Лекция: PyTorch

В предыдущих сериях

1. Принцип максимального правдоподобия
2. Функция softmax - что это и зачем?
3. Какой метод оптимизации чаще всего используется для оптимизации параметров в алгоритмах машинного обучения? В чём заключается этот метод?
4. Стандартный цикл тренировки модели.
5. Обратное распространение ошибки.

Производная: коротко о главном

$$(u(v))' = u'(v) \cdot v'$$

Backpropagation error

1. Неплохая статья с примерами кода
2. Ролик с объяснением на английском с русскими субтитрами

Тервер и матстат?

Изучить книгу “Статистика и котики” – понятно, наглядно, без воды, больше похожа на конспект.

С 20.10 до 26.10 – вам всем будет предложено заполнить гугл-форму с несколькими содержательными вопросами.

Цель библиотек машинного обучения

1. Быстрая реализация и тестирование идей
2. Автоматическое вычисление градиентов
3. Возможность выполнения кода на GPU

PyTorch – базовые элементы

Tensor: numpy-массив, но может вычисляться на GPU.

Autograd: пакет для построения графа вычислений и автоматически вычислять градиент.

Module: слой нейронной сети.

PyTorch VS NumPy

Граф вычислений

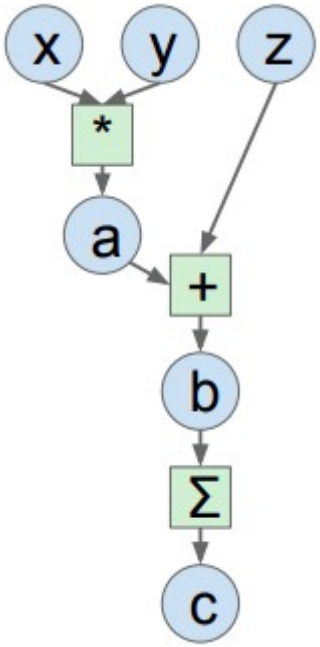
```python
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```




```python
import torch

N, D = 3, 4
x = torch.randn(N, D, requires_grad=True)
y = torch.randn(N, D)
z = torch.randn(N, D)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()
print(x.grad)
```

PyTorch: Tensors

```
np.random.seed(2)
Y = np.random.randn(2, 5)
X = np.random.randn(2, 3)
W = np.random.randn(3, 5)
lam = 0.5

pred = X.dot(W)
pred_loss = np.sum((pred - Y)**2)
reg_loss = lam * np.sum(np.abs(W))
loss = pred_loss + reg_loss
```

Create random tensors for data and weights

```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

Forward pass: compute predictions and loss

Backward pass: manually compute gradients

Gradient descent step on weights

Autograd

```
import torch
Y = torch.Tensor(np.random.randn(2, 5))
X = torch.Tensor(np.random.randn(2, 3))
W = torch.Tensor(np.random.randn(3, 5))
lam = 0.5
pred = X.matmul(W)
pred_loss = torch.sum((pred - Y)**2)
reg_loss = lam * torch.sum(torch.abs(W))
loss = pred_loss + reg_loss
```

forward

```
Y = torch.Tensor(np.random.randn(2, 5))
X = torch.Tensor(np.random.randn(2, 3))
W = torch.Tensor(np.random.randn(3, 5)).requires_grad_(True)
l = 0.5

pred = X.matmul(W)
pred_loss = torch.sum((pred - Y)**2)
reg_loss = l * torch.sum(torch.abs(W))
loss = pred_loss + reg_loss

loss.backward()
print(W.grad)
```

Modules

```
nn = torch.nn.Sequential(
    torch.nn.Linear(3, 10),
    torch.nn.ReLU(),
    torch.nn.Linear(10, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 3))

nn(X)
```

```
import torch.nn.functional as F

class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = torch.nn.Linear(3, 10)
        self.fc2 = torch.nn.Linear(10, 20)
        self.fc3 = torch.nn.Linear(20, 3)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

nn = Net()
nn(X)
```

```
import torch.optim as optim
y = torch.LongTensor(np.random.randint(0,3,size=2))

nn = torch.nn.Sequential(
    torch.nn.Linear(3, 10),
    torch.nn.ReLU(),
    torch.nn.Linear(10, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 3)
)

optimizer = optim.SGD(nn.parameters(), lr=0.01, weight_decay=0.05)

for i in range(100):
    optimizer.zero_grad()
    pred = nn(X)
    criterion = torch.nn.CrossEntropyLoss()
    loss = criterion(pred, y)
    loss.backward()
    optimizer.step()
```

L2