Документация по обучению нейросети на PyTorch с нуля

🔷Введение

Этот документ — подробное руководство по созданию и обучению простой нейронной сети с нуля на языке Python с использованием библиотеки **PyTorch**. Он предназначен для новичков, которые не имеют опыта в машинном обучении и хотят понять, как работает нейросеть, какие параметры на что влияют, и как анализировать результаты.

🔷Содержание

---

1. Загрузка данных

Мы используем **датасет MNIST**, который содержит изображения цифр (от 0 до 9) в виде 28x28 пикселей.

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

transform = transforms.Compose([transforms.ToTensor()])
train_data = datasets.MNIST(root='.', train=True, download=True,
transform=transform)
test_data = datasets.MNIST(root='.', train=False, download=True,
transform=transform)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64)
```

ToTensor() переводит картинку из PIL-формата в тензор (матрицу), который понимает PyTorch.

---

2. Построение модели

Модель — это класс, наследующий `nn.Module`, содержащий:

- Слои (fully-connected): `fc1`, `fc2`
- Forward-функцию: описывает, как данные проходят через сеть

```
class SimpleNet(nn.Module):
    def __init__(self, activation='relu'):
        super(SimpleNet, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 10)
        self.activation = activation

    def forward(self, x):
        x = x.view(-1, 28*28)
        if self.activation == 'relu':
            x = F.relu(self.fc1(x))
        elif self.activation == 'sigmoid':
            x = torch.sigmoid(self.fc1(x))
        elif self.activation == 'tanh':
            x = torch.tanh(self.fc1(x))
        return self.fc2(x)
```

---

3. Активационные функции

Активационная функция — важная часть сети, она решает, передавать ли сигнал дальше.

Название	Особенности	Применение
ReLU	Быстрая, работает лучше всего в целом	по умолчанию
Sigmoid	Сжимает значение от 0 до 1	устарела, но важна для понимания
Tanh	Сжимает от -1 до 1, сглаживает	альтернатива sigmoid

ReLU чаще всего используется, потому что не вызывает затухающего градиента (в отличие от сигмоида).

---

4. Функция потерь (Loss Function)

Loss — это метрика, показывающая **насколько сильно ошибается модель**. Мы использовали:

```
criterion = nn.CrossEntropyLoss()
```

Она используется для задач **многоклассовой классификации**. Чем меньше значение loss, тем точнее предсказание модели.

---

5. Оптимизаторы

Оптимизатор обновляет веса модели для минимизации потерь. Мы использовали:

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Название	Особенности
SGD	простой, но медленный
Adam	адаптивный, быстрый, почти всегда хорош
RMSprop	похож на Adam, но чаще для RNN

---

6. Обучение модели

Модель проходит данные по эпохам (итерациям по всему датасету). На каждой эпохе:

- Делается прямой проход (forward)
- Вычисляется ошибка (loss)
- Делается обратный проход (backpropagation)
- Обновляются веса

```
for epoch in range(1, epochs + 1):
    for X, y in loader:
        optimizer.zero_grad()
        out = model(X)
        loss = criterion(out, y)
        loss.backward()
        optimizer.step()
```

Мы выводили loss и accuracy на **особых эпохах**: 1, 2, 3, 10, 20... 100.

---

7. Анализ результатов

- **Loss**: Если сильно падает, модель учится.

- **Accuracy**: Показывает процент правильных предсказаний.
- Если loss стал почти 0, а accuracy почти 100%, возможен **overfitting** (см. ниже).

---

8. Графики Accuracy и Loss

```
plt.plot(epochs, loss_hist)
plt.plot(epochs, [a * 100 for a in acc_hist])
```

Ты увидишь, как loss убывает, а accuracy растёт. Если на графике появляются "скачки" или выбросы — это могут быть **аномалии**, вызванные случайной ошибкой в обучении или переобучением.

---

9. Переобучение (overfitting)

Что это: модель идеально выучила тренировочные данные, но плохо работает на новых.

Признаки:

- Loss падает, accuracy высокая на тренировке
- Но на тесте accuracy резко хуже

Решения:

- Добавить Dropout
- Уменьшить количество эпох
- Использовать больше данных
- Использовать регуляризацию (например, weight decay)

---

10. Экспорт модели

Чтобы сохранить модель после обучения:

```
torch.save(model.state_dict(), 'model.pth')
```

Чтобы загрузить:

```
model.load_state_dict(torch.load('model.pth'))
model.eval()
```

🔸 Что ещё можно добавить

- Dropout слои для борьбы с переобучением
- Softmax на выходе для получения вероятностей
- Разделение обучения на train/test и валидацию
- Вывод confusion matrix

💙Заключение

Ты теперь умеешь:

- Загружать данные
- Строить и обучать нейросеть
- Использовать активации и оптимайзеры
- Анализировать результаты

И самое главное — ты **разбираешься, что именно происходит на каждом шаге!**