Gradient Descent and Linear Regression with PyTorch

Edoch

04/08/2022

Gradient Descent and Linear Regression with PyTorch

Source: jovian.ai/aakashns/02-linear-regression

In this first section a linear regression model is implemented using PyTorch

```
import numpy as np
import torch
```

In the example the objective is to predict the yields for oranges and apples given the values of temperature, rainfall and humidity. Therefore the (linear) models are going to be

$$yield_{apple} = w_{11} * \text{temp} + w_{12} * \text{rainfall} + w_{13} * \text{humidity} + b_1$$

$$yield_{apple} = w_{21} * \text{temp} + w_{22} * \text{rainfall} + w_{23} * \text{humidity} + b_2$$

Input data

The input data are the following

```
# Input (temp, rainfall, humidity)
inputs = np.array(
    [[73, 67, 43],
    [91, 88, 64],
    [87, 134, 58],
    [102, 43, 37],
```

```
    [69, 96, 70]],
    dtype='float32'
)

inputs
```

```
array([[ 73.,  67.,  43.],
       [ 91.,  88.,  64.],
       [ 87., 134.,  58.],
       [102.,  43.,  37.],
       [ 69.,  96.,  70.]], dtype=float32)
```

The targets (labels) are the following

```
# Targets (apples, oranges)
targets = np.array(
    [[56, 70],
    [81, 101],
    [119, 133],
    [22, 37],
    [103, 119]],
    dtype='float32'
)

targets
```

```
array([[ 56.,  70.],
       [ 81., 101.],
       [119., 133.],
       [ 22.,  37.],
       [103., 119.]], dtype=float32)
```

Transform the input and targets in tensors

```
inputs = torch.from_numpy(inputs)
inputs
```

```
tensor([[ 73.,  67.,  43.],
        [ 91.,  88.,  64.],
```

```
        [ 87., 134.,  58.],
        [102.,  43.,  37.],
        [ 69.,  96.,  70.]])
```

```
targets = torch.from_numpy(targets)
targets
```

```
tensor([[ 56.,  70.],
        [ 81., 101.],
        [119., 133.],
        [ 22.,  37.],
        [103., 119.]])
```

Weights and biases initialization

Weights and biases are initialized with random values following a normal distribution with mean 0 and standard deviation 1. - Weights are collected in a tensor having the where the first dimension (rows) corresponds to the number of models (2, i.e., the number of outputs) and the second dimension (columns) corresponds to the number of features (3, i.e., the number of inputs) - Biases are collected in a tensor having the first dimension equal to the number of models (2)

```
w = torch.randn((2,3), requires_grad=True)
w
```

```
tensor([[ 0.7171, -0.4566,  0.3752],
        [-2.4049, -0.1775,  0.8105]], requires_grad=True)
```

```
b = torch.randn((2), requires_grad=True)
b
```

```
tensor([ 0.4293, -0.6531], requires_grad=True)
```

The model

The model is

$$X \cdot W^T + B$$

This model can be written in python as a function of x

```
def regression_model(x):
    return x @ w.T + b
```

where - `@` is the matrix multiplication operator - `.T` (or `.t()`) outputs the transpose tensor

Moreover, notice that b is a vector, while the result of x @ w.T is a matrix: the summation is computed only after b is broadcasted on the rows to match the matrix shape

The predictions

The predictions of the model are computed calling the function

```
preds = regression_model(inputs)
preds
```

```
tensor([[  38.3222, -153.2487],
        [  49.5216, -183.2430],
        [  23.3985, -186.6518],
        [  67.8257, -223.5927],
        [  32.3434, -126.8929]], grad_fn=<AddBackward0>)
```

If the predictions are compared with the targets it can be seen that they are quite different. That's because the model has not been trained yet, and weights and biases still have random values

Definition of a Loss function

Improving the model, it must be evaluated how well the model is performing. The model predictions are compared with the actual targets using the following method:

- Calculate the difference between the two matrices (`preds` and `targets`).
- Square all elements of the difference matrix to remove negative values.
- Calculate the average of the elements in the resulting matrix.

The result is a single number, known as the **mean squared error** (MSE).

$$\frac{\sum (\text{pred} - \text{target})^2}{N}$$