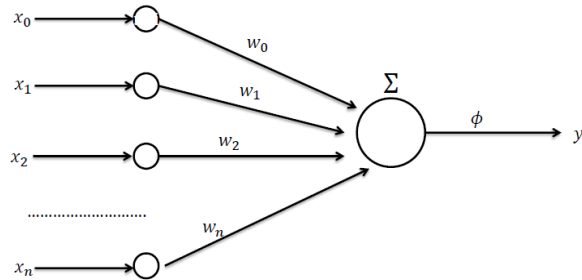


Ведение в искусственные нейронные сети

Линейные классификаторы




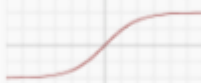


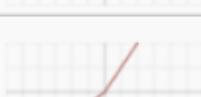


нейрон (персептрон)



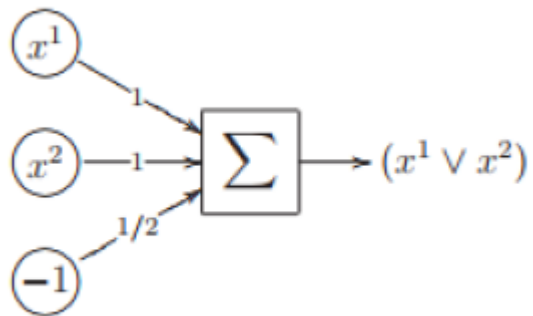
$$y = \phi\left(\sum_{i=0}^n x_i w_i\right)$$

- x_i - входные параметры
- w_i - обучаемые параметры
- ϕ - функция активации
- y - предсказание

activation functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
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) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

персептрон - линейная модель(например если у нас функция активации - логистическая то имеем логрессию)

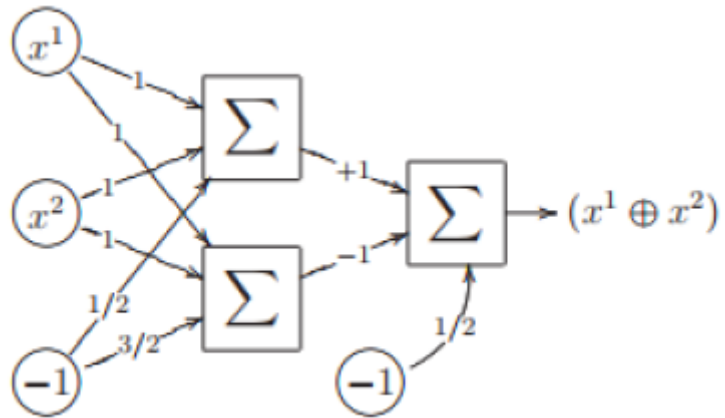


$$x_1 \vee x_2 = [x_1 + x_2 - \frac{1}{2} > 0]$$

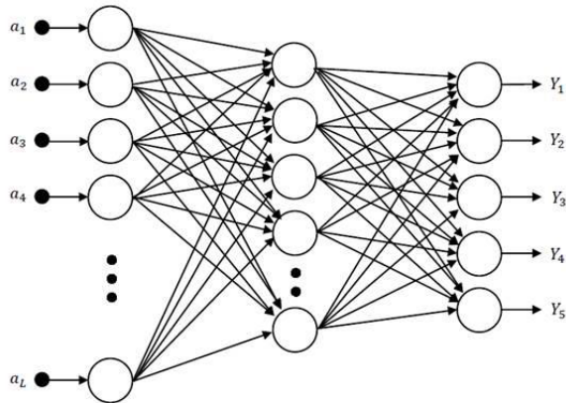
$$x_1 \& x_2 = [x_1 + x_2 - \frac{3}{2} > 0]$$

$$\neg x_1 = [-x_1 + \frac{1}{2} > 0]$$

XOR - \oplus



ПОЛНОСВЯЗНАЯ НЕЙРОННАЯ СЕТЬ (fully-connected neural network)



- ВХОДНОЙ СЛОЙ
- ВЫХОДНОЙ СЛОЙ
- один или более скрытых слоев(hidden layers)

Deep learning

Вычислительные возможности искусственных нейронных сетей.

1. Любая булева функция может быть выражена 2хслойной нейронной сетью.
(дизъюнктивная нормальная форма, конъюнктивная нормальная форма)
2. Из простых геометрических соображений следует, что двухслойная сеть с функциями активации binary step позволяет представить произвольный выпуклый многогранник в n-мерном пространстве характеристик.
3. Трехслойная сеть позволяет вычислить любую конечную линейную комбинацию характеристических функций выпуклых многогранников, поэтому аппроксимировать любые области с непрерывной границей.

