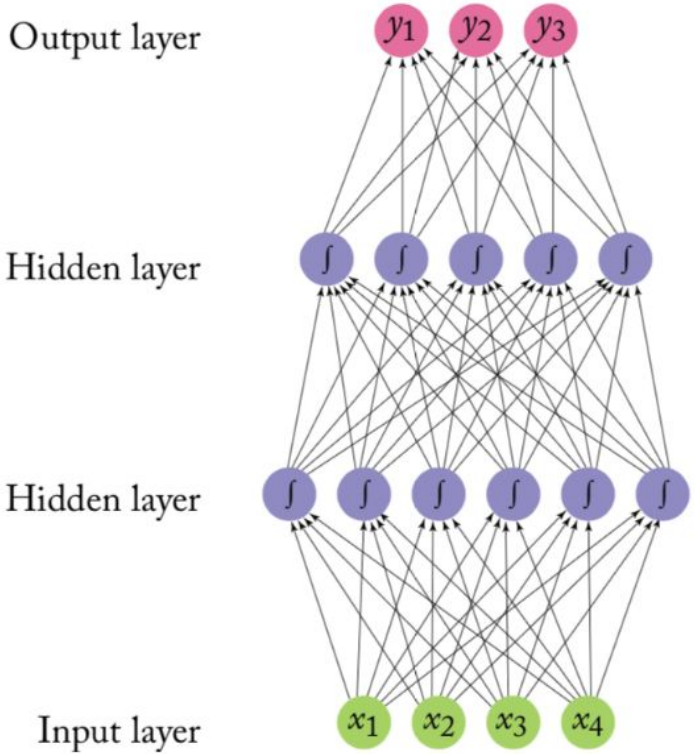
Нейросети. Эмбеддинги слов. WMD.

Маша Шеянова, masha.shejanova@gmail.com

Как устроена нейросеть

нейросеть in a nutshell



На входе — вектор признаков.

На каждой стрелочке — какие-то коэффициенты.

На выходе — вектор вероятностей того или иного класса.

“Нейрон” == один кружочек == функция от выдачи предыдущего слоя.

Перцептрон

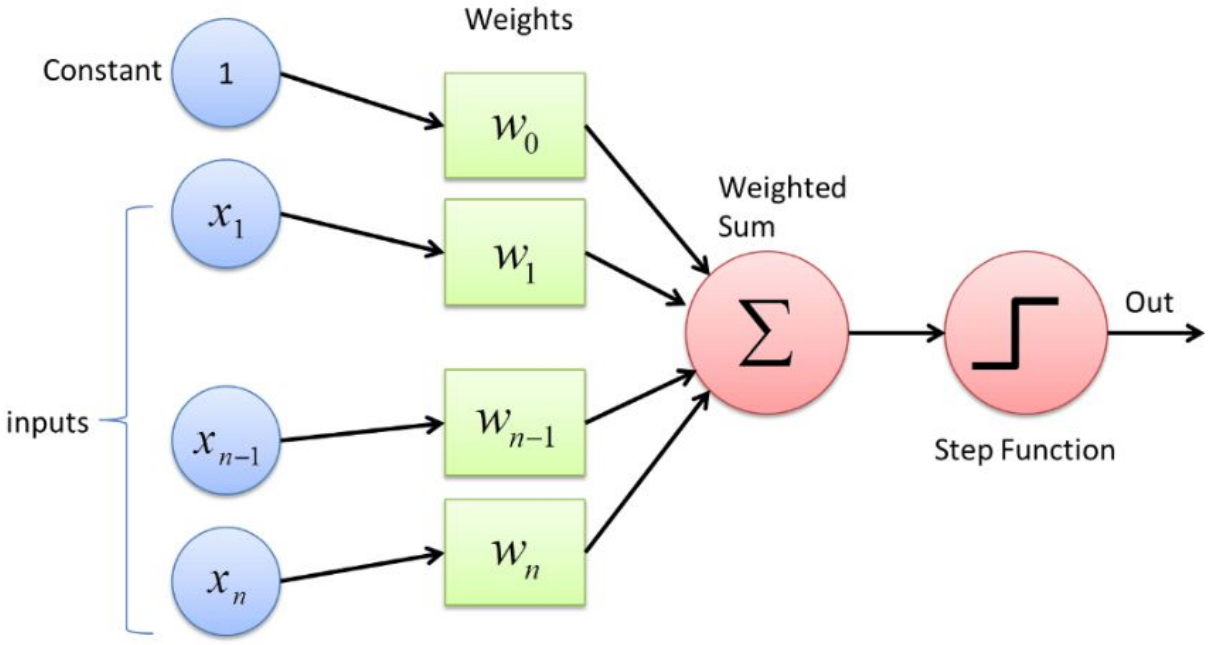
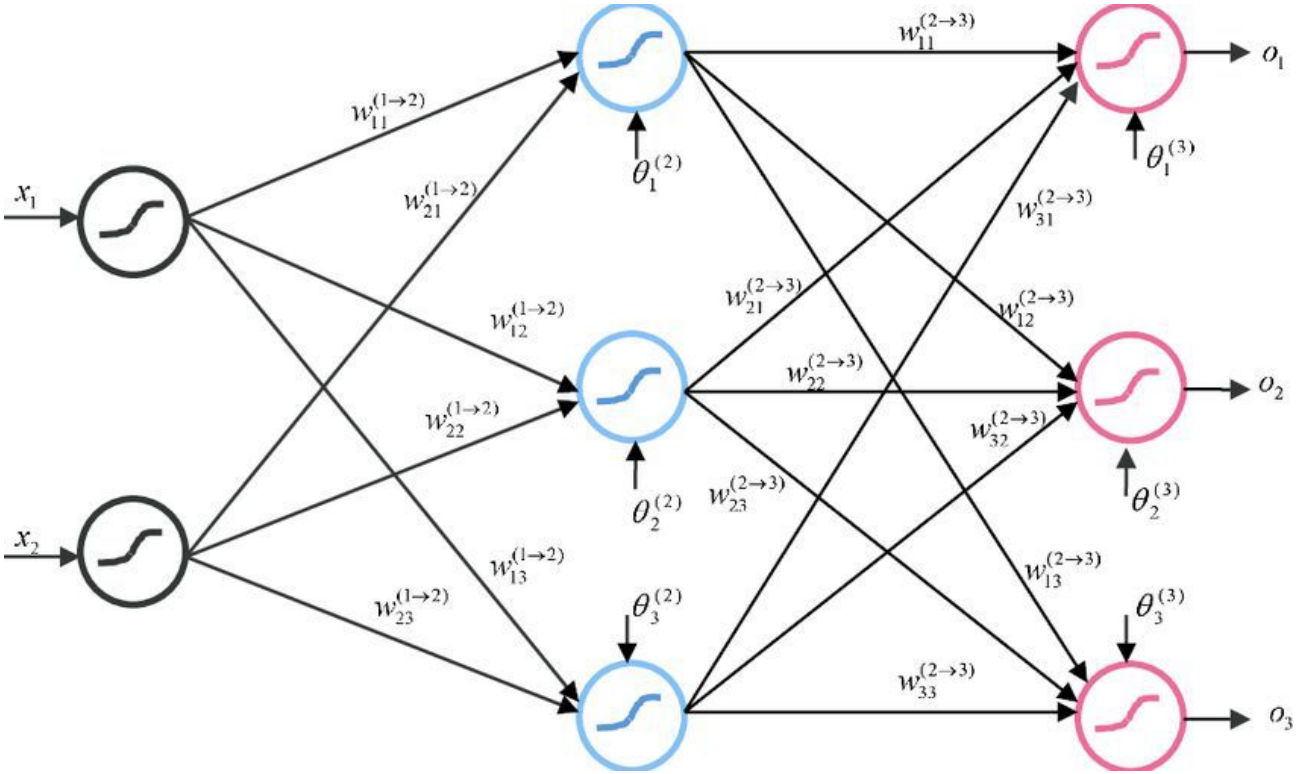


