INFR10086 Machine Learning (MLG) Semester 2, 2023/4

Coursework

Questions

1. In this question, we are going to work out the convergence rate of gradient descent on a particular family of functions.

   (a) Show that if f is convex, then

$$f(y) - f(x) - \nabla f(x)^\top (y - x) \le (\nabla f(y) - \nabla f(x))^\top (y - x) \tag{1}$$

   for any x and y.

   [2 marks]

   By the definition of convexity, $f(x) \ge f(y) + \nabla f(y)^\top (x - y)$. Apply this to $f(y)$ in the question,

$$\begin{aligned} f(y) - f(x) - \nabla f(x)^\top (y - x) &\le f(x) - \nabla f(y)^\top (x - y) - f(x) - \nabla f(x)^\top (y - x) \\ &\le (\nabla f(y) - \nabla f(x))^\top (y - x) \end{aligned}$$

   Remind yourself what Cauchy–Schwarz is. Show that if f is convex, then

$$f(y) - f(x) - \nabla f(x)^\top (y - x) \le \|\nabla f(y) - \nabla f(x)\|_2 \|y - x\|_2 \tag{2}$$

   for any x and y.

   [2 marks]

   By Cauchy–Schwarz,

$$(\nabla f(y) - \nabla f(x))^\top (y - x) \le \|\nabla f(y) - \nabla f(x)\|_2 \|y - x\|_2$$

   A function f is said to be L-smooth if

$$\|\nabla f(x) - \nabla f(y)\|_2 \le L \|x - y\|_2 \tag{3}$$

   for all x and y. Using the above, show that if f is convex and L-smooth, then

$$f(y) \le f(x) + \nabla f(x)^\top (y - x) + L \|x - y\|_2^2 \tag{4}$$

for any x and y.

[2 marks]

The desired inequality follows from the L-smooth assumption.

$$\begin{aligned} f(y) - f(x) - \nabla f(x)^\top (y-x) &\le f(x) - \nabla f(y)^\top (x-y) - f(x) - \nabla f(x)^\top (y-x) \\ &\le (\nabla f(y) - \nabla f(x))^\top (y-x) \\ &\le \|\nabla f(y) - \nabla f(x)\|_2 \|y-x\|_2 \\ &\le L\|y-x\|_2^2 \end{aligned}$$

We can conclude that if f is convex and L-smooth, you can always find a parabola that bounds the function from above.

(b) Let's consider the case where we do gradient descent on a convex and L-smooth function f. The gradient descent algorithm can be summarized as

$$x_t = x_{t-1} - \eta_t \nabla f(x_{t-1}) \tag{5}$$

for $t = 1, \ldots, T$. We know that if f is convex and L-smooth, then

$$f(y) \le f(x) + \nabla f(x)^\top (y-x) + L\|x-y\|_2^2 \tag{6}$$

for any x and y. Plug in $x = x_{t-1}$ and $y = x_t$ to the above and show that

$$f(x_{t-1}) - f(x_t) \ge \eta_t (1 - L\eta_t) \|\nabla f(x_{t-1})\|_2^2. \tag{7}$$

[2 marks]

Following the above instruction, we have

$$\begin{aligned} f(x_t) &\le f(x_{t-1}) + \nabla f(x_{t-1})^\top (x_t - x_{t-1}) + L\|x_t - x_{t-1}\|_2^2 \\ &\le f(x_{t-1}) + \nabla f(x_{t-1})^\top (-\eta_t \nabla f(x_{t-1})) + L\|\eta_t \nabla f(x_{t-1})\|_2^2 \\ &\le f(x_{t-1}) + (L\eta_t - 1)\eta_t \|\nabla f(x_{t-1})\|_2^2. \end{aligned}$$

Consider $g(s) = s(1 - Ls)$. This is a concave[1] parabola. Show that g has a maximum of $\frac{1}{4L}$ when $s = \frac{1}{2L}$, and g is 0 when $s = \frac{1}{L}$.

[2 marks]

[1] A function f is concave if $-f$ is convex.

By completing the square, we have

$$g(s) = -Ls^2 + s = -L\left(s - \frac{1}{2L}\right)^2 + \frac{1}{4L}.$$

This shows that g has a maximum of $\frac{1}{4L}$ at $\frac{1}{2L}$. It's also clear that g is 0 when $s = \frac{1}{L}$.

