



Тверской
государственный
технический
университет

Интеллектуальные информационные системы

Искусственные нейронные сети
Deep Learning Intro

2025 г.

DL \subset **NN** \subset **ML** \subset **AI**

Artificial Intelligence (AI)

- The broad concept of machines performing tasks that typically require **human intelligence**.

Machine Learning (ML)

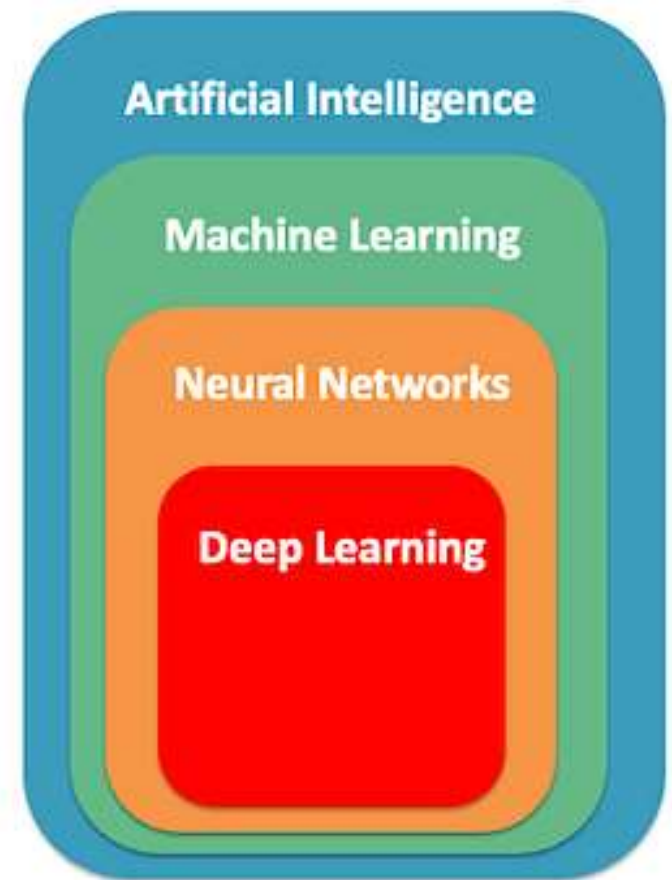
- A subset of AI where systems **learn from data**, identify patterns, and make decisions with minimal human intervention.

Neural Networks (NN)

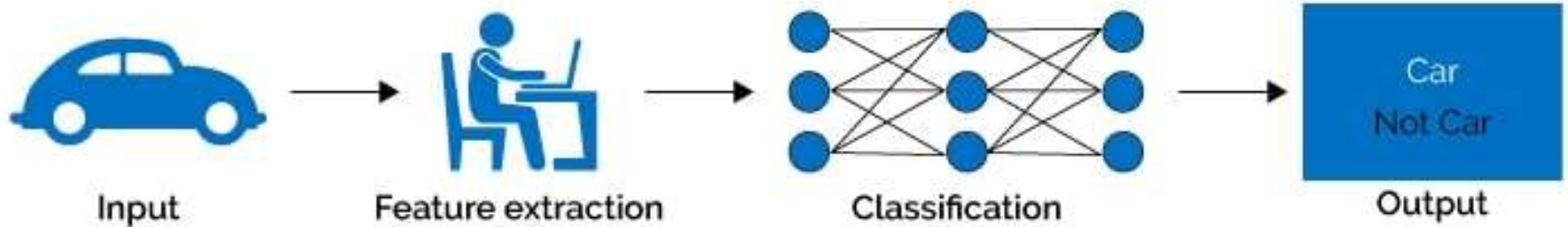
- A subset of ML. **Computational models inspired by the human brain**; used in ML to recognize complex patterns.

Deep Learning (DL)

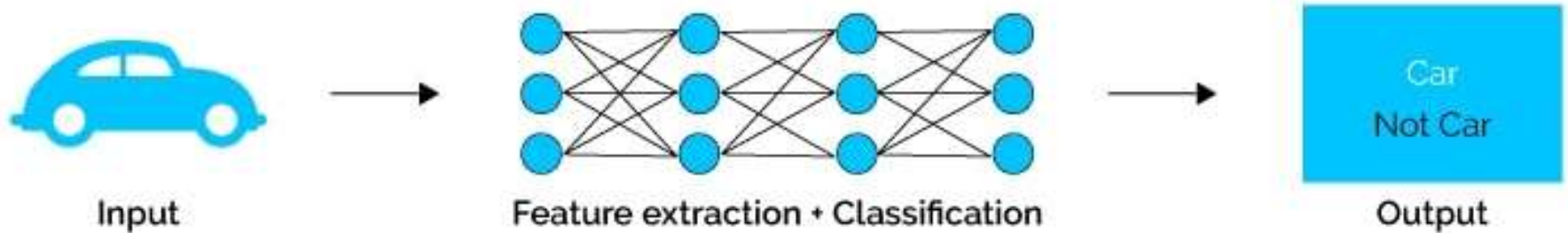
- A subset of NN using **neural networks with many layers** ("deep" architectures).



Machine Learning

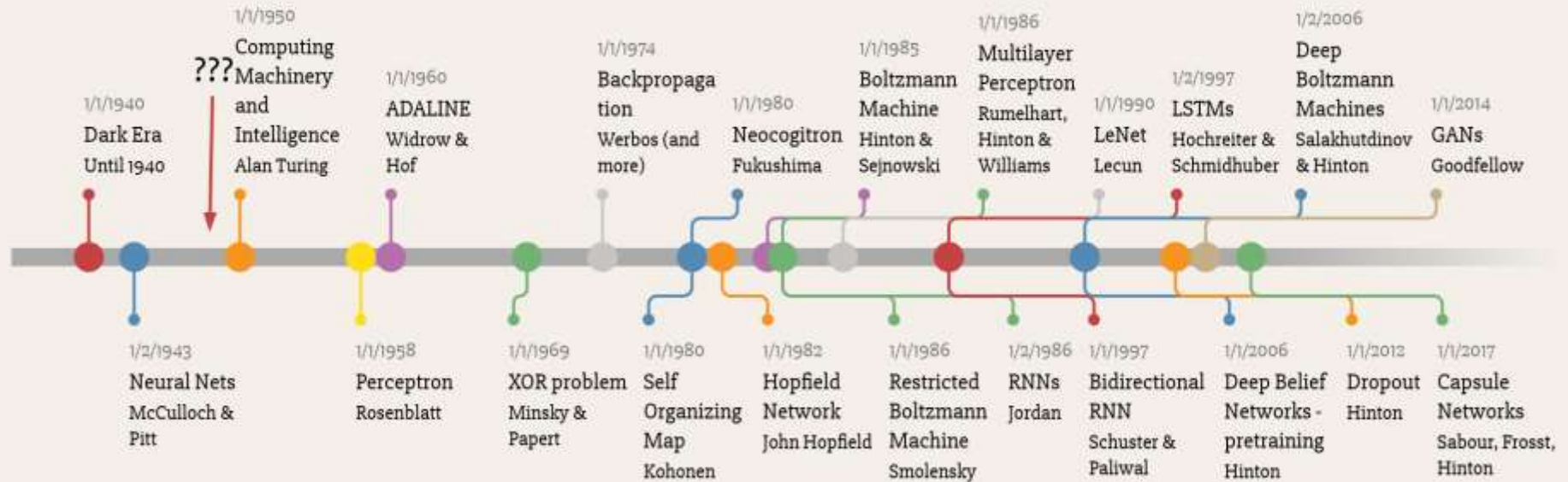


Deep Learning



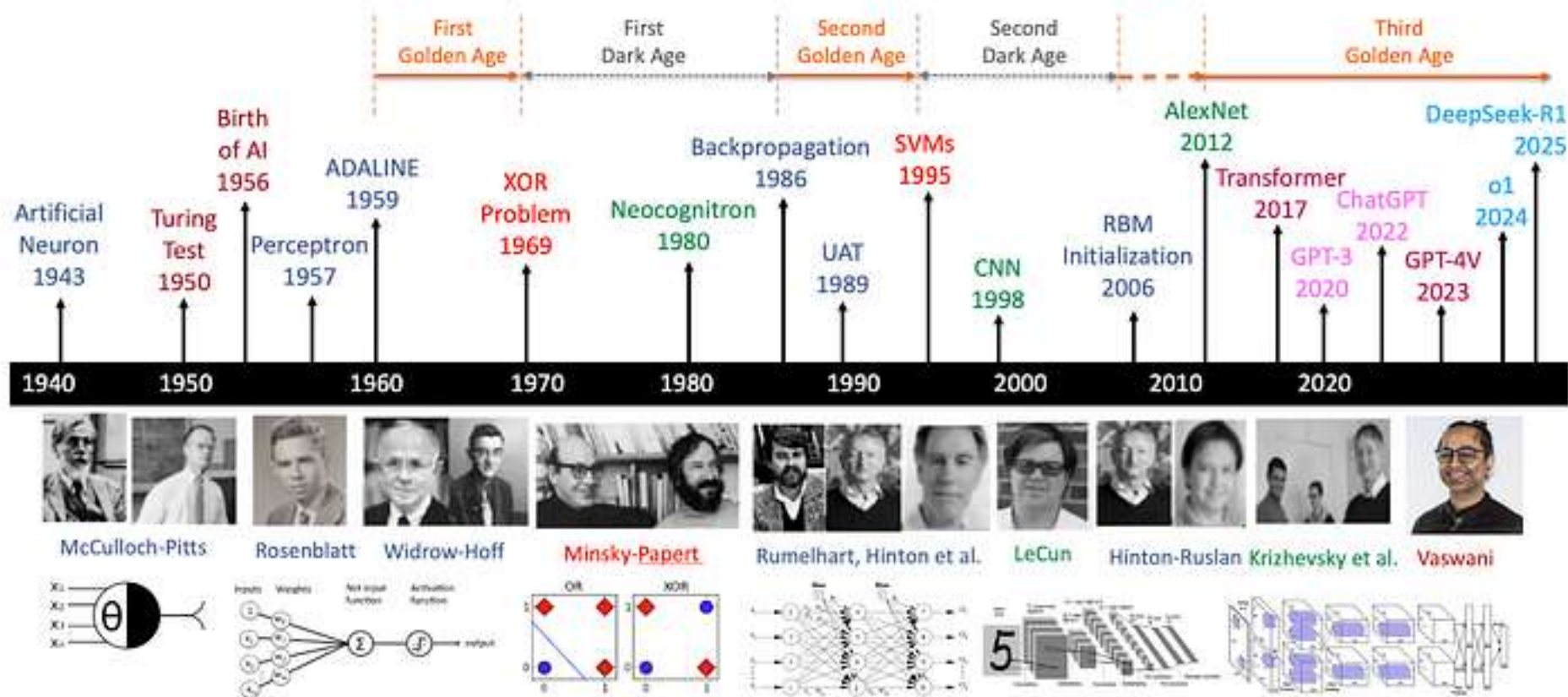
История развития нейронных сетей

Deep Learning Timeline



История развития нейронных сетей

A Brief History of AI with Deep Learning



McCulloch-Pitts Neuron

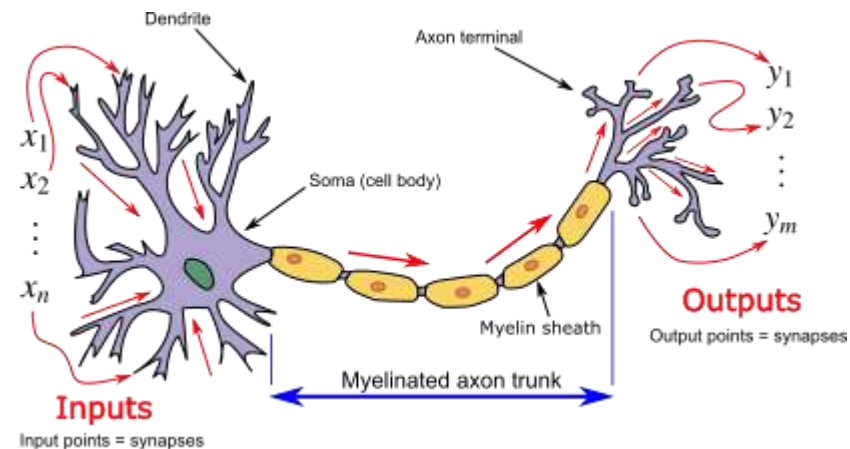
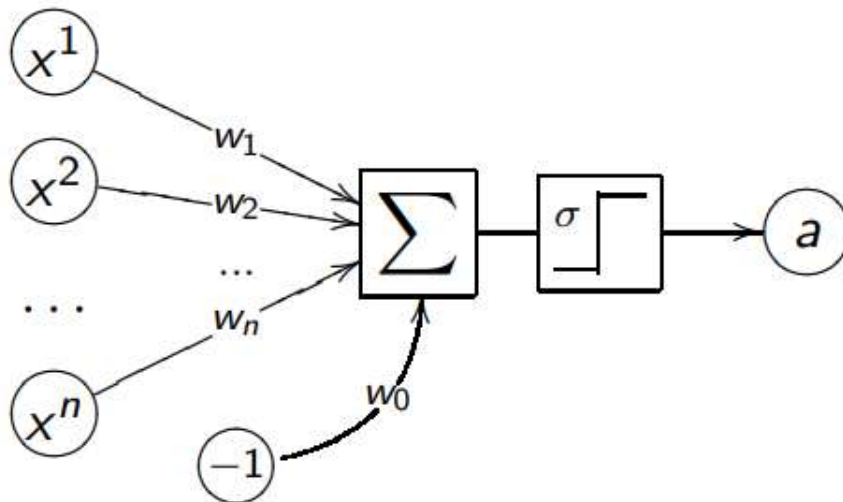
Линейная модель нейрона:

$$a(x, w) = \sigma(\langle w, x \rangle) = \sigma \left(\sum_{j=1}^n w_j f_j(x) - w_0 \right)$$

