Введение в нейронные сети

Урок 3. TensorFlow

Сдавать через Гитхаб.

1. Попробуйте улучшить работу нейронной сети (разобранную на уроке), обучавшейся на датасет Fashion-MNIST. Напишите в комментариях к уроку, какого результата вы добились от нейросети и что помогло улучшить её точность
2. Поработайте с документацией TensorFlow 2. Попробуйте найти полезные команды TensorFlow, неразобранные на уроке
3. * Попробуйте обучить нейронную сеть на TensorFlow 2 на датасете imdb_reviews.

Напишите в комментариях к уроку, какого результата вы добились от нейросети и что помогло улучшить её точность

Задания и ответы:

1. **Попробуйте улучшить работу нейронной сети (разобранную на уроке), обучавшейся на датасет Fashion-MNIST. Напишите в комментариях к уроку, какого результата вы добились от нейросети и что помогло улучшить её точность**

Эксперименты:

Тест 1_1

```
[ ]  model_1_1 = keras.Sequential([
        keras.Input(shape=input_shape),
        Conv2D(32, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
        MaxPooling2D( (2,2), strides = 2),
        Flatten(), # перевод в одномерный массив
        Dense(50), # количество входных нейронов
        Dense(10)  # количество выходных нейронов
        ] )
```

```
[ ]  model_1_1.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
# сразу нормализуем данные и кодируем результат в код вида 3 = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
x_train = x_train / 255
x_test = x_test / 255
y_train_cat = keras.utils.to_categorical(y_train, 10)
y_test_cat = keras.utils.to_categorical(y_test, 10)
```

```
# обучаем сразу с применением валидационной выборки
%%time

history = model_1_1.fit(x_train, y_train_cat, batch_size=32, epochs=5,validation_data =(x_test, y_test_cat) )
```

Получили плохой результат

```
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 9.6768 - accuracy: 0.1006 - val_loss: 9.6918 - val_accuracy: 0.1005
CPU times: user 37.4 s, sys: 2.98 s, total: 40.4 s
Wall time: 39.7 s
```

Результаты печальные...

Тест 1_2

```
[12] # добавим актвацию на каждом слое!
     model_1_2 = keras.Sequential([
           keras.Input(shape=input_shape),
           Conv2D(32, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Flatten(), # перевод в одномерный массив
           Dense(50,activation = 'relu'), # количество входных нейронов
           Dense(10, activation = 'softmax')   # количество выходных нейронов
           ] )
```

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

```
     # обучаем сразу с применением валидационной выборки
     %%time

     history = model_1_2.fit(x_train, y_train_cat, batch_size=32, epochs=5,validation_data =(x_test, y_test_cat) )
```

```
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1808 - accuracy: 0.9324 - val_loss: 0.2554 - val_accuracy: 0.9140
CPU times: user 36.4 s, sys: 3.84 s, total: 40.2 s
Wall time: 36 s
```

Результат сразу улучшился!

Тест 2

```
[15] # сравним с простой моделью, без свёрточных слоёв
     model_2 = keras.Sequential([
           keras.Input(shape=input_shape),
           Flatten(),
           Dense(50, activation = 'relu'),
           Dense(10, activation = 'softmax')
           ] )
```

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

```
[17] %%time

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

```
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3226 - accuracy: 0.8831 - val_loss: 0.3658 - val_accuracy: 0.8683
CPU times: user 30.9 s, sys: 3.7 s, total: 34.6 s
Wall time: 41.9 s
```

У модели со свёрточным слоем результат был лучше, поэтому не будем отказываться от них!

Тест 3

```
# добавляем ещё один свёрточный слой
model_3 = keras.Sequential([
      keras.Input(shape=input_shape),
      Conv2D(32, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
      MaxPooling2D( (2,2), strides = 2),
      Conv2D(64, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
      MaxPooling2D( (2,2), strides = 2),
      Flatten(), # перевод в одномерный массив
      Dense(50,activation = 'relu'), # количество входных нейронов
      Dense(10, activation = 'softmax')   # количество выходных нейронов
      ] )
```

