**Программирование глубоких нейронных сетей на Python**

**Введение в тематику искусственных нейронных сетей**

Цель: изучить факторы, которые необходимы для применения глубоких нейронных сетей, и определить круг задач, решаемых глубокими нейронными сетями

План:

* + 1. Понятие «глубокие нейронные сети» и их преимущества
    2. Задачи, решаемые глубокими нейронными сетями
    3. Факторы, которые привели к возможности практической реализации глубоких нейронных сетей

**Понятие «глубокие нейронные сети» и их преимущества**

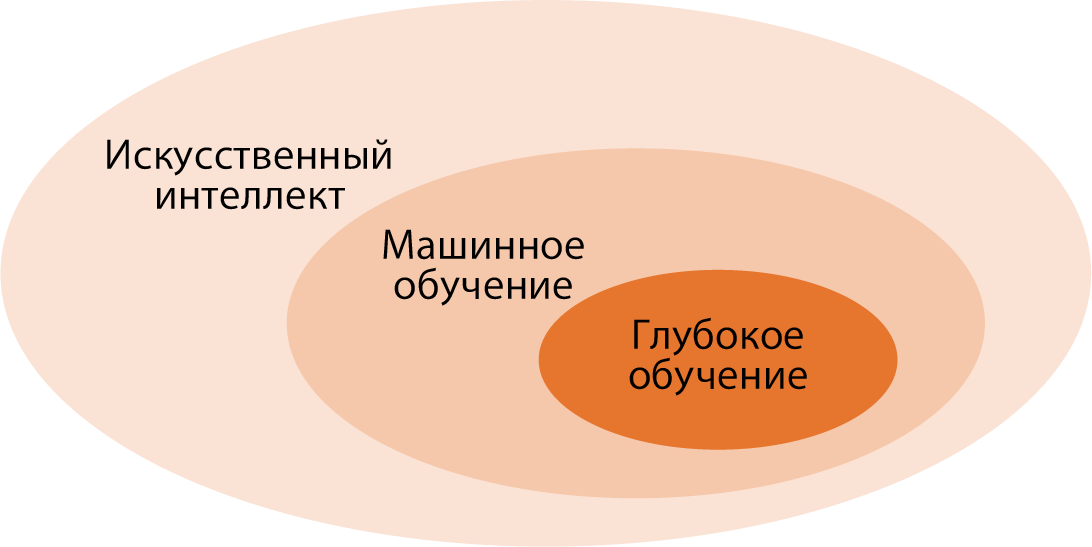


Рис. 1. Связь понятий «Искусственный интеллект», «Машинное» и «Глубокое обучение»

Сфера ИИ (искусственный интеллект) – это автоматизация задач, ранее решаемых лишь человеческим интеллектом. Это огромный спектр разнородных задач включает в себя очень разнообразные методы:

* + машинное обучение;
  + глубокое обучение;
  + генетические алгоритмы;
  + множество подходов, не связанных на прямую с обучением.

Система машинного обучения, как следует из названия, ***обучается***, *но* ***не* *программируется*** в явном виде. На вход она принимает многочисленные и часто разнородные примеры, имеющие отношение к решаемой задаче, а она ищет во входных примерах некую статистическую структуру. В свою очередь, данная структура позволяет алгоритму определить правила для автоматического решения поставленной задачи.

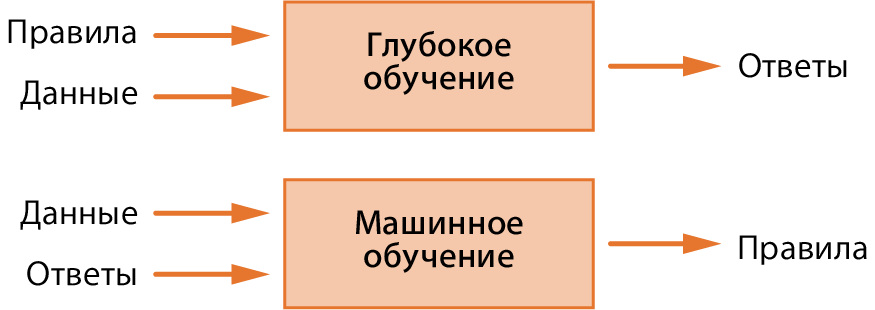


Рис. 2. Разница между классическими алгоритмами и машинным обучением

Модель машинного обучения преобразовывает данные на входе в значимые результаты, «обучаясь» на данных ей примерах входных данных и результатов. Поэтому главная задача машинного и глубокого обучения – значимое преобразование исходных данных или, говоря другими словами, обучение представлению исходных данных, максимально приближающему нас к ожидаемому результату.

***Глубокое обучение*** – это особый раздел машинного обучения: другой подход к поиску представления данных. Он делает упор на анализ последовательных слоев (или уровней) все более значимых представлений. Под глубиной в глубоком обучении подразумевается не более глубокое понимание, которое достигается этим подходом. Идея заключается в многослойном представлении модели. Количество слоев, из которых состоит модель данных, называют глубиной модели. Иными подходящими названиями для этой части машинного обучения могли бы быть: многослойное обучение и иерархическое обучение.

В глубоком обучении многослойные представления изучаются (чаще всего) с применением моделей, называемых нейронными сетями. Их структура представлена в виде слоев, наложенных друг на друга. Понятие нейронной сети заимствовано из нейробиологии. Хотя источником некоторых основополагающих идей глубокого обучения частично являются науки о мозге, модель глубокого обучения не являются моделью мозга человека. Нет фактов, доказывающих, что мозг работает по принципам, подобным механизмам, которые используются в современных моделях глубокого обучения.

Как могут выглядеть представления, полученные алгоритмом глубокого обучения? Их можно представить так, как показано на рисунке 3.

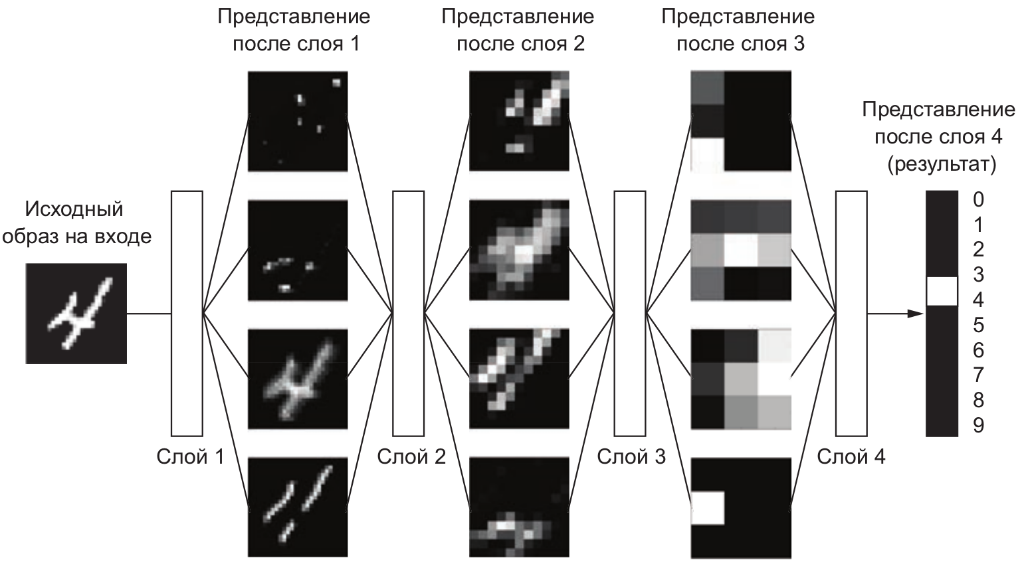


Рис. 3. Глубокая нейронная сеть для классификации цифр

С каждым новым слоем глубокая нейронная сеть уходит все дальше от начального изображения (представленного матрицей размером  пикселей, состоящей из цифр от 0 до 255). Сеть абстрагируется все больше и больше, пока на последнем (выходном) слое не выдаст цифру, которую мы видим на изображении.

