Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Разработка учебно-методических средств для исследования моделей глубокого обучения**

Студент гр. 43501/3 Волкова М.Д.

Санкт-Петербург – 2018

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Работа допущена к защите
зав. кафедрой

_____ В.М. Ицыксон

«____» _____ 2018 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Разработка учебно-методических средств для исследования моделей глубокого обучения**

Направление: 09.03.01 – Информатика и вычислительная техника

Выполнил студент гр. 43501/3 _____ Волкова М.Д.

Научный руководитель,
к. т. н., доц. _____ Никитин К.В.

Санкт-Петербург – 2018

РЕФЕРАТ

Бакалаврская работа, 73 стр., 25 рис., 1 табл., 13 ист., 2 прил.

УЧЕБНО-МЕТОДИЧЕСКИЕ СРЕДСТВА, ЛАБОРАТОРНЫЕ РАБОТЫ, ГЛУБОКОЕ ОБУЧЕНИЕ

В выпускнои работе проводится разработка учебно-методических средств для исследования моделей глубокого обучения. В работе рассмотрены задачи, модели глубокого обучения. Был проведен анализ средств для работы с глубоким обучением. Разработка проводилась на языке Python в Jypiter Notebook.

СОДЕРЖАНИЕ

Глоссарий

RNN (РНС, recurrent neural network) - рекурентная сеть

CNN (СНС, convolutional neural network) - сверточная нейронная сеть

DBN (deep belief network) - глубокая сеть доверия

RBM (restricted Boltzmann machine) - ограниченная машина Больцмана

CC (ПП, Concurrent computing) - параллельные вычисления

ВВЕДЕНИЕ

В настоящее время глубокое обучение является активно развивающейся областью научных исследований. Своими успехами глубокое обучение обязано постоянно увеличивающемуся объему обучающих данных и относительно дешевым графическим процессорам, позволяющим построить эффективную процедуру вычислении.

Глубокое обучение рассматривает методы моделирования высокоуровневых абстракций с помощью множества последовательных нелинейных трансформаций, которые, как правило, представляются в виде искусственных нейронных сетей. На сегодняшний день нейросети успешно используются для решения таких задач, как прогнозирование, распознавание образов и ряда других.

Глубокое обучение является подмножеством методов машинного обучения, в которых применяются искусственные неиронные сети, построенные на базе аналогии со структурои неиронов человеческого мозга. Термин «глубокии» подразумевает наличие большого числа слоев в нейронной сети.

В компаниях Google, Microsoft, Amazon, Apple, Facebook и многих других методы глубокого обучения постоянно используются для анализа больших массивов данных. Теперь эти знания и навыки вышли за рамки чисто академических исследовании и стали достоянием крупных промышленных компании.

Актуальность темы глубокого обучения подтверждается регулярным появлением статей на данную тему.

Целью дипломной работы является разработка учебно-методические средства по исследованию моделей глубокого обучения. В соответствии с целью исследования были поставлены следующие задачи:

- проанализировать существующие лабораторные работы;
- сравнить и выбрать технологии моделирования глубоких нейронных сетей;
- разработать и протестировать несколько лабораторных работ.

1. Обзор существующих технологий глубокого обучения

В данной главе представлен краткий обзор инструментов проектирования и обучения нейросетевых моделей: различных фреймворков, библиотек, программ глубокого обучения. Рассмотрены существующие лабораторные работы, а также модели глубоких нейронных сетей.

1.1. Сравнение программ глубокого обучения

Существующие инструменты глубокого обучения имеют различную функциональность и требуют от пользователя разного уровня знаний и навыков. Правильный выбор инструмента это важная задача, позволяющая добиться необходимого результата за наименьшее время и с меньшей затратой сил. Таблица 1.1 посвящена сравнению некоторых программных инструментов глубокого обучения.

Таблица 1.1. Сравнение программ глубокого обучения

Название	**Язык**	**Интерфейс**	**РНС**	**СНС**
Cafee	C++	Python MATLAB	Да	Да
DeepLearninf4j	Java	Java Scala Clojure Python	Да	Да
Dlib	C++	C++	Нет	Да
Keras	Python	Python	Да	Да
MCT	C++	Python C++ командная строка	Да	Да
MXNet	C++	C++ Python Julia Matlab JavaScript Go , R Scala Perl	Да	Да
ND	C++	Графический интерфейс пользователя	Нет	Нет
OpenNN	C++	C++	Нет	Нет
TF	Python C++	Python C++ Java , Go	Да	Да
Theano	Python	Python	Да	Да
Torch	C Lua	Lua LuaJIT C	Да	Да

Caffe

Caffe - фреймворк глубокого обучения. Янцин Цзя создал проект в процессе подготовки своей диссертации в университете Беркли. Caffe выпускается под лицензией BSD. Написан на языке C++ и поддерживает интерфейс на языке Python [1] .

В библиотеке Caffe топология нейросетей, исходные данные и способ обучения задаются с помощью конфигурационных файлов в формате proto.txt. Файл содержит описание входных данных (тренировочных и тестовых) и слоев нейронной сети.

Разработчики Caffe поддерживают возможности создания, обучения и тестирования свёрточных нейронных сетей, долгой краткосрочной памяти и полносвязных нейронных сетей.

Эта библиотека позволяет использовать готовые промышленные конфигурации нейронных сетей, прошедшие апробацию. В комплект входит, в частности AlexNet, победившую в 2012 году в соревновании по распознаванию изображений ImageNet, и GoogLeNet, победившую в соревнованиях ImageNet 2014 года.

Большое преимущество фреймворка Caffe это скорость, это делает его идеальным для экспериментов и использования в промышленности.

Deeplearning4j

Deeplearning4j - библиотека на языке Java. Используется как фреймворк для глубокого обучения, при этом совместима с Clojure и включает интерфейс прикладного программирования для языка Scala. Кроме того, имеются средства для работы с библиотекой на языке Python через фреймворк Keras [2] .

Deeplearning4j включает реализацию:

- ограниченной машины Больцмана;
- глубокой сети доверия;
- глубокого автокодировщика;
- стекового автокодировщика с фильтрацией шума;
- рекурсивной тензорной нейронной сети;
- word2vec;

- doc2vec;
- GloVe.

Является открытым программным обеспечением, распространяется под лицензией Apache 2.0.

Фреймворк позволяет комбинировать компоненты, объединяя обычные нейронные сети с машинами Больцмана, свёрточными нейронными сетями, автокодировщиками и рекуррентными сетями в одну систему. Кроме того, поддерживаются расширенные средства визуализации. Обучение проводится как с помощью обычных многослойных нейронных сетей, так и для сложных сетей, в которых определён граф вычислений.

Dlib

Dlib - это универсальная кроссплатформенная библиотека, написанная на языке программирования C++. Выпускается под лицензией Software Boost. Она используется в широком диапазоне областей, включая робототехнику, встроенные устройства, смартфоны и большие высокопроизводительные вычислительные среды [3] .

Dlib начал разрабатываться в 2002 году и на сегодняшний день он содержит компоненты для работы с:

- сетями;
- потоками;
- графическими пользовательскими интерфейсами;
- структурами данных;
- линейной алгеброй;
- машинным обучением;
- обработкой изображений;
- интеллектуальным анализом данных;
- анализом XML и текста;
- численной оптимизацией;

- байесовскими сетями ;
- и многими другими задачами.

В последние годы большая часть развития была сосредоточена на создании широкого набора инструментов статистического машинного обучения.

Keras

Keras - это нейросетевая библиотека, написанная на Python [4] . Она была разработана с упором на возможность быстрого экспериментирования. Ее основным автором является Франсуа Шолле, инженер Google. Keras не является самостоятельнои системои, а работает поверх Theano, TensorFlow или CNTK. В 2016 году Keras включили в состав TensorFlow.

Руководящие принципы создателей библиотеки:

- удобство для пользователя;
- модульность;
- масштабируемость;
- работа с Python.

The Microsoft Cognitive Toolkit

Microsoft Cognitive Toolkit (CNTK) - фреймворк для глубокого обучения. Ядро CNTK реализовано на C++. CNTK описывает нейронные сети как серию вычислительных шагов через ориентированный граф [5] .

Эта библиотека делает упор на глубокое обучение нейронных сетей с рекуррентной архитектурой и именно с ними CNTK сильно выигрывает в производительности.

Сильной стороной Cognitive Toolkit является точность и масштабируемость: возможность работы как и с использованием CPU, так и с мощным графическими процессорами производства NVIDIA.

MXNet

MXNet - это библиотека глубокого обучения с открытым исходным кодом, распространяется под лицензией Apache 2.0 [6]. Поддерживает гибкую модель программирования и несколько языков, давая возможность использовать как императивные, так и символические программные конструкции.

MXNet создана на основе планировщика с динамической зависимостью, который анализирует зависимости данных в последовательном коде и автоматически сразу распараллеливает как декларативные, так и императивные операции.

Библиотека берет свое начало в научной среде и является продуктом совместной и индивидуальной работы исследователей из нескольких ведущих университетов, такими как Университет Карнеги, Массачусетский технологический институт, Вашингтонский университет и Гонконгский университет науки и техники.

Neural Designer

Neural Designer - это программный инструмент для анализа данных на основе нейронных сетей [7] .

Основной областью исследований является область искусственного интеллекта. Он был разработан с открытым исходным кодом OpenNN, и содержит графический интерфейс пользователя.

В 2015 году в рамках программы Horizon 2020 Европейская комиссия выбрала Neural Designer в качестве "подрывной инновации"в области информационных и коммуникационных технологий.

OpenNN

OpenNN - это библиотека, написанная на C++, разработана компанией Artelnics, специализирующейся на искусственном интеллекте. Библиотека с открытым исходным кодом, лицензированная по GNU Lesser General Public License [8].

Программное обеспечение реализует любое количество уровней нелинейных блоков обработки для контролируемого обучения. Эта глубокая архитектура позволяет создавать нейронные сети с универсальными свойствами аппроксимации. Кроме того, она обеспечивает многопроцессорное программирование с помощью OpenMP.

Основным преимуществом OpenNN является его высокая производительность. Эта библиотека выделяется с точки зрения скорости выполнения и распределения памяти.

Обычно OpenNN используют в областях бизнес-аналитики здравоохранения и техники (оптимизация производительности и др). OpenNN не занимается компьютерным зрением или обработкой естественного языка.

TensorFlow

TensorFlow - открытая библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронных сетей. Применяется как для исследований, так и для разработки собственных продуктов Google [9].

Является продолжением закрытого проекта DistBelief. Изначально TensorFlow была разработана командой Google Brain для внутреннего использования в Google, в 2015 году система была переведена в свободный доступ с открытой лицензией Apache 2.0.

Вычисления TensorFlow выражаются в виде потоков данных через граф состояний. Название TensorFlow происходит от операций с многомерными массивами данных, которые также называются "тензорами".

В 2016 году компания Google сообщила о применении аппаратного ускорителя собственной разработки - тензорного процессора - специализированной интегральной схемы, адаптированной под задачи для TensorFlow, и обеспечивающей высокую производительность в арифметике пониженной точности и направленной скорее на применение моделей, чем на их обучение. После использования тензорного процессора в собственных задачах Google по обработке данных удалось добиться на порядок лучших показателей продуктивности.

Theano

Theano - библиотека численного вычисления в Python. Вычисления в Theano выражаются синтаксисом NumPy и компилируются, как на обычных центральных процессорах, так и на графических процессорах [10] . Theano является открытым проектом, основным разработчиком которого является группа машинного обучения в Монреальском университете.

28 сентября 2017 года было объявлено о прекращении работы над проектом после выхода релиза 1.0.

Основные возможности Theano:

- работа с тензорами и поддержка множества тензорных операций;
- работа с матрицами и поддержка ряда операций с ними;
- численные методы линейной алгебры;
- возможность создавать новые операции с графами;
- численные операции по преобразованию графов;
- поддержка языка python версий 2 и 3;
- использование архитектуры CUDA для работы с тензорами;
- поддержка стандарта Basic Linear Algebra Subprograms для процедур линейной алгебры.

Torch

Torch - библиотека для научных вычислений с широкой поддержкой алгоритмов машинного обучения. Библиотека написана на языке Lua с использованием C, поддерживается распараллеливание вычислений средствами CUDA и OpenMP [11] .

Torch позволяет создавать сложные нейросети с помощью механизма контейнеров.

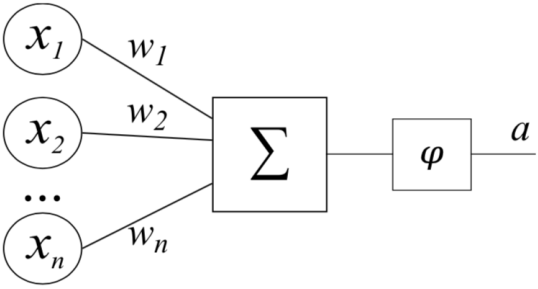
Библиотека состоит из набора модулей, каждый из которых отвечает за различные стадии работы с нейросетями, например, есть модули отвечающие за конфигурирование нейросети (определению слоев, и их параметров), оптимизацию и визуализацию данных. Для расширения функциональности библиотеки можно устанавливать дополнительные модули.

