One at a time

Yihui Xie

2019-09-20

Contents

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Part I

Image Recognition

Chapter 1

Logistic Regression on MNIST digits. IDX images - Yunjey

Restart RStudio before running

Source: https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/01-basics/logistic_regression/main.py

1.1 Code in R

The code in R of this example can be found in Chapter **??**.

1.2 Code in Python

```
library(rTorch)
```

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
```

```
# Hyper-parameters
input_size = 784
num_classes = 10
num_epochs = 5
```

```
batch_size = 100
learning_rate = 0.001
```

```
# MNIST dataset (images and labels)
# IDX format
train_dataset = torchvision.datasets.MNIST(root='../../data',
                                           train=True,
                                           transform=transforms.ToTensor(),
                                           download=True)

test_dataset = torchvision.datasets.MNIST(root='../../data',
                                          train=False,
                                          transform=transforms.ToTensor())

# Data loader (input pipeline)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

```
# Logistic regression model
model = nn.Linear(input_size, num_classes)

# Loss and optimizer
# nn.CrossEntropyLoss() computes softmax internally
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Reshape images to (batch_size, input_size)
        images = images.reshape(-1, 28*28)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
#> Epoch [1/5], Step [100/600], Loss: 2.2214
#> Epoch [1/5], Step [200/600], Loss: 2.1097
#> Epoch [1/5], Step [300/600], Loss: 1.9834
#> Epoch [1/5], Step [400/600], Loss: 1.9353
#> Epoch [1/5], Step [500/600], Loss: 1.8226
#> Epoch [1/5], Step [600/600], Loss: 1.7723
#> Epoch [2/5], Step [100/600], Loss: 1.7138
#> Epoch [2/5], Step [200/600], Loss: 1.6539
#> Epoch [2/5], Step [300/600], Loss: 1.6344
#> Epoch [2/5], Step [400/600], Loss: 1.6265
#> Epoch [2/5], Step [500/600], Loss: 1.5405
#> Epoch [2/5], Step [600/600], Loss: 1.4648
#> Epoch [3/5], Step [100/600], Loss: 1.3308
#> Epoch [3/5], Step [200/600], Loss: 1.3706
#> Epoch [3/5], Step [300/600], Loss: 1.3376
#> Epoch [3/5], Step [400/600], Loss: 1.3485
#> Epoch [3/5], Step [500/600], Loss: 1.2913
#> Epoch [3/5], Step [600/600], Loss: 1.3771
#> Epoch [4/5], Step [100/600], Loss: 1.2499
#> Epoch [4/5], Step [200/600], Loss: 1.1708
#> Epoch [4/5], Step [300/600], Loss: 1.2234
#> Epoch [4/5], Step [400/600], Loss: 1.1204
#> Epoch [4/5], Step [500/600], Loss: 1.1811
#> Epoch [4/5], Step [600/600], Loss: 1.1054
#> Epoch [5/5], Step [100/600], Loss: 1.0072
#> Epoch [5/5], Step [200/600], Loss: 1.0944
#> Epoch [5/5], Step [300/600], Loss: 1.0523
#> Epoch [5/5], Step [400/600], Loss: 1.0789
#> Epoch [5/5], Step [500/600], Loss: 1.0850
#> Epoch [5/5], Step [600/600], Loss: 0.9936
```

```
# Test the model
# In test phase, we don't need to compute gradients (for memory efficiency)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.reshape(-1, 28*28)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum()
```

```
    print('Accuracy of the model on the 10000 test images: {} %'.format(100 * correct /
#> Accuracy of the model on the 10000 test images: 82 %
```

```
# Save the model checkpoint
torch.save(model.state_dict(), 'model.ckpt')
```

Chapter 2

R: Digits recognition on IDX images - DeepLearningWizard

Source: https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_logistic_regression/

```
library(rTorch)

torch       <- import("torch")
torchvision <- import("torchvision")
nn          <- import("torch.nn")
transforms  <- import("torchvision.transforms")
dsets       <- import("torchvision.datasets")
builtins    <- import_builtins()
np          <- import("numpy")

batch_size_train <-  64L
```

2.1 Load datasets

2.1.1 Load training dataset

```
train_dataset = dsets$MNIST(root=file.path(".", 'data'),
                            train=TRUE,
                            transform=transforms$ToTensor(),
```

```
                                download=TRUE)

train_dataset
#> Dataset MNIST
#>     Number of datapoints: 60000
#>     Root location: ./data
#>     Split: Train
builtins$len(train_dataset)
#> [1] 60000
```

2.1.2 Introspection

2.1.3 Class and length of `train_dataset`

```
# R
class(train_dataset)
#> [1] "torchvision.datasets.mnist.MNIST"
#> [2] "torchvision.datasets.vision.VisionDataset"
#> [3] "torch.utils.data.dataset.Dataset"
#> [4] "python.builtin.object"
length(train_dataset)
#> [1] 2
```

```
# Python
builtins$type(train_dataset)
#> <class 'torchvision.datasets.mnist.MNIST'>
builtins$len(train_dataset)
#> [1] 60000
py_len(train_dataset)
#> [1] 60000
```

Note that both similar commands produce different results

```
names(train_dataset)
#>  [1] "class_to_idx"     "classes"          "data"
#>  [4] "download"         "extra_repr"       "extract_gzip"
#>  [7] "processed_folder" "raw_folder"       "root"
#> [10] "target_transform" "targets"          "test_data"
#> [13] "test_file"        "test_labels"      "train"
#> [16] "train_data"       "train_labels"     "training_file"
#> [19] "transform"        "transforms"       "urls"
```

```
reticulate::py_list_attributes(train_dataset)
#>  [1] "__add__"                "__class__"
#>  [3] "__delattr__"            "__dict__"
```

```
#>  [5] "__dir__"                "__doc__"
#>  [7] "__eq__"                 "__format__"
#>  [9] "__ge__"                 "__getattribute__"
#> [11] "__getitem__"            "__gt__"
#> [13] "__hash__"               "__init__"
#> [15] "__init_subclass__"      "__le__"
#> [17] "__len__"                "__lt__"
#> [19] "__module__"             "__ne__"
#> [21] "__new__"                "__reduce__"
#> [23] "__reduce_ex__"          "__repr__"
#> [25] "__setattr__"            "__sizeof__"
#> [27] "__str__"                "__subclasshook__"
#> [29] "__weakref__"            "_check_exists"
#> [31] "_format_transform_repr" "_repr_indent"
#> [33] "class_to_idx"           "classes"
#> [35] "data"                   "download"
#> [37] "extra_repr"             "extract_gzip"
#> [39] "processed_folder"       "raw_folder"
#> [41] "root"                   "target_transform"
#> [43] "targets"                "test_data"
#> [45] "test_file"              "test_labels"
#> [47] "train"                  "train_data"
#> [49] "train_labels"           "training_file"
#> [51] "transform"              "transforms"
#> [53] "urls"
```

```
# this is identical to Python len() function
train_dataset$`__len__`()
#> [1] 60000
```

```
# this is not what we are looking for which is torch.Size([1, 28, 28])
# d0 <- train_dataset$data[0][0]
d0 <- train_dataset$data[1][1]
#> Warning in `[.torch.Tensor`(train_dataset$data, 1): Incorrect number of
#> dimensions supplied. The number of supplied arguments, (not counting any
#> NULL, tf$newaxis or np$newaxis) must match thenumber of dimensions in the
#> tensor, unless an all_dims() was supplied (this will produce an error in
#> the future)
#> Warning in `[.torch.Tensor`(train_dataset$data[1], 1): Incorrect number of
#> dimensions supplied. The number of supplied arguments, (not counting any
#> NULL, tf$newaxis or np$newaxis) must match thenumber of dimensions in the
#> tensor, unless an all_dims() was supplied (this will produce an error in
#> the future)
class(d0)
#> [1] "torch.Tensor"          "torch._C._TensorBase"  "python.builtin.object"
d0$size()
```

```
#> torch.Size([28])
```

```
d0 <- train_dataset$data[0][0]
Error: It looks like you might be using 0-based indexing to extract using `[`. The rTor
```

```
# this is identical to train_dataset.data.size() in Python
train_dataset$data$size()
#> torch.Size([60000, 28, 28])
```

```
# this is not a dimension we are looking for either
train_dataset$data[c(1L)][1L]$size()
#> Warning in `[.torch.Tensor`(train_dataset$data, c(1L)): Incorrect number
#> of dimensions supplied. The number of supplied arguments, (not counting any
#> NULL, tf$newaxis or np$newaxis) must match thenumber of dimensions in the
#> tensor, unless an all_dims() was supplied (this will produce an error in
#> the future)
#> Warning in `[.torch.Tensor`(train_dataset$data[c(1L)], 1L): Incorrect
#> number of dimensions supplied. The number of supplied arguments, (not
#> counting any NULL, tf$newaxis or np$newaxis) must match thenumber of
#> dimensions in the tensor, unless an all_dims() was supplied (this will
#> produce an error in the future)
#> torch.Size([28])
```

```
# py = import_builtins()
enum_train_dataset <- builtins$enumerate(train_dataset)
class(enum_train_dataset)
#> [1] "python.builtin.iterator"  "python.builtin.enumerate"
#> [3] "python.builtin.object"
# enum_train_dataset$`__count__`
reticulate::py_list_attributes(enum_train_dataset)
#>  [1] "__class__"          "__delattr__"        "__dir__"
#>  [4] "__doc__"            "__eq__"             "__format__"
#>  [7] "__ge__"             "__getattribute__"   "__gt__"
#> [10] "__hash__"           "__init__"           "__init_subclass__"
#> [13] "__iter__"           "__le__"             "__lt__"
#> [16] "__ne__"             "__new__"            "__next__"
#> [19] "__reduce__"         "__reduce_ex__"      "__repr__"
#> [22] "__setattr__"        "__sizeof__"         "__str__"
#> [25] "__subclasshook__"
```

```
# this is not a number we were expecting
enum_train_dataset$`__sizeof__`()
#> [1] 48
```

```
train_dataset$data$nelement()  # total number of elements in the tensor
#> [1] 47040000
train_dataset$data$shape       # shape
```

```
#> torch.Size([60000, 28, 28])
train_dataset$data$size()      # size
#> torch.Size([60000, 28, 28])
```

```
# get index, label and image
# the pointer will move forward everytime we run the chunk
obj   <- reticulate::iter_next(enum_train_dataset)
idx   <- obj[[1]]        # index number

image <- obj[[2]][[1]]
label <- obj[[2]][[2]]

cat(idx, label, class(label), "\t")
#> 0 5 integer
print(image$size())
#> torch.Size([1, 28, 28])
```

2.1.4 Introspection training dataset

2.1.4.1 Inspecting a single image

So this is how a single image is represented in numbers. It's actually a 28 pixel x 28 pixel image which is why you would end up with this 28x28 matrix of numbers.

2.1.4.2 Inspecting training dataset first element of tuple

This means to access the image, you need to access the first element in the tuple.

