03_Linear_Regression_Concise

December 23, 2023

[31]:
```python
import torch
```

1 Concise Implementation of Linear Regression

- This notebook shows how linear regression can be implemented using higher-level functionalities provided by PyTorch.

2 Creating the Dataset

[32]:
```python
# Generating the Dataset
def synthetic_data(w, b, num_examples):
    """Generate y = Xw + b + noise."""
    X = torch.normal(0, 1, (num_examples, len(w)))
    y = torch.mm(X, w) + b
    y += torch.normal(0, 0.01, y.size())
    return X, y

true_w = torch.tensor([[2], [-3.4]])
true_b = torch.tensor(4.2)
features, labels = synthetic_data(true_w, true_b, 1000)
```

3 Generating batches

- We will generate batches using high-level functionalities provided by PyTorch classes.
- The abstract class `Dataset` requires implementing a function called `__len__` (called by Python's `len` function) and a function called `__getitem__`, which enables indexing, iterating, and slicing.
- The class `TensorDataset` implements the `Dataset` interface. The constructor of this class accepts an arbitrary number of tensors whose first dimension have the same length. When indexed, a `TensorDataset` returns a tuple with the corresponding elements from each of these tensors.
- The class `DataLoader` enables iterating through minibatches of a `Dataset`.

[33]:
```
dataset = torch.utils.data.TensorDataset(features, labels) # Creates a␣
↪`TensorDataset`
print(dataset[0]) # The first example in our dataset

print()

batch_size = 10
data_iter = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True) #␣
↪Creates a `DataLoader`
next(iter(data_iter)) # Creates an iterator from the `DataLoader` and requests␣
↪the first element
```

```
(tensor([0.3225, 1.2098]), tensor([0.7241]))
```

[33]:
```
[tensor([[ 0.8063, -0.8220],
         [-0.6907,  0.0055],
         [ 0.9742,  0.0982],
         [-0.8409,  0.3336],
         [ 2.4879,  1.4182],
         [-0.1178,  0.0546],
         [-1.4182, -2.2553],
         [ 0.6778,  0.2994],
         [-0.0147,  1.4285],
         [-0.3188, -0.2341]]),
 tensor([[ 8.5917],
         [ 2.7999],
         [ 5.8137],
         [ 1.3876],
         [ 4.3666],
         [ 3.7818],
         [ 9.0093],
         [ 4.5373],
         [-0.6812],
         [ 4.3846]])]
```

4 Defining the Model

- Recall that a linear model can be interpreted as a very simple neural network.
- We will use the neural network functionalities provided by PyTorch to define our model, effectively creating and training a neural network!
- In neural network terms, we need to use a fully-connected linear layer, which is implemented by the `Linear` class in PyTorch.

[34]:
```
num_of_inp = 2 # Number of inputs to the layer
num_of_out = 1 # Number of outputs from the layer
net = torch.nn.Linear(num_of_inp, num_of_out) # Creates our model (a neural␣
↪network with a fully-connected linear layer and a single output)
```

5 Initializing Model Parameters

- We will initialize the parameters of the neural network as in the previous notebook.
- We can modify the parameters by accessing the weights (`net.weight.data`) and the bias (`bias.data`).
- Accessing the `data` of the tensors eliminates the need for `torch.no_grad`.
- The in-place methods `normal_` and `fill_` can be used to overwrite parameter values.

[35]:
```
net.weight.data.normal_(0, 0.01); # Each weight is sampled from a normal␣
↪distribution with mean 0 and standard deviation 0.01.
net.bias.data.fill_(0); # The bias is initialized to 0.
```

6 Defining the Loss Function

- The `MSELoss` class computes the mean squared error.

[36]:
```
loss = torch.nn.MSELoss()
```

7 Defining the Optimization Algorithm

- Stochastic gradient descent is implemented by the `SGD` class.
- The constructor of this class requires a list of parameters to be optimized (which can be obtained through `net.parameters()`) and accepts some hyperparameters (such as the learning rate `lr`).

[37]:
```
optimizer = torch.optim.SGD(net.parameters(), lr=0.5)
```

8 Training Loop

- During each **epoch**:
  - Execute one iteration per minibatch.
  - During each iteration:
    - Obtain the minibatch.
    - Compute predictions and loss using the current model (**forward pass**).
    - Compute the gradients of the loss with respect to model parameters (**backward pass**).
    - Update the model parameters.

[38]:
```python
print('\nInitial parameters:')
print(net.weight)
print(net.bias)

print()

num_epochs = 3
for epoch in range(num_epochs):
    for X, y in data_iter: # Minibatch: `X` and `y`
        y_hat = net(X) # Prediction for the minibatch
        l = loss(y_hat, y) # Loss for the minibatch
        optimizer.zero_grad() # Zeroes the gradient stored inside each parameter
        l.backward() # Computes gradient of `l` with respect to parameters
        optimizer.step() # Updates each parameter based on the gradient stored␣
↪inside it.

    # After each epoch, computes the loss for the entire training dataset
    l = loss(net(features), labels)
    print(f'Epoch {epoch + 1}. Loss: {l:f}')

print('\nLearned parameters:')
print(net.weight)
print(net.bias)

print('\nTrue parameters:')
print(true_w)
print(true_b)
```

```
Initial parameters:
Parameter containing:
tensor([[0.0114, 0.0151]], requires_grad=True)
Parameter containing:
tensor([0.], requires_grad=True)

Epoch 1. Loss: 0.000142
Epoch 2. Loss: 0.000112
Epoch 3. Loss: 0.000136

Learned parameters:
Parameter containing:
tensor([[ 1.9988, -3.3948]], requires_grad=True)
Parameter containing:
tensor([4.2021], requires_grad=True)

True parameters:
tensor([[ 2.0000],
```

```
        [-3.4000]])
tensor(4.2000)
```

9 Evaluation

- Because we created the dataset, we can evaluate our success by comparing the true parameters with the learned parameters.

10 [Storing this notebook as a pdf]

[39]:
```
%%capture
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

!sudo apt-get install texlive-xetex texlive-fonts-recommended
  ↪texlive-plain-generic

# Set the path to this notebook below (add \ before spaces). The output `pdf`
  ↪will be stored in the corresponding folder.
!jupyter nbconvert --to pdf /content/gdrive/My\ Drive/Colab\ Notebooks/nndl/
  ↪week_03/lecture/03_Linear_Regression_Concise.ipynb

# If having issues, save this notebook (File > Save) and restart the session
  ↪(Runtime > Restart session) before running this cell. To debug, remove the
  ↪first line (`%%capture`).
```