Введение


Искусственный Интеллект
Машинное Обучение
Нейросети
Глубокое Обучение
сотни других методов обучения

Машинное обучение

Вход

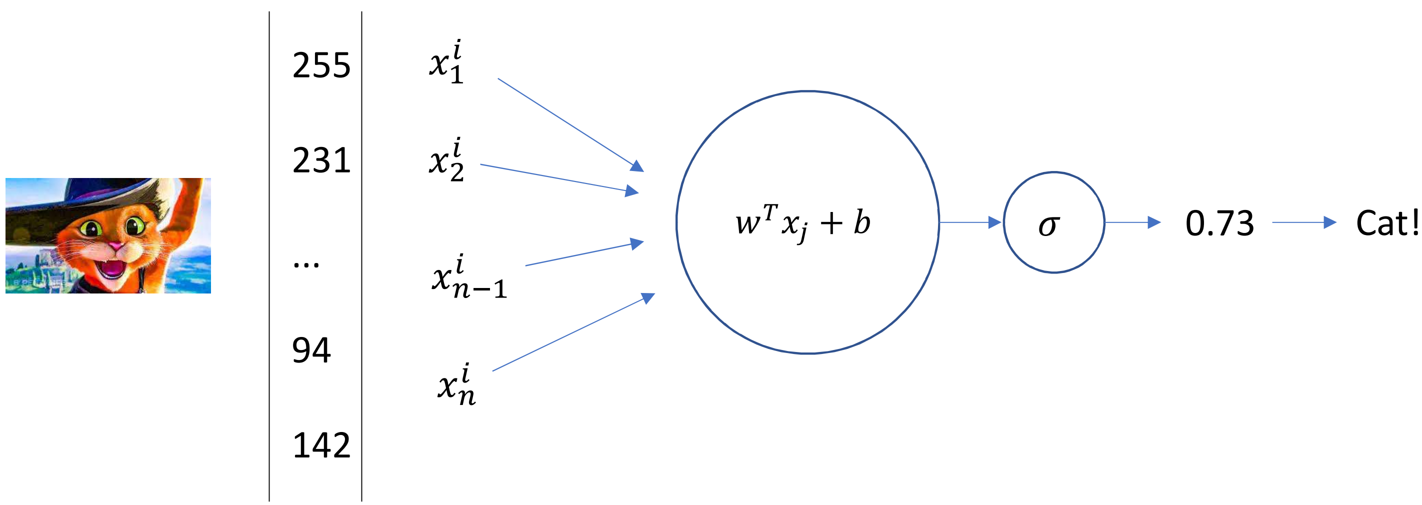
Архитектура
Параметры

“CAT”