**Задачи, решаемые глубокими нейронными сетями**

Глубокое обучение достигло множества прорывов в сложных для машинного обучения областях:

* + классификация изображений на уровне человека;
  + распознавание речи на уровне человека;
  + распознавание рукописного текста на уровне человека;
  + повышение качества машинного перевода с одного языка на другой;
  + улучшение качества чтения текста вслух машиной;
  + создание цифровых помощников: Yandex Алиса, Google Now и Amazon Alexa;
  + управление автомобилем на уровне, сравнимом с человеком;
  + повышение точности целевой рекламы Google, Baidu и Bing;
  + повышение релевантности поиска в Интернете;
  + машины могут отвечать на вопросы, заданные вслух;
  + алгоритм обыграл человека в «Го».

Глубокие нейронные сети автоматизировали одну из сложных задач машинного обучения – выбор признаков, значимых для решения задачи, из множества доступных. Глубокие нейронные сети автоматически извлекают важные признаки в процессе обучения. Но за эту возможность приходится «расплачиваться» очень большими вычислительными ресурсами. Справедливости ради стоит отметить, что это касается лишь обучения нейронной сети. Обученная нейронная сеть работает быстро и требует очень мало ресурсов (способна работать даже на мобильном устройстве). К тому же нейронная сеть дает еще одно преимущество перед другими алгоритмами машинного обучения – она может дообучаться на отдельных примерах без полного переобучения на всех данных.

Методика глубокого обучения имеет две важные характеристики:

* 1. Поэтапно, послойно конструирует все более сложные представления.
  2. Исследует промежуточные представления совместно, за счет чего каждый слой обновляется в соответствии с информацией, полученной от представлений других слоев.

**Факторы, которые привели к возможности практической реализации глубоких нейронных сетей**

Нужно отметить, что машинное обучение и нейронные сети пережили несколько исторических этапов, когда их популярность то возрастала, то убывала (в связи с невозможностью решить некий класс задач).

Различные подходы к машинному обучению и нейронным сетям можно распределить в хронологическом порядке следующим образом:

* + вероятностное моделирование (позволило, к примеру, решить задачу определения спама);
  + полносвязные нейронные сети (сеть LeNet смогла распознать рукописные цифры);
  + ядерные методы (в частности метод опорных векторов);
  + деревья решений и основанные на них случайные леса и градиентный бустинг (долгое время оставляли нейронные сети на втором месте);
  + современные глубокие нейронные сети Inception, ResNet, BERT (вышли в задачах классификации изображений и текстовой информации на уровень человека).

Две ключевые идеи глубокого обучения для решения задач распознавания образов были хорошо известны уже в 1989 году – это сверхточные нейронные сети и алгоритм обратного распространения ошибки. Алгоритм долгой краткосрочной памяти (Long Short-Term Memory, LSTM), который составляет основу глубокого обучения для прогнозирования временных рядов и анализа текстовых последовательностей, был предложен в 1997 году и с тех пор претерпел мало изменений. Тогда почему глубокое обучение стало применяться лишь с 2012 года? Что изменилось за эти два десятилетия?

В целом машинное обучение держится на трех «китах»:

* + оборудование;
  + наборы данных и тесты;
  + алгоритмические достижения.

**Оборудование**

В 2007 году компания NVIDIA выпустила CUDA – программный интерфейс для производимых ими GPU. Теперь несколько GPU могут заменить мощные кластеры на обычных процессорах в ряде задач, которые допускают возможность массового распараллеливания расчетов. Глубокие нейронные сети выполняют (в основном) умножение множества маленьких матриц, поэтому допускают высокую степень распараллеливания. В 2016 году на своей ежегодной конференции Google I/O компания Google презентовала свой проект тензорного процессора (Tensor Processing Unit, TPU). Данный процессор с новой архитектурой предназначается для использования в глубоких нейронных сетях. Он в 10 раз производительнее и энергоэффективнее, чем самые лучшие модели GPU.

**Наборы данных и тесты**

Данные – это своего рода «уголь», «сырье», питающее интеллектуальные машины и глубокое обучение в частности. А экспоненциальный рост емкости устройств хранения информации, наблюдавшийся в последние 20 лет (согласно закону Мура), и перемены в игровом мире вызвали бурный рост интернет-данных, благодаря чему удалось накопить и распространить огромные объемы данных для машинного обучения (нужно отметить, что глубокие нейронные сети требуют сотни тысяч, а иногда и миллионы, обучающих примеров). Сейчас крупные компании работают с наборами изображений, видео и текстовых материалов, которые невозможно было бы собрать без Интернета.

**Алгоритмы**

Кроме оборудования и данных, до конца 2000-х нейронным сетям не хватало эффективного способа обучения по-настоящему глубоких нейронных сетей. Нейронные сети оставались не очень глубокими – имели один или два слоя представления, из-за этого они уступали более совершенным поверхностным методам, таким как метод опорных векторов и случайные леса. Основная проблема заключалась в распространении градиента через глубокие пакеты слоев. Сигнал обратной связи, который используют для обучения нейронных сетей, затухает по мере увеличения количества слоев.

Все изменилось в 2009–2010 годах, когда появились простые, но важные алгоритмические усовершенствования, позволившие улучшить распространение градиента:

* + усовершенствованные функции активации;
  + более оптимальные схемы инициализации весов, начиная с предварительного послойного обучения, от которого быстро отказались;
  + лучшие схемы оптимизации, такие как RMSProp и Adam.

Эти улучшения позволили строить модели с 10 слоями и более. Начался расцвет глубокого обучения. Наконец, в 2014, 2015 и 2016 годах были открыты еще более совершенные способы распространения градиента:

* + пакетная нормализация;
  + обходные связи;
  + отделимые свертки.

В настоящее время можно обучать с нуля модели с тысячами слоев в глубину.

**Введение в тематику искусственных нейронных сетей**

Цель: изучить модель искусственного нейрона, подробно рассмотреть его составляющие, получить общее представление о типах нейронных сетей

План:

* 1. Модель искусственного нейрона
  2. Функции активации и смещение
  3. Классификация нейронных сетей в разрезе распространения сигнала и глубины нейронной сети

**Модель искусственного нейрона**

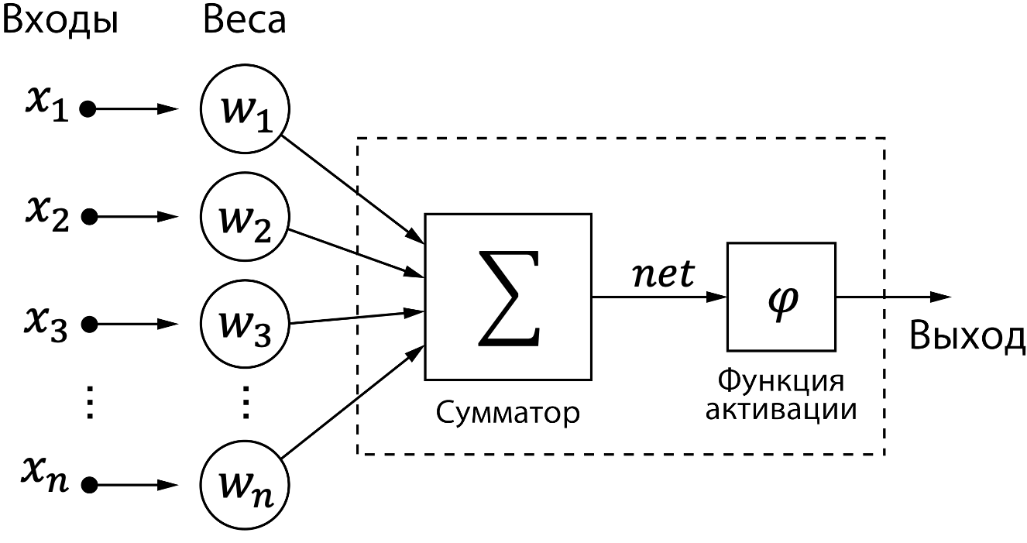


Рис. 1. Модель искусственного нейрона Мак-Каллока и Питса

В математическом выражении все очень просто:

* + Входы – это некий вектор  входных значений.  Являются аналогом дендритов в реальном нейроне.
  + Веса – это вектор обучаемых параметров нейронной сети , которые обозначают значимость каждого входного признака для итогового результата (по сути, аналог синапсов реального нейрона). В конечном итоге, если какой-то вес , то признак  является абсолютно незначимым для нейронной сети. И наоборот, чем больше значение , тем сильнее влияет признак  на результат, выданный нейронной сетью. Длина вектора весов совпадает с длиной вектора входных параметров.
  + Сумматор – функция, которая просто суммирует произведения признаков на их веса. Можно сказать, что это ядро нейрона, которое аккумулирует заряд.
  + Функция активации – функция, которая принимает на вход результат сумматора и выполняет некое преобразование, чтобы превратить сумму взвешенных входов в адекватный выход, который можно интерпретировать с точки зрения решения поставленной задачи. Так как в случае малой величины сумматорной функции функция активации может вернуть 0, то есть «ничего», то она является неким аналогом того механизма в реальном нейроне, которые отвечает за возбуждения нейрона (его «активацию»).

