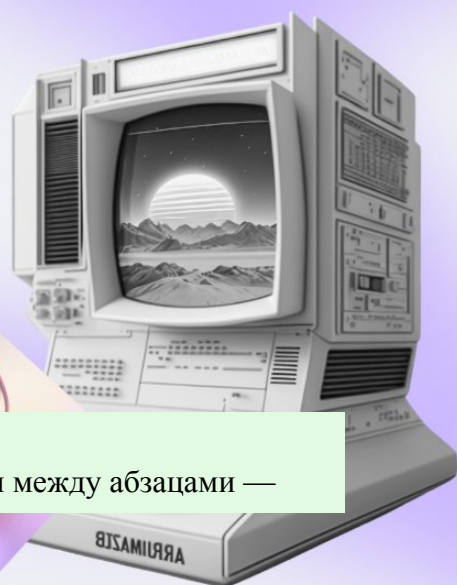


# Перцептрон и многослойный перцептрон

Архитектура нейронных сетей



- Шрифт – IBM Plex Sans  
Размер текста — 12, межстрочный интервал — 1,15, интервал между абзацами —

10, выравнивание по ширине

Размер заголовков первого уровня — 22, второго — 18, третьего — 16

# Оглавление

|   |    |
|---|----|
| Введение  | 3  |
| Словарь терминов  | 3  |
| Перцептроны и многослойные перцептроны                          | 3  |
| Почему глубокое обучение меняет правила игры                    | 3  |
| Все началось с Нейрона  | 4  |
| Персептрон  | 6  |
| Функции активации   | 7  |
| Функция активации сигмоида                                      | 7  |
| Функция активации танга/гиперболического тангенса               | 8  |
| Функция активации ReLU и Leaky ReLU                             | 9  |
| Персептрон для двоичной классификации                           | 10 |
| Собираем все это вместе   | 11 |
| Многослойный персептрон   | 12 |
| Обратное распространение ошибки                                 | 13 |
| Прямое распространение:   | 15 |
| Прямое распространение и Прогнозирование в Нейронных Сетях      | 15 |
| Как прямое распространение связано с обратным распространением? | 16 |
| Заключение  | 17 |

# Введение

Всем привет! Это наша вторая лекция на курсе архитектура нейронных сетей и сегодня мы с вами посмотрим на самую простую, но достаточно эффективную архитектуру перцептрона и многослойного перцептрона. Понимание этих концепций является ключом к вхождению в область глубокого обучения и позволит нам постепенно раскрывать сложные аспекты искусственного интеллекта, а также перейти к изучению более сложных архитектур.

## Словарь терминов

**Перцептрон** — это простейшая форма искусственной нейронной сети

**Многослойный перцептрон** — это расширенная версия перцептрона, которая состоит из нескольких слоев нейронов (или узлов) и позволяет более гибко моделировать сложные зависимости в данных

**Функция активации** — это нелинейная математическая функция, которая применяется к выходу нейрона или нейронной сети после выполнения линейных операций (например, умножения на веса и суммирования входов)

## Перцептроны и многослойные перцептроны

### Почему глубокое обучение меняет правила игры

В традиционном машинном обучении любой, кто строит модель, должен либо быть экспертом в проблемной области, над которой он работает, либо сотрудничать с ним. Без этих экспертных знаний проектирование и разработка функций становятся все более сложной задачей. Качество модели машинного обучения зависит не только от качества набора данных, но и от того, насколько хорошо функции кодируют закономерности в данных.

**Алгоритмы глубокого обучения** используют искусственные нейронные сети в качестве своей основной структуры. Что отличает их от других алгоритмов, так это то, что они не требуют участия экспертов на этапе проектирования и разработки функций. Нейронные сети могут *изучать* характеристики данных.

Алгоритмы глубокого обучения берут набор данных и *изучают* его закономерности. Они *учатся* представлять данные с помощью функций, которые они извлекают самостоятельно. Затем они объединяют различные представления набора данных, каждое из которых идентифицирует определенный шаблон или характеристику, в более абстрактное, высокоуровневое представление набора данных. Такой подход невмешательства, или без особого вмешательства человека в проектирование и извлечение признаков, позволяет алгоритмам гораздо быстрее адаптироваться к имеющимся данным.

**Нейронные сети** основаны на структуре мозга, но не обязательно являются точной моделью. Мы еще многого не знаем о мозге и о том, как он работает, но он послужил источником вдохновения во многих научных областях благодаря своей способности

развиваться и проявлением интеллекта. И хотя существуют нейронные сети, созданные с единственной целью — понять, как работает мозг, глубокое обучение в том виде, в котором мы его знаем сегодня, не предназначено для воспроизведения того, как работает мозг. Вместо этого глубокое обучение фокусируется на создании систем, которые изучают несколько уровней композиции шаблонов.


И, как и любой научный прогресс, **глубокое обучение** началось не со сложных структур и широко распространенных приложений, которые вы видите в современном мире.

Все началось с базовой структуры, напоминающей нейрон мозга .

