ШИФТ
Экосистема

# Основы нейронных сетей

**Юрченко Максим**

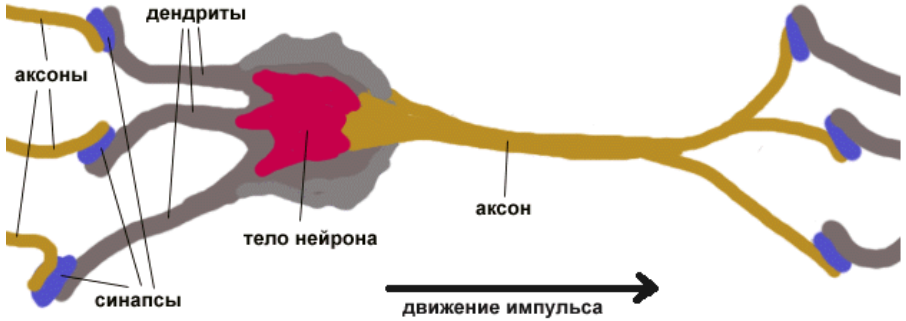
Дата аналитик

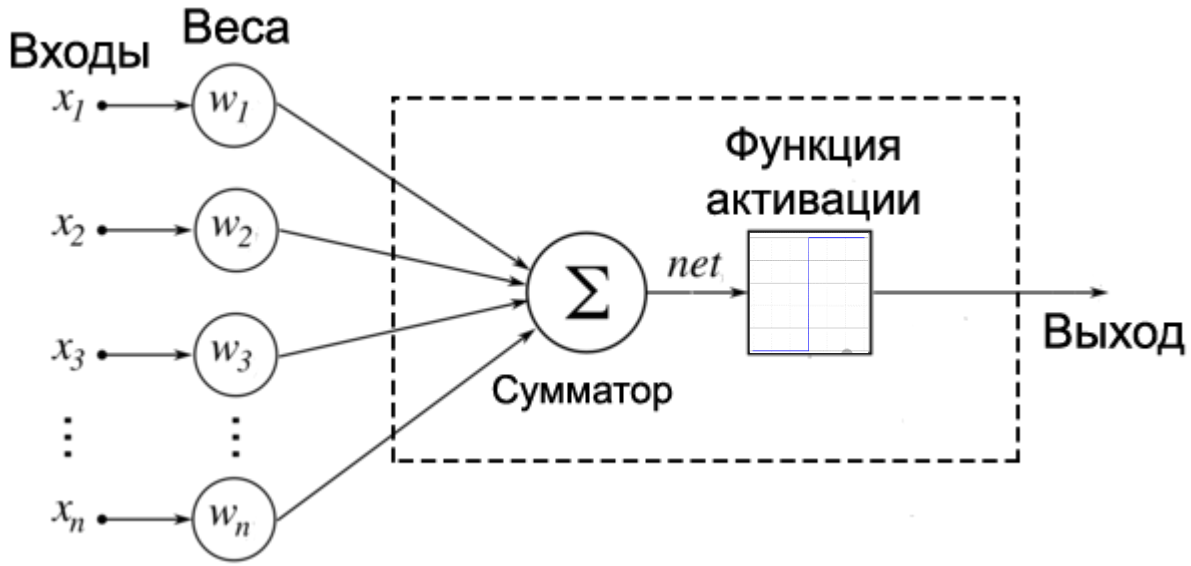
# Структура доклада

1. Механизм предсказаний нейронной сети
2. Обратное распространение ошибки
3. Функции активации
4. Оптимизаторы и стохастический градиентный спуск
5. Немного о computer vision

