Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:
«Машинное обучение»
По лабораторной работе №2
«Реализация глубокой нейронной сети»

Выполнил: Сенькович Дмитрий Сергеевич
магистрант кафедры информатики
группы №858642

Проверил: Стержанов Максим Валерьевич
доцент, кандидат технических наук

Минск 2019

Оглавление

1 Постановка задачи

Имеется большой и маленький наборы изображений цифр 28x28. Требуется обучить глубокую нейронную сеть на различных объемах данных с различными методами регуляризации модели и динамической скоростью обучения.

2 Решение задачи

2.1 Чтение данных

Для начала приведем листинг кода, используемый для чтения данных. По сути, он представляет из себя код чтения данных из предыдущей лабораторной работы с единственным изменением: теперь проверки на нечитабельные файлы нет, так как они предварительно были удалены:

```
def read_data():


    large_dataset_filenames = {}
    for letter in CLASSES:
        for r, d, f in os.walk(LARGE_DATASET_NAME + letter):
            large_dataset_filenames[letter] = f
    print('Large dataset images filenames has been read')
    small_dataset_filenames = {}
    for letter in CLASSES:
        for r, d, f in os.walk(SMALL_DATASET_NAME + letter):
            small_dataset_filenames[letter] = f
    print('Small dataset images filenames has been read')


    t = TicToc('reading datasets')
    t.tic()
    large_dataset_images = {}
    for letter in CLASSES:
        large_dataset_images[letter] = []
        for filename in large_dataset_filenames[letter]:
```

```
            large_dataset_images[letter].append(io.imread(LARGE_DATASET_NAME +
letter + "/" + filename, as_gray=True).ravel() / 255)

    print('Large dataset images has been read')

    small_dataset_images = {}

    for letter in CLASSES:

        small_dataset_images[letter] = []

        for filename in small_dataset_filenames[letter]:

            small_dataset_images[letter].append(io.imread(SMALL_DATASET_NAME +
letter + "/" + filename, as_gray=True).ravel() / 255)

    print('Small dataset images has been read')

    t.toc()

    print(t.elapsed)


    return large_dataset_images, small_dataset_images
```

2.2 Формирование обучающей, валидационной и тестовой выборок

Реализуем вспомогательные функции, формирующие различные выборки для обучения сети:

```
def build_train_set(train_set_size, large_dataset_images):

    x_train = []

    y_train = []

    sample_size = train_set_size / len(CLASSES)

    train_dataset_indices = {}

    for letter in CLASSES:

        train_dataset_indices[letter] = set()

        while len(train_dataset_indices[letter]) < sample_size:

                            train_dataset_indices[letter].add(random.randint(0,
len(large_dataset_images[letter]) - 1))
```

```
    for i in range(len(CLASSES)):
        letter = CLASSES[i]
        indices = train_dataset_indices[letter]
        for index in indices:
            image = large_dataset_images[letter][index]
            x_train.append(image)
            y_train.append(i)
    x_train = np.array(x_train)
    y_train = np.array(y_train)
    print(x_train.shape)
    print(y_train.shape)
    return x_train, y_train, train_dataset_indices


def build_train_and_validation_sets(train_set_size, validation_set_size, large_dataset_images):
     x_train, y_train, train_dataset_indices = build_train_set(train_set_size, large_dataset_images)
    x_val = []
    y_val = []
    sample_size = validation_set_size / len(CLASSES)
    validation_dataset_indices = {}
    for letter in CLASSES:
        validation_dataset_indices[letter] = set()
        while len(validation_dataset_indices[letter]) < sample_size:
            index = random.randint(0, len(large_dataset_images[letter]) - 1)
            if index not in train_dataset_indices[letter]:
                validation_dataset_indices[letter].add(index)
```

```
    for i in range(len(CLASSES)):
        letter = CLASSES[i]
        indices = validation_dataset_indices[letter]
        for index in indices:
            image = large_dataset_images[letter][index]
            x_val.append(image)
            y_val.append(i)
    x_val = np.array(x_val)
    y_val = np.array(y_val)
    print(x_val.shape)
    print(y_val.shape)
    return x_train, y_train, x_val, y_val


def build_test_set(small_dataset_images):
    x_test = []
    y_test = []
    for i in range(len(CLASSES)):
        letter = CLASSES[i]
        for image in small_dataset_images[letter]:
            x_test.append(image)
            y_test.append(i)
    x_test = np.array(x_test)
    y_test = np.array(y_test)
    print(x_test.shape)
    print(y_test.shape)
    return x_test, y_test
```

