

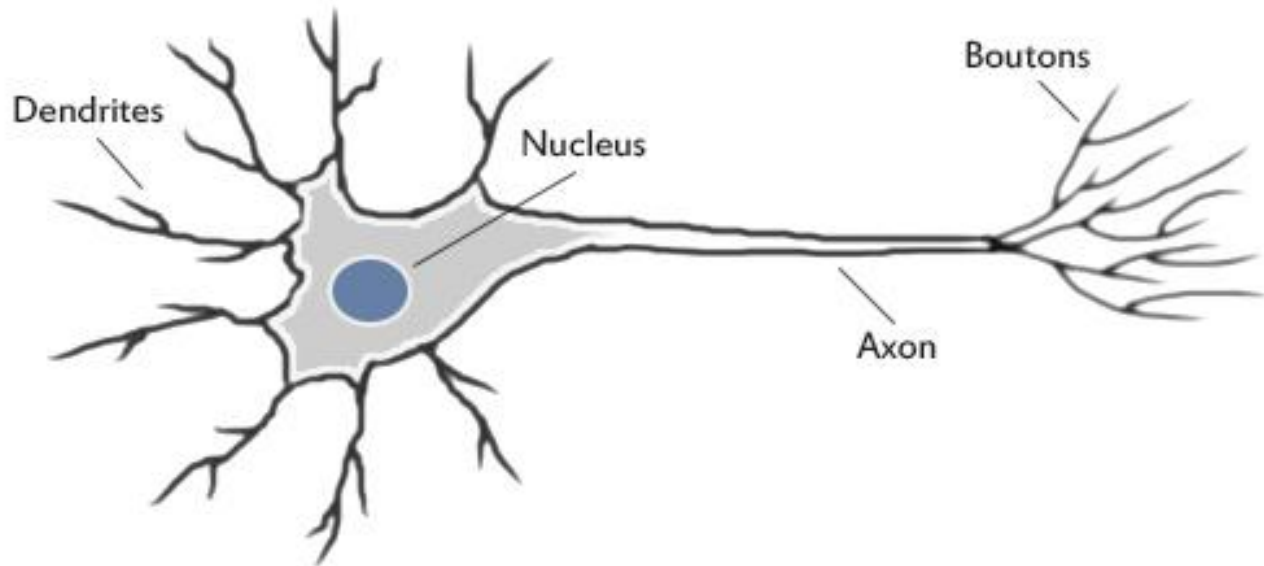
# Введение в нейросети

Grigory Sapunov  CTO / Intento

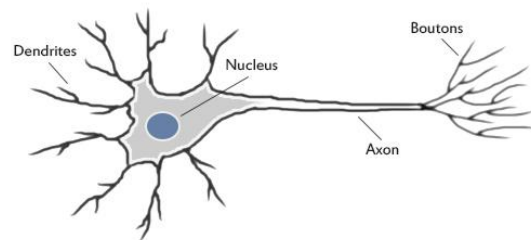
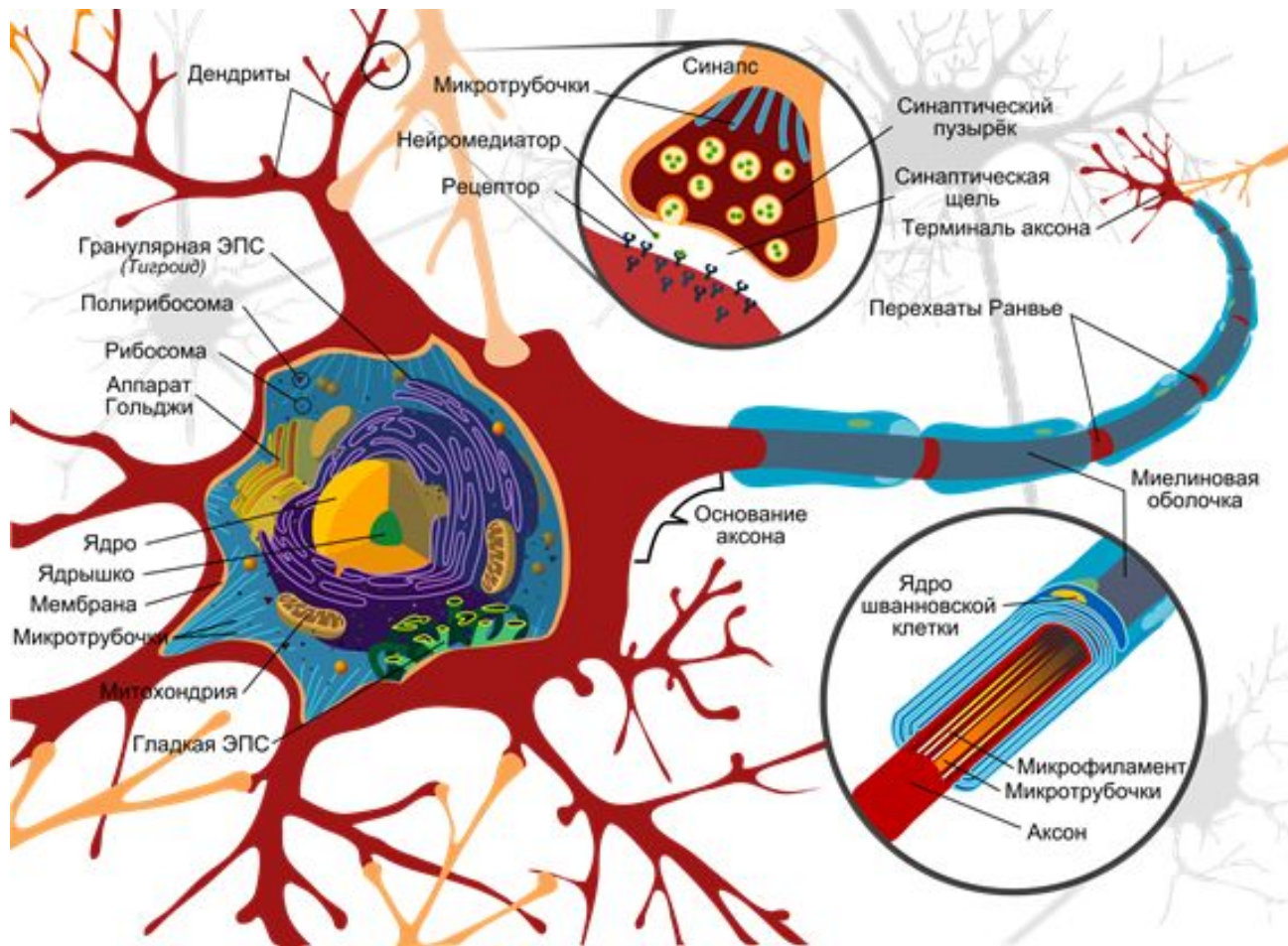
Нейрон

# Естественный нейрон

Нейрон — клетка нервной системы, способная к возбуждению и передаче сигнала.

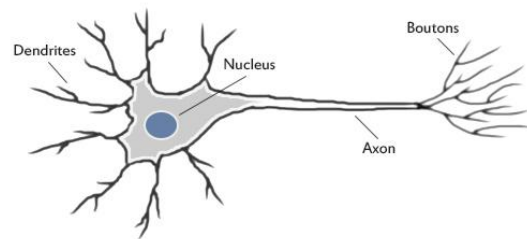
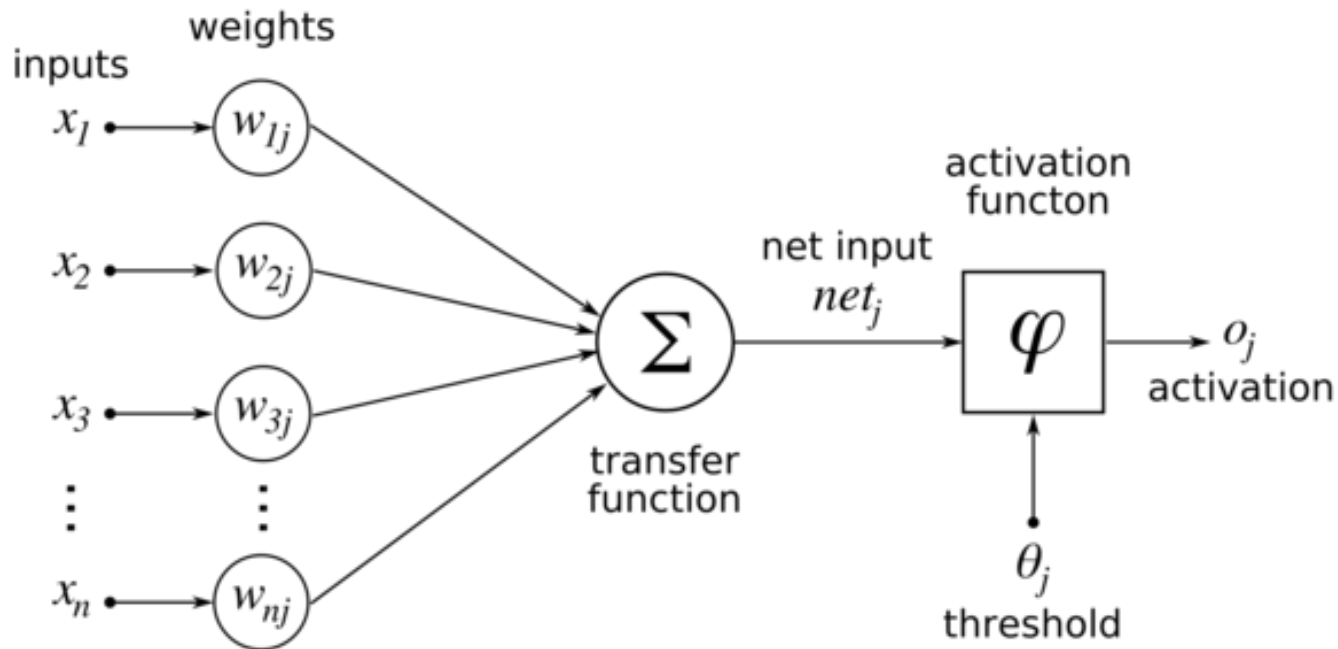


# В реальности всё посложнее...



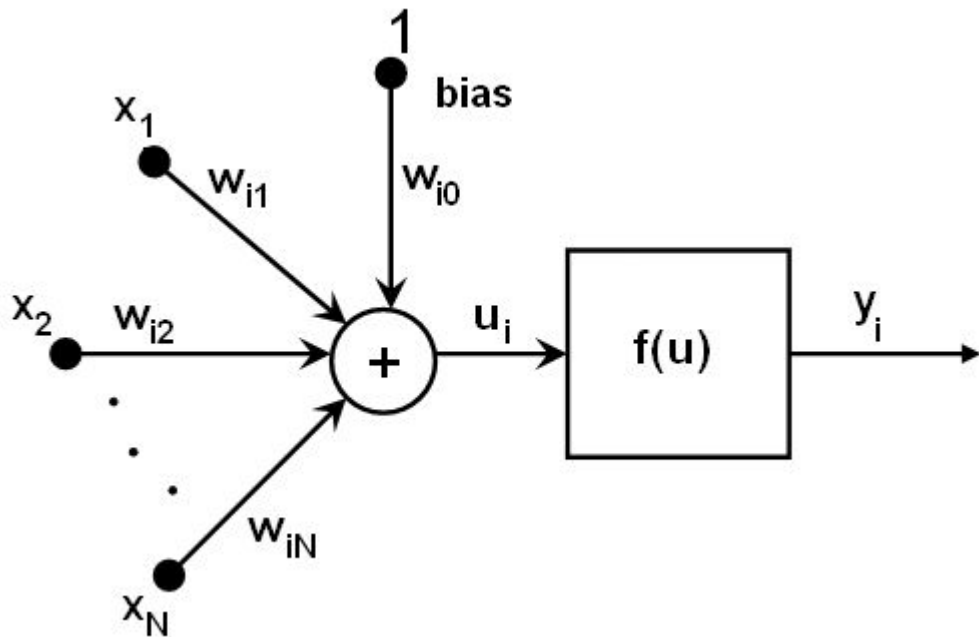
# Искусственный нейрон

Искусственный нейрон — отдалённое подобие биологического. Базовый элемент искусственной нейронной сети



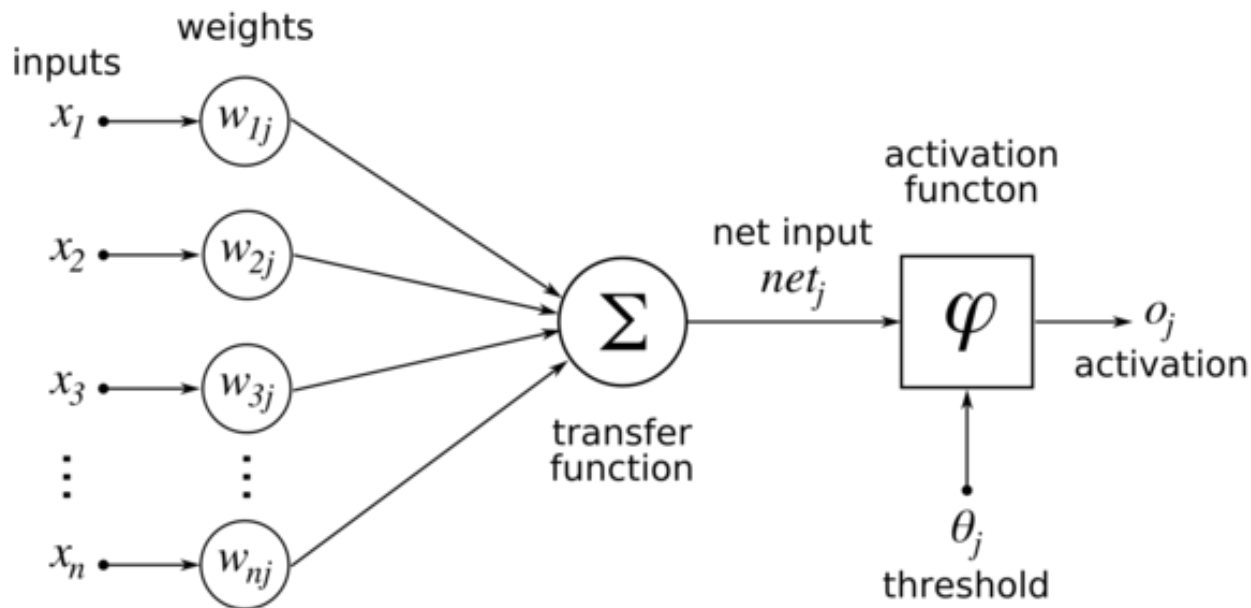
# Модель искусственного нейрона

Обычно у нейрона также есть bias-вход, константное смещение, вес на котором также обучается.



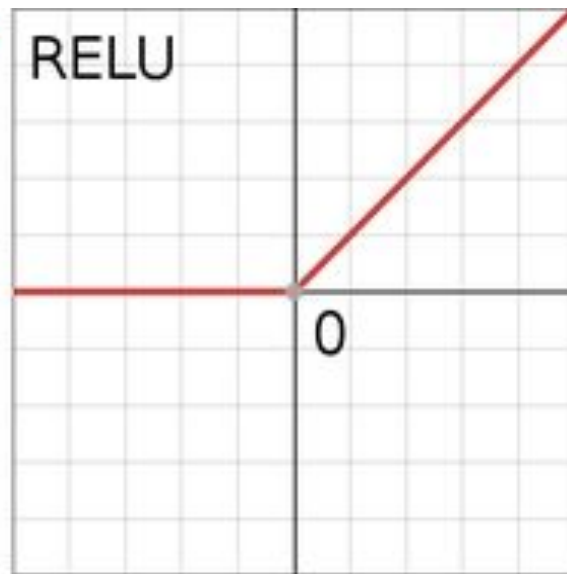
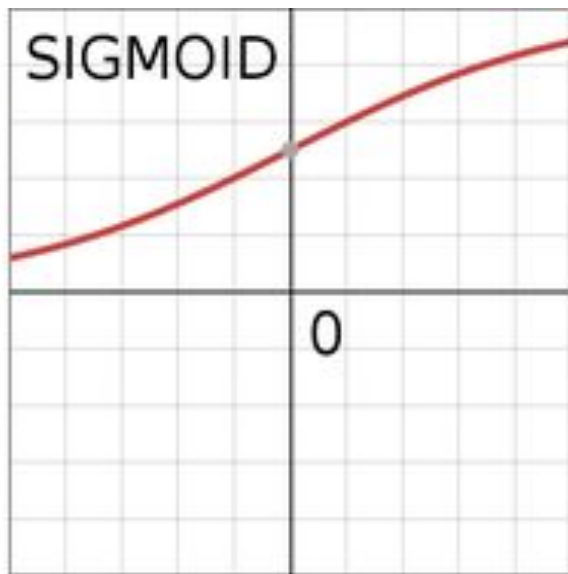
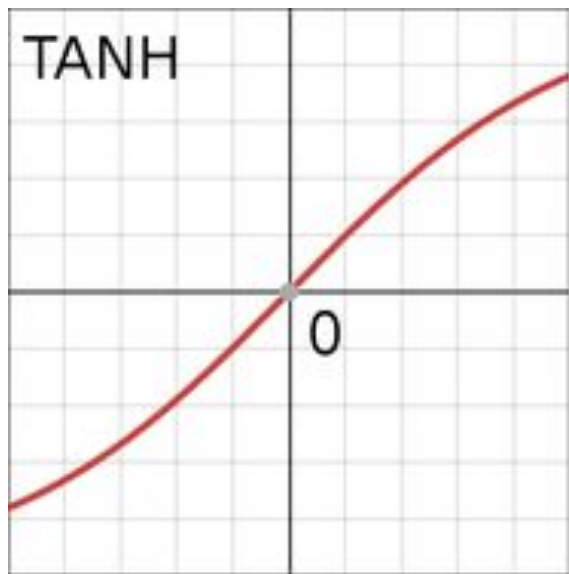
# Модель искусственного нейрона

Отдельный нейрон математически полностью аналогичен логистической регрессии (в случае если используется функция активации сигмоида)



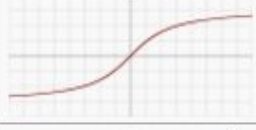





$$f(z) = \frac{1}{1 + e^{-z}}.$$

# Функции активации: Tanh, Sigmoid, ReLU

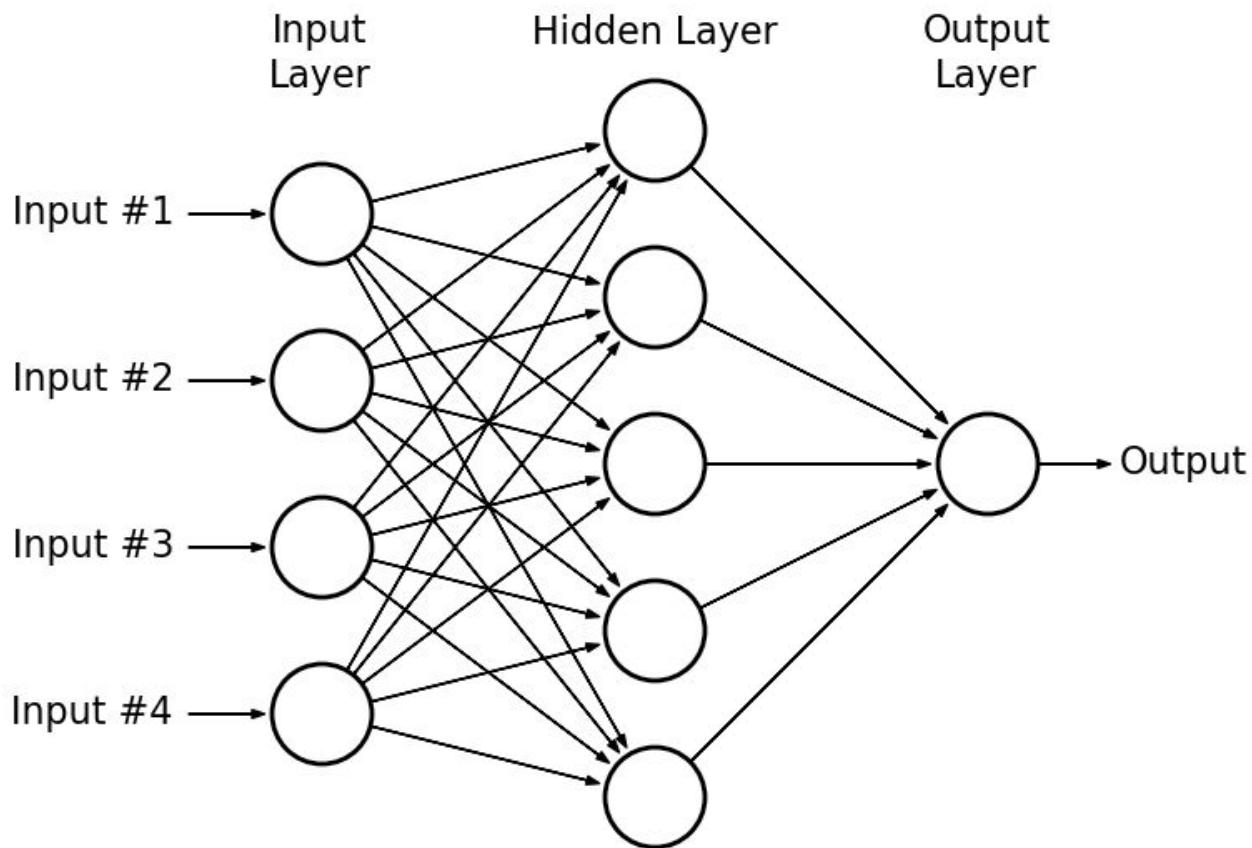




Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) <sup>[2]</sup>		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) <sup>[3]</sup>		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

