

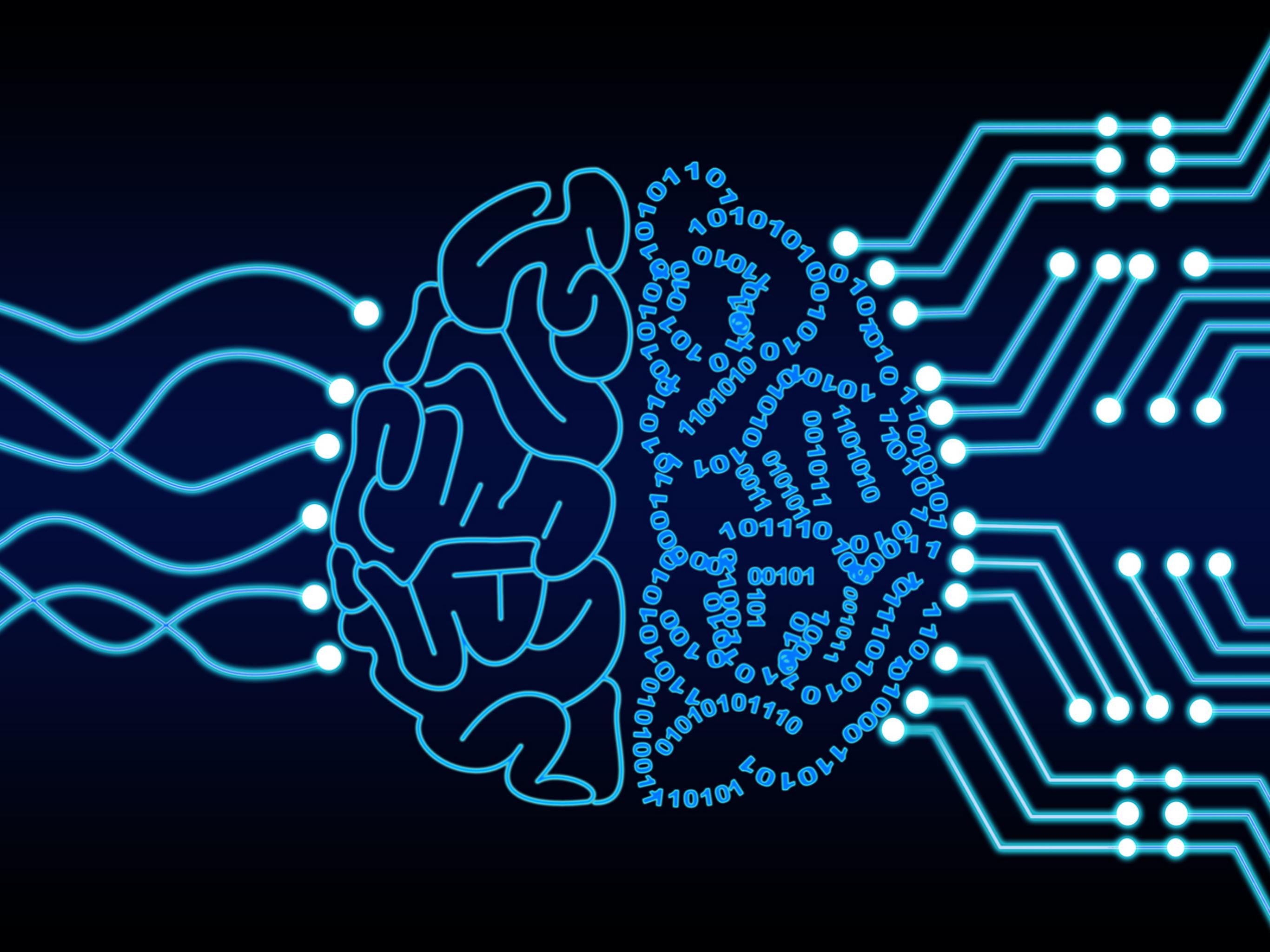
Машинное обучение

Часть III - Dive into Deep Learning

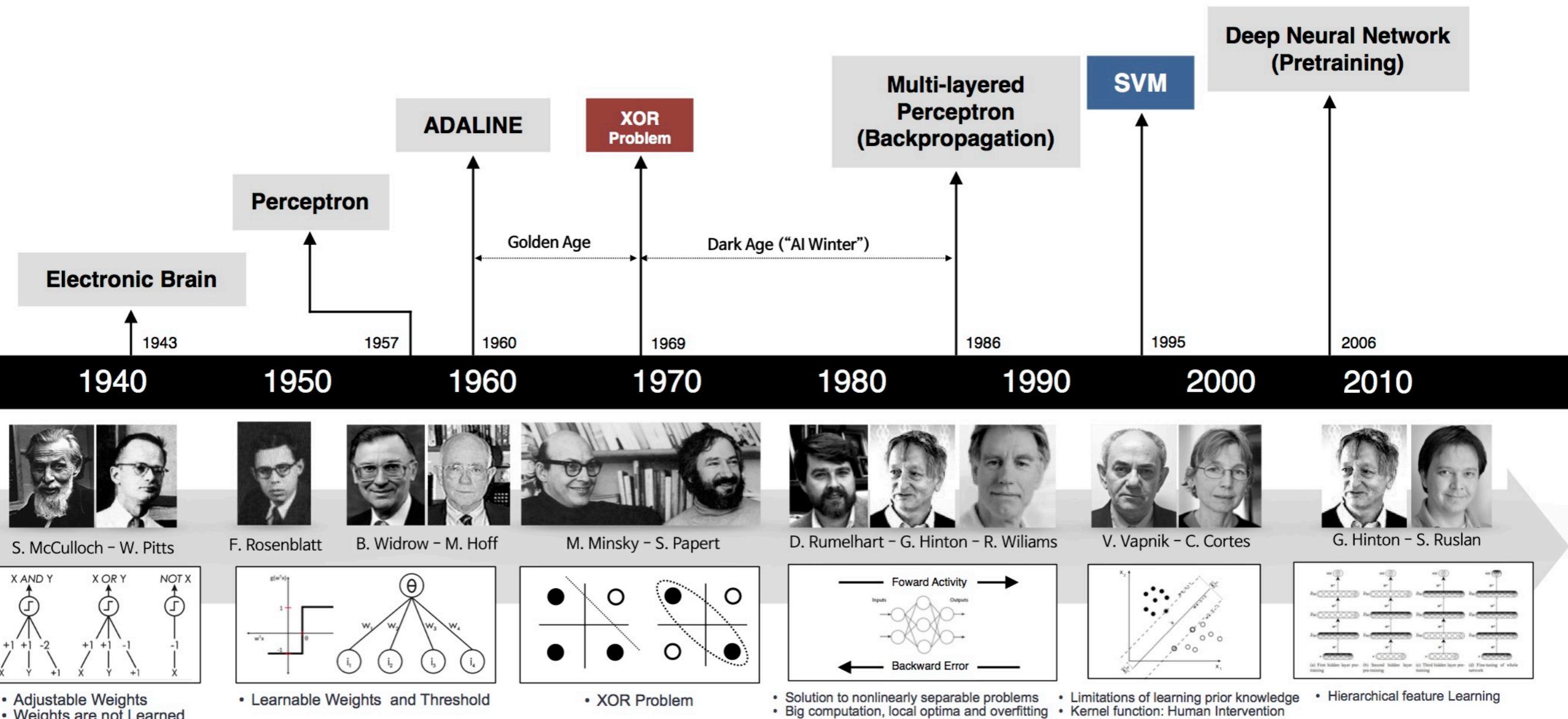
Власов Кирилл Вячеславович



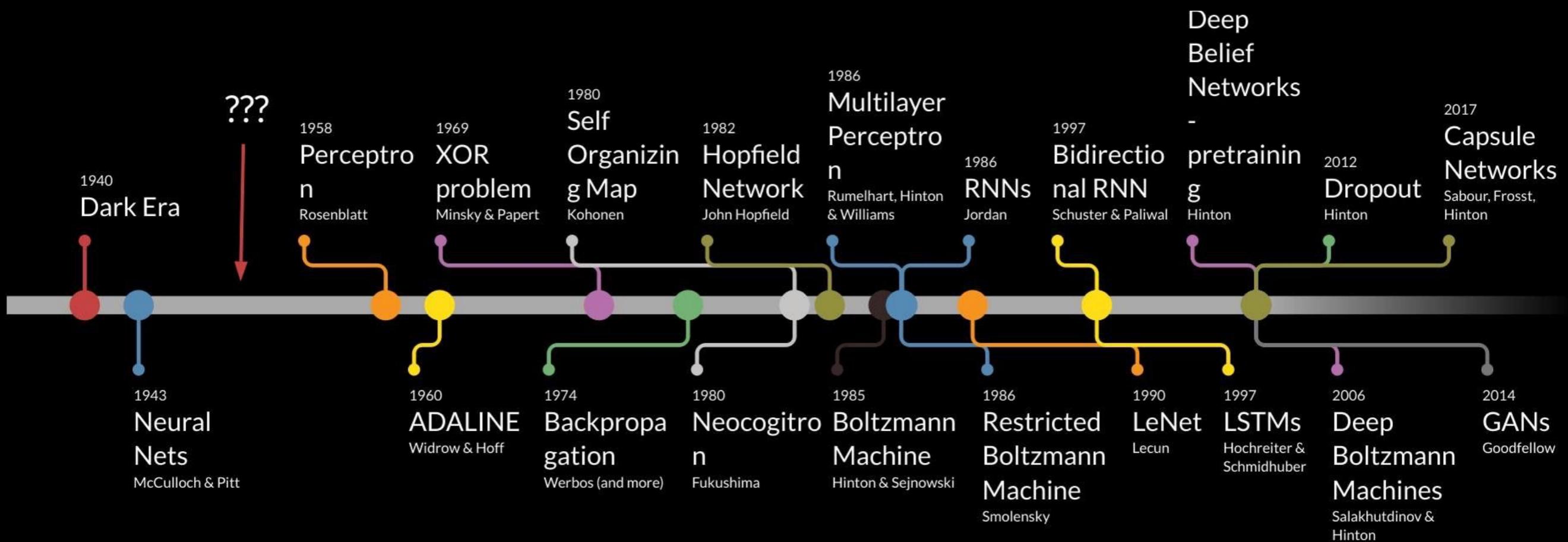
2019



Deep Learning: Timeline

