



> Конспект > 1 урок > Обзор Deep Learning

Содержание

[Содержание](#)

[Что такое DL?](#)

[Чем DL похож на ML?](#)

[Чем отличается ML от DL?](#)

[Устройство нейросетей](#)

[Объединение нейронов](#)

[Как учить нейросети: градиентный спуск](#)

[Как учить нейросети: backpropagation](#)

[Общая архитектура компьютера и роль видеокарт](#)

[Преимущества видеокарт](#)

[Тензоры и их роль в DL](#)

[Методы ускорения вычислений](#)

[Батчи](#)

[Использование чисел с плавающей точкой меньшей точности](#)

[Практика](#)

[Создание тензоров](#)

[Операции с тензорами](#)

[Градиенты в PyTorch*](#)

[Функции потерь](#)

[Слои нейросети](#)

[PyTorch* и GPU](#)

[Важные моменты](#)

[Резюме:](#)

[Дополнительные материалы](#)

Что такое DL?

Глубокое обучение (Deep Learning) — это область машинного обучения, сфокусированная на изучении искусственных нейронных сетей. Её целью является понимание принципов работы нейронных сетей и разработка методов их создания.

Области глубокого обучения и их примеры:

1. Компьютерное зрение (Computer Vision):

- **Определение объектов на изображениях и видео.** Например, распознавание лиц, транспортных средств на дороге, определение животных и т.д. Примеры также включают обнаружение опасных заболеваний по медицинским изображениям, мониторинг аварий на производстве через анализ видео.

2. Обработка естественного языка (Natural Language Processing, NLP):

- **Анализ и генерация текста.** Выявление сущностей в текстовых данных, классификация текстов, создание текста, ответы на вопросы. Например, выявление фейковых новостей, прогнозирование заболеваний по медицинским отчетам, автоматизация анализа текстовых данных.

3. Обработка аудио (Audio Processing):

- **Биометрия и голосовые ассистенты.** Идентификация личности по голосу, разработка голосовых помощников, распознавание и синтез речи.

4. Разработка эффективных алгоритмов глубокого обучения

- Эта область включает в себя разработку и оптимизацию алгоритмов глубокого обучения, чтобы они эффективно использовали ресурсы видеокарт для обработки данных.

Чем DL похож на ML?

Пайплайн работы в DL очень похож на ML



В областях машинного и глубокого обучения (DL) встречаются схожие проблемы,

1. **Переобучение.** Как в ML, так и в DL, модели могут столкнуться с переобучением, когда они слишком точно адаптируются к данным обучения, теряя способность к обобщению. Решение этой проблемы включает сбор большего объема данных, уменьшение количества параметров модели, или применение методов регуляризации.
2. **Недообучение.** Эта проблема возникает, когда модель недостаточно хорошо адаптируется к обучающим данным. Решениями могут быть продолжение обучения, более внимательный мониторинг графиков обучения, и, возможно, увеличение сложности модели.
3. **Отсутствие универсальных решений.** Выбор подходящей модели для конкретной задачи является сложной задачей как в ML, так и в DL. Важно оставаться в курсе последних исследований и популярных решений в сообществе, чтобы найти наиболее подходящие подходы.
4. **Неоптимальные гиперпараметры, утечки данных и грязные данные.** Эти проблемы требуют тщательной проверки разбиения данных на train/test, аккуратного перебора гиперпараметров и очистки данных от выбросов и аномалий. Важно внимательно изучать данные и результаты моделирования, чтобы минимизировать влияние этих проблем.

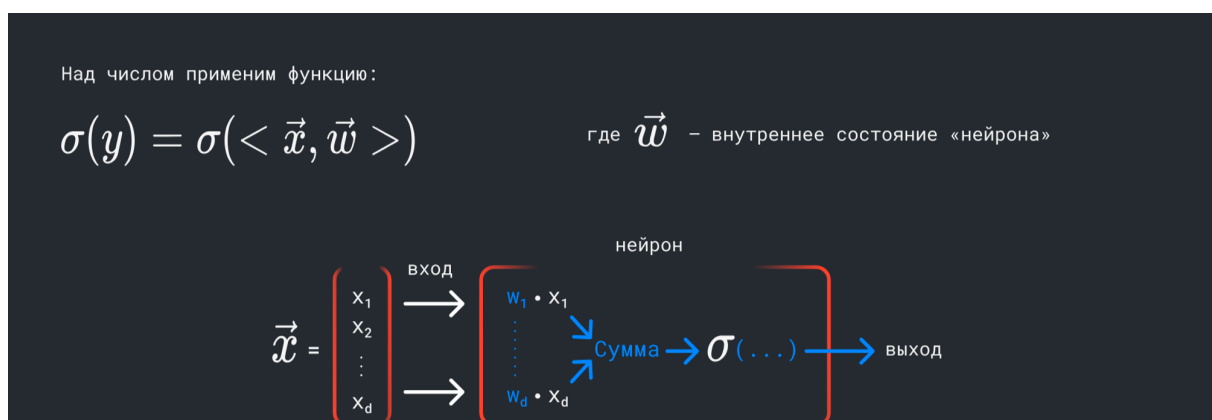
Чем отличается ML от DL?