1.2. Модели нейронных сетей

Модель искусственного нейрона была предложена Уорреном Мак-Каллоком и Уолтером Питтсом в 1943 году [12] . В качестве основы

для своей модели авторы использовали биологический нейрон. Искусственный нейрон МакКаллока - Питтса имеет N входных бинарных величин $x1, ..., xn$, которые трактуются как импульсы, поступающие на вход нейрону 1.2. В нейроне импульсы складываются с весами $w1, ..., wn$.

Рисунок 1.1. Модель искусственного нейрона МакКаллока-Питтса



МакКаллок и Питтс предложили также метод объединения отдельных нейронов в искусственные нейронные сети. Для этого выходные сигналы нейрона передаются на вход следующему нейрону **??**. Нейронная сеть состоит из нескольких слоев, на каждом из которых может находиться несколько нейронов. Слои делятся на входные выходные и скрытные слои.

Однозначного определения, что такое глубокая нейронная сеть, не существует, но принято считать, что это сеть, которая содержит больше одного скрытого слоя.

Далее будут рассмотрены некоторые модели нейронны сетей.

Рекуррентные нейронные сети

Рекуррентные нейронные сети (Recurrent neural networks, RNN) были разработаны в 80 годах прошлого века. Прхитектура таких сетей строится из узлов, каждый из которых соединён со всеми другими узлами. У каждого нейрона порог активации меняется со временем и является вещественным числом. Каждое соединение имеет переменный вещественный вес. Узлы разделяются на входные, выходные и скрытые.

Рисунок 1.2. Искусственная нейронная сеть

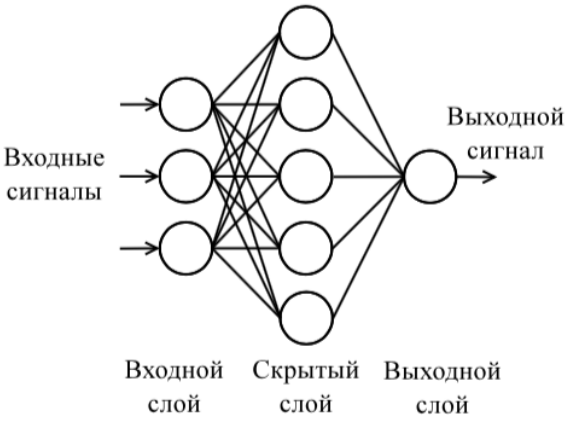
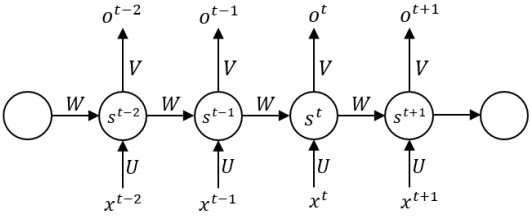
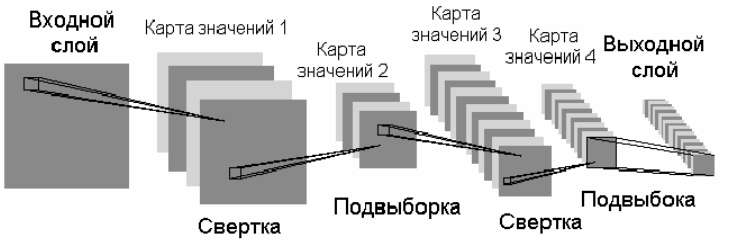


Рисунок 1.3. Рекуррентная нейронная сеть



В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на сегменты, например: распознавание рукописного текста или распознавание речи. Было предложено много различных архитектурных решений для рекуррентных сетей.

Рисунок 1.4. Сверточная нейронная сеть



Свёрточные нейронные сети

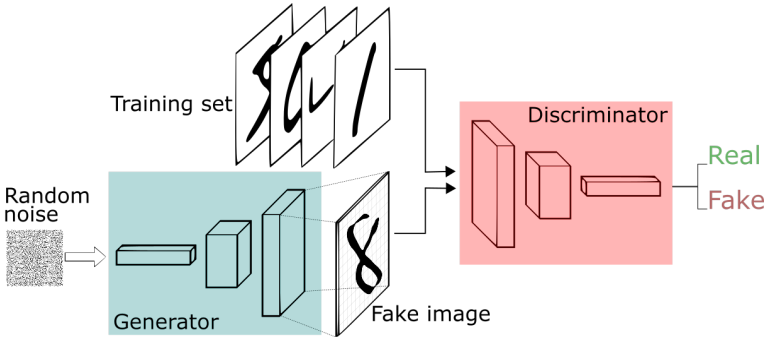
Свёрточные нейронные сети (convolutional neural networks, CNN) отличаются от других сетей. Они используются в основном для обработки изображений. СНС чередует слои свертки и субдискретизации, а на выходе имеет несколько полносвязных слоев. Типичным способом применения CNN является классификация изображений. Такие сети обычно используют «сканер», не обрабатывающий все данные за один раз. Входные данные передаются через свёрточные слои, в которых не все узлы соединены между собой. Вместо этого каждый узел соединен только со своими ближайшими соседями. Эти слои имеют свойство сжиматься с глубиной, причём обычно они уменьшаются на какой-нибудь из делителей количества входных данных, часто используются степени двойки.

Слой свертки и слой субдискретизации чередуясь между собой, формируют входной вектор признаков для многослойного персептрона.

Генеративные состязательные сети

Генеративные состязательные сети (Generative adversarial networks, GAN) принадлежат другому семейству нейросетей. По сути, это две сети, работающие вместе. GAN состоит из любых двух сетей, где одна из сетей генерирует данные (“генератор”), а вторая - анализирует (“дискриминатор”). Дискриминатор получает на вход или обучающие данные, или сгенерированные первой сетью. То, насколько точно дискриминатор сможет определить источник данных, служит потом для оценки ошибок генератора. Таким образом,

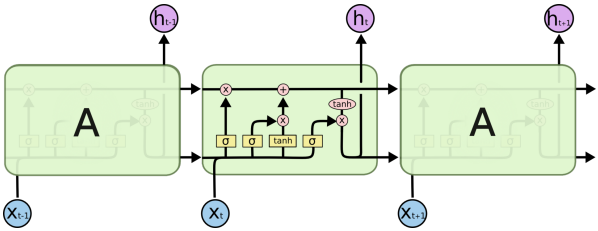
Рисунок 1.5. Генеративно состязательная сеть



происходит своего рода соревнование, где дискриминатор учится лучше отличать реальные данные от сгенерированных, а генератор стремится стать менее предсказуемым для дискриминатора. Это работает отчасти потому, что даже сложные изображения с большим количеством шума в конце концов становятся предсказуемыми, но сгенерированные данные, мало отличающиеся от реальных, сложнее научиться отличать. GAN достаточно сложно обучить, так как задача здесь - не просто обучить две сети, но и соблюдать необходимый баланс между ними. Если одна из частей (генератор или дискриминатор) станет намного лучше другой, то GAN никогда не будет сходиться.

Долгая краткосрочная память

Рисунок 1.6. Долгая краткосрочная память

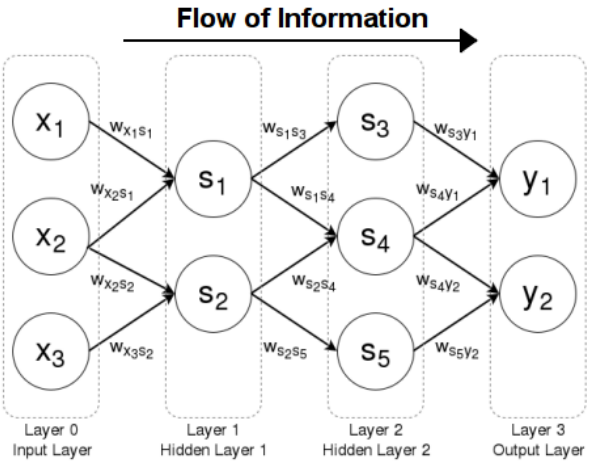


Долгая краткосрочная память (Long short term memory, LSTM)

- попытка побороть проблему взрывного градиента, используя фильтры и блоки памяти. Эта идея пришла, скорее, из области схемотехники, а не биологии. У каждого нейрона есть три фильтра: входной фильтр, выходной фильтр и фильтр забывания. Задача этих фильтров - сохранять информацию, останавливая и возобновляя ее поток. Входной фильтр определяет количество информации с предыдущего шага, которое будет храниться в блоке памяти. Выходной фильтр занят тем, что определяет, сколько информации о текущем состоянии узла получит следующий слой. Наличие фильтра забывания на первый взгляд кажется странным, но иногда забывать оказывается полезно: если нейросеть запоминает книгу, в начале новой главы может быть необходимо забыть некоторых героев из предыдущей. Показано, что LSTM могут обучаться действительно сложным последовательностям, например, подражать Шекспиру или сочинять простую музыку. Стоит отметить, что так как каждый фильтр хранит свой вес относительно предыдущего нейрона, такие сети достаточно ресурсоемкости. Также существуют, двунаправленные LSTM. Разница лишь в том, что эти нейросети связаны не только с прошлым, но и с будущим.

Сети прямого распространения

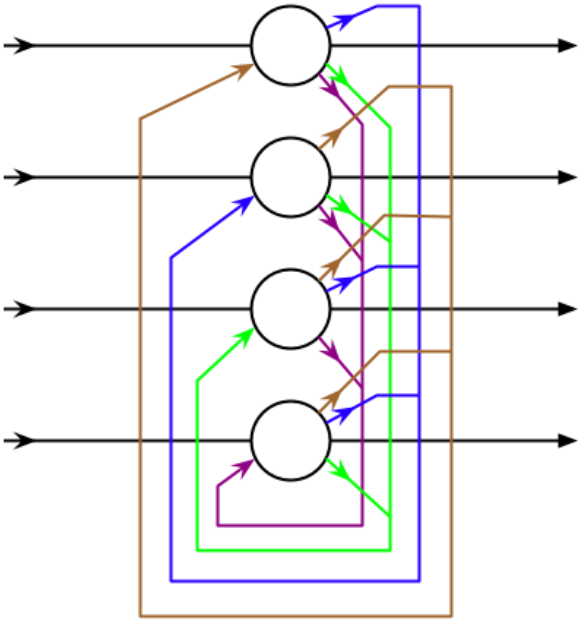
Рисунок 1.7. Сети прямого распространения



Сети прямого распространения (Feed forward neural networks, FF or FFNN) первая и самая простая нейронная сеть. Она передает информацию от входа к выходу без циклов. Нейроны одного слоя между собой не связаны, при этом каждый нейрон этого слоя связан с каждым нейроном соседнего слоя. FFNN обычно обучают методом обратного распространения ошибки, подавая модели на вход пары входных и ожидаемых выходных данных. На практике использование сетей прямого распространения ограничено, и чаще они используются совместно с другими сетями. FFNN с радиально-базисной функцией в качестве функции активации называются сети радиально-базисных функций (radial basis function, RBF).

Нейронная сеть Хопфилда

Рисунок 1.8. Нейронная сеть Хопфилда



Нейронная сеть Хопфилда - полносвязная сеть, это означает, что каждый нейрон соединен с каждым. Сеть Хопфилда является сетью

с обратной связью, что означает, что ее выходы перенаправляются на входные данные. Количество узлов входов равно количеству выходов сети. Кроме того, каждый из нейронов имеет двоичное состояние или значение активации, обычно представленное как 1 или -1. Состояние каждого узла обычно сходится, это означает, что состояние каждого узла становится фиксированным после определенного количества обновлений.

Ограниченная машина Больцмана

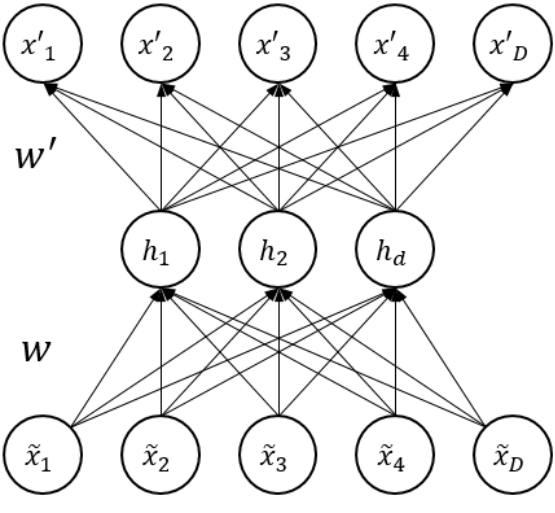
Рисунок 1.9. Ограниченная машина Больцмана

Ограниченная машина Больцмана (Restricted Boltzmann machine, RBM) - вид генеративной стохастической нейронной сети, которая определяет распределение вероятности на входных образцах данных. В них каждый нейрон не связан с каждым, а только каждая группа нейронов соединена с другими группами. Входные нейроны не связаны между собой, нет соединений и между скрытыми нейронами. RBM можно обучать так же, как и сети прямого распространения, за небольшим отличием: вместо передачи данных вперед и последующего обратного распространения ошибки, данные передаются вперед и назад, а затем применяется прямое и обратное распространение.

