b2 relative error: 1.97e-09
```

Solver

Following a modular design, for this assignment we have split the logic for training models into a separate class from the models themselves.

Open the file `deeplearning/solver.py` and read through it to familiarize yourself with the API. After doing so, use a `Solver` instance to train a `TwoLayerNet` that achieves at least `50%` accuracy on the validation set.

In [54]:

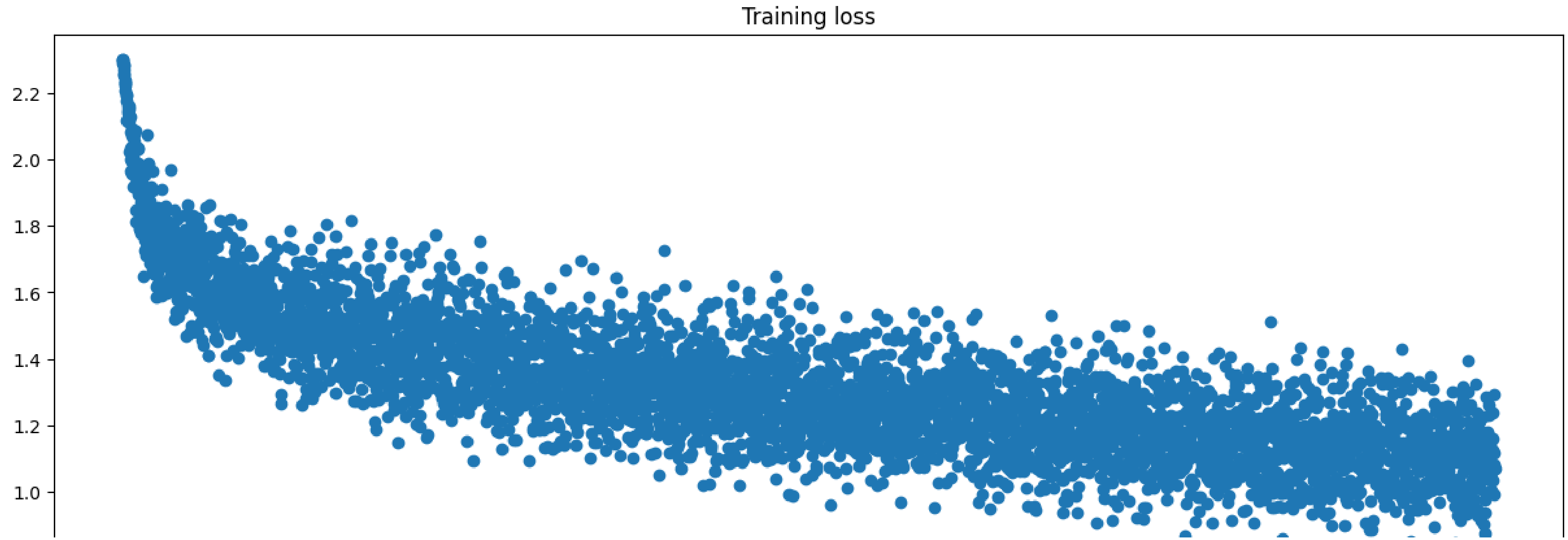
```
model = TwoLayerNet()
solver = None