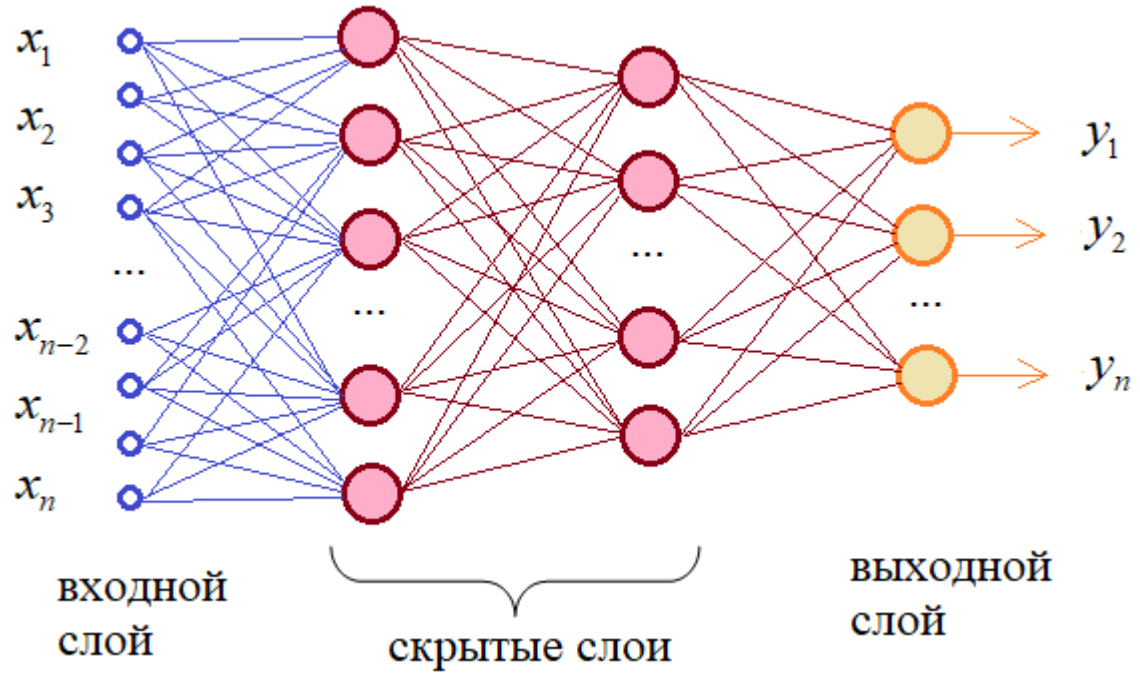
# Идея построения нейронной сети

Механизм принятия решения подсмотрен у биологической нервной системы



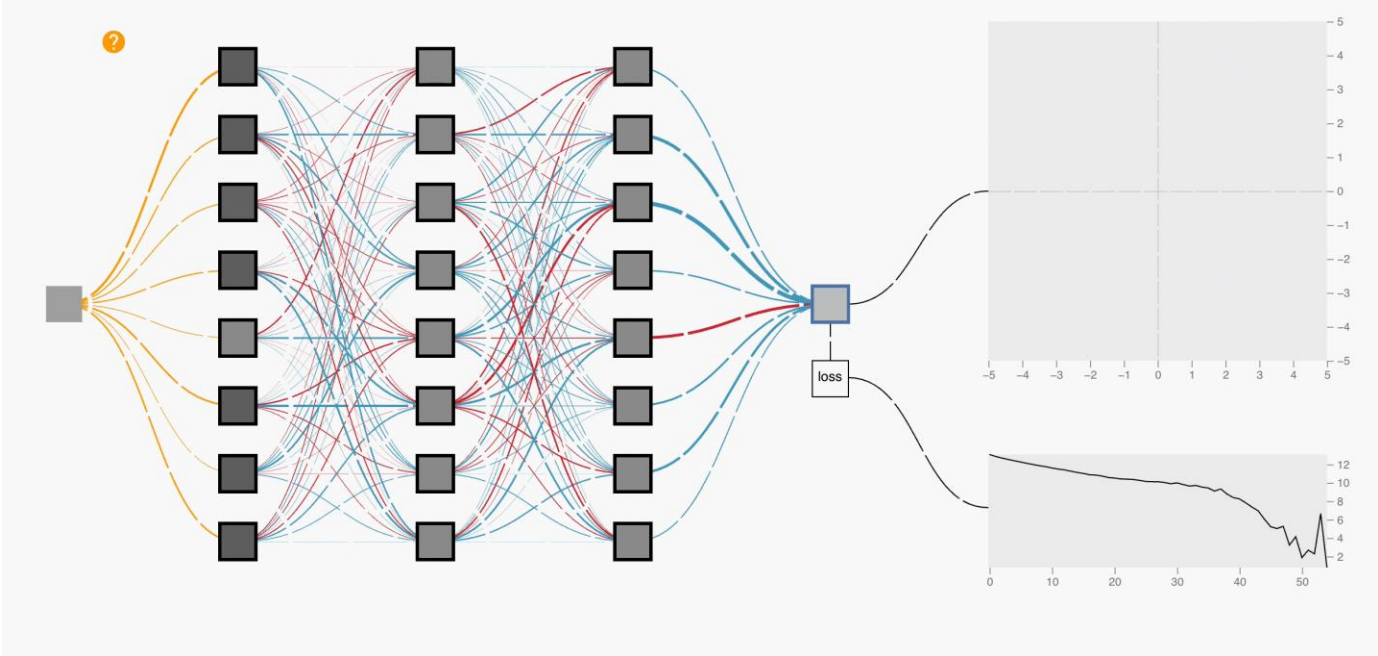
# Нейронная сеть



# Работа полносвязанной нейронной сети

- 1. Прямой проход и предсказание
- 2. Вычисление функции потерь и градиентов
- 3. Обратный проход (Backpropagation) и обновление весов
- 4. Оптимизатор

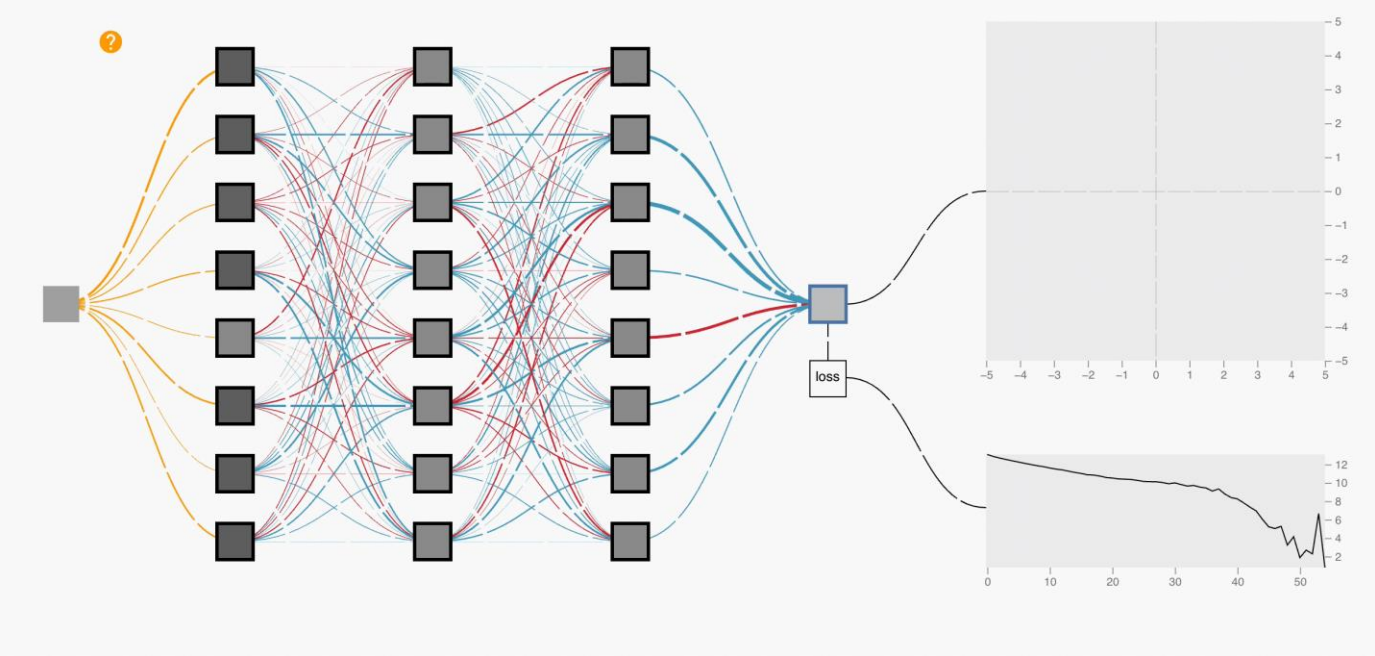
Полносвязанная нейронная сеть прямого распространения




https://xnought.github.io/backprop-explainer/

# Прямой проход

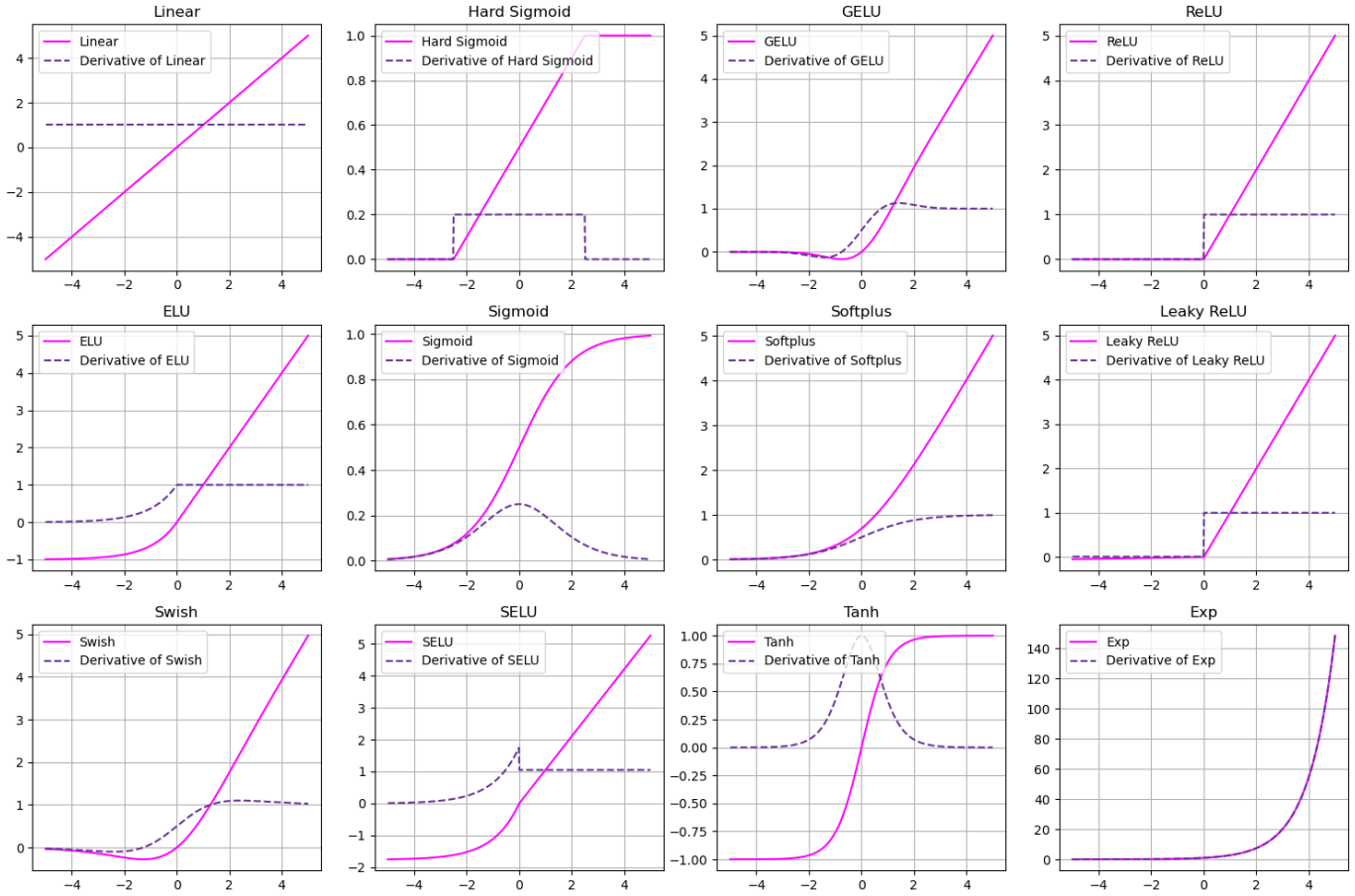
- Вычисляется сумма входных сигналов с учетом веса каждой связи и применяется активационная функция
- torch.nn.functional https://docs.pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity
- keras.activations https://keras.io/api/layers/activations/