Существуют более строгие математические доказательства того, что нейронные сети являются универсальными аппроксимациями функций. Однако ничего не сказано о **необходимом количестве нейронов для аппроксимации произвольной функции** .

Возможности сетей увеличиваются с количеством слоев и количеством нейронов в них.

Обычно используют 2 или 3 **полносвязных слоя fully-connected layers (or dense layer)** для решения задачи классификации или регрессии.

Вопрос: Каковы вычислимые возможности многослойной полносвязной сети с функцией активации $\phi(v) = v$?

метод обратного распространения ошибки(обучение нейросетей) backpropagation method (fitting NN)

Backpropagation обобщенное использование алгоритма оптимизации градиентный спуск для корректировки веса нейронов путем вычисления градиента функции потерь.

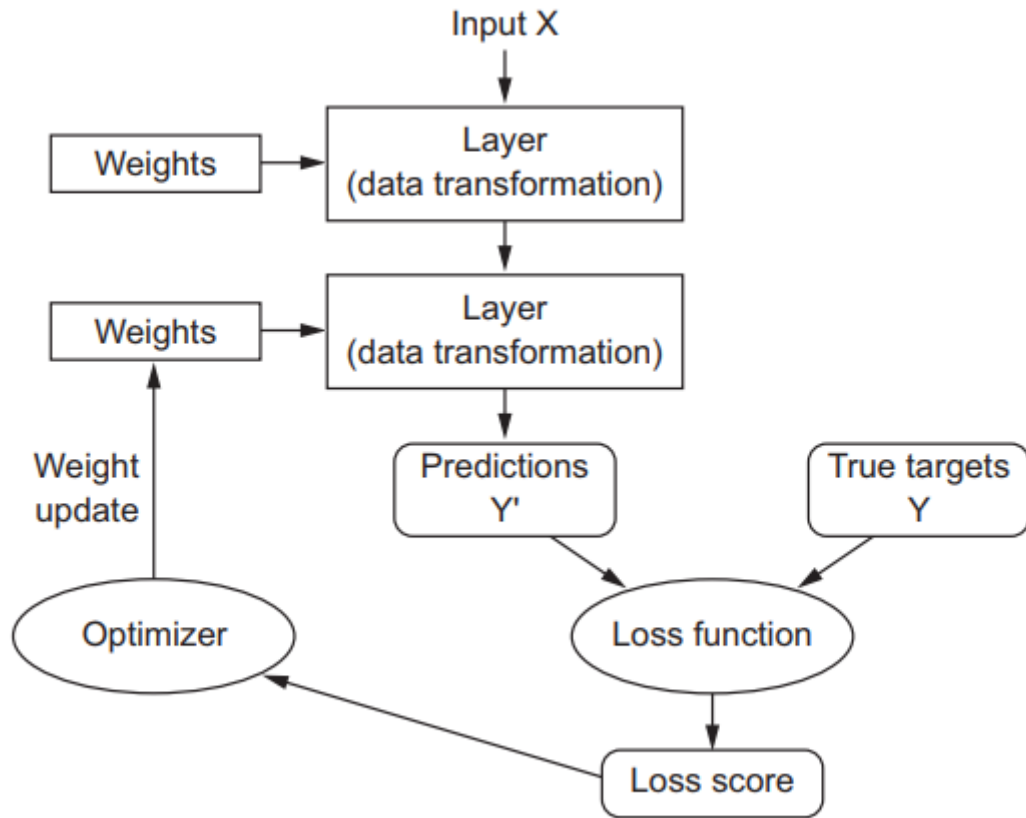
Прямой проход(Forward pass) с сохранением всех значений параметров, затем **обратный проход backward pass** с настройкой весов.

Часто идет обучение по **batch**. Запускается несколько объектов одновременно и на всем batch вычисляется ∇w .

Эпоха (Epoch) - это один из этапов(одна итерация) обучения всему набору учебных материалов, эпоха проходит за несколько итераций.

Шаг обучения(Learning rate) - шаг для градиентного спуска.

Функция потери, эмпирический риск.



