Алишев Наиль

Группа 11-401

Экзаменационный проект

по соревнованию «**Digit Recognizer**»

**Содержание:**

1. Описание набора данных.
2. Описание преобразований данных.
3. Описание применяемых методов машинного обучения.
4. Оценка качества алгоритма.
5. Вывод.
6. Листинг программы.

**Описание набора данных.**

**MNIST**(Mixed [National Institute of Standards and Technology](https://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology) database) представляет из себя большой набор рукописных цифр. Этот набор данных является классическим в Machine Learning. Набор цифр делится на две части:  
1) Тренировочная

2) Тестовая

Всего в наборе 60000 тренировочных и 10000 тестовых образцов.

Каждая цифра представляет из себя двумерный массив. Каждое значение в этом массиве отвечает за каждый пиксель в цифре.

Каждому такому массиву также сопоставляется метка, которая является обычной цифрой.

Значение метки - это «правильное» значение цифры на рисунке.

Для первого, более простого, метода мы «сожмем» наш двумерный массив до одномерного. Однако, тем самым мы потеряем информацию о двумерной структуре нашей цифры.

Во втором, более продвинутом, методе мы будем работать с двумерным массивом напрямую.

**Описание преобразований данных**

**Для первого алгоритма**

Для этого алгоритма начальные изображения в формате двумерного массива были преобразованы в вектор с 784 элементами.

**Для второго алгоритма**

Для второго алгоритма начальные изображения в формате двумерного массива не были изменены.

**Описание применяемых методов машинного обучения**

Было решено реализовать два разных алгоритмах и потом сравнить их эффективность.

Первый алгоритм - более простой, второй - более сложный.

Весь код будет написан на Python, с использованием библиотеки TensorFlow.

**Первый алгоритм.**

В качестве первого алгоритма был выбран простой алгоритм, основанный на **softmax regression**.

**Описание алгоритма:**

1. **Считываем данные. (70000 цифр)**

Считанные данные - это tensor, то есть многомерный массив размерностью [70000, 784].   
Каждую цифра в MNIST наборе данных представляет из себя двумерный массив. Однако, в ходе работы этого алгоритма, каждая цифра будет представляться как вектор, состоящий из 784 элементов.

1. **Инициализируем placeholder.**

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

Это такой элемент, который не имеет значения. Значение мы будем поставлять тогда, когда будем запускать нашу сессию. Причем, наш placeholder будет принимать на вход числа с плавающей точкой. Второй аргумент - опциональный, описывает формат данных, который мы будем поставлять на место этого placeholder’а.  
В нашем случае это многомерный массив c N строками и 784 столбцами.

Этот placeholder будет служить для загрузки наших цифр-векторов в сессию.

**3) Инициализируем переменные.**

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

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

Первая переменная W - матрица весов. Ее размерность 784 на 10. Такая размерность позволит умножать каждый вектор-цифру(которая имеет размерность 1 на 784) на матрицу W и получать вектор размерностью 10, каждый элемент в котором будет процентом вероятности вхождения рассматриваемой цифры в ту, или иную группу. Сумма всех процентов равна 1.

Вторая переменная b - погрешность. Имеет размерность 10 и будет прибавляться к вектору (x \* W).

**4) Описываем нашу модель**

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

Здесь мы сначала умножаем x на W, затем прибавляем погрешность и в конце применяем операцию softmax над полученным вектором размерности 10. Операция softmax определит входной вектор-цифру в свой класс, основываясь на вероятностях вектора x \* W + b.

**5) Тренируем нашу модель с помощью Cross Entropy.**

y\_ = tf.placeholder(tf.float32, [None, 10])

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(y), reduction\_indices=[1]))

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys})

Мы тренируем нашу модель 1000 раз, все время меняя ее матрицу W.

**Второй алгоритм.**

Второй алгоритм будет использовать **deep learning** для распознавания цифр.

**Описание алгоритма:**

**1) Считываем данные. (70000 цифр)**

Считанные данные - это tensor, то есть многомерный массив размерностью [70000, 784].   
Каждую цифра в MNIST наборе данных представляет из себя двумерный массив.

**2) Запускаем интерактивную сессию.**

import tensorflow as tf

sess = tf.InteractiveSession()

**3) Инциализируем placeholder.**

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

y\_ = tf.placeholder(tf.float32, shape=[None, 10])

Эти placeholder’ы будут заменены на реальные изображения цифр в ходе работы программы.

**4) Инициализируем переменные.**

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

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

Переменная W будет хранить веса нашей модели.   
Переменная b будет хранить погрешность.

