Московский Государственный Технический Университет имени Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»

Отчёт по лабораторной работе № 2 по дисциплине «Проектирование интеллектуальных систем»

Исполнитель:
Саврасов П.А.
Группа ИУ5-24М
«__» _____ 2021 г.

Преподаватель:
Терехов В.И.
_____
«__» _____ 2021 г.

Москва, 2021 г.

1. Цель работы

Познакомиться с примерами работы с моделями на примере набора рукописных цифр MNIST

2. Логистическая регрессия для распознавания набора данных MNIST

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
```

Загрузим датасет MNIST и рассмотрим его формат

```
(xTrain, yTrain), (xTest, yTest) = mnist.load_data()

print(xTrain.shape, yTrain.shape)
print(xTest.shape, yTest.shape)
```

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

Как видно, формат датасета (обучающей выборки) следующий:

- 60000 образцов изображений
- 28 пикселей в высоту
- 28 пикселей в ширину

То есть это трёхмерный массив данных. Такой формат не подходтит для обучения нейросетей, требуется изименить форму датасета на двухмерный массив. Число образцов оставим неизменным, а двумерный массив пикселей преобразуем в одномерный:

```
xTrain = tf.reshape(xTrain, (60000, 28 * 28))
print(xTrain.shape)
xTest = tf.reshape(xTest, (10000, 28 * 28))
print(xTest.shape)
```

```
(60000, 784)
(10000, 784)
```

Рассмотрим формат одного элемента массива:

```
print(xTrain[0].dtype)
print(xTrain[0][200:250])
```

```
<dtype: 'uint8'>
tf.Tensor(
[  0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56
  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253 253 253
 253 253 198 182 247 241   0   0   0   0   0   0   0   0], shape=(50,), dtype=uint8)
```

Формат элемента является 8 битным беззнаковым целым, диапазон значений которого лежит в промежутке от 0 до 255. Такой формат для обучения нейросестей не самый удачный. Произведём масштабирование и преобразуем диапазон от 0 до 1. При этом нужно учесть, что сначала нужно преобразовать тип данных в float32

```
xTrain = tf.cast(xTrain, dtype = tf.float32) / 255
xTest = tf.cast(xTest, dtype = tf.float32) / 255
print(xTrain[0].dtype)
print(xTrain[0][200:250])
```

```
<dtype: 'float32'>
tf.Tensor(
[0.         0.         0.         0.19215687 0.93333334 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.9843137  0.3647059  0.32156864 0.32156864 0.21960784
 0.15294118 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.07058824 0.85882354 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.7764706  0.7137255  0.96862745 0.94509804
 0.         0.         0.         0.         0.         0.
 0.         0.        ], shape=(50,), dtype=float32)
```

Датасет преобразован и готов для обучения моделей
Начнём с обучения логистической регрессии

3. Логистическая регрессия для распознавания набора данных MNIST

4. Нейронная сеть с 5 полносвязными слоями (200, 100, 60, 30, 10) для распознавания набора данных MNIST

Теперь обучим нейронную сеть следующими параметрами:

- Входной слой: 784
- Скрытый слой 1: 200
- Скрытый слой 2: 100
- Скрытый слой 3: 60
- Скрытый слой 4: 30
- Выходной слой: 10

Сначала расмотрим упрощённый вариант на основе Sequential API

```
model = keras.Sequential(
    [
        layers.InputLayer(input_shape=(28*28)),
        layers.Dense(200, activation='relu'),
        layers.Dense(100, activation='relu'),
        layers.Dense(60, activation='relu'),
        layers.Dense(30, activation='relu'),
        layers.Dense(10, activation='relu')
    ]
)
```

Установим модели функцию потерь и оптимизатор (способ минимизации функции потери)

```
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(from_logits = True),
    optimizer = keras.optimizers.SGD(learning_rate = 0.5),
    metrics = ['accuracy']
)
```

Обучим модель со следующими параметрами:
Размер батча (порция данных): 32 образца
Число эпох: 5

```
model.fit(xTrain, yTrain, batch_size=32, epochs=5, verbose=2)
model.evaluate(xTest, yTest, batch_size=32, verbose=2)
```