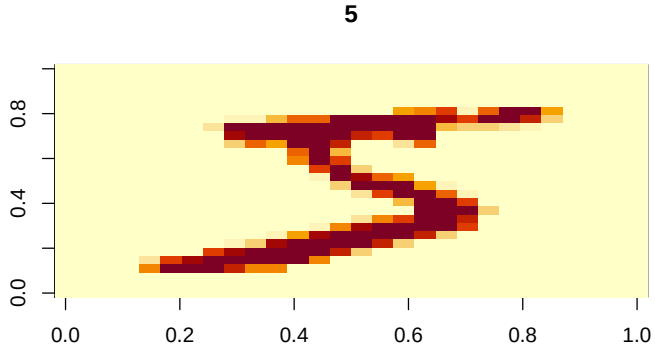
```
# Input Matrix
image$size()
#> torch.Size([1, 28, 28])

# A 28x28 sized image of a digit
# torch.Size([1, 28, 28])
```

2.1.5 MNIST image from training dataset

```
class(image$numpy())
#> [1] "array"
dim(image$numpy())
#> [1]  1 28 28
```

2.1.5.1 Plot one image

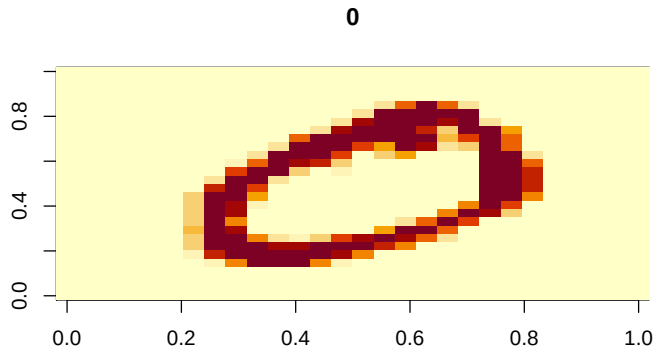
```
rotate <- function(x) t(apply(x, 2, rev))   #function to rotate the matrix

# read label for digit
label
#> [1] 5

# read tensor for image
# img_tensor_2d <- image[0]
img_tensor_2d <- image[1L]
#> Warning in `[.torch.Tensor`(image, 1L): Incorrect number of dimensions
#> supplied. The number of supplied arguments, (not counting any NULL,
#> tf$newaxis or np$newaxis) must match thenumber of dimensions in the tensor,
#> unless an all_dims() was supplied (this will produce an error in the
#> future)
img_tensor_2d$shape       # shape of the 2D tensor: torch.Size([28, 28])
#> torch.Size([28, 28])

# convert tensor to numpy array
img_mat_2d <- img_tensor_2d$numpy()
dim(img_mat_2d)
#> [1] 28 28

# show digit image
image(rotate(img_mat_2d))
title(label)
```



2.1.6 Plot a second image

```
# iterate to the next tensor
obj <- reticulate::iter_next(enum_train_dataset)   # iterator
```

```
idx <- obj[[1]]
img <- obj[[2]][[1]]
lbl <- obj[[2]][[2]]

img_tensor_2d <- img[1]            # get 2D tensor
#> Warning in `[.torch.Tensor`(img, 1): Incorrect number of dimensions
#> supplied. The number of supplied arguments, (not counting any NULL,
#> tf$newaxis or np$newaxis) must match thenumber of dimensions in the tensor,
#> unless an all_dims() was supplied (this will produce an error in the
#> future)
img_mat_2d <- img_tensor_2d$numpy()  # convert to 2D array

# show digit image
image(rotate(img_mat_2d))            # rotate and plot
title(lbl)                           # label as plot title
```



2.1.7 Loading the test dataset

```
test_dataset = dsets$MNIST(root = '../../data',
                           train=FALSE,
                           transform=transforms$ToTensor())

py_len(test_dataset)
#> [1] 10000
```

2.1.7.1 Introspection of the test dataset

```
# we'll get all the attributes of the class
reticulate::py_list_attributes(test_dataset)
#>  [1] "__add__"                "__class__"
#>  [3] "__delattr__"            "__dict__"
```

```
#>  [5] "__dir__"                "__doc__"
#>  [7] "__eq__"                 "__format__"
#>  [9] "__ge__"                 "__getattribute__"
#> [11] "__getitem__"            "__gt__"
#> [13] "__hash__"               "__init__"
#> [15] "__init_subclass__"      "__le__"
#> [17] "__len__"                "__lt__"
#> [19] "__module__"             "__ne__"
#> [21] "__new__"                "__reduce__"
#> [23] "__reduce_ex__"          "__repr__"
#> [25] "__setattr__"            "__sizeof__"
#> [27] "__str__"                "__subclasshook__"
#> [29] "__weakref__"            "_check_exists"
#> [31] "_format_transform_repr" "_repr_indent"
#> [33] "class_to_idx"           "classes"
#> [35] "data"                   "download"
#> [37] "extra_repr"             "extract_gzip"
#> [39] "processed_folder"       "raw_folder"
#> [41] "root"                   "target_transform"
#> [43] "targets"                "test_data"
#> [45] "test_file"              "test_labels"
#> [47] "train"                  "train_data"
#> [49] "train_labels"           "training_file"
#> [51] "transform"              "transforms"
#> [53] "urls"
```

```
# get the Python type
builtins$type(test_dataset$`__getitem__`(0L))   # in Python a tuple gets converted to
#> <class 'list'>
# in Python: type(test_dataset[0]) -> <class 'tuple'>
```

```
# the size of the first and last image tensor
test_dataset$`__getitem__`(0L)[[1]]$size()      # same as test_dataset[0][0].size()
#> torch.Size([1, 28, 28])
test_dataset$`__getitem__`(9999L)[[1]]$size()
#> torch.Size([1, 28, 28])
```

This is the same as:

```
py_to_r(test_dataset)
#> Dataset MNIST
#>     Number of datapoints: 10000
#>     Root location: ../../data
#>     Split: Test
```

```
# the size of the first and last image tensor
# py_get_item(test_dataset, 0L)[[1]]$size()
```

```
py_get_item(test_dataset, 0L)[[1]]$size()
#> torch.Size([1, 28, 28])
py_get_item(test_dataset, 9999L)[[1]]$size()
#> torch.Size([1, 28, 28])
# same as test_dataset[0][0].size()
```

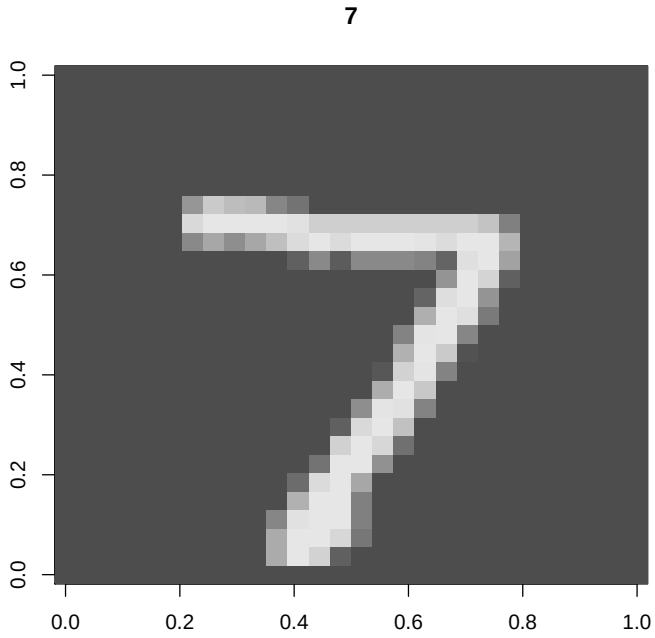
```
# the label is the second list member
label <- test_dataset$`__getitem__`(0L)[[2]]  # in Python: test_dataset[0][1]
label
#> [1] 7
```

2.1.8 Plot image test dataset

```
# convert tensor to numpy array
.show_img <- test_dataset$`__getitem__`(0L)[[1]]$numpy()
dim(.show_img)                                          # numpy 3D array
#> [1]  1 28 28

# reshape 3D array to 2D
show_img <- np$reshape(.show_img, c(28L, 28L))
dim(show_img)
#> [1] 28 28
```

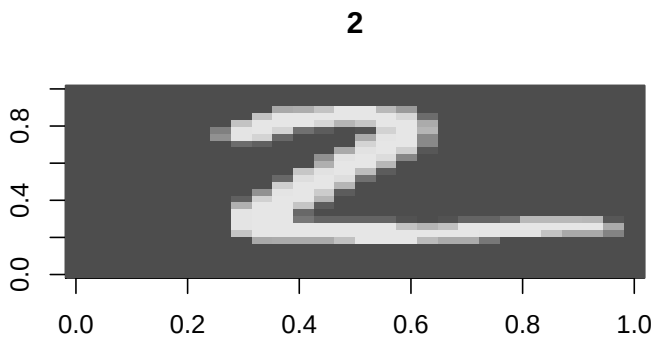
```
# another way to reshape the array
show_img <- np$reshape(test_dataset$`__getitem__`(0L)[[1]]$numpy(), c(28L, 28L))
dim(show_img)
#> [1] 28 28
```

```
# show in grays and rotate
image(rotate(show_img), col = gray.colors(64))
title(label)
```



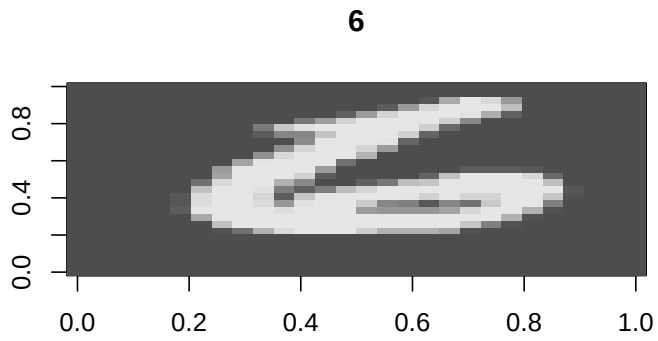
2.1.9 Plot a second test image

```
# next image, index moves from (0L) to (1L), and so on
idx <- 1L
.show_img <- test_dataset$`__getitem__`(idx)[[1]]$numpy()
show_img  <- np$reshape(.show_img, c(28L, 28L))
label     <- test_dataset$`__getitem__`(idx)[[2]]

image(rotate(show_img), col = gray.colors(64))
title(label)
```



2.1.10 Plot the last test image

```
# next image, index moves from (0L) to (1L), and so on
# first image is 0, last image would be 9999
idx <- py_len(test_dataset) - 1L
.show_img <- test_dataset$`__getitem__`(idx)[[1]]$numpy()
show_img  <- np$reshape(.show_img, c(28L, 28L))
label     <- test_dataset$`__getitem__`(idx)[[2]]

image(rotate(show_img), col = gray.colors(64))
title(label)
```



2.2 Defining epochs

When the model goes through the whole 60k images once, learning how to classify 0-9, it's consider **1 epoch**.

However, there's a concept of batch size where it means the model would look at 100 images before updating the model's weights, thereby learning. When the model updates its weights (parameters) after looking at all the images, this is considered 1 iteration.

```
batch_size <- 100L
```

We arbitrarily set 3000 iterations here which means the model would update 3000 times.

```
n_iters <- 3000L
```

One epoch consists of 60,000 / 100 = 600 iterations. Because we would like to go through 3000 iterations, this implies we would have 3000 / 600 = 5 epochs as each epoch has 600 iterations.

```
num_epochs = n_iters / (py_len(train_dataset) / batch_size)
num_epochs = as.integer(num_epochs)
```

```
num_epochs
#> [1] 5
```

2.3 Create iterable objects: training and testing dataset

```
train_loader = torch$utils$data$DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=TRUE)
```

```
# Iterable object
test_loader = torch$utils$data$DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=FALSE)
```

2.3.1 Check iteraibility

```
collections <- import("collections")

builtins$isinstance(train_loader, collections$Iterable)
#> [1] TRUE
builtins$isinstance(test_loader, collections$Iterable)
#> [1] TRUE
```

2.4 Building the model

```
# Same as linear regression!
main <- py_run_string(
"
import torch.nn as nn

class LogisticRegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LogisticRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        out = self.linear(x)
        return out
```

```
")

# build a Linear Rgression model
LogisticRegressionModel <- main$LogisticRegressionModel
```

2.4.1 Instantiate model class based on input and out dimensions

```
# feeding the model with 28x28 images
input_dim = 28L*28L

# classify digits 0-9 a total of 10 classes,
output_dim = 10L

model = LogisticRegressionModel(input_dim, output_dim)
```

2.4.2 Instantiate Cross Entropy Loss class

```
# need Cross Entropy Loss to calculate loss before we backpropagation
criterion = nn$CrossEntropyLoss()
```

2.4.3 Instantiate Optimizer class

Similar to what we've covered above, this calculates the parameters' gradients and update them subsequently.

```
# calculate parameters' gradients and update
learning_rate = 0.001

optimizer = torch$optim$SGD(model$parameters(), lr=learning_rate)
```

2.4.4 Parameters introspection

You'll realize we have 2 sets of parameters, 10x784 which is A and 10x1 which is b in the $y = AX + b$ equation, where X is our input of size 784.