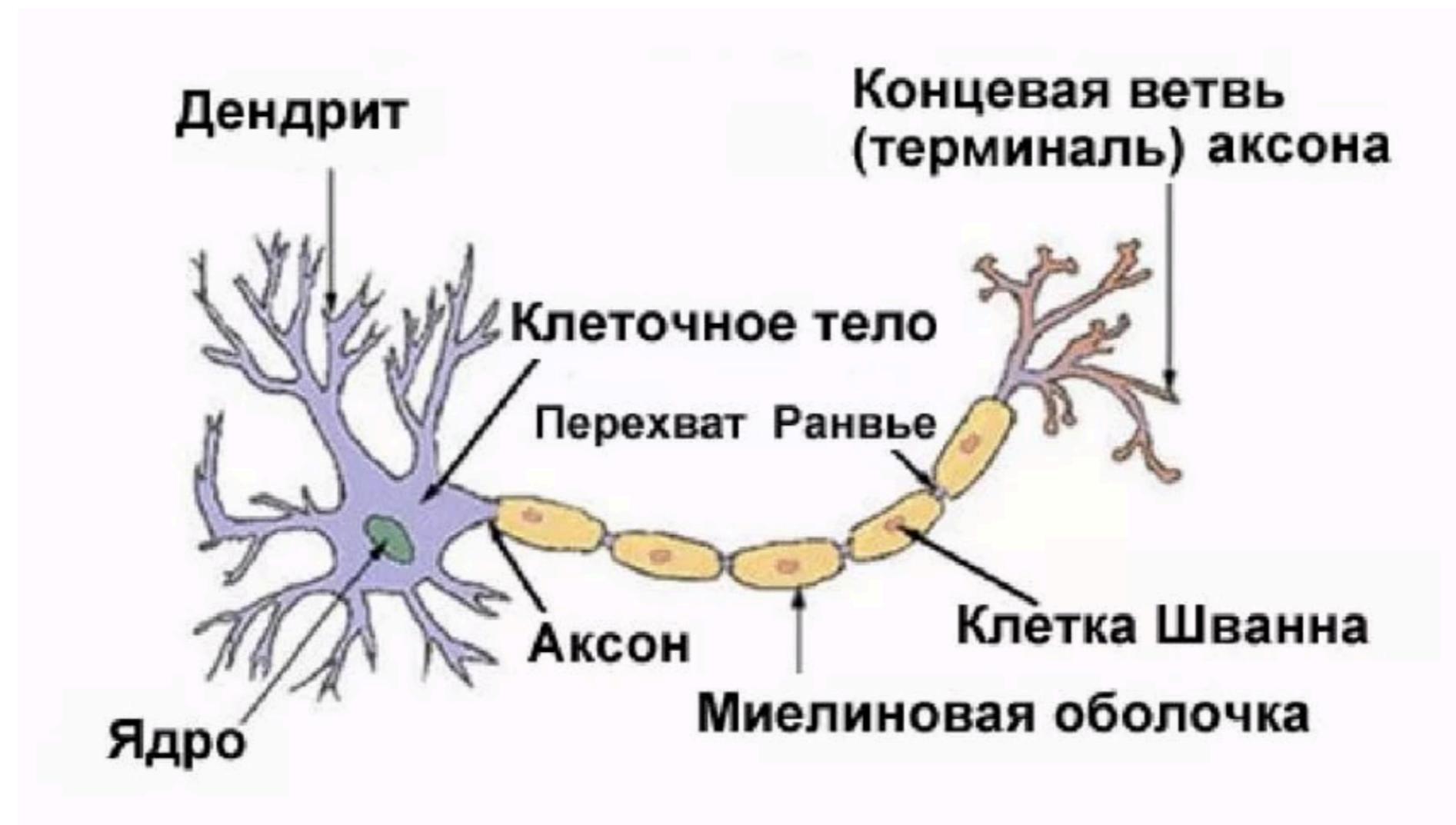


Deep Learning: Timeline



Made by Favio Vázquez

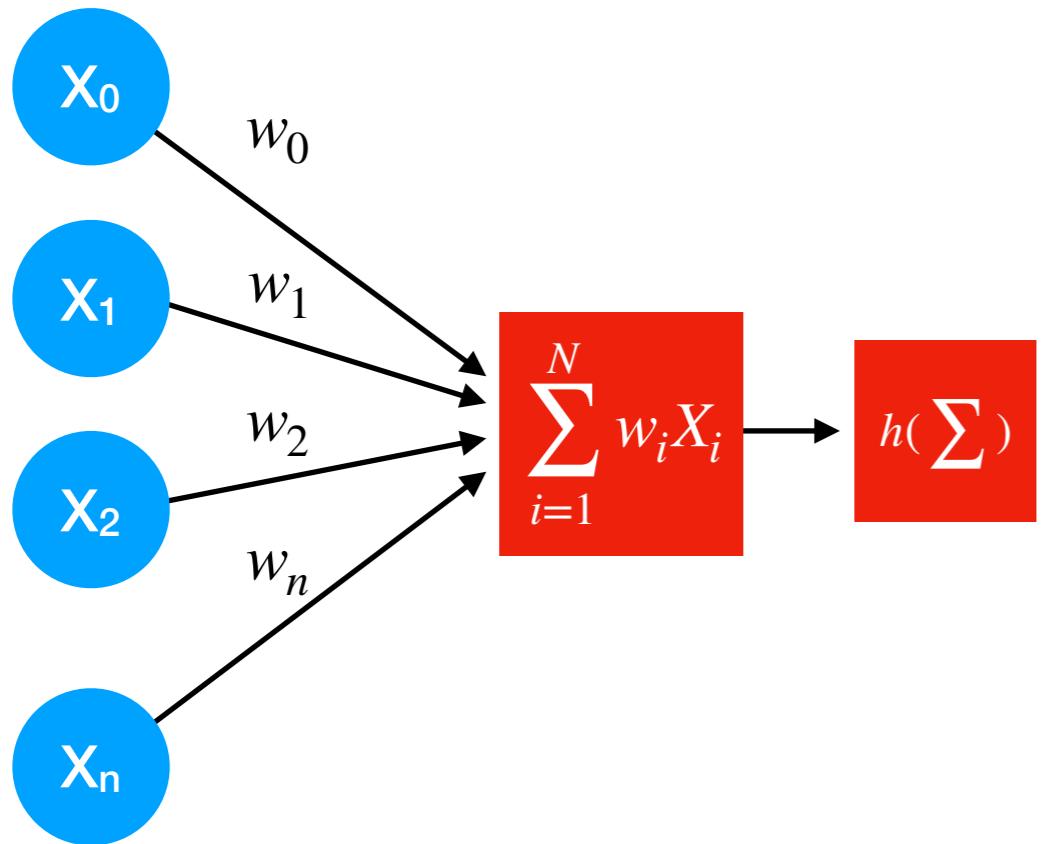
Аналогия с биологическим нейроном



Искусственный нейрон

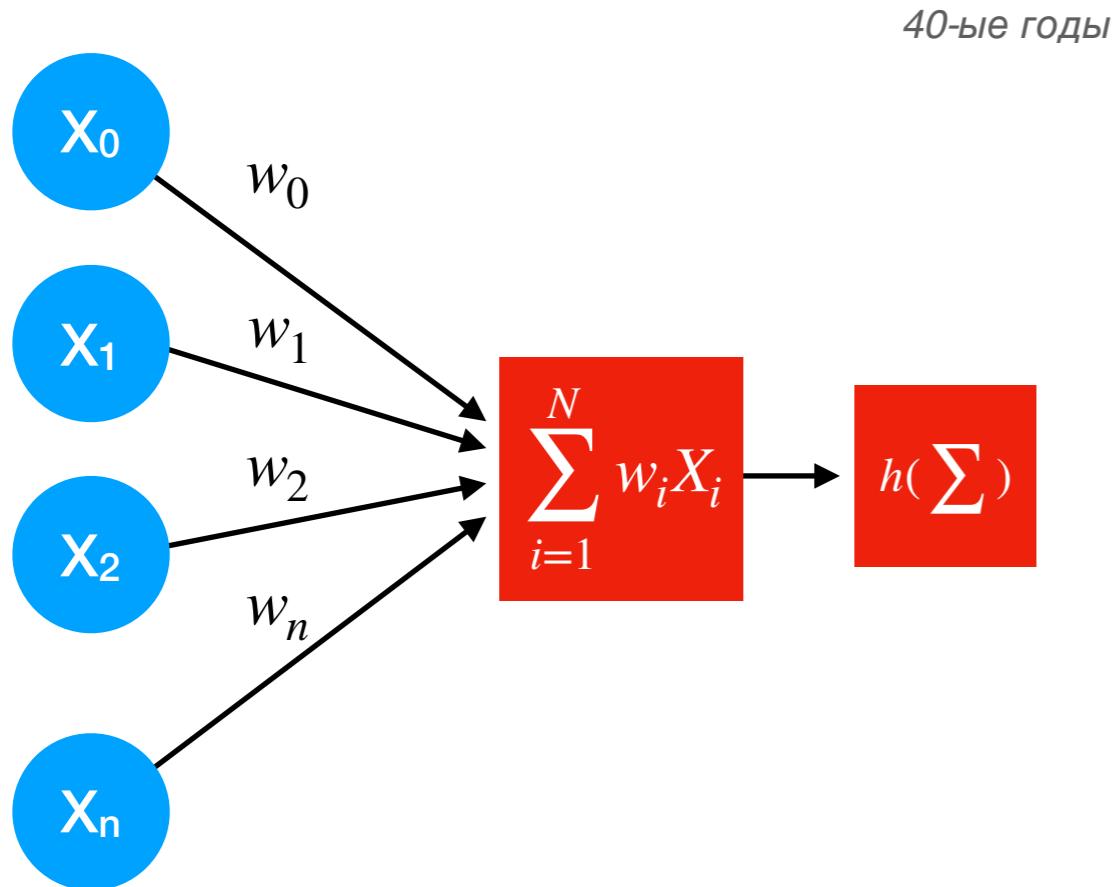
Математический нейрон Маккаллока – Питтса