##############################################################################
# TODO: Use a Solver instance to train a TwoLayerNet that achieves at least  #
# 50% accuracy on the validation set.                                        #
##############################################################################
optim_config = {}
optim_config['learning_rate'] = 0.001
solver = Solver(model, data, optim_config=optim_config, lr_decay=0.9, num_epochs=10)
solver.train()
##############################################################################
#                             END OF YOUR CODE                               #
##############################################################################
```

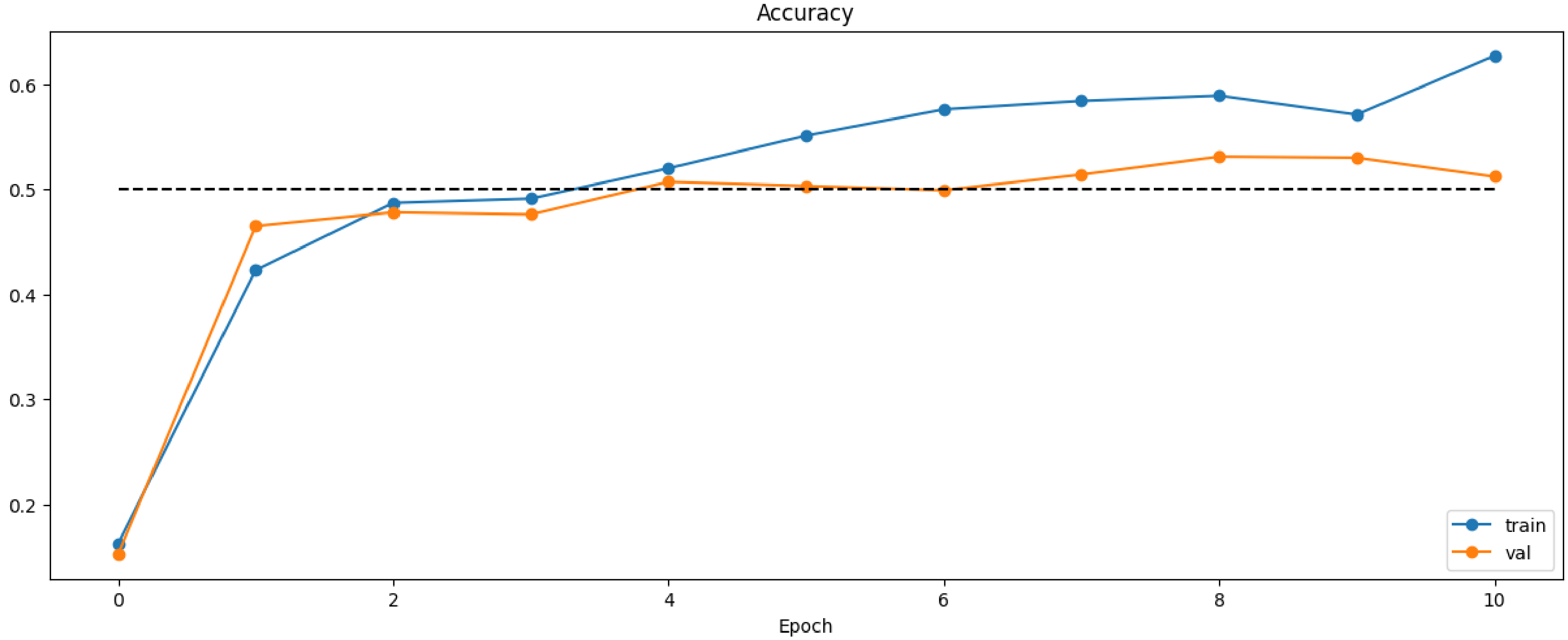
executed in 2m 28s, finished 11:14:00 2023-09-01

```
(Iteration 1 / 4900) loss: 2.302200
(Epoch 0 / 10) train acc: 0.163000; val_acc: 0.153000
(Iteration 11 / 4900) loss: 2.231917
(Iteration 21 / 4900) loss: 2.142166
(Iteration 31 / 4900) loss: 1.999730
(Iteration 41 / 4900) loss: 2.010403
(Iteration 51 / 4900) loss: 1.928361
(Iteration 61 / 4900) loss: 1.945485
(Iteration 71 / 4900) loss: 1.882856
(Iteration 81 / 4900) loss: 1.728381
(Iteration 91 / 4900) loss: 1.906969
(Iteration 101 / 4900) loss: 1.753786
(Iteration 111 / 4900) loss: 1.755652
(Iteration 121 / 4900) loss: 1.616164
(Iteration 131 / 4900) loss: 1.769610
(Iteration 141 / 4900) loss: 1.909818
(Iteration 151 / 4900) loss: 1.597272
(Iteration 161 / 4900) loss: 1.743147
(Iteration 171 / 4900) loss: 1.612413
```

In [55]:
```
# Run this cell to visualize training loss and train / val accuracy, and save the log file of the
# experiment for submission.

plt.subplot(2, 1, 1)
plt.title('Training loss')
plt.plot(solver.loss_history, 'o')
plt.xlabel('Iteration')

plt.subplot(2, 1, 2)
plt.title('Accuracy')
plt.plot(solver.train_acc_history, '-o', label='train')
plt.plot(solver.val_acc_history, '-o', label='val')
plt.plot([0.5] * len(solver.val_acc_history), 'k--')
plt.xlabel('Epoch')
plt.legend(loc='lower right')
plt.gcf().set_size_inches(15, 12)
plt.show()

