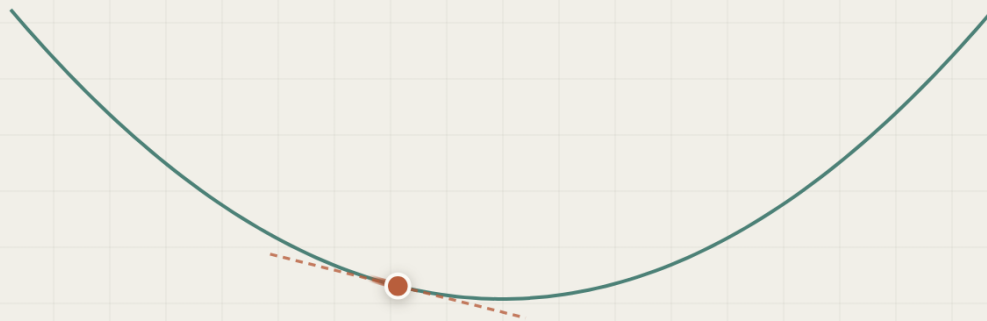


Использование производной при создании искусственного

Как понятие, придуманное в XVII веке для касательных, сегодня обучает нейросети — через градиентный спуск и обратное распространение ошибки.



- ◆ ИИ не программируют правилами «если — то», его **обучают на данных**.
- ◆ У модели есть параметры — **веса** $w_1, w_2, \dots, w_n$, их миллионы и миллиарды.
- ◆ Качество измеряет **функция ошибки** $L(w)$: насколько ответ модели расходится с правильным.
- ◆ Обучение = подобрать веса, при которых $L(w)$ **минимальна**.

$$\min_w L(w) \implies \text{нужна } \frac{\partial L}{\partial w}$$

А минимум ищут там, где **производная** подсказывает направление вниз. Поэтому производная — в самом сердце ИИ.

04

- 04

04



04

Функция потерь

- ◆ Пример — **среднеквадратичная ошибка** (MSE) для одного примера.
- ◆ $\hat{y}$ — ответ модели, y — правильный ответ.
- ◆ Множитель $\frac{1}{2}$ — чтобы производная была «чистой»: $dL/d\hat{y} = \hat{y} - y$.
- ◆ По всей выборке берут **среднее** ошибок. Чем меньше L — тем точнее модель.

$$L = \frac{1}{2} (\hat{y} - y)^2$$

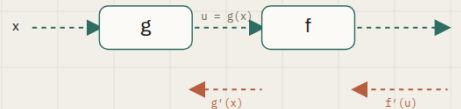
$$\frac{dL}{d\hat{y}} = \hat{y} - y$$

ошибка прямо задаёт направление правки

Цепное правило

$$y = f(g(x)) \Rightarrow \frac{dy}{dx} = f'(g(x)) \cdot g'(x)$$

- ◆ Нейросеть — это **композиция** функций: слой за слоем.
- ◆ Ошибка зависит от веса в начале **через всю цепочку** слоёв.
- ◆ Цепное правило позволяет «протащить» производную сквозь все слои.
- ◆ Без него обучить глубокую сеть **невозможно**.



$$\frac{dy}{dx} = \underbrace{f'(u)}_{\text{внешняя}} \cdot \underbrace{g'(x)}_{\text{внутренняя}}$$

Обратное распространение ошибки

$$\mathbf{1} \quad \delta^L = (a^L - y) \odot \sigma'(z^L)$$

$$\mathbf{2} \quad \delta^l = ((W^{l+1})^\top \delta^{l+1}) \odot \sigma'(z^l)$$

$$\mathbf{3} \quad \frac{\partial L}{\partial W^l} = \delta^l (a^{l-1})^\top$$

$$\boxed{4} \quad \frac{\partial L}{\partial b^l} = \delta^l$$

Это **цепное правило**, применённое от выхода ко входу. В PyTorch / TensorFlow его делает **автодифференцирование** (autograd).



Один шаг обучения ВЖИВУЮ

Нейрон $a = \sigma(w \cdot x)$, цель y . Нажмите «Один шаг» — цепное правило посчитает $\partial L / \partial w$, и вес обновится. «Обучать» — повтор до сходимости.

