МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Искусственные нейронные сети»

Тема: Бинарная классификация отраженных сигналов радара

Студент гр. 8383 _____ Федоров И.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Реализовать классификацию между камнями (R) и металлическими цилиндрами (M) на основе данных об отражении сигналов радара от поверхностей. 60 входных значений показывают силу отражаемого сигнала под определенным углом. Входные данные нормализованы и находятся в промежутке от 0 до 1.

Задачи

- Ознакомиться с задачей бинарной классификации
- Загрузить данные
- Создать модель ИНС в tf.Keras
- Настроить параметры обучения
- Обучить и оценить модель
- Изменить модель и провести сравнение. Объяснить результаты

Выполнение работы.

Были импортированы необходимые для работы класы, модули и функции.

```
import pandas
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import plot_model
```

Набор данных загружается из файла "sonar.csv", находящегося в той же директории, что и проект. После чего данные перемешиваются с помощью функции `shuffle()` (т.к. данные в файле идут последовательно). Затем данные разделяются на входные данные *X* (характеристик) и выходные *Y* (строка названия - метки).

```
dataframe = pandas.read_csv("sonar.csv", header=None)
dataset = dataframe.values
np.random.shuffle(dataset)
X = dataset[:,0:60].astype(float)
Y = dataset[:,60]
```

Для подготовки (переход к категориальному вектору) текстовых меток был использован метод прямого кодирования (one hot encoding).

```
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
dummy_y = to_categorical(encoded_Y)
```

Часть данных была выделена для контрольного набора:

```
test_index = int(len(X)-len(X)*0.2)
test_data_x = X[test_index:]
X = X[:test_index]
test_index = int(len(encoded_Y)-len(encoded_Y)*0.2)
test_data_y = encoded_Y[test_index:]
encoded_Y = encoded_Y[:test_index]
```

Для создания модели была реализована простая функция `get_model()`, возвращающая модель. Полный текст программы приведен в приложении А.

Для построения графиков потерь и точности при обучении и тестировании была использована функция из предыдущей работы `plot_model_loss_and_acc()`, которая принимает на вход объект `history`, который возвращает метод обучения модели `fit()`. При построении используется библиотека `matplotlib`.

При компиляции модели используется функция потерь `binary_crossentropy`, т.к. в данном случае стоит задача бинарной классификации.

Т.к. объем данных достаточно небольшой, то для более качественного оценивания модели был применен метод перекрестной проверки по *k* блокам. Она заключается в разбиении данных на *k* блоков равного размера. Для каждого блока выполняется обучение исследуемой модели на остальных *k-1* блоках, а оценка на текущем блоке. Конечная оценка рассчитывается как среднее всех промежуточных оценок. Схематично метод выглядит как на рис. 1.

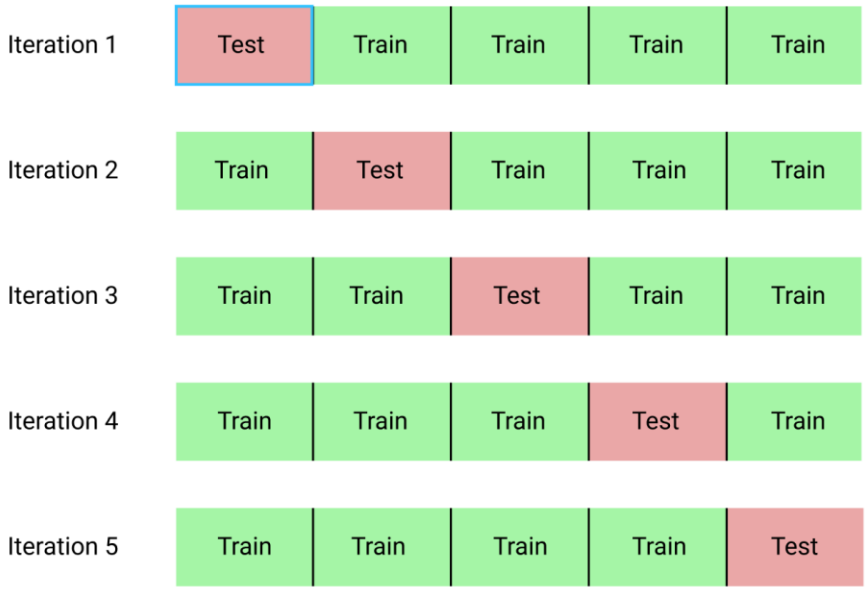

Рисунок 1 - Перекрестная проверка

В работе также использовалась встроенная в Keras визуализация моделей:

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

С помощью метода `fit()` осуществляется обучение модели:

```
history = model.fit(train_data_x, train_data_y, epochs=100,
batch_size=10, verbose=0)
```

Первая модель имеет вид, представленный на рис. 2.

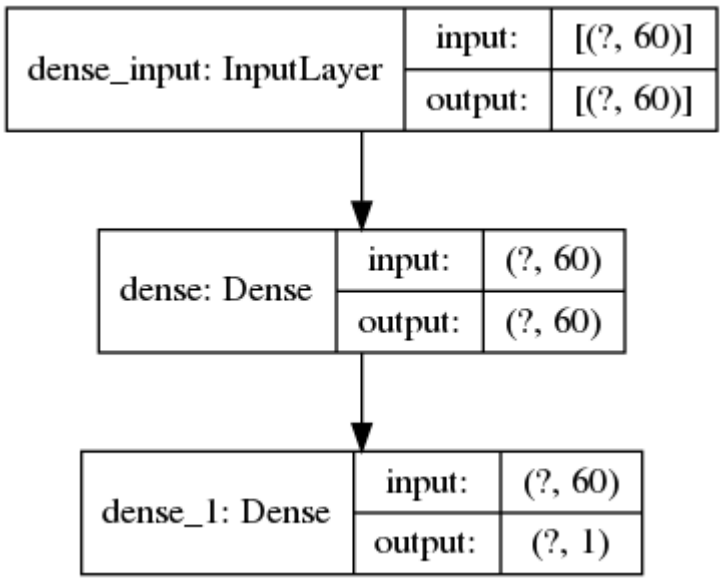

Рисунок 2 - Схема модели 1

В данном случае "?" в схеме означает размер одного пакета данных - batch (т.к. стоит символ "?", то он может быть любым).

После "прогонки" данной модели через цикл, реализующий перекрестную проверку, и проверки на контрольном наборе получены результаты, представленные в табл. 1.

Таблица 1

Общая оценка потерь и точности по методу	
0.429441563	0.8109756103
Оценки на контрольном наборе	
0.438867112	0.7957114072

Графики потерь и точности при обучении и тестировании (уже окончательном, на контрольном наборе) показаны на рис. 3.

