**БАЗОВЫЕ ПОНЯТИЯ**

Поступившие на входы сигналы умножаются на свои веса. Сигнал первого входа ​x1​ умножается на соответствующий этому входу вес ​w1​. В итоге получаем ​x1w1​. И так до ​n​-ого входа. В итоге на последнем входе получаем ​xnwn​.

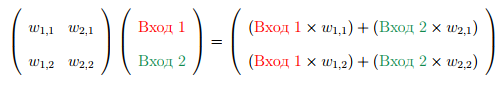
Теперь все произведения передаются в сумматор нейрона. Уже исходя из его названия можно понять, что он делает. Он просто суммирует все входные сигналы, умноженные на соответствующие веса. Это называется **взвешенной суммой** (*weighted sum*).

Роль сумматора очевидна – он агрегирует все входные сигналы (которых может быть много) в какое-то одно число – взвешенную сумму, которая характеризует поступивший на нейрон сигнал в целом. Еще взвешенную сумму можно представить как степень общего возбуждения нейрона.

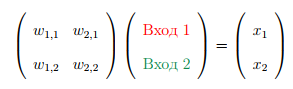
[Здесь с примерами и все очень понятно.](https://neuralnet.info/chapter/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D0%B8%D0%BD%D1%81/#%D0%92%D1%85%D0%BE%D0%B4%D1%8B-%D0%B2%D0%B5%D1%81%D0%B0-%D0%B8-%D1%81%D1%83%D0%BC%D0%BC%D0%B0%D1%82%D0%BE%D1%80)

Для расчета взвешенной суммы используют **матрицы**. С ними получается намного удобнее в случае, когда у нас много слоев или нейронов в каждом слое (алгоритмически по очереди вычисления будут очень громоздкими и могут привести к ошибке).

Для трехслойной нейросети с двумя нейронами это наглядно выглядит так (матрица связей между первым и вторым слоем):



Записывая сокращенно, то, что мы получаем на выходе - и есть взвешенные суммы.



**Формула расчета матрицы** взвешенных сумм такова:

**X = W** ✖ **I**

где **Х** - итоговая матрица взвешенных сумм (аргумент функции-активатора)

**W** - матрица всех весов связей. Для связей между каждым слоем нужна собственная матрица (размерность: число нейронов первого слоя ✖ число нейронов второго слоя). То есть, если слоя три (входной, скрытый, выходной), то понадобится две матрицы (вход-скрытый, скрытый-выход)

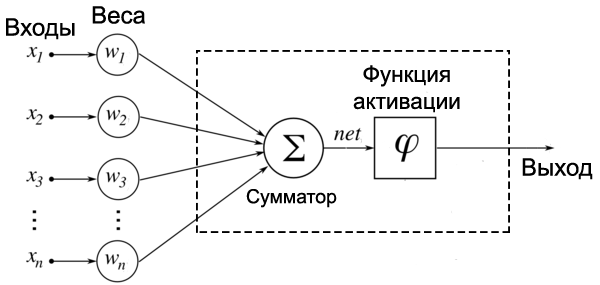
**I** - матрица входов сети

На выходе из сумматора получается число, но как на основе этого числа выявить ответ нейросети? Очевидно, нам нужно как-то преобразовать нашу взвешенную сумму и получить ответ.

Вот на сцену выходит функция активации. Просто так подавать взвешенную сумму на выход достаточно бессмысленно. Нейрон должен как-то обработать ее и сформировать адекватный выходной сигнал. Именно для этих целей и используют функцию активации.

**Функция активации** (*Activation function*) (​ϕ(net)​) — функция, принимающая взвешенную сумму как аргумент. Значение этой функции и является выходом нейрона (​out​).

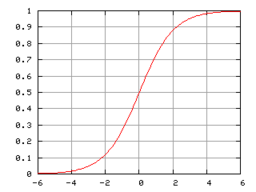
**out=ϕ(net)**

****

**Виды функций активаций:**

* **функция единичного скачка**  - самый простой вид функции активации. Выход нейрона может быть равен только 0 или 1. Если взвешенная сумма больше определенного порога ​b​, то выход нейрона равен 1. Если ниже, то 0.
* **сигмоидальная функция**

**Сигмо́ида** — это гладкая монотонная нелинейная функция, имеющая форму буквы "S", которая часто применяется для «сглаживания» значений некоторой величины. Возрастающая функция. Имеет непрерывную производную.