40-ые годы

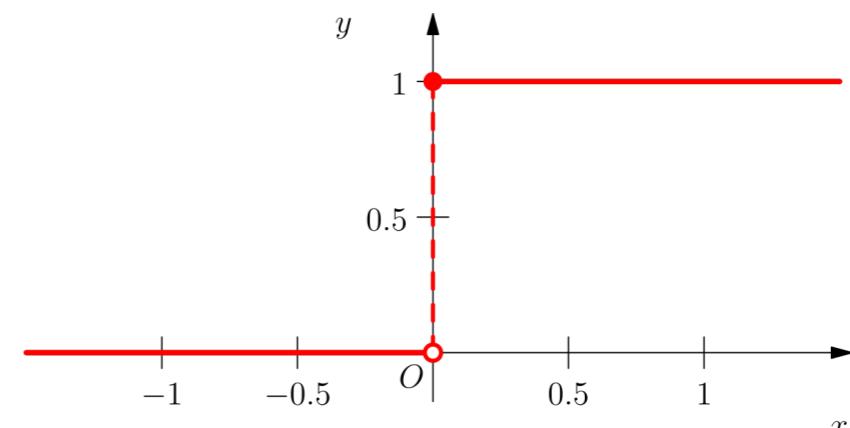


Искусственный нейрон

Математический нейрон Маккаллока – Питтса

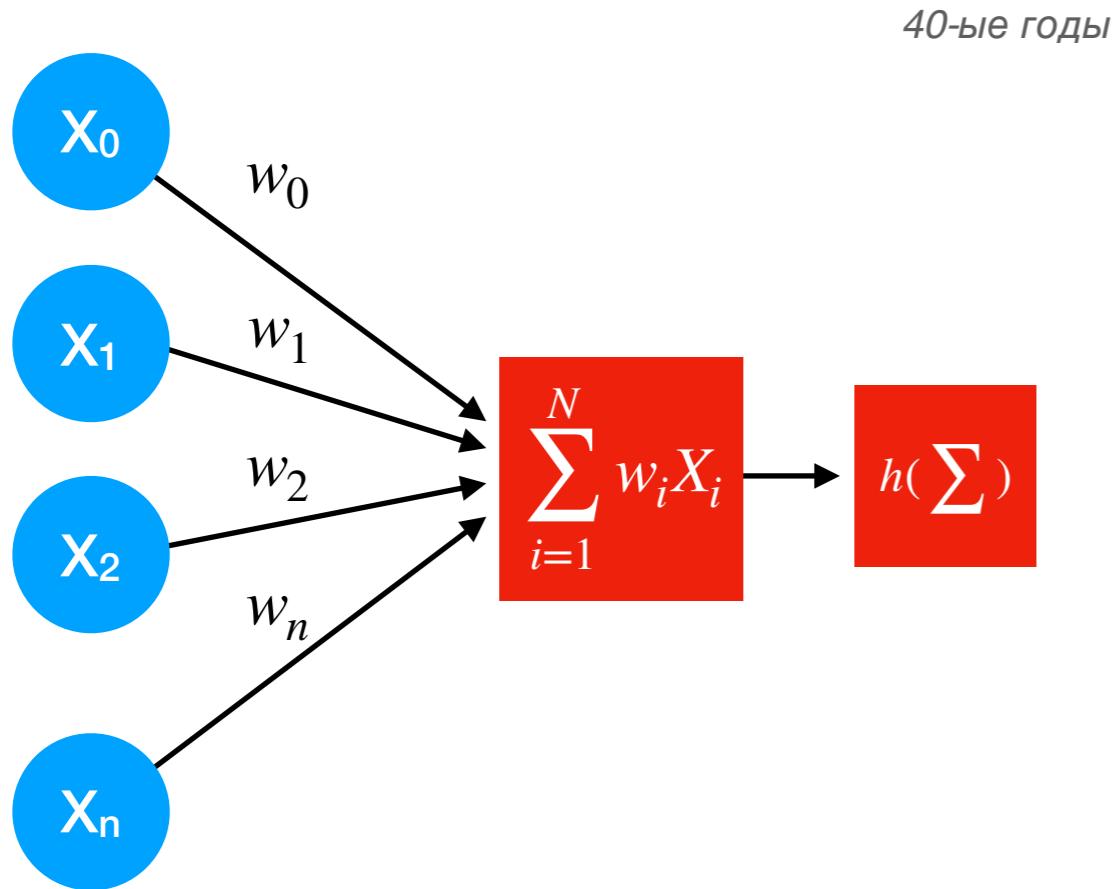


Функция Хевисайда

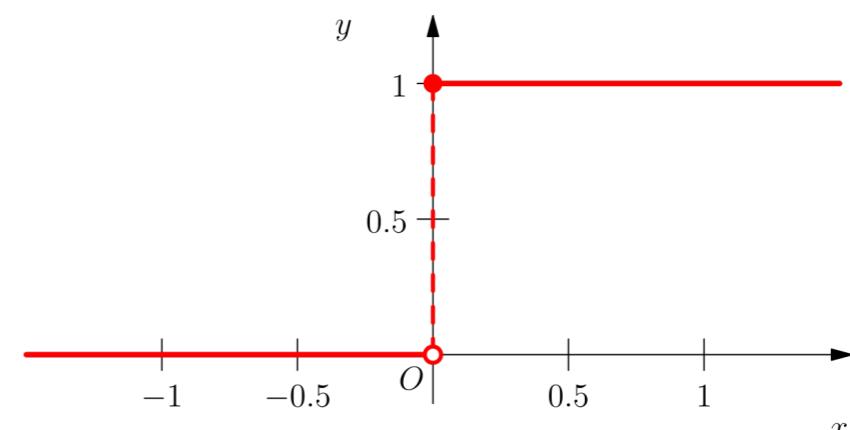


Искусственный нейрон

Математический нейрон Маккаллока – Питтса



Функция Хевисайда

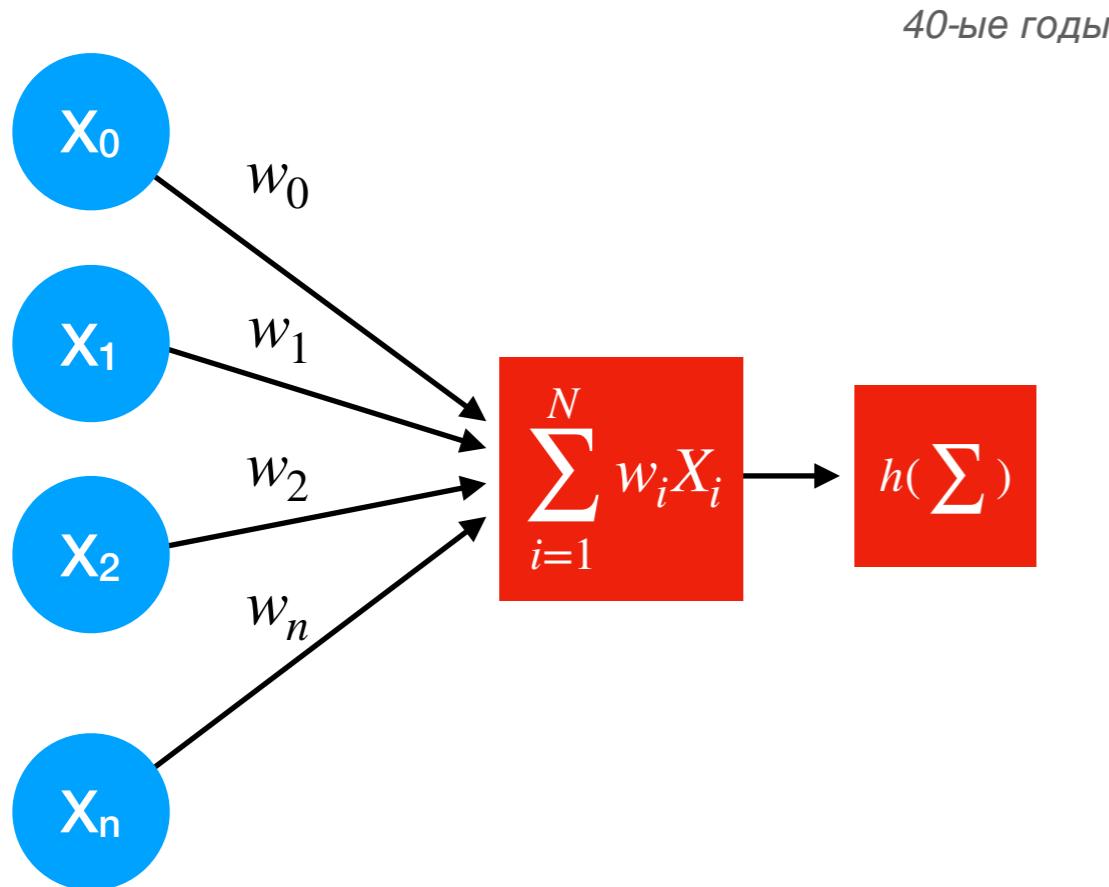


Персепtron Розенблатта

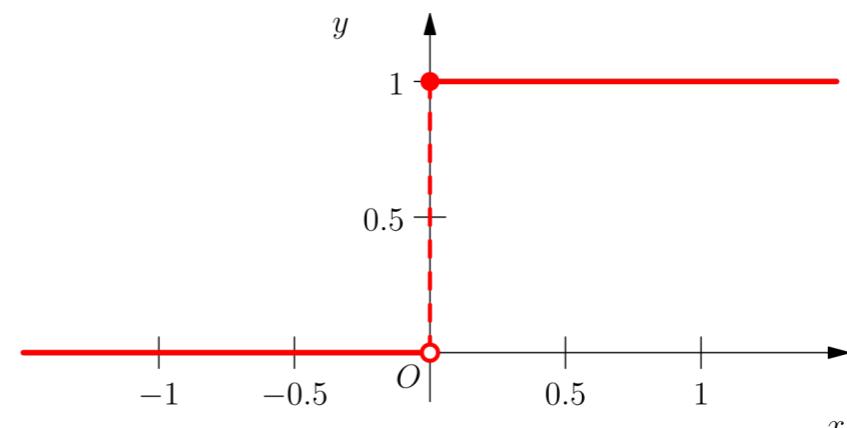
- Первое правило Хебба – *Если сигнал персептрана неверен и равен нулю, то необходимо увеличить веса тех входов, на которые была подана единица.*
- Второе правило Хебба – *Если сигнал персептрана неверен и равен единице, то необходимо уменьшить веса тех входов, на которые была подана единица.*