Using the above results, we can choose $\eta_t = \frac{1}{2L}$ and conclude

$$f(x_{t-1}) - f(x_t) \geq \frac{1}{4L}\|\nabla f(x_{t-1})\|_2^2 \geq 0. \tag{8}$$

In other words, if we know the function is convex and L-smooth and we do gradient descent with a step size of $\frac{1}{2L}$, we are guaranteed to decrease the objective after every gradient update.

(c) A function f is said to be μ-strongly convex if

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2}\|y - x\|_2^2 \tag{9}$$

for any x and y. Note the direction of the inequality. Compared to L-smooth functions that have parabolas bounded from above, strongly convex functions have parabolas bounded from below.

Show that, for a particular x, the quadratic function on the right hand side

$$g(z) = f(x) + \nabla f(x)^\top (z - x) + \frac{\mu}{2}\|z - x\|_2^2 \tag{10}$$

is minimized when $z = x - \frac{1}{\mu}\nabla f(x)$.

[2 marks]

By taking the gradient of g with respect to z, we have

$$\nabla_z g(z) = \nabla f(x) + \frac{\mu}{2}(2z - 2x).$$

The minimum can be found by solving $\nabla_z g(z) = 0$.

Based on the result above, show that

$$f(y) \geq g(y) \geq \min_z g(z) = f(x) - \frac{1}{2\mu}\|\nabla f(x)\|_2^2, \tag{11}$$

for any x and y.

[2 marks]

Plugging in $x - \frac{1}{\mu}\nabla f(x)$, we have

$$\begin{aligned} f(y) &\geq f(x) + \nabla f(x)^\top \left(-\frac{1}{\mu}\nabla f(x)\right) + \frac{\mu}{2}\left\|\frac{1}{\mu}\nabla f(x)\right\|_2^2 \\ &= f(x) - \frac{1}{2\mu}\|\nabla f(x)\|_2^2. \end{aligned}$$

Now choose $y = x^*$ where x^* is the minimizer of f, and conclude that

$$f(x) - f(x^*) \leq \frac{1}{2\mu}\|\nabla f(x)\|_2^2. \tag{12}$$

[2 marks]

Plugging in x^*, we have

$$f(x^*) \geq f(x) - \frac{1}{2\mu}\|\nabla f(x)\|_2^2.$$

In other words, if a function is strongly convex, we know how far the optimal solution is just by looking at the norm of the gradient.

(d) We know that if we perform gradient descent on a L-smooth function f then

$$f(x_t) \leq f(x_{t-1}) - \frac{1}{4L}\|\nabla f(x_{t-1})\|_2^2. \tag{13}$$

We can subtract $f(x^*)$ from both sides to get

$$f(x_t) - f(x^*) \leq f(x_{t-1}) - f(x^*) - \frac{1}{4L}\|\nabla f(x_{t-1})\|_2^2. \tag{14}$$

If we further assume that f is μ-strongly convex, show that

$$\begin{aligned} f(x_t) - f(x^*) &\leq f(x_{t-1}) - f(x^*) + \frac{\mu}{2L}(f(x^*) - f(x_{t-1})) \qquad (15) \\ &= \left(1 - \frac{\mu}{2L}\right)(f(x_{t-1}) - f(x^*)). \qquad (16) \end{aligned}$$

[2 marks]

Using the fact that $f(x) - f(x^*) \leq \frac{1}{2\mu}\|\nabla f(x)\|_2^2$, we get

$$\begin{aligned} f(x_t) - f(x^*) &\leq f(x_{t-1}) - f(x^*) - \frac{1}{4L}\|\nabla f(x_{t-1})\|_2^2 \\ &\leq f(x_{t-1}) - f(x^*) - \frac{1}{4L}2\mu(f(x^*) - f(x_{t-1})) \\ &\leq f(x_{t-1}) - f(x^*) + \frac{\mu}{2L}(f(x^*) - f(x_{t-1})) \\ &= \left(1 - \frac{\mu}{2L}\right)(f(x_{t-1}) - f(x^*)). \end{aligned}$$

Apply this result repeatedly, and conclude that

$$f(x_t) - f(x^*) \leq \left(1 - \frac{\mu}{2L}\right)^t (f(x_0) - f(x^*)). \tag{17}$$

[2 marks]

Using the fact that $f(x) - f(x^*) \leq \frac{1}{2\mu}\|\nabla f(x)\|_2^2$, we get

$$\begin{aligned} f(x_t) - f(x^*) &\leq \left(1 - \frac{\mu}{2L}\right)(f(x_{t-1}) - f(x^*)) \\ &\leq \left(1 - \frac{\mu}{2L}\right)^2 (f(x_{t-2}) - f(x^*)) \\ &\leq \left(1 - \frac{\mu}{2L}\right)^t (f(x_0) - f(x^*)). \end{aligned}$$

In sum, this is the convergence rate if we run gradient descent with a constant step size of $\frac{1}{2L}$ on an L-smooth, μ-strongly convex function.
Is the convergence quadratic, linear, or sublinear?

