Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра компьютерной физики и метаматериалов
на базе Саратовского филиала Института
радиотехники и электроники
имени В. А. Котельникова РАН

Технология машинного обучения TenzorFlow

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ
студентки 2 курса
направления 03.03.02 «Физика» физического факультета

Клименко Анны Михайловны

Научный руководитель		
доцент, к.ф.- м.н.		А.С. Ремизов
должность, уч. степень, уч. звание	подпись, дата	инициалы, фамилия

Заведующий кафедрой		
д.ф.- м.н., профессор		В.М. Аникин
должность, уч. степень, уч. звание	подпись, дата	инициалы, фамилия

Саратов, 2017

Введение

Актуальность работы. В последние десятилетия в мире бурно развивается новая прикладная область математики, специализирующаяся на искусственных нейронных сетях, интерес к которым заметно возобновился после ряда новых успехов, начиная с 2012 года [1–16]. Актуальность исследований в этом направлении подтверждается массой различных применений нейросетевых алгоритмов. Это автоматизация процессов распознавания образов, адаптивное управление, аппроксимация функционалов, прогнозирование, создание экспертных систем, организация ассоциативной памяти и многие другие приложения. С помощью нейронных сетей можно, например, предсказывать показатели биржевого рынка, выполнять распознавание оптических или звуковых сигналов, создавать самообучающиеся системы, способные управлять автомашиной при парковке или синтезировать речь по тексту. Существует целый ряд программных продуктов, предоставляющих инструментарий для работы с нейросетями. Одно из наиболее удачных, популярных и при этом бесплатных решений и рассматривается в этой работе - фреймворк для машинного обучения TensorFlow от Google.

Целью данной работы ставится изучение основных возможностей технологии машинного обучения TenzorFlow, описание архитектуры и особенностей этой технологии, а также теоретических аспектов искусственных нейронных сетей, вокруг которых и построен фреймворк. В **задачи** работы входит методическое изложение теоретической базы, необходимой в работе; описание архитектуры фреймворка, его основных возможностей и способов применения при решении задач машинного обучения; программная реализация практических примеров.

Структура и объем работы. Выпускная квалификационная работа изложена на 56 страницах, состоит из введения, 3 глав, заключения и приложения. Библиографический список включает 16 наименований. Текст содержит один листинг программы распознавания рукописных цифр и иллюстрирован 21-м рисунком.

Основное содержание работы

Во введении к работе, помимо целей и задач, представлена предыстория фреймворка TensorFlow.

Первая глава посвящена краткому обзору "классических" нейронных сетей, даются основные модели нейронов, классификация типов нейронных сетей и методов их обучения.

Об относительно недавнем (с 2012 года) возрождении интереса к нейронным сетям рассказывается во **второй главе**. Там же подробно разобрана топология сверточных нейронных сетей, типы слоев, методы обучения, история появления и успеха этой архитектуры.

В **третьей главе** содержится как описание самой технологии машинного обучения TensorFlow, так и практическая часть - реализация примера решения задачи о распознавании рукописных цифр.

В практическом примере для обучения используется база данных MNIST [7], которая содержит 60000 изображений для обучения и 10000 изображений для тестирования. База данных MNIST ("Mixed National Institute of Standards and Technology") - объёмная база данных образцов рукописного написания цифр, является стандартом, предложенным Национальным институтом стандартов и технологий США с целью калибровки и сопоставления методов распознавания изображений с помощью машинного обучения, в первую очередь на основе нейронных сетей.

Данные MNIST размещены на веб-сайте Яна ЛеКуна (Yann LeCun). Подключение базы производится всего двумя строками:

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Данные загружаются в объект mnist, и разделены на три части: 55 000 позиций для обучения (mnist.train), 10 000 тестовых (mnist.test) и 5000 для валидации (mnist.validation).

Каждая точка данных MNIST имеет две части: изображение рукописной цифры и соответствующую метку. Изображения обозначаются как «x» и метки как «y». Оба набора данных для обучения и тестирования содержат изображения и соответствующие метки; Например, изображения для обучения - это mnist.train.images, а соответствующие метки - mnist.train.labels.

Каждое изображение имеет размер 28 пикселей на 28 пикселей - фактически это просто массив чисел, который можно представить в виде вектора из 28x28 = 784 чисел. В такой форме изображения MNIST представляют собой всего лишь множество точек в 784-мерном векторном пространстве. Уменьшение размерности отбрасывает информацию о двумерной структуре изображения, но для рядя лучших методов компьютерного зрения вполне достаточно и одномерного представления.

