Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

Интеллектуальные системы

Учебное методическое пособие по лабораторным занятиям, самостоятельной и индивидуальной работе студентов направления магистратуры 09.04.01 «Информатика и вычислительна техника»

Томск 2023

Суханов А.Я.

Интеллектуальные системы: Учебное методическое пособие по лабораторным занятиям, самостоятельной и индивидуальной работе студентов – 170 с.

Учебное методическое пособие содержит программу и задания для лабораторных занятий, все необходимые формы документов для выполнения лабораторных заданий, а также указания к самостоятельной работе.

Оглавление

Введение

Целью дисциплины «Интеллектуальные системы» является глубокое изучение и систематический обзор современных моделей представления знаний, перспективных направлений развития систем искусственного интеллекта и принятия решений, подготовка магистрантов к созданию и применению интеллектуальных автоматизированных информационных систем, ознакомление студентов с теоретическими основами систем искусственного интеллекта (ИИ) и технологией программирования для ИИ. Большую часть примеров кода для реализации аутоэнкодеров, вариационных автоэнкодеров, DGAN, WGAN, алгоритмов обучения с подкреплением можно найти по ссылке на проект в github: https://github.com/saysaysx/artificial-intelligence/, часть примеров так же приведено в самом методическом пособии. В приложении приводится титульный лист для оформления лабораторных работ.

Задачи дисциплины

Ознакомление студентов с основными моделями и парадигмами искусственного интеллекта, построением моделей представления знаний, разработкой моделей предметных областей. Изучение методологий индуктивного и дедуктивного обучения.

Ознакомление студентов с теоретическими основами систем искусственного интеллекта (ИИ) и технологией программирования для ИИ.

Ознакомление с алгоритмами обучения с подкреплением.

Ознакомление с генеративными моделями.

Ознакомление с методологией разработки экспертных систем с использованием системи прямого вывода и обратного вывода в условиях нечеткости и ненадежности знаний.

Ознакомление с языковыми моделями нейронных сетей.

1 ЛАБОРАТОРНАЯ РАБОТА №1. ИЗУЧЕНИЕ СИСТЕМЫ CLIPS

Цели лабораторной работы:

1. Исследовать предметную область, сформировать для нее поле знаний, список фактов, а также правил для работы с ними.

2. Овладеть базовыми конструкциями языка представления знаний CLIPS, такими как deftemplate, deffacts, defrule, deffunction, defglobal.

3. Освоить принципы поиска решения в экспертных системах, основанных на правилах вида "ЕСЛИ-ТО", формирования последовательности активации правил при выводе результата.

4. Научиться решать типичные задачи искусственного интеллекта.

5. Овладеть методами объектно-ориентированного расширения CLIPS.

Задачи работы:

1. Описать словесно факты и правила для разрабатываемого прототипа, представить возможную иерархию понятий.

2. Перевести факты и правила в синтаксис языка CLIPS.

3. Продемонстрировать работоспособность прототипа на конкретных примерах.

Конкретные задания студентам предлагается выбрать самостоятельно. Примерами могут быть системы диагностики бытовой или теле-, аудиоаппаратуры, компьютерной техники. В качестве основы для изучения предметной области можно воспользоваться руководствами к данным устройствам.

4. Для выбранной задачи искусственного интеллекта описать разрабатываемые классы и их иерархию.

5. Разработать и отладить методы данных классов.

6. Продемонстрировать работоспособность экспертной системы при поиске конечного решения из различных начальных состояний фактов.

В качестве задания можно использовать классические задачи искусственного интеллекта типа: задача фермера (волк, коза, капуста), обезьяна и бананы, каннибалы и миссионеры (три миссионера и три каннибала на одном берегу, надо переправиться на другой всем, если каннибалов на берегу оказывается больше то они съедают миссионеров). Также могут быть выбраны в качестве заданий логические задачи математики по определенному упорядочиванию и др.

1.1 Варианты задач

1. Семья

Отец, мать и двое детей – сын и дочь, должны переправиться через реку. Поблизости случился рыбак, который мог бы одолжить им свою лодку. Однако, в лодке могут поместиться только один взрослый или двое детей.

Как семье переправиться через реку и вернуть рыбаку его лодку?

2. Люди и обезьяны

Три человека, одна большая и две маленькие обезьяны должны переправиться через реку. Есть одна лодка, в которой может поместиться не больше двоих. Только люди и большая обезьяна умеют грести. Нельзя, чтобы оставались вместе больше обезьян, чем людей, иначе обезьяны сожрут людей. Обезьяны могут выпрыгивать на берег, когда лодка причаливает.

Как им переправиться через реку?

3. Боязнь темноты

Одной семье надо пройти на другую сторону длинного, узкого и очень тёмного тоннеля. Отец может пройти сквозь тоннель за 1 минуту, мать – за 2, сын – за 4 и дочь за 5 минут. У них есть один факел, которого хватит ровно на 12 минут. В тоннеле могут идти не больше двух человек с факелом.

Как всей семье перебраться на другую строну тоннеля, если все боятся темноты?

4. Переправа через реку – игра

Цель игры – переправить всех людей через реку соблюдая следующие правила:

На пароме могут находиться не более 2-х человек. Только взрослые (отец, мать и полицейский) могут управлять паромом. Отец не может находиться вместе с девочками в отсутствии матери. Мать не может находиться вместе с мальчиками в отсутствии отца. Вор не может находиться вместе с любыми членами семьи в отсутствии полицейского.

5. Ревнивые мужья

Во время наводнения пять супружеских пар оказались отрезанными от суши водой. В их распоряжении была одна лодка, которая могла одновременно вместить только трех человек. Каждый супруг был настолько ревнив, что не мог позволить своей супруге находиться в лодке или на другом берегу с другим мужчиной (или мужчинами) в его отсутствие. Найти способ переправить на сушу этих мужчин и жен в целости и сохранности.

1.2 Пример решения стандартной задачи в пространстве состояний «волк, коза, капуста»

Приведем пример реализации стандартной задачи (фермер, волк, коза, капуста) с использованием CLIPS.

```
< определения начальных фактов для начального состояния >
< определения начальных фактов для формирования конфликтного множества >
(deffacts fermer
      (sost 0 0 0 0)
      (start 1)
      (start 0)
)
< правила для формирования конфликтного множества >
(defrule conflict
       (start ?x)
=>
       (assert(confl ?x  1 1 0))
       (assert(confl  1  ?x 1 0))
       (assert(confl ?x  0 0 1))
       (assert(confl  0  ?x 0 1))
)
< терминальный факт завершающий вывод и определяющий конечное состояние >
(defrule exit
      (sost 1 1 1 1 $?x)
=>
      (printout t "End of branch 1 1 1 1 " crlf (expand$ $?x) crlf)
)
< правило перевозки волка >
(defrule move_wolf
      (sost ?x ?y ?z ?x $?s)
      (not (confl ~?x  ?y ?z ~?x))
      (not (exists (sost ~?x  ?y ?z ~?x $?)))
```

```
=>
        (assert(sost (- 1 ?x) ?y ?z (- 1 ?x) $?s))
)
< правило перевозки козы >
(defrule move_koza
        (sost ?x ?y ?z ?z $?s)
        (not (confl ?x  ?y ~?z ~?z))
        (not (sost ?x  ?y ~?z ~?z $?))
=>
        (assert(sost  ?x ?y  (- 1 ?z) (- 1 ?z) $?s))
)
< правило перевозки капусты >
(defrule move_kapusta
        (sost ?x ?y ?z ?y $?s)
        (not (confl ?x  ~?y ?z ~?y))
        (not (sost ?x  ~?y ?z ~?y $?))
=>
(assert(sost  ?x (- 1 ?y)  ?z (- 1 ?y) $?s))
)
< правило перехода фермера >
(defrule move_fermer
        (sost ?x ?y ?z ?w $?s)
        (not (confl ?x ?y ?z ~?w))
        (not (sost ?x ?y ?z ~?w $?))
=>
        (assert(sost  ?x  ?y  ?z (- 1 ?w) $?s))
)
```

Предлагается вначале освоить начальную работу с CLIPS на основе файла 6_Primenenie_clips.pdf.

Затем рассмотреть примеры решения задачи «фермера» на языке Prolog в файле logic_tasks_fermer_prolog.pdf и на основе примера «обезьяна банан» из книги Братко И. Параграф 2.5, 35 стр.

1.3 Основные сведения о CLIPS

Экспертные системы, созданные с помощью CLIPS, могут быть запущены тремя основными способами:

- вводом соответствующих команд и конструкторов языка непосредственно в среду CLIPS;
- использованием интерактивного оконного интерфейса CLIPS (например, для версий Windows или Macintosh);
- с помощью программ-оболочек, реализующих свой интерфейс общения с пользователем и использующих механизмы представления знаний и логического вывода CLIPS.

Windows-версия среды CLIPS полностью совместима с базовой спецификацией языка. Ввод команд осуществляется непосредственно в главное окно CLIPS. Однако по сравнению с базовой Windows-версия предоставляет множество дополнительных визуальных инструментов (например, менеджеры фактов или правил), значительно облегчающих жизнь разработчика экспертных систем.

Можно воспользоваться командой (load <имя-файла>) для загрузки правил и фактов, и всех определений из текстового файла. Имя файла должно быть строкой и заключаться в кавычки.

CLIPS поддерживает также команду load*. Эта команда полностью иден-тична load за исключением того, что она не отображает процесса загрузки конструкторов.

(load* <имя-файла>)

CLIPS предоставляет также команду save, которая позволяет сохранять в текстовый файл все конструкторы, определенные в данный момент в системе. Синтаксис этой команды идентичен синтаксису команд load и load*.

Отличительной особенностью CLIPS являются конструкторы для создания баз знаний (БЗ):

- defrule – определение правил;
- deffacts – определение фактов;
- deftemplate – определение шаблона факта;
- defglobal – определение глобальных переменных;
- deffunction – определение функций;
- defmodule – определение модулей (совокупности правил);

- defclass – определение классов;
- defintances – определение объектов по шаблону, заданному defclass;
- defmessagehandler – определение сообщений для объектов;
- defgeneric – создание заголовка родовой функции;
- defmethod – определение метода родовой функции.

CLIPS поддерживает следующие типы данных: integer, float, string, symbol, external-address, fact-address, instance-name, instance-address.

1.3.1 Факты и шаблоны в CLIPS

Факты – одна из основных форм представления данных в CLIPS (существует также возможность представления данных в виде объектов и глобальных переменных, но об этом речь пойдет позже). Каждый факт представляет собой определенный набор данных, сохраняемый в текущем списке фактов – рабочей памяти системы. Список фактов представляет собой универсальное хранилище фактов и является частью базы знаний. Объем списка фактов ограничен только памятью вашего компьютера. Список фактов хранится в оперативной памяти компьютера, но CLIPS предоставляет возможность сохранять текущий список в файл и загружать список из ранее сохраненного файла.

В системе CLIPS фактом является список неделимых (или атомарных) значений примитивных типов данных. CLIPS поддерживает два типа фактов – упорядоченные факты (ordered facts) и неупорядоченные факты или шаблоны (non-ordered facts или template facts). Ссылаться на данные, содержащиеся в факте, можно либо используя строго заданную позицию значения в списке данных для упорядоченных фактов, либо указывая имя значения для шаблонов. Упорядоченные факты состоят из поля, обязательно являющегося данным типа symbol и следующей за ним, возможно пустой, последовательности полей, разделенных пробелами. Ограничением факта служат круглые скобки:

(поле_типа_symbol [поле]*)

Так как упорядоченный факт для представления информации использует строго заданные позиции данных, то для доступа к ней пользователь должен знать, не только какие данные сохранены в факте, но и какое поле содержит эти данные.

Неупорядоченные факты (или шаблоны) предоставляют пользователю возможность задавать абстрактную структуру факта путем назначения имени каждому полю. Для создания шаблонов, которые впоследствии будут применяться для доступа к полям факта по имени,

используется конструктор deftemplate. Конструктор deftemplate аналогичен определениям записей или структур в таких языках программирования, как Pascal или C.

Конструктор deftemplate задает имя шаблона и определяет последовательность из нуля или более полей неупорядоченного факта, называемых также слотами. Слот состоит из имени, заданного значением типа symbol, и следующего за ним, возможно пустого, списка полей. Как и факт, слот с обеих сторон ограничивается круглыми скобками. В отличие от упорядоченных фактов слот неупорядоченного факта может жестко определять тип своих значений. Кроме того, слоту могут быть заданы значения по умолчанию. Синтаксис данного конструктора следующий:

(deftemplate <имя-шаблона> [<комментарии>] [<оперделение-слота>*])

1.3.2 Глобальные переменные CLIPS

Помимо фактов, CLIPS предоставляет еще один способ представления данных – глобальные переменные (globals). В отличие от переменных, связанных со своим значением в левой части правила, глобальная переменная доступна везде после своего создания (а не только в правиле, в котором она получила свое значение). Глобальные переменные CLIPS подобны глобальным переменным в процедурных языках программирования, таких как C или ADA. Однако, в отличие от переменных большинства процедурных языков программирования, глобальные переменные в CLIPS слабо типизированы. Фактически переменная может принимать значение любого примитивного типа CLIPS при каждом новом присваивании значения.

С помощью конструктора defglobal в среде CLIPS могут быть объявлены глобальные переменные и присвоены их начальные значения.

(defglobal [<имя-модуля>] <определение-переменной>*)

<определение-переменной> ::= <имя-переменной> = <выражение>

<имя-переменной> ::= ?*<значение-типа-symbol>*

CLIPS позволяет использовать произвольное количество конструкторов defglobal. Необязательный параметр <имя-модуля> указывает модуль, в котором должны быть определены конструируемые переменные. Если имя модуля не задано, то переменные будут помещены в текущий модуль.

Глобальные переменные применяются в любом месте, где могут быть использованы переменные, созданные в левой части правил с некоторыми исключениями. Во-первых, гло-

бальные переменные не могут использоваться как параметры в конструкторах deffunction, defmethod или обработчиках сообщений. Во-вторых, глобальные переменные не могут использоваться для получения новых значений в левой части правил.

Неверно: (defrule example (fact ?*x*) =>).

Верно: (defrule example (fact ?y & :(> ?y ?*x*)) =>)

П р и м е р 1.

```
(defglobal
?*x* = 3
?*y* = ?*x*
?*z* = (+ ?*x* ?*y*)
?*q* = (create$ a b c))
```

После выполнения данного конструктора в CLIPS появятся 4 глобальные переменные: x, y, z и q. Переменной x присваивается целое значение 3. Переменной y – значение, сохраненное в глобальной переменной x (т.е. 3). Переменной z – сумма значений x и y (т.е. 6).

Обратите внимание, что переменная y не является указателем на переменную x, просто их значения в данный момент совпадают. Если изменить значение x, значения переменных y и z, несмотря ни на что, останутся равными 3 и 6 соответственно.

Добавьте еще один конструктор defglobal, объявляющий переменные вещественного и текстового типа, а также переменную со значением типа symbol.

```
(defglobal
?*d* = 7.8
?*e* = "string"
?*f* = symbol)
```

При выполнении команды reset все глобальные переменные получают начальные значения, определенные в конструкторе. Команда ppdefglobal выводит в диалоговое окно системы определение заданной глобальной переменной. Имя глобальной переменной должно быть задано без вопросительного знака и символов *, т.е. name для переменной

?*name*.

Команда list-defglobals предназначена для отображения в диалоговом окне списка имен всех определенных в системе глобальных переменных.

(list-defglobals [<имя-модуля>])

Если необязательный параметр <имя-модуля> не указан, то данная команда выводит имена глобальных переменных, определенных в текущем модуле. Если параметр содержит имя конкретного модуля, команда list-defglobal выводит список переменных, определенных в заданном модуле. Допускается использование символа *. В этом случае команда выведет в диалоговое окно имена всех глобальных переменные, определенных во всех модулях системы.

Команда show-defglobals, в отличие от команды list-defglobals, выводит в диалоговое окно CLIPS не только имена глобальных переменных, но и их значения. В остальном эти две команды практически идентичны.

(show-defglobals [<имя-модуля>])

Команда undefglobal предназначена для удаления определенных пользователем глобальных переменных.

(undefglobal <имя-глобальной-переменной>)

В качестве параметра <имя-глобальной-переменной> допускается использование символа *. В этом случае команда попытается удалить все определенные пользователем глобальные переменные. Если глобальная переменная указана, например, в определении функции, удаление этой переменной закончится неудачей.

Существуют похожие функции для стандартных операций с конструкторами различных типов (не только для defglobal).

(bind <имя-переменной> <выражение>*)

Параметр выражения является необязательным. Если он не задан, то переменной будет установлено начальное значение, заданное в конструкторе defglobal. В случае, если выражение было задано, то его значение будет вычислено и результат присвоен переменной. Если было задано несколько выражений, все они будут вычислены, из их результатов будет составлено составное поле, которое будет присвоено глобальной переменной.

Функция bind возвращает значение false в случае, если переменной по какой-то причине не было присвоено никакого значения. В противном случае функция возвращает значение, присвоенное переменной.

Поскольку переменные в CLIPS слабо типизированы, типы значений, присваиваемые одной и той же переменной, в разные моменты времени могут не совпадать.

CLIPS поддерживает эвристическую и процедурную парадигму представления знаний. Для представления знаний в процедурной парадигме CLIPS предоставляет такие механизмы, как глобальные переменные, функции и родовые функции.

1.3.3 Правила в CLIPS

Кроме того, существует такой способ представления знаний, как правила. Правила в CLIPS служат для представления эвристик или так называемых "эмпирических правил" действий при возникновении некоторой ситуации. Разработчик экспертной системы определяет набор правил, которые вместе работают над решением некоторой задачи. Правила состоят из предпосылок и следствия. Предпосылки называются также ЕСЛИ-частью правила, левой частью правила или LHS правила (left-hand side of rule). Следствие называется ТО-частью правила, правой частью правила или RHS правила (right-hand side of rule).

Предпосылки правила представляют собой набор условий (или условных элементов), которые должны удовлетвориться для того, чтобы правило выполнилось. Предпосылки правил удовлетворяются в зависимости от наличия или отсутствия некоторых заданных фактов в списке фактов или некоторых созданных объектов, являющихся экземплярами классов, определенных пользователем. Один из наиболее распространенных типов условных выражений в CLIPS – образцы (patterns). Образцы состоят из набора ограничений, которые используются для определения того, удовлетворяет ли некоторый факт или объект условному элементу. Другими словами, образец задает некоторую маску для фактов или объектов. Процесс сопоставления образцов фактам или объектам называется процессом сопоставления образцов (pattern-matching). CLIPS предоставляет механизм, называемый механизмом логического вывода (inference engine), который автоматически сопоставляет образцы с текущим списком фактов и определенными объектами в поисках правил, которые применимы в данный момент.

Следствие правила представляется набором некоторых действий, которые необходимо выполнить в случае, если правило применимо к текущей ситуации. Таким образом, действия, заданные вследствие правила, выполняются по команде механизма логического вывода, если все предпосылки правила удовлетворены. В случае, если в данный момент применимо более одного правила, механизм логического вывода использует так называемую стратегию разрешения конфликтов (conflict resolution strategy), которая определяет, какое именно правило будет выполнено. После этого CLIPS выполняет действия, описанные вследствие выбранного

правила (которые могут оказать влияние на список применимых правил), и приступает к выбору следующего правила. Этот процесс продолжается до тех пор, пока список применимых правил не опустеет.

Чтобы лучше понять сущность правил в CLIPS, их можно представить в виде оператора IF-THEN, используемого в процедурных языках программирования, например, таких как Ada или C. Однако условия выражения IF-THEN в процедурных языках вычисляются тогда, когда поток управления программой непосредственно попадает на данное выражение путем последовательного перебора выражений и операторов, составляющих программу. В CLIPS, в отличии от этого, механизм логического вывода создает и постоянно модифицирует список правил, условия которых в данный момент удовлетворены. Эти правила запускаются на выполнение механизмом логического вывода.

Для добавления новых правил в базу знаний CLIPS предоставляет специальный конструктор defrule. В общем виде синтаксис данного конструктора можно представить следующим образом:

(defrule
<имя-правила>
[<комментарии>]
[<определение-свойства-правила>]
<предпосылки> ; левая часть правила
=>
<следствие> ; правая часть правила
)

Имя правила должно быть значением типа symbol. В качестве имени правила нельзя использовать зарезервированные слова CLIPS, которые были перечислены ранее. Определение правила может содержать объявление свойств правила, которое следует непосредственно после имени правила и комментариев.

В справочной системе и документации по CLIPS для обозначения предпосылок правила чаще всего используется термин "LHS of rule", а для обозначения следствия – "RHS of rule", поэтому в дальнейшем мы будем использовать аналогичную терминологию – левая и правая часть правила.

Левая часть правила задается набором условных элементов, который обычно состоит из условий, примененных к некоторым образцам. Заданный набор образцов используется системой для сопоставления с имеющимися фактами и объектами.

Все условия в левой части правила объединяются с помощью неявного логического оператора and. Правая часть правила содержит список действий, выполняемых при активизации правила механизмом логического вывода. Для разделения правой и левой части правил используется символ →. Правило не имеет ограничений на количество условных элементов или действий. Единственным ограничением является свободная память вашего компьютера. Действия правила выполняются последовательно, но тогда и только тогда, когда все условные элементы в левой части этого правила удовлетворены.

Если в левой части правила не указан ни один условный элемент, CLIPS автоматически подставляет условие образец initialfact или initial-object.

После того как в систему добавлены все необходимые правила и приготовлены начальные списки фактов и объектов, CLIPS готов выполнять правила.

1.3.4 Процесс выполнения действий в CLIPS

В традиционных языках программирования точка входа, точка остановки и последовательность вычислений явно определяются программистом. В CLIPS поток исполнения программы совершенно не требует ясного определения. Знания (правила) и данные (факты и объекты) разделены, и механизм логического вывода, предоставляемый CLIPS, применяет данные к знаниям, формируя список применимых правил, после чего последовательно выполняет их. Этот процесс называется основным циклом выполнения правил (basic cycle of rule execution).

Рассмотрим последовательность действий (шагов), выполняемых системой CLIPS в этом цикле в момент выполнения нашей программы:

1. Если был достигнут предел выполнения правил или не был установлен текущий фокус, выполнение прерывается. В противном случае для выполнения выбирается первое правила модуля, на котором был установлен фокус. Если в текущем плане выполнения нет удовлетворенных правил, то фокус перемещается по стеку фокусов и устанавливается на следующий модуль в списке. Если стек фокусов пуст, выполнение прекращается. Иначе шаг 1 выполняется еще один раз.

2. Выполнение действий, описанных в правой части выбранного правила. Использование функции return может менять положение фокуса в стеке фокусов. Число запусков данного правила увеличивается на единицу для определения предела выполнения правила.

3. В результате выполнения шага 2 некоторые правила могут быть активированы или дезактивированы. Активированные правила (т.е. правила, условия которых удовлетворяются в данный момент) помещаются в план решения задачи модуля, в котором они определены. Размещение в плане определяется приоритетом правила (salience) и текущей стратегией разрешения конфликтов (эти понятия будут описаны ниже). Дезактивированные правила удаляются из текущего плана решения задачи. Если для правила установлен режим просмотра активаций, то пользователь получит соответствующее информационное сообщение при каждой активации или дезактивации правила (режим просмотра активаций можно установить с помощью диалогового окна Watch options. Для этого выберите пункт Watch в меню Execution и установите флажок Activations).

4. Если установлен режим динамического приоритета (dynamic salience), то для всех правил из текущего плана решения задачи вычисляются новые значения приоритета. После этого цикл повторяется с шага 1.

Свойства правил позволяют задавать характеристики правил до описания левой части правила. Для задания свойства правила используется ключевое слово declare. Однако правило может иметь только одно определение свойства, заданное с помощью declare.

<определение-свойства-правила> :: = (declare <свойство-правила>)

<свойство-правила> :: = (salience <целочисленное выражение>) | (auto-focus TRUE | FALSE)

Свойство правила salience позволяет пользователю назначать приоритет для своих правил. Объявляемый приоритет должен быть выражением, имеющим целочисленное значение из диапазона от –10 000 до +10 000. Выражение, представляющее приоритет правила, может использовать глобальные переменные и функции. Однако старайтесь не указывать в этом выражении функций, имеющих побочное действие. В случае, если приоритет правила явно не задан, ему присваивается значение по умолчанию, т.е. 0.

Значение приоритета может быть вычислено в одном из трех случаев: при добавлении нового правила, при активации правила и на каждом шаге основного цикла выполнения правил. Два последних варианта называются динамическим приоритетом (dynamic salience). По

умолчанию значение приоритета вычисляется только во время добавления правила. Для изменения этой установки можно использовать команду set-salience-evaluation.

Каждый метод вычисления приоритета содержит в себе предыдущий (т.е. если приоритет вычисляется на каждом шаге основного цикла выполнения правил, то он вычисляется и при активации правила, а также при его добавлении в систему).

Свойство auto-focus позволяет автоматически выполняться команде focus при каждой активации правила.

План решения задачи – это список всех правил, имеющих удовлетворенные условия при некотором, текущем состоянии списка фактов и объектов (которые еще не были выполнены). Каждый модуль имеет свой собственный план решения задачи.

Выполнение плана подобно стеку (верхнее правило плана всегда будет выполнено первым). Когда активируется новое правило, оно размещается в плане решения задачи, руководствуясь следующими факторами:

1. Только активированное правило помещается выше всех правил с меньшим приоритетом и ниже всех правил с большим приоритетом.

2. Среди правил с одинаковым приоритетом используется текущая стратегия разрешения конфликтов для определения размещения среди других правил с одинаковым приоритетом.

3. Если правило активировано вместе с несколькими другими правилами, добавлением или исключением некоторого факта и с помощью шагов 1 и 2 нельзя определить порядок правила в плане решения задачи, то правило произвольным образом упорядочивается вместе с другими правилами, которые были активированы. Заметьте, что в этом случае порядок, в котором правила были добавлены в систему, оказывает произвольный эффект на разрешение конфликта (который в высшей степени зависит от текущей реализации правил).

1.3.5 Стратегии разрешения конфликтов в CLIPS

CLIPS поддерживает семь различных стратегий разрешения конфликтов: стратегия глубины (depth strategy), стратегия ширины (breadth strategy), стратегия упрощения (simplicity strategy), стратегия усложнения (complexity strategy), LEX (LEX strategy), MEA (MEA strategy) и случайная стратегия (random strategy). По умолчанию в CLIPS установлена стратегия глубины. Текущая стратегия может быть установлена командой set-strategy (которая переупорядочит текущий план решения задачи, базируясь на новой стратегии).

- Стратегия глубины. Только что активированное правило помещается выше всех правил с таким же приоритетом. Например, допустим, что факт – А активировал правила 1 и 2 и факт Б активировал правило 3 и правило 4, тогда, если факт А добавлен перед фактом Б, в плане решения задачи правила 3 и 4 будут располагаться выше, чем правила 1 и 2. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.

- Стратегия ширины. Только что активированное правило помещается ниже всех правил с таким же приоритетом. Например, допустим, что факт А активировал правила 1 и 2 и факт Б активировал правила 3 и 4, тогда, если факт А добавлен перед фактом В, в плане решения задачи правила 1 и 2 будут располагаться выше, чем правила 3 и 4. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.

- Стратегия упрощения. Между всеми правилами с одинаковым приоритетом только что активированные правила размещаются выше всех активированных правил с равной или большей определенностью (specificity). Определенность правила вычисляется по числу сопоставлений, которые нужно сделать в левой части правила. Каждое сопоставление с константой или заранее связанной с фактом переменной добавляет к определенности единицу. Каждый вызов функции в левой части правила, являющийся частью условных элементов : , = или test, также добавляет к определенности единицу. Логические функции and, or и not не увеличивают определенность правила, но их аргументы могут это сделать. Вызовы функций, сделанные внутри функций, не увеличивают определенность правила. Например, следующее правило имеет определенность, равную 5.

```
(defrule example
(item ?x ?y ?x)
(test (and (numberp ?x) (> ?x (+ 10 ?y)) (< ?x 100)))
=> )
```

Сравнение заранее связанной переменной >x с константой и вызовы функций numberp, < и > добавляют единицу к определенности правила. В итоге получаем определенность, равную 5. Вызовы функций and и + не увеличивают определенность правила.

- Стратегия усложнения. Между правилами с одинаковым приоритетом только что активированные правила размещаются выше всех активированных правил с равной или меньшей определенностью.

- Стратегия LEX. Между правилами с одинаковым приоритетом только что активированные правила размещаются с применением одноименной стратегии, впервые использованной в системе OPSS. Для определения места активированного правила в плане решения задачи используется "новизна" образца, который активировал правило. CLIPS маркирует каждый факт или объект временным тегом для отображения относительной новизны каждого факта или объекта в системе. Образцы, ассоциированные с каждой активацией правила, сортируются по убыванию тегов для определения местоположения правила.

Активация правила, выполненная более новыми образцами, располагается перед активацией, осуществленной более поздними образцами. Для определения порядка размещения двух активаций правил, поодиночке сравниваются отсортированные временные теги для этих двух активаций, начиная с наибольшего временного тега. Сравнение продолжается до тех пор, пока не остается одна активация с наибольшим временным тегом. Эта активация размещается выше всех остальных в плане решения задачи.

Если активация некоторого правила выполнена большим числом образцов, чем активация другого правила, и все сравниваемые временные теги одинаковы, то активация другого с большим числом временных тегов помещается перед активацией с меньшим. Если две активации имеют одинаковое количество временных тегов и их значения равны, то правило с большей определенностью помещается перед активацией с меньшей. В отличие от системы OPSS, условный элемент not в CLIPS имеет псевдовременной тег, который также используется в данной стратегии разрешения конфликтов. Временной тег условного элемента not всегда меньше, чем временной тег образца.

В качестве примера рассмотрим следующие шесть активаций правил, приведенные в LEX-порядке (запятая в конце строки активации означает наличие логического элемента not). Учтите, что временные теги фактов не обязательно равны индексу, но если индекс факта больше, то больше и его временной тег. Для данного примера примем, что временные теги равны индексам.

rule-6: f-1, f-4

rule-5: f-1, f-2, f-3,

rule-1: f-1, f-2, f-3

rule-2: f-3, f-1

rule-4: f-1, f-2,

rule-3: f-2, f-1

Далее показаны те же активации с индексами фактов в том порядке, в котором они сравниваются стратегией LEX.

rule-6: f-4, f-1

rule-5: f-3, f-2, f-1,

rule-1: f-3, f-2, f-1

rule-2: f-3, f-1

rule-4: f-2, f-1,

rule-3: f-2, f-1

- Стратегия MEA. Между правилами с одинаковым приоритетом только что активированные правила размещаются с использованием одноименной стратегии, впервые использованной в системе OPSS. Основное отличие стратегии MEA от LEX в том, что в стратегии MEA не производится сортировка образцов, активировавших правило. Сравниваются только временные теги первых образцов двух активаций. Активация с большим тегом помещается в план решения задачи перед активацией с меньшим. Если обе активации имеют одинаковые временные теги, ассоциированные с первым образцом, то для определения размещения активации в плане решения задачи используется стратегия LEX. Как и в стратегии LEX, условный элемент not имеет псевдовременной тег.

В качестве примера рассмотрим следующие шесть активаций, приведенные в MEA-порядке (запятая на конце активации означает наличие логического элемента not).

rule-2: f-3, f-1

rule-3: f-2, f-1

rule-6: f-1, f-4

rule-5: f-1, f-2, f-3,

rule-1: f-1, f-2, f-3

rule-4: f-1, f-2,

- Случайная стратегия. Каждой активации назначается случайное число, которое используется для определения местоположения среди активаций с одинаковым приоритетом. Это случайное число сохраняется при смене стратегий, таким образом, тот же порядок воспроизводится при следующей установке случайной стратегии (среди активаций в плане решения задачи, когда стратегия заменена на исходную).

1.4 Функции CLIPS

Для описания функций пользователя служит следующий конструктор.

(deffunction <имя-функции>

[<комментарии>] <обязательные-параметры>

[<групповой-параметр>] <действия>)

<обязательные-параметры> ::= <выражение-простое-поле>

<групповой-параметр> : := <выражение-составное-поле>

Синтаксис конструктора deffunction включает в себя 5 элементов:

- имя функции;
- необязательные комментарии;
- список из нуля или более параметров;
- необязательный символ групповых параметров для указания того, что функция может иметь переменное число аргументов;
- последовательность действий или выражений, которые будут выполнены (вычислены) по порядку в момент вызова функции.

В зависимости от того, задан ли групповой параметр, функция, созданная конструктором, может принимать точное число параметров или число параметров не меньшее, чем некоторое заданное. Обязательные параметры определяют минимальное число аргументов, которое должно быть передано функции при ее вызове. В действиях функции можно ссылаться на каждый из этих параметров как на обычные переменные, содержащие простые значения. Если был задан групповой параметр, то функция может принимать любое количество аргументов большее или равное минимальному числу. Если групповой параметр не задан, то функция может принимать число аргументов точно равное числу обязательных параметров. Все аргументы функции, которые не соответствуют обязательным параметрам, группируются в одно значение составного поля.

Ссылаться на это значение можно, используя символ группового параметра. Для работы с групповым параметром могут использоваться стандартные функции CLIPS, предназначенные для работы с составными полями, такие как length и nth. Определение функции может содержать только один групповой параметр.

CLIPS>

(deffunction print-args (?a ?b $?c)

```
(printout t ?a " " ?b " and " (length ?c) " extras: " ?c crlf))
CLIPS> (print-args 1 2)
1 2 and 0 extras: ()
CLIPS> (print-args a b c d)
a b and 2 extras: (c d)
CLIPS> (print-args a)
[ARGACCESS4] Function print-args expected at least 2 argument(s)
CLIPS>
```

В данном примере с помощью конструктора deffunction определяется функция print-args, которая принимает два обязательных параметра: ?a и ?b, и имеет групповой параметр $?c. Функция выводит на экран свои обязательные параметры, а также число полей в составном параметре и его содержимое.

При вызове функции интерпретатор CLIPS последовательно выполняет действия в порядке, заданном конструктором.

Функция возвращает значение, равное значению, которое вернуло последнее действие или вычисленное выражение. Если последнее действие не вернуло никакого результата, то выполняемая функция также не вернет результата (как в приведенном выше примере). Если функция не выполняет никаких действий, то возвращенное значение равно FALSE. В случае возникновения ошибки при выполнении очередного действия выполнение функции будет прервано и возвращенным значением также будет FALSE.

Функции могут быть само- и взаимно рекурсивными. Саморекурсивная функция просто вызывает сама себя из списка своих собственных действий. В качестве примера можно привести функцию, вычисляющую факториал.

```
(deffuction factorial (?a)
(if (or (not (integerp ?a)) (< ?a 0)) then
(printout t "Factorial error! " crlf)
else
(if (= ?a 0) then 1 else
(* ?a (factorial (- ?a 1))))))
```

Взаимная рекурсия между двумя функциями требует предварительного объявления одной из этих функций. Для предварительного объявления функции в CLIPS используется конструктор deffunction с пустым списком действий. В следующем примере функция foo

предварительно объявлена и таким образом может быть вызвана из функции bar. Окончательная реализация функции foo выполнена конструктором после объявления функции bar.

(deffunction foo ())

(deffunction bar () (foo))

(deffunction foo () (bar))

Команда ppdeffunction выводит определение заданной функции на экран.

(ppdeffunction <имя-функции>)

Команда list-deffunctions предназначена для отображения в диалоговом окне списка имен всех определенных в системе функций.

(list-deffunctions)

Для удаления функций, определенных пользователем с помощью конструкторов deffunction, предназначена команда

undeffunction.

(undeffunction <имя-функции>)

В качестве параметра <имя-функции> возможно использование символа *. В этом случае команда попытается удалить все определенные пользователем функции. Удаление функции закончится неудачей, если выбранная функция в данный момент используется или выполняется (например, правилом).

CLIPS поддерживает следующие процедурные функции, реализующие возможности ветвления, организации циклов в программах и т.п.:

- If – оператор ветвления;
- While – цикл с предусловием;
- loop-for-count – итеративный цикл;
- prong – объединение действий в одной логической команде;
- prong$ – выполнение набора действий над каждым элементом поля;
- return – прерывание функции, цикла, правила и т.д.;
- break – то же, что и return, но без возвращения параметров;
- switch – оператор множественного ветвления;
- bind – создание и связывание переменных.

Среди логических функций (возвращающих значения true или false) следует выделить такие группы:

- функции булевой логики: and, or, not;

- функции сравнения чисел: =, ≠, >, ≥, <, ≤;
- предикативные функции для проверки принадлежности проверяемому типу: integerp, floatp, stringp, symbolp, pointerp;

(относится ли аргумент к xternal-address), numberp (относится ли аргумент к integer или float), lexemerp (относится ли аргумент к string или symbol), evenp (проверка целого на четность), oddp (проверка целого на нечетность), multifildp (является ли аргумент составным полем);

- функции сравнения по типу и по значению: eq, neq.

Среди математических функций следует выделить следующие группы:

- Стандартные: +, –, *, /, max, min, div (целочисленное деление), abs (абсолютное значение), float (преобразование в тип float), integer (преобразование в тип integer).
- Расширенные: sqrt (извлечение корня), round (округление числа), mod (вычисление остатка от деления).
- Тригонометрические: sin, sinh, cos, cosh, tan, tanh, acos, acosh, acot, acoth, acsc, acsch, asec, asech, asin, asinh, atan, atanh.

cot, coth, csc, csch, sec, sech, deg-grad (преобразование из градусов в секторы), deg-rad (преобразование из градусов в радианы), grad-deg (преобразование из секторов в градусы), rad-deg (преобразование из радиан в градусы).

- Логарифмические: log, log10, exp, pi.

Среди функций работы со строками следует назвать функции:

- str-cat – объединение строк;
- sym-cat – объединение строк в значение типа symbol;
- sub-string – выделение подстроки;
- str-index – поиск подстроки;
- eval – выполнение строки в качестве команды CLIPS;
- build – выполнение строки в качестве конструктора CLIPS;
- upcase – преобразование символов в символы верхнего регистра;
- lowcase – преобразование символов в символы нижнего регистра;
- str-compare – сравнение строк;
- str-length – определение длины строки;
- check-syntax – проверка синтаксиса строки;

- string-to-field – возвращение первого поля строки.

Функции работы с составными величинами являются одной из отличительных особенностей языка CLIPS. В их число входят:

- insert$ – добавление новых элементов в составную величину;
- first$ – получение первого элемента составной величины;
- rest$ – получение остатка составной величины;
- length$ – определение числа элементов составной величины;
- delete-member$ – удаление элементов составной величины;
- replace-member$ – замена элементов составной величины.

Функции ввода-вывода используют следующие логические имена устройств:

- stdin – устройство ввода;
- stdout – устройство вывода;
- wclips – устройство, используемое как справочное;
- wdialog – устройство для отправки пользователю сообщений;
- wdisplay – устройство для отображения правил, фактов и т.п.;
- werror – устройство вывода сообщений об ошибках;
- wwarning – устройство для вывода предупреждений;
- wtrase – устройство для вывода отладочной информации.

Собственно, функции ввода-вывода следующие:

- open – открытие файла (виды доступа r, w, r+, a, wb);
- create$ – создание составной величины;
- nth$ – получение элемента составной величины;
- members – поиск элемента составной величины;
- subset$ – проверка одной величины на подмножество другой;
- delete$ – удаление элемента составной величины;
- explode$ – создание составной величины из строки;
- implode$ – создание строки из составной величины;
- subseq$ – извлечение подпоследовательности из составной величины;
- replace$ – замена элемента составной величины
- insert$ – добавление новых элементов в составную величину;
- first$ – получение первого элемента составной величины;

- rest$ – получение остатка составной величины;
- length$ – определение числа элементов составной величины;
- delete-member$ – удаление элементов составной величины;
- replace-member$ – замена элементов составной величины.

Функции ввода-вывода используют следующие логические имена устройств:

- stdin – устройство ввода;
- stdout – устройство вывода;
- wclips – устройство, используемое как справочное;
- wdialog – устройство для отправки пользователю сообщений;
- wdisplay – устройство для отображения правил, фактов и т.п.;
- werror – устройство вывода сообщений об ошибках;
- wwarning – устройство для вывода предупреждений;
- wtrase – устройство для вывода отладочной информации.

Собственно, функции ввода-вывода следующие:

- open – открытие файла (виды доступа r, w, r+, a, wb);
- close – закрытие файла;
- printout – вывод информации на заданное устройство;
- read – ввод данных с заданного устройства;
- readline – ввод строки с заданного устройства;
- format – форматированный вывод на заданное устройство;
- rename – переименование файла;
- remove – удаление файла.