[2 marks]

Based on L-smoothness and μ-strongly convex, we have $\mu/2 < L$; otherwise, the upper bound would be below the lower bound, a contradiction. In sum, we have

$$f(x_t) - f(x^*) \leq r^t(f(x_0) - f(x^*)),$$

for some $0 < r < 1$. The convergence is linear.

2. In this question, we are going to study the properties of hinge loss.

   (a) Show that

$$\max(a + b, c + d) \leq \max(a, c) + \max(b, d) \tag{18}$$

for any a, b, c, and d.

[2 marks]

Because

$$\begin{aligned} a &\le \max(a, c) \\ b &\le \max(b, d) \end{aligned}$$

we have $a + b \le \max(a, c) + \max(b, d)$. Similarly, because

$$\begin{aligned} c &\le \max(a, c) \\ d &\le \max(b, d) \end{aligned}$$

we have $c + d \le \max(a, c) + \max(b, d)$. Given that $\max(a, c) + \max(b, d)$ is larger than both $a + b$ and $c + d$, we arrive at $\max(a + b, c + d) \le \max(a, c) + \max(b, d)$.

Use the above result and show that if f and g are both convex, then

$$h(x) = \max(f(x), g(x)) \tag{19}$$

is also convex.

[2 marks]

Based on the convextiy of f and g, we have

$$\begin{aligned} h(\alpha x + (1 - \alpha)y) &= \max(f(\alpha x + (1 - \alpha)y), g(\alpha x + (1 - \alpha)y)) \\ &\le \max(\alpha f(x) + (1 - \alpha)f(y), \alpha g(x) + (1 - \alpha)g(y)) \\ &\le \max(\alpha f(x), \alpha g(x)) + \max((1 - \alpha)f(y), (1 - \alpha)g(y)) \\ &\le \alpha \max(f(x), g(x)) + (1 - \alpha) \max(f(y), g(y)) \\ &\le \alpha h(x) + (1 - \alpha)h(y). \end{aligned}$$

(b) In class, we have talked about the hinge loss for binary classification

$$\ell_{\text{hinge}}(w; x, y) = \max(0, 1 - yw^\top x) \tag{20}$$

where (x, y) is a sample and $y \in \{+1, -1\}$. Use the result above and show that the hinge loss is convex in w.

[2 marks]

The first term in the max 0 is a constant and convex in w. The second term $1 - yw^\top x$ is affine in w, so it is also convex in w. Based on the result above, the maximum of the two is convex in w.

(c) Recall that the zero-one loss is

$$\ell_{01}(w; x, y) = \mathbb{1}_{yw^\top x < 0} \tag{21}$$

where (x, y) is a sample and $y \in \{+1, -1\}$. Show that the hinge loss is always an upper bound of the zero-one loss. (Hint: Enumerate the possible values of y and the possible signs of $w^\top x$.)

[2 marks]

When $yw^\top x < 0$, the hinge loss is $1 - yw^\top x$ and is larger than 1. The zero-one loss is 1, so the hinge loss is an upper bound in this case. When $yw^\top x \geq 0$, the zero-one loss is 0. Regardless of how hinge loss behaves, it is bounded by 0 because of $\max(0, \cdot)$, so the hinge loss is the upper bound of zero-one loss in both cases.

(d) Show that if f is convex and g is μ-strongly convex, then

$$h(x) = f(x) + \lambda g(x) \tag{22}$$

is $\lambda\mu$-strongly convex for $\lambda > 0$. (Hint: Use the definition of convex and strongly convex functions.)

