Лабораторная работа №2. Реализация глубокой нейронной сети

In [18]:

```
from __future__ import absolute_import, division, print_function, unicode_literals

# TensorFlow и tf.keras
!python3 -m pip install keras
import tensorflow as tf
from tensorflow import keras
from keras import regularizers


# Вспомогательные библиотеки
import numpy as np
import matplotlib.pyplot as plt
import pdb
from six.moves import cPickle as pickle
import os
from scipy import ndimage
```

```
Requirement already satisfied: keras in /home/ermolkin/study/mo/prev/l
ib/python3.7/site-packages (2.3.1)
Requirement already satisfied: pyyaml in /home/ermolkin/study/mo/prev/
lib/python3.7/site-packages (from keras) (5.3.1)
Requirement already satisfied: numpy>=1.9.1 in /home/ermolkin/study/m
o/prev/lib/python3.7/site-packages (from keras) (1.18.2)
Requirement already satisfied: h5py in /home/ermolkin/study/mo/prev/li
b/python3.7/site-packages (from keras) (2.10.0)
Requirement already satisfied: six>=1.9.0 in /home/ermolkin/study/mo/p
rev/lib/python3.7/site-packages (from keras) (1.14.0)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /home/erm
olkin/study/mo/prev/lib/python3.7/site-packages (from keras) (1.1.0)
Requirement already satisfied: scipy>=0.14 in /home/ermolkin/study/mo/
prev/lib/python3.7/site-packages (from keras) (1.4.1)
Requirement already satisfied: keras-applications>=1.0.6 in /home/ermo
lkin/study/mo/prev/lib/python3.7/site-packages (from keras) (1.0.8)
```

In [20]:

```
def extract_dataset():
    with open('../data/notMNIST_sanit.pickle', 'rb') as f:
        data = pickle.load(f)
    return data

def image_name(index):
  return chr(ord('A') + index)
```

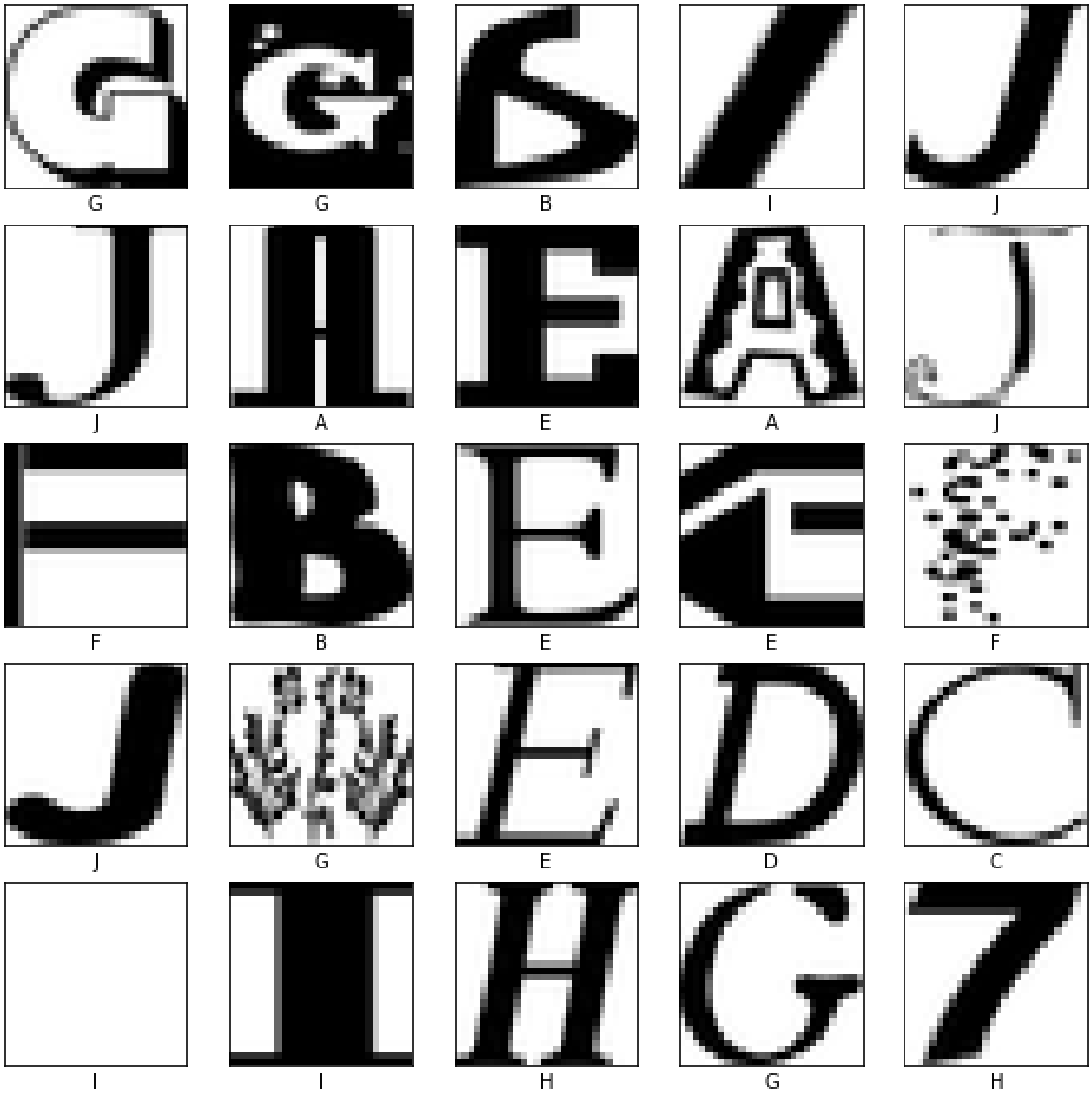
In [22]:

```
dataset = extract_dataset()
train_images = dataset['train_dataset']
train_labels = dataset['train_labels']
valid_images = dataset['valid_dataset']
valid_labels = dataset['valid_labels']
test_images = dataset['test_dataset']
test_labels = dataset['test_labels']
```