Искусственный нейрон

Математический нейрон Маккаллока – Питтса



Функция Хевисайда



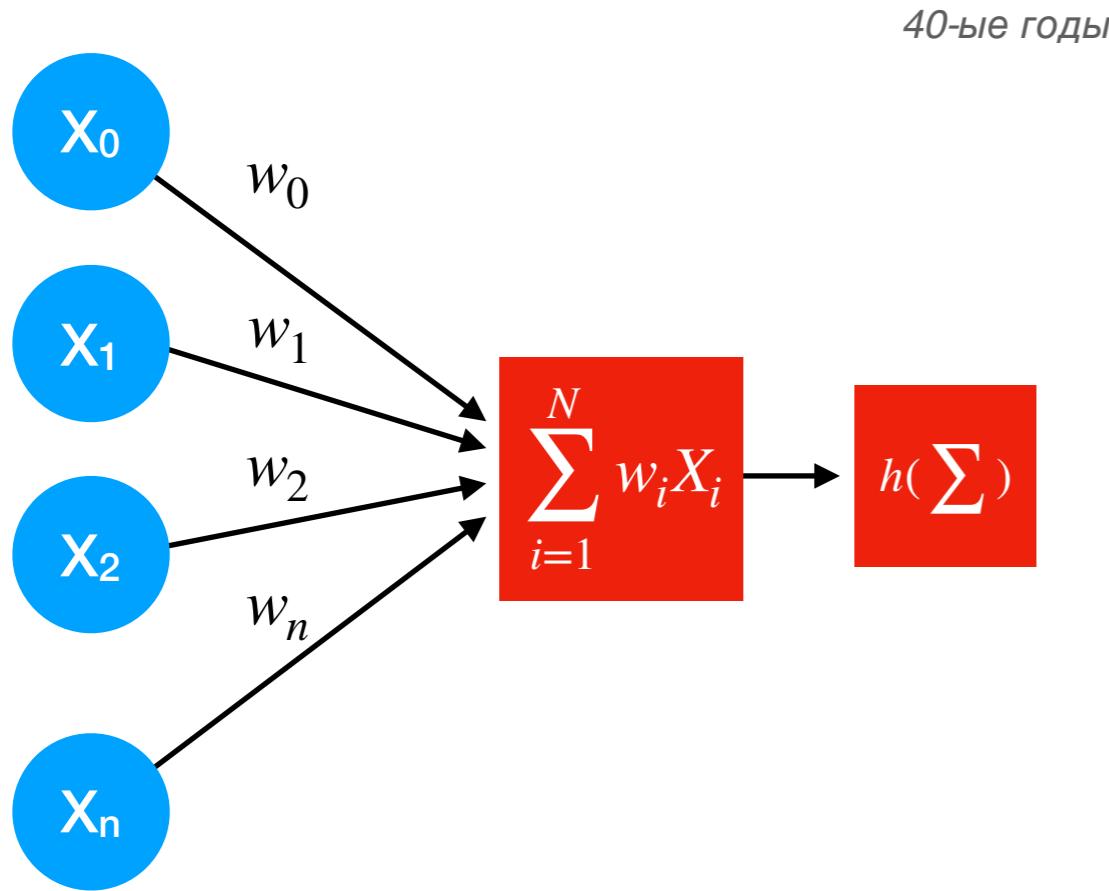
Персепtron Розенблатта

- Первое правило Хебба – *Если сигнал персептрана неверен и равен нулю, то необходимо увеличить веса тех входов, на которые была подана единица.*
- Второе правило Хебба – *Если сигнал персептрана неверен и равен единице, то необходимо уменьшить веса тех входов, на которые была подана единица.*

Биологическая предпосылка:
Если нейрон срабатывает, то синоптическая связь укрепляется

Искусственный нейрон

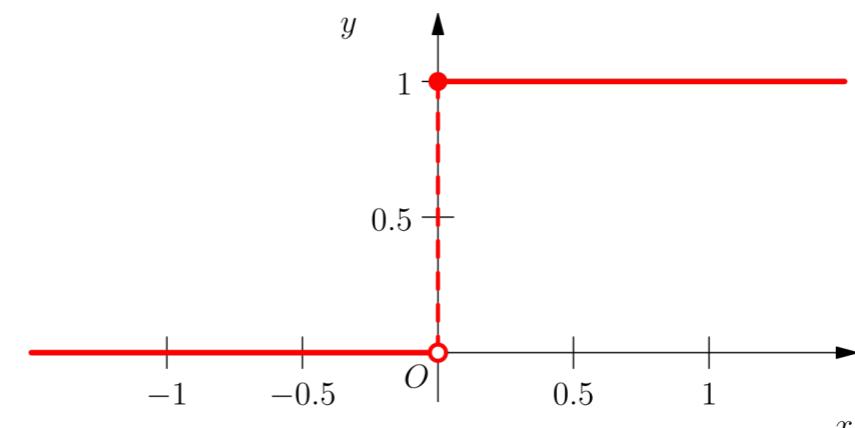
Математический нейрон Маккаллока – Питтса



Персепtron Розенблатта

- Первое правило Хебба – *Если сигнал персептрана неверен и равен нулю, то необходимо увеличить веса тех входов, на которые была подана единица.*
- Второе правило Хебба – *Если сигнал персептрана неверен и равен единице, то необходимо уменьшить веса тех входов, на которые была подана единица.*

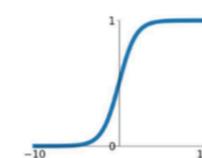
Функция Хевисайда



Другие функции активации:

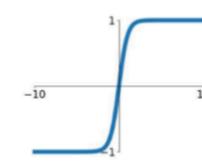
Sigmoid

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



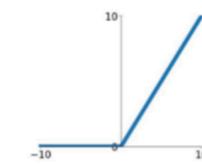
tanh

$$\tanh(x)$$



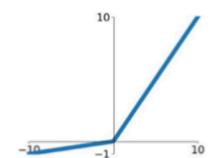
ReLU

$$\max(0, x)$$



Leaky ReLU

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

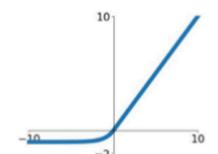


Maxout

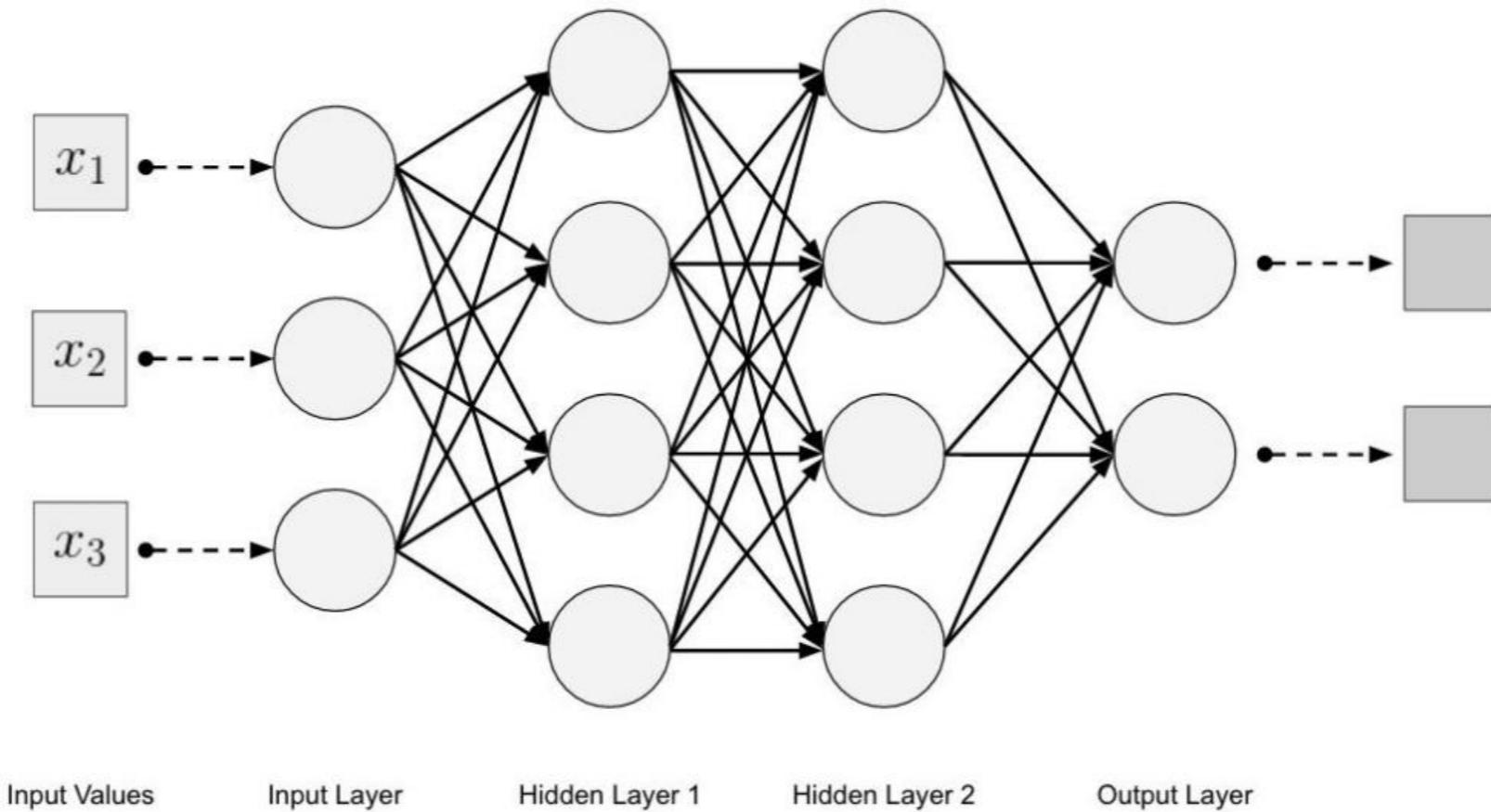
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

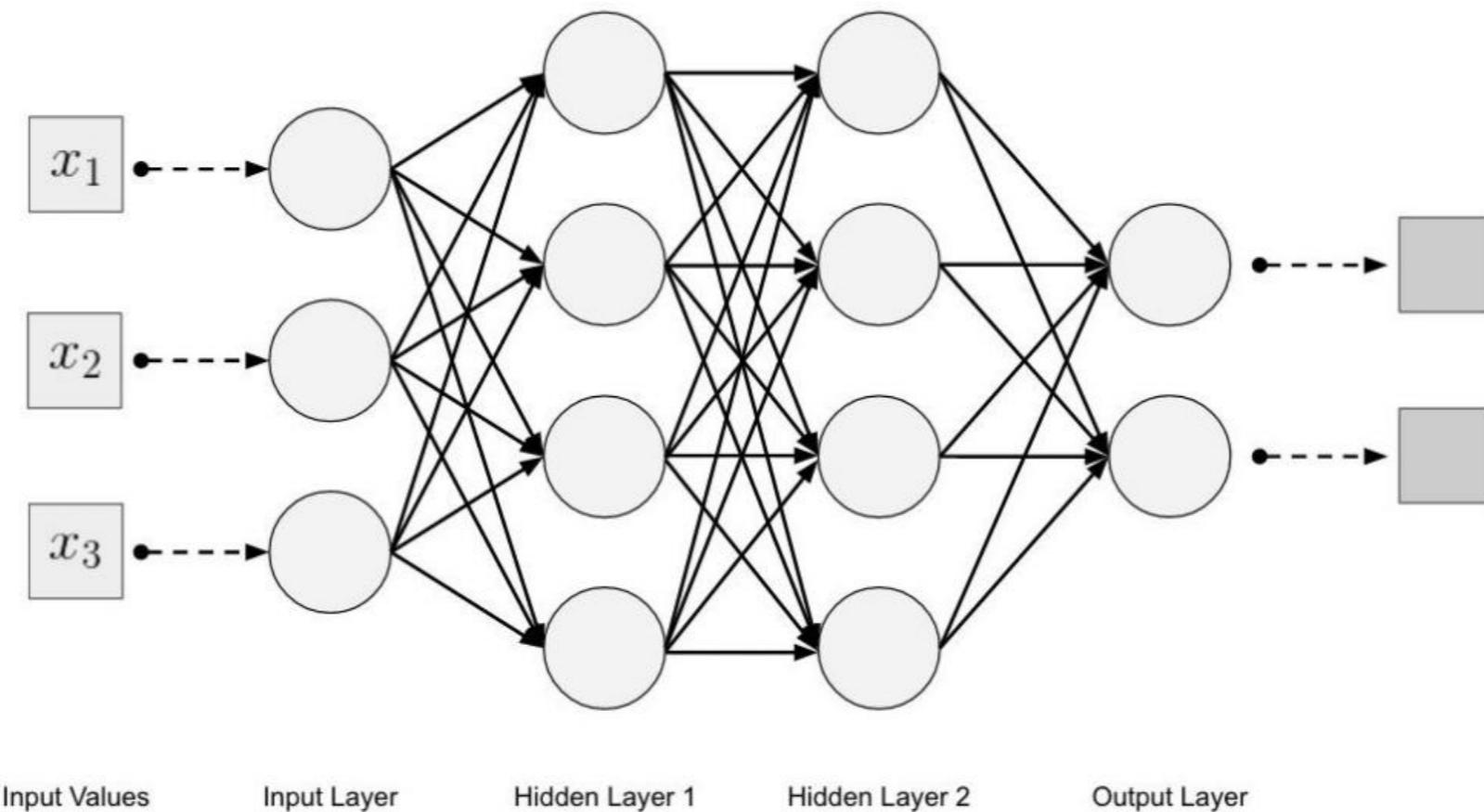
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Нейронные сети



