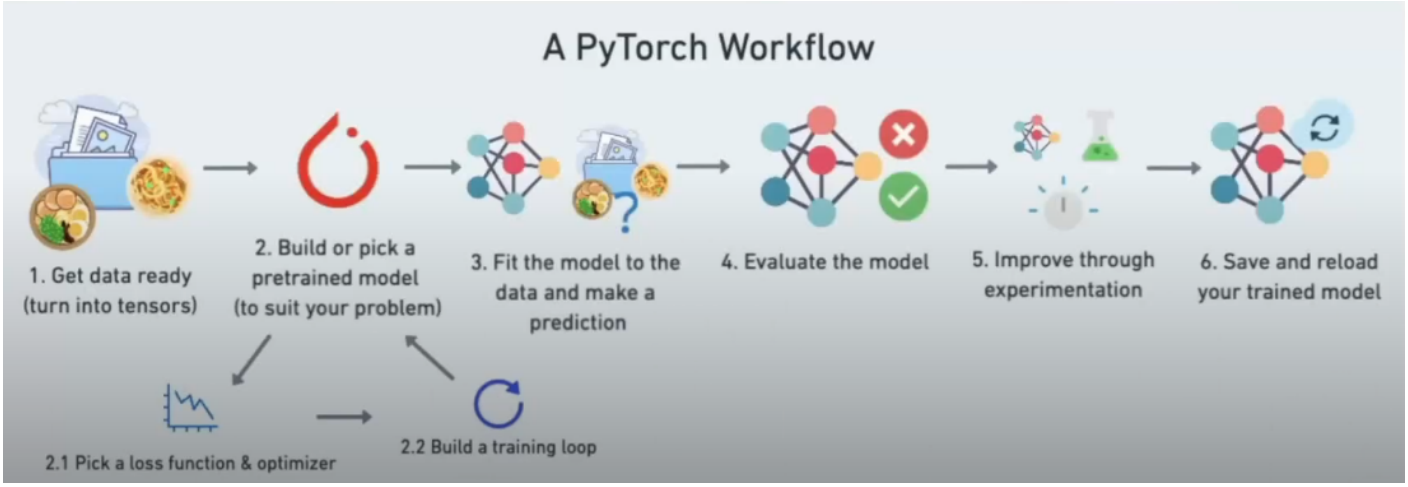
??????? ???????? ?????? ? pytorch



- Класс nn содержит все необходимое для создания нейросети
- nn.Parameter - какие параметры может наша модель пробовать обучить, обычно слой PyTorch будет настраивать их
- nn.Module - базовый класс нейросетей, все модели потомки этого класса. Должен содержать переопределенную процедуру forward
- torch.optim - оптимизатор при создании модели
- torch.inference_mode() - режим вывода при прогнозировании. Убирает очень много дополнительных данных

Приближение модели к нужным параметрам происходит с помощью фукции потерь (Loss function) Она показывает, насколько сильно полученные данные отличаются от требуемых.

Оптимизатор учитывает потери модели и корректирует параметры модели.

```
import torch
from torch import nn
import matplotlib.pyplot as plt

'''
Задача: сделаем данные на основе линейной регрессии (линейного уравнения).
Обучим модель и попытаемся предсказать значения.
'''

def plot_data(train_data=[], train_labels=[],
```

```
                test_data=[], test_labels=[],
                prediction=None):
    plt.figure(figsize=(10, 7))
    plt.scatter(train_data, train_labels, c='b', s=4)
    plt.scatter(test_data, test_labels, c='g', s=4)
    if prediction is not None:
        plt.scatter(test_data, prediction, c='r', s=4)

    plt.show()

class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.weights = nn.Parameter(torch.randn(1,
                                                requires_grad=True,
                                                dtype=torch.float))
        self.bias = nn.Parameter(torch.randn (1,
                                              requires_grad=True,
                                              dtype=torch.float))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.weights * x + self.bias


weight = 0.7
bias = 0.3

# стартовый набор данных
start = 0
end = 1
step = 0.02

X = torch.arange(start, end, step).unsqueeze(dim=1)
Y = weight * X + bias

# Разделим данные на обучающие 60%-80%, тестовые 20%
train_size = int(0.8 * len(X))
X_train, Y_train = X[:train_size], Y[:train_size]
X_test, Y_test = X[train_size:], Y[train_size:]
```

```
torch.manual_seed(42)
model_0 = LinearRegressionModel()
print(list(model_0.parameters()))

# делаем предсказание о качестве метки
with torch.inference_mode():
    y_preds = model_0(X_test)

# Рисуем тестовые данные
plot_data(X_train, Y_train, X_test, Y_test, y_preds)
```

Сохранение модели.

torch.save() / torch.load() Стандартная сериализация и сохранение любого объекта, в частности - torch. Можно через state_dict, Сохраняет текущее состояние модели. По ощущениям проще save/load. Хотя наверное при гигабайтных моделях объект будет весить 1:10.

```
from pathlib import Path

model_0 = LinearRegressionModel()
...

MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parent=True, exist_ok=True)
MODEL_NAME = "testname.pt"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME

torch.save(obj=model_0.state_dict(), f=MODEL_SAVE_PATH)
# новый объект, загруженный из файла
loaded_model = LinearRegressionModel()
loaded_model.load_state_dict(torch.load(f=MODEL_SAVE_PATH))
```

Revision #5
Created 10 April 2026 07:35:46 by Admin
Updated 20 April 2026 02:43:41 by Admin