## Все началось с Нейрона

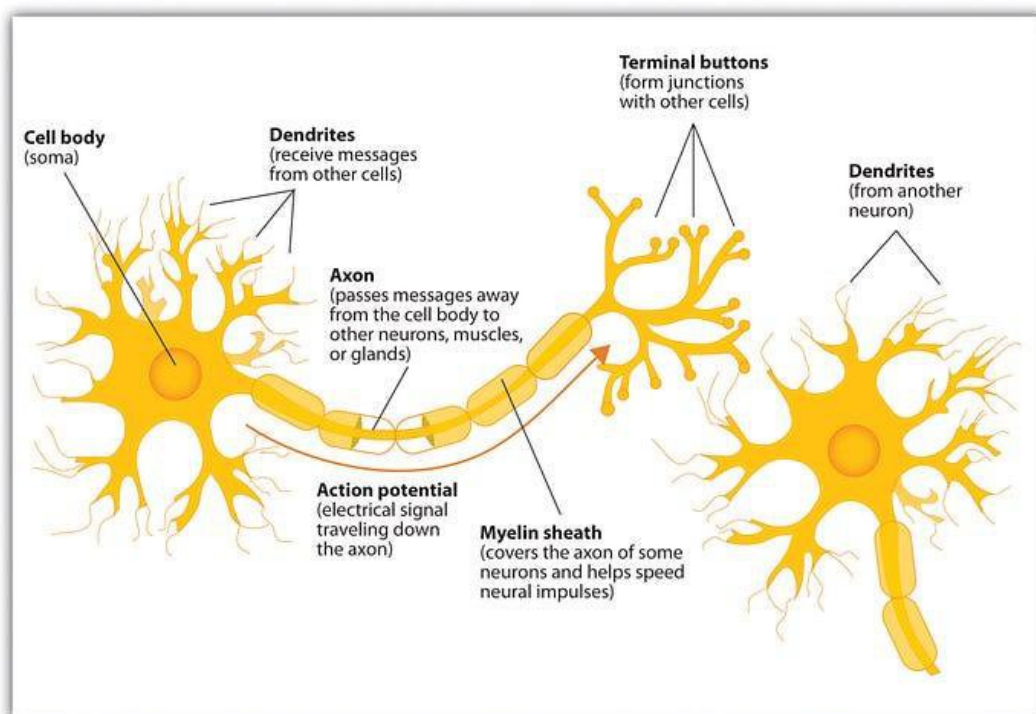
В начале 1940-х годов нейрофизиолог Уоррен Маккалок объединился с логиком Уолтером Питтсом , чтобы создать модель работы мозга. Это была простая линейная модель, которая давала положительный или отрицательный результат при заданном наборе входных данных и весов.

$$\underbrace{f(x, w)}_{\text{output}} = \underbrace{x_1 w_1}_{\text{inputs}} + \dots + x_n w_n$$

  
weights

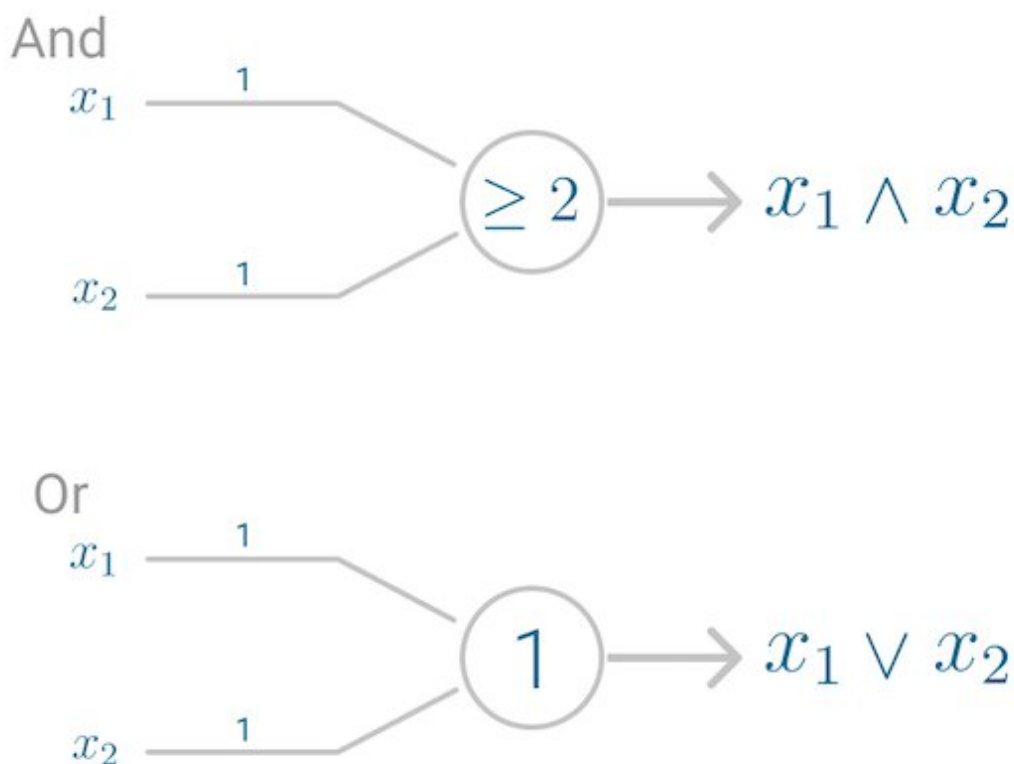
Модель нейронов МакКаллоха и Питтса.

Эту *модель вычислений* намеренно называли **нейронной** , потому что она пыталась имитировать работу основного *строительного блока* мозга. Точно так же, как нейроны мозга получают электрические сигналы, нейрон Маккаллоха и Питтса получал входные данные и, если эти сигналы были достаточно сильными, передавал их другим нейронам.



Нейрон и его разные компоненты. ( [Изображение предоставлено](#) )

Первое применение нейрона воспроизводило логический вентиль , где у вас есть один или два двоичных входа и логическая функция , которая активируется только при наличии правильных входных данных и весов.