We'll go into details subsequently how these parameters interact with our input to produce our 10x1 output.

```
# Type of parameter object
print(model$parameters())
```

```
#> <generator object Module.parameters at 0x7f309296df10>
model_parameters <- builtins$list(model$parameters())

# Length of parameters
print(builtins$len(model_parameters))
#> [1] 2

# FC 1 Parameters
builtins$list(model_parameters)[[1]]$size()
#> torch.Size([10, 784])

# FC 1 Bias Parameters
builtins$list(model_parameters)[[2]]$size()
#> torch.Size([10])
```

```
builtins$len(builtins$list(model$parameters()))
#> [1] 2
```

2.5 Train the model and test per epoch

```
iter = 0

for (epoch in 1:num_epochs) {
  iter_train_dataset <- builtins$enumerate(train_loader) # convert to iterator
  for (obj in iterate(iter_train_dataset)) {
      # get the tensors for images and labels
      images <- obj[[2]][[1]]
      labels <- obj[[2]][[2]]

      # Reshape images to (batch_size, input_size)
      images <- images$view(-1L, 28L*28L)$requires_grad_()

      # Clear gradients w.r.t. parameters
      optimizer$zero_grad()

      # Forward pass to get output/logits
      outputs = model(images)

      # Calculate Loss: softmax --> cross entropy loss
      loss = criterion(outputs, labels)

      # Getting gradients w.r.t. parameters
      loss$backward()
```

```
      # Updating parameters
      optimizer$step()

      iter = iter + 1

      if (iter %% 500 == 0) {
          # Calculate Accuracy
          correct = 0
          total = 0

          # Iterate through test dataset
          iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
          for (obj2 in iterate(iter_test_dataset)) {
              # Load images to a Torch Variable
              images <- obj2[[2]][[1]]
              labels <- obj2[[2]][[2]]
              images <- images$view(-1L, 28L*28L)$requires_grad_()

              # Forward pass only to get logits/output
              outputs = model(images)

              # Get predictions from the maximum value
              .predicted = torch$max(outputs$data, 1L)
              predicted <- .predicted[1L]

              # Total number of labels
              total = total + labels$size(0L)

              # Total correct predictions
              correct = correct + sum((predicted$numpy() == labels$numpy()))
          }
          accuracy = 100 * correct / total

          # Print Loss
          cat(sprintf('Iteration: %5d. Loss: %f. Accuracy: %8.2f \n',
                      iter, loss$item(), accuracy))
      }
  }
}
#> Iteration:   500. Loss: 1.881020. Accuracy:    70.27
#> Iteration:  1000. Loss: 1.517125. Accuracy:    77.97
#> Iteration:  1500. Loss: 1.354405. Accuracy:    80.60
#> Iteration:  2000. Loss: 1.211496. Accuracy:    81.69
#> Iteration:  2500. Loss: 1.132213. Accuracy:    82.59
#> Iteration:  3000. Loss: 1.019603. Accuracy:    83.38
```

2.6 Break down accuracy calculation

As we've trained our model, we can extract the accuracy calculation portion to understand what's happening without re-training the model.

This would print out the output of the model's predictions on your notebook.

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()

    # Forward pass only to get logits/output
    outputs = model(images)

    if (iter_test == 1) {
        print('OUTPUTS')
        print(outputs)
    }
    # Get predictions from the maximum value
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]
}
#> [1] "OUTPUTS"
#> tensor([[-2.0479e-01, -1.3593e+00, -3.0488e-01, -1.4720e-01,  4.0833e-02,
#>          -5.3239e-01, -9.8131e-01,  2.8635e+00, -4.0464e-01,  9.0220e-01],
#>         [ 3.2250e-01,  6.8070e-02,  1.4779e+00,  1.0052e+00, -1.7874e+00,
#>           8.2865e-01,  1.0045e+00, -1.9955e+00,  4.5074e-01, -1.3909e+00],
#>         [-8.4276e-01,  2.2414e+00,  1.2184e-01, -2.9651e-02, -5.4253e-01,
#>          -3.3983e-01, -1.5156e-01, -2.3036e-01,  2.2024e-01, -3.3406e-01],
#>         [ 2.9647e+00, -2.5175e+00, -8.8717e-02, -2.4168e-01, -9.3389e-01,
#>           6.5263e-01,  1.1432e+00,  1.9990e-01, -6.4065e-01, -1.2482e-01],
#>         [-1.4304e-01, -2.0405e+00,  3.6333e-01, -8.0922e-01,  1.7364e+00,
#>          -6.6911e-01,  2.1178e-01,  4.0157e-01, -1.0758e-01,  7.5692e-01],
#>         [-1.2987e+00,  2.7115e+00,  2.7505e-01,  1.3521e-01, -6.1180e-01,
#>          -4.1990e-01, -7.4985e-01,  4.2535e-03,  4.3589e-01, -2.0743e-01],
#>         [-1.2870e+00, -1.3138e+00, -7.4987e-01,  8.9608e-02,  1.5270e+00,
#>          -4.6191e-02, -4.6467e-01,  7.4020e-01,  5.6403e-01,  9.5931e-01],
```

```
#>          [-1.2569e+00, -3.7767e-01, -7.0048e-01, -1.1057e-01,  7.2034e-01,
#>            2.7289e-01,  3.0095e-01, -1.4627e-02,  2.8486e-01,  1.3238e+00],
#>          [ 1.0152e-01, -3.0979e-01,  9.4464e-01, -1.3014e+00,  6.3443e-01,
#>            1.4799e-01,  6.7136e-01, -8.3160e-01,  3.8755e-02,  2.4443e-01],
#>          [-1.2765e-01, -8.8092e-01, -1.0809e+00, -1.1238e+00,  1.1095e+00,
#>           -2.1077e-01, -4.2052e-01,  1.9280e+00,  8.6710e-02,  1.6435e+00],
#>          [ 3.1574e+00, -1.9309e+00,  6.6490e-01,  6.8592e-01, -7.4162e-01,
#>            9.8014e-01, -3.9286e-01, -1.4621e+00,  5.7645e-01, -1.7922e+00],
#>          [ 8.1289e-01, -5.4395e-01,  6.2101e-01, -3.8470e-01,  4.7841e-01,
#>           -5.1522e-01,  9.4043e-01, -8.0787e-01,  2.5931e-01, -5.0283e-01],
#>          [-7.9017e-01, -1.3696e+00, -8.5082e-01, -9.1603e-01,  1.6260e+00,
#>           -7.7058e-02, -1.6040e-01,  1.1269e+00,  2.0468e-01,  2.2587e+00],
#>          [ 3.1616e+00, -2.3800e+00, -2.7943e-01, -2.4139e-01, -3.6120e-01,
#>            7.3184e-01, -3.4811e-01, -4.4688e-01,  4.1694e-01,  2.4069e-01],
#>          [-1.2642e+00,  2.8791e+00, -1.1524e-01,  4.3556e-01, -1.1386e+00,
#>           -2.1253e-01,  3.7044e-02, -1.2600e-01,  3.3838e-01, -1.7522e-01],
#>          [ 3.7750e-01, -9.0595e-01, -1.8303e-01,  1.1638e+00, -2.9494e-01,
#>            7.6012e-01, -4.4813e-01, -4.9704e-01,  6.9392e-01, -5.9989e-01],
#>          [-4.5471e-01, -1.9645e+00,  1.6492e-01, -7.3672e-01,  1.4368e+00,
#>           -6.8745e-01, -1.7953e-01,  7.6996e-01,  2.2585e-01,  1.6515e+00],
#>          [ 2.1165e-01, -1.9543e+00, -4.4266e-01,  3.6177e-01, -3.1148e-01,
#>           -2.6539e-01, -7.6945e-01,  2.6859e+00, -6.0181e-01,  6.2520e-01],
#>          [-6.6666e-01, -6.3672e-01, -4.8508e-02,  1.1945e+00, -6.4643e-01,
#>            3.0000e-01,  1.0074e+00, -5.9322e-01,  1.0006e-01, -3.4331e-01],
#>          [-8.5328e-01, -1.3704e+00, -4.8718e-01, -1.4346e-01,  1.9699e+00,
#>            1.6221e-03, -8.5731e-02,  2.3024e-01,  1.1548e-01,  1.3424e+00],
#>          [-7.7359e-01, -2.8210e-01, -1.3802e+00,  1.9043e-01,  6.7667e-01,
#>            1.1207e-01, -1.2175e+00,  1.7131e+00,  1.4865e-01,  1.6175e+00],
#>          [-5.0142e-01, -1.5318e+00, -2.6184e-02,  4.2558e-01,  3.6907e-01,
#>            6.6034e-01,  2.3316e+00, -1.5406e+00,  4.8098e-01, -1.3304e-01],
#>          [-2.4639e-01, -1.0642e-01,  5.2688e-01, -7.9012e-01,  9.2872e-01,
#>           -1.1287e+00,  1.2245e+00,  1.2407e-01, -3.1249e-01, -2.1534e-03],
#>          [-1.0451e-01, -9.4606e-01, -5.9241e-01,  3.9987e-01, -1.3748e-01,
#>            1.5447e+00,  1.9996e-01, -9.4401e-01,  8.8916e-01,  1.5567e-01],
#>          [-7.7939e-01, -1.1531e+00,  1.6778e-01, -1.4628e-01,  1.3819e+00,
#>           -2.5042e-01, -1.5504e-01,  3.4741e-01, -2.4857e-01,  1.0174e+00],
#>          [ 4.3512e+00, -2.9414e+00,  8.7806e-01, -1.3870e+00, -3.3717e-01,
#>            1.0033e+00,  1.2734e+00, -7.3720e-01,  1.1137e-02, -1.3699e+00],
#>          [-2.8166e-01, -1.3024e+00, -4.5727e-01,  7.2126e-02,  1.7663e-01,
#>           -2.4383e-01, -4.9624e-01,  1.6236e+00, -6.0035e-01,  1.1375e+00],
#>          [-3.7374e-01, -2.1148e+00, -5.2038e-01, -6.6244e-01,  2.3591e+00,
#>            9.7800e-02,  2.6121e-01,  2.0765e-02,  1.6089e-01,  1.4404e+00],
#>          [ 2.7762e+00, -2.4484e+00,  3.7692e-02,  8.9276e-01, -1.1306e+00,
#>            6.9539e-01, -2.5336e-01, -7.3195e-01,  3.6679e-01, -4.0027e-01],
#>          [-1.0824e+00,  1.4544e+00, -4.0074e-01,  2.3187e-01, -4.6137e-01,
```

```
#>          1.5103e-01,  1.8921e-01, -7.8811e-02,  3.1913e-01, -2.7850e-01],
#>        [-8.8933e-01, -4.6435e-01, -8.9819e-01,  2.1559e+00, -1.3384e+00,
#>          1.0193e+00, -3.9569e-01,  1.7003e-01,  1.8287e-01, -1.3543e-01],
#>        [-9.8079e-01,  1.1783e+00, -4.3017e-01,  3.1909e-01, -3.2282e-01,
#>          5.9966e-02,  2.4061e-03,  1.6737e-01,  1.7956e-01,  1.2991e-01],
#>        [-9.0914e-01, -8.1733e-01, -7.7819e-01,  2.4002e+00, -4.6174e-01,
#>          1.4284e+00, -5.5151e-01, -1.2451e+00,  4.4246e-01, -8.2233e-02],
#>        [ 1.8606e+00, -1.8194e+00,  6.8703e-01, -1.8911e+00,  8.0873e-01,
#>          4.2704e-01,  1.7418e+00, -8.1675e-01, -1.1473e-01, -2.8603e-01],
#>        [-8.7625e-01, -2.8564e-01,  7.9909e-01, -3.0945e-01,  6.9551e-02,
#>         -5.6687e-01, -1.3608e+00,  1.7866e+00,  7.8766e-01,  3.9826e-01],
#>        [ 5.2140e-01, -1.2894e+00,  2.5902e+00,  2.8523e-01, -6.7693e-01,
#>          3.0978e-01,  4.7118e-01, -3.0533e-01,  2.6335e-01, -1.7316e+00],
#>        [-6.0106e-01, -1.6226e+00,  1.5349e-01, -7.6830e-03, -1.7812e-02,
#>         -6.0396e-01, -4.6993e-01,  2.3227e+00, -2.9389e-01,  1.1643e+00],
#>        [-1.2897e+00,  2.1840e+00, -5.1492e-01,  1.3564e-01, -6.2325e-01,
#>         -2.8047e-02,  2.5177e-02,  9.4724e-02,  4.6202e-01, -6.7154e-03],
#>        [ 2.5418e-01,  1.9973e-01,  1.0947e+00,  9.8032e-01, -1.8865e+00,
#>          3.9306e-01,  6.4038e-01, -1.3409e+00,  7.8411e-01, -1.2461e+00],
#>        [-1.4295e+00,  3.0419e+00, -2.9736e-01,  3.3749e-01, -1.3243e+00,
#>         -2.3459e-02, -8.5192e-02, -5.8925e-01,  6.2745e-01, -2.1938e-01],
#>        [-7.0331e-01,  1.6353e+00, -3.0117e-02,  1.8527e-01, -5.3464e-01,
#>         -1.5881e-01, -1.8888e-02, -1.5384e-01,  1.3406e-02, -1.7692e-01],
#>        [-7.3279e-01, -9.4782e-01, -2.9877e-02, -4.6342e-01,  2.9866e-02,
#>         -2.3611e-01, -4.4630e-01,  2.0039e+00, -3.7444e-01,  1.2795e+00],
#>        [-2.0154e+00, -3.2036e-01, -3.9857e-02, -3.6067e-01,  2.2775e+00,
#>         -7.1071e-01, -8.0430e-01,  7.2148e-01,  5.2522e-01,  1.7124e+00],
#>        [-4.7475e-01,  7.9963e-01,  1.2281e+00, -1.3052e-01, -7.3618e-02,
#>         -3.4371e-01,  3.0799e-01, -9.8563e-01,  4.7410e-01, -3.3280e-01],
#>        [-1.0504e+00,  7.3264e-02, -7.3355e-02,  1.3657e+00, -8.6593e-01,
#>          5.3903e-01,  4.3451e-01, -4.2514e-01,  7.7801e-02, -2.4562e-01],
#>        [ 7.4037e-02, -1.2902e+00, -7.2728e-01,  1.8037e+00, -6.5481e-01,
#>          1.4527e+00, -1.0006e-01, -1.3808e+00,  1.0110e+00, -4.6459e-01],
#>        [-1.5870e+00,  3.0939e-01, -1.5997e-01,  7.4274e-01, -1.4331e-01,
#>          3.3919e-01,  1.1756e-01,  4.8152e-02,  2.5385e-01,  2.5526e-01],
#>        [-7.3733e-01, -2.3935e-01,  1.6282e+00, -2.4430e-01,  6.6310e-01,
#>         -6.9060e-01,  5.1241e-01, -2.3024e-02,  4.7846e-02,  1.7471e-01],
#>        [-1.0746e+00, -2.5196e+00, -1.3470e+00,  1.5832e-02,  2.8318e+00,
#>          3.6442e-01, -5.6977e-01,  3.9916e-01,  7.3580e-01,  2.1160e+00],
#>        [-9.2246e-02, -2.0376e+00,  6.1287e-01, -7.3385e-01,  2.2582e+00,
#>         -9.1551e-01,  7.1243e-02,  4.2175e-01, -2.2545e-01,  1.1509e+00],
#>        [-3.9357e-03, -1.0385e+00,  1.2450e-01,  3.3858e-01, -3.4441e-01,
#>          5.1258e-01,  2.0654e+00, -1.4793e+00,  8.8135e-02, -2.6334e-01],
#>        [-6.1840e-02, -1.1377e+00, -2.7466e-01,  1.8441e+00, -6.5354e-01,
#>          6.8002e-01,  1.0520e-01, -4.1124e-01, -1.3710e-01,  9.1419e-02],
```

