Введение в нейронные сети

Лекция 1. Основы НС и Keras.

Структура курса

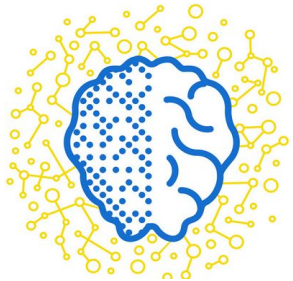
5 лекций и семинаров:

1. Введение в НС и библиотеку Keras
2. Углубление в НС и библиотеку Keras
3. Введение в сверточные *[convolutional]* НС
4. Введение в рекуррентные *[recurrent]* НС
5. Дополнительные темы НС: генеративно-состязательные сети *[Generative Adversarial Networks]*, автоенкодеры *[autoencoders]* & stuff

2-3 домашки:

Coming soon...

Обо мне



tg/linkedin/gmail/slack: waytobehigh

Сегодня мы обсудим...

Базовая теория:

- Искусственные нейроны
- Перцептрон и алгоритм его обучения
- Нейронная сеть *[neural network]*
- Функции активации и зачем нужна нелинейность (sigmod, tanh, relu, leaky_relu?)
- Градиентный спуск и обучение НС методом обратного распространения *[backpropagation]*
- Нейронная сеть vs другие алгоритмы ML. Когда нейронки лучше, а когда - хуже?

Введение в Keras:

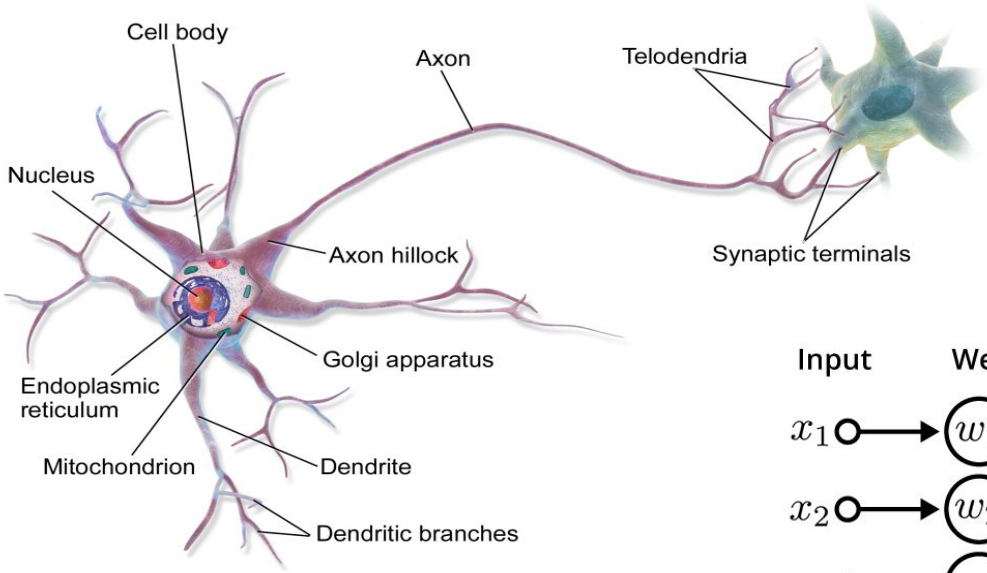
- Знакомство с библиотекой
- Основные строительные блоки - Sequential-модель, базовые слои, SGD-оптимизатор *[SGD-optimizer]*, обучение НС

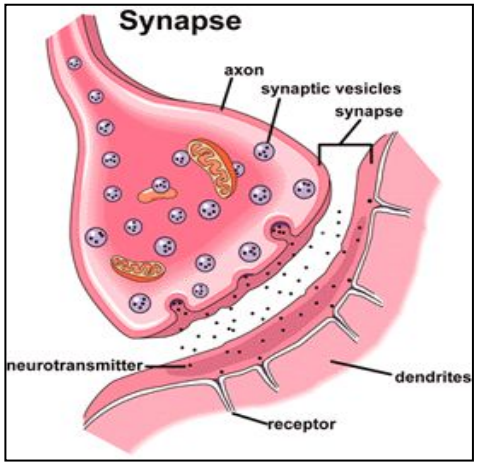
What is Deep Learning?