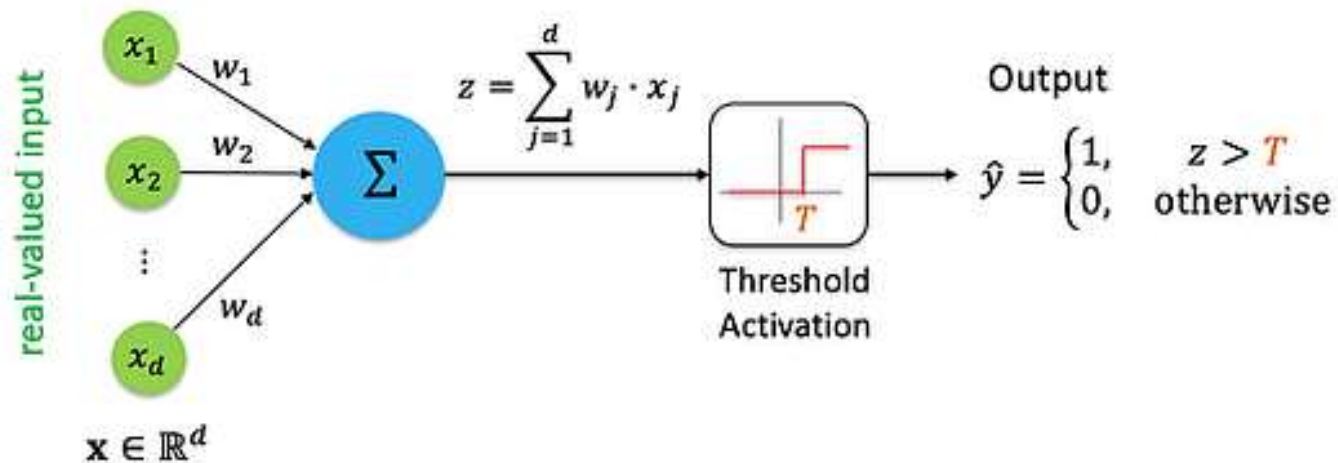
$\sigma(z)$ – функция активации

w_j – весовые коэффициенты синаптических связей

w_0 – порог активации

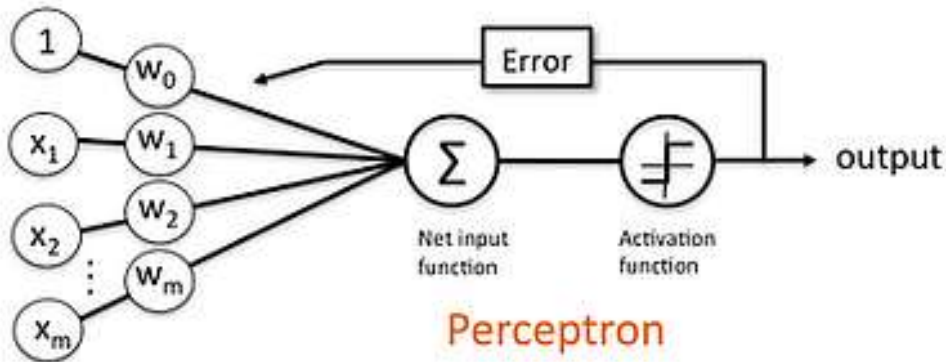


Rosenblatt's Perceptron Model

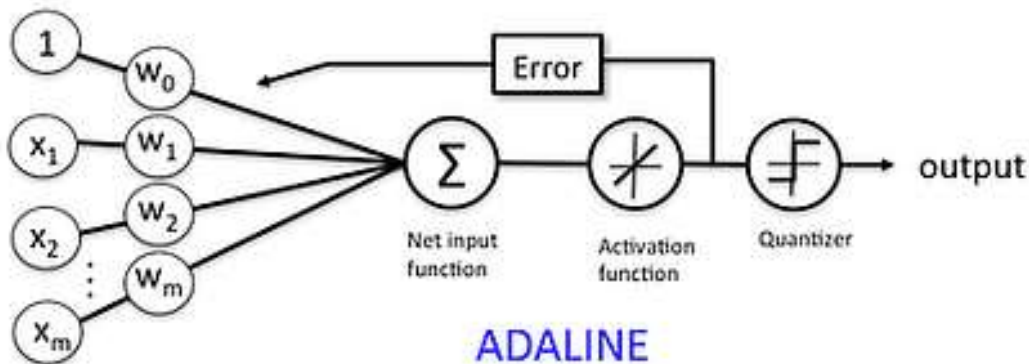


Frank Rosenblatt

ADALINE (Adaptive Linear Neuron aka Delta Learning Rule)



Bernard Widrow

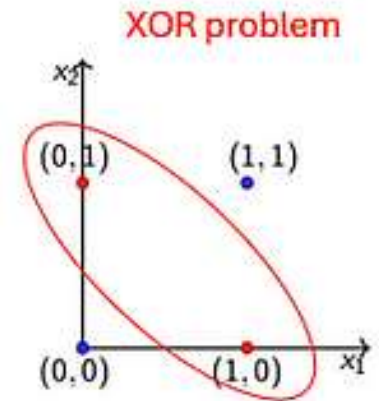
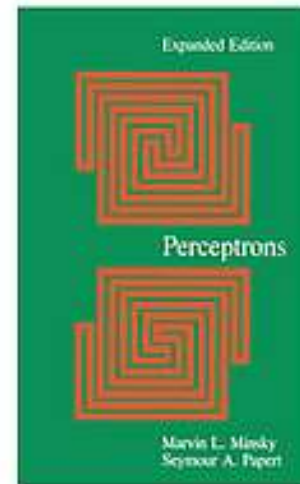


Marcian Hoff

The XOR Problem (1969)

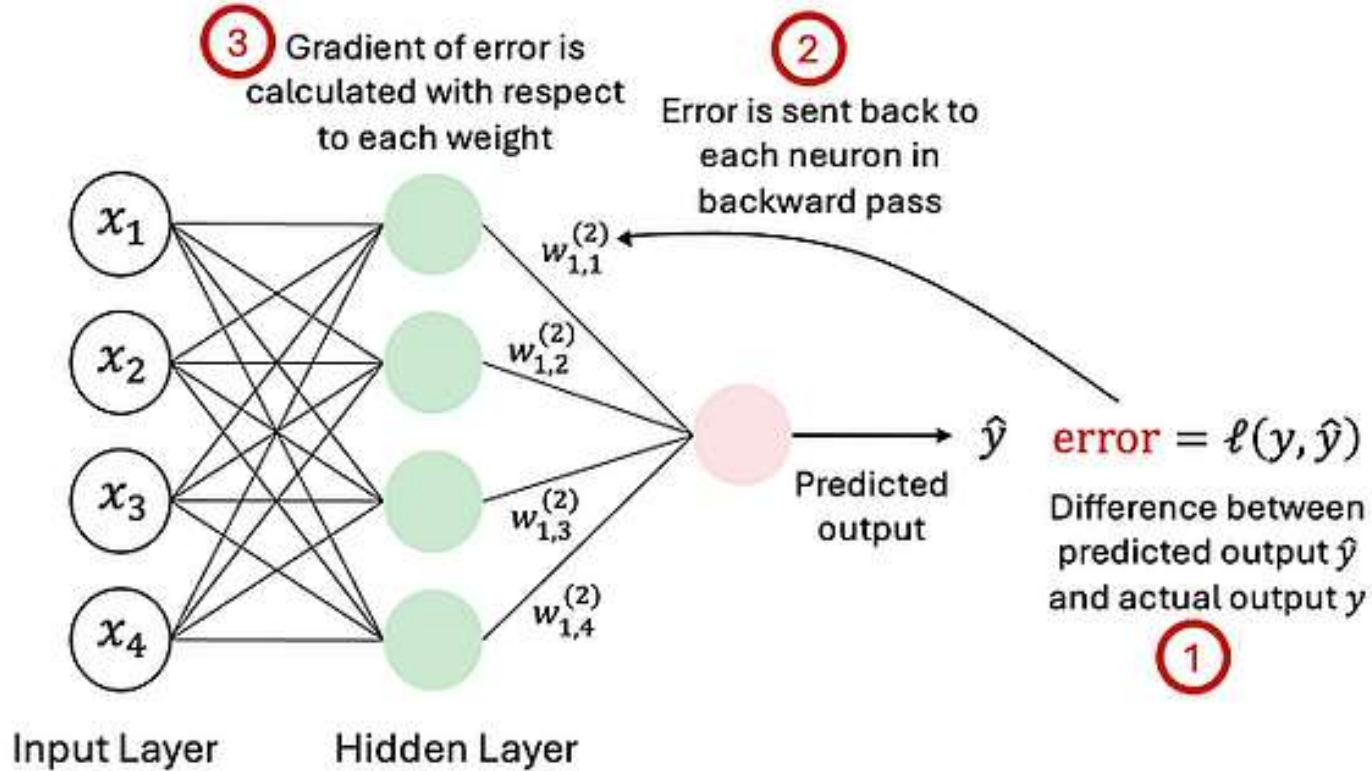


Minsky and Papert: Perceptrons: An introduction to computational geometry. MIT Press, 1969.



В книге “Perceptrons.” математически доказаны ограничения перцептронов. Было показано, что они принципиально не в состоянии выполнять многие из тех функций, которые хотели получить от перцептронов. К тому же, в то время была слабо развита теория о параллельных вычислениях, а перцептрон полностью соответствовал принципам таких вычислений.

Backpropagation (1970s-1980s)



David Rumelhart

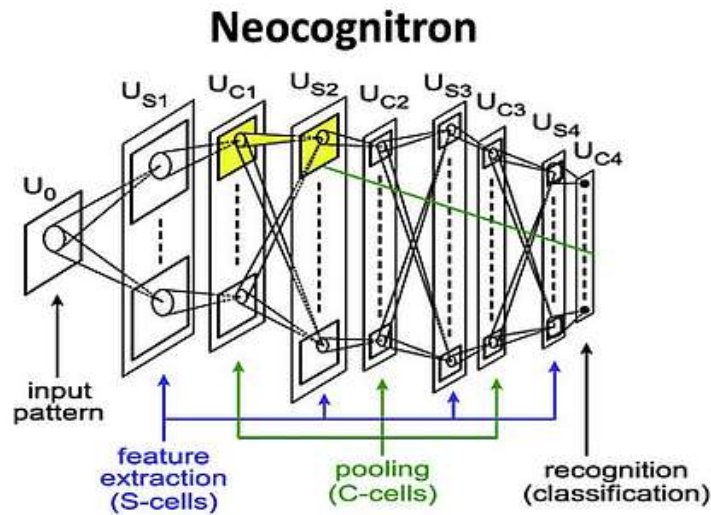


Geoffrey Hinton

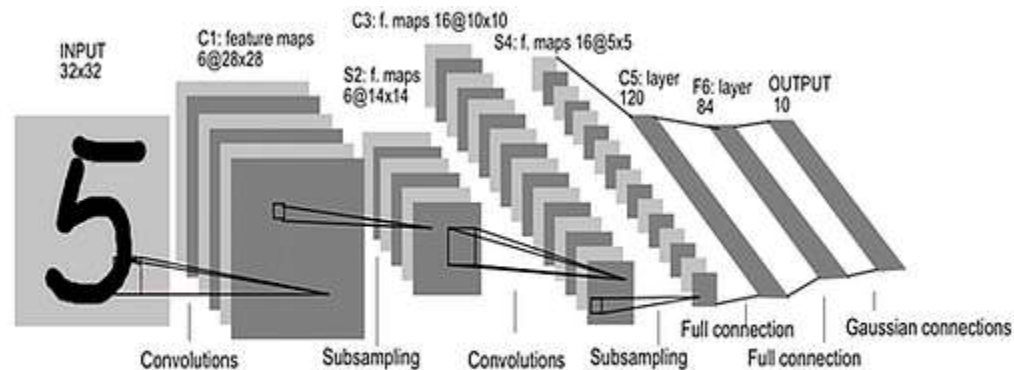


Ronald Williams

Convolutional Neural Networks (1980s — 2010s)



Kenji Fukushima

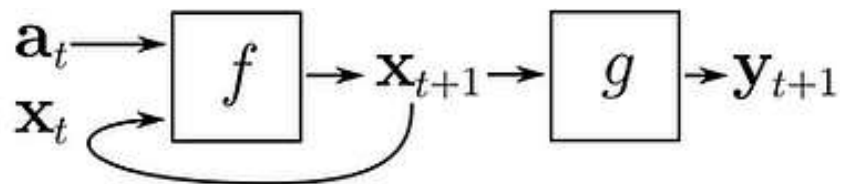
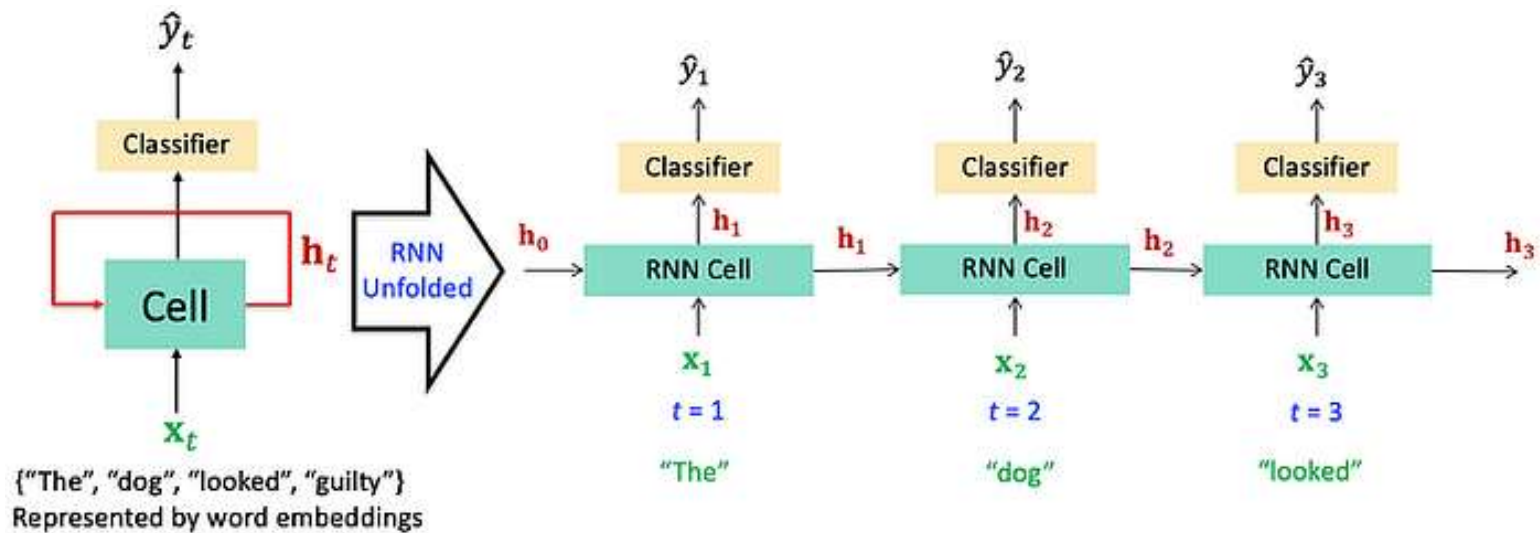