os.makedirs('submission_logs', exist_ok=True)
solver.record_histories_as_npz('submission_logs/train_2layer_fc.npz')
```

executed in 652ms, finished 11:14:12 2023-09-01





Multilayer network

Next you will implement a fully-connected network with an arbitrary number of hidden layers.

Read through the `FullyConnectedNet` class in the file `deeplearning/classifiers/fc_net.py` .

Implement the initialization, the forward pass, and the backward pass. For the moment don't worry about implementing dropout or batch normalization; we will add those features soon.

Initial loss and gradient check

As a sanity check, run the following to check the initial loss and to gradient check the network both with and without regularization. Do the initial losses seem reasonable?

For gradient checking, you should expect to see errors around 1e-6 or less.

In [73]:
```
N, D, H1, H2, C = 2, 15, 20, 30, 10
X = np.random.randn(N, D)
y = np.random.randint(C, size=(N,))

for reg in [0, 3.14]:
    print ('Running check with reg = ', reg)
    model = FullyConnectedNet([H1, H2], input_dim=D, num_classes=C,
                              reg=reg, weight_scale=5e-2, dtype=np.float64)

    loss, grads = model.loss(X, y)
    print ('Initial loss: ', loss)

    for name in sorted(grads):
        f = lambda _: model.loss(X, y)[0]
        grad_num = eval_numerical_gradient(f, model.params[name], verbose=False, h=1e-5)
        print ('%s relative error: %.2e' % (name, rel_error(grad_num, grads[name])))
```

executed in 2.43s, finished 12:33:22 2023-09-01

```
Running check with reg =  0
Initial loss:  2.3015511414743095
W1 relative error: 5.50e-05
W2 relative error: 5.25e-07
W3 relative error: 1.51e-07
b1 relative error: 1.95e-07
b2 relative error: 1.07e-08
b3 relative error: 1.88e-10
Running check with reg =  3.14
Initial loss:  6.895708601318777
W1 relative error: 7.63e-09
W2 relative error: 1.20e-07
W3 relative error: 3.47e-08
b1 relative error: 5.37e-08
b2 relative error: 5.03e-08
b3 relative error: 1.62e-10
```

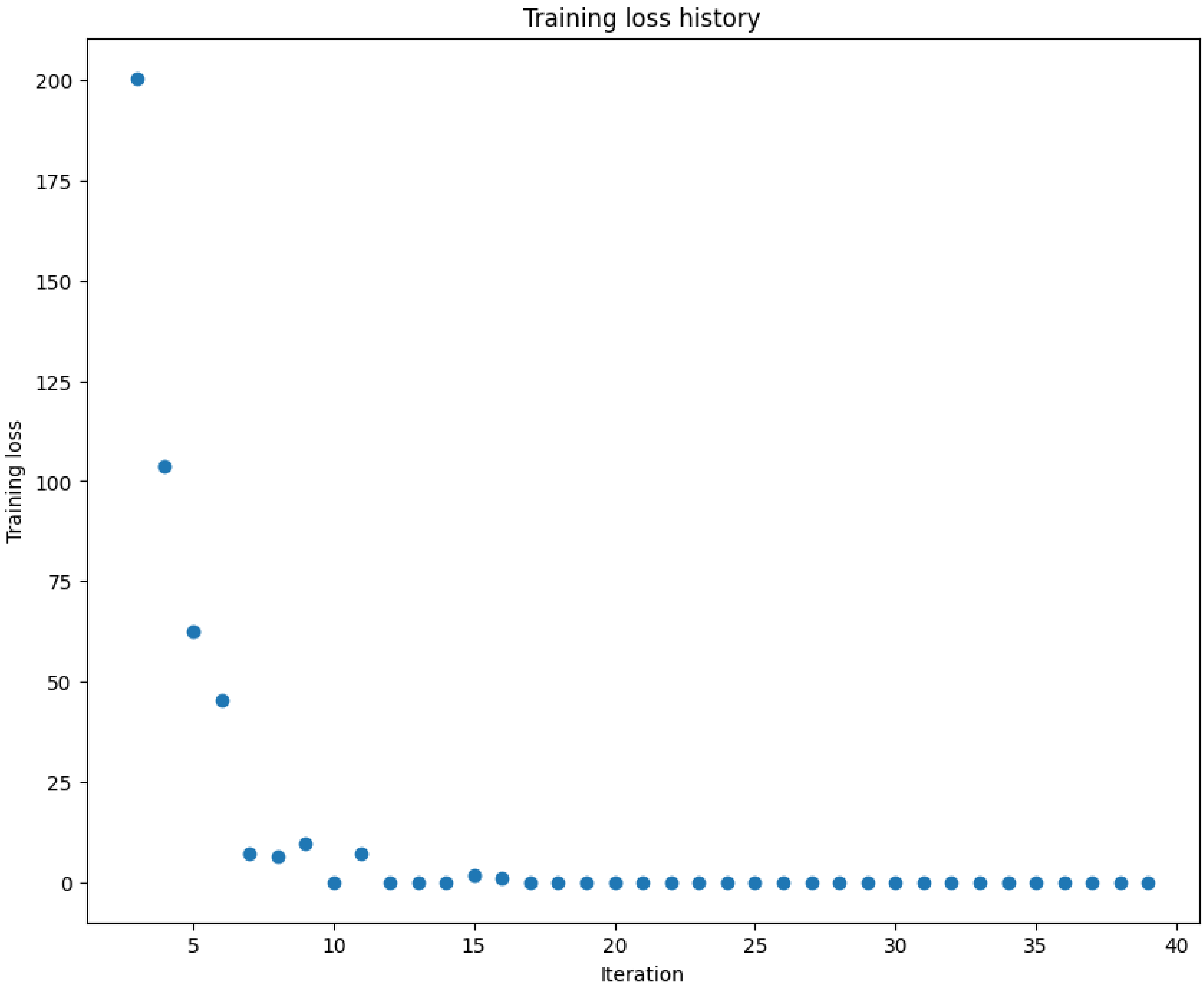
As another sanity check, make sure you can overfit a small dataset of 50 images. First we will try a three-layer network with 100 units in each hidden layer. You will need to tweak the learning rate and initialization scale, but you should be able to overfit and achieve 100% training accuracy within 20 epochs.

In [81]:
```
# TODO: Use a three-layer Net to overfit 50 training examples.