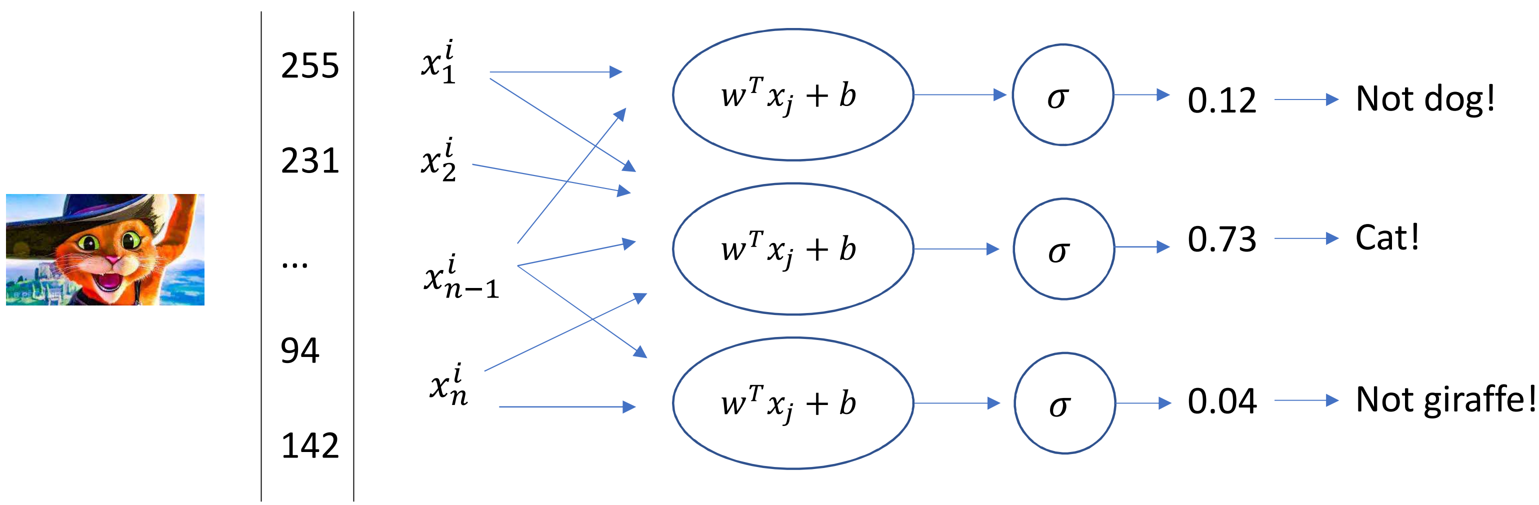
Выход

Что еще нужно? Оптимизатор, функция потерь, метрики, регуляризация

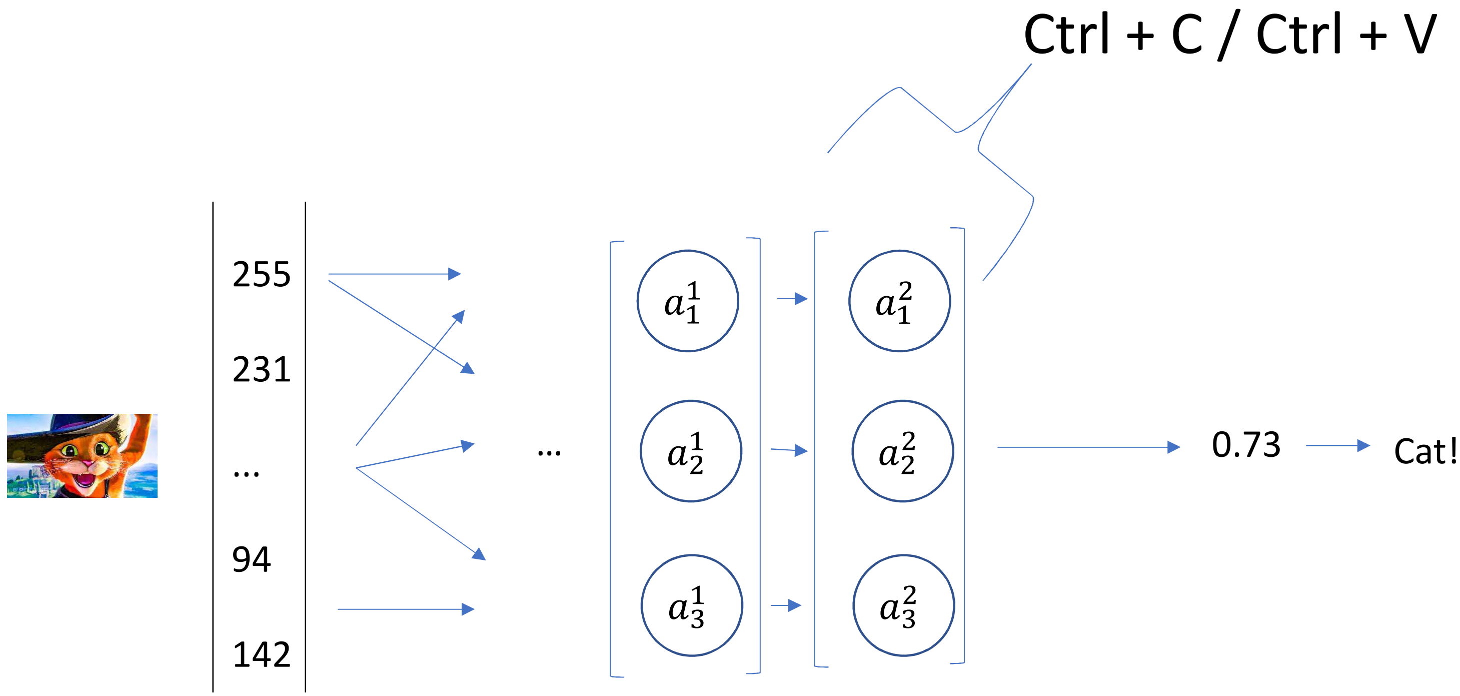
Логистическая регрессия – это тоже нейронная сеть



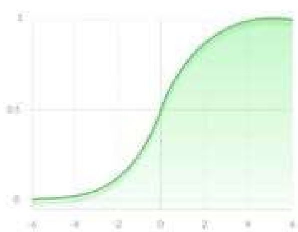
Мультикласс



Многослойная сеть



Функции активации

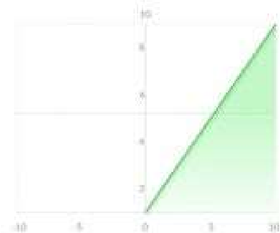


Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

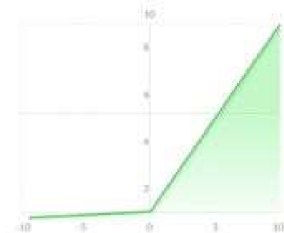
TanH

$$tanh(z) = \frac{2}{1+e^{-2z}} - 1$$



ReLU

$$f(z) = max(0, z)$$



LeakyReLU

$$f(x) = \begin{cases} 0.01x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

Задача обучения

Наша задача - найти функцию хорошо приближающую реальную Зависимость $y(x)$.
Назовем такое решение $\hat{y} : X \to Y$ (эта функция должна быть вычислима на компьютере).