В результате векторизации изображений, mnist.train.images представляет собой тензор (в данной терминологии - n-мерный массив) размера [55000, 784]. Первое измерение - это индекс в список изображений, а второе измерение - индекс для каждого пикселя в каждом изображении. Каждая запись в тензоре представляет собой интенсивность пикселя между 0 и 1 для конкретного изображения. Записи изображений представлены объектом mnist.train.xs.

Каждое изображение в MNIST имеет соответствующую метку - число от 0 до 9, представляющее цифру, нарисованную на изображении. В данной задаче это число-метка кодируется 10ти мерным вектором, координаты которого принимают значения 0 или 1, причем единица может быть только в одной позиции, соответствующей кодируемому десятичному числу. Например, метка для числа 0 представлена вектором [1,0,0,0,0,0,0,0,0,0], а для числа 9 - [0,0,0,0,0,0,0,0,0,1]. В силу такого кодирования, набор меток mnist.train.labels представляет собой тензор [55000, 10].

Регрессионная модель Softmax

Перед нами стоит задача классификации. Мы знаем, что каждое изображение MNIST содержит рукописную цифру от нуля до девяти. Таким образом, существует только десять возможных вариантов классификации для данного изо-

бражения. Мы можем, проанализировав изображение, получить на выходе вероятность того, какая же цифра на нем изображена. Но, поскольку стопроцентное распознавание не гарантировано, то мы получим десять вероятностных оценок - по одной на каждую цифру. Это классический случай, когда в качестве естественной модели подходит softmax-регрессия (одно из обобщений логистической регрессии). В более сложных моделях, как правило, слой с softmax-регрессией присутствует на последнем шаге.

Регрессия softmax применятся в два этапа: сначала мы вычисляем некоторые признаки того, что наши входные данные находятся в определенных классах (обучающая выборка), а затем преобразуем эти признаки в вероятности. При вычислении признака, кроме суммы взвешенных пикселей, используется смещение, некоторый априорный параметр (bias). Это нужно для того, чтобы была возможность влиять на выходные вероятности независимо от входных данных. В результате признак для *i*-го класса и заданных входных данных *x*, вычисляется по формуле:

$$evidence_i = \sum_j W_{i,j} x_j + b_i$$

, где W_i - веса, а b_i - смещение для *i*-го класса, j - индекс для суммирования по пикселям на входном изображении x.

Затем мы преобразуем признаки в предсказательные вероятности y, используя функцию «softmax»:

$$y = softmax(evidence)$$

Здесь softmax служит функцией «активации», формируя вывод нашей линейной функции в нужную нам форму - в распределение вероятности по 10 случаям (для данной задачи). Т.е. происходит преобразование признаков в вероятности для каждого класса. Функция softmax является нормализованной экспонентой:

$$softmax(x) = normalize(\exp(x))$$

и в развернутом виде выглядит как:

$$softmax(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Рассмотренную регрессионную модель можно представить в компактной форме:

$$y = softmax(Wx + b)$$

Реализация регрессии

Для описания вычислений в tensorflow который используется язык Python. Однако на этом языке задается только граф потока данных, реальные вычисления запускаются механизмом сессий, который выполняет операции не на этом языке, а с помощью отдельного высокоэффективного и оптимизированного кода.

Чтобы использовать TensorFlow, сначала нам нужно его импортировать.

import tensorflow as tf

Взаимодействующие операции описываются символическими переменными, создадим одну:

x = tf.placeholder(tf.float32, [None, 784])

здесь x - это не конкретное значение. Это символическая ссылка на местозаполнитель, который будет принимать конкретные значения только тогда, когда мы попросим TensorFlow выполнить вычисление.

Нам нужна возможность работы с любым количеством изображений MNIST, трансформированных в 784-мерный вектор. Символьная ссылка x указывает на двухмерный тензор чисел с плавающей запятой, размерности [None, 784], где None означает, что измерение может иметь любую длину.

Также нам нужно где-то хранить веса и смещения для нашей модели. Можно было бы рассматривать их как дополнительные входные данные, и определить их аналогично x, но TensorFlow предоставляет более удобный способ - объект Variable. Variable - - это изменяемый тензор, который живет на графе взаимодействующих операций. Он может использоваться и даже модифицироваться в процессе вычислений. Для приложений машинного обучения, как правило, параметры модели хранятся в объектах Variable.