```
#>          [ 5.9769e-01, -1.1936e+00, -1.3663e+00,  3.1573e-03,  6.3732e-01,
#>            1.3869e+00, -7.9784e-02, -3.3405e-01, -1.1160e-01,  6.0435e-01],
#>          [ 2.1860e-01, -1.1580e+00, -4.7119e-01,  8.8685e-01, -3.5183e-02,
#>            7.0088e-01, -1.6251e-01, -6.6693e-01,  4.7858e-01, -1.4018e-01],
#>          [ 3.5646e-01, -7.3620e-01,  1.4141e+00, -6.4968e-01,  7.3994e-01,
#>           -1.0560e+00,  6.6393e-01, -5.0864e-01, -1.0077e-01, -1.2591e-01],
#>          [ 1.5414e+00, -2.2194e+00, -6.5543e-01,  6.8563e-01, -7.1479e-01,
#>            9.8243e-01,  2.4973e-01, -8.0620e-01,  1.0722e+00, -1.4052e-01],
#>          [-3.3331e-02, -2.7590e+00, -2.8456e-02, -4.7988e-01,  2.7396e+00,
#>            3.8984e-02,  4.1396e-01, -1.8820e-01,  2.7952e-02,  9.3577e-01],
#>          [-9.0653e-01,  2.2782e+00,  4.2394e-02,  1.7376e-01, -7.5019e-01,
#>           -3.2438e-01, -5.2303e-01, -1.4208e-01,  2.8690e-01, -1.6694e-01],
#>          [-1.9219e-01, -1.8828e+00, -9.3559e-01, -7.7169e-01,  1.7327e+00,
#>           -3.1115e-01, -2.6599e-01,  1.1501e+00, -2.4615e-01,  2.2717e+00],
#>          [ 1.9192e-01,  2.8162e-01, -4.1308e-01, -2.9724e-01,  1.1830e-01,
#>            4.1103e-01, -2.0661e-01,  2.8052e-01,  1.4448e-01, -2.1229e-02],
#>          [-1.0695e-01, -1.6508e+00, -5.7504e-01,  7.4708e-01,  1.4695e-01,
#>           -2.7459e-01, -1.1023e-01,  2.1898e+00, -7.6062e-01,  6.6331e-01],
#>          [ 1.7897e-01, -8.4741e-01,  9.4273e-01, -1.4337e+00, -8.1309e-02,
#>            2.3699e-01,  5.5836e-01, -6.4794e-01,  1.3851e+00,  1.7151e-01],
#>          [-9.5296e-01, -7.7065e-01,  7.5880e-03, -4.1062e-01,  7.7437e-01,
#>            1.5579e-01,  3.7150e-02, -4.6469e-02,  5.4424e-01,  8.8268e-01],
#>          [-8.5767e-01,  1.4701e-01,  1.1428e+00,  7.9135e-01, -1.5296e-01,
#>           -1.5779e-01, -3.0358e-01, -4.6426e-01,  6.7691e-01,  3.2591e-01],
#>          [-9.1459e-01, -8.2141e-01,  4.2666e-01, -3.5101e-01,  9.0355e-01,
#>           -5.0283e-01, -3.2031e-01,  1.8320e+00,  2.8279e-01,  6.2437e-01],
#>          [-1.3905e+00, -6.0861e-01, -6.2662e-01,  5.7454e-01,  5.7814e-01,
#>            4.3941e-01,  7.7798e-02, -2.7515e-01,  4.5189e-01,  8.7133e-01],
#>          [ 5.6257e-01, -4.4437e-01,  9.6367e-01, -2.6671e-01,  4.1857e-01,
#>           -8.0970e-01,  6.7704e-01,  1.6887e-01, -3.3091e-01, -5.0817e-01],
#>          [-7.1065e-01, -1.3373e+00,  5.2476e-01, -9.8477e-01,  2.3167e+00,
#>           -8.8322e-01, -8.2509e-02,  5.4098e-01,  8.6976e-02,  9.3541e-01],
#>          [-1.4806e+00, -4.6107e-01, -2.6849e-01,  2.6974e+00, -4.8143e-01,
#>            8.8503e-01, -9.5357e-01, -1.0841e+00,  1.0180e+00,  8.6351e-02],
#>          [ 2.4628e+00, -1.5071e+00,  7.3117e-01, -6.4603e-01, -1.4958e+00,
#>            3.0829e-01,  4.0907e-01, -2.0519e-01, -1.6792e-01, -4.6206e-01],
#>          [ 5.6877e-01, -1.4969e+00, -6.6706e-01, -6.0053e-02, -3.0994e-01,
#>           -4.9848e-01, -7.6749e-01,  2.8585e+00, -6.1360e-01,  6.5914e-01],
#>          [ 4.4205e+00, -2.8581e+00,  1.0462e+00,  1.5467e-01, -9.9871e-01,
#>            1.0504e+00,  9.4292e-02, -1.4378e+00,  5.7136e-01, -1.6437e+00],
#>          [ 1.6883e+00, -1.3482e+00,  1.8820e+00,  1.2426e+00, -1.2594e+00,
#>            2.9173e-01,  3.9851e-01, -1.0084e+00,  1.4115e-01, -1.9422e+00],
#>          [-1.4171e+00,  8.9162e-01,  2.0539e-01,  9.5908e-02, -6.4104e-01,
#>           -4.3308e-01, -9.6421e-01,  8.9537e-01,  1.1609e+00,  4.8390e-01],
#>          [-1.2176e+00,  2.3829e+00, -4.6562e-01,  8.9853e-02, -9.5672e-01,
```

```
#>           3.1173e-02, -2.4581e-02, -8.8059e-02,  5.9428e-01, -9.2682e-02],
#>         [-1.7307e+00,  5.6224e-01, -3.2265e-01, -4.9043e-01,  7.3022e-01,
#>          -5.9717e-01, -6.5566e-01,  1.8073e+00,  7.3746e-02,  9.3846e-01],
#>         [-4.2901e-01,  1.3826e-01, -3.4844e-01,  2.1697e+00, -1.0114e+00,
#>           9.8128e-01, -4.6798e-01, -1.0823e+00,  6.5561e-01, -7.2020e-01],
#>         [-8.6825e-01, -3.3111e-01,  7.9404e-01, -6.1261e-01, -3.6129e-02,
#>          -6.8388e-01, -2.0317e-02,  1.6777e+00, -3.8377e-01,  5.7952e-01],
#>         [-1.6577e+00,  1.0589e+00, -7.0461e-01,  9.3005e-02, -5.5483e-02,
#>           -2.9957e-02, -6.6914e-01,  3.7134e-01,  9.6462e-01,  8.6798e-01],
#>         [-2.5657e-01, -6.0222e-01, -3.6740e-01, -1.0563e+00,  5.0916e-02,
#>           -6.2516e-01, -1.1167e+00,  3.3912e+00,  1.6242e-01,  8.7198e-01],
#>         [-6.1038e-01, -1.6467e+00, -1.0599e+00, -1.9766e-01,  8.3050e-01,
#>           3.9744e-01, -1.9508e-01,  1.3288e+00, -5.6222e-01,  1.7754e+00],
#>         [ 2.2199e-01, -1.7508e+00,  4.8196e-01, -4.2748e-01,  3.1753e-01,
#>           3.2987e-01,  2.5157e+00, -4.7985e-01, -1.7763e-01,  4.7502e-03],
#>         [-6.0122e-01, -9.8158e-01,  4.0671e+00,  1.2384e-02, -9.1785e-03,
#>           -1.1502e+00,  8.5871e-01, -9.3362e-01,  7.1117e-01, -1.0299e+00],
#>         [-5.8807e-01, -1.8779e+00, -4.9661e-01, -4.8811e-02,  7.0428e-01,
#>           -2.8727e-01, -8.2898e-01,  2.0634e+00, -3.1366e-01,  1.5578e+00],
#>         [-7.6150e-01,  1.0849e-01, -3.3634e-01, -5.2216e-01,  5.9955e-01,
#>           5.9659e-01, -5.6445e-01, -5.5787e-01,  1.2763e+00,  5.5385e-01],
#>         [-1.0205e+00, -2.3503e+00, -7.6701e-01,  1.6570e-01,  3.0343e+00,
#>           1.5617e-01, -1.2460e-01, -1.9061e-01,  4.7652e-01,  1.4649e+00],
#>         [-1.5438e+00,  4.9600e-01, -1.8081e-01, -7.0795e-01, -1.3971e-01,
#>           -9.8486e-01, -1.1418e+00,  2.9865e+00,  1.5504e-01,  1.0458e+00],
#>         [ 1.0249e-01, -1.7000e+00, -1.0777e+00,  1.8996e+00, -2.8052e-01,
#>           1.0650e+00,  7.0742e-01, -5.8938e-01, -1.1658e-01, -1.5945e-01],
#>         [-2.6694e-01, -2.0829e+00,  9.4647e-01, -1.3480e+00,  1.2449e+00,
#>           -4.8505e-01,  2.6142e+00,  5.0344e-02, -5.4981e-01,  1.8449e-01],
#>         [-1.4871e+00,  2.9607e+00,  4.7902e-01,  5.0073e-03, -5.0540e-01,
#>           -3.3616e-01, -2.9949e-01, -2.1439e-01,  5.7632e-01, -4.6274e-01],
#>         [ 1.7111e-01, -6.7651e-01, -2.3165e-01,  2.7955e+00, -1.1800e+00,
#>           6.9570e-01, -1.4718e+00, -4.3201e-01,  8.1469e-01, -8.1572e-01],
#>         [-8.7862e-01, -5.0456e-01,  3.2428e-01, -5.3243e-01,  4.6667e-01,
#>           4.3436e-02,  2.7644e+00, -1.2242e+00,  2.5778e-01, -5.7057e-02],
#>         [-9.5491e-01,  2.5703e-01,  3.4359e-02, -4.5068e-01,  6.8173e-01,
#>           -8.8618e-02, -1.1938e-01, -7.8896e-02,  7.9619e-01,  5.8036e-01],
#>         [-9.4097e-01, -7.9686e-01, -8.5962e-01,  2.4931e+00, -1.5386e+00,
#>           1.0913e+00, -1.1633e+00,  4.2138e-01,  6.8648e-01,  4.1477e-02],
#>         [-2.1038e+00,  1.8299e+00, -3.0775e-01,  2.3694e-01, -5.8992e-01,
#>           1.1753e-01,  5.0003e-01, -2.1849e-01,  7.3212e-01, -6.3468e-02],
#>         [-6.2806e-01, -3.7262e-01, -3.6970e-01, -1.2948e+00,  2.3200e+00,
#>           -4.6523e-01,  4.4164e-01,  2.5105e-01,  9.5926e-02,  1.2765e+00],
#>         [-1.0934e+00,  4.9081e-01, -5.4858e-01,  3.7769e-01, -1.8211e-01,
#>           2.7351e-01,  5.7915e-02,  1.6241e-01,  2.5341e-01,  3.2979e-01],
```

