

01.1 - Функции активации

Функции активации

Определяют выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения.

Применяются к нейрону после того, просчитанному матричным умножением. Определяет, передавать ли полученное значение следующему нейрону (активировать его) или нет, и модифицировать ли как-то полученное значение.

Рассмотрим подробнее такие функции активации, как: - ступенчатая - линейная (linear) - сигмоидная (sigmoid) - гиперболического тангенса (tahn) - ReLU - softmax

Рассмотрим нейрон, у которого взвешенная сумма равна z .

$$z = \sum_i w_i x_i + b$$

где w_i - вес и входное значение i -го входа, а b - смещение.

Полученный результат передается в функцию активации, которая решает рассматривать этот нейрон как активированный, или его можно игнорировать.

Ступенчатая

Ступенчатая функция (англ. binary step function) является пороговой функцией активации. То есть если z больше или меньше некоторого значения, то нейрон становится активированным. Такая функция отлично работает для бинарной классификации. Но она не работает, когда для классификации требуется большее число нейронов и количество возможных классов больше двух.

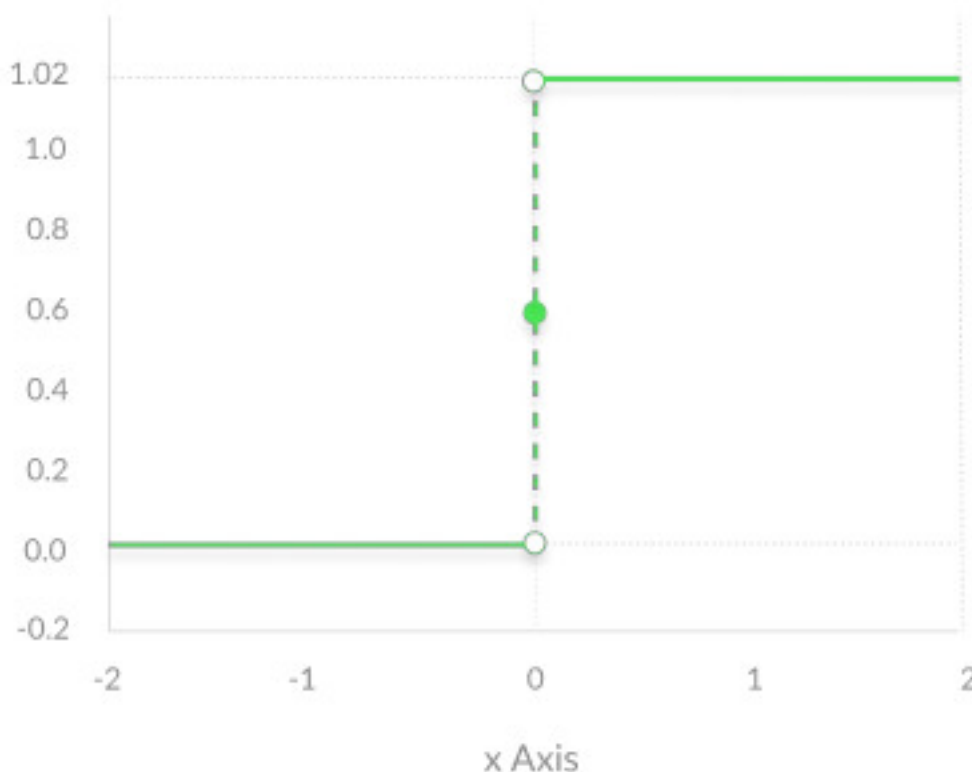


Рис. 1: 140d6bb395dedd0efd98490a80aebbe2.png

Линейная (linear)

Линейная функция (англ. linear function) представляет собой прямую линию, а это значит, что результат этой функции активации пропорционален переданному аргументу. В отличие от предыдущей функции, она позволяет получить диапазон значений на выходе, а не только бинарные 0 и 1, что решает проблему классификации с большим количеством классов.