Создадим ссылки на объекты Variable для тензоров весов и смещений, проинициализировав из нулевыми начальными значениями:

W = tf.Variable(tf.zeros([784, 10]))

b = tf.Variable(tf.zeros([10]))

Обратите внимание, что тензор весов *W* имеет размерность [784, 10], поскольку нам нужно умножить его на 784-мерные векторы изображения, чтобы получить 10-мерные векторы признаков для различных классов. Тензор смещений *b* имеет размерность [10], поскольку его мы добавляем к выходу.

Задав входные данные, остается реализовать нашу модель. Для ее определения требуется всего лишь одна строка!

y = tf.nn.softmax(tf.matmul(x, W) + b)

Здесь мы с помощью метода tf.matmul умножаем тензоры x и W, затем добавляем смещения и к результату применяем встроенный метод tf.nn.softmax.

Вот и все. Нам понадобилась всего одна строка кода, чтобы определить нашу модель, после небольшой предварительной инициализации. Это не означает, что TensorFlow разработан, чтобы особенно просто работать с регрессией, это всего лишь пример того, насколько гибко и просто можно описывать многие виды вычислений - от моделей машинного обучения до физических симуляций. И как только мы задали определение, наша модель может работать на различных устройствах: процессоре компьютера, графической карте и даже на телефоне!

Обучение

Прежде чем подготовить нашу модель к обучению, нужно определить функцию ошибки. Мы должны задать некий критерий, по которому можно было бы определить, насколько вывод модели отличается от желаемого результата, и попытаться минимизировать это отличие.

Воспользуемся одной из распространенных и удобных функций для определения потерь (функция ошибки) - «кросс-энтропией», известной в теории информации:

$$H'_y(y) = -\sum_j y'_i log(y_i)$$

, где y - предсказанное моделью распределение вероятностей, а y'- истинное распределение (вектор с цифровыми метками).

В некотором смысле, кросс-энтропия измеряет близость истинного и оцененного распределения вероятностей.

Определим символьную ссылку на местозаполнитель для правильных ответов:

y_ = tf.placeholder(tf.float32, [None, 10])

Затем реализуем функцию кросс-энтропии по вышеприведенной формуле:

*cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))*

Здесь метод tf.reduce_sum складывает элементы по второму индексу, поскольку указан параметр reduction_indices = [1] (индексы тензоров в данном фреймворке начинаются с нуля), а метод tf.reduce_mean вычисляет среднее значение по всем элементам в партии данных. Но более практично использовать готовый, специально оптимизированный для этой цели метод *tf.nn.softmax_cross_entropy_with_logits*.

Теперь, когда у нас есть функция ошибки, приступим к обучению модели. TensorFlow знает граф наших вычислений, и умеет автоматически использовать алгоритм обратного распространения ошибки (градиентный спуск), эффективно определяя, какие переменные влияют на величину потери (ошибки), которую мы стремимся минимизировать. Он может применить указанный алгоритм оптимизации для изменения переменных и уменьшения потерь:

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

В данном случае в качестве алгоритма оптимизации мы указали градиентный спуск со скоростью обучения 0,5. TensorFlow также предоставляет множество других алгоритмов оптимизации (несколько десятков).

Как уже отмечалось ранее, реальные вычисления производятся в рамках механизма вычислительный сессий, создадим объект сессии:

sess = tf.InteractiveSession()

Прежде чем запустить нашу модель на выполнение, необходимо выполнить операцию инициализации созданных нами переменных:

tf.global_variables_initializer().run()

Теперь можно запустить обучение, пусть число тренировочных шагов будет 1000:

for _ in range(1000):

batch_xs, batch_ys = mnist.train.next_batch(100)

sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

На каждом шаге цикла мы получаем «пакет» ("batch") из ста случайных точек данных из нашей обучающей выборки. Мы запускаем train_step, подавая данные в пакет из соответствующих местозаполнителей.

Использование небольших партий случайных данных называется стохастическим обучением - в данном случае стохастическим градиентным спуском. В идеале, можно было бы использовать все наши данные для каждого этапа обучения, но это слишком накладно в плане вычислений, а результат получается одинаковый.

Оценка работы модели

Насколько хорошо работает наша модель? Для оценки применим несколько вспомогательных функций. Tf.argmax - функция, которая возвращает индекс максимального значения в тензоре вдоль некоторой оси. Например, tf.argmax(y, 1) - это метка, которая, по результату работы модели, наиболее вероятна для каждого входа, а tf.argmax(y_, 1) является правильной меткой. Мы можем использовать tf.equal, чтобы проверить, соответствует ли наше предсказание истине:

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))

В результате мы получим вектор логических значений, например [True, False, True, True]. Чтобы определить финальную оценку, преобразуем логические значения в вещественные, а затем возьмем среднее. Например, [True, False, True, True] станет [1,0,1,1], и после усреднения получим 0.75 (т.е. 3/4):

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

Выведем получившую точность оценок по нашим тестовым данным:

print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))

Результат получился около 92%.