[2 marks]

Based on the definition, we have

$$\begin{aligned} f(x) &\geq f(y) + \nabla f(y)^\top (x - y) \\ g(x) &\geq g(y) + \nabla g(y)^\top (x - y) + \frac{\mu}{2}\|x - y\|_2^2 \end{aligned}$$

We can combine the two inequality to get

$$f(x) + \lambda g(x) \geq f(y) + \lambda g(y) + (\nabla f(y) + \lambda \nabla g(y))^\top (x - y) + \frac{\lambda\mu}{2}\|x - y\|_2^2,$$

which in turn gives

$$h(x) \geq h(y) + \nabla h(y)^\top (x - y) + \frac{\lambda\mu}{2}\|x - y\|_2^2.$$

Show that $g(x) = \frac{1}{2}\|x\|_2^2$ is 1-strongly convex.

[2 marks]

Since

$$\frac{1}{2}\|x\|_2^2 - x^\top y + \frac{1}{2}\|y\|_2^2 = \frac{1}{2}\|x-y\|_2^2,$$

we can arrange the terms to have

$$\begin{aligned} g(x) = \frac{1}{2}\|x\|_2^2 &= -\frac{1}{2}\|y\|_2^2 + x^\top y + \frac{1}{2}\|x-y\|_2^2 \\ &= \frac{1}{2}\|y\|_2^2 + y^\top (x-y) + \frac{1}{2}\|x-y\|_2^2 \\ &= g(y) + \nabla g(y)^\top (x-y) + \frac{1}{2}\|x-y\|_2^2. \end{aligned}$$

If we optimize the loss function

$$\frac{1}{n}\sum_{i=1}^{n} \ell_{\text{hinge}}(w; x_i, y_i) + \frac{\lambda}{2}\|w\|_2^2 \tag{23}$$

on a data set $\{(x_1, y_1), \dots, (x_n, y_n)\}$, conclude that this loss function is λ-strongly convex. [2 marks]

Since the hinge loss is convex, the non-negative sum of hinge losses are also convex. Adding $\frac{\lambda}{2}\|w\|_2^2$, which is λ-strongly convex, gives a λ-strongly convex function.

Compare the above objective to support vector machines, and convince yourself that support vector machines are optimizing the hinge loss.

3. MNIST is a data set consisting of hand-written digits. In this question, we are going to implement a linear classifier using only `numpy`. You are **not** allowed to use any packages other than `numpy`, `matplotlib`, and those in the Python standard library.

   Download the tar ball from `https://homepages.inf.ed.ac.uk/htang2/mlg2023/mnist.tar.gz`. It includes the data set, and a file `mnist.py` to help you load the data set.

   To get full marks for this question, you need to paste all your code and plots in a PDF and submit that with your answers to other questions.

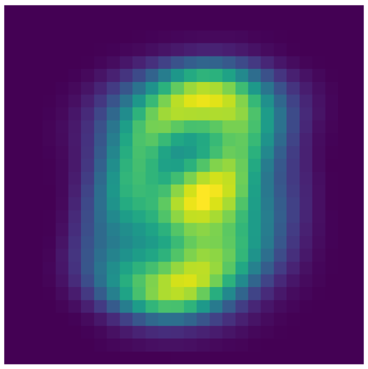
   (a) Use the snippet below to load the training set.

```
import mnist
images = mnist.load_images('train-images-idx3-ubyte')
```

   Since each image is 28×28, we can linearize every image to a 784-dimensional vector. Simply use `numpy.reshape` to reshape the 28×28 matrix to a 784-dimensional vector. Write a script to find the mean of the whole data set. Reshape the mean vector back to a 28×28 matrix and use `pyplot.imshow` to plot the mean "image". What does the mean image look like?

[4 marks]

The mean image looks like the following.



The code is listed at the end of the document.

(b) Remind yourself about the log loss

$$\ell(w; x, y) = -\log p(y|x) = -\log \frac{\exp(w_y^\top x)}{\sum_{y' \in \{0,\dots,9\}} \exp(w_{y'}^\top x)} \tag{24}$$

$$= -w_y^\top x + \log \sum_{y' \in \{0,\dots,9\}} \exp(w_{y'}^\top x) \tag{25}$$

for multiclass classification. Note that you will need a weight vector for each label. In other words, we need weight vectors $w_0, \dots, w_9$ for labels $0, \dots 9$. What is the gradient $\nabla_{w_{\tilde{y}}} \ell$?