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 [24]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(image_name(train_labels[i]))
plt.show()
```



In [26]:

```
# Flatten преобразует формат изображения из двумерного массива (28 на 28 пикселей)
# в одномерный (размерностью 28 * 28 = 784 пикселя)
# Dense - это полносвязные нейронные слои.
baseline_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(20, activation='sigmoid'),
    keras.layers.Dense(20, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

In [28]:

```
baseline_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid'),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

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

baseline_model.summary()

baseline_history = baseline_model.fit(train_images,
                                      train_labels,
                                      epochs=10,
                                      validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = baseline_model.evaluate(test_images,  test_labels, verbose

print('\nТочность на проверочных данных:', test_acc)
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_7 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 100)               78500     
_________________________________________________________________
dense_22 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_23 (Dense)             (None, 10)                1010      
=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 68s 341us/sample -
loss: 0.8498 - accuracy: 0.7832 - sparse_categorical_crossentropy:
0.8498 - val_loss: 0.6804 - val_accuracy: 0.8140 - val_sparse_catego
rical_crossentropy: 0.6804
Epoch 2/10
200000/200000 [==============================] - 66s 331us/sample -
loss: 0.6138 - accuracy: 0.8304 - sparse_categorical_crossentropy:
0.6138 - val_loss: 0.6285 - val_accuracy: 0.8237 - val_sparse_catego
rical_crossentropy: 0.6285
Epoch 3/10
200000/200000 [==============================] - 67s 334us/sample -
loss: 0.5716 - accuracy: 0.8380 - sparse_categorical_crossentropy:
0.5716 - val_loss: 0.5941 - val_accuracy: 0.8294 - val_sparse_catego
rical_crossentropy: 0.5941
Epoch 4/10
200000/200000 [==============================] - 80s 401us/sample -
loss: 0.5418 - accuracy: 0.8431 - sparse_categorical_crossentropy:
0.5418 - val_loss: 0.5701 - val_accuracy: 0.8340 - val_sparse_catego
rical_crossentropy: 0.5701
```

```
Epoch 5/10
200000/200000 [==============================] - 36s 180us/sample -
loss: 0.5195 - accuracy: 0.8475 - sparse_categorical_crossentropy:
0.5195 - val_loss: 0.5503 - val_accuracy: 0.8368 - val_sparse_catego
rical_crossentropy: 0.5503
Epoch 6/10
200000/200000 [==============================] - 37s 185us/sample -
loss: 0.5018 - accuracy: 0.8515 - sparse_categorical_crossentropy:
0.5018 - val_loss: 0.5344 - val_accuracy: 0.8429 - val_sparse_catego
rical_crossentropy: 0.5344
Epoch 7/10
200000/200000 [==============================] - 34s 170us/sample -
loss: 0.4865 - accuracy: 0.8553 - sparse_categorical_crossentropy:
0.4865 - val_loss: 0.5216 - val_accuracy: 0.8458 - val_sparse_catego
rical_crossentropy: 0.5216
Epoch 8/10
200000/200000 [==============================] - 34s 168us/sample -
loss: 0.4738 - accuracy: 0.8584 - sparse_categorical_crossentropy:
0.4738 - val_loss: 0.5114 - val_accuracy: 0.8486 - val_sparse_catego
rical_crossentropy: 0.5114
Epoch 9/10
200000/200000 [==============================] - 28s 140us/sample -
loss: 0.4626 - accuracy: 0.8612 - sparse_categorical_crossentropy:
0.4626 - val_loss: 0.5023 - val_accuracy: 0.8513 - val_sparse_catego
rical_crossentropy: 0.5023
Epoch 10/10
200000/200000 [==============================] - 23s 115us/sample -
loss: 0.4527 - accuracy: 0.8639 - sparse_categorical_crossentropy:
0.4527 - val_loss: 0.4933 - val_accuracy: 0.8539 - val_sparse_catego
rical_crossentropy: 0.4933
8722/8722 - 1s - loss: 0.2838 - accuracy: 0.9150 - sparse_categorica
l_crossentropy: 0.2838

Точность на проверочных данных: 0.9150424
```

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

Логистическая регрессия давала результат в 0.8985 точности, нейронная сеть с тремя слоями выдает результат уже лучше: 0.9144495.