Нейронные сети



Утверждения:

Любая булева функция представима в виде нейронной сети с одним скрытым слоем с нелинейной функцией активации нейрона (но может потребоваться экспоненциально много нейронов в скрытом слое).

Любая непрерывная и ограниченная функция может быть сколь угодно точно аппроксимирована нейронной сетью с одним скрытым слоем с нелинейной функцией активации нейрона.

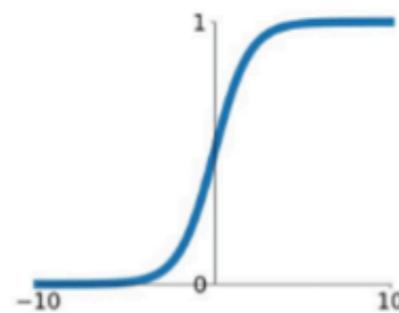
Любая функция может быть сколь угодно точно аппроксимирована нейронной сетью с двумя скрытыми слоями с нелинейной функцией активации нейрона.

Обучение нейронных сетей

ФУНКЦИИ АКТИВАЦИИ

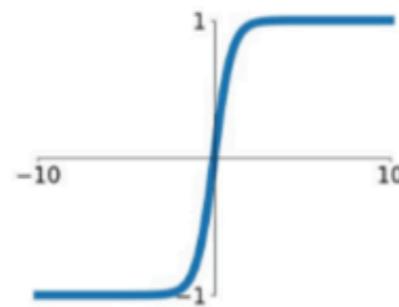
Sigmoid

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



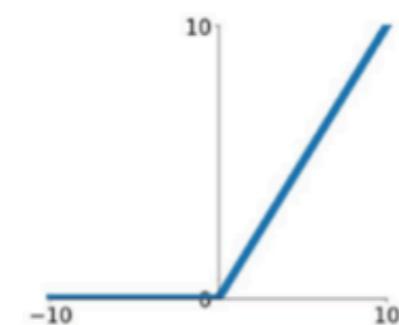
tanh

$$\tanh(x)$$



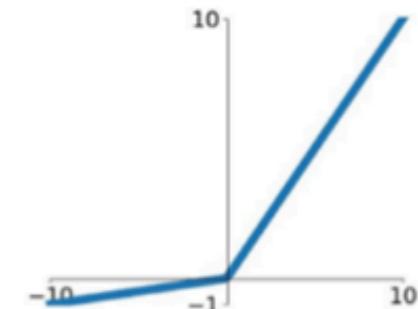
ReLU

$$\max(0, x)$$



Leaky ReLU

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

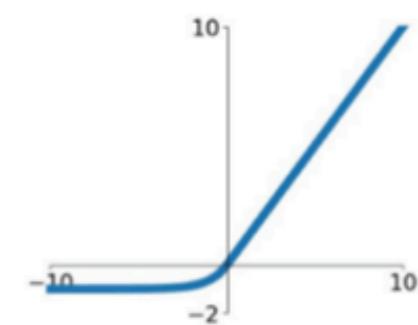


Maxout

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

ELU

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



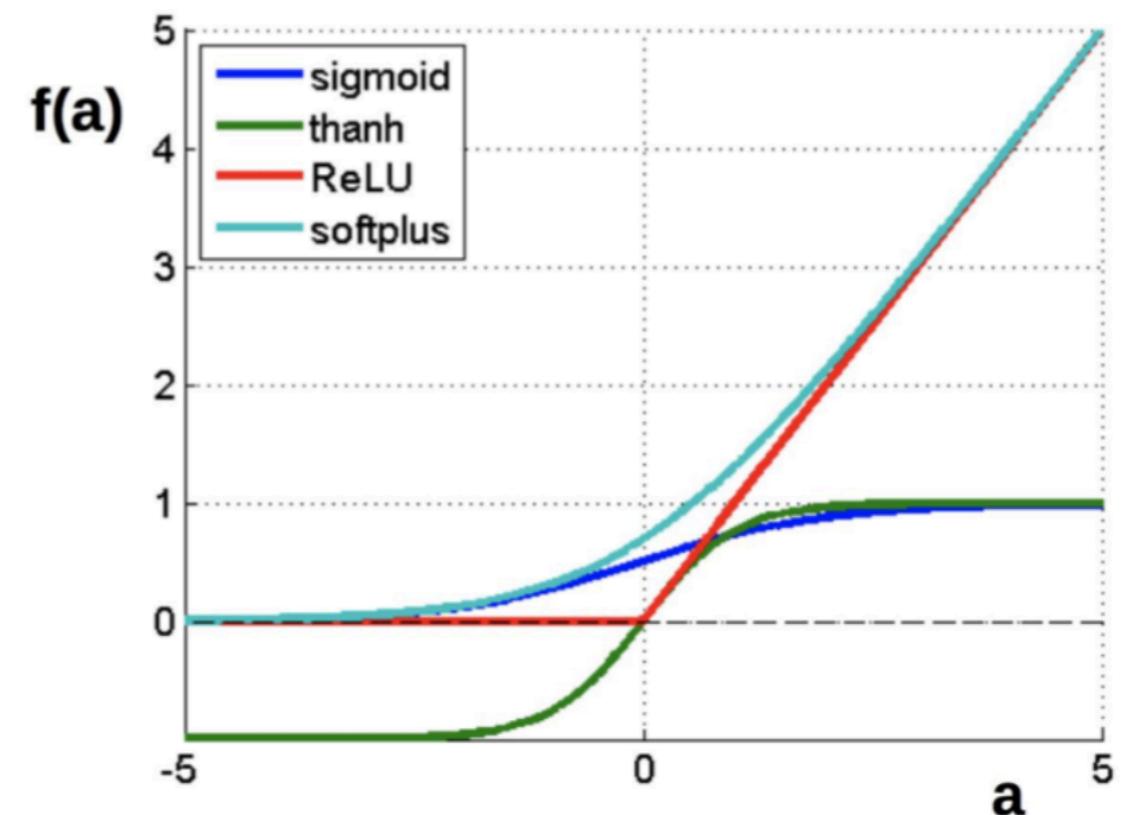
ФУНКЦИИ АКТИВАЦИИ

$$f(a) = \frac{1}{1 + e^a}$$

$$f(a) = \tanh(a)$$

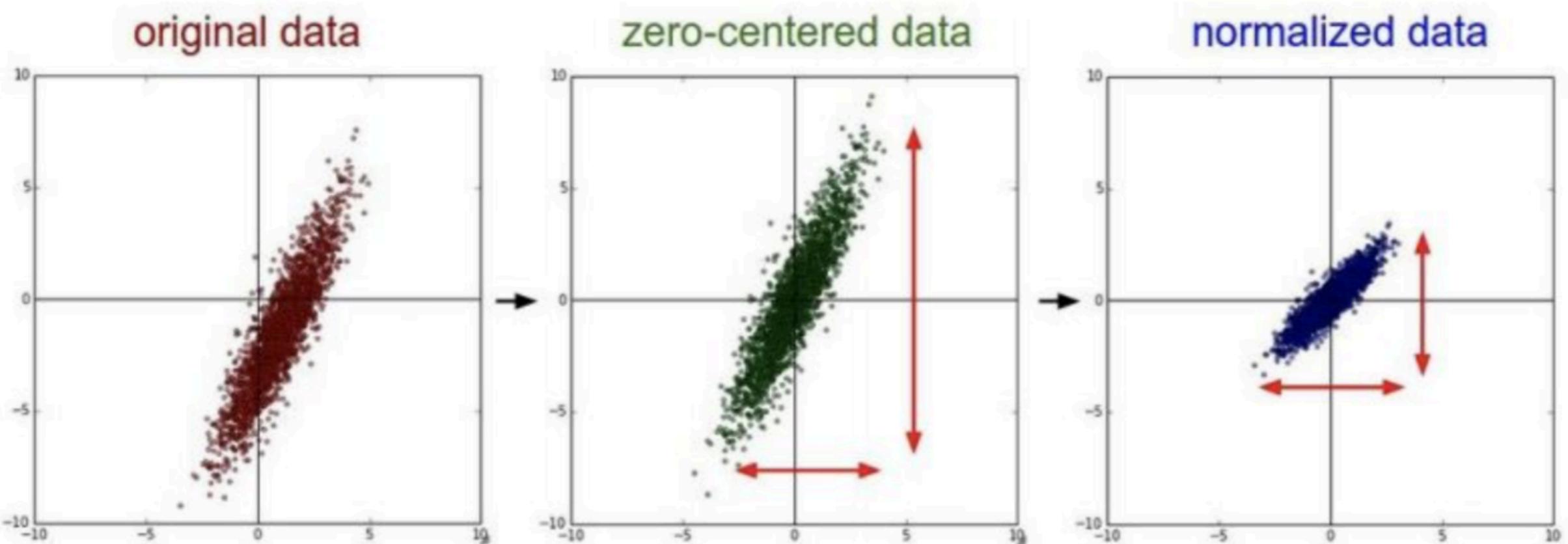
$$f(a) = \max(0, a)$$

$$f(a) = \log(1 + e^a)$$



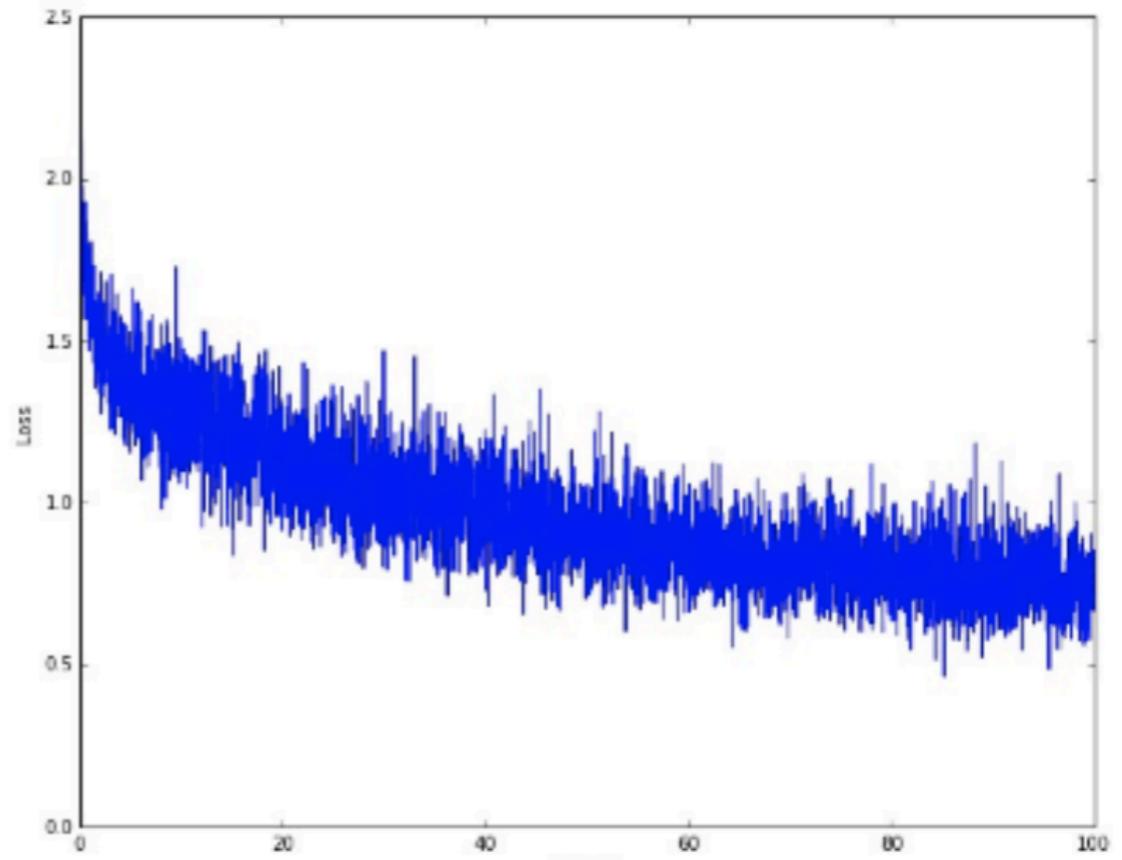
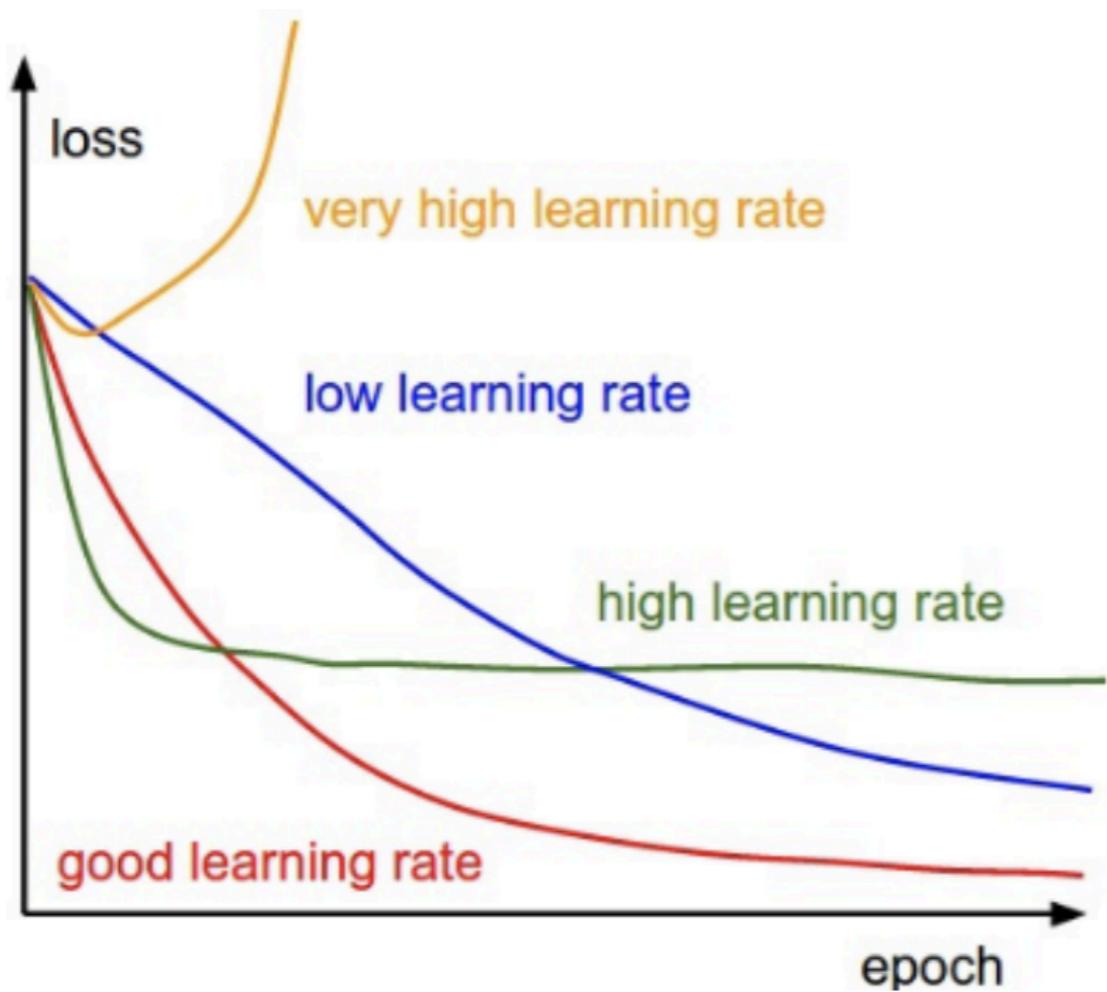
Interactive playground