LeNet-5

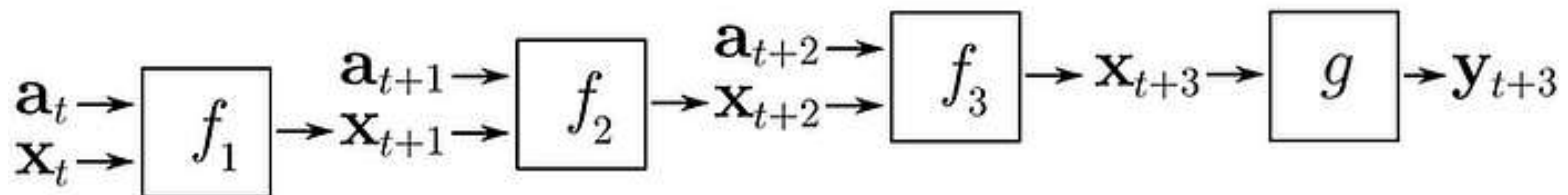


Yann LeCun

Recurrent Neural Networks (1986–2017)

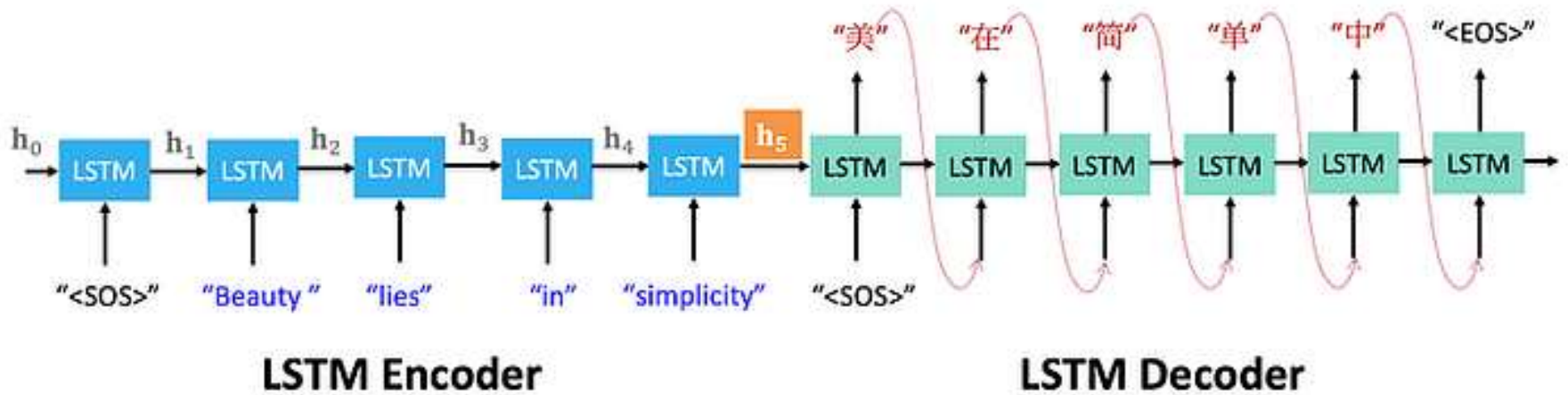
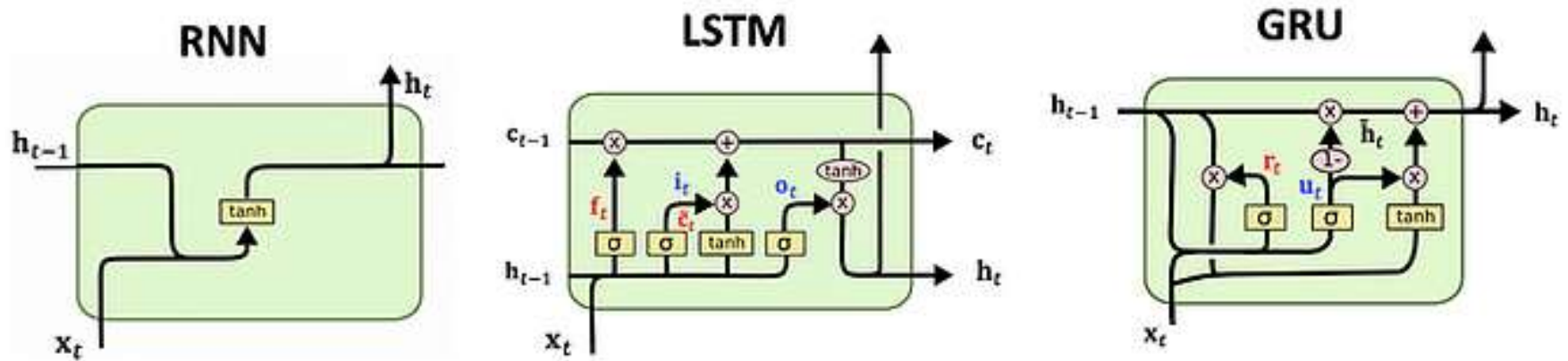


↓ unfold through time ↓



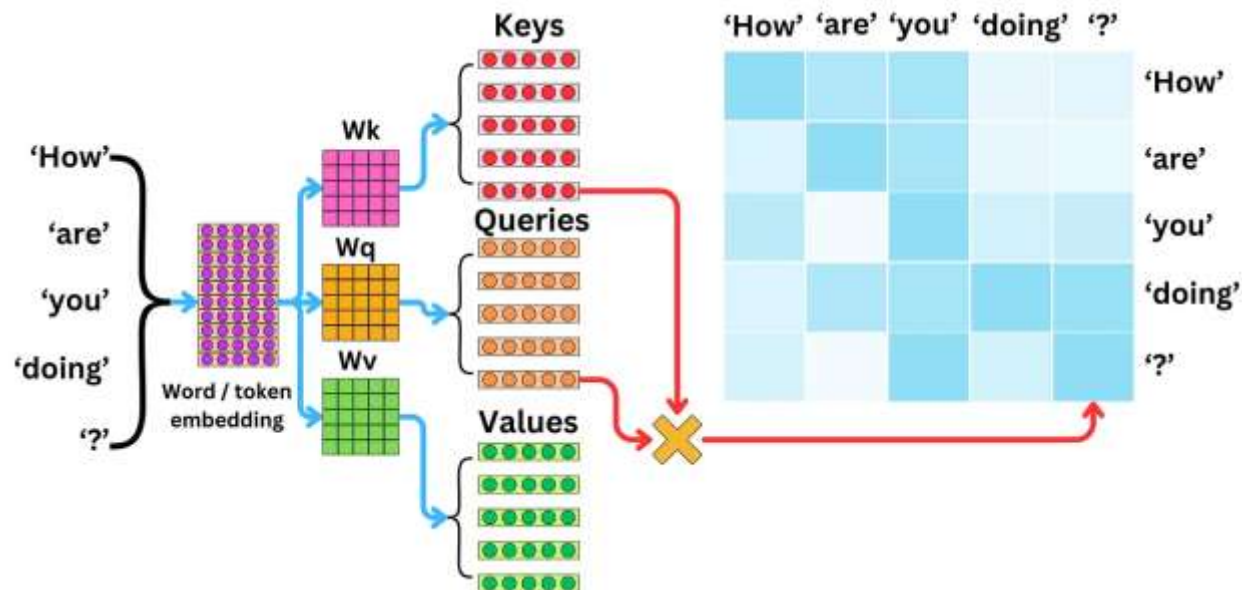
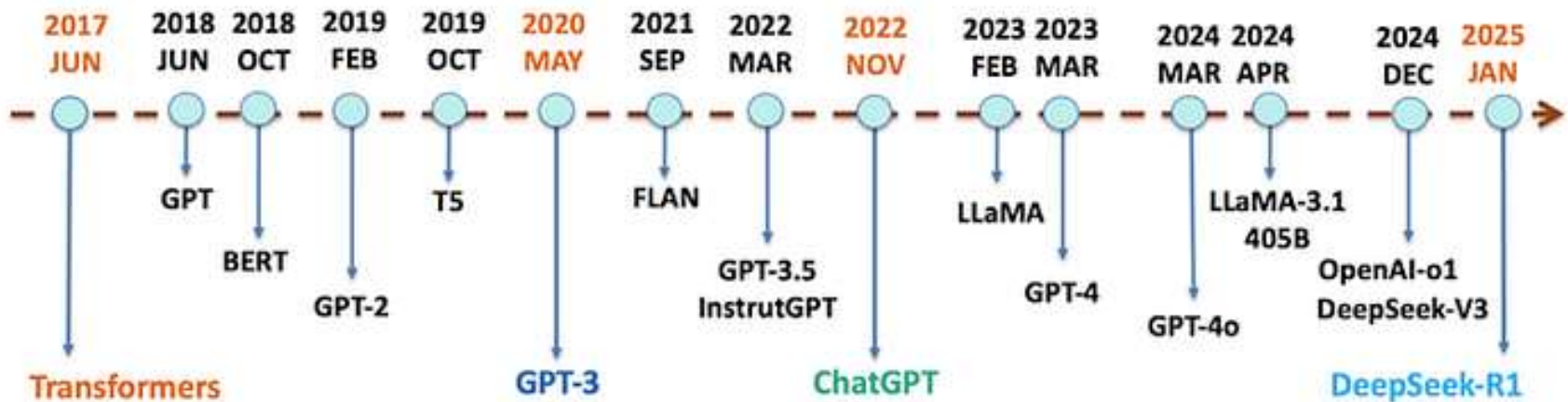
Recurrent Neural Networks

LSTM, GRU and Seq2Seq Models (1997-2014)