Нейросеть

# Искусственная нейросеть: терминология



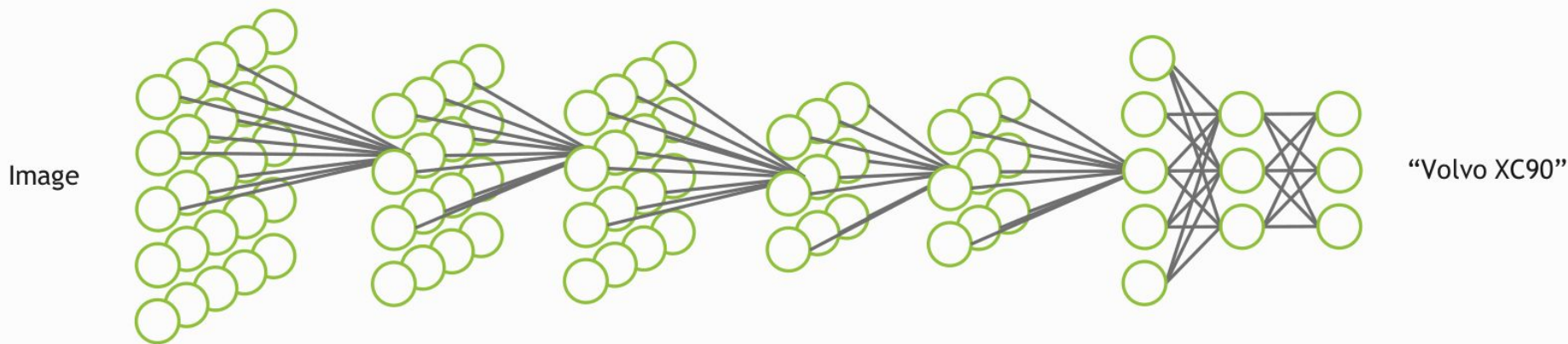
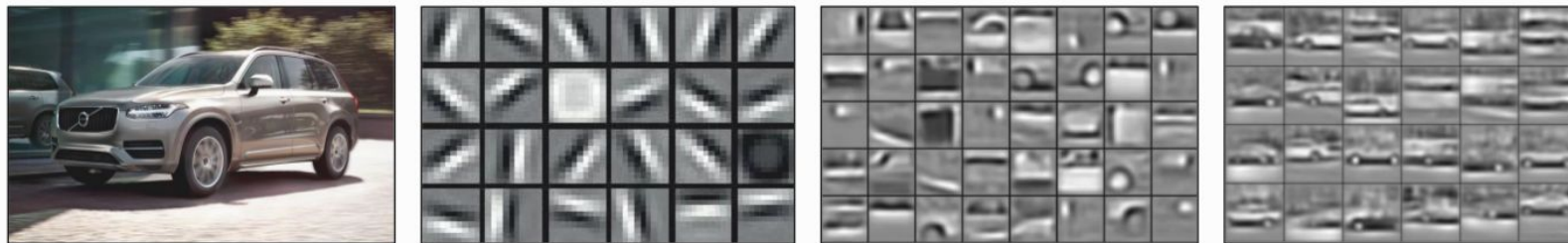
# Использование нейросети

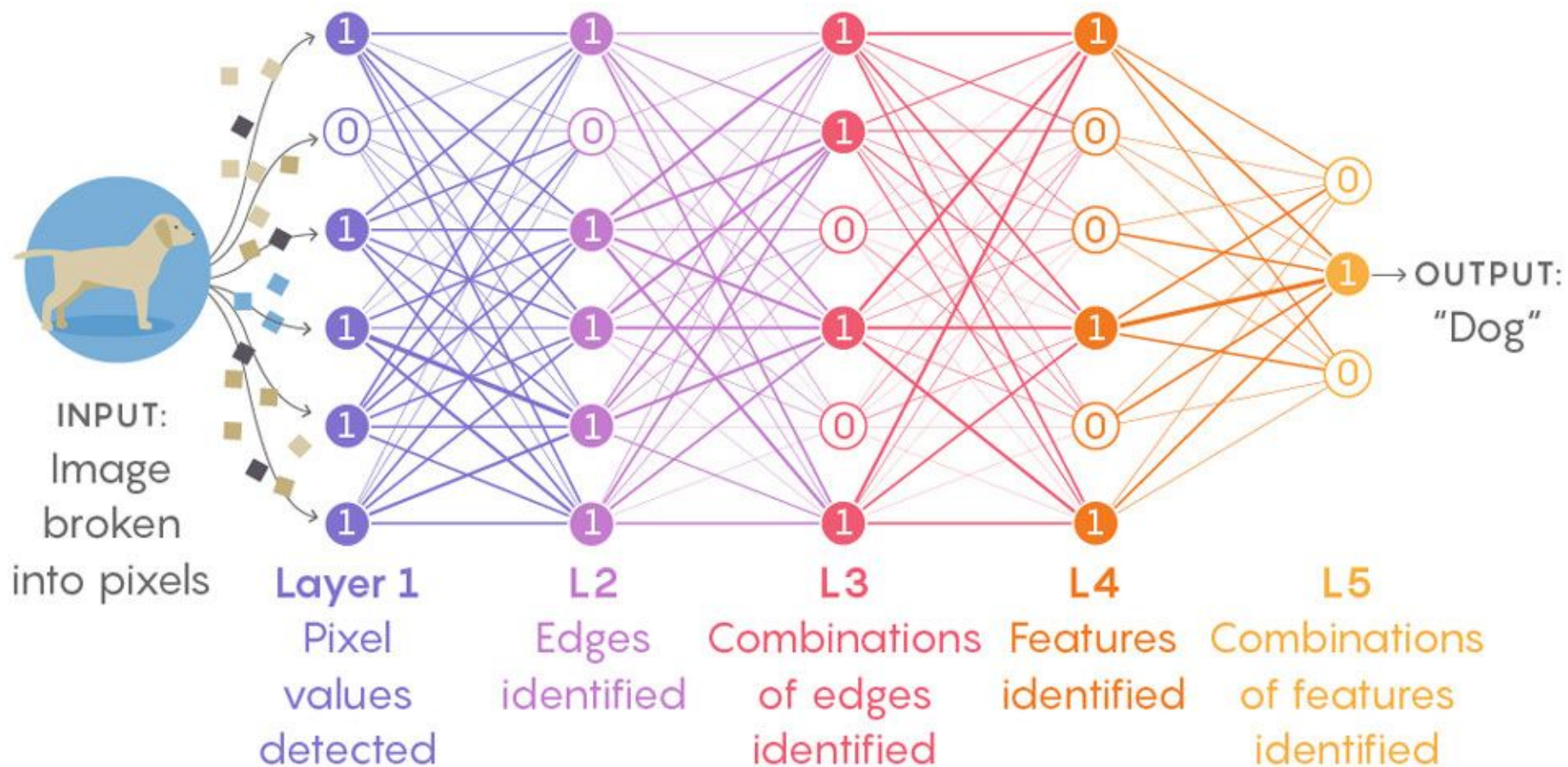
- На вход нейросети для каждого объекта подаётся вектор признаков
- На выходе нейросети получается результат. Вид результата определяется архитектурой нейросети
  - Например, в случае многоклассовой классификации (n классов) выходной слой содержит n нейронов, каждый из которых выдаёт “степень принадлежности” к данному классу (если активация была сигмоидальной)
  - Если нужно получить распределение вероятностей классов, используется функция softmax, нормализующая все выходные значения, так чтобы их сумма была равна 1.

$$\phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}}.$$

# Глубокая нейросеть

Наличие более одного скрытого слоя даёт возможность строить иерархические представления.





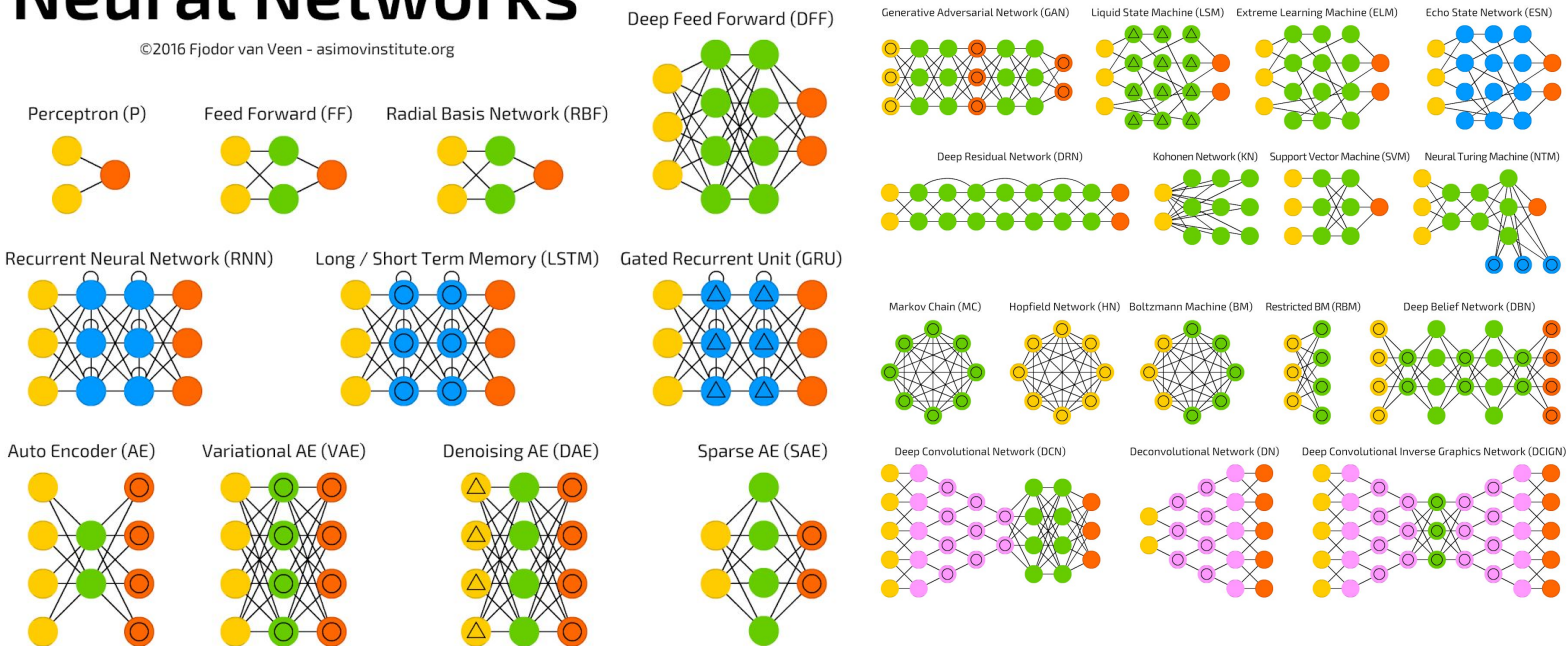


# Существует многообразие архитектур

## A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool



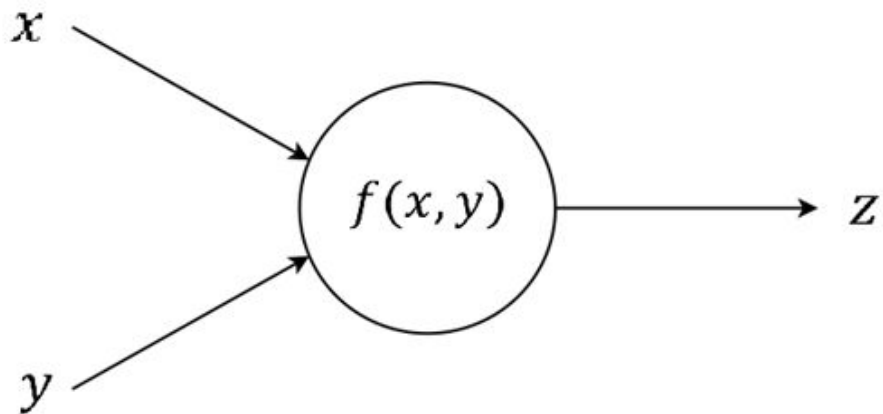
<http://www.asimovinstitute.org/neural-network-zoo/>