Нормализация данных



Обновление весов

$$x_{t+1} = x_t - \text{learning rate} \cdot dx$$

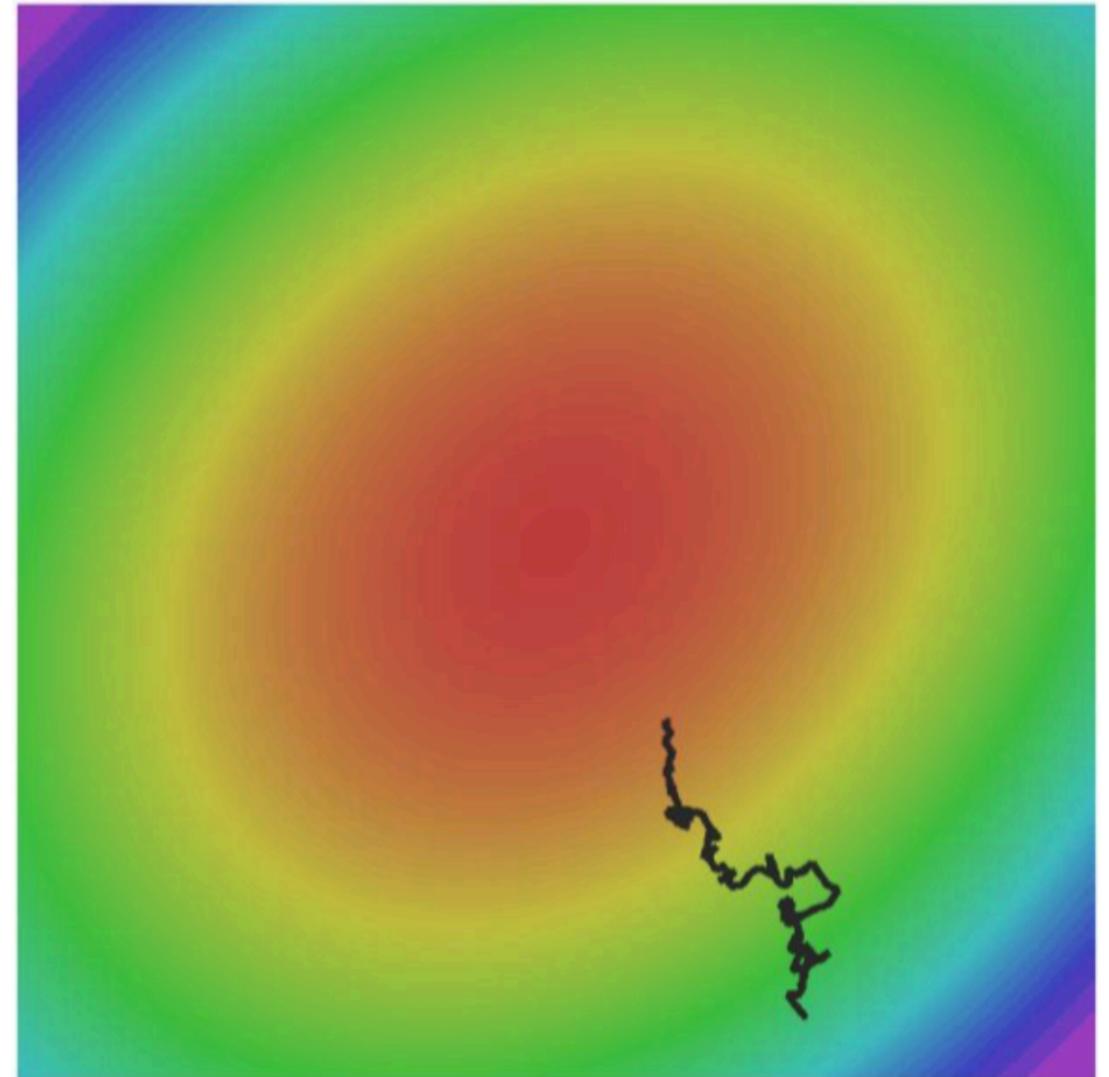


Обновление весов

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



Обновление весов

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

SGD+Momentum

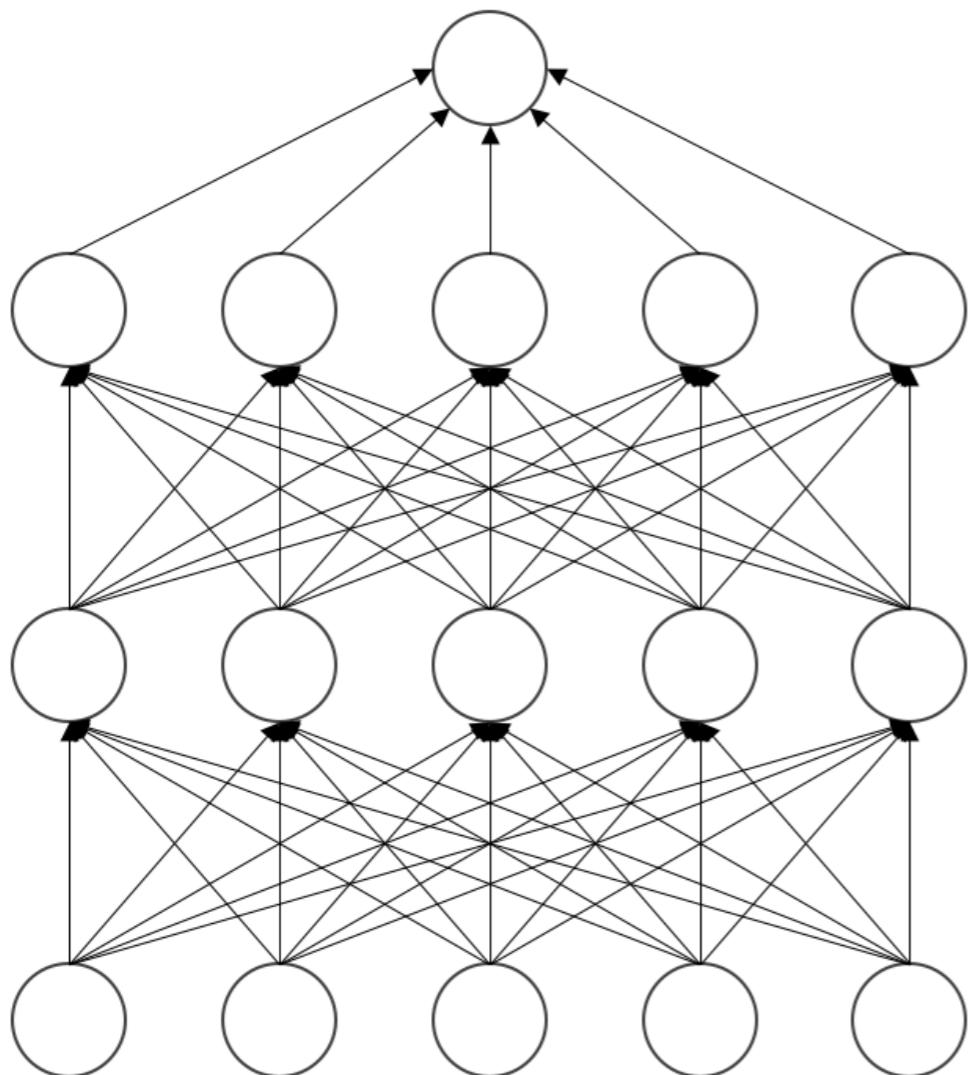
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