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

In [30]:

```
#3.1 Регуляризация
# https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#add_weight_regula

l2_regularization = 1e-4

l2_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid', kernel_regularizer=regularizers.l
    keras.layers.Dense(100, activation='relu', kernel_regularizer=regularizers.l2(l
    keras.layers.Dense(10, activation='softmax')
])

l2_model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy', 'sparse_categorical_crossentropy'])

l2_model_history = l2_model.fit(train_images,
                                train_labels,
                                epochs=10,
                                validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = l2_model.evaluate(test_images,  test_labels, verbose=2)

print('\nТочность на проверочных данных:', test_acc)
```

```
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 29s 146us/sample - lo
ss: 0.8656 - accuracy: 0.7867 - sparse_categorical_crossentropy: 0.835
5 - val_loss: 0.7207 - val_accuracy: 0.8153 - val_sparse_categorical_c
rossentropy: 0.6899
Epoch 2/10
200000/200000 [==============================] - 29s 147us/sample - lo
ss: 0.6569 - accuracy: 0.8297 - sparse_categorical_crossentropy: 0.625
8 - val_loss: 0.6781 - val_accuracy: 0.8214 - val_sparse_categorical_c
rossentropy: 0.6469
Epoch 3/10
200000/200000 [==============================] - 25s 124us/sample - lo
ss: 0.6189 - accuracy: 0.8367 - sparse_categorical_crossentropy: 0.587
5 - val_loss: 0.6426 - val_accuracy: 0.8263 - val_sparse_categorical_c
rossentropy: 0.6110
Epoch 4/10
200000/200000 [==============================] - 27s 134us/sample - lo
ss: 0.5889 - accuracy: 0.8421 - sparse_categorical_crossentropy: 0.557
0 - val_loss: 0.6175 - val_accuracy: 0.8311 - val_sparse_categorical_c
rossentropy: 0.5855
Epoch 5/10
200000/200000 [==============================] - 29s 145us/sample - lo
ss: 0.5648 - accuracy: 0.8472 - sparse_categorical_crossentropy: 0.532
6 - val_loss: 0.5962 - val_accuracy: 0.8365 - val_sparse_categorical_c
rossentropy: 0.5637
Epoch 6/10
200000/200000 [==============================] - 25s 127us/sample - lo
ss: 0.5452 - accuracy: 0.8512 - sparse_categorical_crossentropy: 0.512
6 - val_loss: 0.5779 - val_accuracy: 0.8399 - val_sparse_categorical_c
rossentropy: 0.5450
Epoch 7/10
200000/200000 [==============================] - 26s 129us/sample - lo
```

```
ss: 0.5287 - accuracy: 0.8545 - sparse_categorical_crossentropy: 0.495
7 - val_loss: 0.5636 - val_accuracy: 0.8434 - val_sparse_categorical_c
rossentropy: 0.5304
Epoch 8/10
200000/200000 [==============================] - 26s 130us/sample - lo
ss: 0.5150 - accuracy: 0.8582 - sparse_categorical_crossentropy: 0.481
6 - val_loss: 0.5531 - val_accuracy: 0.8461 - val_sparse_categorical_c
rossentropy: 0.5195
Epoch 9/10
200000/200000 [==============================] - 27s 133us/sample - lo
ss: 0.5032 - accuracy: 0.8607 - sparse_categorical_crossentropy: 0.469
4 - val_loss: 0.5447 - val_accuracy: 0.8478 - val_sparse_categorical_c
rossentropy: 0.5108
Epoch 10/10
200000/200000 [==============================] - 26s 129us/sample - lo
ss: 0.4929 - accuracy: 0.8633 - sparse_categorical_crossentropy: 0.458
7 - val_loss: 0.5327 - val_accuracy: 0.8518 - val_sparse_categorical_c
rossentropy: 0.4984
8722/8722 - 1s - loss: 0.3238 - accuracy: 0.9152 - sparse_categorical_
crossentropy: 0.2895

Точность на проверочных данных: 0.9151571
```

In [32]:

```
def plot_history(histories, key='binary_crossentropy'):
    plt.figure(figsize=(16,10))

    for name, history in histories:
        val = plt.plot(history.epoch, history.history['val_' + key],
                       '--', label=name.title()+' Val')
        plt.plot(history.epoch, history.history[key], color=val[0].get_color(),
                 label=name.title()+' Train')

    plt.xlabel('Epochs')
    plt.ylabel(key.replace('_',' ').title())
    plt.legend()

    plt.xlim([0,max(history.epoch)])

    plt.show()
```

Регуляризация показала худшие результаты, чем исходная модель (0.9094037 против 0.9144495).

In [34]:

```
# 3.2 метод сброса нейронов

dropout_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

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

dropout_model_history = dropout_model.fit(train_images,
                                train_labels,
                                epochs=10,
                                validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = dropout_model.evaluate(test_images,  test_labels, verbose=

print('\nТочность на проверочных данных:', test_acc)
```

```
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 28s 140us/sample -
loss: 1.4125 - accuracy: 0.5353 - sparse_categorical_crossentropy:
1.4125 - val_loss: 0.8015 - val_accuracy: 0.7839 - val_sparse_catego
rical_crossentropy: 0.8015
Epoch 2/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.9434 - accuracy: 0.7279 - sparse_categorical_crossentropy:
0.9434 - val_loss: 0.6895 - val_accuracy: 0.8017 - val_sparse_catego
rical_crossentropy: 0.6895
Epoch 3/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.8337 - accuracy: 0.7610 - sparse_categorical_crossentropy:
0.8337 - val_loss: 0.6497 - val_accuracy: 0.8078 - val_sparse_catego
rical_crossentropy: 0.6497
Epoch 4/10
200000/200000 [==============================] - 27s 137us/sample -
loss: 0.7810 - accuracy: 0.7752 - sparse_categorical_crossentropy:
0.7810 - val_loss: 0.6248 - val_accuracy: 0.8134 - val_sparse_catego
rical_crossentropy: 0.6248
Epoch 5/10
200000/200000 [==============================] - 27s 134us/sample -
loss: 0.7443 - accuracy: 0.7846 - sparse_categorical_crossentropy:
0.7443 - val_loss: 0.6083 - val_accuracy: 0.8155 - val_sparse_catego
rical_crossentropy: 0.6083
Epoch 6/10
200000/200000 [==============================] - 29s 144us/sample -
loss: 0.7180 - accuracy: 0.7915 - sparse_categorical_crossentropy:
0.7180 - val_loss: 0.5965 - val_accuracy: 0.8186 - val_sparse_catego
rical_crossentropy: 0.5965
Epoch 7/10
200000/200000 [==============================] - 23s 116us/sample -
loss: 0.7000 - accuracy: 0.7959 - sparse_categorical_crossentropy:
```

```
0.7000 - val_loss: 0.5881 - val_accuracy: 0.8211 - val_sparse_catego
rical_crossentropy: 0.5881
Epoch 8/10
200000/200000 [==============================] - 23s 117us/sample -
loss: 0.6848 - accuracy: 0.7998 - sparse_categorical_crossentropy:
0.6848 - val_loss: 0.5796 - val_accuracy: 0.8234 - val_sparse_catego
rical_crossentropy: 0.5796
Epoch 9/10
200000/200000 [==============================] - 24s 118us/sample -
loss: 0.6708 - accuracy: 0.8039 - sparse_categorical_crossentropy:
0.6708 - val_loss: 0.5726 - val_accuracy: 0.8246 - val_sparse_catego
rical_crossentropy: 0.5726
Epoch 10/10
200000/200000 [==============================] - 23s 116us/sample -
loss: 0.6632 - accuracy: 0.8054 - sparse_categorical_crossentropy:
0.6632 - val_loss: 0.5674 - val_accuracy: 0.8260 - val_sparse_catego
rical_crossentropy: 0.5674
8722/8722 - 1s - loss: 0.3568 - accuracy: 0.8934 - sparse_categorica
l_crossentropy: 0.3568

Точность на проверочных данных: 0.8933731
```