Автоэнкодеры

Автоэнкодеры (Autoencoders, AE) - подобны сетям прямого распространения. Основная идея автоэнкодеров - автоматическое кодирование информации, отсюда и название. Сеть напоминает по форме песочные часы, так как скрытый слой меньше, чем входной и выходной; к тому же она симметрична относительно средних слоев (одного или двух, в зависимости от четности/нечетности общего количества слоев). Самый маленьких слой почти всегда средний, в нем информация максимально сжата. Все, что расположено до середины — кодирующая часть, выше середины — декодирующая, а в середине - код. AE обучают методом обратного распространения ошибки, подавая входные данные и задавая ошибку равной разницу между входом и выходом. AE можно построить симметричными и с точки зрения весов, выставляя кодирующие веса равными декодирующим.

Рисунок 1.10. Автоэнкодеры



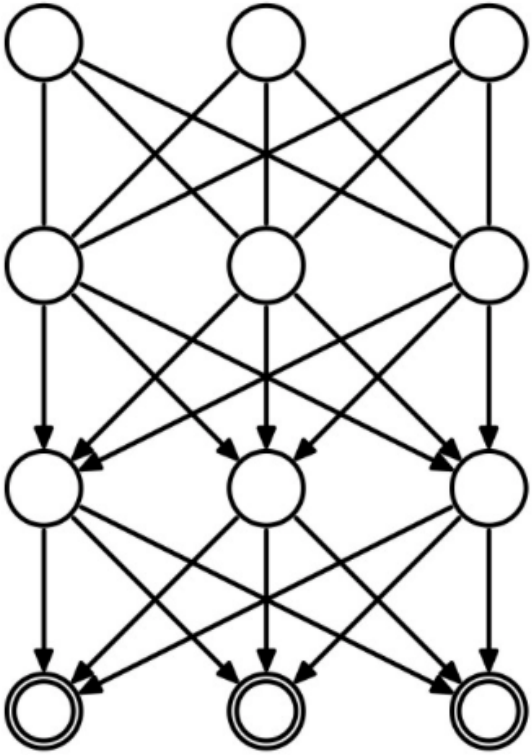
Глубокие сети доверия

Глубокие сети доверия (Deep belief networks, DBN) - сети, представляющие собой композицию нескольких ограниченных машин Больцмана. Такие сети показали себя эффективно обучаемыми одна за другой, когда каждая сеть должна научиться кодировать предыдущую. Этот метод также называют “жадное обучение”, он заключается в принятии оптимального на данный момент решение, чтобы получить подходящий, но, возможно, не оптимальный результат. Глубокие сети доверия могут обучаться методами contrastive divergence или обратным распространением ошибки и учатся представлять данные в виде вероятностной модели, в точности как ограниченные машины Больцмана. Один раз обученную и приведенную к стационарному состоянию модель можно использовать для генерации новых данных.

Нейронные эхо-сети

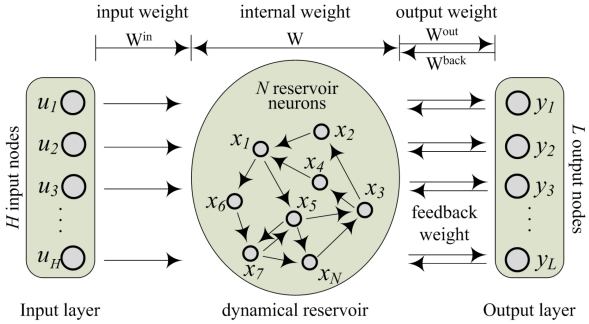
Нейронные эхо-сети (Echo state networks, ESN) — еще один вид рекуррентных нейросетей. Они выделяются тем, что связи между нейронами в них случайны, не организованы в аккуратные слои, и обу-

Рисунок 1.11. Глубокая сеть доверия



чаются они по-другому. Вместо подачи на вход данных и обратного распространения ошибки, мы передаем данные, обновляем состояния нейронов и в течение некоторого времени следим за выходными данными. Входной и выходной слои играют нестандартную роль, так как входной слой служит для инициализации системы, а выходной слой - в качестве наблюдателя за порядком активации нейронов, который проявляется со временем. Во время обучения изменяются связи только между наблюдателем и скрытыми слоями.

Рисунок 1.12. Нейронные эхо-сети



1.3. Существующие лабораторные работы

Институт глубокого обучения NVIDIA (DLI) предлагает пройти практические занятия для разработчиков, специалистов по обработке данных, которые решают сложные задачи с применением технологии глубокого обучения. NVIDIA предлагает узнать о современных техниках проектирования, тренировки и интеграции алгоритмов машинного обучения в самые разные приложения. Курс предлагает научиться работать с популярными фреймворками с открытым исходным кодом и новейшими платформами глубокого обучения с ускорением на NVIDIA GPU.

NVIDIA предлагает следующие области [13]:

- глубокое обучение в области компьютерного зрения;
- глубокое обучения для работы с разными типами данных;
- глубокое обучение для обработки естественных языков;
- глубокое обучение в области геномики;
- глубокое обучение для анализа медицинских изображений;
- восприятие окружающего мира у беспилотных автомобилей;
- создание цифрового контента с помощью генеративно-состязательных сетей и автоэнкодеров;

- основы ускоренных вычислений с CUDA C/C++;
- основы ускоренных вычислений с OpenACC;
- глубокое обучения для разработки стратегии торгового финансирования;
- глубокое обучение для анализа полномасштабного видео.

Эти лабораторные работы не подходят для изучения в рамках университетской программы, так как только первые три из них бесплатные, на прохождение каждой из них дается полтора часа, и основываются на веб-приложение DIGITS. DIGITS (the Deep Learning GPU Training System) - это веб-приложение для обучения глубоким учебным моделям. В настоящее время поддерживаемые структуры: Caffe, Torch и Tensorflow.

2. Анализ задач и моделей глубокого обучения

В этой главе будут подробно рассмотрены и проанализированы используемые в работе задачи и модели глубокого обучения.

2.1. Анализ задач

Задачи были подобраны так, что бы они относились к различнымм областям анализа данных. Далее будут описаны прикладные задачи глубокого обучения, такие как: классификация изображений, локализация объектов и прогнозирование временных рядов.

2.1.1. Классификация изображений

Задача классификации предназначена для определения того, к какой группе наиболее вероятно может быть отнесен каждый объект.

Две основные постановки это задачи - это бинарная (относится ли изображение к данной категории) и многоклассовая (к какой категории относится изображение).

Постановка задачи

Дано множество объектов, каждый их которых принадлежит одному из нескольких классов. Надо определить, какому классу принадлежит данный экземпляр. Каждый объект с номером j можно описать вектором признаков x_j . Каждому объекту можно приписать метку класса y_j .

Бинарная классификация

Дана обучающая выборка

$$X_m = \{(x_1, y_1), ..., (x_m, y_m)\}(x_i, y_i) \in R^m \times Y, Y = \{-1, +1\}$$

Объекты независимы и взяты из некоторого неизвестного распределения

$$(x_i, y_i) \in P(x, y)$$

Цель: для всех новых значений x оценить значения

$$argmaxP(y|x)$$

Многоклассовая классификация

Дана обучающая выборка

$$X_m = \{(x_1, y_1), ..., (x_m, y_m)\}(x_i, y_i) \in R^m \times Y, Y = \{1, ..., K\}$$

Объекты независимы и взяты из некоторого неизвестного распределения

$$(x_i, y_i) \in P(x, y)$$

Цель: для всех новых значений X оценить значения

$$argmaxP(y|x)$$

2.1.2. Локализация объекта

При разработке систем технического зрения часто возникает задача обнаружения и локализации объекта на изображении. Она заключается в проверке наличия на входном изображении некоторого объекта и вычислении его геометрических характеристик типа положения и угловой ориентации.

Постановка задачи

Имеем набор входных изображений $x_1, ., , , x_n \subset \chi$ и связанных с ними аннотаций $y_1, ..., y_n \subset \gamma$, мы хотим получить отображение $g : \chi \mapsto \gamma$, с помощью которого мы можем автоматически маркировать новые образы. Рассмотрим случай, когда выходное пространство состоит из метки ω, указывающей, присутствует ли объект на изображении, и вектора, указывающего верхнюю t, левуюl, нижнюю b и правую r границы рамки внутри изображения:

$$\gamma \equiv \{(\omega, t, l, b, r)|\{+1, -1\}, (t, l, b, r) \in R^4\}.$$

При $\omega = -1$ координатный вектор (t, l, b, r) игнорируется. Мы изучаем это сопоставление, как

$$g(x) = argmax_y f(x, y)$$

, где $f(x, y)$ - дискриминантная функция, которая должна давать большое значение парам (x, y), которые хорошо сопоставляются. Поэтому задача состоит в том, чтобы обучить функцию f, если максимизация выполняется.

2.1.3. Прогнозирование временных рядов

Прогнозирование временных рядов подразумевает, что известно значение некой функции в первых n точках временного ряда. Используя эту информацию необходимо спрогнозировать значение в $n + m$ точке временного ряда.

Постановка задачи

Пусть значения временного ряда доступны в дискретные моменты времени $t = 1, 2, ..., T$. Обозначим временной ряд $Z(t) = Z(1), Z(2), ..., Z(T)$. В момент времени T необходимо определить значения процесса $Z(t)$ в моменты времени $T+1, ..., T+P$. Момент времени T называется моментом прогноза, а величина P - временем упреждения.

Для того чтобы вычислить значений временного ряда в будущие моменты времени требуется определить функциональную зависимость, отражающую связь между прошлыми и будущими значениями этого ряда

$$Z(t) = F(Z(t-1), Z(t-2), Z(t-3), ...) + \varepsilon_t$$

Эта зависимость называется моделью прогнозирования. Требуется создать такую модель прогнозирования, для которой среднее абсолютное отклонение истинного значения от прогнозируемого стремится к минимальному для заданного P

$$E = 1/p \sum_{t=T+1}^{T+P} |\varepsilon_t| \to min$$

2.2. Анализ моделей

Из рассмотренных моделей в предыдущей главе был сделан выбор в пользу сверточной нейронной сети для задачи классификации и lstm-сети для прогнозирования временных рядов.

2.2.1. Сверточная нейронная сеть

В компьютерном обучении сверточная нейронная сеть представляет собой класс глубоких нейронных сетей с прямой связью, наиболее часто используемых для анализа визуальных образов, поскольку СНС вырабатывает необходимую иерархию абстрактных функций, переходя от определенных особенностей изображения к более абстрактным деталям, и далее к ещё более абстрактным деталям вплоть до выделения понятий высокого уровня. Она выделяет важные детали и опускает менее значимые.

Эти признаки, малопонятны и трудны для интерпретации настолько, что в практических системах не рекомендуется пытаться понять содержания этих признаков или пытаться их совершенствовать, вместо этого рекомендуется улучшить структуру и архитектуру сети, чтобы получить лучшие результаты.

Архитектура и принцип работы

Свое название сверточная нейронная сеть получила из-за операции свертки, хотя это название и условно. Математически это кросскорреляция, а не свертка.

В сверточной нейронной сети ограниченная матрица весов используется в операциях свертки, они перемещаются по всему обрабатываемому слою, образуя после каждого шага сигнал активации для нейрона следующего слоя с тем же положением. Для нейронов выходного слоя используется одна и та же матрица весов, она называется ядром свертки. Он интерполируется как графическое кодирование признака. После этого следующий уровень показывает наличие признака в обработанном слое, формируя так называемую карту объектов. Ядра свертки заранее не заложены исследователем, но формируются независимо путем обучения сети методом обратного распространения ошибок. Прохождение каждого набора весов формирует собственный экземпляр карт признаков, что делает нейронную сеть многоканальной.

Шаг в поиске выбирается так, чтобы не перескочить искомый признак. Операция субдискретизации уменьшает размерность карт, что ускоряет вычисление, делает сеть инвариантной к масштабу входного изображения. Сигнал проходит через ряд слоев свертки и субдискретизации. Переменные слои позволяют создавать «карты признаков». На практике это означает способность распознавать сложные иерархии признаков. На выходе слоев свертки сети дополнительно устанавливают несколько слоев полностью подключенной нейронной сети, а карты с предыдущего слоя подают на вход.

Слой свертки

Слой свёртки (convolutional layer) - включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты матричного произведения для каждого фрагмента). Весовые коэффициенты ядра свёртки неизвестны и устанавливаются в процессе обучения. Особенностью свёрточного слоя является сравнительно небольшое количество параметров, устанавливаемое при обучении.

Слой активации

Скалярный результат каждой свёртки попадает на функцию активации, которая представляет собой некую нелинейную функцию. Слой активации обычно логически объединяют со слоем свёртки. Функция нелинейности может быть любой по выбору исследователя, традиционно для этого использовали функции типа гиперболического тангенса или сигмоиды. Однако, по состоянию на 2017 год функция ReLU являются наиболее часто используемыми функциями активации в глубоких нейросетях, в частности, в свёрточных. Она позволила существенно ускорить процесс обучения и одновременно упростить вычисления (за счёт простоты самой функции), что означает блок линейной ректификации, вычисляющий функцию.