Тестовая выборка всегда будет одного размера - все изображения в маленьком датасете. Обучающая выборка будет состоять из неповторяющихся изображений каждой буквы. Валидационная выборка будет сформирована так же, но изображения в ней не будут пересекаться с обучающей выборкой.

2.3 Реализация глубокой нейронной сети

Чтобы получить максимально возможную производительность в нашем случае создадим сеть из пяти полносвязных слоев по 100 нейронов на каждом слое с функцией кусочно-линейной функцией активации ReLU:

```
model = keras.Sequential([

    keras.layers.Dense(100, activation='relu'),

      keras.layers.Dense(100, activation='relu'),

      keras.layers.Dense(100, activation='relu'),

      keras.layers.Dense(100, activation='relu'),

      keras.layers.Dense(100, activation='relu'),

      keras.layers.Dense(10, activation='softmax')

])

model.compile(optimizer='SGD',          loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

print('SGD     accuracy:     %s'     %     train_and_compare_with_test(200000,
large_dataset_images,               small_dataset_images,               model),
file=open("results_no_regularization.txt", "a"))

def      train_and_compare_with_test(train_set_size,      large_dataset_images,
small_dataset_images, model):

     x_train, y_train, large_dataset_indices = build_train_set(train_set_size,
large_dataset_images)

    x_test, y_test = build_test_set(small_dataset_images)


    test_acc = train_and_validate(x_train, y_train, x_test, y_test, model)
```

```
    return test_acc

def train_and_validate(x_train, y_train, x_val, y_val, model):

    t = TicToc('learning')

    t.tic()

    model.fit(x_train, y_train, epochs=EPOCHS_NUMBER)

    test_loss, test_acc = model.evaluate(x_val, y_val, verbose=2)

    print("Accuracy : ", test_acc)

    t.toc()

    print(t.elapsed)

    return test_acc
```

Обучим модель в течение 50 эпох. Получим следующее значение точности на тестовой выборке: 0.94911724, что уже является достаточно высоким показателем. На мой взгляд, объяснить это можно достаточно большим для такой задачи количеством нейронов и слоев.

2.4 Реализация глубокой нейронной сети с L2 регуляризацией

L2-регуляризация в нейронных сетях нужно для той же цели, что и в логистической регрессии: борьба с переобучением. Добавим функцию обучения нейронной сети с L2-регуляризацией и проверки точности предсказания на валидационной выборке. Будем также проверять точность на тестовой выборке:

```
for i in range(-10, -3):

    l = 10**i

    model = keras.Sequential([

                                    keras.layers.Dense(100,    activation='relu',
kernel_regularizer=regularizers.l2(l)),

                                    keras.layers.Dense(100,    activation='relu',
kernel_regularizer=regularizers.l2(l)),

                                    keras.layers.Dense(100,    activation='relu',
kernel_regularizer=regularizers.l2(l)),
```

```
                                    keras.layers.Dense(100,    activation='relu',
kernel_regularizer=regularizers.l2(l)),

                                    keras.layers.Dense(100,    activation='relu',
kernel_regularizer=regularizers.l2(l)),

        keras.layers.Dense(10, activation='softmax')

    ])

         model.compile(optimizer='SGD',  loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

    print('l is %s' % l, file=open("results_l2.txt", "a"))

                        print('SGD     validation     accuracy:     %s'     %
train_and_compare_with_validation(200000, 10000, large_dataset_images, model),
file=open("results_l2.txt", "a"))

    x_test, y_test = build_test_set(small_dataset_images)

    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

    print('SGD test accuracy: %s' % test_acc, file=open("results_l2.txt", "a"))
```