# Механизмы работы

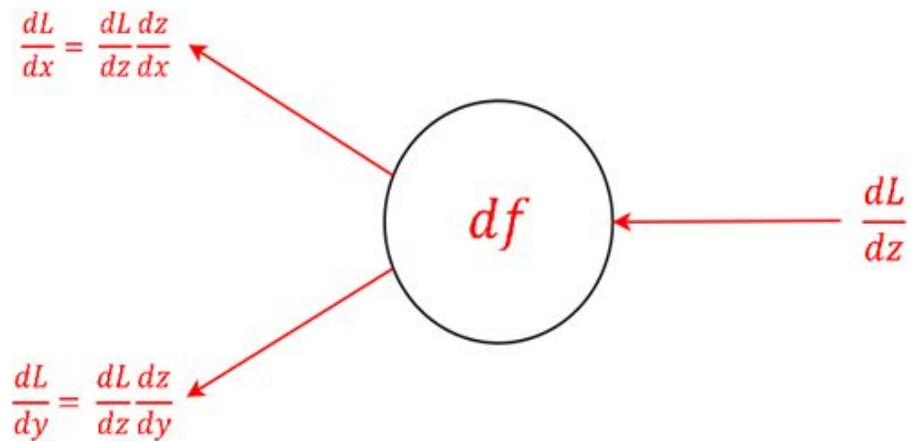


# Два режима работы нейрона и нейросети

Forwardpass



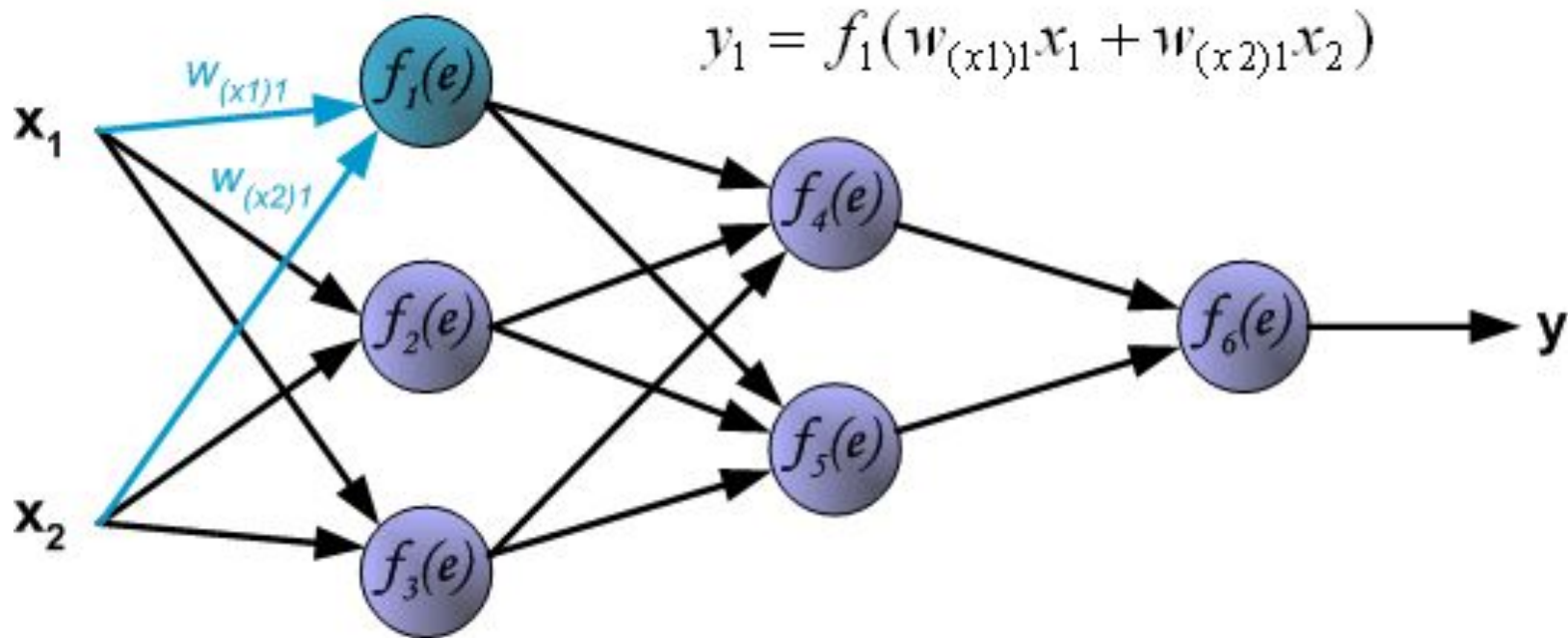
Backwardpass



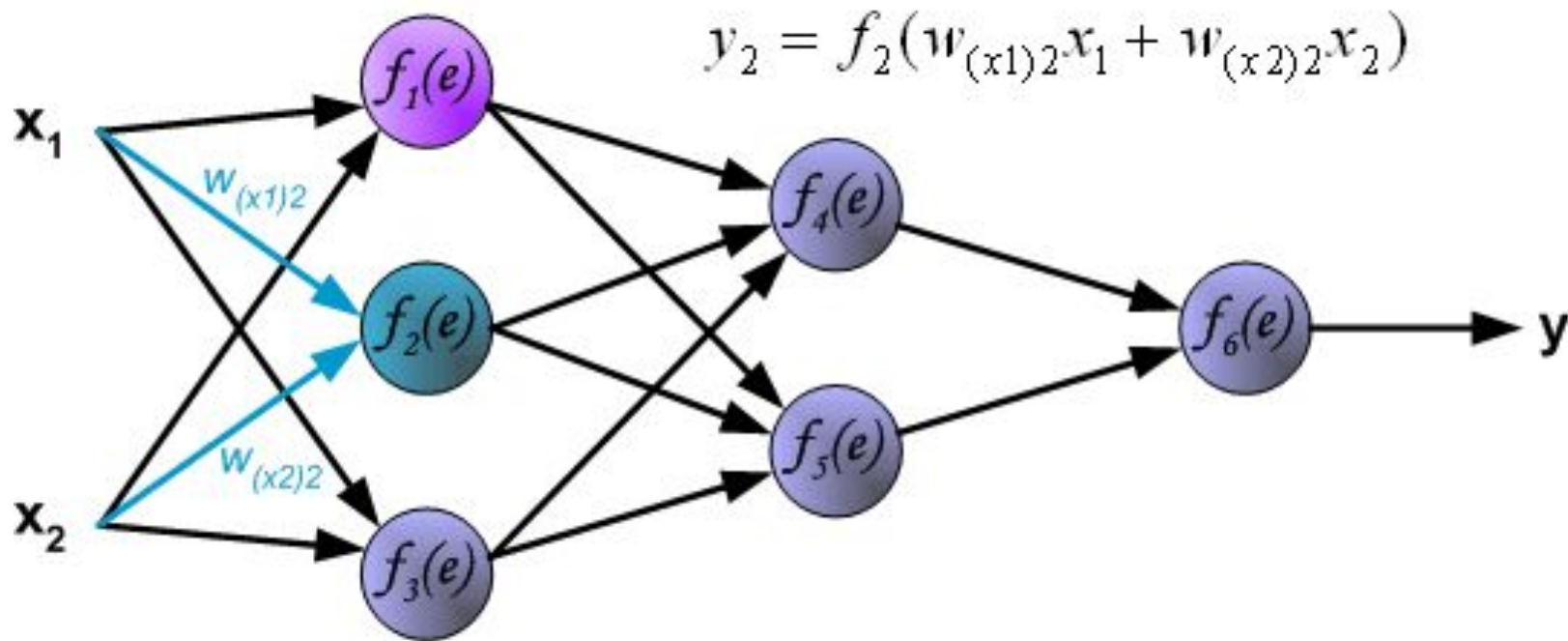
# Механизмы работы

## 1. Forward propagation

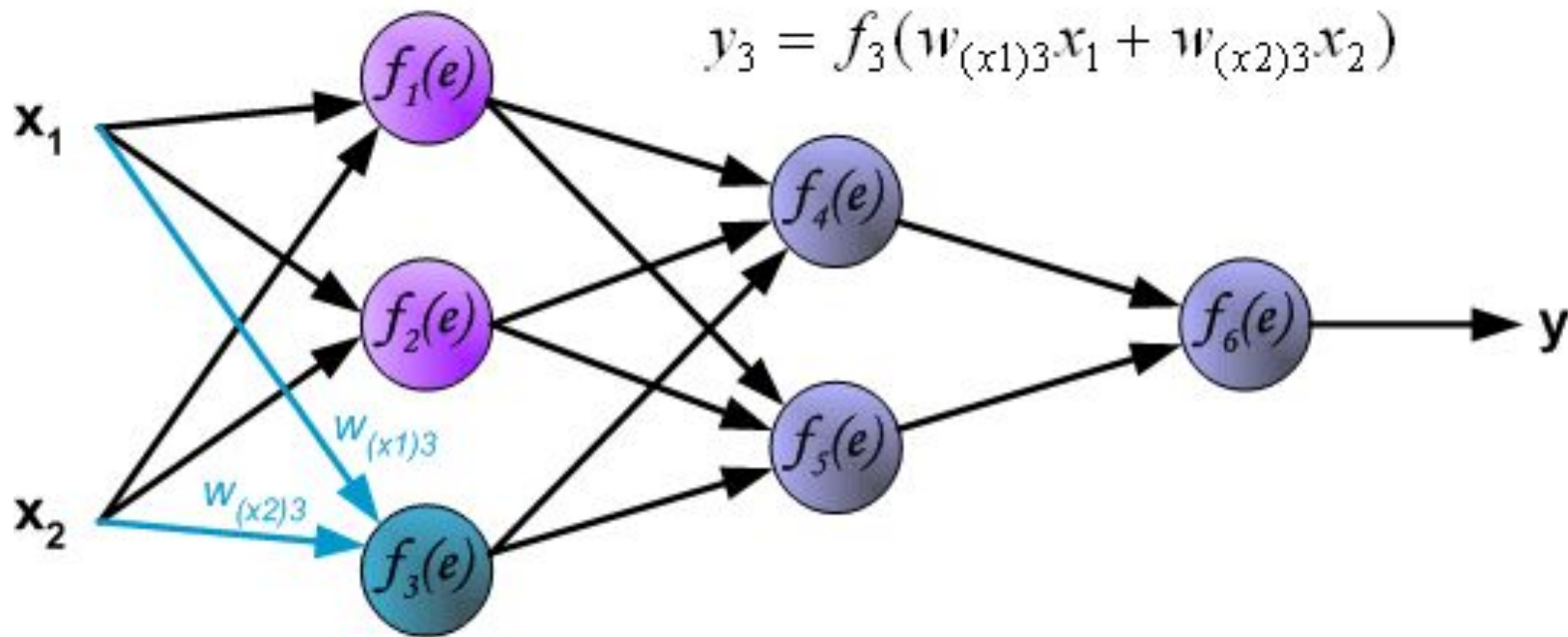
# Forward propagation



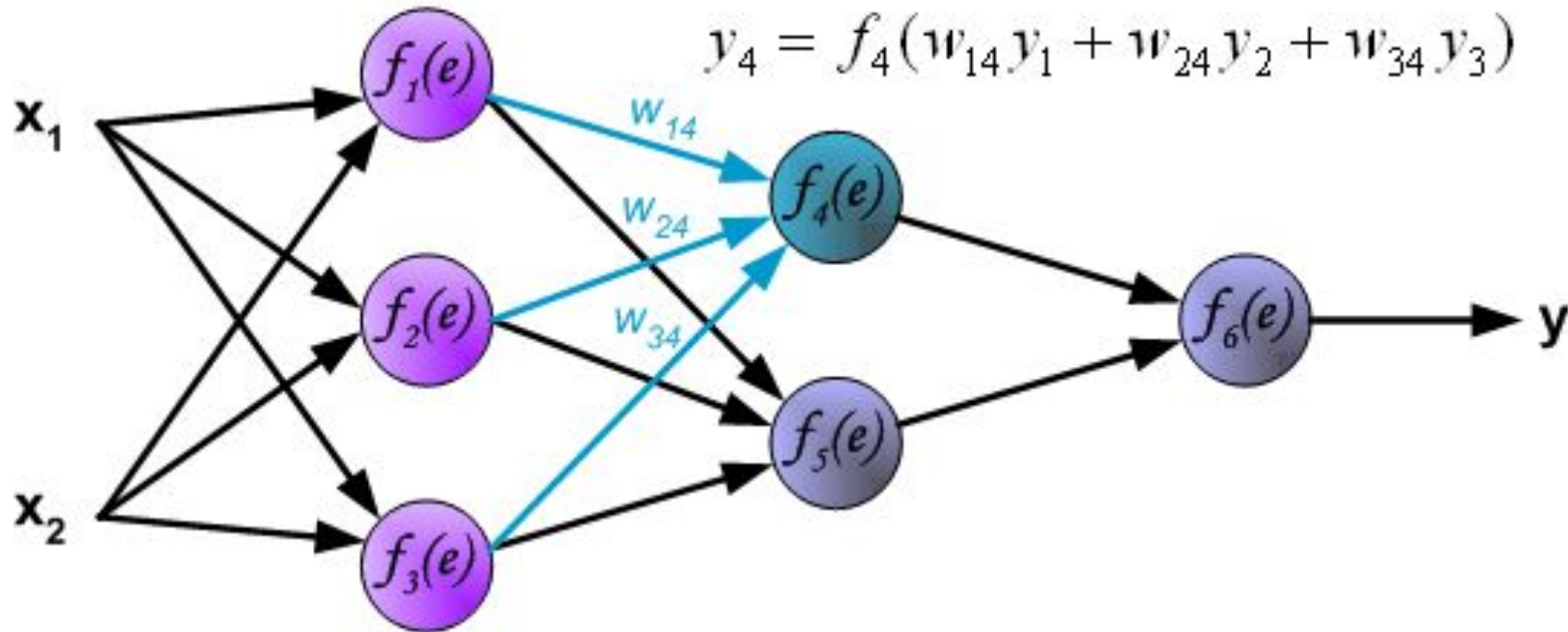
# Forward propagation



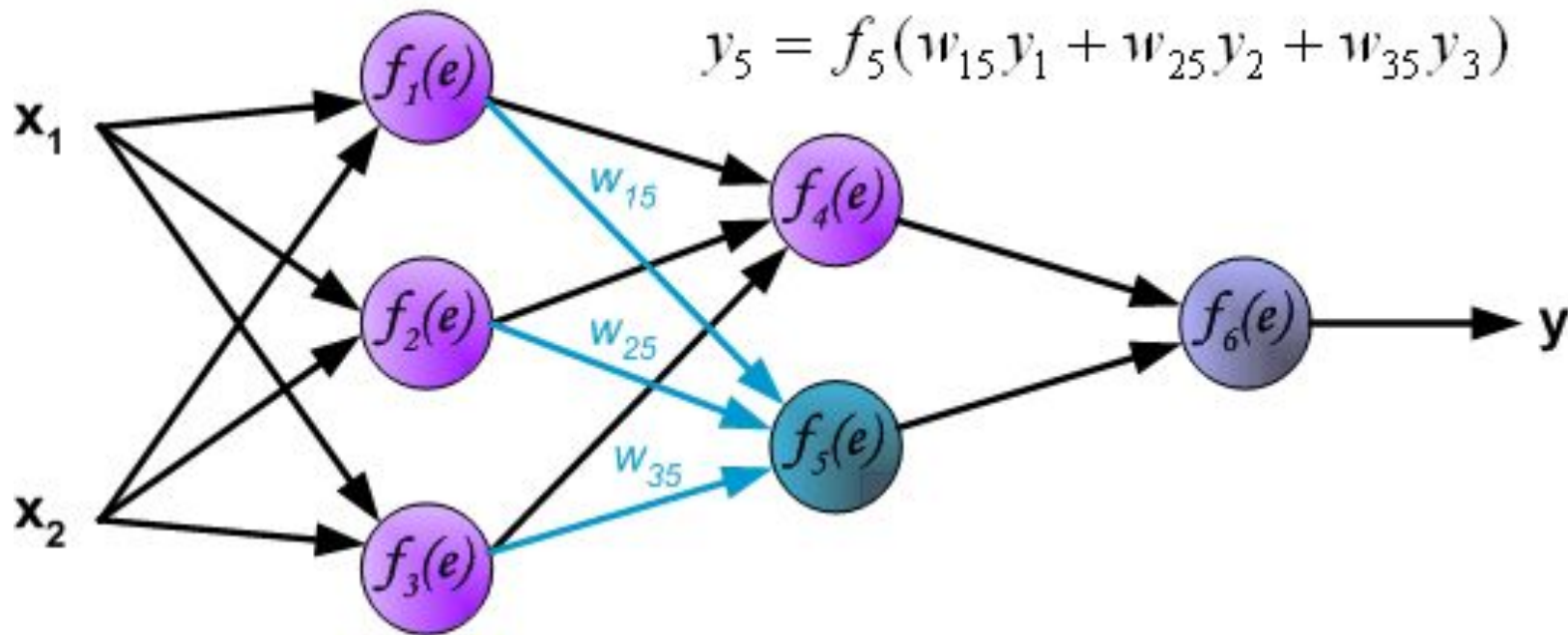
# Forward propagation



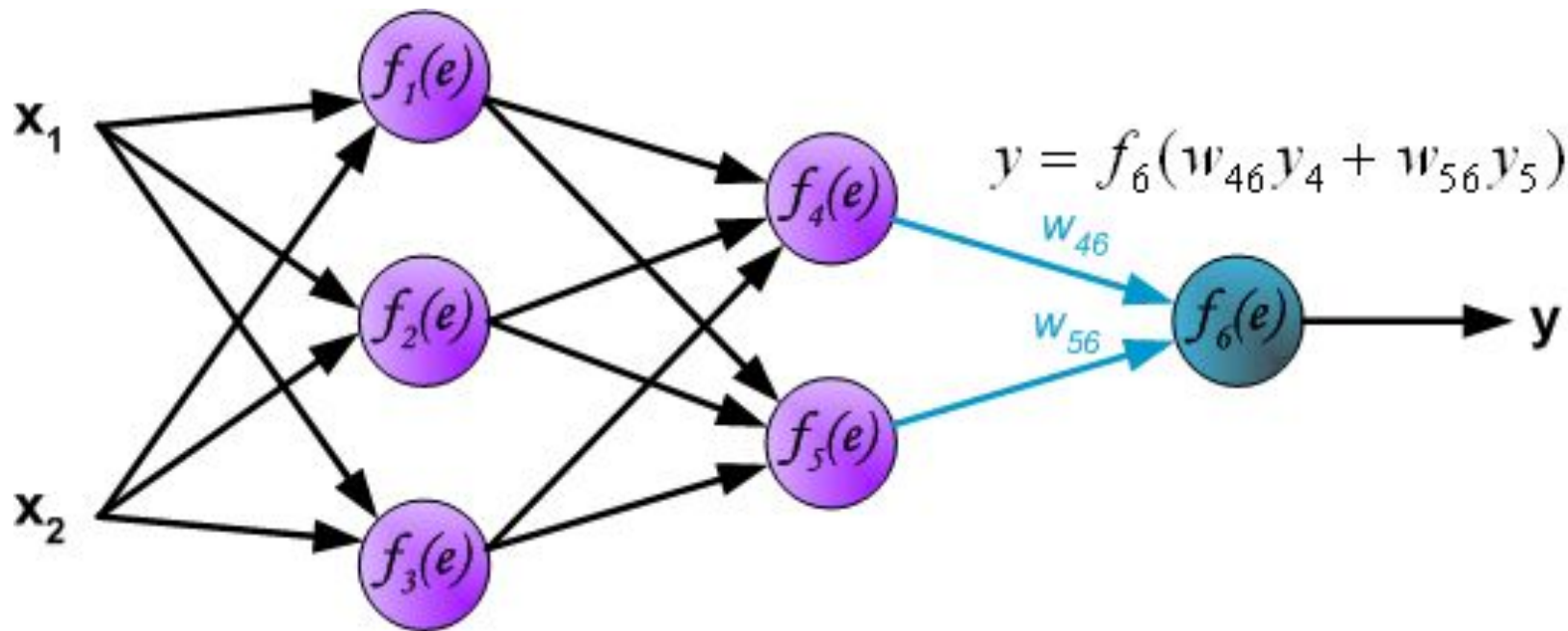
# Forward propagation



# Forward propagation



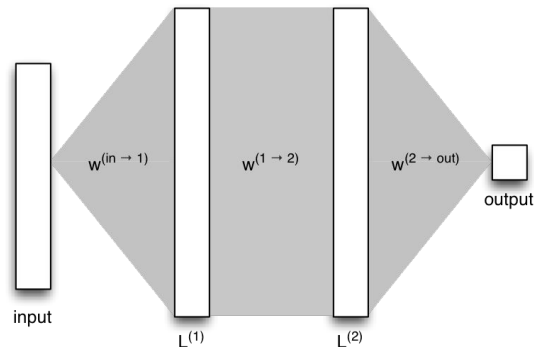
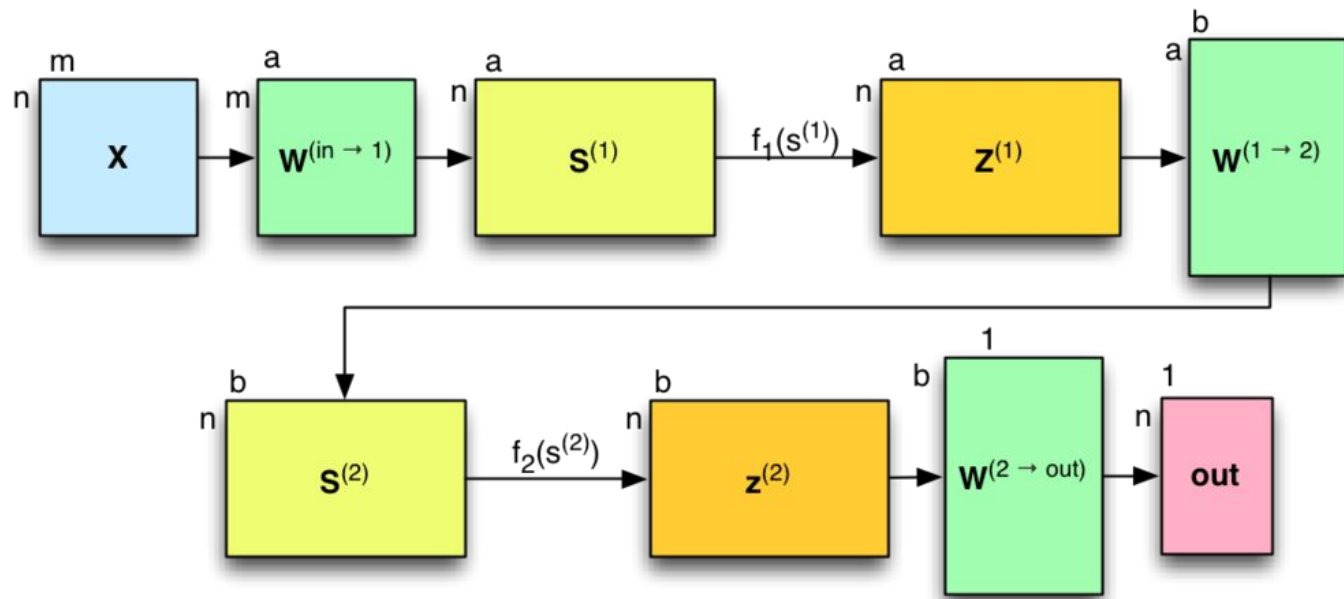
# Forward propagation





# Forward propagation: matrix form

В реальности код векторизуется и вычисляется с помощью матричных операций:



$$S^{(1)} = XW^{(in \rightarrow 1)}$$

$$Z^{(1)} = f_1(S^{(1)})$$

$$S^{(2)} = Z^{(1)}W^{(1 \rightarrow 2)}$$

$$Z^{(2)} = f_2(S^{(2)})$$

$$\hat{y} = f_{out}(Z^{(2)}W^{(2 \rightarrow out)})$$

# Механизмы работы

## 2. Backward propagation

# Оптимизационная задача

**Цель обучения:** например, научиться хорошо классифицировать объекты (задача обучения с учителем, классификация) или предсказывать значения переменной (задача обучения с учителем, регрессия)

**Задача обучения:** минимизировать некую функцию потерь (Loss function, Cost function, Objective), которая отражает близость к идеалу классификации, регрессии или иной задачи.

С помощью хитрого выбора функции потерь можно решать довольно сложные задачи. В будущем увидим это на примере CTC в seq2seq или на примере переноса стиля изображения.

# Функции потерь

В зависимости от типа задачи выбирается архитектура нейросети и в частности функция потерь (<http://cs231n.github.io/neural-networks-2/#losses>)

Примеры для классификации (<http://cs231n.github.io/linear-classify/#loss>)

- Multi-class SVM (hinge) loss  
(or squared hinge loss)

<http://www.pyimagesearch.com/2016/09/05/multi-class-svm-loss/>

$$L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

- Multi-class categorical cross-entropy loss

<http://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/>

$$E = - \sum_i^{n_{class}} t_i \log(y_i)$$

# Функции потерь: регуляризация

К функции потерь может добавляться регуляризация (L1/L2) для предотвращения переобучения.

- L1 loss  
(LASSO)

$$l_{reg}(\vec{w}) = \lambda \sum_{i=1}^N |w_i|$$

- L2 loss  
(Ridge)

$$l_{reg}(\vec{w}) = \lambda \sum_{i=1}^N w_i^2$$

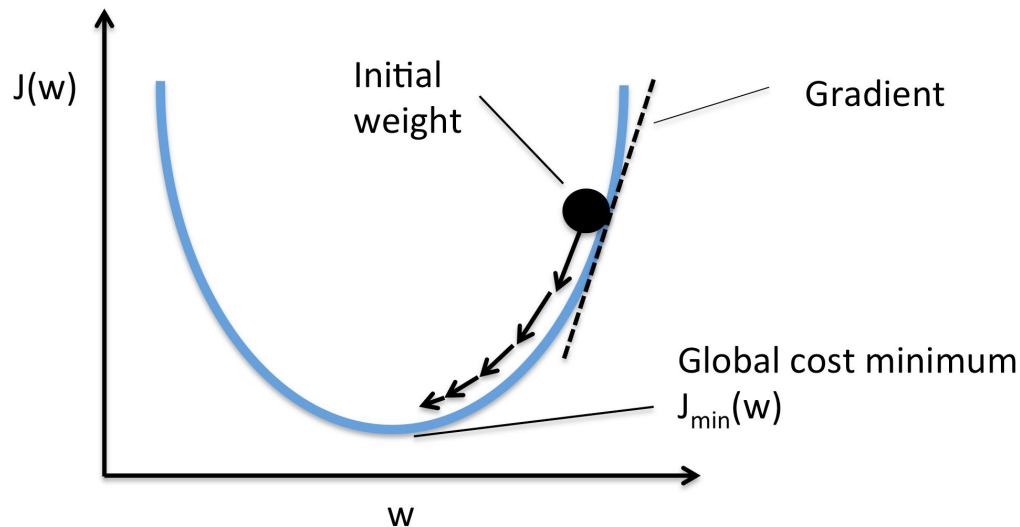
- Elastic-Net

$$l_{reg}(\vec{w}) = \lambda \sum_{i=1}^N \left( \frac{\alpha}{2} w_i^2 + (1 - \alpha) |w_i| \right)$$

# Градиентный спуск

Градиентный спуск — это способ оптимизации дифференцируемых функций.

Идея: если мы можем определить направление наибольшего роста функции потерь (а это и есть градиент этой функции), то, двигаясь в противоположном направлении, мы уменьшим эту функцию (что приблизит нас к идеалу).



# Градиентный спуск

## Обычный градиентный спуск (GD или batch gradient descent)

подразумевает расчёт градиента по полному датасету. Это довольно дорогая процедура, которая приводит к медленной сходимости алгоритма.

Есть несколько разновидностей градиентного спуска:

- **Stochastic gradient descent (SGD, стохастический градиентный спуск)**: обновления происходят на каждом обучающем примере.
- **Mini-batch gradient descent (MB-GD, градиентный спуск с мини-батчами)**: обновления происходят по некоторому числу обучающих примеров. Промежуточный вариант между обычным и стохастическим градиентным спуском.