У линейной функции есть две основных проблемы:

1. Невозможность использования метода обратного распространения ошибки. Так как в основе этого метода обучения лежит градиентный спуск, а для того чтобы его найти, нужно взять производную, которая для данной функции активации — константа и не зависит от входных значений. То есть при обновлении весов нельзя сказать улучшается ли эмпирический риск на текущем шаге или нет.
2. Так как для каждого слоя выходное значение линейно, то они образуют линейную комбинацию, результатом которой является линейная функция. То есть финальная функция активации на последнем слое зависит только от входных значений на первом слое. А это значит, что любое количество слоев может быть заменено всего одним слоем, и, следовательно, нет смысла создавать многослойную сеть.

Главное отличие линейной функции от остальных в том, что ее область определения не ограничена: $(-\infty; +\infty)$. Следовательно, ее нужно использовать, когда выходное значение нейрона должно $\in \mathbb{R}$, а не ограниченному интервалу.

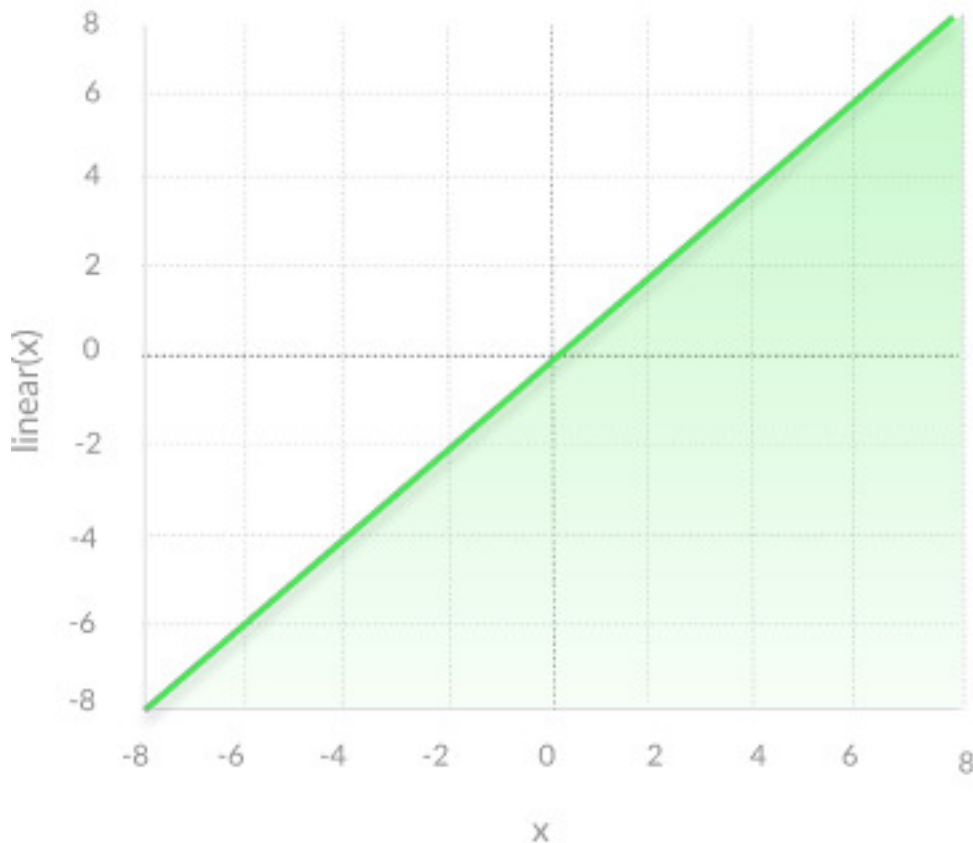


Рис. 2: a62dece54fdb3c84216427a9030a4de6.png

Сигмоидная функция (sigmoid)

Сигмоидная функция (англ. sigmoid function), которую также называют логистической (англ. logistic function), является гладкой монотонно возрастающей нелинейной функцией. Преобразует выходное значение нейрона в значение из диапазона $[0, +1]$, смещает среднее значение с 0 и используется в случаях, когда нужно получить что-то похожее на вероятность.