- initiate weights
- devide to train and validate sample
- Iteration by epoch
 - devide train on batch
 - for each batch:
 - forward pass
 - count loss count, emperical risk
 - backward pass with adjust weight

$$w_i := w_i - \textit{lerning rate} * \left(\frac{\partial(\textit{emperic risk})}{\partial(w_i)} \right)$$

- test on train and val (overfitting)

Batch - зависит от мощности вычислительных ресурсов

EPOCH - пока не переобучимся, например, сохранять веса после эпохи.

LR настроить исходя из скорости обучения.

В машинном обучении есть 2 шага:

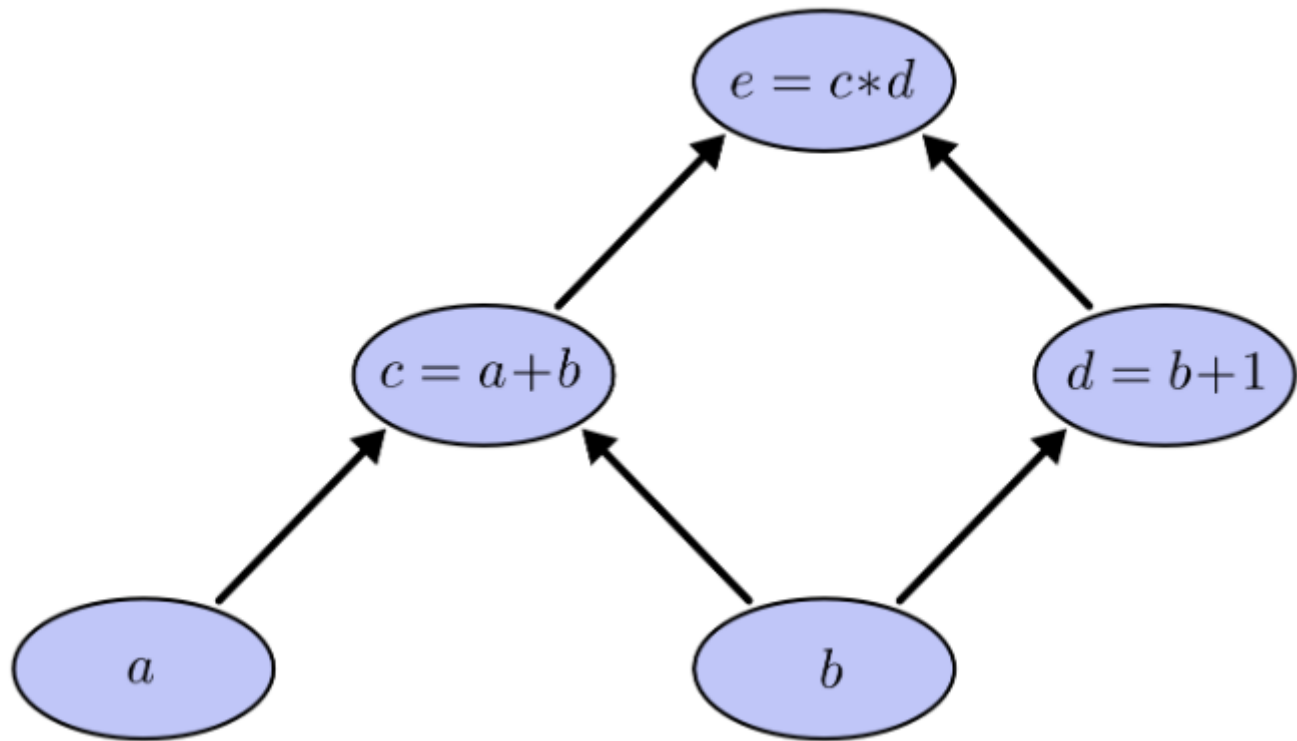
- векторизация
- классификация (регрессия, предсказание)

DL может реализовать оба шага. NN конструирует вторичные признаки.