Ниже представлены результаты обучения для каждого из значения параметра регуляризации:

Значение параметра регуляризации	Точность предсказания на валидационной выборке	Точность предсказания на тестовой выборке
10^{-10}	0.8845	0.9508718
10^{-9}	0.8857	0.9502687
10^{-8}	0.8897	0.9482399
10^{-7}	0.8917	0.9493365
10^{-6}	0.8839	0.9521329
10^{-5}	0.8837	0.94736266
10^{-4}	0.8944	0.95202327

10^{-3}	0.8895	0.9497204

Заметим, что точность предсказания на валидационной выборке на порядок ниже, чем на тестовой, что может объясняться более простыми изображениями в тестовой выборке. Также заметим, что результаты не сильно отличаются друг от друга и в целом не сильно отличаются от результата обучения без регуляризации. Это можно объяснить большим количеством данных в обучающей выборке, из-за чего сеть не может переобучиться и регуляризация не дает большого прироста в точности. Результаты могли бы быть более предсказуемыми при использовании меньшего количества изображений для обучений, скажем, 20.000 экземпляров в обучающем сете.

2.5 Реализация глубокой нейронной сети с dropout

Dropout - метод сброса нейронов - является еще одним способом борьбы с переобучением. Идея заключается в “выключении” некоторой случайно выбранной части нейронов при обучении. В таком случае сеть учится не зависеть от каких-то определенных нейронов, что ведет к регуляризации. Реализуем dropout с различными параметрами drop rate (часть нейронов, которая будет отбрасываться при обучении):

```
for rate in [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]:

    model = keras.Sequential([

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dropout(rate),

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dropout(rate),

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dropout(rate),

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dropout(rate),

        keras.layers.Dense(100, activation='relu'),
```

```
        keras.layers.Dropout(rate),

        keras.layers.Dense(10, activation='softmax')

    ])

    print('rate is %s' % rate, file=open("results_dropout.txt", "a"))

         model.compile(optimizer='SGD',  loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

                        print('SGD     validation     accuracy:     %s'     %
train_and_compare_with_validation(200000, 10000, large_dataset_images, model),
file=open("results_dropout.txt", "a"))

    x_test, y_test = build_test_set(small_dataset_images)

    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

     print('SGD test accuracy: %s' % test_acc, file=open("results_dropout.txt",
"a"))
```

Приведем результаты обучения при различных значениях параметра dropout rate:

Значение параметра регуляризации	Точность предсказания на валидационной выборке	Точность предсказания на тестовой выборке
0.01	0.887	0.9485141
0.05	0.8957	0.95388746
0.1	0.8935	0.952736
0.2	0.8862	0.9486238
0.3	0.8795	0.94500494
0.4	0.8602	0.93327117

Здесь, как и с предыдущим методом регуляризации, сильной разницы между точностью модели без регуляризации не видно. Впрочем, видим, что чем выше часть отбрасываемых нейронов, тем тяжелее сети обучиться за

одинаковое количество эпох. Также видим, что наибольшая точность достигнута в случае dropout rate 0.05, поэтому в финальной модели мы будем использовать именно его.

2.6 Реализация глубокой нейронной сети с динамически изменяемой скоростью обучения