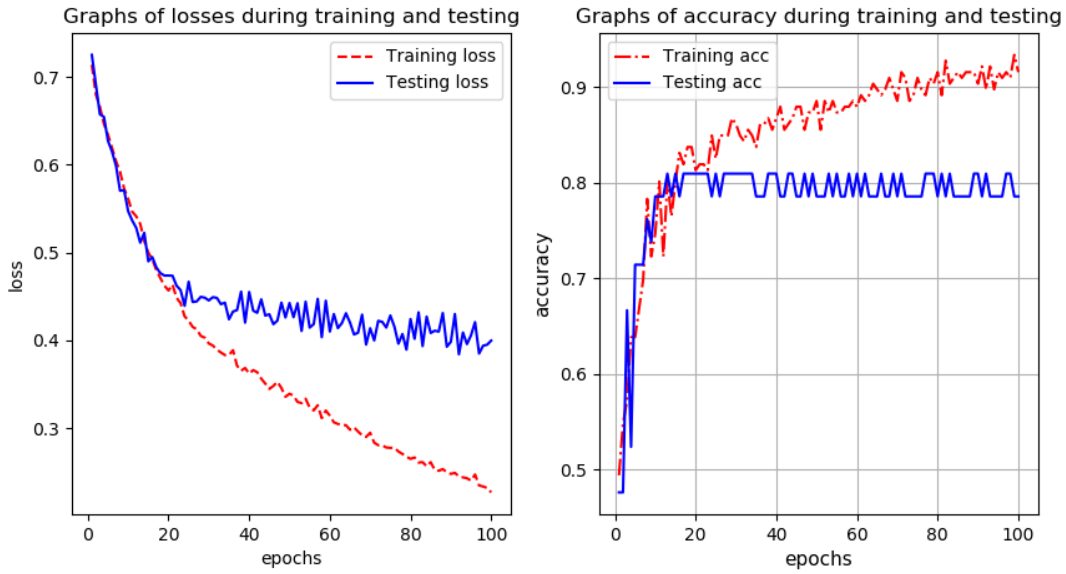

Рисунок 3 - Графики потерь и точности

В представленном наборе данных присутствует некоторая избыточность, т.к. с разных углов описывается один и тот же сигнал. Вероятно, что некоторые углы отражения сигнала имеют большую значимость, чем другие. Изменение количества нейронов во входном слое напрямую влияет на количество признаков, с которыми будет работать нейронная сеть. Уменьшим размер входного слоя в два раза (естественной

данные также предаются в сокращенном варианте) и сравним с результатами первоначальной архитектуры. Схема модели показана на рис. 4.

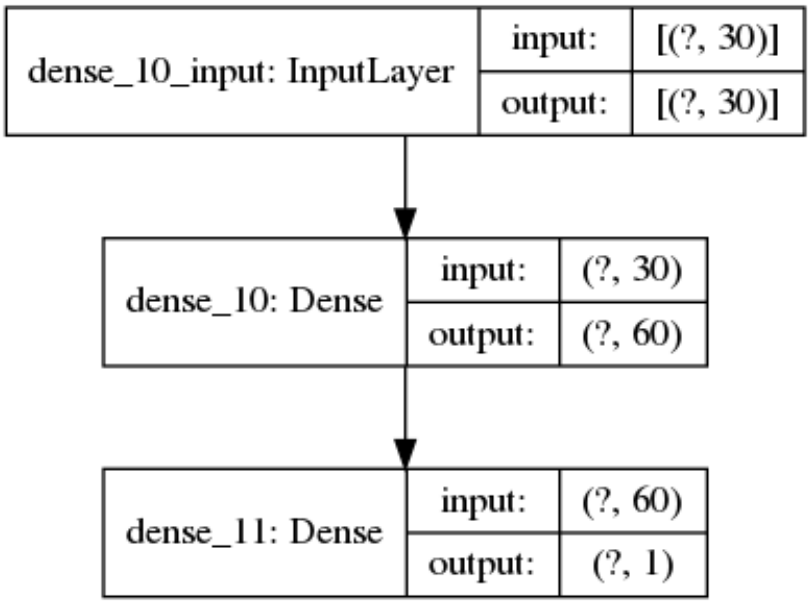

Рисунок 4 - Схема модели 2

После "прогонки" данной модели и проверки на контрольном наборе получены результаты, представленные в табл. 2.

Таблица 2

Общая оценка потерь и точности по методу	
0.525723184	0.7784349103
Оценки на контрольном наборе	
0.45598772	0.7619026112

Графики потерь и точности при обучении и тестировании (уже окончательном, на контрольном наборе) показаны на рис. 5.

Рисунок 5 - Графики потерь и точности

Можно заметить, что произошло увеличение значения потер, а также некоторое снижение оценочной точности и контрольной точности. Вероятно уменьшение размера входного слоя в два раза чрезмено. Попробуем сократить размер входного слоя до 50. Результаты представлены в табл. 3. Схема модели показана на рис. 6. Графики потерь и точности показаны на рис. 7.