**Функции активации**

Итак, предположим, что нам нужно выполнить классификацию объектов на две группы: съедобные грибы и несъедобные. Взвешенная сумма входов может быть любым числом в диапазоне от минус бесконечности до плюс бесконечности, что никак не приближает нас к решению поставленной задачи. Значит необходимо как-то преобразовать это число так, чтобы оно обязательно попадало в строго заданный нами диапазон, и уже в рамках этого диапазона мы могли принимать решение. Для этого были разработаны функции активации.

*Функция единичного скачка*(*она же функция Хэвисайда*)

где  – взвешенная сумма входных параметров.

Данная функция принимает на вход взвешенную сумму входных параметров, и если они меньше 0, то возвращает 0, иначе – 1.

Но что если мы видим, что корректное разделение классов будет происходить не по нулю, а по любому другому значению?

Тут пригодится понятие «порог» (от англ. *bias*) – это значение, которое будет смещать место скачка от 0 в нужное нам место числовой оси

где  – заданный порог,  – взвешенная сумма входных параметров.

**Функция единичного скачка**

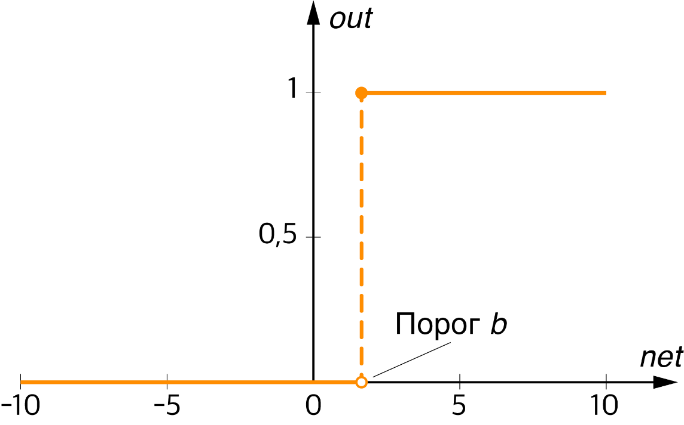


Рис. 2. Функция Хэвисайда (функция единичного скачка)

Но такое строгое разделение на 0 и 1 является слишком грубым и в реальности не используется, так как часто возникают ситуации, когда о принадлежности к классу можно сказать лишь с некой долей вероятности. Поэтому нужна функция, которая позволит выдавать не 0 и 1, а число в этом диапазоне. И чем ближе число к 0, тем больше вероятность, что это, к примеру, несъедобный гриб, а чем ближе к 1, тем больше вероятность, что гриб съедобен. И такая функция была придумана. Рассмотрим ее далее.

*Сигмоидальная функция (она же логистическая функция)*

где  – взвешенная сумма входных параметров.

Кроме того, что она возвращает необходимое нам распределение значений от 0 до 1, данная функция обладает еще одним интересным свойством – она «прижимает» распределения получаемых значений к 0 и к 1, а в центре около 0,5 старается оставить как можно меньше значений. Это происходит за счет очень резкого возрастания функции на участке вокруг 0,5 и плавных переходов около 0 и 1. Таким образом мы получаем минимальное количество значений, которые попадают в зону неопределенности около 0,5.

**Логическая функция**

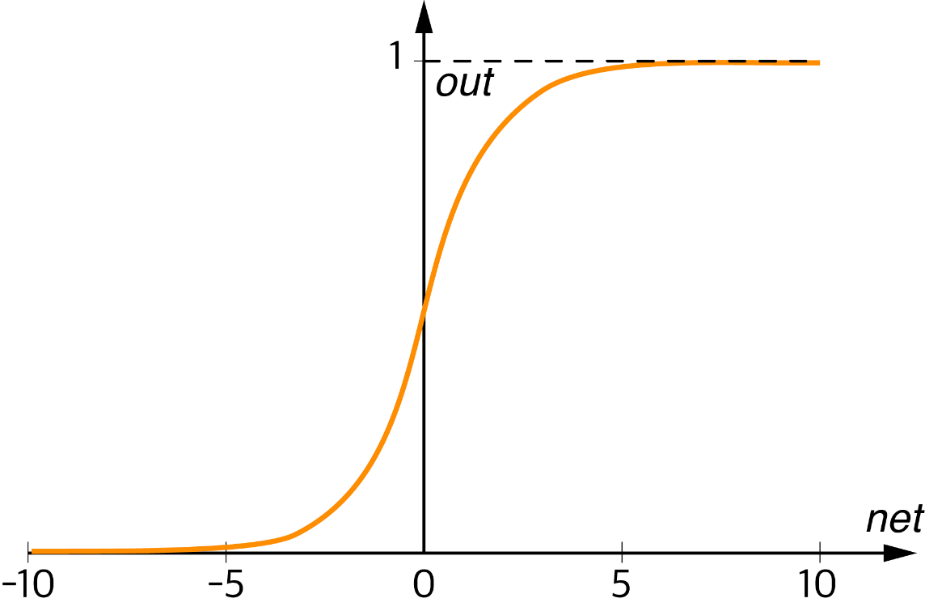


Рис. 3. Сигмоидальная функция (логистическая функция)

*Функция гиперболического тангенса*

Данная функция ведет себя похожим образом с сигмоидой и обладает всеми ее свойствами, но возвращает значения в диапазоне от -1 до 1. Иногда это может быть удобно.

**Гиперболический тангенс**

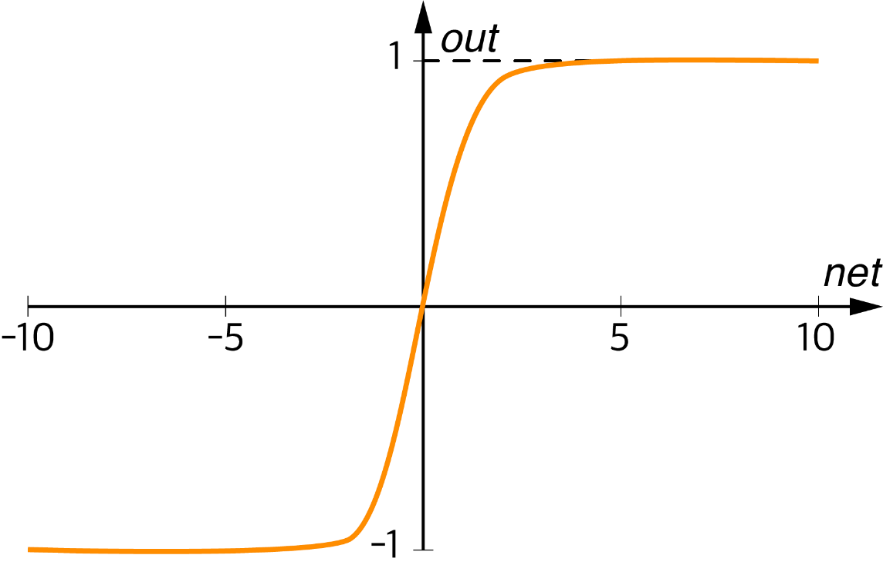


Рис. 4. Функция гиперболического тангенса

*ReLU (rectified linear units)*

Эта функция активации возвращает значение , если х положительно, и 0 в противном случае.

Является хорошим аппроксиматором – любая функция может быть аппроксимирована комбинацией *ReLU.*

Преимуществом также является то, что эта функция создает разреженность при активации нейронов. То есть, в отличие от сигмоиды и гиперболического тангенса, будут активироваться не все нейроны, что уменьшит вычислительную трудоемкость. Это свойство вытекает из того факта, что все отрицательные значения обращаются в ноль. Использование *ReLU* значимо повышает скорость сходимости градиентного спуска по сравнению с сигмоидой и гиперболическим тангенсом.

