О библиотеке PyTorch

расскажи про библиотеку pytorch

PyTorch — это популярная **открытая библиотека для машинного обучения**, разработанная Facebook AI Research. Она широко используется для создания и обучения нейронных сетей благодаря простоте, гибкости и поддержке динамических вычислительных графов.

✅ Особенности PyTorch

1. **Динамический вычислительный граф**
   - Граф строится «на лету» во время выполнения, а не заранее.
   - Это удобно для задач с переменной длиной входа (например, NLP).
2. **Tensor API (как в NumPy)**
   - `torch.Tensor` похож на массив NumPy, но поддерживает вычисления на **GPU**.
3. **Автоматическое дифференцирование**
   - Модуль `torch.autograd` автоматически считает градиенты для обучения нейросетей (backpropagation).
4. **Глубокая интеграция с Python**
   - Можно использовать нативный Python-код, без статических графов, как в TensorFlow 1.x.
5. **Поддержка GPU и TPU**
   - Простая передача данных на GPU (`.to(device)` или `.cuda()`).

✅ Основные модули PyTorch

- **torch** — операции с тензорами.
- **torch.nn** — слои нейросетей и функции активации.
- **torch.optim** — оптимизаторы (SGD, Adam и др.).
- **torch.utils.data** — загрузка и обработка данных (Dataset, DataLoader).
- **torchvision** — работа с изображениями (dataset, трансформации, pretrained модели).

✅ Ключевые функции и объекты

Функция / Класс	Назначение
`torch.tensor()`	Создаёт тензор
`x.to(device)`	Перенос тензора на CPU/GPU
`torch.nn.Linear()`	Полносвязный слой
`torch.nn.ReLU()`	Функция активации
`torch.optim.Adam()`	Оптимизатор Adam
`loss.backward()`	Вычисление градиентов
`optimizer.step()`	Шаг оптимизации

✅ Пример: простая нейронная сеть

python

```python
import torch
import torch.nn as nn
import torch.optim as optim

# 1. Данные
x = torch.randn(100, 3)  # 100 образцов, 3 признака
y = torch.randn(100, 1)  # целевая переменная

# 2. Модель
model = nn.Sequential(
    nn.Linear(3, 10),
    nn.ReLU(),
    nn.Linear(10, 1)
)

# 3. Функция потерь и оптимизатор
criterion = nn.MSELoss()
```

```
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 4. Обучение
for epoch in range(100):
    y_pred = model(x)
    loss = criterion(y_pred, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

print(f'Final loss: {loss.item()}')
```

✅ Применение PyTorch

- **Компьютерное зрение** (CV): классификация, детекция объектов (ResNet, YOLO).
- **Обработка естественного языка (NLP)**: BERT, GPT, LSTM.
- **Рекомендательные системы**.
- **Генеративные модели**: GAN, VAE.
- **Научные исследования и прототипирование**.

Хочешь, я сделаю **сравнение PyTorch и TensorFlow** или **объясню как работает autograd**?

что такое тензоры и операции с тензорами.

Тензоры в PyTorch — это **основная структура данных**, очень похожая на массивы NumPy, но с дополнительными возможностями: поддержка **GPU** и **автоматического дифференцирования**.

✅ Что такое тензор?

Тензор — это обобщение понятий:

- **Скаляр** (одно число) → тензор 0-го порядка.

- **Вектор** (список чисел) → тензор 1-го порядка.
- **Матрица** (таблица чисел) → тензор 2-го порядка.
- **Более высокие размерности** (например, 3D-данные, видео, батчи изображений) → тензоры 3-го и выше порядка.

Пример:

- `torch.tensor(5)` → скаляр (shape = `[]`)
- `torch.tensor([1, 2, 3])` → вектор (shape = `[3]`)
- `torch.tensor([[1, 2], [3, 4]])` → матрица (shape = `[2, 2]`)
- 4D тензоры обычно используются для изображений в формате `(batch, channels, height, width)`.

✅ Создание тензоров

python

```python
import torch

# Из Python списка
a = torch.tensor([1, 2, 3])

# С заданным типом
b = torch.tensor([1.0, 2.0], dtype=torch.float32)

# Единичная матрица 3x3
eye = torch.eye(3)

# Нули и единицы
zeros = torch.zeros((2, 3))
ones = torch.ones((2, 3))

# Случайные значения
rand = torch.rand((2, 3))

print(rand.shape)  # (2, 3)
```

✅ Основные операции с тензорами

Тензоры поддерживают все базовые математические операции и линейную алгебру.

Арифметика

```python
x = torch.tensor([1, 2, 3])
y = torch.tensor([4, 5, 6])

print(x + y)   # [5, 7, 9]
print(x * y)   # [4, 10, 18]
print(x ** 2)  # [1, 4, 9]
```

Матричные операции

```python
A = torch.tensor([[1, 2], [3, 4]])
B = torch.tensor([[5, 6], [7, 8]])

# Скалярное произведение
dot = torch.dot(torch.tensor([1, 2]), torch.tensor([3, 4]))  # 11

# Матричное умножение
matmul = torch.matmul(A, B)  # [[19, 22], [43, 50]]
```

Изменение формы (reshape)

```python
x = torch.arange(12)  # [0,1,2,...,11]
x = x.reshape(3, 4)   # матрица 3x4
```

Объединение (concatenate, stack)

```python
a = torch.tensor([1, 2])
b = torch.tensor([3, 4])
```

```
cat = torch.cat((a, b))       # [1, 2, 3, 4]
stack = torch.stack((a, b))   # [[1, 2], [3, 4]]
```

Срезы и индексация

```python
x = torch.arange(10)
print(x[:5])  # первые 5 элементов
print(x[::2])  # каждый второй элемент
```

✅ Операции на GPU

Тензоры можно переносить на **GPU**:

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
x = torch.ones(3, 3).to(device)
```

Хочешь, я объясню **чем тензоры отличаются от массивов NumPy** или **как работает автоматическое дифференцирование для тензоров**?