Хорошо себя показывает в LSTM-сетях, логистической регрессии.

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

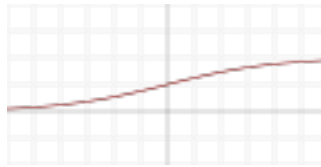


Рис. 3: 7820569e7c12dfc1d9d44cff49cefd89.png

И так как эта функция нелинейна, то ее можно использовать в нейронных сетях с множеством слоев, а также обучать эти сети методом обратного распространения ошибки.

- Сигмоида ограничена двумя горизонтальными асимптотами $y=1$ и $y=0$, что дает нормализацию выходного значения каждого нейрона.
- Кроме того, для сигмоидной функции характерен гладкий градиент, который предотвращает “прыжки” при подсчете выходного значения.
- Помимо всего этого, у этой функции есть еще одно преимущество, для значений $x>2$ и $x<-2$, y “прижимается” к одной из асимптот, что позволяет делать четкие предсказания классов.

Несмотря на множество сильных сторон сигмоидной функции, у нее есть значительный недостаток. Производная такой функции крайне мала во всех точках, кроме сравнительно небольшого промежутка - **проблема исчезающего градиента**. Это сильно усложняет процесс улучшения весов с помощью градиентного спуска. Более того, эта проблема усугубляется в случае, если модель содержит много слоев.

Что касается использования сигмоидной функции, то ее преимущество над другими — в нормализации выходного значения. Иногда, это бывает крайне необходимо. К примеру, когда итоговое значение слоя должно представлять вероятность случайной величины. Кроме того, эту функцию удобно применять при решении задачи классификации, благодаря свойству “прижимания” к асимптотам.

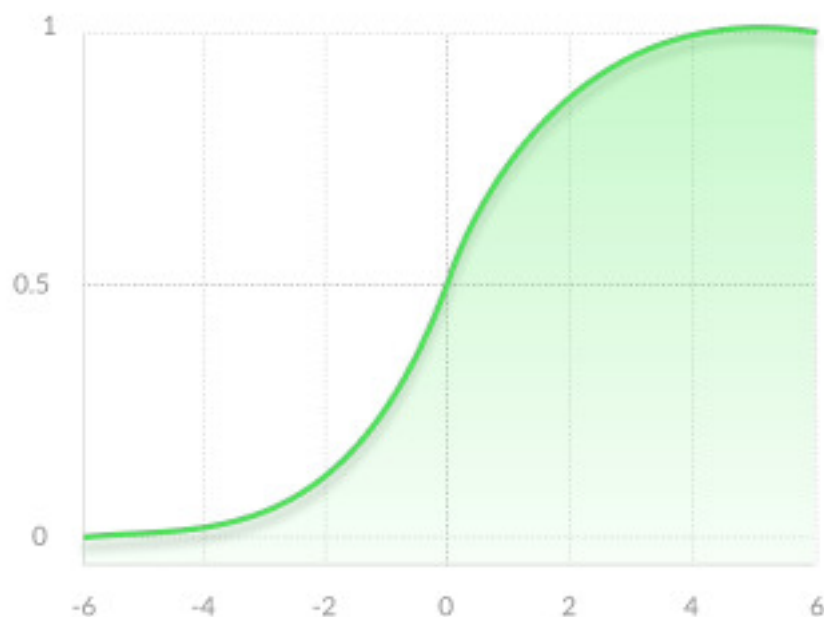


Рис. 4: a11c92f38c962e80d99294445a9f46b1.png

Функция гиперболического тангенса (tahn)

Функция гиперболического тангенса (англ. hyperbolic tangent). Преобразует выходное значение нейрона в значение из диапазона $[-1, +1]$, не смещает среднее значение. Имеет вид:

$$\tanh(z) = \frac{2}{1 + e^{-2z}} - 1 = 2 \cdot \text{sigma}(2z) - 1$$

Эта функция является скорректированной сигмоидной функцией, то есть она сохраняет те же преимущества и недостатки, но уже для диапазона значений $(-1;1)$.