Пример логических элементов И и ИЛИ.

Однако у этой модели была проблема. Она не могла *учиться* , как мозг. Единственным способом получить желаемый результат было заранее установить веса, выполняющие роль катализатора в модели.

*Нервная система представляет собой сеть нейронов, каждый из которых имеет сому и аксон [...] В любой момент времени нейрон имеет некоторый порог, который должно превысить возбуждение, чтобы инициировать импульс.*

Лишь десять лет спустя Фрэнк Розенблатт расширил эту модель и создал алгоритм, который мог *изучать* веса, чтобы генерировать выходные данные.

Основываясь на нейроне Мак-Каллоха и Питта, Розенблатт разработал **перцептрон** .

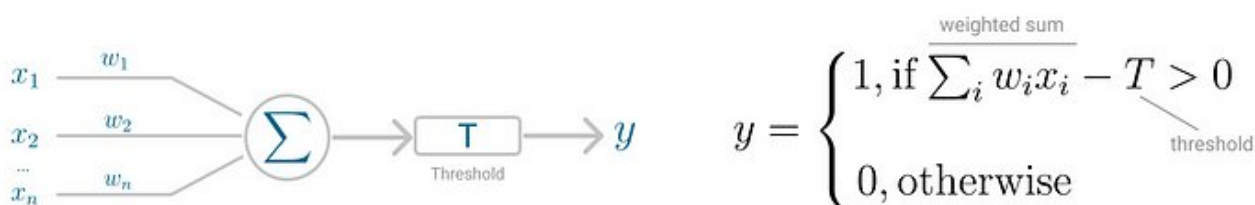
## Персептрон

Хотя сегодня **персептрон** широко известен как алгоритм, изначально он задумывался как машина распознавания изображений. Свое название он получил от выполнения человеческой функции восприятия , видения и распознавания изображений.

*В частности, интерес был сосредоточен на идее машины, которая была бы способна концептуализировать входные данные, поступающие непосредственно из физической среды света, звука, температуры и т. д. — «феноменального мира», с которым мы все знакомы — а не требуя вмешательства человека для переваривания и кодирования необходимой информации.*

Перцептронная машина Розенблатта опиралась на базовую вычислительную единицу — **нейрон** . Как и в предыдущих моделях, у каждого нейрона есть ячейка, которая получает ряд пар входных данных и весов.

Основное отличие модели Розенблатта заключается в том, что **входные данные объединяются во взвешенную сумму** , и, если взвешенная сумма превышает заранее определенный порог, нейрон срабатывает и выдает выходной сигнал.



Модель нейрона перцептронов (слева) и пороговая логика (справа).

Порог  $T$  представляет собой **функцию активации** . Если взвешенная сумма входных данных больше нуля, нейрон выводит значение 1, в противном случае выходное значение равно нулю.

## Функции активации

В большинстве случаев узор, который наши точки данных образуют на плоскости  $XY$ , может не соответствовать образцу прямой линии. В этом случае простая прямолинейная или линейная регрессия в форме

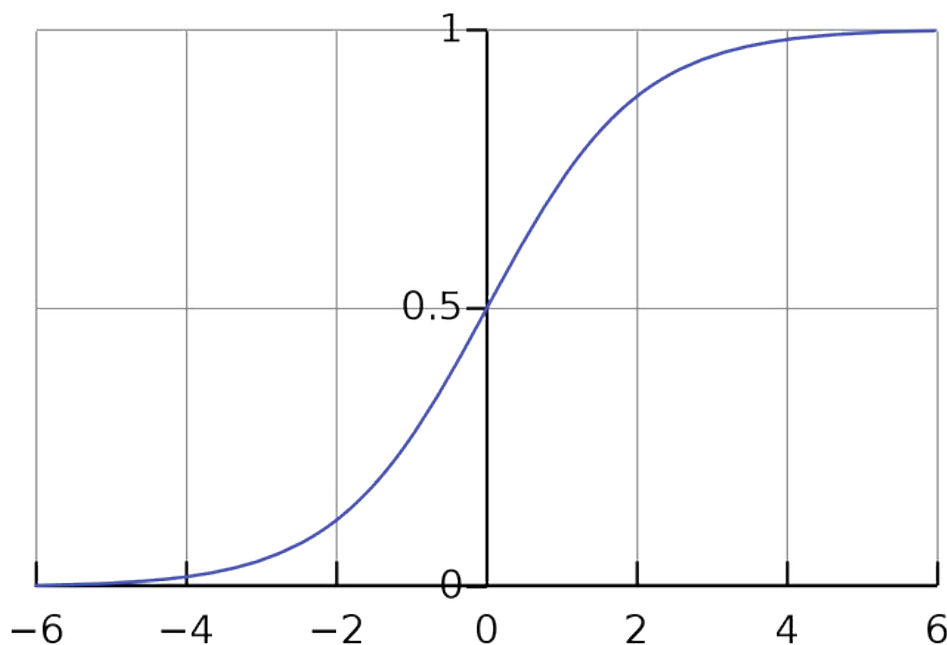
$$y = mx + b$$

была бы недостаточно хороша, чтобы предсказать значения  $X$ . Если мы заставим себя использовать линейную регрессию в подобных случаях, мы получим неподходящую модель.

В этом случае нам нужен инструмент для моделирования точек данных, который позволит нам изогнуть линию регрессии. Мы делаем это с помощью функций активации. Проще говоря, функции активации — это то, что делает нашу регрессию нелинейной. Ниже приведены некоторые из часто используемых функций активации.

