kptan86@gmail.com

Pytorch Implementation of fully connected deep-learning architecture

In [1]:
```
# import libraries
import numpy as np
import torch
import torch.nn
import torch.optim as optim
```

NETWORK ARCHITECTURE

In [2]:
```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        # DEFINE activation functions
        self.sigmoid = torch.nn.Sigmoid()
        self.tanh    = torch.nn.Tanh()
        self.ReLU    = torch.nn.ReLU()
        self.softmax = torch.nn.Softmax(dim=1)

        # NOTE: fully connection is viewed as a linear layer
        self.fully_connect_01 = torch.nn.Linear(3, 10)
        self.fully_connect_02 = torch.nn.Linear(10, 5)
        self.fully_connect_03 = torch.nn.Linear(5, 3)
    # end def

    def init_weights(self):
        torch.nn.init.xavier_uniform_(self.fully_connect_01.weight)
        torch.nn.init.xavier_uniform_(self.fully_connect_02.weight)
        torch.nn.init.xavier_uniform_(self.fully_connect_03.weight)
    # end def

    # DEFINE layers
    def forward(self, _input):
        layer_00 = _input
        layer_01 = self.fully_connect_01(layer_00)
        layer_02 = self.ReLU            (layer_01)
        layer_03 = self.fully_connect_02(layer_02)
        layer_04 = self.sigmoid         (layer_03)
        layer_05 = self.fully_connect_03(layer_04)
        return layer_05
    # end def
# end class
```

Define network

In [3]:
```
net = Net()
```

Define optimizer

In [4]:
```
LEARNING_RATE = 0.01
MOMENTUM      = 0.9
optimizer = optim.SGD(net.parameters(), lr=LEARNING_RATE, momentum=MOMENTUM)
```

Initialize weights for all neurons

kptan86@gmail.com

In [5]:
```
net.init_weights()
```

Set input values

note that pytorch operates on "tensor", which can be viewed as an array of array

In [6]:
```
x = [(0.5, 0.3, 0.8),]
x = torch.from_numpy(np.array(x)).float()
```

Forward pass (compute value)

In [7]:
```
net.forward(x)
```

Out[7]:
```
tensor([[0.8429, 0.7778, 0.4818]], grad_fn=<AddmmBackward>)
```

Compute error

In [8]:
```
# assumed target values
target = [np.array([0.5, 0.5, 0.5]),]
# convert to tensor
targets = torch.from_numpy(np.array(target)).float()
```

(use MSE error for demo here)

In [9]:
```
criterion = torch.nn.MSELoss()
```

In [10]:
```
outputs = net.forward(x)
loss = criterion(outputs, targets)
```

In [11]:
```
loss
```

Out[11]:
```
tensor(0.0650, grad_fn=<MseLossBackward>)
```

Backward pass (Backpropagation)

In [12]:
```
# reset gradients (clear memory from previous batch)
optimizer.zero_grad()
```

In [13]:
```
loss.backward()
```

Update weights

In [14]:
```
optimizer.step()
```

In [15]:
```
# recompute
net.forward(x)
```

Out[15]:
```
tensor([[0.8371, 0.7734, 0.4825]], grad_fn=<AddmmBackward>)
```

Print gradients

In [16]:
```
weigts_l1 = list(net.fully_connect_01.parameters())[0]
weigts_l2 = list(net.fully_connect_02.parameters())[0]
weigts_l3 = list(net.fully_connect_03.parameters())[0]
```

kptan86@gmail.com

In [17]:
```
print('sigmoid:', np.mean(np.abs(weigts_l1.grad.numpy().flatten())))
print('ReLU:', np.mean(np.abs(weigts_l2.grad.numpy().flatten())))
print('(output):', np.mean(np.abs(weigts_l3.grad.numpy().flatten())))
```

```
sigmoid: 0.008226368
ReLU: 0.00858088
(output): 0.06741866
```

GPU Acceleration

Check if cuda is available

In [19]:
```
torch.cuda.is_available()
```

Out[19]:
```
True
```

set device as GPU (fallback to cpu)

In [20]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

Out[20]:
```
device(type='cuda')
```

Copy network to GPU

In [21]:
```
net = net.to(device)
```

Copy variables to GPU

In [22]:
```
x       =       x.to(device)
targets = targets.to(device)
```

Compute as usual

In [23]:
```
outputs = net.forward(x)
outputs
```

Out[23]:
```
tensor([[0.8371, 0.7734, 0.4825]], device='cuda:0', grad_fn=<AddmmBackward>)
```

In [24]:
```
loss = criterion(outputs, targets)
loss
```

Out[24]:
```
tensor(0.0629, device='cuda:0', grad_fn=<MseLossBackward>)
```

copy back to CPU

In [25]:
```
loss = loss.cpu()
loss
```

Out[25]:
```
tensor(0.0629, grad_fn=<CopyBackwards>)
```


DO NOT DISTRIBUTE