

# ????????

## Типы обучения:

- С учителем: есть данные и метки.
- Без учителя: есть данные и ожидаемое количество меток. Проводится классификация, затем говорится: это то, это= это.
- Трансферное обучение: используем некую уже обученную модель и дообучаем ее.
- Обучение с подкреплением: среда, агент и награда за корректный ответ.

## Тензоры

```
scalar = torch.tensor(7)      # создание тензора
scalar.ndim                     # размерность 0 это один на один
scalar.item()                  # элементы тензора
tns = torch.rand(3, 4)        # случайный тензор
zero_tns = torch.zeros(3, 4)  # тензор из нулей
one_tns = torch.ones(3, 4)     # тензор из единиц
one_tns.dtype                  # тип данных в тензоре float32,
one_to_thirtyone = torch.arange(start=1, end=43, step=15) # tensor([ 1, 16, 31])
three_zeros_like = torch.zeros_like(input=one_to_thirtyone) # тензор похожий на шаблонный но
все нули
one_tns.T                      # транспонирование тензора
```

dtype определяет тип данных. По умолчанию float32

```
one_to_ten = torch.tensor([3.0, 6.0, 9.0], dtype=torch.float16)
print(one_to_ten.dtype)
float_16_tensor = float_32_tensor.type(torch.float16) # преобразование типов данных
```

device определяет устройство. По умолчанию cpu. Может быть cuda. Если тензоры на разных устройствах - будет ошибка.

```
tensor_on_gpu = tensor.to("cuda:0")
tensor_on_cpu = tensor_on_gpu.cpu()
```

Тензор на gpu нельзя преобразовать в numpy

requires\_grad=False - расчет градиентов.

### 3 частые ошибки при работе с тензорами:

- неправильный тип данных
- неправильная форма
- тензоры при операции на разных устройствах

### Операции с тензорами

Скалярные сложение, разность, умножение, деление тензора на число - как в python.

Векторное умножение `torch.matmul(tenz, tenz)`

### Дополнительные методы

```
torch.min # само значение
torch.max

torch.argmaxin # индекс минимального значения
torch.argmax

torch.mean # усреднение значений
torch.sum

# reshape - преобразование размера входного тензора в нужный размер
# размер (кол-во элементов, т е произведение размерностей) нового и старого тензоров должны
совпадать.
x_reshaped = x.reshape(3, 3)
# view - возвращает вид входного тензора в указанном размере, но сохраняет память
оригинального тензора
# т е это просто ссылка, а reshape создает копию
x_reviewed = x.view(1,9)
# stack - склеивает тензоры по горизонтали или вертикали
# dim=0 - по горизонтали
# dim=1 - по вертикали
x_new = torch.stack([x,x,x], dim=0)
# squeeze - удаляет все единичные измерения из тензора

# unsqueeze - добавляет одно единичное измерение к тензору

# permute - изменение порядка на входе функции последовательность новой
x_new = x_cur.permute(2,1,0)
```

## Нумерация элементов тензора

```
x = torch.arange(1., 10.)
x_v = x.view(1, 3, 3)
print(x_v[0,2,2]) # третий элемент третьей строки первого блока
print(x_v[:, :, 2]) # tensor([[3., 6., 9.]])
```

*При конвертации в pytorch обязательно помнить о возможном несоответствии типов данных*

Воспроизводимость: получить такой же случайный тензор. Нужно вызывать перед каждой генерацией случайных чисел.

```
torch.manual_seed(SOME_NUMBER)
```

## Функции потерь

Разница в каком-то виде между требуемыми и полученными данными. Их довольно много.

## Запуск на GPU

### Тренды библиотек ИИ

<https://www.learnpytorch.io>

<https://colab.research.google.com/>

Библиотека машинного обучения. Можно запускать предобученные модели. Например для компьютерного зрения:

```
from torchvision import models

print(dir(models))
```

Названия в верхнем регистре - классы, реализующие популярные архитектуры. В нижнем - экземпляры сетей с определенным количеством слоев, нейронов, возможно с весами, т е объекты.

---

Revision #14

Created 9 April 2026 02:12:24 by Admin

Updated 13 April 2026 01:42:34 by Admin