### Функция активации сигмоида

График сигмовидной функции выглядит следующим образом.



Сигмовидная функция. Изображение взято с сайта wikipedia.com.

Уравнение S-образного графика нашей сигмовидной функции выглядит следующим образом.

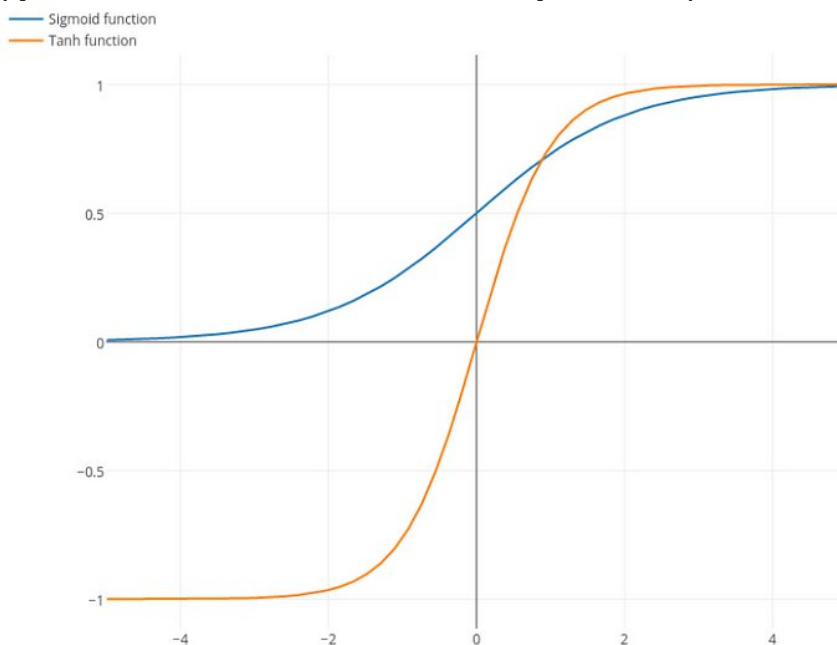
$$S(x) = \frac{1}{1 + e^{-x}}.$$

Поскольку сигмовидная функция принимает значения только от 0 до 1, ее обычно используют для прогнозирования вероятностей. Сигмоидальная функция также дифференцируема, что означает, что производную можно вычислить. Однако производная становится все ближе и ближе к 0 по мере приближения как к положительной, так и к отрицательной бесконечности. Маленькие значения градиента нежелательны для обратного распространения ошибки. Также логистическая сигмовидная функция может привести к зависанию нейронной сети во время обучения.

**С другой стороны, функция активации Softmax** представляет собой более обобщенную логистическую функцию активации для многоклассовой классификации. Это означает, что softmax можно использовать для решения задачи классификации, включающей два или более классов.

## Функция активации гиперболического тангенса

Гиперболическая функция активации тангенса, во многом похожа на сигмовидную функцию. Функция активации гиперболического тангенса также имеет сигмоидальную форму «S», но разница в том, что Тан находится в диапазоне от -1 до 1. График и уравнение для функции активации Тан, сравниваемые параллельно с сигмовидной функцией активации, выглядят следующим образом.



Сигмоида и Тан сравнивались бок о бок. <https://stats.stackexchange.com>.

Тан обычно применяется в задачах классификации между двумя классами. Как и сигмоидальная функция активации, функция tanh является дифференцируемой, но, как следствие, значение градиента также приближается к 0 по мере приближения к обоим бесконечным концам, что нежелательно для обратного распространения ошибки. Гиперболический тангенс удобен, когда приходится выбирать между сигмовидной и tanh. Уравнение для функции tanh выглядит следующим образом.



$$g(x) = \frac{e^x}{1 + e^x}$$

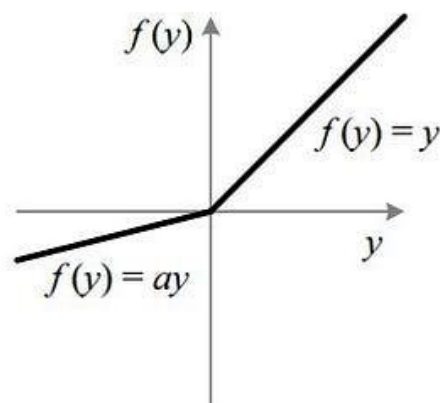
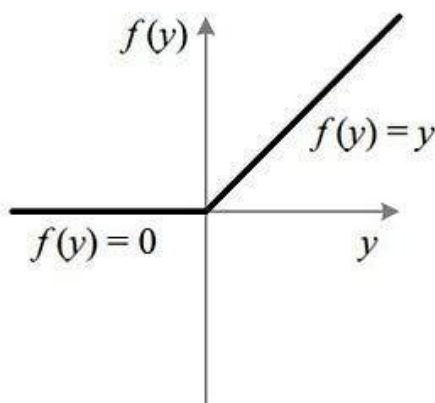
$$\tanh(x) = 2g(2x) - 1$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

## Функция активации ReLU и Leaky ReLU

ReLU означает «Выпрямленная линейная единица» и является наиболее часто используемой функцией активации в нейронных сетях. Функция активации ReLU находится в диапазоне от 0 до бесконечности, причем 0 для значений меньше или равных 0, то есть 0 для отрицательных значений. В целом функция активации ReLU более выгодна по сравнению с сигмовидной и Танговской. Однако он все равно не сможет правильно сопоставить негативные значения, поскольку сразу превращает их в 0.