Точность на проверочных данных: (0.890367 против 0.9144495 у исходной модели).

In [36]:

```
# 3.3 регуляризация + дропаут

l2_dropout_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid', kernel_regularizer=regularizers.l
    keras.layers.Dropout(0.5),
    keras.layers.Dense(100, activation='relu', kernel_regularizer=regularizers.l2(l
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

l2_dropout_model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy', 'sparse_categorical_crossentropy'])

l2_dropout_model_history = l2_dropout_model.fit(train_images,
                                train_labels,
                                epochs=10,
                                validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = l2_dropout_model.evaluate(test_images,  test_labels, verbo

print('\nТочность на проверочных данных:', test_acc)
```

```
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 27s 135us/sample - lo
ss: 1.4132 - accuracy: 0.5480 - sparse_categorical_crossentropy: 1.383
6 - val_loss: 0.8084 - val_accuracy: 0.7903 - val_sparse_categorical_c
rossentropy: 0.7770
Epoch 2/10
200000/200000 [==============================] - 26s 130us/sample - lo
ss: 0.9587 - accuracy: 0.7299 - sparse_categorical_crossentropy: 0.926
1 - val_loss: 0.7125 - val_accuracy: 0.8040 - val_sparse_categorical_c
rossentropy: 0.6790
Epoch 3/10
200000/200000 [==============================] - 26s 129us/sample - lo
ss: 0.8569 - accuracy: 0.7619 - sparse_categorical_crossentropy: 0.822
5 - val_loss: 0.6756 - val_accuracy: 0.8117 - val_sparse_categorical_c
rossentropy: 0.6405
Epoch 4/10
200000/200000 [==============================] - 26s 129us/sample - lo
ss: 0.8073 - accuracy: 0.7777 - sparse_categorical_crossentropy: 0.771
6 - val_loss: 0.6547 - val_accuracy: 0.8154 - val_sparse_categorical_c
rossentropy: 0.6185
Epoch 5/10
200000/200000 [==============================] - 25s 127us/sample - lo
ss: 0.7746 - accuracy: 0.7860 - sparse_categorical_crossentropy: 0.737
9 - val_loss: 0.6432 - val_accuracy: 0.8178 - val_sparse_categorical_c
rossentropy: 0.6061
Epoch 6/10
200000/200000 [==============================] - 26s 129us/sample - lo
ss: 0.7514 - accuracy: 0.7919 - sparse_categorical_crossentropy: 0.713
9 - val_loss: 0.6334 - val_accuracy: 0.8211 - val_sparse_categorical_c
rossentropy: 0.5955
Epoch 7/10
200000/200000 [==============================] - 26s 130us/sample - lo
ss: 0.7339 - accuracy: 0.7969 - sparse_categorical_crossentropy: 0.695
7 - val_loss: 0.6245 - val_accuracy: 0.8240 - val_sparse_categorical_c
```