https://xnought.github.io/backprop-explainer/

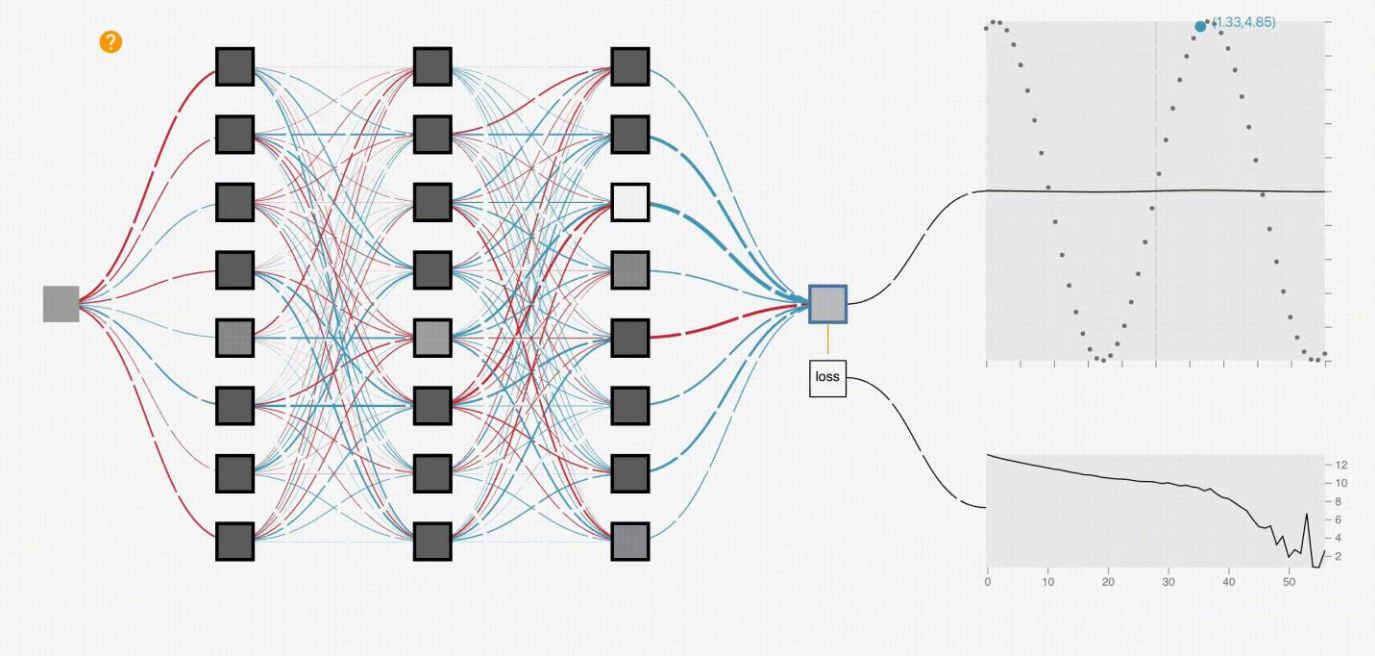
# Функции активации

| Функция | Когда использовать |
|---|---|
| ReLU | Общее применение в скрытых слоях |
| Leaky ReLU / ELU / SELU | Чтобы избежать мертвого ReLU |
| Sigmoid | Выходной слой бинарной классификации |
| Tanh | Для нормализованных выходов (-1 до 1) |
| Softmax | Многоклассовая классификация (не вошла в список) |
| Swish / GELU | Современные альтернативы ReLU |
| Softplus | Плавная замена ReLU |
| Hard Sigmoid | Упрощённая версия сигмоиды (быстрее) |



# Вычисление ошибки с помощью функции потерь

- Функция потерь – это метрика, отражающая расхождение между ожидаемыми и полученными выходными данными
- torch.nn https://docs.pytorch.org/docs/stable/nn.functional.html#loss-functions
- keras.losses https://keras.io/api/losses/