# Общий вид алгоритма градиентного спуска

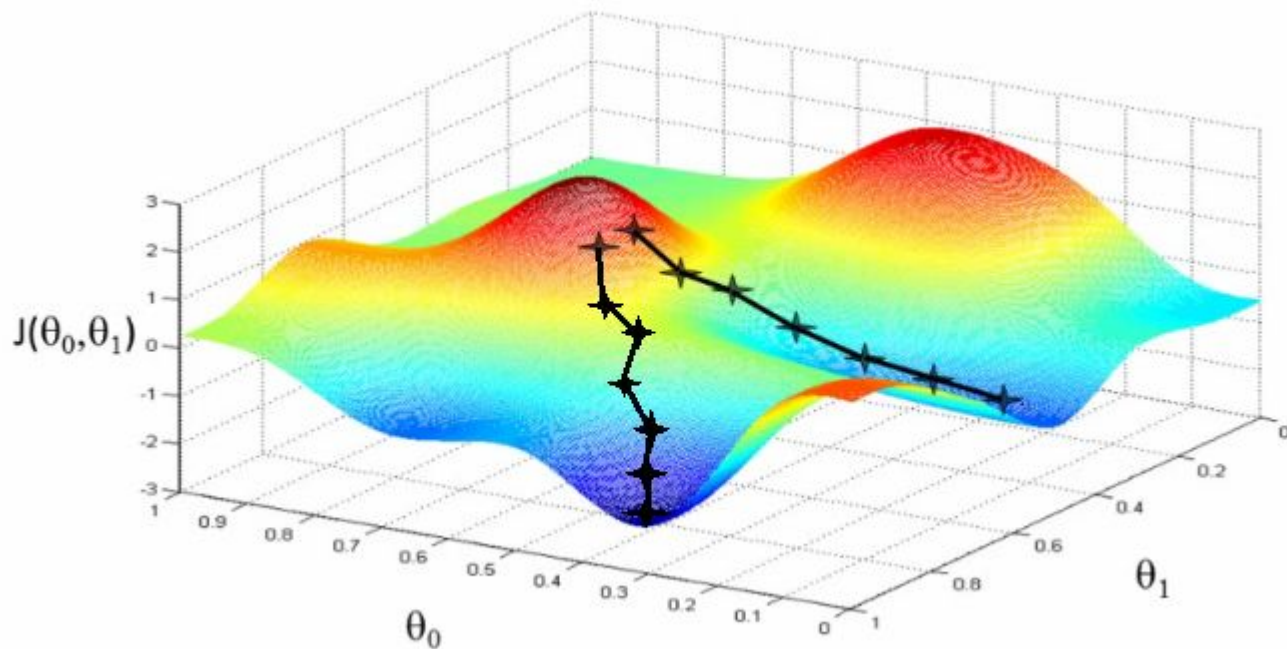
- Задаём параметр  $\eta$  (скорость обучения, learning rate)
- Инициализируем веса нейросети  $W$
- Повторяем пока не достигнут критерий остановки:
  - Получаем мини-батч обучающих примеров  $\langle X, y \rangle$
  - Вычисляем градиент функции потерь  $g$
  - Корректируем веса  $W = W - \eta * g$

Правильный выбор скорости обучения очень важен.



# Градиентный спуск

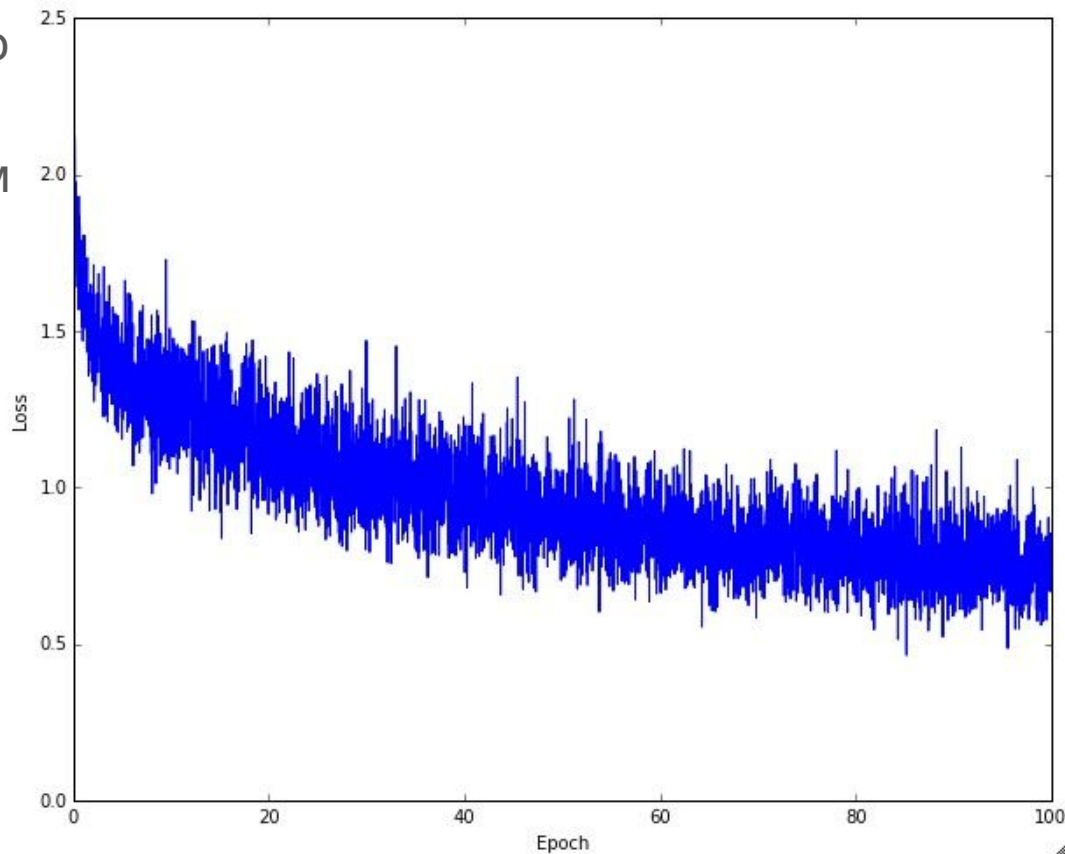
В случае нейросетей ландшафт функции потерь очень сложный, поэтому с градиентным спуском много тонкостей.



# Градиентный спуск

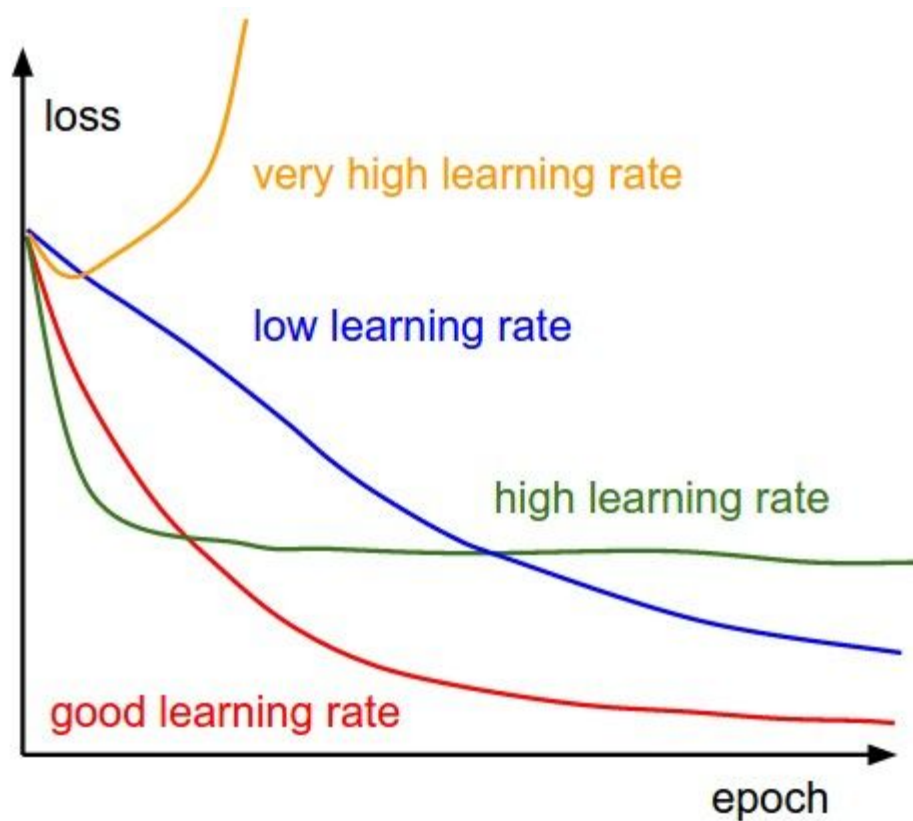
Реальный пример уменьшения со временем функции потерь при обучении нейросети градиентным спуском.

Эпоха = один полный цикл обучения нейросети на всех примерах обучающей выборки.



# Градиентный спуск

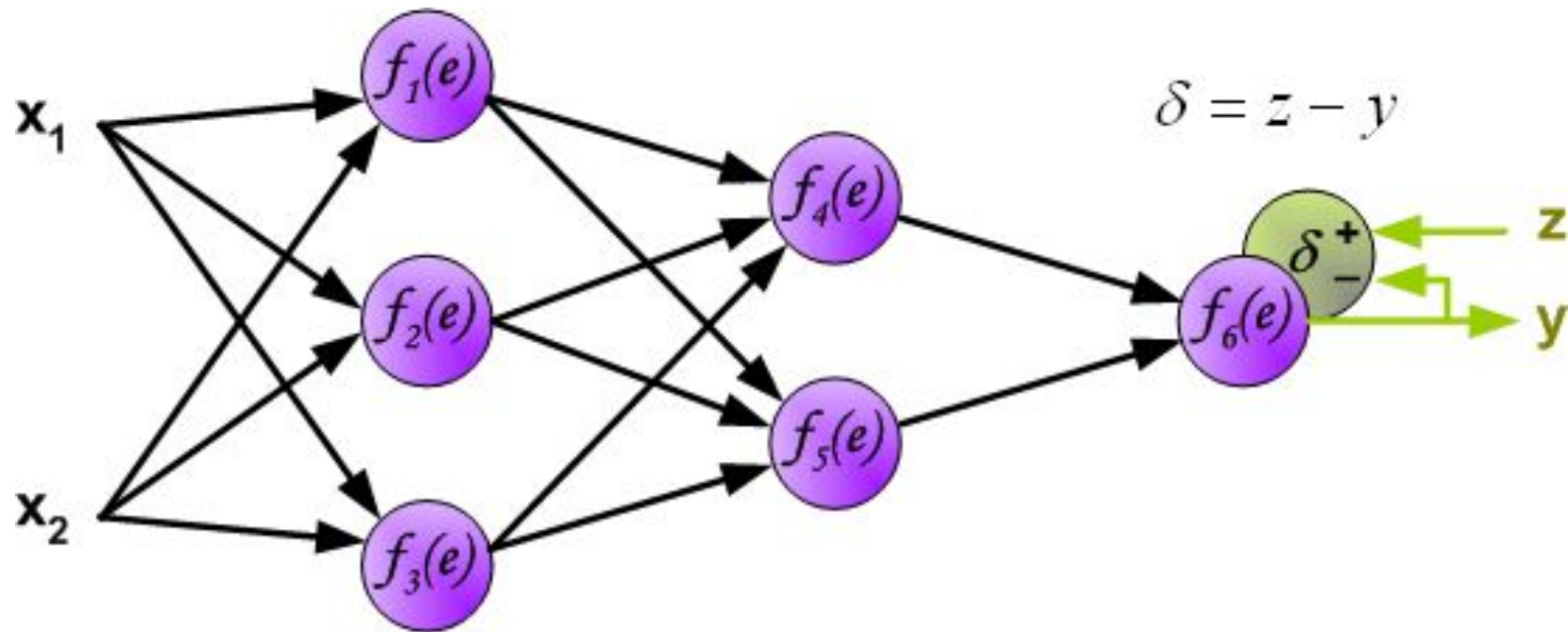
Важность правильного выбора параметра скорости обучения (learning rate) и влияние его на скорость обучения нейросети.



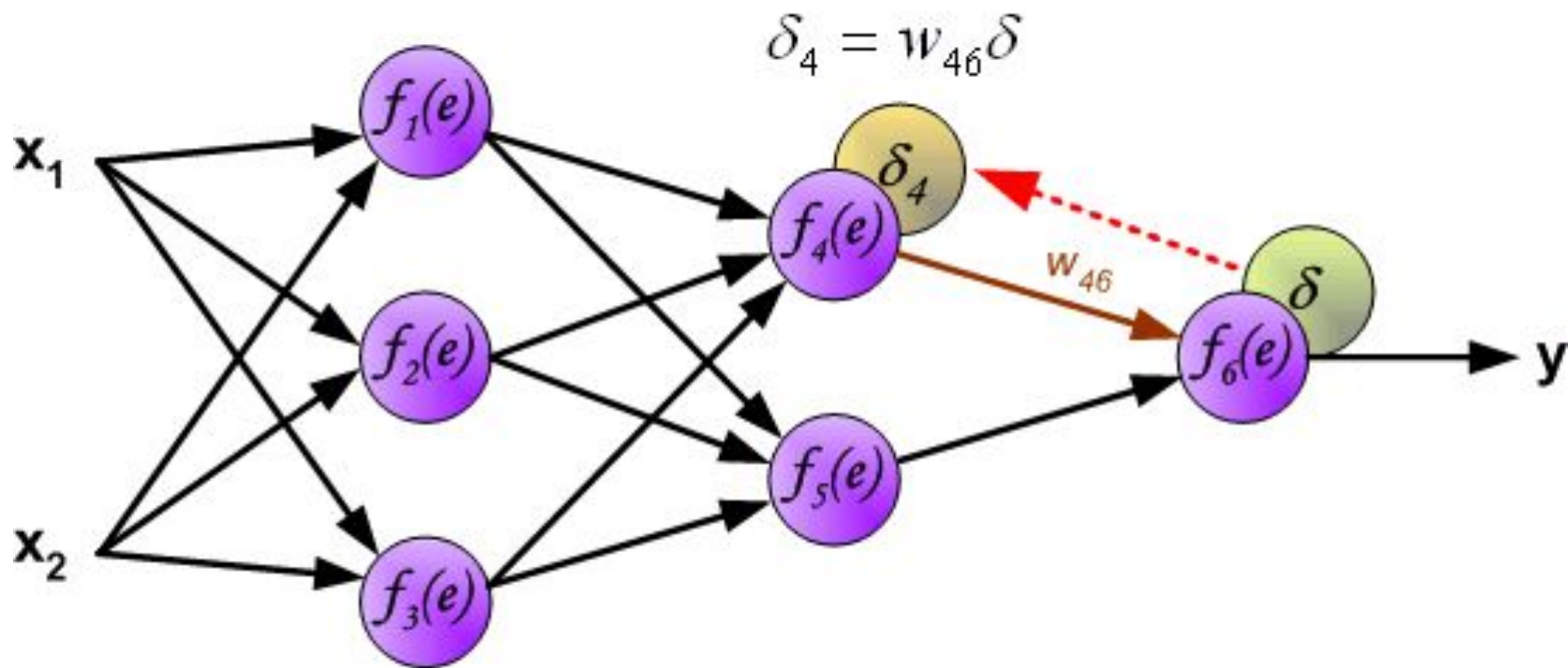
# Алгоритм back-propagation

Алгоритм обратного распространения ошибки (back-propagation, backprop) — это способ расчёта градиента ошибки по каждому из весов в нейросети, чтобы затем каждый вес скорректировать в соответствии с алгоритмом градиентного спуска.

