torch_debugging-help

November 1, 2022

1 Fully connected MLP with PyTorch

[1]:
```python
#import necessary packages.\ (we use torch to present results)
print("PyTorch version:", torch.__version__)
```

```
PyTorch version: 1.12.1
```

[2]:
```python
#read and process data (observations X and ground truth Ytrue)

print('Observations (%d samples x %d variables per sample):' % (X.shape[0], X.
 ↪shape[1]))
print(X)
print('Ground truth labels:', Ytrue.shape)
```

```
Observations (10 samples x 1 variables per sample):
tensor([[0.5503],
        [0.9206],
        [0.5359],
        [0.6081],
        [0.0202],
        [0.8545],
        [0.2357],
        [0.4847],
        [0.3996],
        [0.1957]])
Ground truth labels: torch.Size([10, 1])
```

1.1 Create model

[3]:
```python
#for debugging purposes:
def hook_print_values(m, in_, out_):
    print(m._get_name(), '(row per point)')
    print(out_.cpu().data.numpy().squeeze())


# create model (we did it in form of a class)
#class model
```

```
    def __init__(self, ...):
        #...

    #define functions to do the forward pass
    #we added further hooks to execute printing function every fw-pass

    def forward(self, x):
        #...

    def add_hooks(self):
        #...

    def reset_hooks(self):
        #...

#initialize weights (can also be done in a function)


def print_response(...):
    #...


#initialize number of layers and weights
#call modelclass and print the layers of the model
#for debugging purposes, we consider the tanh-function as an additional layer,␣
↪which is not intended
#for the model construction.
#As an additional layer, it provides information of "before and after effects"␣
↪of the activation function.
#This is not necessary and can be ommitted, since that layer can be given␣
↪implicitly in a dense layer.
```

[3]:

```
MyModel(
  (mlist): ModuleList(
    (0): Linear(in_features=1, out_features=3, bias=True)
    (1): Tanh()
    (2): Linear(in_features=3, out_features=1, bias=True)
  )
)
```

1.2 Perform a forward pass without any training

[4]:

```
print("Initial parameter values:")
for param in model.parameters():
    print(param.data)
```

```
#add and reset hooks for debugging purposes (for prints below see function␣
 ↪above)
#model.add_hooks()
#with torch.no_grad():
#    model(X)
#model.reset_hooks()
```

```
Initial parameter values:
tensor([[-0.0224],
        [-0.4284],
        [ 0.0834]])
tensor([ 0.2521,  0.2649, -0.3557])
tensor([[ 0.2152, -0.1047,  0.1244]])
tensor([-0.1316])
Linear (row per point)
[[ 0.23978113  0.02921478 -0.30986056]
 [ 0.23148532 -0.12941168 -0.27899554]
 [ 0.24010374  0.03538335 -0.31106082]
 [ 0.23848625  0.00445483 -0.30504283]
 [ 0.25165692  0.25629523 -0.3540451 ]
 [ 0.23296615 -0.10109624 -0.28450507]
 [ 0.24682909  0.16398089 -0.33608288]
 [ 0.24125077  0.05731605 -0.3153284 ]
 [ 0.24315725  0.09377057 -0.3224216 ]
 [ 0.24772522  0.18111579 -0.33941695]]
Tanh (row per point)
[[ 0.235289    0.02920647 -0.30031022]
 [ 0.22743732 -0.12869406 -0.2719752 ]
 [ 0.23559374  0.0353686  -0.30140185]
 [ 0.23406544  0.0044548  -0.2959207 ]
 [ 0.24647556  0.25082707 -0.33995804]
 [ 0.22884108 -0.10075322 -0.27706948]
 [ 0.24193566  0.16252673 -0.32397586]
 [ 0.2366768   0.05725336 -0.30527675]
 [ 0.23847568  0.0934967  -0.31169492]
 [ 0.24277915  0.17916106 -0.32695678]]
Linear (row per point)
[-0.12141277 -0.10304759 -0.12212807 -0.11853886 -0.14713818 -0.10630424
 -0.13688318 -0.12466807 -0.12887354 -0.13881387]
MyModel (row per point)
[-0.12141277 -0.10304759 -0.12212807 -0.11853886 -0.14713818 -0.10630424
 -0.13688318 -0.12466807 -0.12887354 -0.13881387]
```

1.3 Create the learning components and train the model for a single epoch

[5]:
```
#choose the loss function
#choose the method of optimization (e.g. gradient descent) and initialize its␣
↪parameters respectivly

def train(model,
          loss_fn,
          optimizer,
          num_iter,
          x_train, y_train):

    #implement the training for the model, return the loss.

#train(model,...)

print("New parameter values:")
for param in model.parameters():
    print(param.data)
```

```
0 loss = 0.48179250955581665
New parameter values:
tensor([[-0.0319],
        [-0.4235],
        [ 0.0778]])
tensor([ 0.2902,  0.2461, -0.3352])
tensor([[ 0.2628, -0.0357,  0.0564]])
tensor([0.0571])
```

1.4 Continue training

[6]:
```
#train the model
```

```
0 loss = 0.4379696846008301
200 loss = 0.15191248059272766
400 loss = 0.09892342239618301
600 loss = 0.18876305222511292
800 loss = 0.13503369688987732
1000 loss = 0.12198492139577866
1200 loss = 0.10045844316482544
1400 loss = 0.08721771836280823
1600 loss = 0.08133644610643387
1800 loss = 0.07906016707420349
2000 loss = 0.07795830816030502
2200 loss = 0.07726430892944336
2400 loss = 0.0767684280872345
2600 loss = 0.07639358192682266
2800 loss = 0.07609931379556656
```

[6]: tensor(0.0759, grad_fn=<MseLossBackward0>)

[1]:
```
#plot results: truth and predicted
```

[ ]:
```

```