assignment_1

March 21, 2023

[ ]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
```

1 Test code

[ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

[ ]:
```
torch.manual_seed(1)
```

[ ]:
```
<torch._C.Generator at 0x111d4b6f0>
```

[ ]:
```
# training data
x_train = torch.FloatTensor([[1],[2],[3]])
y_train = torch.FloatTensor([[1],[2],[3]])
print(x_train)
print(x_train.shape)
```

```
tensor([[1.],
        [2.],
        [3.]])
torch.Size([3, 1])
```

[ ]:
```
W = torch.zeros(1, requires_grad = True)
print(W)
```

```
tensor([0.], requires_grad=True)
```

[ ]:
```
b = torch.zeros(1, requires_grad=True)
hypothesis = x_train * W + b
print(hypothesis)
```

```
tensor([[0.],
        [0.],
        [0.]], grad_fn=<AddBackward0>)
```

[ ]:
```
print((hypothesis - y_train) ** 2)
```

```
tensor([[1.],
        [4.],
        [9.]], grad_fn=<PowBackward0>)
```

[ ]:
```
cost = torch.mean((hypothesis - y_train) ** 2)
print(cost)
```

```
tensor(4.6667, grad_fn=<MeanBackward0>)
```

[ ]:
```
optimizer = optim.SGD([W, b], lr = 0.01)
optimizer.zero_grad()
cost.backward()
optimizer.step()
print(W, "\n")
print(b, "\n")
print(hypothesis)
```

```
tensor([0.0933], requires_grad=True)

tensor([0.0400], requires_grad=True)

tensor([[0.],
        [0.],
        [0.]], grad_fn=<AddBackward0>)
```

[ ]:
```
hypothesis = x_train * W + b
print(hypothesis)
```

```
tensor([[0.1333],
        [0.2267],
        [0.3200]], grad_fn=<AddBackward0>)
```

[ ]:
```
cost = torch.mean((hypothesis - y_train) ** 2)
print(cost)
```

```
tensor(3.6927, grad_fn=<MeanBackward0>)
```

2 Result code

[ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(1, 1)

    def forward(self, x):
        return self.linear(x)

# data
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[1], [2], [3]])

# Initial Model
model = LinearRegressionModel()

# Set up optimizer
optimizer = optim.SGD(model.parameters(), lr=0.01)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # H(x) calculate
    prediction = model(x_train)

    # cost calculate
    cost = F.mse_loss(prediction, y_train)

    # cost to H(x)
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # print log per 100 epoch
    if epoch % 100 == 0:
        params = list(model.parameters())
        W = params[0].item()
        b = params[1].item()
        print('Epoch {:4d}/{} W: {:.3f}, b: {:.3f} Cost: {:.6f}'.format(
            epoch, nb_epochs, W, b, cost.item()
        ))
```

```
Epoch    0/1000 W: 0.303, b: 1.002 Cost: 0.560968
Epoch  100/1000 W: 0.621, b: 0.862 Cost: 0.107054
Epoch  200/1000 W: 0.702, b: 0.677 Cost: 0.066153
Epoch  300/1000 W: 0.766, b: 0.533 Cost: 0.040878
Epoch  400/1000 W: 0.816, b: 0.419 Cost: 0.025260
Epoch  500/1000 W: 0.855, b: 0.329 Cost: 0.015609
Epoch  600/1000 W: 0.886, b: 0.259 Cost: 0.009646
Epoch  700/1000 W: 0.911, b: 0.203 Cost: 0.005960
Epoch  800/1000 W: 0.930, b: 0.160 Cost: 0.003683
Epoch  900/1000 W: 0.945, b: 0.126 Cost: 0.002276
Epoch 1000/1000 W: 0.957, b: 0.099 Cost: 0.001406
```

Reference * Title: AI and deeplearning lecture (INU) * Author: Minsuk Koo, Ph.D. * Availability: koo@inu.ac.kr

---