num_train = 50
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

############################################################################
# TODO: Tune these parameters to get 100% train accuracy within 20 epochs. #
############################################################################
weight_scale = 0.1
learning_rate = 1e-3
############################################################################
#                             END OF YOUR CODE                             #
############################################################################

model = FullyConnectedNet([100, 100],
              weight_scale=weight_scale, dtype=np.float64)
solver = Solver(model, small_data,
                print_every=10, num_epochs=20, batch_size=25,
                update_rule='sgd',
                optim_config={
                  'learning_rate': learning_rate,
                }
         )
solver.train()

plt.plot(solver.loss_history, 'o')
plt.title('Training loss history')
plt.xlabel('Iteration')
plt.ylabel('Training loss')
plt.show()

solver.record_histories_as_npz('submission_logs/overfit_3layer_fc.npz')
```

executed in 4.32s, finished 13:44:51 2023-09-01

```
F:\new_gitee_code\neutral_network\cs182hw1\deeplearning\layers.py:168: RuntimeWarning: divide by zero encountered in log
  dx /= N
```

```
(Iteration 1 / 40) loss: inf
(Epoch 0 / 20) train acc: 0.320000; val_acc: 0.132000
(Epoch 1 / 20) train acc: 0.240000; val_acc: 0.088000
(Epoch 2 / 20) train acc: 0.560000; val_acc: 0.125000
(Epoch 3 / 20) train acc: 0.760000; val_acc: 0.149000
(Epoch 4 / 20) train acc: 0.880000; val_acc: 0.129000
(Epoch 5 / 20) train acc: 0.960000; val_acc: 0.146000
(Iteration 11 / 40) loss: 0.008450
(Epoch 6 / 20) train acc: 0.980000; val_acc: 0.142000
(Epoch 7 / 20) train acc: 0.980000; val_acc: 0.142000
(Epoch 8 / 20) train acc: 0.960000; val_acc: 0.149000
(Epoch 9 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 10 / 20) train acc: 1.000000; val_acc: 0.145000
(Iteration 21 / 40) loss: 0.000000