Именно здесь на помощь приходит Leaky ReLU. Leaky ReLU существует как попытка решить серьезную проблему ReLU. Leaky ReLU не устанавливает отрицательные значения равными нулю, вместо этого он следует линейному уравнению с градиентом, обычно 0,01. Если градиент не равен 0,01, это называется рандомизированным ReLU.



## Перцептрон для двоичной классификации

Благодаря этому дискретному выходу, управляемому функцией активации, перцептрон можно использовать в качестве **модели двоичной классификации**, определяя **линейную границу решения**. Он находит разделяющую гиперплоскость, которая минимизирует расстояние между неправильно классифицированными точками и границей решения.

$$\underbrace{D(w, c)}_{\text{distance}} = - \sum_{\substack{i \in M \\ \text{misclassified observations}}} \overbrace{y_i}^{\text{output}} (x_i w_i + c)$$

Функция потерь перцептрона.

Чтобы минимизировать это расстояние, Перцептрон использует стохастический градиентный спуск в качестве функции оптимизации.

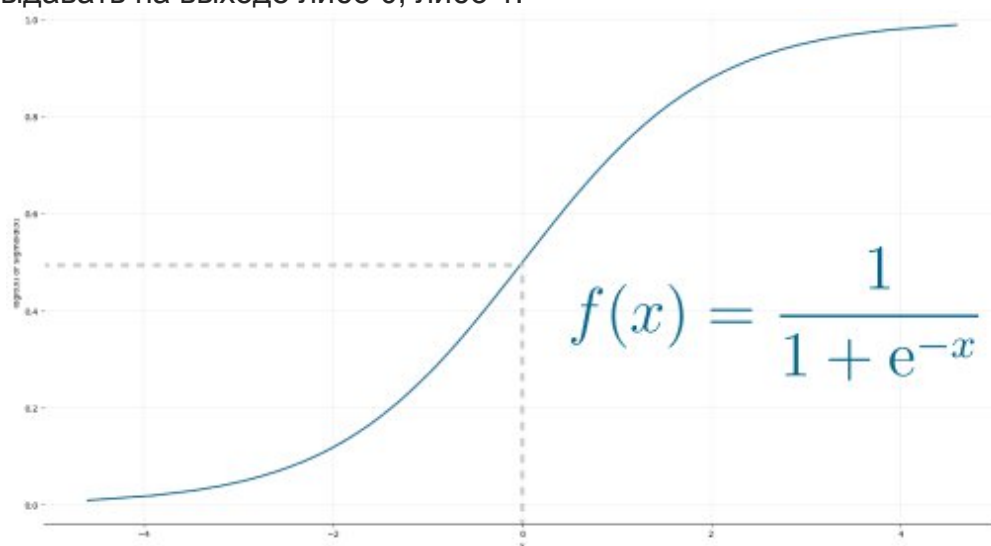
Если данные линейно разделимы, гарантировано, что стохастический градиентный спуск сходится за конечное число шагов.

Последняя часть, которая нужна Перцептрону — это **функция активации**, функция, которая определяет, сработает нейрон или нет.

Первоначальные модели перцептрона использовали сигмовидную функцию, и, просто взглянув на ее форму, это становится понятным!

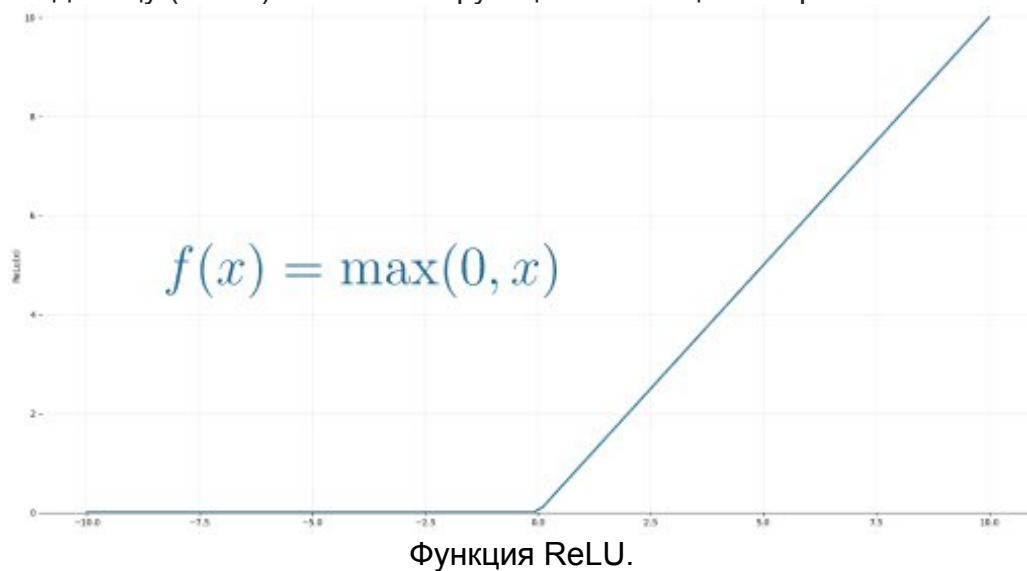
Сигмоидальная функция отображает любой реальный входной сигнал в значение, равное 0 или 1, и кодирует нелинейную функцию.