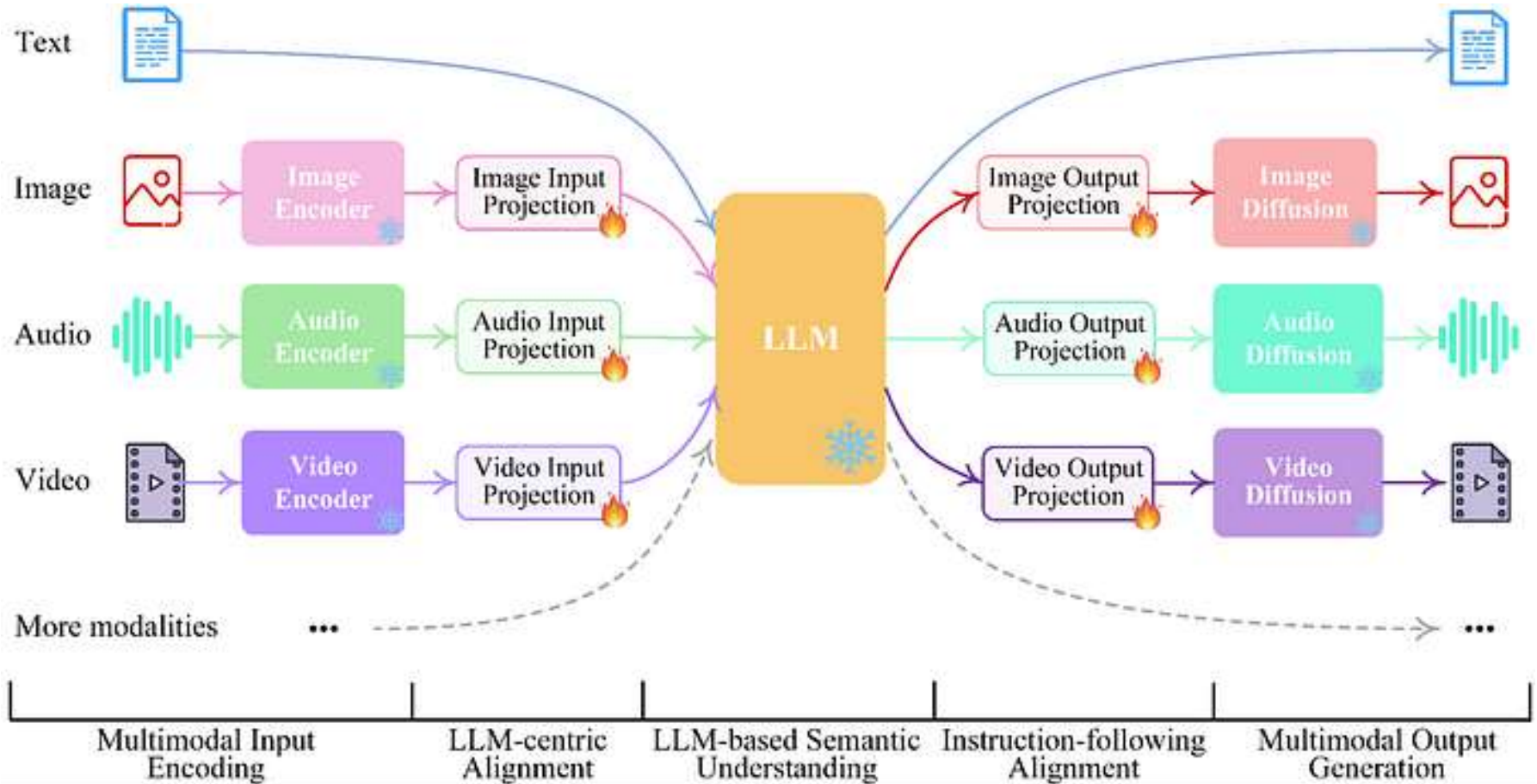


Transformers and the LLM Revolution

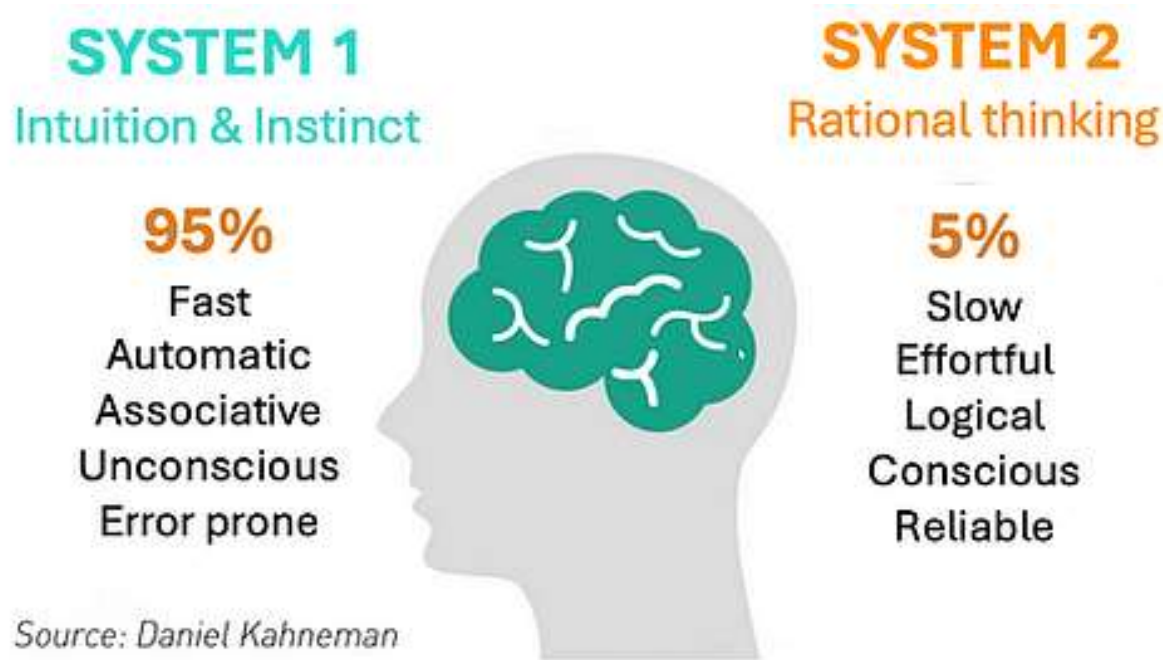
«Attention Is All You Need» (2017)



Multimodal Models



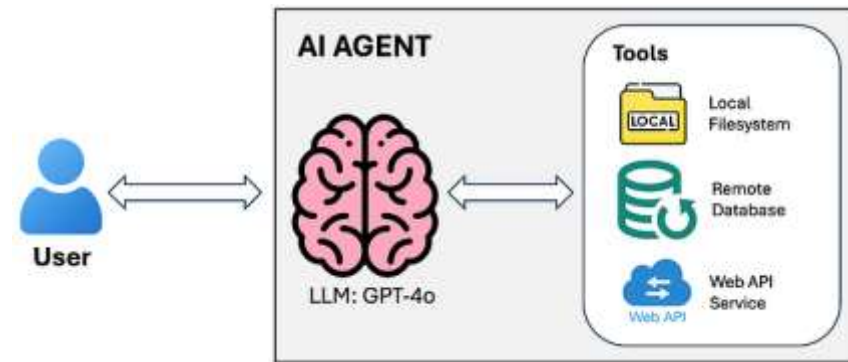
The Reasoning Revolution: From System 1 to System 2 (2024–2025)



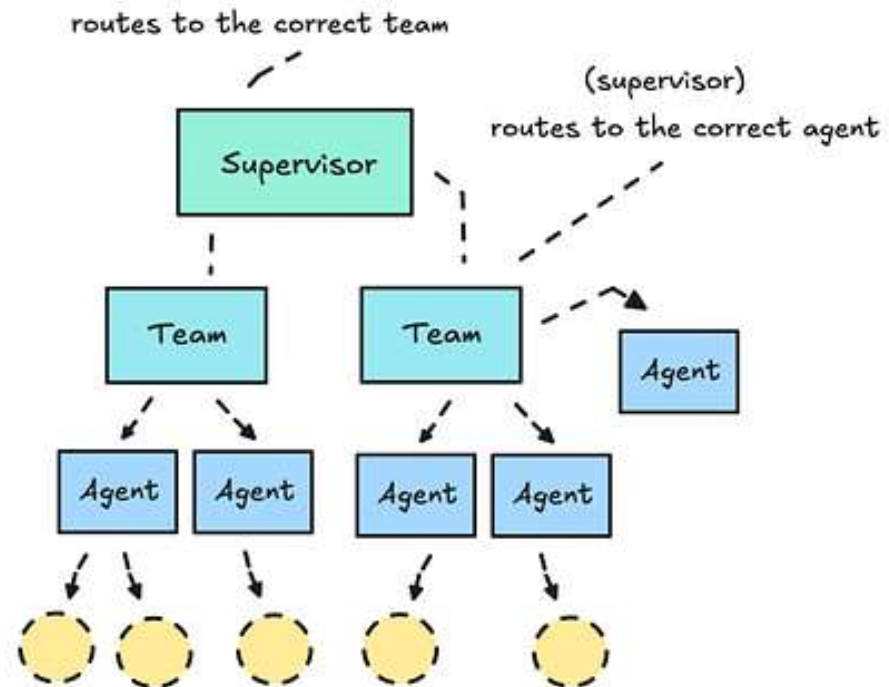
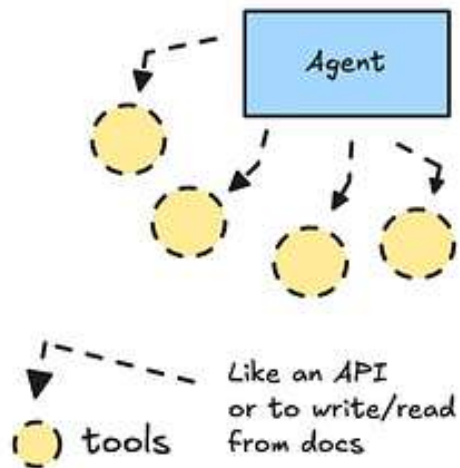
Акцент на улучшении способностей к рассуждению, выход за рамки простого распознавания образов и переход к структурированному аналитическому мышлению. Переход вдохновлён теорией двойного процесса в когнитивной психологии, различающей быстрые, интуитивные реакции (Система 1) и медленные, аналитические рассуждения (Система 2).

LLM-based Agentic AI

Создание автономных агентов, которые могут планировать, выполнять сложные задачи и взаимодействовать с внешними системами, используя LLM в качестве своей когнитивной основы. Эти агентные системы представляют собой переход от пассивного ответа на вопросы к проактивному решению проблем и выполнению задач.



Single vs Multiagent Systems



Нейронная сеть (ИНС, НС) – сложная дифференцируемая функция, задающая отображение из исходного признакового пространства в пространство ответов, все параметры которой могут настраиваться одновременно и взаимосвязано

Нейронная сеть – универсальный аппроксиматор функций

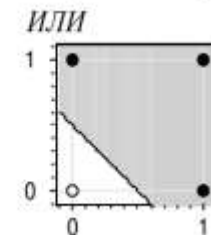
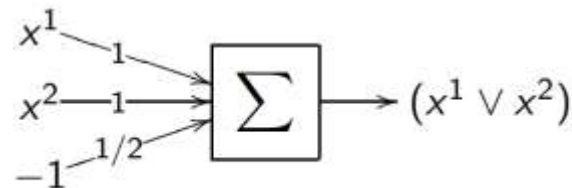
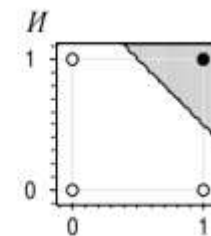
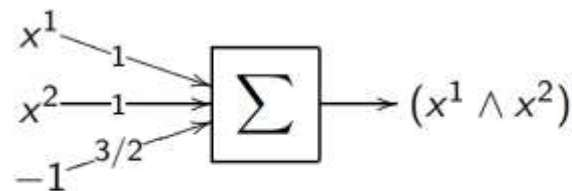
С помощью одного нейрона можно реализовать гиперплоскость

Функции И, ИЛИ, НЕ бинарных признаков x^1, x^2

$$x^1 \wedge x^2 = [x^1 + x^2 - \frac{3}{2} > 0];$$

$$x^1 \vee x^2 = [x^1 + x^2 - \frac{1}{2} > 0];$$

$$\neg x^1 = [-x^1 + \frac{1}{2} > 0];$$



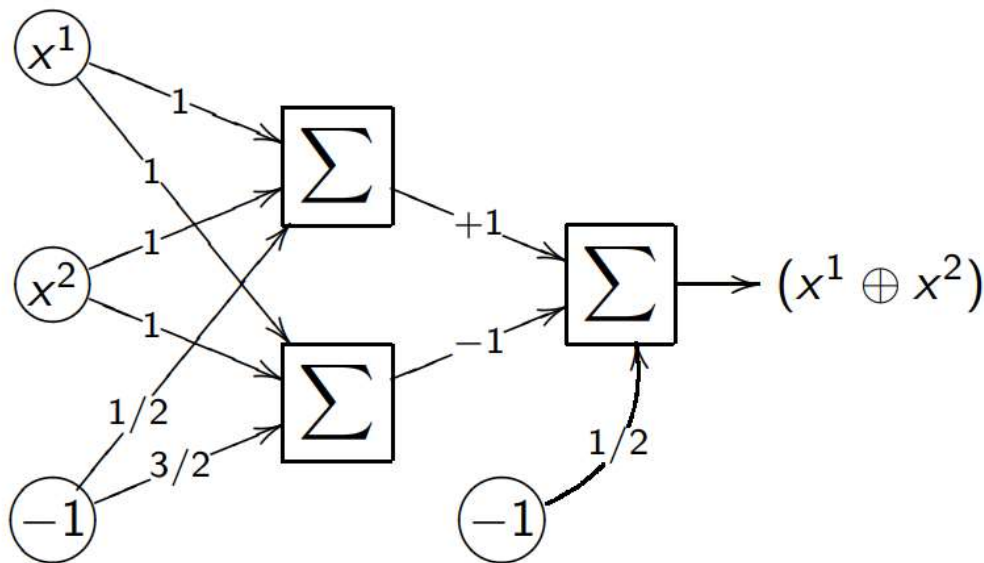
Например, функция XOR не реализуема одним нейроном, но может быть реализована двумя способами:

- Добавлением нелинейного признака:

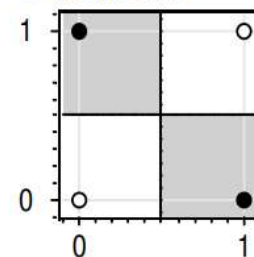
$$x^1 \oplus x^2 = [x^1 + x^2 - 2x^1x^2 - \frac{1}{2} > 0]$$

- Построить нейронную сеть (многослойную суперпозицию) функций И, ИЛИ:

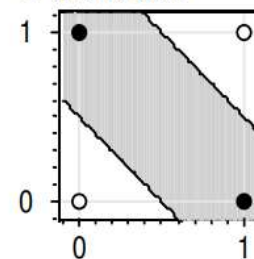
$$x^1 \oplus x^2 = [(x^1 \vee x^2) - (x^1 \wedge x^2) - \frac{1}{2} > 0]$$



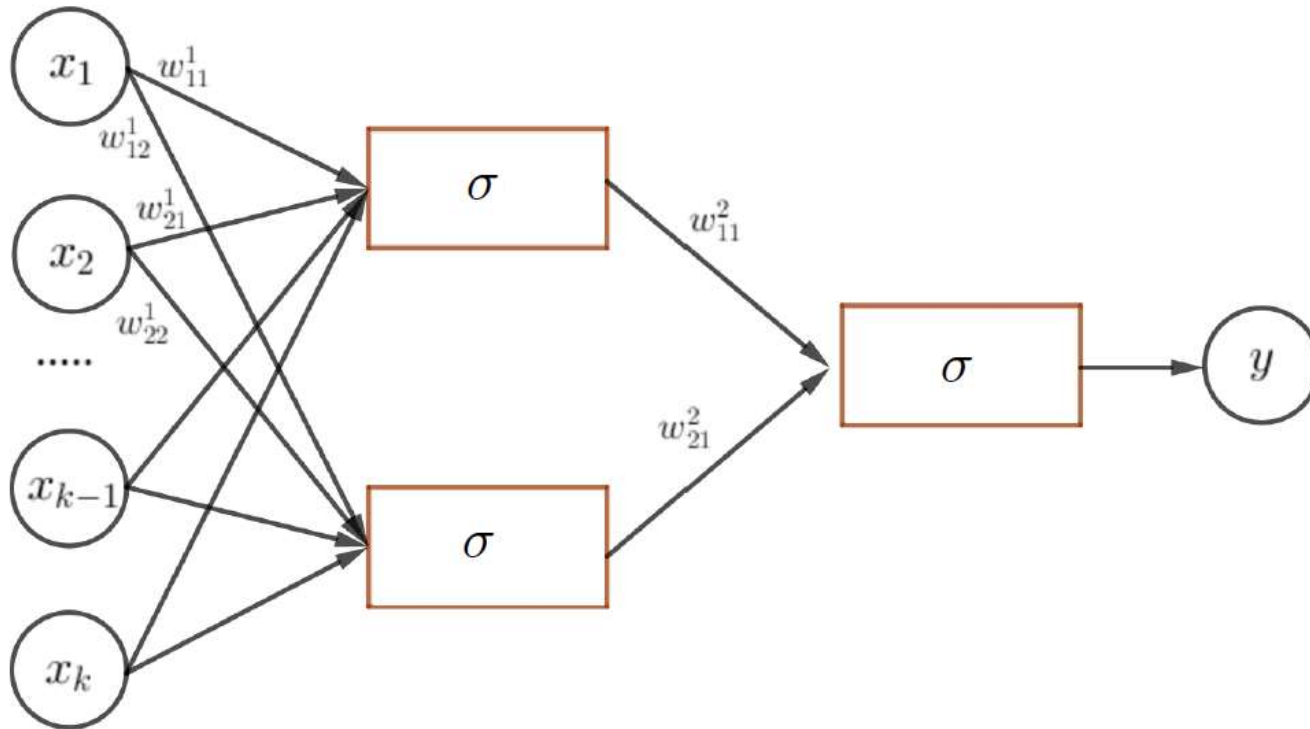
1-й способ



2-й способ



MLP (multi-layer-perceptron)



$$h = \sigma(X \cdot W)$$

$$y = \sigma(h \cdot W)$$

Универсальная теорема аппроксимации

Если $\sigma(z)$ - непрерывная сигмоида, то для любой непрерывной на $[0,1]^n$ функции $f(x)$ существуют такие значения параметров $H, \alpha_h \in R, w_h \in R^n, w_0 \in R$, что двухслойная сеть

$$a(x) = \sum_{h=1}^H \alpha_h \sigma(\langle x, w_h \rangle - w_0)$$

равномерно приближает $f(x)$ с любой точностью ε :
 $|a(x) - f(x)| < \varepsilon$, для всех $x \in [0,1]^n$

