Machine Learning

Basak Guler

University of California, Riverside

Department of Electrical and Computer Engineering

Motivation

- Consider a self-driving car that needs to navigate without a driver

lidar
Self driving car
radar

Motivation

- Consider a self-driving car that needs to navigate without a driver
- For this, the vehicle has to make decisions based on the information it gathers from the environment, using various sensing devices such as cameras, radars, lidars

lidar
Self driving car
radar

Motivation

- Machine learning allows us to train mathematical models that can learn from data or past experience how to make decisions or predictions
- It is an important tool for automating decision-making in real-world applications

Motivation

Machine learning has a large number of real-world applications:

- Image recognition (robotics, autonomous vehicles)
- Speech recognition (smart assistants such as Siri, machine translation)
- Product and content recommendations (video, music, news etc.)
- Navigation (self-driving cars, autonomous vehicles)
- Medical diagnosis
- Social media
- Gaming
- Computer security, fraud and malware detection

Motivation

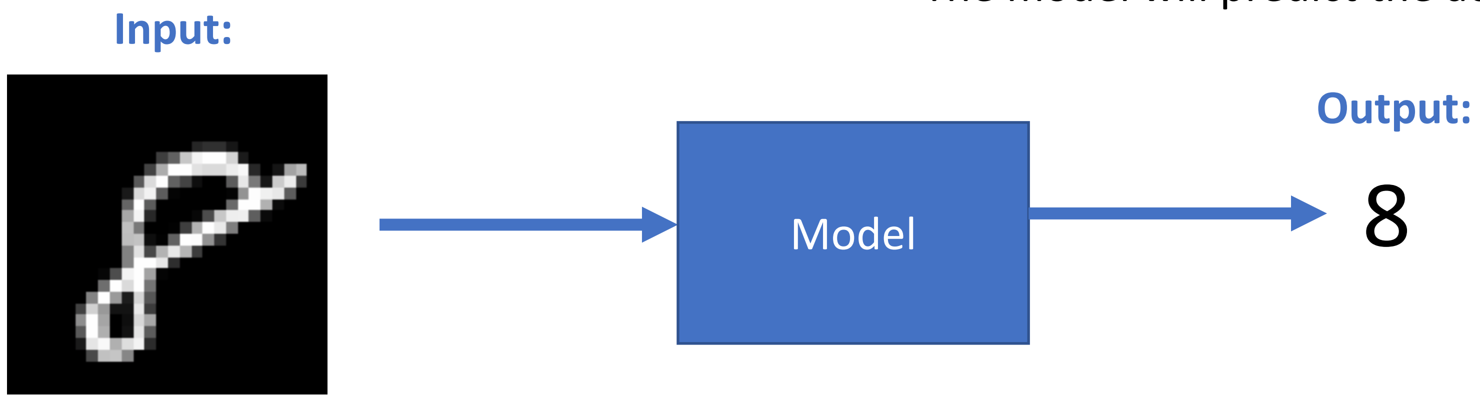
- In this module, we will learn the basic principles of machine learning by training a simple machine learning model
- The most popular programming languages for machine learning are Python and C++
- In our project, we will use the Python programming language
- To learn the functionality of a specific command that appears in the learning module, you can either:
  - Read from the Python tutorial from https://docs.python.org/3/tutorial/
  - Or search " Python + *name of the command*" on Google

Training a machine learning model

- Using Python, we will train a simple machine learning model, called “linear regression”, to predict the digit of a handwritten image

Our input will be an image of an unknown handwritten digit

The model will predict the actual digit in the image



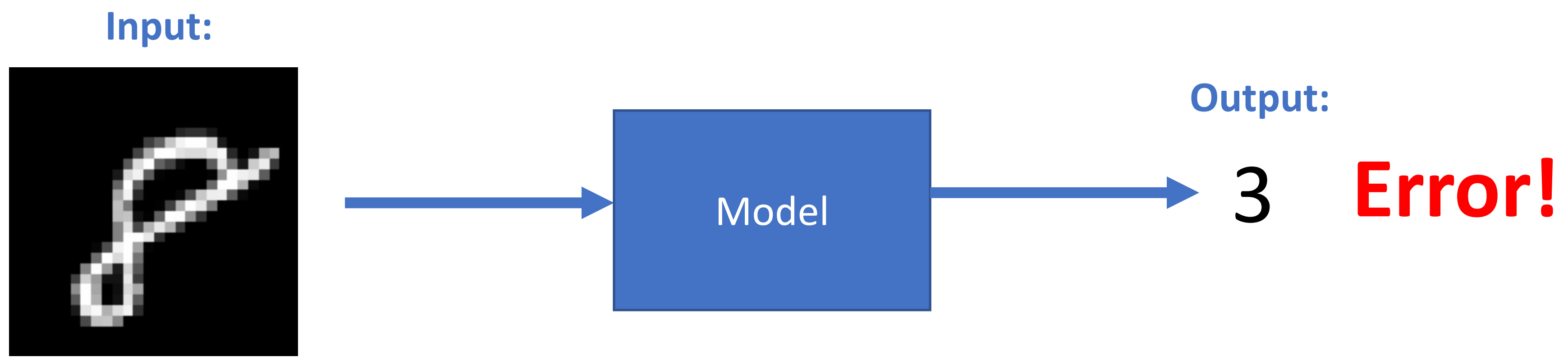
We will call the actual digit of the image as a “label”

In other words, the label of this image is 8

Training a machine learning model

- If the model predicts the wrong digit, an error is made

Our input will be an image of an unknown handwritten digit

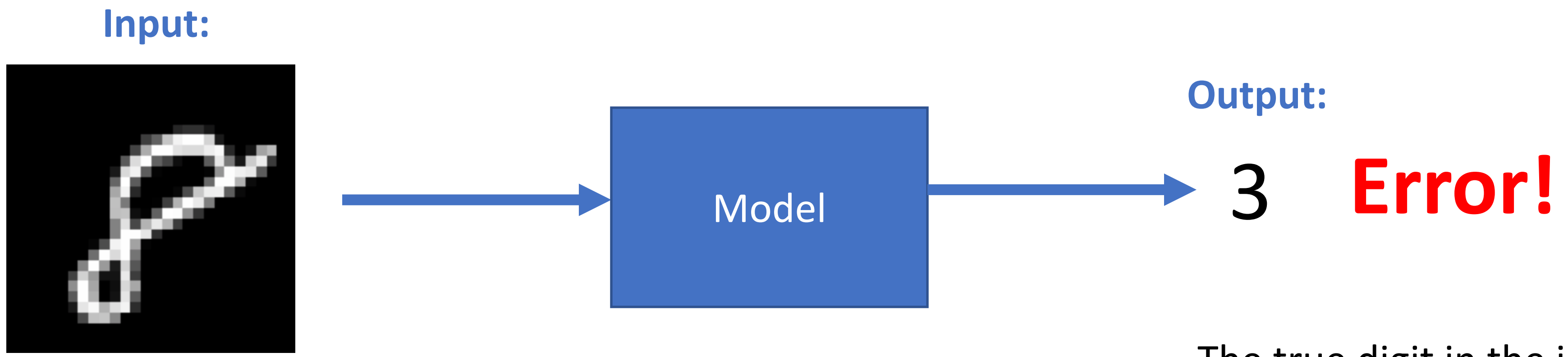


Training a machine learning model

- Our goal is to train a model that minimizes the errors in the predictions

Our input will be an image of an unknown handwritten digit

If the model predicts the wrong digit, an error is made



The true digit in the image is 8,
but the model has predicted 3!

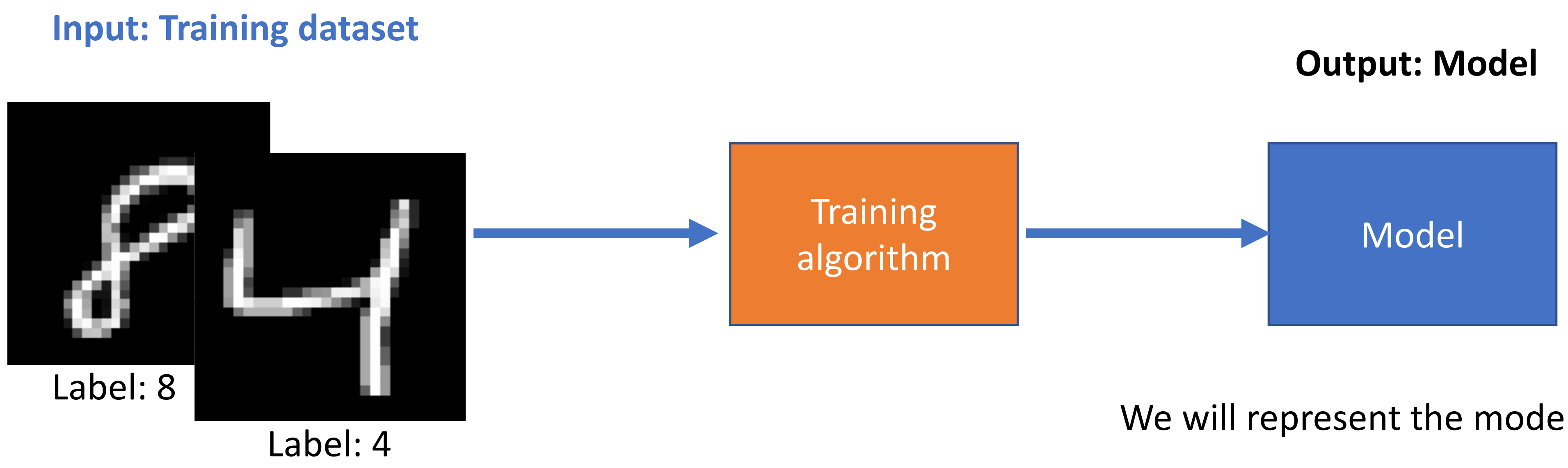
Training a machine learning model

- For this, we will define a loss function, which measures the error between the predicted outcome and the actual label
- The loss function depends on the specific machine learning model.
- Some common ones are squared-error loss and cross-entropy loss

Model		**Loss function**
Linear regression	↔	Squared error loss
Logistic regression	↔	Cross-entropy loss

Training a machine learning model

- We will try to find the model that minimizes the loss function over a known set of images called the training set
- The output of the training algorithm is the model

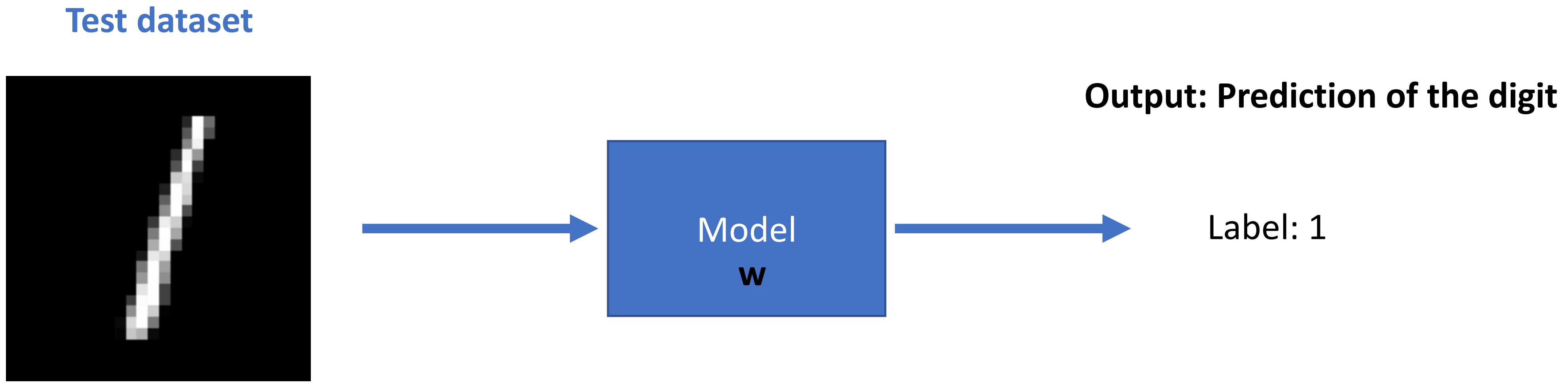


We will represent the model as a vector "**w**"

Training dataset consists of a set of images and their true labels)

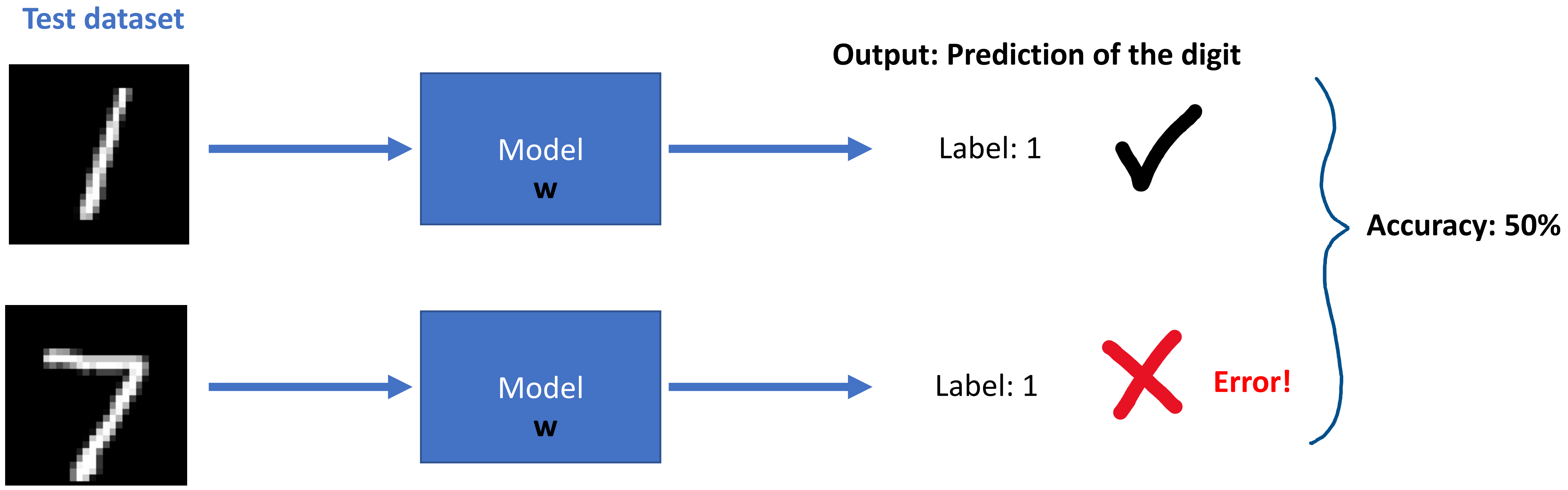
Testing a model

- Then we will test how good this model is on a different set of images, called the "test set"
- The model will try to predict the actual digits of the images in the test set



Testing a model

- We will then compute the percentage of the errors the model has made while predicting the labels of the images in the test set
- We will call this the accuracy of the model
  - The higher the accuracy, the better our model is