```
[19] model_3.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])
```

```
[20] %%time

     history = model_3.fit(x_train, y_train_cat, batch_size=32, epochs=5,validation_data =(x_test, y_test_cat) )
```

```
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1816 - accuracy: 0.9320 - val_loss: 0.2534 - val_accuracy: 0.9085
CPU times: user 41.2 s, sys: 4.2 s, total: 45.4 s
Wall time: 41.6 s
```

У модели с двумя свёрточными слоями результат стал ещё лучше.

Тест 4

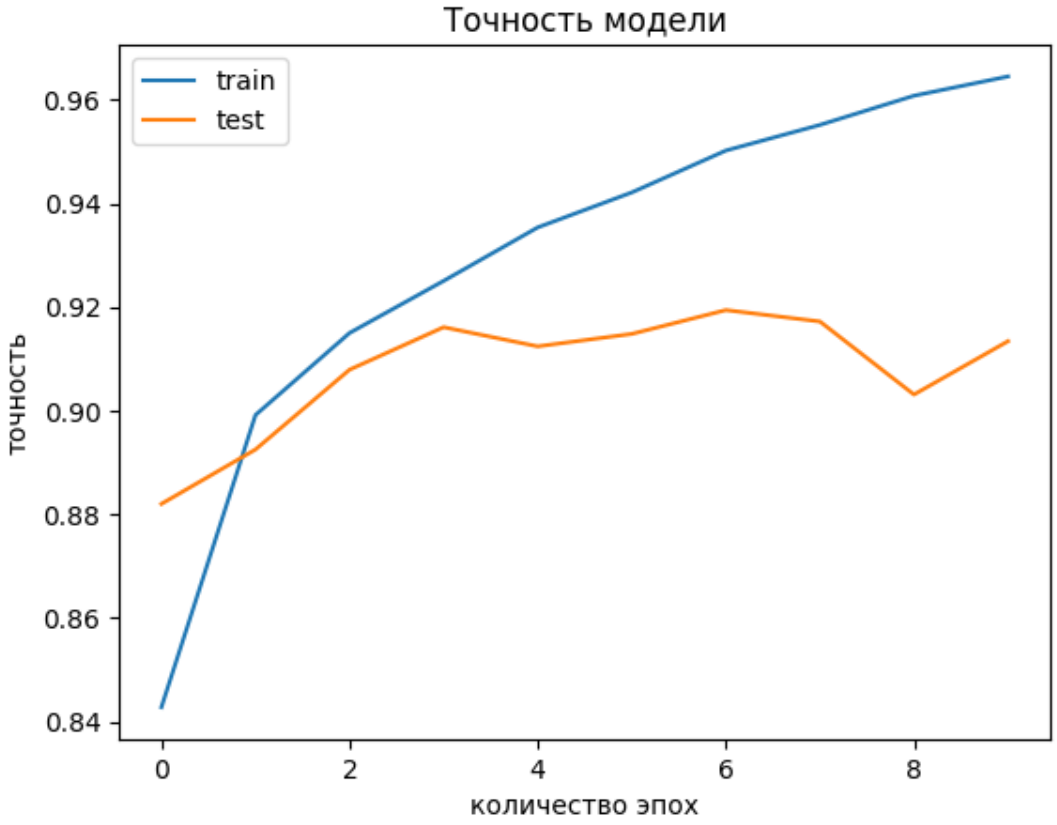
```
[21] # добавим третий свёрточный слой
     # и увеличим количество нейронов на скрытом полносвязном слое
     model_4 = keras.Sequential([
           keras.Input(shape=input_shape),
           Conv2D(32, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Conv2D(64, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Conv2D(128, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Flatten(), # перевод в одномерный массив
           Dense(128,activation = 'relu'), # количество входных нейронов
           Dense(10, activation = 'softmax')   # количество выходных нейронов
           ] )
```

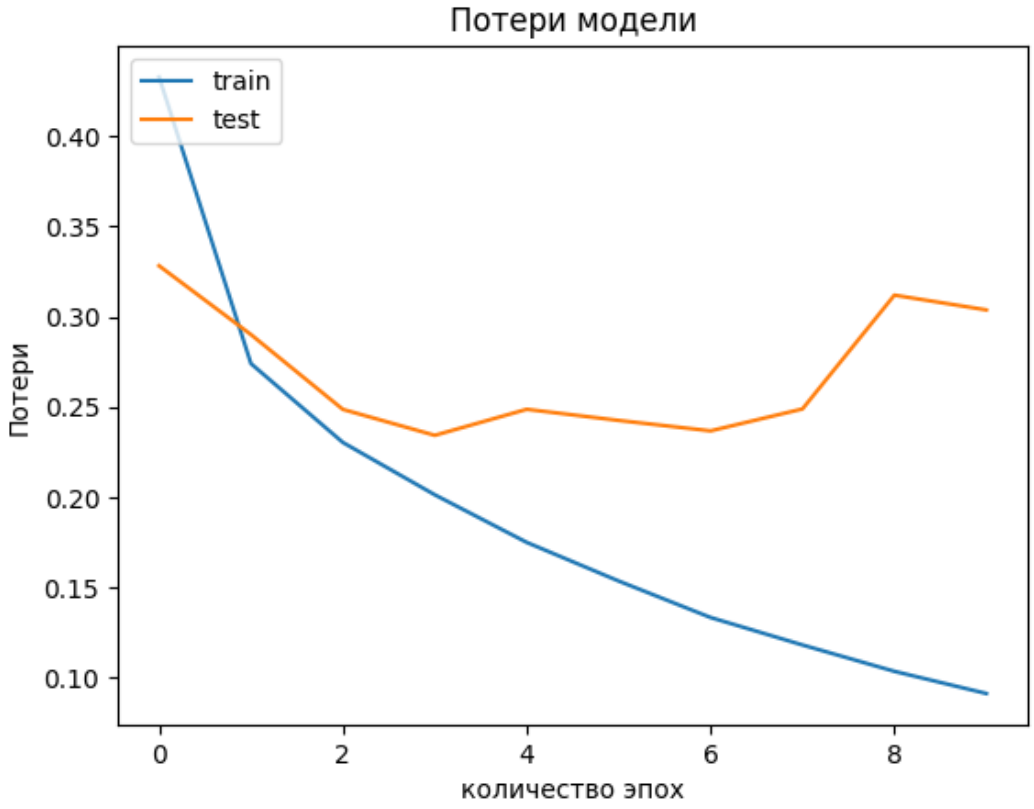
```
[22] model_4.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])
```