Среди двух десятков команд CLIPS следует назвать основные команды при работе со средой CLIPS:

- load – загрузка конструкторов из текстового файла;
- load+ – загрузка конструкторов из текстового файла без отображения;
- reset – сброс рабочей памяти системы CLIPS;
- clear – очистка рабочей памяти системы;
- run – выполнение загруженных конструкторов;
- save – сохранение созданных конструкторов в текстовый файл;
- exit – выход из CLIPS.

1.5 Модули CLIPS

CLIPS предоставляет возможность разбиения базы данных и решения задачи на отдельные независимые модули. Для создания таких модулей служит конструктор defmodule. С помощью модулей можно группировать вместе отдельные элементы базы знаний и управлять процессом доступа к этим элементам во время решения некоторой задачи. Подобный процесс управления доступом к данным напоминает механизмы пространства имен, используемый в C++, и глобальных и локальных областей видимости в языках C и Ada. Однако, в отличие от механизмов в перечисленных выше языках, области видимости в CLIPS строго иерархичны и однонаправлены: если модуль A может видеть данные модуля B, это не означает, что модуль B может видеть данные модуля A. С помощью управления с ограничением доступа к данным, содержащимся в различных модулях, при решении сложных задач модули могут реализовывать концепцию доски объявлений (blackboard strategy – стратегия решения задач с использованием разнородных источников знаний, взаимодействующих через общее информационное поле). В этом случае отдельный модуль позволяет видеть правилам из других модулей строго определенный набор фактов и объектов. Кроме того, модули используются для управления потоком вычисления правил.

(defmodule <имя-молуля> [<комментарий>]
<спецификации-импорта-экспорта>*)
<спецификация-импорта-экспорта> : :=
(export <элемент-спецификация>) |
(import <имя-модуля> <элемент-спецификации>)
<элемент-спецификации> : := ?ALL | ?NONE |
<конструктор> ?ALL | <конструктор> ?NONE |
<конструктор> <имя-конструктора>
<конструкция>:: = deftemplate | defclass |
defglobal | deffunction | defgeneric

После своего создания модуль не может быть переопределен или удален (за исключением системного модуля MAIN, который пользователь может один раз переопределить). Единственный способ удалить существующий модуль – выполнить команду clear. Во время запуска системы и при вызове команды clear CLIPS автоматически создает предопределенный системный модуль: (defmodule MAIN).

Явное задание модуля выполняется с помощью имени модуля, разделенного с именем конструкции при помощи двойного двоеточия :: . Имя модуля и символ :: называются спецификатором модуля (module specifier). Например, запись

MAIN::find-stuff ссылается на конструкцию find-stuff из ____модуля

f-1 (foo (x 3))

f-2 (bar (y 4))

For a total of 2 facts.

CLIPS> (facts B)

f-1 (foo (x 3))

For a total of 1 fact.

CLIPS>

Таким образом, имя объекта можно указать тремя способами.

<имя-объекта> ::= [<имя>] |

[::<имя>] |

[<модуль> :: <имя>]

Скобки являются обязательным синтаксисом CLIPS.

Каждый модуль имеет свой собственный процесс сопоставления образцов для своих правил и свой план решения задачи. По команде run начинает выполняться план решения задачи модуля, на который в данный момент установлен фокус. Команды reset и clear автоматически устанавливают фокус на модуль MAIN. Выполнение правил продолжается до тех пор, пока в плане решения задачи не останется применимых правил, и другой модуль не получит фокус, либо правая часть одного из выполняемых правил не вызовет функцию return. После того как в плане решения задачи модуля, имеющего фокус, заканчиваются правила, текущей модуль удаляется из стека фокусов (focus stack) и находящийся в стеке следующий модуль получает фокус. Перед выполнением правила текущим становится модуль, в котором данное правило определено. Управлять стеком фокусов можно с помощью команды focus.

В завершение следует иметь в виду, что CLIPS может неудовлетворительно работать в реальном времени, когда потребуется время реакции менее 0,1 с. В этом случае надо исследовать на разработанном прототипе механизмы вывода для всего множества правил предметной области на различных по производительности компьютерах. Как правило, современные персональные компьютеры обеспечивают работу с продукционными системами объемом 1000 – 2000 правил в реальном времени. Web-ориентированные средства на базе JAVA (сис-

темы Exsys Corvid, JESS) являются более медленными, чем, например, CLIPS 6 или OPS-2000. Поэтому CLIPS – лучший на сегодня выбор для работы в реальном времени среди распространяемых свободно оболочек ЭС, разработанных на C++.

2 ЛАБОРАТОРНАЯ РАБОТА №2. ИЗУЧЕНИЕ БИБЛИОТЕКИ РАБОТЫ С НС KERAS НА ПРИМЕРЕ ЗАДАЧ РЕГРЕССИИ, КЛАССИФИКАЦИИ И ЗАДАНИЯ СТИЛЯ ИЗОБРАЖЕНИЯ

Цели работы: Реализовать и разобрать алгоритм обратного распространения ошибки на языке python реализованный с параллельными вычислениями (факультативное задание). Задание а. Использовать предобученную сеть библиотеки keras для задачи классификации изображений. Задание б. Реализовать с использованием библиотеки keras нейронную сеть для решения задачи классификации (Задание в) или регрессии (Задание г). Реализовать нейронную сеть для наложения стиля на изображения. Задание д.

2.1 Задание а. Реализация алгоритма обратного распространения ошибки.

2.1.1 Общие сведения о нейронных сетях

В данном параграфе дана краткая теория по нейронным сетям.

Нейронная сеть представляет собой совокупность взаимосвязанных простых элементов нейронов, и способна выдавать на входное возмущение определенный информационный отклик. Математическую модель нейрона можно представить в следующем виде (формула 2.1):

$$y = \phi(g) = \phi\left(\sum_{i=1}^{n} w_i x_i + w_0\right), \quad (2.1)$$

где y - выходной сигнал нейрона, $\phi(g)$ – функция активации нейрона, w_i - весовой коэффициент *i*-го входа, w_0 - начальное состояние (возбуждение) нейрона, x_i - входные сигналы, i=1, 2…n – номера входов нейрона.

Нейрон можно также представить в виде схемы, приведенной на рисунке 2.1.

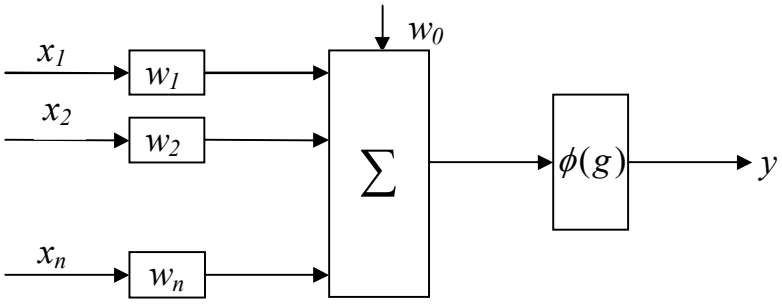


Рисунок 2.1 – Модель формального нейрона

В качестве функции активации могут служить различные функции, например, сигмовидная функция (формула 2.2):

$$\phi(g) = \frac{1}{1 + \exp(-a \cdot g)}. \qquad (2.2)$$

Параметр a определяет наклон сигмовидной функции. График сигмовидной функции представлен на рисунке 2.2.

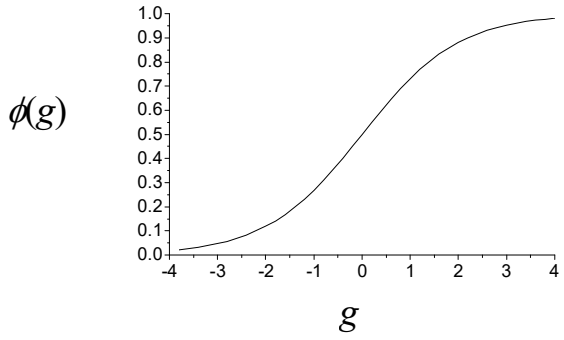


Рисунок 2.2 – Активационная функция

Наклон сигмовидной функции определяет возможности нейрона по различению входных воздействий. Чем круче наклон функции, тем меньше нейрон различает входных воздействий. При пороговой функции активации, представленной на рисунке 2.3, нейрон разделяет все входные воздействия только на два класса.

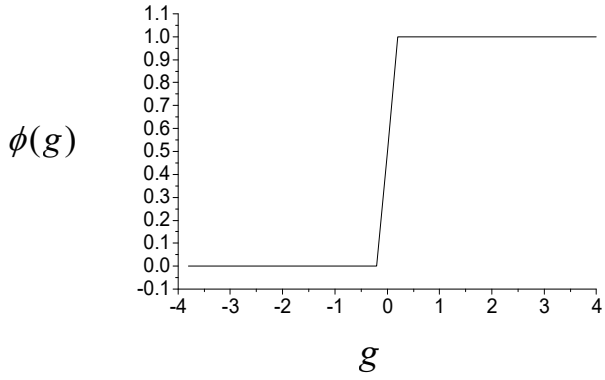


Рисунок 2.3 – Пороговая функция активации

Кроме указанных активационных функций часто используют функцию ReLU (Restricted Linear Unit), принимающая значение g при аргументах больших 0 и 0 при меньших, LeakyReLU при отрицательных значениях аргумента медленно линейно возрастает до 0 и затем равна g, Tanh (гиперболический тангенс) и многие другие.

Например, leakyRELU (формула 2.3)

LeakyReLU(x)=max(0,x)+α∗min(0,x).

$$LReLU(x) = \begin{cases} x, x \geq 0 \\ \alpha x \end{cases} \quad (2.3)$$

где α положительное значение меньше 1 и близкое к 0, например, 0.01.

Существуют различные виды нейронных сетей. Использование конкретного вида зависит от поставленной задачи.

Для того чтобы сеть выполняла решение поставленной задачи, необходимо провести ее обучение. Существуют самообучающиеся нейронные сети и сети с учителем. Обучение с учителем предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются обучающей парой. Обычно сеть обучается на некотором множестве таких пар. Предъявляется входной вектор, вычисляется выход сети и сравнивается с соответствующим целевым вектором, разность (ошибка) с помощью обратной связи подается в сеть, и веса изменяются в соответствии с алгоритмом, стремящимся минимизировать ошибку. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемого низкого уровня. Обучение без учителя не нуждается в целевом векторе для выходов и не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные вектора, то есть, чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Процесс обучения, следовательно, выделяет статистические свойства обучающего множества и группирует сходные векторы в классы. Но до обучения невозможно предсказать какой выход будет производиться данным классом входных векторов. Таким образом, сети с обучением без учителя удобно использовать для задач классификации, для нашей же задачи целесообразнее обучение с учителем, учитывая множество имеющихся в наличии обучающих пар: профилей толщи и профилей концентрации.

Существует различные методы для обучения нейронной сети. Детерминированный метод обучения шаг за шагом осуществляющий процедуру коррекции весов сети, основанную на использовании их текущих значений, а также величин входов, фактических выходов и желаемых выходов. Стохастический метод обучения, выполняющий псевдослучайные изменения величин весов, сохраняя те изменения, которые ведут к улучшению результатов работы сети. Эвристические ал-

горитмы обучения, к которым относится генетический алгоритм поиска, моделирующий процессы природной эволюции и позволяющий из множества решений (популяций) выбрать наилучшее решение. К детерминированным методам относится алгоритм обратного распространения ошибки, к стохастическим методам относится машина Больцмана и машина Коши.

При реализации метода нейронных сетей важно выбрать метод обучения, число слоев, тип нейронов, а также создать обучающую выборку. Создание обучающей выборки, отдельная и сложная задача, необходимо чтобы она была полной, наиболее информативной и не сильно большой по объему.

Математически функционирование трехслойной нейронной сети можно описать следующим выражением (формула 2.4):

$$f_i = \phi_{3,i}\left(\sum_{j1=0}^{L-1} w_{i,j1}^{3} \cdot \phi_{2,j1}\left(\sum_{j2=0}^{M-1} w_{j1,j2}^{2} \cdot \phi_{1,j2}\left(\sum_{j3=0}^{N-1} w_{j2,j3}^{1} \cdot x_{j3}\right)\right)\right) \qquad (2.4)$$

где $\phi_{i,j}$ - активационная функция j-го нейрона i-го слоя;

$w_{i,j}^{n}$ – j-й весовой коэффициент i-го нейрона n-го слоя;

f_i - выходные значения нейронной сети, соответствующие выходу i -го нейрона последнего слоя;

x_i - входные значения нейронной сети, их число равно числу входов каждого входного нейрона нейронной сети.

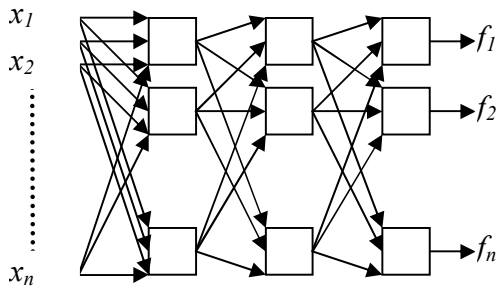


Рисунок 2.4 – Примеры однослойной, двухслойной и трехслойной нейронных сетей

Нейронные сети можно считать неким устройством, позволяющим построить необходимую функциональную зависимость по предлагаемым ей примерам обучения.

2.1.2 Алгоритм обратного распространения ошибки

Алгоритм обратного распространения ошибки — это итеративный градиентный алгоритм. При обучении ставится задача минимизации ошибки нейронной сети.

Обучение осуществляется методом градиентного спуска, т.е. на каждой итерации изменение веса производится по формуле (2.5).

$$w_{ij}(t+1) = w_{ij}(t) - h\frac{\partial E}{\partial w_{ij}}, \tag{2.5}$$

где h – параметр, определяющий скорость обучения.

Функция ошибки в явном виде не содержит зависимости от веса w_{jk}, поэтому воспользуемся формулами неявного дифференцирования сложной функции (формула 2.6):

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial S_j}\frac{\partial S_j}{\partial w_{ij}}, \tag{2.6}$$

где y_j - значение выхода j–го нейрона;

S_j - взвешенная сумма входных сигналов.

При этом множитель $\frac{\partial S_j}{\partial w_{ij}} = x_i$, где x_i– значение i–го выходного нейрона.

Определим первый множитель формулы 2.6 (формула 2.7):

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial S_k}\frac{\partial S_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial S_k} w_{jk}^{(n+1)}, \tag{2.7}$$

где k –число нейронов в слое $n+1$.

Введем вспомогательную переменную $\delta_j^{(n)} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial S_j}$. Тогда можно определить рекурсивную формулу для определения $\delta_j^{(n)}$ n- го слоя, если известно значение этой переменной следующего $(n+1)$-го слоя (формула 2.8):

$$\delta_j^{(n)} = \frac{\partial y_j}{\partial S_j}\sum_k \delta_k^{(n+1)} w_{jk}^{(n+1)}. \tag{2.8}$$

Нахождение $\delta_j^{(n)}$ для последнего слоя нейронной сети не представляет трудностей, так как априорно известен вектор тех значений, который должна выдавать сеть при заданном входном векторе (формула 2.9):

$$\delta_j^{(n)} = (y_j^n - d_j)\frac{\partial y_j}{\partial S_j} \quad (2.9)$$

В результате всех преобразований получим следующее выражение для вычисления приращения весов связи в нейронной сети (формулы 2.10 и 2.11):

$$\Delta w_{ij} = -h\delta_j^{(n)} x_i^n, \quad (2.10)$$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}. \quad (2.11)$$

Таким образом, можно сформулировать полный алгоритм обучения методом обратного распространения ошибки.

1. Подать на вход нейронной сети очередной входной вектор из обучающей выборки и определить значения выходов нейронов в выходном слое.
2. Рассчитать Δw_{ij}, используя формулы (2.10), (2.9) для выходного слоя нейронной сети.
3. Рассчитать Δw_{ij}, используя формулы (2.10), (2.8) для остальных слоев.
4. Скорректировать все веса нейронной сети по формуле (2.11).
5. Если ошибка существенна, вернуться к шагу 1, если нет, то завершить процедуру обучения.

Рассмотренный алгоритм не эффективен в случае, когда производные по различным весам сильно отличаются. Также алгоритм может привести не к глобальному, а локальному минимуму. Для устранения этих недостатков вводят некоторый коэффициент инерционности μ, позволяющий корректировать приращение веса на предыдущей итерации, но он не гарантирует устранения указанных проблем (формула 2.12):

$$\Delta w_{ij}(t) = -h\delta_j^{(n)} x_i^n + \mu\Delta w_{ij}(t). \quad (2.12)$$

2.1.3 Код алгоритма обратного распространения ошибки на языке Python.

```
# класс для сигмовидной активационной функции с производной
class Act:
  def act(self,g):
        if(g<-20):
              g = -20.0
        return 1.0/(exp(-g)+1.0)
  def actdif(self,g):
        if(g<-20):
```

```
                g = -20.0
            ff = exp(-g)
            return ff/(ff+1.0)**2
    def __init__(self):
    # векторизация вычислений
            self.vact = numpy.vectorize(self.act)
            self.vactdif = numpy.vectorize(self.actdif)
    # класс для линейной активационной функции с производной
    class ActLine(Act):
    def act(self,g):
            return g
    def actdif(self,g):
            return 1.0
    # класс для создания слоя (количество нейронов, размер входа нейронов, активационная
функция)
    class Layer:
    def __init__(self,n,neursz,act):
            self.n = n
            self.neursz = neursz
            self.w = np.random.uniform(low=-0.5,high=0.5,size = (n,neursz))
            self.act = act
            self.s = numpy.random.uniform(low=-0.5,high=0.5,size = (n))
            self.y = self.act.vact(self.s)
            self.sg = self.y
    def ask(self,x):
            self.s = numpy.dot(self.w,x)
            self.y = self.act.vact(self.s)
            return self.y
    # класс нейронной сети
    class NeuroNet:
    def __init__(self,nin,lays,acts):
            nins = [nin]+lays
    # создание сети с заданными активационным функциями в слоях
            self.lays = [Layer(lays[i],nins[i],acts[i]) for i in range(len(lays))]
    # прогон сети вперед для получения промежуточных значений и итогового ответа
    def ask(self,x):
            y = self.lays[0].ask(x)
            for i in range(1,len(self.lays)):
                    y = self.lays[i].ask(y)
            return y
    # процедура обратного распространения ошибки от обучающего примера (x,y)
    def back_ppg_learn(self,x,y,al):
            il = len(self.lays)-1
            d = self.ask(x)
```

```
                self.lays[il].sg = (d - y)*self.lays[il].act.vactdif(self.lays[il].s)
                il = il-1
                while il>=0:
                        self.lays[il].sg = numpy.dot(self.lays[il+1].w.T,self.lays[il+1].sg)
                        self.lays[il].sg = self.lays[il].sg*self.lays[il].act.vactdif(self.lays[il].s)
                        il=il-1
                il = 0
                self.lays[il].w = self.lays[il].w  -
numpy.dot(np.array([self.lays[il].sg]).T,np.array([x]))*al
                for il in range(1,len(self.lays)):
                        self.lays[il].w = self.lays[il].w  -
numpy.dot(np.array([self.lays[il].sg]).T,np.array([self.lays[il-1].y]))*al
```

2.2 Задание б. Предобученные нейронные сети классификации keras

Цель лабораторной работы:

Знакомство с библиотекой keras, научиться использовать готовые сети для классификации. Повторить лабораторную и выполнить задание, указанное в конце.

Если есть возможность использовать вместо google colab, https://cloud.yandex.ru/services/datasphere (datasphere yandex) или есть локально установленные IDE с python и соответствующими библиотеками, в частности tensorflow-gpu, то можно использовать их.

Первая часть работы.

Зайдите на сайт https://colab.research.google.com/, зарегистрируйтесь если необходимо в Google. Затем в меню переименуйте текущий NoteBook.

File - > Rename

Назовите свой проект. Вводите код в соответствующую ячейку для кода (рисунок 2.5).

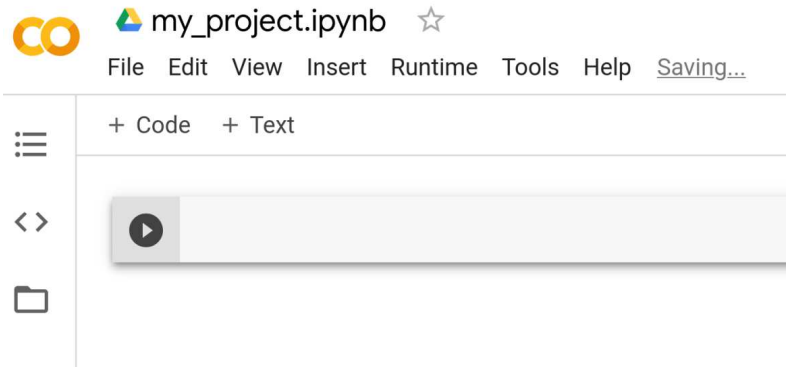


Рисунок 2.5 - Ячейка Cell в Colab

В Runtime, можно использовать Change Runtime type для смены вычислительного устройства на GPU или TPU. Выбрать Python2 или Python3.

Нажав + Code, можно создать новую ячейку (Cell), запускать код или команды, в том числе запускать сам Python.

Можно непосредственно установить нужный пакет. Не забывайте перед командой ставить восклицательный знак (рисунок 2.6).

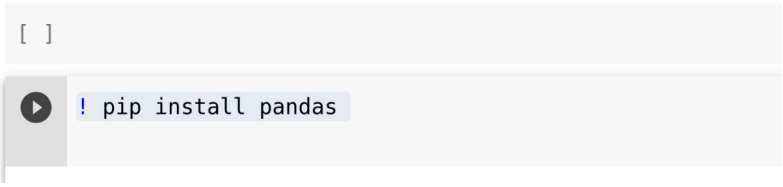


Рисунок 2.6 – Запуск команды из командной строки для установки пакета

Можно клонировать какой-либо проект с github.

! git clone https://github.com/ ……

Для подключения диска Google можно воспользоваться такими командами, которые приведены на рисунке 2.7.

Рисунок 2.7 – Подключение Google Disk

Либо нажав знак «папочка», и затем «Mount Drive» (Рисунок 2.8).

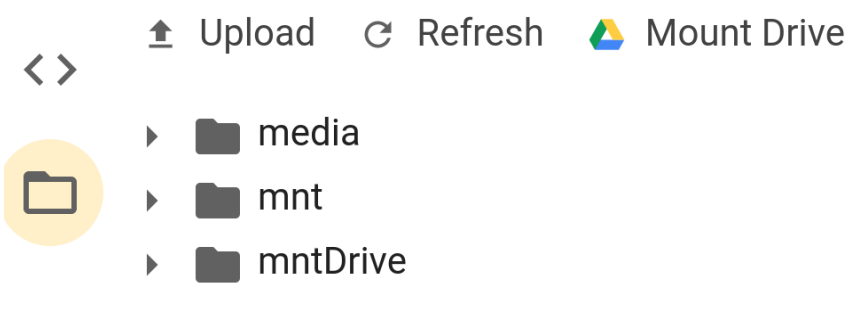


Рисунок 2.8 – Подключение Google Disk

После запуска кода появится сообщение о необходимости ввода кода авторизации, для этого перейдите по указанной ссылке и разрешите сторонним компонентам получать доступ к вашему диску (рисунок 2.9).

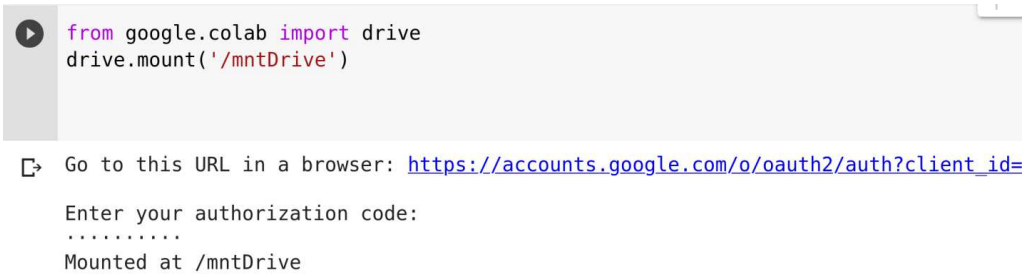


Рисунок 2.9 - Результат подключения диска

Далее приводится пример формы откуда необходимо копировать код авторизации (рисунок 2.10).

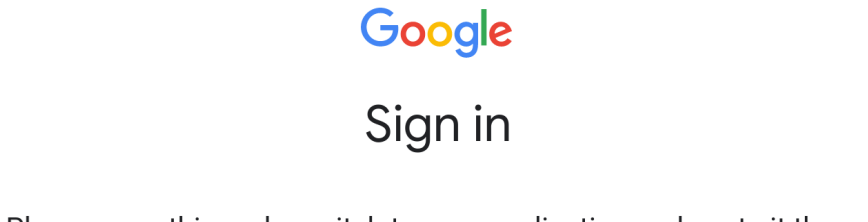


Рисунок 2.10 – Форма откуда копируется ключ авторизации

Ниже должен быть указан код авторизации, который нужно скопировать в поле ввода.

Теперь можно обращаться к файлам на Google диск.

Зайдем в свой Google диск https://drive.google.com, или из google.com.

Дальнейшие действия так же можно совершать не используя Google диска, а используя папки справа, правда те папки являются временными через сутки удаляются, но зато с ними работа гораздо быстрее чем с Google диском.

Создадим директорию python, либо назовите как вам приятнее. Можем теперь записать туда нужный или нужные нам файлы с помощью upload, можно записать целую директорию upload directory (рисунок 2.11).

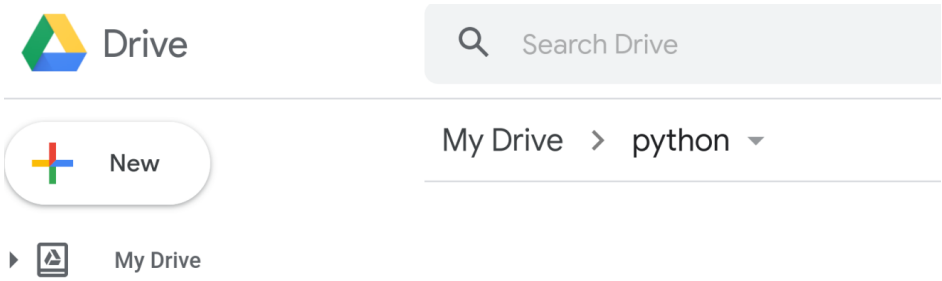


Рисунок 2.11 – Диск с созданным каталогом

Можно вернуться на Colab. Выбрать Download py, и сохранить все Cells в файле на вашем рабочем компьютере.

Вернитесь на гугл диск и создайте папку dataf в папке python и скопируйте туда файлы с изображениями, например, в формате jpg. Возможно, что название для доступа к файлам на google colab уже изменилось, можно посмотреть путь к папке и задать тот путь, который актуален на данный момент.

Ниже приведен код для работы с библиотекой keras и сервисом google colab. В случае чего замените путь к файлам.

Добавьте код, который позволят вам считать изображения.

```
import sys
import os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# добавляем путь к гугл диск
sys.path.append('/mntDrive/My Drive/python')

# функция для чтения изображений
def read_image_files(files_max_count,dir_name):
  files = os.listdir(dir_name)
  if(files_max_count>len(files)): # определяем количество файлов не больше max
    files_count = len(files)
```

```
  else:
    files_count = files_max_count
  image_box = [[]]*files_count
  for file_i in range(files_count): # читаем изображения в список
    image_box[file_i] = Image.open(dir_name+'/'+files[file_i]) #  / ??
  return files_count, image_box

# читаем изображения
files_count , image_box = read_image_files(10,'/mntDrive/My Drive/python/dataf')
```

Добавим код, который реализует классификацию изображений. Изучите и повторите данный код и в отчете опишите результат.

```
import random
import keras
from keras.layers import Input
from keras.models import Model

from keras.applications.resnet50 import preprocess_input, decode_predictions

height = 224
width = 224
# копирование изображений в массив нумпай и изменение их размеров (+ нормализация)
# попробуйте написать код, который сохраняет соотношение сторон
images_resized = [[]]*files_count
for i in range(files_count):
  images_resized[i] = np.array(image_box[i].resize((height,width)))/255.0
images_resized = np.array(images_resized)
nh=224
nw=224
ncol=3
```

```
# описываем входной слой

visible2 = Input(shape=(nh,nw,ncol),name = 'imginp')
# используем предобученную сеть resnet
resnet = keras.applications.resnet_v2.ResNet50V2(include_top=True, weights='imagenet', input_tensor=visible2, input_shape=None, pooling=None, classes=1000)

# с помощью данной закомментированной операции можно привести данные к нормализованному виду, но мы уже сами провели нормализацию выше
# x = preprocess_input(x)
out_net = resnet.predict(images_resized)

# вывод информации о распознанных классах
decode = decode_predictions(out_net, top=1)
for elem in decode:
  print(elem)
```

Далее реализован код для отображения изображений, которые мы распознаем.

```
# рисуем распознаваемые изображения
print(files_count)
fig = plt.figure(figsize=(10,10))
plot_countx = int(files_count**0.5)+1
plot_county = int(files_count**0.5)+1
print(image_box)
viewer = [[]]*files_count # массив саб графиков
for i in range(files_count):
  viewer[i] = fig.add_subplot(plot_countx,plot_county,i+1)
  viewer[i].imshow(np.array(image_box[i])) # делаем график изображения
fig.show()

# выводим информацию о слоях сети
```

```
for layer in resnet.layers:
  print(layer.name)
  print(resnet.get_layer(layer.name).output.shape)

# создаем сеть с выходами в промежуточных слоях resnet

modelres = Model(inputs=visible2, outputs=[
                    resnet.get_layer('avg_pool').output,\
                    resnet.get_layer('conv2_block3_out').output,\
                    resnet.get_layer('conv3_block3_out').output,\
                    resnet.get_layer('conv4_block3_out').output
                    ]
                )

# получаем выходы сети на два изображения

output = modelres.predict(images_resized[0:2])

# отображаем первый выход являющийся слоем с векторным выходом

fig1 = plt.figure()
x = np.linspace(0,len(output[0][0]),len(output[0][0]))
viewer1 = fig1.add_subplot(1,1,1)
viewer1.plot(x,output[0][0])
viewer1.plot(x,output[0][1])
fig1.show()

# отображаем второй выход являющийся сверточным выходом

fig2 = plt.figure(figsize=(12,12))
# меняем порядок осей в массиве, чтобы отобразить карты изображений
outimg = output[1][0].transpose((2,0,1))
```

```
print(outimg.shape)
w_x = int(len(outimg)**0.5)+1
w_y = int(len(outimg)**0.5)+1
for i in range(len(outimg)):
  viewer = fig2.add_subplot(w_y,w_x,i+1)
  viewer.imshow(outimg[i])
fig2.show()
```

Таким образом, мы можем получить выходы промежуточных слоев, иногда они используются в GAN сетях или в других видах сетей при решении различных задач. Часто в задачах стилизации картинок. Реализовать подобный вывод для сети VGG16. Посмотреть разницу между выходами в разных слоях и в одном слое преобразовав изображения промежуточных выходов до одного размера. Результаты отобразить в отчете, попытаться сделать выводы.

2.3 Задание в. Решение нейронной сетью задачи классификации и регрессии

Современные библиотеки реализации нейронных сетей типа keras, caffe, tensorflow (тензорных вычислений), pytorch предоставляют пользователю возможности по созданию структуры сети, ее обучению, часто по использованию возможностей задания своих функций потерь и так далее. Мы будем рассматривать решение задачи классификации и регрессии на примере библиотеки keras.

Общий принцип создания такой сети сводится к заданию входного слоя, промежуточных слоев, получающих на вход выходы предыдущих слоев, и выходного слоя. Затем создается модель сети, к ней применяется процедура компиляции с выбором алгоритма обучения и его параметров, и затем реализуется процедура фитинга с использованием выбранного алгоритма обучения с заданным количеством эпох.

В качестве промежуточных слоев современные библиотеки предлагают большое их разнообразие, включая возможность использовать на входе такого слоя многомерные тензора, изменять структуру передаваемых тензоров и так далее, что реализуется специальными слоями, так же есть возможность выполнять математические или специализированные опе-

рации над данными, включая нормализацию, зашумление, отбрасывание ненужных связей в процессе обучения.

В качестве основных типов слоев для обработки изображений можно выделить сверточные слои

lay1= **Conv2D**(filters=30, kernel_size=(4,4), strides=(1,1), activation='relu', padding='same') (lay).

Filters задает количество выходных карт у тензора (оно же количество применяемых фильтров для сверток), kernel_size – размер фильтра по горизонтали и вертикали, strides – показывает шаг фильтра по вертикали и горизонтали, при strides = (2,2) размер выходной карты будет уменьшен в два раза по вертикали и горизонтали, в качестве activation (активационной функции) можно выбрать linear, relu, sigmoid, tanh, softmax. Softmax например, применяется в последних слоях для задачи классификации, нормируя сумму выходных значений на 1, что дает возможность интерпретировать их как вероятности распознавания класса. Padding = 'same' позволяет сохранить исходный или пропорциональный strides размер исходной карты. Lay – ссылка на предыдущий слой.

Плотный слой (полносвязный):

Dense(units = 30, activation = 'sigmoid', use_bias = 'True')

Units – количество нейронов, и фактически число выходов слоя, use_bias – использовать смещение или нет, есть и другие параметры которые можно посмотреть в документации, они позволяют, например, указать способ задания начальных значений весов. На выходе такого слоя получается одномерный тензор, если не учитывать измерения по количеству примеров обучения в одном батче.

Слой Пулинга, который обычно ставится после сверточного слоя, для выборки из карты данных, путем взятия максимального в некотором окне, здесь окно размера 2 на 2.

MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid") (lay)

Strides указывает на шаг окна, по умолчанию он равен указанному в pool_size.

Padding valid указывает отсутствие заполнения, same указывает тот же размер как у исходной карты. При Пулинге 2 на 2 размер карт уменьшится в 2 раза.

Аналогично работает **AveragePooling2D**, который берет среднее значение в окне.

Для того, чтобы можно было подать один слой на вход другого необходимо чтобы размерность их тензоров совпадала. Например выход сверточного слоя подать на вход полносвязного нельзя, надо преобразовать его с помощью слоя **Flatten() (lay1)** или слоев

GlobalAveragePooling() (lay1) или **GlobalMaxPooling()(lay1).** Первый все выходы предыдущего слоя превращает в одномерный плоский массив, вторые два каждую тензора усредняют, либо берут в ней максимальный элемент, таким образом, получая количество выходов равное количеству карт.

Кроме того, можно использовать слои **Reshape**(target_shape = (5,3)), например, если выходной слой Dense имеет 15 выходов мы преобразуем его в тензор 5 на 3.

Так же есть входной слой Input(shape = (sz1,sz2,sz3…)), естественно не ссылающийся на предыдущие слои. Здесь приведен пример трехмерного входного тензора соответствующему, например, изображению с тремя цветовыми картами, если sz3=3.

Кроме указанных слоев для обучения глубоких сетей бывает трудно обойтись без дополнительных слоев, позволяющих создать Residual слои или ускорить обучение.

В частности, слой нормализации **BatchNormalization**, нормирующий входные данные так, чтобы их область определения попадала в окно наибольшего внимания входного слоя, например, нормируя их от -1 до 1.

Для объединения слоев с разных уровней сети можно использовать **Add** и **Concatenate**.

Для регуляризации и устранения переобучения на обучающей выборке можно использовать слой **Dropout(rate = 0.1),** rate указывает на вероятность отбрасывание весов.

Пример создания модели в последовательном виде.

Ниже приведен пример задания структуры сети в Keras в последовательном виде. Слой дропаут (слой с прореживанием). Слой пакетной нормализации (BatchNormalization). Простой слой (Dense) с функцией активации leakyrelu. Предварительно нужно подключить все указанные компоненты используя import.

```
model = Sequential()
model.add(Dense(64, input_dim=334))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dropout(0.2))
model.add(Dense(16))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dropout(0.05))
```

```
model.add(Dense(11))
model.add(Activation('softmax'))
```

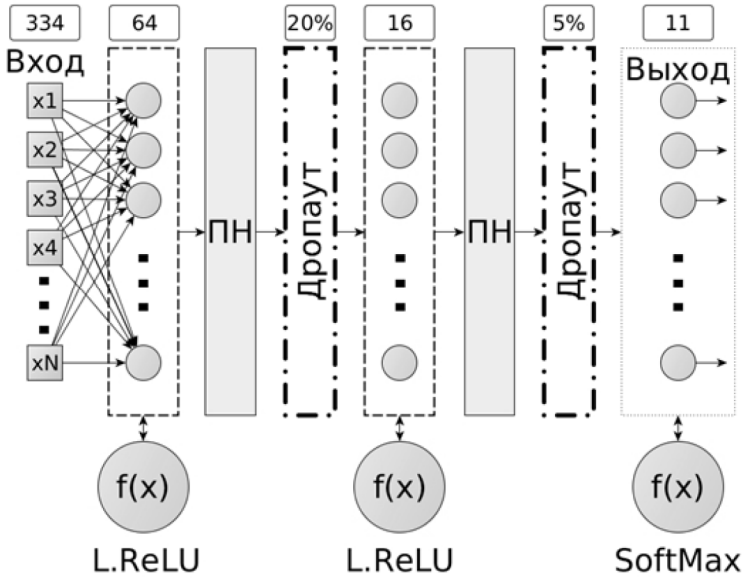


Рисунок 2.12 – Полученный вид модели

Рассмотрим пример решения задачи классификации с использованием стандартного датасета cifar10, содержащего 10 классов различных изображений.

Листинг. Пример классификации.

```
import tensorflow.keras as krs
import tensorflow.keras.utils as ut

# два датасета cifar10 и cifar100
from keras.datasets import cifar10
from keras.datasets import cifar100
# загрузка данных (обучающую и тестовую выборки)
(trainx, trainy), (testx, testy) = cifar10.load_data()
(trainx1, trainy1), (testx1, testy1) = cifar100.load_data()
# выводим размерности многомерного массива
print(trainx.shape)
print(trainy.shape)
print(trainx1.shape)
print(trainy1.shape)

# выводим первые 10 выходных примеров на экран
```

```
print(trainy[0:10])

# нормализация данных от -1 до 1
trainx = (trainx / 255.0-0.5)*1.999
testx = (testx / 255.0-0.5)*1.999
trainx1 = (trainx1 / 255.0-0.5)*1.999
testx1 = (testx1 / 255.0-0.5)*1.999
# преобразование выходных данных в категориальный вид, из (0..9) в (1 0 0 0 0 0 0 0 0 0)
tr_y = ut.to_categorical(trainy)
tst_y = ut.to_categorical(testy)
nh , nw , nc = trainx.shape[1:]
print(f"nh = {nh}, nw = {nw}, nc = {nc}")
# входной слой имеющий размерность входного изображения датасета
input = krs.layers.Input(shape = (nh, nw, nc))
# сверточный слой с пятью выходнымии картами и размером ядра 5x5
lay = krs.layers.Conv2D(filters = 5, kernel_size=(5,5), padding='same', activation='relu') (input)
# слой пулинга уменьшающий размер карты в два раза по обеим осям
lay = krs.layers.MaxPooling2D(pool_size=(2,2)) (lay)
# слой нормализации
lay = krs.layers.BatchNormalization()(lay)
# слой отбрасывания весов
lay = krs.layers.Dropout(rate = 0.2)(lay)
lay = krs.layers.Conv2D(filters = 10, kernel_size=(4,4), padding='same', activation='relu') (lay)
lay = krs.layers.MaxPooling2D(pool_size=(2,2)) (lay)
lay = krs.layers.BatchNormalization()(lay)
lay = krs.layers.Dropout(rate = 0.2)(lay)
lay = krs.layers.Conv2D(filters = 20, kernel_size=(3,3), padding='same', activation='relu') (lay)
lay = krs.layers.MaxPooling2D(pool_size=(2,2)) (lay)
lay = krs.layers.BatchNormalization() (lay)
# слой делающим выходной тензор плоским
lay = krs.layers.Flatten()(lay)
# полносвязный слой из 150 нейронов
lay = krs.layers.Dense(150, activation='relu')(lay)
lay = krs.layers.BatchNormalization()(lay)
lay = krs.layers.Dense(50, activation='relu')(lay)
lay = krs.layers.BatchNormalization()(lay)
# выходной софтмакс слой из 10 выходных нейронов
output = krs.layers.Dense(10,activation='softmax')(lay)
# создаем модель нейронной сети
model = krs.Model(input,output,name = "classifier_of_cifar10")
# рисуем полученную модель
krs.utils.plot_model(model, to_file='model.png', show_shapes=True)
# компилируем модель с алгоритмов обучения Adam и функцией потерь бинарной кроссэн-
тропии
model.compile(optimizer= krs.optimizers.Adam(lr=0.005), loss = 'binary_crossentropy', metrics =
['accuracy'])
# обучаем модель с количеством эпох 80 и размером батча обучения 4000
# проводим кроссвалидацию на тестовой выборке
```

```
model.fit(x = trainx, y = tr_y, batch_size=4000,epochs=80, validation_data = (testx,tst_y))
# проверяем работу на самой же обучающией выборке
out = model.predict(trainx[0:15])
print(out.argmax(axis=1))
print(trainy[0:15].flatten())
```

Реализуйте такую же сеть, но для датасета cifar100, проверьте точность работы обучаемой сети, найдите наиболее оптимальную структуру сети, дающую минимальную функцию потерь, или наибольшую точность на валидационной выборке.