Comments

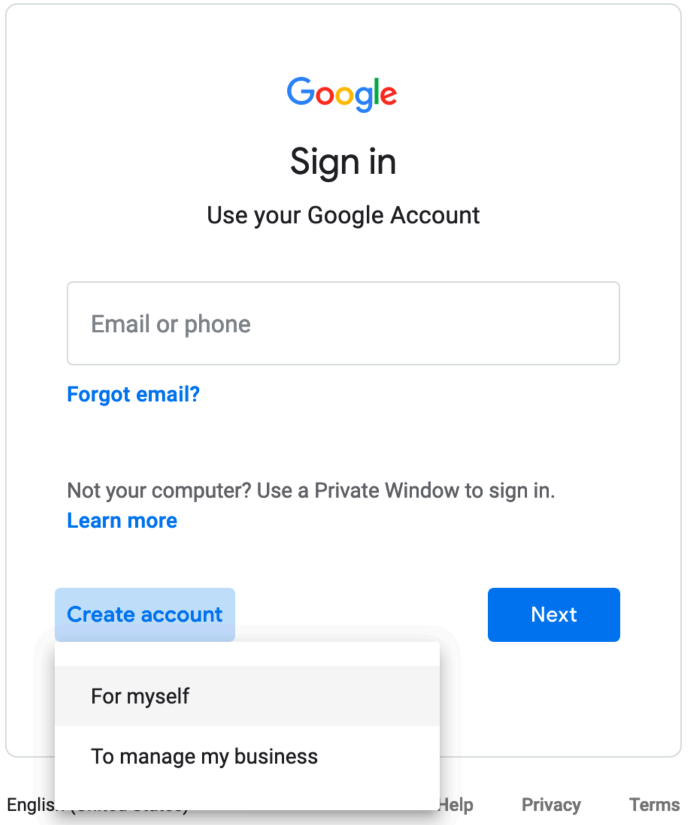
- Note that training and testing are done on different images, in other words, the model has never “seen” the images in the test set during training
- Instead, the model “learns” how to interpret images on the training set, and uses this “information” to interpret previously unseen images, which are the images in the test set
- Next, we will train a simple machine learning model using the Python programming language

How to run your Python code on a web browser

- For this, we will use Google Colab, which is a free service created by Google that allows us to run our Python code on a web browser
  - The code runs on Google’s machines, not our local machine
  - We can modify our code and see the results in real time
  - No installation required, but you need an active internet connection
  - Make sure to save a copy of your code and results (when your session ends or internet connection gets cut, you may lose your progress otherwise)
  - Can be integrated with Google drive (can save your code on Google drive)
- To use it, we need a Google account
- Create a Google account
  - If you have a gmail account, you can skip this step
- Sign in to your Google account

How to run your Python code on a web browser

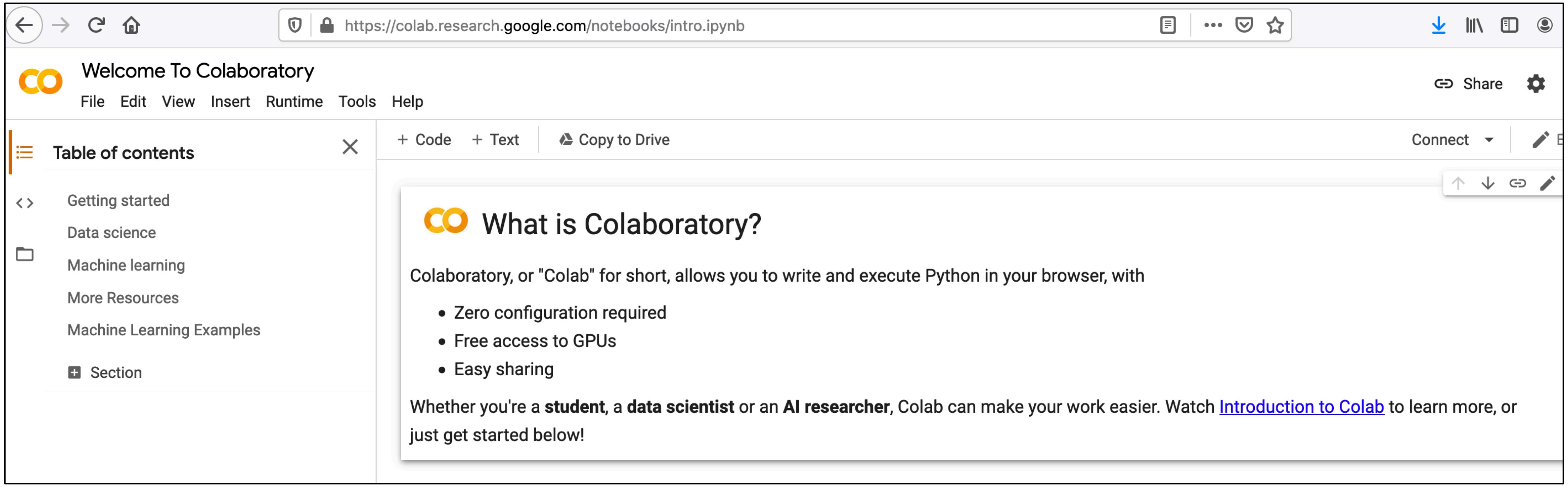
- To use it, we need a Google account
- Create a Google account by going to www.google.com and click on "Sign In" at the top right, then click Create Account -> For myself
  - If you have a Google account already, you can skip this step
- Then, sign in to your Google account



How to run your Python code on a web browser

- Open the Google Colab web page in your web browser:

  https://colab.research.google.com



What is Colaboratory?

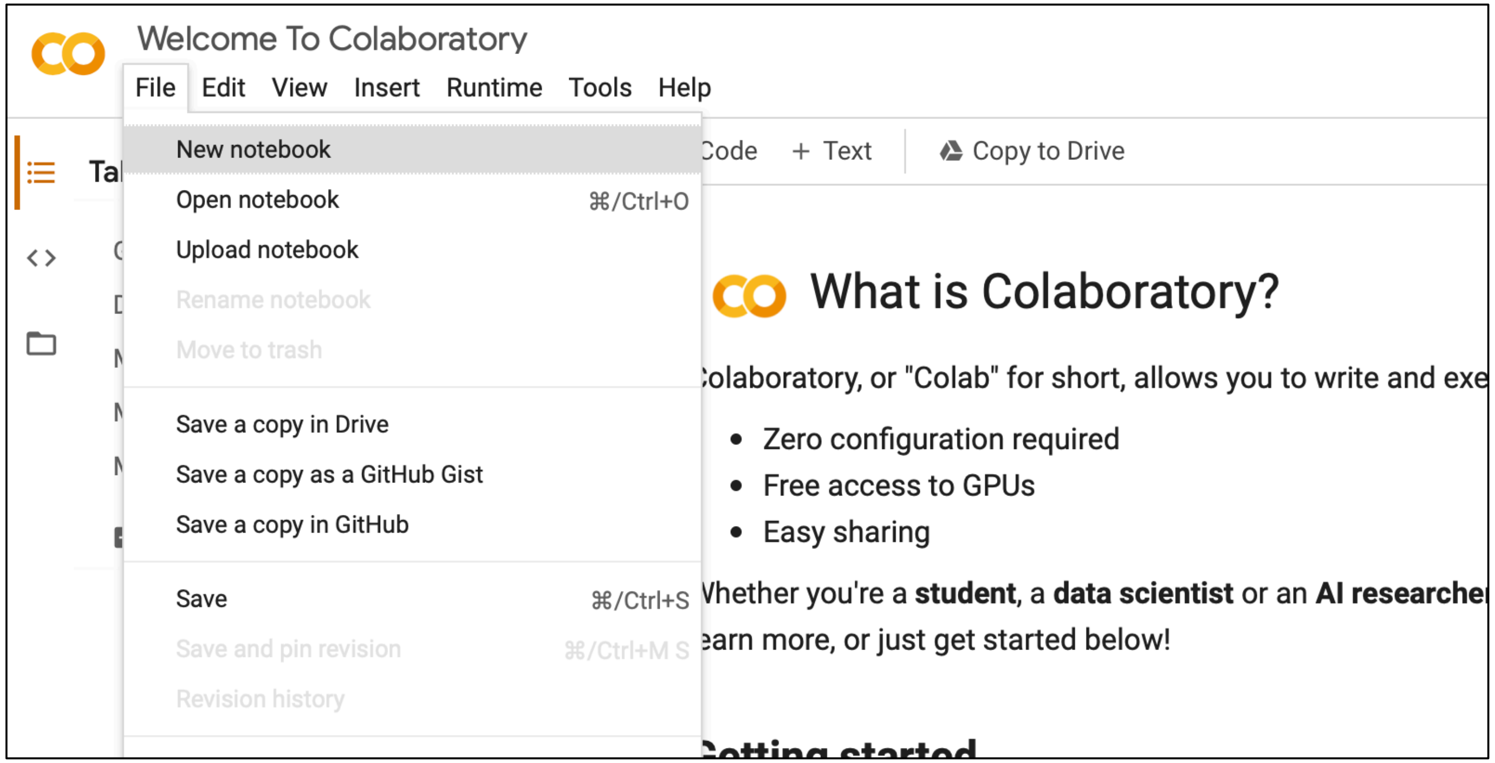
Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

How to run your Python code on a web browser

- After opening the Google Colab website, click on File -> New Notebook



What is Colaboratory?

olaboratory, or "Colab" for short, allows you to write and exe

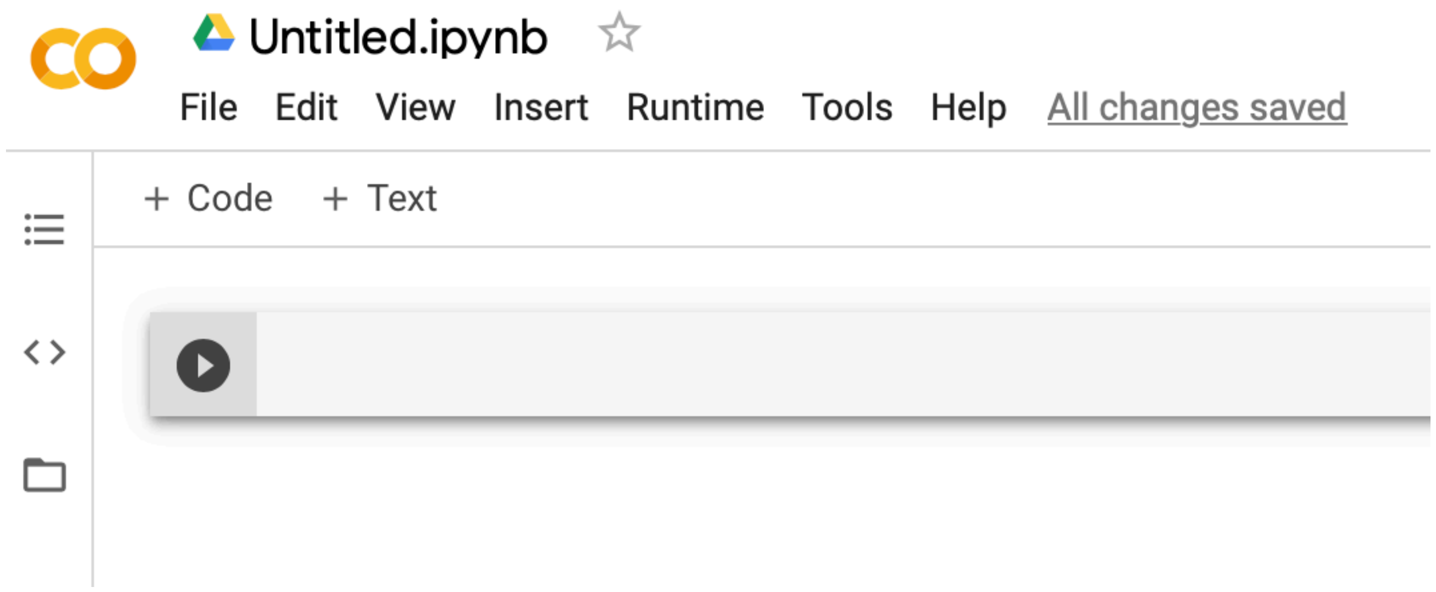
- Zero configuration required
- Free access to GPUs
- Easy sharing

Vhether you're a **student**, a **data scientist** or an **AI researche**

earn more, or just get started below!

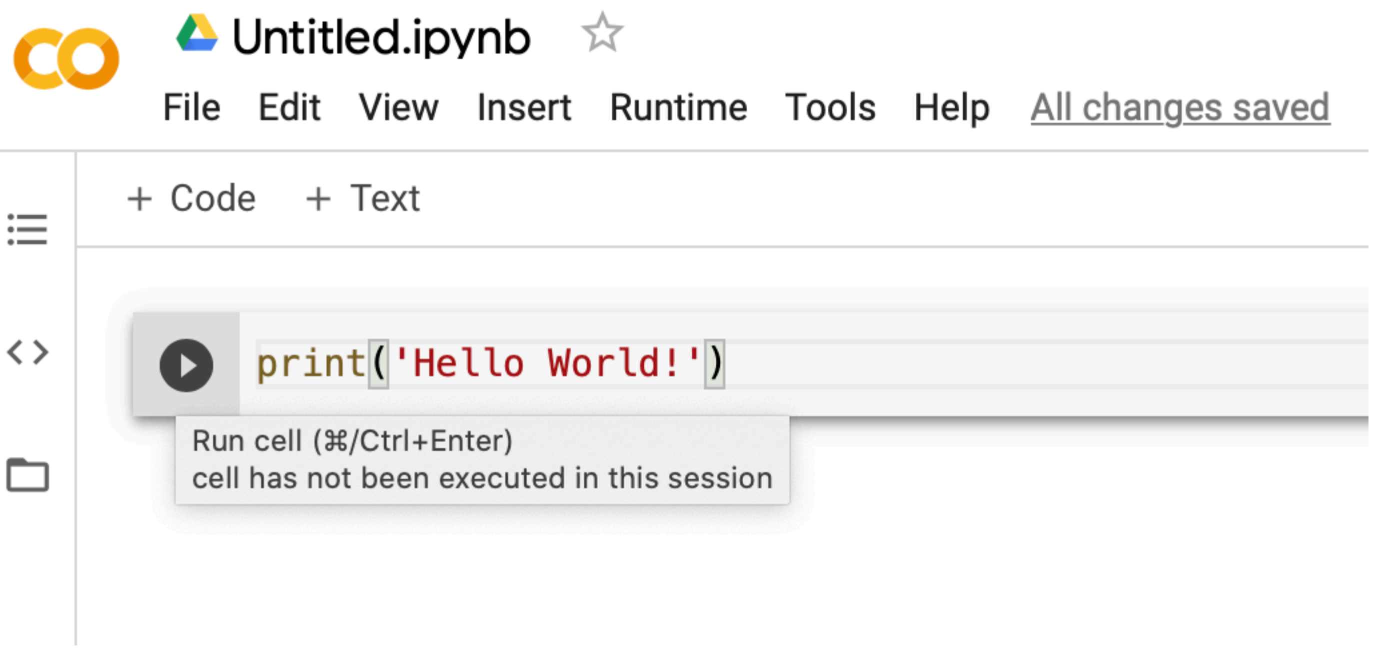
How to run your Python code on a web browser

- The you will see a new page where you can type your code – we call this a “notebook”



How to run your Python code on a web browser

- Lets write a code that prints “Hello world” on the screen to test it
- After typing your code, click on the “run cell” button