.

* + **логистическая функция**

Использование логистической функции в качестве функции активации приведет к тому, что вы будете получать цифру между 0 и 1. Причем чем больше взвешенная сумма, тем ближе выход будет к 1 (но никогда не будет точно ей равен). И наоборот, чем меньше взвешенная сумма, тем ближе выход нейрона будет к 0. Используется чаще всего.

Чем она так хороша?

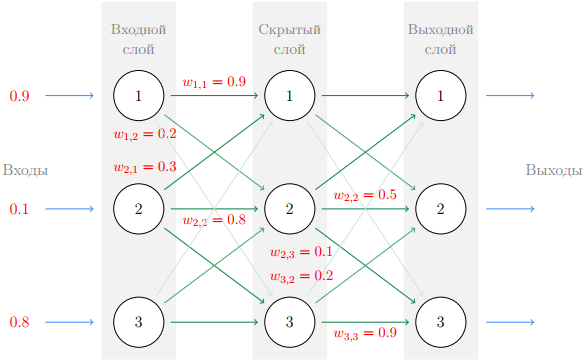
* + - она является «сжимающей» функцией, то есть вне зависимости от аргумента (взвешенной суммы), выходной сигнал всегда будет в пределах от 0 до 1
    - она более гибкая, чем функция единичного скачка – ее результатом может быть не только 0 и 1, но и любое число между ними
    - во всех точках она имеет производную, и эта производная может быть выражена через эту же функцию
  + **гиперболический тангенс**

Он применяется в качестве функции активации биологами для более реалистичной модели нервной клетки.

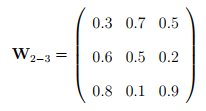
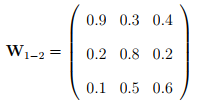
Такая функция позволяет получить на выходе значения разных знаков (например, от -1 до 1), что может быть полезным для ряда сетей. В остальном все плюсы точно такие же, как и в логистической функции (и график такой же).

**АЛГОРИТМ ПОДСЧЕТА ВЫХОДА**

* если сеть состоит из двух слоев (вход и выход), достаточно применить формулу X = W ✖ I.
* если в сети больше одного слоя (есть скрытые):



1. Составляем матрицу весов для первой пары слоев (размерность: количество нейронов второго слоя ✖ количество нейронов первого слоя)



1. Умножаем входной сигнал на полученную матрицу: X = W ✖ I
2. Каждый элемент матрицы Х (взвешенные суммы) используем как аргумент функции-активатора О = сигмоида (Х)
3. Получаем новую матрицу, которую используем в качестве входного сигнала I на следующем слое. Если данный слой - последний, то полученная матрица - и есть отклик нейросети. Если нет, см. п. 5
4. Возвращаемся к пункту 1.

**ПРАВИЛА ОБУЧЕНИЯ**

**ПРАВИЛО ХЕББА**

**= обучение с коррекцией ошибки**

Допустим, что задача - распознавание определенного символа (цифру пять из всех чисел от 0 до 9). Смотреть [код](https://github.com/leramorozova/Neurostudies/blob/master/percepton_number_recogniser.py).

Что важно учесть:

* если нейросеть правильно распознала цифру пять, ничего делать не надо (ведь все и так хорошо)
* если нейросеть ошиблась и
  + распознала неверную цифру как пять, то нужно уменьшить веса тех связей, по которым прошел сигнал
  + не распознала цифру пять, но нужно увеличить веса тех связей, по которым прошел сигнал