Обычно мы выбираем решение из некоторого параметризованного семейства.

$\mathcal{F} = \{\hat{y}_\theta \mid \theta \in \Theta\}, \Theta -$ *множество параметров.*

Задача обучения (2)

Определим функцию $L(y, \hat{y}(x))$, ее значение показывает насколько сильно наше предсказание отличается от реального значения.

Пример:

Задача предсказания цены дома из предыдущих примеров.

Возможные функции потерь:

$L(y_{\text{true}}, \hat{y}(x)) = (y_{\text{true}} - \hat{y}(x))^2$ --- квадратичная функция потерь

$L(y_{\text{true}}, \hat{y}(x)) = |y_{\text{true}} - \hat{y}(x)|$ --- абсолютная функция потерь

$L(y_{\text{true}}, \hat{y}(x)) = (y_{\text{true}} - \hat{y}(x))^2 + 7$ --- ?

Задача обучения (3)

Эмпирический риск:

Определим эмпирический риск как среднее значение функции потерь на обучающем датасете.
Часто функцию эмпирического риска также называют лоссом.

Обучение:

$$\theta_{\text{best}} = \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{\text{dataset size}} \sum_{i} L(y^i_{\text{true}}, \hat{y}_\theta(x^i))$$

(Это общее математическое определение. Конкретный алгоритм получения лучшего параметра для каждой модели свой.)

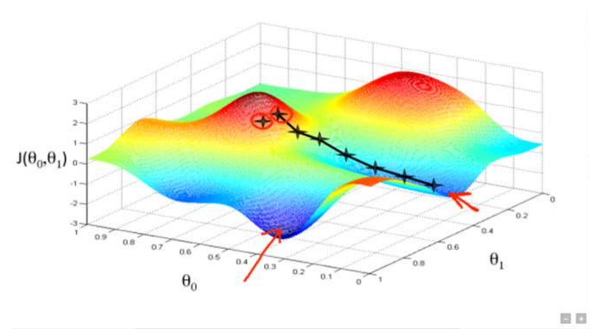
Функция среднеквадратичной ошибки (MSE)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Производная относительно ВЕСА

$$\frac{\partial}{\partial\theta_j}\operatorname{MSE}(\theta) = \frac{2}{m}\sum_{i=1}^{m}(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)})\, x_j^{(i)}$$

Алгоритм градиентного спуска



Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial\theta_j} J(\theta)$$

}

SGD

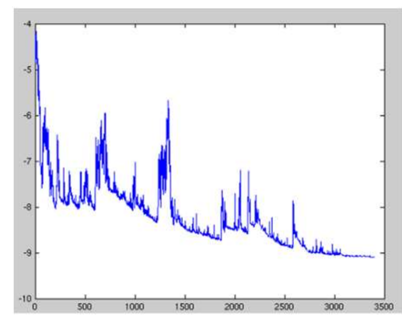
$$\Delta\theta = -\alpha\nabla_{\theta}J(\theta)$$
$$\theta = \theta + \Delta\theta = \theta - \alpha\nabla_{\theta}J(\theta)$$

J - функция потерь, θ - параметры

Может перепрыгнуть в другой локальный минимум

Быстрая сходимость, часто к хорошему минимуму

Сильные флуктуации, для стабилизации используют мини-батчи



Adam

1) Храним затухающую сумму квадратов предыдущих градиентов m
2) Храним затухающее среднее градиентов v

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$ 1й момент

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$ 2й момент

Если инициализировать v и m нулями, то у нас появляется сдвиг в их сторону, особенно на ранних шагах

Чтобы избавиться от этого, оценки корректируются:

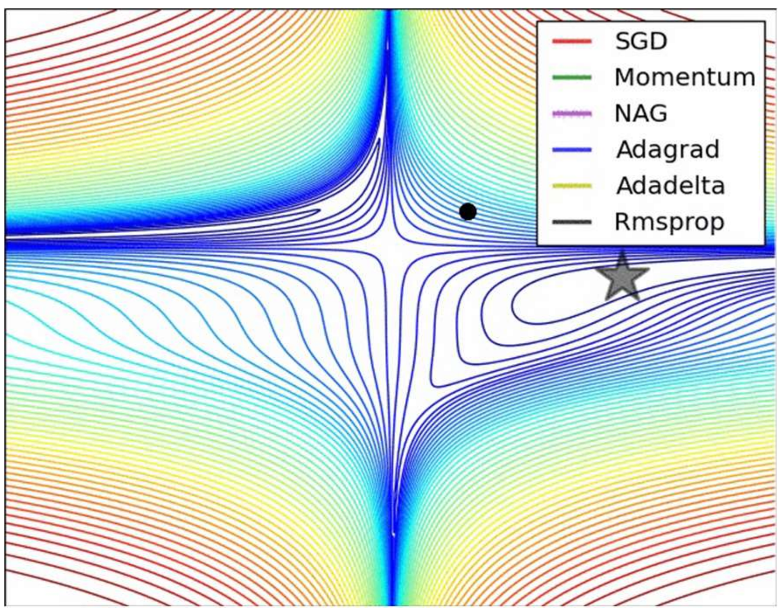
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