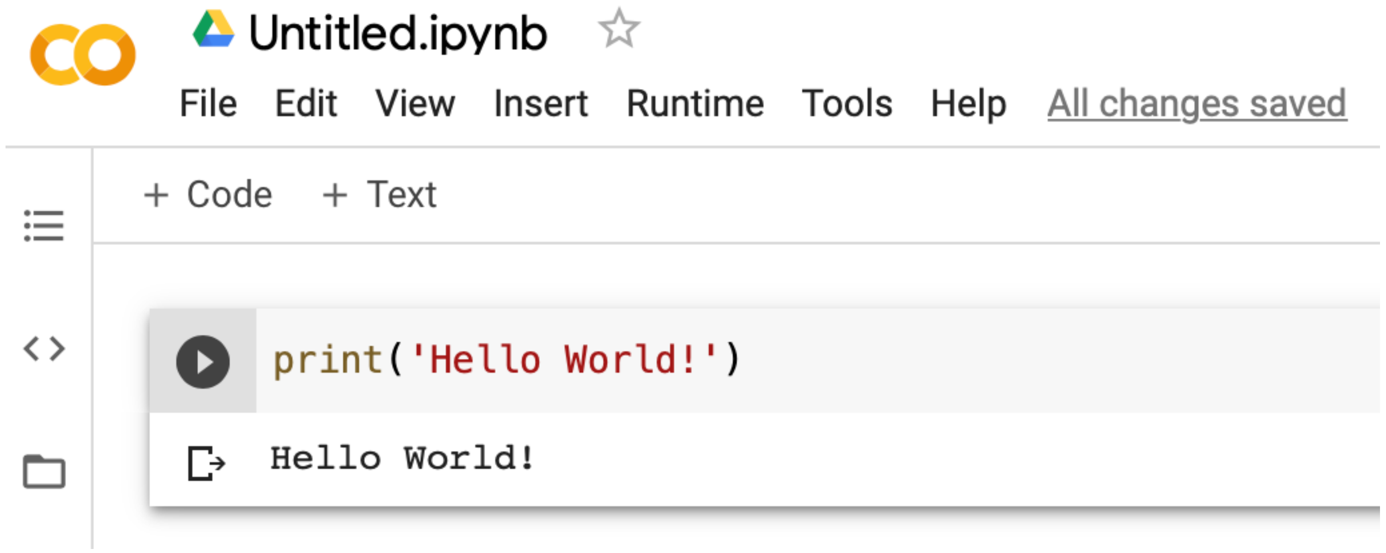


```python
print('Hello World!')
```

Run cell (⌘/Ctrl+Enter)
cell has not been executed in this session

How to run your Python code on a web browser

- After the execution of the code ends, you will see the result on the screen



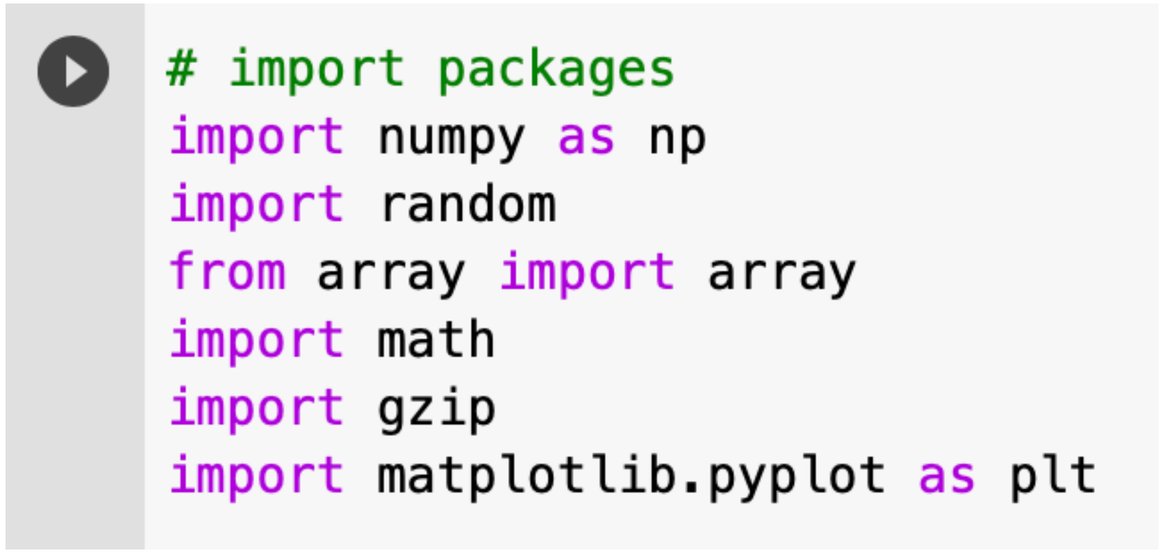
```
print('Hello World!')
```

```
Hello World!
```

Next, we will see how to train a simple machine learning model using Python and the Colab framework

How to train a simple machine learning model

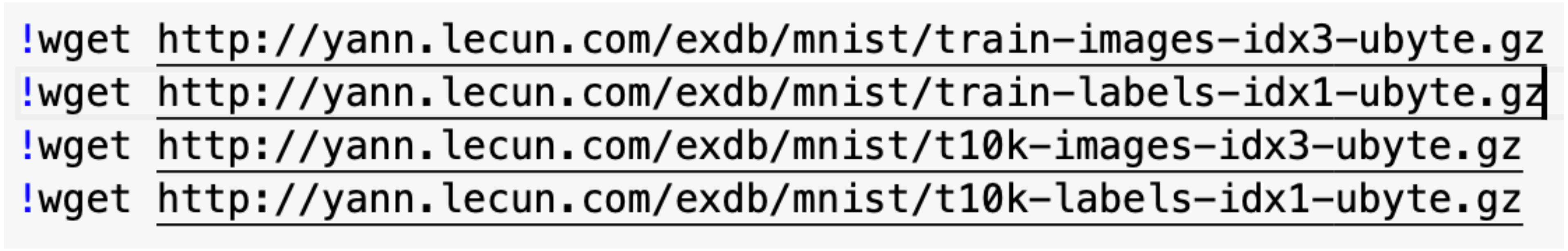
- Now we will use Google Colab for training our model
- In the following, we will go through each step of the code
- First, lets import some Python packages that we will need



```python
# import packages
import numpy as np
import random
from array import array
import math
import gzip
import matplotlib.pyplot as plt
```

How to train a simple machine learning model

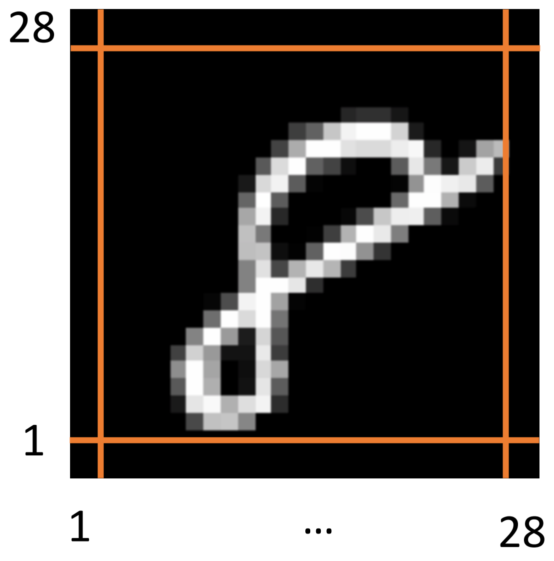
- Next, we will download the handwritten images dataset from http://yann.lecun.com/exdb/mnist/
- This dataset is called the MNIST dataset, and is a popular dataset used in machine learning
- The dataset consists of four files (test images, test labels, train images, train labels)
- The following piece of code will download the dataset



```
!wget http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
!wget http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
!wget http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
!wget http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
```

How to train a simple machine learning model

- The dataset has 60000 images in the training set, and 10000 images in the test set
- Each image in the dataset is an 28x28 image, meaning that it consists of 28 pixels in each dimension
- We will train a model with size 28x28 =784



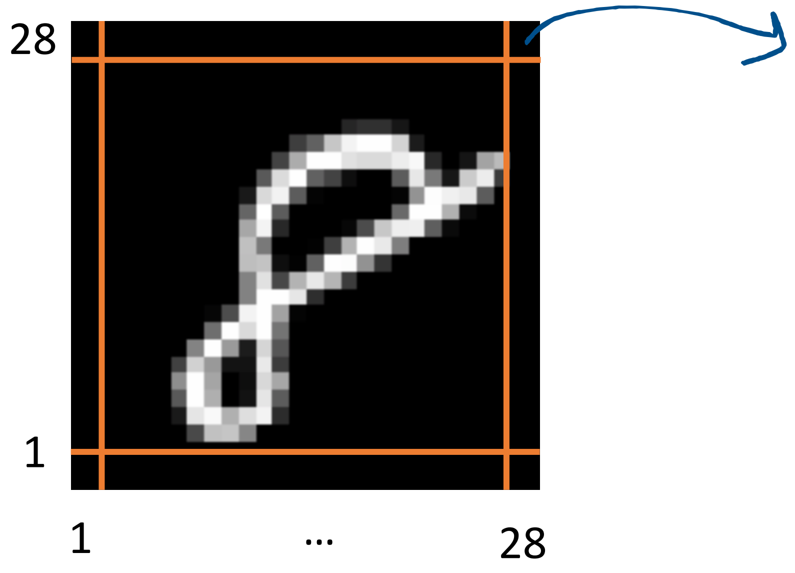
28x 28 image

We will now define these variable in the program:

```
image_size = 28 # each image has dimension 28x28
num_train_images = 60000 # number of images in the training set
num_test_images = 10000 # number of images in the test set
d = image_size*image_size # model size, equal to number of pixels for each image
```

How to train a simple machine learning model

- These are grayscale images, and each pixel value is between 0 and 255, where 0=black, 255=white, and the numbers in between correspond to gray.

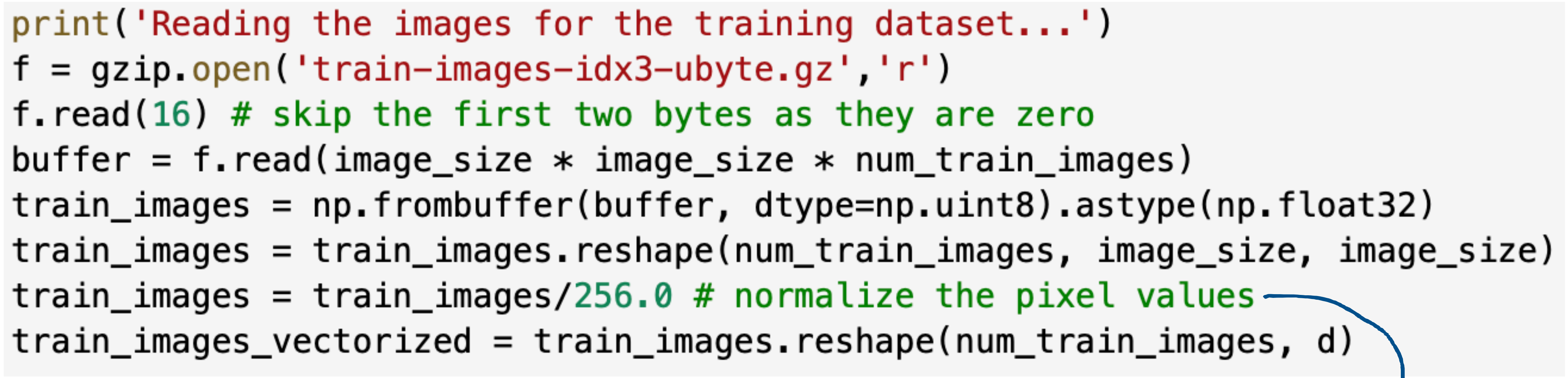


Pixel value = 0 (black)

28x 28 image

How to train a simple machine learning model

- Next, our program will read the contents of the dataset and store them
- First, we will read the contents of the images in the training set
- These correspond to the pixel values of the images in the training set



```python
print('Reading the images for the training dataset...')
f = gzip.open('train-images-idx3-ubyte.gz','r')
f.read(16) # skip the first two bytes as they are zero
buffer = f.read(image_size * image_size * num_train_images)
train_images = np.frombuffer(buffer, dtype=np.uint8).astype(np.float32)
train_images = train_images.reshape(num_train_images, image_size, image_size)
train_images = train_images/256.0 # normalize the pixel values
train_images_vectorized = train_images.reshape(num_train_images, d)
```

This step normalizes the pixel values so that they are between 0 and 1 instead of 0 and 255

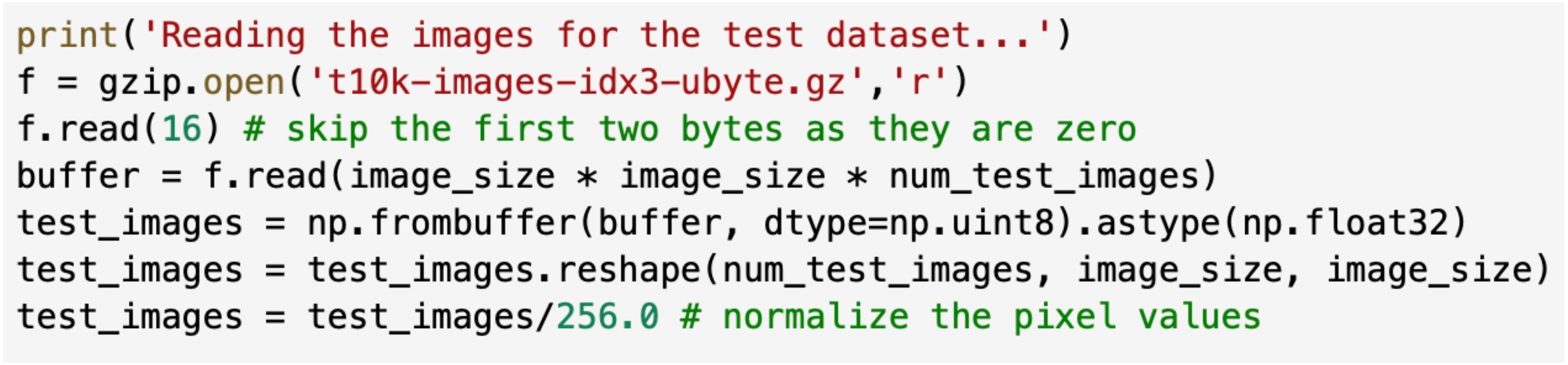
How to train a simple machine learning model

- Next, we will read the labels that correspond to the images in the training set

```
print('Reading the labels for the training dataset...')
f = gzip.open('train-labels-idx1-ubyte.gz','r')
f.read(8)
buffer = f.read(num_train_images)
train_labels = np.frombuffer(buffer, dtype=np.uint8).astype(np.int64)
```

How to train a simple machine learning model

- Then, we will read the contents of the images in the test set



```
print('Reading the images for the test dataset...')
f = gzip.open('t10k-images-idx3-ubyte.gz','r')
f.read(16) # skip the first two bytes as they are zero
buffer = f.read(image_size * image_size * num_test_images)
test_images = np.frombuffer(buffer, dtype=np.uint8).astype(np.float32)
test_images = test_images.reshape(num_test_images, image_size, image_size)
test_images = test_images/256.0 # normalize the pixel values
```

How to train a simple machine learning model

- Finally, we will read the labels that correspond to the images in the test set
- The labels of the test set are needed for computing the accuracy of the model, to check whether the label predicted by the model matches the actual label of the images in the test set

```
print('Reading the labels for the test dataset...')
f = gzip.open('t10k-labels-idx1-ubyte.gz','r')
f.read(8)
buffer = f.read(num_test_images)
test_labels = np.frombuffer(buffer, dtype=np.uint8).astype(np.int64)
```

