PyTorch

Области применения

- распознавание образов на изображениях;
- компьютерное зрение, обнаружение движущихся объектов;
- поиск закономерностей, анализ данных, в том числе неструктурированных;
- обработка естественного языка, распознавание речи и машинный перевод;
- создание машинных описаний для изображений;
- анализ текстов и поиск в них информации;
- генерация текстового контента и картинок.

Отличия PyTorch от других ML-фреймворков

- Динамические вычисления

Отличия PyTorch от других ML-фреймворков

- Динамические вычисления
- Автоматическое дифференцирование

Отличия PyTorch от других ML-фреймворков

- Динамические вычисления
- Автоматическое дифференцирование
- CUDA

Структура PyTorch

- Тензоры

Особенности тензоров в PyTorch:

•Они оптимизированы для автоматического дифференцирования, то есть поиска производной «на каждом шаге».
•Их можно запускать и на основных процессорных мощностях, и на видеокарте, перемещать с обычного процессора на графический и обратно.

Структура PyTorch

- Тензоры
- Встроенные операции

Структура PyTorch

- Тензоры
- Встроенные операции
- Datasets

Структура PyTorch

- Тензоры
- Встроенные операции
- Datasets
- Data Loader

Модули PyTorchrch

- Autograd
- Optim
- nn

PyTorch

- позволяет строить вычислительные динамические графы
- больше действий для экспериментов с моделями
- более гибкая настройка моделей
- необходимо самостоятельно описывать процедуру обучения
- более сложный, но более мощный инструмент

Keras

- предлагает последовательные и простые API-интерфейсы
- сводит к минимуму количество действий пользователя, необходимых для типичных случаев использования
- содержит обширную документацию и руководства для разработчиков
- модели строятся по принципу конструктора из готовых блоков
- быстрый запуск обучения

PyTorch	TensorFlow
• PyTorch тесно связан с фреймворком Torch, который активно используется в Facebook.	• TensorFlow разработан Google Brain и активно используется в Google.
• PyTorch является относительно новым по сравнению с другими конкурентными технологиями.	• TensorFlow не является новым и рассматривается в качестве инструмента, необходимого для работы.
• PyTorch включает в себя все в императивной и динамичной манере.	• TensorFlow включает в себя статические и динамические графики в виде комбинации.
• Граф вычислений в PyTorch определяется во время выполнения.	• TensorFlow не включает опцию времени выполнения.
• PyTorch включает в себя возможность развертывания для мобильных и встроенных платформ.	• TensorFlow лучше работает для встроенных фреймворков.

Keras. Пример полносвязной сети

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy’])

x_train, y_train = ...

model.fit(x_train, y_train, epochs=150, batch_size=10)

y_test = model.predict(x_test)
```

PyTorch. Пример полносвязной сети

```
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
class Network(nn.Module):
        def __init__(self):
                super(Network, self).__init__()
                self.l1 = nn.Linear(8, 12)
                self.relu1 = nn.ReLU()
                self.l2 = nn.Linear(12, 8)
                self.relu2 = nn.ReLU()
                self.l3 = nn.Linear(8, 1)
        def forward(self, x):
                x = self.l1(x)
                x = self.relu1(x)
                x = self.l2(x)
                x = self.relu2(x)
                x = self.l3(x)
                return F.log_softmax(x)
```

PyTorch. Пример полносвязной сети

```
model = Network()
optimizer = optim.Adam(model.parameters())
loss_func = nn.CrossEntropyLoss()

x_train, y_train = ...
batch_size = 10
epochs = 150

for e in range(epochs):
        for i in range(0, x_train.shape[0], batch_size):
                x_var = Variable(x_train[i:i + batch_size])
                y_var = Variable(y_train[i:i + batch_size])

                optimizer.zero_grad()
                net_out = model(x_var)
                loss = loss_func(net_out, y_var)

                loss.backward()
                optimizer.step()
        print('Epoch: {} - Loss: {:.6f}'.format(e, loss.data[0]))
net_out = model(x_test)
```

Тензоры

```
import torch
x = torch.Tensor(2, 3)

x = torch.rand(2, 3)

x = torch.ones(2,3)
y = torch.ones(2,3) * 2
x + y

y[:,1] = y[:,1] + 1
```

Автоматическое дифференцирование в PyTorch

```
x = Variable(torch.ones(2, 2) * 2, requires_grad=True

z = 2 * (x * x) + 5 * x

z.backward(torch.ones(2, 2))
print(x.grad)
```

Создание нейронной сети в PyTorch

Класс для построения нейронной сети

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
      def __init__(self):
            super(Net, self).__init__()
            self.fc1 = nn.Linear(28 * 28, 200)
            self.fc2 = nn.Linear(200, 200)
            self.fc3 = nn.Linear(200, 10)
```

Класс для построения нейронной сети

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

Класс для построения нейронной сети

```
net = Net()
print(net)
```

```
Net (
(fc1): Linear (784 -> 200)
(fc2): Linear (200 -> 200)
(fc3): Linear (200 -> 10)
)
```

Тренировка сети

```
# Осуществляем оптимизацию путем
стохастического градиентного спуска
optimizer = optim.SGD(net.parameters(),
            lr=learning_rate, momentum=0.9)


# Создаем функцию потерь
criterion = nn.NLLLoss()
```