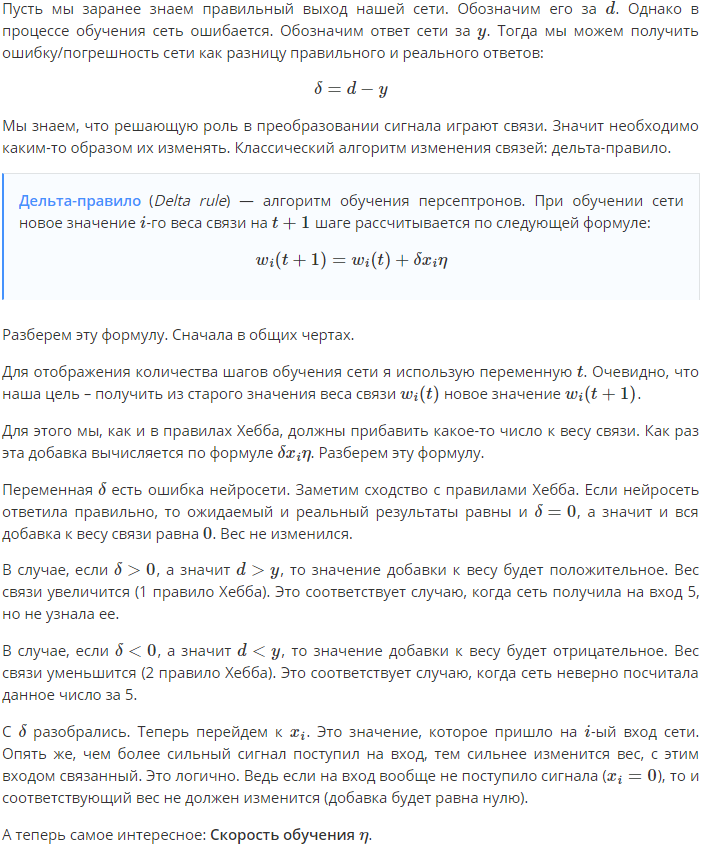
Алгоритм для программы:

1. Подать на входы нейросети цифру в строковом формате.
2. Если цифра распознана/отвергнута верно, то перейти к шагу 1.
3. Если сеть ошиблась и распознала неверную цифру как 5, то вычесть из всех связей, связанных с возбудившимися S-элементами единицу.
4. Если сеть ошиблась и отвергла цифру 5, то добавить единицу ко всем связям, связанным с возбудившимися S-элементами.

Можно вычитать и прибавлять и не единицу - выбор **шага обучения** влияет на эффективность.

**ДЕЛЬТА-ПРАВИЛО**

**обобщение правила Хебба на любые входы (не только 0 и 1)**

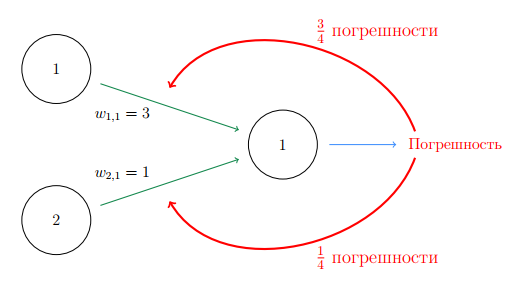
[****](https://neuralnet.info/chapter/%D0%BF%D0%B5%D1%80%D1%81%D0%B5%D0%BF%D1%82%D1%80%D0%BE%D0%BD%D1%8B/#%D0%9F%D0%BE%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B8)

**МЕТОД ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ**

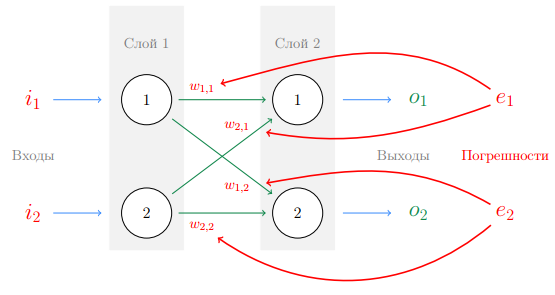
На одном слое это выглядит так: находим погрешность (ожидаемое - полученное) мы. Мы не знаем, по какой именно связи пришел ошибочный сигнал, поэтому не можем вычитать погрешность из определенного веса. Поэтому мы просто поделим ее между всеми весами, которые передают сигнал в выходной нейрон. Однако делим мы его не равномерно - чем больше вес, тем большую часть погрешности мы ему отдадим. Дело в том, что чем больше вес связи, тем большее участие она приняла в формировании ошибочного ответа, тем большее наказание должна понести.