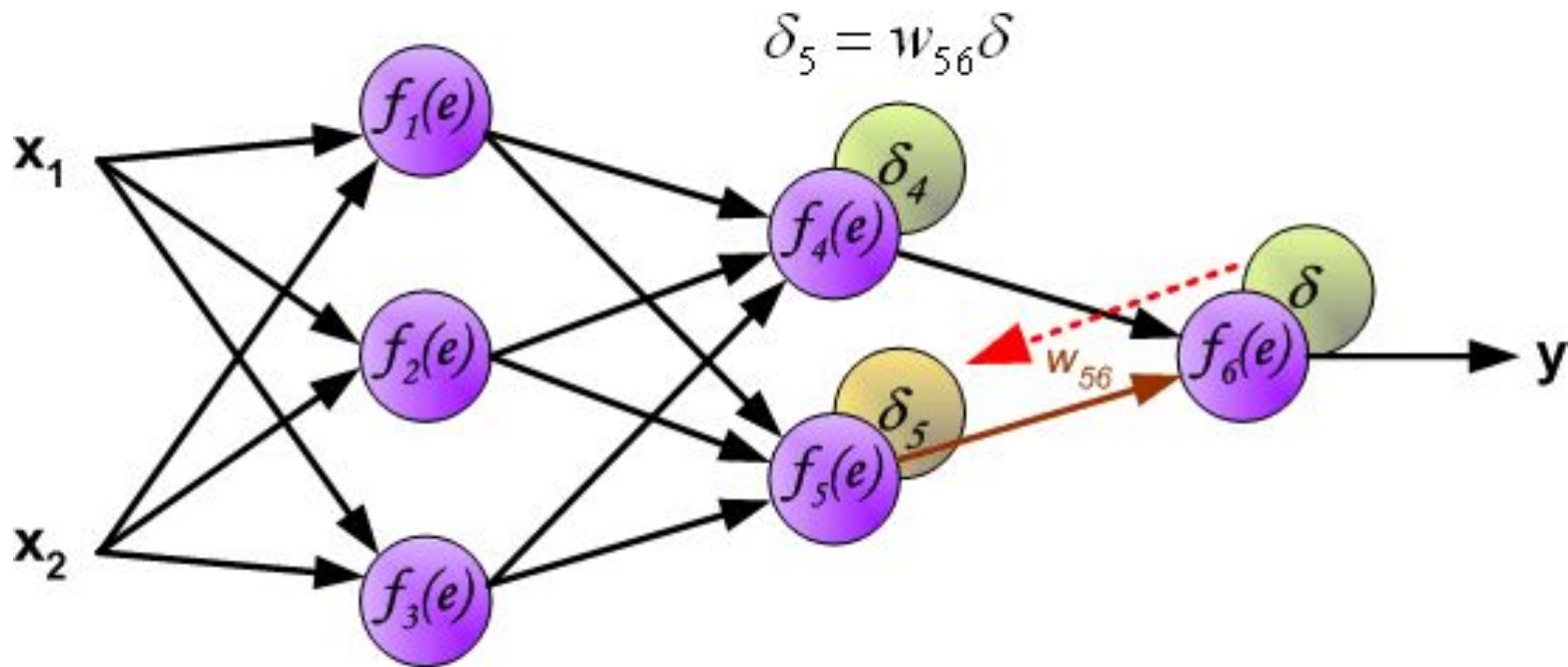
# Backward propagation



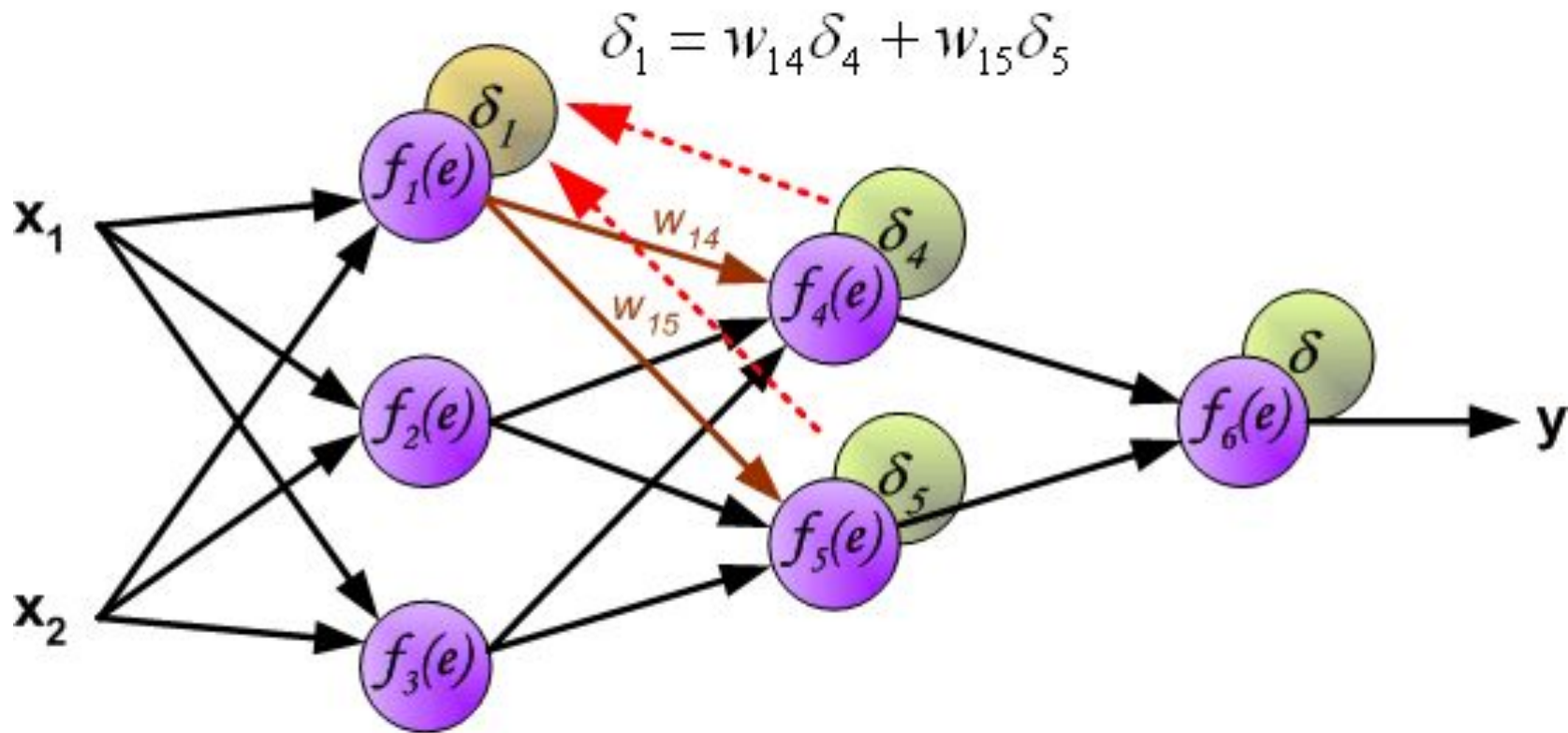
# Backward propagation



# Backward propagation

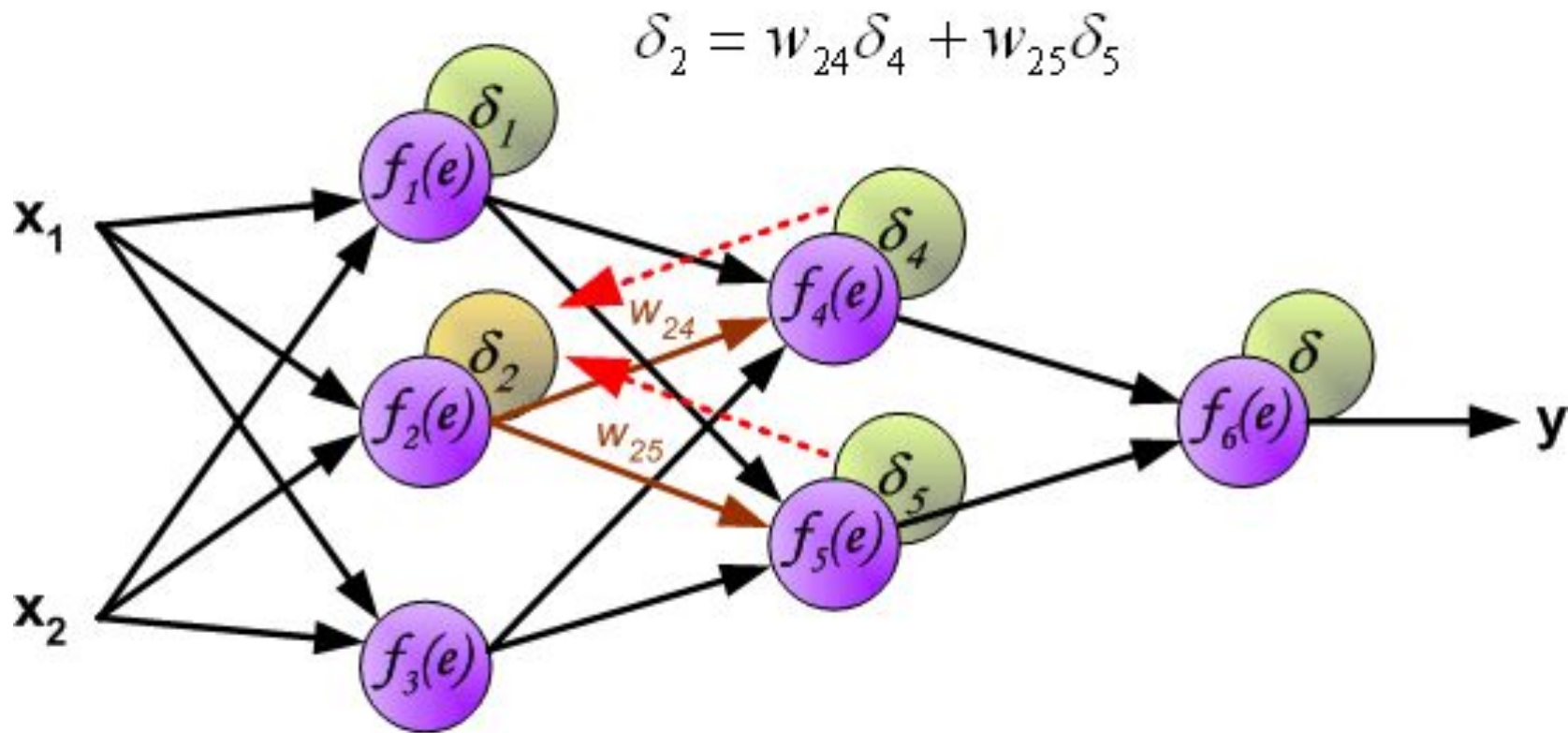


# Backward propagation

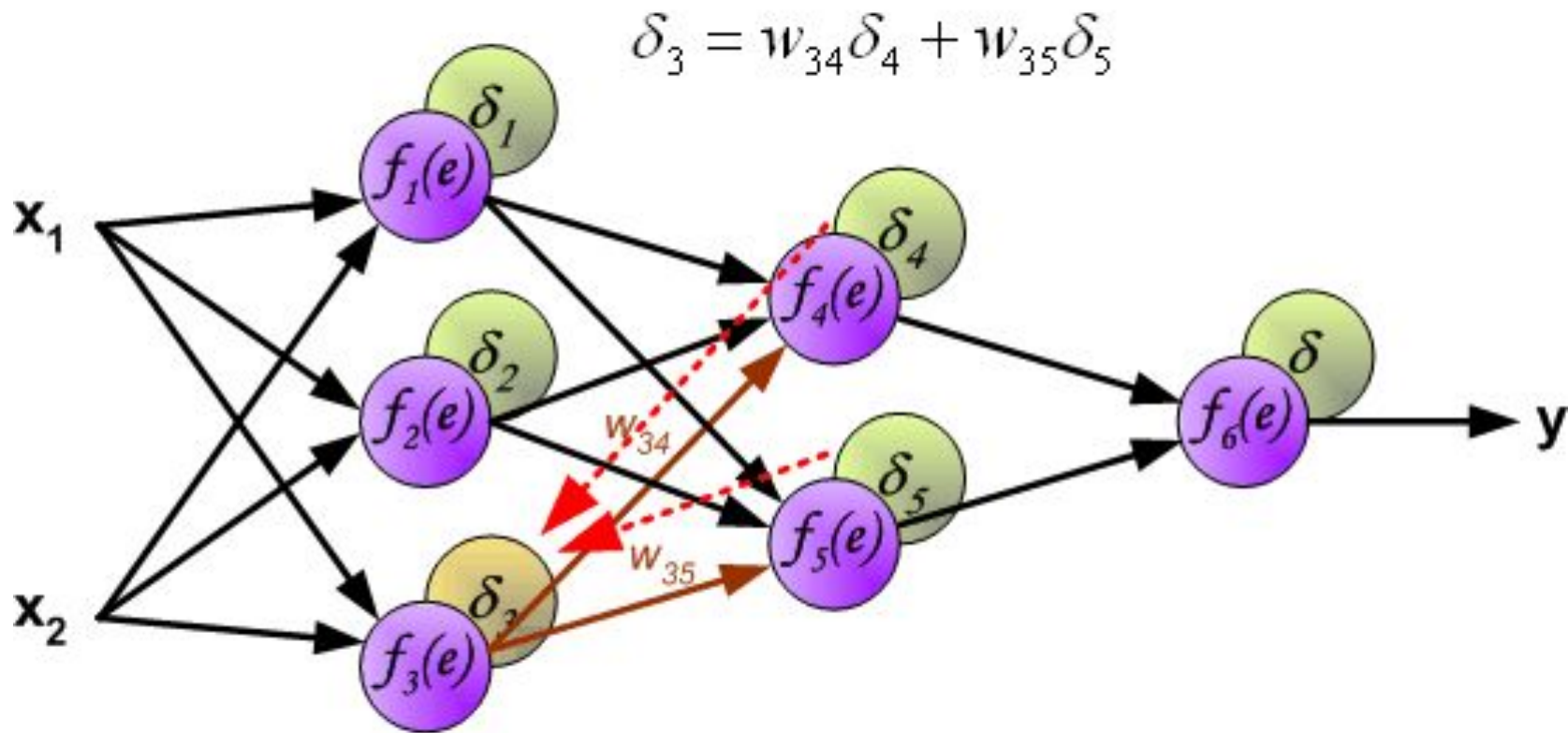




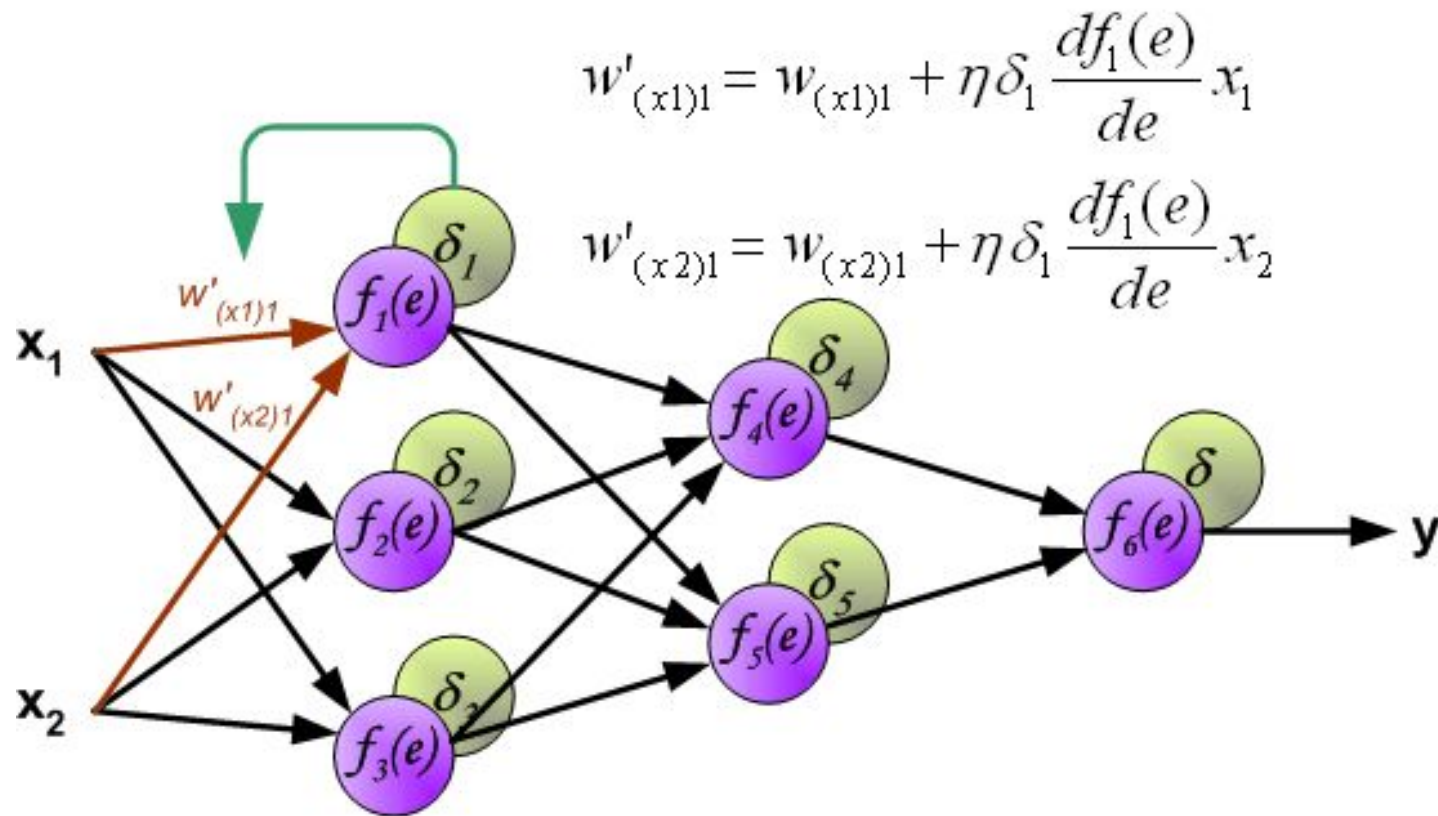
# Backward propagation



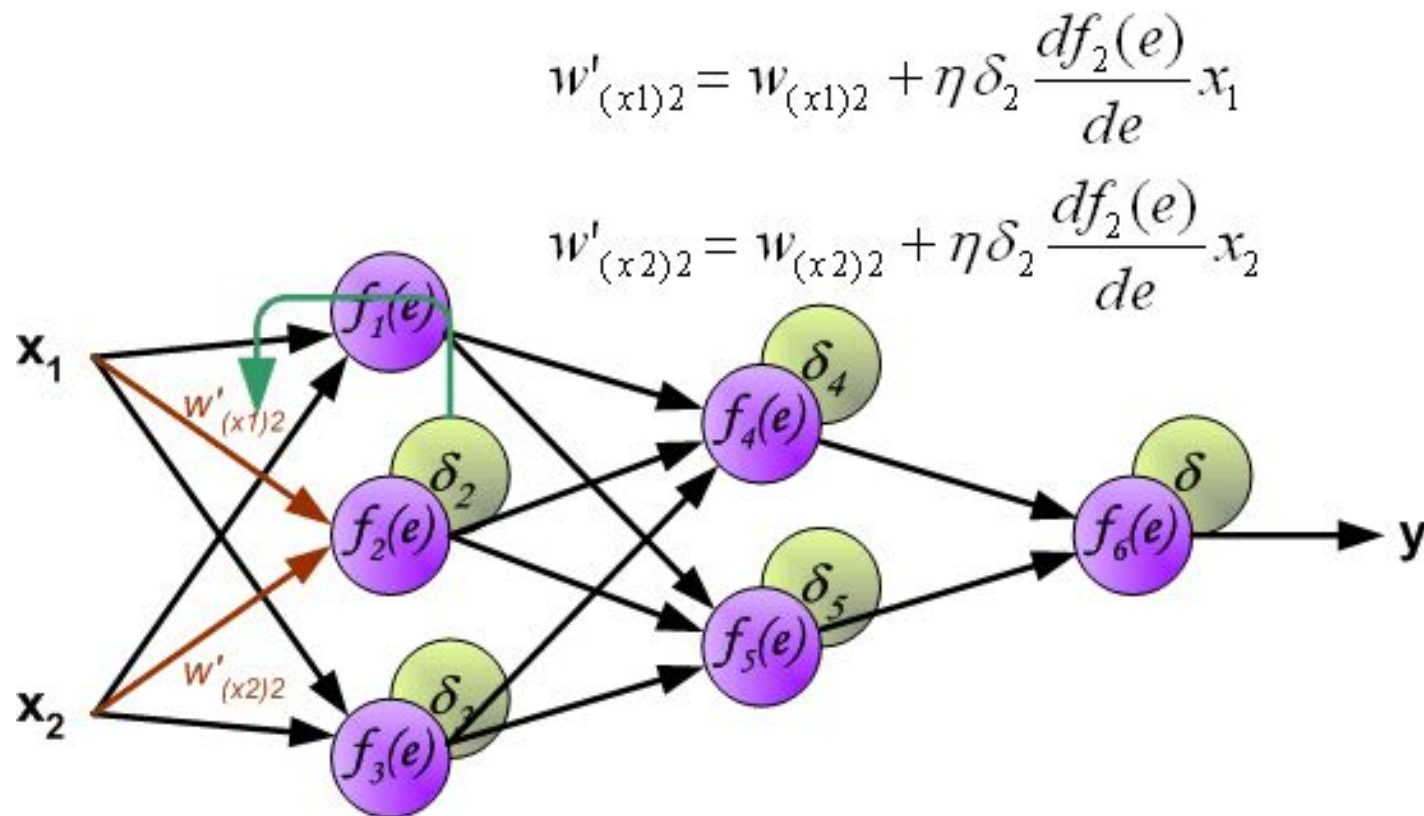
# Backward propagation



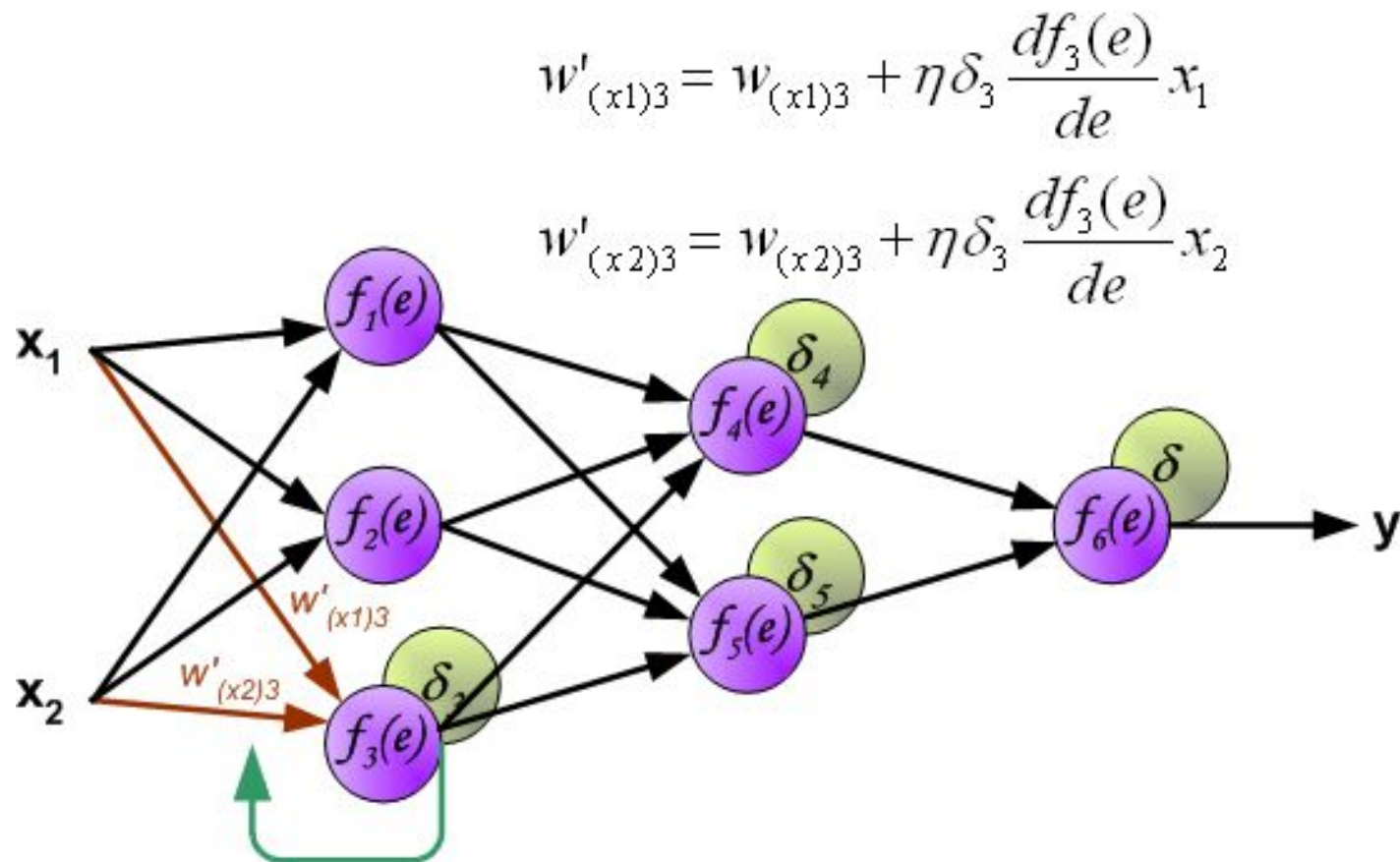
# Backward propagation



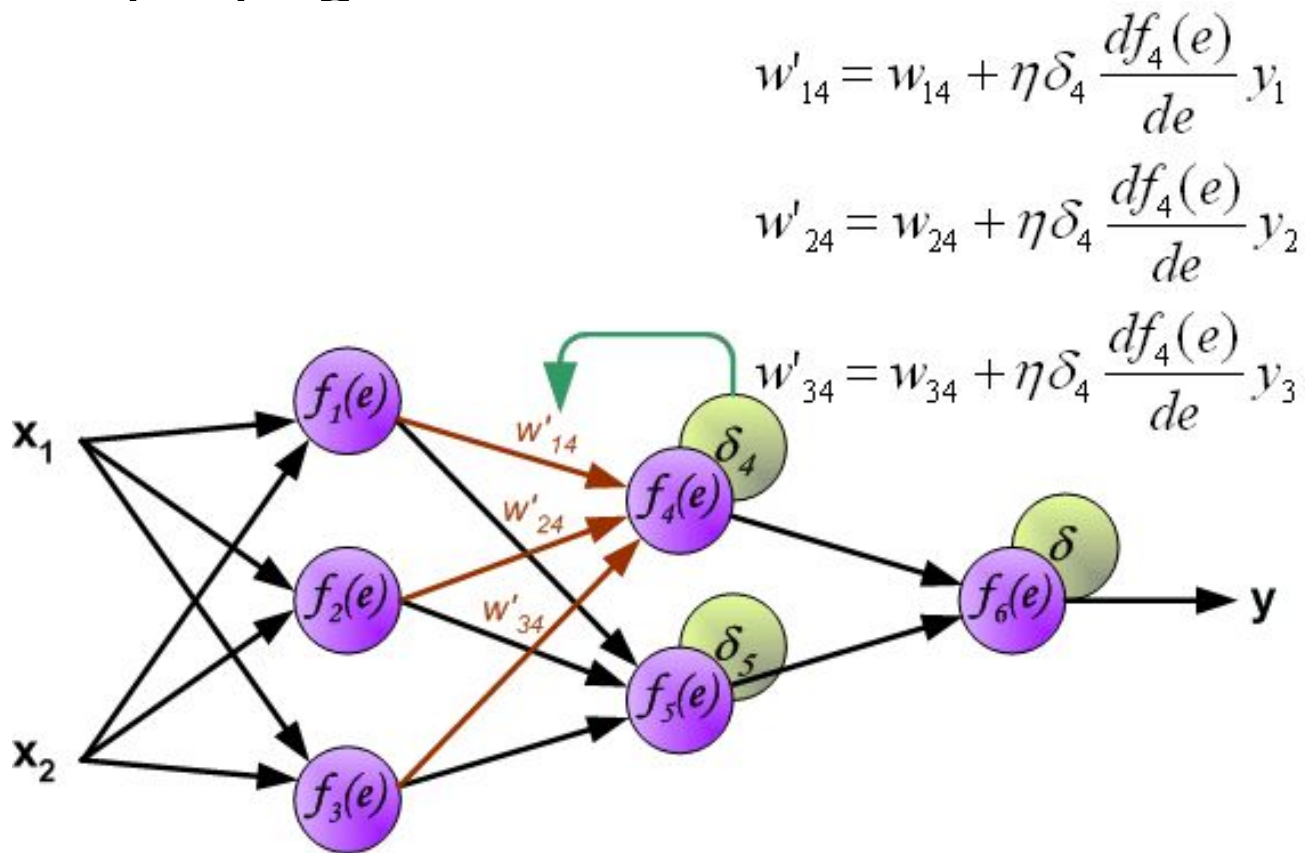
# Backward propagation



# Backward propagation

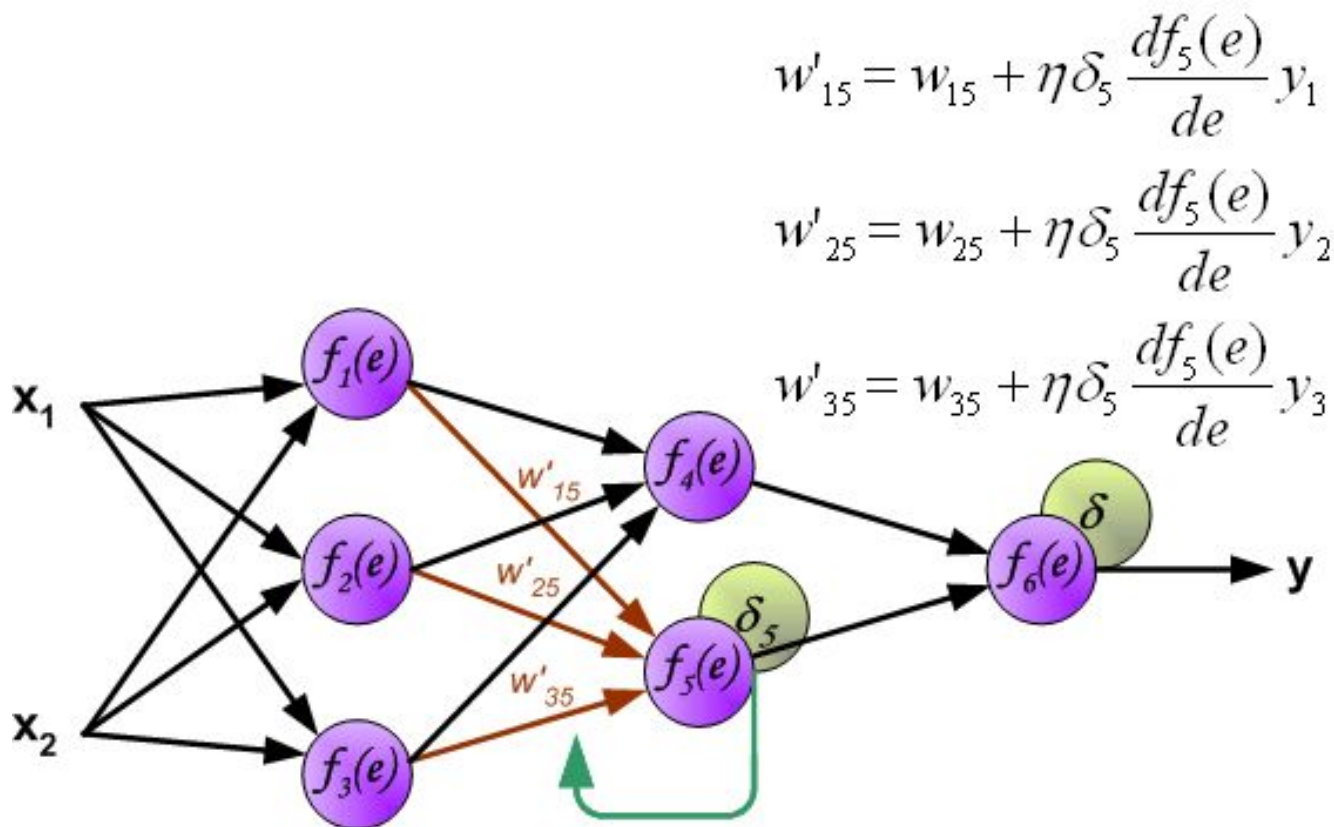


# Backward propagation

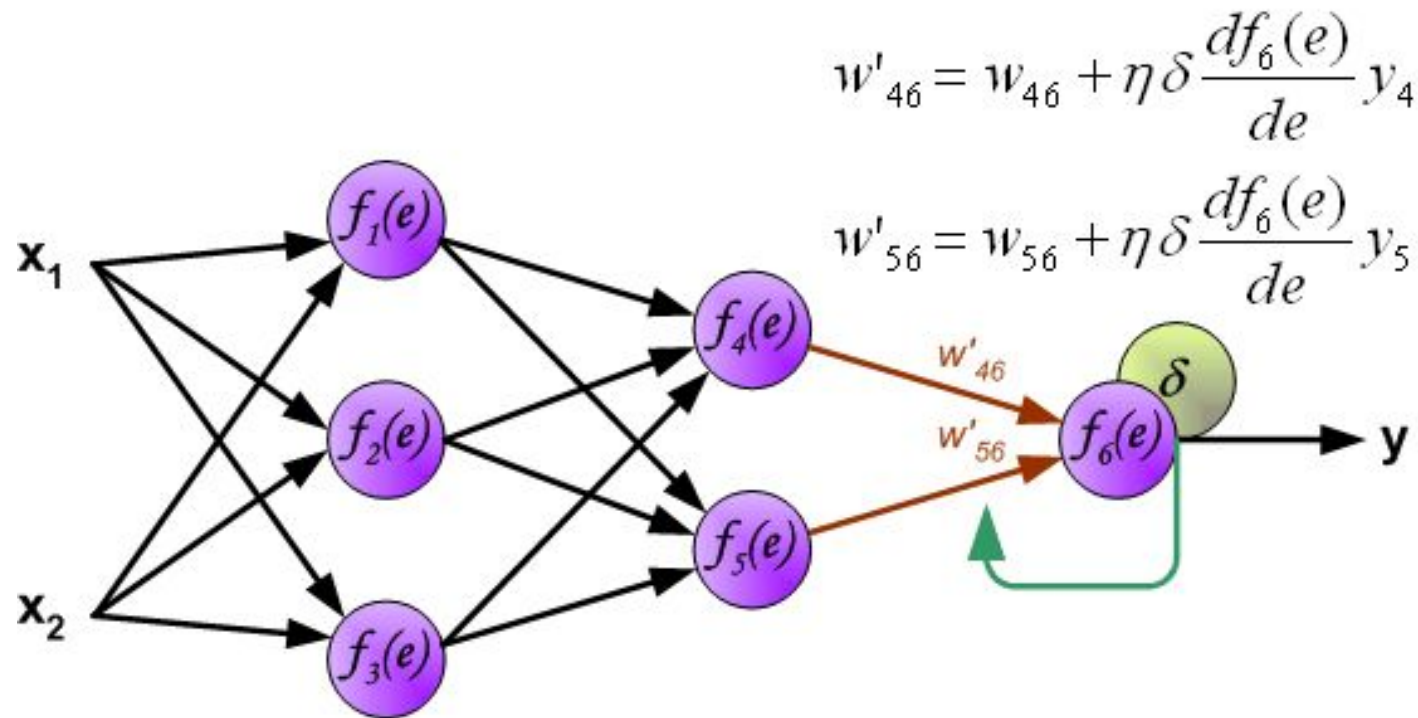




# Backward propagation



# Backward propagation





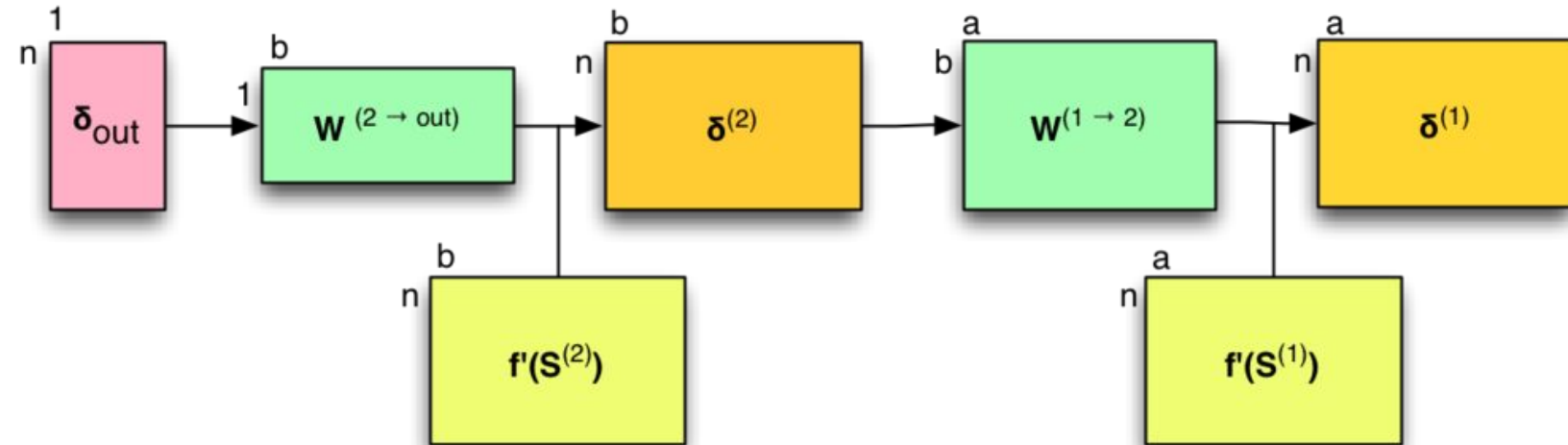
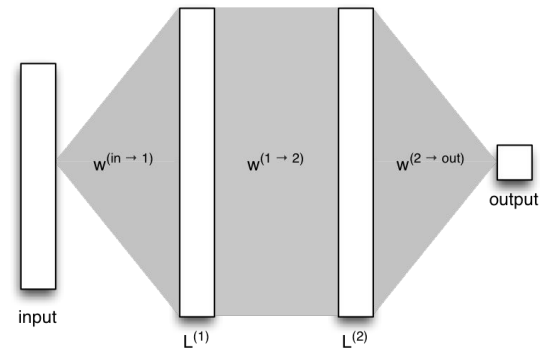
# Backward propagation: matrix form

$$D^{(out)} = (Z^{(out)} - Y)^T$$

$$D^{(i)} = F^{(i)} \odot W^{(i)} D^{(j)}$$

$$\Delta W^{(in \rightarrow 1)} = -\eta (D^{(1)} X)^T$$

$$\Delta W^{(i \rightarrow j)} = -\eta (D^{(j)} Z^{(i)})^T$$