[2 marks]

The gradient of log loss is

$$\nabla_{w_{\tilde{y}}} \ell = -\mathbb{1}_{y=\tilde{y}} x + \frac{\exp(w_{\tilde{y}}^\top x)}{\sum_{y' \in \{0,\dots,9\}} \exp(w_{y'}^\top x)} x.$$

Mind the different y's. The symbol y is used for the ground truth label, the symbol y' is used in the sum, and the symbol $\tilde{y}$ is some label that we take gradient of.

(c) Use the snippet below to load the labels of the data set.

```python
import mnist
labels = mnist.load_labels('train-labels-idx1-ubyte')
```

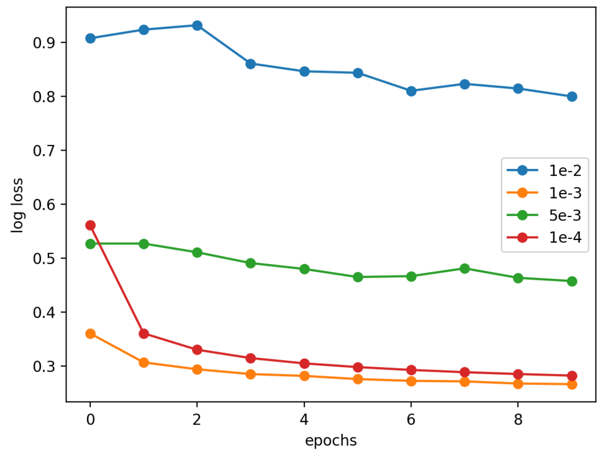
Implement stochastic gradient descent (SGD) to optimize the log loss above with a batch size of 1. Recall that an epoch is a pass over the data set. The training set have 60,000 samples, so one epoch should give you 60,000 gradient updates. Print the loss value on a sample before the gradient update. Run SGD for 10 epochs. Average the losses printed, and you should end up with 10 average loss values. It is a good practice to save a classifier after every epoch, in case you want to continue from where you left off.

Plot the losses where the x-axis is the number of epochs and the y-axis is the averaged loss values per epoch. Repeat this process for different step sizes and overlay the loss curves on top of each other.

What plot do you get after overlaying the loss curves? What is the best step size that leads to the lowest loss values?

[6 marks]

Depending on the choice of step sizes, we get different loss curves.



The code is listed at the end of the document.

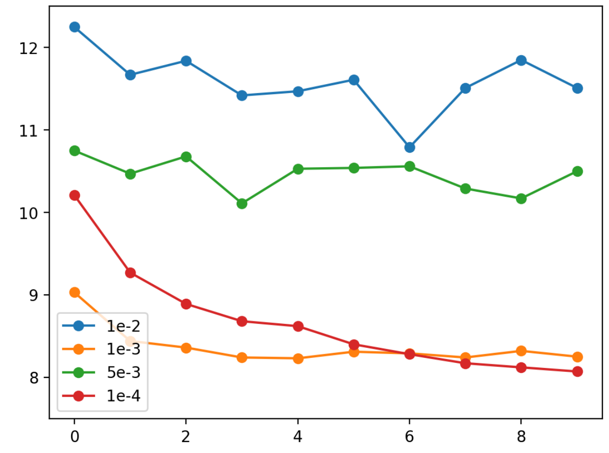
Write a script to load the two files `t10k-images-idx3-ubyte` and `t10k-labels-idx1-ubyte`. Implement

$$\hat{y} = \operatorname*{argmax}_{y \in \{0,\dots,9\}} w_y^\top x \tag{26}$$

for prediction, and measure the zero-one loss of the classifiers you have. What is the zero-one loss that you get for your classifiers?

[2 marks]

The best zero-one loss 8.1% is achieved with a step size of 10^{-4} at epoch 10.



The code is listed at the end of the document.

```python
import array
import sys
import numpy

def load_images(filename):
    f = open(filename, 'rb')

    sig = f.read(4)
    dim1 = int.from_bytes(f.read(4), byteorder='big', signed=False)
    dim2 = int.from_bytes(f.read(4), byteorder='big', signed=False)
    dim3 = int.from_bytes(f.read(4), byteorder='big', signed=False)

    data = numpy.array(array.array('B', f.read()), dtype=numpy.dtype(float))
    result = data.reshape(dim1, dim2, dim3)

    f.close()

    return result


def load_labels(filename):
    f = open(filename, 'rb')

    sig = f.read(4)
    dim1 = int.from_bytes(f.read(4), byteorder='big', signed=False)

    result = numpy.array(array.array('B', f.read()), dtype=numpy.dtype(float))

    f.close()

    return result
```