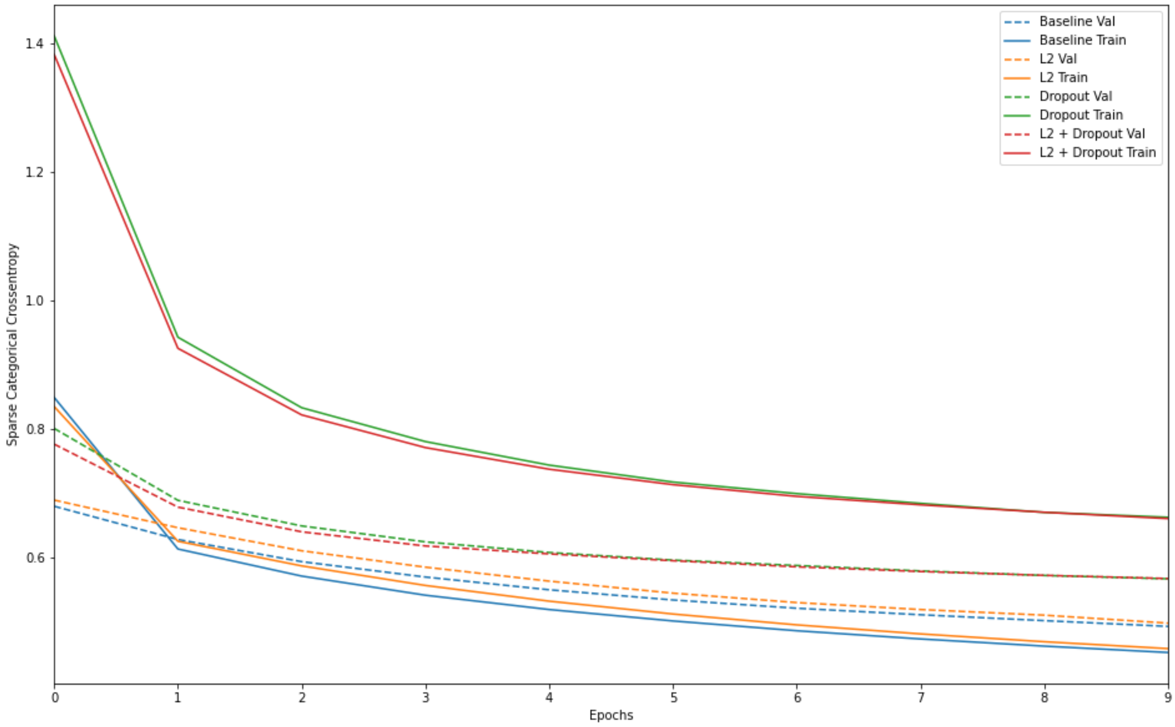
```
rossentropy: 0.5860
Epoch 8/10
200000/200000 [==============================] - 26s 128us/sample - lo
ss: 0.7214 - accuracy: 0.7993 - sparse_categorical_crossentropy: 0.682
6 - val_loss: 0.6178 - val_accuracy: 0.8258 - val_sparse_categorical_c
rossentropy: 0.5787
Epoch 9/10
200000/200000 [==============================] - 25s 127us/sample - lo
ss: 0.7103 - accuracy: 0.8031 - sparse_categorical_crossentropy: 0.671
0 - val_loss: 0.6124 - val_accuracy: 0.8269 - val_sparse_categorical_c
rossentropy: 0.5728
Epoch 10/10
200000/200000 [==============================] - 26s 130us/sample - lo
ss: 0.7010 - accuracy: 0.8048 - sparse_categorical_crossentropy: 0.661
1 - val_loss: 0.6079 - val_accuracy: 0.8277 - val_sparse_categorical_c
rossentropy: 0.5678
8722/8722 - 1s - loss: 0.3973 - accuracy: 0.8934 - sparse_categorical_
crossentropy: 0.3573

Точность на проверочных данных: 0.8933731
```

Точность на проверочных данных: (0.89220184 против 0.9144495 у исходной модели).

In [37]:

```python
plot_history(
    [
        ('baseline', baseline_history),
        ('l2', l2_model_history),
        ('dropout', dropout_model_history),
        ('l2 + dropout', l2_dropout_model_history)
    ],
             key='sparse_categorical_crossentropy')
```



В итоге, дропаут только ухудшил результат модели, регуляризация её не ухудшила, но и лучших результатов не показала. Можно сделать вывод, что так как эти методы применяются для борьбы с переобучением модели, то если они ничего не улучшают, значит модель изначально не была переобучена.

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 [38]:

```
# Adagrad

adagrad_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid'),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

optimizer = keras.optimizers.Adagrad(lr=0.01, epsilon=1e-08, decay=0.0)
adagrad_model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy', 'sparse_categorical_crossentropy'])

adagrad_model.summary()

adagrad_history = adagrad_model.fit(train_images,
                                    train_labels,
                                    epochs=10,
                                    validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = adagrad_model.evaluate(test_images,  test_labels, verbose=

print('\nТочность на проверочных данных:', test_acc)
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_14 (Flatten)         (None, 784)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 100)               78500     
_________________________________________________________________
dense_43 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_44 (Dense)             (None, 10)                1010      
=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 28s 138us/sample - lo
ss: 0.7055 - accuracy: 0.8119 - sparse_categorical_crossentropy: 0.705
5 - val_loss: 0.6278 - val_accuracy: 0.8242 - val_sparse_categorical_c
rossentropy: 0.6278
Epoch 2/10
200000/200000 [==============================] - 25s 127us/sample - lo
ss: 0.5611 - accuracy: 0.8418 - sparse_categorical_crossentropy: 0.561
1 - val_loss: 0.5784 - val_accuracy: 0.8333 - val_sparse_categorical_c
rossentropy: 0.5784
Epoch 3/10
200000/200000 [==============================] - 25s 125us/sample - lo
ss: 0.5230 - accuracy: 0.8495 - sparse_categorical_crossentropy: 0.523
0 - val_loss: 0.5506 - val_accuracy: 0.8391 - val_sparse_categorical_c
rossentropy: 0.5506
Epoch 4/10
```

```
200000/200000 [==============================] - 26s 128us/sample - lo
ss: 0.4985 - accuracy: 0.8545 - sparse_categorical_crossentropy: 0.498
5 - val_loss: 0.5327 - val_accuracy: 0.8439 - val_sparse_categorical_c
rossentropy: 0.5327
Epoch 5/10
200000/200000 [==============================] - 25s 125us/sample - lo
ss: 0.4811 - accuracy: 0.8583 - sparse_categorical_crossentropy: 0.481
1 - val_loss: 0.5170 - val_accuracy: 0.8476 - val_sparse_categorical_c
rossentropy: 0.5170
Epoch 6/10
200000/200000 [==============================] - 25s 126us/sample - lo
ss: 0.4674 - accuracy: 0.8622 - sparse_categorical_crossentropy: 0.467
4 - val_loss: 0.5072 - val_accuracy: 0.8486 - val_sparse_categorical_c
rossentropy: 0.5072
Epoch 7/10
200000/200000 [==============================] - 25s 126us/sample - lo
ss: 0.4565 - accuracy: 0.8647 - sparse_categorical_crossentropy: 0.456
5 - val_loss: 0.4973 - val_accuracy: 0.8528 - val_sparse_categorical_c
rossentropy: 0.4973
Epoch 8/10
200000/200000 [==============================] - 25s 125us/sample - lo
ss: 0.4472 - accuracy: 0.8671 - sparse_categorical_crossentropy: 0.447
2 - val_loss: 0.4888 - val_accuracy: 0.8549 - val_sparse_categorical_c
rossentropy: 0.4888
Epoch 9/10
200000/200000 [==============================] - 25s 126us/sample - lo
ss: 0.4392 - accuracy: 0.8693 - sparse_categorical_crossentropy: 0.439
2 - val_loss: 0.4837 - val_accuracy: 0.8564 - val_sparse_categorical_c
rossentropy: 0.4837
Epoch 10/10
200000/200000 [==============================] - 25s 125us/sample - lo
ss: 0.4321 - accuracy: 0.8713 - sparse_categorical_crossentropy: 0.432
1 - val_loss: 0.4765 - val_accuracy: 0.8577 - val_sparse_categorical_c
rossentropy: 0.4765
8722/8722 - 1s - loss: 0.2712 - accuracy: 0.9199 - sparse_categorical_
crossentropy: 0.2712

Точность на проверочных данных: 0.91985786
```