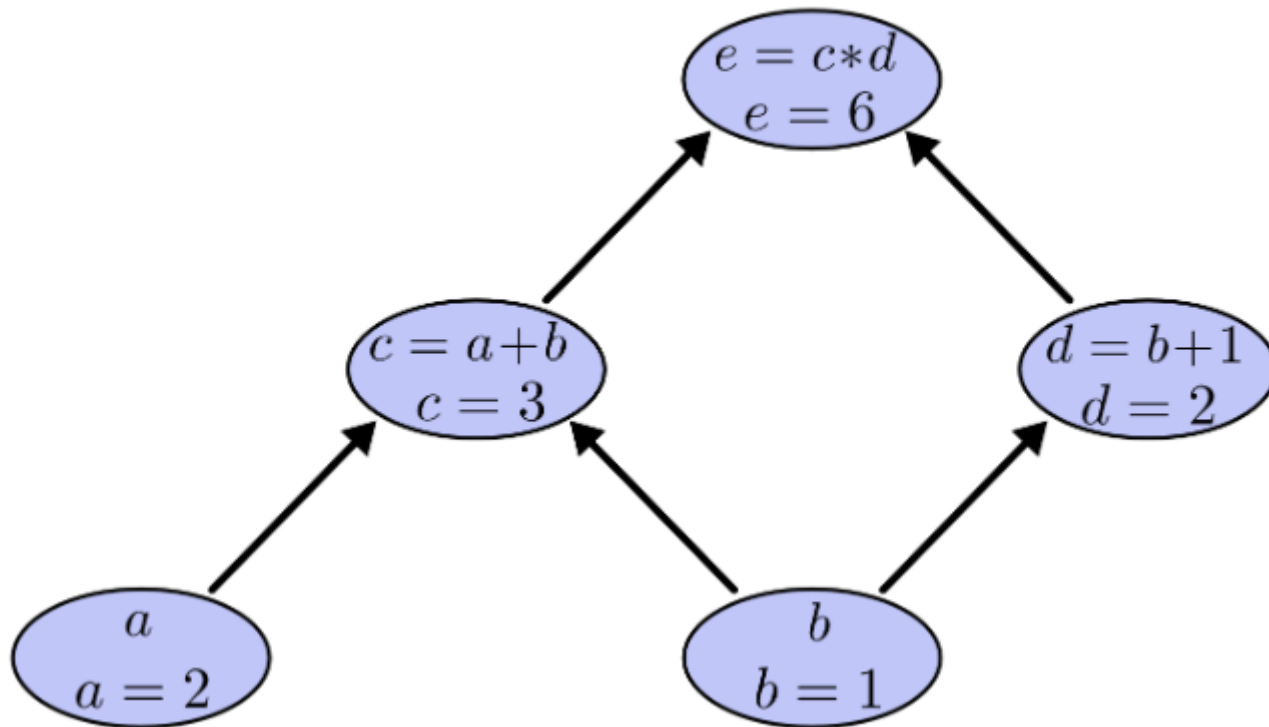
Граф вычисления

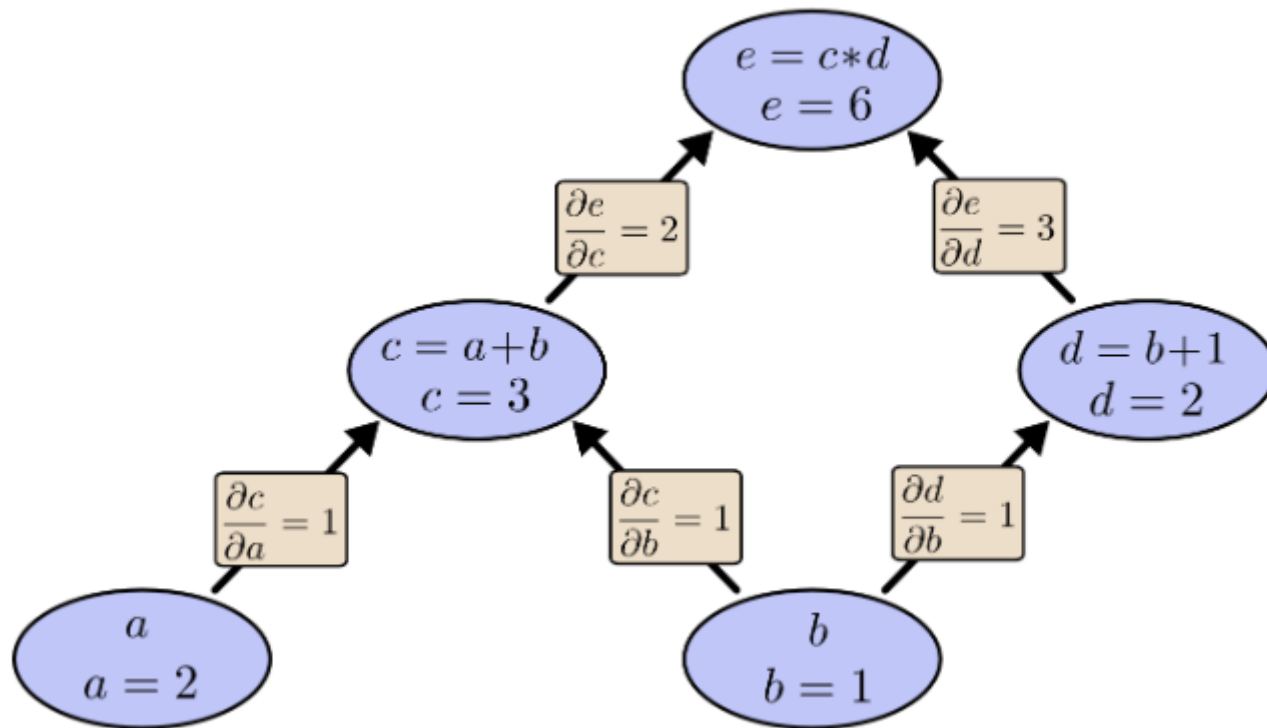
$$\begin{aligned}c &= a + b \\d &= b + 1 \\e &= c * d\end{aligned}$$