Слой субдискретизации

Слой субдискретизации представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей уплотняется до одного пикселя, проходя нелинейное преобразование. Наиболее употребительна при этом функция максимума. Преобразования затрагивают

непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция субдискретизации позволяет существенно уменьшить пространственный объём изображения. Субдискретизация интерпретируется так. Если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. Слой субдискретизации, как правило, вставляется после слоя свёртки.

Полносвязный слой

После нескольких прохождений свёртки изображения и уплотнения с помощью субдискретизации система перестраивается от конкретной сетки пикселей с высоким разрешением к более абстрактным картам при знаков, как правило на каждом следующем слое увеличивается число каналов и уменьшается размерность изображения в каждом канале. В конце концов остаётся большой набор каналов, хранящих небольшое число данных, которые интерпретируются как самые абстрактные понятия, выявленные из исходного изображения. Эти данные объединяются и передаются на обычную полносвязную нейронную сеть, которая тоже может состоять из нескольких слоёв. При этом полносвязные слои уже утрачивают пространственную структуру пикселей и обладают сравнительно небольшой размерностью.

2.2.2. Долгая краткосрочная память

LSTM-сеть — это искусственная нейронная сеть, содержащая LSTM-модули вместо или в дополнение к другим сетевым модулям.

Архитектура

LSTM-сеть является рекуррентным сетевым модулем, который может хранить значения как для коротких, так и для длительных периодов времени. Ключом к этой функции является то, что LSTM-модуль не использует функцию активации внутри своих рекуррентных компонентов. Таким образом, хранимое значение не изменяется во времени, и градиент или штраф не исчезают при использовании обратного распространения ошибки во времени при тренировки сети.

Модули LSTM часто группируются в «блоки», содержащие разные модули LSTM. Такое устройство типично для «глубоких» многослойных нейронных сетей и обеспечивает параллельные вычисления. LSTM-блоки содержат три или четыре ««вентиля»», которые используются для контроля потоков информации на входах и на выходах. Этивентили реализованы как логистическая функция для расчета значений в диапазоне [0; 1]. Умножение на это значение используется для частичного разрешения или блокировки потока информации внутри и вне памяти. «Выходной вентиль» контролирует степень, в которой значение, хранящееся в памяти, используется для вычисления функции активации выхода для блока.

3. Реализация комплекса

Курс разработан так, чтобы выполнять его можно было с любой операционной системы. Для работы с учебниками, надо установить Python 3 и Jypiter. Количество сторонних библиотек сведено к минимуму.

3.1. IPython и Jupyter Notebook

IPython является мощным инструментом для работы с языком Python.

Jupyter Notebook представляет собой графическую веб-оболочку для интерактивных вычислений, кроме того это удобный инструмент для создания красивых аналитических отчетов, так как он позволяет хранить вместе код, изображения, комментарии, формулы и графики:

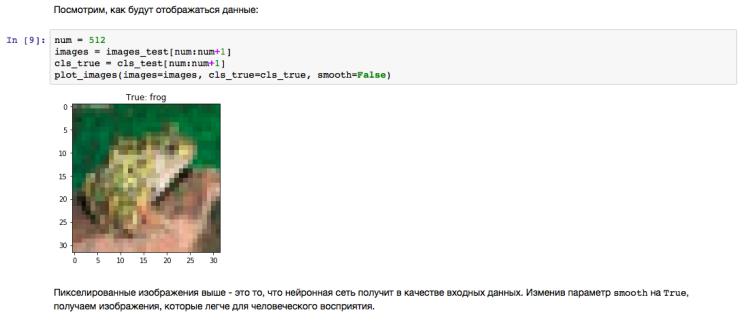


Рисунок 3.1. Jupyter Notebook

Основные особенности этой платформы – это комплексная интроспекция объектов, сохранение истории ввода на протяжении всех сеансов, кэширование выходных результатов, магические команды, журналирование сессии, дополнительный командный синтаксис, доступ к системной оболочке. Ко всему прочему это еще и свободное программное обеспечение.

Экспортировать блокнот можно в формате IPython Notebook (.ipynb), но есть и другие варианты:

- преобразовать блокнот в html-файл;

- опубликовать его в gists, где можно обрабатывать файлы этого формата;

- сохранить блокнот на облако, а затем открыть ссылку в nbviewer

Ко всему прочему, в блокноте есть magics-команды. Под "магией"в IPython понимаются дополнительные команды, выполняемые в рамках оболочки, которые облегчают процесс разработки и расширяют ваши возможности.

Готовые учебники - это файлы, в которых хранится исходный код, входные и выходные данные, полученные в рамках сессии. Фактически, он является записью работы, но при этом позволяет заново выполнить код.

3.2. Описание готовых работ

В даннои главе будут представлены примеры разработанных лабораторных работ по дисциплине "исследование моделей глубокого обучения". Программы были разработаны в Jupyter Notebook на языке Python, с использованием библиотек Tensorflow и Keras.

Учебники были разработаны так, чтобы на их примере студенты смогли научиться извлекать признаки из разнородных данных, какие бывают при этом проблемы и как их решать. Также они научатся сводить задачи к формальным постановкам задачи машинного обучения и поймут, как проверять качество построенной модели на разных данных.

Пройдя этот курс, студент может получить знания о распространенных типах прикладных задач и суметь найти схемы их решения.

3.2.1. Структура курса

Курс содержит три лабораторных работы, которые решают задачи: классификации, локализации изображений и прогнозирование временных рядов.

Каждая лабораторная состоит из:

- название;

- введение - краткое описание решаемых задач;

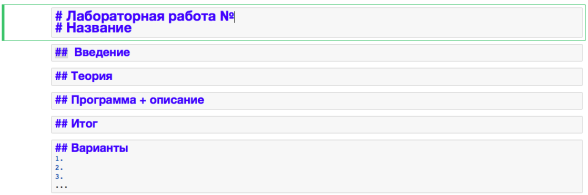


Рисунок 3.2. Структура работ

- теорию - информацию о наборах данных, сетях, использованных в работе и алгоритмах;
- разработанную программу и ее подробное описание;
- итог;
- варианты.

3.2.2. Реализация первой лабораторной работы

В разработанном учебнике показано, как создать простой классификатор изображений на основе сверточной нейронной сети и наборе данных CIFAR-10, с использованием библиотек TensorFlow и prettytensor - это прикладной интерфейс высокого уровня для TensorFlow, в разы упрощает работу с глубокими сетями.

На рисунке **??** показана структура нейронной сети. Сначала каждое изображение проходит через функцию искажения. Затем сеть будет состоять из двух сверточных слоев и слоев субдискретизации. После этого выходное изображение слоя подвыборки трансформируется в одномерный вектор (слоем Flatten) и проходит два полносвязных слоя. На всех слоях, кроме выходного полносвязного слоя, используется функция активации ReLU, последний же слой использует softmax.

ReLU - простой выпрямитель, который имеет формулу

$$f(x) = max(0, x)$$

. Softmax используется для того, что бы преобразовать вектор со второго полносвязного слоя в массив из 10 чисел (= количеству классов) с вероятностными значениями принадлежности к классу, которые в сумме дали бы 1.

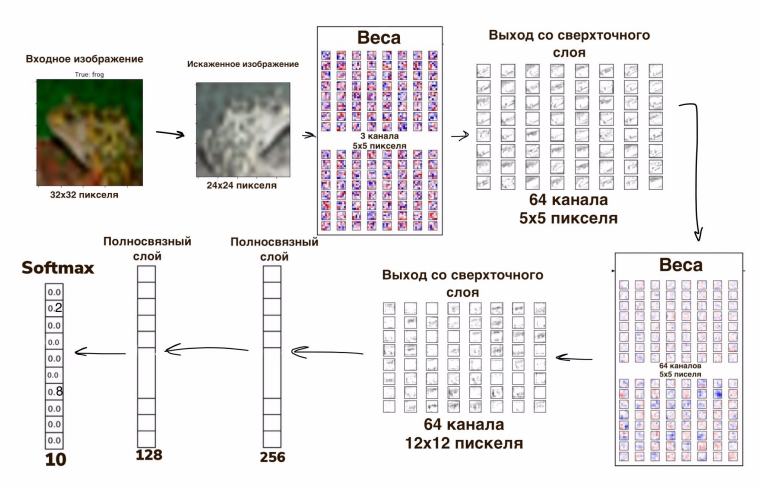


Рисунок 3.3. Структура сверточной нейронной сети

Структура сети

Функция случайного искажения изображения, принимает на вход изображение из набора данных CIFAR и случайно обрезает, переворачивает, изменяет контраст, яркость, насыщенность и оттенок. Это делается для того, чтобы искусственно увеличить объем выборки.

Результат показан на картинке 3.4. Слева представлена картинка с набора данных CIFAR, справа 9 разных искаженных изображений.

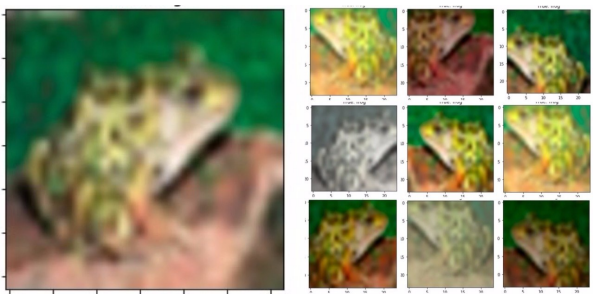


Рисунок 3.4. Пример работы функции "distort$_image$"

Сеть имеет следующие параметры:

- batch size = 32— количество обучающих образцов, обрабатываемых одновременно за одну итерацию алгоритма градиентного спуска;
- num epochs = 1000 — количество итераций обучающего алгоритма по всему обучающему множеству;
- kernel size = 5 — размер ядра в сверточных слоях;
- pool size = 2 — размер подвыборки в слоях подвыборки;
- depth = 64 — количество ядер в сверточных слоях;
- fc size1 = 256 — количество нейронов в 1ом полносвязном слое;
- fc size2 = 128.

При большом количестве итераций обучение может занять много времени, если обучать сеть без использования графического процессора. Поэтому в учебник был добавлен Saver. Saver может сохранять переменные в контрольные точки. Также он может найти последнюю контрольную точку и достать из нее переменные.

Таким образом процесс обучения можно отложить, а затем снова начать не потеряв результата.

В качестве функции потерь используется кросс-энтропия.

Модель достигает точности около 90 процентов на тестовом множестве, учитывая относительную простоту модели, это вполне достойный результат.

Варианты

Студентам предлагается выбрать себе набор данных из 15-ти представленных (Fashion-MNIST, Caltech-256, Linnaeus 5 и другие) и обучить сеть, опираясь на уже представленный код. В отчете построить и проанализировать веса и выходы с сверточных слоев.

3.2.3. Реализация второй лабораторной работы

Во второй лабораторной работе студентам предлагается создать свой набор данных и переобучить существующую модель глубокого обучения для задачи детектирования объектов, используя Tensorflow Object Detection API.

Введение

Первая часть работы заключается в том, чтобы показать как подключить и запустить уже существующую замороженную модель для локализации объектов. Во второй части предстоит собрать изображения, промаркировать их, разделить изображения на изображения для тренировки и теста, создать tfrecords для обеих выборок, настроить .config-файл , переобучить сеть, создать граф модели и протестировать на тестовой выборке.

Tensorflow Object Detection API - это платформа с открытым исходным кодом, созданная на основе TensorFlow. Она содержит несколько моделей машинного обучения, способных локализовать и классифицировать несколько объектов на одном изображении. Модели обучались на таких наборах данных как: COCO, Kitti и Open Images.

Модель для демонстрирования примера - ssd mobilenet v1 coco, так как она самая быстрая из представленных. Она использует алгоритм - Single Shot detectors [**?**].

Варианты

В качестве вариантов, предлагается выбрать объект, которого нет в словаре сети и переобучить модель.

3.2.4. Реализация третьей лабораторной работы

В этом примере показано, как прогнозировать данные временных рядов с использованием сети с длинной короткосрочной памятью (LSTM).

Чтобы прогнозировать значения будущих шагов времени последовательности, достаточно обучить сеть LSTM с последовательной регрессией, где ответы представляют собой обучающие последовательности со значениями, сдвинутыми на один временной шаг. То есть на каждом временном шаге входной последовательности сеть LSTM учится прогнозировать значение следующего шага времени.

В этом примере используется набор данных "monthly milk production pound". Набор охватывает период с января 1962 года по декабрь 1975 и содержит информацию о количестве произведенного молока в месяц коровой.

Набор данных

Загружаем этот набор данных с помощью библиотеки Pandas. Для предсказания нам не интересна дата, учитывая, что каждое наблюдение разделяется одним и тем же интервалом в один месяц. Поэтому мы можем исключить первый столбец.

```
In [5]: # первые пять строк набора данных
        dataframe = pandas.read_csv('data/monthly-milk-production-pounds-p.csv')
        dataframe.head()
```

Out[5]:

	Month	Monthly milk production: pounds per cow. Jan 62 ? Dec 75
0	1962-01	589.0
1	1962-02	561.0
2	1962-03	640.0
3	1962-04	656.0
4	1962-05	727.0

Рисунок 3.5. Структура нейронное сети

Пример обучает сеть LSTM прогнозировать количество молока, которое может произвести корова, учитывая ее предыдущие показатели.