Таблица 3

Общая оценка потерь и точности по методу	
0.43512169	0.8203179524
Оценки на контрольном наборе	
0.42272453	0.8095513814

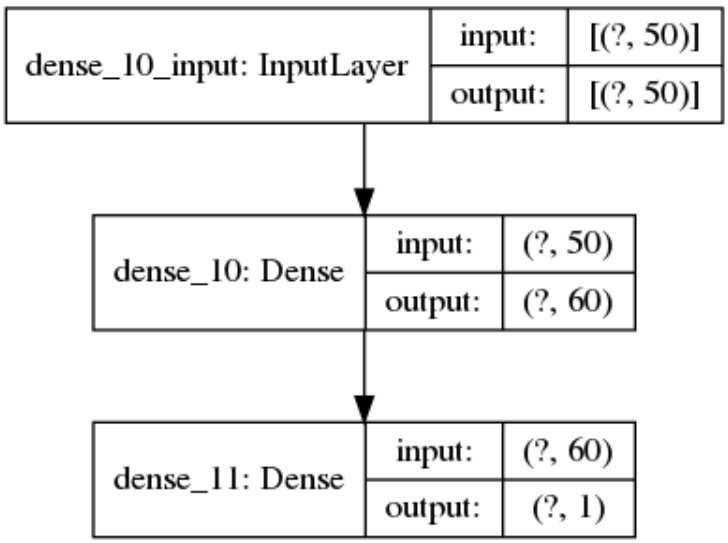

Рисунок 6 - Схема модели 3

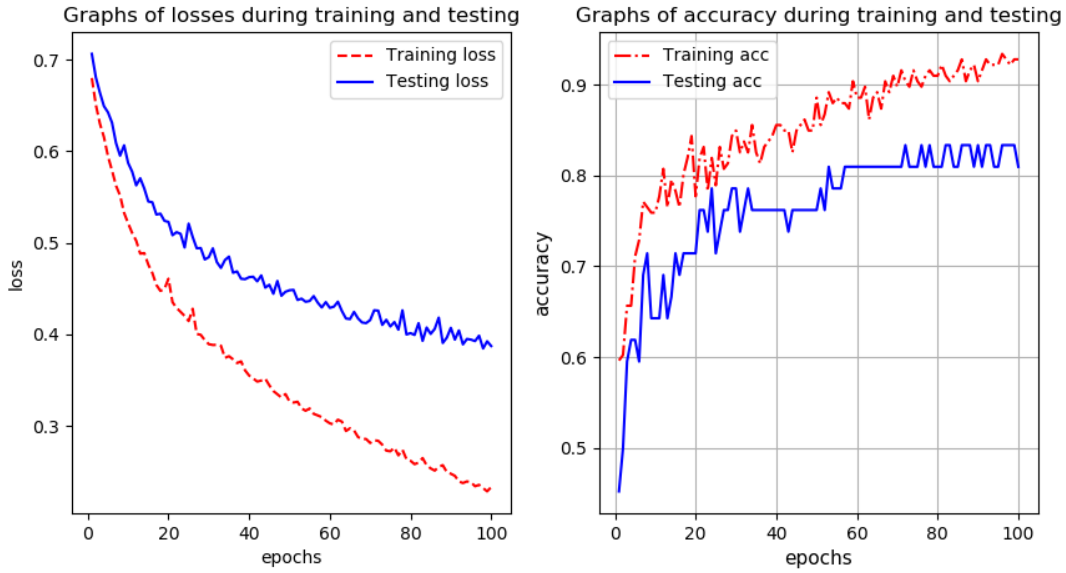

Рисунок 7 - Графики потерь и точности

Можно заметить, что результаты модели 3 очень схожи с результатами модели 1, значит данное уменьшение входного слоя является корректным.

Нейронная сеть с несколькими слоями позволяет находить закономерности не только во входных данных, но и в их комбинации. Также, дополнительные слои позволяют ввести нелинейность в сеть, что позволяет получать более высокую точность. Добавим промежуточный (скрытый) слой *Dense* с 15 нейронами в изначальную архитектуру сети. Схема модели показана на рис. 8.