- Обычно, \tanh является предпочтительнее сигмоиды в случаях, когда нет необходимости в нормализации. Это происходит из-за того, что область определения данной функции активации центрирована относительно нуля, что снимает ограничение при подсчете градиента для перемещения в определенном направлении.
- Кроме того, производная гиперболического тангенса значительно выше вблизи нуля, давая большую амплитуду градиентному спуску, а следовательно и более быструю сходимость.

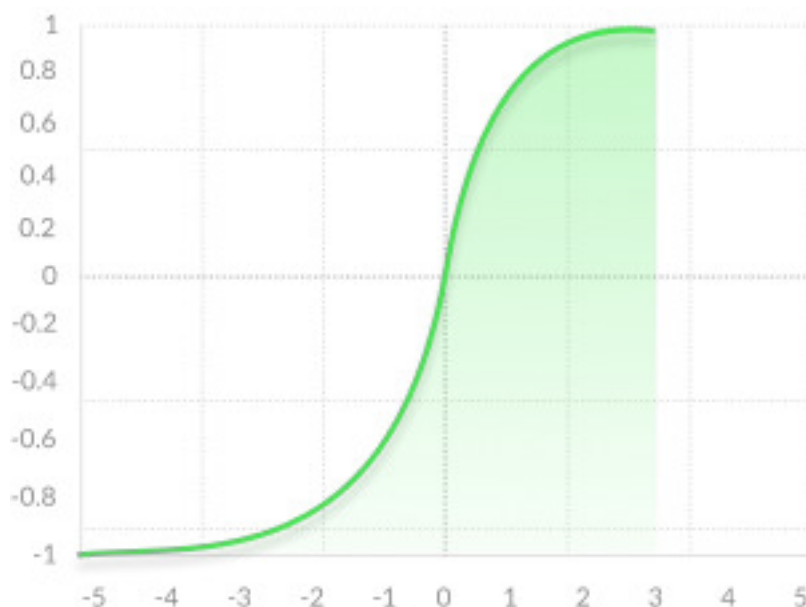


Рис. 5: 5f05cbc65e31de6e59909b4d29b14386.png

Функция ReLU

Функция ReLU (Rectified Linear Unit) — это наиболее часто используемая функция активации при глубоком обучении. Данная функция возвращает 0, если принимает отрицательный аргумент, в случае же положительного аргумента, функция возвращает само число. То есть она может быть записана как $f(z)=\max(0,z)$.

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

Рис. 6: bc2ac67e46bd850cd90bd4bafd2a6231.png

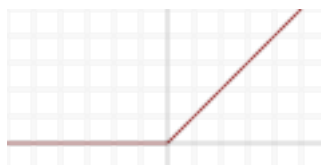


Рис. 7: a0514ab4753e3c3424d875de50ce4ff9.png

На первый взгляд может показаться, что она линейна и имеет те же проблемы что и линейная функция, но это не так и ее можно использовать в нейронных сетях с множеством слоев. Функция ReLU обладает несколькими преимуществами перед сигмоидой и гиперболическим тангенсом:

- Очень быстро и просто считается производная. Для отрицательных значений — 0, для положительных — 1.
- Разреженность активации. В сетях с очень большим количеством нейронов использование сигмоидной функции или гиперболического тангенса в качестве активационной функции влечет активацию почти всех нейронов, что может сказаться на производительности обучения

модели. Если же использовать ReLU, то количество включаемых нейронов станет меньше, в силу характеристик функции, и сама сеть станет легче.