```
#>          [-1.7189e+00,  1.2996e+00, -4.9783e-01,  1.2690e-01, -2.8852e-01,
#>            5.7689e-03, -1.1065e-01,  8.5996e-01,  3.7666e-01,  4.4925e-01],
#>          [ 1.1295e+00, -1.1470e+00,  7.2451e-01, -8.1282e-01, -6.1164e-02,
#>            5.1537e-01,  1.9665e+00, -1.3334e+00,  6.0729e-02, -5.3673e-01],
#>          [-1.2605e+00, -1.6713e+00,  9.0893e-02, -6.1368e-01,  1.4691e+00,
#>           -9.3188e-01, -3.8812e-01,  8.1698e-01,  1.7610e-01,  2.3959e+00]],
#>         grad_fn=<AddmmBackward>)
print(predicted)
#> tensor([8, 9, 0, 1, 2, 7, 4, 8, 6, 7, 1, 0, 2, 2, 9, 4, 7, 8, 4, 7, 8, 6, 1, 1,
#>         4, 7, 8, 4, 4, 7, 0, 1, 9, 2, 8, 7, 8, 2, 6, 0, 0, 6, 3, 8, 8, 9, 1, 4,
#>         0, 6, 1, 0, 0, 0, 0, 8, 1, 7, 7, 1, 4, 6, 0, 7, 0, 3, 6, 8, 7, 1, 3, 2,
#>         4, 4, 4, 2, 6, 4, 1, 7, 2, 6, 6, 0, 1, 2, 8, 4, 5, 6, 7, 8, 4, 0, 1, 2,
#>         3, 4, 8, 6])
```

2.6.1 Printing output size

This produces a 100x10 matrix because each iteration has a batch size of 100 and each prediction across the 10 classes, with the largest number indicating the likely number it is predicting.

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()

    # Forward pass only to get logits/output
    outputs = model(images)

    if (iter_test == 1) {
        print('OUTPUTS')
        print(outputs$size())
    }
    # Get predictions from the maximum value
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]
}
#> [1] "OUTPUTS"
#> torch.Size([100, 10])
print(predicted$size())
```

```
#> torch.Size([100])
```

The `predicted` and `output` tensors have the same number of 1D members. It is obvious because `predicted` is calculated from the `output` maximum values.

2.6.2 Printing one output

This would be a 1x10 matrix where the largest number is what the model thinks the image is. Here we can see that in the tensor, position 7 has the largest number, indicating the model thinks the image is 7.

```
number 0: -0.4181
number 1: -1.0784
...
number 7: 2.9352
```

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()

    # Forward pass only to get logits/output
    outputs = model(images)

    if (iter_test == 1) {
        print('OUTPUTS')
        print(outputs[1])    # show first tensor of 100
        print(outputs[99])   # show last tensor of 100
    }
    # Get predictions from the maximum value for 100 tensors
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]
}
#> [1] "OUTPUTS"
#> Warning in `[.torch.Tensor`(outputs, 1): Incorrect number of dimensions
#> supplied. The number of supplied arguments, (not counting any NULL,
#> tf$newaxis or np$newaxis) must match thenumber of dimensions in the tensor,
#> unless an all_dims() was supplied (this will produce an error in the
#> future)
#> tensor([-0.2048, -1.3593, -0.3049, -0.1472,  0.0408, -0.5324, -0.9813,  2.8635,
```

```
#>         -0.4046,  0.9022], grad_fn=<SelectBackward>)
#> Warning in `[.torch.Tensor`(outputs, 99): Incorrect number of dimensions
#> supplied. The number of supplied arguments, (not counting any NULL,
#> tf$newaxis or np$newaxis) must match thenumber of dimensions in the tensor,
#> unless an all_dims() was supplied (this will produce an error in the
#> future)
#> tensor([ 1.1295, -1.1470,  0.7245, -0.8128, -0.0612,  0.5154,  1.9665, -1.3334,
#>          0.0607, -0.5367], grad_fn=<SelectBackward>)
print(predicted)
#> tensor([8, 9, 0, 1, 2, 7, 4, 8, 6, 7, 1, 0, 2, 2, 9, 4, 7, 8, 4, 7, 8, 6, 1, 1,
#>         4, 7, 8, 4, 4, 7, 0, 1, 9, 2, 8, 7, 8, 2, 6, 0, 0, 6, 3, 8, 8, 9, 1, 4,
#>         0, 6, 1, 0, 0, 0, 0, 8, 1, 7, 7, 1, 4, 6, 0, 7, 0, 3, 6, 8, 7, 1, 3, 2,
#>         4, 4, 4, 2, 6, 4, 1, 7, 2, 6, 6, 0, 1, 2, 8, 4, 5, 6, 7, 8, 4, 0, 1, 2,
#>         3, 4, 8, 6])
```

2.6.3 Printing prediction output

Because our output is of size 100 (our batch size), our prediction size would also of the size 100.

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()

    # Forward pass only to get logits/output
    outputs = model(images)

    # Get predictions from the maximum value for a batch
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]

    if (iter_test == 1) {
        print('PREDICTION')
        print(predicted$size())
    }
}
#> [1] "PREDICTION"
#> torch.Size([100])
```

2.6.4 Print prediction value

We are printing our prediction which as verified above, should be digit 7.

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()

    # Forward pass only to get logits/output
    outputs = model(images)

    # Get predictions from the maximum value
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]

    if (iter_test == 1) {
        print('PREDICTION')
        print(predicted[1L])
    }
}
#> [1] "PREDICTION"
#> tensor(7)
```

2.6.5 Print prediction, label and label size

We are trying to show what we are predicting and the actual values. In this case, we're predicting the right value 7!

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()
    # Forward pass only to get logits/output
    outputs = model(images)
    # Get predictions from the maximum value
```

```
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]

    if (iter_test == 1) {
        print('PREDICTION')
        print(predicted[1])

        print('LABEL SIZE')
        print(labels$size())

        print('LABEL FOR IMAGE 0')
        print(labels[1]$item())  # extract the scalar part only
    }
}
#> [1] "PREDICTION"
#> tensor(7)
#> [1] "LABEL SIZE"
#> torch.Size([100])
#> [1] "LABEL FOR IMAGE 0"
#> [1] 7
```

2.6.6 Print second prediction and ground truth

It should be the digit 2.

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()
    # Forward pass only to get logits/output
    outputs = model(images)
    # Get predictions from the maximum value
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]

    if (iter_test == 1) {
        print('PREDICTION')
        print(predicted[1])

        print('LABEL SIZE')
```

```
        print(labels$size())

        print('LABEL FOR IMAGE 0')
        print(labels[1]$item())
    }
}
#> [1] "PREDICTION"
#> tensor(7)
#> [1] "LABEL SIZE"
#> torch.Size([100])
#> [1] "LABEL FOR IMAGE 0"
#> [1] 7
```

2.6.7 Print accuracy

Now we know what each object represents, we can understand how we arrived at our accuracy numbers.

One last thing to note is that `correct.item()` has this syntax is because correct is a PyTorch tensor and to get the value to compute with total which is an integer, we need to do this.

```
# Iterate through test dataset
iter_test <- 0
iter_test_dataset <- builtins$enumerate(test_loader) # convert to iterator
for (test_obj in iterate(iter_test_dataset)) {
    iter_test <- iter_test + 1
    # Load images to a Torch Variable
    images <- test_obj[[2]][[1]]
    labels <- test_obj[[2]][[2]]
    images <- images$view(-1L, 28L*28L)$requires_grad_()
    # Forward pass only to get logits/output
    outputs = model(images)
    # Get predictions from the maximum value
    .predicted = torch$max(outputs$data, 1L)
    predicted <- .predicted[1L]
    # Total number of labels
    total = total + labels$size(0L)
    # Total correct predictions
    correct = correct + sum((predicted$numpy() == labels$numpy()))
}
accuracy = 100 * correct / total
print(accuracy)
#> [1] 83.4
```

2.7 Saving PyTorch model

This is how you save your model. Feel free to just change `save_model = TRUE` to save your model

```
save_model = TRUE
if (save_model) {
    # Saves only parameters
    torch$save(model$state_dict(), 'awesome_model.pkl')
}
```

Part II

PyTorch and R data structures

Chapter 3

Working with data.frame

3.1 Load PyTorch libraries

```
library(rTorch)

torch       <- import("torch")
torchvision <- import("torchvision")
nn          <- import("torch.nn")
transforms  <- import("torchvision.transforms")
dsets       <- import("torchvision.datasets")
builtins    <- import_builtins()
np          <- import("numpy")
```

3.2 Dataset iteration batch settings

```
# folders where the images are located
train_data_path = '~/mnist_png_full/training/'
test_data_path  = '~/mnist_png_full/testing/'
```

```
# read the datasets without normalization
train_dataset = torchvision$datasets$ImageFolder(root = train_data_path,
    transform = torchvision$transforms$ToTensor()
)

print(train_dataset)
#> Dataset ImageFolder
#>     Number of datapoints: 60000
```

```
#>     Root location: /home/msfz751/mnist_png_full/training/
```

3.3 Summary statistics for tensors

3.4 using `data.frame`

```
library(tictoc)
tic()

fun_list <- list(
    size  = c("size"),
    numel = c("numel"),
    sum   = c("sum",    "item"),
    mean  = c("mean",   "item"),
    std   = c("std",    "item"),
    med   = c("median", "item"),
    max   = c("max",    "item"),
    min   = c("min",    "item")
    )

idx <- seq(0L, 599L)    # how many samples

fun_get_tensor <- function(x) py_get_item(train_dataset, x)[[1]]

stat_fun <- function(x, str_fun) {
  fun_var <- paste0("fun_get_tensor(x)", "$", str_fun, "()")
  sapply(idx, function(x)
    ifelse(is.numeric(eval(parse(text = fun_var))),  # size return chracater
           eval(parse(text = fun_var)),              # all else are numeric
           as.character(eval(parse(text = fun_var)))))
}

df <- data.frame(ridx = idx+1,      # index number for the sample
  do.call(data.frame,
          lapply(
              sapply(fun_list, function(x) paste(x, collapse = "()$")),
              function(y) stat_fun(1, y)
          )
  )
)
head(df)
#>   ridx                    size numel sum  mean   std med max min
```

```
#> 1    1 torch.Size([3, 28, 28])  2352 366 0.156 0.329   0   1   0
#> 2    2 torch.Size([3, 28, 28])  2352 284 0.121 0.297   0   1   0
#> 3    3 torch.Size([3, 28, 28])  2352 645 0.274 0.420   0   1   0
#> 4    4 torch.Size([3, 28, 28])  2352 410 0.174 0.355   0   1   0
#> 5    5 torch.Size([3, 28, 28])  2352 321 0.137 0.312   0   1   0
#> 6    6 torch.Size([3, 28, 28])  2352 654 0.278 0.421   0   1   0
toc()
#> 13.016 sec elapsed
# 59    1.663s
#   599  13.5s
#  5999  54.321 sec; 137.6s
# 59999 553.489 sec elapsed
```

Chapter 4

Working with data.table

4.1 Load PyTorch libraries

```
library(rTorch)

torch       <- import("torch")
torchvision <- import("torchvision")
nn          <- import("torch.nn")
transforms  <- import("torchvision.transforms")
dsets       <- import("torchvision.datasets")
builtins    <- import_builtins()
np          <- import("numpy")

## Dataset iteration batch settings
# folders where the images are located
train_data_path = '~/mnist_png_full/training/'
test_data_path  = '~/mnist_png_full/testing/'

# read the datasets without normalization
train_dataset = torchvision$datasets$ImageFolder(root = train_data_path,
    transform = torchvision$transforms$ToTensor()
)

print(train_dataset)
#> Dataset ImageFolder
#>     Number of datapoints: 60000
#>     Root location: /home/msfz751/mnist_png_full/training/
```

4.1.1 Using 'data.table

```
library(data.table)
library(tictoc)
tic()

fun_list <- list(
    numel = c("numel"),
    sum   = c("sum",    "item"),
    mean  = c("mean",   "item"),
    std   = c("std",    "item"),
    med   = c("median", "item"),
    max   = c("max",    "item"),
    min   = c("min",    "item")
    )

idx <- seq(0L, 5999L)

fun_get_tensor <- function(x) py_get_item(train_dataset, x)[[1]]

stat_fun <- function(x, str_fun) {
  fun_var <- paste0("fun_get_tensor(x)", "$", str_fun, "()")
  sapply(idx, function(x)
    ifelse(is.numeric(eval(parse(text = fun_var))),  # size return chracater
           eval(parse(text = fun_var)),              # all else are numeric
           as.character(eval(parse(text = fun_var)))))
}

dt <- data.table(ridx = idx+1,
  do.call(data.table,
          lapply(
            sapply(fun_list, function(x) paste(x, collapse = "()$")),
            function(y) stat_fun(1, y)
          )
  )
)

head(dt)
#>    ridx numel sum  mean   std med max min
#> 1:    1  2352 366 0.156 0.329   0   1   0
#> 2:    2  2352 284 0.121 0.297   0   1   0
#> 3:    3  2352 645 0.274 0.420   0   1   0
#> 4:    4  2352 410 0.174 0.355   0   1   0
#> 5:    5  2352 321 0.137 0.312   0   1   0
#> 6:    6  2352 654 0.278 0.421   0   1   0
```

```
toc()
#> 105.771 sec elapsed

#   60   1.266 sec elapsed
#  600   11.798 sec elapsed; 12.9s
# 6000  119.256 sec elapsed; 132.9s
# 1117.619 sec elapsed
```

Part III

PyTorch with Rmarkdown

Chapter 5

Simple Regression with PyTorch

This examples combine Python and R code together.

Source: https://www.guru99.com/pytorch-tutorial.html

5.1 Creating the network model

Our network model is a simple Linear layer with an input and an output shape of 1.