```
# увеличим в два раза количество эпох
%%time

history = model_4.fit(x_train, y_train_cat, batch_size=32, epochs=10,validation_data =(x_test, y_test_cat) )
```

```
Epoch 10/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0913 - accuracy: 0.9645 - val_loss: 0.3037 - val_accuracy: 0.9134
CPU times: user 1min 28s, sys: 7.5 s, total: 1min 35s
Wall time: 2min 23s
```

У модели с тремя свёрточными слоями результат пока самый лучший





На 7-й эпохе был самый лучший результат на Тестовых данных

val_loss: 0.2367 - val_accuracy: 0.9194

После 7-й эпохи наша модель немного переобучилась.

1. Попробуйте улучшить работу нейронной сети (разобранную на уроке), обучавшейся на датасет Fa
MNIST.

Напишите в комментариях к уроку, какого результата вы добились от нейросети и что помогло улучшить её точность

Тест 5_1

Попробуем совместить лучшую Модель 3_5 из ПЗ №2 и текущую Модель_4

```
[30] # добавим второй полносвязный скрытый слой
     model_5_1 = keras.Sequential([
           keras.Input(shape=input_shape),
           Conv2D(32, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Conv2D(64, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Conv2D(128, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Flatten(), # перевод в одномерный массив
           Dense(128,activation = 'relu'), # количество входных нейронов
           Dense(64,activation = 'relu'),
           Dense(10, activation = 'softmax')   # количество выходных нейронов
           ] )
```

```
[31] model_5_1.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
```

```
[32] # увеличим размер батча в 4 раза, чтобы долго не ждать
     %%time

     history = model_5_1.fit(x_train, y_train_cat, batch_size=128, epochs=10, validation_data =(x_test, y_test_cat) )
```