Нейрон может получать на вход отрицательные числа, но при этом он все равно сможет выдавать на выходе либо 0, либо 1.



Сигмовидная функция

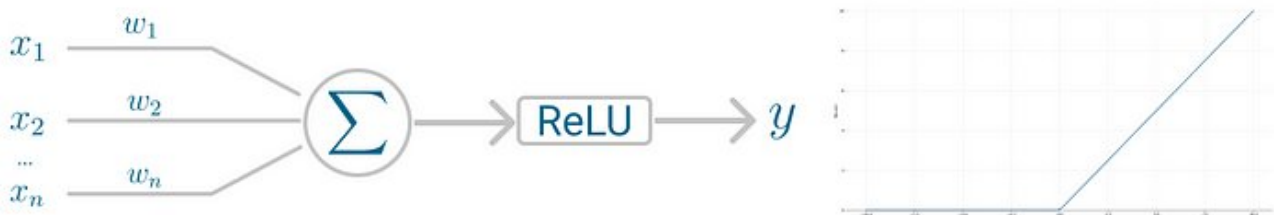
Но если вы посмотрите на статьи и алгоритмы глубокого обучения за последнее десятилетие, вы увидите, что большинство из них используют выпрямленную линейную единицу (ReLU) в качестве функции активации нейрона.



Причина, по которой ReLU стала более распространенной, заключается в том, что она обеспечивает лучшую оптимизацию с использованием стохастического градиентного спуска, более эффективные вычисления и является масштабно-инвариантной, то есть на ее характеристики не влияет масштаб входных данных.

## Собираем все это вместе

Нейрон получает входные данные и случайным образом выбирает начальный набор весов. Они объединяются во взвешенную сумму, а затем ReLU, функция активации, определяет значение вывода.



Модель нейрона перцептрона (слева) и функция активации (справа).

Но вам может быть интересно: разве *Персептрон на самом деле не изучает веса*?

Он делает! Персептрон использует стохастический градиентный спуск, чтобы найти или, можно сказать, *изучить* набор весов, который минимизирует расстояние между неправильно классифицированными точками и границей решения. Как только стохастический градиентный спуск сходится, набор данных разделяется на две области линейной гиперплоскостью.

Хотя было сказано, что персептрон может представлять любую схему и логику, самая большая критика заключалась в том, что он не мог представлять логический

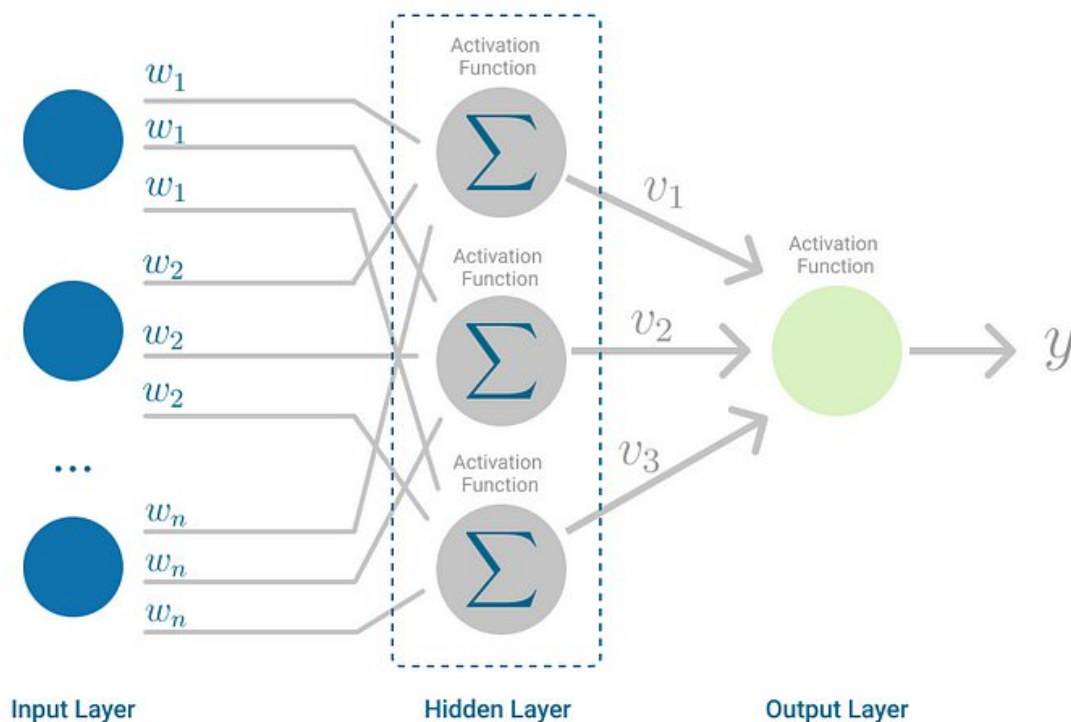
элемент исключающее *ИЛИ* , где модель возвращает 1 только в том случае, если входные данные различны.

Это было доказано почти десять лет спустя Мински и Папертом в 1969 году и подчеркивает тот факт, что перцептрон, имеющий только один нейрон, не может быть применен к нелинейным данным.

## Многослойный перцептрон

**Многослойный перцептрон** был разработан для устранения этого ограничения. Это нейронная сеть, в которой отображение входных и выходных данных нелинейно.

Многослойный перцептрон имеет входные и выходные слои, а также один или несколько **скрытых слоев** со множеством нейронов, сложенных вместе. И хотя в перцептроне нейрон должен иметь функцию активации, устанавливающую порог, например ReLU или сигмовидную функцию, нейроны в многослойном перцептроне могут использовать любую произвольную функцию активации.