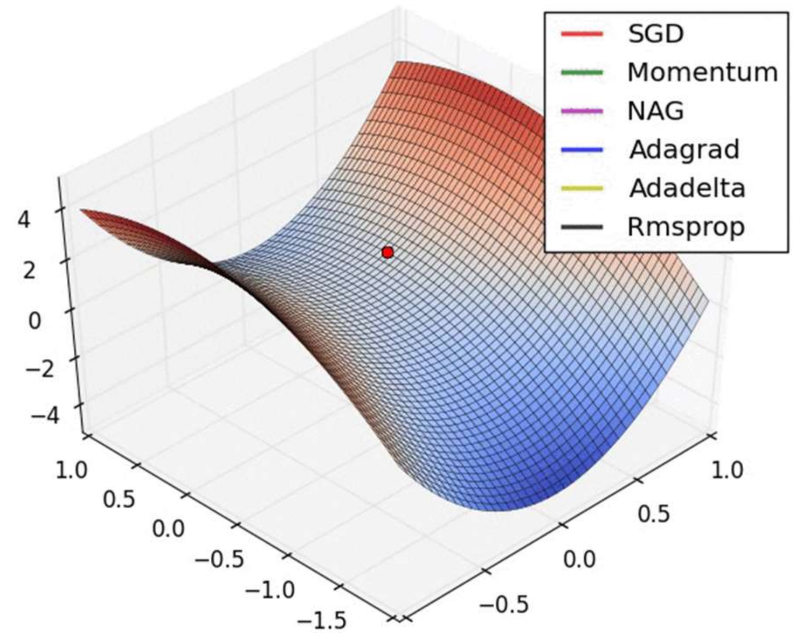
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Итоговое выражения для весов: $$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

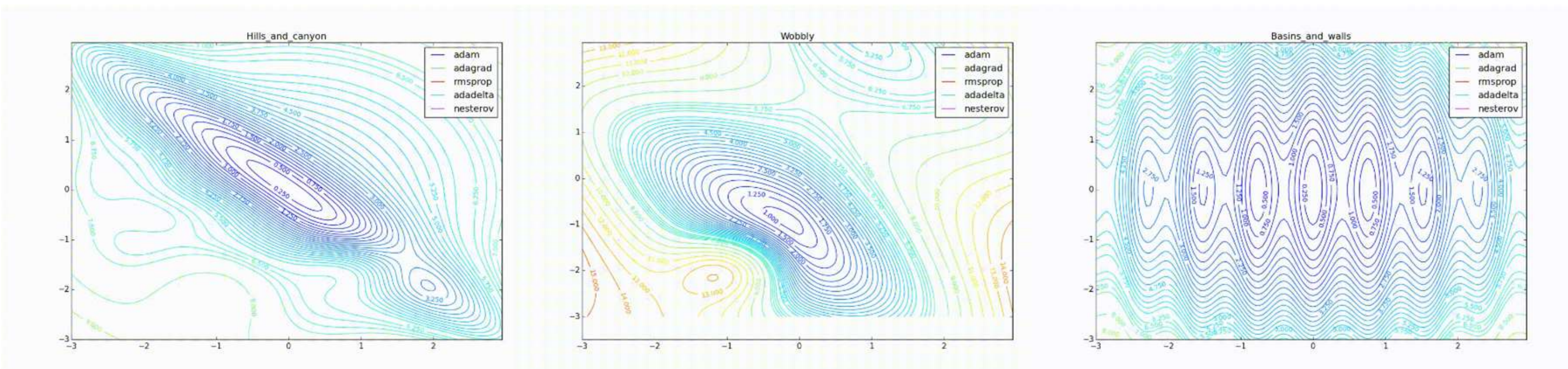
теперь у нас не просто тяжелый мячик, а мячик с трением

Примеры работы алгоритмов





Примеры работы алгоритмов



Источники:

- Материалы http://cs231n.stanford.edu/
- Материалы https://cs230.stanford.edu/
- https://habr.com/ru/post/318970/
- https://www.ruder.io/optimizing-gradient-descent/
- Практики реализации нейронных сетей