```
Epoch 1/5
1875/1875 - 4s - loss: 1.3936 - accuracy: 0.5317
Epoch 2/5
1875/1875 - 4s - loss: 0.7025 - accuracy: 0.7898
Epoch 3/5
1875/1875 - 4s - loss: 0.1642 - accuracy: 0.9573
Epoch 4/5
1875/1875 - 4s - loss: 0.1280 - accuracy: 0.9669
Epoch 5/5
1875/1875 - 4s - loss: 0.1111 - accuracy: 0.9706
313/313 - 0s - loss: 0.1453 - accuracy: 0.9667

[0.1453077793121338, 0.96670001745224]
```

Теперь повторим тоже самое, но уже с применением Functional API

```
inpLayer = keras.Input(shape=(28*28))
hidLayer1 = layers.Dense(200, activation='relu')(inpLayer)
hidLayer2 = layers.Dense(100, activation='relu')(hidLayer1)
hidLayer3 = layers.Dense(60, activation='relu')(hidLayer2)
hidLayer4 = layers.Dense(30, activation='relu')(hidLayer3)
outLayer = layers.Dense(10, activation='relu')(hidLayer4)

model = keras.Model(inputs=inpLayer, outputs=outLayer)
```

```
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(from_logits = True),
    optimizer = keras.optimizers.SGD(learning_rate = 0.5),
    metrics = ['accuracy']
)
```

```
model.fit(xTrain, yTrain, batch_size=32, epochs=5, verbose=2)
model.evaluate(xTest, yTest, batch_size=32, verbose=2)
```

```
Epoch 1/5
1875/1875 - 4s - loss: 1.1223 - accuracy: 0.6493
Epoch 2/5
1875/1875 - 4s - loss: 0.8117 - accuracy: 0.7653
Epoch 3/5
1875/1875 - 4s - loss: 0.7794 - accuracy: 0.7744
Epoch 4/5
1875/1875 - 4s - loss: 0.7640 - accuracy: 0.7793
Epoch 5/5
1875/1875 - 4s - loss: 0.7529 - accuracy: 0.7824
313/313 - 0s - loss: 0.7632 - accuracy: 0.7790

[0.7632175087928772, 0.7789999842643738]
```

5. Выводы по работе:

Познакомился с примерами работы с моделями на примере набора рукописных цифр MNIST

6. Контрольные вопросы

6.1. Что такое Variable?

В TensorFlow для хранения значений модели существует специальный тип tf.Variable. В отличие от других Tensor объектов которые заново обновляются при каждом запуске сессии, переменные (Variable) хранят фиксированное значение в графе.

Это является важным, т.к. при текущее значение переменной влияет на вывод в вычисляемой итерации. Как и другие Tensor объекты, переменные можно использовать как входные значения в графе.

6.2. Что такое placeholder?

Для добавления входных данных извне модели в TensorFlow используется специальный тип - плейсхолдер (placeholder). Плейсхолдер можно представить в виде пустой переменной который будет заполняться данными позже. Сперва их используют для создания графа и заполняют данными при выполнении сессии.

6.3. Что такое функция потерь?

Функция потерь(стоимости) – используется в качестве метрики для определения качества модели. Это расстояние(разница) между предсказанием модели и истинным значением входного вектора.

6.4. Какие другие названия функции потери?

Функция стоимости.

6.5. Зачем нужна функция потери?

Функция потерь(стоимости) – используется в качестве метрики для определения качества модели. Это расстояние(разница) между предсказанием модели и истинным значением входного вектора.

6.6. Как запустить обучение модели?

Для Tensorflow 1: В метод Tf.Session().run() передаем шаг градиентного спуска и значения для placeholder.

Для Tensorflow 2: В метод fit класса модели передать обучающие данные, число эпох, размер минибатча и тип отображения процесса обучения.

6.7. Что делает tf.global_variables_initializer()?

Вызывается при вызове метода сессии .run() для создания в оперативной памяти области для хранения переменных и их исходных значений.

6.8. Что такое minibatch?

Небольшая порция примеров из общего датасета. Обычно объем данной подвыборки варьируется от 50 до 500 примеров.

6.9. Какие бывают активационные функции?

Логистическая, тангенсальная и ReLU (Rectified Linear Unit) активационные функции.