In [39]:

```
# Adadelta

# Adadelta - это расширение Adagrad,
# которое стремится уменьшить свою агрессивную,
# монотонно уменьшающуюся скорость обучения.

adadelta_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid'),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

optimizer = keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=1e-08, decay=0.0)
adadelta_model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy', 'sparse_categorical_crossentropy'])

adadelta_model.summary()

adadelta_history = adadelta_model.fit(train_images,
                                      train_labels,
                                      epochs=10,
                                      validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = adadelta_model.evaluate(test_images,  test_labels, verbose

print('\nТочность на проверочных данных:', test_acc)
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_15 (Flatten)         (None, 784)               0         
_________________________________________________________________
dense_45 (Dense)             (None, 100)               78500     
_________________________________________________________________
dense_46 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_47 (Dense)             (None, 10)                1010      
=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.5920 - accuracy: 0.8335 - sparse_categorical_crossentropy:
0.5920 - val_loss: 0.5313 - val_accuracy: 0.8425 - val_sparse_catego
rical_crossentropy: 0.5313
Epoch 2/10
200000/200000 [==============================] - 27s 133us/sample -
loss: 0.4633 - accuracy: 0.8609 - sparse_categorical_crossentropy:
0.4633 - val_loss: 0.4800 - val_accuracy: 0.8576 - val_sparse_catego
rical_crossentropy: 0.4800
Epoch 3/10
200000/200000 [==============================] - 27s 133us/sample -
```

```
loss: 0.4196 - accuracy: 0.8731 - sparse_categorical_crossentropy:
0.4196 - val_loss: 0.4520 - val_accuracy: 0.8636 - val_sparse_catego
rical_crossentropy: 0.4520
Epoch 4/10
200000/200000 [==============================] - 26s 131us/sample -
loss: 0.3912 - accuracy: 0.8810 - sparse_categorical_crossentropy:
0.3912 - val_loss: 0.4384 - val_accuracy: 0.8687 - val_sparse_catego
rical_crossentropy: 0.4384
Epoch 5/10
200000/200000 [==============================] - 27s 136us/sample -
loss: 0.3708 - accuracy: 0.8869 - sparse_categorical_crossentropy:
0.3708 - val_loss: 0.4303 - val_accuracy: 0.8703 - val_sparse_catego
rical_crossentropy: 0.4303
Epoch 6/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.3540 - accuracy: 0.8917 - sparse_categorical_crossentropy:
0.3540 - val_loss: 0.4190 - val_accuracy: 0.8762 - val_sparse_catego
rical_crossentropy: 0.4190
Epoch 7/10
200000/200000 [==============================] - 26s 132us/sample -
loss: 0.3406 - accuracy: 0.8960 - sparse_categorical_crossentropy:
0.3406 - val_loss: 0.4184 - val_accuracy: 0.8761 - val_sparse_catego
rical_crossentropy: 0.4184
Epoch 8/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.3294 - accuracy: 0.8997 - sparse_categorical_crossentropy:
0.3294 - val_loss: 0.4220 - val_accuracy: 0.8755 - val_sparse_catego
rical_crossentropy: 0.4220
Epoch 9/10
200000/200000 [==============================] - 27s 133us/sample -
loss: 0.3198 - accuracy: 0.9023 - sparse_categorical_crossentropy:
0.3198 - val_loss: 0.4090 - val_accuracy: 0.8804 - val_sparse_catego
rical_crossentropy: 0.4090
Epoch 10/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.3112 - accuracy: 0.9050 - sparse_categorical_crossentropy:
0.3112 - val_loss: 0.4173 - val_accuracy: 0.8796 - val_sparse_catego
rical_crossentropy: 0.4173
8722/8722 - 1s - loss: 0.2112 - accuracy: 0.9351 - sparse_categorica
l_crossentropy: 0.2112

Точность на проверочных данных: 0.93510664
```