Доля ошибки - это вес данной связи, деленный на сумму всех весов, связанных с данных нейроном:

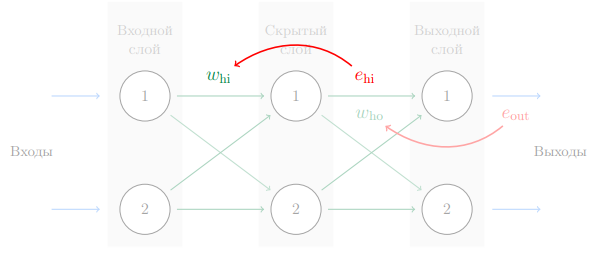




Если выходных нейронов с ошибочным откликом больше одного, то проводим подобную операцию над каждым нейроном.

****

А если слоев больше, чем два? Тогда нам нужно повторить операцию над всеми связями к нейронам скрытого слоя (***whi****)*. Но для этого нам необходимо выяснить погрешности этих нейронов (***ehi***). Напрямую они не находятся, но их возможно вычислить.



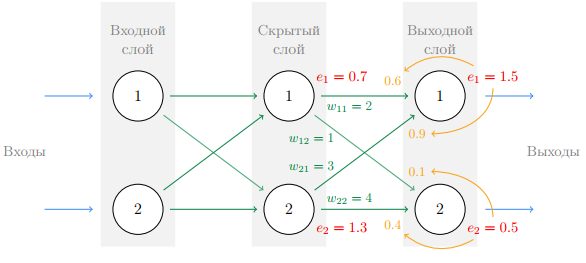
Сначала повторяем известный алгоритм:

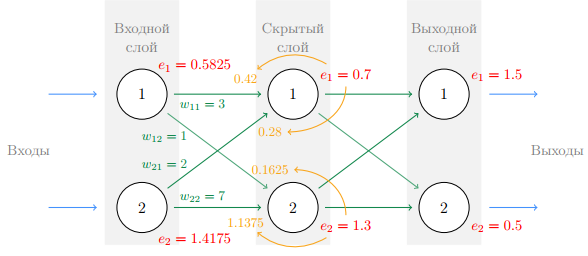
1. Находим погрешности нейронов выходного слоя (***eout***)
2. Делим погрешность пропорционально между связями (***whо****)*

Теперь кое-что новенькое:

1. Складываем погрешности, получившуюся на всех связях (***whо****)*, которые ведут к нужному нам нейрону скрытого слоя
2. ????
3. PROFIT

Так мы нашли погрешность нейрона скрытого слоя. Далее надо повторять эту операцию до тех пор, пока мы не дойдем до входного слоя.



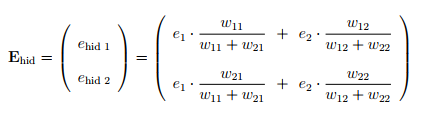


Погрешность высчитывается от начала к концу, поэтому этот метод корректировки весов называется **методом обратного распространения**.

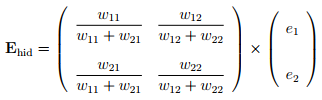
Расчет погрешности одного нейрона скрытого слоя можно собрать в такую формулу:



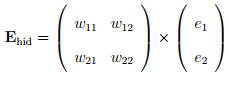
А все погрешности можно представить в виде матрицы, как всегда:



А в этой матрице спряталось умножение:



А знаменатели можно убрать (это повлияет на корректировку, но компенсируется в ходе обучения, зато проще использовать:



В итоге первый множитель у нас - транспонированная матрица связей **W**, которую мы использовали для подсчета выхода сигнала. Круто!