Данные были разделены на тренировочный и тестовый набор в соотношении 90/10.

LSTM состоит из двух скрытых слоев, на первом -14 нейронов, на втором - 8 и выходным слоем. Функция активации relu, оптимизатор использует метод стохастических градиентов.

Среднеквадратичная ошибка составила 43.45 единиц для тренировочного набора и 45.4 единиц для тестового.

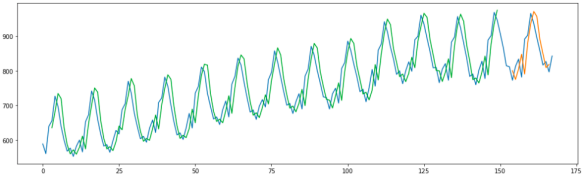


Рисунок 3.6. Структура нейронное сети

Варианты

В качестве вариантов предлагается студентам выбрать набор данных и сделать предсказания. Попытаться, варьируя количество нейронов в слоях, количество слоев и количество эпох - минимизировать ошибку.

4. Апробация курса лабораторных работ

В качестве тестирования курса, было прорешено по одному варианту для каждой лабораторной работы опираясь на представленный код. Отчеты были сохранены в формате notebook, с выходами (графики, ошибки по эпохам и прочее) и выводами.

4.1. Лабораторная работа 1

Был написан классификатор на наборе данных FASHION-mnist.

Архитектура сети аналогична представленной в учебнике из предыдущего раздела. Количество эпох обучения - 5000, что составило примерно 8 минут, на пк без графического процессора. Итоговая точность классификации: 90 процентов.

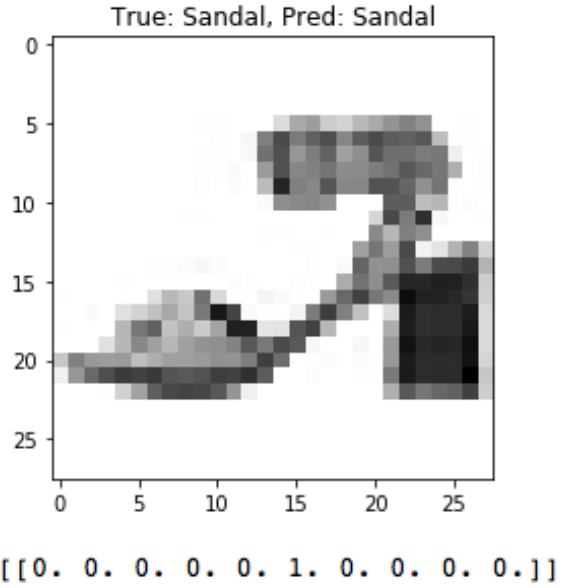


Рисунок 4.1. Матрица с предсказаниями

В отчете представлены графики выходов со сверточных слоев и

значения фильтров (веса).

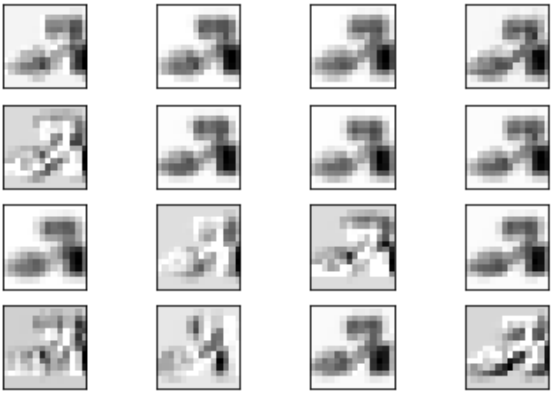


Рисунок 4.2. Выход с первого сверточного слоя

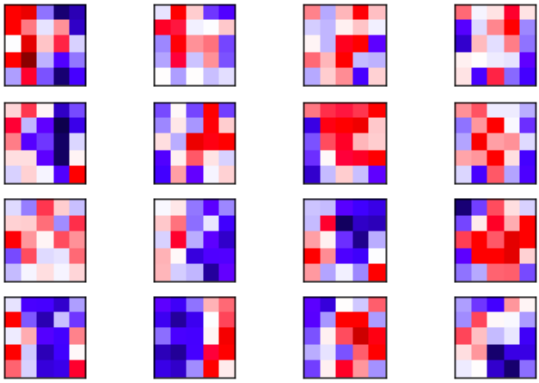


Рисунок 4.3. Значения весов (красные - положительные, синие - отрицательные значения)

4.2. Лабораторная работа 2

Для тестирования второй лабораторной работы, был создан набор данных: собраны и промаркированы изображения, конвертированы метки в формат TFrecord; переобучена модель ssd mobilenet v1 coco на более чем 1000 эпохах, что составило, примерно, 6 часов.

В отчете приведены графики ошибок классификации, локализации и общая ошибка обучения с помощью инструмента визуализации TensorBoard.

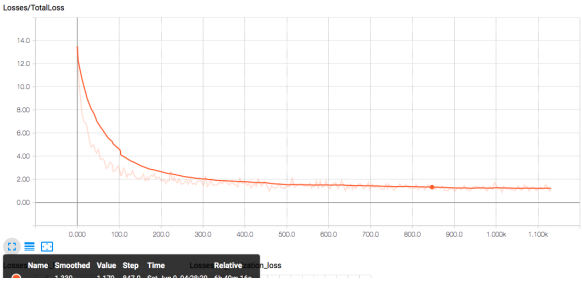


Рисунок 4.4. График ошибки обучения

Общая ошибка составила значение около единицы.

Рисунок 4.5. Результат обучения

В итоге, получили модель, которая может распознавать и детек-

тировать лягушку на изображениях.

4.3. Лабораторная работа 3

Аналогично первой лабораторной, был решен первый вариант из представленных.

Набор данных содержит информацию о температуре за период времени с 1 января 2010 до 31 декабря 2014 года. Также был разделен набор данных на тестовый и тренировочный в отношении 90 к 10.

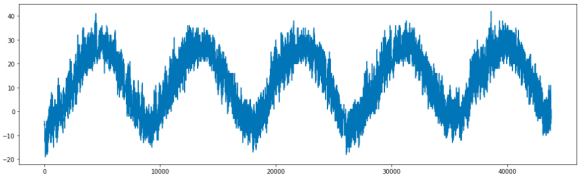


Рисунок 4.6. Исходный временной ряд

Для обучения использовали сеть с 1 входом, 1 скрытым слоем, с 8 нейронами и выходным слоем. Обучение было произведено на 200 эпохах и заняло около 15 минут.

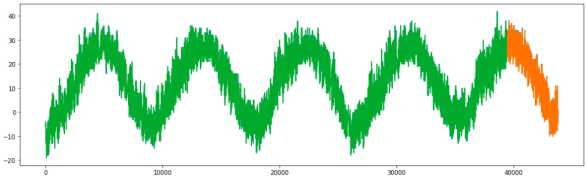


Рисунок 4.7. Результат предсказания (изображено желтым цветом)

Среднеквадратичная ошибка на тренировочном наборе составила - 1.5, на тестовом - 1.48 единиц. Это хороший результат, благодаря тому, что на исходной выборке явно прослеживался периодичный тренд.

ЗАКЛЮЧЕНИЕ

В результате выполнения бакалаврской работы было произведено исследование технологий глубокого обучения: рассмотрены существующие модели нейронных сетей и поставлены задачи для которых они применяются, а также рассмотрены различные программы глубокого обучения. Был произведен анализ лабораторных работ, выложенных в открытый доступ.

Среди множества программных инструментов для глубокого обучения были выбраны библиотеки TensorFlow и Keras. Они отвечают потребностям разрабатываемого курса, а именно подходит для решения поставленных задач.

Для визуализации полученных результатов была выбрана графическая веб-оболочка Jupyter Notebook. Проектирование проводилось в графической веб-оболочке Jupyter Notebook. В ней же и были сохранены сами лабораторные работы.

Практическая значимость разработки учебно-методического комплекса лабораторных работ, заключается в возможности внедрения её в учебный процесс.

Процесс прохождения курса лабораторных работ по дисциплине "исследование моделей глубокого обучения"состоит из 3 работ. Каждая лабораторная направлена на изучение различных задач: классификации, локализации изображений и прогнозирование временных рядов. По их завершению обучающиеся научатся сводить разнообразные задачи к формальным постановкам задач машинного обучения и применять необходимые технологии и схемы для их решения.

Задачи бакалаврской работы были выполнены, а цели - достигнуты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Caffe DocumentationElectronic resource. — URL: http://caffe.berkeleyvision.org.
2. Deeplearning4j DocumentationElectronic resource. — URL: https://deeplearning4j.org/documentation.
3. Dlib DocumentationElectronic resource. — URL: http://dlib.net.
4. Keras DocumentationElectronic resource. — URL: https://keras.io.
5. CNTK DocumentationElectronic resource. — URL: https://github.com/Microsoft/CNTK/tree/master/Documentation.
6. Mxnet DocumentationElectronic resource. — URL: https://mxnet.incubator.apache.org/faq/why_mxnet.html.
7. Neural Designer IntroductionElectronic resource. — URL: https://www.neuraldesigner.com.
8. opennn DocumentationElectronic resource, opennn. — URL: http://www.opennn.net/documentation/.
9. Tensorflow DocumentationElectronic resource. — URL: https://www.tensorflow.org/api_docs/.
10. Theano DocumentationElectronic resource. — URL: http://deeplearning.net/software/theano/.
11. Torch DocumentationElectronic resource. — URL: http://torch.ch/docs/getting-started.html.
12. McCulloch W.S. Pitts W. A Logical Calculus of the Ideas Immanent in Nervous Activity. — The Bulletin of Mathematical Biophysics, 1943. — 133 p.
13. NVIDIAElectronic resource. — URL: https://www.nvidia.ru/deep-learning-ai/education/.

ПРИЛОЖЕНИЕ 1
Настройка рабочей среды

Установка Python.

```
sudo apt-get install python3
```

Установка для CPU. Установка TensorFlow:

```
pip install --ignore-installed --upgrade tensorflow
```

Для проверки корректности установки TensorFlow, запустите python и выполните программу:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

В результате должно быть напечатано:

```
b'Hello, TensorFlow!'
```

Установка для GPU.

Замечание! TensorFlow поддерживает GPU компании NVIDIA с CUDA Compute Capability 3.5 и выше. Видеокарты AMD и других производителей для TensorFlow не подходят.

Установка TensorFlow:

```
pip install --upgrade tensorflow-gpu
```

Проверка установки. Для проверки корректности установки TensorFlow, запустите python и выполните программу:

```
import tensorflow as tf
tf.test.gpu_device_name()
```

Установка Keras.

Замечание! Начиная с версии 1.4 TensorFlow включает Keras. Поэтому отдельно устанавливать Keras не обязательно: pip install keras

Установка Jupyter Notebook.

Разверните среду разработки Python 3, в которую вы хотите установить Jupyter Notebook (в данном руководстве среда условно называется "my $_e nv$").

```
cd ~/environments

. my_env/bin/activate
```

Затем нужно обновить pip:

```
pip install --upgrade pip
```

Чтобы установить Jupyter Notebook, запустите:

```
pip install jupyter
```

Для того что бы запустить jupyter notebook, достаточно ввести в терминале

```
jupyter notebook
```

Также можно ввести в строку браузера адрес http://localhost:8080 (по умолчанию).

ПРИЛОЖЕНИЕ 2

Лабораторная работа. Классификация изображений

1 Лабораторная работа 1

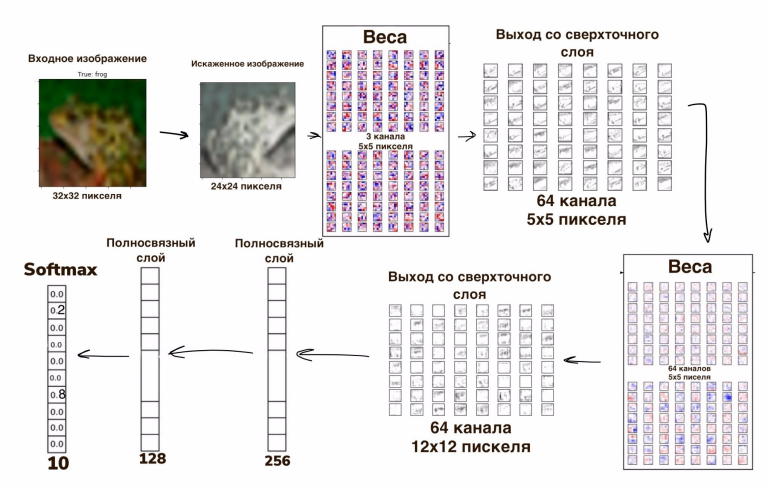
1.1 Классификация изображений

1.2 Введение

В этом примере показано, как создать простой классификатор изображений на основе сверточной нейронной сети и наборе данных CIFAR-10.