In [ ]:

```

```

In [40]:

```
# RMSprop

# RMSprop очень просто настраивает метод Адаграда,
# пытаясь уменьшить его агрессивное, монотонно убывающее обучение.

rms_prop_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid'),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
rms_prop_model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy', 'sparse_categorical_crossentropy'])

rms_prop_model.summary()

rms_prop_history = rms_prop_model.fit(train_images,
                                      train_labels,
                                      epochs=10,
                                      validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = rms_prop_model.evaluate(test_images,  test_labels, verbose

print('\nТочность на проверочных данных:', test_acc)
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_16 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_48 (Dense)             (None, 100)               78500
_________________________________________________________________
dense_49 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_50 (Dense)             (None, 10)                1010
=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 28s 140us/sample -
loss: 0.5174 - accuracy: 0.8470 - sparse_categorical_crossentropy:
0.5174 - val_loss: 0.4721 - val_accuracy: 0.8589 - val_sparse_catego
rical_crossentropy: 0.4721
Epoch 2/10
200000/200000 [==============================] - 27s 133us/sample -
loss: 0.4083 - accuracy: 0.8751 - sparse_categorical_crossentropy:
0.4083 - val_loss: 0.4405 - val_accuracy: 0.8673 - val_sparse_catego
rical_crossentropy: 0.4405
Epoch 3/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.3771 - accuracy: 0.8848 - sparse_categorical_crossentropy:
```

```
0.3771 - val_loss: 0.4405 - val_accuracy: 0.8709 - val_sparse_catego
rical_crossentropy: 0.4405
Epoch 4/10
200000/200000 [==============================] - 27s 137us/sample -
loss: 0.3601 - accuracy: 0.8907 - sparse_categorical_crossentropy:
0.3601 - val_loss: 0.4446 - val_accuracy: 0.8709 - val_sparse_catego
rical_crossentropy: 0.4446
Epoch 5/10
200000/200000 [==============================] - 27s 136us/sample -
loss: 0.3499 - accuracy: 0.8934 - sparse_categorical_crossentropy:
0.3499 - val_loss: 0.4419 - val_accuracy: 0.8698 - val_sparse_catego
rical_crossentropy: 0.4419
Epoch 6/10
200000/200000 [==============================] - 27s 136us/sample -
loss: 0.3428 - accuracy: 0.8950 - sparse_categorical_crossentropy:
0.3428 - val_loss: 0.4422 - val_accuracy: 0.8718 - val_sparse_catego
rical_crossentropy: 0.4422
Epoch 7/10
200000/200000 [==============================] - 27s 133us/sample -
loss: 0.3356 - accuracy: 0.8971 - sparse_categorical_crossentropy:
0.3356 - val_loss: 0.4612 - val_accuracy: 0.8661 - val_sparse_catego
rical_crossentropy: 0.4612
Epoch 8/10
200000/200000 [==============================] - 27s 135us/sample -
loss: 0.3324 - accuracy: 0.8986 - sparse_categorical_crossentropy:
0.3324 - val_loss: 0.4667 - val_accuracy: 0.8700 - val_sparse_catego
rical_crossentropy: 0.4667
Epoch 9/10
200000/200000 [==============================] - 27s 133us/sample -
loss: 0.3289 - accuracy: 0.9002 - sparse_categorical_crossentropy:
0.3289 - val_loss: 0.4693 - val_accuracy: 0.8717 - val_sparse_catego
rical_crossentropy: 0.4693
Epoch 10/10
200000/200000 [==============================] - 28s 138us/sample -
loss: 0.3251 - accuracy: 0.9016 - sparse_categorical_crossentropy:
0.3250 - val_loss: 0.5169 - val_accuracy: 0.8695 - val_sparse_catego
rical_crossentropy: 0.5169
8722/8722 - 1s - loss: 0.2558 - accuracy: 0.9280 - sparse_categorica
l_crossentropy: 0.2558

Точность на проверочных данных: 0.9279982
```

In [41]:

```
# Adam

# Adam - это обновление оптимизатора RMSProp,
# похожее на RMSprop с динамикой.

adam_model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='sigmoid'),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08
adam_model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy', 'sparse_categorical_crossentropy'])

adam_model.summary()

adam_history = adam_model.fit(train_images,
                                      train_labels,
                                      epochs=10,
                                      validation_data=(valid_images, valid_labels))

test_loss, test_acc, _ = adam_model.evaluate(test_images,  test_labels, verbose=2)

print('\nТочность на проверочных данных:', test_acc)
```