Искусственная нейронная сеть прямой связи с одним скрытым слоем может аппроксимировать любую непрерывную функцию многих переменных с любой точностью при условии достаточного количества нейронов и подбора оптимального вектора весов

- Линейный слой (linear layer, dense layer) – линейное преобразование над входящими данными. Его обучаемые параметры – матрица весов W и вектор w_0 . Слой преобразует d -мерные векторы в k -мерные:

$$x \rightarrow xW + w_0, W \in R^{d \times k}, x \in R^d, w_0 \in R^k$$

- Функция активации (activation function) – нелинейное преобразование, поэлементно применяющееся к пришедшим на вход данным. Благодаря функциям активации НС способны порождать более информативные признаковые описания, преобразуя данные нелинейным образом



Epoch
000,100

Learning rate

0.03

Activation

Linear

Regularization

None

Regularization rate

0

Problem type

Classification

DATA

Which dataset do you want to use?



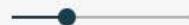
Ratio of training to test data: 50%



Noise: 0



Batch size: 10



REGENERATE

FEATURES

Which properties do you want to feed in?



+ - 1 HIDDEN LAYER

+ -

2 neurons

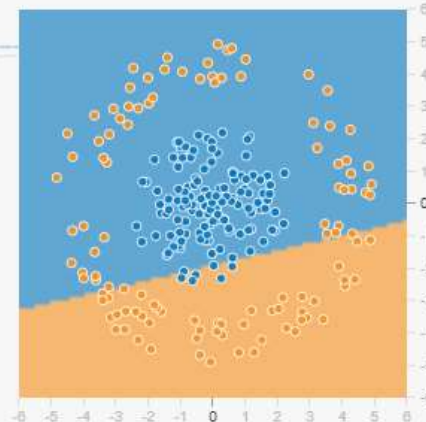


This is the output from one neuron. Hover to see it larger.

OUTPUT

Test loss 0.552

Training loss 0.483



Colors shows data, neuron and weight values.



☐ Show test data

☒ Discretize output



Epoch
001,000

Learning rate
0.03

Activation
Linear

Regularization
None

Regularization rate
0

Problem type
Classification

DATA

Which dataset do you want to use?



Ratio of training to test data: 50%



Noise: 0



Batch size: 10



REGENERATE

FEATURES

Which properties do you want to feed in?



+ - 1 HIDDEN LAYER

+ -

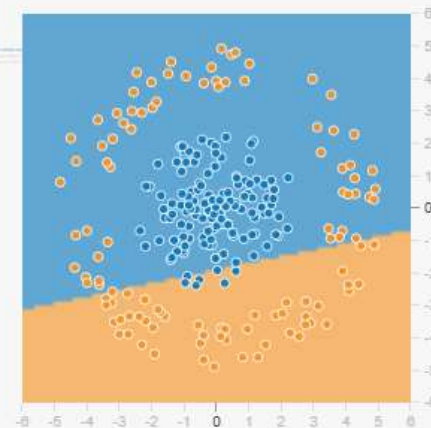
3 neurons



This is the output from one neuron. Hover to see it larger.

OUTPUT

Test loss 0.551
Training loss 0.483



Colors shows data, neuron and weight values.



☐ Show test data ☒ Discretize output



Epoch
001,000

Learning rate
0.03

Activation
Sigmoid

Regularization
None

Regularization rate
0

Problem type
Classification

DATA

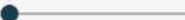
Which dataset do you want to use?



Ratio of training to test data: 50%



Noise: 0



Batch size: 10



REGENERATE

FEATURES

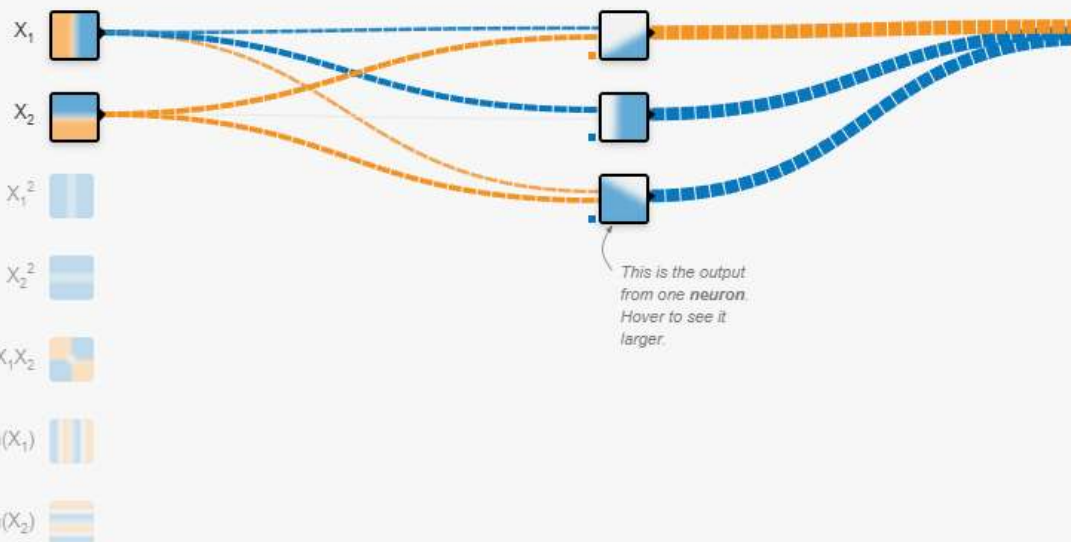
Which properties do you want to feed in?

- X_1
- X_2
- X_1^2
- X_2^2
- $X_1 X_2$
- $\sin(X_1)$
- $\sin(X_2)$

+ - 1 HIDDEN LAYER

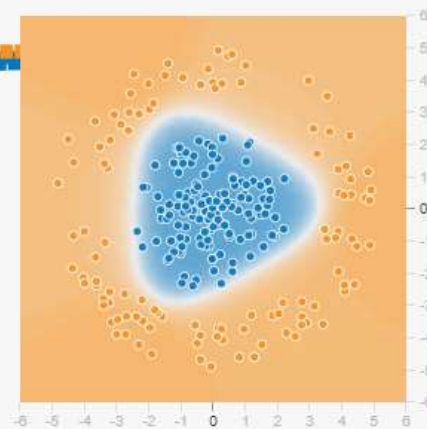
+ -

3 neurons



OUTPUT

Test loss 0.018
Training loss 0.015



Colors shows data, neuron and weight values.



☐ Show test data

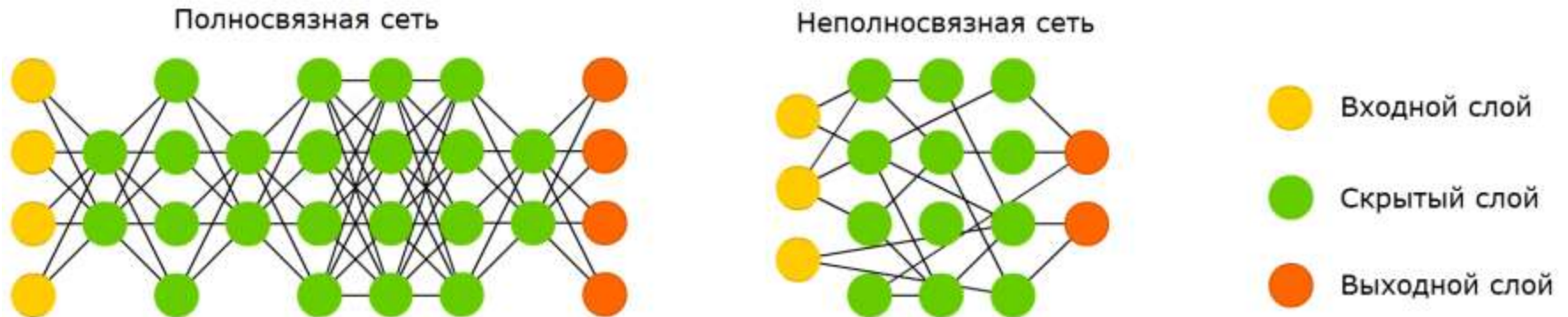
☐ Discretize output

Входной слой (input layer) – вход нейросети, который получает исходные данные в виде матрицы «объекты-признаки»

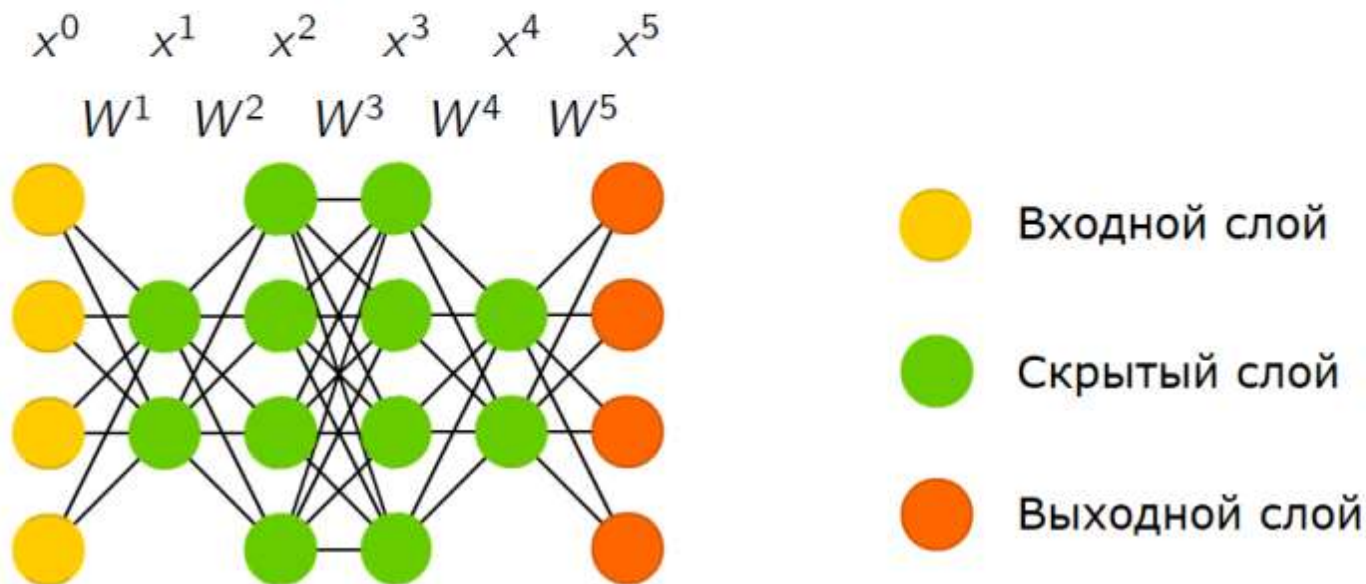
Скрытый слой (hidden layer) – преобразовывают исходные данные в промежуточные (внутренние, скрытые) представления

Выходной слой (output layer) – финальное преобразование промежуточного представления в пространство ответов

Полносвязная НС (fully connected, multilayer perceptron, MLP с одним слоем) – нейросеть, в которой есть только линейные слои и различные функции активации.