What is Deep Learning?

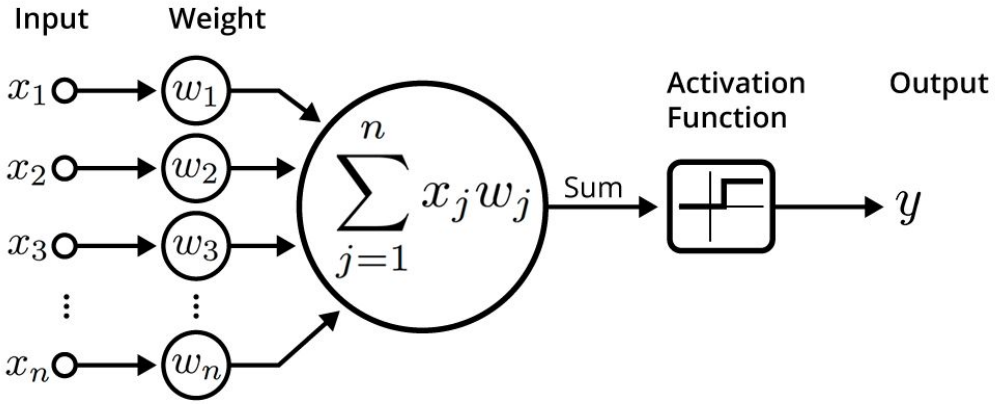
- Construct & train
- Construct from some **blocks** (parameterized modules)
- Train via **gradient-based optimization**