Насколько это хороший результат? На самом деле, результат довольно посредственный. Произошло это по потому, что мы использовали очень простую модель. С небольшими изменениями, легко достичь 97%. Простая модификация модели путем добавление двух сверточных слоев, показала результат 99,2%. Лучшие модели могут получить более 99,7% точности.

Заключение

В данной работе был рассмотрен новый инструмент для проведения распределенных вычислений TensorFlow, который представляет вычисления в виде потока данных и производит оптимизацию всех входящих в систему параметров, используя обобщенный алгоритм обратного распространения ошибки. А также приведен некоторый теоретический минимум по теории искусственных нейронных сетей, включая последние достижения.

Помимо работы с обычными и свёрточными нейронными сетями в TensorFlow поддерживается работа с рекуррентными нейронными сетями, текстовыми данными, в том числе алгоритм word2vec; кроме того, несложным является и написание собственных вычислительных единиц.

Помимо вычислительной составляющей, в данном инструменте поддерживается встроенное построение графиков, гистограмм, трехмерных проекций многомерных тензоров и визуализация графа вычислений, что позволяет сделать описание моделей более наглядным.

TensorFlow позволяет ускорить процесс моделирования вычислительных систем благодаря удобному программному интерфейсу, хорошо оптимизированному коду, простому распараллеливанию, средствам визуализации и поддержке вычислений на графических процессорах.

Список использованных источников

1. *Haykin S*, Нейронные сети: Полный курс. М. : Вильямс, 2008.
2. *LeCun, Yann*. LeNet-5, convolutional neural networks [Электронный ресурс]. URL: http://yann.lecun.com/exdb/lenet/
3. *Fukushima Kunihiko Neocognitron*: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. // Biological Cybernetics 1980, 36 (4)
4. *Борисов Е.* Классификатор на основе многослойной нейронной сети. [Электронный ресурс] URL:http://mechanoid.kiev.ua/neural-net-perceptron-multi-layer-classifier.html
5. *Bouvrie Jake*. Notes on Convolutional Neural Networks. Center for Biological and Computational Learning. Department of Brain and Cognitive Sciences. Massachusetts Institute of Technology Cambridge, MA 02139, 2006, 8p
6. *Stephan Dreiseitl, Lucila Ohno-Machado*. Logistic regression and artificial neural network classification models: a methodology review. Journal of Biomedical Informatics, 2002. Vol. 35, iss. 5–6 (October 2002), p. 352–359.
7. The MNIST database of handwritten digits. [Электронный ресурс] URL: http://yann.lecun.com/exdb/mnist/
8. Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation. DeepLearning 0.1. LISA Lab. [Электронный ресурс] URL: http://deeplearning.net/tutorial/lenet.html
9. Официальный сайт TensorFlow. [Электронный ресурс] URL: https://www.tensorflow.org
10. Международный проект онлайн-книга "Neural Networks and Deep Learning". [Электронный ресурс] URL: http://neuralnetworksanddeeplearning.com/index.html
11. *Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E.* (University of Toronto). ImageNet Classification with Deep Convolutional Neural Networks. Part of: Advances in Neural Information Processing Systems 25 (NIPS 2012), 9p
12. *Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun*. Deep Residual Learning for Image Recognition. Microsoft Research. arXiv:1512.03385v1 [cs.CV] 10 Dec 2015, 12p. [Электронный ресурс] URL: https://arxiv.org/pdf/1512.03385v1.pdf
13. *Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun*. Identity Mappings in Deep Residual Networks. Microsoft Research. arXiv:1603.05027v2 [cs.CV] 12 Apr 2016, p15. https://arxiv.org/pdf/1603.05027v2.pdf
14. *Stephan Dreiseitl, Lucila Ohno-Machado*. Logistic regression and artificial neural network classification models: a methodology review. Available at: http://dx.doi.org/10.1016/S1532-0464(03)00034-0
15. *Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi*. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. Cornell University Library. Submitted on 23 Feb 2016. https://arxiv.org/abs/1602.07261
16. Материалы конференции TensorFlow Dev Summit 2017, February 15th, 2017, Mountain View, CA. [Электронный ресурс] URL: https://events.withgoogle.com/tensorflow-dev-summit/