How to train a simple machine learning model

- We can print some information about our dataset and model
- The following commands will print the model size, number of images in the training set, and the number of images in the test set:

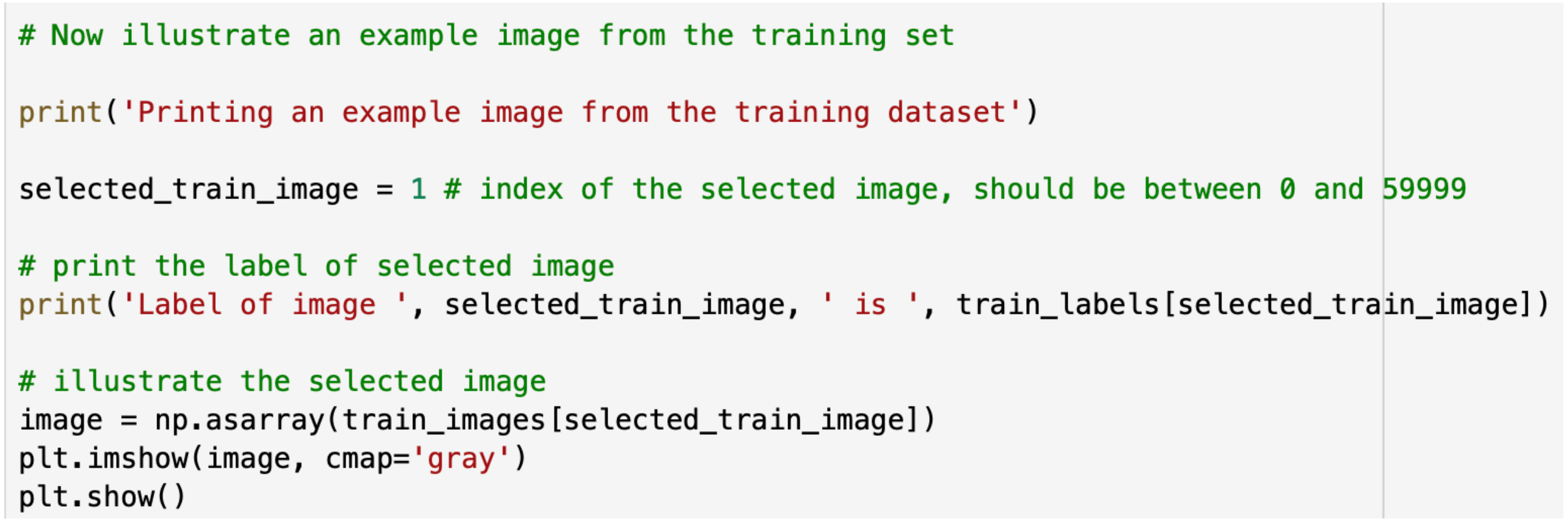
```
print('Model size', d)
print('Number of images in the training set:', num_train_images)
print('Number of images in the test set:', num_test_images)
```

Output of the program after execution

```
Model size 784
Number of images in the training set: 60000
Number of images in the test set: 10000
Printing an example image from the training dataset
```

How to train a simple machine learning model

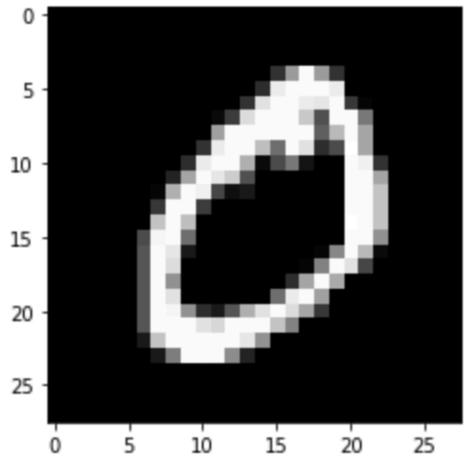
- We can also illustrate an example image from the training set



```
# Now illustrate an example image from the training set

print('Printing an example image from the training dataset')

selected_train_image = 1 # index of the selected image, should be between 0 and 59999

# print the label of selected image
print('Label of image ', selected_train_image, ' is ', train_labels[selected_train_image])

# illustrate the selected image
image = np.asarray(train_images[selected_train_image])
plt.imshow(image, cmap='gray')
plt.show()
```

```
Printing an example image from the training dataset
Label of image  1  is  0
```

Output of the program after execution

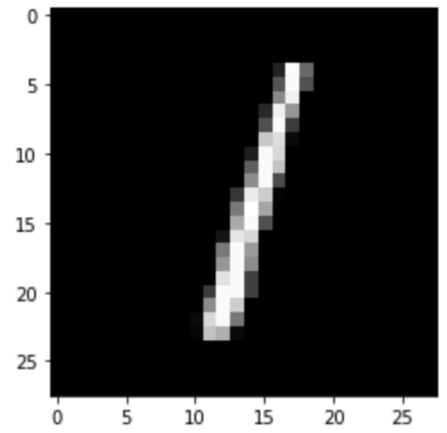


How to train a simple machine learning model

- Similarly, we can illustrate an example image from the test set

```
# Now illustrate an example image from the test set

print('Printing an example image from the test dataset')

selected_test_image = 2 # index of the selected image, should be between 0 and 9999

# print the label of selected image
print('Label of image ', selected_test_image, ' is ', test_labels[selected_test_image])

# illustrate the selected image
image = np.asarray(test_images[selected_test_image])
plt.imshow(image, cmap='gray')
plt.show()
```

```
Printing an example image from the test dataset
Label of image  2  is  1
```



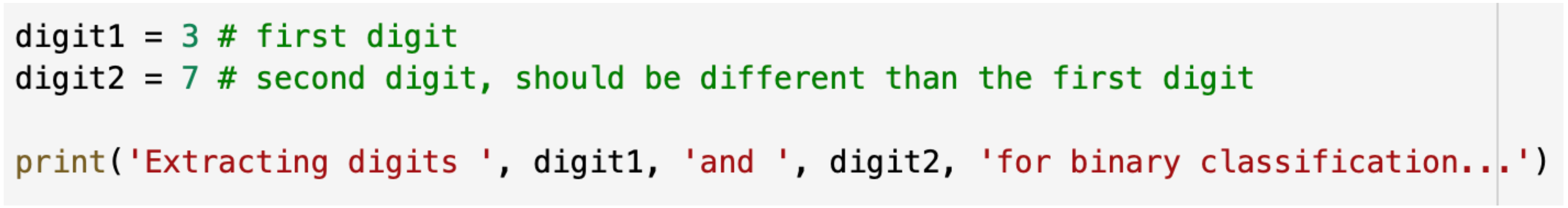
Output of the program after execution

How to train a simple machine learning model

- The original MNIST dataset contains images for 10 digits
- In our example, we will extract only the images corresponding to two specific digits
- Our goal will be to predict the digit of a given image
- In the example below, we extract the images and labels corresponding to digits 3 and 7

How to train a simple machine learning model

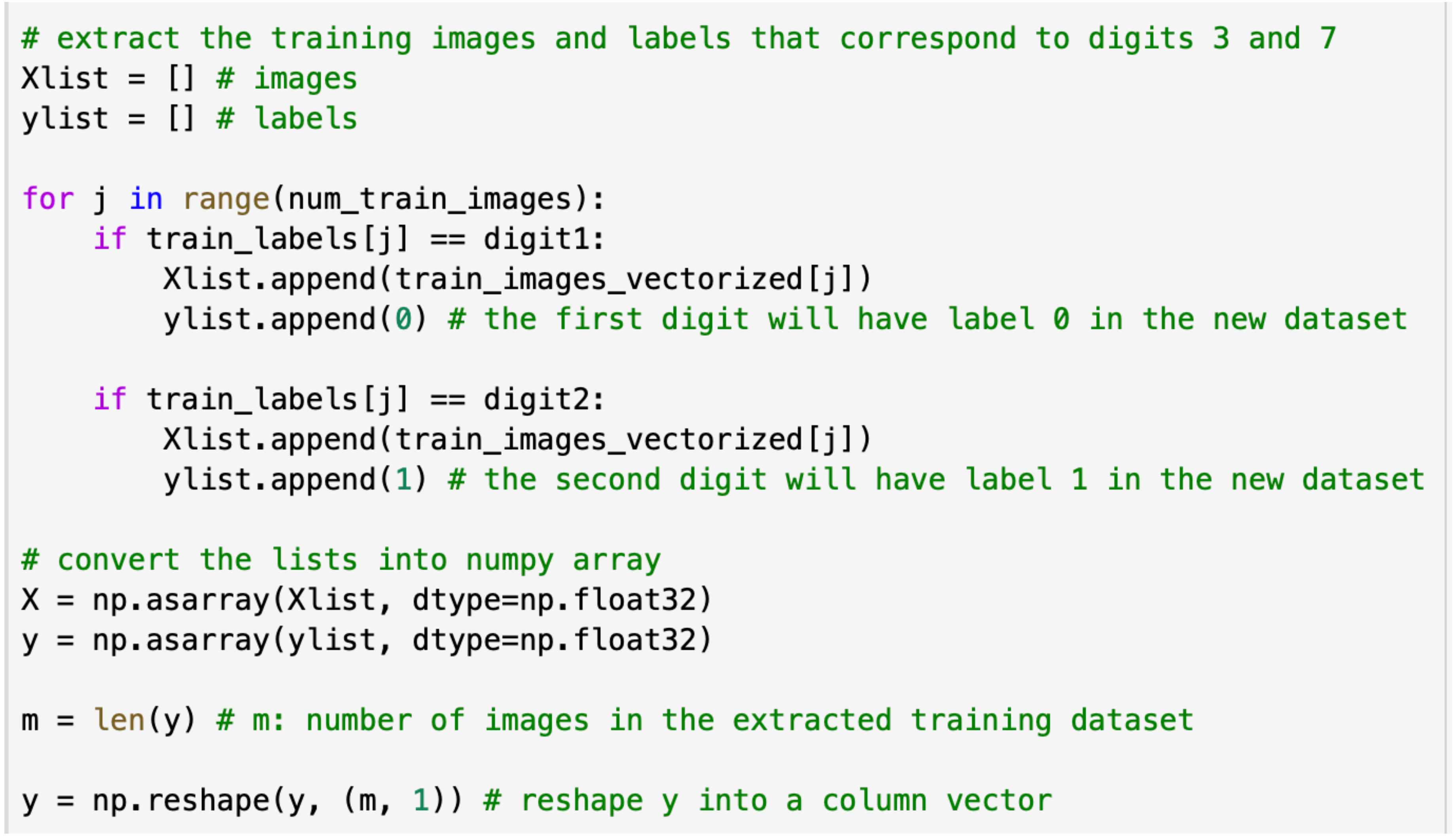
- The original MNIST dataset contains images for 10 digits
- In our example, we will extract only the images corresponding to two specific digits
- Our goal will be to predict the digit of a given image
- In the example below, we extract the images and labels corresponding to digits 3 and 7



```python
digit1 = 3 # first digit
digit2 = 7 # second digit, should be different than the first digit

print('Extracting digits ', digit1, 'and ', digit2, 'for binary classification...')
```

How to train a simple machine learning model

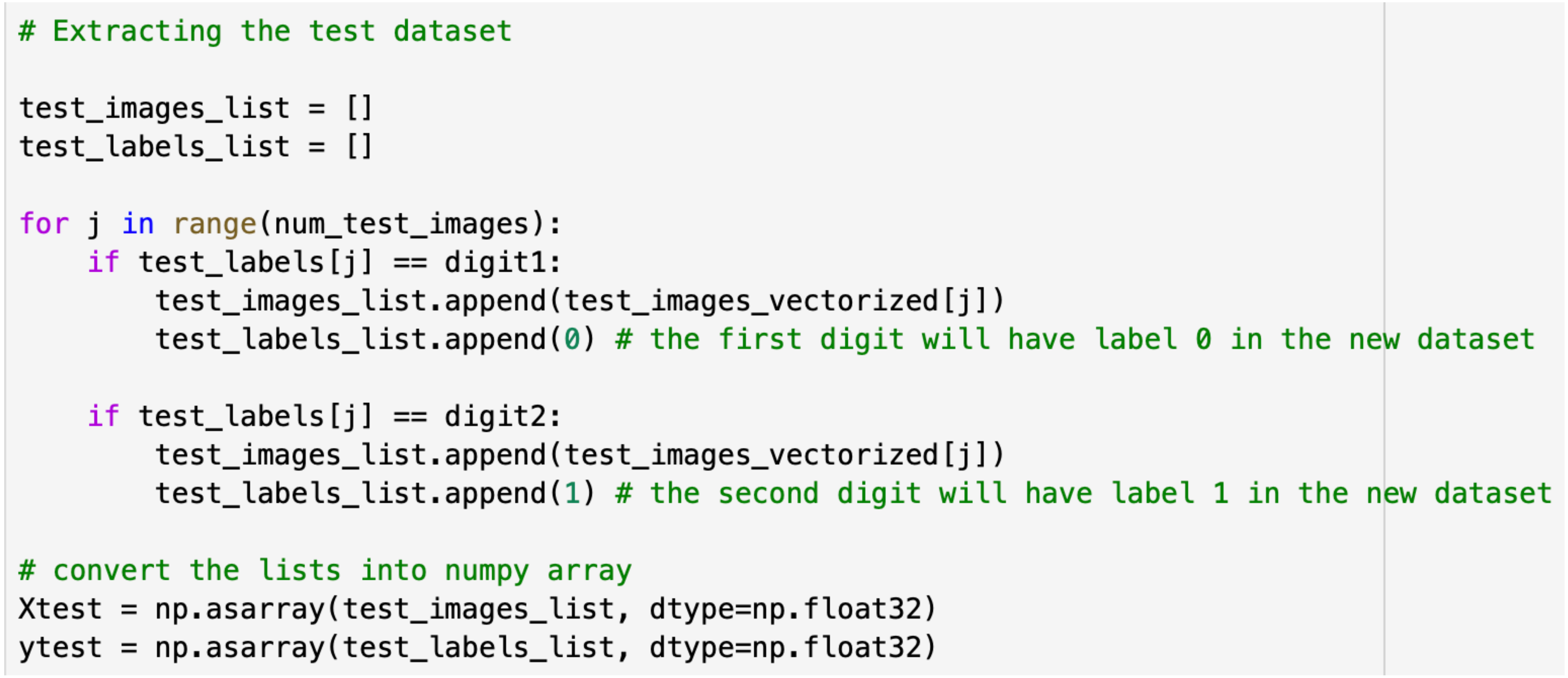
- Extracting the training images and labels



```
# extract the training images and labels that correspond to digits 3 and 7
Xlist = [] # images
ylist = [] # labels

for j in range(num_train_images):
    if train_labels[j] == digit1:
        Xlist.append(train_images_vectorized[j])
        ylist.append(0) # the first digit will have label 0 in the new dataset

    if train_labels[j] == digit2:
        Xlist.append(train_images_vectorized[j])
        ylist.append(1) # the second digit will have label 1 in the new dataset

# convert the lists into numpy array
X = np.asarray(Xlist, dtype=np.float32)
y = np.asarray(ylist, dtype=np.float32)

m = len(y) # m: number of images in the extracted training dataset

y = np.reshape(y, (m, 1)) # reshape y into a column vector
```