Реальные и искусственные нейроны



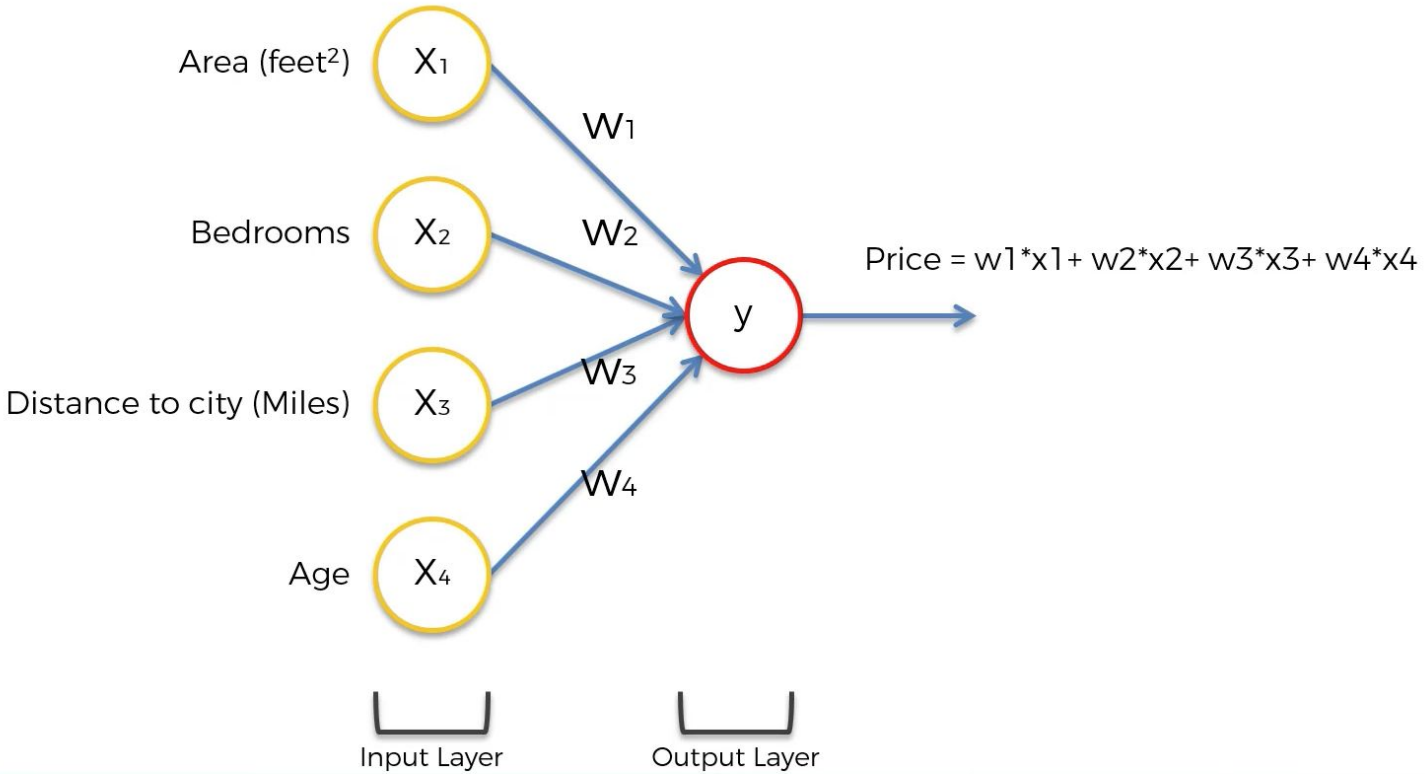


$$y = H(\sum_{j=1}^{n} w_j x_j)$$

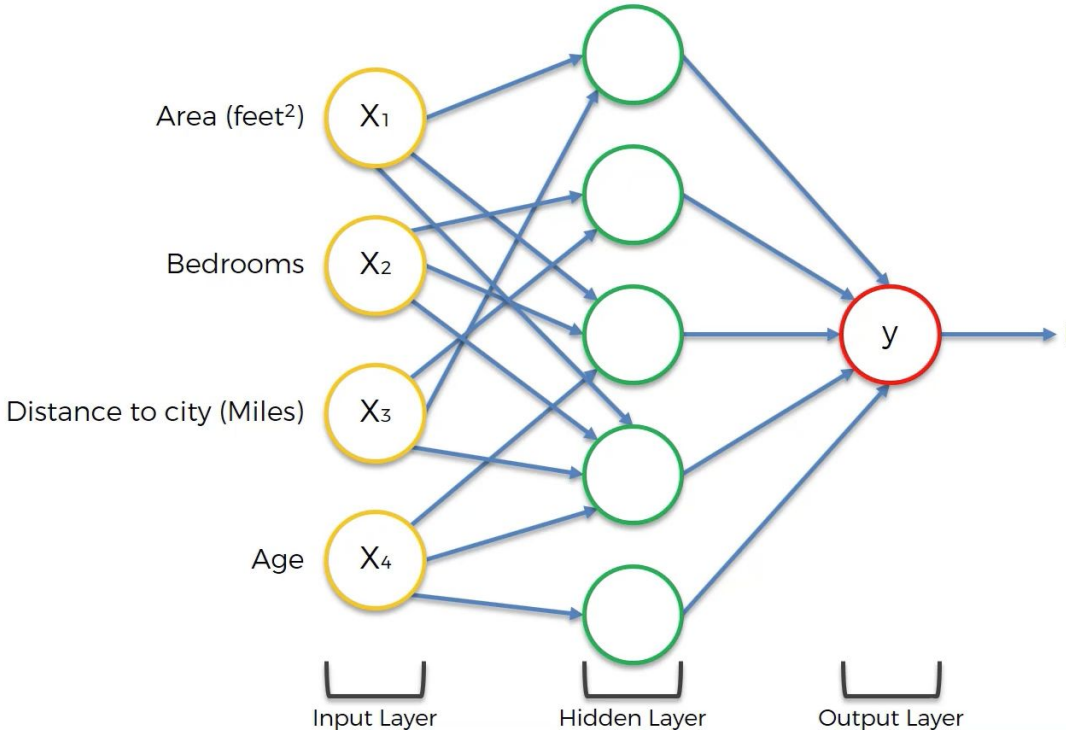


An illustration of an artificial neuron. Source: Becoming Human.