2.4 Задание г. По вариантам. Решение задачи регрессии или классификации

Изучить код, представленный ниже, повторить его. Выполнить задания по вариантам. Для некоторых заданий вместо бинарной кроссэнтропии можно использовать МНК функцию потерь - 'mean_squared_error'.

Для заданий, где графики автоматически масштабируются, можно использовать другой график, например, прямых, которые будут ограничивать пространство графика.

Варианты заданий:

1) Сделать программу, которая позволит распознавать также линейные функции, полукруги или круги разных радиусов с различным положением по центру. Использовать второй график для рисования круга.

2) Сделать программу, которая позволит распознавать количество окружностей на графике. Для этого использовать график с несколькими функциями полуокружностей. Окружности могут быть с разными центрами и разных радиусов.

3) Сделать программу, которая позволит отличать окружности от эллипсов . Для этого использовать график с несколькими функциями полуокружностей. Окружности и эллипсы могут быть с разными центрами и разных радиусов.

4) Сделать программу, которая позволит указывать количество прямых на графике. Для этого использовать график с несколькими функциями.

5) Сделать программу, которая определит радиус круга на графике.

6) Сделать программу, которая определит координаты центра круга на графике

7) Сделать программу, которая определит угол наклона прямой на графике

8) Сделать программу, которая определит точку пересечения двух прямых по рисунку.

9) Сделать программу, которая определит количество разных функций на графике

10) Сделать программу, которая посчитает количество колебаний синусов

Подключение Google диска.

```
from google.colab import drive
drive.mount("/mntDrive")
```

Пример создания примеров для загрузки на гугл диск, входные файлы изображения с графиками, выходные массивы numpy. Классификация функция синус и квадратичной.

```
from math import *
import numpy
import matplotlib.pyplot as plt
from PIL import Image
from random import random

def sin_f(a,w,f,x):
  return a*sin(w*x+f)
def quad(a,b,c,x):
  return a*x*x+b*x+c

def make_examples(n):
  a = -10
  b = 10
  n_x = 200
  for i in range(n):
        # задаем сетку
        x = [a+i*(b-a) for i in range(n_x)]
        # выбираем случайно первый или второй класс
        if random() > 0.5:
              # случайно задаем параметры функции
              a = random()*10
              w = 0.1+random()*10
              f = 2*pi*random()
```

```
            # задаем таблично функцию
            f = [sin_f(a,w,f,x[i]) for i in range(n_x)]
            # задаем класс принадлежности синус функциям
            ans = numpy.array([1,0])
        else:
            a = (random()-0.5)*10
            b = (random()-0.5)*10
            c = (random()-0.5)*10
            f = [quad(a,b,c,x[i]) for i in range(n_x)]
            # задаем класс принадлежности квадратичным функциям
            ans = numpy.array([0,1])
        # рисуем график
        plt.plot(x,f)
        # сохраняем график функции в файл
        plt.savefig(f'/mntDrive/MyDrive/python/data/f{i}.png')
        plt.close()
        # сохраняем класс принадлежности в файл
        numpy.save(f'/mntDrive/MyDrive/python/data/ans{i}',ans)
# создаем тысячу примеров обучения
make_examples(1000)
```

Сделайте сохранение примеров, используя numpy массив примеров, а не по каждому примеру отдельно.

Далее представлен код считывающий примеры обучения и обучающий нейронную сеть.

```
from PIL import Image
import numpy
from math import *
import matplotlib.pyplot as plt
from random import random
```

```
# функция считывающая входные изображения и выходные примеры
def read_image_files(start,files_max_count,dir_name):
  image_box = [[]]*(files_max_count-start)
  height = 128
  width = 128
  ans = [[]]*(files_max_count-start)
  for file_i in range(start,files_max_count):
        num = file_i-start
        print(num)
        image_box[num] = Image.open(dir_name+'/'+f"f{file_i}.png")
        image_box[num] = numpy.array(image_box[num].resize((height,width)))/255.0
        ans[num] = numpy.load(f'/mntDrive/MyDrive/python/data/ans{file_i}.npy')
  # возвращаем примеры обучения в виде массивов numpy
  return numpy.array(image_box), numpy.array(ans)
# считываем обучающие примеры
train_in, train_out = read_image_files(0,900,"/mntDrive/MyDrive/python/data")
# отображаем один пример на графике
plt.imshow(train_in[0])
print(train_in[0].shape)

import keras
from keras.layers import Input
from keras.models import Model
from keras.layers import Dense,Flatten,Conv2D,
GlobalAveragePooling2D,GlobalMaxPooling2D , MaxPooling2D, AveragePooling2D
from keras.layers import BatchNormalization
from keras import optimizers
from keras.utils import plot_model
nh = train_in[0].shape[0]
nw = train_in[0].shape[1]
ncol = train_in[0].shape[2]
```

```
# задаем входной слой
visible1 = Input(shape=(nh, nw, ncol), name='imgdiscr')
# задаем сверточный слой
# 4 — количество фильтров
# (5,5) — размеры окна фильтра
# relu — активационная функция ReLU
# padding — заполнение same, выходная карта имеет тот же размер что и входная
lay = Conv2D(4, (5, 5), activation='relu', padding='same') (visible1)
# Двумерный пулинг слой , карта разбивается на квадраты 2 на 2, в которых берется одно максимальное значение, карта сокращается в 2 раза по каждому измерению
lay = MaxPooling2D(pool_size=(2, 2))(lay)
# Слой нормализации выходных значения предыдущего слоя
lay = BatchNormalization()(lay)
lay = Conv2D(8, (3, 3), activation='relu', padding='same') (lay)
lay = MaxPooling2D(pool_size=(2, 2))(lay)
lay = BatchNormalization()(lay)
lay = Conv2D(16, (5, 5), activation='relu', padding='same') (lay)
lay = MaxPooling2D(pool_size=(2, 2))(lay)
lay = BatchNormalization()(lay)
lay = Conv2D(32, (3, 3), activation='relu', padding='same') (lay)
lay = MaxPooling2D(pool_size=(2, 2))(lay)
lay = BatchNormalization()(lay)
lay = Conv2D(64, (3, 3), activation='relu', padding='same') (lay)
lay = MaxPooling2D(pool_size=(2, 2))(lay)
lay = BatchNormalization()(lay)
# Слой делающий выходные карты плоскими, вектором (одномерный слой)
lay = Flatten()(lay)
# одномерный слой
lay = Dense(4*64,activation='relu')(lay)
lay = BatchNormalization()(lay)
lay = Dense(64,activation='relu')(lay)
lay = BatchNormalization()(lay)
```

```
# выходная активационная функция softmax
lay = Dense(2,activation='softmax')(lay)
# задаем модель сети
model = Model(inputs = visible1,outputs = lay, name = 'discr')
# рисуем модель сети
plot_model(model, to_file='/mntDrive/MyDrive/python/dataf/df.png', show_shapes=True)
# компилируем модель сети с использованием в качестве функции потерь бинарной кроссэнтропии и алгоритма обучения Adam, скорость обучения 0.0003
model.compile(loss='binary_crossentropy',
optimizer=optimizers.Adam(lr=0.0003),
metrics=['accuracy'])
# обучаем сеть с 100 эпохами, 1 эпоха прогон всех примеров, размер батча 200, по которому проходит вычисление градиента и его усреднение
model.fit(train_in,train_out,batch_size=200, epochs=100)

# тестируем сеть на примерах
test_in, test_out = read_image_files(900,920,"/mntDrive/MyDrive/python/data")
print(test_in.shape)
res = model.predict(test_in)
print(test_out, res)
print(test_out - numpy.array(res))
```

2.5 Задание д. Стилизация изображения

Наложение стиля — это попытка перенести стиль одного изображения на другое, где есть какие-либо объекты. Стиль соответственно берется с изображения стиля. Таким образом, выделяется изображение стиля и изображение контента, куда переносится данный стиль.

Задача наложения может быть сведена к задаче минимизации следующего функционала (формула 2.13):

$$E(Y) = (1-\lambda)\sum_{ist}^{n}\left(G_{net}(Y) - G_{vgg}(X_{st})\right)^2 + \lambda\sum_{ic}^{m}\left(C_{net}(Y) - C_{vgg}(X_c)\right)^2 \qquad (2.13)$$

где G – это матрица грамма, полученная от выходов предобученной сети со слоев соответствующих слоям стиля, C – выход получаемых из предобученной сети соответствующий слою

контента (содержимого). Фактически реализуется минимум разности между выходами сети от исходных изображений, хранящих стиль X_{st}, содержимое X_c и итоговым искомым изображением Y.

Матрица грамма представляет собой матрицу скалярных произведений векторизированных матричных объектов из сверточных слоев. Двумерный сверточный слой предобученной нейронной сети возвращает «тензор» или трехмерный массив, условной размерности высота на ширину и на количество карт. Обычное изображение, например, состоит из трех цветовых карт (r,g,b). Таким образом, например если у нас исходный выходной тензор представляет собой матрицу размерности h*w*c, то далее матрица h*w векторизируется, что дает вектор s = h*w, далее получаем матрицу размерности s*c, нужно каждый столбец этой матрицы перемножить с остальными столбцами, таким образом, так же будет получена матрица, фактически матрица ковариаций. Матрично операция выглядит как B(c*s)*A(s*c), где B транспонированная к матрице A, матрица частично векторизированного выхода.

Задачу минимизации можно реализовать, используя современные библиотеки нейронных сетей, например, keras. Общая идея будет заключаться в том, что мы добавим слой в с одним входом, который будет передаваться на входные нейроны количество которых будет равно размерности изображения, фактически, таким образом, получим ситуацию при которой весовые коэффициенты и будут задавать изображение, но мы тут так же добавим смешение с изображением контента и слой свертки. Указанная сеть будет подавать свой выход на вход следующей предобученной сети, например, vgg19, для которой будут взяты выходы, соответствующие слоям стиля и контента. При реализации процедуры минимизации разности между выходами данной сети и результата сети vgg19 от исходных изображений мы получим искомое стилизованное изображение Y.

Подключим требуемые библиотеки.

```
# This is a sample Python script.

import os
import PIL
import matplotlib.pyplot as plt
import numpy
from keras.applications import vgg19 as vgg19v
import keras
from keras.layers import Permute
from keras.layers import Conv2D, LocallyConnected2D, Conv2DTranspose
from keras.layers import Reshape
from keras.layers import Input
```

```
from keras.layers import Dot, Dense
from keras.layers import Lambda, Multiply, Add
from keras.layers import BatchNormalization
from keras.utils import plot_model
from keras import backend as K

# размеры изображения и количество цветов
nw = 224
nh = 224
ncol = 3
```

Добавим функцию, которая будет возвращать исходный формат изображения, после преобразование его для подачи на вход сети vgg19.

```
# функция обратного преобразования изображения
def deprocess_image(x):
   x = x.reshape((nh, nw, 3))
   x[:, :, 0] += 103.939
   x[:, :, 1] += 116.779
   x[:, :, 2] += 123.68
   # 'BGR'->'RGB'
   x = x[:, :, ::-1]
   x = numpy.clip(x, 0, 255).astype("uint8")
   return x
```

Считаем изображение стиля и контента из папки dataf.

```
# имена файлов стиля и контента
files = ["3s.jpeg","3c.jpeg"]
box = []
# читаем файлы
for item in files:
  image = PIL.Image.open(f"dataf/{item}")
  image = image.resize((nh,nw))
  image = numpy.array(image)
  # преобразование изображений для нейронной сети vgg19
  box.append(vgg19v.preprocess_input(image))

box = numpy.array(box)
# добавление изображений стиля и контента на графики
fig = plt.figure(figsize = (7,7))
ax1 = fig.add_subplot("221")
ax1.imshow(deprocess_image(box[0].copy()))
ax2 = fig.add_subplot("222")
ax2.imshow(deprocess_image(box[1].copy()))
# берем стандартную сеть VGG19
visible2 = Input(shape=(nh,nw,ncol),name = 'imginp')
```

```
vgg19 = keras.applications.VGG19(
    include_top=True,
    weights="imagenet",
    input_tensor=visible2,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)

# имена слоев стиля и контента в сети vgg19
style_n = [
    "block1_conv1",
    "block2_conv1",
    "block3_conv1",
    "block4_conv1",
    "block5_conv1"
]

content_n = ["block5_conv2"]

# функция нормировки
def div_norm(x):
    return x/(x.shape[1]*x.shape[2])**2

def normalize(lay):
    lam = Lambda(div_norm)(lay)
    return lam

# расчет матрицы грамма от выходов слоя lay
def addgramm(lay):
    # изменяем размерности матриц стилей, было например (224*224*512) стало
(50176*512)
    visd = Reshape(target_shape = (lay.shape[1]*lay.shape[2],lay.shape[3])) (lay)
    # транспонируем матрицу 50176*512
    vist = Permute((2,1),input_shape = (lay.shape[i] for i in range(1,4))) (visd)
    # умножаем матрицу 512*50176 на 50176*512
    dt = Dot(axes = (2,1)) ([vist,visd])
    c_dt = Lambda(lambda x : x/(4*(lay.shape[1]*lay.shape[2]*lay.shape[3]))) (dt)
    return c_dt

# добавление выходных слоев стиля с матрицей грамма
outst = []
for st in style_n:
    outst.append(addgramm(vgg19.get_layer(st).output))
# добавление выходных слоев контента
outcont = []
for st in content_n:
```

```
    outcont.append(vgg19.get_layer(st).output)

# получаем выходные слои
allnet = outst+outcont

input = vgg19.layers[0].output
print(f"inpt sh {input.shape}")
# создаем новую сеть с выходами нужных слоев стиля и контента
modelst = keras.Model(inputs=input, outputs=outst)
# получаем значения выходных слоев стиля - матрицы грамма
v1 = modelst.predict(numpy.array([box[0]]))
modelcont = keras.Model(inputs=input, outputs=outcont)
# получаем значения выходных слоев контента
v2 = modelcont.predict(numpy.array([box[1]]))

# объединяем получая требуемый выходной пример,
# к нему приводится обучаемая сеть
# в v требуемый нами выход к которому необходимо стремиться
# состоящий из пяти выходов стилей и одного выхода контента
v = v1+[v2]

# функция для установки слоев сети, которые будут или не будут обучаться
def trainable(model,flag):
    model.trainable = flag
    for l in model.layers:
        l.trainable = flag
    return
# задаем сеть, которая будет формировать изображение
# данная сеть принимает на вход 1, и два изображения контента
# можно ограничиться одним изображением, или попытаться
# подать изображение стиля
vis1 = Input(shape=(1,),name = 'our_input1')
vis2 = Input(shape=(nh,nw,3),name = 'our_input2')
vis3 = Input(shape=(nh,nw,3),name = 'our_input3')
# векторизуем входные изображения меняя размерность
vis22 = Reshape(target_shape = (nh*nw*3,)) (vis2)
vis33 = Reshape(target_shape = (nh*nw*3,)) (vis3)
# добавляем слой с nh*nw*3 нейронами на каждый из которых поступает
# один входной элемент, мы зададим его равным 1
dens =  Dense(nh*nw*3,activation= 'linear') (vis1)
# слой который суммирует слой изображения и слой контента
dens = Add()([dens,vis22])
dens1 =  Dense(nh*nw*3,activation= 'linear') (vis1)
dens1 = Add()([dens1,vis33])
# слой который суммирует первый и второй слой сумматоров
dens = Add()([dens,dens1])
# изменяем векторизированный выход в трехмерный тензор изображения
# которое и будет итоговым стилизованным изображением после
```

```
# применения слоя свертки тремя фильтрами 3 на 3
# что порождает три цветовых карты.
imgout = Reshape(target_shape = (nh,nw,3)) (dens)
# добавляем слой свертки для прохождения фильтра
imgout = Conv2D(filters=3,kernel_size=(3,3),padding='same') (imgout)
# формируем модель которая будет создавать стилизованное изображение
imgmodel = keras.Model(inputs=[vis1,vis2,vis3], outputs=imgout)
# указываем что модель будет обучаться и ее коэффициенты
trainable(imgmodel,True)

# создаем модель «сверху» которой будет сеть формирующая искомое
# стилизованное изображение, а «снизу» предобученная сеть
model = keras.Model(inputs=visible2, outputs=allnet)
modelall = model(imgout)
gener = keras.Model([vis1,vis2,vis3], modelall, name = 'gen_discr')
# отключаем изменяемость весов сети vgg19 при обучении
trainable(model,False)


# задаем функцию потерь (по методу наименьших квадратов)
# задаем скорость обучения lr
# задаем веса для каждого из квадратов разности, последний
# коэффициент, например, при разности контентов, у нас она одна
# и пять разностей для выходов стилей
gener.compile(loss='mean_squared_error',
        optimizer=keras.optimizers.Adam(lr=0.01),
        metrics=['accuracy'],loss_weights = [0.5,0.5,0.5,0.5,0.5,0.7])
# отображаем модель сети
plot_model(gener, to_file='./out/modelall.png', show_shapes=True)
# задаем входные и выходные вектора
# вход 1, изображение контент, изображение контент
N = 1
x1 = numpy.zeros(N)+1
x2 = numpy.repeat(numpy.array([box[1]]),N,axis=0)
x3 = numpy.repeat(numpy.array([box[1]]),N,axis=0)

x = [x1,x2,x3]

# выход стиль1,стиль2,стиль3,стиль4, стиль5,контент1

y = [[]]*len(v)
for i in range(len(v)):
    y[i] = numpy.repeat(v[i],N,axis=0)


# обучаем сеть

for i in range(1):
```

```
    #gener.load_weights("./out/s5c.dat")
    gener.fit(x,y,batch_size=N, epochs=10000)
    gener.save_weights("./out/s5c.dat")

ans = imgmodel.predict(x)


ax3 = fig.add_subplot("223")
ax3.imshow(deprocess_image(ans[0].copy()))
plt.show()
```

Задание.

Реализовать и запустить на коллаб заданную сеть, либо использовать tensorflow, материалы можно найти в сети. Исследовать влияние коэффициента скорости сходимости на наложение стиля.

Далее выполнить следующие исследования по вариантам.

1) Проверить влияние активационных функций, relu, sigmoid, linear и других в слоях обучаемой сети.

2) Добавить еще один сверточный слой, сравнить результаты.

3) Добавить вместо изображения контента изображение стиля на один из входов сети и сравнить результаты.

4) Изменить размерность фильтра в слое свертки и сравнить результаты.

5) Добавить слои типа Conv2DTranspose и MaxPooling2D с целью проверки влияния на наложение стиля

6) Изменить алгоритм обучения на sgd и сравнить результаты

7) Изменить алгоритм обучения на AdaDelta и сравнить результаты

8) Изменить алгоритм обучения на AdaGrad и сравнить результаты

9) Добавить слой активационной функции layers.LeakyReLU()(x) и проверить ее влияние.

10) Добавить активационную функцию selu и проверить результаты для разных параметров.

3 ЛАБОРАТОРНАЯ РАБОТА №3. АУТОЭНКОДЕР. ВАРИАЦИОННЫЙ АУТОЭНКОДЕР

3.1 Автоэнкодер

Нейронная сеть автоэнкодер решает задачу восстановления исходного изображения при прохождении его через нейронную сеть со сжимающим слоем в пространство меньшей размерности. Такой слой можно считать промежуточным и разделяющим автоэнкодер на сеть энкодера и декодера. Сам по себе слой возвращает скрытые признаки, кодирующие изображения обучающей выборки, так же можно считать их дескрипторами объектов, присутствующими в обучающей выборке.

В автоэнкодерах входной сигнал совпадает с выходным. Автоэнкодер состоит из 3 компонентов: энкодера, кода и декодера. Кодер сжимает ввод и создает код, затем декодер восстанавливает ввод, используя только этот код. Код представляет собой компактное «обобщение» или «сжатие» входных данных, также называемое представлением в скрытом пространстве.

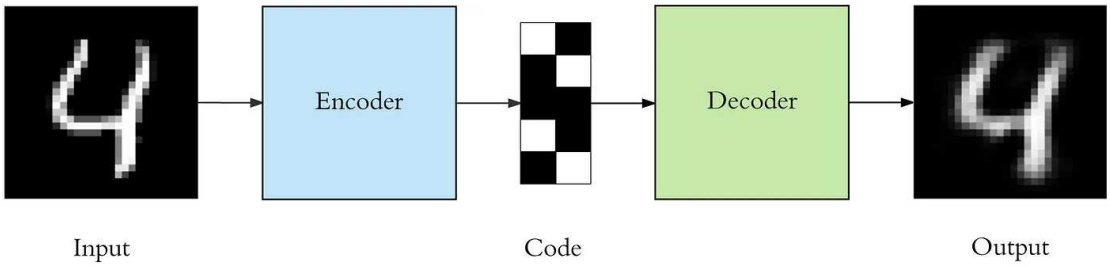


Рисунок 3.1 - Общая схема устройства автоэнкодера

Чтобы построить автоэнкодер нужны 3 составляющих: метод кодирования, метод декодирования и функция потерь для сравнения вывода с целью.

Автоэнкодеры в основном представляют собой алгоритм уменьшения размерности (или сжатия) с парой важных свойств:

Специфичные для данных: автоэнкодеры могут осмысленно сжимать только те данные, на которых они были обучены. Поскольку они изучают функции, специфичные для данных обучающих данных, они отличаются от стандартного алгоритма сжатия данных, такого как gzip. Таким образом, мы не можем ожидать, что автоэнкодер, обученный рукописным цифрам, будет сжимать пейзажные фотографии.

Вывод автоэнкодера не будет точно таким же, как ввод, это будет близкое, но ухудшенное представление. Если нужно сжатие без потерь, они не подходят.

Автоэнкодеры считаются методом обучения без учителя, поскольку для обучения им не нужны явные метки. Но если быть более точным, они самоконтролируются, потому что они генерируют свои собственные метки на основе обучающих данных.

Автоэнкодер можно использовать не только для сжатия изображения, но и создания дескриптивного описания изображений или объектов, довольно часто их для этого и используют. Далее эти дескрипторы можно использоваться для решения задачи классификации. Таким образом, например, получаем дескрипторы, обучаем на них классификатор и далее будем для классификации брать полученные энкодером дескриптор. Более того, если у нас мало размеченных данных, то мы можем потом обучать классификатор на части дескрипторов, имеющих разметку.

Если детально рассмотреть кодировщик, кода и декодер, то и кодировщик, и декодер представляют собой полностью связанные нейронные сети с прямой связью, по сути, ИНС. Код — это один слой ИНС с размерностью по нашему выбору. Количество узлов в слое кода (размер кода) — это гиперпараметр, который мы задаем перед обучением автоэнкодера.

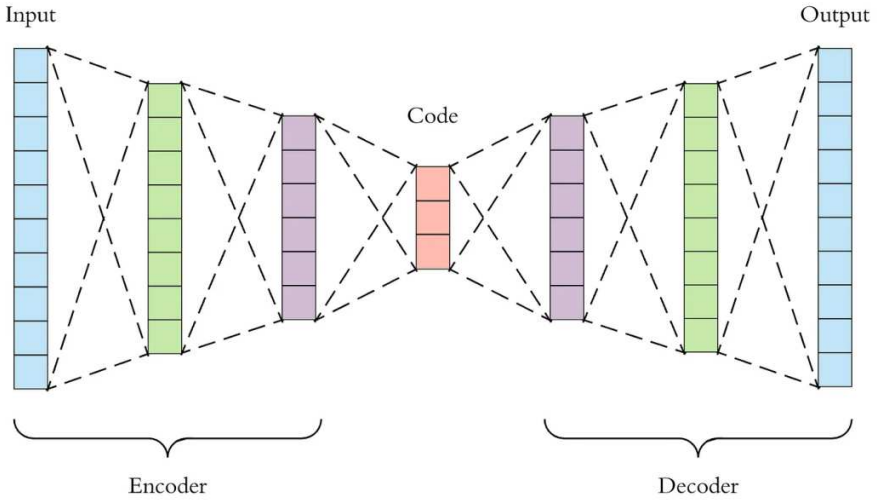


Рисунок 3.2 – Устройство автоэнкодера

Сначала вход проходит через кодировщик, который представляет собой полносвязную ИНС, для создания кода. Декодер, который имеет аналогичную структуру ANN, затем производит вывод только с использованием кода. Цель состоит в том, чтобы получить результат, идентичный входу. Обратите внимание, что архитектура декодера является зеркальным от-

ражением кодировщика. Это не требование, но обычно так и есть. Единственное требование - размерность ввода и вывода должна быть одинаковой. Со всем, что находится посередине, можно играть.

Есть 4 гиперпараметра, которые нам нужно установить перед обучением автоэнкодера:

Размер кода: количество узлов в среднем слое. Меньший размер приводит к большему сжатию.

Количество слоев: автоэнкодер может быть сколь угодно глубоким. На рисунке выше у нас есть 2 слоя как в кодере, так и в декодере, без учета ввода и вывода.

Количество узлов на слой: архитектура автоэнкодера, над которой мы работаем, называется стековым автоэнкодером, поскольку слои располагаются один за другим. Обычно сложенные автоэнкодеры выглядят как «бутерброд». Количество узлов на слой уменьшается с каждым последующим уровнем кодировщика и увеличивается обратно в декодере. Кроме того, декодер симметричен кодеру с точки зрения структуры уровней. Как отмечалось выше, в этом нет необходимости, и мы имеем полный контроль над этими параметрами.

Функция потерь: мы используем либо среднеквадратичную ошибку (mse), либо бинарную кроссэнтропию. Если входные значения находятся в диапазоне [0, 1], то мы обычно используем кроссэнтропию, в противном случае мы используем среднеквадратичную ошибку. Для более подробной информации посмотрите это видео.

Автоэнкодеры обучаются так же, как и ИНС, посредством обратного распространения ошибки.

Привдем пример кода реализующего типичный автоэнкодер. Кроме всего прочего данный код строит двумерный дескриптор изображений используемый, далее на графике выводятся кластера.

```
import matplotlib.pyplot as plt
import tensorflow.keras as krs
import numpy
from keras.datasets import mnist
nw = 28
nh = 28
num_hide = 98
# загружаем примеры обучения mnist (рукописные цифры)
(trainx, trainy), (testx, testy) = mnist.load_data()
# нормируем от -1 до 1 изображения цифр
all_image = (trainx/255.0-0.5)*1.999
# добавляем дополнительное измерение соответствующее одной цветовой карте
```

```
all_image = numpy.expand_dims(all_image, axis=3)
# задаем входной слой экодера высота на ширину на количество карт
encoder_input = krs.layers.Input(shape=(nw,nh,1))
# задаем сверточный слой с 32 фильтрами-картами и фильтрами 3 на 3
# оставляет тот же размер карты 28*28
lay = krs.layers.Conv2D(32, (3, 3), strides = (2,2), activation='relu', padding='same')(encoder_input)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(64, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
# добавляем слой прореживания
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(128, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(256, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
# слой который многомерный тензорный слой превращает в плоский вектор
lay = krs.layers.Flatten()(lay)
# выходной кодирующий слой
lay_out_encoder = krs.layers.Dense(num_hide, activation="linear", name='den4')(lay)
# создаем сеть энкодера
encoder = krs.Model(encoder_input, lay_out_encoder)

# создание сети декодера, входной слой
decoder_input = krs.layers.Input(shape=(num_hide,))
lay = krs.layers.Dense(128*7*7)(decoder_input)
# преобразуем плоский слой в многомерный тензор 7*7*128
lay = krs.layers.Reshape(target_shape=(7,7,128))(lay)
lay = krs.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(lay)
# повышаем размерность карты в два раза, будет 14*14
# можно использовать билинейную интерполяцию если хотите
lay = krs.layers.UpSampling2D(size=(2,2))(lay)
lay = krs.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(lay)
lay = krs.layers.UpSampling2D(size=(2,2))(lay)
lay_out_decoder = krs.layers.Conv2D(1, (3, 3), activation='tanh', padding='same')(lay)
# создаем сеть декодера
decoder = krs.Model(decoder_input,lay_out_decoder)

# объединяем обе сети в автоэнкодер
lay_out = decoder(lay_out_encoder)
autoencoder = krs.Model(encoder_input,lay_out)
# полученный вид модели сохраняем в файле в виде изображения
krs.utils.plot_model(autoencoder, to_file='./out/autoencoder.png', show_shapes=True)
# компилируем модель автоэнкодера с функцией потерь mse и скоростью обучения 0.0002
autoencoder.compile(loss='mean_squared_error', optimiz-
er=krs.optimizers.Adam(learning_rate=0.0002),
                    metrics=['accuracy'])
# запускаем 40 эпох обучения с размером батча 4000
ep = 40
autoencoder.fit(x = all_image,y = all_image,batch_size = 4000,epochs = ep)
```

```
# получаем выход автоэнкодера, изображения который он получает
index = numpy.random.randint(0,len(all_image),9)
out_img = autoencoder.predict(all_image[index])
# выводим их на графике
fig = plt.figure(figsize=(5,5))
for i in range(3):
    for j in range(3):
        ax = fig.add_subplot(3,3,i*3+j+1)
        ax.imshow(out_img[i*3+j][:,:,0])
plt.show()
# реализуем работу  с энкодером получая скрытый кодовый слой
from scipy.cluster.vq import kmeans2
out_vec = encoder.predict(all_image)
# получим центроиды кластеров для 10 кластеров
centroid, label = kmeans2(out_vec, 10, minit='++')

# получим центроиды кластеров для 2 кластеров
centroid1, label1 = kmeans2(out_vec, 2, minit='++')

# считаем координаты кластера как разность с центроидом
out_vec1 = (out_vec - centroid1[0])**2
out_vec2 = (out_vec - centroid1[1])**2
# берем среднее значение
outm = out_vec1.mean(axis=1)
outstd = out_vec2.mean(axis=1)

coutm = centroid.mean(axis=1)
coutstd = centroid.mean(axis=1)

fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(1,1,1)
# рисуем на графике кластер объектов в виде среднее дисперсия
for i in range(10):
    mask = label == i
    ax.scatter(outm[mask],outstd[mask])
plt.show()

# реализуем другой способ кластеризации, с помощью автоэкодера с дескриптором размером
2
# эти два значения и будут использоваться как двумерные координаты кластера
encoder_input1 = krs.layers.Input(shape=(num_hide))
lay = krs.layers.Dense(2000, activation="relu")(encoder_input1)
lay = krs.layers.Dense(500, activation="relu")(lay)
lay = krs.layers.Dense(100, activation="relu")(lay)
lay_out_encoder1 = krs.layers.Dense(2, activation="linear", name='den')(lay)
encoder1 = krs.Model(encoder_input1, lay_out_encoder1)
decoder_input1 = krs.layers.Input(shape=(2,))
lay = krs.layers.Dense(100, activation="relu")(decoder_input1)
```

```
lay = krs.layers.Dense(500, activation="relu")(lay)
lay = krs.layers.Dense(2000, activation="relu")(lay)
lay_out_decoder1 = krs.layers.Dense(num_hide, activation="linear")(lay)
decoder1 = krs.Model(decoder_input1,lay_out_decoder1)
lay_out1 = decoder1(lay_out_encoder1)
autoencoder1 = krs.Model(encoder_input1,lay_out1)
krs.utils.plot_model(autoencoder1, to_file='./autoencoder1.png', show_shapes=True)
autoencoder1.compile(loss='mean_squared_error', optimiz-
er=krs.optimizers.Adam(learning_rate=0.0002),
                metrics=['accuracy'])

autoencoder1.fit(x = out_vec,y = out_vec,batch_size = 4000,epochs = ep*2)
out_vec = encoder1.predict(out_vec)
fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(1,1,1)
centroid, label = kmeans2(out_vec, 10, minit='random')
print(label)

# рисуем полученные кластера цифр
for i in range(10):
  mask = label == i
  ax.scatter(out_vec[mask,0],out_vec[mask,1])
  plt.text(centroid[i,0], centroid[i,1], i , fontdict=None)
plt.show()
```

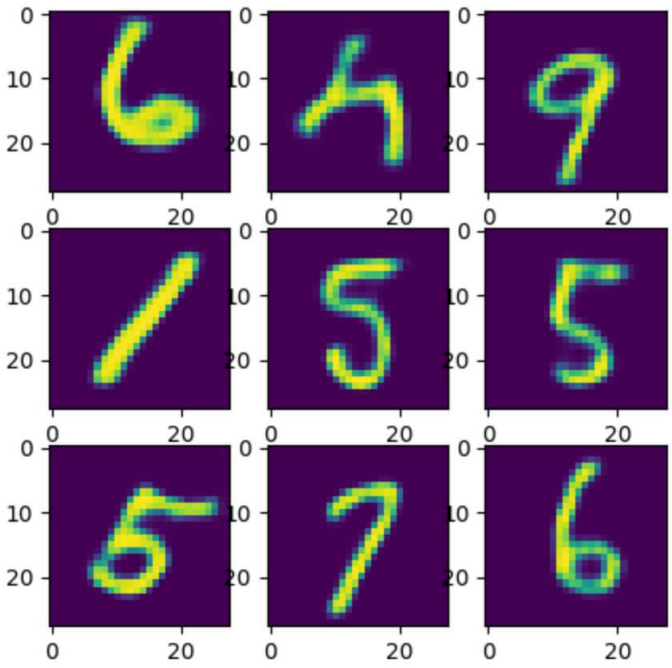


Рисунок 3.3 – Результат вывода автоэнкодера

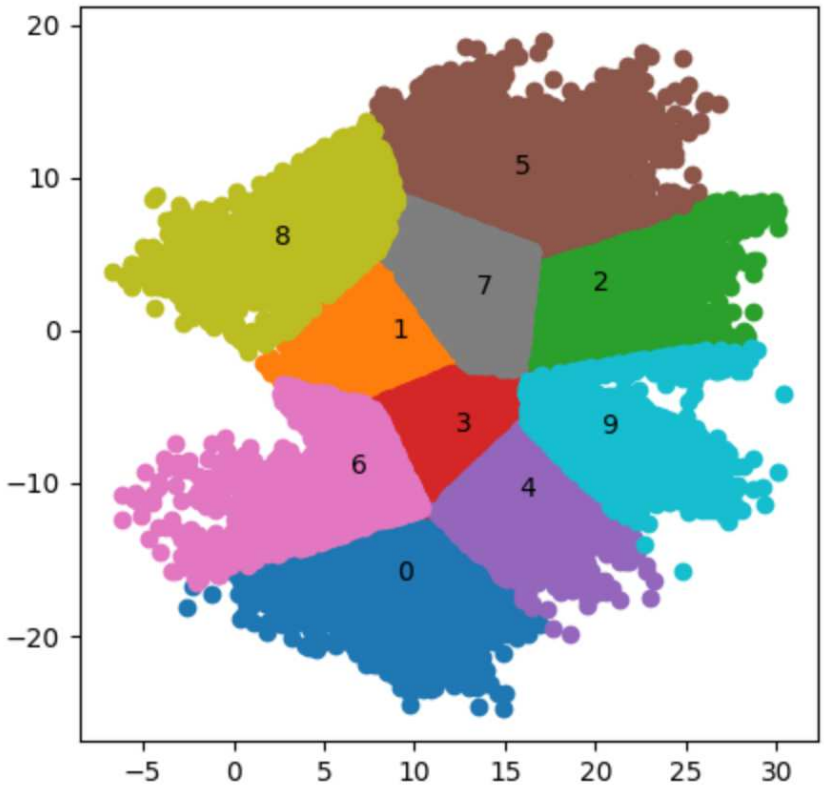


Рисунок 3.4 – Результаты кластеризации с помощью автоэнкодера

3.2 Вариационный автоэнкодер

Автоэнкодер можно использовать не только для сжатия изображений или получения дескрипторов, но и для генерации. То есть, например, если подать на вход декодера сгенерированный из области распределения дескрипторов скрытый вектор, то мы получим сгенерированное изображение. Осталось задаться вопросом, а какое вероятностное распределение у дескрипторов, ведь оно может быть любым и нам сложно будет сгенерировать вектор обладающим нужными свойствами, чтобы не выйти за область определения и не получить непонятное изображение.

Потому были разработаны вариационные автоэнкодеры, которые позволяют генерировать изображения из области заданного распределения скрытых векторов, делается это путем создания специальной сети и использования дополнительной функции потерь, которая минимизирует получаемое распределение случайного скрытого вектора к заданному виду распределения, обычно в качестве такое распределеняи берется нормальное. Как вы знаете нормальное распределение задается двумя параметрами: средним и дисперсией.

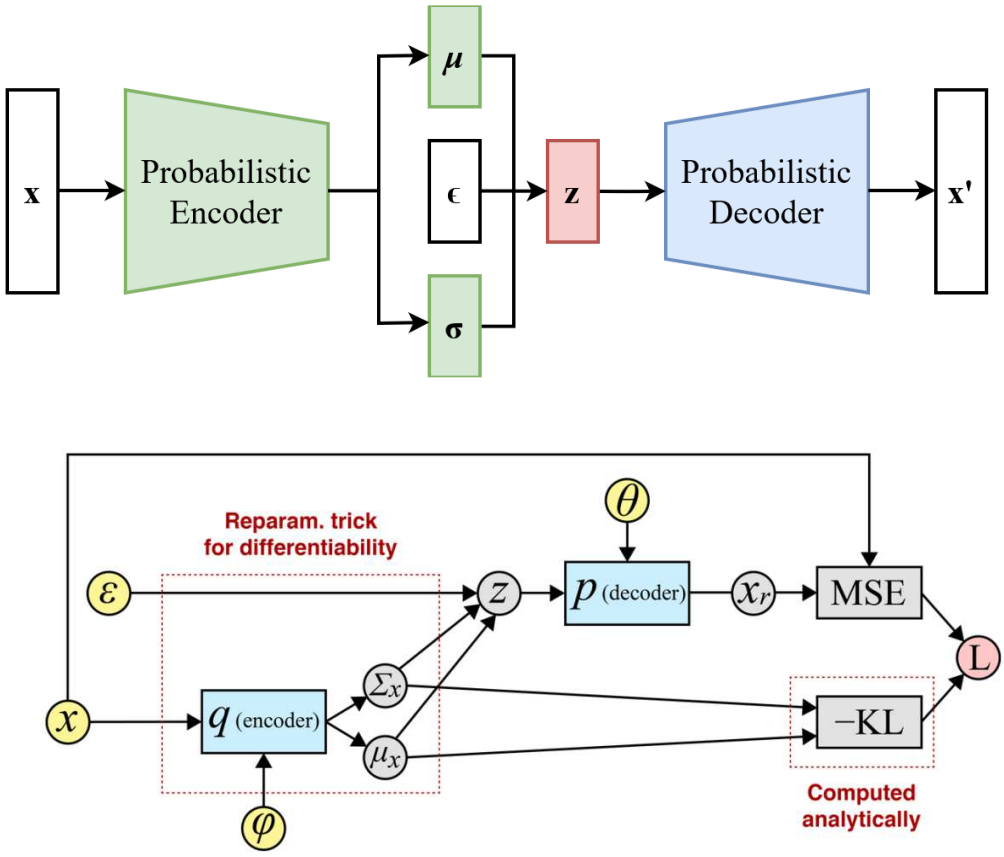


Рисунок 3.5 – Структурная схема вариационного аватоэгкодера

На данной схеме, как видите (рисунок 3.5), присутствует слой, который получает среднее, дисперсию, а также используя эти значение по генератору ε формируется случайный нормальный вектор z который подается на декодер, который формирует новое x (допустим, изображение) из пространства распределения исходных изображений.

Для того, чтобы получить вектора распределенные близкие к заданному виду распределения используется дивергенция Кулбака-Лайблера, она ассиметрична, потому это не расстояние или не мера, грубо говоря расчет Q относительно P даст другой результат. Фактически она показывает похожесть одного распределения на другое (формула 3.1).

$$D(P||Q) = \int p(x) \cdot log \frac{p(x)}{q(x)} dx \tag{3.1}$$

Пусть есть генерируемое скрытое пространство Z из котрого генерируются z, и пространство наблюдаемых объектов X с элементами x, при этом есть рапраделение для x p(x).

Обучая авоэнкодер, мы получаем распределение z при условии x, p(z|x) но оно не удовлетворяет нашим требованиям по возможности использования для генерации далее по z объектов x. Тогда попытаемся сделать замену на q(z) обладающую нужными нам свойствами минимизируя дивергенции между q(z) и p(z|x). Обычно q(z) формируют с учетом нормального распределения с параметром среднего и дисперсии.

Рассмотрим формальный вывод для реализации вариационного автоэнкодера.

