Лабораторная работа №2

Реализация глубокой нейронной сети

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 (https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz) (большой набор данных);
- https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz (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 (http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html)

Задание 1

Реализуйте полносвязную нейронную сеть с помощью библиотеки *TensorFlow*. 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 [1]:

```python
from google.colab import drive

drive.mount('/content/drive', force_remount = True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=
email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2f
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%
2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleap
i.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803
-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=ur
n%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%
2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.co
m%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.re
adonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2'

import sys

sys.path.append(BASE_DIR)

import os

os.chdir(BASE_DIR)
```

In [0]:

```
import pandas as pd

dataframe = pd.read_pickle("./large.pkl")
```

In [4]:

```
dataframe['data'].shape
```

Out[4]:

```
(461946,)
```

In [5]:

```
! pip install tensorflow-gpu --pre --quiet

! pip show tensorflow-gpu
```

```
     |████████████████████████████████| 516.2MB 26kB/s
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: termcolor, tensorflow-estimator, opt-einsum, wrapt, absl-py, proto
buf, tensorboard, astunparse, keras-preprocessing, h5py, numpy, wheel, scip
y, gast, six, google-pasta, grpcio
Required-by:
```

In [0]:

```
import tensorflow as tf
```

In [0]:

```
import numpy as np
```

In [0]:

```
dataframe_test = dataframe.sample(frac = 0.1)

dataframe = dataframe.drop(dataframe_test.index)
```

In [9]:

```
x = np.asarray(list(dataframe['data']))[..., np.newaxis]

x = tf.keras.utils.normalize(x, axis = 1)

x.shape
```

Out[9]:

```
(415751, 28, 28, 1)
```

In [31]:

```
x_test = np.asarray(list(dataframe_test['data']))[..., np.newaxis]

x_test = tf.keras.utils.normalize(x_test, axis = 1)

x_test.shape
```

Out[31]:

```
(46195, 28, 28, 1)
```

In [0]:

```
IMAGE_DIM_0, IMAGE_DIM_1 = x.shape[1], x.shape[2]
```

In [11]:

```
from tensorflow.keras.utils import to_categorical

y = to_categorical(dataframe['label'].astype('category').cat.codes.astype('int32'))

y.shape
```

Out[11]:

```
(415751, 10)
```

In [32]:

```
y_test = to_categorical(dataframe_test['label'].astype('category').cat.codes.astype('int32'

y_test.shape
```

Out[32]:

```
(46195, 10)
```

In [0]:

```
LAYER_WIDTH = 5000
```

In [0]:

```
CLASSES_N = y.shape[1]
```

In [0]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Reshape

model = tf.keras.Sequential()

model.add(Reshape((IMAGE_DIM_0 * IMAGE_DIM_1,), input_shape = (IMAGE_DIM_0, IMAGE_DIM_1, 1)
model.add(Dense(LAYER_WIDTH, activation = 'relu'))
model.add(Dense(LAYER_WIDTH, activation = 'sigmoid'))
model.add(Dense(LAYER_WIDTH, activation = 'tanh'))
model.add(Dense(LAYER_WIDTH, activation = 'elu'))
model.add(Dense(LAYER_WIDTH, activation = 'softmax'))
model.add(Dense(CLASSES_N))
```

In [0]:

```
def cat_cross_from_logits(y_true, y_pred):
    return tf.keras.losses.categorical_crossentropy(y_true, y_pred, from_logits = True)

model.compile(optimizer = 'sgd',
              loss = cat_cross_from_logits,
              metrics = ['categorical_accuracy'])
```

In [16]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 5000)              3925000
_________________________________________________________________
dense_1 (Dense)              (None, 5000)              25005000
_________________________________________________________________
dense_2 (Dense)              (None, 5000)              25005000
_________________________________________________________________
dense_3 (Dense)              (None, 5000)              25005000
_________________________________________________________________
dense_4 (Dense)              (None, 5000)              25005000
_________________________________________________________________
dense_5 (Dense)              (None, 10)                50010
=================================================================
Total params: 103,995,010
Trainable params: 103,995,010
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
VAL_SPLIT_RATE = 0.1
```

In [0]:

```
EPOCHS_N = 10
```

In [19]:

```
model.fit(x = x, y = y, epochs = EPOCHS_N, validation_split = VAL_SPLIT_RATE)
```

```
Epoch 1/10
11693/11693 [==============================] - 103s 9ms/step - loss: 2.2195
- categorical_accuracy: 0.1122 - val_loss: 4.8722 - val_categorical_accurac
y: 0.0000e+00
Epoch 2/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2042
- categorical_accuracy: 0.1130 - val_loss: 5.4886 - val_categorical_accurac
y: 0.0000e+00
Epoch 3/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2025
- categorical_accuracy: 0.1126 - val_loss: 5.8407 - val_categorical_accurac
y: 0.0000e+00
Epoch 4/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2018
- categorical_accuracy: 0.1127 - val_loss: 6.0791 - val_categorical_accurac
y: 0.0000e+00
Epoch 5/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2015
- categorical_accuracy: 0.1123 - val_loss: 6.2565 - val_categorical_accurac
y: 0.0000e+00
Epoch 6/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2013
- categorical_accuracy: 0.1127 - val_loss: 6.3940 - val_categorical_accurac
y: 0.0000e+00
Epoch 7/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2012
- categorical_accuracy: 0.1134 - val_loss: 6.5053 - val_categorical_accurac
y: 0.0000e+00
Epoch 8/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2011
- categorical_accuracy: 0.1130 - val_loss: 6.5967 - val_categorical_accurac
y: 0.0000e+00
Epoch 9/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2010
- categorical_accuracy: 0.1134 - val_loss: 6.6732 - val_categorical_accurac
y: 0.0000e+00
Epoch 10/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2010
- categorical_accuracy: 0.1134 - val_loss: 6.7377 - val_categorical_accurac
y: 0.0000e+00
```