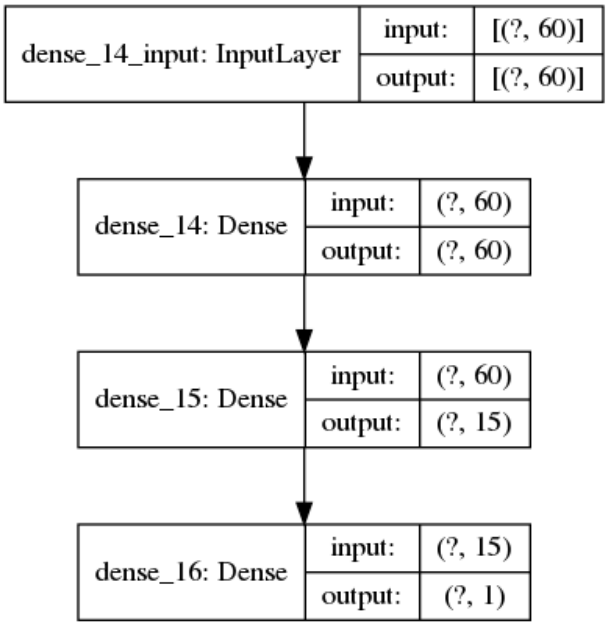

Рисунок 8 - Схема модели 4

После "прогонки" модели и проверки на контрольном наборе получены результаты, представленные в табл. 4. Графики потерь и точности: рис. 9.

Таблица 4

Общая оценка потерь и точности по методу	
0.506578232	0.8357629731
Оценки на контрольном наборе	
0.399876453	0.884463964

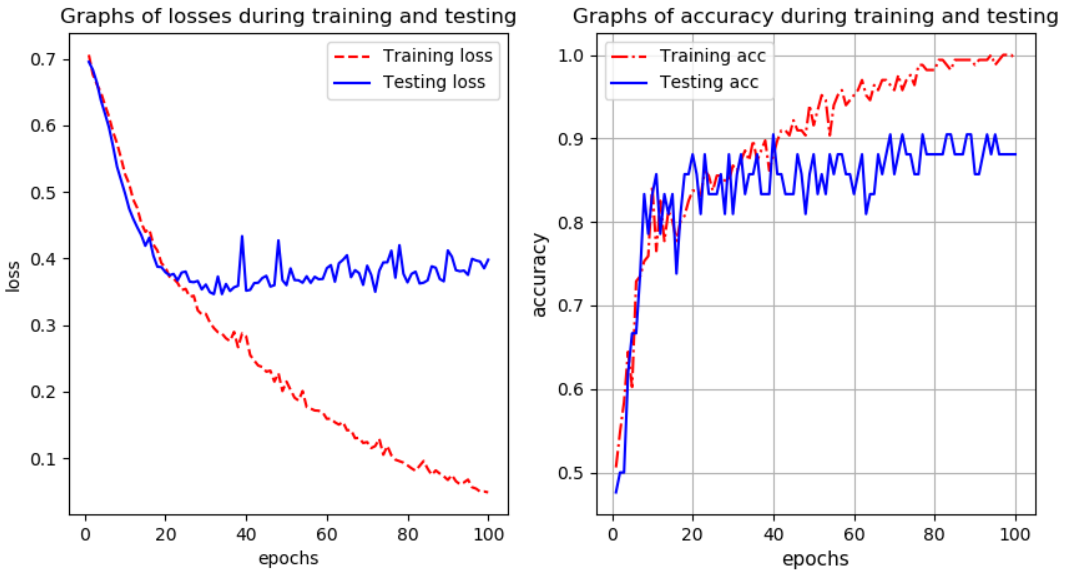

Рисунок 9 - Графики потерь и точности

Можно заметить, что повысилась оценочная и контрольная точность, что говорит о том, что добавление промежуточного слоя положительно сказалось на качетсве модели. Кроме того, увеличилась "скорость" обучения модели, т.к. уже примерно на 40 эпохе наступает переобучение модели.