Fig : Perceptron

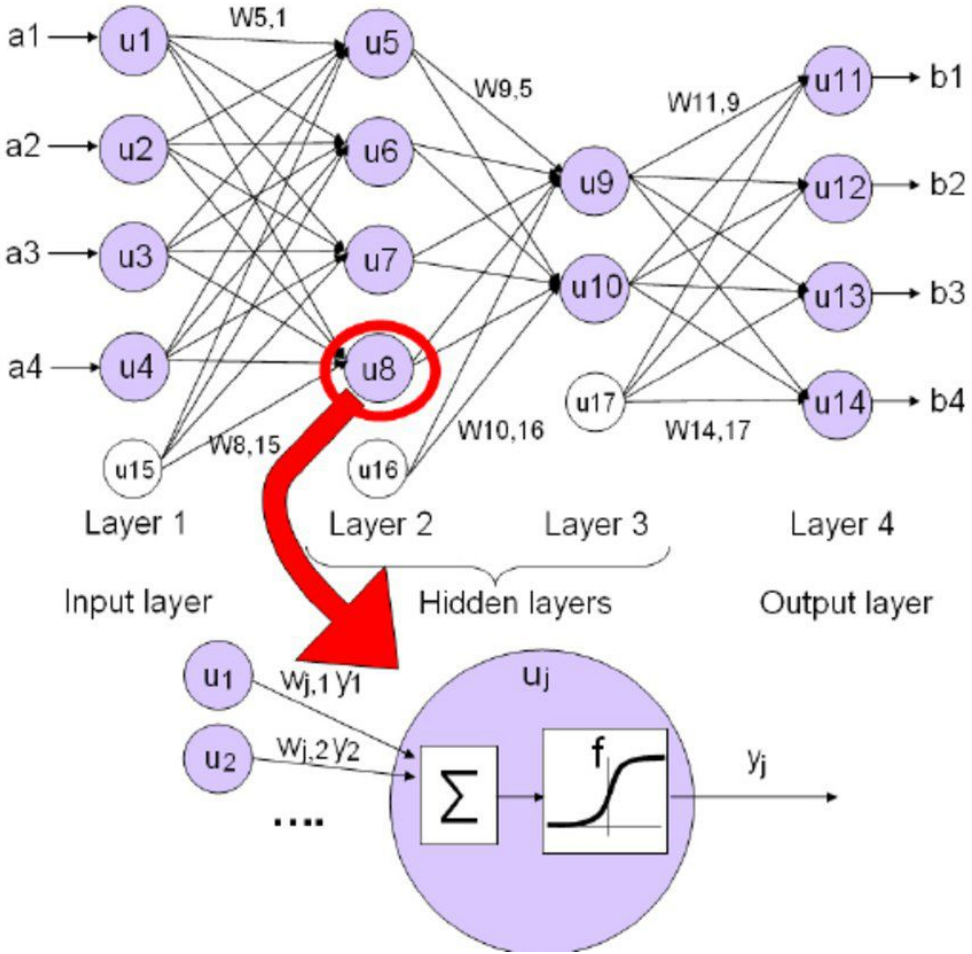
Однослойная нейросеть. С неё всё началось.

Нейросеть как функция



Один нейрон — функция от **вектора** параметров (w_{11}, w_{21}), умноженного на **вектор** объекта x (+ b).
*f(**wx** + b)*

Слой — функция от **матрицы** параметров * x + **вектор** b.
*f(**Wx** + **b**).*



Четырёхслойная нейросеть.

Универсальная теорема аппроксимации: любую функцию можно приблизить нейросетью.

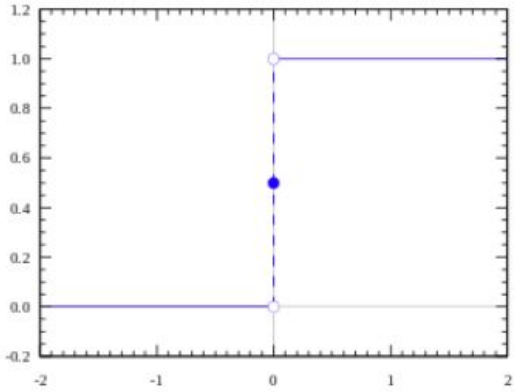
Функции активации

Почему “функция активации”?

… по аналогии с естественными нейросетями.

Справа — “step function”: нейрон активировался (1) или нет (0). (Источник картинки)

Но для artificial NN нужно что-то дифференцируемое.



Почти все картинки этого раздела взяты отсюда.