https://xnought.github.io/backprop-explainer/

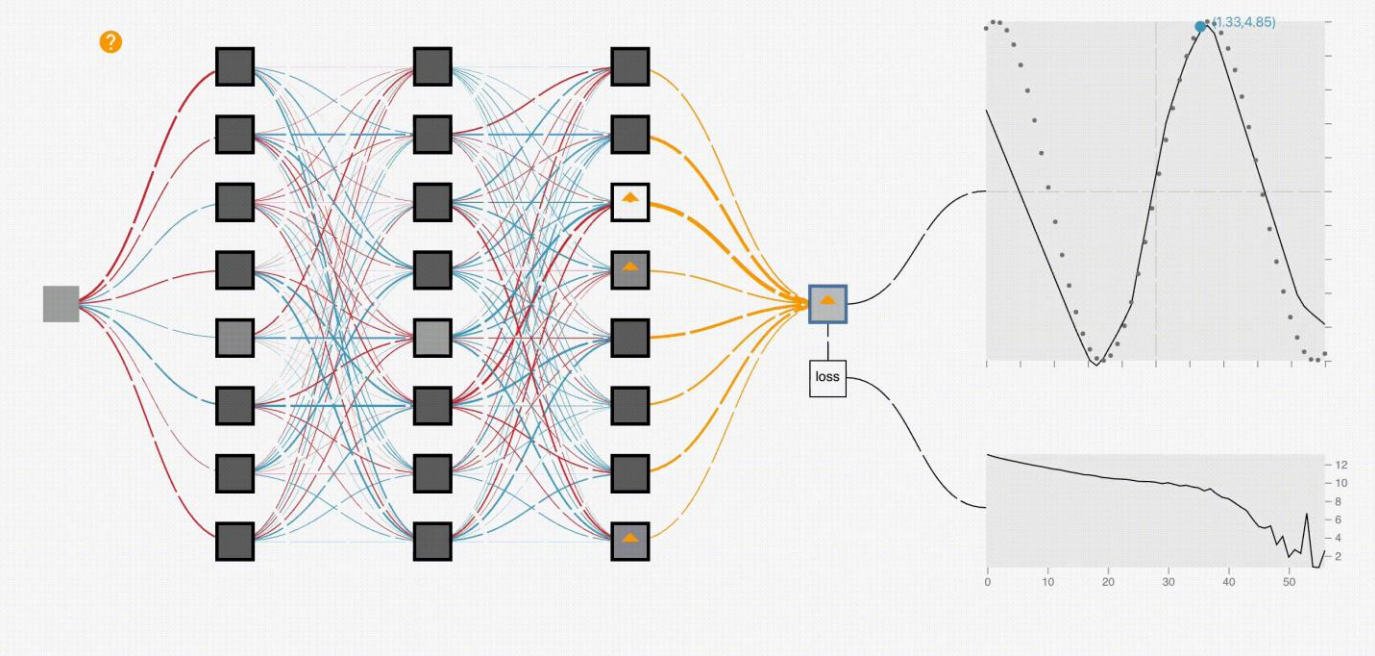
# Функции потерь

| Название функции | Тип задачи | Описание | Когда использовать |
|---|---|---|---|
| MSE / L2 Loss | Регрессия | Среднеквадратичная ошибка между предсказанием и истинным значением. | Когда ошибки распределены нормально, и важно минимизировать большие отклонения. Чувствительна к выбросам. |
| MAE / L1 Loss | Регрессия | Средняя абсолютная ошибка. Менее чувствительна к выбросам, чем MSE. | Если данные содержат выбросы или аномалии. Более устойчива к шуму. |
| Huber Loss | Регрессия | Гибрид MSE и MAE: квадратичная часть для малых ошибок, линейная — для больших. | Хорошо подходит, если нужно сбалансировать точность и устойчивость к шуму. |
| Log-Cosh | Регрессия | Гладкая версия Huber Loss. Устойчива к выбросам. | Альтернатива Huber Loss, когда нужна дифференцируемость и стабильность. |
| Binary Crossentropy | Классификация (бинарная) | Логистические потери для задачи с 2 классами. Выходной сигнал — сигмоида. | В бинарной классификации (например, спам/не спам). |
| Categorical Crossentropy | Классификация (многоклассовая) | Используется, если метки закодированы one-hot. Выходной слой — softmax. | Для многоклассовой классификации (например, MNIST). |
| Sparse Categorical Crossentropy | Классификация (многоклассовая) | То же, что и Categorical Crossentropy, но метки задаются как целые числа (индексы классов). | Когда нет необходимости кодировать метки one-hot. |
| Hinge Loss | Классификация (бинарная) | Подходит для задач с большими разделяющими границами. | В задачах с четкой разделяющей гиперплоскостью. Метки: -1 и +1. |
| Kullback-Leibler Divergence (KL) | Вероятностные модели | Измеряет "расстояние" между двумя распределениями. | В вариационных автокодировщиках (VAE), генеративных моделях. |
| Focal Loss | Классификация (многоклассовая) | Взвешенная версия crossentropy, акцентирует внимание на редких классах. | При несбалансированных данных (например, объект редко встречается). |
| Dice Loss | Сегментация изображений | Оценивает пересечение областей. | В задачах сегментации изображений, где важна точность пересечения. |
| Contrastive Loss | Обучение схожести | Минимизирует расстояние между похожими примерами и увеличивает между различными. | В face verification, парных сравнениях. |

# Обратный проход

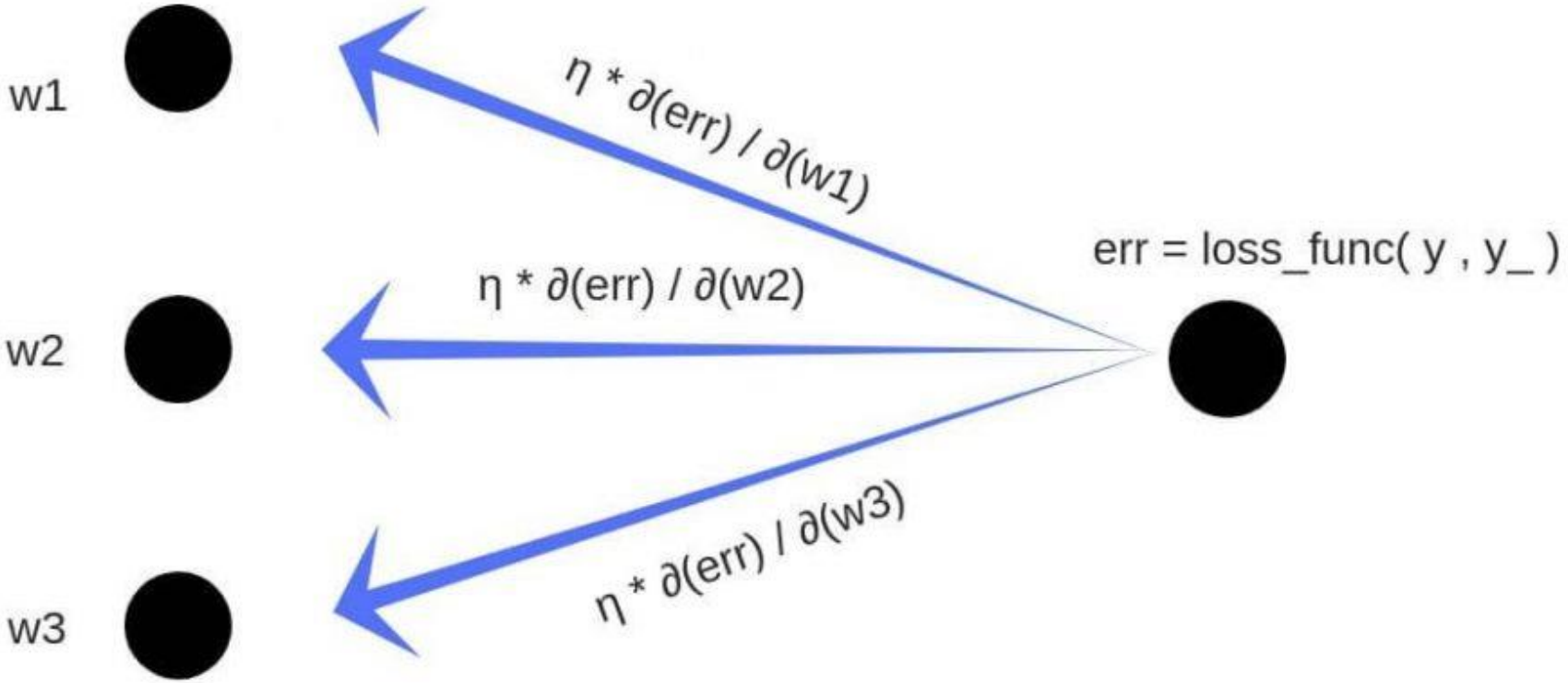
- Вычисленные антиградиенты используются для обновления весов;
- Частные производные не вычисляются аналитически, используются численные методы;
- Веса обновляются в соответствии с величиной частной производной функции потерь:

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w}$$



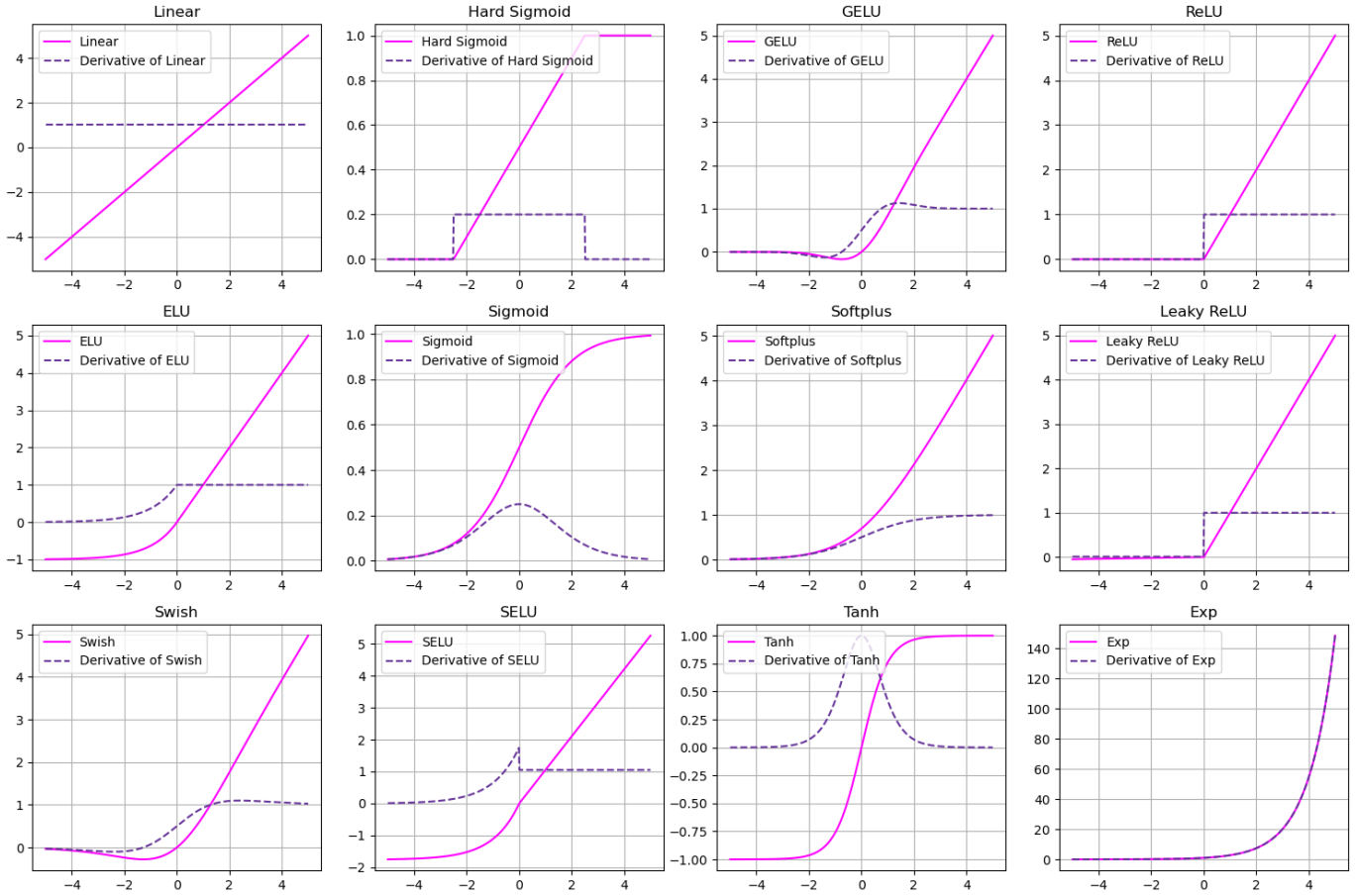

https://xnought.github.io/backprop-explainer/

# Обратный проход



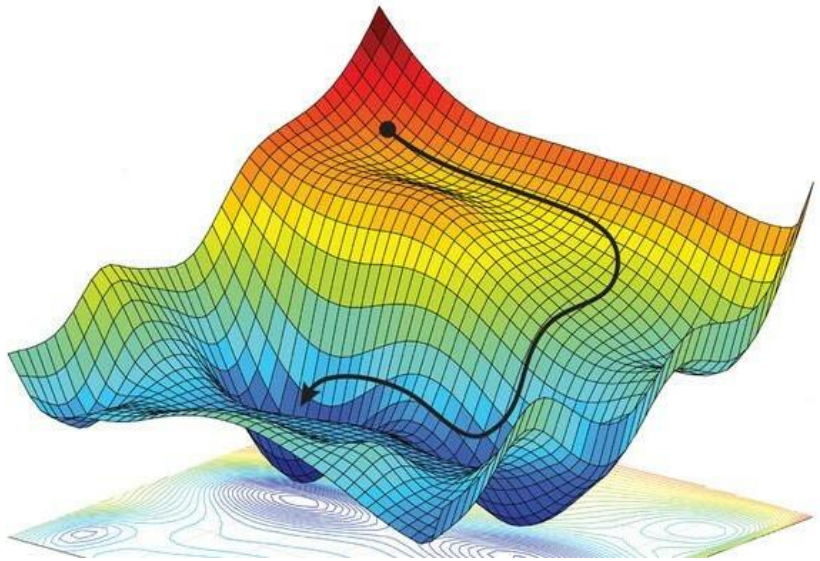
# Функции активации

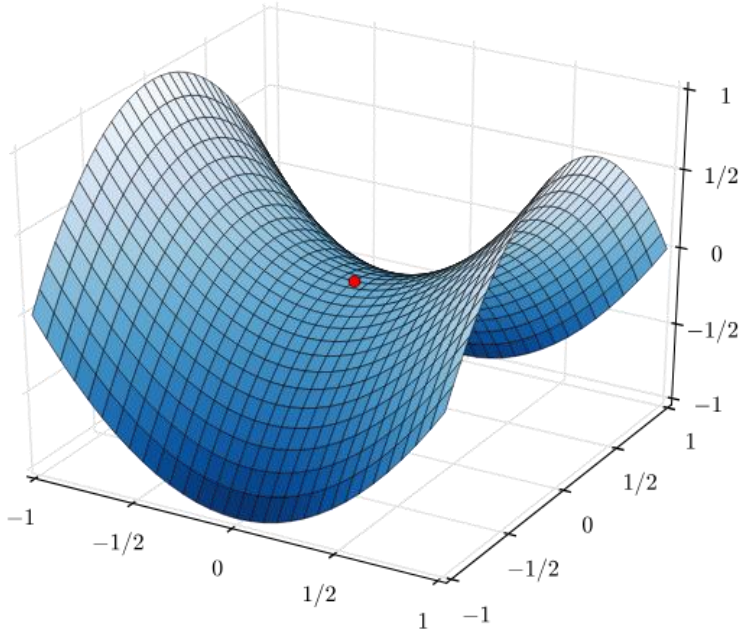
| Функция | Когда использовать |
|---|---|
| ReLU | Общее применение в скрытых слоях |
| Leaky ReLU / ELU / SELU | Чтобы избежать мертвого ReLU |
| Sigmoid | Выходной слой бинарной классификации |
| Tanh | Для нормализованных выходов (-1 до 1) |
| Softmax | Многоклассовая классификация (не вошла в список) |
| Swish / GELU | Современные альтернативы ReLU |
| Softplus | Плавная замена ReLU |
| Hard Sigmoid | Упрощённая версия сигмоиды (быстрее) |

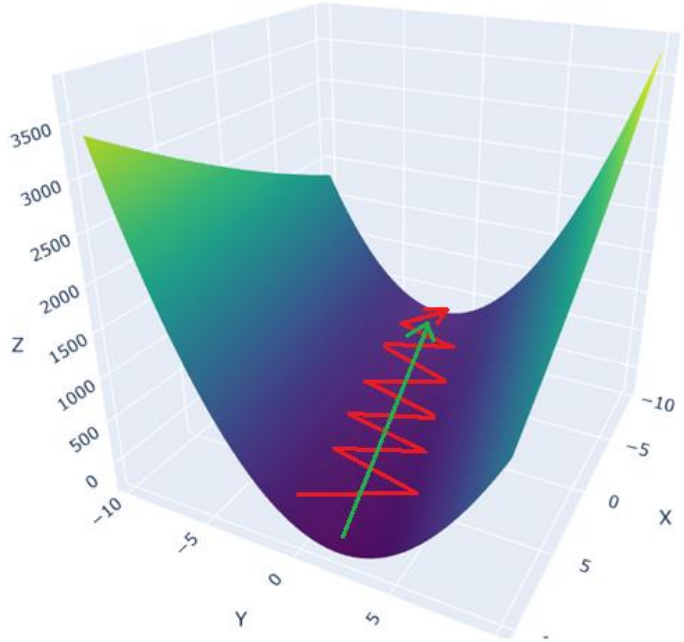


# Оптимизаторы

Самой распространённой техникой оптимизации, используемой большинством нейронных сетей, является алгоритм градиентного спуска.