Линейная регрессия как НС



Многослойный персептрон *[multilayer perceptron]*



$$\mathbf{x} = (x_1 \dots x_n), \;\; y_i = f(\sum_j W_{ij} x_j)$$

$$\mathbf{y^1} = f_1(W_1 \mathbf{x})$$

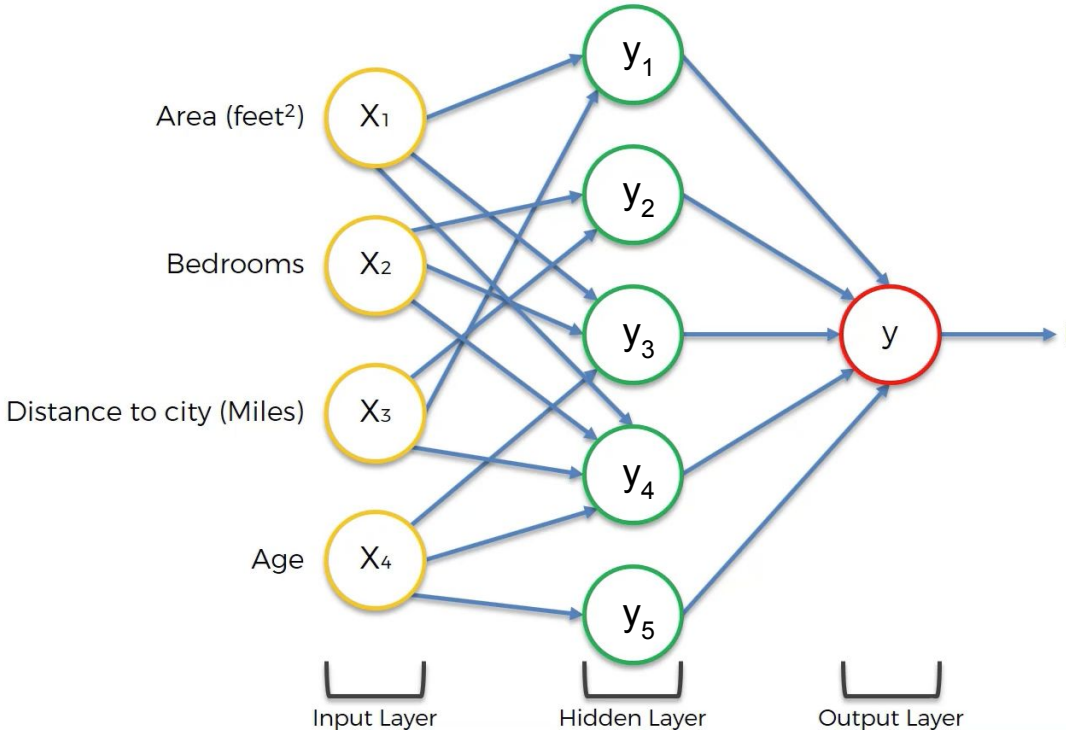
$$\mathbf{y^2} = f_2(W_2 \mathbf{y^1})$$

$$\dots$$

$$\mathbf{y^n} = f_n(W_3 \mathbf{y^{n-1}})$$

f одна и та же для каждого нейрона в слое. Никакого математического смысла, удобнее векторизировать (e.g. on GPU).

Многослойный персептрон *[multilayer perceptron]*



$$\mathbf{x} = (x_1 \dots x_n), \;\; y_i = f(\sum_j W_{ij} x_j)$$

$$\mathbf{y^1} = f_1(W_1 \mathbf{x})$$

$$\mathbf{y^2} = f_2(W_2 \mathbf{y^1})$$

$$\dots$$

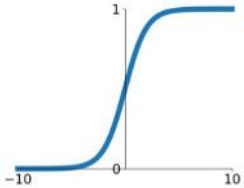
$$\mathbf{y^n} = f_n(W_3 \mathbf{y^{n-1}})$$

f одна и та же для каждого нейрона в слое. Никакого математического смысла, удобнее векторизировать (e.g. on GPU).

