Введение в нейронные сети

Урок 2. Keras

Сдавать через Гитхаб или Гугл Колаб

1. Попробуйте обучить, нейронную сеть на Keras (рассмотренную на уроке) на датасете MNIST с другими параметрами. Напишите в комментарии к уроку:
   --- Какого результата вы добились от нейросети?
   --- Что помогло вам улучшить её точность?
2. (по желанию) Нарисуйте цифру от руки, преобразуйте как надо и подайте на вход вашей обученной нейронности, оцените точность предсказания.
3. (по желанию) Попробуйте применить нейронный классификатор от Sklearn на датасет MNIST.

Задания и ответы:

1.

Попробуйте обучить, нейронную сеть на Keras (рассмотренную на уроке) на датасете MNIST с другими параметрами.

Эксперименты:

тест 1

```
[10] model_1 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(50),
           Dense(1)
           ] )
```

```
[11] model_1.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])
```

```
%%time

history = model_1.fit(x_train, y_train, batch_size=32, epochs=5)
```

Получили плохой результат

```
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 5.3095e-07 - accuracy: 0.0441
```

тест 1_2

```
[13] # приведём все числа в диапазон от 0 до 1
     # т.е. нормализуем данные
     x_train = x_train / 255
     x_test = x_test / 255
```

```
[14] model_1_2 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(50),
           Dense(1)
           ] )
```

```
[15] model_1_2.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
[ ] %%time

    model_1_2.fit(x_train, y_train, batch_size=32, epochs=5)
```

Получили немного лучший, но всё равно плохой результат

```
Epoch 5/5
1875/1875 [==============================] - 7s 3ms/step - loss: 5.3095e-07 - accuracy: 0.0988
```

тест 1_3

```
[17] # закодируем все цифрры в десятипозиционный код,
     # например 1 = 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, а 9 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
     y_train_cat = keras.utils.to_categorical(y_train, 10)
     y_test_cat = keras.utils.to_categorical(y_test, 10)
```

```
[18] model_1_3 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(50),
           Dense(10)
           ] )
```

```
[19] model_1_3.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
%%time

history = model_1_3.fit(x_train, y_train_cat, batch_size=32, epochs=5)
```

Получили ещё немного лучший, но всё ещё плохой результат

```
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 8.5137 - accuracy: 0.1462
```

тест 1_4

```
[21] # поработаем с функцией активации на выходном слое (принадлежность только одной цифре!)
     model_1_4 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(50),
           Dense(10, activation = 'softmax')
           ] )
```

```
[22] model_1_4.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
```

```
%%time

history = model_1_4.fit(x_train, y_train_cat, batch_size=32, epochs=5)
```

А вот результата уже радует

```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2702 - accuracy: 0.9247
```

тест 2

```
[24] # поработаем с функцией активации по слойно
     model_2 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(50,activation = 'relu'),
           Dense(10, activation = 'softmax')
           ] )
```

```
[25] model_2.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
```

Результат ещё больше порадовал

```
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0813 - accuracy: 0.9746
```

Тест 3_1

```
[34] model_3_1 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(50,activation = 'relu'),
           Dense(10, activation = 'softmax')
           ] )
```

```
[35] # заменим оптимизатор
     model_3_1.compile(optimizer='RMSprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
[36] %%time

     history = model_3_1.fit(x_train, y_train_cat, batch_size=32, epochs=5)
```

Замена оптимизатора на RMSprop не улучшила результат, оставляем adam

```
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0953 - accuracy: 0.9721
```

Тест 3_2

```
[37] # немного поменяем модель, добавим ещё один внутренний слой
     model_3_2 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(100,activation = 'relu'),
           Dense(50,activation = 'relu'),
           Dense(10, activation = 'softmax')
           ] )
```

```
[38] model_3_2.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

Добавление ещё одного внутреннего слоя улучшило показатели работы модели

```
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0475 - accuracy: 0.9848
CPU times: user 43.1 s, sys: 2.96 s, total: 46 s
Wall time: 38.8 s
```

Тест 3_3

```
[41] model_3_3 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(100,activation = 'relu'),
           Dense(50,activation = 'relu'),
           Dense(10, activation = 'softmax')
           ] )
```