Чем отличается ML от DL?

1. **Сложность и разнообразие моделей.** В DL используются более сложные и разнообразные модели по сравнению с традиционным ML. Это позволяет DL моделям изучать более сложные закономерности в данных.

2. **Создание контента.** Благодаря своей сложности, модели глубокого обучения способны создавать новый контент, например, видео, что выходит за рамки возможностей большинства классических ML моделей.
3. **Количество параметров.** Модели DL обладают значительно большим количеством параметров по сравнению с моделями классического ML, что требует больших объемов данных для обучения, чтобы каждый параметр мог быть адекватно скорректирован.
4. **Требования к вычислительным ресурсам.** Более сложные модели DL требуют более мощного вычислительного оборудования, что делает их использование затратным и не всегда доступным для всех компаний.
5. **Автоматическое выделение признаков.** В отличие от традиционного ML, где разработчики часто вручную создают дополнительные признаки из исходных данных, модели DL способны автоматически выявлять необходимые признаки для эффективного обучения, уменьшая тем самым необходимость вручную создавать признаки.

Устройство нейросетей



На изображении представлена схема работы нейрона в нейронной сети, объясняющая основы их функционирования. Входной вектор \vec{x} подаётся на вход нейрона, где каждый элемент этого вектора умножается на соответствующий вес w_i из вектора весов \vec{w} . Эти произведения суммируются, получаясь скалярное произведение векторов \vec{x} и \vec{w} .

Далее результат скалярного произведения подаётся в функцию активации σ , которая преобразует линейный вход в нелинейный выход. Функция активации добавляет нелинейность в модель, позволяя нейронной сети аппроксимировать более сложные функции.

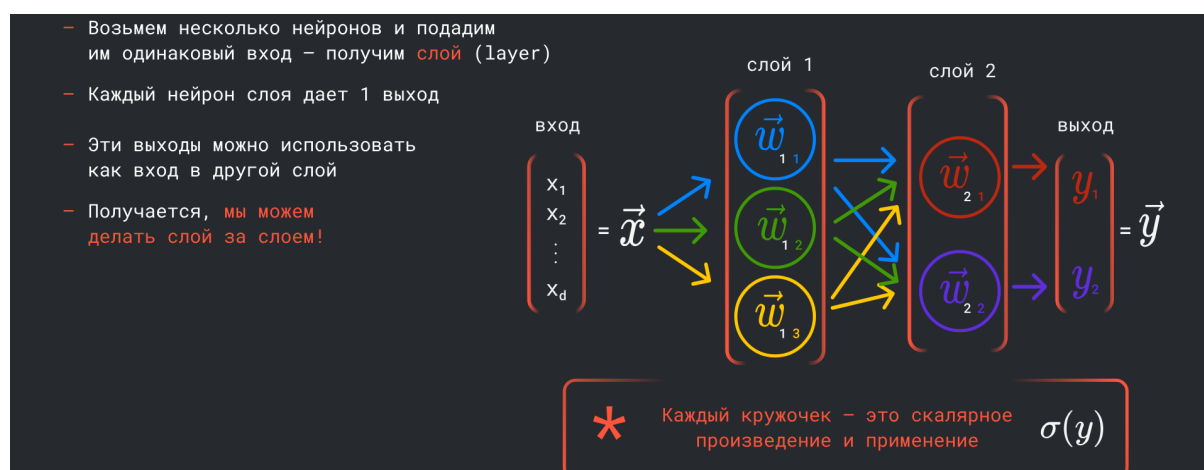
Общая формула выхода нейрона выглядит следующим образом:

$\sigma(y) = \sigma(\langle \vec{x}, \vec{w} \rangle)$, где $\langle \vec{x}, \vec{w} \rangle$ обозначает скалярное произведение входного вектора и вектора весов, а σ — функцию активации.