Функции активации *[activation functions]*

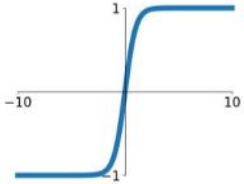
Sigmoid

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



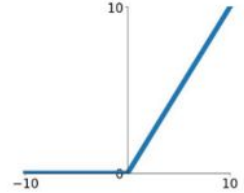
tanh

$$\tanh(x)$$



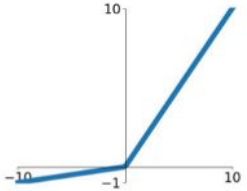
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

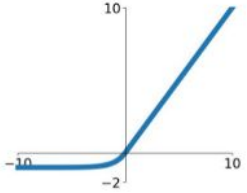


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

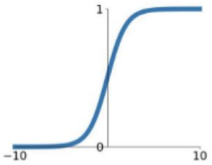


If we remove them, everything collapses to a single matrix multiplication!

Классификация vs. регрессия

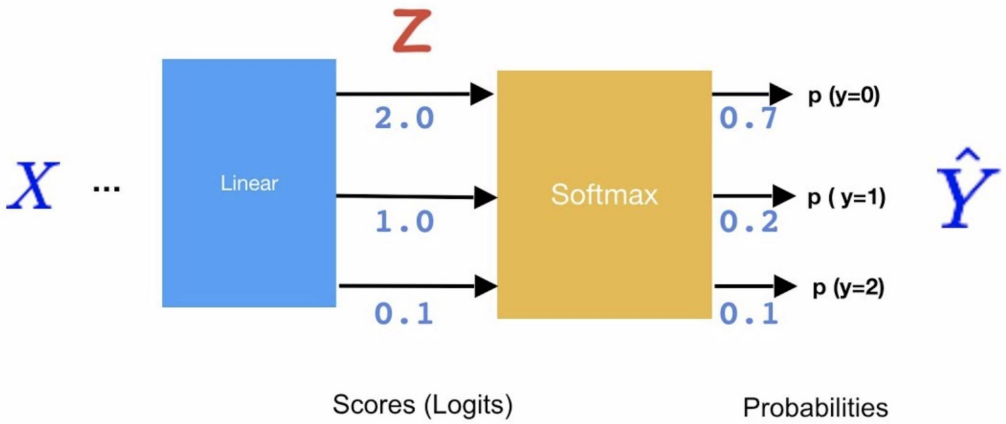
Sigmoid

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



Meet Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



So we've designed a **block**…
What about **training**?