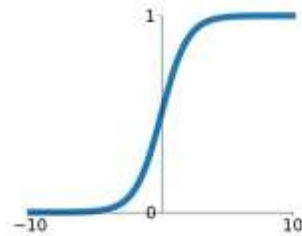
Архитектура сети: H_l — число нейронов в l -м слое, $l = 1, \dots, L$
 $x^0 = x = (f_j(x))_{j=0}^n$ — вектор признаков на входе сети, $H_0 = n$
 $x^l = (x_h^l)_{h=0}^{H_l}$ — вектор признаков на выходе l -го слоя, $x_0^l = -1$
 $x^L = a(x) = (a_m(x))_{m=1}^M$ — выходной вектор сети, $H_L = M$
 $W^l = (w_{kh}^l)$ — матрица весов l -го слоя, размера $(H_{l-1} + 1) \times H_l$



Наиболее популярные функции активации

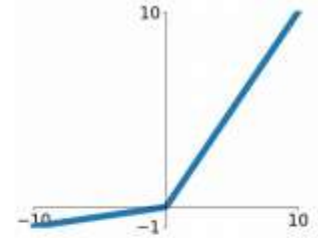
Sigmoid

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



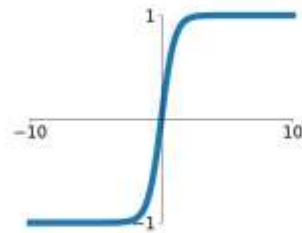
Leaky ReLU

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



tanh

$$\tanh(x)$$

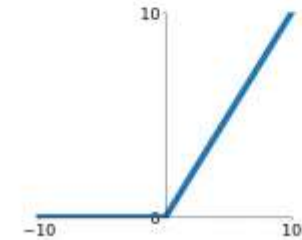


Maxout

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

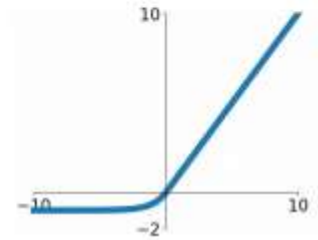
ReLU

$$\max(0, x)$$



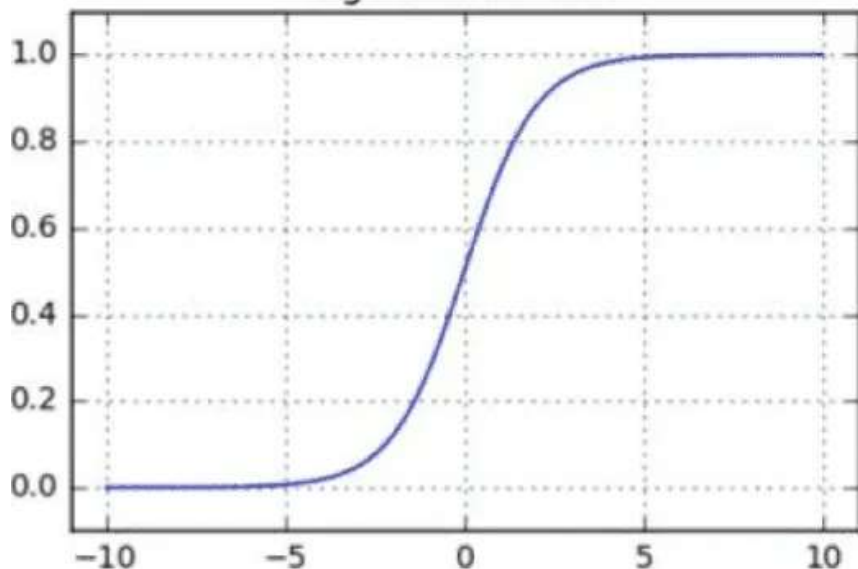
ELU

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

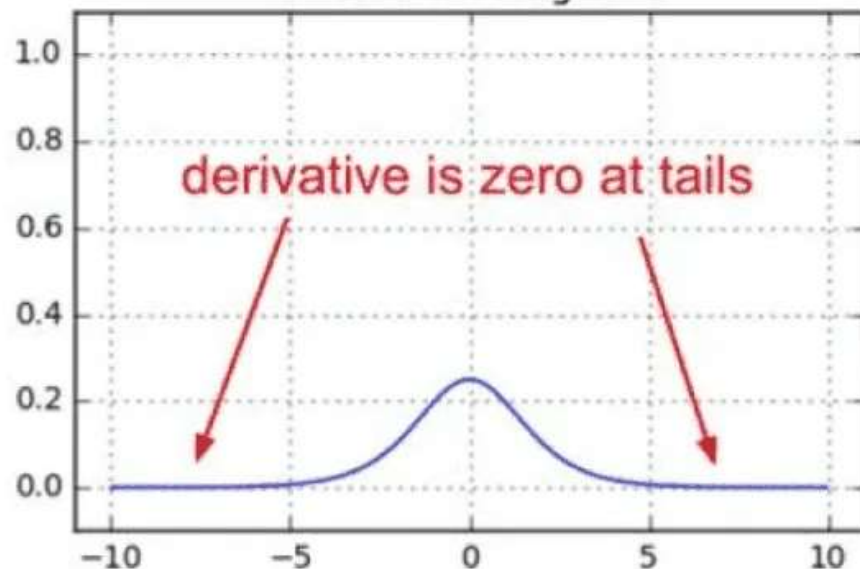


- Затухание градиента (vanishing gradient, паралич сети) – значение градиента настолько малое (приближается к нулю), что веса практически не изменяются и обучение НС фактически останавливается
- Признаки:
 - точность модели растёт медленно или вообще не растёт
 - веса модели экспоненциально уменьшаются во время обучения либо стремятся к нулю

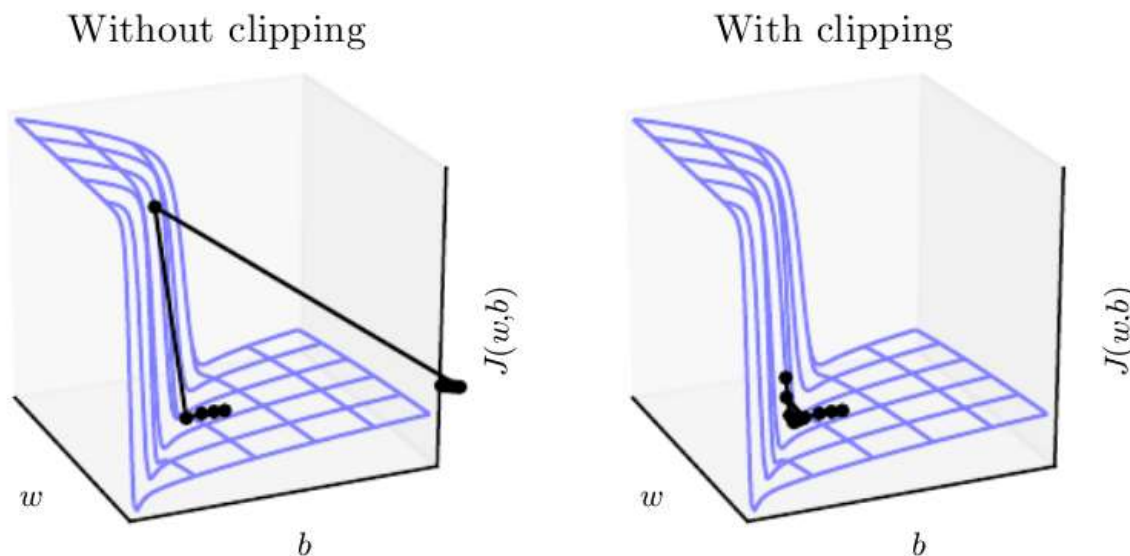
sigmoid function



derivative of sigmoid



- Взрыв градиента (gradient exploding) – «взрывной рост» градиента
- Признаки:
 - модель плохо обучается на данных, функция потерь имеет высокие значения
 - функция потерь меняется скачкообразно
 - веса модели растут экспоненциально, принимают значение NaN
- Для решения проблемы можно применить эвристику Gradient Clipping: если $\|g\| > \theta$, то $g := g\theta/\|g\|$

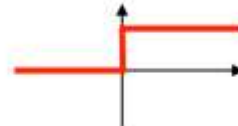
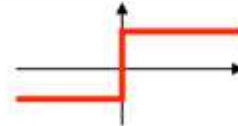




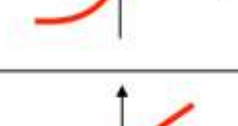
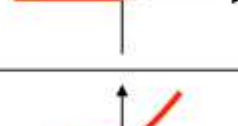


- Rectified linear unit (ReLU):
 - + простота вычисления активации и производной, достаточно сравнить значение производной с нулем, за счет этого быстрее, чем сигмоида
 - область значений является смещенной относительно нуля
 - для отрицательных значений производная равна нулю, что может привести к затуханию градиента
- Leaky ReLU и Parametric ReLU (PReLU) - позволяет получить более симметричную относительно нуля область значений за счет гиперпараметра α , обеспечивающего наклон слева от нуля
- Exponential ReLU (ELU) – гладкая аппроксимация ReLU, но так как надо считать экспоненту, редко используется на практике
- Sigmoid – исторически одна из первых функций активации, рассматривалась как гладкая аппроксимация пороговой функции, эмулирующая активацию естественного нейрона. На практике используется в задачах бинарной классификации
 - область значений смещена относительно нуля
 - на хвостах обладает практически нулевой производной, что может привести к затуханию градиента
 - максимальное значение производной составляет 0.25, что так же приводит к затуханию градиента









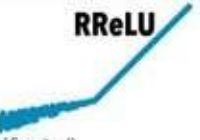
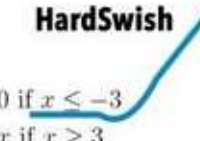
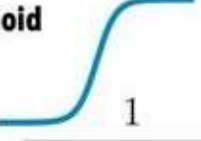
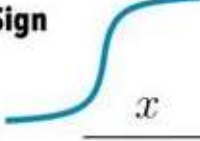

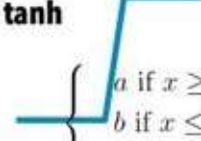
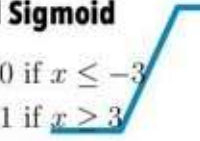


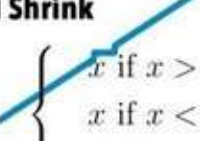
- Гиперболический тангенс, Tanh:
- + имеет ограниченную область значений, как и сигмоида
- + эта область значений симметрична, в отличие от сигмоиды
- требуется вычисление экспоненты, что является достаточно сложной вычислительной операцией
- на хвостах производная близка к нулю, что может привести к затуханию градиента

Как выбрать функцию активации?

- Для задач классификации - Sigmoid (бинарная) или Softmax (многоклассовая), если хотим получить вероятности классов в качестве выходных данных
- Для задач регрессии - используйте ReLU или его модификации, такие как LeakyReLU или ELU. Эти функции обычно дают лучшую производительность в задачах регрессии
- Для моделей глубокого обучения, ReLU является общим выбором для скрытых слоев, так как она может ускорить обучение, но можно также использовать другие функции, например, PReLU
- Для рекуррентных нейронных сетей, обычно используются функции активации Tanh.
- Если не уверены, какую функцию активации использовать – начинаем с ReLU, так как она быстро считается, затем подбираем в процессе валидации. Не забываем оценивать значения градиента и отслеживать взрыв или затухание, а так же значение функции потерь

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

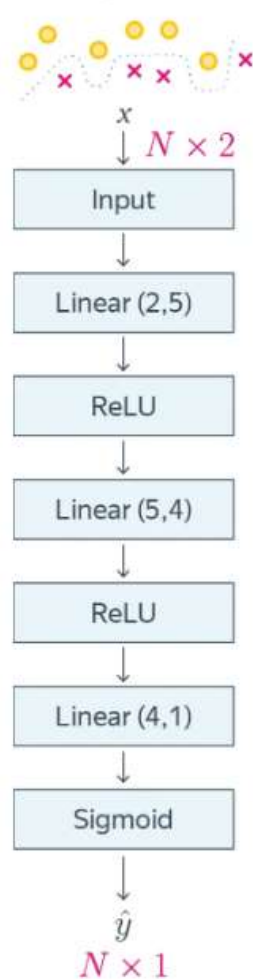
Neural Network Activation Functions: a small subset!

ReLU  $\max(0, x)$	GELU  $\frac{1}{2} \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + ax^3) \right) \right)$	PReLU  $\max(0, x)$
ELU  $\begin{cases} x & \text{if } x > 0 \\ \alpha(x \exp x - 1) & \text{if } x < 0 \end{cases}$	Swish  $\frac{x}{1 + \exp -x}$	SELU  $\alpha(\max(0, x) + \min(0, \beta(\exp x - 1)))$
SoftPlus  $\frac{1}{\beta} \log(1 + \exp(\beta x))$	Mish  $x \tanh \left(\frac{1}{\beta} \log(1 + \exp(\beta x)) \right)$	RReLU  $\begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \text{ with } a \sim \mathcal{R}(l, u) \end{cases}$
HardSwish  $\begin{cases} 0 & \text{if } x \leq -3 \\ x & \text{if } x \geq 3 \\ x(x+3)/6 & \text{otherwise} \end{cases}$	Sigmoid  $\frac{1}{1 + \exp(-x)}$	SoftSign  $\frac{x}{1 + x }$
Tanh  $\tanh(x)$	Hard tanh  $\begin{cases} a & \text{if } x \geq a \\ b & \text{if } x \leq b \\ x & \text{otherwise} \end{cases}$	Hard Sigmoid  $\begin{cases} 0 & \text{if } x \leq -3 \\ 1 & \text{if } x \geq 3 \\ x/6 + 1/2 & \text{otherwise} \end{cases}$
Tanh Shrink  $x - \tanh(x)$	Soft Shrink  $\begin{cases} x - \lambda & \text{if } x > \lambda \\ x + \lambda & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$	Hard Shrink  $\begin{cases} x & \text{if } x > \lambda \\ x & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$

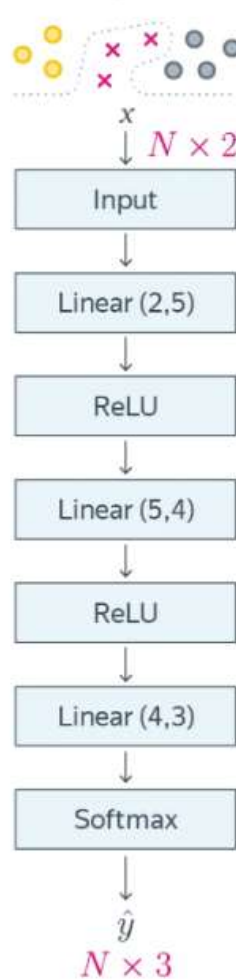
Почему Deep Learning?