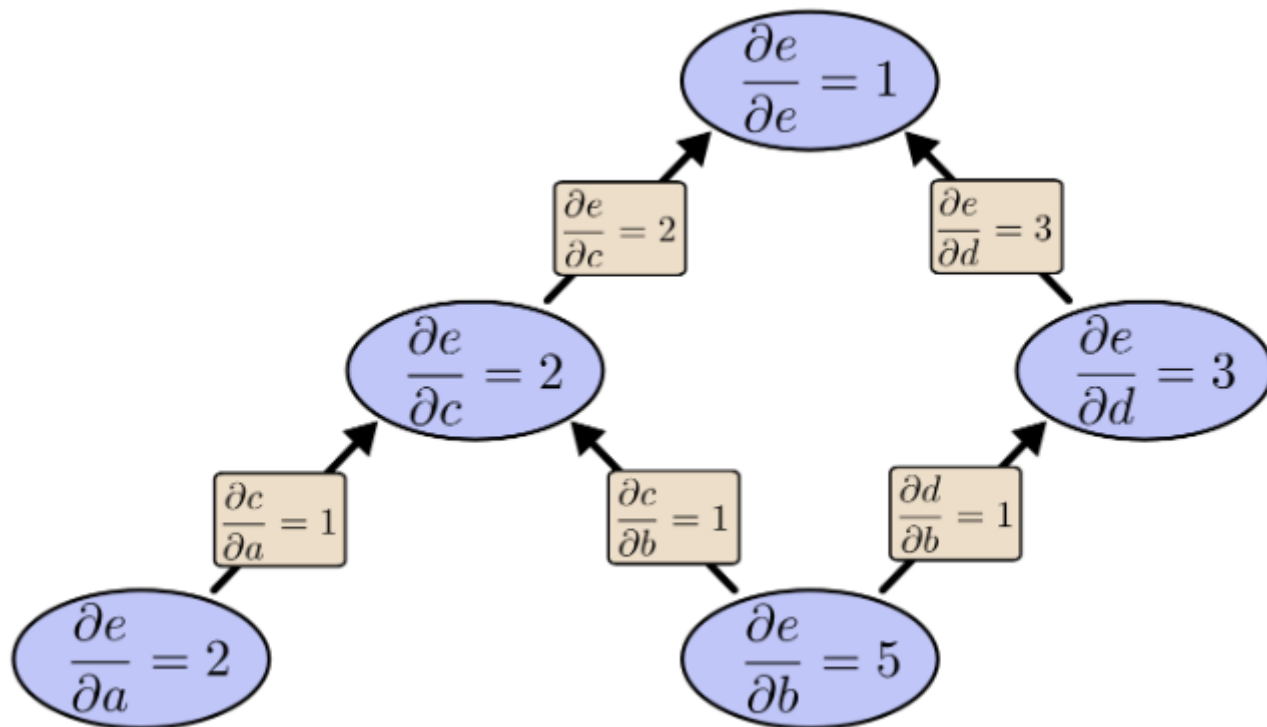
$$e = (a + b) * (b + 1)$$



Пусть $a = 2$ и $b = 1$







УСЛОВИЯ ДЛЯ СЛОЯ

- *forward(X, W) return result*
- *backward(*gradient*) return dX, dW*

Еще раз схема тренировки

1. взять минибатч (случайное разбиение данных)
2. forward pass посчитать значения
3. backward pass посчитать градиенты
4. обновить параметры

$$\overline{(W_i)} := \overline{(W_i)} - lr * \overline{\nabla_{W_i} Q}$$

препроцесс

нормализовать данные

затухание градиента

Если где то градиент станет 0, то дальше не будет обучения!!!