Out[19]:

```
<tensorflow.python.keras.callbacks.History at 0x7f399433fbe0>
```

In [34]:

```
results = model.evaluate(x_test, y_test)

print('Test loss, test accuracy:', results)
```

```
1444/1444 [==============================] - 5s 4ms/step - loss: 2.6571 - ca
tegorical_accuracy: 0.1008
Test loss, test accuracy: [2.657094955444336, 0.10081177949905396]
```

Задание 2

Как улучшилась точность классификатора по сравнению с логистической регрессией?

Стало хуже — на тестовой выборке точность составила 10%. Похоже, что данная модель совершенно не подходит для решения этой задачи.

Задание 3

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

In [0]:

```
REG_RATE = 0.001
```

In [0]:

```
from tensorflow.keras.regularizers import l2

l2_reg = l2(REG_RATE)
```

In [0]:

```
DROPOUT_RATE = 0.2
```

In [0]:

```
from tensorflow.keras.layers import Dropout

dropout_layer = Dropout(DROPOUT_RATE)
```

In [0]:

```
model_2 = tf.keras.Sequential()

model_2.add(Reshape((IMAGE_DIM_0 * IMAGE_DIM_1,), input_shape = (IMAGE_DIM_0, IMAGE_DIM_1,
model_2.add(Dense(LAYER_WIDTH, activation = 'relu', kernel_regularizer = l2_reg))
model_2.add(dropout_layer)
model_2.add(Dense(LAYER_WIDTH, activation = 'sigmoid', kernel_regularizer = l2_reg))
model_2.add(dropout_layer)
model_2.add(Dense(LAYER_WIDTH, activation = 'tanh', kernel_regularizer = l2_reg))
model_2.add(dropout_layer)
model_2.add(Dense(LAYER_WIDTH, activation = 'sigmoid', kernel_regularizer = l2_reg))
model_2.add(dropout_layer)
model_2.add(Dense(LAYER_WIDTH, activation = 'relu', kernel_regularizer = l2_reg))
model_2.add(dropout_layer)
model_2.add(Dense(CLASSES_N))
```

In [0]:

```
model_2.compile(optimizer = 'sgd',
                loss = cat_cross_from_logits,
                metrics = ['categorical_accuracy'])
```

In [26]:

```
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
reshape_1 (Reshape)          (None, 784)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 5000)              3925000   
_________________________________________________________________
dropout (Dropout)            (None, 5000)              0         
_________________________________________________________________
dense_7 (Dense)              (None, 5000)              25005000  
_________________________________________________________________
dense_8 (Dense)              (None, 5000)              25005000  
_________________________________________________________________
dense_9 (Dense)              (None, 5000)              25005000  
_________________________________________________________________
dense_10 (Dense)             (None, 5000)              25005000  
_________________________________________________________________
dense_11 (Dense)             (None, 10)                50010     
=================================================================
Total params: 103,995,010
Trainable params: 103,995,010
Non-trainable params: 0
_________________________________________________________________
```

In [27]:

```
model_2.fit(x = x, y = y, epochs = EPOCHS_N, validation_split = VAL_SPLIT_RATE)
```

```
Epoch 1/10
11693/11693 [==============================] - 210s 18ms/step - loss: 19.270
5 - categorical_accuracy: 0.1154 - val_loss: 19.6030 - val_categorical_accur
acy: 0.0000e+00
Epoch 2/10
11693/11693 [==============================] - 210s 18ms/step - loss: 12.889
9 - categorical_accuracy: 0.1193 - val_loss: 15.0547 - val_categorical_accur
acy: 0.0000e+00
Epoch 3/10
11693/11693 [==============================] - 210s 18ms/step - loss: 8.4027
- categorical_accuracy: 0.3462 - val_loss: 12.9659 - val_categorical_accurac
y: 0.0000e+00
Epoch 4/10
11693/11693 [==============================] - 210s 18ms/step - loss: 5.6802
- categorical_accuracy: 0.4805 - val_loss: 11.1155 - val_categorical_accurac
y: 0.0000e+00
Epoch 5/10
11693/11693 [==============================] - 210s 18ms/step - loss: 4.0491
- categorical_accuracy: 0.5286 - val_loss: 9.5503 - val_categorical_accurac
y: 0.0000e+00
Epoch 6/10
11693/11693 [==============================] - 210s 18ms/step - loss: 3.0297
- categorical_accuracy: 0.5674 - val_loss: 8.9475 - val_categorical_accurac
y: 0.0000e+00
Epoch 7/10
11693/11693 [==============================] - 210s 18ms/step - loss: 2.3880
- categorical_accuracy: 0.5944 - val_loss: 8.4980 - val_categorical_accurac
y: 0.0000e+00
Epoch 8/10
11693/11693 [==============================] - 210s 18ms/step - loss: 1.9772
- categorical_accuracy: 0.6292 - val_loss: 7.8002 - val_categorical_accurac
y: 0.0000e+00
Epoch 9/10
11693/11693 [==============================] - 210s 18ms/step - loss: 1.7196
- categorical_accuracy: 0.6475 - val_loss: 7.6441 - val_categorical_accurac
y: 0.0000e+00
Epoch 10/10
11693/11693 [==============================] - 211s 18ms/step - loss: 1.5578
- categorical_accuracy: 0.6599 - val_loss: 7.5225 - val_categorical_accurac
y: 0.0000e+00
```