How to train a simple machine learning model

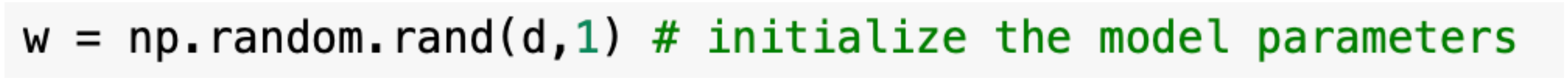
- Extracting the test images and labels



```python
# Extracting the test dataset

test_images_list = []
test_labels_list = []

for j in range(num_test_images):
    if test_labels[j] == digit1:
        test_images_list.append(test_images_vectorized[j])
        test_labels_list.append(0) # the first digit will have label 0 in the new dataset

    if test_labels[j] == digit2:
        test_images_list.append(test_images_vectorized[j])
        test_labels_list.append(1) # the second digit will have label 1 in the new dataset

# convert the lists into numpy array
Xtest = np.asarray(test_images_list, dtype=np.float32)
ytest = np.asarray(test_labels_list, dtype=np.float32)
```

How to train a simple machine learning model

- After we have finished extracting the dataset, we will now start training the model
- First, we will initialize the model randomly



```python
w = np.random.rand(d,1) # initialize the model parameters
```

How to train a simple machine learning model

- We will train the model using a method called “gradient descent”
- “Gradient” is a vector that tells us the direction of the fastest increase of a multivariate function (function that takes several variables as input)
  - Negative of the gradient tells us the direction of the fastest decrease
- The gradient of a function $f(w_1, w_2, \ldots, w_d)$ is defined as:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix} \rightarrow \text{Each element is a partial derivative of f}$$

How to train a simple machine learning model

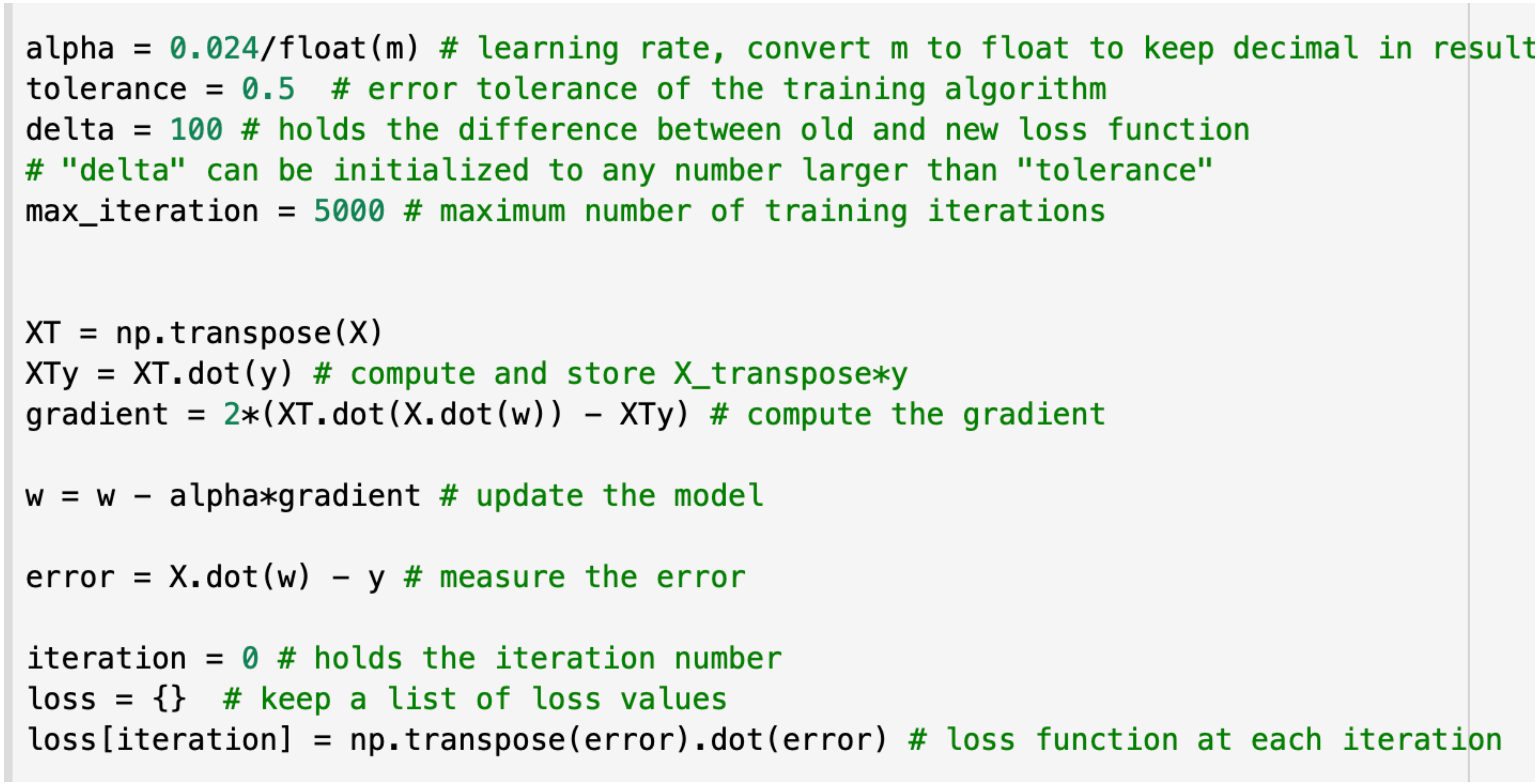
- We will train the model using a method called "gradient descent"
- "Gradient" is a vector that tells us the direction of the fastest increase of a multivariate function (function that takes several variables as input)
  - Negative of the gradient tells us the direction of the fastest decrease
- The gradient of a function $f(w_1, w_2, \ldots, w_d)$ is defined as:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{bmatrix} \rightarrow \text{Each element is a partial derivative of f}$$

In the training problem, the function f will be the loss function and we will minimize the loss function by updating the model in the negative direction of the gradient

How to train a simple machine learning model

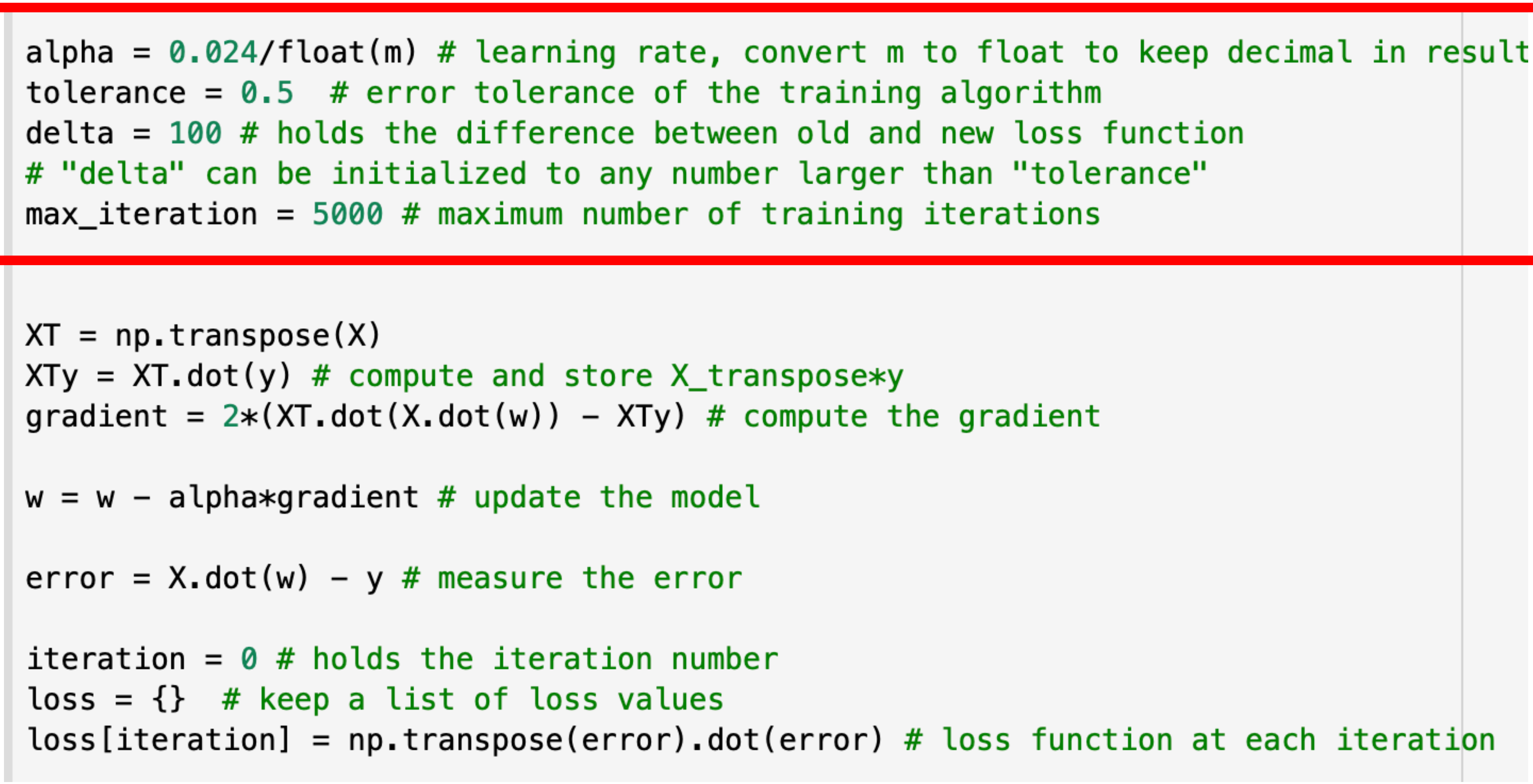
This can be done with the following steps



```
alpha = 0.024/float(m) # learning rate, convert m to float to keep decimal in result
tolerance = 0.5  # error tolerance of the training algorithm
delta = 100 # holds the difference between old and new loss function
# "delta" can be initialized to any number larger than "tolerance"
max_iteration = 5000 # maximum number of training iterations


XT = np.transpose(X)
XTy = XT.dot(y) # compute and store X_transpose*y
gradient = 2*(XT.dot(X.dot(w)) - XTy) # compute the gradient

w = w - alpha*gradient # update the model

error = X.dot(w) - y # measure the error

iteration = 0 # holds the iteration number
loss = {}  # keep a list of loss values
loss[iteration] = np.transpose(error).dot(error) # loss function at each iteration
```

How to train a simple machine learning model

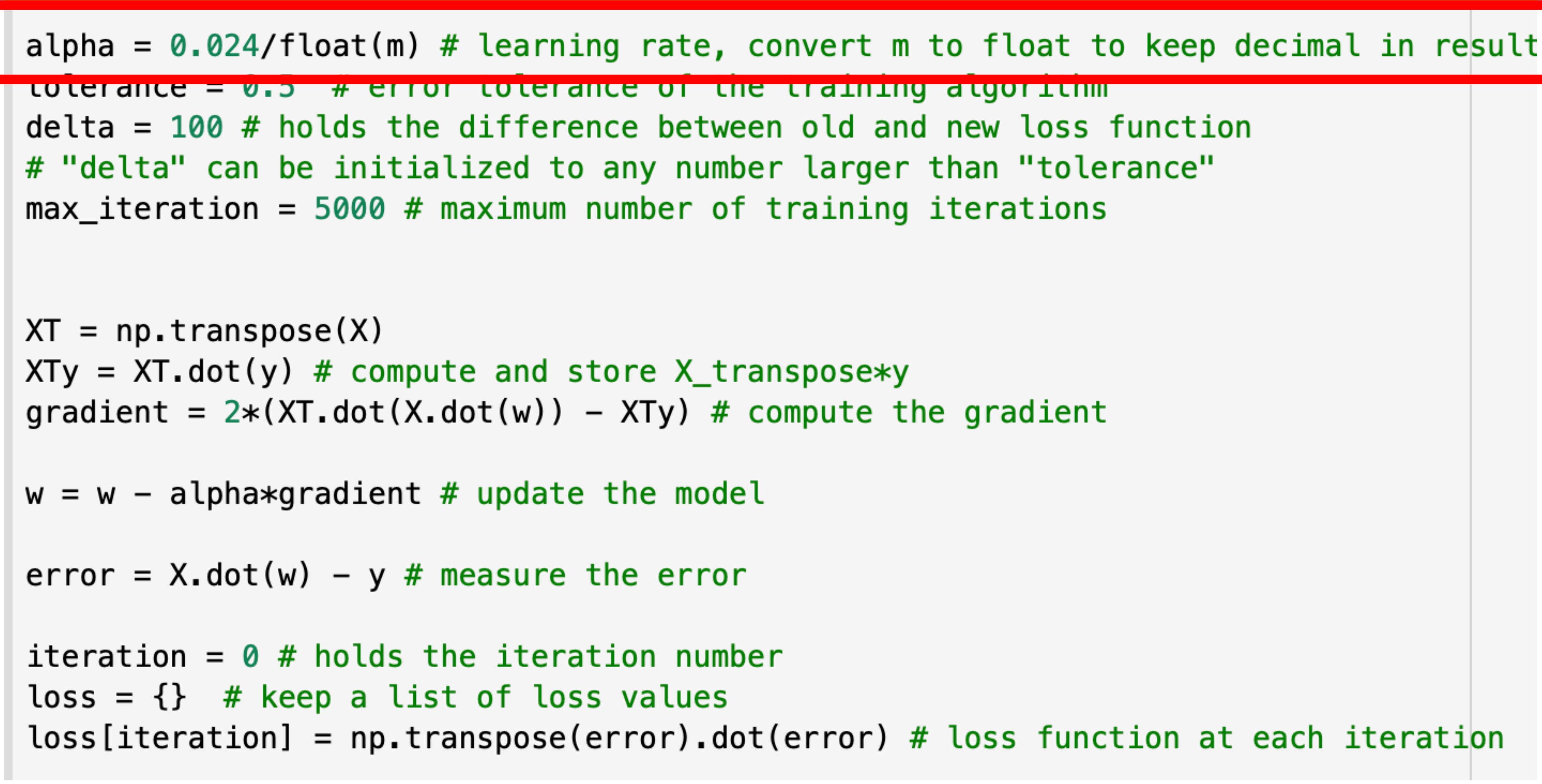
First we initialize some important parameters, such as learning rate and error tolerance of the algorithm



```
alpha = 0.024/float(m) # learning rate, convert m to float to keep decimal in result
tolerance = 0.5  # error tolerance of the training algorithm
delta = 100 # holds the difference between old and new loss function
# "delta" can be initialized to any number larger than "tolerance"
max_iteration = 5000 # maximum number of training iterations


XT = np.transpose(X)
XTy = XT.dot(y) # compute and store X_transpose*y
gradient = 2*(XT.dot(X.dot(w)) - XTy) # compute the gradient

w = w - alpha*gradient # update the model

error = X.dot(w) - y # measure the error

iteration = 0 # holds the iteration number
loss = {}  # keep a list of loss values
loss[iteration] = np.transpose(error).dot(error) # loss function at each iteration
```