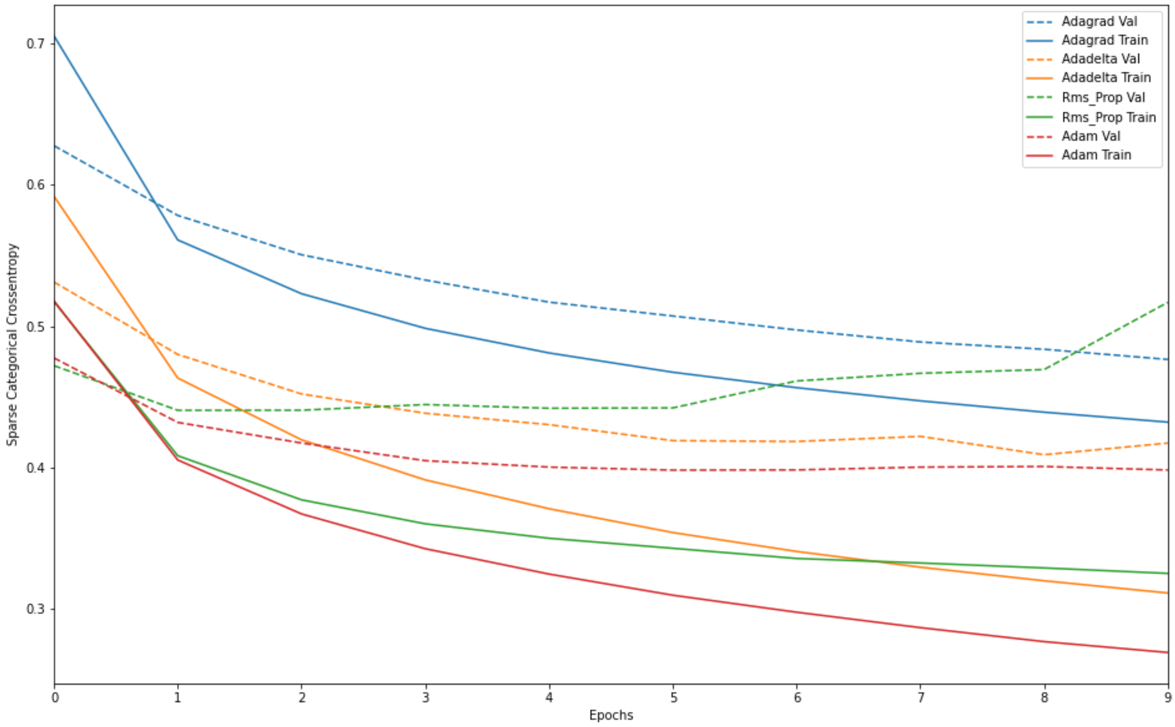
```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_17 (Flatten)         (None, 784)               0         
_________________________________________________________________
dense_51 (Dense)             (None, 100)               78500     
_________________________________________________________________
dense_52 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_53 (Dense)             (None, 10)                1010      
=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
_________________________________________________________________
Train on 200000 samples, validate on 16911 samples
Epoch 1/10
200000/200000 [==============================] - 26s 128us/sample -
loss: 0.5179 - accuracy: 0.8463 - sparse_categorical_crossentropy:
0.5179 - val_loss: 0.4775 - val_accuracy: 0.8578 - val_sparse_catego
rical_crossentropy: 0.4775
Epoch 2/10
200000/200000 [==============================] - 24s 122us/sample -
loss: 0.4053 - accuracy: 0.8762 - sparse_categorical_crossentropy:
0.4053 - val_loss: 0.4319 - val_accuracy: 0.8715 - val_sparse_catego
rical_crossentropy: 0.4319
Epoch 3/10
200000/200000 [==============================] - 25s 126us/sample -
loss: 0.3671 - accuracy: 0.8874 - sparse_categorical_crossentropy:
```

```
0.3671 - val_loss: 0.4173 - val_accuracy: 0.8730 - val_sparse_catego
rical_crossentropy: 0.4173
Epoch 4/10
200000/200000 [==============================] - 25s 123us/sample -
loss: 0.3425 - accuracy: 0.8950 - sparse_categorical_crossentropy:
0.3425 - val_loss: 0.4048 - val_accuracy: 0.8793 - val_sparse_catego
rical_crossentropy: 0.4048
Epoch 5/10
200000/200000 [==============================] - 25s 124us/sample -
loss: 0.3246 - accuracy: 0.8998 - sparse_categorical_crossentropy:
0.3246 - val_loss: 0.4003 - val_accuracy: 0.8798 - val_sparse_catego
rical_crossentropy: 0.4003
Epoch 6/10
200000/200000 [==============================] - 25s 124us/sample -
loss: 0.3096 - accuracy: 0.9044 - sparse_categorical_crossentropy:
0.3096 - val_loss: 0.3982 - val_accuracy: 0.8824 - val_sparse_catego
rical_crossentropy: 0.3982
Epoch 7/10
200000/200000 [==============================] - 25s 123us/sample -
loss: 0.2976 - accuracy: 0.9080 - sparse_categorical_crossentropy:
0.2976 - val_loss: 0.3983 - val_accuracy: 0.8832 - val_sparse_catego
rical_crossentropy: 0.3983
Epoch 8/10
200000/200000 [==============================] - 25s 126us/sample -
loss: 0.2866 - accuracy: 0.9110 - sparse_categorical_crossentropy:
0.2866 - val_loss: 0.4003 - val_accuracy: 0.8839 - val_sparse_catego
rical_crossentropy: 0.4003
Epoch 9/10
200000/200000 [==============================] - 27s 137us/sample -
loss: 0.2767 - accuracy: 0.9140 - sparse_categorical_crossentropy:
0.2767 - val_loss: 0.4007 - val_accuracy: 0.8808 - val_sparse_catego
rical_crossentropy: 0.4007
Epoch 10/10
200000/200000 [==============================] - 29s 147us/sample -
loss: 0.2691 - accuracy: 0.9160 - sparse_categorical_crossentropy:
0.2691 - val_loss: 0.3982 - val_accuracy: 0.8840 - val_sparse_catego
rical_crossentropy: 0.3982
8722/8722 - 1s - loss: 0.2054 - accuracy: 0.9393 - sparse_categorica
l_crossentropy: 0.2054

Точность на проверочных данных: 0.93934876
```

In [42]:

```
plot_history(
    [
        ('adagrad', adagrad_history),
        ('adadelta', adadelta_history),
        ('rms_prop', rms_prop_history),
        ('adam', adam_history)
    ],
    key='sparse_categorical_crossentropy')
```



In [ ]:

```

```