- tanh
- relu
- Leaky relu
- exponential relu

Инициализация весов

$$W = a * random_normal(input, output)$$

- a - большое веса будут сильно увеличиваться
- a - маленькое веса будут сильно уменьшаться

Xavier initialization

$$a = \frac{1}{\sqrt{input}}$$

He initialization

$$a = \frac{1}{\sqrt{\frac{input}{2}}}$$

Relu

Batch-normalization

With different measure of data, gradient descent does not work very well.

Batch-normalization - normalize the input data (expected value = 0, variance = 1).
Normalization is performed before entering layer.

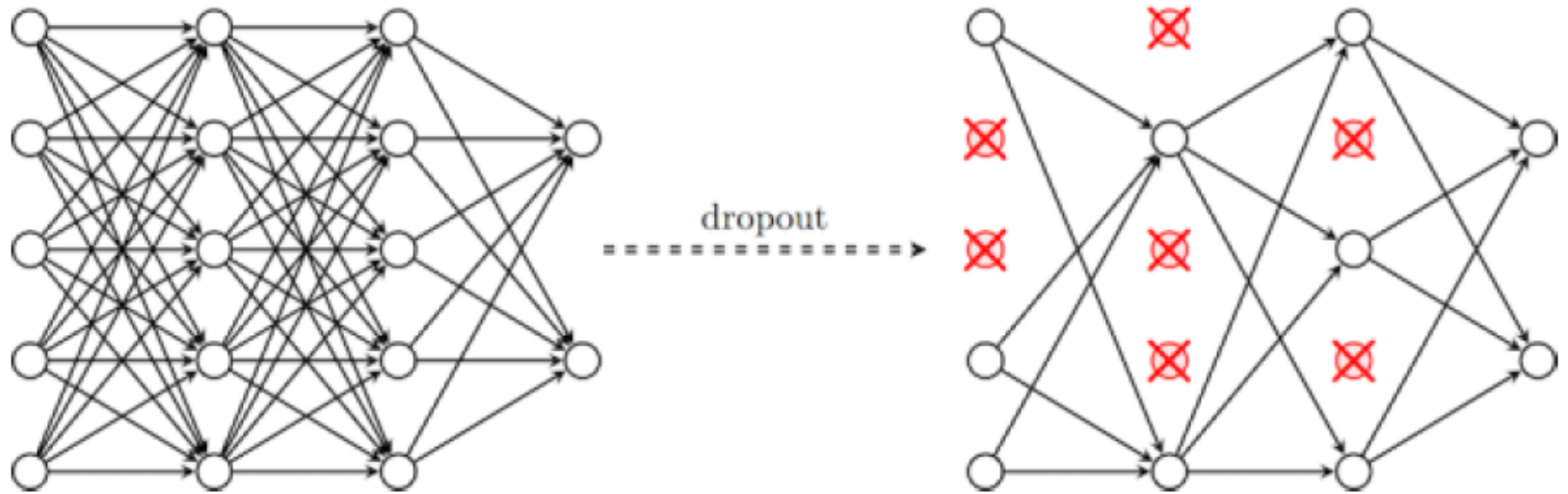
- ускоряет и стабилизирует тренировку
- регуляризует
- не так важна инициализация

regularisation in NN

motivation: there is no possibility to run a CV

- Weight penalty L1 and L2
- Early stopping
- Dataset **augmentation**

dropout layer



обновление параметров

$$w = w - lr * gradient$$

momentum

маленькая скорость сходимости.

$$velocity = momentum * velocity - lr * gradient$$

$$w = w + velocity$$

adagrad

Проблема локального минимума. Часто попадаются седловые точки.

$$accumulated = accumulated + gradient^2$$

$$adaptive_lr = \frac{lr}{\sqrt{accumulated}}$$

$$w = w - adaptive_lr * gradient$$

RMSProp

Движение все медленнее, хоть мы и выходим из седловых точек.