(Epoch 11 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 12 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 13 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.145000
(Iteration 31 / 40) loss: 0.000033
(Epoch 16 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.145000
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.145000
```


Training loss history
Training loss
Iteration
200
175
150
125
100
75
50
25
0
5
10
15
20
25
30
35
40

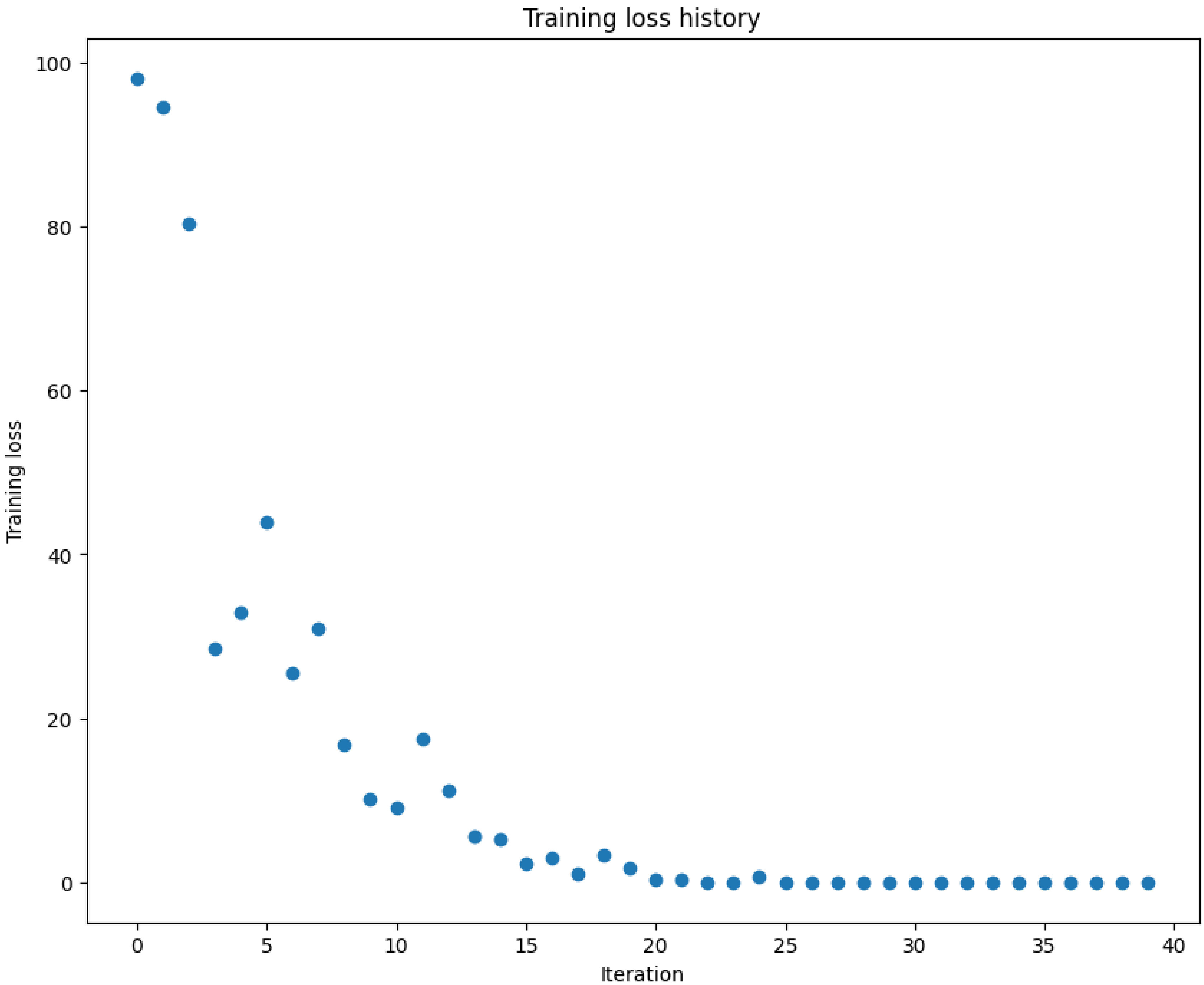
Now try to use a five-layer network with 100 units on each layer to overfit 50 training examples. Again you will have to adjust the learning rate and weight initialization, but you should be able to achieve 100% training accuracy within 20 epochs.

In [85]:
```
## TODO: Use a five-layer Net to overfit 50 training examples.

num_train = 50
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}

############################################################################
# TODO: Tune these parameters to get 100% train accuracy within 20 epochs. #
############################################################################
weight_scale = 0.1
learning_rate = 2e-4
############################################################################
#                             END OF YOUR CODE                             #
############################################################################

model = FullyConnectedNet([100, 100, 100, 100],
                weight_scale=weight_scale, dtype=np.float64)
solver = Solver(model, small_data,
                print_every=10, num_epochs=20, batch_size=25,
                update_rule='sgd',
                optim_config={
                  'learning_rate': learning_rate,
                }
         )
solver.train()

plt.plot(solver.loss_history, 'o')
plt.title('Training loss history')
plt.xlabel('Iteration')
plt.ylabel('Training loss')
plt.show()

solver.record_histories_as_npz('submission_logs/overfit_5layer_fc.npz')
```

executed in 4.37s, finished 13:50:47 2023-09-01

