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 (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)

In [1]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

from scipy.io import savemat
import numpy, glob, sys, os
from PIL import Image
from scipy.io import loadmat
```

In [2]:

```
def generateDataset(folder, target):
    max_count = 0
    for (root, dirs, files) in os.walk(folder):
        for f in files:
            if f.endswith('.png'):
                max_count += 1
    print('Found %s files' % (max_count,))
    data = numpy.zeros((28,28,max_count))
    labels = numpy.zeros((max_count,))
    count = 0
    for (root, dirs, files) in os.walk(folder):
        for f in files:
            if f.endswith('.png'):
                try:
                    img = Image.open(os.path.join(root,f));
                    data[:,:,count]=numpy.asarray(img)
                    surround_folder = os.path.split(root)[-1]
                    assert len(surround_folder)==1
                    labels[count]=ord(surround_folder)-ord('A')
                    count+=1
                except:
                    pass

    print('Saving to ', target)
    savemat(target,{'images': data[:,:,:count],'labels': labels
```

In [3]:

```
generateDataset("notMNIST_small", "test_dataset.mat")
generateDataset("notMNIST_large", "train_dataset.mat")
```

```
Found 18726 files
Saving to  test_dataset.mat
Found 529119 files
Saving to  train_dataset.mat
```

In [4]:
```
def getDataset(file):
    matfile = loadmat(file)
    x = matfile['images'] / 255
    y = matfile['labels']
    return x.T, y.T

trainX, trainY = getDataset('train_dataset.mat')
print(trainX.shape, trainY.shape)

testX, testY = getDataset('test_dataset.mat')
print(testX.shape, testY.shape)
```

```
(529115, 28, 28) (529115, 1)
(18724, 28, 28) (18724, 1)
```

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 [5]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

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

model.fit(trainX, trainY, epochs=15)
```

```
WARNING:tensorflow:From /Users/alinadolmatovich/anaconda2/envs/ten
sorflow-env/lib/python3.6/site-packages/tensorflow_core/python/ops
/resource_variable_ops.py:1630: calling BaseResourceVariable.__ini
t__ (from tensorflow.python.ops.resource_variable_ops) with constr
aint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Train on 529115 samples
Epoch 1/15
529115/529115 [==============================] - 199s 375us/sample
- loss: 0.4140 - acc: 0.8734
```

```
Epoch 2/15
529115/529115 [==============================] - 192s 362us/sample
- loss: 0.3290 - acc: 0.8979
Epoch 3/15
529115/529115 [==============================] - 190s 360us/sample
- loss: 0.3008 - acc: 0.9061
Epoch 4/15
529115/529115 [==============================] - 185s 349us/sample
- loss: 0.2836 - acc: 0.9110
Epoch 5/15
529115/529115 [==============================] - 178s 337us/sample
- loss: 0.2704 - acc: 0.9148
Epoch 6/15
529115/529115 [==============================] - 179s 338us/sample
- loss: 0.2609 - acc: 0.9177
Epoch 7/15
529115/529115 [==============================] - 174s 328us/sample
- loss: 0.2534 - acc: 0.9199
Epoch 8/15
529115/529115 [==============================] - 181s 342us/sample
- loss: 0.2470 - acc: 0.9218
Epoch 9/15
529115/529115 [==============================] - 168s 318us/sample
- loss: 0.2403 - acc: 0.9236
Epoch 10/15
529115/529115 [==============================] - 191s 362us/sample
- loss: 0.2366 - acc: 0.9248
Epoch 11/15
529115/529115 [==============================] - 202s 382us/sample
- loss: 0.2324 - acc: 0.9259
Epoch 12/15
529115/529115 [==============================] - 179s 337us/sample
- loss: 0.2280 - acc: 0.9273
Epoch 13/15
529115/529115 [==============================] - 187s 353us/sample
- loss: 0.2246 - acc: 0.9287
Epoch 14/15
529115/529115 [==============================] - 188s 356us/sample
- loss: 0.2213 - acc: 0.9293
Epoch 15/15
529115/529115 [==============================] - 181s 342us/sample
- loss: 0.2184 - acc: 0.9304
```

Out[5]: `<tensorflow.python.keras.callbacks.History at 0x13face4a8>`

Задание 2. Как улучшилась точность классификатора по сравнению с логистической регрессией?