Но есть и недостатки. Все из-за того же обнуления всех отрицательных значений градиент на этой части также равен 0. Поэтому веса не будут корректироваться во время градиентного спуска. А значит нейроны, пребывающие в таком состоянии, не будут реагировать на ошибки и обучаться. Данное явление также называют проблемой умирающего *ReLU.*

Существуют модификации *ReLU*, которые частично решают эту проблему: *Leaky ReLU*, Parametric *ReLU*, *Randomized ReLU*.

**Классификация нейронных сетей в разрезе распространения сигнала и глубины нейронной сети**

По типу распространения сигнала нейронные сети можно разделить на:

* + нейронные сети с прямым распространением сигнала;
  + нейронные сети с наличием по крайней мере одной обратной связи (рекуррентные нейронные сети).

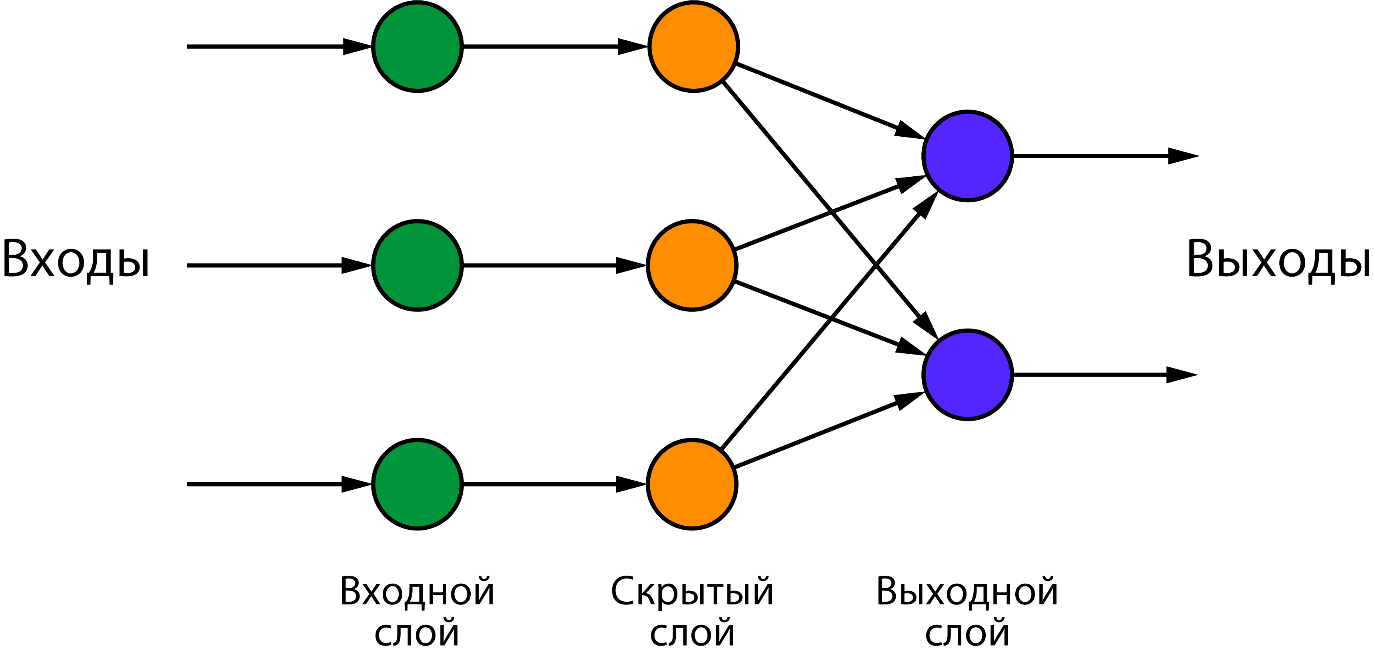


Рис. 5. Нейронная сеть с прямым распространением сигнала

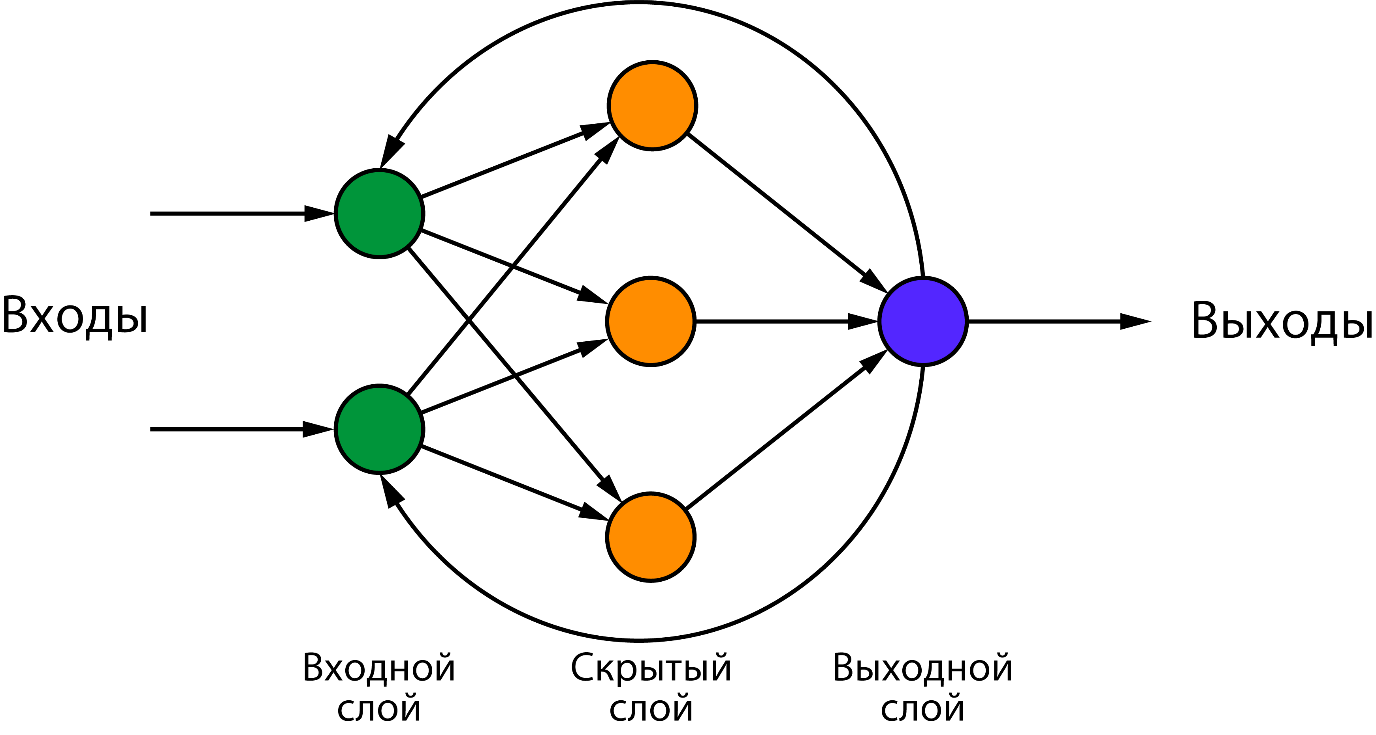


Рис. 6. Нейронная сеть с обратной связью (рекуррентная нейронная сеть)

Сети с прямым распространением сигнала решают множество задач классификации и регрессии на табличных данных, практически все задачи на изображениях и часть задач обработки естественного языка.

Необходимость в рекуррентных нейронных сетях возникает при решении задач, в которых требуется помнить информацию о предыдущих элементах обучающей последовательности: временные ряды, обработка текста с учетом последовательности слов.

Также архитектуры нейронных сетей можно разделить на однослойные и многослойные (они же глубокие) нейронные сети.

Однослойной считается нейронная сеть, имеющая лишь входной и выходной слой, без наличия скрытых слоев. В такой сети сигналы попадают сразу с входного слоя на выходной. Сеть подобного рода способна настроиться не под все сложные зависимости, очень чувствительна к предварительной обработке признаков и требует их тщательного отбора. Преимуществом является достаточно быстрая скорость обучения.