```
library(rTorch)
```

```
from __future__ import print_function

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable

torch.manual_seed(123)
#> <torch._C.Generator object at 0x7fae1a382a90>
class Net(nn.Module):
   def __init__(self):
       super(Net, self).__init__()
       self.layer = torch.nn.Linear(1, 1)

   def forward(self, x):
```

```
        x = self.layer(x)
        return x

net = Net()
print(net)
#> Net(
#>   (layer): Linear(in_features=1, out_features=1, bias=True)
#> )
```

And the network output should be like this

```
Net(
  (hidden): Linear(in_features=1, out_features=1, bias=True)
)
```

5.1.1 Code in R

This would be the equivalent code in R:

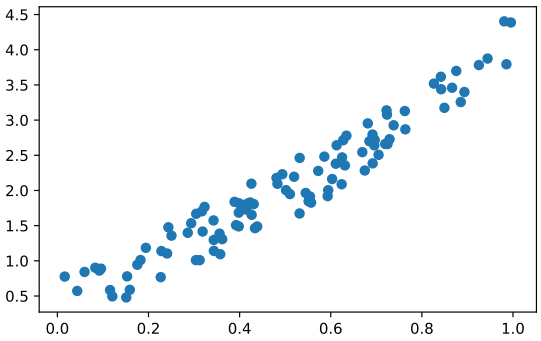
```
library(reticulate)
#>
#> Attaching package: 'reticulate'
#> The following objects are masked from 'package:rTorch':
#>
#>     conda_install, conda_python

torch <- import("torch")
nn    <- import("torch.nn")
Variable <- import("torch.autograd")$Variable

torch$manual_seed(123)
#> <torch._C.Generator>

main = py_run_string(
"
import torch.nn as nn

class Net(nn.Module):
   def __init__(self):
       super(Net, self).__init__()
       self.layer = torch.nn.Linear(1, 1)

   def forward(self, x):
       x = self.layer(x)
       return x
```

```
")


# build a Linear Rgression model
net <- main$Net()
print(net)
#> Net(
#>   (layer): Linear(in_features=1, out_features=1, bias=True)
#> )
```

5.2 Datasets

Before you start the training process, you need to know our data. You make a random function to test our model. $Y = x3sin(x) + 3x + 0.8rand(100)$

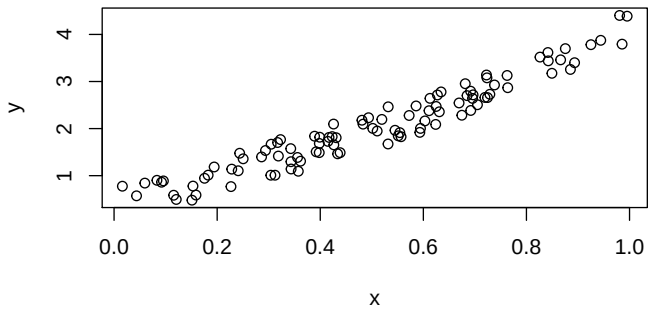
```
# Visualize our data
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(123)

x = np.random.rand(100)
y = np.sin(x) * np.power(x,3) + 3*x + np.random.rand(100)*0.8

plt.scatter(x, y)
plt.show()
```



This is the code in R:

```
np    <- import("numpy")

np$random$seed(123L)
```

```
x = np$random$rand(100L)
y = np$sin(x) * np$power(x, 3L) + 3*x + np$random$rand(100L)*0.8

plot(x, y)
```



Before you start the training process, you need to convert the numpy array to Variables that supported by Torch and autograd.

```
# convert numpy array to tensor in shape of input size
x = torch.from_numpy(x.reshape(-1,1)).float()
y = torch.from_numpy(y.reshape(-1,1)).float()
print(x, y)
#> tensor([[0.6965],
#>         [0.2861],
#>         [0.2269],
#>         [0.5513],
#>         [0.7195],
#>         [0.4231],
#>         [0.9808],
#>         [0.6848],
#>         [0.4809],
#>         [0.3921],
#>         [0.3432],
#>         [0.7290],
#>         [0.4386],
#>         [0.0597],
#>         [0.3980],
#>         [0.7380],
#>         [0.1825],
#>         [0.1755],
#>         [0.5316],
#>         [0.5318],
#>         [0.6344],
#>         [0.8494],
#>         [0.7245],
```

```
#>         [0.6110],
#>         [0.7224],
#>         [0.3230],
#>         [0.3618],
#>         [0.2283],
#>         [0.2937],
#>         [0.6310],
#>         [0.0921],
#>         [0.4337],
#>         [0.4309],
#>         [0.4937],
#>         [0.4258],
#>         [0.3123],
#>         [0.4264],
#>         [0.8934],
#>         [0.9442],
#>         [0.5018],
#>         [0.6240],
#>         [0.1156],
#>         [0.3173],
#>         [0.4148],
#>         [0.8663],
#>         [0.2505],
#>         [0.4830],
#>         [0.9856],
#>         [0.5195],
#>         [0.6129],
#>         [0.1206],
#>         [0.8263],
#>         [0.6031],
#>         [0.5451],
#>         [0.3428],
#>         [0.3041],
#>         [0.4170],
#>         [0.6813],
#>         [0.8755],
#>         [0.5104],
#>         [0.6693],
#>         [0.5859],
#>         [0.6249],
#>         [0.6747],
#>         [0.8423],
#>         [0.0832],
#>         [0.7637],
#>         [0.2437],
```

```
#>          [0.1942],
#>          [0.5725],
#>          [0.0957],
#>          [0.8853],
#>          [0.6272],
#>          [0.7234],
#>          [0.0161],
#>          [0.5944],
#>          [0.5568],
#>          [0.1590],
#>          [0.1531],
#>          [0.6955],
#>          [0.3188],
#>          [0.6920],
#>          [0.5544],
#>          [0.3890],
#>          [0.9251],
#>          [0.8417],
#>          [0.3574],
#>          [0.0436],
#>          [0.3048],
#>          [0.3982],
#>          [0.7050],
#>          [0.9954],
#>          [0.3559],
#>          [0.7625],
#>          [0.5932],
#>          [0.6917],
#>          [0.1511],
#>          [0.3989],
#>          [0.2409],
#>          [0.3435]]) tensor([[2.7166],
#>          [1.3983],
#>          [0.7679],
#>          [1.8464],
#>          [2.6614],
#>          [1.8297],
#>          [4.4034],
#>          [2.7003],
#>          [2.1778],
#>          [1.5073],
#>          [1.2966],
#>          [2.7287],
#>          [1.4884],
#>          [0.8423],
```

```
#>         [1.4895],
#>         [2.9263],
#>         [1.0114],
#>         [0.9445],
#>         [1.6729],
#>         [2.4624],
#>         [2.7788],
#>         [3.1746],
#>         [2.6593],
#>         [2.3800],
#>         [3.1382],
#>         [1.7665],
#>         [1.3082],
#>         [1.1390],
#>         [1.5341],
#>         [2.3566],
#>         [0.8612],
#>         [1.4642],
#>         [1.8066],
#>         [2.2308],
#>         [2.0962],
#>         [1.0096],
#>         [1.6538],
#>         [3.3994],
#>         [3.8747],
#>         [2.0045],
#>         [2.0884],
#>         [0.5845],
#>         [1.7039],
#>         [1.7285],
#>         [3.4602],
#>         [1.3581],
#>         [2.0949],
#>         [3.7935],
#>         [2.1950],
#>         [2.6425],
#>         [0.4948],
#>         [3.5188],
#>         [2.1628],
#>         [1.9643],
#>         [1.5740],
#>         [1.0099],
#>         [1.8123],
#>         [2.9534],
#>         [3.6986],
```

```
#>         [1.9485],
#>         [2.5445],
#>         [2.4811],
#>         [2.4700],
#>         [2.2838],
#>         [3.4392],
#>         [0.9015],
#>         [2.8687],
#>         [1.4766],
#>         [1.1847],
#>         [2.2782],
#>         [0.8885],
#>         [3.2565],
#>         [2.7141],
#>         [3.0781],
#>         [0.7763],
#>         [2.0038],
#>         [1.8270],
#>         [0.5882],
#>         [0.7793],
#>         [2.6416],
#>         [1.4162],
#>         [2.3851],
#>         [1.9140],
#>         [1.8385],
#>         [3.7822],
#>         [3.6160],
#>         [1.0941],
#>         [0.5721],
#>         [1.6683],
#>         [1.6848],
#>         [2.5068],
#>         [4.3876],
#>         [1.3866],
#>         [3.1286],
#>         [1.9197],
#>         [2.7949],
#>         [0.4797],
#>         [1.8171],
#>         [1.1042],
#>         [1.1414]])
```

5.2.1 Code in R

Notice that before converting to a Torch tensor, we need first to convert the R numeric vector to a `numpy` array:

```
# convert numpy array to tensor in shape of input size
x <- r_to_py(x)
y <- r_to_py(y)
x = torch$from_numpy(x$reshape(-1L, 1L)) #$float()
y = torch$from_numpy(y$reshape(-1L, 1L)) #$float()
print(x, y)
#> tensor([[0.6965],
#>         [0.2861],
#>         [0.2269],
#>         [0.5513],
#>         [0.7195],
#>         [0.4231],
#>         [0.9808],
#>         [0.6848],
#>         [0.4809],
#>         [0.3921],
#>         [0.3432],
#>         [0.7290],
#>         [0.4386],
#>         [0.0597],
#>         [0.3980],
#>         [0.7380],
#>         [0.1825],
#>         [0.1755],
#>         [0.5316],
#>         [0.5318],
#>         [0.6344],
#>         [0.8494],
#>         [0.7245],
#>         [0.6110],
#>         [0.7224],
#>         [0.3230],
#>         [0.3618],
#>         [0.2283],
#>         [0.2937],
#>         [0.6310],
#>         [0.0921],
#>         [0.4337],
#>         [0.4309],
#>         [0.4937],
#>         [0.4258],
```

```
#>          [0.3123],
#>          [0.4264],
#>          [0.8934],
#>          [0.9442],
#>          [0.5018],
#>          [0.6240],
#>          [0.1156],
#>          [0.3173],
#>          [0.4148],
#>          [0.8663],
#>          [0.2505],
#>          [0.4830],
#>          [0.9856],
#>          [0.5195],
#>          [0.6129],
#>          [0.1206],
#>          [0.8263],
#>          [0.6031],
#>          [0.5451],
#>          [0.3428],
#>          [0.3041],
#>          [0.4170],
#>          [0.6813],
#>          [0.8755],
#>          [0.5104],
#>          [0.6693],
#>          [0.5859],
#>          [0.6249],
#>          [0.6747],
#>          [0.8423],
#>          [0.0832],
#>          [0.7637],
#>          [0.2437],
#>          [0.1942],
#>          [0.5725],
#>          [0.0957],
#>          [0.8853],
#>          [0.6272],
#>          [0.7234],
#>          [0.0161],
#>          [0.5944],
#>          [0.5568],
#>          [0.1590],
#>          [0.1531],
#>          [0.6955],
```

```
#>         [0.3188],
#>         [0.6920],
#>         [0.5544],
#>         [0.3890],
#>         [0.9251],
#>         [0.8417],
#>         [0.3574],
#>         [0.0436],
#>         [0.3048],
#>         [0.3982],
#>         [0.7050],
#>         [0.9954],
#>         [0.3559],
#>         [0.7625],
#>         [0.5932],
#>         [0.6917],
#>         [0.1511],
#>         [0.3989],
#>         [0.2409],
#>         [0.3435]], dtype=torch.float64)
```

5.3 Optimizer and Loss

Next, you should define the Optimizer and the Loss Function for our training process.