Объединение нейронов

Одним из фундаментальных принципов глубокого обучения является возможность объединения нейронов в слои для обработки входных данных. В такой структуре каждый нейрон в слое принимает одинаковый входной сигнал, обрабатывает его, используя уникальные для себя веса, и производит на выходе определенный результат.

Когда несколько нейронов работают с одним и тем же входом, они генерируют на выходе набор чисел, или вектор, который может быть использован в качестве входных данных для следующего слоя нейронов.



Такая модель работы позволяет выходу одного слоя стать входом для другого, создавая тем самым многослойную нейронную сеть. Это ключевой аспект, позволяющий моделировать сложные зависимости в обрабатываемых данных.

Благодаря этому глубокое обучение становится мощным инструментом для решения разнообразных задач машинного обучения, обеспечивая возможность построения сложных и эффективных моделей.

Как учить нейросети: градиентный спуск

Подбирать веса модели можно так же, как и в обычных моделях машинного обучения — градиентным спуском

1. **Начальные условия.** У нас есть нейросеть с несколькими слоями, которые после обработки входных данных выдают числовые значения (например, вектор или одно число), которые могут использоваться для прогнозирования.
2. **Функция потерь.** Для оценки точности прогноза нейросети используется функция потерь $L(\mathbf{x}, \theta)$, которая является скалярной и зависит от выходных данных сети и реальных значений. Например, если нейросеть предсказывает цену хлеба, функция потерь может быть среднеквадратичным отклонением между предсказанной и истинной ценой.
3. **Градиент функции потерь.** Вычисляется градиент функции потерь $\nabla_{\theta} L$ по параметрам модели, что позволяет определить направление для корректировки весов для уменьшения ошибки.
4. **Обновление параметров.** Параметры модели обновляются по формуле $\theta_{\text{new}} = \theta_{\text{old}} - \eta \nabla_{\theta} L$ где η — это скорость обучения, которая определяет размер шага при корректировке весов.
5. **Итерационный процесс.** Этот процесс повторяется множество раз, где на каждой итерации параметры модели корректируются для минимизации функции потерь. Со временем это должно приводить к улучшению точности предсказаний нейросети.

Как учить нейросети: backpropagation

Backpropagation является ключевым методом обучения нейронных сетей в глубоком обучении. Отличие глубокого обучения от классических методов машинного обучения в значительной мере связано именно с этим алгоритмом.

Основная задача, которую решает backpropagation, заключается в вычислении градиентов функции потерь по весам сети, что необходимо для их оптимизации в процессе обучения.

Пример:
хотим $\frac{\partial L}{\partial w_1}$

Chain rule:

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(t)}{\partial t} \frac{\partial g(x)}{\partial x}$$

Можем идти справа налево и обчислять все градиенты по пути

Рассмотрим нейронную сеть с двумя слоями. Первый слой принимает входные данные x и преобразует их, получая промежуточные выходы t_1 и t_2 . Эти промежуточные значения затем передаются во второй слой, который выдает окончательное предсказание — например, цену хлеба. Для оценки качества предсказания используются потери, часто в виде среднеквадратичного отклонения.

Проблема возникает, когда нам нужно вычислить, как изменение весов первого слоя (например, w_1) влияет на функцию потерь. Прямой подсчет этой производной может быть сложным из-за многоступенчатой зависимости потерь от w_1 : необходимо учесть умножение на w_1 , применение нелинейной функции, последующее умножение, сложение и возведение в квадрат.

Для решения этой задачи применяется правило цепочки

Правило цепочки. Для функций f и g и переменной x , правило цепочки гласит, что

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(t)}{\partial t} \cdot \frac{\partial g(x)}{\partial x}$$

Используя это правило, мы можем последовательно вычислить производные функции потерь по всем параметрам, начиная с конечного слоя и двигаясь обратно к начальным слоям. Например, сначала вычисляется производная потерь по выходу второго слоя (который зависит от w_3), а затем — по параметрам первого слоя (w_1 и w_2).

Процесс вычисления градиентов начинается с выходного слоя и продвигается в обратном направлении, отсюда и название "backpropagation" — обратное распространение ошибки. На каждом этапе градиенты для каждого параметра пересчитываются, позволяя обновлять веса сети таким образом, чтобы минимизировать функцию потерь.