Out[27]:

```
<tensorflow.python.keras.callbacks.History at 0x7f399345a9b0>
```

In [36]:

```
results_2 = model_2.evaluate(x_test, y_test)

print('Test loss, test accuracy:', results_2)
```

```
1444/1444 [==============================] - 8s 6ms/step - loss: 1.4656 - ca
tegorical_accuracy: 0.7228
Test loss, test accuracy: [1.4655554294586182, 0.7227838635444641]
```

Регуляризация и сброс нейронов значительно помогли — модель показывает 72% точности на тестовой выборке!

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 [28]:

```
from tensorflow.keras.optimizers import SGD

dyn_lr_sgd = SGD(lr = 0.01, momentum = 0.9)

model_2.compile(optimizer = dyn_lr_sgd,
                loss = cat_cross_from_logits,
                metrics = ['categorical_accuracy'])

model_2.fit(x = x, y = y, epochs = EPOCHS_N, validation_split = VAL_SPLIT_RATE)
```

```
Epoch 1/10
11693/11693 [==============================] - 240s 20ms/step - loss: 1.3780
- categorical_accuracy: 0.6521 - val_loss: 6.3145 - val_categorical_accurac
y: 0.0000e+00
Epoch 2/10
11693/11693 [==============================] - 239s 20ms/step - loss: 1.0968
- categorical_accuracy: 0.7462 - val_loss: 6.1084 - val_categorical_accurac
y: 0.0000e+00
Epoch 3/10
11693/11693 [==============================] - 239s 20ms/step - loss: 1.0805
- categorical_accuracy: 0.7522 - val_loss: 6.2995 - val_categorical_accurac
y: 0.0000e+00
Epoch 4/10
11693/11693 [==============================] - 238s 20ms/step - loss: 1.0759
- categorical_accuracy: 0.7534 - val_loss: 6.1664 - val_categorical_accurac
y: 0.0000e+00
Epoch 5/10
11693/11693 [==============================] - 239s 20ms/step - loss: 1.0707
- categorical_accuracy: 0.7564 - val_loss: 6.4368 - val_categorical_accurac
y: 0.0000e+00
Epoch 6/10
11693/11693 [==============================] - 238s 20ms/step - loss: 1.0186
- categorical_accuracy: 0.7755 - val_loss: 6.6284 - val_categorical_accurac
y: 0.0000e+00
Epoch 7/10
11693/11693 [==============================] - 239s 20ms/step - loss: 0.9949
- categorical_accuracy: 0.7805 - val_loss: 6.6951 - val_categorical_accurac
y: 0.0000e+00
Epoch 8/10
11693/11693 [==============================] - 239s 20ms/step - loss: 0.9902
- categorical_accuracy: 0.7824 - val_loss: 6.1755 - val_categorical_accurac
y: 0.0000e+00
Epoch 9/10
11693/11693 [==============================] - 239s 20ms/step - loss: 0.9873
- categorical_accuracy: 0.7840 - val_loss: 6.3812 - val_categorical_accurac
y: 0.0000e+00
Epoch 10/10
11693/11693 [==============================] - 239s 20ms/step - loss: 0.9844
- categorical_accuracy: 0.7852 - val_loss: 6.3063 - val_categorical_accurac
y: 0.0000e+00
```

Out[28]:

```
<tensorflow.python.keras.callbacks.History at 0x7f39922ee0f0>
```

In [38]:

```
results_3 = model_2.evaluate(x_test, y_test)

print('Test loss, test accuracy:', results_3)
```

```
1444/1444 [==============================] - 8s 6ms/step - loss: 1.4656 - ca
tegorical_accuracy: 0.7228
Test loss, test accuracy: [1.4655554294586182, 0.7227838635444641]
```

Динамически изменяемая скорость обучения не улучшила результат — 72% на тестовой выборке.

Можно сделать вывод, что модель с полносвязными слоями может использоваться для решения задачи распознавания изображений, однако она очевидно не является наилучшей.