Многослойный перцептрон. (Изображение автора)

Многослойный перцептрон подпадает под категорию алгоритмов прямой связи , поскольку входные данные объединяются с начальными весами во взвешенную сумму и подвергаются функции активации, как и в перцептроне. Но разница в том, что каждая линейная комбинация распространяется на следующий уровень.

Каждый уровень *передает* следующему результат своих вычислений, свое внутреннее представление данных. Это проходит через все скрытые слои к выходному слою.

Но это еще не все.

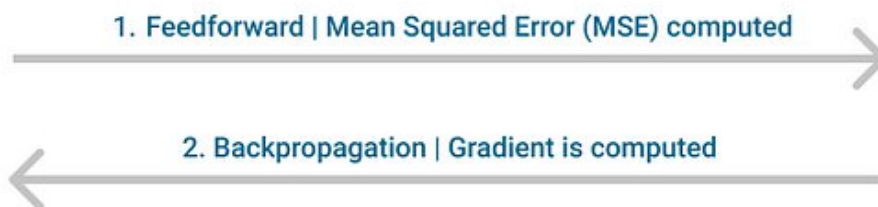
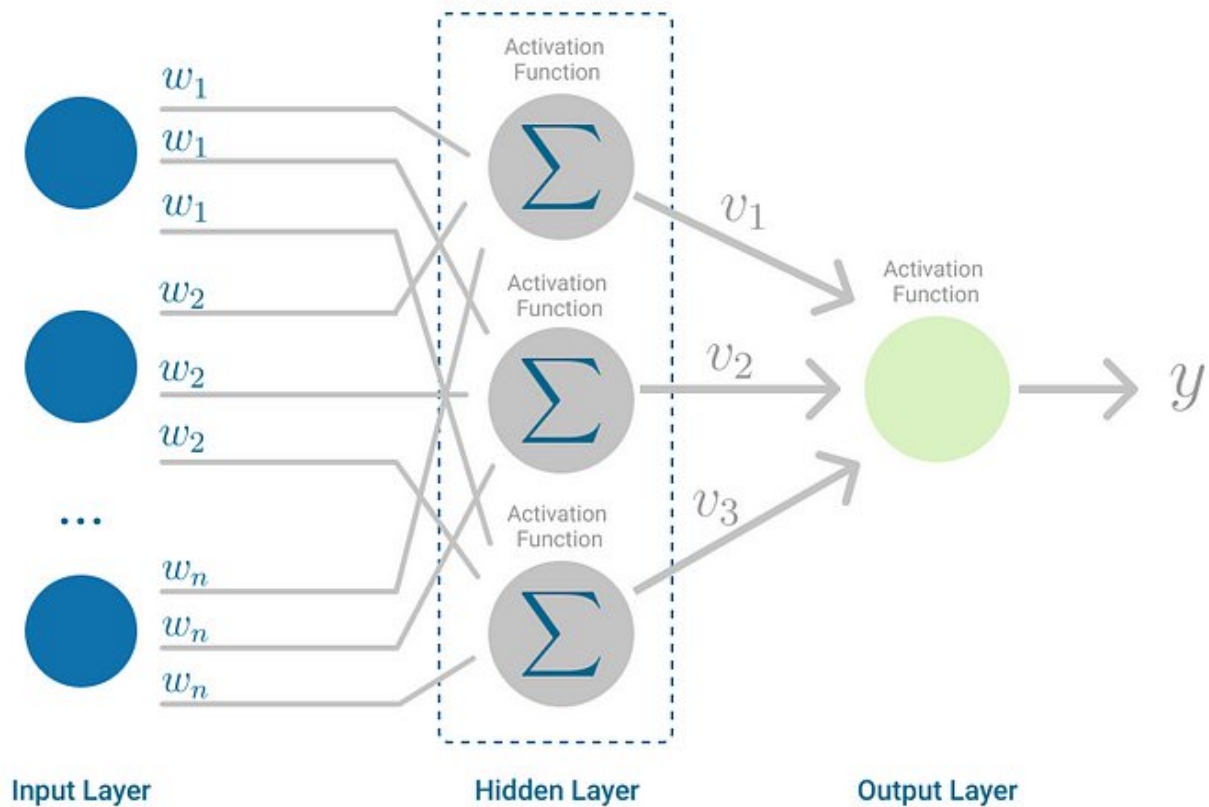
Если бы алгоритм только вычислял взвешенные суммы в каждом нейроне, распространял результаты на выходной слой и останавливался на этом, он не смог бы узнать *веса*, которые минимизируют функцию стоимости. Если бы алгоритм вычислял только одну итерацию, фактического обучения не было бы.

Именно здесь в игру вступает обратное распространение.

## Обратное распространение ошибки

Обратное распространение ошибки — это механизм обучения, который позволяет многослойному перцептрону итеративно корректировать веса в сети с целью минимизации функции стоимости.

Для правильной работы обратного распространения ошибки существует одно жесткое требование. Функция, которая объединяет входные данные и веса в нейроне, например, взвешенная сумма, и пороговая функция, например ReLU, должны быть дифференцируемыми. Эти функции должны иметь **ограниченную производную**, поскольку градиентный спуск обычно является функцией оптимизации, используемой в многослойном перцептроне.



Многослойный персептрон с выделением этапов прямого распространения и обратного распространения ошибки.