Важно понимать, что во время прямого распространения (когда нейросеть делает предсказание) информация движется от входных данных к выходным. Во время backpropagation происходит обратный процесс — информация о градиентах распространяется от выходов к входам. Это позволяет нам адаптировать веса нейронной сети, чтобы с каждой итерацией повышать точность предсказаний.

Общая архитектура компьютера и роль видеокарт

Компьютер состоит из нескольких ключевых компонентов:

1. процессора,
2. видеокарты
3. оперативной памяти.

В процессе работы эти компоненты тесно взаимодействуют между собой. Процессор и видеокарта выполняют вычисления, в то время как оперативная память хранит промежуточные данные. Особенностью видеокарт является наличие собственного процессора и оперативной памяти, что позволяет эффективно обрабатывать данные без нагрузки на основной процессор компьютера.

Преимущества видеокарт

Видеокарты предназначены для обработки больших массивов данных и специализированы на выполнении параллельных вычислений. Благодаря большому количеству ядер видеокарты могут одновременно обрабатывать множество операций над тензорами (матрицами и векторами), что делает их идеальными для задач глубокого обучения.

На начальных этапах разработки моделей предпочтение отдают процессорам из-за их способности к быстрому обнаружению и исправлению ошибок в коде. Тем не менее, для обучения серьезных нейросетей и выполнения интенсивных вычислений используют видеокарты из-за их высокой производительности.

Тензоры и их роль в DL

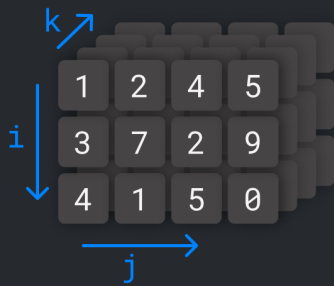
Тензоры — это обобщение матриц на многомерный случай, ключевой элемент в глубинном обучении. Они представляют собой многомерные массивы данных, которые могут хранить в себе всевозможные типы информации, используемые в машинном обучении. Работа с тензорами позволяет эффективно организовать и обрабатывать большие объемы данных, что необходимо для обучения нейросетей.

Тензор — это многомерная таблица с числами.

Весь DL построен на тензорах: в них лежат данные и параметры модели.

$$\begin{pmatrix} 12 & -1 \\ -5 & 0 \end{pmatrix}$$

Двумерный тензор
(матрица)



Трехмерный тензор

Методы ускорения вычислений

Когда вы обучаете нейросеть на большом количестве данных, они могут не влезать в оперативную память, поэтому мы учимся на кусках данных, в этом случае есть два решения:

Батчи

GPU проще, когда приходят все данные сразу, а не по одному, поэтому придумали **батчи (batch) — упаковку нескольких тензоров в один.**

Батчи позволяют разбивать датасет на меньшие части и обучать модель порциями. Это не только облегчает работу с большими объемами данных, но и позволяет эффективнее использовать вычислительные мощности GPU, поскольку графические процессоры более эффективны при работе с большими объемами данных за один раз.

Например, если раньше тензор имел размерность (3, 1920, 1080), что соответствует изображению с 3 каналами и размером 1920x1080 пикселей, то теперь, используя батчи размером 64, размерность тензора становится (64, 3, 1920, 1080). Это означает, что в одном батче теперь содержится 64 изображения с указанными параметрами, что оптимизирует процесс обучения нейросети.

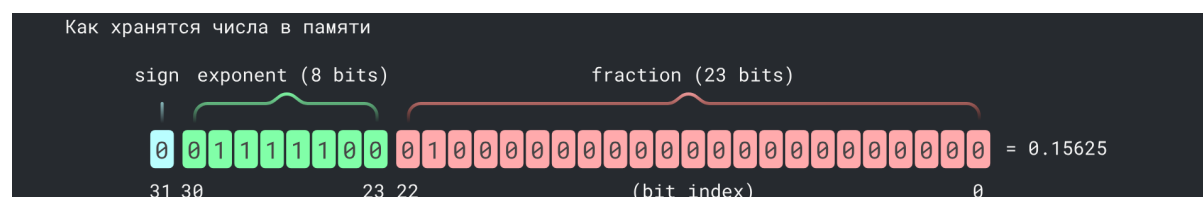
Использование чисел с плавающей точкой меньшей точности

Числа с плавающей точкой в компьютерах хранятся в бинарном формате и разделяются на несколько частей:

- sign
- exponent
- fraction

FP32 и FP16 отличаются количеством бит, выделенных на эти компоненты. FP32 обеспечивает большую точность и диапазон за счет использования 32 бит на число, тогда как FP16 сокращает это количество до 16 бит, что позволяет сэкономить память и ускорить вычисления, особенно на архитектурах, оптимизированных под операции с таким типом данных, как современные видеокарты.