# Дифференцирование

Ещё несколько лет назад программирование нейросети заключалось в ручном задании всех слоёв, функций активации и функции потерь, аналитического расчёта их производных, а также проверки корректности этих производных с помощью численного дифференцирования. Эта процедура была подвержена ошибкам и имела высокий порог входа.

В последние годы необходимости в ручном задании производных нет, так как большинство библиотек поддерживают один из двух вариантов автоматизации этого процесса:

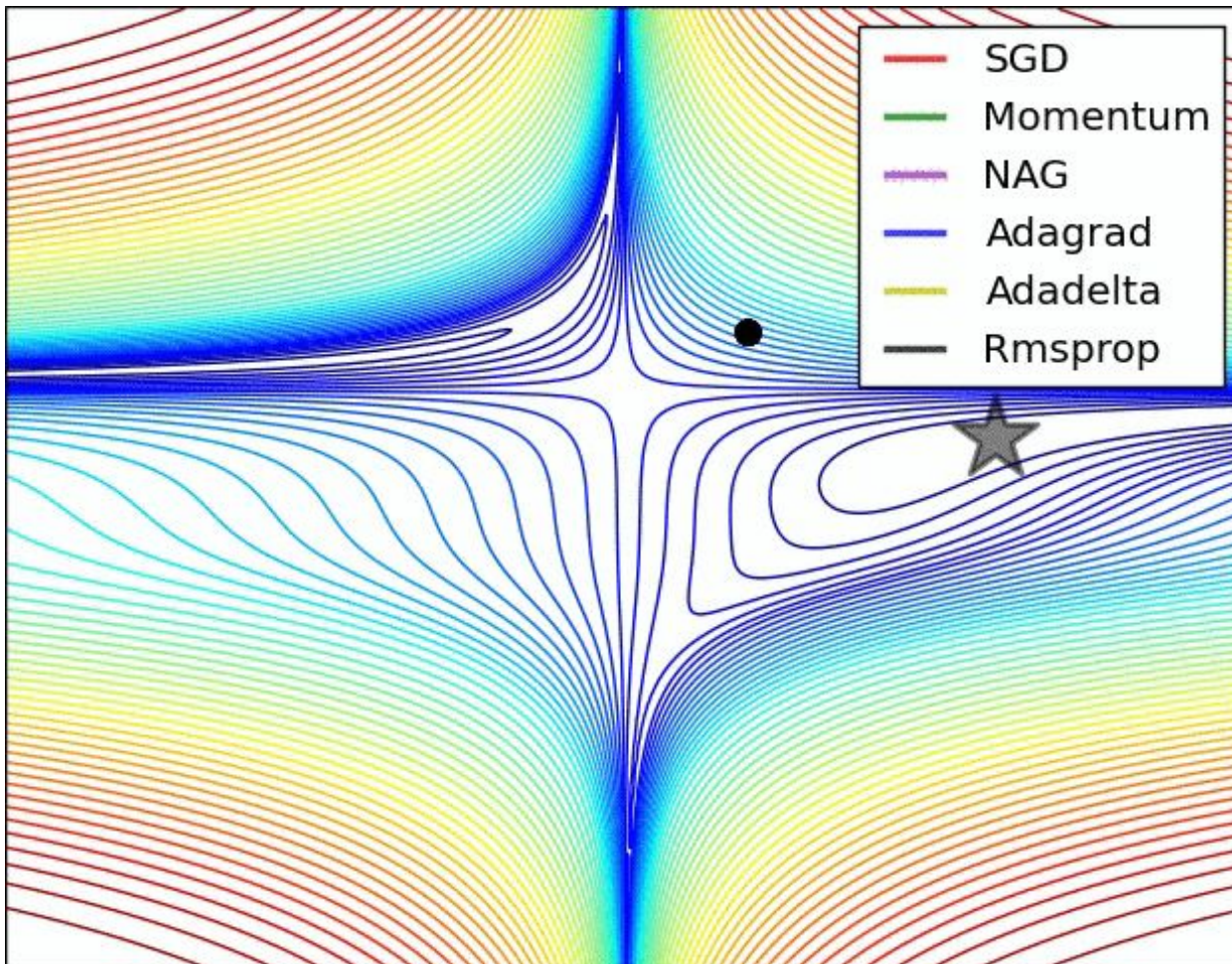
- Символьное дифференцирование (Theano)
- Автоматическое дифференцирование (Tensorflow, Torch)

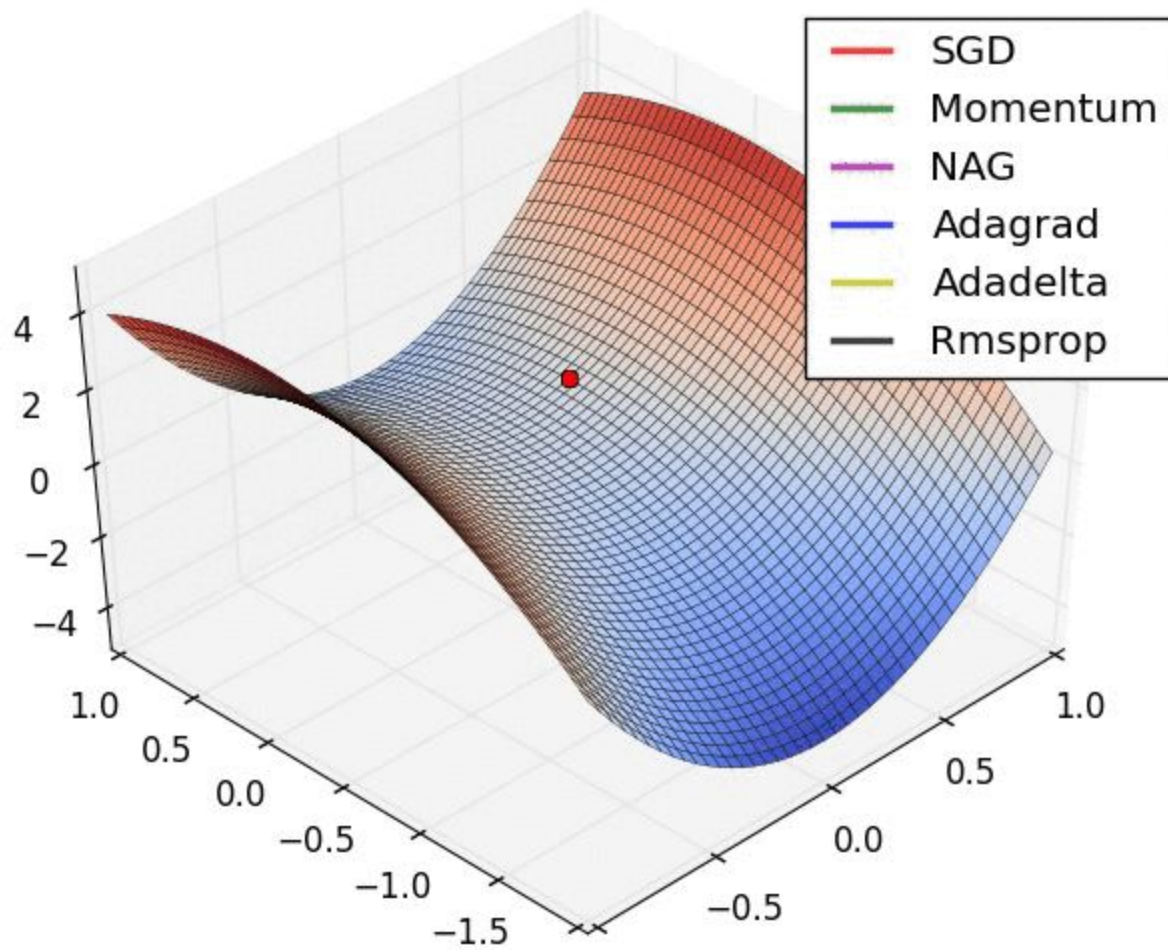
<https://domluna.me/automatic-differentiation-a.k.a-please-save-me-from-backprop/>

# Вариации и улучшения backprop

Существует множество модификаций backprop

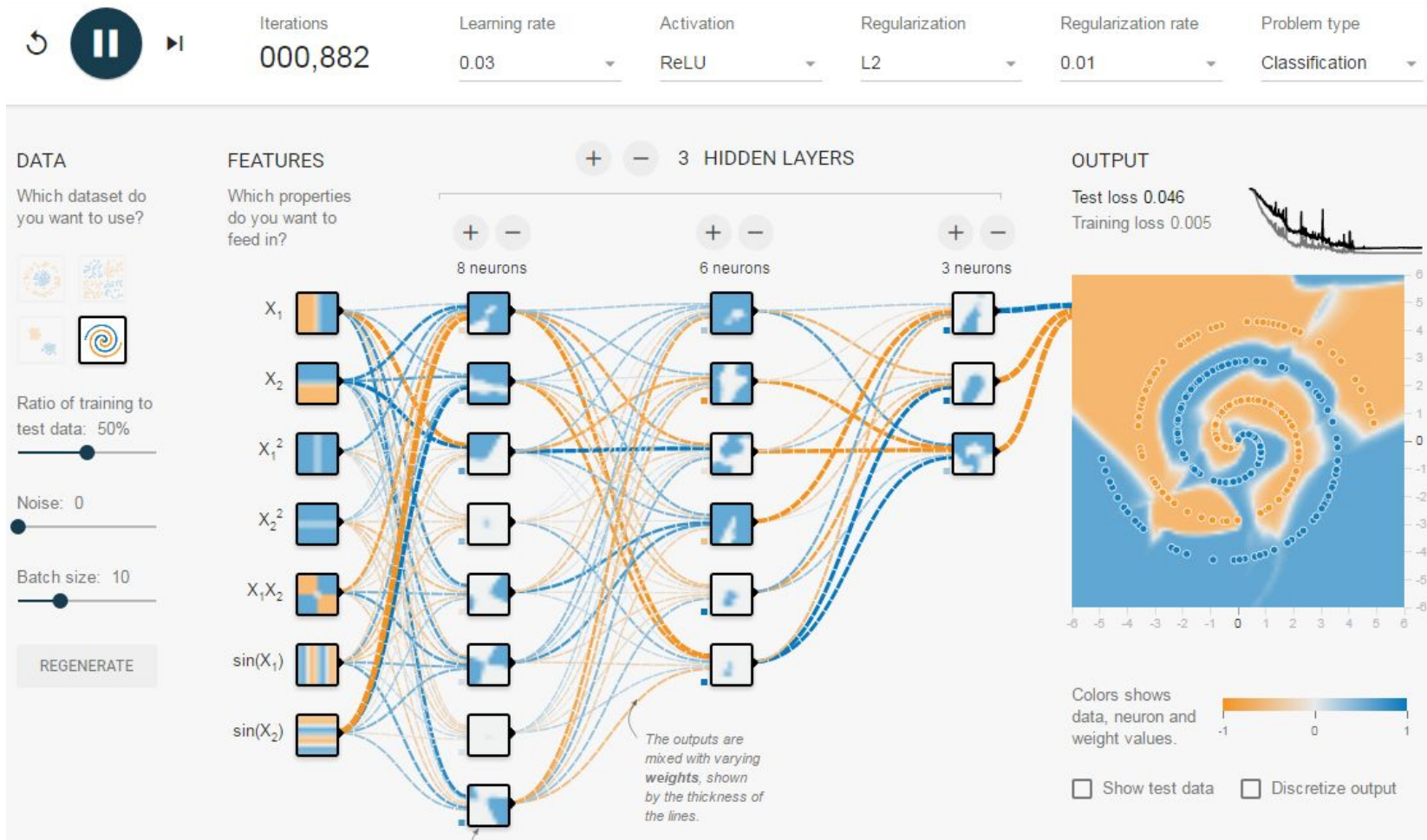
- Изменяемый learning rate (постепенное уменьшение, уменьшение на порядок каждые N эпох, ...)
- Использование момента инерции (momentum update, Nesterov momentum)
- Адаптивные методы (Adagrad, Adadelata, RMSprop, Adam, ...)
- Методы второго порядка (L-BFGS, Conjugate gradient, Hessian Free)