Многослойной (глубокой) нейронной сетью считается сеть с, как минимум, одним скрытым слоем (входной и выходной, естественно, также остаются). Способна самостоятельно от слоя к слою преобразовывать входные признаки и проводить их отбор за счет уменьшения/увеличения весов. Но наличие большого количества слоев, а значит и обучаемых параметров, также ведет к увеличению времени обучения (некоторые нейронные сети обучались по несколько месяцев). А еще из-за очень сложной структуры данный тип сетей подвержен переобучению при малом количестве обучающих данных. То есть нейронная сеть идеально подстраивается под обучающий набор, но в дальнейшем, на реальных данных, показывает очень плохое качество.

**Библиотеки для обучения нейронных сетей**

Цель: познакомиться с базовыми понятиями обучения нейронной сети, узнать основные типы обучения, подходы к организации обучения, познакомиться с самыми распространенными библиотеками для обучения нейронных сетей

План:

* 1. Обучение нейронной сети
  2. Библиотеки для обучения нейронной сети
  3. Стек технологий, который будет использован в курсе

**Обучение нейронной сети**

Итак, нейронная сеть представляет собой некий набор: входных сигналов; весов, которые как-то учитывают силу влияния входных сигналов; нейронов, которые преобразовывают входные сигналы и передают их следующим нейронам или на выходной слой. Как было сказано ранее, функций, которые преобразовывают сигнал (функций активации), существует множество и выбирать их необходимо исходя из задачи. Также можно создавать свои функции активации. Количество нейронов в слое и слоев в нейронной сети можно варьировать как угодно. Из чего следует, что нейронная сеть – достаточно гибкий инструмент.

Но основная сила нейронной сети не в этом. Даже при всех вышеописанных плюсах, если бы веса нейронной сети устанавливались один раз в начале обучения (как это происходит с количеством слоев и нейронов в слое, а также с функциями активации), то был бы получен крайне ограниченный инструмент, неспособный найти обобщенное решение для множества разнородных наблюдений. В лучшем случае нейронная сеть запомнит правильный ответ на одно из наблюдений (к примеру, запомнит, что на этой конкретной фотографии кошка), но уже следующее наблюдение, с немного измененными параметрами, заставит нейронную сеть ошибиться.

Сила нейронных сетей в обучении. В обучении весов связей между нейронами. В ходе множества итераций нейронная сеть сравнивает свой выход с настоящим ответом и, в случае ошибки, корректирует все свои веса таким образом, чтобы на следующей итерации добиться верного ответа. Алгоритм такого обучения называется «алгоритм обратного распространения ошибки». Это связано с тем, что нейронная сеть понимает, ошиблась она или нет, только на выходе, и затем информацию об ошибке она распространяет в обратном порядке от выхода ко входу.

Чем больше в нейронной сети слоев и нейронов, тем больше связей (а значит и весов, эти связи характеризующих) и тем более тонкой может быть настройка нейронной сети при обучении. Но также и выше опасность подкрутить веса так, что они идеально будут решать задачу на обучающих данных, но будут выдавать абсолютно неверный ответ на новых данных из реального мира. В этом случае говорят, что нейронная сеть переобучилась.

Но учиться можно по-разному. Рассмотрим варианты.

**1. Обучение с учителем**

Это достаточно привычный нам способ, когда кто-то нас проверяет. В случае с нейронной сетью создаются специальные учебные наборы данных. В них парами идут входные данные и ответ (см. табл. 1). Наборы формируются людьми (это может быть сам исследователь или нанятые им сторонние разметчики данных).

Таблица 1

Примеры формирования тренировочного набора для обучения с учителем

|  |  |  |
| --- | --- | --- |
| **Прикладная сфера** | **Входные данные** | **Верный ответ** |
| Классификация изображений | Матрицы изображений рукописных цифр (одинаковой размерности) | Число от 0 до 9 – цифра, которая реально написана на изображении |
| Классификация текстов | Вектор, полученный из текста новостной ленты | Тема новостей: спорт, политика, финансы и т. д. |
| Регрессия | Вектор с параметрами дома: площадь, количество комнат, удаленность от центра и т. д. | Число, обозначающее реальную стоимость дома |
| Временной ряд | Вектор с последовательным почасовым трафиком посетителей на сайте | Число (или вектор чисел), обозначающее число посетителей в следующий момент времени (ряд последовательных моментов времени) |

Таким образом нейронная сеть получает возможность исправлять свои ошибки, как прилежный ученик, опираясь на авторитетное мнение учителя. Данный класс задач наиболее изучен и дает наиболее качественные результаты.

Сразу же хочется отметить, что из данных таблицы 1 следуют два основных класса задач обучения с учителем:

* + классификация – множество ответов нейронной сети конечно, представляет собой некий конечный набор классов;
  + регрессия и аппроксимация – множество ответов нейронной сети бесконечно, представляет собой действительное число или вектор действительных чисел.

Временной ряд можно отнести к задаче регрессии, но с условием, что выборка наблюдений берется не случайно, а строго упорядоченно во времени и с одинаковым интервалом. К примеру цена на конкретный дом фиксировалась на протяжении 10 лет с интервалом в 1 год и нужно предсказать цену данного дома на одиннадцатый год.

**2. Обучение без учителя**

Как понятно из названия, тут уже нейронная сеть должна «посмотреть» на данные и сделать какие-то свои выводы, попытаться найти какие-то шаблоны, которые выделяют те или иные части набора данных. Тут заранее стоит предупредить, что шаблоны, найденные нейронной сетью, далеко не всегда могут быть интерпретированы в человеческих терминах, то есть не всегда ясно, по какому признаку нейронная сеть разделила те или иные группы, или вывела какие-то правила. К задачам обучения без учителя относятся:

* + кластеризация – задача разбиения выборки на *N* (число кластеров может задаваться заранее, а может вычисляться алгоритмом в процессе обучения) кластеров по заранее неведомым признакам. Отличие от классификации в том, что метки классов заранее не заданы и полученные кластеры необходимо затем интерпретировать в язык человеческих терминов;
  + сокращения размерности – задача представления многомерного исходного вектора в вектор меньшей размерности с минимальной потерей информации. Таким образом может происходить отсев малоинформативных признаков, что уменьшит шум в данных и увеличит скорость обработки данных алгоритмом, а также может повысить качество какой-нибудь последующей задачи (например, классификации). Вырожденные случаем сокращения размерности (до двумерных или трехмерных векторов) можно использовать для визуализации данных.

**3. Обучение с подкреплением**

Это направление в машинном обучении, при котором модель не имеет информации о системе, но при этом может производить некие действия, влияющие на систему. При воздействии модели на систему та переходит в новое состояние. В зависимости от того, приближает это модель к желаемой цели или отдаляет от нее, система посылает модели положительное или отрицательное вознаграждение.

Например, есть сеть улиц, по которым едет машина. Модель может управлять скоростью машины и ее поворотами в заданных пределах. Целевую функцию можно сформулировать, как «доехать из пункта A в пункт B за минимальное время», а к тому же можно еще наложить ограничение, запрещающее врезаться в стены. Тогда модель будет управлять машиной, а система будет ее штрафовать за удар о стену или поощрять за оптимально выбранный маршрут.

Еще стоит отметить, что бывает несколько подходов к реализации обучения:

* + полное обучение (пакетный метод) – подход, при котором все обучающие данные подаются одним единым блоком. Преимуществом является то, что происходит значительная экономия времени на обучение, но есть высокая вероятность, что пострадает точность модели;
  + онлайн-обучение (стохастическое обучение) – подход, при котором обучающие примеры подаются по одному и после каждого из них происходит процесс обратного распространения ошибки. Метод более затратный по времени и ресурсам. Есть риск попасть в локальный минимум. Также модель может «забыть» информацию о более ранних примерах;
  + обучение на мини-выборках (мини-пакетах) – попытка найти золотую середину между двумя предыдущими вариантами. За счет того, что мини-выборки формируются случайным образом и усредняют ошибку, по всей подвыборке снижается риск попасть в локальный минимум. Метод быстрее онлайн-обучения. Размер мини-выборки можно регулировать исходя из технических возможностей машины, а также подбирать экспериментально для схождения к лучшему результату.