In [8]:
```
testLoss, testAcc = model.evaluate(testX,  testY, verbose=2)
print('Точность на проверочных данных:', testAcc)
```

```
18724/18724 - 2s - loss: 0.1378 - acc: 0.9621
Точность на проверочных данных: 0.9621342
```

Задание 3. Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество классификации?

In [9]:
```
onModel = keras.models.Sequential([
yes.Flatten(input_shape=(28, 28)),
yes.Dense(256, kernel_regularizer=keras.regularizers.l2(0.001), act
yes.Dense(128, kernel_regularizer=keras.regularizers.l2(0.001), act
yes.Dense(64, kernel_regularizer=keras.regularizers.l2(0.001), acti
yes.Dense(10, activation='softmax')


onModel.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

onModel.fit(trainX, trainY, epochs=15)
```

```
Train on 529115 samples
Epoch 1/15
529115/529115 [==============================] - 251s 475us/sample
- loss: 0.6194 - acc: 0.8538
Epoch 2/15
529115/529115 [==============================] - 264s 499us/sample
- loss: 0.5291 - acc: 0.8673
Epoch 3/15
529115/529115 [==============================] - 295s 558us/sample
- loss: 0.5174 - acc: 0.8694
Epoch 4/15
529115/529115 [==============================] - 306s 579us/sample
- loss: 0.5108 - acc: 0.8707
Epoch 5/15
529115/529115 [==============================] - 291s 550us/sample
- loss: 0.5062 - acc: 0.8709
Epoch 6/15
529115/529115 [==============================] - 312s 589us/sample
- loss: 0.5040 - acc: 0.8715
Epoch 7/15
529115/529115 [==============================] - 295s 557us/sample
- loss: 0.5038 - acc: 0.8713
Epoch 8/15
529115/529115 [==============================] - 283s 536us/sample
- loss: 0.5027 - acc: 0.8714
Epoch 9/15
```

```
529115/529115 [==============================] - 244s 460us/sample
- loss: 0.5017 - acc: 0.8715
Epoch 10/15
529115/529115 [==============================] - 239s 452us/sample
- loss: 0.4999 - acc: 0.8710
Epoch 11/15
529115/529115 [==============================] - 244s 461us/sample
- loss: 0.4994 - acc: 0.8715
Epoch 12/15
529115/529115 [==============================] - 239s 451us/sample
- loss: 0.4991 - acc: 0.8710
Epoch 13/15
529115/529115 [==============================] - 243s 459us/sample
- loss: 0.4976 - acc: 0.8716
Epoch 14/15
529115/529115 [==============================] - 251s 475us/sample

- loss: 0.4980 - acc: 0.8716
Epoch 15/15
529115/529115 [==============================] - 249s 470us/sample
- loss: 0.4981 - acc: 0.8712
```

Out[9]:
```
<tensorflow.python.keras.callbacks.History at 0x65c8dde10>
```

In [11]:
```
loss, testRegularizationAcc = regularizationModel.evaluate(testX,  te
print('Точность на проверочных данных с регуляризацией:', testRegularizationAcc)
```

```
18724/18724 - 3s - loss: 0.3101 - acc: 0.9301
Точность на проверочных данных с регуляризацией: 0.9300897
```

