université de BORDEAUX

PyTorch: a framework for deep learning

LE Van Linh

van-linh.le@u-Bordeaux.fr

http://www.labri.fr/perso/vle

Contents

université de BORDEAUX

1. Overview

2. PyTorch's main packages

3. Use PyTorch to implement a Deep N-Layer Neural Network

van-linh.le@u-bordeaux.fr

Overview

université de BORDEAUX

- A deep learning framework with Tensor computation and GPU acceleration
- Easy convert from other Python packages: Numpy, SciPy, Cython,…
- Free framework, distributed under BSD-style licensed
- Install pytorch: https://pytorch.org/get-started/locally/

van-linh.le@u-bordeaux.fr

Overview - PyTorch's component

université de BORDEAUX

Component	Description
Torch	A Tensor library like Numpy with strong GPU support
Torch.autograd	Provides all differentiable Tensor operations in torch
Torch.nn	Library for neural networks
Torch.multiprocessing	Python multiprocessing
Torch.utils	DataLoader, Trainer and other utility functions
Torch.legacy	Legacy code which has inherited from Torch for backward propagation

van-linh.le@u-bordeaux.fr

Contents

université de BORDEAUX

1. Overview

2. **PyTorch's main packages**

3. Use PyTorch to implement a Deep N-Layer Neural Network

van-linh.le@u-bordeaux.fr

PyTorch's main packages – Torch.Autograd

université de BORDEAUX

- The central of all neural network in PyTorch
- Provide automatic differentiation for all operations on Tensors
- Enable to compute the gradient automatically (`.requires_grad` attribute)
- Use function `.backward()` to call the computation for gradient

van-linh.le@u-bordeaux.fr

PyTorch's main packages – Torch.nn

université de BORDEAUX

- Contains all components to build a neural network
- **`nn.Module`** is a base class for all neural network
- Your neural network should inherited from this class. It must contain:
  - The layers
  - A method **`forward`** to return the **`output`**

van-linh.le@u-bordeaux.fr

Contents

université de BORDEAUX

1. Overview

2. PyTorch's main packages

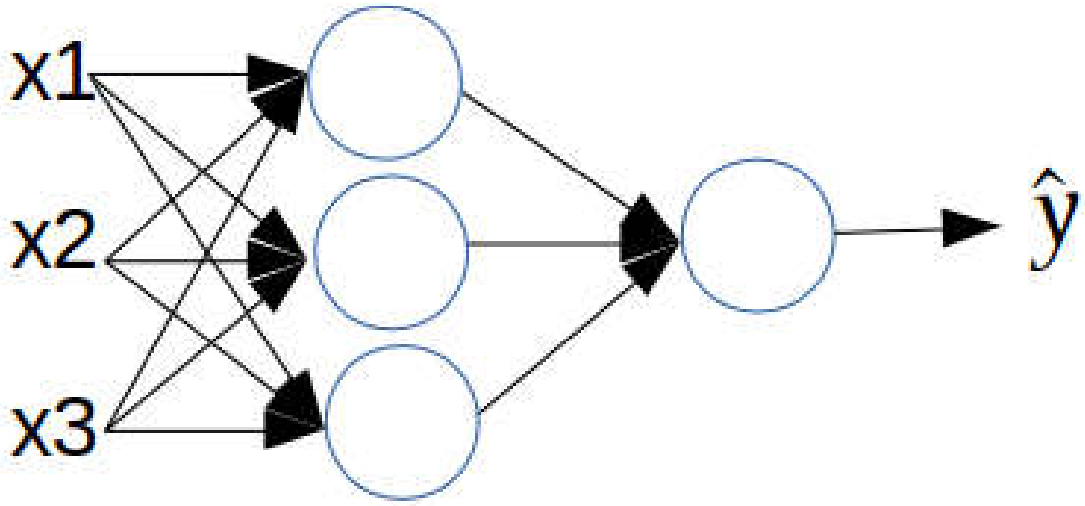
3. **Use PyTorch to implement a Deep N-Layer Neural Network**

van-linh.le@u-bordeaux.fr

Use PyTorch to design a NN

université de BORDEAUX

Lets we consider a neural network with one hidden layer



van-linh.le@u-bordeaux.fr

Use PyTorch to design a NN

université de BORDEAUX

Process:

- Define the neural network with learnable parameters
- Define (choose) the loss function
- Iterate over training data to train the network model, calculate the forward and backward propagation
- Update the parameters

van-linh.le@u-bordeaux.fr

Step 1: Define neural network

université de BORDEAUX

- Inherited from `nn.Module` class
- Implement the `forward` method

```python
import torch
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # define the layers of the network
        self.layer1 = nn.Linear(3,4)
        self.layer2 = nn.Linear(4,1)

    def forward(self, x):
        x = F.relu(self.layer1(x)) # use ReLU function for layer 1
        x = F.sigmoid(self.layer2(x)) # use Sigmoid function for layer 2
        return x

net = Net()
print(net)
```


van-linh.le@u-bordeaux.fr

Step 2: Choose the loss function

université de BORDEAUX

- PyTorch provides several different loss functions
- Distributed in nn package

- *For example*: We use Cross-Entropy loss.

```
criterion = nn.CrossEntropyLoss() # declare a CrossEntropy loss function
# Calculate the loss, where output is prediction of network, target is ground truth
loss = criterion(output, target)
print(loss)
```


van-linh.le@u-bordeaux.fr

Step 3: Calculate the backward propagation

université de BORDEAUX

- Clear the existing gradient (if it exists)
- Call the function: `loss.backward()` to run backward process.

van-linh.le@u-bordeaux.fr

Step 4: Update the parameters

université de BORDEAUX

For example, the simplest rule is used Stochastic Gradient Descent (SGD)

$$weight = weight - learning_{rate} * gradient$$

van-linh.le@u-bordeaux.fr

Step 4: Update the parameters

université de BORDEAUX

`Torch.optim` package

Implement different update rules which are used in Neural Networks such as SGD, Nesterov-SGD, Adam, …

```
1  import torch.optim as optim
2
3  # create your optimizer
4  optimizer = optim.SGD(net.parameters(), lr=0.01)
5
6  # in your training loop:
7  optimizer.zero_grad()   # zero the gradient buffers
8
9  # 1. Train the model (to predict the output)
10 # 2. Compute the loss
11 # 3. Calculate the gradient
12
13 optimizer.step()    # Update the parameters
```


van-linh.le@u-bordeaux.fr

Step 5: Train the network

université de BORDEAUX

Loop over data iterator, and feed the inputs to the network and optimize.

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```


van-linh.le@u-bordeaux.fr

Summary

université de BORDEAUX

- PyTorch framework
- How to use PyTorch to design a Neural Network
- How to use PyTorch to design other variants architectures of deep learning?

van-linh.le@u-bordeaux.fr


université
de BORDEAUX