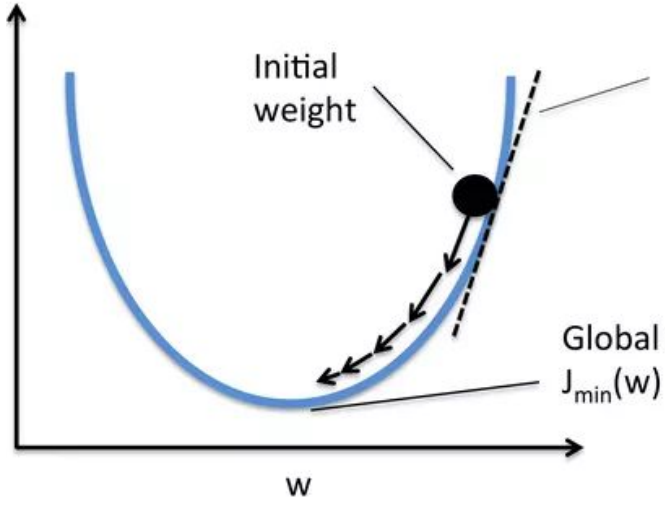
Градиентный спуск *[Gradient descent]*

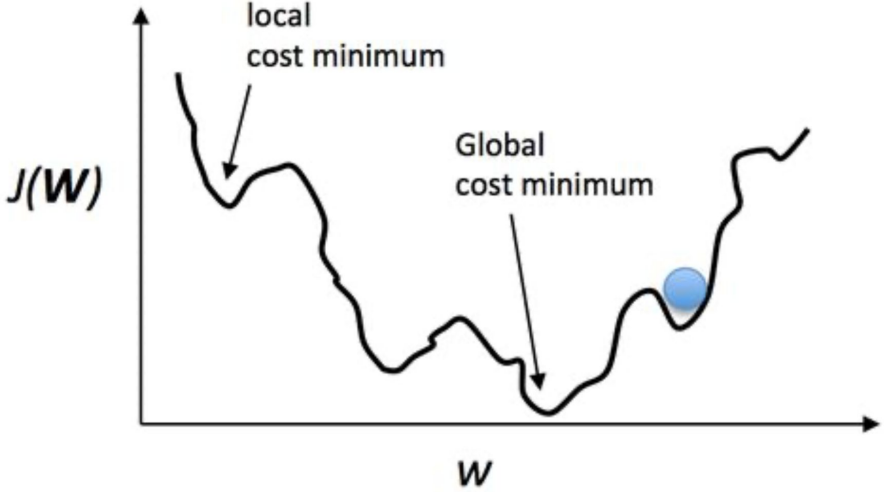
- Минимизируем L(x, y, w), например

$$L(x, y, w) = (f(x, w) - y)^2 \to \min_w$$

f(x, w) -- НС, (x, y) -- пример из выборки

- Инициализируем w_0
- На каждом шаге считаем градиент функции L: $L'_w(x, y, w_i)$
- Делаем шаг $w_{i+1} = w_i - \alpha L'_w(x, y, w_i)$




local
cost minimum
Global
cost minimum
J(W)
W

Обратное распространение ошибки *[Backpropagation]*

- Давайте распишем многослойный персептрон f(x, w):

$$f(x, w) = f_n(W_n f_{n-1}(W_{n-1} \dots f_1(W_1 x)))$$

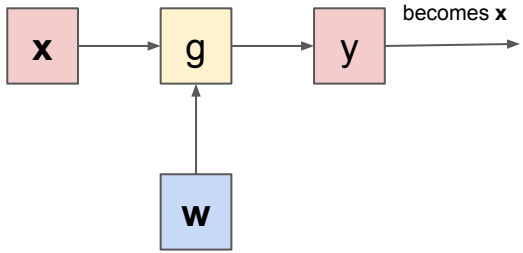
- Чтобы сделать шаг, нам нужно знать все $\frac{\partial L}{W_1} \cdots \frac{\partial L}{W_n}$

$$y = g(x, w)$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w}; \quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

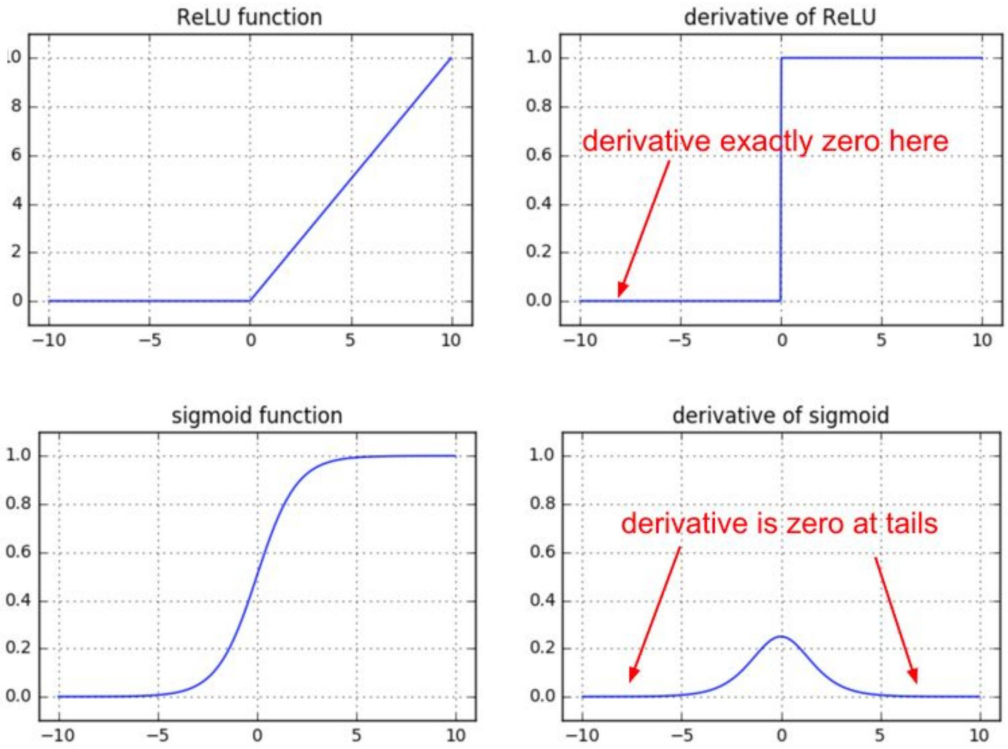
$$y = w \cdot x \Rightarrow \frac{\partial y}{\partial w} = x \Rightarrow \frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}x$$