Выводы.

В ходе выполнения данной работы была построена модель классификации между камнями и металлическими цилиндрами на основе данных об отражении сигналов радара от поверхностей. Были проведены эксперименты с уменьшением/увеличением размера входного слоя. Были построены и сравнены модели с разным числом нейронов и числом слоев.

Приложение А

```
import pandas
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import plot_model

# X - входные данные, У-выходные, данные перемешиваются
dataframe = pandas.read_csv("sonar.csv", header=None)
dataset = dataframe.values
np.random.shuffle(dataset)

X = dataset[:,0:60].astype(float)
Y = dataset[:,60]


#Переход от R,M меток к категориальному вектору
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)    #[1, 1, 1, 1, ... , 0 , 0, 0]

###### отбор тестовых данных
test_index = int(len(X)-len(X)*0.2)
test_data_x = X[test_index:]

X = X[:test_index]
test_index = int(len(encoded_Y)-len(encoded_Y)*0.2)
test_data_y = encoded_Y[test_index:]
encoded_Y = encoded_Y[:test_index]


# графики потерь и точности при обучении и тестирования
def plot_model_loss_and_accuracy(history, figsize_=(10,5)):
    plt.figure(figsize=figsize_)
    train_loss = history.history['loss']
    test_loss = history.history['val_loss']
    train_acc = history.history['acc']
    test_acc = history.history['val_acc']
    epochs = range(1, len(train_loss) + 1)

    plt.subplot(121)
    plt.plot(epochs, train_loss, 'r--', label='Training loss')
    plt.plot(epochs, test_loss, 'b-', label='Testing loss')
    plt.title('Graphs of losses during training and testing')
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend()
    plt.subplot(122)
    plt.plot(epochs, train_acc, 'r-.', label='Training acc')
    plt.plot(epochs, test_acc, 'b-', label='Testing acc')
    plt.title('Graphs of accuracy during training and testing')
    plt.xlabel('epochs', fontsize=11, color='black')
    plt.ylabel('accuracy', fontsize=11, color='black')
    plt.legend()
    plt.grid(True)

    plt.show()
```

```
def get_model():
    model = Sequential()
    model.add(Dense(60, input_dim=60, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam',loss='binary_crossentropy',
metrics=['accuracy'])
    return model

k = 4
num_valid_data = len(X) // k
valid_scores_loss = []
valid_scores_acc = []

#X = X[:, 0:50]
#test_data_x = test_data_x[:, 0:50]
#перекрестная проверка по К блокам
for fold in range(k):
    validation_data_x = X[num_valid_data * fold: num_valid_data * (fold + 1)]
    validation_data_y = encoded_Y[num_valid_data * fold: num_valid_data *
(fold + 1)]

    train_data_x = np.concatenate((X[:num_valid_data * fold] ,
X[num_valid_data * (fold + 1):]))
    train_data_y = np.concatenate((encoded_Y[:num_valid_data * fold] ,
encoded_Y[num_valid_data * (fold + 1):]))

    model = get_model()
    history = model.fit(train_data_x, train_data_y, epochs=100,
batch_size=10, verbose=0)
    results = model.evaluate(validation_data_x, validation_data_y,
batch_size=10,verbose=0)
    valid_scores_loss.append(results[0])
    valid_scores_acc.append(results[1])

valid_score_acc = np.average(valid_scores_acc)
valid_score_loss = np.average(valid_scores_loss)
print("Loss: ", valid_score_loss)
print("Accuracy: ", valid_score_acc)

#обучение на всех данных, кроме контрольного набора
model = get_model()
history = model.fit(X, encoded_Y, epochs=100, batch_size=10,
validation_data=(test_data_x, test_data_y),verbose=2)
test_score = model.evaluate(test_data_x,test_data_y, batch_size=10,verbose=0)
print(test_score)
plot_model_loss_and_accuracy(history)
plot_model(model, to_file='model.png', show_shapes=True)
```