На каждой итерации, после того как взвешенные суммы передаются по всем слоям, градиент среднеквадратической **ошибки** вычисляется для всех пар входных и выходных данных. Затем, чтобы распространить его обратно, веса первого скрытого слоя обновляются значением градиента. Вот как веса передаются обратно в начальную точку нейронной сети!

$$\underbrace{\Delta_w(t)}_{\substack{\text{Gradient} \\ \text{Current Iteration}}} = \underbrace{-\varepsilon}_{\substack{\text{Bias} \\ \text{Error}}} \underbrace{\frac{dE}{dw(t)}}_{\substack{\text{Weight vector}}} + \underbrace{\alpha \Delta_w(t-1)}_{\substack{\text{Learning Rate} \\ \text{Gradient} \\ \text{Previous Iteration}}}$$

Одна итерация градиентного спуска.

Этот процесс продолжается до тех пор, пока градиент для каждой пары ввода-вывода не сойдется, что означает, что вновь вычисленный градиент не изменился больше, чем заданный *порог сходимости*, по сравнению с предыдущей итерацией.

## Прямое распространение:

Мы поговорили об обратном распространении ошибки, но также необходимо отметить что самый простейший алгоритм это прямое распространение ошибки. Прямое распространение - это процесс, при котором данные передаются через нейронную сеть, начиная с входного слоя и заканчивая выходным слоем, чтобы получить прогноз или выходное значение. В этом процессе каждый нейрон выполняет два основных действия:

**Линейная комбинация входов:** Нейрон суммирует взвешенные входные данные, умножая каждый вход на соответствующий вес и добавляя смещение. Это создает взвешенную сумму входов.

**Применение функции активации:** Результат линейной комбинации подвергается функции активации. Функция активации определяет, активируется ли нейрон (передает сигнал) или остается неактивным.

Процесс прямого распространения выполняется последовательно для всех нейронов и слоев, пока не достигнется выходной слой, который предоставляет окончательный результат или прогноз сети. Этот процесс позволяет сети принимать входные данные и генерировать соответствующие выходы.

## Прямое распространение и Прогнозирование в Нейронных Сетях

Прямое распространение - это процесс, при котором данные передаются через нейронную сеть, начиная с входного слоя и заканчивая выходным слоем, чтобы получить прогноз или выходное значение. Это ключевой этап в работе нейронных сетей для генерации предсказаний.

Процесс прямого распространения можно разбить на следующие шаги:

1. **Подача данных:** Входные данные подаются на входной слой нейронной сети. Эти данные могут представлять собой различные характеристики, изображения, текст или другую информацию, в зависимости от задачи.
2. **Линейная комбинация весов:** Каждый нейрон в каждом слое принимает входные данные и выполняет линейную комбинацию, умножая каждый вход на соответствующий вес и суммируя результаты. Это создает взвешенную сумму входов.
3. **Применение функции активации:** Результат линейной комбинации подвергается функции активации. Функция активации введена для внесения нелинейности в модель и определяет, активируется ли нейрон (передает сигнал) или остается неактивным.
4. **Передача к следующему слою:** Выход нейронов текущего слоя становится входом для нейронов следующего слоя. Процесс линейной комбинации и применения функции активации повторяется для каждого нейрона в каждом слое.
5. **Выходной слой и прогноз:** Процесс продолжается до тех пор, пока данные не достигнут выходного слоя. Выходной слой генерирует окончательный прогноз или значение в соответствии с задачей, которую решает нейронная сеть. Например, в задаче классификации, выходной слой может предсказать класс объекта, а в задаче регрессии - числовое значение.

Прогнозирование - это результат работы нейронной сети во время прямого распространения. Полученное значение на выходном слое представляет собой прогноз или оценку, сделанную моделью на основе входных данных. Нейронные сети используются для прогнозирования в различных областях, включая компьютерное зрение, обработку естественного языка, финансовый анализ, медицинские диагностики и многое другое.

Прямое распространение и прогнозирование - это ключевые шаги в работе нейронных сетей, и они позволяют сети использовать свои веса и параметры, чтобы сделать прогнозы на новых данных, обучаясь на предыдущих примерах.

## Как прямое распространение связано с обратным распространением?

Для обучения нейронная сеть использует как прямое, так и обратное распространение. Обратное распространение ошибки используется в машинном обучении и интеллектуальном анализе данных для повышения точности прогнозирования за счет производных, рассчитанных с помощью обратного распространения ошибки. Обратное распространение — это процесс перемещения справа (выходной слой) налево (входной слой). Прямое распространение — это способ перемещения данных слева (входной слой) вправо (выходной слой) в нейронной сети.

Нейронную сеть можно понимать как совокупность связанных узлов ввода/вывода. Точность узла выражается как функция потерь или частота ошибок. Обратное распространение ошибки вычисляет наклон функции потерь других весов в нейронной сети.



# Заключение

В заключение нашей лекции, мы рассмотрели важные аспекты перцептрона и многослойного перцептрона, которые представляют собой фундаментальные блоки в мире искусственных нейронных сетей. Перцептрон, с его способностью к бинарной классификации, является стартовой точкой для понимания работы нейронных сетей. Однако, многослойный перцептрон, с несколькими скрытыми слоями, демонстрирует силу и гибкость глубокого обучения. Понимание перцептрона и многослойного перцептрона - это только начало вашего увлекательного путешествия в эту захватывающую область, где вы сможете применить эти знания для решения различных задач и разработки инновационных приложений, а также понимание более сложных архитектур нейронных сетей!