```
# Define Optimizer and Loss Function
optimizer = torch.optim.SGD(net.parameters(), lr=0.2)
loss_func = torch.nn.MSELoss()
print(optimizer)
#> SGD (
#> Parameter Group 0
#>     dampening: 0
#>     lr: 0.2
#>     momentum: 0
#>     nesterov: False
#>     weight_decay: 0
#> )
print(loss_func)
#> MSELoss()
```

5.3.1 Equivalent code in R

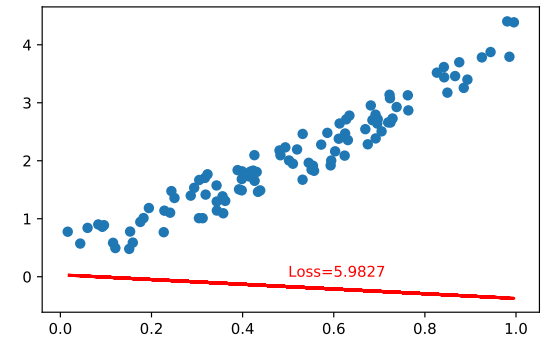
```
# Define Optimizer and Loss Function
optimizer <- torch$optim$SGD(net$parameters(), lr=0.2)
loss_func <- torch$nn$MSELoss()
print(optimizer)
#> SGD (
#> Parameter Group 0
#>     dampening: 0
#>     lr: 0.2
#>     momentum: 0
#>     nesterov: False
#>     weight_decay: 0
#> )
print(loss_func)
#> MSELoss()
```

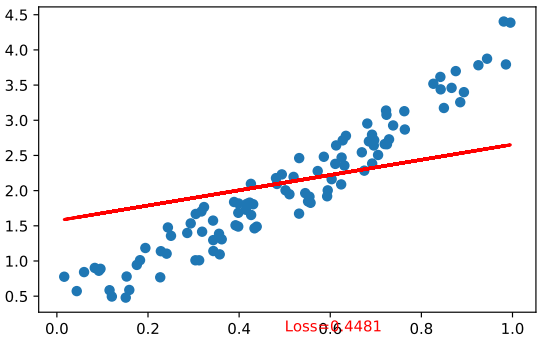
5.4 Training

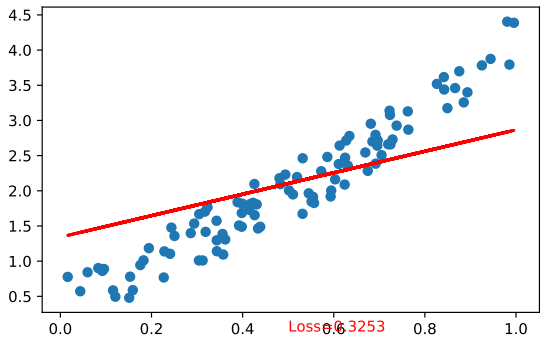
5.4.1 Code in Python

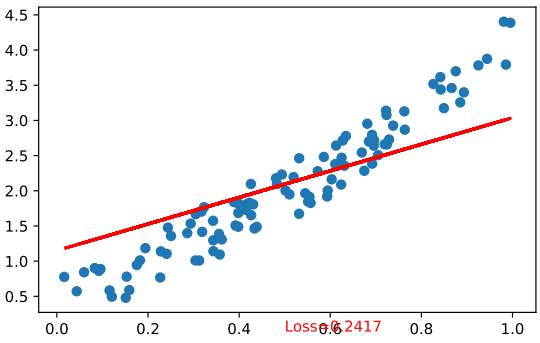
Now let's start our training process. With an epoch of 250, you will iterate our data to find the best value for our hyperparameters.

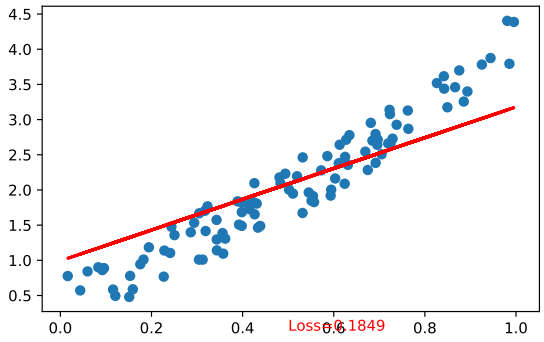
```
inputs = Variable(x)
outputs = Variable(y)
for i in range(250):
   prediction = net(inputs)
   loss = loss_func(prediction, outputs)
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()

   if i % 10 == 0:
       # plot and show learning process
       plt.cla()
       plt.scatter(x.data.numpy(), y.data.numpy())
       plt.plot(x.data.numpy(), prediction.data.numpy(), 'r-', lw=2)
       plt.text(0.5, 0, 'Loss=%.4f' % loss.data.numpy(), fontdict={'size': 10, 'color'
       plt.pause(0.1)
```

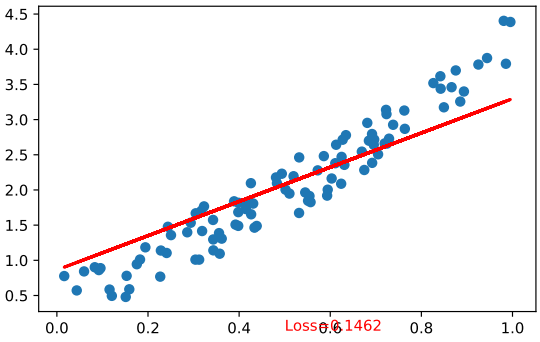

Loss=5.9827

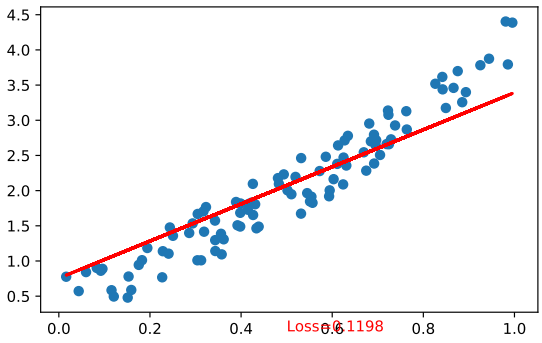

Loss=0.4481

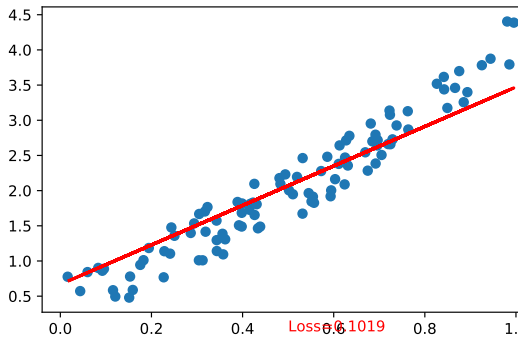

Loss=0.3253

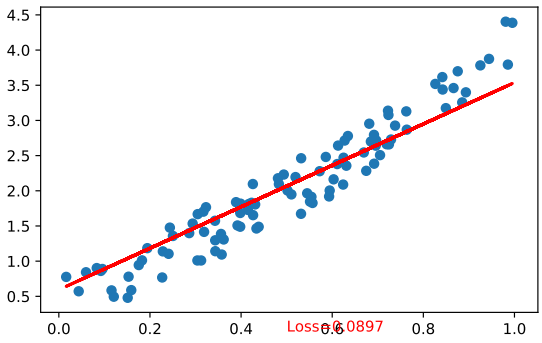

Loss=0.2417

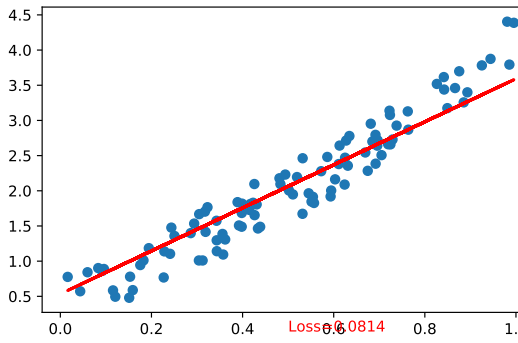

Loss=0.1849

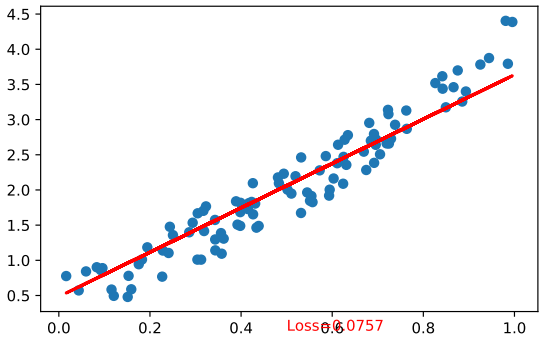

Loss=0.1462

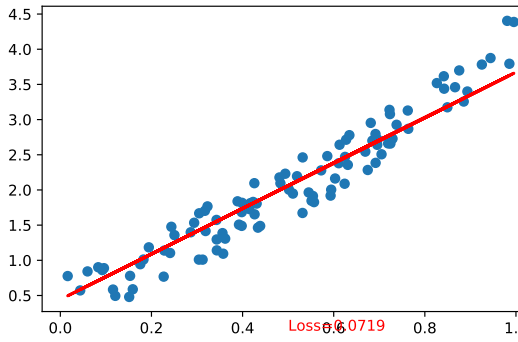

Loss: 0.1198

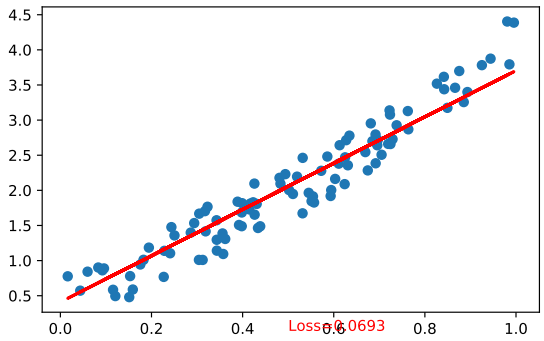

Loss: 0.1019

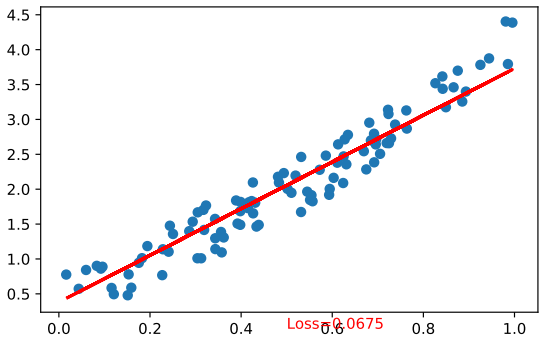

Loss: 0.0897

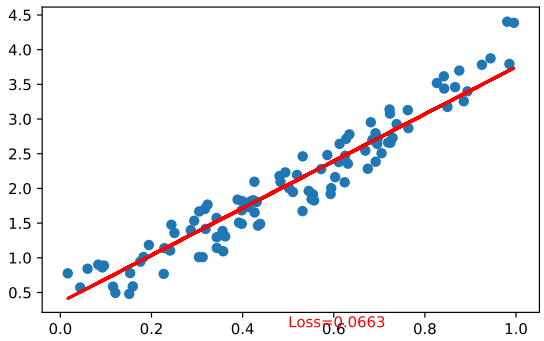

Loss: 0.0814

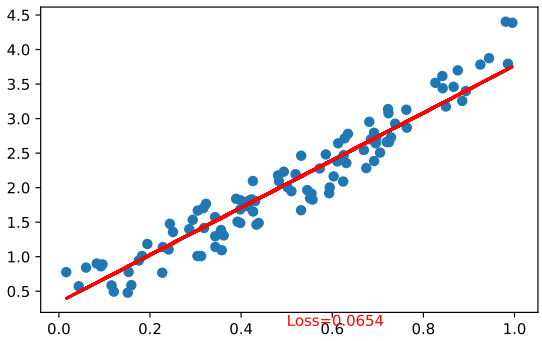

Loss: 0.0757

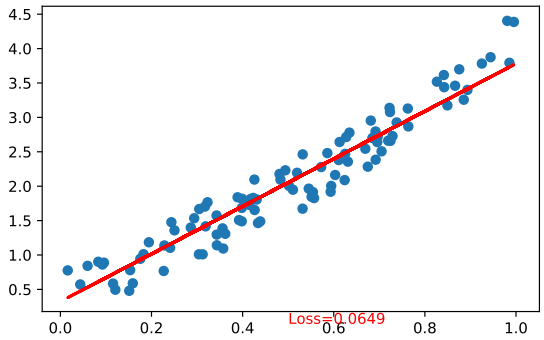

Loss: 0.0719

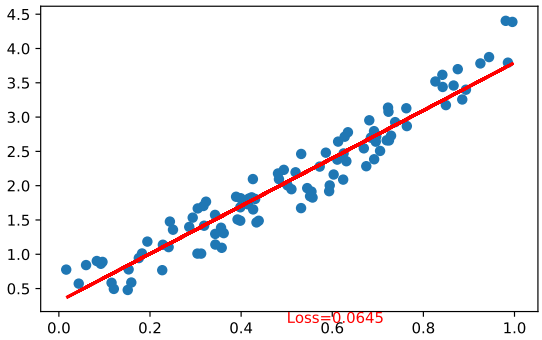

Loss=0.0693

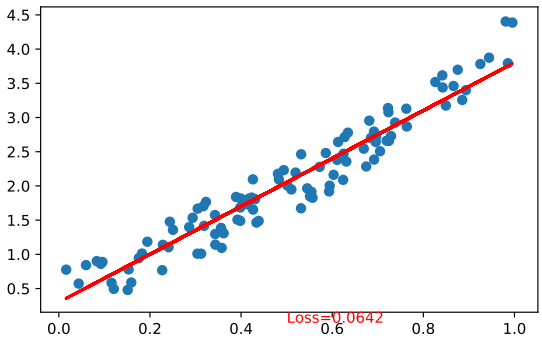

Loss=0.0675

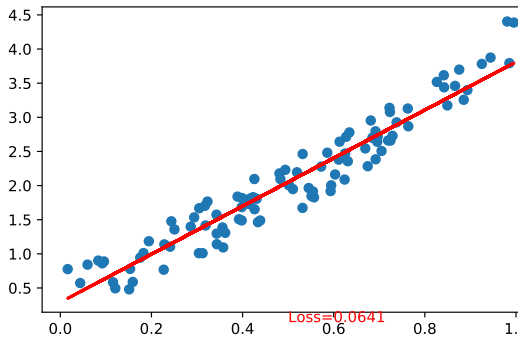

Loss=0.0663

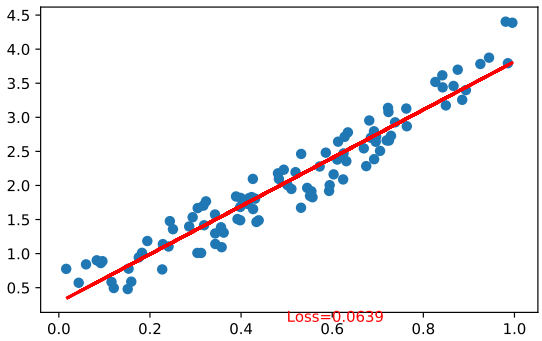

Loss=0.0654

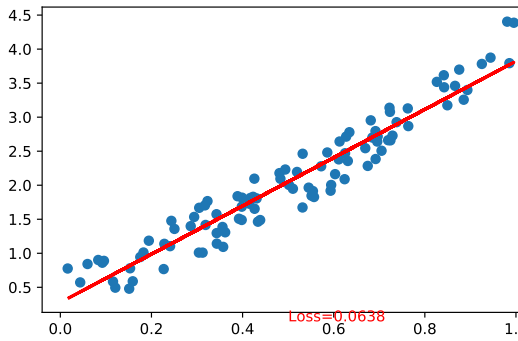

Loss=0.0649

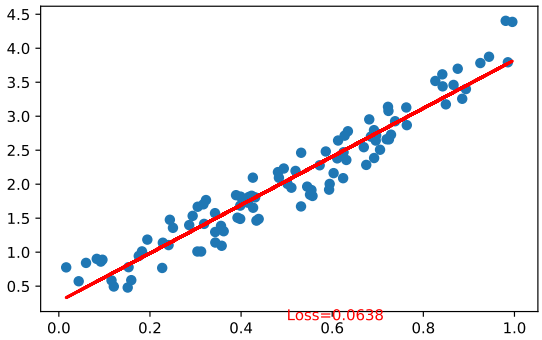

Loss=0.0645

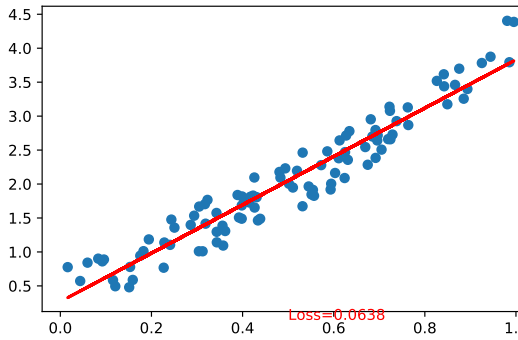

Loss=0.0642

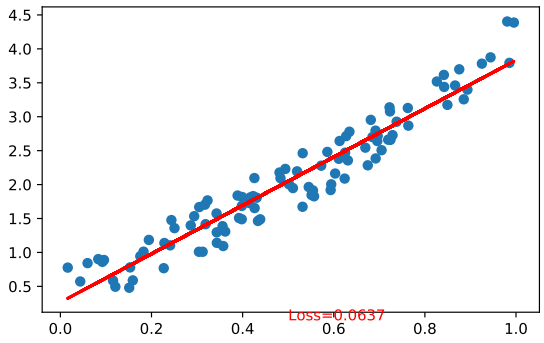

Loss=0.0641


Loss=0.0639


Loss=0.0638


Loss=0.0638


Loss=0.0638

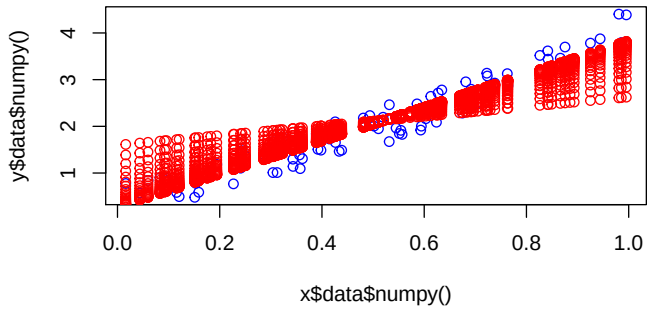