Тренировка сети

```
# запускаем главный тренировочный цикл
for epoch in range(epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
      data, target = Variable(data), Variable(target)
# изменим размер с (batch_size, 1, 28, 28) на (batch_size, 28*28)
      data = data.view(-1, 28*28)
      optimizer.zero_grad()
      net_out = net(data)
      loss = criterion(net_out, target)
      loss.backward()
      optimizer.step()
      if batch_idx % log_interval == 0:
         print('Train Epoch: {} [{}/{} ({:.0f}%)]tLoss: {:.6f}'.format(
           epoch, batch_idx * len(data), len(train_loader.dataset),
             100. * batch_idx / len(train_loader), loss.data[0]))
```

Тренировка сети

```
# запускаем главный тренировочный цикл
for epoch in range(epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
# изменим размер с (batch_size, 1, 28, 28) на (batch_size, 28*28)
        data = data.view(-1, 28*28)
        optimizer.zero_grad()
        net_out = net(data)
        loss = criterion(net_out, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                    100. * batch_idx / len(train_loader), loss.data[0]))
```

Тренировка сети

```
# запускаем главный тренировочный цикл
for epoch in range(epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
      data, target = Variable(data), Variable(target)
# изменим размер с (batch_size, 1, 28, 28) на (batch_size, 28*28)
      data = data.view(-1, 28*28)
      optimizer.zero_grad()
      net_out = net(data)
      loss = criterion(net_out, target)
      loss.backward()
      optimizer.step()
      if batch_idx % log_interval == 0:
         print('Train Epoch: {} [{}/{} ({:.0f}%)]tLoss: {:.6f}'.format(
           epoch, batch_idx * len(data), len(train_loader.dataset),
             100. * batch_idx / len(train_loader), loss.data[0]))
```

Тренировка сети

```
# запускаем главный тренировочный цикл
for epoch in range(epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
# изменим размер с (batch_size, 1, 28, 28) на (batch_size, 28*28)
        data = data.view(-1, 28*28)
        optimizer.zero_grad()
        net_out = net(data)
        loss = criterion(net_out, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                    100. * batch_idx / len(train_loader), loss.data[0]))
```

Тренировка сети

```
Train Epoch: 9 [52000/60000 (87%)] Loss: 0.015086

Train Epoch: 9 [52000/60000 (87%)] Loss: 0.015086

Train Epoch: 9 [54000/60000 (90%)] Loss: 0.030631

Train Epoch: 9 [56000/60000 (93%)] Loss: 0.052631

Train Epoch: 9 [58000/60000 (97%)] Loss: 0.052678
```

Тестирование сети

```
test_loss = 0
correct = 0
for data, target in test_loader:
    data, target = Variable(data, volatile=True), Variable(target)
    data = data.view(-1, 28 * 28)
    net_out = net(data)
# Суммируем потери со всех партий
    test_loss += criterion(net_out, target).data[0]
# получаем индекс максимального значения
    pred = net_out.data.max(1)[1]
    correct += pred.eq(target.data).sum()

test_loss /= len(test_loader.dataset)
print('nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)n'.forma
       test_loss, correct, len(test_loader.dataset),
       100. * correct / len(test_loader.dataset)))
```

Тестирование сети

```
test_loss = 0
correct = 0
for data, target in test_loader:
    data, target = Variable(data, volatile=True), Variable(target)
    data = data.view(-1, 28 * 28)
    net_out = net(data)
# Суммируем потери со всех партий
    test_loss += criterion(net_out, target).data[0]
# получаем индекс максимального значения
    pred = net_out.data.max(1)[1]
    correct += pred.eq(target.data).sum()

test_loss /= len(test_loader.dataset)
print('nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)n'.forma
       test_loss, correct, len(test_loader.dataset),
       100. * correct / len(test_loader.dataset)))
```

Тестирование сети

```
test_loss = 0
correct = 0
for data, target in test_loader:
    data, target = Variable(data, volatile=True), Variable(target)
    data = data.view(-1, 28 * 28)
    net_out = net(data)
# Суммируем потери со всех партий
    test_loss += criterion(net_out, target).data[0]
# получаем индекс максимального значения
    pred = net_out.data.max(1)[1]
    correct += pred.eq(target.data).sum()

test_loss /= len(test_loader.dataset)
print('nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)n'.forma
       test_loss, correct, len(test_loader.dataset),
       100. * correct / len(test_loader.dataset)))
```

Тестирование сети

```
test_loss = 0
correct = 0
for data, target in test_loader:
    data, target = Variable(data, volatile=True), Variable(target)
    data = data.view(-1, 28 * 28)
    net_out = net(data)
# Суммируем потери со всех партий
    test_loss += criterion(net_out, target).data[0]
# получаем индекс максимального значения
    pred = net_out.data.max(1)[1]
    correct += pred.eq(target.data).sum()

test_loss /= len(test_loader.dataset)
print('nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)n'.forma
       test_loss, correct, len(test_loader.dataset),
       100. * correct / len(test_loader.dataset)))
```

Тестирование сети

```
Test set: Average loss: 0.0003, Accuracy: 9783/10000 (98%)
```