```
[46] # поменяем loss-функцию (функцию потерь) на низкочувствительную к выбросам
     model_3_3.compile(optimizer='adam',
                  loss='categorical_hinge',
                  metrics=['accuracy'])
```

```
[48] %%time

     history = model_3_3.fit(x_train, y_train_cat, batch_size=32, epochs=5)
```

Результат остался таким же отличным, наверное, у нас нет в данных существенных выбросов

```
Epoch 5/5
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0380 - accuracy: 0.9811
```

Тест 3_4

```
[49] model_3_4 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(100,activation = 'relu'),
           Dense(50,activation = 'relu'),
           Dense(10, activation = 'softmax')
           ] )
```

```
[50] model_3_4.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
[51] # увеличим количество эпох в 2 раза
     %%time

     history = model_3_4.fit(x_train, y_train_cat, batch_size=32, epochs=10)
```

Результат ещё немного улучшился, но время обучение увеличилось

```
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0217 - accuracy: 0.9931
CPU times: user 1min 33s, sys: 5.01 s, total: 1min 38s
Wall time: 1min 41s
```

Тест 3_5

```
[52] model_3_5 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(100,activation = 'relu'),
           Dense(50,activation = 'relu'),
           Dense(10, activation = 'softmax')
           ] )
```

```
[53] model_3_5.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
[55] # увеличим размер батча тоже в 4 раза
     %%time

     history = model_3_5.fit(x_train, y_train_cat, batch_size=128, epochs=10)
```

Это самый лучший результат за самое быстрое время обучения!

```
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0091 - accuracy: 0.9968
CPU times: user 31.8 s, sys: 1.8 s, total: 33.5 s
Wall time: 28.9 s
```

<u>Сравним работу model_2 (Тест 2) и model_3_5 (Тест 3_5)</u>

```
[29] # посмотрим, не произошло ли переобучения? Оценим работу нашей модели на тестовых данных!
     model_2.evaluate(x_test, y_test_cat)

     313/313 [==============================] - 2s 5ms/step - loss: 0.1107 - accuracy: 0.9669
     [0.11070998758077621, 0.9668999910354614]
```

```
[57] # посмотрим, не произошло ли переобучения? Оценим работу нашей модели на тестовых данных!
     model_3_5.evaluate(x_test, y_test_cat)

     313/313 [==============================] - 1s 3ms/step - loss: 0.1058 - accuracy: 0.9785
     [0.1058313250541687, 0.9785000085830688]
```

```
# можно обучать сразу с оценкой работы нашей модели
%%time

history = model_2.fit(x_train, y_train_cat, batch_size=32, epochs=5, validation_data =(x_test, y_test_cat) )
```

```
Epoch 1/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0703 - accuracy: 0.9786 - val_loss: 0.0932 - val_accuracy: 0.9704
Epoch 2/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0606 - accuracy: 0.9808 - val_loss: 0.0926 - val_accuracy: 0.9715
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0542 - accuracy: 0.9833 - val_loss: 0.0910 - val_accuracy: 0.9728
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0473 - accuracy: 0.9852 - val_loss: 0.1061 - val_accuracy: 0.9693
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0420 - accuracy: 0.9867 - val_loss: 0.0961 - val_accuracy: 0.9717
CPU times: user 33.4 s, sys: 1.67 s, total: 35.1 s
Wall time: 42 s
```

```
[59] # можно обучать сразу с оценкой работы нашей модели
     %%time

     history = model_3_5.fit(x_train, y_train_cat, batch_size=128, epochs=10, validation_data =(x_test, y_test_cat) )