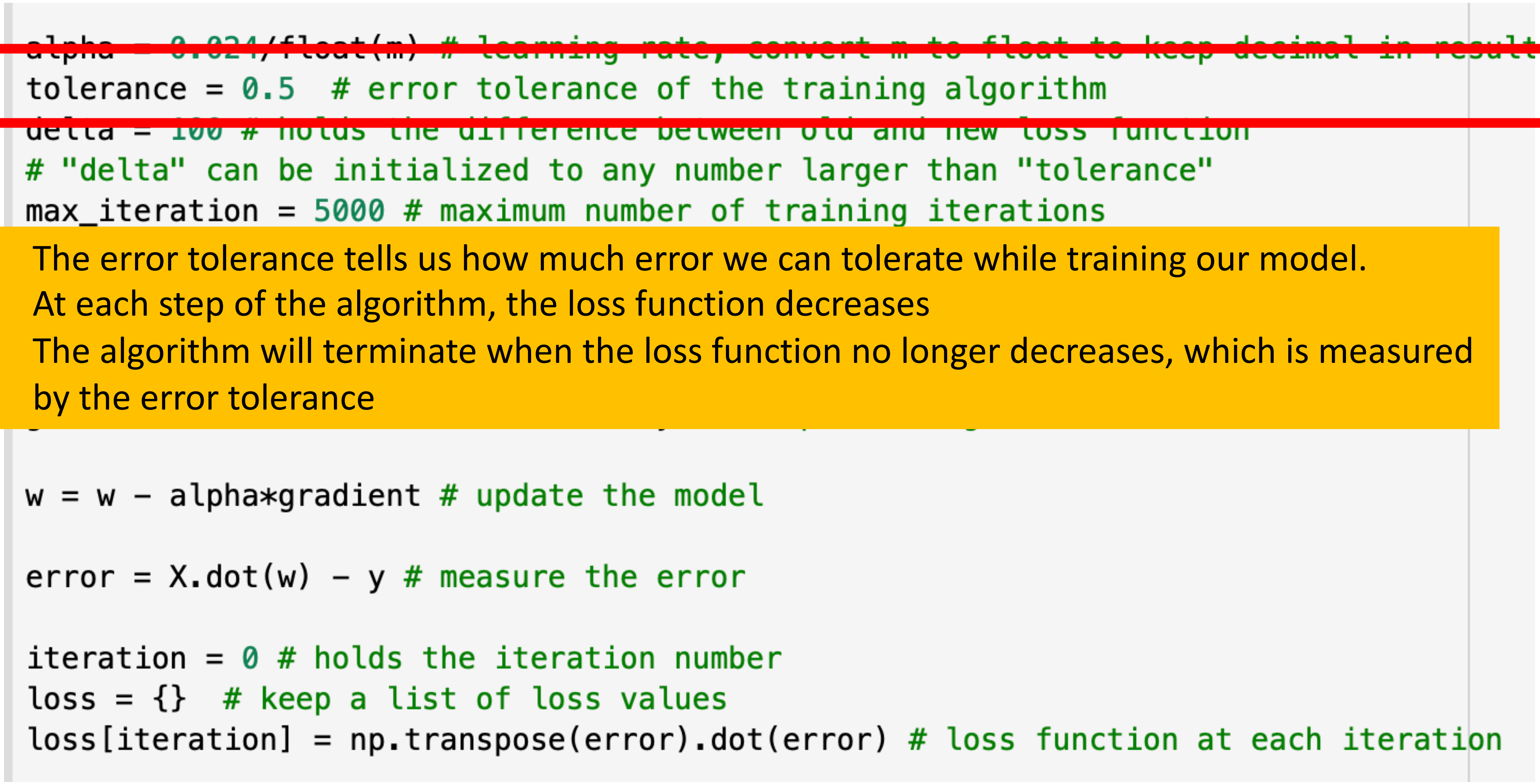
How to train a simple machine learning model

The learning rate tells us how much the model will change from one iteration to another, the larger the learning rate is, the larger the change is



```
alpha = 0.024/float(m) # learning rate, convert m to float to keep decimal in result
tolerance = 0.5  # error tolerance of the training algorithm
delta = 100 # holds the difference between old and new loss function
# "delta" can be initialized to any number larger than "tolerance"
max_iteration = 5000 # maximum number of training iterations


XT = np.transpose(X)
XTy = XT.dot(y) # compute and store X_transpose*y
gradient = 2*(XT.dot(X.dot(w)) - XTy) # compute the gradient

w = w - alpha*gradient # update the model

error = X.dot(w) - y # measure the error

iteration = 0 # holds the iteration number
loss = {}  # keep a list of loss values
loss[iteration] = np.transpose(error).dot(error) # loss function at each iteration
```

How to train a simple machine learning model

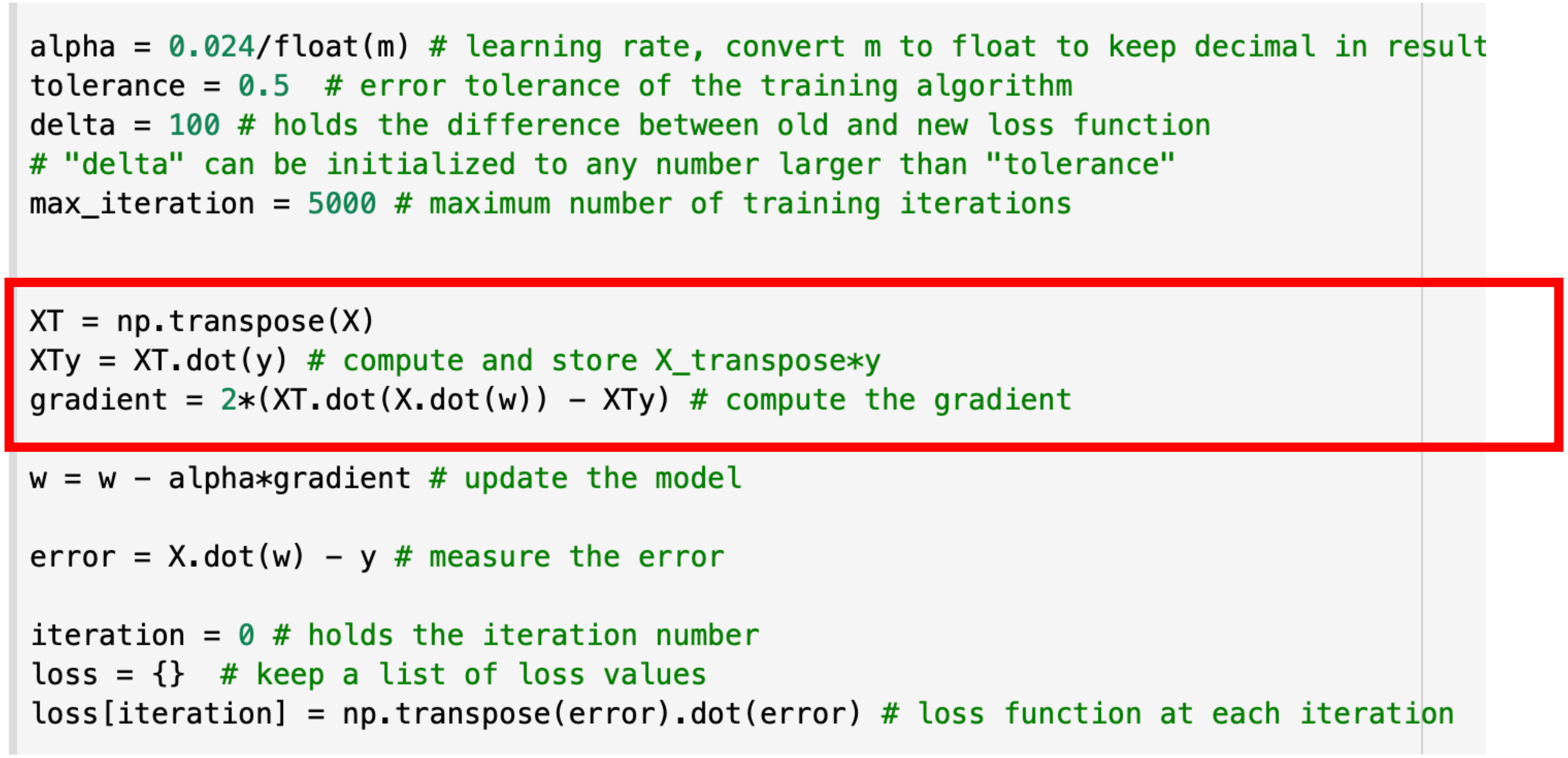


```
alpha = 0.024/float(m) # learning rate, convert m to float to keep decimal in result
tolerance = 0.5  # error tolerance of the training algorithm
delta = 100 # holds the difference between old and new loss function
# "delta" can be initialized to any number larger than "tolerance"
max_iteration = 5000 # maximum number of training iterations

w = w - alpha*gradient # update the model

error = X.dot(w) - y # measure the error

iteration = 0 # holds the iteration number
loss = {}  # keep a list of loss values
loss[iteration] = np.transpose(error).dot(error) # loss function at each iteration
```

- The error tolerance tells us how much error we can tolerate while training our model.
- At each step of the algorithm, the loss function decreases
- The algorithm will terminate when the loss function no longer decreases, which is measured by the error tolerance

How to train a simple machine learning model

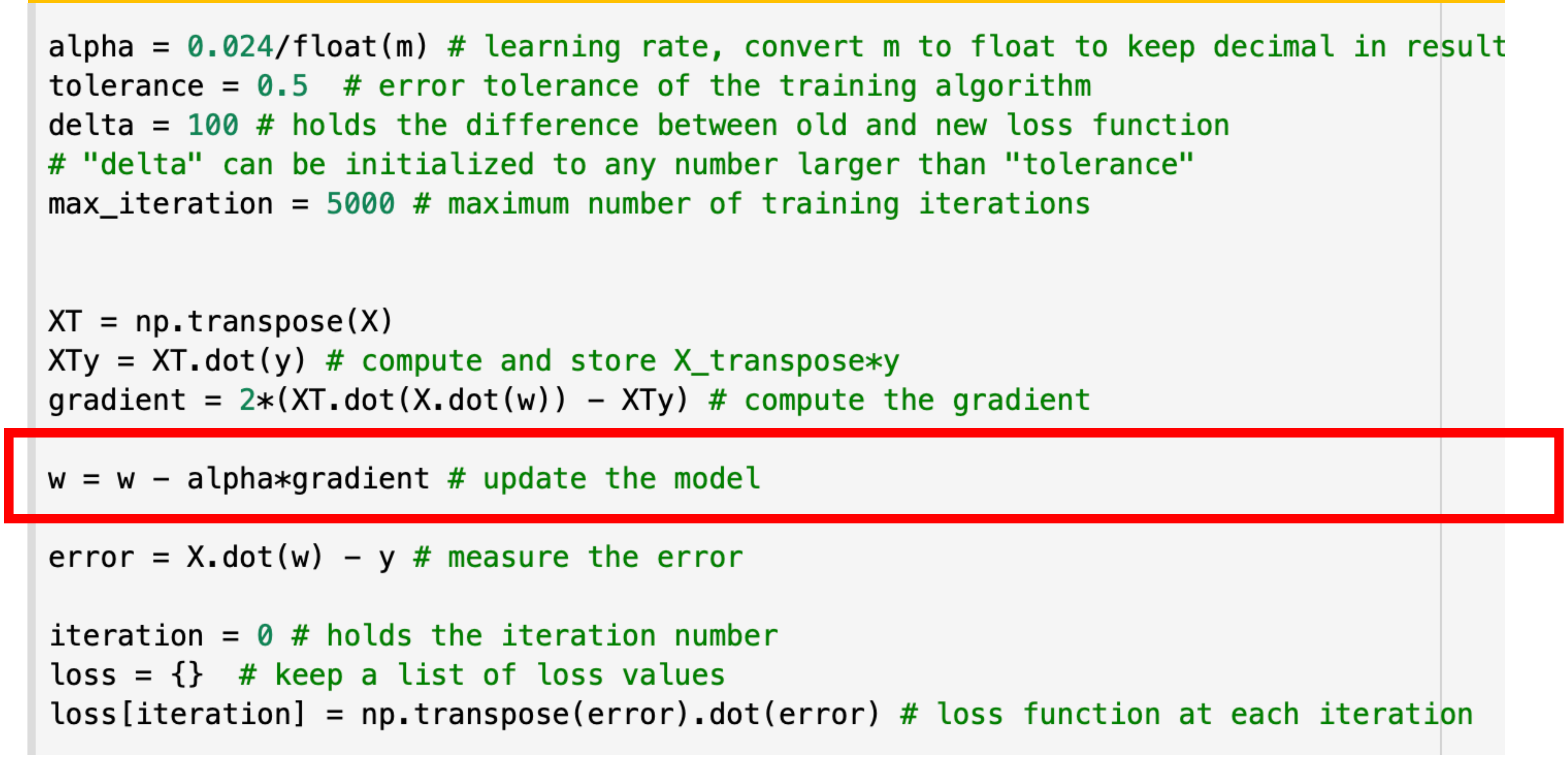
Then we compute the gradient for our loss function, which is the mean-squared error loss.



```
alpha = 0.024/float(m) # learning rate, convert m to float to keep decimal in result
tolerance = 0.5  # error tolerance of the training algorithm
delta = 100 # holds the difference between old and new loss function
# "delta" can be initialized to any number larger than "tolerance"
max_iteration = 5000 # maximum number of training iterations


XT = np.transpose(X)
XTy = XT.dot(y) # compute and store X_transpose*y
gradient = 2*(XT.dot(X.dot(w)) - XTy) # compute the gradient

w = w - alpha*gradient # update the model

error = X.dot(w) - y # measure the error

iteration = 0 # holds the iteration number
loss = {}  # keep a list of loss values
loss[iteration] = np.transpose(error).dot(error) # loss function at each iteration
```