**Библиотеки для обучения нейронной сети**

Реализовывать все это многообразие методов и архитектур нейронных сетей с нуля очень трудоемко и нерационально. Поэтому были созданы библиотеки, в которых все основные архитектуры и методы реализованы и оптимизированы под параллельные вычисления. Таким образом создание нейронной сети превращается из сложной задачи низкоуровневого программирования и оптимизации в задачу непосредственно экспериментирования с нейронными сетями на высоком уровне абстракции. На уровне слоев и готовых функций активации, оптимизации и обратного распространения ошибки. Исследователь может полностью сосредоточиться на сути нейронных сетей и изучать их поведение при разных конфигурациях. Необходимость опускать на уровень ниже может возникать лишь при опытах с новыми экспериментальными функциями активации, конфигурациями соединения нейронов в слое или слоев между собой. Но начинающий исследователь может абстрагироваться от сложной математики и низкоуровневых программных реализаций, постепенно погружаясь в мир нейронных сетей.

Рассмотрим самые распространенные библиотеки для обучения нейронных сетей (см. табл. 2).

Таблица 2

Библиотеки для обучения нейронных сетей

|  |  |
| --- | --- |
| **Название библиотеки** | **Описание** |
| **TensorFlow** | Базовый язык С++, но имеет API для Python. Разработан Google. Вычисления с использованием графов потоков данных. Предлагает мощные средства мониторинга процесса обучения моделей и визуализации. Поддерживает распределенное обучение. Имеет достаточно высокий входной порог |
| **Theano** | Базовый язык Python. Разработан университетом Монреаля. Вычисления с использованием графов потоков данных. Эффективная обработка тензоров. Вычисления выражаются [NumPy](https://ru.wikipedia.org/wiki/NumPy" \t "_blank) – подобным синтаксисом.  Имеет высокий входной порог |
| **PyTorch** | Базовый язык Lua. Поддерживает API для Lua, Python, Java, C++. Разработан Ронаном Коллабертом. Имеет множество модульных элементов, которые легко комбинировать. Легко писать собственные типы слоев и работать на GPU.  Имеет API разных уровней абстракции |
| **CNTK** | Базовый язык С++. Поддерживает API для С++, С#, Python, Java. Разработана Microsoft. Обеспечивает скорость обучения, сравнимую с TensorFlow, а на рекуррентных сетях превосходит его. Имеет API разных уровней абстракции |
| **Caffe** | Базовый язык С++. Поддерживает API для С++, Python. Разработан в центре компьютерного зрения и обучения Беркли. Имеет небольшую скорость относительно других библиотек |
| **Keras** | Библиотека верхнего уровня. Поддерживает Python. В качестве вычислительного back-end использует TensorFlow или Theano. Позволяет создавать и обучать нейронные сети на очень высоком уровне абстракции. Имеет низкий порог вхождения |

Стоит отметить, что для решения задач машинного обучения также существуют очень мощные и удобные инструменты: Scikit-learn, Xgboost, LightGBM, CatBoost. Мы не коснемся их в курсе, но стоит знать хотя бы о их существовании, так как ряд задач с разнородными табличными данными до сих пор решается лучше именно с помощью них, а не нейронных сетей.

**Стек технологий, который будет использован в курсе**

Как же выбрать нужное из такого большого объема инструментов? В рамках курса остановимся на TensorFlow с использованием API библиотеки Keras. TensorFlow на данный момент является лидером промышленной разработки, поддерживает все типы архитектур нейронных сетей, имеет огромное сообщество, качественное руководство, большое количество готовых к использованию заранее обученных моделей, а свою сложность компенсирует тем, что превратил Keras из просто «обертки» в официальное и рекомендуемое API, а уровень вхождения у Keras – в один из самых высоких.

**Распознавание предметов одежды. Обзор набора данных и выбор архитектуры нейронной сети**

Цель: познакомиться с понятием полносвязной нейронной сети, научиться строить модель нейронной сети с помощью TensorFlow на основе информации о полученном наборе данных, узнать базовые объекты и параметры объектов TensorFlow.

План:

* 1. Полносвязная нейронная сеть прямого распространения
  2. Анализ набора данных с точки зрения дальнейшего построения нейронной сети
  3. Базовые объекты и параметры объектов глубоких нейронных сетей в TensorFlow

**Полносвязная нейронная сеть прямого распространения**

Она же Fully Connected Feed-Forward Neural Networks, FNN

Как видно из названия, нейроны данной сети полностью связаны между собой. Каждый нейрон связан со всеми нейронами предыдущего слоя, собственно, как и со всеми последующего. Это же касается и входов, и выходов нейронной сети – все входы подаются на каждый нейрон, а выходы всех нейронов последнего слоя подаются на выходы нейронной сети.

С помощью FNN достаточно хорошо решаются многие задачи классификации и регрессии.

Но существует два крупных недостатка, которые нужно учитывать при выборе данной архитектуры:

* 1. У данной архитектуры слишком быстро с ростом числа входных данных растет число параметров, которые нужно обучить. Например, для цветного изображения размерностью 100х100 пикселей только входных параметров будет 100х100х3 = 30 000. А первый же скрытый слой хотя бы в 500 нейронов приведет к увеличению параметров до 30 000х500 = 15 000 000.
  2. Проблема затухающего градиента. При обучении нейронной сети ошибка, которую оценивают на выходе нейронной сети, распространяется обратно по всем весам нейронной сети к ее входу. И при наличии относительно большого количества слоев ошибка ближе ко входу уменьшается до значений, близких к нулю (затухает), а значит, веса нейронной сети, которые находятся в первых слоях, перестают обучаться.

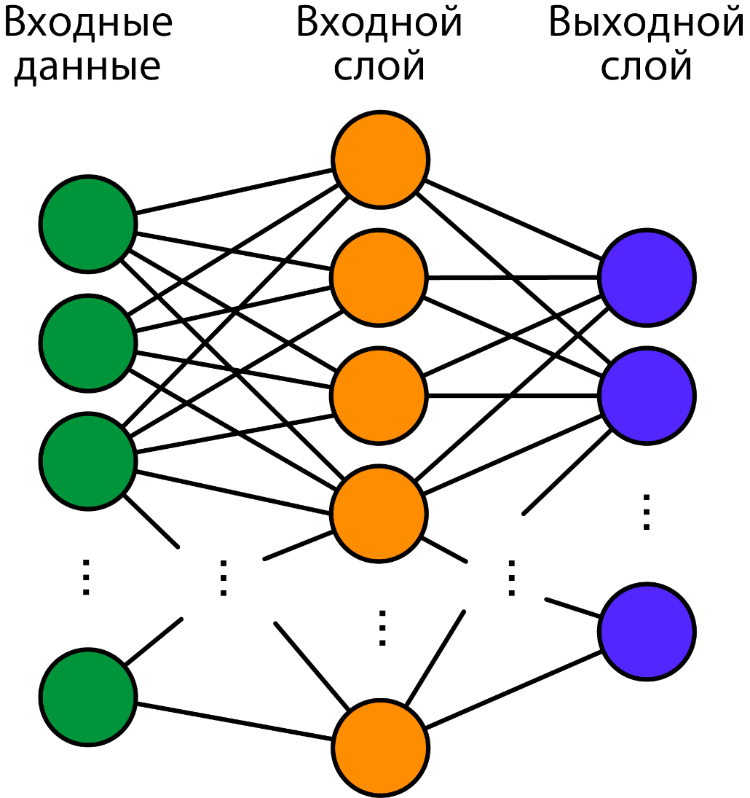


Рис. 1. Пример полносвязной (Fully Connected) нейронной сети

**Анализ набора данных с точки зрения дальнейшего построения нейронной сети**

Теперь следует определиться, какие параметры набора данных непосредственно влияют на параметры архитектуры нейронной сети.

Предположим, что у нас есть набор изображений животных, представленный следующими классами: медведь, волк, рысь. Изображения имеют расширение 100х100 пикселей и являются цветными. Яркость пикселей определена значениями от 0 до 255. Так как изображение цветное, то мы имеем три канала передачи цвета – RGB (red, green, blue). Отсюда следует, что данное изображение – не одна матрица размерностью 100х100 со значениями в диапазоне 0–255, а три подобных матрицы, наложенные друг на друга и в сочетании дающие некие привычные нам цвета.

