Train, test, etc

Тема	Содержание
1. Подготовка данных	Данные могут быть практически любыми, но для начала мы собираемся создать простую прямую линию.
2. Построение модели	Здесь мы создадим модель для изучения закономерностей в данных, мы также выберем **функцию потерь**, **оптимизатор** и создайте **цикл обучения**.
3. Подгонка модели к данным (обучение)	У нас есть данные и модель, теперь давайте позволим модели (попытаться) найти закономерности в (**обучении**) данные.
4. Прогнозирование и оценка модели (вывод).	Наша модель обнаружила закономерности в данных, давайте сравним ее результаты с фактическими (**тестированием**) данными. .
5. Сохранение и загрузка модели.	Возможно, вы захотите использовать свою модель в другом месте или вернуться к ней позже, здесь мы об этом расскажем.
6. Собираем все вместе	Давайте возьмем все вышеперечисленное и объединим.

Выборка

Расколоть	Цель	Общий объем данных	Как часто он используется?
Обучающий набор	Модель учится на этих данных (например, на материалах курса, которые вы изучаете в течение семестра).	~60-80%	Всегда
Набор проверки	Модель настраивается на основе этих данных (например, практический экзамен, который вы сдаете перед выпускным экзаменом).	~10-20%	Часто, но не всегда
Тестовый набор	Модель оценивается на основе этих данных, чтобы проверить, чему она научилась (например, выпускной экзамен, который вы сдаете в конце семестра).	~10-20%	Всегда

Создание модели

```
class LinearRegressionModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.weights = nn.Parameter(torch.randn(1, dtype=torch.float
    self.bias = nn.Parameter(torch.randn(1, dtype=torch.float),
  def forward(self, x:torch.Tensor):
    return self.weights * x + self.bias
```

Модуль PyTorch	Что оно делает?
`torch.nn`	Содержит все строительные блоки для вычислительных графов (по сути, серии вычислений, выполняемых определенным образом).
`torch.nn.Parameter`	Сохраняет тензоры, которые можно использовать с `nn.Module`. Если `requires_grad=True` градиенты (используемые для обновления параметров модели посредством **градиентного спуска**) рассчитываются автоматически, это часто называют " ;автоград".
`torch.nn.Module`	Базовый класс для всех модулей нейронных сетей, все строительные блоки нейронных сетей являются подклассами. Если вы создаете нейронную сеть в PyTorch, ваши модели должны быть подклассом `nn.Module`. Требуется реализация метода `forward()`.
`torch.optim`	Содержит различные алгоритмы оптимизации (они сообщают параметрам модели, хранящимся в `nn.Parameter`, как лучше всего их изменить, чтобы улучшить градиентный спуск и, в свою очередь, уменьшить потери).
`def forward()`	Всем подклассам `nn.Module` требуется метод `forward()`, который определяет вычисления, которые будут выполняться с данными, передаваемыми в конкретный =3> (например, формула линейной регрессии выше). `nn.Module`