How to train a simple machine learning model

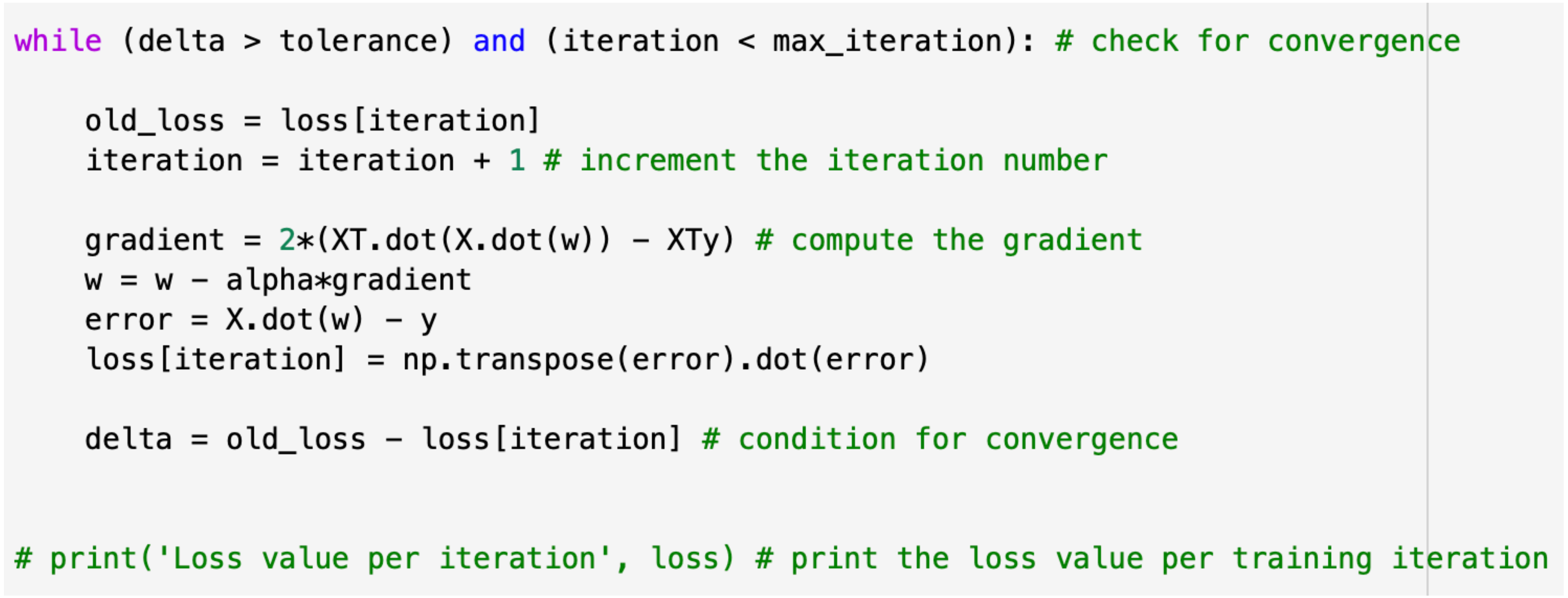
Finally, we update the gradient in the negative direction of the gradient, weighted with a parameter called the learning rate.



```
alpha = 0.024/float(m) # learning rate, convert m to float to keep decimal in result
tolerance = 0.5  # error tolerance of the training algorithm
delta = 100 # holds the difference between old and new loss function
# "delta" can be initialized to any number larger than "tolerance"
max_iteration = 5000 # maximum number of training iterations


XT = np.transpose(X)
XTy = XT.dot(y) # compute and store X_transpose*y
gradient = 2*(XT.dot(X.dot(w)) - XTy) # compute the gradient

w = w - alpha*gradient # update the model

error = X.dot(w) - y # measure the error

iteration = 0 # holds the iteration number
loss = {}  # keep a list of loss values
loss[iteration] = np.transpose(error).dot(error) # loss function at each iteration
```

How to train a simple machine learning model

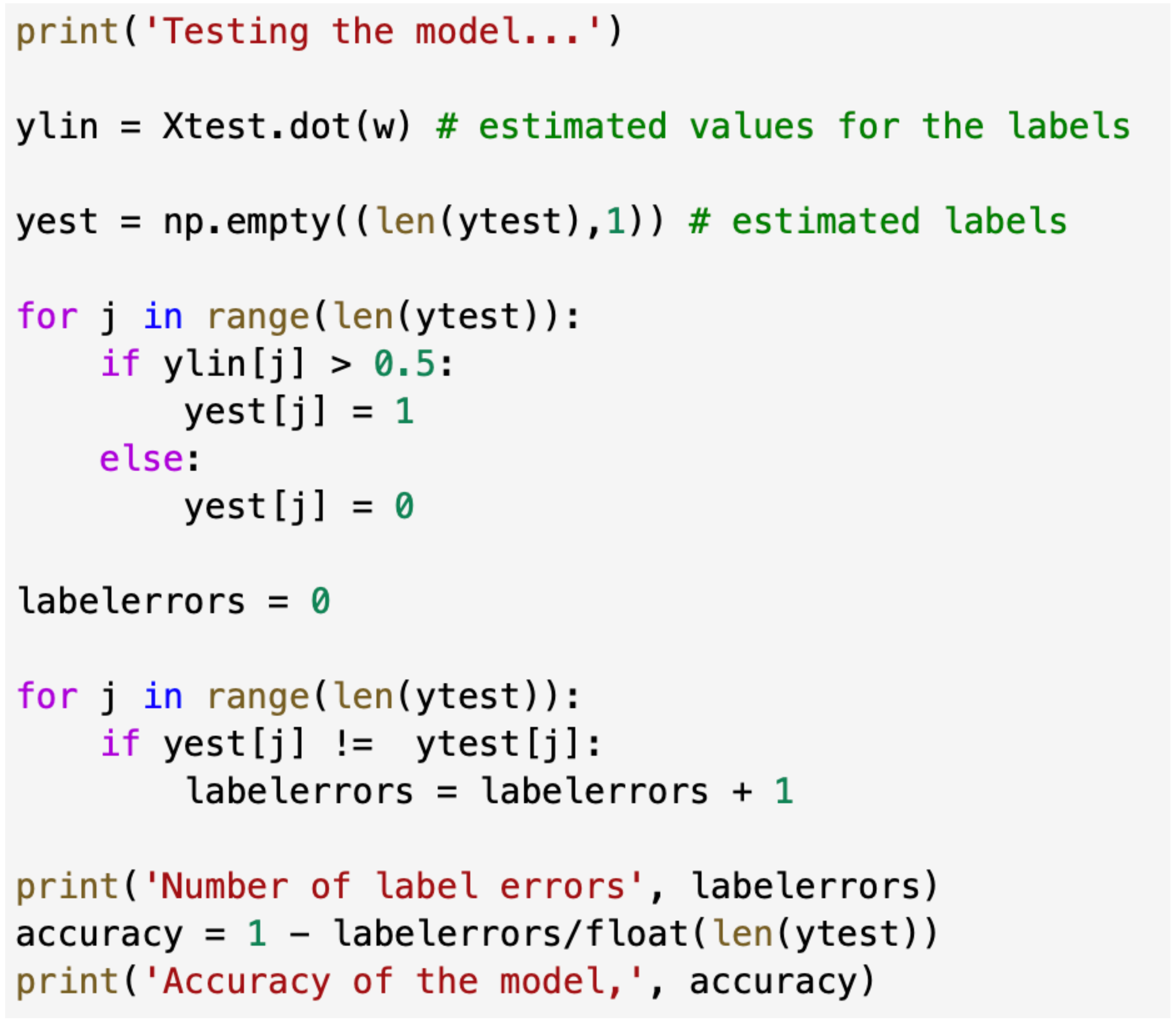
We repeat the gradient computation and model update steps until the algorithm terminates



```
while (delta > tolerance) and (iteration < max_iteration): # check for convergence

    old_loss = loss[iteration]
    iteration = iteration + 1 # increment the iteration number

    gradient = 2*(XT.dot(X.dot(w)) - XTy) # compute the gradient
    w = w - alpha*gradient
    error = X.dot(w) - y
    loss[iteration] = np.transpose(error).dot(error)

    delta = old_loss - loss[iteration] # condition for convergence


# print('Loss value per iteration', loss) # print the loss value per training iteration
```

How to train a simple machine learning model

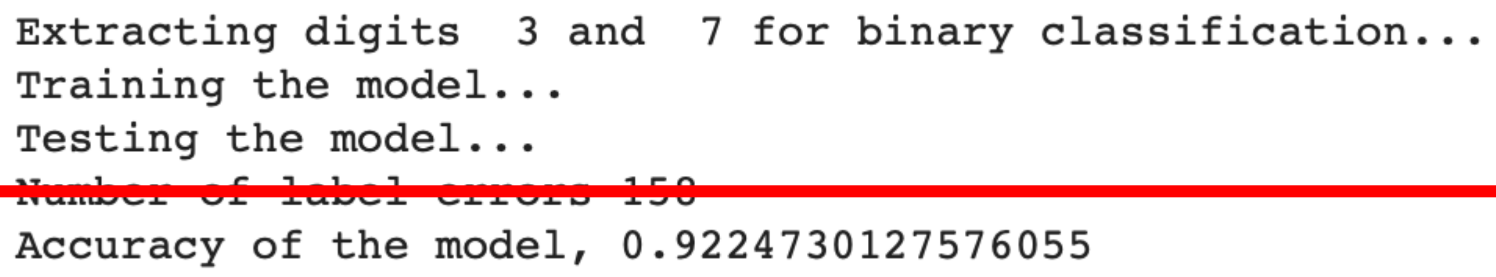
Now we have finished training our model. The final step is to test its accuracy on the test set.



```
print('Testing the model...')

ylin = Xtest.dot(w) # estimated values for the labels

yest = np.empty((len(ytest),1)) # estimated labels

for j in range(len(ytest)):
    if ylin[j] > 0.5:
        yest[j] = 1
    else:
        yest[j] = 0

labelerrors = 0

for j in range(len(ytest)):
    if yest[j] !=  ytest[j]:
        labelerrors = labelerrors + 1

print('Number of label errors', labelerrors)
accuracy = 1 - labelerrors/float(len(ytest))
print('Accuracy of the model,', accuracy)
```

How to train a simple machine learning model

Now we can execute the code, and wait for the program to terminate.
When the program terminates, we will observe the results:



```
Extracting digits  3 and  7 for binary classification...
Training the model...
Testing the model...
Number of label errors 158
Accuracy of the model, 0.9224730127576055
```

Comments

- Remember that the loss function is computed with respect to the training set, whereas the accuracy is measured with respect to the test set.
- That is, we train the model by minimizing the loss function with respect to the training set and then use that trained model to test the accuracy with respect to the test set.
- If our error tolerance is very very small, it means we are "fitting" our model too much to the training data. The model will work perfectly on the training data. For example, if we computed the accuracy with respect to the training data you would get very high accuracy, but the model will lose its flexibility to interpret new examples.
- As a result, our accuracy on the test set will start to decrease, because the model hasn't seen these examples while training.
- This issue is called **overfitting**.
- Ideally, we want our error tolerance to be **small enough but not too small to cause overfitting.**

How are training sets labeled?

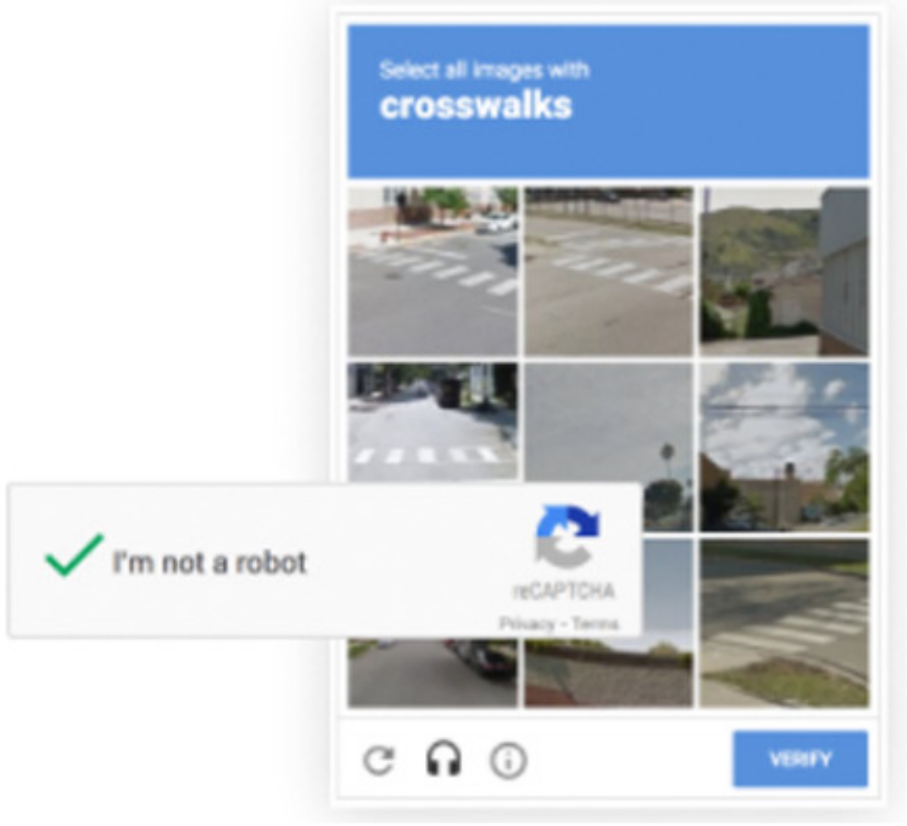
- In this learning module, we learned how to train a linear regression model
- The training set contained both the images and their labels
- But, how are the training sets labeled?

How are training sets labeled?

- Mostly created manually
    - For example, the dataset creator can manually label each image

How are training sets labeled?

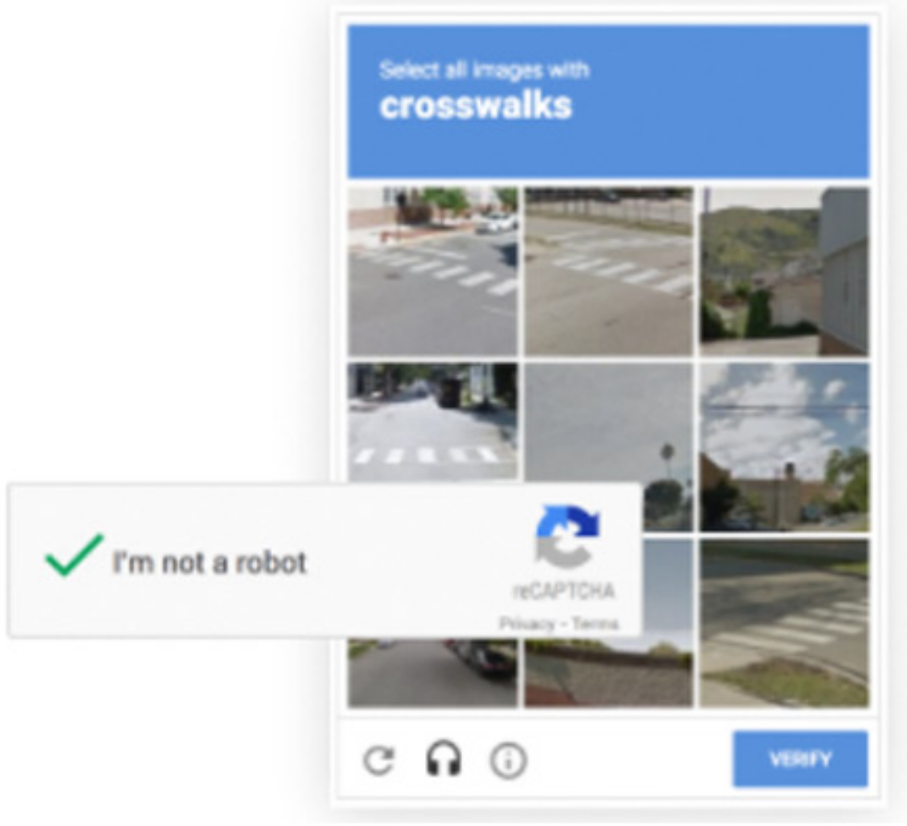
- Mostly created manually
  - For example, the dataset creator can manually label each image
  - Another example is applications such as **reCAPTCHA** – for distinguishing humans from malicious software when someone wants to access a website - the main purpose is security, but it is also used for labling new images in the process



https://www.google.com/recaptcha/about/

How are training sets labeled?