$$y = w \cdot x \Rightarrow \frac{\partial y}{\partial x} = x \Rightarrow \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}w$$



Vanishing gradients

$$f(x, w) = Wx$$
$$f_j(x, w) = W_{jk} x_k$$
$$\frac{\partial f_j}{\partial x_i} = W_{ji}$$
$$\frac{\partial f}{\partial x} = W^T$$

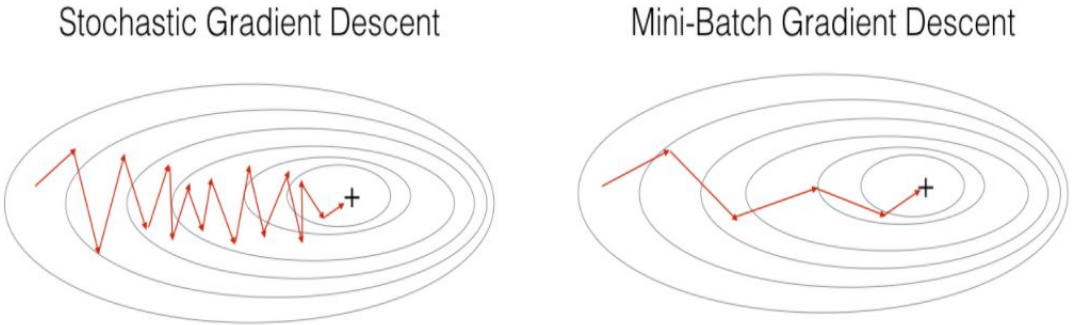


Стохастический градиентный спуск *[SGD]*

Можно считать градиент не по всей выборке и усреднять, а по подвыборке меньшего размера, называемой батч *[batch]*. Ее размер -- *batch size*.

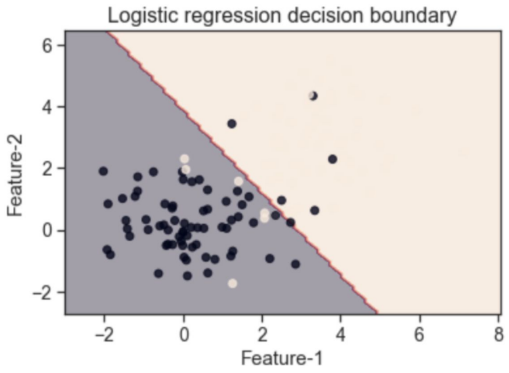
Это ускоряет сходимость, т.к. сетка сходится обычно за то же число шагов, но сами шаги занимают много меньше времени: batch_size << n_samples.

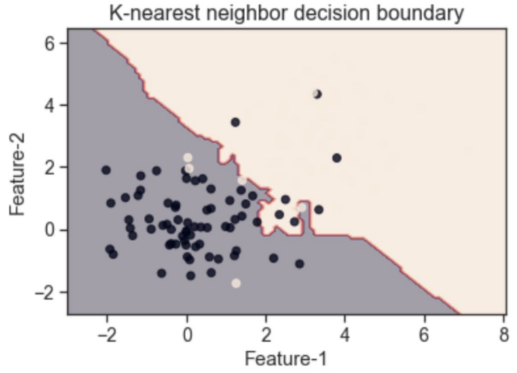
Всегда пользуйтесь этой вариацией, не считайте градиент по всей выборке!