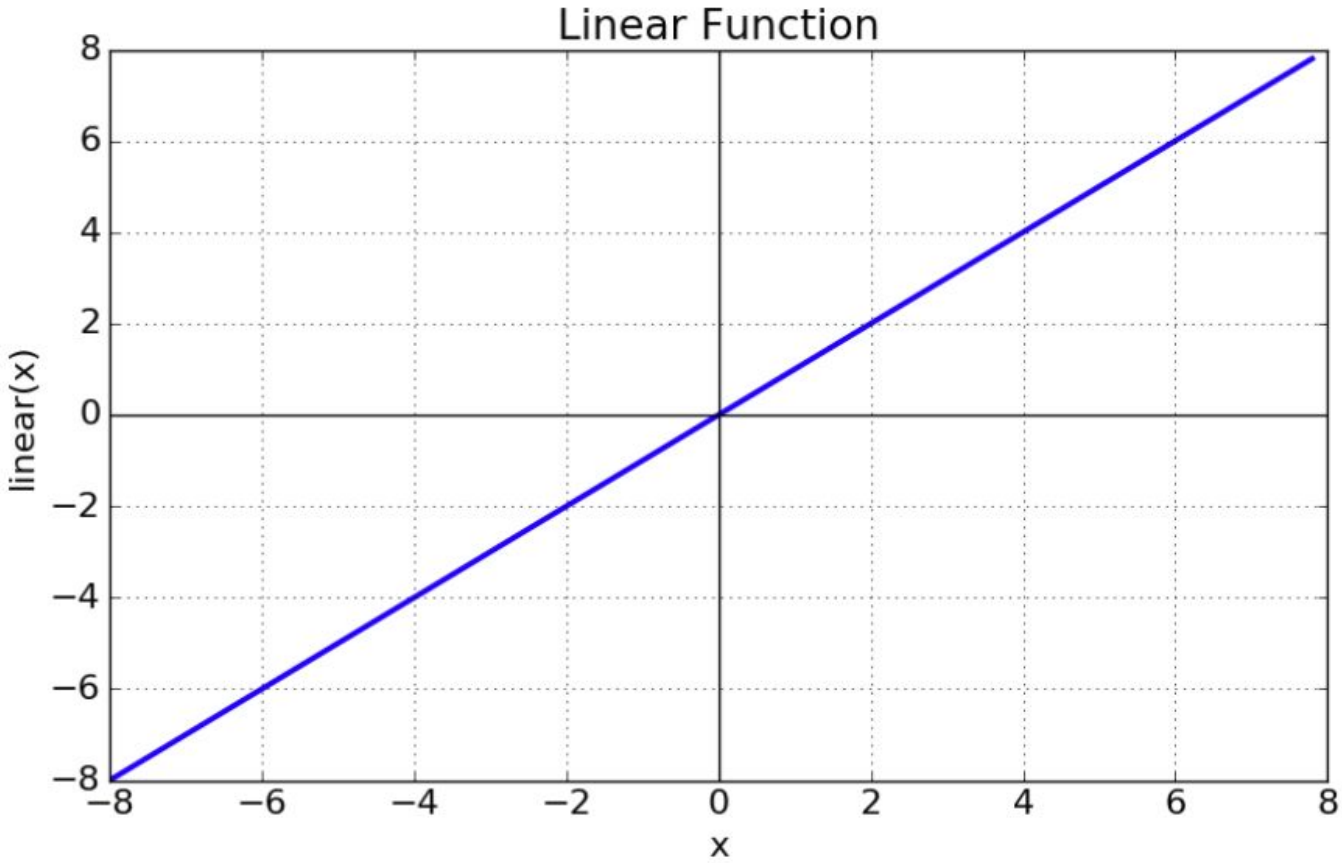
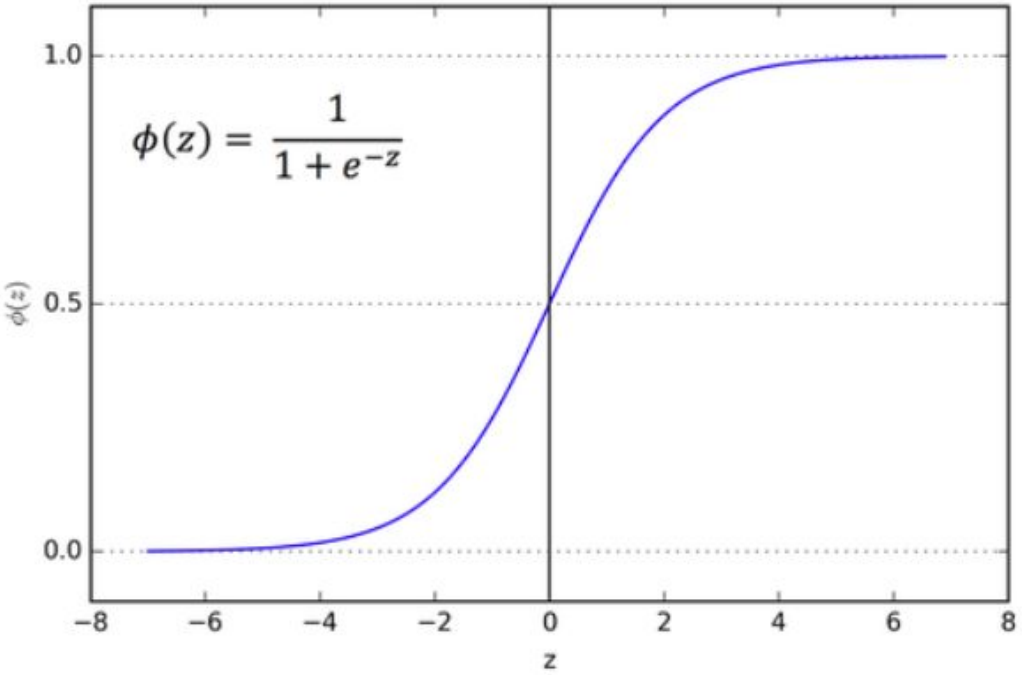