Но полносвязная нейронная сеть принимает на вход вектор. А значит необходимо преобразовать матрицу в вектор. В случае черно-белого изображения достаточно просто последовательно удлинить первую строку матрицы второй строкой, затем третьей и так до последней строки матрицы. В результате получится одна длинная строка 100х100=10 000 элементов. В случае цветного изображения необходимо сложить строки матриц всех трех цветов. В итоге получится вектор длиной 100х100х3 = 30 000 элементов. Это и будет размерность входного слоя нейронной сети. Это первый параметр, строго завязанный на входные данные.

Далее из набора известно, что есть три класса животных: медведь, волк, рысь. А это значит, что нейронная сеть должна научиться разделять все полученные изображения на три класса (решать задачу классификации). Это число строго определяет количество выходных нейронов в построенной архитектуре сети.

Значения пикселей не влияют на архитектуру нейронной сети. Но для обучения нейронной сети лучше подавать значения от 0 до 1 (либо от -1 до 1). А значит, перед подачей данных на вход нейронной сети необходимо предусмотреть их масштабирование.

Количество скрытых слоев и количество нейронов в них строго не определено. Данные параметры нейронной сети подбирают экспериментально.

Из параметров набора данных можно понять еще пару важных вещей.

Если количество изображений каждого класса (волк, рысь, медведь) одинаково, то говорят, что набор сбалансирован, иначе — нет. Желательно давать на обучение сбалансированные наборы, иначе нейронная сеть может лучше обучиться под класс, представленный большим количеством примеров, а под другой класс обучиться некорректно.

Также, если изображения имеют слишком большое разрешение (скажем больше, чем 512х512), то у нейронной сети может получиться слишком много параметров и при мини-пакетном обучении такой мини-пакет просто не войдет в оперативную память или память GPU. А при этом какого-то прироста информации может и не быть. Давайте в рамках курса условимся считать 512х512 пороговым значением расширения изображений. Если изображение больше, то будем уменьшать. Но, в целом, с этим параметром можно и нужно экспериментировать.

**Базовые объекты и параметры объектов глубоких нейронных сетей в TensorFlow**

На данном этапе рассмотрим лишь пару необходимых объектов и их параметры:

* 1. tensorflow.keras.models.Sequential – это базовая модель нейронной сети, которая, по сути, является контейнером для последовательно помещенных в нее слоев. Не имеет параметров;
  2. tensorflow.keras.layers.Dense – это объект полносвязного слоя нейронной сети. Определяется следующими параметрами:
     + units — это количество нейронов в данном слое;
     + input\_dim – размерность входного слоя (только для слоя, непосредственно следующего за входными данными, для последующих слоев входная размерность определяется автоматически исходя из размерности предыдущего слоя);
     + activation – функция активации, которая будет использована для данного слоя («relu», «softmax» и др.).

Есть и другие параметры данного слоя, но на данном этапе погружение в них может лишь запутать.

Пример объявление простой полносвязной нейронной сети

* 1. Импортируем необходимые объекты

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

* 1. Создаем объект последовательной модели нейронной сети

model = Sequental()

* 1. Помещаем необходимые слои в модель
     1. Слой обрабатывающий входные данные размерностью 784 элемента. Количество нейронов в слое зададим 700, а функцию активации relu

model.add(Dense(units=700,

input\_dim=784,

activation="relu"))

* + 1. Выходной слой, выдающий вероятность принадлежности к одному из **5** классов с функцией активации softmax.

model.add(Dense(5, activation="softmax"))

В целом архитектура сети построена. Дальнейшие действия рассмотрим в следующем разделе.

**Распознавание предметов одежды. Построение архитектуры нейронной сети и ее обучение**

Цель: познакомиться с понятием метрик качества моделей обучения, функциями потерь и оптимизаторами обучения, научиться строить с нуля и обучать нейронную сеть с помощью Keras и TensorFlow

План:

* 1. Метрики качества
  2. Функции потерь и оптимизаторы обучения
  3. Объекты, функции и параметры объектов глубоких нейронных сетей в TensorFlow
  4. Библиотека pandas

**Метрики качества**

Для чего вообще нужны метрики? Предположим, ваш знакомый пробежал некую дистанцию, и вы хотите кому-то об этом рассказать. И вот вы говорите, что он пробежал «много». В ответ вы тут же услышите вопрос: «Много – это сколько?».

Мы привыкли к неким мерам. Расстояние может измеряться в метрах, милях, верстах. Вес – в килограммах, фунтах. И вот метры – это метрика измерения расстояния, фунты – метрика измерения веса. Теперь каждый поймет, что ваш друг пробежал действительно много, если это была дистанция в 100 км.

Так и с нейронной сетью. Обучение нейронной сети происходит не просто так ради удовольствия. Нейронные сети решают большой спектр прикладных задач и в результате нужно понять, а насколько хорошо они это делают. Или при каких параметрах нейронная сеть решает задачу лучше. Чтобы иметь возможность сравнивать, необходимо говорить о качестве модели нейронной сети в разрезе неких величин – метрик качества нейронной сети.

Метрики под каждую задачу могут выбираться индивидуально. Можно даже создавать свои метрики. Но есть ряд общепринятых и часто употребляемых метрик качества.

Для задач регрессии чаще всего применяются:

* + среднее квадратичное отклонение (mean square error – MSE)
  + среднее абсолютное отклонение (mean absolute error – MAE)

**MSE** – это среднее квадратов всех отклонений предсказанных значений от истинных. Квадрат нужен для того, чтобы избежать отрицательных значений. Нам, по сути, не важно, была ошибка в плюс или в минус – важен размер ошибки. Таким образом данная метрика тем лучше, чем ближе ее значению к нулю (отсюда диапазон значений данной метрики от 0 до +∞). У нее есть определенные недостатки. Из-за наличия квадрата она завышает значение больших ошибок (больше единицы) и преуменьшает значение ошибок, значение которых меньше единицы.

**MAE** – это среднее абсолютных значение всех отклонений. Выполняет те же функции, что и MSE, но является более интерпретируемой, так как берет ошибки по модулю, не выполняя нелинейного преобразования.

Существует еще масса модификаций этих метрик (RMSE, MAPE, MSLE), которые можно увидеть в документации TensorFlow <https://www.tensorflow.org/api_docs/python/tf/keras/metrics>

Для задач классификации существуют свои метрики. Рассмотрим их для простоты понимания на примере бинарной классификации, то есть классификации на два класса: отрицательный и положительный (класс 0 и класс 1):

* + accuracy – полная точность

где TP – это количество верно классифицированных меток положительного класса;

TF – это количество верно классифицированных меток отрицательного класса;

FP – это количество неверно классифицированных меток положительного класса;

FN – это количество неверно классифицированных меток отрицательного класса

* + precision – точность для положительного класса
  + recall
  + Cross Entropy

где p – настоящая вероятность класса (это наша ручная разметка, и она имеет значения либо 0, либо 1, так как мы, в отличие от нейронной сети, точно уверены в значениях классов);

ai – вероятность класса i-го объекта, предсказанная нейронной сетью.

**Accuracy** – это точность в целом по выборке. Просто считаются все верные ответы по всем классам и эта сумма делится на общее количество классов. Это одна из часто применяемых метрик, но все она имеет свой недостаток. Например, класс 1 – «клиент не откажется от услуг», а 0 – «откажется». Обычно в задачах подобного рода мы имеем несбалансированную выборку. Предположим есть 100 клиентов, из них 90 отказались, 10 – нет. Таким образом, если нейронная сеть будет увеличивать accuracy, то ей достаточно прогнозировать для всех клиентов, что они откажутся от услуг, и тогда она получит качество (0(TP) + 90(TN)) / 100 = 0,90 или 90 %. Получается, что в случае дисбаланса классов применять данную метрику опасно.

**Precision** – это также точность, но как видно по формуле, она следит лишь за положительным классом. И в ситуации с предыдущим примером даст честные 0(TP) / (0(TP) + 0(FP)) = 0 % (0/0 на самом деле неопределенность, но функция в TensorFlow обработает это исключение). Но у данной метрики также есть недостаток. Предположим, что на той же задаче модель верно определила 5 клиентов, которые не откажутся от услуги (класс 1), а всех остальных отнесла в класс 0. Тогда 5(TP) / (5(TP) + 0(FP)) = 1 или 100 %. Что мы видим? Данную метрику интересуют доля верных ответов среди всех ответов, отнесенных к 1-му классу, она следит, чтобы не было ложно положительных ответов (то есть истинно нулевых, отнесенных к 1-му классу).