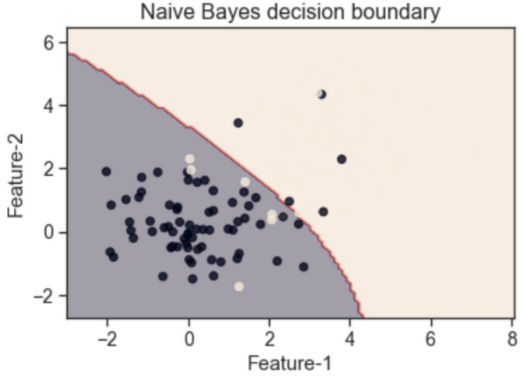


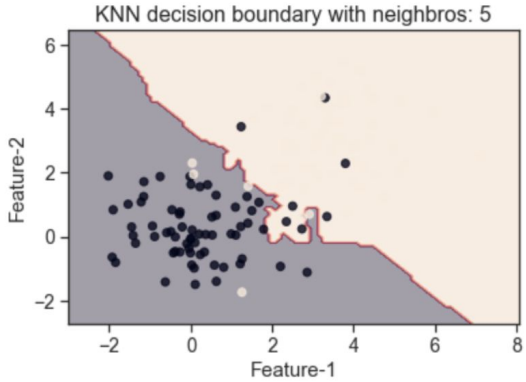
So why?

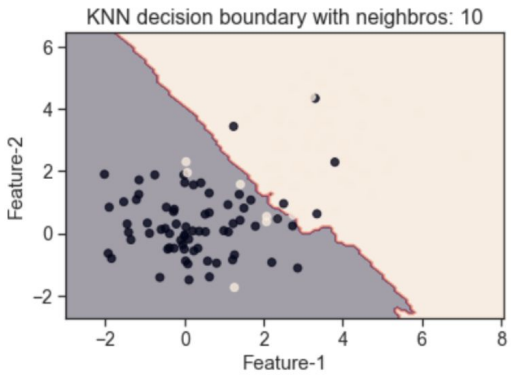
Затухающие градиенты *[Vanishing gradients]*

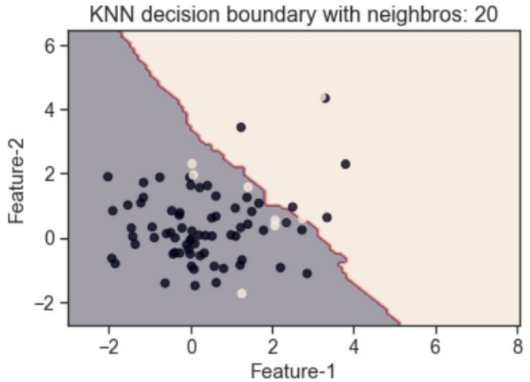












Сетки: pros. & cons.

+ Неограниченная генерализующая способность (могут находить сложнейшие закономерности в данных любой структуры)
+ Блоки и их устройство могут содержать информацию о реальном мире (RNN, CNN), упрощая поиск разделяющей поверхности
+ Могут выучивать собственное представление для структурированных данных, которое потом можно переиспользовать *[embeddings]*
+ Multitasking & transfer learning

- Могут переобучаться быстрее стандартных алгоритмов
- Dim of input space should be >> dim search space
- Тонна трюков в виде разных оптимайзеров, gradient penalty/clipping, regularization, scheduling, pre-/self- training, normalization, ... иногда просто чтобы оно р а б о т а л о

Summary: очень хороши на мультимодальных данных (картинка, текст, звук), но обычно проигрывают классическим алгоритмам в табличках (не time series).

Further reading

- Interactive neural network playground in your browser
- Backprop in depth by cs231
- **A recipe for training neural networks**
- Примерно то же самое, что я рассказал, только от Стенфорда и на английском
- **Official intro to Keras**
- Deep learning frameworks (russian) - video
- Deep learning frameworks (english) - video

Пример вычислительного графа

tensorflow 1.x (static):

```
sess = tf.Session(...)

x = tf.placeholder(...)

y = tf.placeholder(...)

out = x ** 2 * y + (y + 2)

print(sess.run(out, {x: 1, y: 2}))

# 6
```