# Оптимизаторы

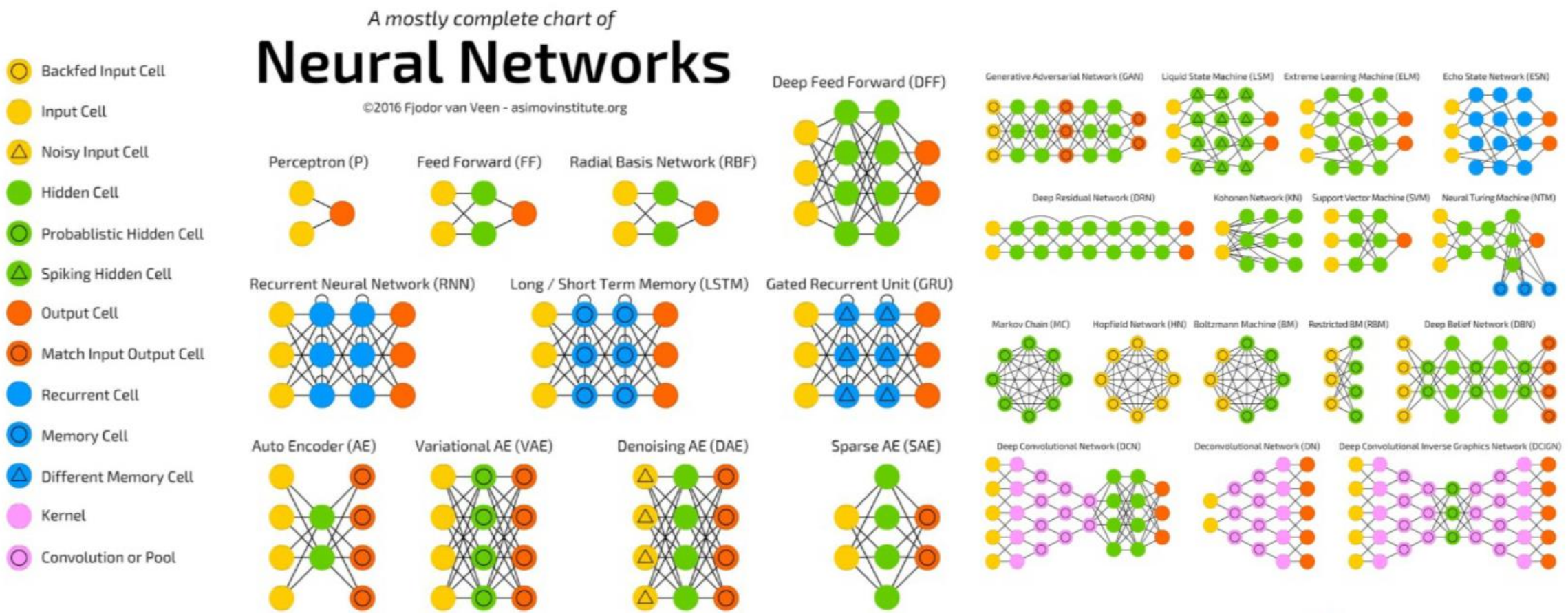
| Оптимизатор | Адаптивный LR | Использует момент | Лучше всего подходит |
|---|---|---|---|
| **SGD** | ✗ | ✗ | Простые задачи |
| **Momentum** | ✗ | ✓ | Сложные овраги |
| **Nesterov Accelerated Gradient (NAG)** | ✗ | ✓ | Более точный Momentum |
| **Adagrad** | ✓ | ✗ | Разреженные данные |
| **RMSProp** | ✓ | ✗ | Если перестали обновляться веса |
| **Adam** | ✓ | ✓ | Общее применение |
| **AdamW** | ✓ | ✓ | Улучшение регуляризации |
| **RAdam** | ✓ | ✓ | Добавление стабильности в начало обучения |
| **AdaBelief** | ✓ | ✓ | Исследования |
| **LBFGS** | ✗ | ✗ | Малые модели |