**Recall** – это полнота. Она смотрит уже на количество верных ответов для положительного класса. В предыдущей ситуации полнота даст нам 5(TP) / (5(TP) + 5(FN)) = 0,5 или 50 %. Ведь мы верно угадали лишь 50 % из всех правильных ответов. Но можно заметить, что данная метрика будет вводит нас в заблуждение в другой ситуации. Если модель предскажет всех как положительный класс, то получим 10(TP) / (10(TP) + 0(FN)) = 1 или 100 %. Но при этом мы захватили в положительный класс 90 реально ушедших клиентов и даже не заметили. С это ситуацией как раз боролась метрика presision.

Итак presision и recall – это две стороны одной медали и в реальных задачах нужно балансировать между ними, исходя из специфики задачи. К тому же у них есть общий недостаток – они применимы лишь для бинарной классификации. А вот accuracy легко применима и для много классовой.

Так же стоит отметить, что метрики accuracy, precision, recall имеют диапазон значений от 0 (если все неверно) до 1 (если все верно).

**Cross Entropy** – данная метрика применима как для бинарной, так и для много классовой классификации. На несбалансированном классе ведет себя нормально, так как она дает большой штраф, как за неверную классификацию в сторону отрицательного класса, так и в сторону положительного. Допустим, класс должен быть положительным, тогда p = 1, а нейронная сеть предсказала достаточно уверенно (с вероятностью a = 0,01), что класс отрицательный. Тогда -1\*log(0,01) – (1-1)\*log(1-0,01) = -1\*(-4,605) – (0)\*−0,01 = 4,605. И это только на одном примере. А если вспомнить наш пример с отказом клиентов от услуг, где модель начинает всем предсказывать, что они откажутся от услуги, то есть кидать их в отрицательный класс 0, а на самом деле 10 человек должны быть в положительном, то получим метрику 10\*4,605 =46,05, что весьма и весьма плохо. Из этих же выкладок следует, что диапазон данной метрики от 0 до +∞, где чем ближе к 0, тем лучше.

Как и в случае с регрессией, это не все возможные метрики. Про остальные можно почитать по ссылке на документацию TensorFlow, предоставленной выше.

**Функции потерь и оптимизаторы обучения**

Нейронной сети, чтобы обучаться, также нужна какая-то мера. Она должна уметь сравнить свой ответ с верным ответом из учебного ответа, оценить, насколько ее ответ близок к истине или далек от нее, а затем с помощью специального алгоритма обратного распространения ошибки обновить все свои веса.

Но, в отличие от человека, у нейронной сети есть дополнительное требование к функции, которая считает метрику. Эта функция должна быть дифференцируемой, чтобы в дальнейшем нейронная сеть смогла посчитать производные по этой функции и определить градиент данной функции. Так как данная функция определяет, насколько нейронная сеть ошиблась, то ее принято называть функцией потерь (loss function). Из вышеперечисленных метрик функциями потерь могут послужить MAE, MSE, Cross Entropy и другие метрики, не упомянутые в курсе, но удовлетворяющие требованию дифференцируемости. Полный список функций потерь можно увидеть в документации TensorFlow <https://www.tensorflow.org/api_docs/python/tf/keras/losses>.

За сам процесс распространения посчитанной ошибки обратно на все веса модели нейронной сети отвечают так называемые оптимизаторы. Это функции, которые распространяют ошибку, используя различные стратегии:

* + SGD – стохастический градиентный спуск;
  + RMSprop – модификация градиентного спуска;
  + Adam – модификация градиентного спуска.

Полный список данных функций можно увидеть в документации TensorFlow <https://www.tensorflow.org/api_docs/python/tf/keras/optimizers>. А подробнее изучить их, и даже реализовать самостоятельно вы сможете во второй части данного курса.

**Объекты, функции и параметры объектов глубоких нейронных сетей в TensorFlow**

В прошлой теме мы уже частично познакомились с функционалом TensorFlow, но теперь стоит углубить свои знания, чтобы суметь построить пусть и простую, но уже полноценную модель классификации объектов.

Итак в прошлой теме мы построили простенькую модель:

# Создаем последовательную модель

model = Sequential()

# Входной полносвязный слой, 800 нейронов, 784 входа в каждый нейрон

model.add(Dense(800, input\_dim=784, activation="relu"))

# Выходной полносвязный слой, 10 нейронов (по количеству рукописных цифр)

model.add(Dense(10, activation="softmax"))

Но на данный момент нейронная сеть существует лишь на экране. Чтобы модель построилась и была готова к использованию, ее необходимо скомпилировать.

model.compile(loss="categorical\_crossentropy", optimizer="SGD", metrics=["accuracy"])

И вот тут мы видим три понятия, которые в той или иной степени, были упомянута в данном материале: metrics (метрика, на основе которой мы делаем вывод о качестве модели), loss (функция потерь, на основе которой нейронная сеть понимает на сколько она ошиблась), optimizer (функция, которая реализует определенную стратегию обратного распространения ошибки по весам нейронной сети).  Данные параметры можно менять, с учетом того какую задачу вы решаете, и подбирать наиболее удачную конфигурацию данных функций. Возможно в вашей задаче лучше подойдет другой оптимизатор или другая функция потерь.

Затем остается обучить нейронную сеть:

model.fit(x\_train, y\_train,

batch\_size=200,

epochs=10,

verbose=1)

где x\_train – матрица, в которой каждая строка это отдельный объект, а каждый столбец – это отдельный признак объекта;

train – вектор меток объектов (его длина должна совпадать с количеством строк в матрице);

batch\_size – количество объектов из матрицы *x\_train*, которые делают проход по нейронной сети за один раз и после которых выполняется обновление весов нейронной сети;

epochs – количество эпох, на протяжении которых будет учиться нейронная сеть. Одна эпоха – это полный проход всех учебных данных через нейронную сеть. То есть к концу первой эпохи нейронная сеть уже увидит все обучающие данные и в последующих эпохах будет видеть их же, но в другом сочетании внутри одного *batch*. Последующие эпохи могут помочь сделать результат лучше;

verbose – отвечает за выведение информации о процессе обучения. Если значение «0», то информация выводиться не будет, если «1», то мы увидим эту информацию по всем эпохам обучения.

По окончании обучения можно применять модель нейронной сети для предсказаний

predictions = model.predict(x\_to\_predict)

Теперь в переменной predictions будут храниться результаты предсказаний по всем примерам, которые были поданы в матрице x\_to\_predict.

Также модель можно сохранить:

model.save('model')

И в дальнейшем использовать без предварительного обучения

loaded\_model = tf.keras.models.load\_model('model')

**Библиотека pandas**

Отдельно стоит привести несколько примеров использования библиотеки pandas, которая применяется для обработки табличных данных. Многие операции по считыванию данных, их предварительной обработке и сохранению, много проще делать именно в ней.

* + Импортируем библиотеку и присваиваем ей короткое имя, которое уже стало, по сути, стандартом во всей литературе

import pandas as pd

* + Считываем файл в переменную df:

df = pd.read\_csv('train.csv', index\_col='id')

где 'train.csv' – путь к файлу в формате «.csv». Если файл в другом формате, то нужно использовать для его считывания другую функцию, которая начинается на «read\_» и заканчивается названием расширения файла, если pandas поддерживает данный формат.

index\_col – колонка, которая станет индексом для таблицы с данными. Это необязательный параметр. Если его не указать, то pandas создаст столбец с индексами автоматически.

Существует еще много полезных параметров данной функции: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html>

* + Выбираем необходимый столбец, просто указав его имя в квадратных скобках:

y\_train = df['label']

* + Если нужны более тонкие настройки выбора столбцов и строк, то:

x\_train = df.iloc[ : , : -1]

iloc – позволяет указывать диапазон строк, которые хотим получить (например, со второго по четвертый включительно 2 : 5) и через запятую диапазон столбцов (например, с третьего и до предпоследнего 3 : -1). Пустота перед «:» означает с самого начала, пустота после – до самого конца. Поддерживается обратная нумерация.

* + Если хотим перейти от таблицы pandas к numpy массиву, который будет подаваться на вход нейронной сети, то нужно использовать метод «values»:

y\_train = train\_df['label'].values

* + Для записи таблицы в файл, применяйте to\_csv('name.csv')