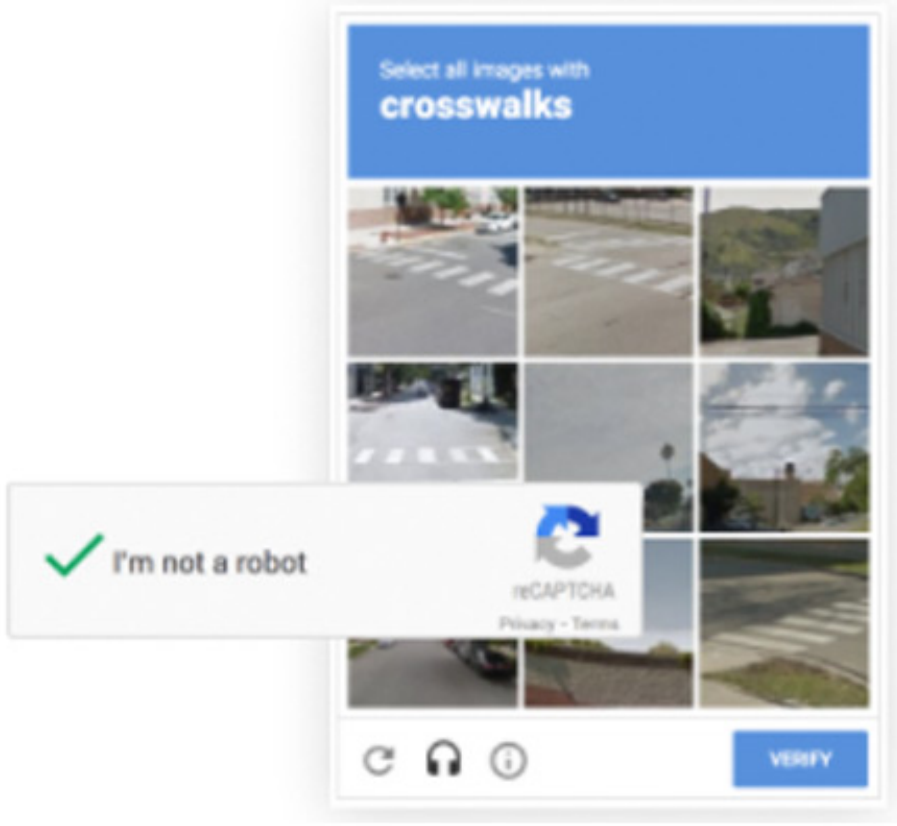
- Mostly created manually
  - For example, the dataset creator can manually label each image
  - Another example is applications such as **reCAPTCHA** – for distinguishing humans from malicious software when someone wants to access a website - the main purpose is security, but it is also used for labling new images in the process



https://www.google.com/recaptcha/about/

How are training sets labeled?

- Mostly created manually
  - For example, the dataset creator can manually label each image
  - Another example is applications such as **reCAPTCHA** – for distinguishing humans from malicious software when someone wants to access a website - the main purpose is security, but it is also used for labling new images in the process



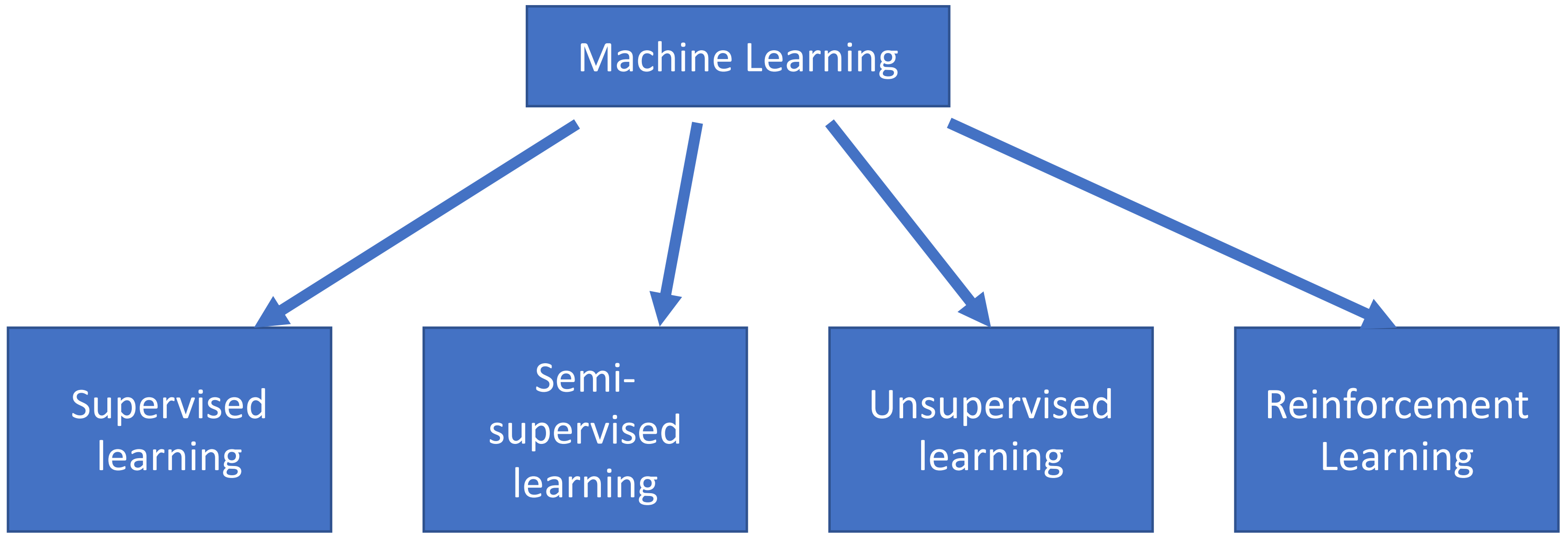
https://www.google.com/recaptcha/about/

How are training sets labeled?

- Data labeling process is relatively easy for simple images such as digits, cars, dogs, but can be very costly and can take weeks in some other applications
- For example, in medical applications, the data may need to be labelled by a medical professional
- In applications where it is hard to label data, we can use different machine learning techniques that requires fewer or no labeled data
    - These techniques use the similarity between different data points to make predictions

Machine Learning

- Overall, we can group machine learning algorithms into four main groups:



All of the data samples (images) in the training set are labeled
E.g., linear/logistic regression, deep neural networks

Some samples (images) in the training set are labeled, but the majority is unlabeled
E.g., graph-based techniques

None of the data samples (images) in the training set are labeled
E.g., clustering algorithms

Learn the optimal action strategy to maximize a reward function by interacting with the environment (trial and error)
Used for motion control in robotics, or to find the best strategy in games like chess, go etc.

Limitations of Machine Learning

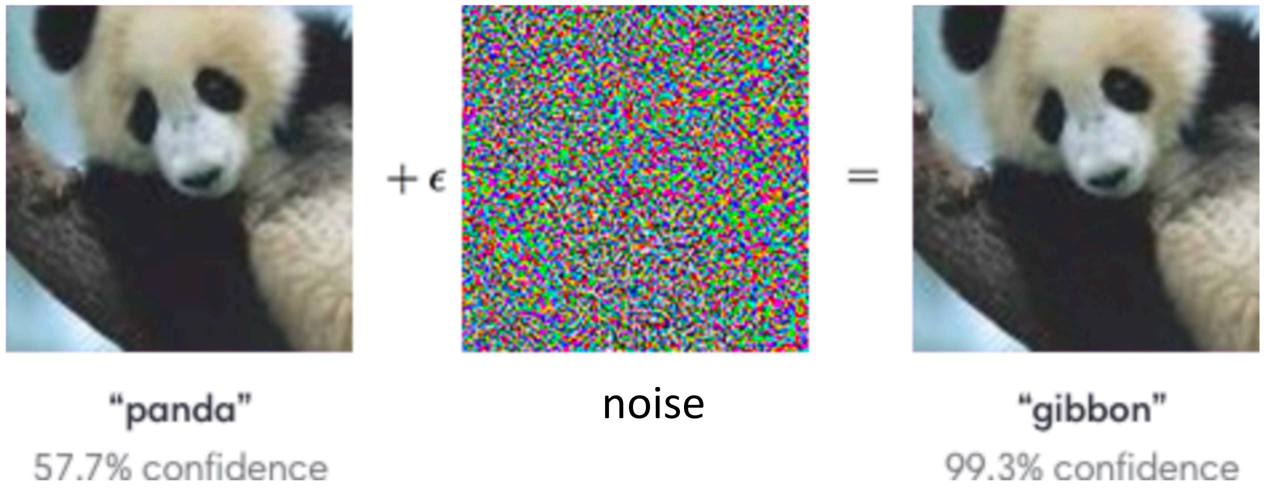
- Next, we will briefly talk about some important limitations and security challenges of machine learning

Bias and Fairness

- As we have seen in our example, the training algorithm is highly dependent on the training dataset
- If the training dataset has low quality, the performance of the training algorithm will be poor
- For instance, suppose that the dataset is dominated by a certain digit while the other digit almost never appears
- Then, the model will favor the samples that are dominant in the training set, and be biased against the underrepresented samples
- This can be a significant problem in real-world applications (e.g. an machine-learning-based tool to automate job application reviews)
- Fairness in machine learning is a very active area of research

Security – Adversarial Attacks

- It is possible to create adversarial inputs to a machine learning system to cause the model to make a mistake
- These are done by adding structured noise to the image that can't be perceived by the human eye but fools the machine learning system



An adversarial input, overlaid on a typical image, can cause a classifier to miscategorize a panda as a gibbon.

from https://openai.com/blog/adversarial-example-research/

Security – Adversarial Attacks

- Similarly, an adversary can fool speech recognition systems by adding structured noise to the speech waveforms

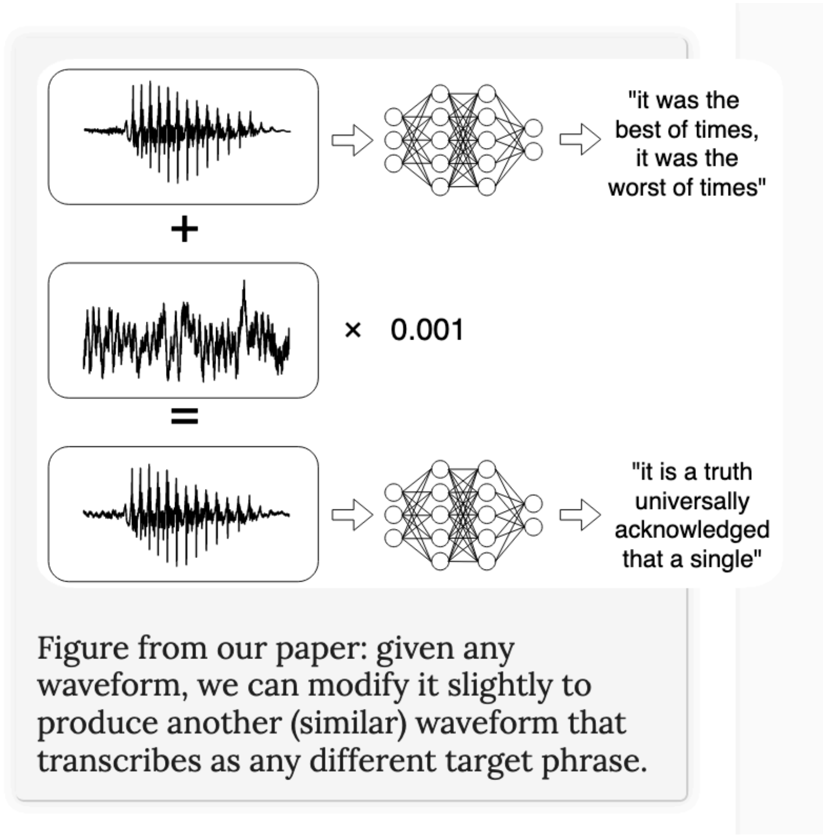
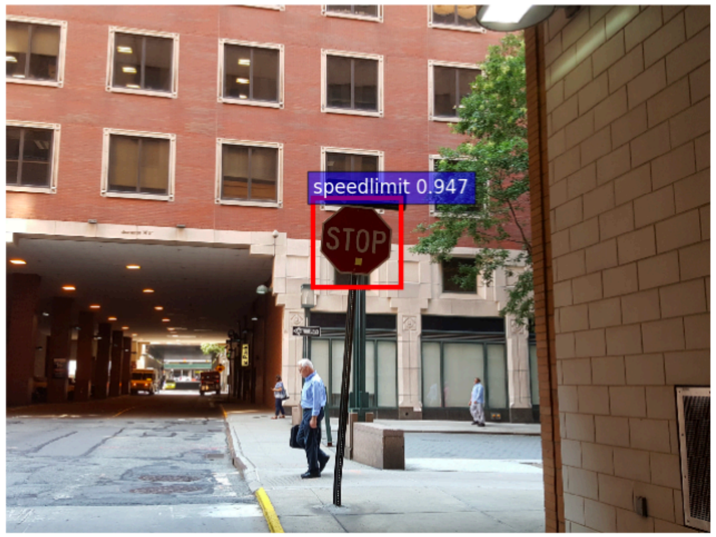

"it was the best of times, it was the worst of times"
+
× 0.001
=
"it is a truth universally acknowledged that a single"

Figure from our paper: given any waveform, we can modify it slightly to produce another (similar) waveform that transcribes as any different target phrase.

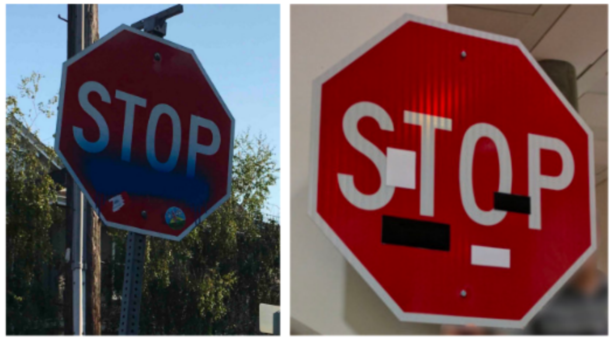
Example from https://nicholas.carlini.com/code/audio_adversarial_examples

Security - Backdoor Attacks

- An adversary can also poison the dataset by placing special “marks” on the images that may appear normal to humans but will cause the machine learning algorithm to make an error



from https://arxiv.org/pdf/1708.06733v1.pdf



https://arxiv.org/pdf/1707.08945.pdf

- The rectangle marks placed on the stop sign causes the machine learning model to classify a stop sign as a speed limit
- This can lead to security and safety risks in applications such as autonomous driving, self-driving cars

A Google Colab implementation is available at: https://towardsdatascience.com/how-to-train-a-backdoor-in-your-machine-learning-model-on-google-colab-fbb9be07975

Next steps to complete the learning module

Can you implement the code described in this learning module on Colab and answer the following questions?

1) How does the accuracy change by changing the learning rate?

2) How does the accuracy change by changing the error tolerance?

3) Can you plot the loss value per iteration and observe its behavior (increasing, decreasing)?

4) Try to run the program a few times, you may notice small changes in the accuracy results. Can you guess why we get different numbers when we run the same algorithm again?

Next steps to complete the learning module

5) Try different pairs of digits, other than 3 and 7.

- Are the results different?
- Are some pairs harder to differentiate than others? Such as 3 and 8?
- Can you try to visualize the images?

6) Can you try to find an example (image) where the model predicts the wrong digit?

7) Can you now flip all the labels in the training dataset but not the test set? How is the accuracy affected?

Next steps to complete the learning module

8) Bonus question:

- In this example we learned how to train a linear regression model.
- The problem we studied here is called binary classification, which means that we want to classify a given data point (in our example an image) between two classes (in our example two digits).
- For these types of problems (i.e., binary classification problems), a more suitable approach is called "logistic regression".
- For the details of logistic regression, please see pages 16-19 in:

  http://cs229.stanford.edu/notes/cs229-notes1.pdf
- Can you try to modify this code for training and testing a logistic regression model and repeat the exercises?

Conclusion

- This concludes our learning module, thanks for your attention and I hope you enjoyed it!