```
plt.show()
```

5.4.2 Code in R

```
x = x$type(torch$FloatTensor)    # make it a a FloatTensor
y = y$type(torch$FloatTensor)

inputs = Variable(x)
outputs = Variable(y)
plot(x$data$numpy(), y$data$numpy(), col = "blue")
for (i in 1:250) {
   prediction = net(inputs)
   loss = loss_func(prediction, outputs)
   optimizer$zero_grad()
   loss$backward()
   optimizer$step()

   if (i %% 10 == 0) {
```

```
        # plot and show learning process
       # points(x$data$numpy(), y$data$numpy())
       points(x$data$numpy(), prediction$data$numpy(), col="red")
        # cat(i, loss$data$numpy(), "\n")
   }
}
```



5.5 Result

As you can see below, you successfully performed regression with a neural network. Actually, on every iteration, the red line in the plot will update and change its position to fit the data. But in this picture, you only show you the final result

Chapter 6

Autograd

Source: https://github.com/jcjohnson/pytorch-examples#pytorch-autograd

```
library(rTorch)
```

6.1 Python code

```
# Do not print from a function. Similar functionality to R invisible()
# https://stackoverflow.com/a/45669280/5270873
import os, sys

class HiddenPrints:
    def __enter__(self):
        self._original_stdout = sys.stdout
        sys.stdout = open(os.devnull, 'w')

    def __exit__(self, exc_type, exc_val, exc_tb):
        sys.stdout.close()
        sys.stdout = self._original_stdout
```

```
# Code in file autograd/two_layer_net_autograd.py
import torch
device = torch.device('cpu')
# device = torch.device('cuda') # Uncomment this to run on GPU

torch.manual_seed(0)

# N is batch size; D_in is input dimension;
```

```
# H is hidden dimension; D_out is output dimension.
#> <torch._C.Generator object at 0x7feb342abb90>
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random Tensors to hold input and outputs
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)

# Create random Tensors for weights; setting requires_grad=True means that we
# want to compute gradients for these Tensors during the backward pass.
w1 = torch.randn(D_in, H, device=device, requires_grad=True)
w2 = torch.randn(H, D_out, device=device, requires_grad=True)

learning_rate = 1e-6

for t in range(5):
  # Forward pass: compute predicted y using operations on Tensors. Since w1 and
  # w2 have requires_grad=True, operations involving these Tensors will cause
  # PyTorch to build a computational graph, allowing automatic computation of
  # gradients. Since we are no longer implementing the backward pass by hand we
  # don't need to keep references to intermediate values.
  y_pred = x.mm(w1).clamp(min=0).mm(w2)

  # Compute and print loss. Loss is a Tensor of shape (), and loss.item()
  # is a Python number giving its value.
  loss = (y_pred - y).pow(2).sum()
  print(t, loss.item())

  # Use autograd to compute the backward pass. This call will compute the
  # gradient of loss with respect to all Tensors with requires_grad=True.
  # After this call w1.grad and w2.grad will be Tensors holding the gradient
  # of the loss with respect to w1 and w2 respectively.
  loss.backward()

  # Update weights using gradient descent. For this step we just want to mutate
  # the values of w1 and w2 in-place; we don't want to build up a computational
  # graph for the update steps, so we use the torch.no_grad() context manager
  # to prevent PyTorch from building a computational graph for the updates
  with torch.no_grad():
    w1 -= learning_rate * w1.grad
    w2 -= learning_rate * w2.grad

    # Manually zero the gradients after running the backward pass
    with HiddenPrints():   # this would be the equivalent of invisible() in R
      w1.grad.zero_()
```

```
      w2.grad.zero_()
#> 0 29428666.0
#> 1 22739450.0
#> 2 20605262.0
#> 3 19520376.0
#> 4 17810228.0
```

6.2 R code

```
# library(reticulate) # originally qwe used reticulate
library(rTorch)

torch  = import("torch")
device = torch$device('cpu')
# device = torch.device('cuda') # Uncomment this to run on GPU

torch$manual_seed(0)
#> <torch._C.Generator>

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N <- 64L; D_in <- 1000L; H <- 100L; D_out <- 10L

# Create random Tensors to hold inputs and outputs
x = torch$randn(N, D_in, device=device)
y = torch$randn(N, D_out, device=device)

# Create random Tensors for weights; setting requires_grad=True means that we
# want to compute gradients for these Tensors during the backward pass.
w1 = torch$randn(D_in, H, device=device, requires_grad=TRUE)
w2 = torch$randn(H, D_out, device=device, requires_grad=TRUE)

learning_rate = torch$scalar_tensor(1e-6)

for (t in 1:5) {
  # Forward pass: compute predicted y using operations on Tensors. Since w1 and
  # w2 have requires_grad=True, operations involving these Tensors will cause
  # PyTorch to build a computational graph, allowing automatic computation of
  # gradients. Since we are no longer implementing the backward pass by hand we
  # don't need to keep references to intermediate values.
  y_pred = x$mm(w1)$clamp(min=0)$mm(w2)

  # Compute and print loss. Loss is a Tensor of shape (), and loss.item()
```

```
  # is a Python number giving its value.
  loss = (torch$sub(y_pred, y))$pow(2)$sum()
  cat(t, "\t", loss$item(), "\n")

  # Use autograd to compute the backward pass. This call will compute the
  # gradient of loss with respect to all Tensors with requires_grad=True.
  # After this call w1.grad and w2.grad will be Tensors holding the gradient
  # of the loss with respect to w1 and w2 respectively.
  loss$backward()

  # Update weights using gradient descent. For this step we just want to mutate
  # the values of w1 and w2 in-place; we don't want to build up a computational
  # graph for the update steps, so we use the torch.no_grad() context manager
  # to prevent PyTorch from building a computational graph for the updates
  with(torch$no_grad(), {
    w1$data = torch$sub(w1$data, torch$mul(w1$grad, learning_rate))
    w2$data = torch$sub(w2$data, torch$mul(w2$grad, learning_rate))

    # Manually zero the gradients after running the backward pass
    w1$grad$zero_()
    w2$grad$zero_()
  })
}
#> 1     29428666
#> 2     22739450
#> 3     20605262
#> 4     19520376
#> 5     17810228
```

6.3 Observations

If the seeds worked the same in Python and R, we should see similar results in the output.