Intro. to Machine Learning / Deep Learning

shpre1236

shpre1236

2025.12.01

Previously...

- What are Machine Learning and Deep Learning?
- What is gradient descent?
- How does gradient descent work?

Homework review

Q1...Q3 will be covered in a programming section. (But strongly recommended to solve it by hand)

Homework review

Q4.

$$y = 3x + 4$$
$$y = 2x + 2$$

$$\begin{bmatrix} 3 & -1 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -4 \\ -2 \end{bmatrix}$$
$$\mathrm{Ax} = \mathrm{b}$$

$$\mathrm{x} = \mathrm{A}^{-1}\mathrm{b}$$
$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{3 \cdot (-1) - 2 \cdot (-1)} \begin{bmatrix} -1 & 1 \\ -2 & 3 \end{bmatrix} \begin{bmatrix} -4 \\ -2 \end{bmatrix}$$

Homework review

Q5.

$$y_1 = \mathrm{A}\mathrm{x} + b$$
$$\frac{dy_1}{d\mathrm{x}} = \mathrm{A}^T$$

$$y_2 = \mathrm{x}^T\mathrm{A} + b$$
$$\frac{dy_2}{d\mathrm{x}} = \mathrm{A}$$

$$y_3 = \mathrm{x}^T\mathrm{B}\mathrm{x}$$
$$\frac{dy_3}{d\mathrm{x}} = \mathrm{B} + \mathrm{B}^T$$

Homework review

Q5.

$$
\begin{aligned}
Q &= (\mathrm{Ax} - \mathrm{b})^2 \\
&= (\mathrm{Ax} - \mathrm{b})^T (\mathrm{Ax} - \mathrm{b}) \\
\frac{dQ}{d\mathrm{x}} &= \cdots
\end{aligned}
$$

Today's Topic

- gradient descent implementation
- Pytorch introduction
- Building Deep learning model

Gradient Descent

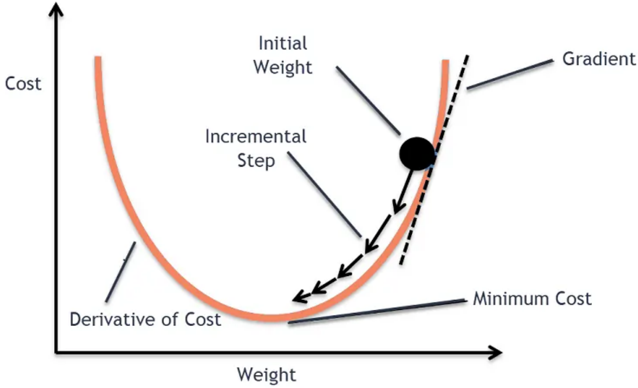

Figure 1: Gradient descent

Implement the algorithm in python

Pytorch

Pytorch is an open-source deep learning library.

autograd feature in pytorch enables the user to design and deploy a deep learning model easily.
It automatically calculate the gradient of model with respect to the input variable.

Pytorch

day3.py > ...

```python
x0 = 5.0
num_epoch = 5
learning_rate = 1.0

x = x0
for i in range(num_epoch):
    # initialize
    grad = 0

    # (optional) calculate output
    y = (x - 3) ** 2 + 4

    # calculate gradient
    grad = 2 * (x - 3)

    # update x variable
    x = x - grad * learning_rate

    # print states
    print(y, grad, x)
```

(a) Gradient Descent Implementation

```python
x0 = 5.0
num_epoch = 10
learning_rate = 0.1

x = torch.tensor(x0, requires_grad=True)

loss_func = nn.MSELoss()
optimizer = optim.SGD([x], lr=learning_rate)

for i in range(num_epoch):
    # initialize
    optimizer.zero_grad()

    # (optional) calculate output
    y = (x-3)**2 + 4

    # calculate gradient
    y.backward()

    # update x variable
    optimizer.step()

    # print states
    print(y.item(), x.grad.item(), x.item())
```

(b) Pytorch implementation

Neural Network model

```python
num_data = 10000
num_epoch = 10000
learning_rate = 0.01

x = init.uniform_(torch.Tensor(num_data,1),0,6)
y = (x - 3)**2 + 4
noise = init.normal_(torch.FloatTensor(num_data,1), std=0.5)
y_noise = y + noise

model = nn.Sequential(
    nn.Linear(1,10),
    nn.ReLU(),
    nn.Linear(10,30),
    nn.ReLU(),
    nn.Linear(30,10),
    nn.ReLU(),
    nn.Linear(10,1)
)

loss_func = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

loss_array = []
for i in range(num_epoch):
    #initialize
    optimizer.zero_grad()

    # calculate output and loss
    output = model(x)
    loss = loss_func(output, y_noise)

    # calculate gradient
    loss.backward()

    # update
    optimizer.step()

    # (optional) save loss value
    loss_array.append(loss.item())

print(model(torch.FloatTensor([3.0])).item())
print(model(torch.FloatTensor([4.0])).item())
print(model(torch.FloatTensor([2.0])).item())
```

Figure 3: Neural Network Model

CNN

LeNet-5

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.layer = nn.Sequential(
            nn.Conv2d(1,16,5),
            nn.ReLU(),
            nn.Conv2d(16,32,5),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(32,64,5),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
        )
        self.fc_layer = nn.Sequential(
            nn.Linear(64*3*3, 100),
            nn.ReLU(),
            nn.Linear(100,10)
        )

    def forward(self,x):
        out = self.layer(x)
        out = out.view(batch_size,-1)
        out = self.fc_layer(out)
        return out


device = torch.device("cuda:0")
model = CNN().to(device=device)
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

loss_arr = []
for i in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        x = image.to(device)
        y = label.to(device)

        optimizer.zero_grad()
        output = model.forward(x)
        loss = loss_func(output,y)
        loss.backward()
        optimizer.step()

```

Figure 4: LeNet-5

Today's summary

- Implementation of Gradient descent
- Pytorch introduction
- Deep Neural Network implementation