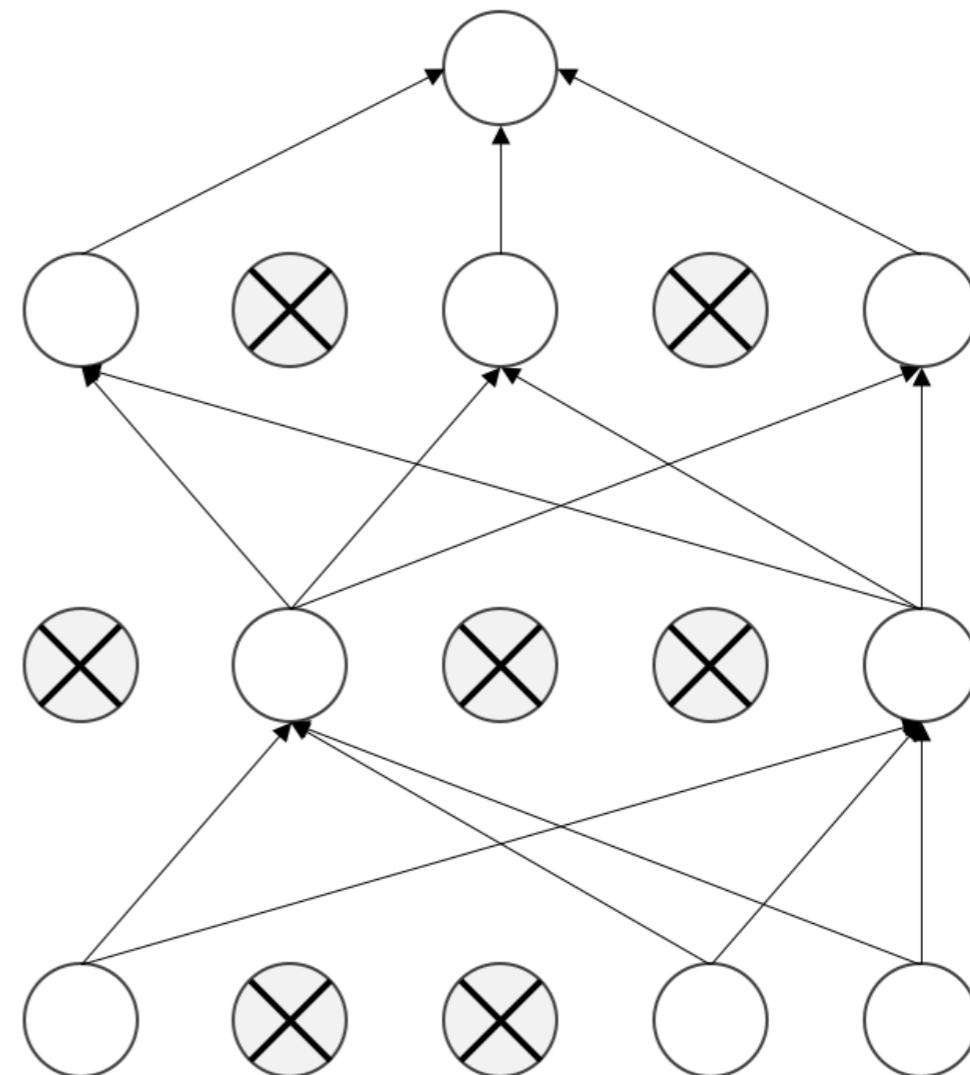
```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Регуляризация: DropOut

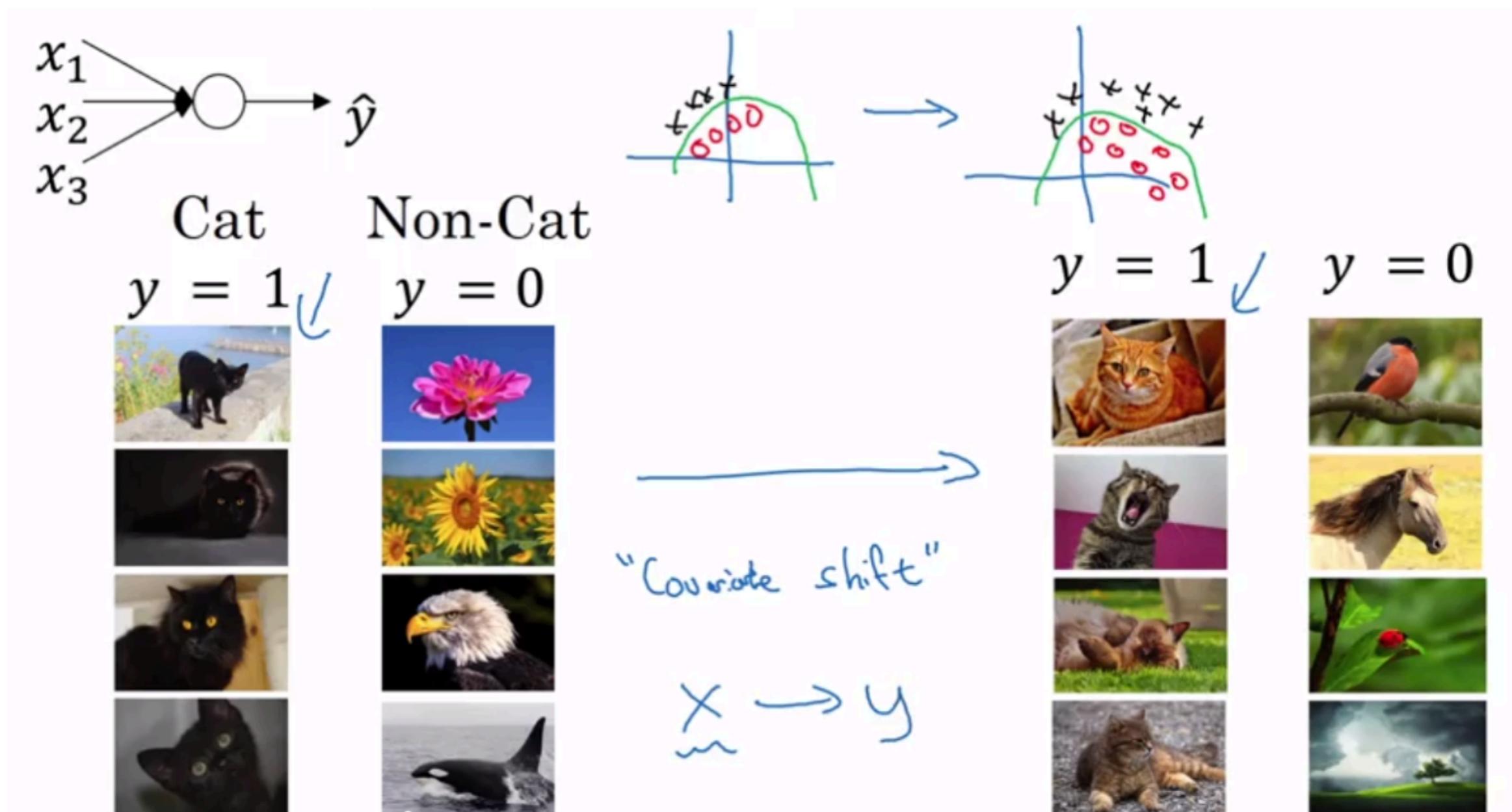


Standard Neural Net



After applying dropout

Регуляризация: Batch Normalisation



- Оригинальная статья: [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)
- [Andrew Ng: Why Does Batch Norm Work?](#)
- [How Does Batch Normalization Help Optimization?](#)

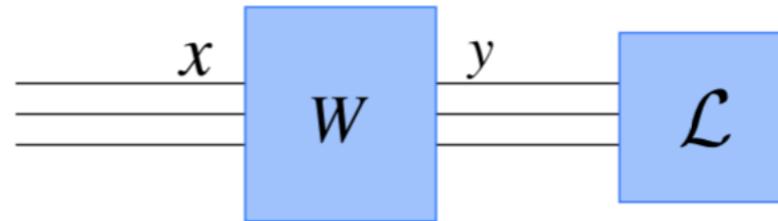
Регуляризация: Batch Normalisation

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

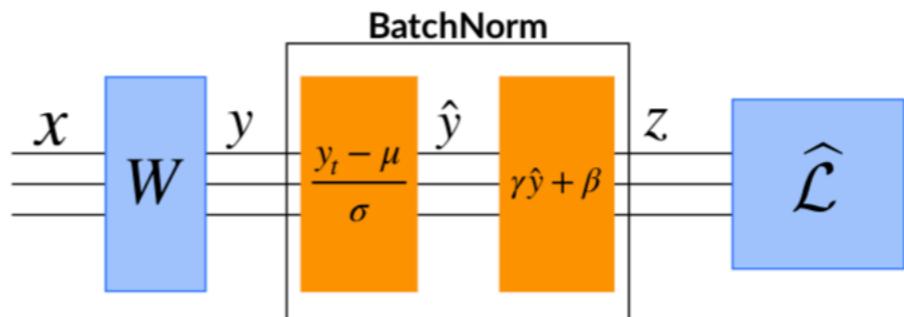
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.