У нас есть выборка с объектами, имеющая распределение p(x) и так же вектор скрытых параметров z, который мы хотим использовать для генерации x. Таким образом, для генерации мы должны получить распределение z и получить p(x|z).

Рассмотрим дивергенцию Кулбака-Лайблера, которую будем минимизировать по q(z) (формула 3.2):

$$D(Q||P) = \int q(z|x) \cdot log \frac{q(z|x)}{p(z|x)} dz =$$
$$= E\left[log \frac{q(z|x)}{p(z|x)}\right] = E\left[log \frac{q(z|x)p(x)}{p(x,z)}\right] = \log p(x) - E\left[log \frac{p(x,z)}{q(z|x)}\right] \quad (3.2)$$

В нашем случае при минимизации $\log p(x)$ этот член выражения является константой независящей от q(z), тогда максимизируя (формула 3.3):

$$E\left[log \frac{p(x,z)}{q(z|x)}\right] \underset{q(z)}{\longrightarrow} max \quad (3.3)$$

можно найти нужное нам распределение. Данное выражение часто обозначают как ELBO (Evidence lower bound – нижняя граница достоверности, вариационная нижняя граница). Является полезной нижней границей логарифмического правдоподобия наблюдаемых данных (формула 3.4).

$$ELBO = E\left[log \frac{p(x,z)}{q(z|x)}\right] = E\left[log \frac{p(x|z)p(z)}{q(z|x)}\right] =$$
$$= E[logp(x|z)] + E[logp(z)] - E[logq(z|x)] =$$
$$= -E\left[log \frac{q(z|x)}{p(z)}\right] + E[logp(x|z)] =$$
$$= -\int log \frac{q(z|x)}{p(z)} q(z|x)dz + E[logp(x|z)] =$$
$$= -D(q(z|x)||p(z)) + E[logp(x|z)] \quad (3.4)$$

Таком образом, решается задача минимизации дивергенции Кулбака-Лайблера и максимизации распределения объектов в зависимости x от z. При этом q(z|x) воспроизводит

скрытое пространство по x, а p(x|z) воспроизводит связь реальных объектов от скрытых состояний, что близко к энкодеру и декодеру.

Общая задача максимизации по параметрам функции ELBO выглядит следующим образом (формула 3.5):

$$-D(q(z|x,\varphi)||p(z)) + E_{q(z|x,\varphi)}[logp(x|z,\theta)] \underset{\varphi,\theta}{\longrightarrow} max \quad (3.5)$$

Если в качестве распределения p(z) взять нормальное распределение с нулевым средним и единичной дисперсией, а в качестве q(z|x) диагональную многомерную нормальную случайную величину, то для такого случая дивергенция Кульбака-Лайблера может быть записана следующим образом (формула 3.6):

$$D_{KL}(N((\mu_1,\mu_2..\mu_n)^T, diag(\sigma_1^2,\sigma_2^2..\sigma_n^2))||N(0,I)) = \frac{1}{2}\sum_i^n \mu_i^2(x) + \sigma_i^2(x) - log\sigma_i^2(x) - 1 \quad (3.6)$$

Задача максимизации $E_{q(z|x,\varphi)}[logp(x|z,\theta)]$ представляет собой типичную задачу максимума правдоподобия, которая может быть сведена к методу наименьших квадратов между моделью x и реальными объектами x при нормально распределенных скрытых параметрах, таким образом, мы получим типично обучаемую часть автоэнкодера.

Для того, чтобы получить диагональное многомерное нормальное распределение как уже было сказано выше и используется трюк репараметризации, где на нормально распределенный вектор с нулевым средним и единичной дисперсией умножается вектор среднеквадратического отклонения и добавляется вектор средних значений получаемые нейронной сетью, нейронная сеть учится их получать.

Далее представлен код реализующий вариационный автоэнкодер.

```
import matplotlib.pyplot as plt
import tensorflow.keras as krs
import numpy
nw = 28
nh = 28
from keras.datasets import mnist
(trainx, trainy), (testx, testy) = mnist.load_data()
all_image = ((trainx/255)-0.5)*1.99

num_hide = 100

# слой свертки для дискриминатора
def conv(filt,size,x):
    y = krs.layers.Conv2D(filters = filt, kernel_size = (size,size),padding='same') (x)
    lay = krs.layers.LeakyReLU(alpha=0.2) (y)
    #lay = krs.layers.Dropout(rate = 0.2)(lay)
    return lay
```

```
# слой деконволюции для генератора
def dconv(filt,size,x, strides=2):
    y = krs.layers.UpSampling2D(size=strides)(x)
    y = krs.layers.Conv2D(filters = filt,kernel_size=size,padding='same') (y)
    lay = krs.layers.LeakyReLU(alpha=0.2) (y)
    return lay

encoder_input = krs.layers.Input(shape = (nh,nw,1))
lay = conv(4,3,encoder_input)
lay = krs.layers.MaxPooling2D((2,2)) (lay)
lay = conv(8,3,lay)
lay = krs.layers.MaxPooling2D((2,2)) (lay)
lay = conv(16,3,lay)
lay = krs.layers.MaxPooling2D((2,2)) (lay)
lay = conv(32,3,lay)
lay = krs.layers.Flatten()(lay)
lay = krs.layers.Dense(num_hide) (lay)
lay = krs.layers.LeakyReLU(alpha=0.2) (lay)
lay = krs.layers.Dense(num_hide) (lay)

z_mean = krs.layers.Dense(num_hide, activation='linear', name = 'z_mean') (lay)
z_std = krs.layers.Dense(num_hide, activation='relu', name = 'z_std') (lay)

noise_input = krs.layers.Input(shape  =(num_hide,))
noise = krs.layers.GaussianNoise(stddev = 1) (noise_input)
batch_size = 100

mult = krs.layers.Multiply()([z_std,noise])
out = krs.layers.Add() ([mult,z_mean])

# слой учитывающий расчет дивергенцию Кулбака-Лайблера

def vae_layer(args):
    z_mean, z_std = args
    z_std_sq = z_std*z_std
    z_mean_sq = z_mean*z_mean
    all = -(1+krs.backend.log(z_std_sq+1e-19)-z_std_sq-z_mean_sq)*0.5/num_hide
    return all

# функция потерь для учета расстояния Кулбака-Лайблера
def vae_loss(y_true,x_pred):
    return krs.backend.mean(x_pred)

out_loss = krs.layers.Lambda(vae_layer) ([z_mean,z_std])

encoder = krs.Model(encoder_input, outputs = [z_mean,z_std],name = 'encoder')

decoder_input = krs.layers.Input(shape = (num_hide,))
```

```
lay = krs.layers.Dense(7 * 7 * 128, activation='relu') (decoder_input)
lay = krs.layers.Reshape(target_shape=(7,7,128)) (lay)
lay = dconv(16,3,lay)
lay = dconv(8,3,lay)
lay = krs.layers.Conv2D(filters = 1,kernel_size=7,padding='same',activation='tanh') (lay)
decoder = krs.Model(decoder_input, outputs = lay, name = 'decoder')

modelall = decoder(out)
vae = krs.Model([encoder_input,noise_input], [modelall,out_loss], name = 'vae')
vae.compile(loss = ['mean_absolute_error',vae_loss],optimizer=krs.optimizers.Adam(lr=0.00002),
                        metrics=['accuracy'])

krs.utils.plot_model(vae, to_file='./modelvae.png', show_shapes=True)
krs.utils.plot_model(encoder, to_file='./encoder1.png', show_shapes=True)
x = all_image

x_noise = numpy.zeros((x.shape[0],num_hide))
y_noise = numpy.zeros((x.shape[0]))

#vae.load_weights("./vae_weights6.h5")
vae.fit([all_image,x_noise],[all_image,y_noise],batch_size=1000,epochs=100)
vae.save_weights("./vae_weights6.h5")
nex = 5
pred = vae.predict([all_image[0:nex],x_noise[0:nex]])
print(pred[0].shape)
print(pred[1].shape)

pred_enc = encoder.predict(all_image[0:nex])

rx  = numpy.random.randn(nex,num_hide)
inp_dec = pred_enc[0]+pred_enc[1]*rx*1
pred_dec = decoder.predict(inp_dec)

encoder.save("./encoder7.h5")
decoder.save("./decoder7.h5")
fig = plt.figure(figsize= (7,7))
ax = [[],[],[],[]]
for i in range(4):
   ax[i] = fig.add_subplot(2,2,i+1)
   ax[i].imshow(((pred_dec[i][:,:,0]+1)/2))
plt.show()
```

3.3 Задание

Реализуйте вариационный автоэнкодер и проверьте его работу с различными параметрами. Попробуйте изменить активационные функции, количество фильтров и слоев, проведите исследования для, по крайней мере, трех вариантов сетей и оцените результаты работы.

4. ЛАБОРАТОРНАЯ РАБОТА №4. ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНЫЕ СЕТИ (GAN)

GAN – генеративно-состязательная нейросеть (Generative adversarial network, GAN) – один из алгоритмов классического машинного обучения, обучения без учителя. Суть идеи в комбинации двух нейросетей, при которой одновременно работает два алгоритма (две модели сети) “генератор” и “дискриминатор”. Задача генератора – генерировать образы заданной категории (генерировать так называемый «фейковый» образ). Задача дискриминатора – пытаться распознать созданный образ (или распознать «фейк» или подлинное изображение). Так как сети G и D имеют противоположные цели — создать образцы и отбраковать образцы, то между ними возникает антагонистическая игра.

В качестве дискриминатора и решения задачи распознавания используются чаще всего сверточные нейронные сети (CNN). Дискриминатор — это обычный бинарный классификатор, который выдает значение близкое к 1 если объект распознан или близкое 0 в ином случае.

В качестве генератора используется сеть деконволюции. Формирование изображений начинается с генерации произвольного шума, на котором постепенно начинают проступать фрагменты искомого изображения, при этом изначальный вектор постепенно путем использования слоев деконволюции и изменения формы превращается в матрицу, обычно из трех цветовых карт.

Впервые GANы были предложены в статье Яна Гудфеллоу из компании Google в 2014 году [1]. Высокоуровнево эта модель может быть описана, как две подмодели, которые соревнуются друг с другом, и одна из этих моделей (генератор), пытается научиться в некотором смысле обманывать вторую (дискриминатор). Для этого генератор генерирует случайные объекты, а дискриминатор пытается отличить эти сгенерированные объекты от настоящих объектов из тренировочной выборки. В процессе обучения генератор генерирует все более похожие на выборку объекты и дискриминатору становится все сложнее отличить их от настоящих. Таким образом, генератор превращается в генеративную модель, которая генерирует объекты из некого сложного распределения, например, из распределения фотографий человеческих лиц.

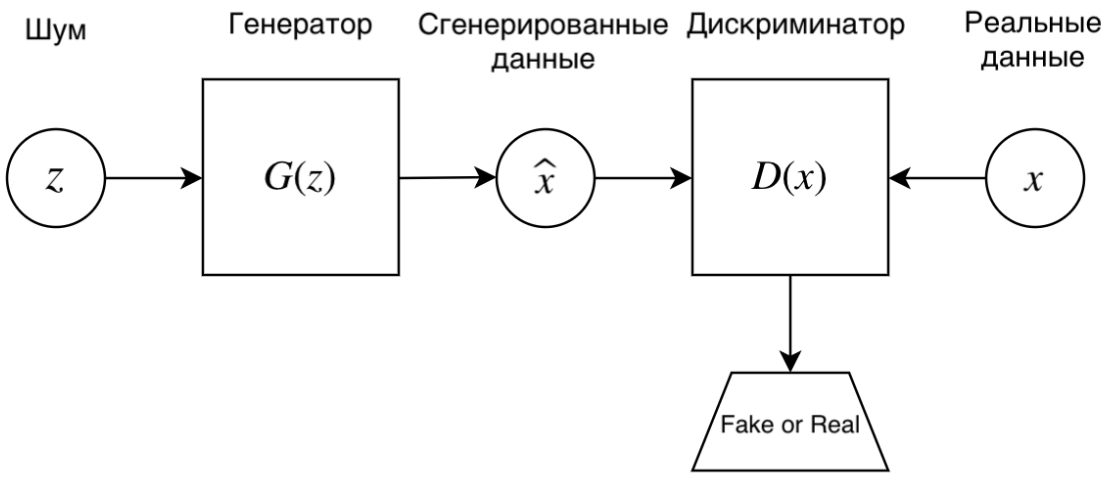


Рисунок 4.1 - Оригинальная архитектура GAN

Существует очень много различных видов GAN сетей, которые представлены на рисунке 4.2.

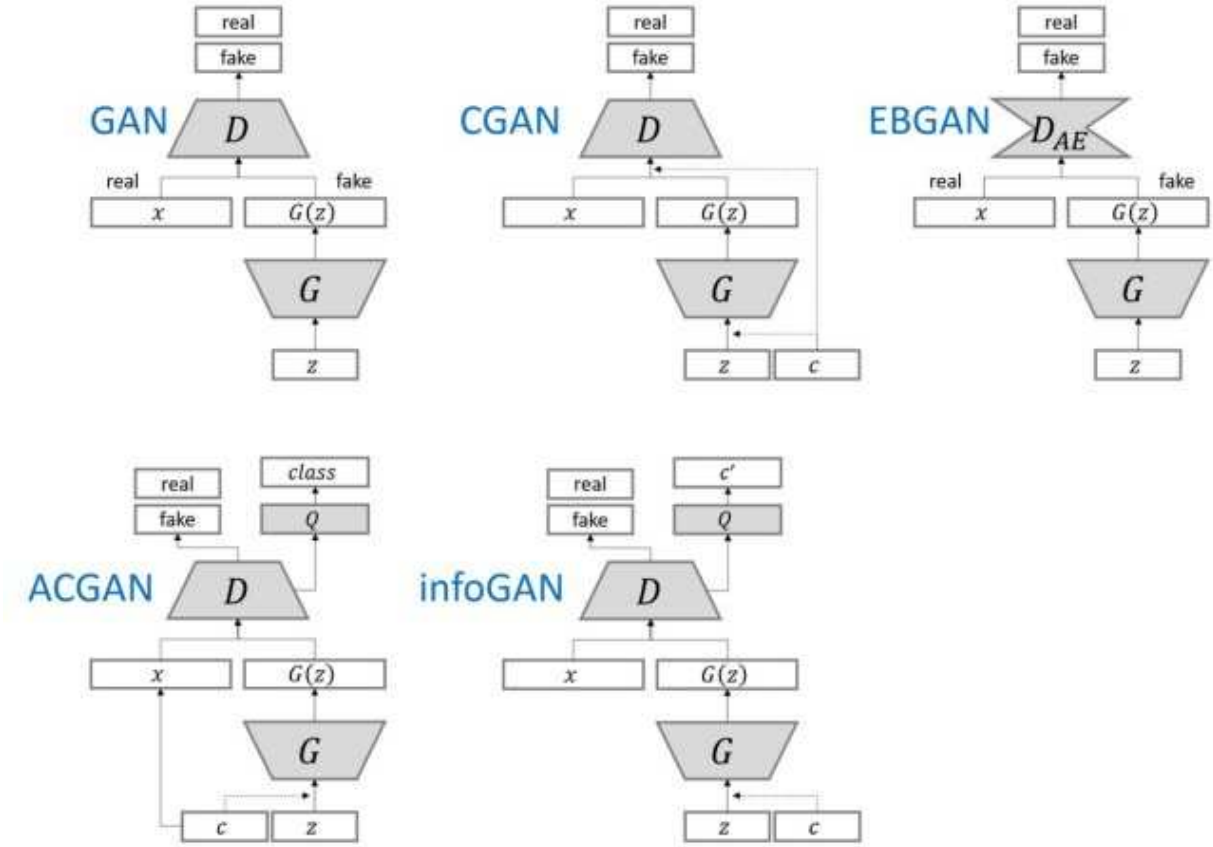


Рисунок 4.2 – Различные структуры GAN

- c — это класс.
- z — шум, подаваемый на генератор.
- x — реальные данные, подаваемые для обучения дискриминатора.
- G — генератор.
- D — дискриминатор (критик).

Ниже приводится рисунок, на котором изображены результаты работы такой нейронной сети при генерации человеческих лиц, то есть в реальности этих людей не существует (рисунок 4.3).

Рисунок 4.3 - Примеры изображений, созданных GAN

Все эти изображения сгенерированы системой на базе генеративно-состязательных нейронных сетей, часть из них выглядит не слишком реалистично, но другая часть весьма правдоподобна.

GAN – довольно молодой метод (он появился только в 2014 году, а первые “приличные” результаты появились в 2016-2017 годах), но, тем не менее, на момент 2023 года уже существует множество ресурсов генерирующих весьма реалистичные изображения. Обзорам новых вариаций GAN и способам их применения посвящено много статей.

Основные проблемы GAN алгоритмов:

- Схлопывание мод распределения (англ. mode collapse): генератор коллапсирует, то есть выдает ограниченное количество разных образцов.

- Проблема стабильности обучения (англ. non-convergence): параметры модели дестабилизируются и не сходятся.
- Исчезающий градиент (англ. diminished gradient): дискриминатор становится слишком "сильным", а градиент генератора исчезает и обучение не происходит.
- Проблема запутывания (англ. disentanglement problem): выявление корреляции в признаках, не связанных (слабо связанных) в реальном мире.
- Высокая чувствительность к гиперпараметрам.

Существуют различные виды GAN, которые мы рассмотрим далее, в том числе и более подробно.

C-GAN (генеративные состязательные сети с условием, condition).

В данной модификации генератор и дискриминатор получают не только данные в виде картинки/файла, но и описание класса к которому они принадлежат. То есть при обучении дискриминатор будет оперировать не только собственными наблюдениями, но и приходящими с данными обозначениями классов к которым эти данные возможно относятся. Генератор же выступает в роли художника в шумном месте. То есть его задача состоит не только в том, чтобы нарисовать максимально похожее на запрошенное, но и понять, что именно запрашивали.

Тем самым данная модификация позволяет тренировать устойчивые к помехам сети. Данные тренированные сети используются впоследствии в качестве телефонных роботов, шумоподавления и прочих сферах, где влияние сторонних шумов высоко.

Вассерштейн ГАН (в дальнейшем именуемый **WGA**N), успешно добился следующих прорывов:

Полностью решить проблему нестабильности обучения GAN, больше не нужно тщательно балансировать уровни обучения генератора и дискриминатора.

В основном решаем проблему режима коллапса и обеспечиваем разнообразие создаваемых образцов.

В процессе обучения, наконец, есть значение, такое как кросс-энтропия и точность, чтобы регулировать ход обучения. Чем меньше значение, тем лучше обучение GAN и тем выше качество изображения, генерируемого генератором.

Правда в своем чистом варианте обучение WGAN требует своей стабилизации иначе появляется расходимость процесса обучения.

Для стабилизации и улучшения GAN и WGAN существует несколько работ, основные из них это "Towards Principled Methods for Training Generative Adversarial Networks, в которой выдвинуто группа теорем, формул и теоретически проанализированы проблемы исходного GAN и даны основные методики улучшения, а во второй статье Wasserstein GAN предложено так же множество методик улучшения.

Хотя GAN Вассерштейна (WGAN) способен стабильно обучаться, но иногда может генерировать только выборки низкого качества или не сходиться. Обнаружено, что эти проблемы часто возникают из-за использования отсечения веса в WGAN для принудительного ограничения Липшица, что может привести к нежелательному поведению. Предложена альтернатива отсечению весов: штрафовать норму градиента критика по отношению к его входу. Предлагаемый метод работает лучше, чем стандартная WGAN, и обеспечивает стабильное обучение самых разных архитектур GAN практически без настройки гиперпараметров, включая 101-уровневые сети ResNet и языковые модели на дискретных данных. Данный метод улучшения получил название «*Градиентное пенальти*» (англ. gradient penalty). Его использование позволяет снизить вероятность доминирования дискриминатора сети на порядок тем самым частично решив проблему потери градиента. Проблема исчезающего градиента (англ. diminished gradient)

Квадратичная регуляризация: для ограничения роста весов к ним добавляется штрафное слагаемое тем самым приводя к постоянному уменьшению весов, что решает проблему переобучения. Однако у данного решения есть своя основная проблема: рассчитывать штрафное слагаемое приходится на каждом шагу, что приводит к повышенной вычислительной сложности алгоритма и, следовательно, к большим затратам времени.

Выбивание из локальных минимумов: при каждой стабилизации алгоритма в вектор весов вносятся случайные модификации в широком диапазоне для принудительного «выбивания» из полученной стабильной позиции. Данный метод является хорошим вариантом страховки от застревания в локальных экстремумах, но он же не позволяет завершить поиск при получении определенной точности, что требует иных способов определения завершения работы.

4.1 Модель GAN

Пусть X некоторое пространство объектов. Например, картинки 64*64*3 пикселя. На некотором вероятностном пространстве Ω задана векторная случайная величина $x:\Omega \to X$ с

распределением вероятностей, имеющим ненулевую плотность вероятности $p_{data}(x)$ появления, допустим, человеческого лица. Пусть дана случайная выборка фотографий лиц для величины $\{x_i, i\epsilon[1,N], x_i \sim p(x)\}$. Дополнительно определим вспомогательное пространство $Z = R^n$ и случайную величину $z: \phi \to Z$ с распределением вероятностей, имеющим плотность $p_z(z)$.

$D: X \to (0,1)$— функция-дискриминатор. Эта функция принимает на вход объект $x\epsilon X$ (в нашем примере — картинку соответствующего размера) и возвращает вероятность того, что входная картинка является фотографией человеческого лица.

$G: Z \to X$ — функция-генератор. Она принимает значение $z\epsilon Z$ и выдает объект пространства X, то есть, в нашем случае, картинку (обычно z представляет собой вектор нормально распределенных величин с нулевым средним и единичной дисперсией).

Процесс обучения похож на игру минимум-максимум для двух игроков со следующей целевой функцией (формула 4.1):

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}\big[\log\left(1 - D\big(G(z)\big)\right)\big], \quad (4.1)$$

где x – реальное изображение из истинного распределения данных p_{data};

z – вектор шума, выбранный из распределения p_z (например, равномерное или гауссово распределение).

Фактически если попытаться описать указанную формулу то сначала максимизируется функция модели дискриминатора по его параметрам, так чтобы на реальных данных он выдавал значение близкое к 1, а на данных формируемых генератором близкое к 0, а затем минизируется функция модели дискриминатора по параметрам генератора, так чтобы функция дискриминатора давала значение близкое к 1 от образов генерируемых генератором.

Такая форма записи позволяет еще использовать в качестве функции потерь классическую бинарную кроссэнтропию реализованную во всех современных библиотеках обучения нейронных сетей.

4.2 Обучение GAN

Следующие шаги выполняются в прямом и обратном направлении, позволяя GAN справиться с проблемами генерации, которые иначе оказались бы неразрешимыми:

- Шаг 1. Выбираем несколько реальных изображений из тренировочного набора

- Шаг 2. Генерируем несколько фейковых изображений. Для этого мы создаем несколько случайных векторов шума и создаем из них изображения с помощью генератора.
- Шаг 3. Обучаем дискриминатор на протяжении одной или большего количества эпох, используя как реальные, так и фейковые изображения. При этом будут обновляться только веса дискриминатора, поскольку мы пометим все реальные изображения как 1, а фейковые как 0.
- Шаг 4. Создаем еще несколько фейковых изображений.
- Шаг 5. Обучаем полную модель GAN на протяжении одной или большего количества эпох, используя только фейковые изображения. При этом будут обновляться только веса генератора, а всем фейковым изображениям будет назначена метка 1.

Большинство GAN'ов подвержено следующим проблемам:

- Схлопывание мод распределения (англ. mode collapse): генератор коллапсирует, то есть выдает ограниченное количество разных образцов.
- Проблема стабильности обучения (англ. non-convergence): параметры модели дестабилизируются и не сходятся.
- Исчезающий градиент (англ. diminished gradient): дискриминатор становится слишком "сильным", а градиент генератора исчезает и обучение не происходит.
- Проблема запутывания (англ. disentanglement problem): выявление корреляции в признаках, не связанных (слабо связанных) в реальном мире.
- Высокая чувствительность к гиперпараметрам.

Не существует универсального подхода для решения этих проблем, но существует список рекомендаций, которые могут помочь при обучении GAN:

- Нормализация данных. Все признаки в диапазоне $[-1;1]$;
- Сэмплирование из многомерного нормального распределения вместо равномерного;
- Использовать нормализационные слои (например, batch normalization или layer normalization) в G и D;
- Использовать метки для данных, если они имеются, то есть обучать дискриминатор еще и классифицировать образцы.

Способы решения проблемы схлопывания мод распределения

В процессе обучения генератор может прийти к состоянию, при котором он будет всегда выдавать ограниченный набор выходов. При этом пространство, в котором распределены сгенерированные изображения, окажется существенно меньше, чем пространство исходных изображений. Главная причина этого в том, что генератор обучается обманывать дискриминатор, а не воспроизводить исходное распределение. Если генератор начинает каждый раз выдавать похожий выход, который является максимально правдоподобным для текущего дискриминатора, то зависимость от z падает, а следовательно, и градиент G(z) стремиться к 0. Лучшей стратегией для дискриминатора будет улучшение детектирования этого конкретного изображения. Так на следующих итерациях наиболее вероятно, что генератор придет к другому изображению, хорошо обманывающему текущий дискриминатор, а дискриминатор будет учиться отличать конкретно это новое изображение. Этот процесс не будет сходиться и количество представленных мод не будет расти, поэтому приблизиться к исходному распределению не удастся.

На текущий момент mode collapse является одной из главных проблем GAN, эффективное решение которой ещё ищется. Возможные решения проблемы mode collapse:

- WGAN — использование метрики Вассерштейна (англ. Wasserstein Loss) внутри функции ошибки, позволяет дискриминатору быстрее обучаться выявлять повторяющиеся выходы, на которых стабилизируется генератор;
- UGAN (Unrolled GAN) — для генератора используется функция потерь, которая не только от того, как текущий дискриминатор оценивает выходы генератора, но и от выходов будущих версий дискриминатора.

Способы решения проблемы стабильности обучения

Задача обучения дискриминатора и генератора в общем смысле не является задачей поиска локального или глобального минимума функции, а является задачей поиска точки равновесия двух игроков. В теории игр эта точка называется точкой равновесия Нэша (англ. Nash equilibrium) в которой оба игрока больше не получают выгоды, хотя следуют оптимальной стратегии. Рассмотрим задачу поиска этой точки на игрушечном примере, где G хочет максимизировать произведение

xy, а D — минимизировать. Будем обновлять параметры x и y на основе градиентного спуска: $\Delta x = \alpha \frac{\delta\,(x*y)}{\delta(x)}\ \Delta y = \alpha \frac{\delta\,(x*y)}{\delta(y)}$. Если изобразить на графике (рисунок 4.3) поведение x,y

и xy то станет ясно, что они не сойдутся, а амплитуда их движения будет только увеличиваться.

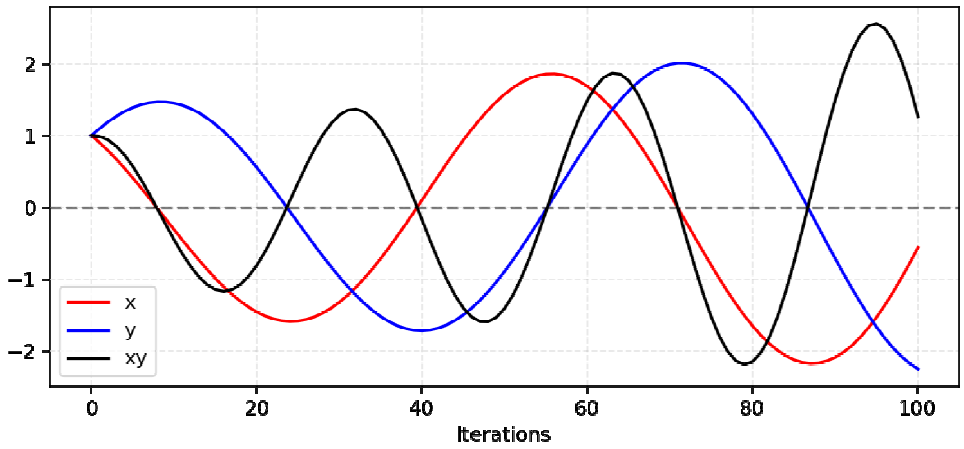


Рисунок 4.3 - Симуляция изменения x, y и xy

Возможные решения проблемы стабильности:

- регуляризация — добавление шума ко входам дискриминатора и соответствующая настройка гиперпараметров дискриминатора;
- PGGAN (Progressive Growing of GANs) — в процессе работы разрешение изображений увеличивается от очень малых (4 на 4 пикселя), до конечных (1024 на 1024 пикселя), что позволяет тренировать сначала выявление крупных черт, а затем более мелких, что крайне положительно сказывается на стабильности;
- WGAN — В качестве функции дивергенции используется метрика Вассерштейна, которая в большинстве случаев решает проблему расходимости интеграла в функции Дженсена-Шеннона.

4.3. Рассмотрение видов GAN

4.3.1 CGAN

Условные порождающие состязательные сети (англ. Conditional Generative Adversarial Nets, CGAN) − это модифицированная версия алгоритма GAN, которая может быть сконструирована при помощи передачи дополнительных данных ***y***, являющихся условием для генератора и дискриминатора. ***y*** может быть любой дополнительной информацией, например, меткой класса, изображением или данными из других моделей, что может позволить контро-

лировать процесс генерации данных. Например, можно подавать параметр ***y***, как условие на класс для генерации чисел, похожих на MNIST. Создание таких картинок, в случае передачи картинки в качестве ***y*** является задачей трансляции изображений.

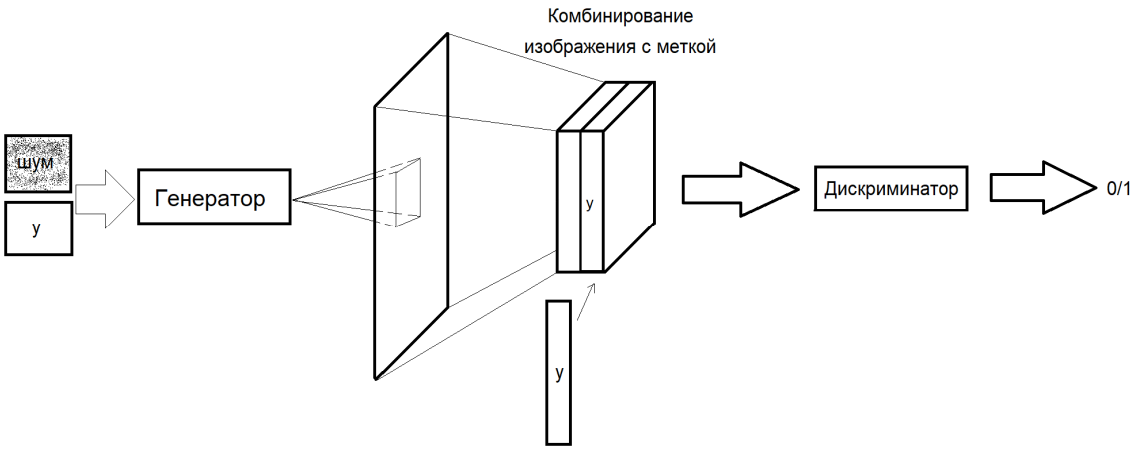


Рисунок 4.4 - Генерация при использовании CGAN

4.3.2 DCGAN

DCGAN (Deep Convolutional Generative Adversarial Nets) − модификация алгоритма GAN, в основе которых лежат сверточные нейронные сети (CNN). Задача поиска удобного представления признаков на больших объемах не размеченных данных является одной из наиболее активных сфер исследований, в частности представление изображений и видео. Одним из удобных способов поиска представлений может быть DCGAN (рисунок 4.4). Использование сверточных нейронных сетей напрямую не давало хороших результатов, поэтому было внесены ограничения на слои сверток. Эти ограничения и лежат в основе DCGAN:

- замена всех пулинговых слоев на страйдинговые свертки (strided convolutions) в дискриминаторе и частично-страйдинговые свертки (fractional-strided-convolutions) в генераторе, что позволяет сетям находить подходящие понижения и повышения размерностей;
- использование батчинговой нормализации для генератора и дискриминатора, то есть нормализация входа так, чтобы среднее значения было равно нулю, и дисперсия была равна единице. Не стоит использовать батч-нормализация для выходного слоя генератора и входного дискриминатор;
- удаление всех полносвязных скрытых уровней для более глубоких архитектур;

- использование ReLU в качестве функции активации в генераторе для всех слоев, кроме последнего, где используется tanh;
- использование LeakyReLU в качестве функции активации в дискриминаторе для всех слоев.

Помимо задачи генерации объектов, данный алгоритм хорошо показывает себя в извлечении признаков. Данный алгоритм был натренирован на наборе данных Imagenet-1k, после чего были использованы значения со сверточных слоев дискриминатора, подвергнутые max-pooling'y, чтобы образовать матрицы 4×4 и получить общий вектор признаков на их основе. L2-SVM, с полученным представлением, на наборе данных CIFAR-10 превосходит по точности решения, основанные на алгоритме K-Means.

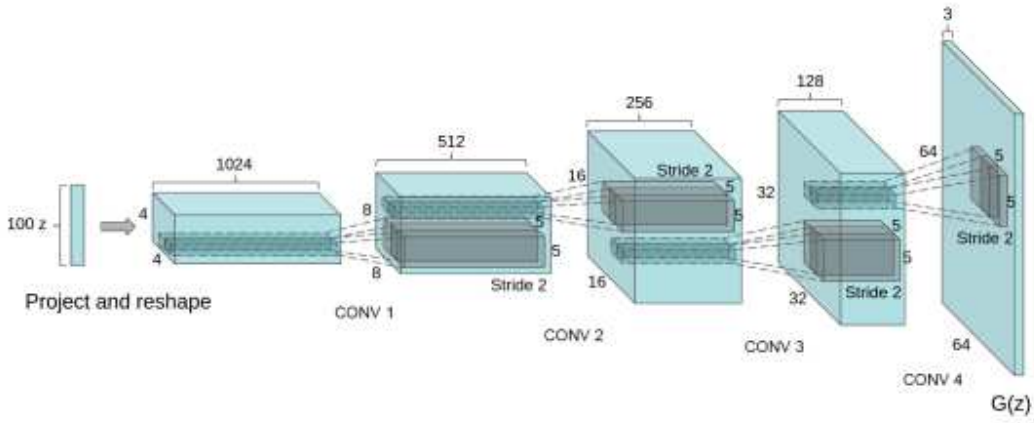


Рисунок 4.4 - Архитектура генератора в DCGAN

4.3.3 StackGAN

StackGAN (Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks) − порождающая состязательная сеть для генерации фотореалистичных изображений (256x256) исходя из текстового описания. Генерировать фотореалистичные изображения на обычных GAN сложно, поэтому была придумана двухэтапная модель генерации. Stage-I GAN рисует скетчи с примитивными формами и цветами, основанные на текстовом описании, в низком разрешении. Stage-II GAN принимает на вход изображения с первого этапа и текстовое описание и генерирует изображение в высоком разрешении с фотореалистичными деталями. Чтобы улучшить разнообразие синтезированных изображений и стабилизировать обучение, вместо CGAN использовался метод Conditioning Augmentation.

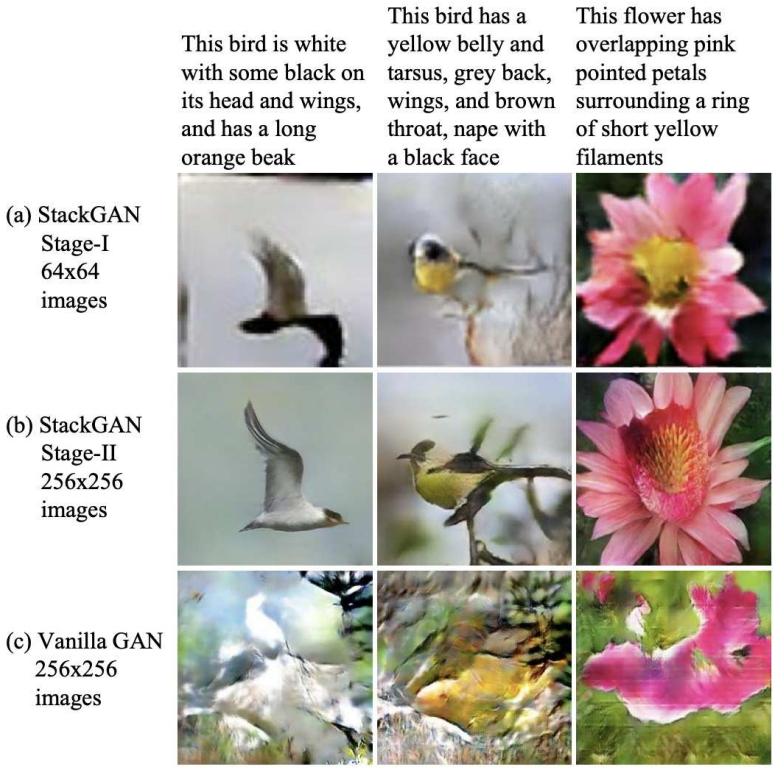


Рисунок 4.5 - Работа StackGAN

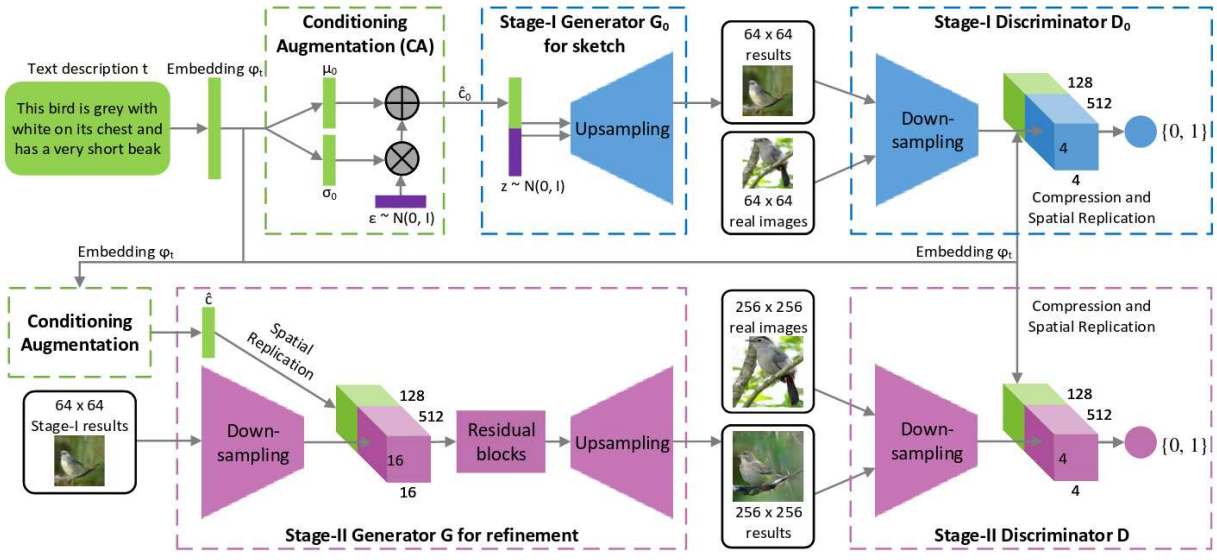


Рисунок 4.6 – Архитектура StackGAN

Архитектура модели StackGAN состоит из следующих компонентов:

1) Embedding: преобразует входной текст переменной длины в вектор фиксированной длины;

2) Conditioning Augmentation (CA);

3) Stage 1 Generator: Генерирует изображения с низким разрешением (64×64);

4) Stage 1 Discriminator;

5) Residual Blocks;

6) Stage 2 Generator: Генерирует изображения с высоким разрешением (256×256);

7) Stage 2 Discriminator.

Embedding

При подаче данных в нейронную сеть необходимо сопоставлять все слова с некоторыми конкретными числами, поскольку нейронные сети не могут понимать язык обычных людей.

Встраивание слов – это метод представления слова с помощью вектора чисел. Проще говоря, встраивание слов означает текст в виде чисел.

Conditioning Augmentation

Текстовое описание t сначала кодируется кодировщиком, что дает вложение текста φ_t. В предыдущих работах встраивание текста нелинейно преобразовывалось, чтобы генерировать скрытые обусловливающие переменные в качестве входных данных генератора. Однако скрытое пространство для встраивания текста обычно имеет более высокие размерности (более 100). Обычно это вызывает неравномерность множества скрытых данных, что невыгодно для обучения генератора. Чтобы решить эту проблему, был введен метод conditioning augmentation (блок условного дополнения) для создания дополнительных conditioning переменных $\hat{c}$. Conditioning Augmentation дает больше обучающих пар при небольшом количестве пар изображение-текст и, таким образом, способствует устойчивости к небольшим возмущениям вдоль многообразия conditioning.