Fig: Linear Activation Function

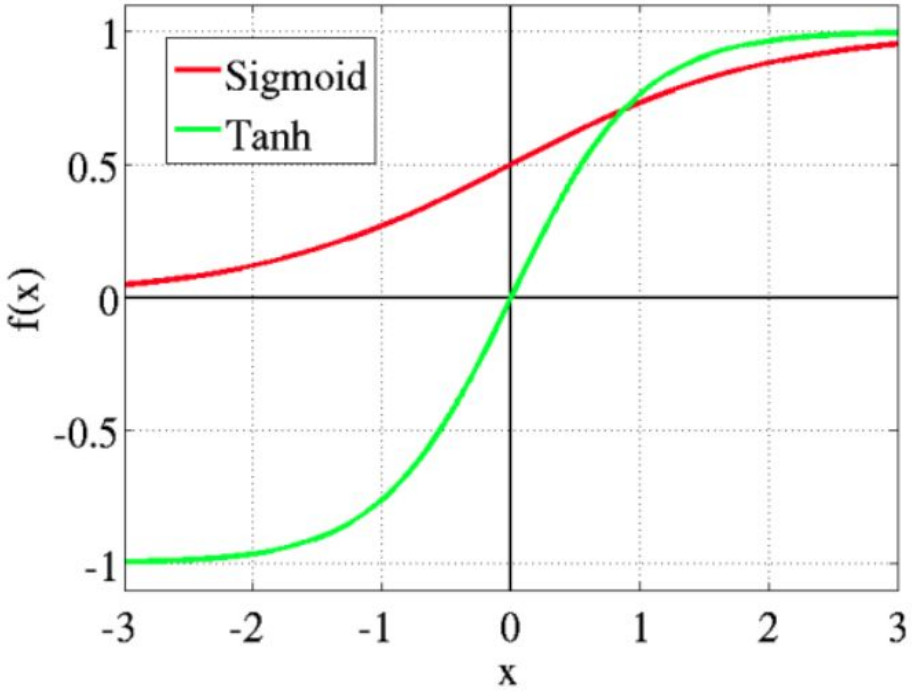
Sigmoid function



Это то же самое, что и логистическая регрессия.

Изменяется от 0 до 1 (и поэтому — хороший выбор для выдачи вероятностей).

Tanh or hyperbolic tangent Activation Function



Похожа на предыдущую, но изменяется от -1 до 1.

ReLU и Leaky ReLU

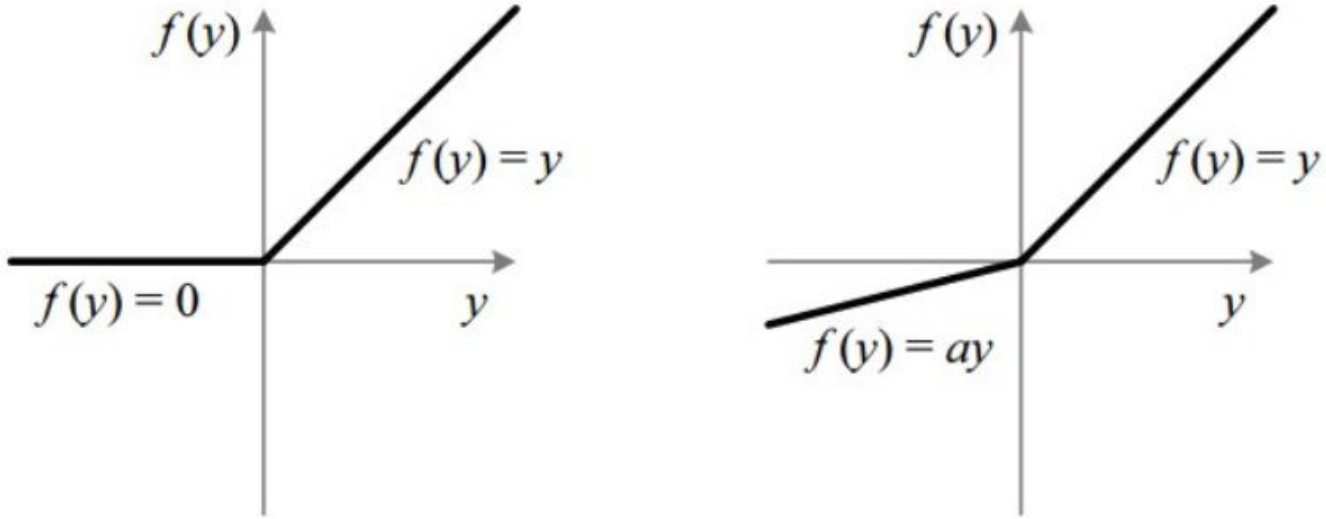


Fig : ReLU v/s Leaky ReLU

Backpropagation

Градиентный спуск

Производная

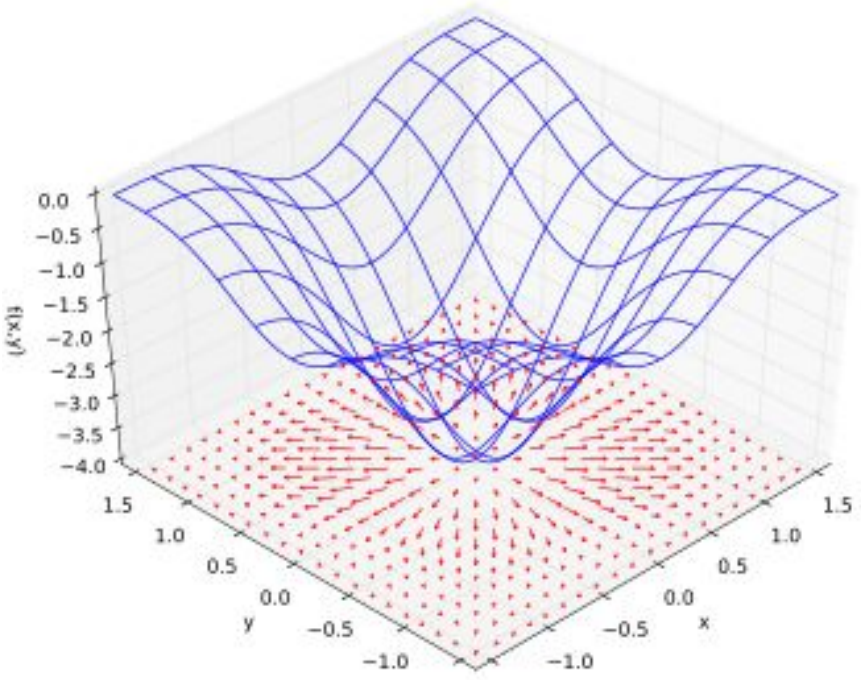
Производная — это мера, насколько быстро растёт функция.