In [12]:
```
dropoutModel = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

dropoutModel.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

dropoutModel.fit(trainX, trainY, epochs=15)
```

```
Train on 529115 samples
Epoch 1/15
```

```
529115/529115 [==============================] - 284s 536us/sample
- loss: 0.7617 - acc: 0.7801
Epoch 2/15
529115/529115 [==============================] - 284s 537us/sample
- loss: 0.6313 - acc: 0.8222
Epoch 3/15
529115/529115 [==============================] - 280s 529us/sample
- loss: 0.6077 - acc: 0.8285
Epoch 4/15
529115/529115 [==============================] - 280s 529us/sample
- loss: 0.5949 - acc: 0.8326
Epoch 5/15
529115/529115 [==============================] - 287s 542us/sample
- loss: 0.5863 - acc: 0.8340
Epoch 6/15
529115/529115 [==============================] - 279s 527us/sample
- loss: 0.5816 - acc: 0.8358
Epoch 7/15
529115/529115 [==============================] - 306s 579us/sample
- loss: 0.5772 - acc: 0.8367
Epoch 8/15
529115/529115 [==============================] - 284s 537us/sample
- loss: 0.5737 - acc: 0.8382
Epoch 9/15
529115/529115 [==============================] - 314s 593us/sample
- loss: 0.5706 - acc: 0.8389
Epoch 10/15
529115/529115 [==============================] - 365s 691us/sample
- loss: 0.5695 - acc: 0.8391
Epoch 11/15
529115/529115 [==============================] - 381s 720us/sample
- loss: 0.5662 - acc: 0.8403
Epoch 12/15
529115/529115 [==============================] - 435s 822us/sample
- loss: 0.5639 - acc: 0.8414- lo
Epoch 13/15
529115/529115 [==============================] - 406s 767us/sample
- loss: 0.5646 - acc: 0.8411
Epoch 14/15
529115/529115 [==============================] - 325s 615us/sample
- loss: 0.5609 - acc: 0.8420
Epoch 15/15
529115/529115 [==============================] - 344s 650us/sample
- loss: 0.5606 - acc: 0.8418
```

Out[12]: <tensorflow.python.keras.callbacks.History at 0x65c884fd0>

In [13]:

```
1 testDropoutLoss, testDropoutAcc = dropoutModel.evaluate(testX,
2 print('Точность на проверочных данных с dropout\'ом:', testDrop
```

```
18724/18724 - 4s - loss: 0.2280 - acc: 0.9383
Точность на проверочных данных с dropout'ом: 0.93826103
```

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 [14]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

adamOptimizer = keras.optimizers.Adam(learning_rate=0.01)
model.compile(optimizer=adamOptimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(trainX, trainY, epochs=15)
```

```
Train on 529115 samples
Epoch 1/15
529115/529115 [==============================] - 238s 450us/sample
- loss: 0.5622 - acc: 0.8365
Epoch 2/15
529115/529115 [==============================] - 227s 429us/sample
- loss: 0.5079 - acc: 0.8517
Epoch 3/15
529115/529115 [==============================] - 211s 398us/sample
- loss: 0.4919 - acc: 0.8560
Epoch 4/15
529115/529115 [==============================] - 216s 408us/sample
- loss: 0.4838 - acc: 0.8582
Epoch 5/15
529115/529115 [==============================] - 226s 426us/sample
- loss: 0.4795 - acc: 0.8593
Epoch 6/15
529115/529115 [==============================] - 225s 425us/sample
- loss: 0.4743 - acc: 0.8611
Epoch 7/15
529115/529115 [==============================] - 268s 506us/sample
- loss: 0.4696 - acc: 0.8624
Epoch 8/15
529115/529115 [==============================] - 246s 464us/sample
- loss: 0.4654 - acc: 0.8628
Epoch 9/15
529115/529115 [==============================] - 221s 418us/sample
- loss: 0.4641 - acc: 0.8621
Epoch 10/15
529115/529115 [==============================] - 249s 471us/sample
```

```
- loss: 0.4596 - acc: 0.8641
Epoch 11/15
529115/529115 [==============================] - 254s 480us/sample
- loss: 0.4609 - acc: 0.8635
Epoch 12/15
529115/529115 [==============================] - 243s 459us/sample
- loss: 0.4555 - acc: 0.8657
Epoch 13/15
529115/529115 [==============================] - 238s 449us/sample
- loss: 0.4509 - acc: 0.8673
Epoch 14/15
529115/529115 [==============================] - 281s 531us/sample
- loss: 0.4529 - acc: 0.8657
Epoch 15/15
529115/529115 [==============================] - 250s 473us/sample
- loss: 0.4518 - acc: 0.8658
```

Out[14]: `<tensorflow.python.keras.callbacks.History at 0x652009e10>`

In [15]:

```
1 testDropoutLoss, testDropoutAcc = model.evaluate(testX,  testY,
2 print('Точность на проверочных данных с learning rate оптимизац
```

```
18724/18724 - 4s - loss: 0.2816 - acc: 0.9214
Точность на проверочных данных с learning rate оптимизацией: 0.921
38433
```