torch.optim https://docs.pytorch.org/docs/stable/optim.html

keras.optimizers https://keras.io/api/optimizers/

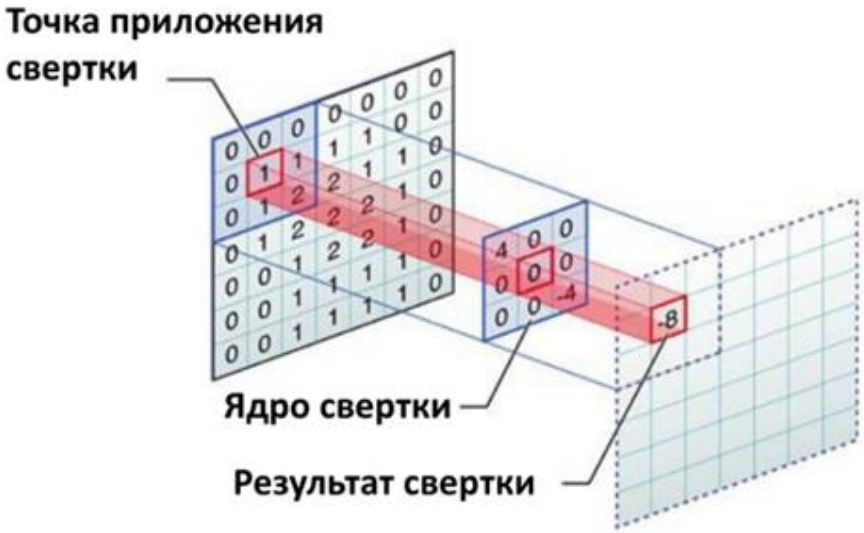
https://habr.com/ru/articles/318970/

# Архитектуры нейронных сетей

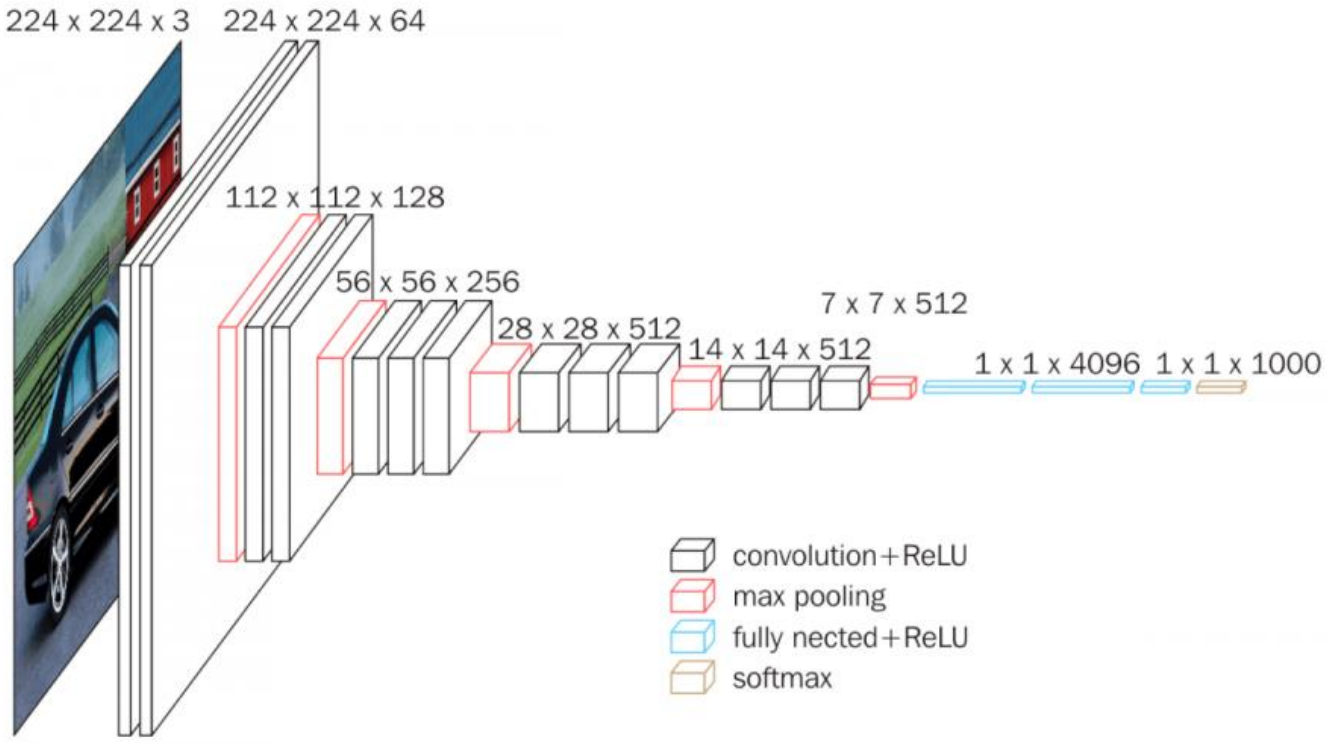


# Свёрточные нейронные сети

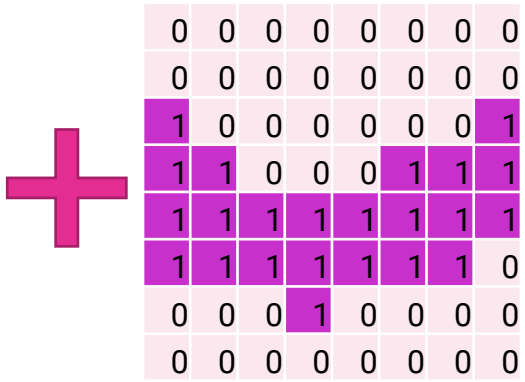
Основаны на операциях «свёртки»