Демонстрация работы





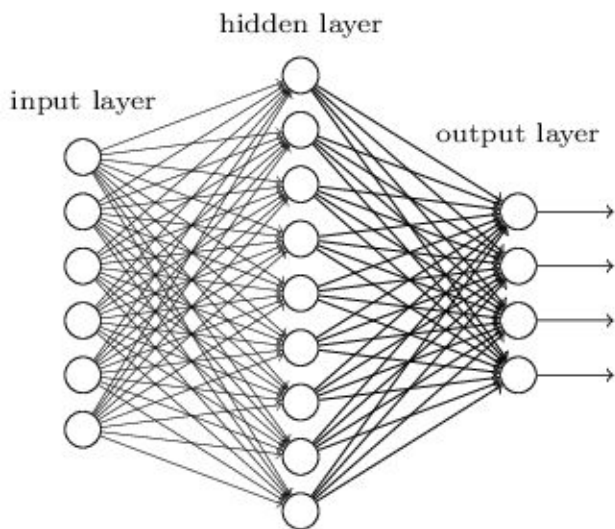
<http://playground.tensorflow.org/>

Классические нейросети  
прямого распространения  
(Feed-Forward Neural Networks, FNN)

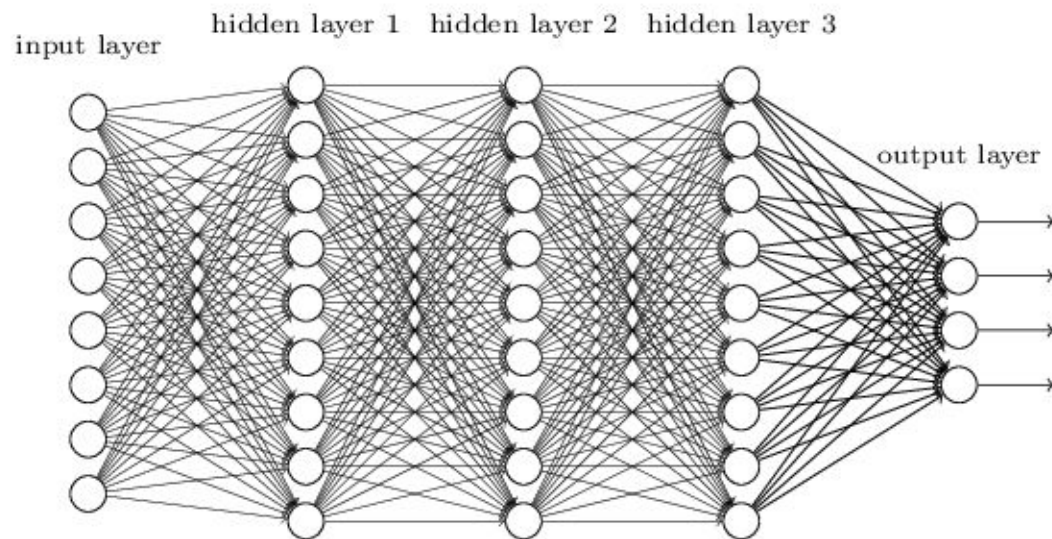


# Полносвязная нейросеть

"Non-deep" feedforward  
neural network



Deep neural network

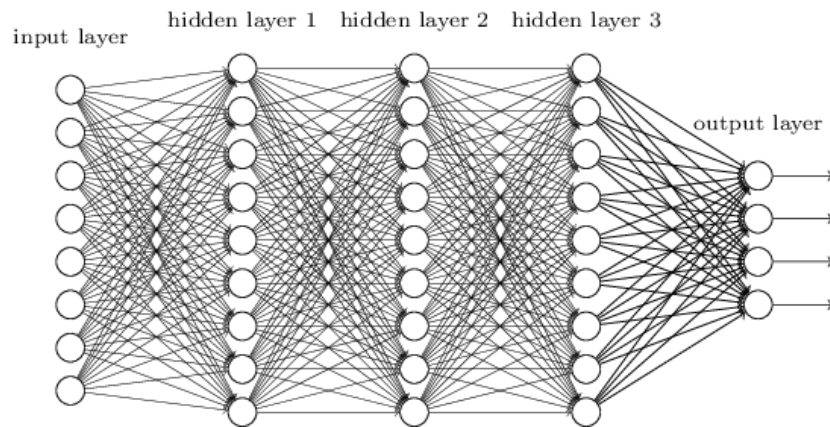


# Multilayer Perceptron (MLP)

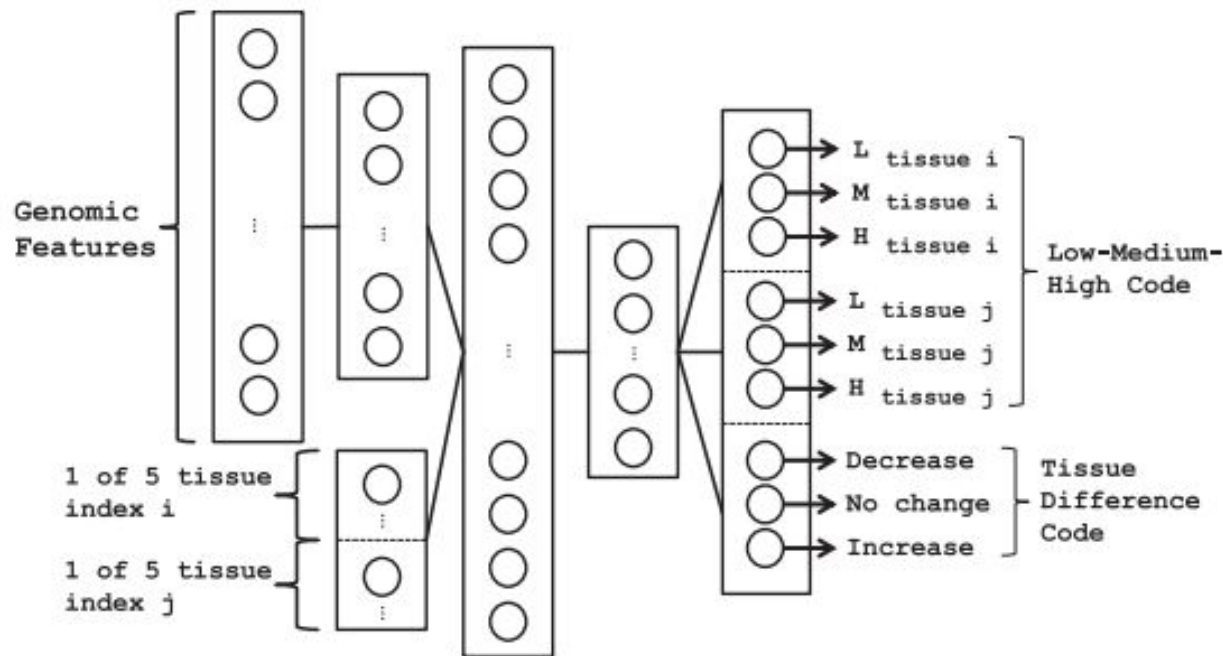
Классика FNN. Хорошо работают для классификации, но есть трудности:

- Много параметров
  - Для сети, у которой на входе картинка 100x100, три скрытых слоя по 100 нейронов каждый, и выходом на 10 классов, число параметров будет примерно 1M  
( $10000 \cdot 100 + 100 \cdot 100 + 100 \cdot 100 + 100 \cdot 10$ )
- Затухающие градиенты (если слоёв много)

Как следствие — трудно обучать.



# Пример с MLP



**Fig. 1.** Architecture of the DNN used to predict AS patterns. It contains three hidden layers, with hidden variables that jointly represent genomic features and cellular context (tissue types)

Deep learning of the tissue-regulated splicing code

<http://www.psi.toronto.edu/publications/2014/DeepSplicingCode.pdf>

# Пример с MLP

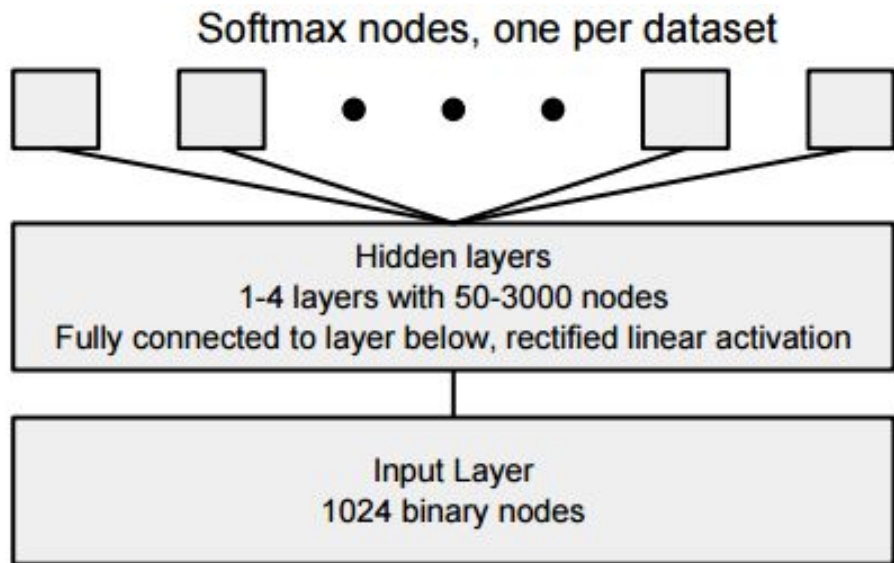
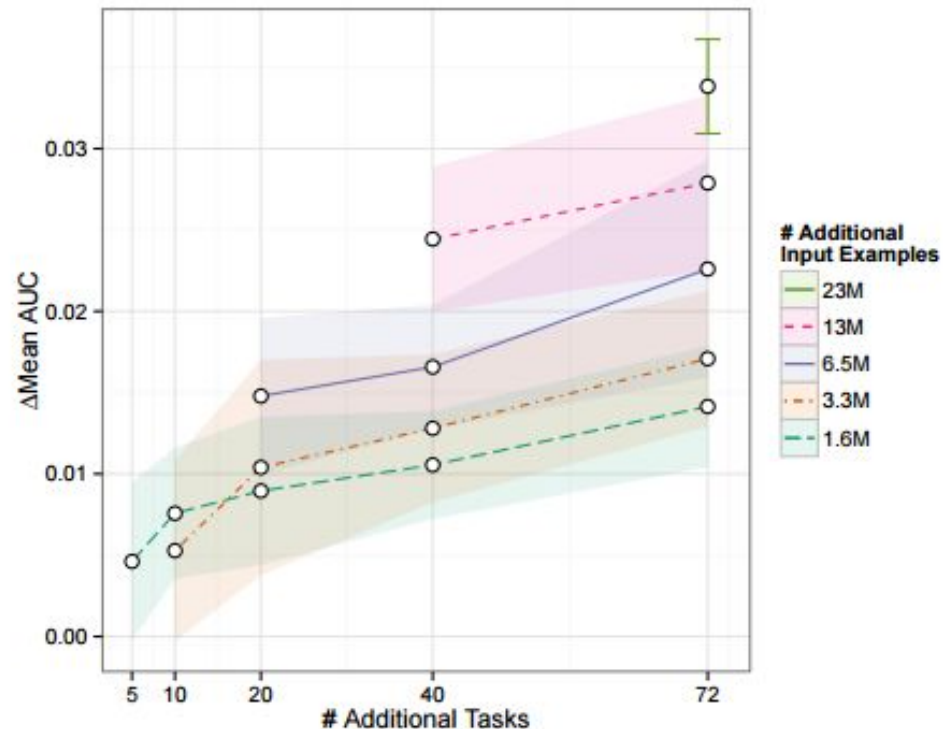


Figure 1. Multitask neural network.



Massively Multitask Networks for Drug Discovery

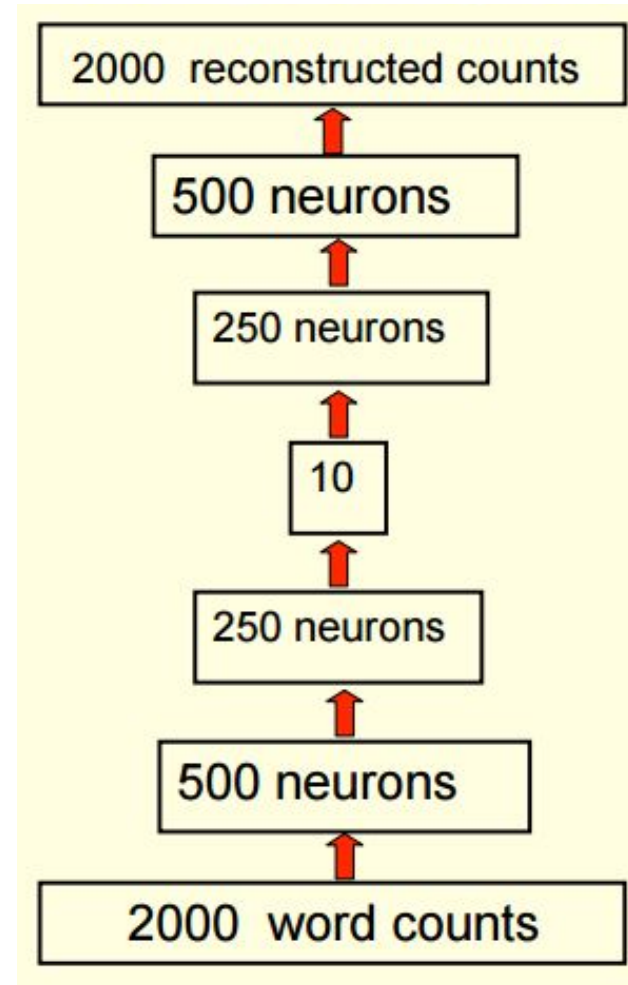
<https://arxiv.org/abs/1502.02072>

# Вариации FNN: Автоэнкодер (АЕ)

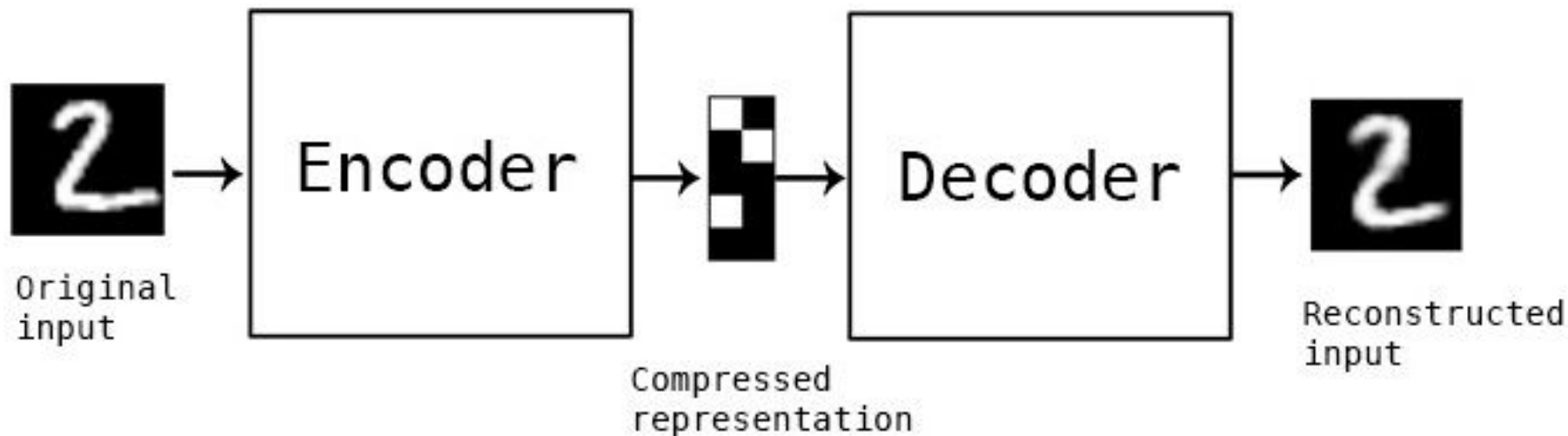
Учится создавать компактное описание входных данных.

Используется для уменьшения размерности и получения новых высокоуровневых признаков.

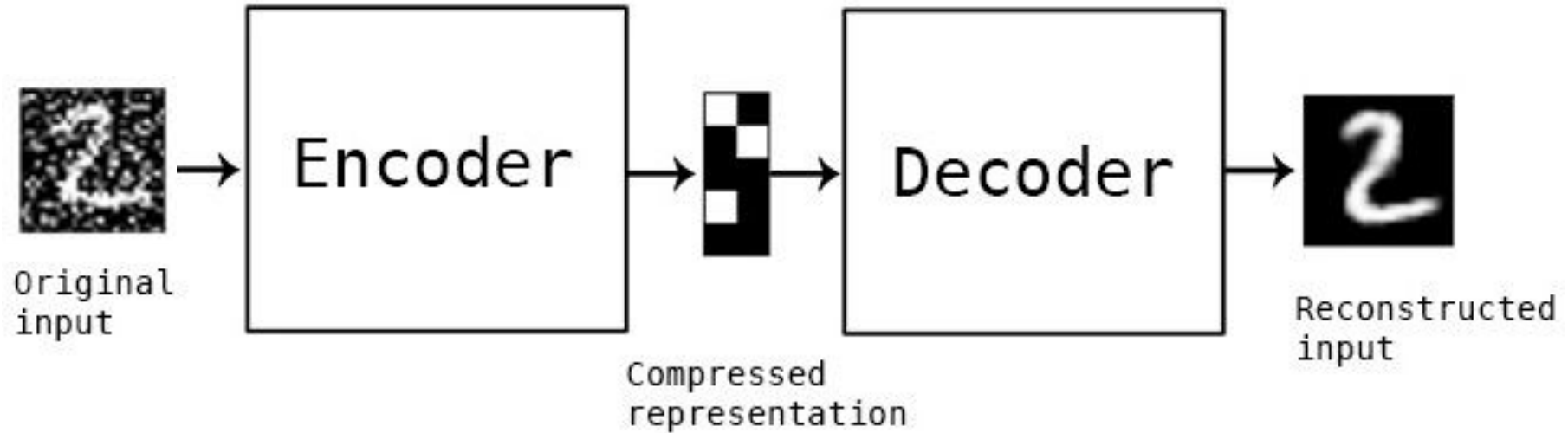
Может быть глубоким (многослойным).