     Epoch 1/10
     469/469 [==============================] - 3s 6ms/step - loss: 0.0043 - accuracy: 0.9988 - val_loss: 0.0968 - val_accuracy: 0.9812
     Epoch 2/10
     469/469 [==============================] - 3s 7ms/step - loss: 9.8434e-04 - accuracy: 0.9999 - val_loss: 0.0944 - val_accuracy: 0.9809
     Epoch 3/10
     469/469 [==============================] - 3s 6ms/step - loss: 5.3887e-04 - accuracy: 1.0000 - val_loss: 0.0957 - val_accuracy: 0.9814
     Epoch 4/10
     469/469 [==============================] - 3s 6ms/step - loss: 4.0491e-04 - accuracy: 1.0000 - val_loss: 0.0962 - val_accuracy: 0.9816
     Epoch 5/10
     469/469 [==============================] - 5s 10ms/step - loss: 3.2619e-04 - accuracy: 1.0000 - val_loss: 0.0980 - val_accuracy: 0.9818
     Epoch 6/10
     469/469 [==============================] - 4s 8ms/step - loss: 2.6447e-04 - accuracy: 1.0000 - val_loss: 0.0989 - val_accuracy: 0.9819
     Epoch 7/10
     469/469 [==============================] - 3s 6ms/step - loss: 2.2152e-04 - accuracy: 1.0000 - val_loss: 0.0990 - val_accuracy: 0.9817
     Epoch 8/10
     469/469 [==============================] - 3s 5ms/step - loss: 1.8873e-04 - accuracy: 1.0000 - val_loss: 0.1002 - val_accuracy: 0.9821
     Epoch 9/10
     469/469 [==============================] - 2s 5ms/step - loss: 1.5930e-04 - accuracy: 1.0000 - val_loss: 0.1010 - val_accuracy: 0.9819
     Epoch 10/10
     469/469 [==============================] - 3s 7ms/step - loss: 1.3382e-04 - accuracy: 1.0000 - val_loss: 0.1023 - val_accuracy: 0.9824
     CPU times: user 34 s, sys: 1.84 s, total: 35.9 s
     Wall time: 41.5 s
```

Это просто супер результат!

```
loss: 1.3382e-04 - accuracy: 1.0000 - val_loss: 0.1023 - val_accuracy: 0.9824
```

Напишите в комментарии к уроку:
--- Какого результата вы добились от нейросети?

Test_3_5

loss: 0.0091 - accuracy: 0.9968