$$accum = \rho * accum + (1 - \rho) * grad^2$$

$$adaptive_lr = \frac{lr}{\sqrt{accumulated}}$$

$$w = w - adaptive_lr * gradient$$

adam

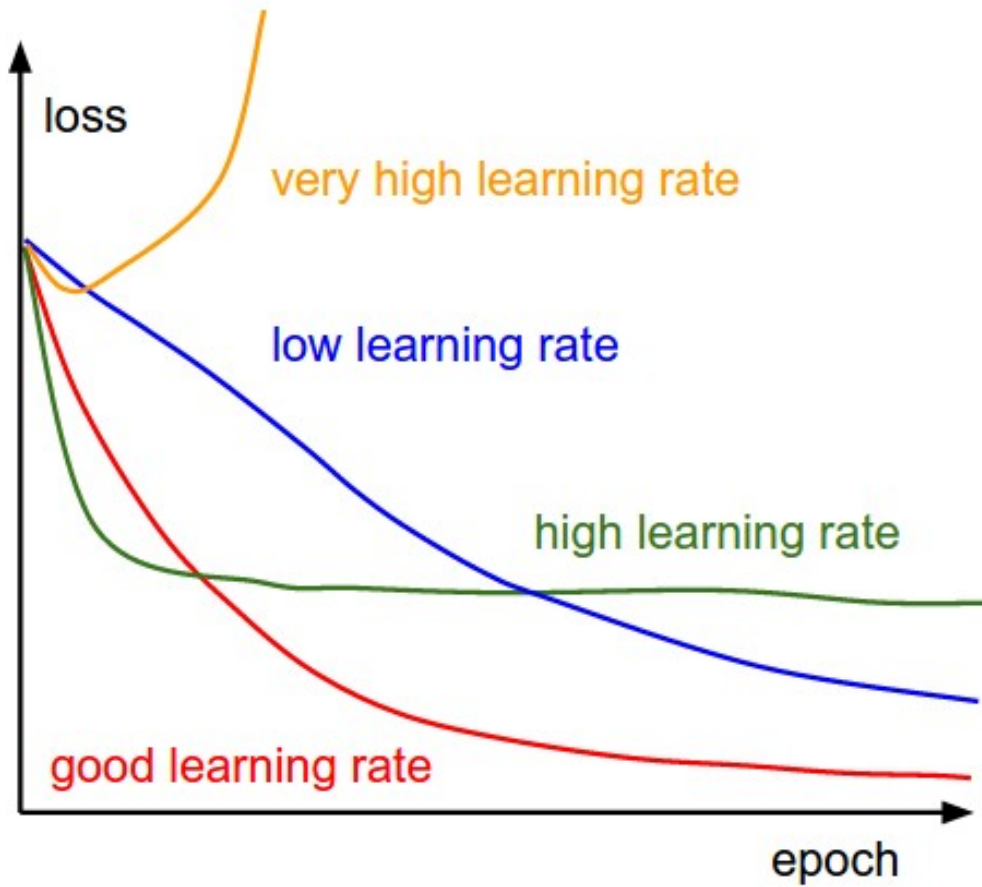
$$velocity = \beta * velocity + (1 - \beta) * gradient$$