- Архитектура НС – структура слоев и связей между ними, позволяющая наделять сеть нужными свойствами

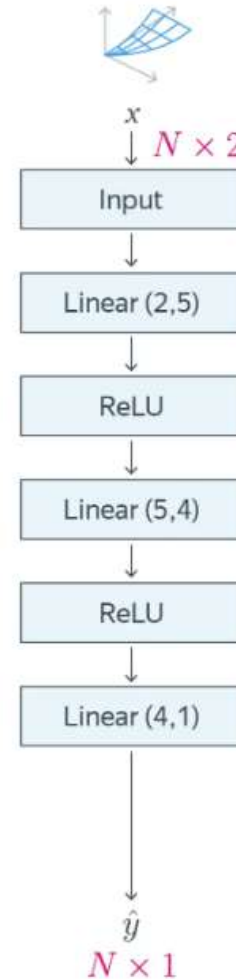
Бинарная классификация



Многоклассовая классификация

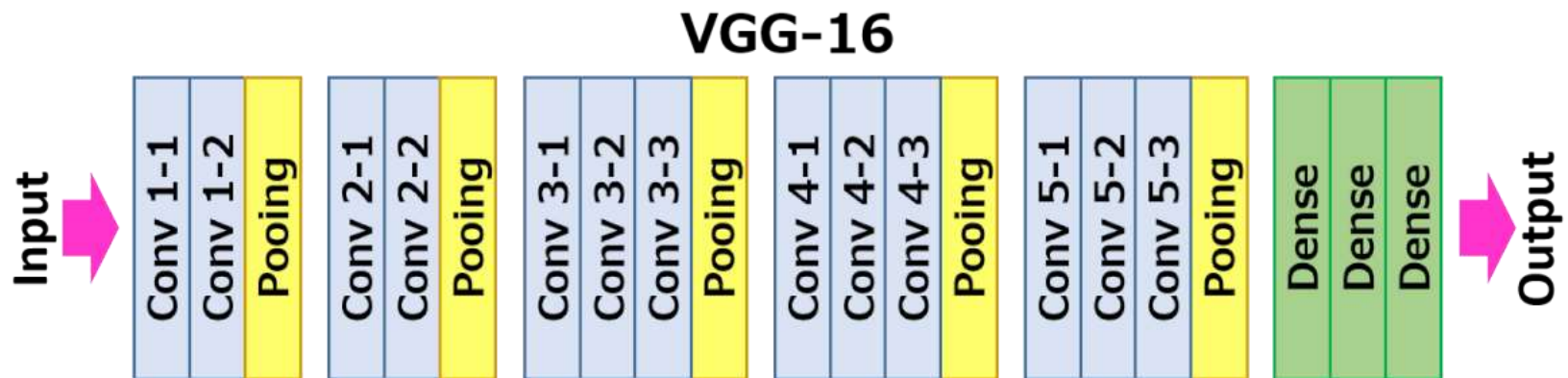


Регрессия

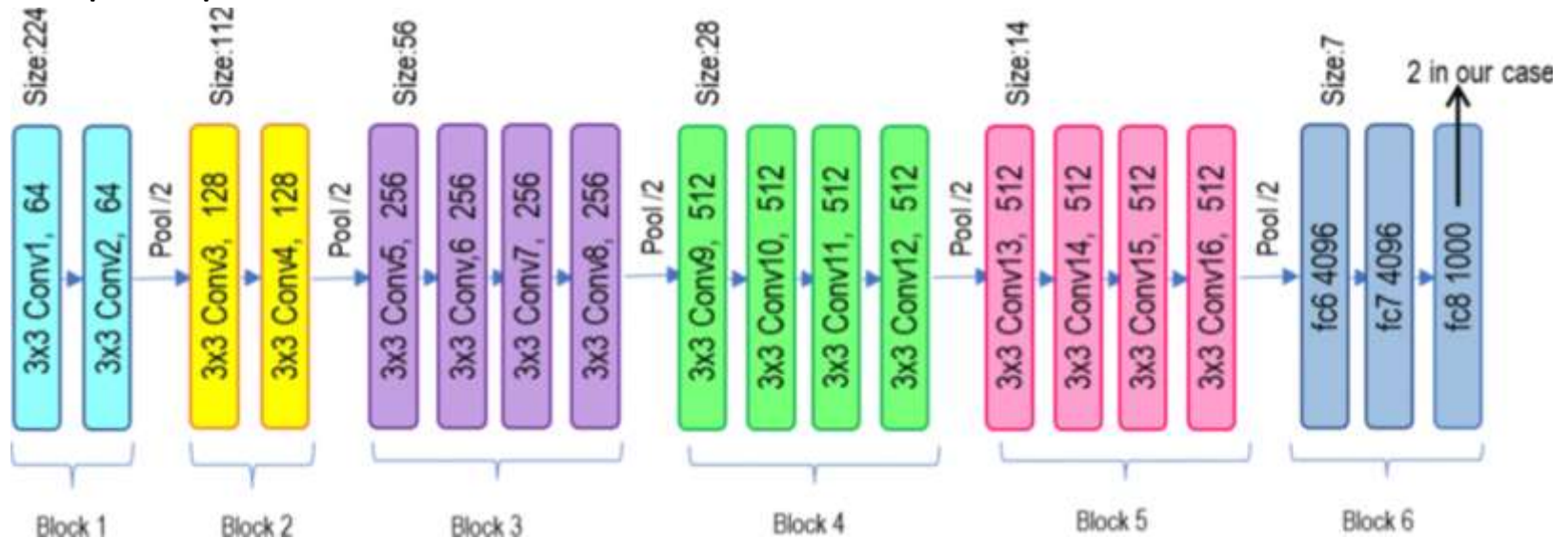


Почему Deep Learning?

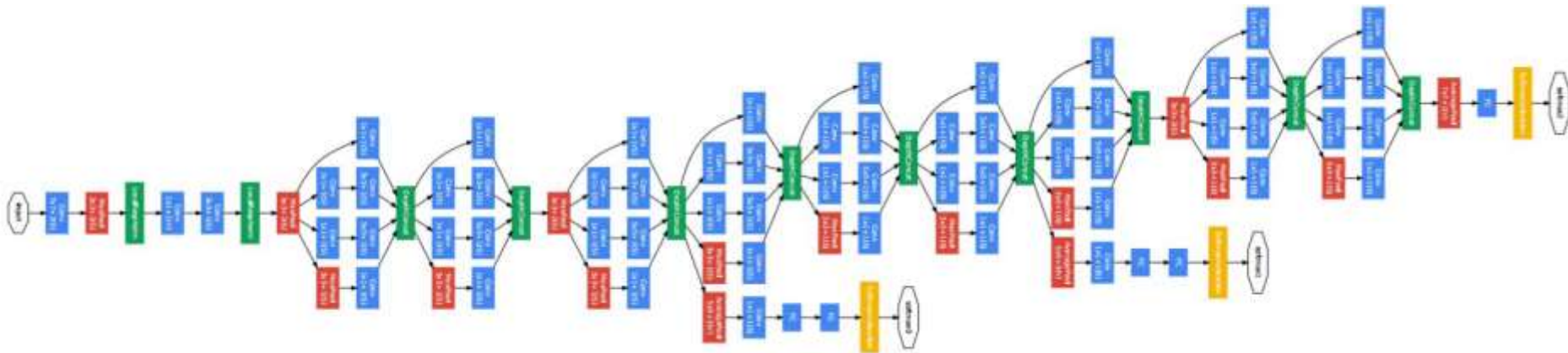
- Глубокие НС – это сети с несколькими слоями между входным и выходным слоем. Чем больше слоев, тем глубже сеть. Иногда говорят, что НС с более чем 3 скрытыми слоями уже может считаться глубокой, но общепринятого мнения нет
- Глубина (L слоев) важнее ширины (N нейронов в каждом слое)
- Глубокие НС позволяют принимать на входе и генерировать на выходе сложно структурированные данные
- Пример: VGG16 – одна из самых известных НС, топ-5 при тестировании на ImageNet (14 миллионов изображений из 1000 классов) с точностью 92.7% на соревновании ILSVRC-2014



- Пример: VGG19



- Пример: GoogLeNet (Inception-v1) – SotA на ILSVRC 2014

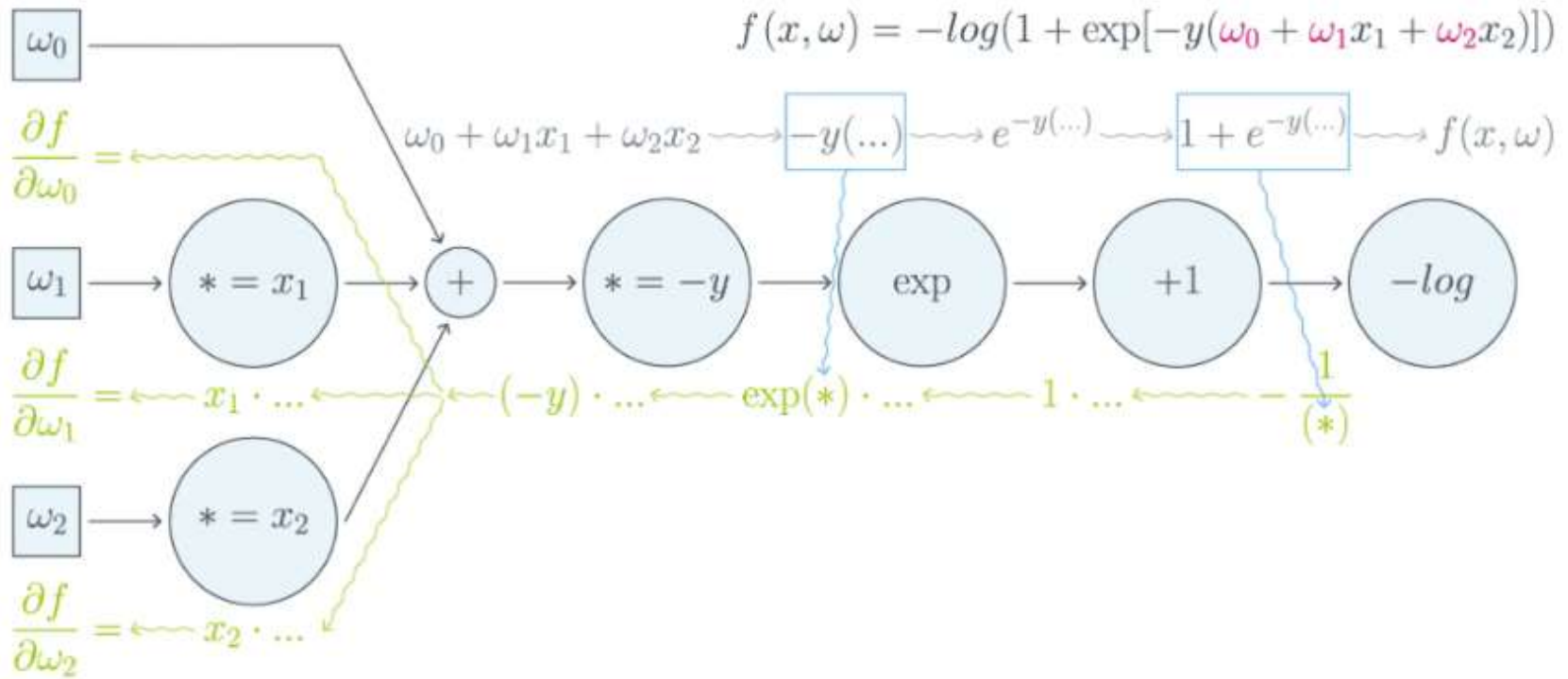


Обучение нейронной сети

- Метод обратного распространения ошибки (error backward propagation, BackProp) – метод вычисления градиента, который используется при обновлении весов НС.
- Был предложен в 1986 году Дэвидом Э. Руммельхартом, Джеффри Э. Хинтоном и Рональдом Дж. Вильямсом, а также независимо и одновременно красноярскими математиками С.И. Барцевым и В.А. Охониным
- Суть метода: исходя из формулы производного сложной функции (chain rule) градиенты можно вычислять последовательно в ходе обратного прохода, умножая каждый раз на частные производные предыдущего слоя

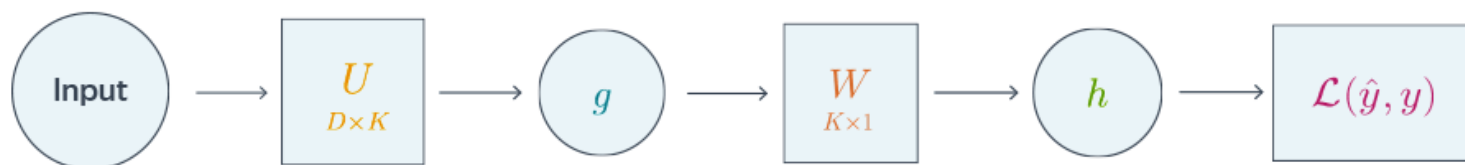
$$f(x) = g_m \left(g_{m-1} \left(\dots \left(g_1(x) \right) \dots \right) \right)$$
$$\frac{\partial f}{\partial x} = \frac{\partial g_m}{\partial g_{m-1}} \frac{\partial g_m}{\partial g_{m-1}} \dots \frac{\partial g_2}{\partial g_1} \frac{\partial g_1}{\partial x}$$

- BackProp в одномерном случае на примере функции потерь логистической регрессии на одном объекте:



- Сначала выполняем прямой проход (forward propagation) для вычисления всех промежуточных значений, которые необходимо хранить в памяти
- Затем выполняется backprop, на котором за один проход вычисляются все градиенты

- BackProp в общем виде:
- Инициализируем значения весов W_0^i , выделяем X – мини-батч
- Сначала выполняем прямой проход (forward propagation) вычисляя и сохраняя все промежуточные представления $X = X^0, X^1, \dots, X^m = \hat{y}$
- Вычисляем все градиенты с помощью backprop
- С помощью градиентов совершаем шаг SGD



Вход: выборка $(x_i, y_i)_{i=1}^{\ell}$, архитектура $(H_l)_{l=1}^L$, параметры η, λ ;

Выход: вектор весов всех слоёв $w = (W^1, \dots, W^L)$;

инициализировать веса w ;

повторять

выбрать объект x_i из X^{ℓ} (например, случайно);

прямой ход: для всех $l = 1..L, h = 1..H_l$

$$S_{ih}^l := \sum_{k=0}^{H_{l-1}} w_{kh}^l x_{ik}^{l-1}; \quad x_{ih}^l := \sigma_h^l(S_{ih}^l); \quad z_{ih}^l := (\sigma_h^l)'(S_{ih}^l);$$

$$\varepsilon_{hi}^L := \frac{\partial \mathcal{L}_i(w)}{\partial x_h^L}, \quad h = 1..H_L;$$

обратный ход: для всех $l = L..2, k = 0..H_{l-1}$

$$\varepsilon_{ik}^{l-1} = \sum_{h=0}^{H_l} \varepsilon_{ih}^l z_{ih}^l w_{kh}^l;$$

градиентный шаг: для всех $l = 1..L, k = 0..H_{l-1}, h = 1..H_l$

$$w_{kh}^l := w_{kh}^l - \eta \varepsilon_{ih}^l z_{ih}^l x_{ik}^{l-1};$$

пока значения Q и/или веса w не стабилизируются;

Автоматическое дифференцирование выражений (autograd) реализовано во всех основных библиотеках – PyTorch, TensorFlow и т.д.