Stage-1

Работа упрощается, благодаря первоначальному созданию изображения с низким разрешением с помощью Stage-1 GAN, которая фокусируется на рисовании только грубой формы и правильных цветов для объекта, а не на непосредственном создании изображения с высоким разрешением на основе текстового описания.

Stage-2

На изображениях Stage-1 с низким разрешением иногда отсутствуют яркие элементы объектов и могут быть искажения формы. На первом этапе могут быть удалены некоторые текстовые детали, которые очень важны для создания фотореалистичной графики. Для получения изображений с высоким разрешением Stage-2 основана на результатах Stage-1. Чтобы исправить ошибки в результатах Stage-1, он основан на фотографиях с низким разрешением, а также на встраивании текста. Stage-2 заполняет пробелы в текстовых данных, которые ранее игнорировались, что приводит к более фотореалистичным функциям.

Residual Blocks

Residual block – это набор слоев, в котором выходные данные одного слоя берутся и добавляются к слою, находящемуся глубже в блоке. После этого нелинейность применяется путем объединения ее с выходом соответствующего слоя на основном пути.

4.3.4 LAPGAN

LAPGAN (Laplacian Pyramid of Adversarial Networks) − генеративная параметрическая модель, представленная пирамидой лапласианов с каскадом сверточных нейронных сетей внутри, которая генерирует изображения постепенно от исходного изображения с низким разрешением к изображению с высоким. На каждом уровне пирамиды обучается сверточная генеративная модель, используя подход порождающих состязательных сетей. Такая стратегия позволяет декомпозировать задачу генерации изображений на последовательность уровней, что упрощает ее решение.

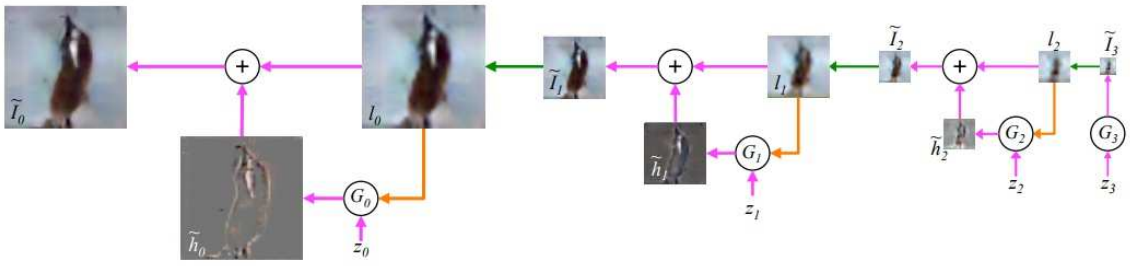


Рисунок 4.7 - Процедура сэмплинга для модели LAPGAN

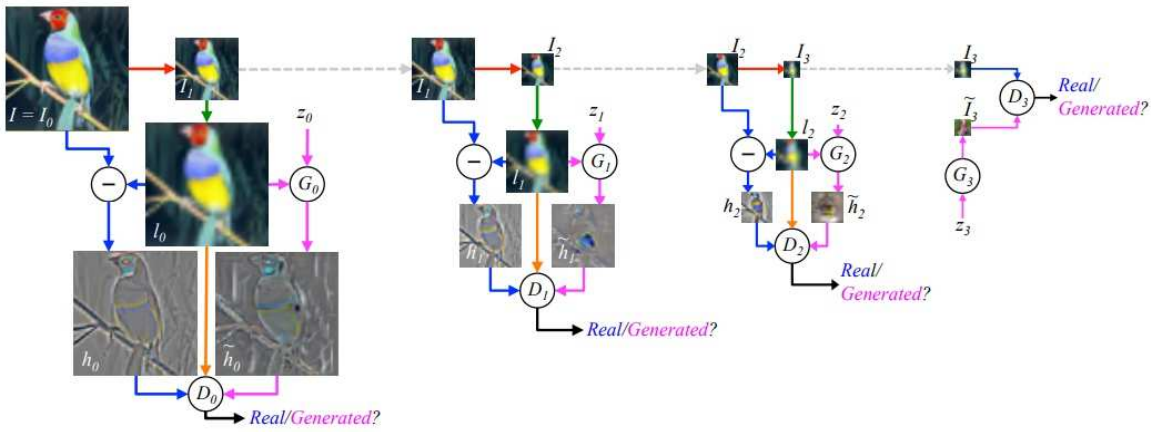


Рисунок 4.8 - Процедура обучения модели LAPGAN

4.3.5 ControlGAN

Контролируемые порождающие состязательные сети (англ. Controllable Generative Adversarial Nets, ControlGAN) − модифицированная версия алгоритма GAN, состоящая из трех нейронных сетей: генератор, дискриминатор, классификатор. Концепт модели ControlGAN (рисунок 4.9). Как и в обычной версии алгоритма, генератор пытается обмануть дискриминатор, и одновременно с этим пытается быть классифицированным как нужный класс в классификаторе.

Хоть CGAN и являются самыми популярными моделями для генерации образцов, зависимых от внешних данных, но лучше они умеют генерировать образцы с заданными ярко отличительными чертами (цвет волос, веснушки), но менее явные детали (форма бровей, сережки) вызывают затруднения (Но более поздний StyleGAN2 справляется и с этой задачей). С помощью отделения классификатора от дискриминатора, ControlGAN позволяет контролировать черты образцов. К тому же и само качество сгенерированных изображений может быть улучшено за счет того, что такое разделение на три составляющие дает возможность дискриминатору лучше выполнять свою главную задачу.

Более того, аугментация данных может помешать некоторым сетям, например, Auxiliary Classifier GAN (ACGAN) обучаться, хотя сам способ может улучшить качество классификации. К тому же в случае контролируемой генерации нет необходимости размечать тренировочные данные, выбираются желаемые характеристики объектов для генерации, а не условная информация (например, метка объекта).

Иллюстрация принципа работы сети (рисунок 4.10). Зеленые линии − результат работы классификатора; оранжевые − дискриминатора. Серые фигуры − образцы из разных классов. Результат генератора обозначается голубыми участками, которыми он показывает распределение образцов, как и пытается быть классифицированным верно.

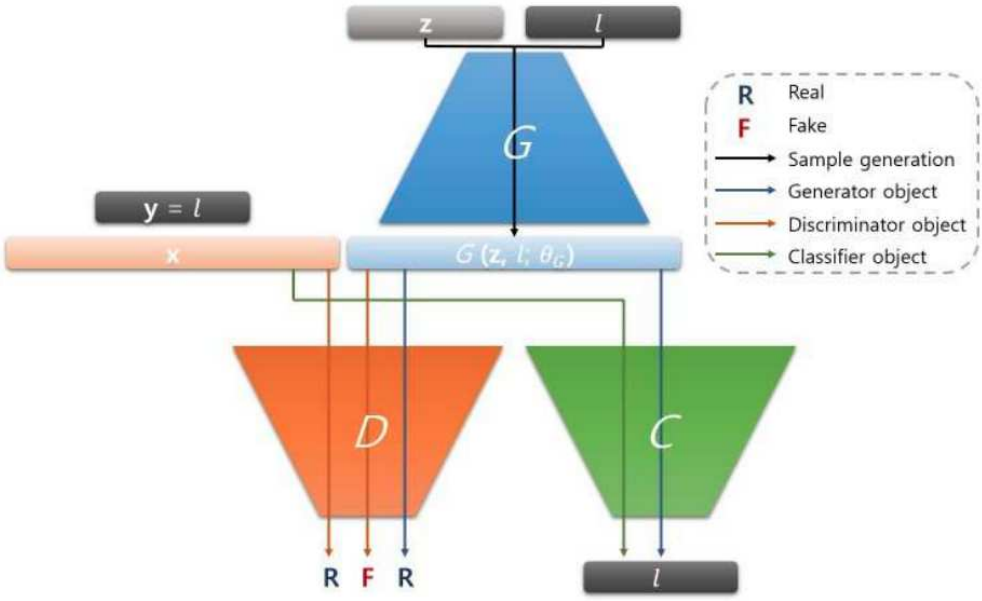


Рисунок 4.9 - Концепт модели ControlGAN

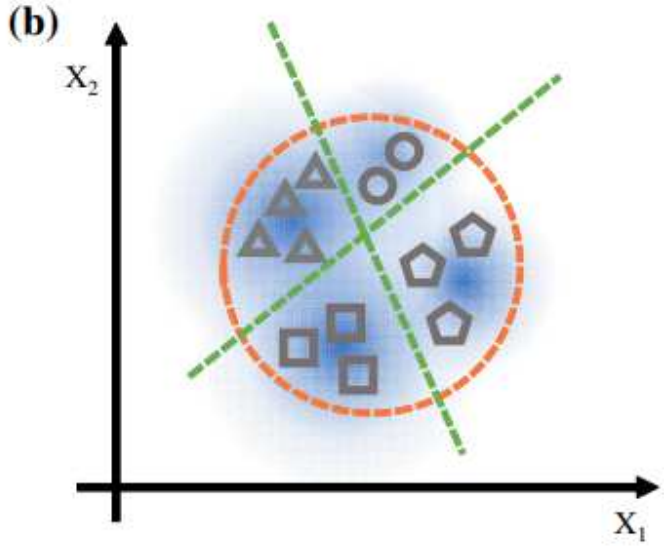


Рисунок 4.10 - Принцип работы ControlGAN

4.3.6 Super resolution GAN

Идея данной архитектуры заключается в восстановлении более подробных текстур из изображения при его увеличении, без потери качества. Существуют и другие методы, такие как билинейная интерполяция, которые могут быть использованы для выполнения этой задачи, но они страдают от возможных потерь информации с изображения и сглаживания изображения.

Архитектура

Подобно архитектуре GAN, Super Resolution GAN также состоит из двух частей - генератора и дискриминатора, где генератор производит некоторые данные на основе вероятностного распределения, а дискриминатор пытается определить откуда поступают данные, из входного набора данных или это данные, созданные генератором. Генератор пытается оптимизировать генерируемые данные таким образом, чтобы они могли обмануть дискриминатор. Ниже, на рисунке 1 приведена общая архитектура SRGAN:

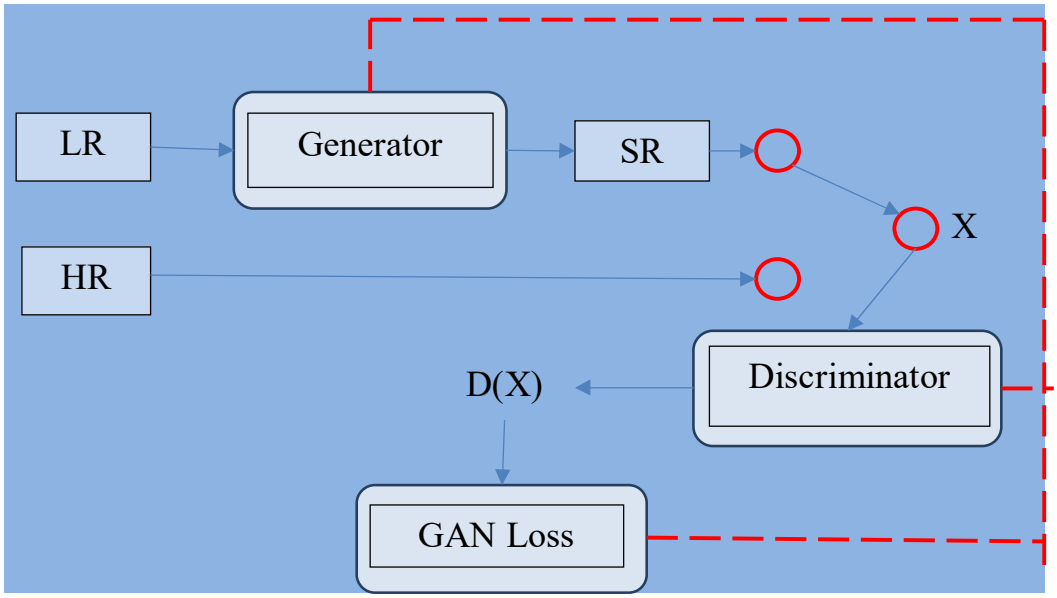


Рисунок 4.11 - Архитектура SRGAN

где LR - изображение с низким разрешением;

HR - изображение с высоким разрешением;

SR - сверхразрешение.

Архитектура генератора

Архитектура генератора содержит остаточную сеть вместо глубоких сверточных сетей, поскольку остаточные сети легче обучить и позволяют существенно углубить их для получения лучших результатов. Это происходит потому, что в остаточной сети используется тип соединений, называемый пропускными соединениями.

Архитектура генератора сети SRGAN состоит из входного сигнала низкого разрешения, который проходит через начальный сверточный слой из 9×9 ядер и 64 карт признаков, за которым следует слой Parametric ReLU. Заметно, что вся архитектура генератора использует Parametric ReLU в качестве основной функции активации. Причина выбора Parametric ReLU заключается в том, что это одна из лучших нелинейных функций для конкретной задачи составления карт изображений низкого разрешения для изображений высокого разрешения.

Активационные функции такие как ReLU, также могут выполнить данную задачу, но существуют проблемы, которые могут возникнуть из-за концепции мертвых нейронов, когда значения меньше нуля картируются непосредственно на ноль. Альтернативным вариантом является Leaky ReLU, где значения меньше нуля картируются на число, заданное пользователем. Однако в случае Parametric ReLU мы можем позволить нейронной сети самостоятельно выбрать наилучшее значение, поэтому для нашей задачи данная функция предпочтительнее.

Следующий слой полностью сверточной модели SRGAN с обратной связью, использует набор остаточных блоков. Каждый из остаточных блоков содержит сверточный слой с ядрами 3×3 и 64 картами признаков, затем слой групповой нормализации, функцию активации Parametric ReLU, еще один сверточный слой с групповой нормализацией и заключительный метод поэлементной суммы. Метод поэлементного суммирования использует выход прямой передачи вместе с выходом пропускающего соединения для получения окончательного результирующего выхода.

Ключевым аспектом, который следует отметить в архитектуре нейронной сети, является то, что каждый из сверточных слоев использует аналогичную подкладку, чтобы размер следующих входов и выходов не менялся. В отличие от других полностью сверточных сетей, таких как архитектура U-Net, зачастую в ней используются объединяющие слои для уменьшения размера изображения. Однако для нашей модели SRGAN это не требуется, поскольку размер изображения не нужно уменьшать.

Как только остаточные блоки были созданы, строится остальная часть модели генератора. В модели архитектуры генератора используется перестановка пикселей после 4-х крат-

ного апсемплинга сверточного слоя для получения изображений со сверхразрешением. Для перемешивания пикселей берутся значения из измерения канала и вставляются в измерения высоты и ширины. В этом случае высота и ширина умножаются на два, пока канал делится на два. Архитектура генератора изображена на рисунке 4.12.

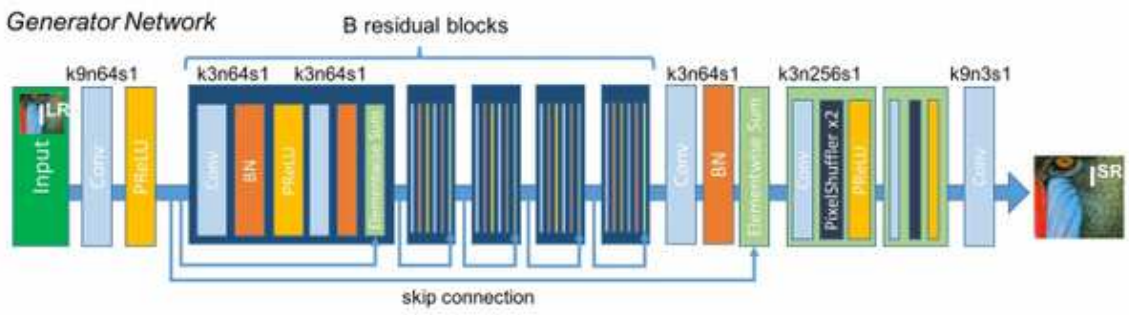


Рисунок 4.12 - Архитектура генератора

Архитектура дискриминатора

Архитектура дискриминатора построена для наилучшей поддержки типичной GAN. И генератор, и дискриминатор конкурируют друг с другом и одновременно совершенствуются. В то время как сеть дискриминатора пытается найти поддельные изображения, генератор пытается создать реалистичные изображения, чтобы избежать обнаружения со стороны дискриминатора. Работа дискриминатора в случае SRGAN также заключается в этом. Генеративная модель G пытается обмануть дифференцируемый дискриминатор D, который обучен отличать изображения со сверхразрешением от реальных.

Таким образом, архитектура дискриминатора, показанная на рисунке 4.13, работает для различения изображений со сверхразрешением и реальных изображений. Построенная модель дискриминатора направлена на решение состязательной задачи minmax. Общая формула этого уравнения может быть интерпретирована следующим образом (формула 4.2):

$$\min_{\theta g} \max_{\theta D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}\left[\log D_{\theta_D}(I^{HR})\right] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}\left[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{HR})))\right] \quad (4.2)$$

В дискриминаторе используется начальный сверточный слой, за которым следует функция активации Leaky ReLU. Затем у нас есть несколько повторяющихся блоков сверточных слоев, за которыми следует слой групповой нормализации и функция активации Leaky ReLU. После пяти таких повторяющихся блоков у нас есть плотные слои, за которыми следует сигмовидная функция активации для выполнения действия классификации. Начальный

стартовый размер свертки составляет 64 x 64, который умножается на 2 после каждого второго блока, пока мы не достигнем 8-кратного коэффициента масштабирования 512 x 512. Данная модель дискриминатора помогает генератору обучаться более эффективно и выдавать лучшие результаты.

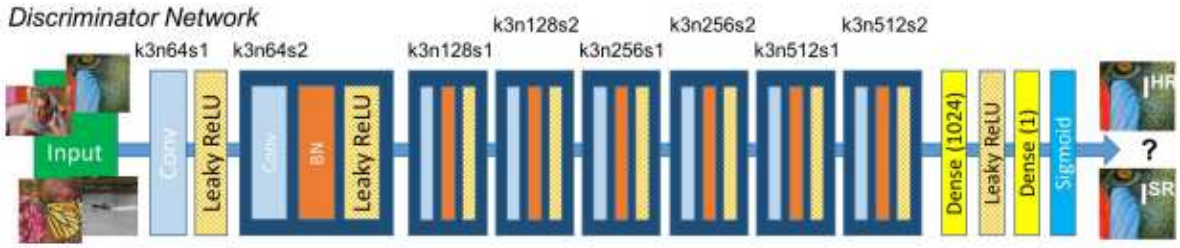


Рисунок 4.13 – Архитектура дискриминатора

Функции потерь

SRGAN использует бессрочную функцию потерь (L_{SR}), которая является взвешенной суммой двух компонентов потерь: потери содержания и состязательные потери. Эти потери являются очень важными для производительности генератора

Потери содержания (content loss)

Используется две функции потери содержания: пиксельная потеря MSE для архитектуры SRResNet, которая является наиболее распространенной функцией потерь для изображений со сверхразрешением (Simple Content Loss) (формула 4.3):

$$l_{MSE}^{SR} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2 \quad (4.3)$$

Однако простая функция потерь не может справиться с высокочастотным содержанием в изображении, что приводит к получению слишком гладких изображений. Поэтому следует использовать функцию потерь различных слоев VGG. Данная функция VGG основана на активации слоев ReLU предварительно обученной 19-слойной сети VGG. Функция потерб VGG (VGG content loss):

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}(\varphi_{i,j}(I^{HR})_{x,y} - \varphi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \quad (4.4)$$

Состязательные потери

Состязательные потери (Adversarial Loss) - это функция потерь, которая заставляет генератор создавать более похожие изображения на изображения высокого разрешения, используя дискриминатор, который обучен различать изображения высокого и сверхвысокого разрешения (формула 4.5):

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -logD_{\theta_D}(G_{\theta_G}(I^{LR})) \quad (4.5)$$

Общая потеря содержания данной архитектуры (Perceptual loss) (формула 4.6):

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR} \quad (4.6)$$

где l_X^{SR} - функция потерь содержательной части,

$10^{-3} l_{Gen}^{SR}$ - функция потерь для состязательной части GAN.

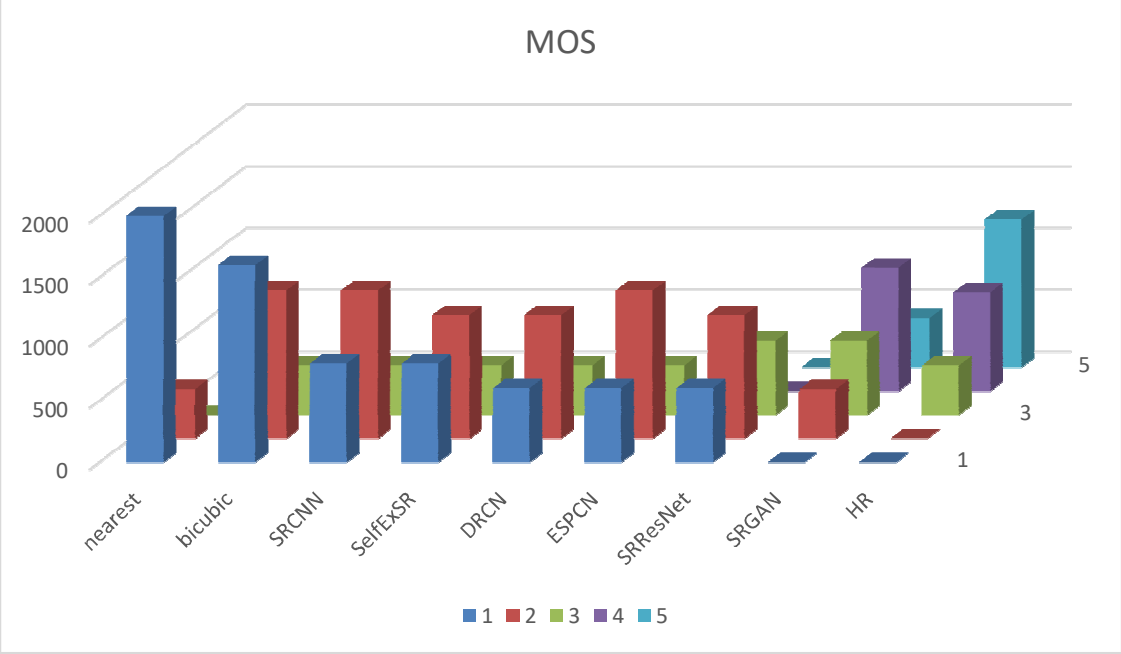


Рисунок 4.14 – Сравнение различных методов повышения разрешения

Глядя на результаты экспериментов MOS, оценивающие среднее значение видно, что SRGAN смогла получить отличные результаты на трех наборах данных.

Построение архитектуры генератора SRGAN

Архитектура генератора SRGAN построена в точности так, как подробно обсуждалось в предыдущем разделе. Архитектура модели разделена на несколько функций. Определяется блок перестановки пикселей и соответствующая функция, которая будет апсемплировать наши данные вместе со слоем перестановки пикселей. Далее определяется еще одна функция для остаточных блоков, содержащих непрерывную комбинацию сверточного слоя 3×3 ядер и

64 карт признаков, затем слой групповой нормализации, функцию активации Parametric ReLU, еще один сверточный слой с групповой нормализацией и заключительный метод по-элементной суммы, который использует соединение feed-forward и skip.

```
upsamples_per_scale = {
   2: 1,
   4: 2,
   8: 3
}
pretrained_srresnet_models = {
   "srresnet_bicubic_x4": {
      "url": "https://image-super-resolution-weights.s3.af-south-
1.amazonaws.com/srresnet_bicubic_x4/generator.h5",
      "scale": 4
   }
}
def pixel_shuffle(scale):
   return lambda x: tf.nn.depth_to_space(x, scale)

def upsample(x_in, num_filters):
   x = Conv2D(num_filters, kernel_size=3, padding='same')(x_in)
   x = Lambda(pixel_shuffle(scale=2))(x)
   return PReLU(shared_axes=[1, 2])(x)

def residual_block(block_input, num_filters, momentum=0.8):
   x = Conv2D(num_filters, kernel_size=3, padding='same')(block_input)
   x = BatchNormalization(momentum=momentum)(x)
   x = PReLU(shared_axes=[1, 2])(x)
   x = Conv2D(num_filters, kernel_size=3, padding='same')(x)
   x = BatchNormalization(momentum=momentum)(x)
   x = Add()([block_input, x])
   return x

def build_srresnet(scale=4, num_filters=64, num_res_blocks=16):
   if scale not in upsamples_per_scale:
      raise ValueError(f"available scales are: {upsamples_per_scale.keys()}")

   num_upsamples = upsamples_per_scale[scale]

   lr = Input(shape=(None, None, 3))
   x = Lambda(normalize_01)(lr)

   x = Conv2D(num_filters, kernel_size=9, padding='same')(x)
   x = x_1 = PReLU(shared_axes=[1, 2])(x)

   for _ in range(num_res_blocks):
      x = residual_block(x, num_filters)
```

```
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x)
    x = BatchNormalization()(x)
    x = Add()([x_1, x])

    for _ in range(num_upsamples):
        x = upsample(x, num_filters * 4)

    x = Conv2D(3, kernel_size=9, padding='same', activation='tanh')(x)
    sr = Lambda(denormalize_m11)(x)

    return Model(lr, sr)
```

Построение модели дискриминатора и архитектуры SRGAN

Архитектура дискриминатора строится точно так же, как было разобрано в предыдущем разделе. Используются сверточные слои с функцией активации Leaky ReLU, которая использует альфа-значение 0,2. Добавляется сверточный слой и функция активации Leaky ReLU для первого блока. В остальных пяти блоках архитектуры дискриминатора используются сверточные слои, затем слой групповой нормализации и, наконец, добавляется слой функции активации Leaky ReLU. Последними слоями архитектуры являются полностью связанные узлы с 1024 параметрами, слой Leaky ReLU и последний полносвязный узел с сигмовидной функцией активации для классификации.

```
def discriminator_block(x_in, num_filters, strides=1, batchnorm=True, momentum=0.8):
    x = Conv2D(num_filters, kernel_size=3, strides=strides, padding='same')(x_in)
    if batchnorm:
        x = BatchNormalization(momentum=momentum)(x)
    return LeakyReLU(alpha=0.2)(x)

def build_discriminator(hr_crop_size):
    x_in = Input(shape=(hr_crop_size, hr_crop_size, 3))
    x = Lambda(normalize_m11)(x_in)

    x = discriminator_block(x, 64, batchnorm=False)
    x = discriminator_block(x, 64, strides=2)

    x = discriminator_block(x, 128)
    x = discriminator_block(x, 128, strides=2)

    x = discriminator_block(x, 256)
    x = discriminator_block(x, 256, strides=2)
    x = discriminator_block(x, 512)
    x = discriminator_block(x, 512, strides=2)
```

```
x = Flatten()(x)

x = Dense(1024)(x)
x = LeakyReLU(alpha=0.2)(x)
x = Dense(1, activation='sigmoid')(x)

return Model(x_in, x)
```

4.4 Метрика Вассерштейна

Расстояние Вассерштейна - это минимальная стоимость транспортировки массы при преобразовании распределения данных *д* к распространению данных *п*. Расстояние Вассерштейна для реального распределения данных P_r и сгенерированного распределения данных P_θ математически определяется как наибольшая нижняя граница (инфимум) для любого транспортного плана (т. е. стоимость самого дешевого плана) (формула 4.7):

$$W(P_r, P_\theta) = \inf_{\gamma \in \prod(P_r, P_\theta)} \sum_{x,y}^{\gamma} [||x - y||] \quad (4.7)$$

$W(P_r, P_\theta)$ обозначает множество всех совместных распределений *γ(x, y)*, маргиналами которых являются соответственно P_r и P_θ.

Вместо добавления шума Wasserstein GAN (WGAN) предлагает новую функцию стоимости, использующую расстояние Вассерштейна, которое имеет более плавный градиент повсюду. WGAN учится независимо от того, работает генератор или нет. На приведенной ниже диаграмме повторяется аналогичный график значения D(X) как для GAN, так и для WGAN. Для GAN (красная линия) он заполняет области с уменьшающимся или взрывным градиентом. Для WGAN (синяя линия) градиент везде более плавный и лучше учится, даже если генератор не дает хороших изображений, рисунок 4.15.

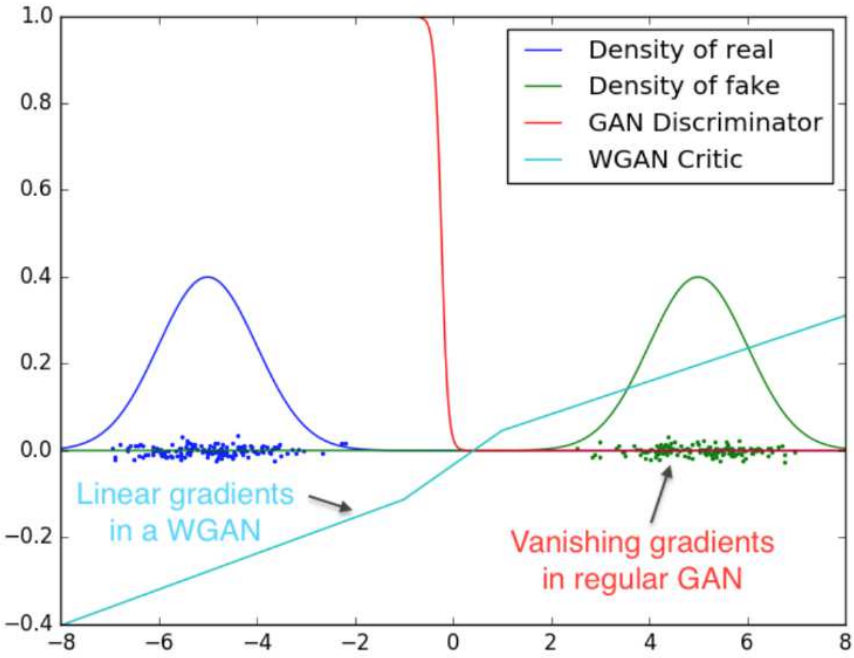


Рисунок 4.15 - Диаграмма GAN и WGAN

Уравнение для расстояния Вассерштейна достаточно сложное в решение. Используя двойственность Канторовича-Рубинштейна, можно упростить вычисление до (формула 4.8):

$$W(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \left(\sum_x^{P_r} [f(x)] - \sum_x^{P_\theta} [f(x)] \right) \quad (4.8)$$

где sup — наименьшая верхняя граница, а f — 1-липшицева функция.

В реузльтате, чтобы вычислить расстояние Вассерштейна, необходимо найти 1-липшицевую функцию. Как и в случае с другой проблемой глубокого обучения, можно построить глубокую сеть, чтобы изучить ее. Эта сеть очень похожа на дискриминатор *D*, только без сигмовидной функции и выводит скалярную оценку, а не вероятность. Эту оценку можно интерпретировать как то, насколько реальны входные изображения. В обучении с подкреплением такая функция называется функция ценности, которая измеряет, насколько хорошим является состояние (вход). Дискриминатор в WGAN переименован в Critic, чтобы отразить его новую роль. На рисунке 4.16 и 4.17 изображены GAN и WGAN.

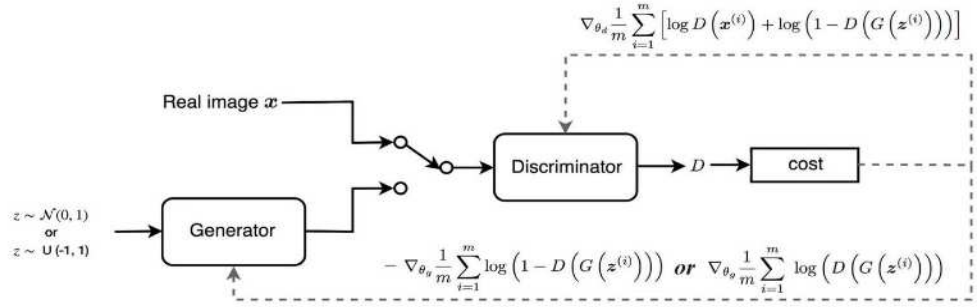


Рисунок 4.16 - GAN

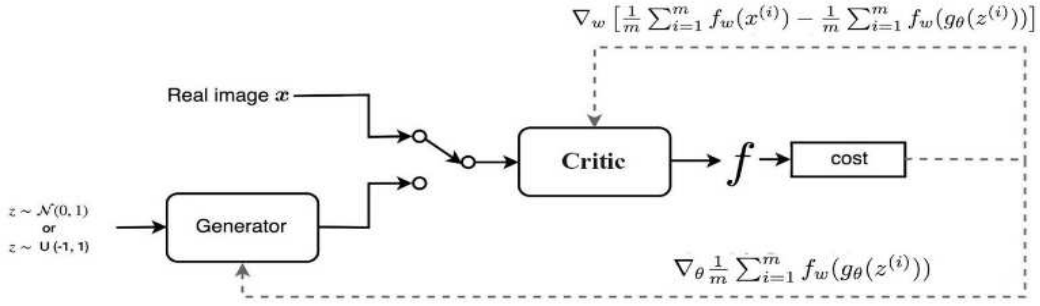


Рисунок 4.17 - WGAN

Схема сети WGAN почти такая же как GAN, за исключением того, что у Critic нет выходной сигмовидной функции. Основное различие заключается только в функции стоимости:

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G(z^{(i)})\right)\right)$
WGAN	$\nabla_{\omega} \frac{1}{m} \sum_{i=1}^{m} [f(x^{(i)}) - f(G(z^{(i)}))]$	$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^{m} f(G(z^{(i)}))$

Однако не хватает одной важной вещи. *f* должна быть 1-липшицевой функцией. Чтобы обеспечить соблюдение ограничения, WGAN применяет очень простое отсечение, чтобы ог-

раничить максимальное значение веса в *f*, то есть веса дискриминатора должны находиться в пределах определенного диапазона, контролируемого гиперпараметрами *c* (формулы 4.9, 4.10):

$$\omega \leftarrow \omega + \alpha * RMSProp(\omega, g_\omega) \tag{4.9}$$

$$\omega \leftarrow clip(\omega, -c, c) \tag{4.10}$$

DCGAN использует версию стохастического градиентного спуска Адама с небольшой скоростью обучения и скромным импульсом.

WGAN рекомендует вместо этого использовать RMSProp с небольшой скоростью обучения 0,00005.

4.6 Градиент Пенальти (Штраф градиента)

Идея Градиент Пенальти состоит в том, чтобы установить ограничение, чтобы градиенты выходных данных Crit модели (WGAN) относительно входных данных имели единичную норму

Так, по формуле 4.11:

$$L = \sum_{\tilde{x}}^{P_\theta}[f(\tilde{x})] - \sum_{x}^{P_r}[f(x)] + \lambda \sum_{\hat{x}}^{P_{\hat{x}}}[(\left|\left|\nabla_{\hat{x}} f(\hat{x})\right|\right|_2 - 1)^2] \tag{4.11}$$

члены слева от суммы представляют собой первоначальные Crit потери, а члены справа от суммы представляют собой штраф за градиент.

$P_{\hat{x}}$ — распределение, полученное путем равномерной выборки по прямой линии между реальным P_r и сгенерированным P_θраспределениями. Это сделано, чтобы оптимальная Crit имела прямые линии с единичной нормой градиента между выборками, связанными из реальной и сгенерированной выборки.

λ — коэффициент штрафа, используется для взвешивания штрафа за градиент.

4.7 Ограничение Липшица

Отсечение позволяет нам применить ограничение Липшица к модели Critic для вычисления расстояния Вассерштейна (формула 4.12):

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \tag{4.12}$$

Трудность в WGAN заключается в обеспечении ограничения Липшица. Обрезка проста, но создает некоторые проблемы. Модель может по-прежнему создавать изображения

низкого качества и не сходиться, в частности, когда гиперпараметр c настроен неправильно (формула 4.13):

$$\omega \leftarrow clip(\omega, -c, c) \tag{4.13}$$

Производительность модели очень чувствительна к этому гиперпараметру. На диаграмме ниже, когда пакетная нормализация отключена, дискриминатор переходит от уменьшающихся градиентов к взрывным градиентам, когда с увеличивается с 0,01 до 0,1.

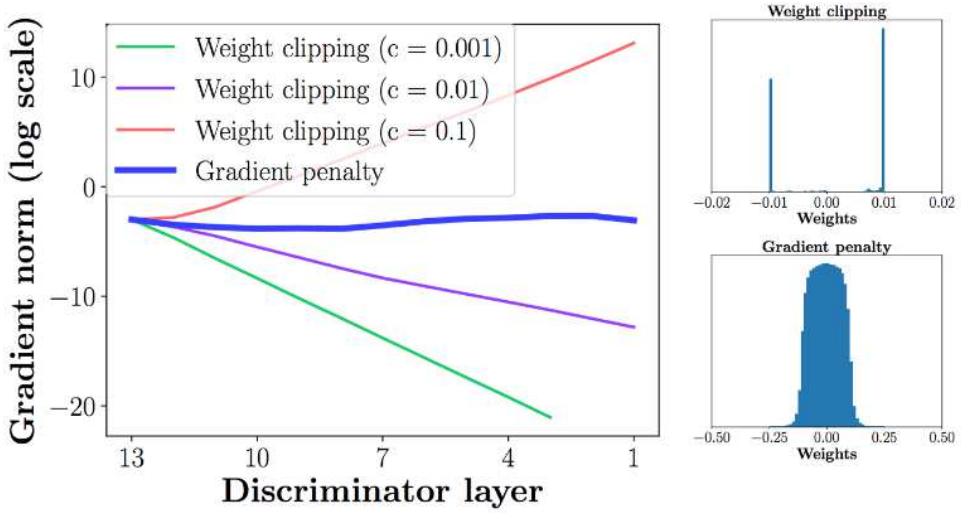


Рисунок 4.18 - Сравнение штрафа градиента и метода ограничения весов

4.8 Введение в задачу генерации изображений

Генерация изображений или объектов с заданными свойствами является отдельным направлением в интеллектуальных системах. Одним из подходов является генерация объектов на основе индуктивной модели обучения, когда присутствует набор уже существующих объектов или изображений, после чего по ним реализуется процедура поиска параметрического алгоритма (обучение) аппроксимирующего обратную функцию распределения данных объектов, либо функцию, позволяющую генерировать из пространства одного распределения, например нормального в искомую область распределения. Для аппроксимации подобных функций может быть использована глубокая нейронная сеть. Например, для задач генерации изображений часто используют так называемые вариационные автоэнкодеры [2], которые позволяют получать из области генерации векторов, распределенных нормально изображения из области распределения обучающей выборки. При этом базовая процедура обучения, использующая автоэнкодер предполагает получить сеть, в которой в промежуточном слое происхо-

дит сжатие исходного признакового пространства в скрытое признаковое пространство в виде вектора меньшего размера, нежели исходное изображение. При этом полученный вектор изначально может иметь различные вероятностные характеристики, для придания нужных ему свойств используют дополнительную функцию потерь на основе расхождения Кульбека-Лейблера [1]. Так же функция потерь автоэнкодера является квадратичной разностью двух изображений, исходного из обучающей выборки и восстанавливаемого декодером. Такая мера часто дает сглаженное изображение и человеком воспринимается как размытое, человек воспринимает изображения как естественные, если они более резкие, но для них порой мера квадратичной разности может быть и гораздо большей. Интуитивно это можно объяснить следующим образом, сдвинем одно изображение относительно другого всего лишь на один пиксель и получим большую меру расхождения, хотя изображение по-прежнему будет восприниматься человеком как четкое. Для создания реалистичного изображения в настоящее время применяется подход на основе генеративно-состязательных сетей GAN.

4.8.1 Генеративно-состязательные сети (GAN)

Генеративно-состязательные сети представляют, как последовательную минимаксную игру сети дискриминатора и генератора, в которой осуществляется поиск равновесия Нэша [3]. Данные сети чаще всего представляют собой глубокие нейронные сети со слоями свертки или обратной свертки (конволюции и деконволюции). Роль генератора сводится к генерации на основе случайного входного вектора изображения, а дискриминатора к попытке разделять изображения на два класса: генерируемые генератором и изображения обучающей выборки, часто их в англоязычных статьях описывают как fake и real. В общем случае задачу можно описать как поиск минимакса от суммы логарифма результата дискриминатора от реальных изображений и логарифма обратной вероятности распознавания дискриминатором фейковых изображений. Часто такое обучение сводится к тому, что сначала изменяются весовые коэффициенты дискриминатора, с помощью градиентного спуска, когда он учится классифицировать реальные и фейковые изображения, а затем генератор настраивается так, чтобы максимизировать вероятность распознавания дискриминатором генерируемого изображения как реального.