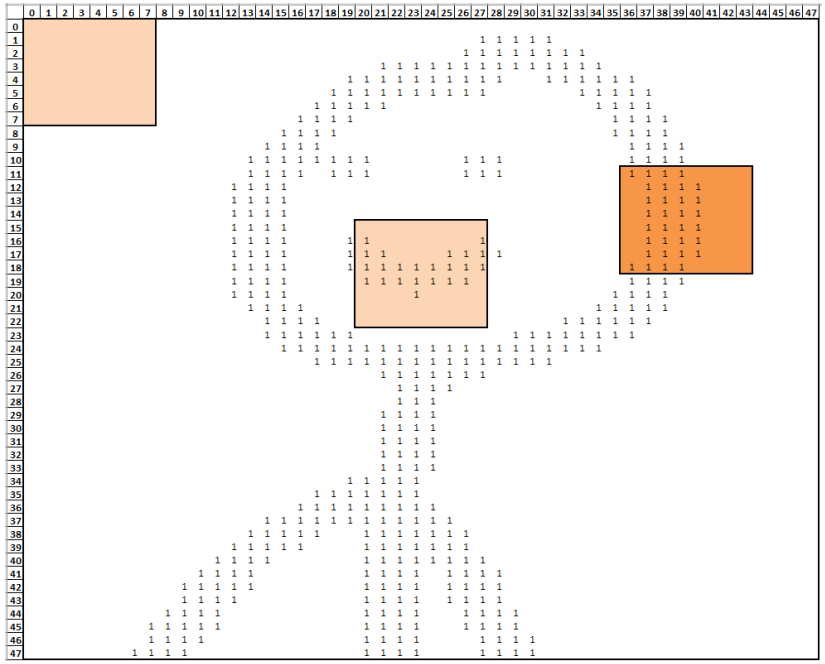

Архитектура VGG16



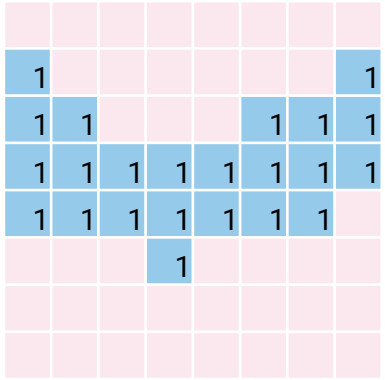
# Свёрточные нейронные сети



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

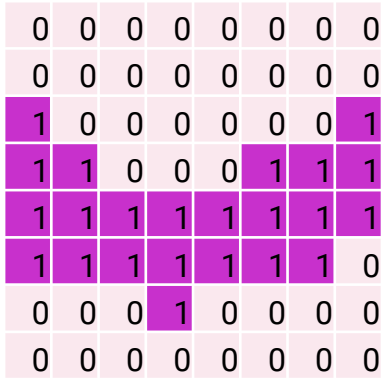


# Свёрточные нейронные сети

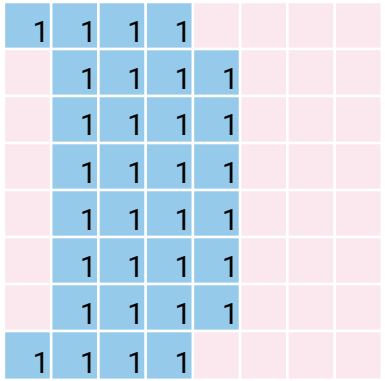



Сумма 23

×

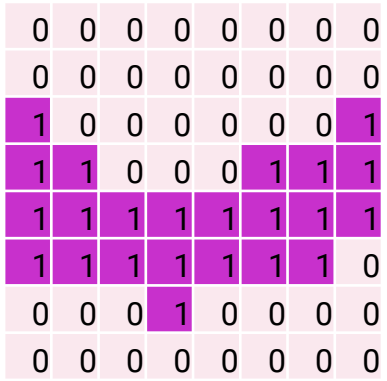



= 15



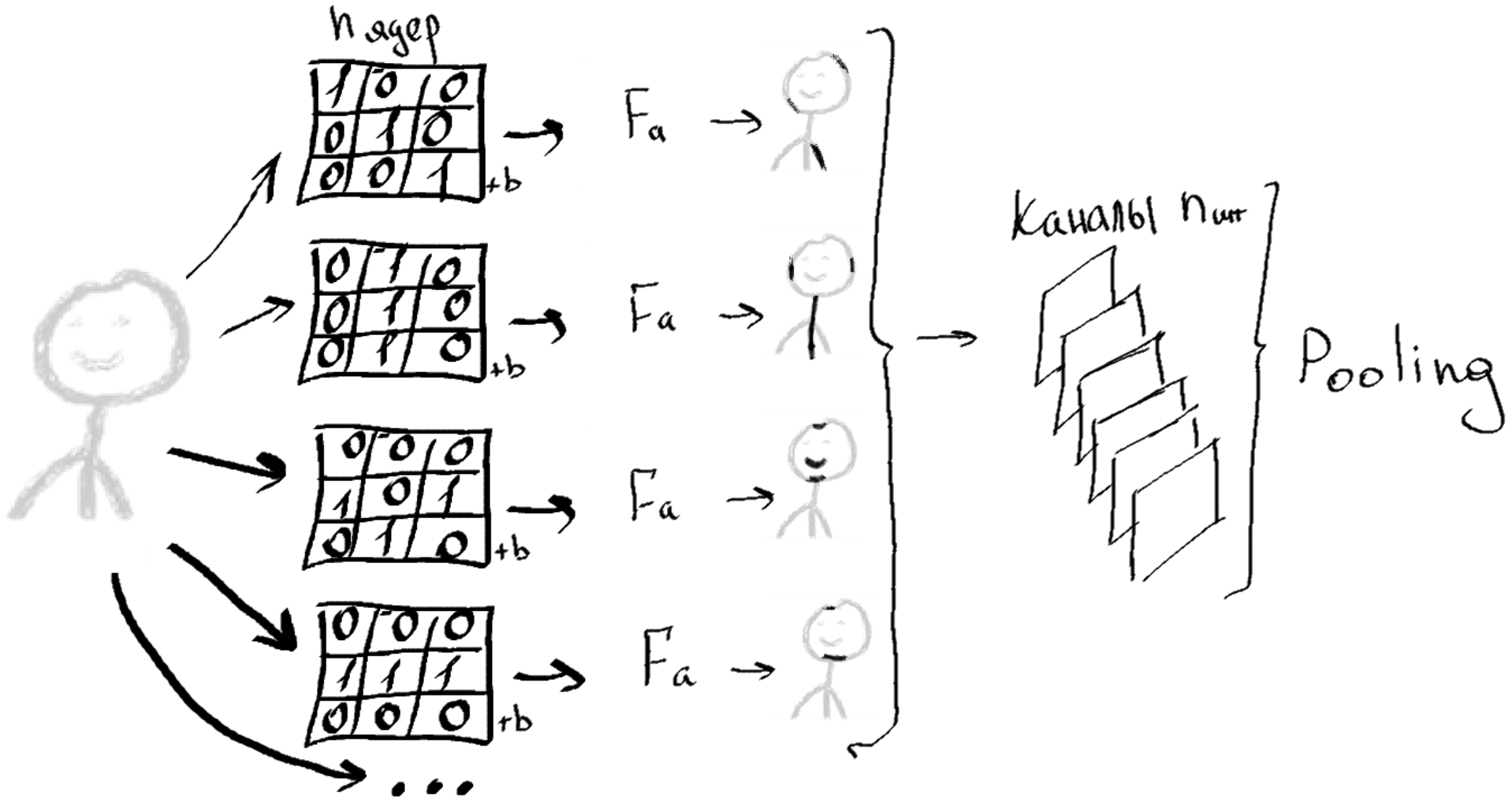

Сумма 32

×

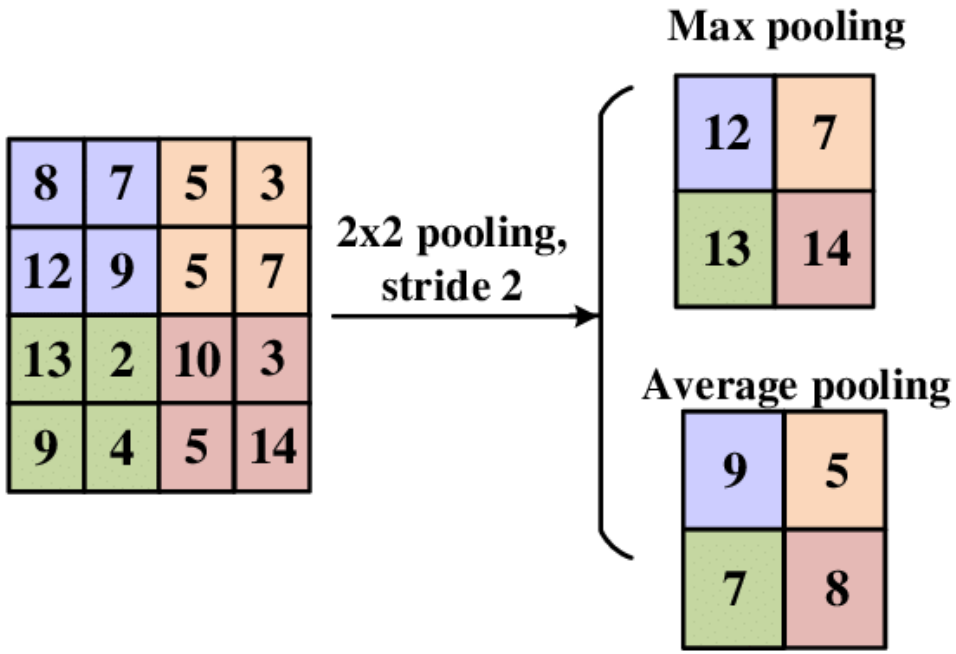


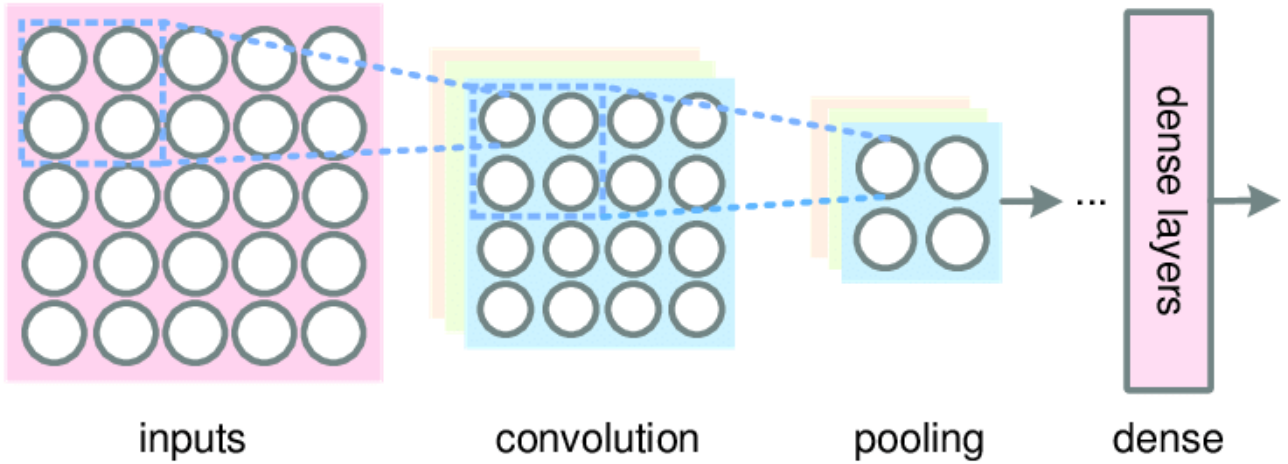

= 10

# Свёрточные нейронные сети



# Pooling


Max pooling
8 7 5 3
12 9 5 7
13 2 10 3
9 4 5 14
2x2 pooling, stride 2
12 7
13 14
Average pooling
9 5
7 8


inputs
convolution
pooling
...
dense layers
dense

# Блок вопросов