- Регуляризация нейронных сетей

- Изменение функции потерь:
- Изменение структуры сети
- Изменение данных

- Регуляризация через изменение функции потерь:

Weight Decay – штраф за высокие значения весов НС с коэффициентом регуляризации λ :

$$L_{regularization} = L_{original} + \lambda \|W\|_2$$

Для задачи классификации можно использовать энтропия распределения предсказаний:

$$L_{regularization} = L_{original} - \lambda \sum_k \hat{p}_k \log \hat{p}_k$$

\hat{p} – предсказанные вероятности

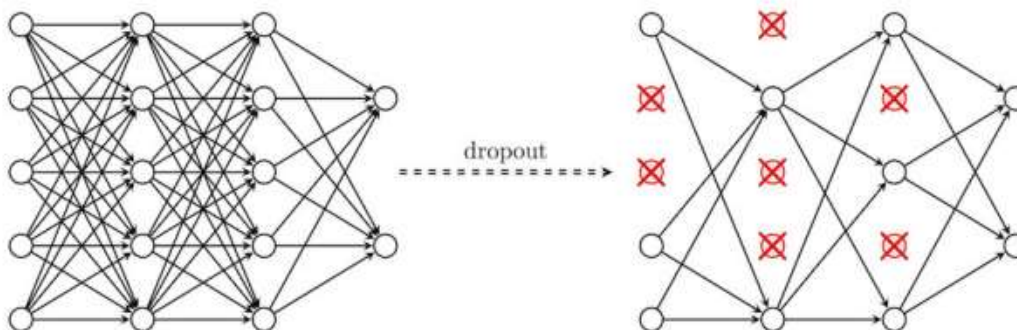
- Регуляризация через ограничение структуры модели:
- Dropout – случайным образом (с заданной вероятностью) «выключаем» доступ к некоторым координатам внутренних представлений на этапе обучения

Этап обучения: делая градиентный шаг $\mathcal{L}_i(w) \rightarrow \min_w$, отключаем h -ый нейрон l -го слоя с вероятностью p_l :

$$x_{hi}^l = \xi_h^l \sigma_h^l \left(\sum_k w_{kh}^l x_{ki}^{l-1} \right), \quad P(\xi_h^l = 0) = p_l$$

Этап применения: включаем все нейроны, но с поправкой:

$$x_{hi}^l = (1 - p_l) \sigma_h^l \left(\sum_k w_{kh}^l x_{ki}^{l-1} \right)$$

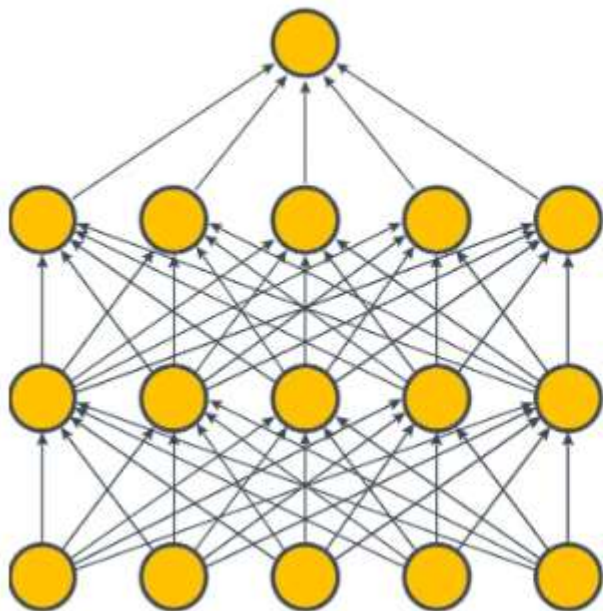


N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov.

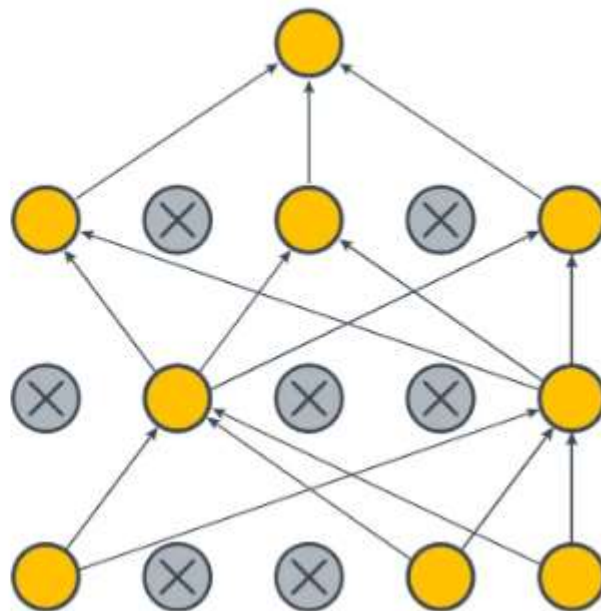
Dropout: a simple way to prevent neural networks from overfitting. 2014.

- Dropout можно применять и к входным данным – первым ставим слой dropout, это равносильно отбору признаков. Например, если в данных множество мультиколлинеарных признаков, или данные сильно зашумлены, или сильно разрежены, применение dropout в качестве первого слоя может приводить к получению более качественных результатов

Standard Neural Net



After applying dropout



- Batch normalization (batchNorm) – добавления слоя batch normalization, на котором текущий батч приводится к нулевому среднему и единичной дисперсии:

$$X^{k+1} = \frac{X^k - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Input: Network N with trainable parameters Θ ;
subset of activations $\{x^{(k)}\}_{k=1}^K$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

- 1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
- 2: **for** $k = 1 \dots K$ **do**
- 3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)}, \beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
- 4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
- 5: **end for**
- 6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$
- 7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
- 8: **for** $k = 1 \dots K$ **do**
- 9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
- 10: Process multiple training mini-batches \mathcal{B} , each of size m , and average over them:

$$\mathbb{E}[x] \leftarrow \mathbb{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\text{Var}[x] \leftarrow \frac{m}{m-1} \mathbb{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$
- 11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma, \beta}(x)$ with

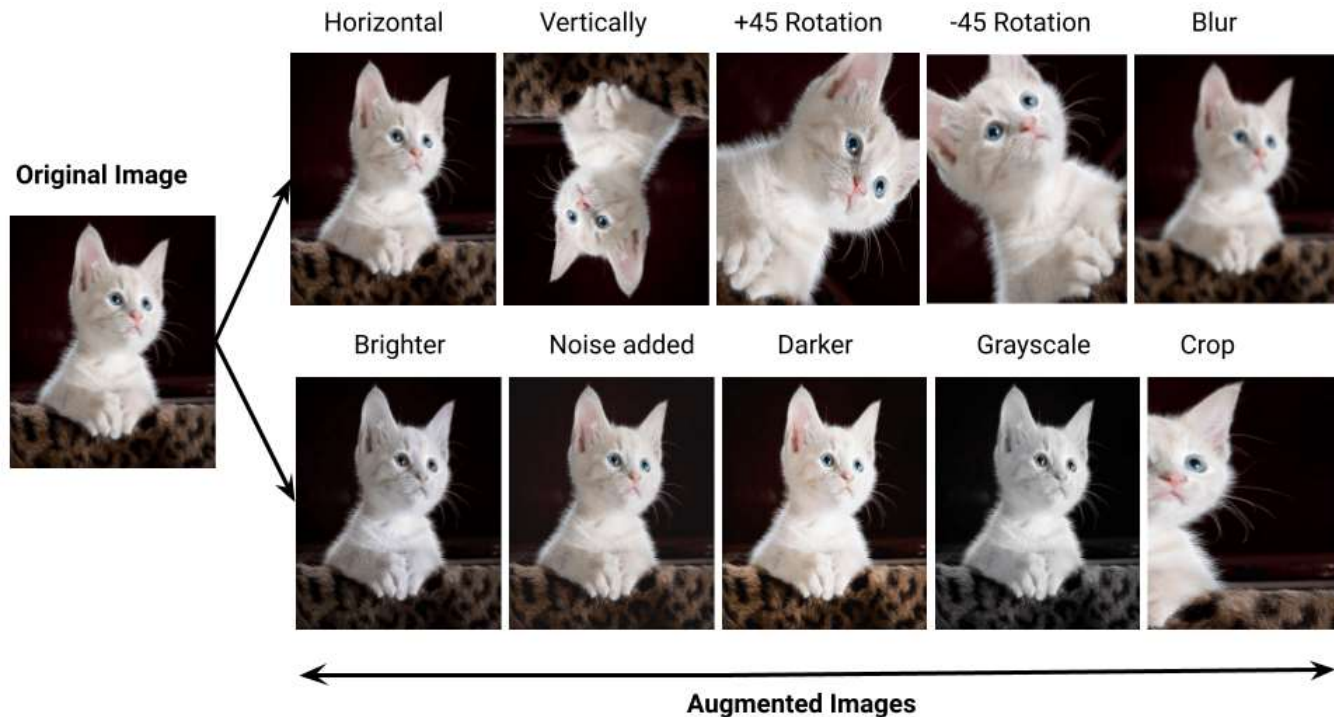
$$y = \frac{\gamma}{\sqrt{\text{Var}[x] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \right)$$
- 12: **end for**

Algorithm 2: Training a Batch-Normalized Network

- Регуляризация через изменение данных

Внесение изменений в данные (аугментация данных) позволяет увеличить объем обучающей выборки и дают понять модели, какие преобразования являются допустимыми. На примере изображений:

- Поворот и отражения
- Изменение масштаба
- Изменение яркости, контраста и насыщенности
- Добавление шума, искажений
- Обрезка (crop)
- Размытие



Original image



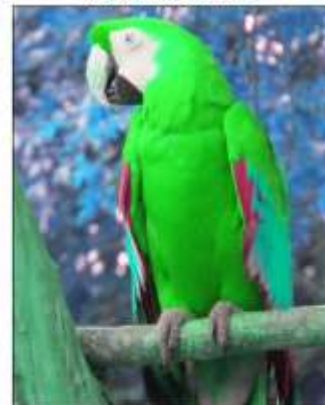
RGBShift



HueSaturationValue



ChannelShuffle



CLAHE



RandomContrast



RandomGamma



RandomBrightness



Blur



MedianBlur



ToGray



JpegCompression



PyTorch example

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        # First 2D convolutional layer, taking in 1 input channel (image),
        # outputting 32 convolutional features, with a square kernel size of 3
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        # Second 2D convolutional layer, taking in the 32 input layers,
        # outputting 64 convolutional features, with a square kernel size of 3
        self.conv2 = nn.Conv2d(32, 64, 3, 1)

        # Designed to ensure that adjacent pixels are either all 0s or all active
        # with an input probability
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)

        # First fully connected layer
        self.fc1 = nn.Linear(9216, 128)
        # Second fully connected layer that outputs our 10 labels
        self.fc2 = nn.Linear(128, 10)

my_nn = Net()
print(my_nn)
```



```

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    # x represents our data
    def forward(self, x):
        # Pass data through conv1
        x = self.conv1(x)
        # Use the rectified-linear activation function over x
        x = F.relu(x)

        x = self.conv2(x)
        x = F.relu(x)

        # Run max pooling over x
        x = F.max_pool2d(x, 2)
        # Pass data through dropout1
        x = self.dropout1(x)
        # Flatten x with start_dim=1
        x = torch.flatten(x, 1)
        # Pass data through ``fc1``
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)

        # Apply softmax to x
        output = F.log_softmax(x, dim=1)
        return output

```

```

1 import numpy as np
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5
6 # load the dataset, split into input (X) and output (y) variables
7 dataset = np.loadtxt('pima-indians-diabetes.csv', delimiter=',')
8 X = dataset[:,0:8]
9 y = dataset[:,8]
10
11 X = torch.tensor(X, dtype=torch.float32)
12 y = torch.tensor(y, dtype=torch.float32).reshape(-1, 1)
13
14 # define the model
15 model = nn.Sequential(
16     nn.Linear(8, 12),
17     nn.ReLU(),
18     nn.Linear(12, 8),
19     nn.ReLU(),
20     nn.Linear(8, 1),
21     nn.Sigmoid()
22 )
23 print(model)
24
25 # train the model
26 loss_fn = nn.BCELoss() # binary cross entropy
27 optimizer = optim.Adam(model.parameters(), lr=0.001)
28
29 n_epochs = 100
30 batch_size = 10
31
32 for epoch in range(n_epochs):
33     for i in range(0, len(X), batch_size):
34         Xbatch = X[i:i+batch_size]
35         y_pred = model(Xbatch)
36         ybatch = y[i:i+batch_size]
37         loss = loss_fn(y_pred, ybatch)
38         optimizer.zero_grad()
39         loss.backward()
40         optimizer.step()
41     print(f'Finished epoch {epoch}, latest loss {loss}')
42
43 # compute accuracy (no_grad is optional)
44 with torch.no_grad():
45     y_pred = model(X)
46 accuracy = (y_pred.round() == y).float().mean()
47 print(f"Accuracy {accuracy}")

```

Model definition

Train loop

- **Summary:**

- Нейрон это линейная модель классификации или регрессии
- Нейронная сеть это суперпозиция нейронов с нелинейной функцией активации
- Универсальная теорема аппроксимации – теоретически, двух-трех слоев достаточно для решения широкого класса задач
- Глубокие нейронные сети автоматизируют выделение признаков из сложно структурированных данных
- BackProp это быстрое дифференцирование суперпозиций (НС), метод позволяет обучать сети практически любой архитектуры
- Некоторые эвристики по улучшению сходимости: регуляризация, функции активации, модификации градиентного спуска