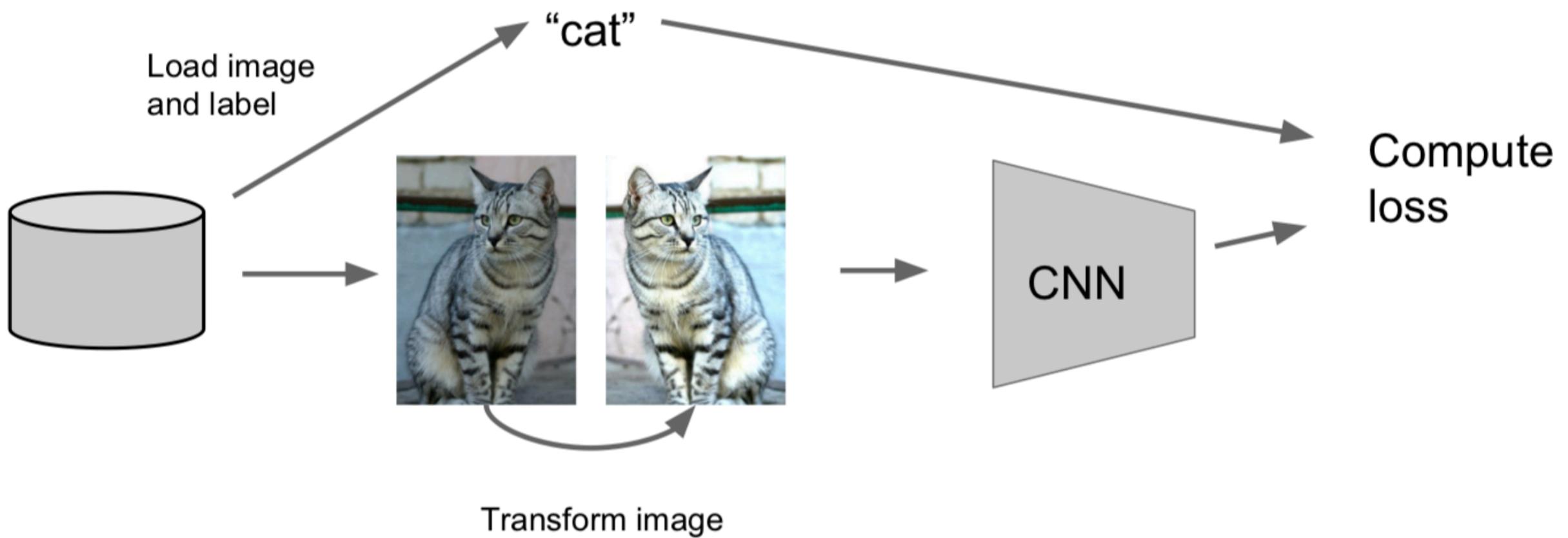
(a) Vanilla Network



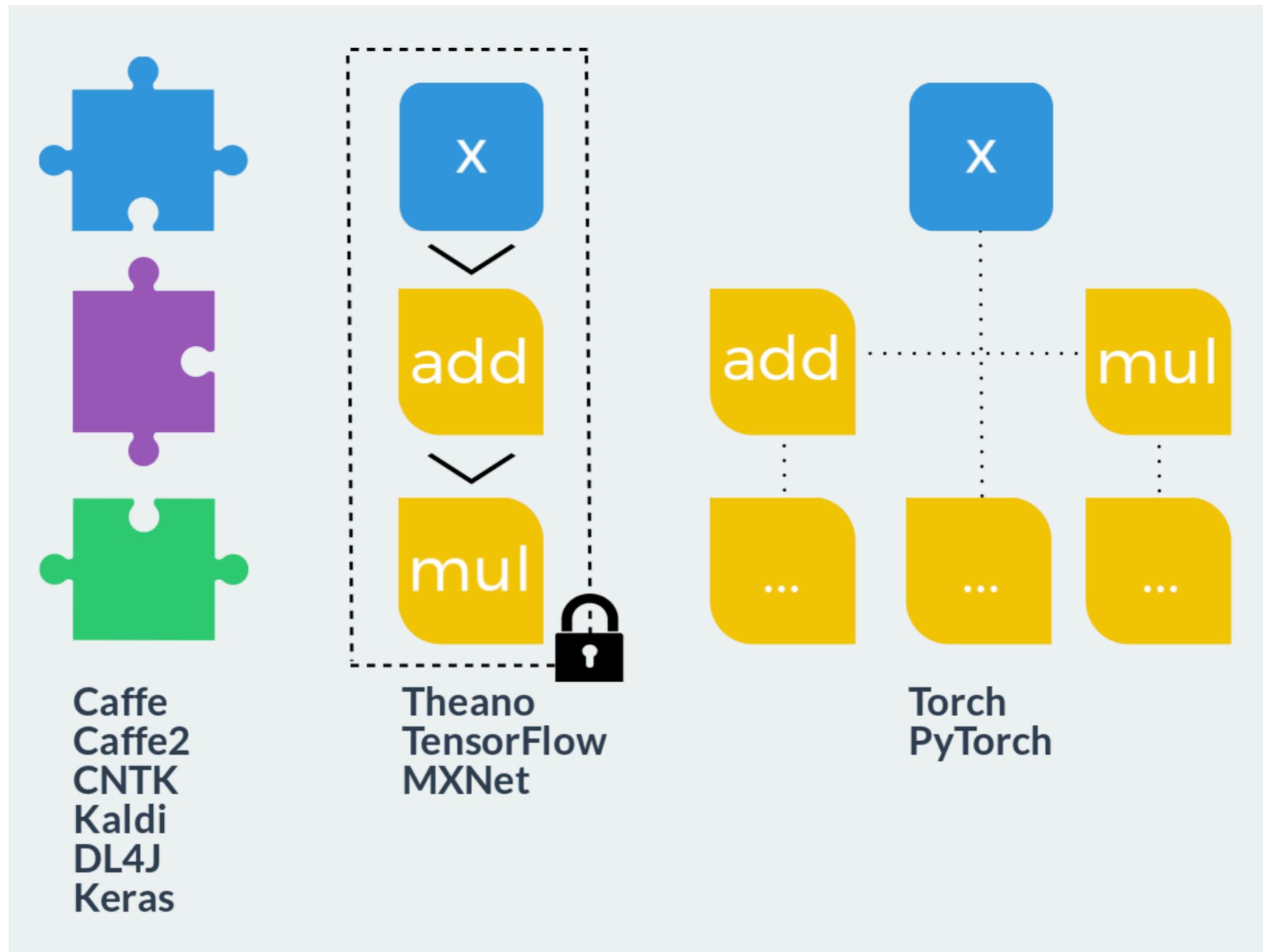
(b) Vanilla Network + BatchNorm Layer

- Оригинальная статья: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
- Andrew Ng: Why Does Batch Norm Work?
- How Does Batch Normalization Help Optimization?

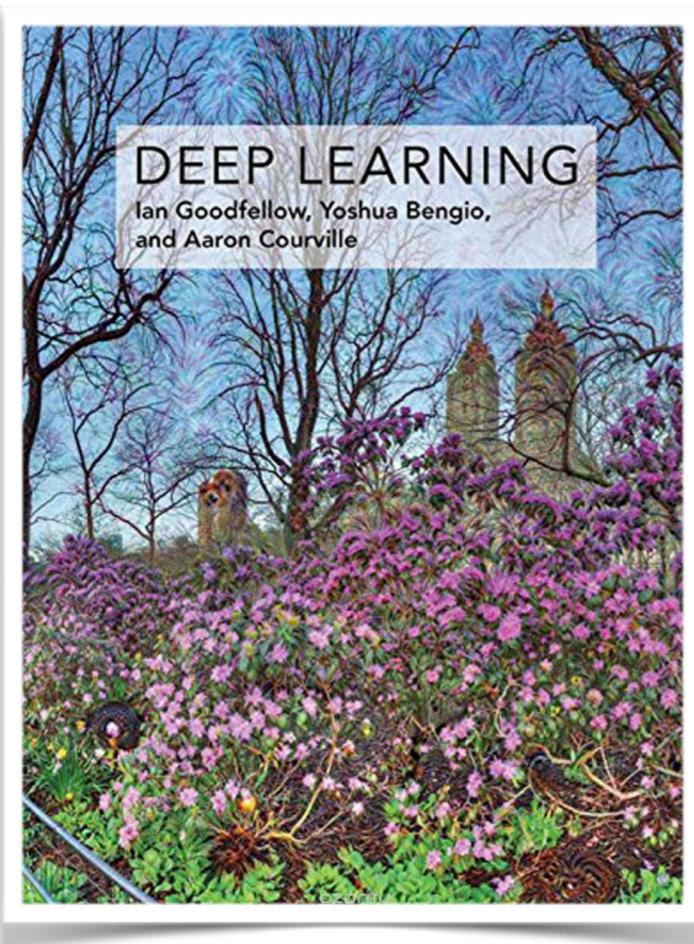
Регуляризация: Аугментация



Сравнение библиотек



Литература и ссылки



<http://www.deeplearningbook.org>

Deep Learning на пальцах

CS231n: Convolutional Neural Networks for Visual Recognition

CS224n: Natural Language Processing with Deep Learning