У данной функции есть один недостаток, называющийся **проблемой умирающего ReLU**. Так как часть производной функции равна нулю, то и градиент для нее будет нулевым, а то это значит, что веса не будут изменяться во время спуска и нейронная сеть перестанет обучаться.

Функцию активации ReLU следует использовать, если нет особых требований для выходного значения нейрона, вроде неограниченной области определения. Но если после обучения модели результаты получились не оптимальные, то стоит перейти к другим функциям, которые могут дать лучший результат.

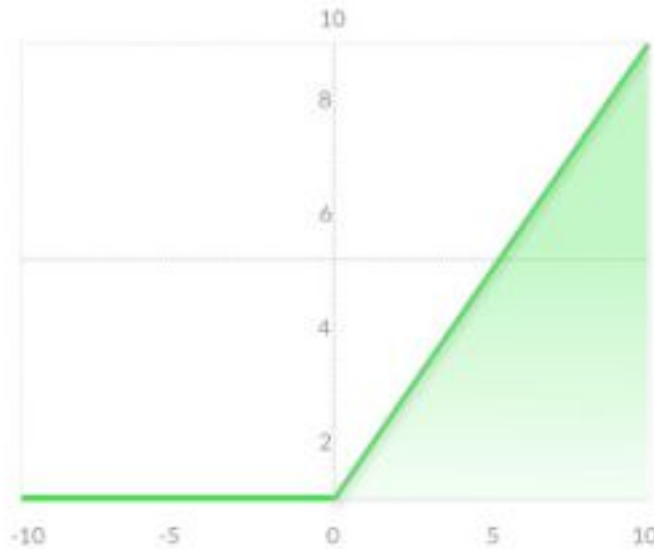


Рис. 8: d28ec2d22208eeb9b328b5f3878801ef.png

Функция Leaky ReLU

Одной из проблем стандартного ReLU является затухающий, а именно нулевой, градиент при отрицательных значениях. При использовании обычного ReLU некоторые нейроны умирают, а отследить умирание нейронов не просто. Чтобы решить эту проблему иногда используется подход ReLU с «утечкой» (leak) — график функции активации на отрицательных значениях образует не горизонтальную прямую, а наклонную, с маленьким угловым коэффициентом (порядка 0,01).

$$f(x) = \begin{cases} 0.01x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

Рис. 9: 3aa10f7c992e09ad97c38e699278baf0.png

Такое небольшое отрицательное значение помогает добиться ненулевого градиента при отрицательных значениях. Однако, функция Leaky ReLU имеет некоторые недостатки:

- Сложнее считать производную, по сравнению со стандартным подходом (так как значения уже не равны нулю), что замедляет работу каждой эпохи.
- Угловой коэффициент прямой также является гиперпараметром, который надо настраивать.
- На практике, результат не всегда сильно улучшается относительно ReLU.

Стоит отметить, что помимо проблемы умирающих нейронов, у ReLU есть и другая — проблема затухающего градиента. При слишком большом количестве слоев градиент будет принимать очень маленькое значение, постепенно уменьшаясь до нуля. Из-за этого нейронная сеть работает нестабильно и неправильно. Leaky ReLU (LReLU) решает первую проблему, но в по-настоящему глубоких сетях проблема затухания градиента все еще встречается и при использовании этого подхода.

На практике LReLU используется не так часто. Требуется дополнительно настраивать гиперпараметр (уровень наклона при отрицательных значениях), что требует определенных усилий.

Еще одной проблемой является то, что результат LReLU не всегда лучше чем при использовании обычного ReLU, поэтому чаще всего такой подход используют как альтернатива. Довольно часто на практике используется PReLU (Parametric ReLU), который позволяет добиться более значительных улучшений по сравнению с ReLU и LReLU.