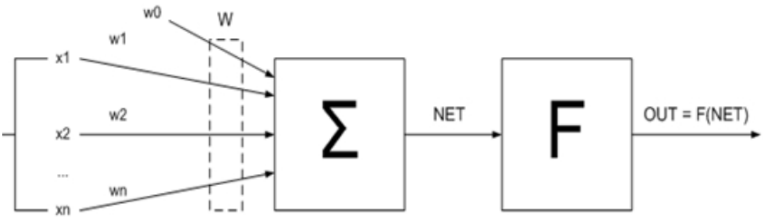
1.3 Структура сверточной нейронной сети

CHC состоит из разных видов слоев: сверточные (convolutional) слои, субдискретизирующие (subsampling, pooling) слои и полносвязный слои.



Первые два типа слоев (convolutional, subsampling), чередуясь между собой, формируют входной вектор признаков для многослойного персептрона.

1.4 Модель нейрона



$$NET = \sum_{i=1}^{n} w_i + x_i + w_0,$$

где * w_i - вес i нейрона; * x_i - выход i нейрона;
* w_0 - смещение; * n - количество синаптических связей, входящих в нейрон.

1.5 Входной слой

В нашем случае, входные данные представляют из себя цветные изображения, размера 32x32 пикселей. Если размер будет слишком велик, то вычислительная сложность повысится, соответственно ограничения на скорость ответа будут нарушены, определение размера в данной задаче решается методом подбора. Если выбрать размер слишком маленький, то сеть не сможет выявить ключевые признаки. Каждое изображение разбивается на 3 канала: красный, синий, зеленый. Таким образом получается 3 изображения размера 32x32 пикселей.

Входной слой учитывает двумерную топологию изображений и состоит из нескольких карт (матриц), карта может быть одна, в том случае, если изображение представлено в оттенках серого.

Входные данные каждого конкретного значения пикселя нормализуются в диапазон от 0 до 1, с помощью функции нормализации:

$$f(p, min, max) = \frac{p - min}{max - min},$$

где * p - значение конкретного цвета пикселя (от 0 до 255); * min - минимальное значение пискеля;
* max - максимальное значение пикселя.

1.6 Сверточный слой

Сверточный слой представляет из себя набор карт признаков, у каждой карты есть синаптическое ядро (фильтр).

Количество карт определяется требованиями к задаче, если взять большое количество карт, то повысится качество распознавания, но увеличится вычислительная сложность. Исходя из анализа научных статей, в большинстве случаев предлагается брать соотношение один к двум,

то есть каждая карта предыдущего слоя (например, у первого сверточного слоя, предыдущим является входной) связана с двумя картами сверточного слоя.

Размер у всех карт сверточного слоя - одинаковы и вычисляются по формуле:

$$(w,h) = (mW - kW + 1, mH - kH + 1),$$

где * (w,h) - вычисляемый размер сверточной карты; * mW - ширина предыдущей карты; * mH - высота предыдущей карты;
* kW - ширина ядра;
* kH - высота ядра.

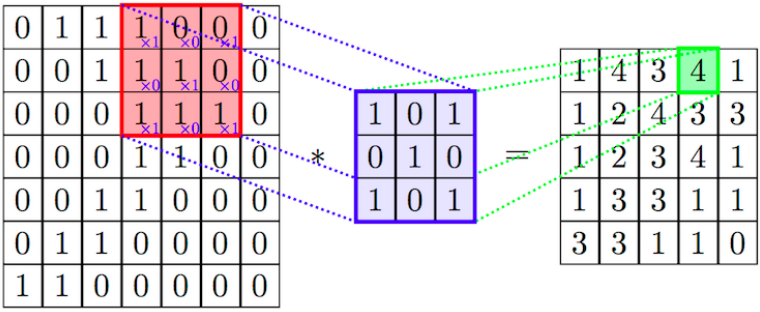
Ядро представляет из себя фильтр или окно, которое скользит по всей области предыдущей карты и находит определенные признаки объектов. Размер ядра обычно берут в пределах от 3x3 до 7x7. Если размер ядра маленький, то оно не сможет выделить какие-либо признаки, если слишком большое, то увеличивается количество связей между нейронами. Также размер ядра выбирается таким, чтобы размер карт сверточного слоя был четным, это позволяет не терять информацию при уменьшении размерности в подвыборочном слое, описанном ниже.

Ядро представляет собой систему разделяемых весов или синапсов, это одна из главных особенностей сверточной нейросети. В обычной многослойной сети очень много связей между нейронами, то есть синапсов, что весьма замедляет процесс детектирования. В сверточной сети – наоборот, общие веса позволяет сократить число связей и позволить находить один и тот же признак по всей области изображения.

Изначально значения каждой карты сверточного слоя равны 0. Значения весов ядер задаются случайным образом в области от -0.5 до 0.5. Ядро скользит по предыдущей карте и производит операцию свертка, которая часто используется для обработки изображений, формула:

$$(f * g)[m,n] = \sum f[m-k, n-l] * g[k,l],$$

где * f - исходная матрица изображения; * g - ядро свертки.

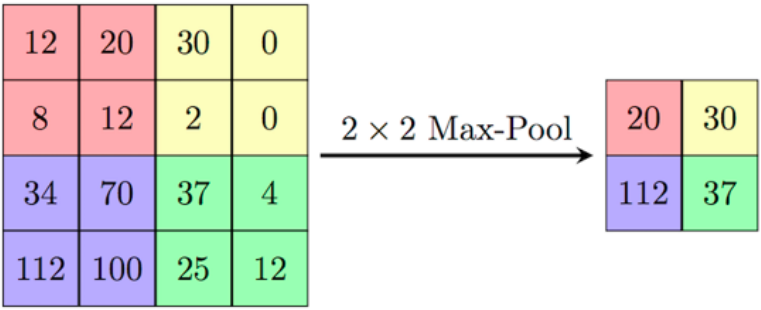


Неформально эту операцию можно описать следующим образом — окном размера ядра g проходим с заданным шагом все изображение f, на каждом шаге поэлементно умножаем содержимое окна на ядро g, результат суммируется и записывается в матрицу результата.

1.7 Подвыборочный слой

Подвыборочный слой также, как и сверточный имеет карты, но их количество совпадает с предыдущим слоем. Цель слоя – уменьшение размерности карт предыдущего слоя. Если на предыдущей операции свертки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. В процессе сканирования ядром подвыборочного слоя (фильтром) карты предыдущего слоя, сканирующее ядро не пересекается в отличие от сверточного слоя. Обычно, каждая карта имеет ядро размером 2x2, что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Вся карта признаков разделяется на ячейки 2x2 элемента, из которых выбираются максимальные по значению.

Обычно в подвыборочном слое применяется функция активации RelU. Операция подвыборки изображена на картинке:



Формально слой может быть описан формулой:

$$x^l = f(a^l * subsample(x^{l-1}) + b^l),$$

где * x^l - выход слоя l; * $f()$ - функция активации; * a^l, b^l - коэффициенты сдвига слоя l; * $subsample()$ - операция выборки локальных максимальных значений.

1.8 Полносвязный слой

Последний из типов слоев это слой обычного многослойного персептрона. Цель слоя – классификация, моделирует сложную нелинейную функцию, оптимизируя которую, улучшается качество распознавания.

Нейроны каждой карты предыдущего подвыборочного слоя связаны с одним нейроном скрытого слоя. Таким образом число нейронов скрытого слоя равно числу карт подвыборочного слоя, но связи могут быть не обязательно такими, например, только часть нейронов какой-либо из карт подвыборочного слоя быть связана с первым нейроном скрытого слоя, а оставшаяся часть со вторым, либо все нейроны первой карты связаны с нейронами 1 и 2 скрытого слоя. Вычисление значений нейрона можно описать формулой:

$$x_j^l = f(\sum_i x_j^{l-1} * w_i^{l-1} + b_j^{l-1}),$$

где * x_j^l - карта признаков j (выход слоя l); * $f()$ - функция активации; * b^l - коэффициенты сдвига слоя l; * w_i^{l-1} - матрица весовых коэффициентов слоя l.

1.9 Выходной слой

Выходной слой связан со всеми нейронами предыдущего слоя. Количество нейронов соответствует количеству распознаваемых классов, то есть в нашем случае 10.

Для уменьшения количества связей и вычислений для бинарного случая можно использовать один нейрон и при использовании в качестве функции активации гиперболический тангенс, выход нейрона со значением -1 означает, что на картинке нет распозноваемого класса, напротив выход нейрона со значением 1 – означает принадлежность к классу.

1.10 Выбор функции активации

Одним из этапов разработки нейронной сети является выбор функции активации нейронов. Вид функции активации во многом определяет функциональные возможности нейронной сети и метод обучения этой сети. Классический алгоритм обратного распространения ошибки хорошо работает на двухслойных и трехслойных нейронных сетях, но при дальнейшем увеличении глубины начинает испытывать проблемы. Одна из причин — так называемое затухание градиентов. По мере распространения ошибки от выходного слоя к входному на каждом слое происходит домножение текущего результата на производную функции активации. Производная у традиционной сигмоидной функции активации меньше единицы на всей области определения, поэтому после нескольких слоев ошибка станет близкой к нулю. Если же, наоборот, функция активации имеет неограниченную производную (как, например, гиперболический тангенс), то может произойти взрывное увеличение ошибки по мере распространения, что приведет к неустойчивости процедуры обучения.

В данной работе в качестве функции активации в скрытых и выходном слоях применяется ReLU.

Преимущества использования ReLU:

- ее производная равна либо единице, либо нулю, и поэтому не может произойти разрастания или затухания градиентов, т.к. умножив единицу на дельту ошибки мы получим дельту ошибки, если же мы бы использовали другую функцию, например, гиперболический тангенс, то дельта ошибки могла, либо уменьшиться, либо возрасти, либо остаться такой же, то есть, производная гиперболического тангенса возвращает число с разным знаком и величиной, что можно сильно повлиять на затухание или разрастание градиента. Более того, использование данной функции приводит к прореживанию весов;
- вычисление сигмоиды и гиперболического тангенса требует выполнения ресурсоемких операций, таких как возведение в степень, в то время как ReLU может быть реализован с помощью простого порогового преобразования матрицы активаций в нуле;
- отсекает ненужные детали в канале при отрицательном выходе.

Из недостатков можно отметить, что ReLU не всегда достаточно надежна и в процессе обучения может выходить из строя. Например, большой градиент, проходящий через ReLU, может привести к такому обновлению весов, что данный нейрон никогда больше не активируется. Если это произойдет, то, начиная с данного момента, градиент, проходящий через этот нейрон, всегда будет равен нулю. Соответственно, данный нейрон будет необратимо выведен из строя. Например, при слишком большой скорости обучения, может оказаться, что до 40%

ReLU никогда не активируются. Эта проблема решается посредством выбора надлежащей скорости обучения.

1.11 Импорты

```
In [1]: %matplotlib inline
        from sklearn.metrics import confusion_matrix
        import matplotlib.pyplot as plt
        from datetime import timedelta
        import prettytensor as pt
        import tensorflow as tf
        import urllib.request
        import numpy as np
        import tarfile
        import zipfile
        import pickle
        import math
        import time
        import sys
        import os
```

1.12 Загружаем набор данных

Изображение в наборе данных CIFAR-10 цветные (то есть у них три канала: красный, зеленый и синий) содержат 10 классов, имеют размер 32x32 пикселя. Объявим эти переменные, так как дальше они будут часто использоваться.

```
In [2]: img_size = 32
        cropped_size = 24

        num_clsss = 10
        num_channels = 3
```

Для начала загрузим набор данных cifar-10:

```
In [3]: def download_and_extract():
            """
            Загружает и распаковывает набор данных cifar-10, если его еще нет в data_path
            """

            url="https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz"
            download_dir="data/CIFAR-10/"
            filename = url.split('/')[-1]
            file_path = os.path.join(download_dir, filename)

            if not os.path.exists(file_path):
                # создает директорию для загрузки, если она не существует
                if not os.path.exists(download_dir):
                    os.makedirs(download_dir)
```

```
                # скачивание файла с интернета
                file_path, _ = urllib.request.urlretrieve(url=url,
                                                          filename=file_path)

                if file_path.endswith(".zip"):
                    # распаковка zip
                    zipfile.ZipFile(file=file_path, mode="r").extractall(download_dir)
                elif file_path.endswith((".tar.gz", ".tgz")):
                    # распаковка tar
                    tarfile.open(name=file_path, mode="r:gz").extractall(download_dir)

                print("Done")
            else:
                print("Data has already been downloaded and unpacked")

        download_and_extract()
```

```
Data has already been downloaded and unpacked
```

и названия классов:

```
In [4]: def return_data(filename):
            file_path = os.path.join("data/CIFAR-10/",
                                     "cifar-10-batches-py/",
                                     filename)
            with open(file_path, mode='rb') as file:
                data = pickle.load(file, encoding='bytes')
            return data

        def load_class_names():
            raw = return_data(filename="batches.meta")[b'label_names']
            names = [x.decode('utf-8') for x in raw]
            return names

        class_names = load_class_names()
        class_names
```

```
Out[4]: ['airplane',
         'automobile',
         'bird',
         'cat',
         'deer',
         'dog',
         'frog',
         'horse',
         'ship',
         'truck']
```

Набор представляет 10 различных классов: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик.