Из этого у нас получается совсем простая формула подсчета погрешности нейронов скрытого слоя:

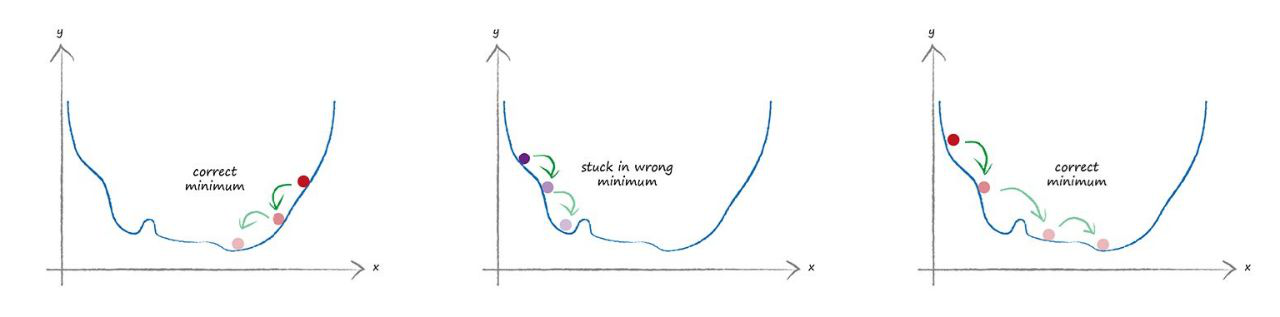
Ehid = WT ✖ Eout

Означает ли это, что мы можем выставить случайные веса и просто корректировать их методом обратного распространения ошибки? Нет!

Так как все веса связаны, изменения веса, , связанного с нейроном, принесшим ошибочный результат, приведет к тому, что вес повлияет и на другие нейроны (которые с ошибкой могли не иметь ничего общего и выдавать правильный результат). В итоге вместо оптимизации весов мы можем просто испортить их. Поиск оптимального варианта методом подбора может занять очень много времени (чем больше нейронов и связей, тем дольше), пока наконец не найдется единственная наиболее оптимальная комбинация.

Что же с этим делать? Не рубить с плеча и использовать **метод градиентного спуска**. Мы корректируем веса в ту сторону, в которой величина ошибки уменьшается. Величина, на которую мы корректируем значения весов, называется **шагом обучения**. Если взять слишком большой шаг, то мы можем случайно “перепрыгнуть” минимум. Если шаг слишком маленький, то достижение минимума будет идти слишком долго. Оптимальный вариант - брать большой шаг и уменьшать его по мере приближения к минимуму.

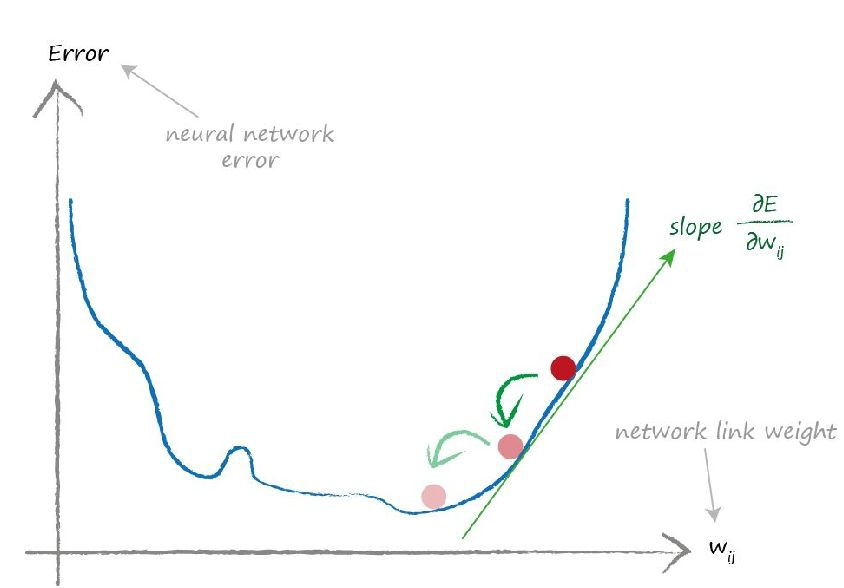
Как найти действительно наименьший минимум, а не застрять в тупике, как на картинке? Начинать путь с разных точек = проводить обучение по разным выборкам, чтобы понять, какой результат действительно наименьший.