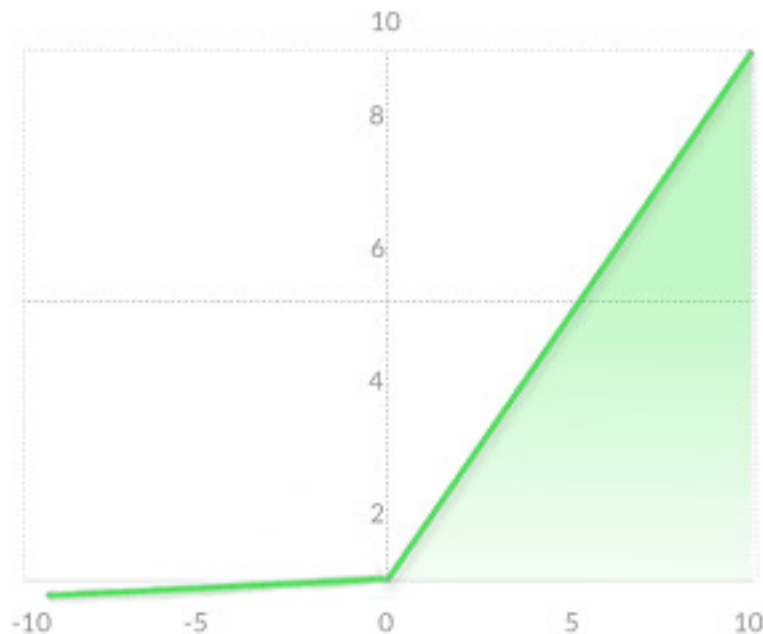


Рис. 10: 46195d6dcf2f4d30a15985b54483f4b7.png

Функция Softmax

Softmax — это обобщение логистической функции (уравнение Ферхюльста) для многомерного случая.

Функция преобразует вектор x размерности K в вектор σ той же размерности, где каждая координата σ_i полученного вектора представлена вещественным числом в интервале $[0, 1]$ и сумма координат равна 1.

Координаты σ_i вычисляются следующим образом:

Функция Softmax применяется для задач классификации, когда количество возможных классов больше двух (для двух классов используется логистическая функция). Координаты σ_i полученного вектора при этом трактуются как вероятности того, что объект принадлежит к классу i .

Давайте рассмотрим принцип работы функции softmax на простом примере. У нас есть входной вектор inp из 4 значений:

```
inp = [-4, 5, 2, -8]
```

Напишем функцию для расчета i -той координаты:

```
import numpy as np
def soft(i, data):
    param1 = np.exp(data[i])           # Получение значения в числителе
    param2 = np.exp(data).sum()        # Получение значения в знаменателе
    return param1/param2
```

А теперь получим результат применения функции softmax к нашему набору:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{k=1}^k e^{x_k}}$$

Рис. 11: d245cc8bb08ca4c853b6d35d2c09e3c1.png

```

out1 = soft(0, inp)           # Получение результата работы софтмакс-функции
    ↪ к первому элементу входных данных
out2 = soft(1, inp)           # Получение результата работы софтмакс-функции
    ↪ ко второму элементу входных данных
out3 = soft(2, inp)           # Получение результата работы софтмакс-функции
    ↪ к третьему элементу входных данных
out4 = soft(3, inp)           # Получение результата работы софтмакс-функции
    ↪ к четвертому элементу входных данных
out = [out1, out2, out3, out4] # Формирование результирующего вектора
print(out)
[0.00011754291529297411, 0.9524601077108533, 0.047420196500263315, 2.1528735904351842e-
06]

```

Проверим полученный результат (сумма всех значений результирующего вектора должна равняться единице):

```

sum(out)
1.0

```

Немного дополнительной теории

Активационные функции являются неотъемлемым звеном в архитектуре нейронной сети.