● СТЕНД · ОДИН ШАГ ОБУЧЕНИЯ НЕЙРОНА

$$\frac{\partial L}{\partial w} = \underbrace{(a - y)}_{-0.354} \cdot \underbrace{a(1 - a)}_{0.229} \cdot \underbrace{x}_{1.00} = -0.0811$$

$$w \leftarrow w - \eta \frac{\partial L}{\partial w} = 0.600 - 0.50 \cdot (-0.0811) = \mathbf{0.641}$$

$x = 1.00 \rightarrow xw \rightarrow z = 0.600 \rightarrow \sigma \rightarrow a = 0.646 \rightarrow L \rightarrow L = 0.0628$

Скорость обучения

$\eta = 0.50$

Цель

$y = 1.00$

Один шаг

Обучать

Сброс

ОШИБКА L · ШАГ 0

<

>

10 / 12 обзор ▢

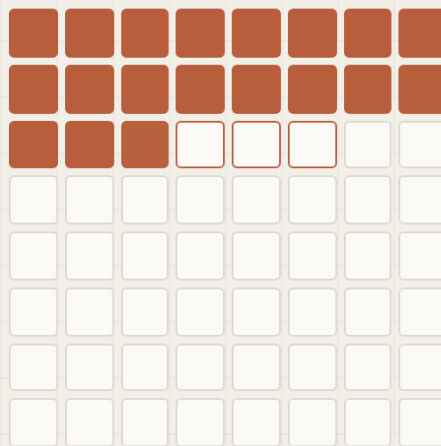
$a = \mathbf{0.646}$

$w = \mathbf{0.600}$

Почему для ИИ нужны видеокарты

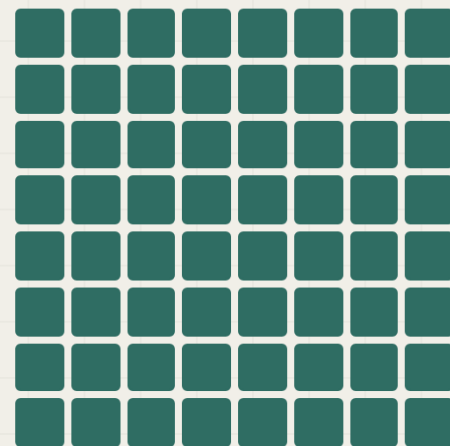
- ◆ В большой модели — **миллиарды весов**, и на каждом шаге нужно столько же производных.
- ◆ Backprop = огромное число операций над **матрицами**.
- ◆ Производные считаются **независимо** → их можно вычислять **параллельно**.
- ◆ **GPU** заточены под параллельное умножение матриц → ускорение в десятки-сотни раз.

CPU - по одной операции



Такт 22 / 64

GPU · целый ряд параллельно

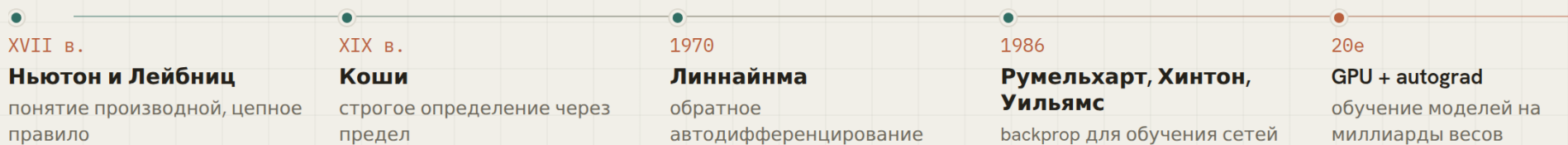


8 ТАКТОВ • ГОТОВО

Одна и та же работа: CPU — по клетке, GPU — целыми рядами сразу.

Производная — фундамент обучения ИИ

- ◆ Обучение = минимизация функции ошибки $L(w)$.
- ◆ Градиент ∇L задаёт направление спуска.
- ◆ Цепное правило + backprop считают все производные эффективно.
- ◆ GPU выполняют это в гигантском масштабе.



Инструмент, придуманный **300 лет назад** для геометрии и механики, сегодня обучает искусственный интеллект.