2-linear-regression-pytorch

December 25, 2021

1 Implementation

[15]:
```
# Import Numpy & PyTorch
import numpy as np
import torch
```

1.1 Linear Regression Model using PyTorch built-ins

Let's re-implement the same model using some built-in functions and classes from PyTorch.

And now using two different targets: Apples and Oranges

[16]:
```
# Imports
import torch.nn as nn
```

[17]:
```
# Input (temp, rainfall, humidity)
inputs = np.array([[73, 67, 43], [91, 88, 64], [87, 134, 58], [102, 43, 37],␣
 ↪[69, 96, 70], [73, 67, 43], [91, 88, 64], [87, 134, 58], [102, 43, 37], [69,␣
 ↪96, 70], [73, 67, 43], [91, 88, 64], [87, 134, 58], [102, 43, 37], [69, 96,␣
 ↪70]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[56, 70], [81, 101], [119, 133], [22, 37], [103, 119],
                    [56, 70], [81, 101], [119, 133], [22, 37], [103, 119],
                    [56, 70], [81, 101], [119, 133], [22, 37], [103, 119]],␣
 ↪dtype='float32')
```

[18]:
```
inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)
```

1.1.1 Dataset and DataLoader

We'll create a `TensorDataset`, which allows access to rows from `inputs` and `targets` as tuples. We'll also create a DataLoader, to split the data into batches while training. It also provides other utilities like shuffling and sampling.

[19]:
```
# Import tensor dataset & data loader
from torch.utils.data import TensorDataset, DataLoader
```

[20]:
```
# Define dataset
```

[21]:
```
# Define data loader
```

1.1.2 nn.Linear

Instead of initializing the weights & biases manually, we can define the model using `nn.Linear`.

[22]:
```
# Define model
```

1.1.3 Optimizer

Instead of manually manipulating the weights & biases using gradients, we can use the optimizer `optim.SGD`.

[ ]:
```
#Obtain Parameters
```

[23]:
```
# Define optimizer
```

1.1.4 Loss Function

Instead of defining a loss function manually, we can use the built-in loss function `mse_loss`.

[24]:
```
# Import nn.functional
import torch.nn.functional as F
```

[25]:
```
# Define loss function
loss_fn = F.mse_loss
```

[26]:
```
#loss = loss_fn(? , ?)
#print(loss)
```

1.1.5 Train the model

We are ready to train the model now. We can define a utility function `fit` which trains the model for a given number of epochs.

[27]:
```
# Define a utility function to train the model
#def fit(num_epochs, model, loss_fn, opt):
 #   for epoch in range(num_epochs):
  #      for xb,yb in train_dl:
             # Generate predictions
           # pred = model(?)
             #loss = loss_fn(?,?)
             # Perform gradient descent
             #loss.backward()
             #opt.step()
             #opt.zero_grad()
    #print('Training loss: ', loss_fn(model(inputs), targets))
```

[28]:
```
# Train the model for 100 epochs
#fit(? , ? , ?, ?)
```

[29]:
```
# Generate predictions
#preds = model(?)
#preds
```

[30]:
```
# Compare with targets
#targets
```

Now we can define the model, optimizer and loss function exactly as before.

[31]:
```
#fit(?,?,?,?)
```

#Exercise 1: Try Linear Regression just using numpy (Without Tensorflow/Pytorch or other torch library). You can optionally use sklearn (if you want)
Exercise 2: Try Linear regression on same prediction data using Tensorflow