Определение активационной функции:

Активационная функция - это универсальная математическая формула, согласно которой сумма произведений входных значений и весовых коэффициентов преобразуется в упорядоченные выходные значения в рамках одного слоя нейронной сети.

В библиотеке Keras имеется хороший выбор активационных функций, алгоритм которых написан заранее для удобства разработчиков.

В архитектуре сети различают входной, скрытые и выходной типы слоёв. Активационные функции применяются для каждого скрытого и выходного слоёв индивидуально - указываются как гиперпараметр:

```

model.add(Dense(1, activation= "linear" ))

```

Чтобы получить наилучшую точность модели сети разработчик должен уметь правильно подобрать функцию активации (она же - активационная функция) для конкретного слоя.

Так как про область применения функций активации уже было сказано, закрепим данный пункт - необходимо провести условное различие между функциями для скрытых слоёв и функцией выходного слоя.

Для скрытых слоёв наиболее часто используются следующие функции активации:

Sigmoid - преобразует малые значения (< -5) в близкие к нулю, а большие значения (> 5) в близкие к единице.

Вычисляется по программной формуле (Python): $\text{sigmoid}(x) = 1 / (1 + \exp(-x))$

ReLU - возвращает входные значения без изменения, если знак значений положительный; отрицательные значения преобразует к нулю.

Вычисляется по программной формуле (Python): $\text{relu} = \max(x, 0)$

Tanh - функция гиперболического тангенса; возвращает различные значения в диапазоне $(-1; 1)$.

Вычисляется по программной формуле (Python): $\tanh(x) = \sinh(x) / \cosh(x) = ((\exp(x) - \exp(-x)) / (\exp(x) + \exp(-x)))$

Скрытые слои предусматривают использование одной и той же функции для каждого из них, однако понимание типа предназначения каждого слоя может помочь разработчику подобрать функцию в индивидуальном порядке. Так, функция активации ReLU даёт хорошие результаты точности в полносвязных слоях (Dense layers) и свёрточных слоях (Convolutional layers). А функция гиперболического тангенса Tanh наравне с функцией Sigmoid используются в рамках моделей LSTM (Long Short Term Memory).

Важно запомнить, что упомянутые функции преобразуют входные значения, а именно - сумму произведений входных данных и весовых коэффициентов, в новые значения. В некоторых случаях разработчику требуется передать входные значения на выход слоя без изменений. Тогда лучший вариант - активационная функция linear (линейная функция). Линейная функция вычисляется по математической формуле $A = cx$, где A - значение функции, c - весовые коэффициенты, x - входные значения. Эта функция была приведена в качестве примера гиперпараметра в начале ноутбука и применяется как к скрытым, так и выходным слоям.

Подробнее о подборе функции для выходного слоя. Выходной слой на выходе сообщает сами значения предсказания модели, а значит при подборе функции в данном случае разработчик должен чётко разбираться, с каким типом задачи работает модель. Конечно, для выходного слоя можно привести всё тот же полный список возможных функций активации. Но для лучшего запоминания полный список эффективнее отфильтровать по наиболее часто применяемым:

Softmax - распределяет выходные значения пропорционально вероятности каждого из них в диапазоне $(0; 1)$; это означает, что сумма выходных значений всегда будет равна 1 (что эквивалентно 100% общей вероятности); данная функция хорошо подходит для задач по многоклассовой классификации.

Вычисляется по программной формуле (Python), применимой к каждому входному вектору индивидуально:

$\exp(x) / \text{tf.reduce_sum}(\exp(x))$

Linear (рассмотрена ранее в ноутбуке). Sigmoid (рассмотрена ранее в ноутбуке). Линейная функция активации linear используется, например, для задач регрессии. Для бинарной и многокритериальной классификации - функция sigmoid, а для многоклассовой, как уже было сказано, используется функция softmax.

Помните, что количество значений предсказания модели не зависит от функции активации, а определяется числом нейронов выходного слоя. Активационные функции определяют сами выходные значения, а не их число.