Загрузим тренировочный и тестовый наборы:

In [5]:
```
def label_generation(class_numbers, num_clsss=None):
    """
    Генерирует метки классов в формате 1-мерного массива заполненым нулями
    с одним единичным элементом, чей индекс = номеру класса
    """

    if num_clsss is None:
        num_clsss = np.max(class_numbers) + 1

    return np.eye(num_clsss, dtype=float)[class_numbers]
```

In [6]:
```
def convert_images(raw):
    """
    Конвертирует изображения с набора данных CIFAR-10
    в формат 4ех мерный массив [image_number, height, width, channel]
    """

    # конвертирует изображения из формата raw в точки float
    raw_float = np.array(raw, dtype=float) / 255.0

    # преобразует массив в 4-ех мерный
    images = raw_float.reshape([-1, num_channels, img_size, img_size])

    # изменяет порядок индексов
    images = images.transpose([0, 2, 3, 1])

    return images
```

In [7]:
```
def load_data(filename):
    """
    Загружает файл данных из набора данных CIFAR-10
    и возвращает конвертированное изображение с номером класса для каждого изображения
    """

    # загружаем файл
    data = return_data(filename)
    raw_images = data[b'data']

    # загружаем номер класса
    cls = np.array(data[b'labels'])

    # конвертируем изображение
    images = convert_images(raw_images)

    return images, cls
```

```
In [8]: def load_training_data():
            """
            Загружает все картинки из CIFAR-10.
            Возвращает картинки, номера классов и матрицы принадлежностей.
            """

            # инициализируем массивы
            images = np.zeros(shape=[5 * 10000, img_size, img_size, num_channels], dtype=float)
            cls = np.zeros(shape=[5 * 10000], dtype=int)

            begin = 0

            for i in range(5):
                # загружаем изображение и номер класса
                images_batch, cls_batch = load_data(filename="data_batch_" + str(i + 1))
                num_images = len(images_batch)
                end = begin + num_images
                images[begin:end, :] = images_batch
                cls[begin:end] = cls_batch
                begin = end

            return images, cls, label_generation(class_numbers=cls, num_clsss=num_clsss)

In [9]: def load_test_data():
            """
            Загружает все картинки для тренировки из CIFAR-10.
            Возвращает картинки, номера классов и матрицы принадлежностей.
            """

            images, cls = load_data(filename="test_batch")
            return images, cls, label_generation(class_numbers=cls, num_clsss=num_clsss)

In [10]: images_train, cls_train, labels_train = load_training_data()
         images_test, cls_test, labels_test = load_test_data()
```

Набор данных CIFAR-10 теперь загружен и состоит из 60 000 изображений и связанных меток. Он разбит на 2 взаимоисключающих подмножества: тренировочный набор (50000 изображений) и тестовый набор (10000 изображений).

```
In [11]: print("Training-set:\t" + str(len(images_train)))
         print("Test-set:\t" + str(len(images_test)))
```

```
Training-set:        50000
Test-set:        10000
```

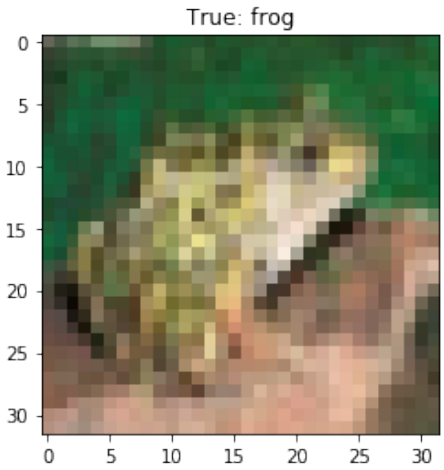
1.12.1 Функция для построения изображений

```
In [12]: def plot_image(image, cls_true, cls_pred=None):
             """
```

```
    Выводим изображение и его истинный класс и если есть - предсказанный
    """

    plt.imshow(image[0, :], interpolation='nearest')
    cls_true_name = class_names[cls_true[0]]

    # отображение названия истинного класса
    if cls_pred is None:
        xlabel = "True: {0}".format(cls_true_name)
    else:
        # отображение названия прогнозируемого класса
        xlabel = "True: {0}\nPred: {1}".format(cls_true_name, class_names[cls_pred[0]])

    plt.title(xlabel)
    plt.imshow(image[0, :, :, :], interpolation='nearest')
    plt.cls_true_name = class_names[cls_true[0]]
    plt.show()
```

Проверим, как будут отображаться данные:

```
In [13]: image_number = 512
         image = images_test[image_number:image_number+1]
         cls_true = cls_test[image_number:image_number+1]
         plot_image(image = image, cls_true=cls_true)
```



1.13 Граф тензора

Цель TensorFlow состоит в том, чтобы получить так называемый вычислительный граф, который может выполняться намного эффективнее, чем если бы те же вычисления выполнялись непосредственно в Python. TensorFlow может быть более эффективным, чем NumPy, потому что TensorFlow знает весь граф вычислений, который должен быть выполнен, а NumPy знает только вычисление одной математической операции за раз.

TensorFlow также может автоматически вычислять градиенты, которые необходимы для оптимизации переменных графа, чтобы сделать модель более эффективной. Это связано с тем, что граф представляет собой комбинацию простых математических выражений, поэтому градиент всего графа может быть рассчитан с использованием цепного правила для производных.

1.13.1 Placeholder variables

Placeholder – это просто переменная, предназначенная для определения позже. Эти переменные позволяют нам определять наши операции и строить граф вычислений без предварительного ввода данных. В терминологии TensorFlow через placeholder-ы мы передаем данные графу. Работает это следующим образом:

```
In [14]: x = tf.placeholder(tf.float32, shape=[None, img_size, img_size, num_channels], name='x'
```

Эта переменная будет хранить в себе истинные метки.

```
In [15]: lab_true = tf.placeholder(tf.float32, shape=[None, num_clsss], name='lab_true')
```

Мы могли бы также создать placeholder-переменную для номеров истинных классов, но легче вернуть индекс элемента равного 1.

```
In [16]: true_cls = tf.argmax(lab_true, dimension=1)
```

1.13.2 Функция для создания предварительной обработки

Эта функция добавляет узел к вычислительному графу для TensorFlow и предназначена для "расширения"обучающей выборки. Она рандомно искажает каждое изображение.

```
In [17]: def distort_image(img, training):
             """
             Принимает на вход изображение и значение типа bool, которое отслеживает
             изображение взято из тренировочной выборки или тестовой
             """

             if training:
                 # Рандомно обрезает изображение
                 img = tf.random_crop(img, size=[cropped_size, cropped_size, num_channels])

                 # Рандомно переворачивает изображение по горизонтали
                 img = tf.image.random_flip_left_right(img)

                 # Изменяет контраст, яркость, насыщенность и оттенок
                 img = tf.image.random_hue(img, max_delta=0.05)
```

```
            img = tf.image.random_contrast(img, lower=0.3, upper=1.0)
            img = tf.image.random_brightness(img, max_delta=0.2)
            img = tf.image.random_saturation(img, lower=0.0, upper=2.0)

            # Защита от переполнения
            img = tf.minimum(img, 1.0)
            img = tf.maximum(img, 0.0)
        else:
            # Обрезает изображение вокруг центра, так, чтобы по размеру оно было такое же,
            # как и изображение из тренировочной выборки
            img = tf.image.resize_image_with_crop_or_pad(img,
                                                         target_height = cropped_size,
                                                         target_width = cropped_size)

        return img

    def handling(images, training):
        images = tf.map_fn(lambda image: distort_image(image, training), images)
        return images

    distorted_images = handling(images=x, training=True)
```

1.13.3 Создание сверточной сети

С помощью функции `network` конструируем сеть. Мы будем использовать Pretty Tensor. В этой функции создается сеть с двумя сверточными слоями (`conv_1`, `conv_2`), с 64 ядрами размером = 5, двумя слоями субдискретизации (`max_pool`), которые уменьшают изображение в 2 раза, двумя полносвязными слоями (`fc_1` - 256 нейронов, `fc_2` - 128 нейронов) и слоем классификации (`softmax_classifier`).

In [18]:
```
def network(images, training):
    # складываем изображение в объект Pretty Tensor
    x_pretty = pt.wrap(images)

    if training:
        phase = pt.Phase.train
    else:
        phase = pt.Phase.infer

    # создаем сверточную нейронную сеть используя Pretty Tensor
    with pt.defaults_scope(activation_fn=tf.nn.relu, phase=phase):
        y_pred, loss = x_pretty.\
            conv2d(kernel=5, depth=64, name='conv_1', batch_normalize=True).\
            max_pool(kernel=2, stride=2).\
            conv2d(kernel=5, depth=64, name='conv_2').\
            max_pool(kernel=2, stride=2).\
            flatten().\
            fully_connected(size=256, name='fc_1').\
```

```
                fully_connected(size=128, name='fc_2').\
                softmax_classifier(num_classes=num_clsss, labels = lab_true)

        return y_pred, loss
```

Функция `create_network` создает нейронную сеть. Нейронная сеть объявлена как `network`.

```
In [19]: def create_network(training):
             with tf.variable_scope('network', reuse=not training):
                 images = handling(images=x, training=training)
                 y_pred, loss = network(images=images, training=training)
             return y_pred, loss
```

Переменная `global_step` будет отслеживать количество эпох оптимизации. В дальнейшем она нам понадобится для сохранения процесса обучения. `trainable=False` означает, что TensorFlow не будет пытаться оптимизировать эту переменную.

```
In [20]: global_step = tf.Variable(initial_value=0,
                                   name='global_step', trainable=False)
```

Создаем нейронную сеть, которая будет использоваться для обучения.

```
In [21]: _, loss = create_network(training=True)
```

Создаем оптимизатор, который минимизирует `loss`. Также передаем переменную `global_step` в оптимизатор, чтобы она была увеличена на единицу после каждой итерации.

`learning_rate` - это коэффициент скорости обучения, он выбирается в диапазоне от 0 до 1. Ноль указывать бессмысленно, поскольку в этом случае корректировка весов вообще производиться не будет. Выбор параметра противоречив. Большие значения (0,7 – 1) будут соответствовать большому значению шага коррекции. При этом алгоритм будет работать быстрее (т.е. для поиска минимума функции ошибки потребуется меньше шагов), однако может снизиться точность настройки на минимум, что потенциально увеличит ошибку обучения. Малые значения коэффициента (0,1 – 0,3) соответствуют меньшему шагу коррекции весов. При этом число шагов (или эпох), требуемое для поиска оптимума, как правило, увеличивается, но возрастает и точность настройки на минимум, что потенциально уменьшает ошибку обучения. В данном случае он равен `1e-4` или `0,0001`. Чаще всего на практике этот коэффициент выбирают экспериментально.

```
In [22]: optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(loss, global_step=globa
```

Теперь создайте нейронную сеть для этапа тестирования.

```
In [23]: y_pred, _ = create_network(training=False)
```

`y_pred` представляет собой массив с 10 элементами. В `pred_cls` будет записан индекс самого большого элемента в массиве.

```
In [24]: pred_cls = tf.argmax(y_pred, dimension=1)
```

Затем мы создаем вектор булевых строк, показывающий нам, является ли предсказанный класс равным истинному классу каждого изображения.

```
In [25]: correct_pred = tf.equal(pred_cls, true_cls)
```

Вектор `correct_pred` переводится из bool в float, так что: False становится 0, а True становится 1. Далее, расчитывается точность классификации, как среднее полученных чисел.

```
In [26]: acc = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

1.13.4 Saver

Объект типа Saver создается для того чтобы сохранить переменные нейронной сети, чтобы их можно было перезагрузить быстро, без необходимости снова тренировать сеть.

```
In [27]: saver = tf.train.Saver()
```

1.13.5 Веса и выходы с слоев

Мы использовали имена `conv_1` и `conv_2` для двух сверточных слоев. Pretty Tensor автоматически дает имена переменным, которые он создает для каждого слоя, поэтому мы можем получить веса для слоя с использованием имени.

```
In [28]: def get_weights(layer_name):
             with tf.variable_scope("network/" + layer_name, reuse=True):
                 variable = tf.get_variable('weights')

             return variable
```

Получаем переменные по каждому слою.

```
In [29]: weights_conv_1 = get_weights(layer_name='conv_1')
         weights_conv_2 = get_weights(layer_name='conv_2')
```

Аналогичным образом полуаем выходы со слоев. Функция немного отличается от выше приведенной, потому что нас интересует лишь последний тензор.

```
In [30]: def get_layer_output(layer_name):
             tensor_name = "network/" + layer_name + "/Relu:0"
             tensor = tf.get_default_graph().get_tensor_by_name(tensor_name)
             return tensor
```

Получаем выходы со сверточных слоев:

```
In [31]: output_conv_1 = get_layer_output(layer_name='conv_1')
         output_conv_2 = get_layer_output(layer_name='conv_2')
```

1.13.6 Создаем сессию TensorFlow

Объект Session инкапсулирует среду, в которой выполняются объекты операции, и оцениваются объекты тензора.