```
(Iteration 1 / 40) loss: 98.139997
```

```
(Epoch 0 / 20) train acc: 0.160000; val_acc: 0.116000
(Epoch 1 / 20) train acc: 0.220000; val_acc: 0.116000
(Epoch 2 / 20) train acc: 0.300000; val_acc: 0.119000
(Epoch 3 / 20) train acc: 0.360000; val_acc: 0.103000
(Epoch 4 / 20) train acc: 0.520000; val_acc: 0.122000
(Epoch 5 / 20) train acc: 0.560000; val_acc: 0.131000
(Iteration 11 / 40) loss: 9.054900
(Epoch 6 / 20) train acc: 0.600000; val_acc: 0.133000
(Epoch 7 / 20) train acc: 0.680000; val_acc: 0.126000
(Epoch 8 / 20) train acc: 0.760000; val_acc: 0.114000
(Epoch 9 / 20) train acc: 0.760000; val_acc: 0.138000
(Epoch 10 / 20) train acc: 0.940000; val_acc: 0.130000
(Iteration 21 / 40) loss: 0.347506
(Epoch 11 / 20) train acc: 0.960000; val_acc: 0.129000
(Epoch 12 / 20) train acc: 0.980000; val_acc: 0.128000
(Epoch 13 / 20) train acc: 1.000000; val_acc: 0.130000
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.130000
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.129000
(Iteration 31 / 40) loss: 0.000211
(Epoch 16 / 20) train acc: 1.000000; val_acc: 0.129000
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.129000
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.129000
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.129000
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.128000
```


Training loss history
Training loss
Iteration
100
80
60
40
20
0
0
5
10
15
20
25
30
35
40

Collect your submissions

In [86]:
```
!rm -f cs182hw1_submission.zip
!zip -r cs182hw1_submission.zip . -x "*.git*" "*deeplearning/datasets*" "*.ipynb_checkpoints*" "*README.md" ".env/*" "*.pyc" "*deeplea
```

executed in 669ms, finished 16:01:52 2023-09-01

```
'rm' 不是内部或外部命令，也不是可运行的程序
或批处理文件。

  adding: .idea/ (260 bytes security) (stored 0%)
  adding: .idea/cs182hw1.iml (172 bytes security) (deflated 43%)
  adding: .idea/inspectionProfiles/ (260 bytes security) (stored 0%)
  adding: .idea/inspectionProfiles/profiles_settings.xml (172 bytes security) (deflated 27%)
  adding: .idea/misc.xml (172 bytes security) (deflated 28%)
  adding: .idea/modules.xml (172 bytes security) (deflated 37%)
  adding: .idea/vcs.xml (172 bytes security) (deflated 23%)
  adding: .idea/workspace.xml (172 bytes security) (deflated 61%)
  adding: deeplearning/ (260 bytes security) (stored 0%)
  adding: deeplearning/classifiers/ (260 bytes security) (stored 0%)
  adding: deeplearning/classifiers/fc_net.py (172 bytes security) (deflated 77%)
  adding: deeplearning/data_utils.py (172 bytes security) (deflated 69%)
  adding: deeplearning/gradient_check.py (172 bytes security) (deflated 69%)
  adding: deeplearning/layers.py (172 bytes security) (deflated 77%)
  adding: deeplearning/layer_utils.py (172 bytes security) (deflated 79%)
  adding: deeplearning/optim.py (172 bytes security) (deflated 53%)
  adding: deeplearning/setup.py (172 bytes security) (deflated 47%)
  adding: deeplearning/solver.py (172 bytes security) (deflated 69%)
  adding: deeplearning/vis_utils.py (172 bytes security) (deflated 66%)
  adding: deeplearning/__init__.py (172 bytes security) (stored 0%)
  adding: FullyConnectedNets.ipynb (172 bytes security) (deflated 42%)
  adding: submission_logs/ (260 bytes security) (stored 0%)
  adding: submission_logs/overfit_3layer_fc.npz (172 bytes security) (deflated 55%)
  adding: submission_logs/overfit_5layer_fc.npz (172 bytes security) (deflated 48%)
  adding: submission_logs/train_2layer_fc.npz (172 bytes security) (deflated 8%)
```

Question

Did you notice anything about the comparative difficulty of training the three-layer net vs training the five layer net?

Please include your response in the written assignment submission.