4.8.2 Проблемы с обучением GAN

Обучение такой сети является достаточно сложной процедурой, так как часто подобный процесс приводит к коллапсу генератора, чаще всего это ситуация выражается в генера-

ции на различные входные случайные вектора одинаковых изображений. Для того чтобы избежать подобных проблем в ранних статьях, посвященных GAN предлагались решения и советы по использованию определенных структур и подходов в сетях генератора и дискриминатора, найденные эмпирическим путем [4]. Так, например, предлагается избавиться от всех полносвязных слоев, использовать слои вида Conv2Transpose, избавиться от слоев пуллинга (MaxPool), преобразовывать случайный одномерный вектор к размерности результирующего тензора изображения, добавлять в дискриминатор слои прореживания (Dropout с вероятностью прореживания 0.5), использовать преобразование входных образов к диапазону от -1 до 1, используя в качестве выходной активационной функции tanh, а также в обязательном порядке слои пакетной нормализации (BatchNormalization). Так же часто в статьях встречается проблема появления изображения в виде шахматной доски, что связывают с особенностями прохождения сверточного фильтра по картам изображения, для борьбы с этим эффектом предлагается использовать слои Upsampling2D с билинейной интерполяцией.

Так же предлагается в качестве функции активации в дискриминаторе использовать не ReLU (ограниченную линейную функцию), а LeakyReLU (так же линейная функция, но для отрицательных значений убывающая медленнее).

Проведенные в данной работе исследования показали, что пакетная нормализация приводит к последующей деградации генератора, потому в данной работе они не используются.

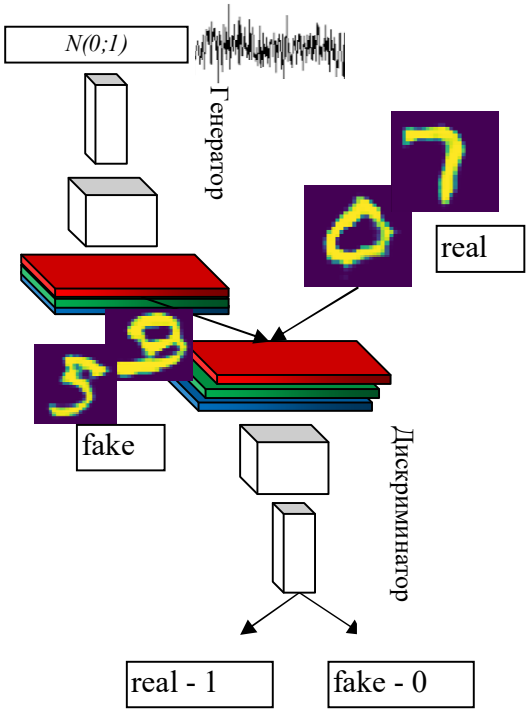


Рисунок 4.19 - Модель генеративно-состязательной сети

Во многих работах слои пакетной нормализации модифицируются, либо для них так же подбираются определенные параметры, но как показали эксперименты это не всегда помогает. Потому для выполнения лабораторных работ по данному направлению выработаны определенные рекомендации, которые позволят получить студенту положительный результат на базовой обучающей выборке и далее продолжить собственные исследования.

Так как выполнение подобных работ требует значительных вычислительных ресурсов, то в качестве среды разработки предлагается использовать ресурс Colaboratory Google (colab.research.google.com), язык Python и библиотеку Keras. Данный ресурс позволяет выделить вычислительные мощности графических и тензорных процессоров.

В первой части работы предлагается создать стандартную DCGAN (Deep Convolution GAN) реализующую генерацию на основе стандартной обучающей выборки рукописных цифр mnist. Во второй части предлагается создать WGAN (Wasserstein GAN) [5]. Теоретически было показано, что мерой расходимости распределений для стандартной GAN является мера Дженсона-Шеннона, являющейся суммой двух расхождений (дивергенций) Кульбека-Лейблера. Назначение данных расхождений оценить похожесть двух распределений случайных величин. Но расхождение Кульбека-Лейблера несимметрично, то есть рассчитанное та-

ким образом расстояние одного распределения относительно другого не равно расстоянию, рассчитанному в обратном отношении (не выполняется неравенство треугольника), потому иногда эту меру не называют мерой или расстоянием. При этом как показывается в работе [5] мера Вассерштейна имеет преимущества, с одной стороны это действительно расстояние, а с другой учитывает пространственную метрику, фактически показывая какую часть условной вероятностной массы нужно перенести от одного распределения, чтобы получить другое (фактически какую работу для этого нужно совершить). В работе [5] предлагается альтернативный подход для расчета меры Вассерштейна, при этом налагая на параметрически заданную искомую функцию условие 1-Липшицовсти в виде $||f(x)||_L<1$, в работе [5] данное условие предлагается выполнять за счет ограничение на веса, в других работах предлагается добавлять штраф градиента. Использование ограничения на весовые коэффициенты является весьма спорным, что признают сами авторы статьи, и для задач более сложных чем генерация цифр процесс обучения расходится, более того выбор других параметров обучения (в частности алгоритма) так же приводит к дестабилизации и взрывному росту градиента. Нами предлагается налагать ограничение в прямом виде в передаваемом пакете обучающих примеров. В качестве 1-Липшевости рассматривается условие $|f(x)-f(y)|<|x-y|L$ и соответствующий штраф, далее будет приведен код на Keras реализующий данный подход.

4.8.3 Код, реализующий GAN с использованием Keras

Далее приводится код на python, так как сам по себе он вполне понятно описывает структуру сети и алгоритм обучения GAN.

Слои использующиеся для создания дискриминатора.

```
def conv(filt,size,x):
    y = layers.Conv2D(filters = filt, kernel_size = (size,size),padding='same') (x)
    lay = layers.LeakyReLU(alpha=0.2) (y)
    lay = layers.Dropout(rate = 0.2)(lay)
    return lay
```

Слои использующиеся для создания генератора.

```
def dconv(filt,size,x, strides=2):
    y = layers.UpSampling2D(size=strides)(x)
    y = layers.Conv2D(filters = filt,kernel_size=size,padding='same') (y)
```

```
    lay = layers.LeakyReLU(alpha=0.2) (y)
    return lay
```

Описание сети дискриминатора.

```
discriminator_input = krs.layers.Input(shape = (nh,nw,1))
lay = conv(32,3,encoder_input)
lay = layers.MaxPooling2D((2,2)) (lay)
lay = conv(64,3,lay)
lay = layers.MaxPooling2D((2,2)) (lay)
lay = conv(128,3,lay)
lay = layers.MaxPooling2D((2,2)) (lay)
lay = conv(256,3,lay)
lay = layers.Flatten()(lay)
lay = layers.Dense(1, activation = 'sigmoid') (lay)
```

Описание сети генератора.

```
generator_input = layers.Input(shape = (num_hide,))
lay = layers.Dense(7 * 7 * 128, activation='relu') (generator_input)
lay = layers.Reshape(target_shape=(7,7,128)) (lay)
lay = dconv(128,3,lay)
lay = dconv(64,3,lay)
lay = layers.Conv2D(filters = 1,kernel_size=7,padding='same',activation='tanh') (lay)
```

Для студентов уже реализовавших сверточную нейронную сеть (CNN) данный код не должен вызывать вопросов. Единственно, что здесь стоит пояснить это, то что дискриминатор как и ожидается имеет один выход, соответствующий нейрону с сигмовидной функцией активации, далее дискриминатор будет обучаться с использованием бинарной кроссэнтропии, функции потерь содержащей в своем выражении логарифмы, что при определенном подходе к обучению и даст классическое выражение для минимаксной игры.

Сама процедура обучения выглядит следующим образом:

```
for i_learn in range(n_learn):
```

```
indexes = numpy.random.randint(0, all_images.shape[0], n_batch)
real_images = all_images[indexes]
rx = numpy.random.randn(n_batch, num_hide)
fake_images = generator.predict(rx)
rf = numpy.concatenate([real_images,fake_images])
oz = numpy.concatenate([ones,zeros])
dloss = descriminator.train_on_batch(rf, oz)
gloss = gan_model.train_on_batch(rx, ones)
```

Здесь легко прослеживаются основные этапы процедуры динамического процесса игры, когда сначала обучается дискриминатор, затем генератор. В процессе идет получение генерируемых генератором изображений и передача их наряду с реальными изображениями в дискриминатор для обучения на одном так называемом батче или пакете примеров (сформированным случайным образом). Затем дискриминатор максимизирует вероятность распознавания генерируемых изображений (решая задачу минимизации разности с единицей). При этом обучение происходит через слои замороженного дискриминатора.

Указание на то, что веса дискриминатора должны обучаться.

```
descriminator.trainable = True
descriminator.compile(loss='binary_crossentropy', optimizer=krs.optimizers.Adam(0.00001),
                      metrics=['accuracy'])
```

В объединенной сети дискриминатора и генератора веса дискриминатора не меняются. Выход сети генератора поступает на вход дискриминатора в общей модели обучения.

```
descriminator.trainable = False
gan = descriminator(generator.layers[-1].output)
gan_model = krs.Model(inputs=generator.layers[0].input, outputs=gan)
gan_model.compile(loss='binary_crossentropy', optimizer=krs.optimizers.Adam(0.00001),
                  metrics=['accuracy'])
```

Пример работы сети генератора приведен на рисунке 4.20. Размерность случайного вектора 100.

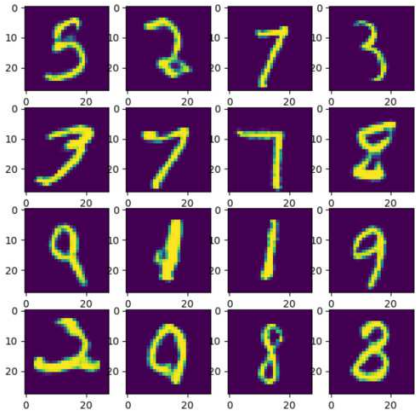


Рисунок 4.20 - Пример работы сети генератора при генерации цифр

4.8.4 Код, реализующий WGAN с использованием Keras

Для WGAN также приведем основные части реализующего данный алгоритм, за описанием самого базового алгоритма можно обратиться к его авторам [5].

В качестве предлагаемой минимизируемой функции дискриминатора предлагается следующая:

```
def model_wloss_descr(input):
    def wloss(y_true,x_pred):
        xval = krs.backend.mean(krs.backend.abs(input[:,None] - input[None,:]))
        fval = krs.backend.mean(krs.backend.abs(x_pred[:,None]-x_pred[None,:]))
        return krs.backend.mean(y_true*x_pred)+(krs.backend.relu(fval-xval))
    return wloss
```

Сам дискриминатор уже имеет в качестве выходной функции активации линейную функцию. В качестве алгоритма оптимизации как и советуют авторы статьи [5] выбран RMSProp.

```
descriminator.trainable = True
descriminator.compile(loss=model_wloss_descr(descriminator.input), optimizer=krs.optimizers.RMSprop(0.00005))
```

Далее опишем функции потерь генератора и объединенную модель.

```
def model_wloss_gan(input):
    def wloss(y_true,x_pred):
```

```
        return krs.backend.mean(y_true*x_pred)
    return wloss

descriminator.trainable = False
gan = descriminator(generator.layers[-1].output)
gan_model = krs.Model(inputs=generator.layers[0].input, outputs=gan)
gan_model.compile(loss=model_wloss_gan(generator.layers[-1].output), optimizer=krs.optimizers.RMSprop(0.00005))
```

Сам код реализующий алгоритм выглядит так:

```
ones = numpy.ones((n_batch,1))
neg_ones = -numpy.ones((n_batch,1))
n_critic = 1
for i_learn in range(n_learn):
    for j in range(n_critic):
        indexes = numpy.random.randint(0,all_image.shape[0],n_batch)
        real_image = all_image[indexes]
        rx  = numpy.random.randn(n_batch,num_hide)
        fake_image = generator.predict(rx)
        trainx = numpy.concatenate([real_image,fake_image])
        trainy = numpy.concatenate([neg_ones,ones])
        dloss, _ = descriminator.train_on_batch(trainx,trainy)
        #for l in descriminator.layers:
        #       weights = l.get_weights()
        #       weights = [numpy.clip(w, -0.01, 0.01) for w in weights]
        #       l.set_weights(weights)
    rx  = numpy.random.randn(n_batch,num_hide)
    gloss, _ = gan_model.train_on_batch(rx,neg_ones)
```

В указанном коде присутствует комментарий реализующий ограничение, предложенное в работе [5].

Кроме указанных здесь подходов к реализации GAN существует целое множество типов таких сетей, модифицирующих соответствующую антогонистическую игру. Например, Condition GAN, Control GAN и прочие подобные разновидности, добавляющие к входам генератора и дискриминатора метки классов объектов или дополнительные характеристики объектов, что стабилизирует сходимость. Кроме того, существуют GAN для повышения разрешения, когда, например, на вход сети генератора подается изображение с плохим разрешением или меньшего размера и генерируется изображение более высокого разрешения, дополнительно функция потерь учитывает потери контента (содержимого) на основе использования предобученной нейронной сети vgg16. Stacked GAN использует похожий механизм повышения разрешения при генерации изображений по словесному описанию. Подобная сеть (RuDali) была, например, недавно представлена Сбером. В целом данное направление развивается стремительно и за последние буквально несколько лет различными исследователями получены значительные результаты.

1. Kullback S., Leibler R.A. On information and sufficiency // Annals of Mathematical Statistics. – 1951. Vol. 22, No. 1. – P. 79–86. doi:10.1214/aoms/1177729694

2. Doersch C. Tutorial on variational autoencoders // arXiv preprint arXiv:1606.05908. – 2016. – P. 1–23.

3. Goodfellow I. Generative adversarial nets / I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley and others // Advances in Neural Information Processing Systems 27. - Curran Associates, Inc. – 2014. – P. 2672-2680. - URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

4. Saliman T., Goodfellow I., Zaremba W., Cheung V. , Radford A., Chen X. Improved techniques for training GAN // Advances in Neural Information Processing Systems. – 2016. pp. 2234-2242. URL: https:// arxiv. org/abs/1606.03498

5. Martin Arjovsky, Soumith Chintala, and L´eon Bottou. Wasserstein GAN. CoRR, abs/1701.07875, 2017.

4.9 Примеры кода реализующие проверку работы различных видов GAN

4.9.1 Реализация Conditional GAN

```
# conditional gan

import matplotlib.pyplot as plt
```

```
import tensorflow.keras as krs

import numpy
import pandas as pd

from keras.datasets import mnist

# функция устанавливающая обучаемость слоев модели, False - веса фиксированные
def trainable(model, flag):
    model.trainable = flag
    for l in model.layers:
        l.trainable = flag
    return


nw = 28
nh = 28
num_hide = 196


# загружаем обучающий датасет
(trainx, trainy), (testx, testy) = mnist.load_data()

all_image = (trainx/255.0-0.5)*1.999
all_image = numpy.expand_dims(all_image, axis=3)

# преобразуем метки классов из вида 0..9 в бинарный вектор
all_labels = krs.utils.to_categorical(trainy)

print(all_image.shape)

# создаем сеть дескриминатор с двумя входами
desc_input = krs.layers.Input(shape=(nw,nh,1),name = 'inpd1')
# второй вход для метки класса образа
desc_input1 = krs.layers.Input(shape=(10,), name = 'inpd2')
lay1 = krs.layers.Dense(7*7, name = 'd2') (desc_input1)
lay1 = krs.layers.Dense(28*28, name = 'd3') (lay1)
lay1 = krs.layers.Reshape((28,28,1), name = 'd4') (lay1)
# объединяем два параллельных слоя
layc = krs.layers.Concatenate(name = 'd5') ([desc_input, lay1])


lay = krs.layers.Conv2D(32, (3, 3), strides = (2,2), activation='relu', padding='same', name =
'd6')(layc)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(64, (3, 3), strides = (2,2),activation='relu', padding='same', name =
'd7')(lay)
lay = krs.layers.Dropout(0.15)(lay)
```

```
lay = krs.layers.Conv2D(128, (3, 3), strides = (2,2),activation='relu', padding='same', name =
'd8')(lay)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(256, (3, 3), strides = (2,2),activation='relu', padding='same', name =
'd9')(lay)
lay = krs.layers.Flatten(name = 'd10')(lay)
lay_out = krs.layers.Dense(1, activation="sigmoid", name='den4')(lay)

# определяем модель дескриминатора
descriminator = krs.Model([desc_input,desc_input1], lay_out)
descriminator.trainable = True
descriminator.compile(loss='binary_crossentropy', optimizer=krs.optimizers.Adam(learning_rate =
0.0002),
                      metrics=['accuracy'])

# создаем сеть генератора с двумя входами
gen_input = krs.layers.Input(shape=(num_hide,),name = 'ginp1')
# вход для метки класса образа
gen_input1 = krs.layers.Input(shape=(10,),name = 'ginp2')

layc = krs.layers.Concatenate(name = 'g3')([gen_input,gen_input1])
lay = krs.layers.Dense(128*7*7,name = 'g4')(layc)
lay = krs.layers.Reshape(target_shape=(7,7,128),name = 'g5')(lay)
lay = krs.layers.Conv2D(128, (3, 3), activation='relu', padding='same',name = 'g6')(lay)
lay = krs.layers.UpSampling2D(size=(2,2),name = 'g7')(lay)
lay = krs.layers.Conv2D(64, (3, 3), activation='relu', padding='same',name = 'g8')(lay)
lay = krs.layers.UpSampling2D(size=(2,2),name = 'g9')(lay)
lay_out = krs.layers.Conv2D(1, (3, 3), activation='tanh', padding='same',name = 'g10')(lay)

generator = krs.Model([gen_input,gen_input1],lay_out)



krs.utils.plot_model(descriminator, to_file='./descriminator.png', show_shapes=True)
krs.utils.plot_model(generator, to_file='./generator.png', show_shapes=True)

#  отключаем обучение весов дескриминатора
descriminator.trainable = False

# создаем объединенную сеть дескриминатора и генератора
gan = descriminator([generator.layers[-1].output, gen_input1])
gan_model = krs.Model(inputs=[gen_input, gen_input1], outputs=gan)
# binary_crossentropy
gan_model.compile(loss='binary_crossentropy', optimiz-
er=krs.optimizers.Adam(learning_rate=0.0002),
                  metrics=['accuracy'])
```

```
krs.utils.plot_model(gan_model, to_file='./gan_model.png', show_shapes=True)

# binary_crossentropy

krs.utils.plot_model(gan_model, to_file='./gan_model.png', show_shapes=True)
n_learn = 50000
n_batch = 32
n_batch_check = 4000
ones = numpy.ones((n_batch, 1))   -1e-13
zeros = numpy.zeros((n_batch, 1))   +1e-13
oz = numpy.concatenate([ones, zeros])
zo = numpy.concatenate([zeros, ones])

ones_c = numpy.ones((n_batch_check, 1))   -1e-13
zeros_c = numpy.zeros((n_batch_check, 1))   +1e-13
oz_c = numpy.concatenate([ones_c, zeros_c])
zo_c = numpy.concatenate([zeros_c, ones_c])

fig = plt.figure(figsize = (5,5))
bx = [[]]*16
for i in range(16):
    bx[i] = fig.add_subplot(4, 4, i + 1)
    bx[i].imshow((all_image[i][:, :, 0]+1)*0.5)
plt.show()


#descriminator.load_weights("./out/model_descriminator12.h5")
#generator.load_weights("./out/model_generator12.h5")
#inv_descriminator.load_weights("./out/imodel_descriminator12.h5")

ax = [[], [], [], [], [], [], [], [],[], [], [], [], [], [], [], []]
classificator = krs.models.load_model("class_mnist.h5")

error = []

for i_learn in range(n_learn):
    # выбираем батч реальных образов и меток их классов
    indexes = numpy.random.randint(0, all_image.shape[0], n_batch)
    real_image = all_image[indexes]
    real_labels = all_labels[indexes]

    # генерируем вектора скрытого пространства и метки классов
    rx = numpy.random.randn(n_batch, num_hide)

    fake_labels = numpy.random.randint(0, 10, n_batch)
    fake_labels = krs.utils.to_categorical(fake_labels, num_classes = 10)
```

```
    # получаем генерируемые образы
    fake_image = generator.predict([rx,fake_labels])

    # склеиваем в один батч реальные и фейковые образы
    rf = numpy.concatenate([real_image,fake_image])
    oz = numpy.concatenate([ones,zeros])

    # склеиваем метки реальных и фейковх образов
    labels = numpy.concatenate([real_labels,fake_labels])

    # обучаем дескриминатор
    dloss = descriminator.train_on_batch([rf,labels], oz)

    # обучаем генератор
    gloss1 = gan_model.train_on_batch([rx,fake_labels], ones)
    zo = numpy.concatenate([zeros,ones])

    # контроль обучения, сохранение результатов
    if (i_learn % 1000 == 0 ):
        indexes = numpy.random.randint(0, all_image.shape[0], n_batch_check)
        real_image = all_image[indexes]
        real_labels = all_labels[indexes]

        rx = numpy.random.normal(0, 1, (n_batch_check, num_hide))#numpy.random.randn(n_batch,
num_hide)
        inp_dec =  rx

        fake_labels = numpy.random.randint(0, 10, n_batch_check)
        fake_labels = krs.utils.to_categorical(fake_labels, num_classes = 10)


        fake_image = generator.predict([inp_dec, fake_labels])
        images = numpy.concatenate([real_image, fake_image])

        labels = numpy.concatenate([real_labels,fake_labels])
        pred = descriminator.predict_on_batch([images, labels])
        dloss = numpy.abs(pred - oz_c).mean()

        # для расчета gloss
        ans_gan = gan_model.predict_on_batch([inp_dec,fake_labels])

        # получаем метки сгенерированных образов с помощью предобученного классификатора
        ans_cls = classificator.predict_on_batch(fake_image)

        # получаем гистограмму распределения классов
        arr = numpy.bincount(ans_cls.argmax(axis=1))
        print(arr)
        # получаем вероятность появления класса
```

```
        parr = arr/arr.sum()+1e-20
        # считае энтропию
        H = (parr*numpy.log(1/parr)).sum()
        error.append([i_learn, H , dloss])
        print(f"entropy {H}")

        gloss = numpy.abs(ans_gan - ones_c).mean()

        print(f"index {i_learn} dloss {dloss}  gloss {gloss} ")

        if (i_learn % 2000 == 0 and i_learn > 4000):
            # сохраняем веса дескриминатора и генератора
            descriminator.save_weights("./model_descriminator13.h5")
            generator.save_weights("./model_generator13.h5")

        # рисуем 16 примеров сгенерированных изображений

        rx = numpy.random.normal(0, 1, (16, num_hide))#numpy.random.randn(16,num_hide)

        fake_labels = numpy.random.randint(0, 10, 16)
        fake_labels = krs.utils.to_categorical(fake_labels, num_classes = 10)

        inp_dec = rx
        pred_dec = generator.predict([inp_dec, fake_labels])


        fig = plt.figure(figsize = (5,5))
        for i in range(16):
            ax[i] = fig.add_subplot(4, 4, i + 1)
            ax[i].imshow((pred_dec[i][:, :, 0]+1)*0.5)
        plt.show()

# отображаем поведение энтропии в процессе обучения
error  = numpy.array(error)
fig = plt.figure(figsize = (10,5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
ax1.plot(error[:,0],error[:,1])
ax1.set_ylabel('Entropy')
ax1.set_xlabel('Number of iteration')

rx = numpy.random.randn(n_batch_check,num_hide)

fake_labels = numpy.random.randint(0, 10, 16)
fake_labels = krs.utils.to_categorical(fake_labels, num_classes = 10)


fake_image = generator.predict([rx,fake_labels])
```

```
ans_cls = classificator.predict_on_batch(fake_image)
arr = numpy.bincount(ans_cls.argmax(axis=1))

ax2.bar([0,1,2,3,4,5,6,7,8,9], arr)
ax2.set_ylabel('Historgamm')
ax2.set_xlabel('Classes')

td = pd.DataFrame({"index":error[:,0], "entropy":error[:,1], "dloss":error[:,2] })

td.to_excel("error.xls", sheet_name = "error")
td.to_csv("error.txt", sep = " ")
plt.show()

# mnist classifier

import keras.layers
import matplotlib.pyplot as plt
import tensorflow.keras as krs
import numpy

from keras.datasets import mnist
import tensorflow.keras.utils as ut

import tensorflow as tf

print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpus = tf.config.list_physical_devices('GPU')
print("GPUs Available: ", gpus)

tf.config.experimental.set_visible_devices(gpus[0], 'GPU')

nw = 28
nh = 28

(trainx, trainy), (testx, testy) = mnist.load_data()
all_image = (trainx/255.0-0.5)*1.999
all_image = numpy.expand_dims(all_image, axis=3)

all_out = ut.to_categorical(trainy)

testx = (testx/255.0-0.5)*1.999
testy = ut.to_categorical(testy)

desc_input = krs.layers.Input(shape=(nw,nh,1))
lay = krs.layers.Conv2D(32, (3, 3), strides = (2,2), activation='relu', padding='same')(desc_input)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(64, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Dropout(0.15)(lay)
```

```
lay = krs.layers.Conv2D(128, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(256, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Flatten()(lay)
lay_out = krs.layers.Dense(10, activation="softmax", name='den4')(lay)

classificator = krs.Model(desc_input, lay_out)
classificator.trainable = True
classificator.compile(loss='binary_crossentropy', optimizer=krs.optimizers.Adam(learning_rate =
0.0002),
                      metrics=['accuracy'])

classificator.fit(all_image, all_out, batch_size = 8000 ,epochs = 150, validation_data=(testx,testy))

classificator.save("./out/class_mnist.h5")
```

4.9.2 Реализация стандартной GAN

```
# обычный gan

%matplotlib
%matplotlib inline

import matplotlib.pyplot as plt
import tensorflow.keras as krs
import numpy
import pandas as pd
from keras.datasets import mnist

def trainable(model, flag):
    model.trainable = flag
    for l in model.layers:
        l.trainable = flag
    return

nw = 28
nh = 28
num_hide = 196

(trainx, trainy), (testx, testy) = mnist.load_data()
all_image = (trainx/255.0-0.5)*1.999
all_image = numpy.expand_dims(all_image, axis=3)

print(all_image.shape)

desc_input = krs.layers.Input(shape=(nw,nh,1))
```

```
lay = krs.layers.Conv2D(32, (3, 3), strides = (2,2), activation='relu', padding='same')(desc_input)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(64, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(128, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(256, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Flatten()(lay)
lay_out = krs.layers.Dense(1, activation="sigmoid", name='den4')(lay)

descriminator = krs.Model(desc_input, lay_out)
descriminator.trainable = True
descriminator.compile(loss='binary_crossentropy', optimizer=krs.optimizers.Adam(learning_rate =
0.0002),
                      metrics=['accuracy'])
gen_input = krs.layers.Input(shape=(num_hide,))
lay = krs.layers.Dense(128*7*7)(gen_input)
lay = krs.layers.Reshape(target_shape=(7,7,128))(lay)
lay = krs.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(lay)
lay = krs.layers.UpSampling2D(size=(2,2))(lay)
lay = krs.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(lay)
lay = krs.layers.UpSampling2D(size=(2,2))(lay)
lay_out = krs.layers.Conv2D(1, (3, 3), activation='tanh', padding='same')(lay)

generator = krs.Model(gen_input,lay_out)

krs.utils.plot_model(descriminator, to_file='./descriminator.png', show_shapes=True)

descriminator.trainable = False
gan = descriminator(generator.layers[-1].output)
gan_model = krs.Model(inputs=generator.layers[0].input, outputs=gan)
# binary_crossentropy
gan_model.compile(loss='binary_crossentropy', optimiz-
er=krs.optimizers.Adam(learning_rate=0.0002),
                  metrics=['accuracy'])

# binary_crossentropy

krs.utils.plot_model(gan_model, to_file='./gan_model.png', show_shapes=True)
n_learn = 50000
n_batch = 32
n_batch_check = 4000
ones = numpy.ones((n_batch, 1))   -1e-13
zeros = numpy.zeros((n_batch, 1))   +1e-13
oz = numpy.concatenate([ones, zeros])
zo = numpy.concatenate([zeros, ones])

ones_c = numpy.ones((n_batch_check, 1))   -1e-13
```

```
zeros_c = numpy.zeros((n_batch_check, 1))  +1e-13
oz_c = numpy.concatenate([ones_c, zeros_c])
zo_c = numpy.concatenate([zeros_c, ones_c])

fig = plt.figure(figsize = (5,5))
bx = [[]]*16
for i in range(16):
    bx[i] = fig.add_subplot(4, 4, i + 1)
    bx[i].imshow((all_image[i][:, :, 0]+1)*0.5)
plt.show()

#descriminator.load_weights("./out/model_descriminator12.h5")
#generator.load_weights("./out/model_generator12.h5")
#inv_descriminator.load_weights("./out/imodel_descriminator12.h5")

ax = [[], [], [], [], [], [], [], [],[], [], [], [], [], [], [], []]
classificator = krs.models.load_model("class_mnist.h5")

error = []

for i_learn in range(n_learn):
    indexes = numpy.random.randint(0, all_image.shape[0], n_batch)
    real_image = all_image[indexes]
    rx = numpy.random.randn(n_batch, num_hide)
    fake_image = generator.predict(rx)
    rf = numpy.concatenate([real_image,fake_image])
    oz = numpy.concatenate([ones,zeros])

    dloss = descriminator.train_on_batch(rf, oz)

    gloss1 = gan_model.train_on_batch(rx, ones)
    zo = numpy.concatenate([zeros,ones])

    if (i_learn % 1000 == 0 ):
        indexes = numpy.random.randint(0, all_image.shape[0], n_batch_check)
        real_image = all_image[indexes]
        #mu_dx = [pred_enc[0][indexes], pred_enc[1][indexes]]
        rx = numpy.random.normal(0, 1, (n_batch_check, num_hide))#numpy.random.randn(n_batch,
num_hide)
        inp_dec =  rx
        fake_image = generator.predict(inp_dec)
        images = numpy.concatenate([real_image, fake_image])
        pred = descriminator.predict_on_batch(images)
        dloss = numpy.abs(pred - oz_c).mean()

        ans_gan = gan_model.predict_on_batch(inp_dec)

        ans_cls = classificator.predict_on_batch(fake_image)
```

```
        #print(ans_cls.shape)
        #print(ans_cls[0:5])
        #print(ans_cls.argmax(axis=1))
        arr = numpy.bincount(ans_cls.argmax(axis=1))
        print(arr)
        parr = arr/arr.sum()+1e-20
        H = (parr*numpy.log(1/parr)).sum()
        error.append([i_learn, H , dloss])
        print(f"entropy {H}")

        gloss = numpy.abs(ans_gan - ones_c).mean()

        print(f"index {i_learn} dloss {dloss}  gloss {gloss} ")

        if (i_learn % 2000 == 0 and i_learn > 4000):
            descriminator.save_weights("./model_descriminator13.h5")
            generator.save_weights("./model_generator13.h5")

        rx = numpy.random.normal(0, 1, (16, num_hide))#numpy.random.randn(16,num_hide)
        #inds = numpy.random.randint(0, n_batch, 16)
        inp_dec = rx #pred_enc[0][inds] + pred_enc[1][inds] * rx[inds]
        pred_dec = generator.predict(inp_dec)


        fig = plt.figure(figsize = (5,5))
        for i in range(16):
            ax[i] = fig.add_subplot(4, 4, i + 1)
            ax[i].imshow((pred_dec[i][:, :, 0]+1)*0.5)
        plt.show()

#тут надо сохранять график
error  = numpy.array(error)
fig = plt.figure(figsize = (10,5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
ax1.plot(error[:,0],error[:,1])
ax1.set_ylabel('Entropy')
ax1.set_xlabel('Number of iteration')

rx = numpy.random.randn(n_batch_check,num_hide)
fake_image = generator.predict(rx)
ans_cls = classificator.predict_on_batch(fake_image)
arr = numpy.bincount(ans_cls.argmax(axis=1))


ax2.bar([0,1,2,3,4,5,6,7,8,9], arr)
ax2.set_ylabel('Historgamm')
ax2.set_xlabel('Classes')
```

```
td = pd.DataFrame({"index":error[:,0], "entropy":error[:,1], "dloss":error[:,2] })

td.to_excel("error.xls", sheet_name = "error")
td.to_csv("error.txt", sep = " ")
plt.show()
```

4.9.3 Реализация WGAN

```
# WGAN with clips comments and проверкой на условие липшица
import matplotlib.pyplot as plt
import tensorflow.keras as krs

import numpy

import tensorflow as tf
import pandas as pd

print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpus = tf.config.list_physical_devices('GPU')
print("GPUs Available: ", gpus)

tf.config.experimental.set_visible_devices(gpus[1], 'GPU')

import gc
from keras.datasets import mnist

from tensorflow.python.framework.ops import disable_eager_execution
disable_eager_execution()


def trainable(model, flag):
    model.trainable = flag
    for l in model.layers:
        l.trainable = flag
    return

def model_wloss_gan(input):

    def wloss(y_true,x_pred):
        return krs.backend.mean(y_true*x_pred)
    return wloss


def model_wloss_descr(input):
    def wloss(y_true,x_pred):
        xval = krs.backend.mean(krs.backend.abs(input[:,None] - input[None,:]))
```

```
        fval = krs.backend.mean(krs.backend.abs(x_pred[:,None] - x_pred[None,:]))
        return krs.backend.mean(y_true*x_pred)+(krs.backend.relu(fval-xval))*0.85
    return wloss
nw = 28
nh = 28
num_hide = 196

(trainx, trainy), (testx, testy) = mnist.load_data()
all_image = (trainx/255.0-0.5)*1.999
all_image = numpy.expand_dims(all_image, axis=3)

print(all_image.shape)

desc_input = krs.layers.Input(shape=(nw,nh,1))
lay = krs.layers.Conv2D(32, (3, 3), strides = (2,2), activation='relu', padding='same')(desc_input)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(64, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(128, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Dropout(0.15)(lay)
lay = krs.layers.Conv2D(256, (3, 3), strides = (2,2),activation='relu', padding='same')(lay)
lay = krs.layers.Flatten()(lay)
lay_out = krs.layers.Dense(1, activation="linear", name='den4')(lay)

descriminator = krs.Model(desc_input, lay_out)
descriminator.trainable = True
descriminator.compile(loss=model_wloss_descr(descriminator.input), optimiz-
er=krs.optimizers.RMSprop(0.0001), metrics=['accuracy'])
gen_input = krs.layers.Input(shape=(num_hide,))
lay = krs.layers.Dense(128*7*7)(gen_input)
lay = krs.layers.Reshape(target_shape=(7,7,128))(lay)
lay = krs.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(lay)
lay = krs.layers.UpSampling2D(size=(2,2))(lay)
lay = krs.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(lay)
lay = krs.layers.UpSampling2D(size=(2,2))(lay)
lay_out = krs.layers.Conv2D(1, (3, 3), activation='tanh', padding='same')(lay)

generator = krs.Model(gen_input,lay_out)

krs.utils.plot_model(descriminator, to_file='./out/descriminator.png', show_shapes=True)

descriminator.trainable = False
gan = descriminator(generator.layers[-1].output)
gan_model = krs.Model(inputs=generator.layers[0].input, outputs=gan)

# binary_crossentropy
gan_model.compile(loss = model_wloss_gan(generator.layers[-1].output), optimiz-
er=krs.optimizers.RMSprop(0.0001),
```

```
            metrics=['accuracy'])

# binary_crossentropy
krs.utils.plot_model(gan_model, to_file='./out/gan_model.png', show_shapes=True)

n_learn = 50000
n_batch = 32
ax = [[], [], [], [], [], [], [], [],[], [], [], [], [], [], [], []]
classificator = krs.models.load_model("./out/class_mnist.h5")

error = []

n_batch_check = 4000
ones_c = numpy.ones((n_batch_check, 1))  -1e-13

ones = numpy.ones((n_batch,1))
neg_ones = -numpy.ones((n_batch,1))
n_critic = 1

for i_learn in range(n_learn):
    for j in range(n_critic):
        indexes = numpy.random.randint(0,all_image.shape[0],n_batch)
        real_image = all_image[indexes]
        rx  = numpy.random.randn(n_batch,num_hide)
        fake_image = generator.predict(rx)

        trainx = numpy.concatenate([real_image,fake_image])
        trainy = numpy.concatenate([neg_ones,ones])
        dloss, _ = descriminator.train_on_batch(trainx,trainy)
        #for l in descriminator.layers:
        #    weights = l.get_weights()
        #    weights = [numpy.clip(w, -0.01, 0.01) for w in weights]
        #    l.set_weights(weights)
    rx  = numpy.random.randn(n_batch,num_hide)
    gloss, _ = gan_model.train_on_batch(rx,neg_ones)

    if (i_learn % 1000 == 0):
        print(f"index {i_learn} dloss {dloss}  gloss {gloss}  ")

        rx = numpy.random.randn(n_batch_check,num_hide)

        #ans_gan = generator.predict(inp_dec)
        fake_image = generator.predict(rx)

        ans_cls = classificator.predict_on_batch(fake_image)
        #print(ans_cls.shape)
        #print(ans_cls[0:5])
        #print(ans_cls.argmax(axis=1))
```

```
        arr = numpy.bincount(ans_cls.argmax(axis=1))
        print(arr)
        parr = arr/arr.sum()+1e-20
        H = (parr*numpy.log(1/parr)).sum()
        error.append([i_learn, H ])
        print(f"entropy {H}")
        if (i_learn % 2000 == 0 and i_learn > 4000):
            descriminator.save_weights("./out/model_descriminator13.h5")
            generator.save_weights("./out/model_generator13.h5")

        rx = numpy.random.normal(0, 1, (16, num_hide))#numpy.random.randn(16,num_hide)
        #inds = numpy.random.randint(0, n_batch, 16)
        inp_dec = rx #pred_enc[0][inds] + pred_enc[1][inds] * rx[inds]
        pred_dec = generator.predict(inp_dec)

        fig = plt.figure(figsize = (5,5))
        for i in range(16):
            ax[i] = fig.add_subplot(4, 4, i + 1)
            ax[i].imshow((pred_dec[i][:, :, 0]+1)*0.5)
        plt.show()

error  = numpy.array(error)
fig = plt.figure(figsize = (10,5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
ax1.plot(error[:,0],error[:,1])
ax1.set_ylabel('Entropy')
ax1.set_xlabel('Number of iteration')

rx = numpy.random.randn(n_batch_check,num_hide)
fake_image = generator.predict(rx)
ans_cls = classificator.predict_on_batch(fake_image)
arr = numpy.bincount(ans_cls.argmax(axis=1))

ax2.bar([0,1,2,3,4,5,6,7,8,9], arr)
ax2.set_ylabel('Historgamm')
ax2.set_xlabel('Classes')

td = pd.DataFrame({"index":error[:,0], "entropy":error[:,1] })

td.to_excel("./out/error.xls", sheet_name = "error")
td.to_csv("./out/error.txt", sep = " ")
plt.show()
```

4.10 Задание

Реализовать три вида GAN в соответствии с указанным в пособии кодом, разобраться с работой данных программ. Изменить параметры слоев, алгоритмов обучения, посмотреть имзенения.

5. ЛАБОРАТОРНАЯ РАБОТА №.5 ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

5.1 Общая постановка задачи обучения с подкреплением

Обучение с подкреплением применятся в областях, где алгоритм должен быть получен в процессе изучения среды. Данные подходы нашли большое распространение и в задачах, где ранее только человек показывал хорошие результаты, например, игра Go. Так же показано, как ИИ способен выиграть в старкрафт и т.д. В том числе алгоритмы обучения с подкреплением применяются в дообучении таких сетей как ChatCPT (например, PPO). Обычно алгоритмы обучения с подкреплением тестируются на играх типа Atari и прочих искусственных средах.

Дадиим общую постановку задачи обучения с подкреплением (рисунок 5.1).

Рисунок 5.1 - Общая постановка задачи обучения с подкреплением

Дан Марковский процесс принятия решений и соответственно функции вероятности перехода $P(s, a, s') = P(s' \vee a, s)$ из состояния s в состояние s', a – действие.

Policy $\pi(a, s) = P(a \vee s)$. Ставится задача поиска политики максимизирующей функционал (формула 5.1):

$$J(\pi) = E[\sum_{t=0}^{\infty} \gamma^t R_t \vee \pi] \quad (5.1)$$

Функция ценности состояния (value function) (формула 5.2):

$$v(s_{t'}) = E[\sum_{t=t'}^{\infty} \gamma^t R_t \vee \pi, s_{t'}] \quad (5.2)$$

Функция ценности действия из состояния (q-value function) (формула 5.3):

$$q(a_{t'}, s_{t'}) = E\left[\sum_{t=t'}^{\infty} \gamma^t R_t \vee \pi, s_{t'} a_{t'}\right] \quad (5.3)$$

Функции оптимальности Беллмана для ценности состояния (формула 5.4):

$$v(s) = \max_{a \in A} \sum_{s' \in S}^{\infty} P(s' \vee s, a) \cdot \left(R(a, s) + \gamma \cdot v(s')\right) \quad (5.4)$$

Функции оптимальности Беллмана для ценности действия в состоянии (формула 5.5)

$$q(a, s) = \sum_{s' \in S}^{\infty} P(s' \vee s, a) \cdot \left(R(a, s) + \gamma \cdot \max_{a' \in A} q(a', s')\right) \quad (5.5)$$

Каскад существующий методов обучения с подкреплением, представлен на рисунке 5.2.