$$accum = \rho * accum + (1 - \rho) * grad^2$$

$$adaptive_lr = \frac{lr}{\sqrt{accumulated}}$$

$$w = w - adaptive_lr * velocity$$

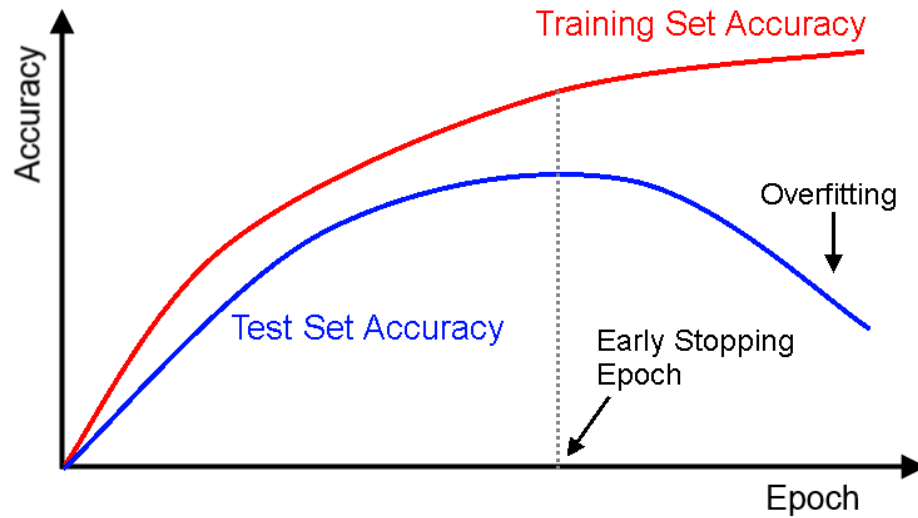
learning rate

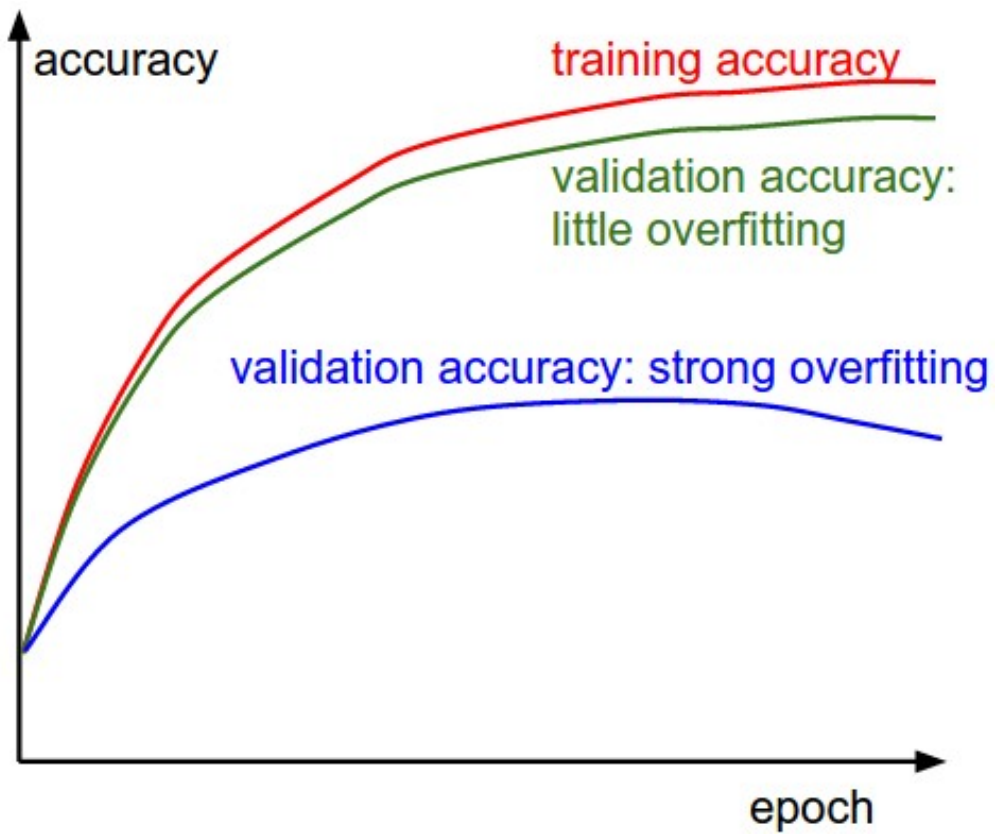


Annealing

- каждые n эпох умножать на $\lambda < 1$
- $lr = lr_0 * e^{-\alpha * t}$
- уменьшать при выходе на плато

early stopping

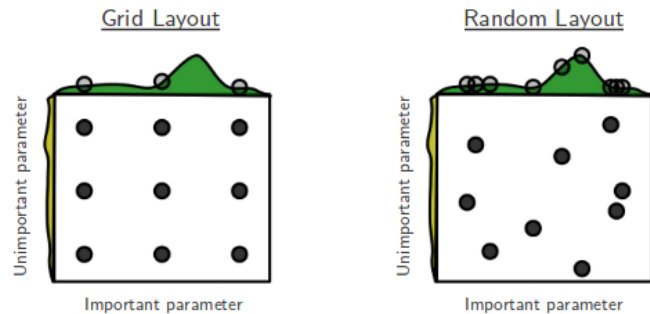




Подбор гиперпараметров

- lr
- коэффициент annealing
- размер батча?
- доля dropout

CV с одним фолдом. train and val.



Итого

- нормализуем данные
- активация RELU
- HE initialization
- Batch normalization
- оптимизатор adam или rmsprop
- уменьшаем LR
- перебираем гиперпараметры (настраиваем по val)
- следим за графиками (цифрами)