```
Epoch 9/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1499 - accuracy: 0.9442 - val_loss: 0.2441 - val_accuracy: 0.9151
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1353 - accuracy: 0.9501 - val_loss: 0.2624 - val_accuracy: 0.9148
CPU times: user 32.4 s, sys: 2.42 s, total: 34.8 s
Wall time: 34 s
```

val_loss: 0.2367 - val_accuracy: 0.9194

vs

val_loss: 0.2441 - val_accuracy: 0.9151

Результат почти не изменился

Тест 5_2

```
[33] # добавим третий полносвязный скрытый слой
     model_5_2 = keras.Sequential([
           keras.Input(shape=input_shape),
           Conv2D(32, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Conv2D(64, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Conv2D(128, padding = 'same', kernel_size = (3,3), activation = 'relu' ),
           MaxPooling2D( (2,2), strides = 2),
           Flatten(), # перевод в одномерный массив
           Dense(128,activation = 'relu'), # количество входных нейронов
           Dense(64, activation= 'tanh'), # немного выровняем
           Dense(64, activation= 'relu'),
           Dense(10, activation = 'softmax')   # количество выходных нейронов
           ] )
```

```
[34] model_5_2.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
```

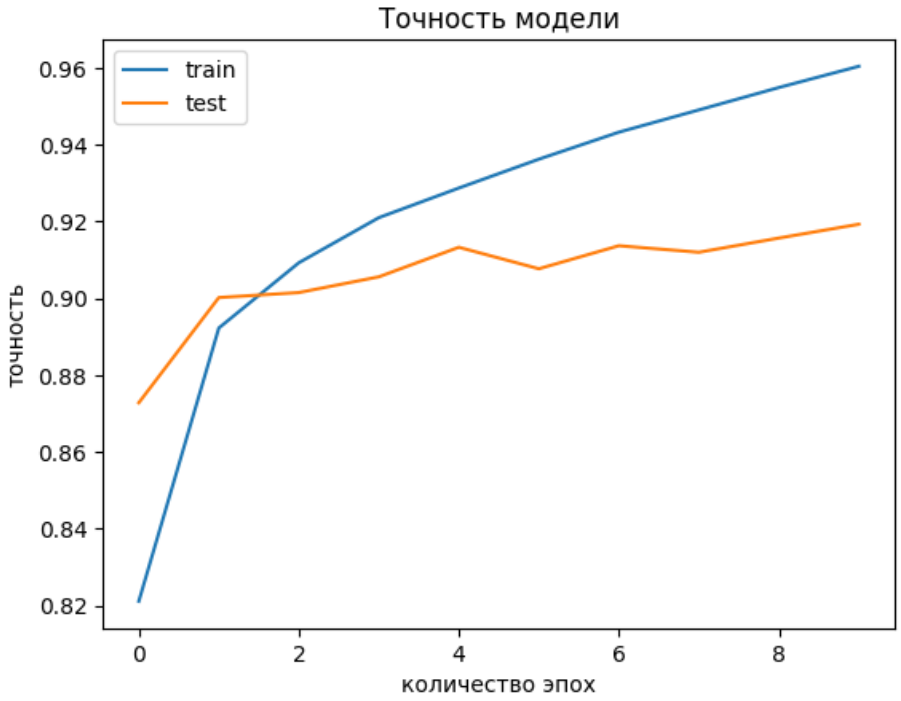
```
# увеличим размер батча в 2 раза (а не в 4), чтобы долго не ждать
%%time

history = model_5_2.fit(x_train, y_train_cat, batch_size=64, epochs=10, validation_data =(x_test, y_test_cat) )
```

