Лабораторная работа №2. Реализация глубокой нейронной сети

Данные: В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A … J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке: https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz (большой набор данных); https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz (маленький набор данных);

Описание данных на английском языке доступно по ссылке: http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html

Задание 1.

Реализуйте полносвязную нейронную сеть с помощью библиотеки Tensor Flow. В качестве алгоритма оптимизации можно использовать, например, стохастический градиент (Stochastic Gradient Descent, SGD). Определите количество скрытых слоев от 1 до 5, количество нейронов в каждом из слоев до нескольких сотен, а также их функции активации (кусочно-линейная, сигмоидная, гиперболический тангенс и т.д.).

In [2]:

```
import os
import tarfile

data_folder = '../data'

def extract(name):
    path = os.path.join(data_folder, name)

    with tarfile.open(path) as tar:
        tar.extractall(data_folder)
```

In [3]:

```
active_dataset = 'notMNIST_small'
```

In [3]:

```
extract(active_dataset + '.tar.gz')
```

In [4]:

```
import numpy as np
import matplotlib.image as mpimg

def load_data(name, classes, n):
    X = []
    y = []

    path = os.path.join(data_folder, name)

    for letter_path, dir_names, file_names in os.walk(path):
        for file_name in file_names:
            try:
                img_path = os.path.join(letter_path, file_name)
                img = mpimg.imread(img_path)
                img = img.reshape(1, n).T

                X.append(img)

                letter_class = os.path.basename(letter_path)
                index = classes.index(letter_class)

                y.append(index)
            except:
                pass

    m = len(X)
```

```
    X = np.array(X).T.reshape((n, m))
    y = np.array(y).T.reshape((1, m))

    return X, y
```

In [5]:

```
h = 28
w = 28
n = h * w
classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
X, y = load_data(active_dataset, classes, n)
```

In [6]:

```
from sklearn.model_selection import train_test_split

X_train, X_dev, y_train, y_dev = train_test_split(X.T, y.T)
```

In [7]:

```
def dummy_model():
    model = keras.models.Sequential([
        keras.layers.Dense(100, activation='relu', input_dim=n),
        keras.layers.Dense(len(classes), activation='softmax')
    ])

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

    return model
```

In [8]:

```
from tensorflow import keras

def train(model, epochs):
    model.fit(X_train, y_train, epochs=epochs)

    return model.evaluate(X_dev, y_dev)
```

In [9]:

```
epochs = 3

result = train(dummy_model(), epochs)
print(result)
```

```
Train on 14043 samples
Epoch 1/3
14043/14043 [==============================] - 2s 115us/sample - loss: 0.8971 - accuracy: 0.7733
Epoch 2/3
14043/14043 [==============================] - 2s 115us/sample - loss: 0.5009 - accuracy: 0.8692
Epoch 3/3
14043/14043 [==============================] - 2s 126us/sample - loss: 0.4452 - accuracy: 0.8800
4681/4681 [==============================] - 1s 129us/sample - loss: 0.4570 - accuracy: 0.8735
[0.45695628228031704, 0.8735313]
```

Задание 2.

Как улучшилась точность классификатора по сравнению с логистической регрессией?

0.81 vs 0.87

Задание 3.

Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество

Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество классификации?

In [10]:

```
def l2_dropout_model(lmbd, p):
    model = keras.models.Sequential([
        keras.layers.Dense(100, activation='relu', input_dim=n,
                           kernel_regularizer=keras.regularizers.l2(lmbd)),
        keras.layers.Dropout(p),
        keras.layers.Dense(len(classes), activation='softmax')
    ])

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

    return model
```

In [11]:

```
import math

lmbd = math.pow(.3,  1)
p = .1

result = train(l2_dropout_model(lmbd, p), epochs)
print(result)
```

```
Train on 14043 samples
Epoch 1/3
14043/14043 [==============================] - 2s 163us/sample - loss: 11.4490 - accuracy: 0.7629
Epoch 2/3
14043/14043 [==============================] - 3s 214us/sample - loss: 1.1405 - accuracy: 0.8541
Epoch 3/3
14043/14043 [==============================] - 2s 139us/sample - loss: 1.0162 - accuracy: 0.8539
4681/4681 [==============================] - 1s 173us/sample - loss: 1.0012 - accuracy: 0.8479
[1.0011670858374622, 0.84789574]
```

Задание 4.

Воспользуйтесь динамически изменяемой скоростью обучения (learning rate). Наилучшая точность, достигнутая с помощью данной модели составляет 97.1%. Какую точность демонстрирует Ваша реализованная модель?

In [12]:

```
def dynamic_rate_l2_dropout_model(lmbd, p, decay):
    model = keras.models.Sequential([
        keras.layers.Dense(100, activation='relu', input_dim=n,
                           kernel_regularizer=keras.regularizers.l2(lmbd)),
        keras.layers.Dropout(p),
        keras.layers.Dense(len(classes), activation='softmax')
    ])

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

    return model
```

In [16]:

```
decay = .01

result = train(dynamic_rate_l2_dropout_model(lmbd, p, decay), epochs)
print(result)
```

```
Train on 14043 samples
Epoch 1/3
14043/14043 [==============================] - 2s 162us/sample - loss: 18.7172 - accuracy: 0.7318
Epoch 2/3
14043/14043 [==============================] - 3s 187us/sample - loss: 6.0581 - accuracy: 0.8361
```

```
Epoch 3/3
14043/14043 [==============================] - 2s 168us/sample - loss: 3.9292 - accuracy: 0.8424
4681/4681 [==============================] - 1s 186us/sample - loss: 3.3774 - accuracy: 0.8549
[3.377443164211388, 0.85494554]
```