```
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0091 - accuracy: 0.9968
CPU times: user 31.8 s, sys: 1.8 s, total: 33.5 s
Wall time: 28.9 s
```

--- Что помогло вам улучшить её точность?

Помогло улучшить точность предсказаний:

- нормализация исходных данных
- кодирование в позиционный код результатов работы модели для однозначного предсказания, потому что не должно быть предсказано, что данная цифра немного 3, а ещё похожа на 5 ;-)
- функция активации softmax на последнем (выходном) слое помогает сделать более точную многоклассовую классификацию
- функции активации на внутренних (скрытых) слоях
- добавление внутреннего (скрытого) слоя
- разумное увеличение количества эпох улучшает результат, но увеличивает время обучения
- разумное увеличение размера batch-а уменьшает время обучения
- наличие validation_data позволяет избежать переобучения модели

Не повлияло на точность работы модели (результат остался примерно таким же):

- замена оптимизатора с adam на RMSprop не улучшила результат
- замена функции потерь с categorical_crossentropy на categorical_hinge не улучшила результат

2.

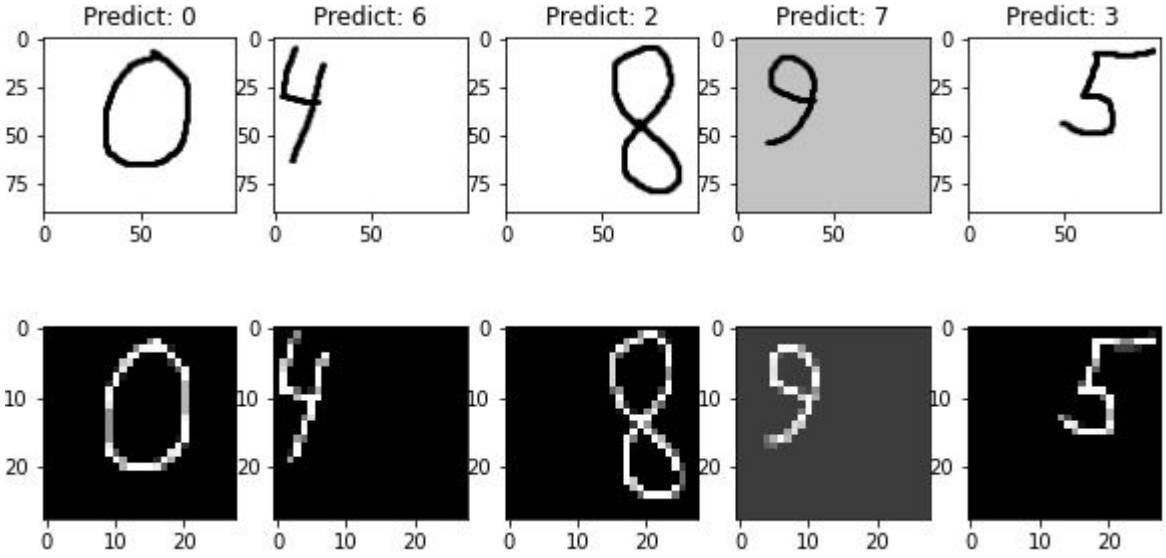
(по желанию) Нарисуйте цифру от руки, преобразуйте как надо и подайте на вход вашей обученной нейронности, оцените точность предсказания.

Основная проблема при распознавании собственной цифры состоит в том, что произвольная картинка сильно отличается от базы картинок MNIST:

- исходные MNIST-овские цифры помещаются в квадратную картинку 20x20 пикселей.
- затем вычисляется центр масс изображения и оно располагается на поле размера 28x28 пикселей таким образом, чтобы центр масс совпадал с центром поля.

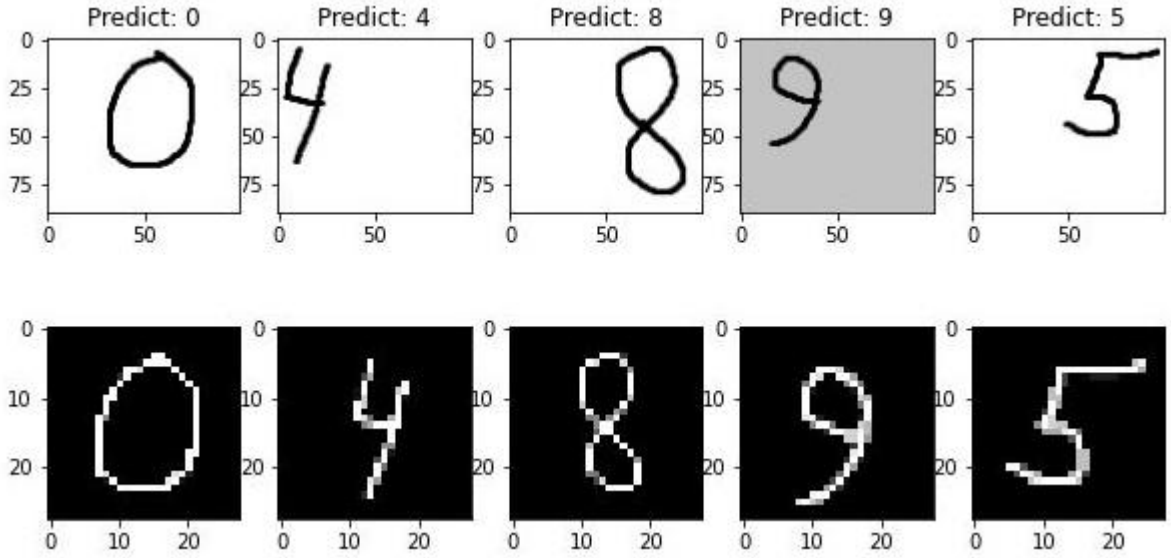
Именно к такому виду мы и должны подгонять наши данные, преобразуя все картинки к такому формату. Также важно, что если фон не совсем белый, то мы получим что-то, сильно отличающееся от мнистовского датасета. Необходимо добавить пороговую обработку после считывания изображения и инвентировать цвета.

Пример предсказаний до обработки изображений:



Ноль распознался нормально, потому что находится по центру и расположен в целом довольно удачно. С остальными числами плохо.
Получается, точность на 5 тестовых картинках всего 20 процентов.

Можно добиться очень хороших результатов, используя предобработку изображений:



Получается, точность на 5 тестовых картинках 100 процентов.

Информация взята из статьи на Habr-е.