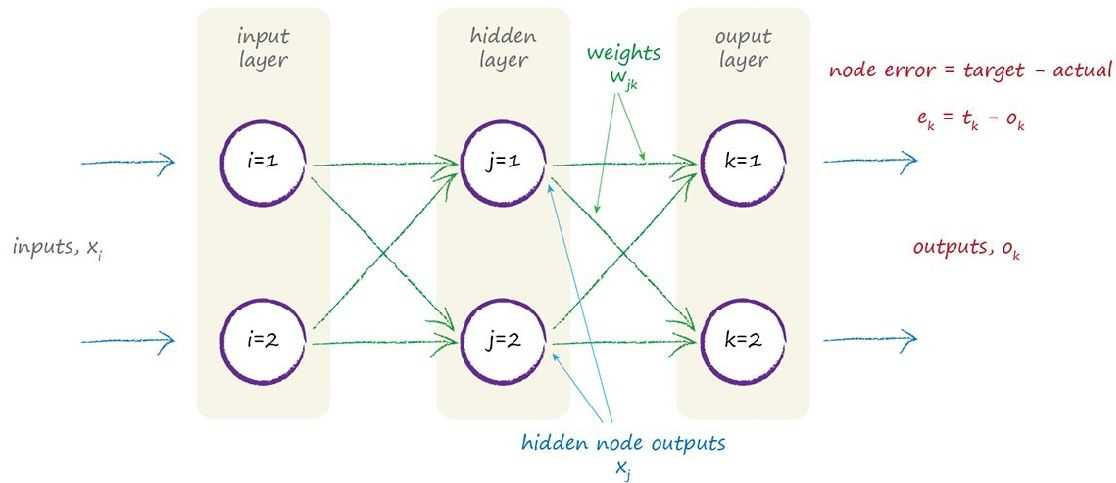
Сам по себе выход нейрона - не функция ошибки, но последнюю получить очень просто (цель - реальный результат). Брать значение в чистом виде не стоит, потому что в случаях, когда значения ошибок противоположны друг другу (1 и -1, например), их вычитание из веса взаимоисключает друг-друга, и мы будем топтаться на месте, вместо того чтобы идти к минимуму функции. Модуль тоже не поможет - на V-образном графике мы спускаемся к минимуму равномерно, так что по значениям будет не понятно, когда мы приблизимся к минимуму, и можем его перескочить. Выход? Взять **квадрат ошибки**. Это удобно по нескольким причинам:

* квадратичная функция непрерывная и плавная
* она удобнее для дифференцирования (это пригодится дальше)
* градиент будет уменьшаться по мере приближения к минимуму, что снижает вероятность перескакивания через точку минимума

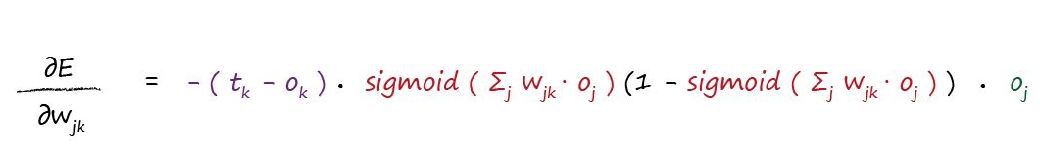
Дифференцирование нужно нам для того, чтобы определить значение веса связи при заданной величине ошибки. Для этого нам нужна функция ошибки - с помощью дифференцирования мы ее и получим.