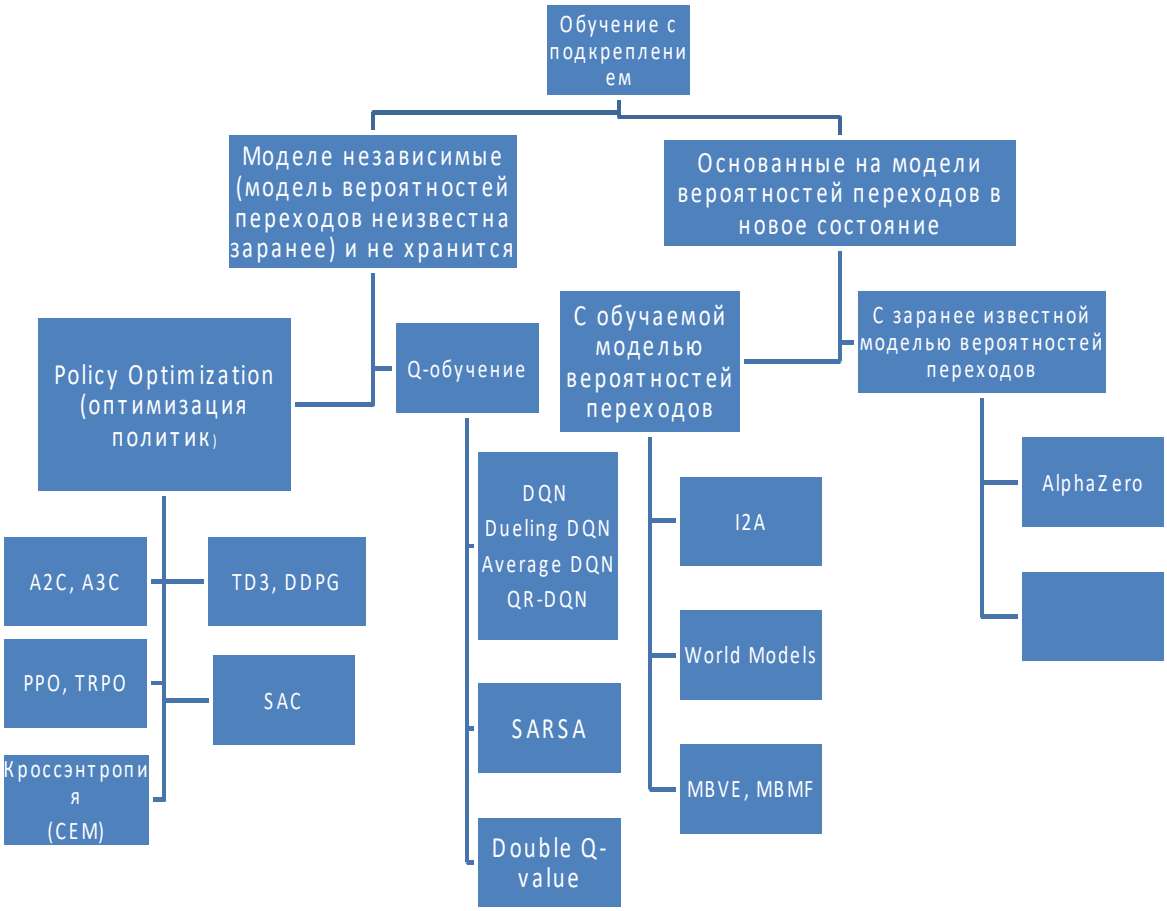


Рисунок 5.2 - Каскад существующий методов обучения с подкреплением

[2] A2C / A3C (Asynchronous Advantage Actor-Critic): Mnih et al, 2016

[3] PPO (Proximal Policy Optimization): Schulman et al, 2017

[4] TRPO (Trust Region Policy Optimization): Schulman et al, 2015

[5] DDPG (Deep Deterministic Policy Gradient): Lillicrap et al, 2015

[6] TD3 (Twin Delayed DDPG): Fujimoto et al, 2018 (*ищут оптимальную стратегию как решение уравнения оптимальности Беллмана*)

[7] SAC (Soft Actor-Critic): Haarnoja et al, 2018

[8] DQN (Deep Q-Networks): Mnih et al, 2013

[9] C51 (Categorical 51-Atom DQN): Bellemare et al, 2017

[10] QR-DQN (Quantile Regression DQN): Dabney et al, 2017

[11] HER (Hindsight Experience Replay): Andrychowicz et al, 2017

[12] World Models: Ha and Schmidhuber, 2018

[13] I2A (Imagination-Augmented Agents): Weber et al, 2017

[14] MBMF (Model-Based RL with Model-Free Fine-Tuning): Nagabandi et al, 2017

[15] MBVE (Model-Based Value Expansion): Feinberg et al, 2018

[16] AlphaZero: Silver et al, 2017

CEM (Кроссэнтропийный метод)

Инициализировать policy

Повторить:

Сэмплировать 100 сессий

Взять 25 лучших сессий (элитных)

Изменить policy по лучшим сессиям, использовав обучение с учителем, на примерах state, actions.

В качестве policy можно выбрать нейронную сеть с выходным слоем softmax, функцией потерь бинарная кроссэнтропия, на входе состояния, на выходе сделанное действие с вероятностью 1.

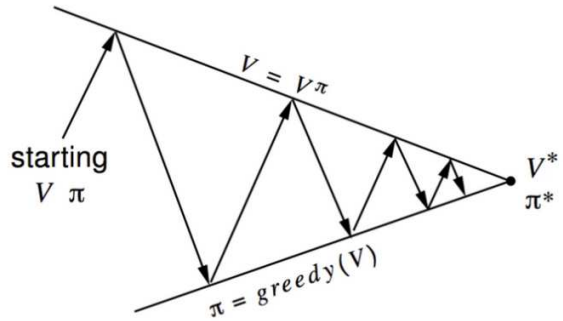
5.2 Итерация политики (Policy iteration).

Основанный на модели метод. Model-Based.

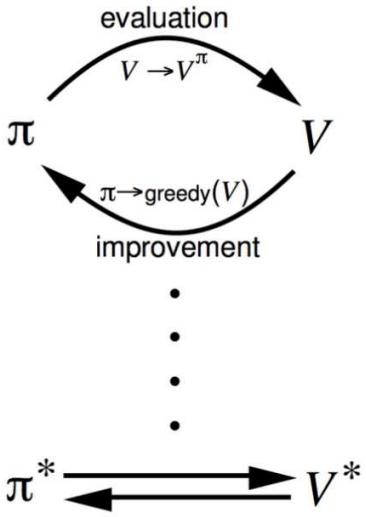
Итерация policy запускает цикл между оценкой и улучшением policy.

Policy Evaluation оценивает функцию стоимости V с помощью жадной policy, полученной в результате последнего policy Improvement. Policy Improvement, с другой стороны, обновляет policy с помощью действия, которое максимизирует V для каждого

состояния. Уравнения обновления основаны на уравнении Беллмана. Итерация продолжается до сходимости.



Policy evaluation Estimate v_π
Any policy evaluation algorithm
Policy improvement Generate $\pi' \geq \pi$
Any policy improvement algorithm



1. Инициализация	
	$V(s) \in R$ и $\pi(s) \in A(s)$ произвольно для всех $s \in S$
2. Policy Evaluation	
	Повторить $\Delta \leftarrow 0$ Для каждого $s \in S$: $\vartheta \leftarrow V(s)$ $V(s) \leftarrow \sum_{s',r} p(s', r\|s, \pi(s))\,[r + \gamma V(s')]$ $\Delta \leftarrow \max\ (\Delta, \|\vartheta - V(s)\|)$ Пока $\Delta <$ 0 (маленькое положительное число)
3. Policy Improvement	
	Policy-stable←истина Для каждого $s \in S$: $a \leftarrow \pi(s)$ $\pi(s) \leftarrow arg\max_a \sum_{s',r} p(s', r\|s, a)\,[r + \gamma V(s')]$ Если $a \neq \pi(s)$, то *policy-stable ←ложь* *Если policy-stable*, то остановиться и вернуть V и π; иначе перейти 2

5.3 Итерация значения (функции ценности) (Value iteration)

Значение Iteration содержит только один компонент. Он обновляет функцию значения V на основе оптимального уравнения Беллмана (формула 5.6).

$$\vartheta_*(s) = \max_\alpha E[R_{t+1} + \gamma\vartheta_*(S_{t+1}) \mid S_t = s, A_t = \alpha] = \max_\alpha \sum_{s',r} p(s', r|s, a)[r + \gamma\vartheta_*(s')] \quad (5.6)$$

Инициализировать массив V произвольно (например, $V(s) = 0$ для всех $s \in S^+$)

Повторить

$\Delta \leftarrow 0$

Для каждого $s \in S$:

$\vartheta \leftarrow V(s)$

$V(s) \leftarrow max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |\vartheta - V(s)|)$

Пока $\Delta < 0$ (маленькое положительное число)

Выведите детерминированную policy, π, такую, что

$\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

5.4 Q-learning

Один из самых часто используемых алгоритмов ОП является алгоритм Q-обучения (Воткинс & Дайан, 1992). Этот алгоритм основан на простом обновлении итерации значения (Беллман, 1957), непосредственно оценивающем функцию оптимального значения Q*. Табличное Q-обучение представляет собой таблицу, которая содержит старые оценки функции значений действий и заранее посчитанных обновлений с использованием следующего правила обновления (формула 5.7):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q\left(s_{t,} a_t\right)\right] \quad (5.7)$$

Где s' - результирующее состояние после применения действия a в состоянии s, r - немедленное вознаграждение, применяемое за действие а в состоянии s, γ - коэффициент дисконтирования, а α - коэффициент обучения.

Когда число состояний велико, ведение таблицы со всеми возможными значениями пар состояние-действие в памяти нецелесообразно. Распространенным решением этой проблемы является использование функции приближения, параметризованной θ, такой, что

$$Q(s,a) \approx Q(s,a;\ \theta). \tag{5.8}$$

$$Q^{new}(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right) \tag{5.9}$$

Инициализировать Q(s,a) произвольно

Повторить (для каждого эпизода):

Инициализировать S

Повторить (для каждого шага эпизода):

Выберите a из s, используя policy, производную от Q

Выполнить действие a, наблюдайте r,s'

Обновлять

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a, Q(s',a') - Q(s,a)]$$

$s \leftarrow s'$;

Пока s не будет результатирующим

5.5 Double Q – learning

$$Q^A_{t+1}(s_t,a_t) = Q^A_t(s_t,a_t) + \alpha_t(s_t,a_t)\left(r_t + \gamma Q^B_t(s_{t+1}, arg\max_a Q^A_t(s_{t+1},a)) - Q^A_t(s_t,a_t)\right) \tag{5.10}$$

$$Q^B_{t+1}(s_t,a_t) = Q^B_t(s_t,a_t) + \alpha_t(s_t,a_t)\left(r_t + \gamma Q^A_t(s_{t+1}, arg\max_a Q^B_t(s_{t+1},a)) - Q^B_t(s_t,a_t)\right) \tag{5.11}$$

Алгоритм 1 Двойное Q-обучение

Инициализировать Q^A, Q^B, s

повторить

Выбрать a, исходя из $Q^A(s,\cdot)$ и $Q^B(s,\cdot)$, соблюдать r, s'

Выбрать (например, случайным образом) либо ОБНОВИТЬ(A) или ОБНОСТЬ(B)

если ОБНОВИТЬ(A) **то**

Определить $a^* = \arg max_a Q^A(s',a)$

$$Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$$

иначе если ОБНОВИТЬ(B) **то**

Определить $b^* = argmax_a Q^B(s',a)$

$$Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)\left(r + \gamma Q^A(s',a^*) - Q^B(s,a)\right)$$

иначе если

$s \leftarrow s'$

пока не закончится

5.6 Sarsa

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (5.12)$$

САРСА очень напоминает Q-learning. Ключевое различие между SARSA и Q-learning заключается в том, что SARSA - это алгоритм на основе policy. Это означает, что SARSA изучает значение Q на основе действия, выполняемого текущей policy вместо жадной policy.

САРСА (внутриполитический контроль TD) для оценки $Q \approx q_*$

Инициализировать Q(s,a), для всех s ∈ S, a ∈ A(s), произвольно, и Q(конечное состояние, •)=0

Повторить (для каждого эпизода):

Инициализировать S

Выберите a из s, используя policy, производную от Q (например., ∈-жадный)

Повторить (для каждого шага эпизода):

Выполнить действие A, наблюдать R, S'

Выбрать A' из S' используя policy, производную от Q (например., ∈-жадный)

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

пока S не станет конечным(результатирующим)

5.7 DQN

Алгоритм 1: глубокое Q-обучение с воспроизведением опыта.

Инициализировать память воспроизведения D до емкости N

Инициализировать функцию "действие-значение" Q со случайными весами θ

Инициализировать целевую функцию ценности действия $\hat{Q}$ с весами $\theta^- = \theta$

Для эпизода=1, M **делать**

Инициализировать последовательность $s_1 = \{x_1\}$ и предварительно обработанная последовательность $\varphi_1 = \varphi(s_1)$

Для t=1, T **делать**

С вероятностью ε выберите случайное действие α_t

В противном случае выберите $\alpha_t = argmax_a Q(\varphi(s_t), a;\ \theta)$

Выполнить действие α_t в эмуляторе и наблюдать вознаграждение r_t и изображение x_{t+1}

Задать $s_{t+1}=s_t, \alpha_t, x_{t+1}$ и предварительно обработать s $\varphi_{t+1}= \varphi\ (s_{t+1})$

Перейти $(\varphi_{\ t}, \alpha_t, r_t\ \ \varphi_{\ t+1})$ в D

Выборка случайной мини-партии переходов $(\varphi_{\ j}, \alpha_j, r_j\ \ \varphi_{\ j+1})$ из D

Задать $y_j = \begin{cases} r_j \text{ если эпизод завершается на шаге } j+1 \\ r_j + \gamma\ max_{a'} \hat{Q}(\varphi_{j+1}, a'; \theta^-) \text{ в противном случае} \end{cases}$

Выполните шаг градиентного спуска по $\left(y_i - Q(\varphi_j, a_j, \theta)\right)^2$ относительно

параметров сети θ

Каждыйшаг C сбрасывается $\hat{Q} = Q$

Конец цикла

Конец цикла

```
def learn(self):
        size_block = 100
        arr = numpy.random.randint(0,self.max_size-self.nmem,size_block)
        states = []
        states_last = []
        for i in range(self.nmem):
                states.append(self.states[arr+i+1])
                states_last.append(self.states[arr+i])
        res = self.target.predict_on_batch(states)
        res_last = self.model.predict_on_batch(states_last)
        acts = self.acts[arr+1]
        rews = self.rews[arr+1]
        dones = self.dones[arr+1]
        outs = res_last.copy()
        nums = numpy.expand_dims(acts.max(axis=1),axis=1)
        mnums = acts >= nums
        numnet = numpy.expand_dims(res.max(axis=1),axis=1)
        mnumnet = res >= numnet
        try:
                outs[mnums] = rews + self.gamma*res[mnumnet]*(1-dones)
        except:
                pass
        self.model.train_on_batch(states_last, outs)
        if(self.index%5000==0):
                self.target.set_weights(self.model.get_weights())
                self.target.save_weights("out/1.file")
                print(self.index)
```

5.7.1 Dueling DQN

Мы хотим разделить преимущества действий, зависящих от состояния, и значения состояния на два отдельных потока. Мы также определяем прямой проход сети с прямым отображением.

```
# Со сверточными сетями
class
ConvDuelingDQN(nn.Module):


                    def __init__(self, input_dim, output_dim):
                    super(ConvDuelingDQN, self).__init__()
                    self.input_dim = input_dim
                        self.output_dim = output_dim
                        self.fc_input_dim = self.feature_size()

                        self.conv = nn.Sequential(
                           nn.Conv2d(input_dim[0], 32, kernel_size=8, stride=4),
                           nn.ReLU(),
                           nn.Conv2d(32, 64, kernel_size=4, stride=2),
                           nn.ReLU(),
                           nn.Conv2d(64, 64, kernel_size=3, stride=1),
                           nn.ReLU()
                        )


                        self.value_stream = nn.Sequential(
                           nn.Linear(self.fc_input_dim, 128),
                           nn.ReLU(),
                           nn.Linear(128, 1)
                        )


                        self.advantage_stream = nn.Sequential(
                           nn.Linear(self.fc_input_dim, 128),
                           nn.ReLU(),
                           nn.Linear(128, self.output_dim)
                        )
```

```
    def forward(self, state):
        features = self.conv(state)
        features = features.view(features.size(0), -1)
        values = self.value_stream(features)
        advantages = self.advantage_stream(features)
        qvals = values + (advantages - advantages.mean())

        return qvals


    def feature_size(self):
        return self.conv(autograd.Variable(torch.zeros(1,
*self.input_dim))).view(1, -1).size(1)
```

```
# без сверточных сетей
class
DuelingDQN(nn.Module):
```

```
def __init__(self, input_dim, output_dim):
    super(DuelingDQN, self).__init__()
    self.input_dim = input_dim
    self.output_dim = output_dim

    self.feauture_layer = nn.Sequential(
        nn.Linear(self.input_dim[0], 128),
        nn.ReLU(),
        nn.Linear(128, 128),
        nn.ReLU()
    )


    self.value_stream = nn.Sequential(
        nn.Linear(128, 128),
        nn.ReLU(),
        nn.Linear(128, 1)
    )
```

```
            self.advantage_stream = nn.Sequential(
                nn.Linear(128, 128),
                nn.ReLU(),
                nn.Linear(128, self.output_dim)
            )


        def forward(self, state):
            features = self.feauture_layer(state)
            values = self.value_stream(features)
            advantages = self.advantage_stream(features)
            qvals = values + (advantages - advantages.mean())

            return qvals
```

Реализация функции обновления:

```
def com-
pute_loss(self,
batch):
                states, actions, rewards, next_states, dones = batch
                states = torch.FloatTensor(states).to(self.device)
                actions = torch.LongTensor(actions).to(self.device)
                rewards = torch.FloatTensor(rewards).to(self.device)
                next_states = torch.FloatTensor(next_states).to(self.device)
                dones = torch.FloatTensor(dones).to(self.device)


                curr_Q = self.model.forward(states).gather(1, actions.unsqueeze(1))
                curr_Q = curr_Q.squeeze(1)
                next_Q = self.model.forward(next_states)
                max_next_Q = torch.max(next_Q, 1)[0]
                expected_Q = rewards.squeeze(1) + self.gamma * max_next_Q


                loss = self.MSE_loss(curr_Q, expected_Q)

                return loss
```

```
def update(self, batch_size):
    batch = self.replay_buffer.sample(batch_size)
    loss = self.compute_loss(batch)


    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

5.7.2 Average DQN

Алгоритм Averaged-DQN (Алгоритм 2) является расширением алгоритма DQN. Averaged-DQN использует K ранее полученных оценок Q-значений для получения текущей оценки значения действия (строка 5). Алгоритм Averaged-DQN стабилизирует процесс обучения (рисунок 5.3), уменьшая дисперсию целевой погрешности аппроксимации, как мы подробно рассмотрим в разделе 5. Вычислительные затраты по сравнению с DQN состоят в том, что в K-раз больше итераций проходит через Q-сеть при этом минимизируется проигрыш DQN (строка 7). Количество обновлений обратного распространения (которое является самым требовательным вычислительным элементом) остается таким же, как в DQN. Выходные данные алгоритма — это среднее значение за последние K ранее обученных Q-сетей.

Жирные линии представляют собой среднее значение по семи независимым учебным испытаниям. Каждые 1 миллион кадров проводился тест производительности с использованием метода жадного исследования с коэффициентом $\epsilon = 0{,}05$ для 500000 кадров. Затененная область представляет одно стандартное отклонение.

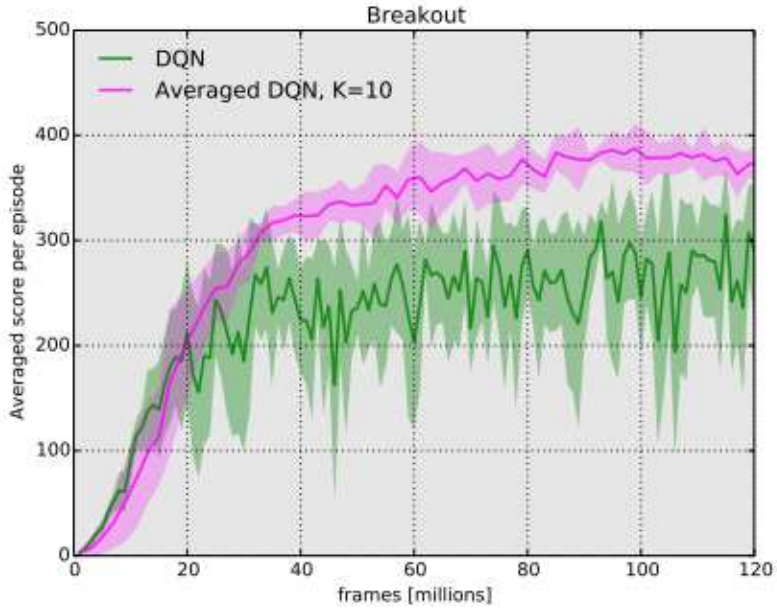


Рисунок 5.3 - Производительность DQN и Averaged-DQN в игре Atari BREAKOUT.

1. Инициализируйте Q (s, a; θ) со случайными весами θ_0
2. Инициализировать буфер воспроизведения опыта (BO) B
3. Инициализация процедуры исследования. Explore (·)
4. Для $i = 1, 2, \ldots, N$ делать
5. $Q_{i-1}^{A}(s,a) = \frac{1}{K}\sum_{k=1}^{K} Q(s,a;\ \theta_{i-k})$
6. $y_{s,a}^{i} = \mathrm{E_B}[r + \ \gamma max_{\alpha'} Q_{I-1}^{A}(s',a')|\ s,a]$
7. $\theta_i \approx argmin_{\theta} \mathrm{E_B}[(y_{s,a}^{i} - Q(s,a;\ \theta))^2]$
8. Изучить (·), обновить B
9. Конец цикла

Вывести $Q_N^A(s,a) = \frac{1}{K}\sum_{k=0}^{K-1} Q(s,a;\ \theta_{N-k})$

5.8 DDPG

Глубокий детерминированный градиент policy (DPG) - это алгоритм, который одновременно изучает Q-функцию и policy. Он использует данные, не относящиеся к policy, и уравнение Беллмана для изучения Q-функции и использует Q-функцию для изучения policy.

Этот подход тесно связан с Q-обучением и мотивирован таким же образом: если вы знаете оптимальную функцию Q*(s,a), то в любом заданном состоянии оптимальное действие a*(s) может быть найдено путем решения (формула 5.13).

$$a^x(s) = arg\max_a Q^*(s,a) \quad (5.13)$$

Алгоритм 1: Глубокий детерминированный градиент policy

Входные данные: начальные параметры policy θ, Q-функциональные параметры φ, пустой буфер воспроизведения D

Установите целевые параметры равными основным параметрам $\theta_{targ} \leftarrow \theta, \varphi_{targ} \leftarrow \varphi$

Повторять

Наблюдайте за состояниями и выбирайте действие $\mathrm{a} = \mathrm{clip}\left(\mu_\theta(\mathrm{s}) + \varepsilon, \mathrm{a}_{\mathrm{Low}}, \mathrm{a}_{\mathrm{High}}\right)$, где $\varepsilon \sim$ N

Выполните a в среде

Наблюдайте за следующим состоянием s', вознаграждение r и завершенный сигнал d, указывающий, является ли s' терминальным

Сохранить (s, a, r, s', d) в буфере воспроизведения D

Если s' является терминальным, сбросьте состояние среды

Если пришло время обновить, **то**

Для каждого обновления **делать**

Произвольная выборка пакета переходов, B={(s, a, r, s', d)} из D

Вычислять целевые показатели по формуле (5.14)

$$\mathrm{y}(\mathrm{r}, \mathrm{s}', \mathrm{d}) = \mathrm{r} + \gamma(1-\mathrm{d}) * \mathrm{Q}_{\varphi\mathrm{targ}}(\mathrm{s}', \mu_{\theta\mathrm{targ}}(\mathrm{s}')) \quad (5.14)$$

Обновите Q-функцию на один шаг градиентного спуска, используя формулу (5.15).

$$\nabla_\varphi \frac{1}{|\mathrm{B}|} \sum_{(\mathrm{s,a,r,s}',\mathrm{d})\in \mathrm{B}} (\mathrm{Q}_\varphi(\mathrm{s},\mathrm{a}) - \mathrm{y}(\mathrm{r},\mathrm{s}',\mathrm{d}))^2 \quad (5.15)$$

Обновите policy на один шаг подъема по градиенту, используя формулу (5.16).

$$\nabla_\varphi \frac{1}{|\mathrm{B}|} \sum_{\mathrm{x}\in \mathrm{B}} \mathrm{Q}_\varphi(\mathrm{s}, \mu_\theta(\mathrm{s})) \quad (5.16)$$

Обновите целевую сеть с помощью формул (5.17), (5.18).

$$\varphi_{\mathrm{targ}} \leftarrow \rho\varphi_{\mathrm{targ}} + (1-\rho)\varphi \quad (5.17)$$

$$\theta_{\mathrm{targ}} \leftarrow \rho\theta_{\mathrm{targ}} + (1-\rho)\theta \quad (5.18)$$

Конец для

Конец если

до сближения

5.9 TD3

В то время как DDPG иногда может достигать высокой производительности, он часто хрупок в отношении гиперпараметров и других видов настройки. Распространенный режим сбоя для DDPG заключается в том, что изученная Q-функция начинает резко завышать значения Q, что затем приводит к нарушению policy, поскольку она использует ошибки в Q-функции. DDPG с двойной задержкой (TD3) - это алгоритм, который решает эту проблему, вводя три критических приема:

Трюк первый: Сокращенное обучение по методу Double-Q. TD3 запоминает две Q-функции вместо одной (отсюда и “двойная”) и использует меньшее из двух Q-значений для формирования целевых значений в функциях потери ошибок Беллмана.

Уловка вторая: “Отложенные” обновления policy. TD3 обновляет policy (и целевые сети) реже, чем Q-функция. В документе рекомендуется одно обновление политики на каждые два обновления Q-функции.

Трюк третий: Целенаправленное сглаживание policy. TD3 добавляет шум к целевому действию, чтобы затруднить policy использование ошибок Q-функции путем сглаживания Q при изменениях в действии.

Алгоритм 1: Двойная задержка DDPG

Входные данные: начальные параметры policy θ, Q-функциональные параметры $\varphi_1,\ \varphi_2$ пустой буфер воспроизведения D

Установите целевые параметры равными основным параметрам $\theta_{targ} \leftarrow \theta, \varphi_{targ1} \leftarrow \varphi_1, \varphi_{targ2} \leftarrow \varphi_2$

Повторять

Наблюдайте за состояниями и выбирайте действие $a = \mathrm{clip}\left(\mu_\theta(s) + \varepsilon, a_{\mathrm{Low}}, a_{\mathrm{High}}\right),$ где $\varepsilon \sim N$

Выполните a в среде

Наблюдайте за следующим состоянием s’, вознаграждение r и завершенный сигнал d, указывающий, является ли s’ терминальным

Сохранить (s, a, r, s’, d) в буфере воспроизведения D

Если s’ является терминальным, сбросьте состояние среды

Если пришло время обновить, **то**

Для j в диапазоне (сколько бы обновлений ни было) **делать**

Произвольная выборка пакета переходов, B={(s, a, r, s’, d)} из D

Вычислять целевые действия по формуле (5.19).

$$a'(s') = clip\big(\mu_{\theta targ}(s') + clip(\varepsilon, -c, c), \mathrm{a_{Low}}, \mathrm{a_{High}}\big), \varepsilon \sim N(0, \sigma) \quad (5.19)$$

Вычислять целевые показатели по формуле (5.20).

$$\mathrm{y(r, s', d) = r + \gamma(1 - d) * \min_{i=1,2} Q_{\varphi targ}(s', a'(s')} \quad (5.20)$$

Обновите Q-функцию на один шаг градиентного спуска, используя формулу (5.21):

$$\nabla_{\varphi} \frac{1}{|\mathrm{B}|} \sum_{(\mathrm{s,a,r,s',d}) \in \mathrm{B}} (\mathrm{Q_{\varphi i}(s, a)} - \mathrm{y(r, s', d)})^2 \quad for\ i = 1,2 \quad (5.21)$$

Если задержка policy j mod = 0, то

Обновите policy на один шаг подъема по градиенту, используя формулу (5.22):

$$\nabla_{\theta} \frac{1}{|\mathrm{B}|} \sum_{\mathrm{x} \in \mathrm{B}} \mathrm{Q_{\varphi i}(s, \mu_{\theta}(s))} \quad (5.22)$$

Обновите целевую сеть с помощью формул (5.23), (5.24):

$$\varphi_{\mathrm{targ}} \leftarrow \rho \varphi_{\mathrm{targ}} + (1 - \rho)\varphi_i \quad for\ i = 1,2 \quad (5.23)$$

$$\theta_{\mathrm{targ}} \leftarrow \rho \theta_{\mathrm{targ}} + (1 - \rho)\theta \quad (5.24)$$

Конец если

Конец для

Конец если

до сближения

5.10 SAC

Soft Actor Critic (SAC) - это алгоритм, который оптимизирует стохастическую policy нестандартным способом, образуя мост между оптимизацией стохастической policy и подходами в стиле DDPG. Он не является прямым преемником TD3 (был опубликован примерно одновременно), но он включает в себя обрезанный трюк с двойным Q, и из-за присущей policy стохастичности в SAC он также выигрывает от чего-то вроде сглаживания policy target.

Центральной особенностью SAC является регуляризация энтропии. Policy обучена максимизировать компромисс между ожидаемой доходностью и энтропией, мерой случайности в policy. Это тесно связано с компромиссом между разведкой и эксплуатацией: увеличение энтропии приводит к увеличению объема исследований, что в дальнейшем может ускорить процесс обучения. Это также может предотвратить преждевременное приближение policy к плохому локальному оптимуму.

Алгоритм 1: Мягкий актер-критик

Входные данные: начальные параметры policy θ, Q-функциональные параметры φ_1, φ_2 пустой буфер воспроизведения D

Установите целевые параметры равными основным параметрам $\varphi_{targ1} \leftarrow \varphi_1$, $\varphi_{targ2} \leftarrow \varphi_2$

Повторять

Observe state s and select action a~$\pi_\theta(* \mid s)$

Выполните действие a в среде

Наблюдайте за следующим состоянием s', вознаграждение r и завершенный сигнал d, указывающий, является ли s' терминальным

Сохранить (s, a, r, s', d) в буфере воспроизведения D

Если s' является терминальным, сбросьте состояние среды

Если пришло время обновить, **то**

Для j в диапазоне (сколько бы обновлений ни было) **делать**

Произвольная выборка пакета переходов, B={(s, a, r, s', d)} из D

Вычислите целевые значения для функций Q (формула 5.25):

$$y(r, s', d) = r + \gamma(1 - d) * (\min_{i=1,2} Q_{\varphi targi}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s'),\ \tilde{a}' \sim \pi_\theta(* \mid s') \quad (5.25)$$

Обновите Q-функцию на один шаг градиентного спуска, используя формулу (5.26):

$$\nabla_\varphi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\varphi i}(s, a) - y(r, s', d))^2 \quad for\ i = 1,2 \quad (5.26)$$

Обновите политику на один шаг подъема по градиенту, используя формулу (5.27):

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (\min_{i=1,2} Q_{\varphi 1}(s, \widetilde{a_\theta}(s)) - \alpha \log \pi_\theta(\widetilde{a_\theta(s)}|s) \quad (5.27)$$

где $\widetilde{a_\theta}(s)$ это образец из$\pi_\theta(* \mid s)$ это выборка, из которой можно дифференцировать wrt θ с помощью метода репараметризации

Обновите целевую сеть с помощью формулы (5.28):

$$\varphi_{targ} \leftarrow \rho \varphi_{targ} + (1 - \rho)\varphi_i \quad for\ i = 1,2 \quad (5.28)$$

Конец для

Конец если

до сближения

5.11 Policy gradients

Рассматривается Марковский процесс принятия решений (МППР), имеющий терминальное состояние. Задача ставится как максимизация суммы всех выигрышей $\mathbf{R=r_0+r_1+\cdots+r_T}$, где T — шаг, на котором произошел переход в терминальное состояние.

Пусть τ некоторый сценарий — последовательность состояний и произведенных в них действий: $\tau=(s_1,a_1,s_2,a_2,...s_T,a_T)$.

Сумма всех выигрышей, полученных в ходе сценария, записывается как (формула 5.29)

$$R_\tau = \sum_\tau r(s_t, a_t) \tag{5.29}$$

Не все сценарии равновероятны. Вероятность реализации сценария зависит от поведения среды, которое задается вероятностями перехода между состояниями $p(s_{t+1}|s_t,a_t)$, распределением начальных состояний $p(s_1)$ и поведения агента, которое определяется его стохастической стратегией $\pi_\theta(a_t|s_t)$. Вероятностное распределение над сценариями задается как (формула 5.30):

$$p_\theta(\tau) = p_\theta(s_1, a_1, \dots s_T, a_T) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t), \tag{5.30}$$

Предполагаем, что вероятности переходов между состояниями агенту неизвестны, то есть у агента нет модели поведения окружающей среды (model-free learning).

Нужно выбрать такой набор параметров агента θ, задающий $\pi_\theta(a|s)$, чтобы максимизировать матожидание суммы полученных выигрышей (формула 5.31):

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}[R_\tau] = \int p_\theta(\tau)R_\tau d\tau, \text{->max} \tag{5.31}$$

Будем максимизировать функцию J(θ) методом градиентного подъема (формула 5.32):

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|\theta_k \tag{5.32}$$

Для этого необходимо рассчитывать ее градиент (формула 5.33):

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau)R_\tau d_\tau \tag{5.33}$$

Мы не можем подсчитать $\nabla_\theta p_\theta(\tau)$ напрямую, потому что в выражение для $p_\theta(\tau)$ входят вероятности переходов между состояниями, которые агенту неизвестны. Однако, так как (формула 5.34):

$$p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau), \tag{5.34}$$

То можно сделать замену (здесь $p_\theta(\tau) = P(\tau \vee \theta)$) (формула 5.35):

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P\,(\tau|\theta) \quad (5.35)$$

Отсюда (формула 5.36)

$$\nabla_\theta J(\theta) = \int p_\theta(\tau)\,\nabla_\theta \log p_\theta(\tau) R_\tau d\tau = E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)\, R_\tau], \quad (5.36)$$

Теперь нужно рассмотреть градиент $\nabla_\theta log p_\theta(\tau)$ (формулы 5.37 и 5.38).

$$\log p_\theta(\tau) = \log(p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)) = \log p(s_1) + \sum_{t=1}^{T}(\log\pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t,a_t)), \quad (5.37)$$

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \log p_\theta(s_1) + \sum_{t=1}^{T}(\nabla_\theta \log\pi_\theta(a_t|s_t) + \nabla_\theta \log p(s_{t+1}|s_t,a_t)) = \sum_{t=1}^{T}\nabla_\theta \log\pi_\theta(a_t|s_t) \quad (5.38)$$

Отсюда (формула 5.39)

$$\nabla_\theta J(\theta) = E_{\tau\sim p_\theta(\tau)}[(\sum_{t=1}^{T}\nabla_\theta \log\pi_\theta(a_t|s_t))R_\tau] \quad (5.39)$$

Весь расчет

! Производная для градиента базовой policy		
$\nabla_\theta J(\pi_\theta) = \nabla_{\theta_{\tau\sim\pi_\theta}} E[R(\tau)]$		
	$=\nabla_\theta \int_\tau P(\tau\|\theta)R(\tau)$	Расширение ожидания
	$=\int_\tau \nabla_\theta P(\tau\|\theta)R(\tau)$	Привести градиент к интегралу
	$=\int_\tau P(\tau\|\theta)\nabla_\theta \log P(\tau\|\theta)R(\tau)$	Лог-производная
	$=\underset{\tau\sim\pi_\theta}{E}[\nabla_\theta \log P(\tau\|\theta)R(\tau)]$	Возврат к форме ожидания
$\therefore \nabla_\theta J(\pi_\theta)$	$=\underset{\tau\sim\pi_\theta}{E}[\sum_{t=0}^{T}\nabla_\theta \log\pi_\theta(a_t\|s_t)R(\tau)]$	Выражение для grad-log-prob

Оценку среднего по различным сценариям можно провести методом Монте-Карло (формула 5.40):

$$\hat{g} = \frac{1}{|\mathrm{D}|}\sum_{\tau\in\mathrm{D}}\sum_{t=0}^{T}\nabla_\theta \log\pi_\theta(a_t|s_t)R(\tau) \quad (5.40)$$

$\mathrm{D} = \{\tau_i\}_{i=1..N}$ - это количество траекторий, всего N траекторий. В данном выражении вероятности переходов между состояниями уже не входят в расчет, таким образом их не нужно знать.

Таким образом имея N сценариев можно посчитать величину градиента (формула 5.41):

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log\pi_\theta\left(a_t^i|s_t^i\right)\right)R_{\tau^i} = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T^i}\nabla_\theta \log\pi_\theta\left(a_t^i|s_t^i\right)\right)\left(\sum_{t=1}^{T^i} r\left(a_t^i|s_t^i\right)\right) \quad (5.41)$$

Очевидно такой методы расчета очень долгий, кроме того необходимо считать до терминального состояния.

Таким образом, оптимизировать $J(\theta)$ можно с помощью следующего простого алгоритма (REINFORCE):

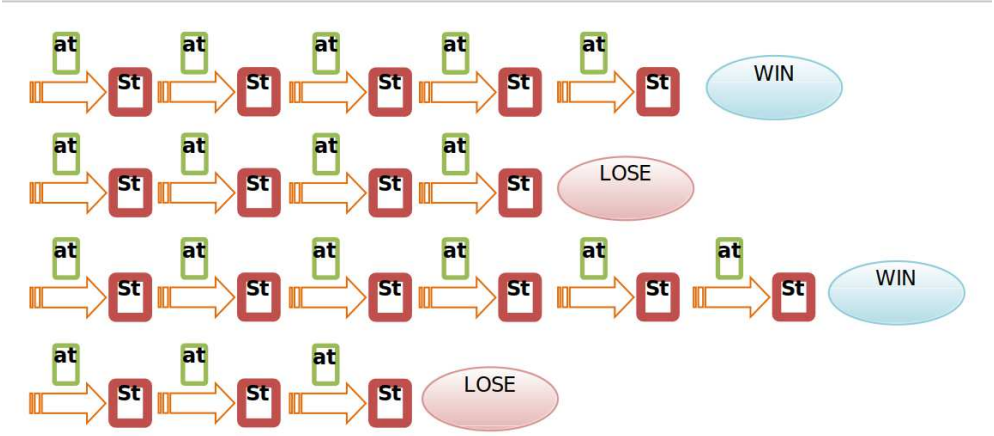
1. Прогнать N сценариев τ_i со стратегией $\pi_\theta(a|s)$;

2. Посчитать среднее арифметическое

$$\nabla_\theta J(\theta) \leftarrow \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T^i}\nabla_\theta \log \pi_\theta\left(a_t^i|s_t^i\right)\right)\left(\sum_{t=1}^{T^i} r\left(s_t^i, a_t^i\right)\right)$$

3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

4. Если не сошлись к экстремуму, повторить с пункта 1.



Интуитивное объяснение обучения с подкреплением методов без знания модели. Повышаем правдоподобие сценариев с высоким значением выигрыша и понижаем с низким.

Преимущества:

- Легко обобщается на задачи с большим множеством действий, в том числе на задачи с непрерывным множеством действий;
- По большей части избегает конфликта между эксплуатацией (exploitation) и исследованием (exploration), так как оптимизирует напрямую стохастическую стратегию $\pi\theta(a|s)$;
- Имеет более сильные гарантии сходимости: если Q-learning гарантированно сходится только для МППР с конечными множествами действий и состояний, то policy

gradient, при достаточно точных оценках $\nabla\theta J(\theta)$ (т. е. при достаточно больших выборках сценариев), сходится к локальному оптимуму всегда, в том числе в случае бесконечных множеств действий и состояний, и даже для частично наблюдаемых Марковских процессов принятия решений (ЧНМППР, англ. *partially observed Markov decision process, POMDP*).

Недостатки:

- Очень низкая скорость работы — требуется большое количество вычислений для оценки $\nabla_\theta J(\theta)$ по методу Монте-Карло, так как:
- для получения всего одного семпла требуется произвести T взаимодействий со средой;
- случайная величина $\nabla_\theta \log p_\theta(\tau) R_\tau$ имеет большую дисперсию, так как для разных τ значения R_τ могут очень сильно различаться, поэтому для точной оценки $\nabla_\theta J(\theta) = E_{\tau \sim p\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) R_\tau]$ требуется много семплов;
- семплы, собранные для предыдущих значений θ, никак не переиспользуются на следующем шаге, семплирование нужно делать заново на каждом шаге градиентного спуска.
- В случае конечных МППР Q-learning сходится к глобальному оптимуму, тогда как policy gradient может застрять в локальном.

Способы усовершенствования алгоритма:

Первый способ основан на использовании опорного значения (формула 5.42).

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)\right)\right] \quad (5.42)$$

Если b - константа относительно τ, то (формула 5.43):

$$E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(R_\tau - b)] = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) R_\tau] \quad (5.43)$$

Это вытекает из формулы 5.44:

$$E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) b] = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) b d\tau = \int \nabla_\theta p_\theta(\tau) b d\tau = b \nabla_\theta \int p_\theta(\tau) dt = b \nabla_\theta 1 = 0 \quad (5.44)$$

Таким образом, если выигрыш меняется на константу это не меняет среднюю оценку градиента выигрыша. Что, кстати, показывает возможность изменения наград на константу при действиях.

Дисперсия меняется

Но при этом изменяется дисперсия градиента, что дает возможность снижать эту дисперсию регулируя опорное значение b и улучшать сходимость (формула 5.45).

$$Var_{\tau \sim p_\theta(\tau)[\nabla_\theta \log p_\theta(\tau)(R_\tau - b)] = E_{\tau \sim p_\theta(\tau)}\left[(\nabla_\theta \log p_\theta(\tau)(R_\tau - b))^2\right] - E_{\tau \sim p_\theta(\tau)}\left[(\nabla_\theta \log p_\theta(\tau)(R_\tau - b))^2\right]} \quad (5.45)$$

Где $E_{\tau\sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(R_\tau - b))^2]$ *первое зависит от b,*

$E_{\tau\sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(R_\tau - b))^2]$ *второе* $= E[\nabla_\theta \log p_\theta(\tau) R_\tau]^2$

5.11.1 Использование будущего выигрыша вместо полного выигрыша

Рассмотрим выражение для градиента полного выигрыша (формула 5.46):

$$\nabla_\theta J(\theta) = E_{\tau\sim p_\theta(\tau)}[\,(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))R_\tau\,] = E_{\tau\sim p_\theta(\tau)}[\,(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=1}^{T} r(s_t,a_t))] \quad (5.46)$$

Так как в момент времени t от действия a_t зависят только $r(s_{t'},a_{t'})$ для $t'\leq t$, это выражение можно переписать как (формула 5.47)

$$\nabla_\theta J(\theta) \approx E_{\tau\sim p_\theta(\tau)}\Big[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Big(\sum_{t'=t}^{T} r(s_{t'},a_{t'})\Big)\Big] \quad (5.47)$$

Величина $Q_{\tau,t}$ — *будущий выигрыш* (reward-to-go) на шаге t в сценарии τ.

5.12 Алгоритм актора-критика с преимуществом (англ. Advantage Actor Critic, A2C)

Используем полученное приближение (формула 5.48):

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\big(a_t^i|s_t^i\big) Q_{\tau_i,t} \quad (5.48)$$

$Q_{\pi,t}$ — оценка будущего выигрыша из состояния s^i_t при условии действия a^i_t, которая базируется только на одном сценарии τ_i.

Это плохое приближение ожидаемого будущего выигрыша — истинный ожидаемый будущий выигрыш выражается формулой (5.49).

$$Q^\pi(s_t,a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'},a_{t'})|s_t,a_t] \quad (5.49)$$

Вместо ожидаемого будущего выигрыша при оценке $\nabla_\theta J(\theta)$ будем использовать функцию *преимущества* (advantage) $A^\pi(s_t,a_t)$ (формула 5.50):

$$A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t) \quad (5.50)$$

Преимущество действия a_t в состоянии s_t — величина, характеризующая то, насколько выгоднее в состоянии s_t выбрать именно действие a_t. Такое допустимо так как мы применяем метод опорного значения, вычитая константу.

Таком образом используем формулу (5.51):

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\big(\big(a_t^i|s_t^i\big) A^\pi(s_t^i,a_t^i) \quad (5.51)$$

Как достаточно точно и быстро оценить $A_\pi(s^i_t,a^i_t)$, Сведем задачу к оценке $V_\pi(s_t)$ (формулы 5.52 и 5.53):

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + E_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}[V^\pi(s_{t+1})] \approx r(s_t, a_t) + V^\pi(s_{t+1}), \quad (5.52)$$

$$A^\pi(s^i_t, a^i_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t) \quad (5.53)$$

Теперь нужно уметь оценивать (формула 5.54):

$$V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t] \quad (5.54)$$

Можно делать это с помощью метода Монте-Карло — так мы получим несмещенную оценку. Но это будет работать очень медленно, не существенно быстрее, чем обычный policy gradient.

Заметим, что при фиксированных s_t и a_t выполняется (формула 5.55):

$$V^\pi(s_t) = r(s_t, a_t) + V^\pi(S_{t+1}) \quad (5.55)$$

Таким образом, если мы имеем некоторую изначальную оценку $V_\pi(s)$ для всех s, то мы можем обновлять эту оценку путем, аналогичным алгоритму Q-learning (формула 5.56):

$$V^\pi(S_t) \leftarrow (1-\beta)V^\pi(S_t) + \beta(r(s_t, a_t) + V^\pi(s_{t+1})) \quad (5.56)$$

Здесь β — коэффициент обучения (*learning rate*) для функции ценности. Такой пересчет мы можем производить каждый раз, когда агент получает вознаграждение за действие. Так мы получим оценку ценности текущего состояния, не зависящую от выбранного сценария развития событий τ, а значит, и оценка функции преимущества не будет зависеть от выбора конкретного сценария.

Это сильно снижает дисперсию случайной величины $\nabla_\theta \log\pi_\theta(a_{it}|s_{it})A_\pi(s_{it},a_{it})$, что делает оценку $\nabla_\theta J(\theta)$ достаточно точной даже в том случае, когда мы используем всего один сценарий для ее подсчета (формула 5.57):

$$\nabla_\theta J(\theta) \approx \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\,(a_t|s_t)A^\pi(s_t, a_t) \quad (5.57)$$

На практике же мы можем аппроксимировать $\nabla_\theta J(\theta)$ на каждом шаге (в онлайне), основываясь на всего одном действии каждый раз.

Алгоритм в итоге будет следующим:

1. производим действие $a \square \pi_\theta(a|s)$, переходим в состояние s' и получаем вознаграждение r

2. $V_\pi(s) \leftarrow (1-\beta)V_\pi(s)+\beta(r+V_\pi(s'))$

3. $A_\pi(s,a) \leftarrow r + V_\pi(s') - V_\pi(s)$

4. $\nabla_\theta J(\theta) \leftarrow \nabla_\theta \log \pi_\theta(a|s) A_\pi(s,a)$

5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

6. Если не сошлись к экстремуму, повторить с пункта 1.

Такой алгоритм называется алгоритмом актора-критика с преимуществом (Advantage Actor-Critic). Актором здесь называется компонента, которая оптимизирует стратегию $\pi_\theta(a|s)$, а критиком — компонента, которая подсчитывает ценности состояний $V_\pi(s)$.

Актор определяет дальнейшее действие, а критик оценивает, насколько то или иное действие выгодно, основываясь на функции преимущества (advantage).

Алгоритм актора-критика считается гибридным, так как актор работает в соответствии с принципом policy gradient, а критик работает аналогично алгоритму Q-routing.

5.13 Асинхронный актор-критик (англ. Asynchronous Advantage Actor-Critic, A3C)

Проблема с обучением с подкреплением в онлайне заключается в том, что данные, поступающие на вход алгоритму обучения, сильно скоррелированы: каждое следующее состояние непосредственно зависит от предпринятых агентом действий. Обучение на сильно скоррелированных данных приводит к переобучению. Таким образом, для того, чтобы успешно обучить стратегию, обобщаемую на большое количество состояний среды, нам все еще необходимо обучаться на эпизодах из различных сценариев. Одним из способов достичь этого является запуск множества агентов параллельно. Все агенты находятся в разных состояниях и выбирают различные конкретные действия согласно стохастической стратегии $\pi\theta(a|s)$, тем самым достигается устранение корреляции между наблюдаемыми данными. Однако все агенты используют и оптимизируют один и тот же набор параметров θ.

Идея алгоритма асинхронного актора-критика заключается в том, чтобы запустить N агентов параллельно, при этом на каждом шаге каждый из агентов рассчитывает обновления для значений $V_\pi(s)$ и θ. Однако, вместо того, чтобы просто продолжить работу, каждый агент обновляет $V_\pi(s)$ и θ, общие для всех агентов. Перед обработкой каждого нового эпизода агент копирует текущие глобальные значения параметра θ и использует его, чтобы определить собственную стратегию на этот эпизод. Агенты не ждут, пока остальные агенты завершат обработку своих эпизодов, чтобы обновить глобальные параметры (отсюда *асинхронный*). Поэто-

му пока один из агентов обрабатывает один эпизод, глобальное значение θ может изменяться вследствие действий других агентов.

В большинстве современных исследований стратегия $\pi_\theta(a|s)$ и функция ценности $V_\pi(s)$ задаются с помощью нейросетей. Каждая из функций может в принципе использовать отдельную нейросеть, но на практике чаще всего применяется совмещенная нейросеть с двумя выходными слоями — для стратегии и для функции ценности. Такой подход, как правило, приводит к лучшим результатам, так как функция ценности, вообще говоря, зависит от текущей стратегии.

Реализация алгоритма асинхронного актора-критика инициализирует глобальную нейросеть (master network) и запускает N дочерних процессов (workers), в каждом из которых агент взаимодействует со средой. Нейросеть каждого агента является копией материнской нейросети. Перед началом каждого эпизода веса из материнской нейросети заново копируются в нейросеть агента. Градиенты, посчитанные по агентской нейросети, применяются в итоге к материнской.

5.14 Алгоритм TRPO (trust region policy optimization)

TRPO обновляет policy, делая максимально возможный шаг для повышения производительности, при этом соблюдая специальное ограничение на то, насколько близки новые и старые policy, которым разрешено быть. Ограничение выражается в терминах KL-дивергенции, меры (что-то вроде, но не совсем) расстояния между распределениями вероятностей.

Это отличается от обычного градиента policy, при котором новые и старые policy находятся рядом в пространстве параметров. Но даже кажущиеся незначительными различия в пространстве параметров могут иметь очень большие различия в производительности — так что один неверный шаг может привести к снижению производительности policy. Это делает опасным использование больших размеров шага с градиентами ванильной policy, тем самым снижая эффективность выборки. TRPO позволяет избежать такого рода сбоев и, как правило, быстро и монотонно повышает производительность.

Краткие факты:

- TRPO - это алгоритм, основанный на policy.
- TRPO может использоваться для сред с дискретными или непрерывными пространствами действий.

- Расширенная реализация TRPO поддерживает распараллеливание с MPI.

Ключевые уравнения

Пусть π_θ обозначим политику параметрами θ. Теоретическое обновление TRPO является (формулы 5.58, 5.59):

$$\theta_{k+1} = arg\max_\theta \tau(\theta_k, \theta) \tag{5.58}$$

$$\text{s.t.} \quad \overline{D}_{KL}(\theta||\theta_k) \leq \delta \tag{5.59}$$

где $\tau(\theta_k, \theta)$ находится *суррогатное преимущество*, показатель того, как policy π_θ работает по сравнению со старой политикой π_{θ_k}, используя данные из старой policy (формула 5.60):

$$\tau(\theta_k, \theta) = \underset{s,a\sim\pi_{\theta_k}}{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)\right] \tag{5.60}$$

и $\overline{D}_{KL}(\theta||\theta_k)$ является средним KL расхождением между policy в разных государствах, посещенных старой policy (формула 5.61):

$$\overline{D}_{KL}(\theta||\theta_k) = \underset{s,a\sim\pi_{\theta_k}}{E}[D_{KL}(\pi_\theta(\cdot|s)\|\pi_{\theta_k}(\cdot|s))] \tag{5.61}$$

Теоретическое обновление TRPO не самое простое в работе, поэтому TRPO делает некоторые приближения, чтобы быстро получить ответ. Мы, Тейлор, расширяем цель и ограничения, чтобы навести порядок вокруг θ_k (формулы 5.62 и 5.63):

$$\tau(\theta_k, \theta) \approx g^T(\theta - \theta_k) \tag{5.62}$$

$$\overline{D}_{KL}(\theta||\theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \tag{5.63}$$

в результате возникает проблема приблизительной оптимизации (формулы 5.64 и 5.65):

$$\theta_{k+1} = \arg\max_\theta g^T(\theta - \theta_k) \tag{5.64}$$

$$\text{s.t.}\ \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta \tag{5.65}$$

Эта приближенная задача может быть аналитически решена методами лагранжевой двойственности, что дает решение (формула 5.66):

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{5.66}$$

Если бы мы остановились на этом и просто использовали этот конечный результат, алгоритм точно рассчитал бы естественный градиент policy. Проблема заключается в том, что

из-за ошибок аппроксимации, вносимых расширением Тейлора, это может не удовлетворять ограничению KL или фактически улучшать суррогатное преимущество. TRPO добавляет модификацию к этому правилу обновления: поиск по строке с обратным отслеживанием (формула 5.67),

$$\theta_{k+1} = \theta_k + a^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{5.67}$$

где $\propto \in (0,1)$ - коэффициент обратного отслеживания, а j - наименьшее неотрицательное целое число, такое, что $\pi_{\theta_{k+1}}$ удовлетворяет ограничению KL и дает положительное суррогатное преимущество.

Наконец: вычисление и хранение обратной матрицы H^{-1} , является чрезвычайно дорогостоящим процессом при работе с policy нейронной сети с тысячами или миллионами параметров. TRPO обходит проблему, используя алгоритм сопряженного градиента для решения $Hx = g$ для $x = H^{-1}g$, требующий только функции, которая может вычислять произведение матрицы на вектор Hx вместо непосредственного вычисления и сохранения всей матрицы H. Это не так уж сложно сделать: мы настраиваем символьную операцию для вычисления (формула 5.68):

$$Hx = \nabla_\theta((\nabla_\theta \bar{D}_{KL}(\theta \,\|\theta_k))^T x) \tag{5.68}$$

которая дает нам правильный вывод без вычисления всей матрицы

5.15 Алогритм PPO (Proximal polcy optimization)

5.15.1 Реализация PPO.