Listing 1: File `mnist.py`.

```
import numpy
import math


def log_loss(w, x, y, K):
    scores = [numpy.dot(w[i], x) for i in range(K)]
    m = max(enumerate(scores), key=lambda t: t[1])
    logZ = scores[m[0]] + math.log(sum(math.exp(s - scores[m[0]]) for s in scores))

    return -scores[y] + logZ


def grad_log_loss(w, x, y, K):
    scores = [numpy.dot(w[i], x) for i in range(K)]
    m = max(enumerate(scores), key=lambda t: t[1])
    logZ = scores[m[0]] + math.log(sum(math.exp(s - scores[m[0]]) for s in scores))
    grad = numpy.zeros_like(w)
    for i in range(K):
        grad[i] = ((-1.0 if i == y else 0.0) + math.exp(scores[i] - logZ)) * x

    return grad


def load_mean_var(filename):
    f = open(filename)
    mean = numpy.array([float(e) for e in f.readline()[1:-2].split(',')])
    var_ = numpy.array([float(e) for e in f.readline()[1:-2].split(',')])
    f.close()
    return mean, var_


def load_param(filename):
    f = open(filename)
    w = []
    for i in range(10):
        w.append([float(e) for e in f.readline()[1:-2].split(',')])
    f.close()
    return numpy.array(w)


def save_param(w, filename):
    f = open(filename, 'w')
    for v in w:
        print(list(v), file=f)
    f.close()
```

Listing 2: File `linear.py`

```
#!/usr/bin/env python3

import sys
import math
import numpy
import mnist
import linear


param = sys.argv[1]
param_out = sys.argv[2]
step_size = 5e-3

images = mnist.load_images('train-images-idx3-ubyte')
labels = mnist.load_labels('train-labels-idx1-ubyte')

mean, var_ = linear.load_mean_var('train.mean-var')
var_[var_ == 0.0] = 1.0

w = linear.load_param(param)

sample = 1

for img, lab in zip(images, labels):
    img = img.reshape(28 * 28)
    img = (img - mean) / numpy.sqrt(var_)

    loss = linear.log_loss(w, img, int(lab), 10)
    grad = linear.grad_log_loss(w, img, int(lab), 10)

    w -= grad * step_size

    print('sample: {}'.format(sample))
    print('loss: {:.6}'.format(loss))
    print('grad norm: {:.6}'.format(numpy.linalg.norm(w)))
    print()

    sample += 1

linear.save_param(w, param_out)
```

Listing 3: File `train-linear.py`

```
#!/usr/bin/env python3

import sys
import math
import numpy
import mnist
import linear


param = sys.argv[1]

images = mnist.load_images('t10k-images-idx3-ubyte')
labels = mnist.load_labels('t10k-labels-idx1-ubyte')

mean, var_ = linear.load_mean_var('train.mean-var')
var_[var_ == 0.0] = 1.0

w = linear.load_param(param)

log_loss = 0
zero_one_loss = 0
K = 10

samples = 1

for img, lab in zip(images, labels):
    img = img.reshape(28 * 28)
    img = (img - mean) / numpy.sqrt(var_)

    loss = linear.log_loss(w, img, int(lab), 10)
    m = max(enumerate([numpy.dot(w[i], img) for i in range(K)]), key=lambda t: t[1])

    zero_one_loss += (1.0 if m[0] != lab else 0.0)
    log_loss += loss

    print('sample: {}'.format(samples))
    print('log loss: {:.6}'.format(loss))
    print('zero-one loss: {:.6}'.format(1.0 if m[0] != lab else 0.0))
    print('')

    samples += 1

print('avg log loss: {:.6}'.format(log_loss / samples))
print('avg zero-one loss: {:.6}'.format(zero_one_loss / samples))
```

Listing 4: File `eval-linear.py`

```python
#!/usr/bin/env python3

import mnist
import numpy

images = mnist.load_images('train-images-idx3-ubyte')

ex = numpy.zeros(28 * 28)
ex2 = numpy.zeros(28 * 28)
samples = 0

for img in images:
    img = img.reshape(28 * 28)

    ex += img
    ex2 += img * img
    samples +=1

mean = ex / samples

print(list(mean))
print(list(ex2 / samples - mean ** 2))
print(samples)
```

Listing 5: File `mean-var.py`