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x}$$

У функции от n переменных f(x1, x2, ... x_n) нет одной общей производной — зато есть n частные производные.

$$\frac{\partial f}{\partial x_k}(a_1, \cdots, a_n) = \lim_{\Delta x \to 0} \frac{f(a_1, \ldots, a_k + \Delta x, \ldots, a_n) - f(a_1, \ldots, a_k, \ldots, a_n)}{\Delta x}$$

Что такое градиент



Градиент — это вектор, элементы которого — значения всех возможных частных производных в конкретной точке.

Градиент соответствует вектору, указывающему направление наибольшего роста функции.

Идея

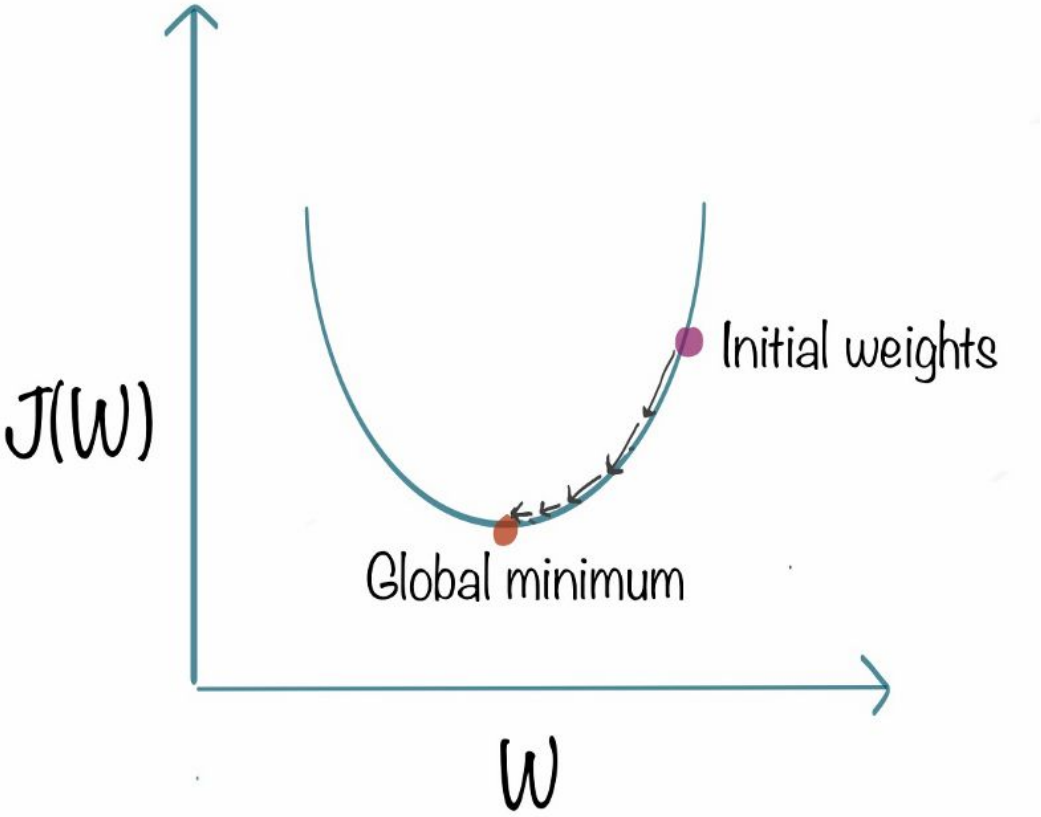
loss function = cost function = error function = функция потерь = J(W)

Её мы хотим минимизировать.

Теперь мы умеем находить, в каком направлении функция растёт быстрее всего. Но нам нужен минимум функции потерь, а не максимум!

Решение очевидно: найдём градиент и пойдём в обратную сторону.

С какой скоростью? Растёт быстро — с большой, медленно — с маленькой.



Источник картинки — очень понятно про то, как оно работает и какое бывает.

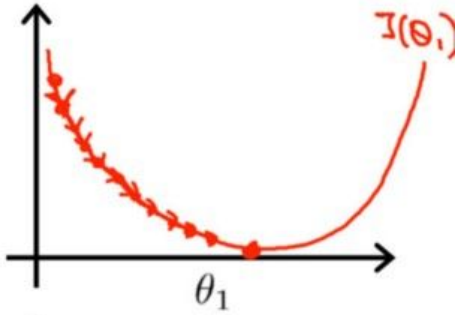
Шаги:

- подобрать случайные коэффициенты
- вычислить градиент функции потерь в этой точке
- обновить коэффициенты
- повторять, пока не сойдётся

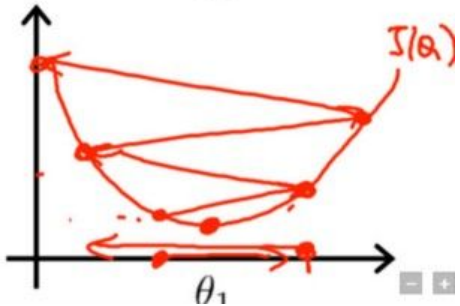
Learning rate

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. (отсюда)

Каким бывает градиентный спуск

- **Batch gradient descent**

  Считает градиент функции потерь с параметрами W сразу для всех обучающих данных. Работает жутко медленно.

- **Stochastic gradient descent (SGD)**

  Рандомно выбирает точку данных каждый раз

- **Mini-batch gradient descent**

  Выбираем кусочек выборки и по нему считаем

Что делать, если всё ещё ничего непонятно

Непонимание градиентного спуска, в принципе, не помешает вам решать типичные задачи готовыми инструментами. Но может помешать улучшать модель и решать проблемы, если что-то пойдет не так.