```
# proximal policy optimization

import numpy
import time
from random import choices
import tensorflow as tf
import gym
import cv2


print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpus = tf.config.list_physical_devices('GPU')
print("GPUs Available: ", gpus)
```

```
tf.config.experimental.set_visible_devices(gpus[0], 'GPU')

import tensorflow.keras as keras
from tensorflow.keras.layers import Dense, Input, concatenate, BatchNormalization, Dropout,
Conv1D, Reshape, Flatten
import tensorflow as tf


config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True

#tf.config.run_functions_eagerly(True)
sess = tf.compat.v1.Session(config=config)
sess.as_default()


class environment():
    def __init__(self):
        #self.env = gym.make("Breakout-ram-v0")
        self.env = gym.make("Breakout-ram-v0")
        n_action = self.env.action_space.n

        self.step = 0
        self.acts = numpy.zeros(n_action)
        self.acts[0] = 1.0
        self.reward = 0
        self.index = 0

        self.state_max = self.env.observation_space.high.max()
        self.state_min = self.env.observation_space.low.min()
        self.igame  = 0
        self.xnew = []
        self.ynew = []
        self.observ = [[],[],[],[]]
        self.ind_obs = 0
        self.n_obs = 1
        self.env.reset()
        self.env_reset()
        self.state()



    def get_state(self, act):
        self.igame = self.igame + 1
        self.step = self.step + 1
        if(self.index == 0):
            self.reward  = 0
```

```
        self.ind_obs = 0
        next_observation, reward, done, info = self.env.step(act)

        for i in range(3):
            self.observ[i] = self.observ[i+1]
        self.observ[-1] = next_observation

        self.field = numpy.concatenate(self.observ)

        self.reward = self.reward+reward
        self.index = self.index + 1
        if done:
            self.env_reset()

        return done, self.state(), reward

    def env_reset(self):
        self.index = 0

        next_observation = self.env.reset()
        self.observ[0] = next_observation
        self.observ[1] = next_observation
        self.observ[2] = next_observation
        self.observ[3] = next_observation
        self.field = numpy.concatenate(self.observ)
    def get_image(self):
        return self.env.render(mode='rgb_array')

    def state(self):
        state = self.field.flatten()
        return state/255.0-0.5#(( state - self.state_min) / (self.state_max - self.state_min)-0.5)*2

    def field(self):
        return self.field

    def get_len_state(self):
        return len(self.state())


class multi_environment:
    def __init__(self,n = 1):
        self.N = n
        self.envs = [environment() for i in range(n)]

    def get_len_state(self):
        return len(self.envs[0].state())
    def get_len_acts(self):
```

```
        return len(self.envs[0].acts)

    def get_step(self, i, act):
        return self.envs[i].get_state(act)

    def get_cur_state(self, i):
        return self.envs[i].state()

    def get_image(self,i):
        return self.envs[i].get_image()


class ppo:
    def __init__(self,envs):
        self.envs = envs
        self.index = 0

        self.max_size = self.index
        self.flag = False

        self.len_state = envs.get_len_state()
        self.len_act = envs.get_len_acts()
        self.gamma = 0.99


        self.max_t = 10
        self.start_time = time.time()

        self.T = 64
        self.N = envs.N


        self.rews = numpy.zeros((self.N, self.T),dtype=numpy.float32)
        self.acts = numpy.zeros((self.N, self.T),dtype=numpy.int32)
        self.policies = numpy.zeros((self.N,self.T, self.len_act),dtype=numpy.float32)
        self.states = numpy.zeros((self.N, self.T, self.len_state),dtype=numpy.float32)
        self.previous_states = numpy.zeros((self.N, self.T, self.len_state),dtype=numpy.float32)
        self.dones = numpy.zeros((self.N,self.T),dtype=numpy.float32)
        self.cur_rewards = numpy.zeros((self.N),dtype=numpy.int32)
        self.NSTEP = 100
        self.all_rewards = numpy.zeros((self.N, self.NSTEP),dtype=numpy.int32)
        self.cur_step = numpy.zeros((self.N),dtype=numpy.int32)


        inp = Input(shape = ( self.len_state,))
        lay = Dense(self.len_state//2,activation = "tanh") (inp)
        lay = Dense(self.len_state//4,activation = "relu") (lay)
```

```
    lay = Dense(self.len_state//10,activation = "linear") (lay)
    lay = Dense(self.len_act, activation="softmax") (lay)
    self.model = keras.Model(inputs=inp, outputs=[lay])


    inp = Input(shape = ( self.len_state,))
    lay = Dense(self.len_state//2,activation = "tanh") (inp)
    lay = Dense(self.len_state//4,activation = "relu") (lay)

    lay = Dense(self.len_state//10,activation = "tanh") (lay)
    layv = Dense(1,activation = "linear", name = "vvalue") (lay)
    self.model_v = keras.Model(inputs=inp, outputs=[layv])

    self.optimizer = tf.keras.optimizers.Adam(learning_rate=0.0002)
    self.optimizerv = tf.keras.optimizers.Adam(learning_rate=0.0002)

    self.nwin = "Main"
    cv2.namedWindow(self.nwin)
    cv2.setMouseCallback(self.nwin,self.capture_event)
    self.show_on = True

def capture_event(self,event,x,y,flags,params):
    if event==cv2.EVENT_LBUTTONDBLCLK:
        self.show_on = not self.show_on
        print("St")


@tf.function
def get_net_res(self,l_state):
    out = self.model(l_state, training = False)
    return out

@tf.function
def get_value_res(self,l_state):
    val = self.model_v(l_state, training = False)
    return val

def get_net_act(self,l_state):

    out = self.get_net_res(l_state)

    i = choices([i for i in range(self.len_act)], out[0])[0]
    return out[0], i

def calc_value(self, vnext):
    T = self.T
    N = self.N
```

```
    vst = numpy.zeros((N, T))
    vst[:,T-1] = self.rews[:,T-1]+vnext[:,T-1]*self.gamma*(1-self.dones[:,T-1])
    for i in range(T-2,-1,-1):
        vst[:,i] = self.rews[:,i]+self.gamma*vst[:,i+1]*(1-self.dones[:,i])
    return vst

def calc_advantage(self, vst, value):
    T = self.T
    sig = vst - value
    #adv = numpy.zeros(sig.shape)
    lam = 0.99
    #for i in range(T-2, -1, -1):
    #    adv[:,i] = sig[:,i] + lam*self.gamma*adv[:,i+1]
    adv = sig
    #advm = adv.mean(axis=1).reshape(adv.shape[0],1)
    #adv_minus_mean = adv - advm
    #advm2 = (((adv_minus_mean)**2)**0.5).mean(axis=1).reshape(adv.shape[0],1)
    #adv = adv_minus_mean/(advm2+1e-10)

    return adv

@tf.function
def train_critic(self,inp, vst, valold):
    with tf.GradientTape() as tape:
        v = self.model_v(inp, training = True)

        vclip = tf.square(tf.clip_by_value(v-valold,-0.3,0.3))
        vdif = tf.square(v - vst)
        vd = tf.maximum(vclip,vdif)

        loss_value = tf.reduce_mean(vd)
        trainable_vars = self.model_v.trainable_variables
    grads = tape.gradient(loss_value, trainable_vars)
    self.optimizerv.apply_gradients(zip(grads, trainable_vars))
    return

@tf.function
def train_actor(self, inp, adv, acts, pol):
    with tf.GradientTape() as tape:
        y_pi = self.model(inp, training = True)
        i_nums = tf.range(0,y_pi.shape[0])
        ind = tf.cast(tf.transpose(tf.stack([i_nums, acts])),tf.int32)
        advm = tf.reduce_mean(adv)
        adv2 = tf.sqrt(tf.reduce_mean(tf.square(adv-advm)))
        advs = tf.divide(adv-advm, adv2+1e-9)
        y_pi2 = tf.gather_nd(y_pi,ind)
        y_old = tf.gather_nd(pol, ind)
        rel = tf.divide(y_pi2,y_old+1e-9)
```

```
        relclip = tf.clip_by_value(rel,0.9,1.1)

        relmin = tf.minimum(rel*tf.stop_gradient(advs),relclip*tf.stop_gradient(advs))
        loss_value = tf.reduce_sum(-relmin)
        entr = tf.reduce_sum(y_pi*tf.math.log(y_pi+1e-25),axis=1)
        kb = tf.reduce_sum((y_pi-pol)*tf.math.log((y_pi+1e-25)/(pol+1e-25)),axis=1)

        loss_value = loss_value + 0.01*tf.reduce_sum(entr)
        trainable_vars = self.model.trainable_variables
    grads = tape.gradient(loss_value, trainable_vars)
    #grads, gnorm = tf.clip_by_global_norm(grads, 0.4)
    self.optimizer.apply_gradients(zip(grads, trainable_vars))
    return tf.reduce_mean(kb)

def learn_all(self):
    T = self.T
    N = self.N

    prev_st = numpy.concatenate([self.previous_states,self.states[:,-1:]],axis=1)
    prev_states = prev_st.reshape(N*(T+1), prev_st.shape[-1])
    res  = self.get_value_res(prev_states).numpy()
    res = res.reshape(N, T+1)
    resst = res[:,0:-1]
    resstnext = res[:,1:]
    pstates = self.previous_states.reshape(N*T, self.previous_states.shape[-1])

    #vst = self.rews+self.gamma*resstnext*(1-self.dones)
    vst =  self.calc_value(resstnext)


    adv = self.calc_advantage(vst, resst)

    adv = adv.reshape(N*T)
    acts = self.acts.reshape(N*T)
    pol = self.policies.reshape(N*T, self.len_act)


    EP = 4
    S = N*T//EP

    for i in range(EP):

        index = list(range(i*S,S*(i+1)))
        st_c = tf.cast(pstates[index] ,tf.float32)
        adv_c = tf.cast(adv[index] ,tf.float32)
        acts_c = tf.cast(acts[index] ,tf.int32)
        pol_c = tf.cast(pol[index] ,tf.float32)
```

```
            ret = self.train_actor(st_c, adv_c, acts_c, pol_c)
            #print(ret)

        vst_c =  tf.cast(vst.reshape(N*T), tf.float32)
        vals = tf.cast(resst.reshape(N*T),tf.float32)
        pstates_c = tf.cast(pstates,tf.float32)
        self.train_critic(pstates_c, vst_c, vals)
        return

    def step(self):
        for i in range(self.N):
            for j in range(self.T):
                prev_state = envs.get_cur_state(i)

                policy, act = self.get_net_act(numpy.expand_dims(prev_state, axis=0))

                done, state, reward = envs.get_step(i, act)

                self.previous_states[i,j] = prev_state
                self.rews[i,j] = reward
                self.dones[i,j] = done
                self.policies[i,j] = policy
                self.states[i,j] = state
                self.acts[i,j] = act
                self.cur_rewards[i] = self.cur_rewards[i]+reward
                if done:
                    self.all_rewards[i,self.cur_step[i]] = self.cur_rewards[i]
                    self.cur_rewards[i] = 0
                    self.cur_step[i]=self.cur_step[i]+1
                    if(self.cur_step[i]>=self.NSTEP):
                        self.cur_step[i] = 0
                if(i==0 and self.show_on):
                    self.show(0)

        self.learn_all()

        if self.index%10==0:

            print(f"step is {self.index}")
            print(f"rewards sum is {self.all_rewards.sum(axis=1).mean()}  rewards mean
{self.all_rewards.mean(axis=1).mean()}")
            print(f"policy  {self.policies[0,-1]}")
            #res1 = self.get_value_res(self.states[0,0:10]).numpy().flatten()
            #res2 = self.get_value_res(self.states[0,-10:]).numpy().flatten()
            #print(f"valuef {res1}")
            #print(f"valuef {res2}")
        self.index = self.index +  1
```

```
        return

    def show(self, i):
        cv2.imshow(self.nwin,self.envs.get_image(i))
        if cv2.waitKey(1)==27:
            print("27")
        return

N = 48
envs = multi_environment(N)
ppo1 = ppo(envs)
for i in range(100000000):
    ppo1.step()
```

6 ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ ПО ЯЗЫКОВЫМ МОДЕЛЯМ

Языковые модели — это программные алгоритмы, которые используются для обработки естественного языка. Они используют статистические методы для анализа и понимания языка, а также для генерации новых текстов на основе имеющихся данных.

Языковые модели могут быть обучены на больших объемах текстовых данных и использоваться для различных задач, таких как машинный перевод, классификация текстов, генерация текста и многое другое. Они играют важную роль в обработке естественного языка и находят широкое применение в различных областях, включая поисковые системы, чат-боты, анализ социальных медиа и многое другое.

6.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) — это языковая модель, разработанная компанией Google, которая использует технологию трансформеров для обработки естественного языка. Она была представлена в 2018 году исследователями Якобом Девлином, Минг-Вей Ченгом, Кентоном Ли и Кристиной Тутановой (Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova) и является одной из самых мощных языковых моделей на сегодняшний день.

BERT обучается на больших объемах текстовых данных и может использоваться для решения различных задач, таких как классификация текстов, ответы на вопросы, генерация текста и многое другое. Он стал одним из самых важных инструментов в обработке естественного языка и находит широкое применение в различных областях, включая поисковые системы, чат-боты, анализ социальных медиа и многое другое. BERT также стал основой для многих последующих языковых моделей, таких как GPT-2 и RoBERTa.

В отличие от других моделей языкового представления, BERT предназначен для предварительной подготовки глубоких двунаправленных представлений из неразмеченного текста путем совместной обработки как левого, так и правого контекста на всех уровнях. В результате предварительно обученная модель BERT может быть точно настроена с помощью всего лишь одного дополнительного слоя для создания современных моделей для широкого круга задач, таких как ответы на вопросы и языковой вывод, без существенных изменений архитектуры для конкретных задач.

Новый слой нейронов имеет тот же формат, что старые слои: он имеет классифицирующий токен C – предсказывая его, модель также предсказывает, как связаны между собой предложения (определяет их в класс противоречащих, взаимно подтверждающих или нейтральных). Нейросеть учитывает свое предсказание о том, следуют ли два предложения друг за другом (с помощью токена [CLS] с этапа предобучения) и на эмбеддинги слов. Вся эта информация подсказывает, как нужно классифицировать предложения в новой задаче (рисунок 6.1).

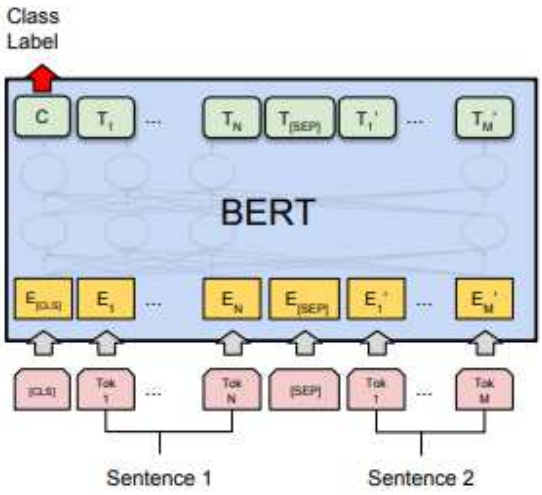


Рисунок 6.1 – Представление задачи классификации предложения

Задачи классификации пар предложений решаются с помощью предсказания токена C на основе проверки BERT. Эти задачи могут быть разными, например, QQP - задача на определение одинаковых по смыслу вопросов с Quora, где вопросы классифицируются как "одинаковые" или "неодинаковые". Еще одна задача - QNLI, где модель должна правильно определить, содержит ли пара предложений вопрос и подходящий к нему ответ, классифицируя их как "содержит" или "не содержит". Задача SWAG заключается в выборе логически подходящего продолжения к заданной фразе из четырех предложенных вариантов.

Каждый раз, когда создаются новые данные, их формат подстраивается под формат данных, используемых для предварительного обучения. Например, в задаче SWAG используются четыре входных предложения, каждое из которых состоит из заданной начальной фразы и одного из четырех продолжений. Задача модели BERT заключается в предсказании классификационного токена C. Для этого выбирается тренировочный пример, в котором значение токена C максимально, и пара логически связанных фраз становится ответом. Значение

токена C, в свою очередь, предсказывается на основе токена CLS, который модель BERT научилась предсказывать на этапе предварительного обучения.

В задаче SST-2 по заданному предложению из рецензий к фильмам нужно определить его эмоциональную окраску. В задаче CoLA нужно угадать, является ли данное предложение «лингвистически приемлемой» английской фразой.

Бывают задачи, решаемые на уровне отдельных токенов: в них, нужно предсказать не токен классификации предложения, а, например, токен начала и конца ответа на заданный вопрос (рисунок 6.2). Так работает задача SQUAD, в которой дан вопрос и абзац текста, где предположительно может быть дан ответ на поставленный вопрос. Задача нейросети - определить, на каком токене начинается и заканчивается конкретный ответ.

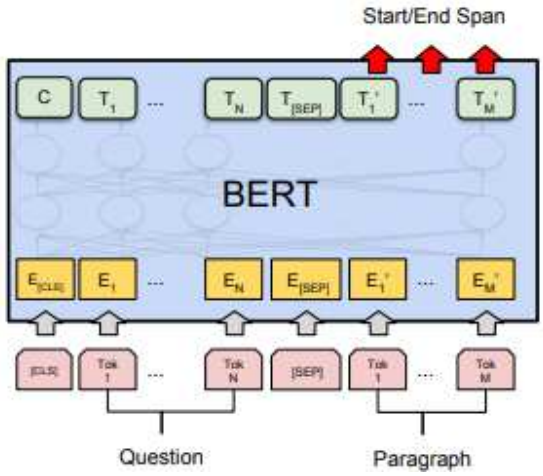


Рисунок 6.2 – Представление задачи нахождения начала и конца вопроса

Последний тип задач, решаемый моделью, является задача классификации токенов в отдельном предложении (рисунок 6.3). В задаче каждое слово нужно классифицировать как имя собственное или имя нарицательное.

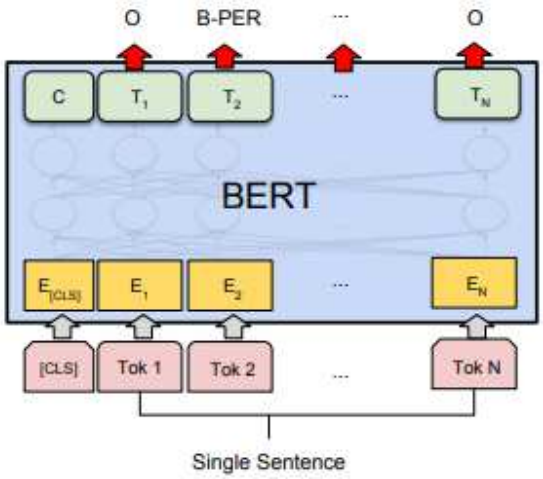


Рисунок 6.3 – Представление задачи классификации токенов

6.2 GPT-3

GPT-3 (Generative Pre-trained Transformer) — это авторегрессионная языковая модель третьего поколения, которая использует глубокое обучение для создания текста, похожего на человеческий. Или, проще говоря, это вычислительная система, предназначенная для генерации последовательностей слов, кода или других данных, начиная с исходного ввода, называемого подсказкой. Он используется, например, в машинном переводе для статистического прогнозирования последовательности слов. Языковая модель обучается на немаркированном наборе данных, состоящем из текстов, таких как Википедия и многие другие сайты, преимущественно на английском, но также и на других языках. Эти статистические модели необходимо обучать на больших объемах данных, чтобы получать соответствующие результаты.

Первая итерация GPT в 2018 году использовала 110 миллионов параметров обучения (т.е. значений, которые нейронная сеть пытается оптимизировать во время обучения). Год спустя GPT-2 использовала 1,5 миллиарда из них. Сегодня GPT-3 использует 175 миллиардов параметров. Он обучается на суперкомпьютере искусственного интеллекта Microsoft Azure (Scott 2020). Этот вычислительный подход подходит для широкого спектра вариантов использования, включая обобщение, перевод текста, исправление грамматики, ответы на вопросы, чат-бот, составление электронных писем и многое другое.

На рисунке 6.4 представлены размеры и архитектуры 8 различных по размерам моделей GPT-3.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Рисунок 6.4 – Параметры языковых моделей

n_{params} — это общее количество обучаемых параметров, n_{layers} — общее количество слоев, d_{model} — количество единиц в bottleneck слое (это слой, в котором меньше нейронов, чем в предыдущем, что заставляет нейросеть понижать размерность и делать признаки более компактными. В каждой модели есть feedforward слой, в котором сигнал распространяется строго от входного слоя к выходному, он в четыре раза больше bottleneck слоя ($d_{ff} = 4 * d_{model}$), а d_{head} — это размер каждой головы внимания. Все модели используют контекстное окно, n_{ctx} = 2048 токенов.

GPT решает задачу языкового моделирования. Языковое моделирование — это предсказание следующего слова (или куска слова) с учётом предыдущего контекста.

При генерации продолжения текста с помощью GPT происходит следующее:

1. Входной текст токенизируется в последовательность чисел (токенов).
2. Список токенов проходит через Embedding Layer (линейный слой) и конвертируется в список эмбеддингов.
3. К каждому эмбеддингу прибавляется positional embedding.
4. Далее список эмбеддингов проходит через несколько одинаковых блоков (Transformer Decoder Block).
5. После того как список эмбеддингов пройдёт через последний блок, эмбеддинг, соответствующий последнему токену матрично умножается на всё тот же входной, но уже транспонированный Embedding Layer и после применения SoftMax получается распределение вероятностей следующего токена.
6. Из этого распределения выбирается следующий токен (например, с помощью функции argmax).
7. Этот токен добавляется к входному тексту и повторяются шаги 1-6.

Модель GPT построена с помощью блоков декодера Трансформера. BERT же, напротив, использует блоки энкодера. Ключевое различие состоит в том, что GPT, как и все традиционные языковые модели, генерирует на выходе один токен за раз.

GPT-3 — это авторегрессионная модель, а BERT — двунаправленная. В то время как GPT-3 при прогнозировании учитывает только «левый» контекст, BERT учитывает как «левый», так и «правый» контекст. Это делает BERT более подходящим для таких задач, как анализ настроений или NLU (natural language understanding, понимание естественного языка), где важно понимать полный контекст предложения или фразы.

Еще одно различие между двумя моделями заключается в их обучающих наборах данных. В то время как обе модели обучались на больших наборах текстовых данных из таких источников, как Википедия и книги, GPT-3 обучался на 45 ТБ данных, а BERT — на 3 ТБ данных. Таким образом, GPT-3 имеет доступ к большему количеству информации, чем BERT, что может дать ему преимущество в определенных задачах, таких как обобщение или перевод, где доступ к большему количеству данных может быть полезен.

Наконец, есть различия и в размерах. GPT-3 значительно больше, чем его предшественник, из-за гораздо более обширного набора обучающих данных (в 470 раз больше, чем тот, который использовался для обучения BERT).

На рисунке 6.5 представлены результаты теста GLUE по состоянию на 2018 год, оцененные сервером оценки.

Число под каждым заданием обозначает количество обучающих примеров.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Рисунок 6.5 – Результаты теста GLUE

Здесь оба BERT BASE и BERT LARGE значительно превосходят все системы по всем задачам, получая среднее повышение точности на 4,5% и 7,0%. За самую крупную и широко освещаемую задачу теста GLUE, MNLI, BERT показал повышение точности на 4,6 %. В таблице лидеров GLUE, BERT LARGE набирает 80,5 баллов по сравнению с OpenAI GPT, который получает 72.8.

Сказать какая из языков моделей работает лучше не предоставляется возможным. GPT-3 и BERT зарекомендовали себя как ценные инструменты для выполнения различных задач NLP (Natural Language Processing, обработка естественного языка) с разной степенью точности. Однако из-за различий в архитектуре и размере набора обучающих данных каждая модель лучше подходит для одних задач, чем для других.

Например, GPT-3 лучше подходит для суммирования или перевода, а BERT — для анализа тональности или NLU (natural language understanding, понимание естественного языка). В конечном счете, выбор между двумя моделями будет зависеть от конкретных потребностей и от того, какую задачу необходимо выполнить.

7 УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

7.1. Основная литература

1. Бессмертный, Игорь Александрович. Интеллектуальные системы [Электр.ресурс] :1. учебник и практикум для вузов. - М. : Юрайт , 2020 on-line[Электронный ресурс]: — Режим доступа: https://urait.ru/viewer/intellektualnye-sistemy-451101#page/1.

2. Кудрявцев, Валерий Борисович. Интеллектуальные системы [Электр.ресурс] : учебник и практикум для вузов. - М. : Юрайт , 2020 on-line[Электронный ресурс]: — Режим доступа: https://urait.ru/viewer/intellektualnye-sistemy-452226#page/1.

3. Станкевич, Лев Александрович. Интеллектуальные системы и технологии [Электр.ресурс] : учебник и практикум для вузов. - М. : Юрайт , 2020 on-line[Электронный ресурс]: — Режим доступа: https://urait.ru/viewer/intellektualnye-sistemy-i-tehnologii-450773#page/1.

7.2. Дополнительная литература

1. Гасанов, Эльяр Эльдарович. Интеллектуальные системы. Теория хранения и поиска информации [Электр.ресурс] : учебник для вузов. - М. : Юрайт , 2020 on-line[Электронный ресурс]: — Режим доступа: https://urait.ru/viewer/intellektualnye-sistemy-teoriya-hraneniya-i-poiska-informacii-452220#page/1.

2. Иванов, Владимир Михайлович. Интеллектуальные системы [Электр.ресурс] : учебное2. пособие для вузов. - М. : Юрайт , 2020 on-line[Электронный ресурс]: — Режим доступа: https://urait.ru/viewer/intellektualnye-sistemy-453212#page/1.

7.3. Дополнительные полезные ресурсы

1. https://www.perplexity.ai/
2. https://you.com/search?q=who+are+you&tbm=youchat&cfr=chat
3. https://docs.cleanrl.dev/
4. https://spinningup.openai.com/en/latest/index.html

ПРИЛОЖЕНИЕ А. Оформление титульного листа

Министерство образования и науки
Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

КАФЕДРА АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

Лабораторная работа №1

по курсу «Интеллектуальные системы»

Название лабораторной работы

Выполнил:
Студент 1-го курса
Магистратуры гр. номер
_____Фамилия И.О.
подпись

_____
дата

Принял:
к.т.н., доцент
_____ _____Фамилия И.О.
Оценка подпись

_____
дата

Томск 2023