На изображении показано сколько бит выделяется в FP32 и FP16.



Для совмещения преимуществ высокой точности и эффективного использования памяти и вычислительной мощности применяется **техника Mixed Precision**. Она заключается в использовании FP16 для большинства операций, что позволяет уменьшить время выполнения операций и потребление памяти.

Одновременно, в критически важных моментах, где необходима высокая точность (например, при расчете градиентов), временно используются числа в формате FP32. Это позволяет уменьшить в 1.5 - 2 раза объем памяти.

Практика

Создание тензоров

Тензоры в PyTorch* — это многомерные массивы, аналогичные numpy массивам, но с возможностью вычисления на GPU.

```
import torch

# Создание тензора с произвольными данными
t = torch.Tensor(2, 3, 4)

# Тензор заполненный нулями
zeros = torch.zeros((5, 3))

# Тензор заполненный единицами
ones = torch.ones((2, 3, 4))

# Тензор из нормального распределения
randn = torch.randn((2, 3, 2, 4))
```

Операции с тензорами

PyTorch* поддерживает множество операций над тензорами, включая арифметические, матричные и логические операций над тензорами, включая арифметические, матричные и логические операции.

```
# Сложение
sum = 2 * torch.ones((2, 3)) + 3

# Поэлементное умножение
mul = torch.eye(3) * torch.tensor([[1., 2., 3.], [4., 5., 6.],
[7., 8., 9.]])
```

```
# Матричное умножение
a = torch.tensor([[3, 5]])
b = torch.tensor([[2, 4], [4, 2]])
matmul = a @ b
```

Градиенты в PyTorch*

PyTorch* автоматически вычисляет градиенты, что делает его идеальным для обучения нейросетей.

```
w = torch.tensor([[1, 1], [2, 2]], dtype=float, requires_grad=True)
x = torch.tensor([[5], [3]], dtype=float)
y = w @ x
y_scalar = y.sum()
y_scalar.backward()
print(w.grad)
```

Функции потерь

PyTorch* предоставляет множество встроенных функций потерь, например, MSE (среднеквадратичная ошибка) и BCE (бинарная кросс-энтропия).

```
loss = torch.nn.functional.mse_loss(torch.tensor([2.0]), torch.tensor([5.0]))
```

Слои нейросети

В PyTorch* уже реализованы многие стандартные слои, такие как полносвязные слои, сверточные слои и слои активации.

```
import torch.nn as nn

# Полносвязный слой
linear = nn.Linear(2, 1)
# Слой активации
sigmoid = nn.Sigmoid()
# Сборка в последовательную модель
sequential = nn.Sequential(nn.Linear(2, 1), nn.Sigmoid())
```

PyTorch* и GPU

PyTorch* позволяет легко перенести вычисления на GPU для ускорения обучения.

```
# Перенос тензора на GPU
tensor_gpu = tensor.to("cuda")

# Обратно на CPU
tensor_cpu = tensor_gpu.to("cpu")
```

Важные моменты

- PyTorch* удобен для работы как на CPU, так и на GPU.
- Важно помнить, что все операции в рамках одной операции должны выполняться на одном и том же устройстве.
- PyTorch* предоставляет удобные инструменты для автоматического вычисления градиентов и построения нейросетей.
- Встроенные функции потерь и слои упрощают создание и обучение моделей глубокого обучения.

Резюме:

- Узнали об отличиях и сходствах классического ML и DL
- Узнали о способах обучения нейронных сетей и об оптимизации этих методов
- Научились работе с тензорами в PyTorch*

Дополнительные материалы

<https://pytorch.org/tutorials/>



* Pytorch является продуктом финансируемый компанией Meta. Компания Meta Platforms Inc. по решению Тверского районного суда города Москвы от 21.03.2022 признана экстремистской организацией, ее деятельность на территории России запрещена