Со временем обучение может значительно замедлиться при достижении в ходе обучения некоторой довольно близкой к оптимуму окрестности. Одна из причин может быть слишком высокий коэффициент обучения, который не дает градиентному спуска двигаться в сторону оптимума. В таких случаях было бы хорошо уменьшать коэффициент обучения. Оптимум достичь не удастся, но алгоритм сможет блуждать все ближе и ближе вокруг него. Для этого в Tensorflow и Keras есть параметр модели learning rate decay, который используется для вычисления нового значения коэффициента обучения в ходе обучения:

```
for decay_rate in [0.1, 0.01, 0.001, 0.0001]:

    learning_rate = 0.1

    model = keras.Sequential([

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dense(100, activation='relu'),

        keras.layers.Dense(10, activation='softmax')

    ])

                print('initial learning rate %s' % learning_rate,
file=open("results_learning_rate_decay.txt", "a"))

                  print('learning rate decay %s' % decay_rate,
file=open("results_learning_rate_decay.txt", "a"))

    sgd = keras.optimizers.SGD(lr=learning_rate, decay=decay_rate)
```

```
          model.compile(optimizer=sgd,  loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

                      print('SGD     validation     accuracy:     %s'     %
train_and_compare_with_validation(200000, 10000, large_dataset_images, model),
file=open("results_learning_rate_decay.txt", "a"))

    x_test, y_test = build_test_set(small_dataset_images)

    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

                  print('SGD    test    accuracy:    %s'    %    test_acc,
file=open("results_learning_rate_decay.txt", "a"))
```

Теперь модель будет иметь 2 дополнительных параметра: learning rate - коэффициента обучения в начале обучения, возьмем его 0.1 (значения параметра по умолчанию в Keras для метода оптимизации SGD), learning rate decay - параметр, влияющий на значение коэффициента обучения, будем проверять значения от 0.1 до 0.0001. По умолчанию, в Keras для метода обучения SGD (стохастического градиентного спуска) реализована time-based decay:

```
lr *= (1. / (1. + self.decay * self.iterations))
```

Здесь lr - очередное значение коэффициента обучения при достижении следующей эпохи, decay - значение learning rate decay, a iterations - количество эпох.

Ниже представлены результаты обучения:

Learning rate decay	Точность предсказания на валидационной выборке	Точность предсказания на тестовой выборке
0.1	0.7907	0.8800307
0.01	0.8406	0.91578025
0.001	0.8766	0.9451146
0.0001	0.8833	0.9490076

2.7 Финальная модель

Наша финальная модель будет отличаться от модели с динамическим коэффициентом обучения только добавлением dropout слоев, так как обучать ее мы намерены значительно дольше и немного разным значением learning rate decay (будет отражено ниже в таблице результатов):

```
learning_rate = 0.1

decay_rate = 0.0001

dropout_rate = 0.05

model = keras.Sequential([

    keras.layers.Dense(100, activation='relu'),

    keras.layers.Dropout(dropout_rate),

    keras.layers.Dense(100, activation='relu'),

    keras.layers.Dropout(dropout_rate),

    keras.layers.Dense(100, activation='relu'),

    keras.layers.Dropout(dropout_rate),

    keras.layers.Dense(100, activation='relu'),

    keras.layers.Dropout(dropout_rate),

    keras.layers.Dense(100, activation='relu'),

    keras.layers.Dropout(dropout_rate),

    keras.layers.Dense(10, activation='softmax')

])

print('initial       learning       rate       %s'       %       learning_rate,
file=open("results_final_model.txt", "a"))

print('learning        rate        decay        %s'        %        decay_rate,
file=open("results_final_model.txt", "a"))

print('dropout  rate  %s'  %  dropout_rate,  file=open("results_final_model.txt",
"a"))
```

```
sgd = keras.optimizers.SGD(lr=learning_rate, decay=decay_rate)

model.compile(optimizer=sgd,            loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

print('SGD validation accuracy: %s' % train_and_compare_with_validation(200000,
10000, large_dataset_images, model), file=open("results_final_model.txt", "a"))

x_test, y_test = build_test_set(small_dataset_images)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print('SGD test accuracy: %s' % test_acc, file=open("results_final_model.txt",
"a"))
```

Модель будем обучать в течение 50, 100 и 200 эпох:

Количество эпох	Точность на валидационной выборке	Точность на тестовой выборке
50	0.8918	0.9546551
100	0.897	0.9547648
200	0.896	0.9551486

Максимально полученная точность 0.9551486% правильных классификаций.

Замечание: при learning rate decay 0.0002, количеством эпох 200, momentum = 0.95 и nesterov = False удалось получить точность 0.9568483%.