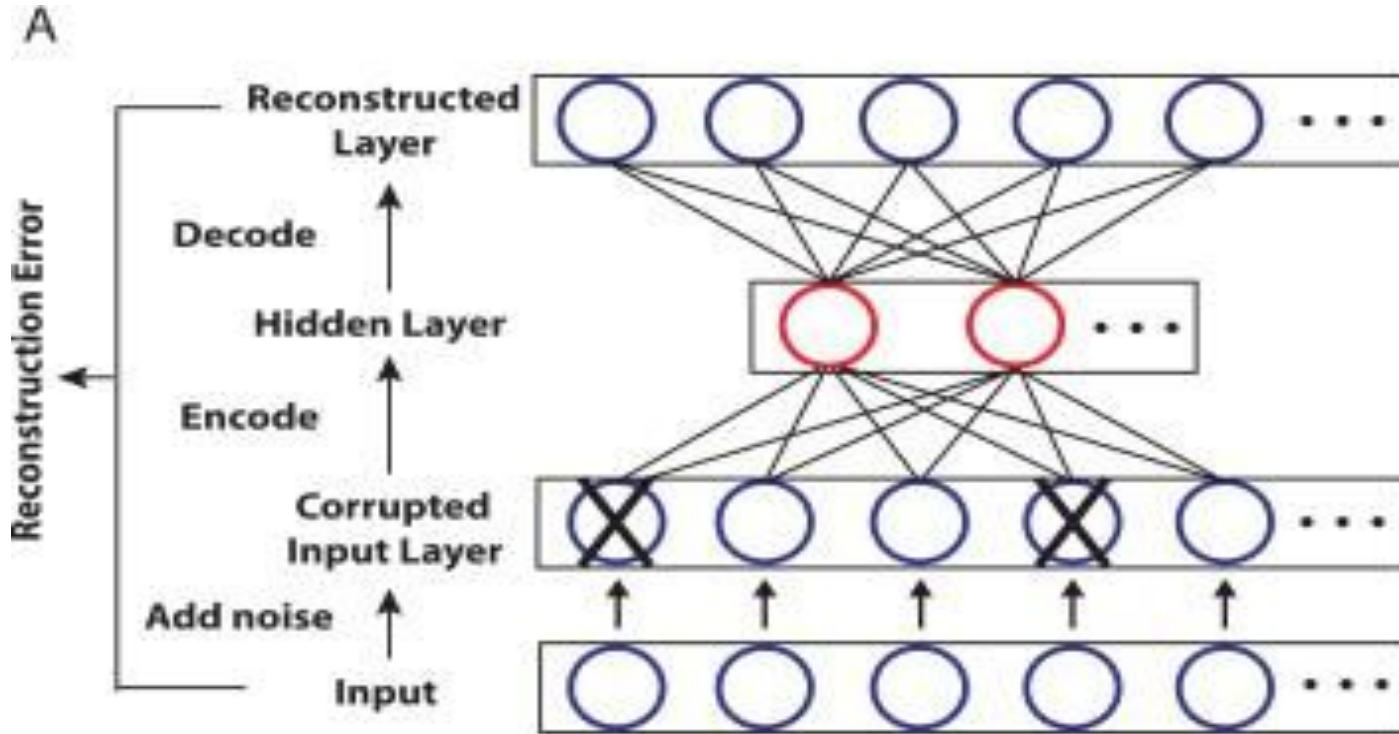
# Autoencoder



# Denoising Autoencoder



# Denoising Autoencoder

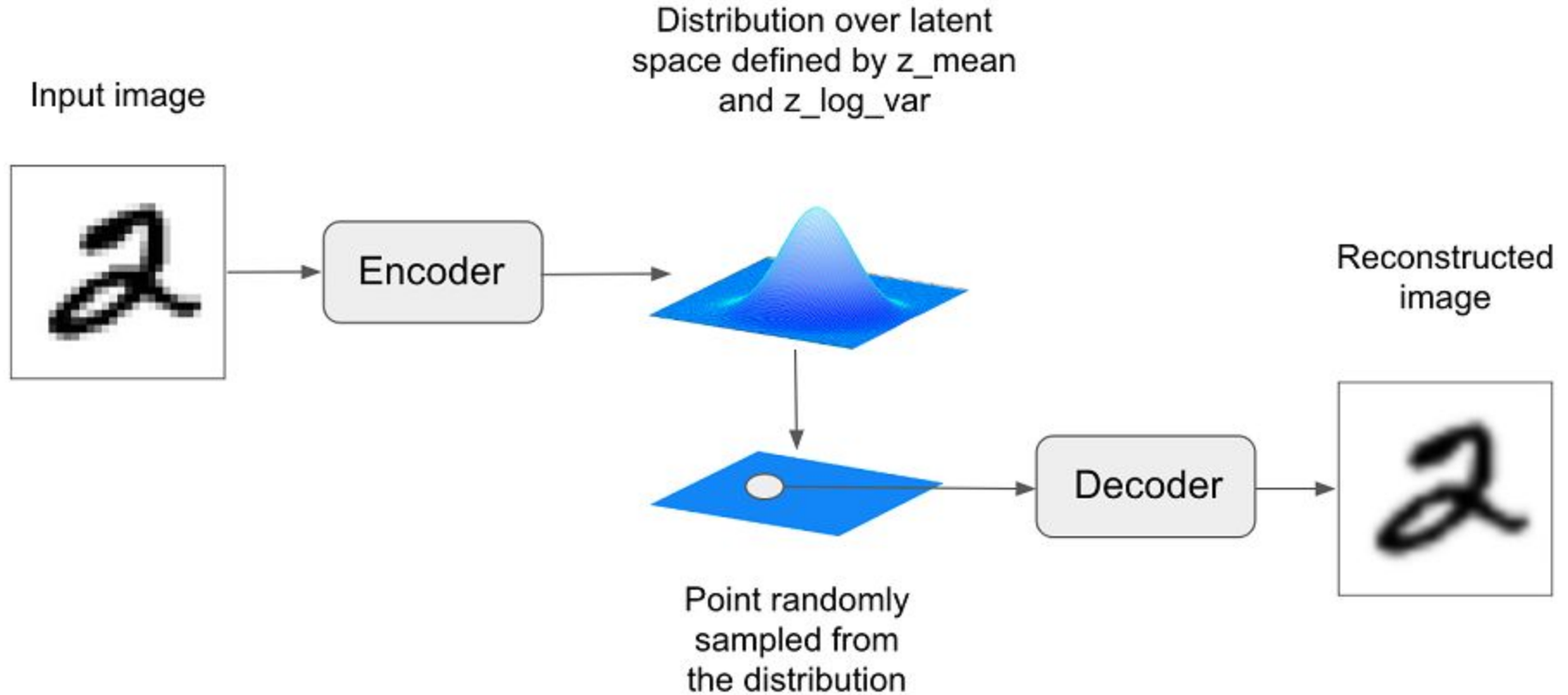


UNSUPERVISED FEATURE CONSTRUCTION AND KNOWLEDGE EXTRACTION FROM GENOME-WIDE ASSAYS OF BREAST CANCER WITH DENOISING AUTOENCODERS,

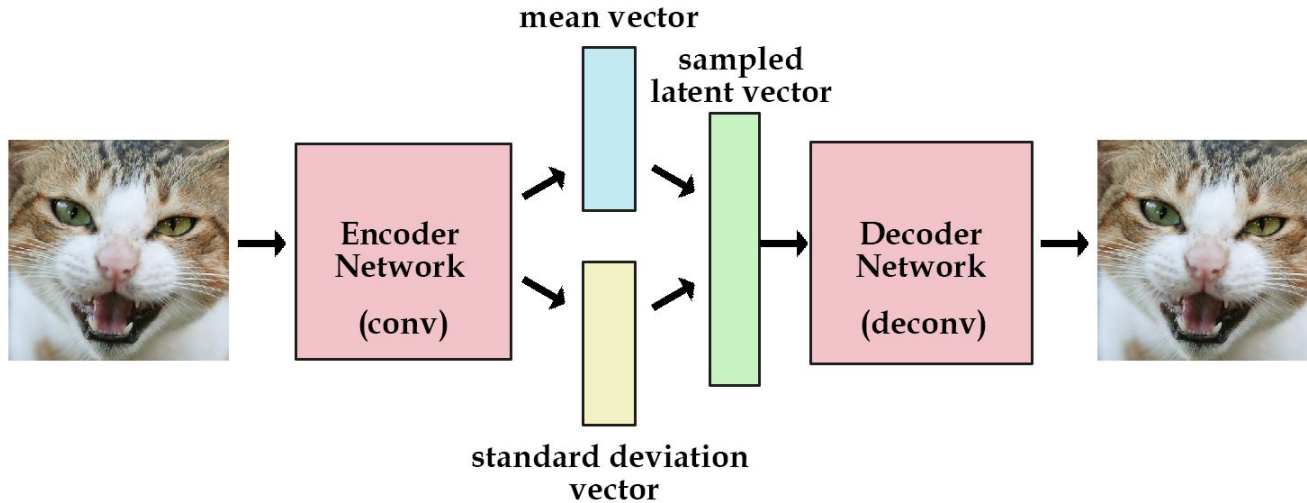
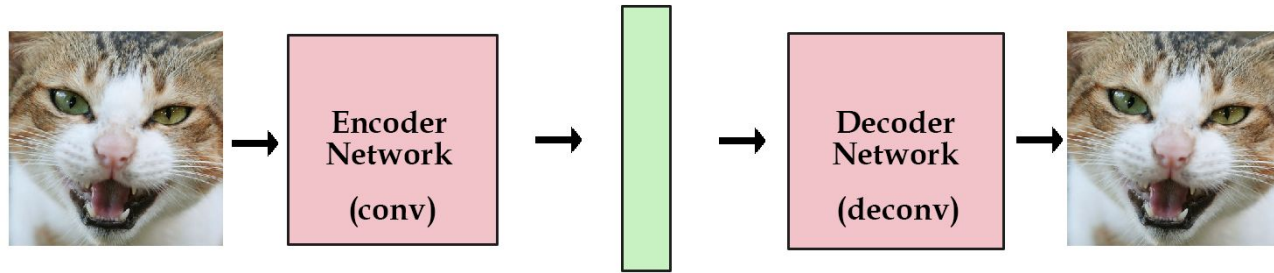
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4299935/>



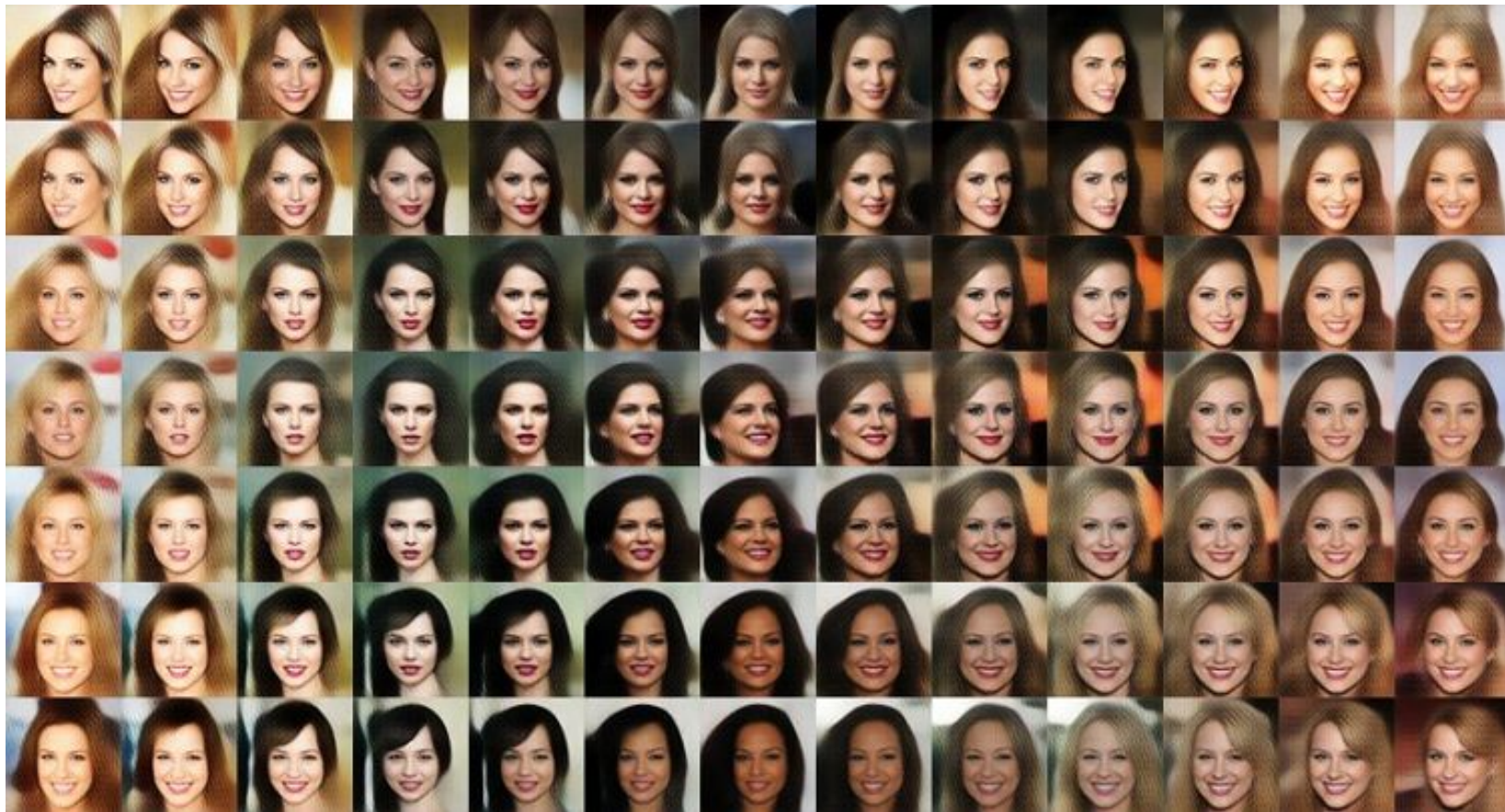
# Variational Autoencoder (VAE)



# AE vs. VAE



# VAE on images



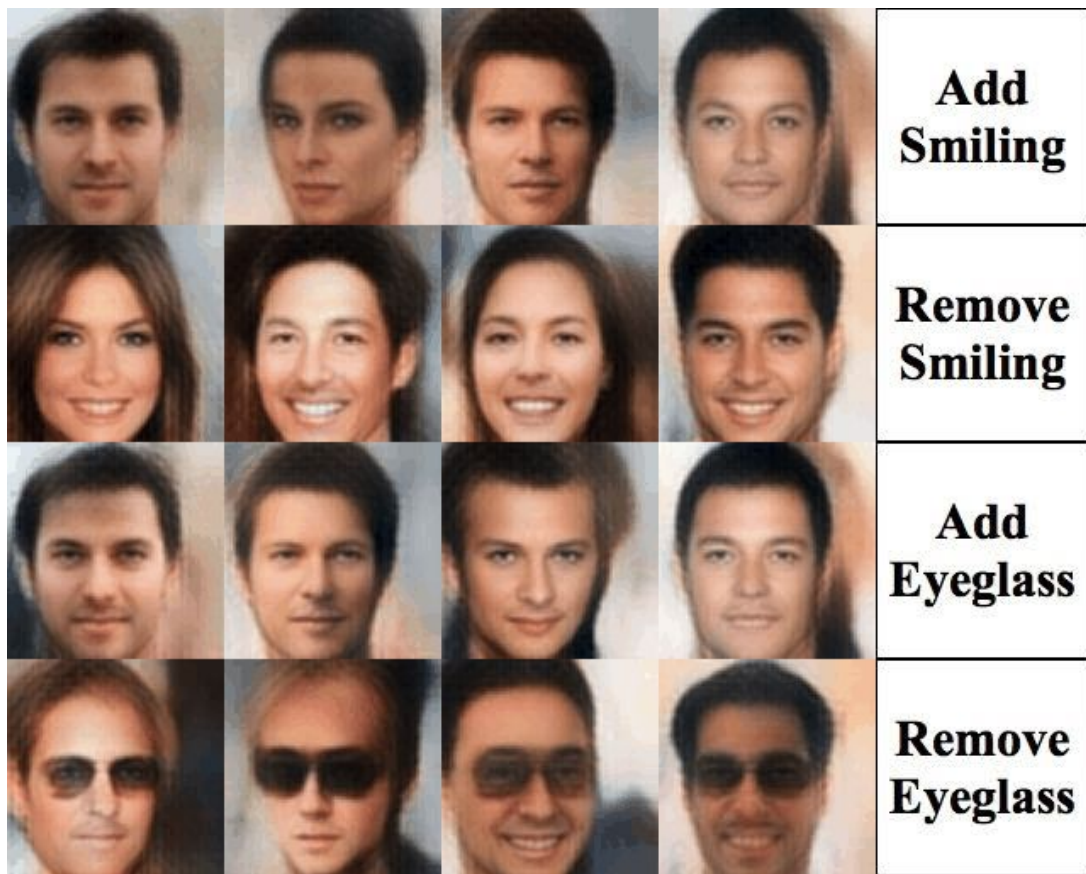
<https://www.manning.com/books/deep-learning-with-python>

# VAE on images and the “smile vector”





# VAE on images and the “smile vector”

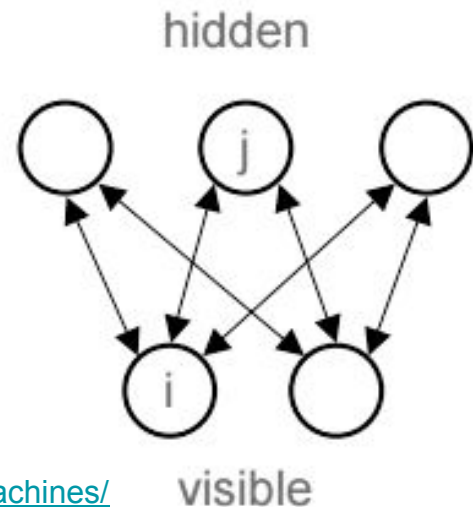


<https://github.com/houxianxu/DFC-VAE>

# Вариации FNN: RBM

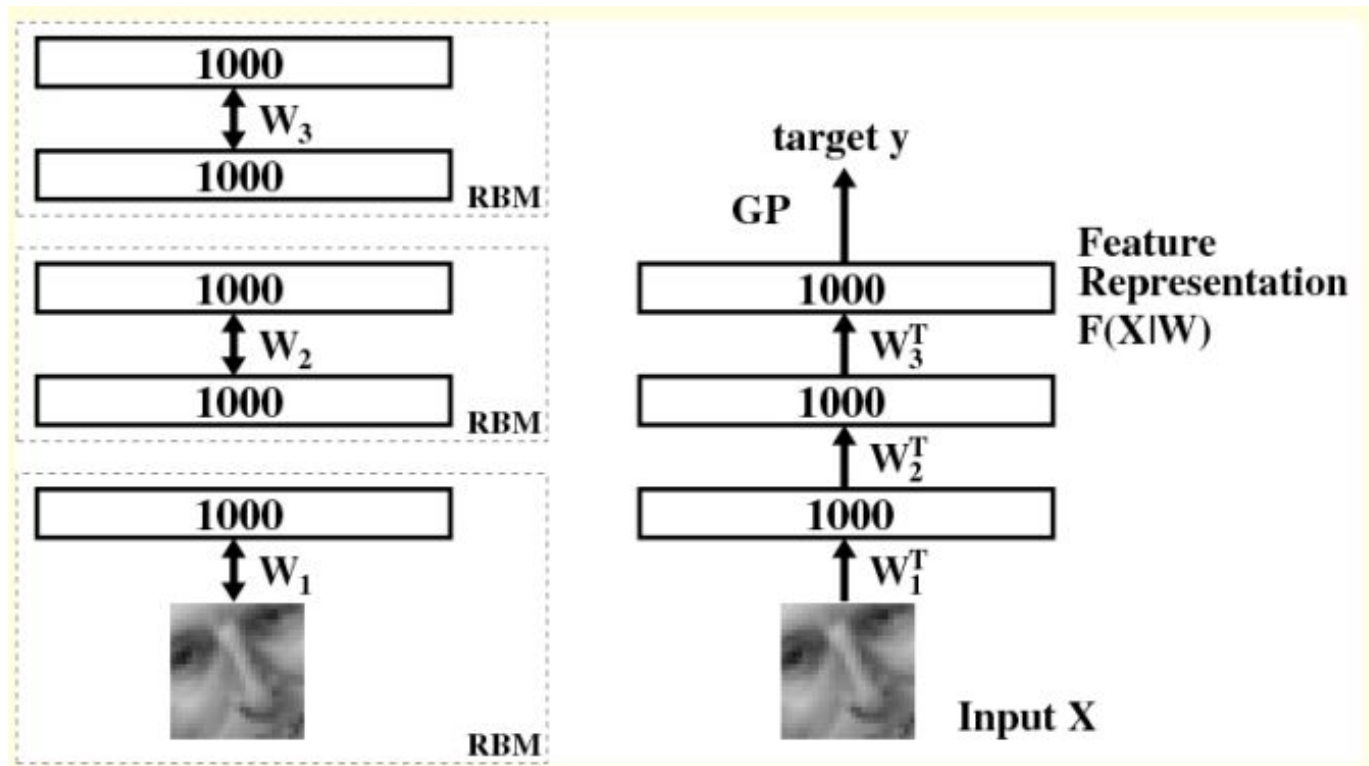
## Restricted Boltzmann Machine (RBM)

Неглубокая генеративная модель, которая учится генерировать данные с заданными характеристиками. По факту очень похожа на автоэнкодер, но в отличие от автоэнкодера стохастическая.



# Вариации FNN: DBN

Deep Belief Networks (DBN) — фактически способ обучения глубоких сетей, при котором каждый уровень сети учится как отдельная RBM.



Ресурсы



# Ресурсы

- “Deep Learning in Neural Networks: An Overview”, Juergen Schmidhuber, <https://arxiv.org/abs/1404.7828>
- “Deep Learning”, An MIT Press book, Ian Goodfellow and Yoshua Bengio and Aaron Courville, <http://www.deeplearningbook.org/>
- “A Tutorial on Deep Learning”, Q.V. Le, <http://www-cs-faculty.stanford.edu/~quocle/>
- [http://eclass.cc/courselists/117\\_deep\\_learning](http://eclass.cc/courselists/117_deep_learning)