```
In [32]: session = tf.Session()
```

1.13.7 Контрольные точки

Обучение нейронной сети может занять много времени, особенно если у вас нет графического процессора. Поэтому мы сохраняем контрольные точки во время обучения, для того чтобы продолжить обучение в другое время (например, в ночное), а также для проведения анализа без необходимости тренировать нейронную сеть каждый раз, когда мы хотим ее использовать.

```
In [33]: # каталог для хранения контрольных точек
         save_dir = 'checkpoints/'

         # создаем каталог, если она еще не существует
         if not os.path.exists(save_dir):
             os.makedirs(save_dir)

         save_path = os.path.join(save_dir, 'cifar10_cnn')
```

Попробуйте восстановить контрольную точку:

```
In [34]: try:
             # нахождение контрольной точки
             last_chk_path = tf.train.latest_checkpoint(checkpoint_dir=save_dir)

             # пробуем восстановить последнюю кт
             saver.restore(session, save_path=last_chk_path)

             print("Restored checkpoint from:", last_chk_path)
         except:
             session.run(tf.global_variables_initializer())
```

```
INFO:tensorflow:Restoring parameters from checkpoints/cifar10_cnn-11110
Restored checkpoint from: checkpoints/cifar10_cnn-11110
```

1.13.8 Объем партии

В тренировочном наборе 50 000 изображений. Для вычисления градиента модели с использованием всех этих изображений требуется много времени. Поэтому мы используем только небольшую партию изображений на каждой итерации.

Это число можно варьировать. С увеличением этой переменной можно сокращать количество эпох и наоборот. При высоком значении - RAM будет перегружаться, а при маленьком будет высокая ошибка.

Эта функция выбирает случайный индекс из массива тренировочной выборки. И возвращает последующие batch_size картинок и их метки.

```
In [35]: batch_size = 100

         def random_batch():
             i = np.random.choice(len(images_train), size=batch_size, replace=False)
             return images_train[i, :, :, :], labels_train[i, :]
```

1.13.9 Оптимизация

Эта функция выполняет ряд эпох оптимизации. На каждой эпохе из набора тренировок выбирается новая партия данных, а затем TensorFlow выполняет оптимизатор с использованием этих данных. Прогресс печатается каждые 10 эпох. Контрольная точка сохраняется каждые 1000 эпох, а также после последней эпохи.

```
In [36]: def optimize(num_iterations):
             # используем таймер, для того, чтобы увидеть время
             start_time = time.time()

             for i in range(num_iterations):
                 x_batch, y_true_batch = random_batch()

                 feed_dict_train = {x: x_batch,
                                    lab_true: y_true_batch}

                 # запускаем оптимизатор
                 i_global, _ = session.run([global_step, optimizer],
                                           feed_dict=feed_dict_train)

                 # печатаем результат каждые 10 итераций
                 if (i_global % 10 == 0) or (i == num_iterations - 1):
                     # Срассчитываем ассигасу на тренировочной выборке
                     batch_acc = session.run(acc,
                                             feed_dict=feed_dict_train)

                     # выводим статус
                     msg = "Epoch: {0:>6}, Accuracy: {1:>6.1%}"
                     print(msg.format(i_global, batch_acc))

                 # сохраняется последняя кт и после каждых 1000 эпох
                 if (i_global % 1000 == 0) or (i == num_iterations - 1):
                     saver.save(session,
                                save_path=save_path,
                                global_step=global_step)

                     print("Saved checkpoint.")

             # рассчитываем и выводим конечное время
             end_time = time.time()
             time_dif = end_time - start_time
             print("Time usage: " + str(timedelta(seconds=int(round(time_dif)))))
```

1.13.10 Построение сверточных весов

```
In [37]: def plot_conv_weights(weights, input_channel=0):
             w = session.run(weights)
```

```
        w_min = np.min(w)
        w_max = np.max(w)

        abs_max = max(abs(w_min), abs(w_max))

        num_filters = w.shape[3]
        num_grids = math.ceil(math.sqrt(num_filters))

        fig, axes = plt.subplots(num_grids, num_grids)

        for i, ax in enumerate(axes.flat):
            if i<num_filters:
                img = w[:, :, input_channel, i]
                ax.imshow(img, vmin=-abs_max, vmax=abs_max,
                          interpolation='nearest', cmap='seismic')

            ax.set_xticks([])
            ax.set_yticks([])

        return plt
```

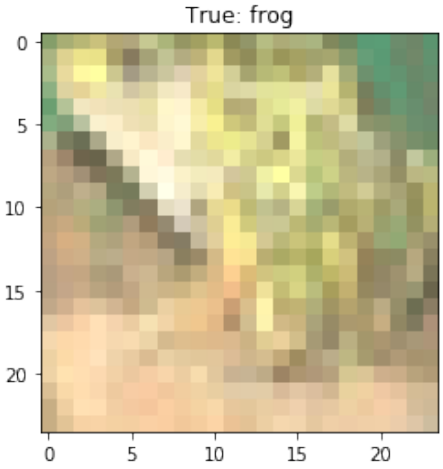
1.13.11 Построение выходов с светочных слоев

```
In [38]: def plot_layer_output(layer_output, image):
        feed_dict = {x: [image]}

        values = session.run(layer_output, feed_dict = feed_dict)

        values_min = np.min(values)
        values_max = np.max(values)

        num_images = values.shape[3]
        num_grids = math.ceil(math.sqrt(num_images))

        fig, axes = plt.subplots(num_grids, num_grids)

        for i, ax in enumerate(axes.flat):
            if i < num_images:
                img = values[0, :, :, i]

                ax.imshow(img, vmin = values_min, vmax = values_max,
                          interpolation = 'nearest', cmap = 'binary')

            ax.set_xticks([])
            ax.set_yticks([])

        plt.show()
```

1.14 Пример искаженных изображений

```
In [39]: def plot_distorted_image(img, cls_true):
             img_2 = img[np.newaxis, :, :, :]
             feed_dict = {x: img_2}
             result = session.run(distorted_images, feed_dict=feed_dict)
             plot_image(image = result, cls_true = np.repeat(cls_true, 1))
```

```
In [40]: img = images_test[image_number, :, :, :]
         cls = cls_test[image_number]
         plot_distorted_image(img, cls)
```



1.15 Запускаем оптимизацию

От количества итераций и аппаратных показателей вашего компьютера зависит время обучения.

Поскольку мы сохраняем контрольные точки во время оптимизации и восстанавливаем последнюю контрольную точку при перезапуске кода, мы можем остановить и продолжить оптимизацию позже.

```
In [62]: optimize(num_iterations=1)
```

```
Epoch:  11120, Accuracy:  73.0%
Saved checkpoint.
Time usage: 0:00:01
```

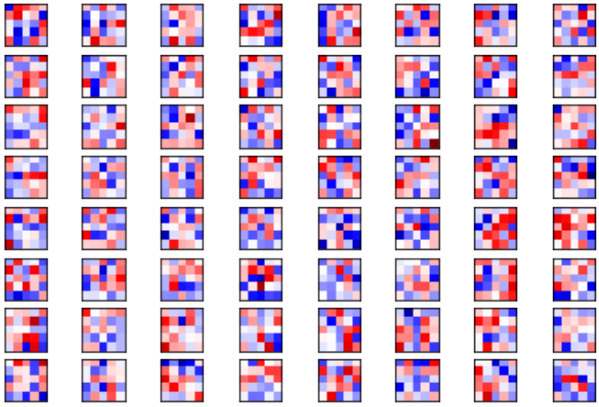
1.15.1 Веса

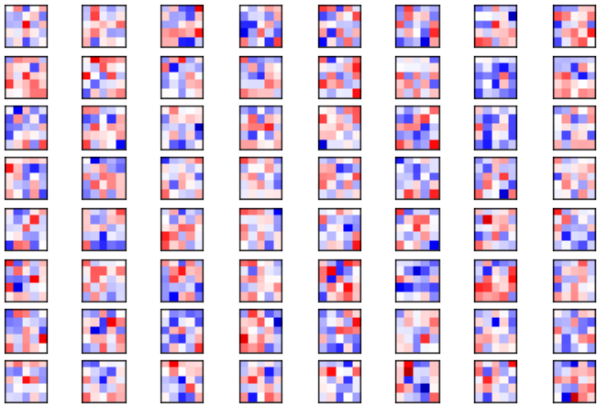
Посмотрим на веса с двух сверточных слев. Меняйте переменную `input_channel` (в диапазоне 0-2 для первого сверточного слоя и 0-63 для второго), чтобы посмотреть веса с других каналов.

Положительные веса - красные, отрицательные - синие.

```
In [42]: plot_conv_weights(weights=weights_conv_1, input_channel=0)
         plot_conv_weights(weights=weights_conv_2, input_channel=63)
```

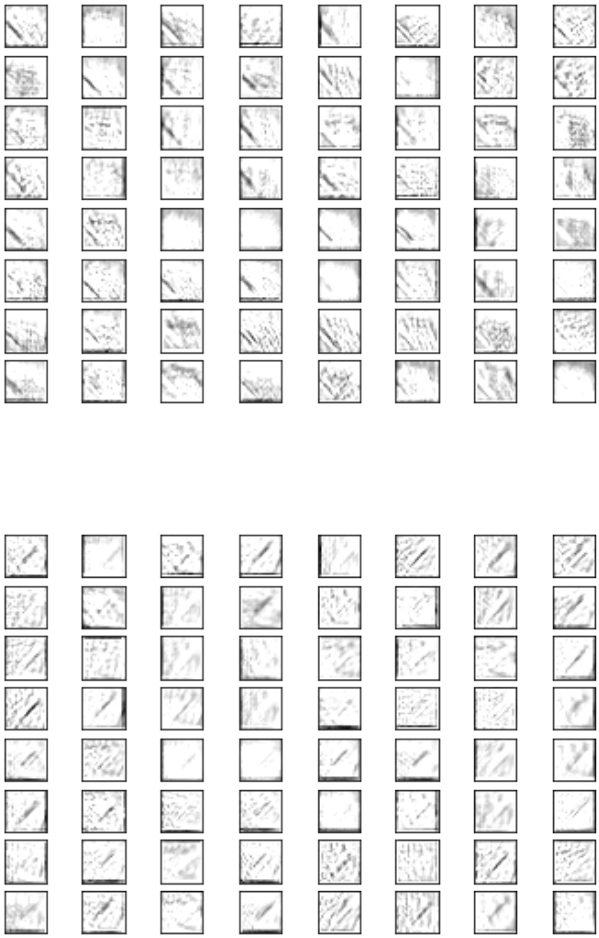
```
Out[42]: <module 'matplotlib.pyplot' from '/anaconda3/lib/python3.6/site-packages/matplotlib/pyp
```





2 Выходы с сверточных слоев

```
In [43]: plot_layer_output(output_conv_1, image = img)
         plot_layer_output(output_conv_1, image = img)
```



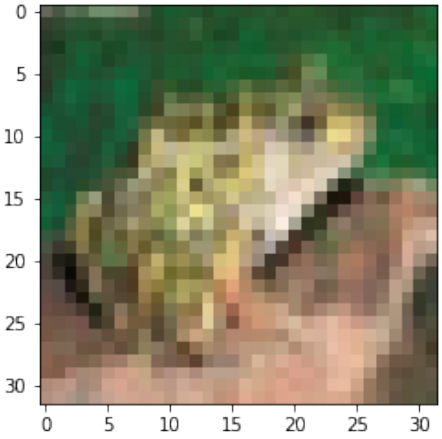
На данном графике сложно различить, какие же признаки сеть выявила после обучения. Но nhoэкспериментировав на разных классах можно заметить, что для машин это, например, колеса, прямые линии. Для лягушки - фактура.

2.1 Предсказания

Посмотрим на результаты предсказания:

```
In [53]: lbl_pred, cls_pred = session.run([y_pred, pred_cls],
                                          feed_dict={x: [img]})

         np.set_printoptions(precision=3, suppress=True)

         img = images_test[image_number, :, :, :]
         cls = cls_test[image_number]
         plt.imshow(img)
         print(lbl_pred[0])
```

```
[0.     0.     0.     0.     0.001  0.     0.999  0.     0.     0.  ]
```



Предсказание представляет из себя массив длинной 10. Каждый элемент массива показывает вероятность пренадлежности соответствующему классу.

В нашем случае на картинке лягушка, что соответствует седьмому классу и наше предсказание тоже считает, что это лягушка с вероятностью 0.999.

2.2 Закрытие TensorFlow сессии

По окончанию исспользования TensorFlow, обязательно закрывайте сессию.

```
In [45]: # session.close()
```

2.3 Заключение

В этом учебнике показано, как создать свернутую нейронную сеть для классификации изображений в наборе данных CIFAR-10. Точность классификации составляла около 80% на тестовом наборе.

2.4 Варианты

Создать свой классификатор на основе вышеприведенного на следующих наборах данных:

1. CIFAR-100 (взять любые 2-10 классов)
2. Fashion-MNIST
3. Linnaeus 5 dataset
4. Caltech-256
5. Caltech 101
6. CORe50

7.Cityscapes Dataset

8. HASYv2
9. MNIST