Эта дробь в левом верхнем углу означает следующее: насколько изменение ошибки зависит от изменения веса? Вот схема для того, чтобы понимать дальнейшие условные обозначения:

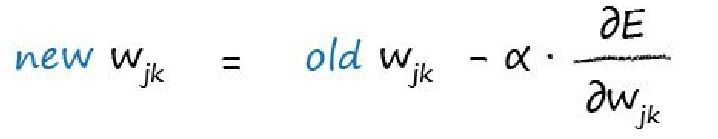


Путем нехитрых преобразований мы получаем такое выражение:

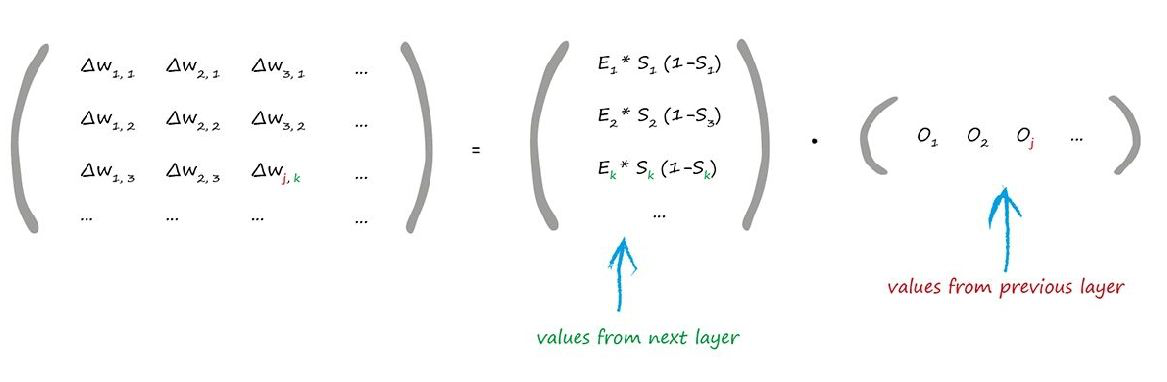


Мы расписали Е, потом продифференцировали полученное по цепному правилу (aka как сложную функцию). Потом расписали выход из нейрона k (выражение внутри сигмоиды - это просто взвешенная функция, а потом нашли производную и от сигмоиду тоже. Этим же равенством можно выразить ошибку на любом слое, надо только симметрично изменить коэффициенты.

Как нам с помощью этого обновлять веса? Вот формула:



Но, как всегда, если применить все на матрицах, все оказывается намного проще.



Если расписать ошибку каждого нейрона, можно найти умножение матриц и расписать его (альфы здесь нет - она сейчас несущественна). А вторая матрица является ничем иным, как транспонированной матрицей выходов сигналов из прошлого слоя нейронов J. Это открытие позволяет нам упростить формулу еще:



**ПОДГОТОВКА ДАННЫХ**

1. **Входные значения** не должны быть ни слишком большими, ни слишком маленькими, т. к. это может повлиять на вычислительную точность. В идеале числа должны от 0.01 до 1.0. Инпуты в виде нуля проблемны - весовые коэффициенты не повлияют на ответ (потому что при умножение нуля на любое число будет получен ноль), что стопорит обучение.
2. **Целевые выходные данные** при обучении должны находиться строго в диапазоне 0 < x < 1 (числа вне этого диапазона сигмоида просто не сможет породить).
3. Просто рандомные **веса** - не самое оптимальное решение. Веса, представляющие из себя набор одинаковых констант (особенно нулей) - тоже. Если все веса будут одинаковы, все выходы будут одинаковы. Значит, у нас будут одинаковые ошибки, которые мы будем одинаково делить между связями. Нужно помнить, что в прототипической нейросети все веса разные. Нули еще хуже - они просто аннулируют входной сиграл.

По п*равилу большого пальца*, лучший вариант:

* 1. посчитать, сколько связей входит в каждый нейрон
  2. взять квадрат этого числа (назовем полученный квадрат Х)
  3. брать веса в диапазоне от -(1/Х) < вес < (1/Х)

Это сделает сумму весов достаточно маленькой, чтобы не насытить функцию-активатор. К тому же, чем больше вес, тем большее отклонение он будет давать (если он неправильный).

**ПО ЧАСТИ ПРОГИ**

Обозначения:

**bias** - (иногда) порог взвешенной суммы для функции-активации

**net** - взвешенная сумма весов

**delta** - погрешность/ошибка сети по дельта-правилу

Богоподобное [объяснение](https://cloud.mail.ru/public/FHKA/kzwQdnFMs) объектно-ориентированного программирования.