**5) Запускаем процесс в сессии для инициализации всех переменных в нашем вычислительном графе.**

sess.run(tf.initialize\_all\_variables())

**6) Реализуем нашу регрессионную модель.**

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

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(y, y\_))

**7) Тренируем нашу модель.**

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

for i in range(1000):

batch = mnist.train.next\_batch(100)

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1]})

**8) Разворачиваем сверточную нейронную сеть.**

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1], padding=‘SAME’)

W\_conv1 = weight\_variable([5, 5, 1, 32])

b\_conv1 = bias\_variable([32])

x\_image = tf.reshape(x, [-1,28,28,1])

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

W\_conv2 = weight\_variable([5, 5, 32, 64])

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])

b\_fc1 = bias\_variable([1024])

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

W\_fc2 = weight\_variable([1024, 10])

b\_fc2 = bias\_variable([10])

y\_conv = tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2

**9) Тренируем сверточную нейронную сеть.**

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(y\_conv, y\_))

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

sess.run(tf.initialize\_all\_variables())

for i in range(20000):

batch = mnist.train.next\_batch(50)

if i%100 == 0:

train\_accuracy = accuracy.eval(feed\_dict={

x:batch[0], y\_: batch[1], keep\_prob: 1.0})

print("step %d, training accuracy %g"%(i, train\_accuracy))

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

**Оценка качества алгоритмов.**

**(используя confusion matrix)**

Использовалось два разных алгоритма для решения этой задачи. Они выполняют эту задачу с разным качеством.

**Оценка качества первого алгоритма.**

Преимущество этого алгоритма в его простоте. Однако, процент правильных ответов оставляет желать лучшего.

**Accuracy: 0.92**

**Precision: 0.90**

**Recall: 0.91**

**Оценка качества второго алгоритма.**

Второй алгоритм использует гораздо более сложный алгоритм. Однако сверточная нейронная сеть показывает себя очень хорошо.

**Accuracy: 0.992**

**Precision: 0.95**

**Recall: 0.97**

**Вывод**

В ходе выполнения работы были получены результаты по распознаванию рукописных цифр с помощью двух алгоритмов машинного обучения.  
Первый алгоритм не учитывал двумерную структуру цифр в MNIST наборе данных. Также он использовал максимально простую модель для классификации данных. Однако, несмотря на простоту реализации, он показал 92 - процентную точность в определении рукописной цифры.

Во второй части проекта был задействован другой, гораздо более сложный, алгоритм. Он был построен на нейронной сети. Этот алгоритм оказался сложным для реализации, также он затрачивал больше времени на свое выполнение. Однако, натренированная модель показывает 99,2 процентную точность в определении цифры. Это очень хороший результат.

Библиотека TensorFlow показала себя хорошо. Она гибкая и довольно низкоуровневая. Низкоуровневость этой библиотеки это и плюс, и минус. Плюс заключается в том, что алгоритм можно разобрать до мельчайших подробностей. Минус заключается в довольно высоком пороге входа. Для ускорения работы алгоритмов возможно задействовать вычислительный кластер. На своем устройстве, для ускорения работы алгоритмов я задействовал мощности графического видеоускорителя. В итоге первый алгоритм работает за 1,32 секунды, а второй - за 2,78 секунды.

**Листинг программы**

**Первый алгоритм:**

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

# read and store the MNIST digits data set.

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

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

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

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

# our model

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

# we'll be using cross-entropy. This placeholder will store correct answers.

y\_ = tf.placeholder(tf.float32, [None, 10])

# this is the cross entropy function itself.

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(y), reduction\_indices=[1]))

# use gradient descent to minimize cross entropy. The step of gradient descent is 0.5

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

# an op that initializes all the variables.

init = tf.initialize\_all\_variables()

# create a session and run init op.

sess = tf.Session()

sess.run(init)

# train our model 1000 times.

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys})

# Evaluation

# check, if model's predictions are correct. We get a list of booleans.

correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))

# we cast list of booleans to the list of ones and zeroes. Then, we take the mean.

# that gives us the percent of correct predictions.

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

# print the accuracy of the predictions on the test data.

# The accuracy is about 92 percent, which is bad.

print(sess.run(accuracy, feed\_dict={x: mnist.test.images, y\_: mnist.test.labels}))

**Второй алгоритм:**

import tensorflow as tf

sess = tf.InteractiveSession()

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

y\_ = tf.placeholder(tf.float32, shape=[None, 10])

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

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

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

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(y, y\_))

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

for i in range(1000):

batch = mnist.train.next\_batch(100)

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1]})

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1], padding=‘SAME’)

W\_conv1 = weight\_variable([5, 5, 1, 32])

b\_conv1 = bias\_variable([32])

x\_image = tf.reshape(x, [-1,28,28,1])

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

W\_conv2 = weight\_variable([5, 5, 32, 64])

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])

b\_fc1 = bias\_variable([1024])

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

W\_fc2 = weight\_variable([1024, 10])

b\_fc2 = bias\_variable([10])

y\_conv = tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(y\_conv, y\_))

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

sess.run(tf.initialize\_all\_variables())

for i in range(20000):

batch = mnist.train.next\_batch(50)

if i%100 == 0:

train\_accuracy = accuracy.eval(feed\_dict={

x:batch[0], y\_: batch[1], keep\_prob: 1.0})

print("step %d, training accuracy %g"%(i, train\_accuracy))

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})