Если всё ещё ничего непонятно, keep calm and:

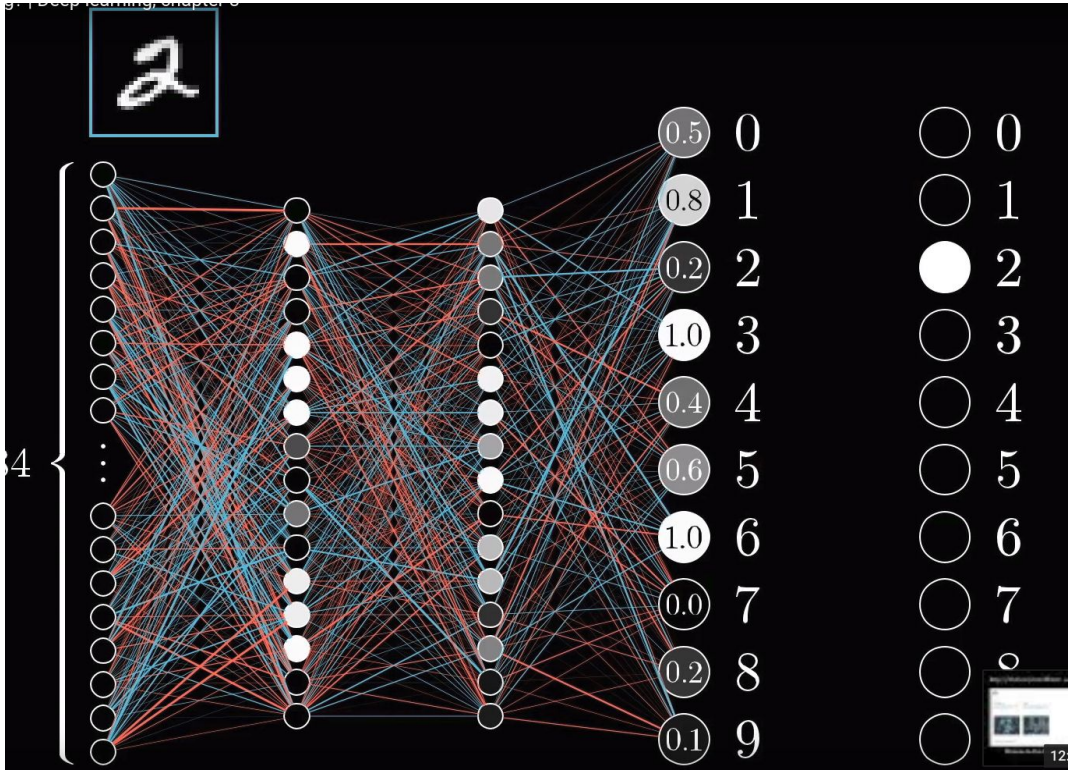
- пройдите небольшой курс по multivariate calculus на khan academy
- посмотрите вот это видео про градиентный спуск
- прочитайте эту и эту статью
- если удастся сформулировать вопросы, feel free to ask

Интуиция за backpropagation

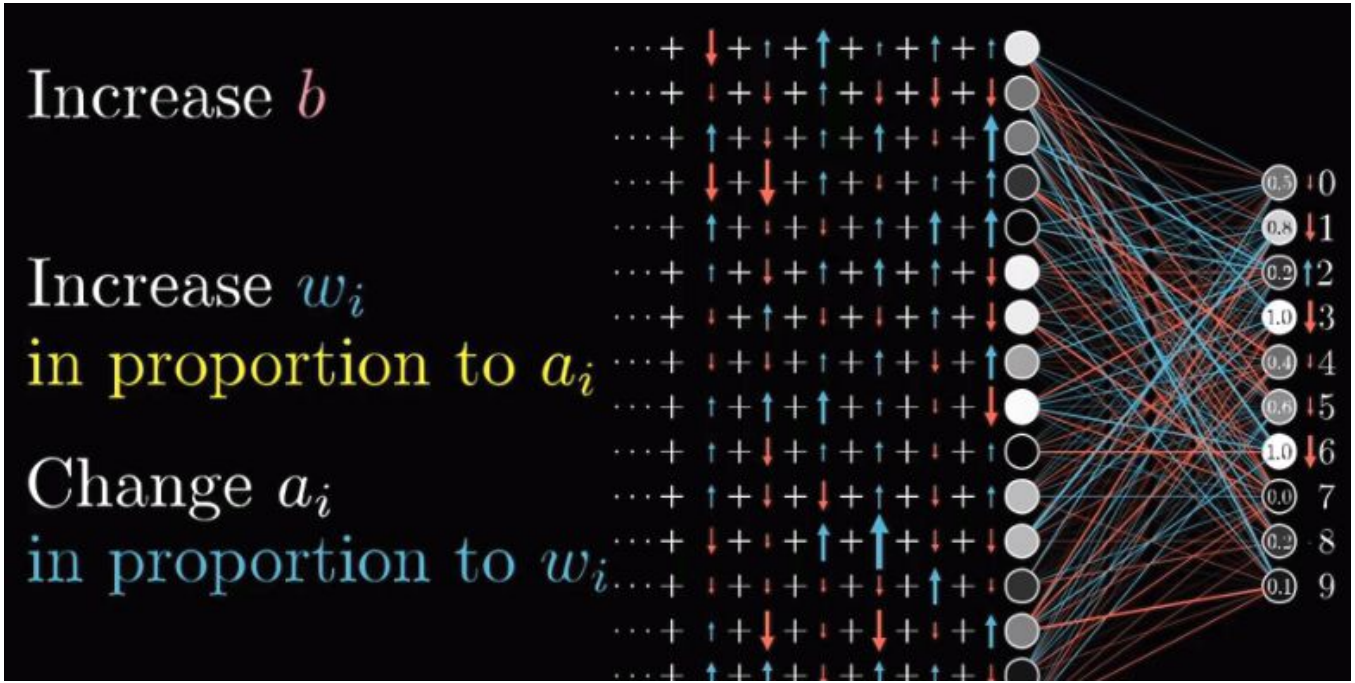
Функция потерь

Она может быть разной, например так:

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2,$$

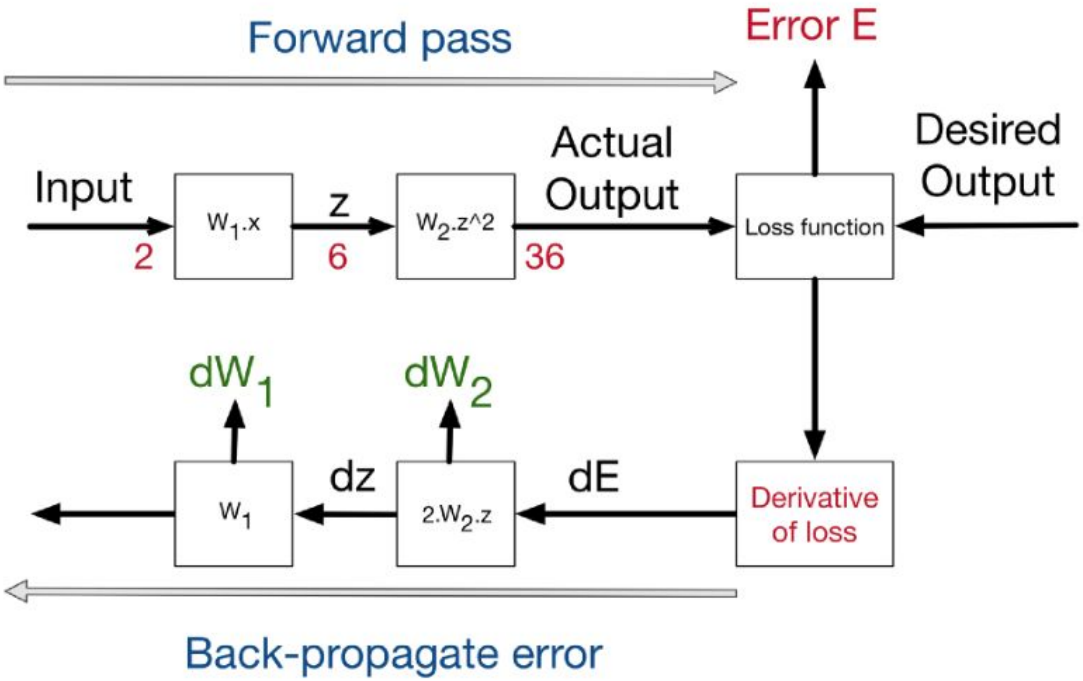


Суммирование ошибки



А дальше оптимизируем функцию потерь градиентным спуском.

Обратное распространение



Теория за backpropagation

Нейросеть — это тоже функция

x — входные данные

h1 = f1(W1 * x + b1)

h2 = f2(W2 * h1 + b2)

y_pred = f3(W3 * h2 + b3)

y_pred = f3(W3 * f2(W2 * f1(W1 * x + b1) + b2) + b3)

C = avg_sum(**y_pred** - **y_true**)^2

Композиция функций

В математике:

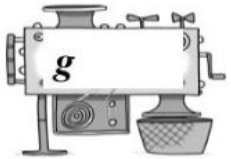
Substituting a function or it's value into **another** function.

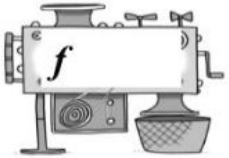
Second

$$f\left(g\left(x\right)\right)$$

First

(inside parentheses always first)





OR

$$f \circ g\left(x\right)$$

В программировании — то же самое!

Chain Rule

Это правило про то, как брать производную от композиции функций.

$$(f \circ g)' = (f' \circ g) \cdot g'.$$

This may equivalently be expressed in terms of the variable. Let $F = f \circ g$, or equivalently, $F(x) = f(g(x))$ for all x. Then one can also write

$$F'(x) = f'(g(x))g'(x).$$

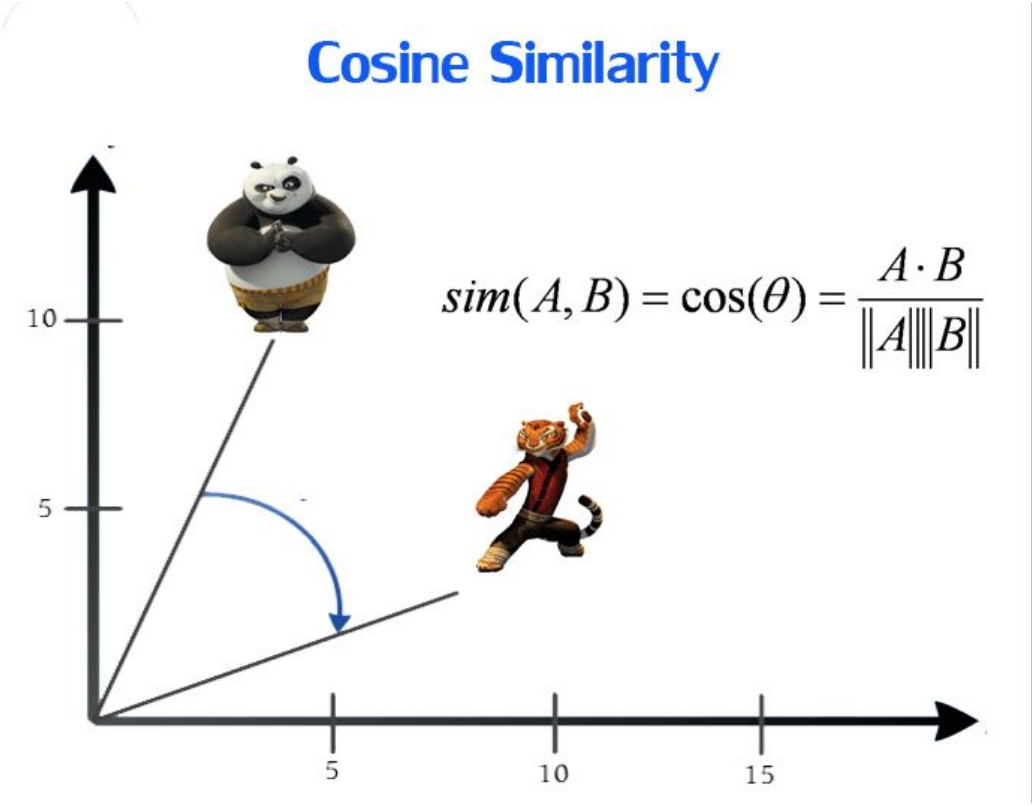
Псевдокод для backprop слоя

```
def backpropagation(loss):
  for layer in NN:
    layer.layer_loss *= layer.f_by_w(loss)
    loss *= layer.f_by_x(loss)
   return loss
```

А вот здесь есть код.

Эмбеддинги

Как найти, насколько близки слова?



- надо найти способ превратить слова в вектора так, чтобы они отражали **контекст**
- найти расстояние между этими векторами одним из способов

Источник картинки.