```
Epoch 10/10
938/938 [==============================] - 5s 5ms/step - loss: 0.1077 - accuracy: 0.9604 - val_loss: 0.2647 - val_accuracy: 0.9193
CPU times: user 54.5 s, sys: 4.72 s, total: 59.2 s
Wall time: 1min 23s
```

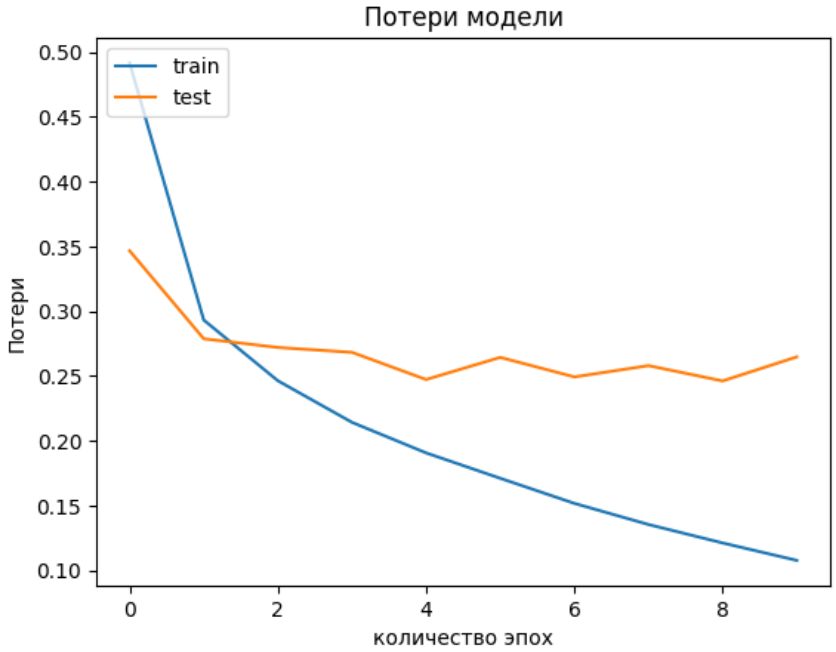
Время работы модели уменьшилось в два раза при той же точности предсказаний!

Epoch 10/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0913 - accuracy: 0.9645 - val_loss: 0.3037 - val_accuracy: 0.9134
CPU times: user 1min 28s, sys: 7.5 s, total: 1min 35s
Wall time: 2min 23s
VS
Epoch 10/10
938/938 [==============================] - 5s 5ms/step - loss: 0.1077 - accuracy: 0.9604 - val_loss: 0.2647 - val_accuracy: 0.9193
CPU times: user 54.5 s, sys: 4.72 s, total: 59.2 s
Wall time: 1min 23s

Визуализируем результат работы Модели 5_2



[37]



На 10-й эпохе был самый лучший результат на Тестовых данных

val_loss: 0.2367 - val_accuracy: 0.9194
vs
val_loss: 0.2647 - val_accuracy: 0.9193
Результаты схожи, но времени затрачено в 2 раза меньше!

Таким образом удалось создать модель с другим количеством слоёв, которая работает в два раза быстрее при той же точности. Это Модель 5_2.

2.

Поработайте с документацией TensorFlow 2. Попробуйте найти полезные команды TensorFlow, неразобранные на уроке

2. Поработайте с документацией TensorFlow 2. Попробуйте найти полезные команды TensorFlow, неразобранные на уроке

В TensorFlow 2 есть команда tf.random.Generator.
Это класс, который предоставляет генератор случайных чисел для создания случайных тензоров с различными распределениями.
С помощью этого класса можно создавать случайные тензоры, например, с нормальным или равномерным распределением.

Базовые основные команды:

1. tf.constant(value): Создает тензор с постоянным значением.
2. tf.Variable(initial_value): Создает переменную тензора, которая может изменяться в процессе обучения.
3. tf.matmul(tensor1, tensor2): Умножает два тензора.
4. tf.nn.relu(tensor): Применяет функцию активации ReLU к тензору.
5. tf.keras.layers.Dense(units): Создает полносвязный слой с указанным количеством нейронов.
6. tf.keras.optimizers.Adam(learning_rate): Создает оптимизатор Adam с указанным коэффициентом обучения.
7. tf.keras.losses.BinaryCrossentropy(): Создает функцию потерь бинарной кросс-энтропии для бинарной классификации.
8. tf.data.Dataset.from_tensor_slices(data): Создает датасет из массива данных.