Как сделать из слов вектора?

Итак, основная идея — **учитывать контекст**. Но как? Про это есть большая наука.

Самый простой-наивный метод — **счётный**. Идея: для каждого слова возьмём ближайшие в некотором окне (например, -5 +5). Сделаем такой же мешок слов, как делали для документов (CountVectorizer, TfidfVectorizer). Можно делать “скользящее окно”.

Плюсы: легко и быстро.

Минусы: для большого корпуса — очень большие вектора.

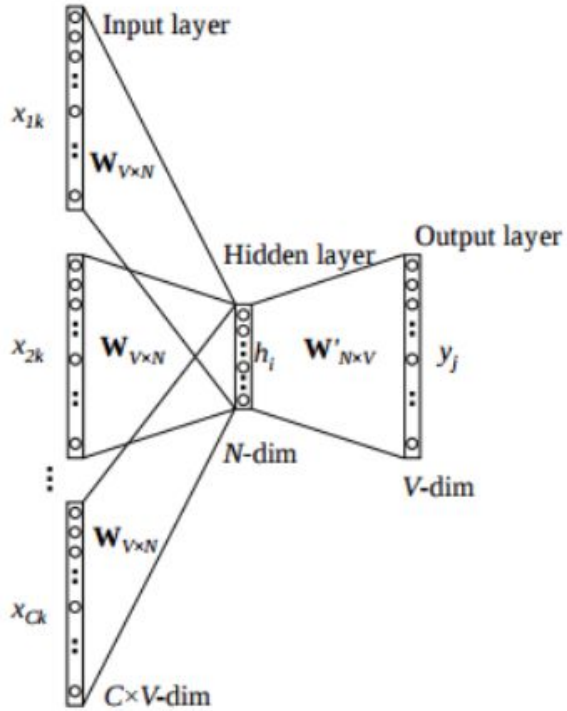
Word2vec

В двух словах, Word2Vec — это метод строить гораздо более компактные эмбеддинги с помощью нейросетей.

Методы:

- CBOW (Common Bag Of Words)
- skipgram

CBOW (common bag of words)



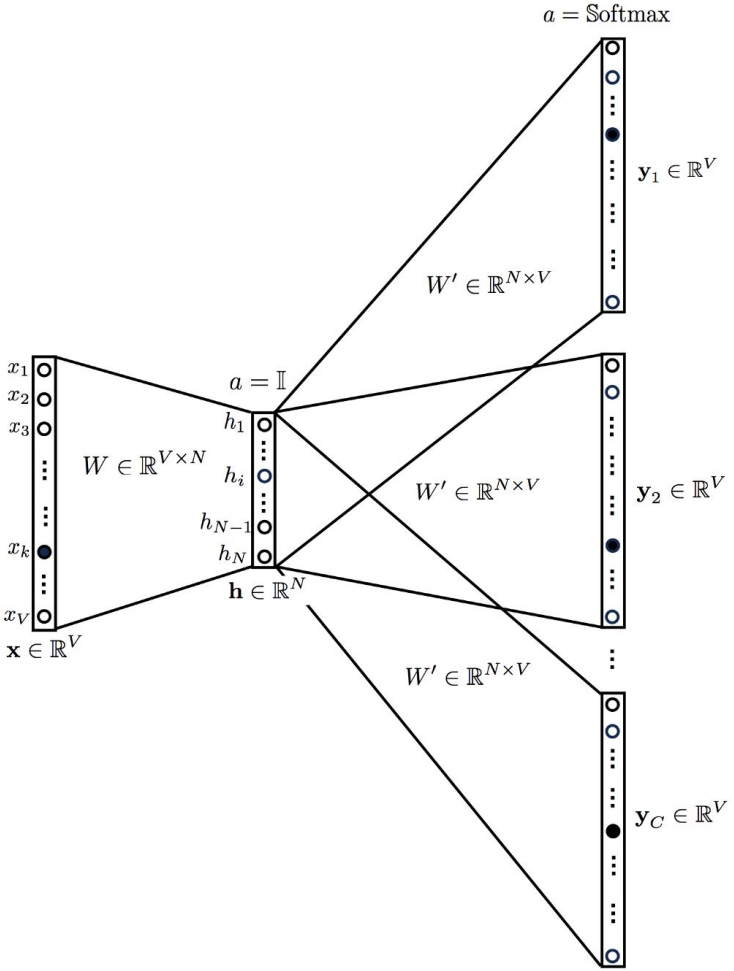
Источник картинки

Метод CBOW пытается **предсказать слово по его контексту**. Он берёт каждое слово из контекста слова Y и пытается по нему предсказать слово Y.

skipgram

skipgram, в отличие от CBOW, пытается предсказывать контекст по слову.

- **Skip Gram** хорошо работает с маленьким объёмом данных и **лучше представляет редкие слова**
- **CBOW** работает быстрее и **лучше представляет наиболее частые слова**



Веб-интерфейсы и ресурсы про word2vec

rusvectores — для русского

tutorial по word2vec — для английского

хорошее объяснение про word2vec и fasttext (англ)

word2vec tutorial на kaggle

Fasttext

Fasttext — почтиии то же самое, что и word2vec, но работает на уровне меньше, чем слово.

Идея такая: разбиваем каждое слово на *символьные нграммы*. Например, так: **apple → app**, **ppl**, **ple**

Обучаем нейросетку так, чтобы получить эмбеддинги этих кусочков. Финальный эмбеддинг слова — сумма эмбеддингов его кусочков.

В чём профит? Умеем представлять даже слова, которых не было в корпусе!

Где взять готовые эмбеддинги

Можно обучить свои эмбеддинги. Но это долго и не всегда нужно. Есть ли уже обученные эмбеддинги? Конечно!

Rusvectores! (для русских слов)

WMD

WMD

Ресурсы

Почитать

- Understanding Activation Functions in Neural Networks
- Activation Functions in Neural Networks
- Neural networks and back-propagation explained in a simple way
- Introduction to Word Embedding and Word2Vec
- Word2Vec and FastText Word Embedding with Gensim
- про WMD

Посмотреть (про нейросети)

Отличная серия видео про нейросети понятным языком:

- But what *is* a Neural Network
- Understanding Gradient Descent
- What backpropagation is really doing?
- Math for backpropagation

Livecoding a NN library (на странноватом новом питоне).