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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Искусственные нейронные сети»
Тема: Многоклассовая классификация цветов

Студентка гр. 8382 _____ Кузина А.М.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы

Реализовать классификацию сортов растения ирис (Iris Setosa - 0, Iris Versicolour - 1, Iris Virginica - 2) по четырем признакам: размерам пестиков и тычинок его цветков.

Задачи:

- Ознакомиться с задачей классификации
- Загрузить данные
- Создать модель ИНС в Keras
- Настроить параметры обучения
- Обучить и оценить модель

Требования:

- Изучить различные архитектуры ИНС (Разное кол-во слоев, разное кол-во нейронов на слоях)
- Изучить обучение при различных параметрах обучения (параметры ф-ций fit)
- Построить графики ошибок и точности в ходе обучения
- Выбрать наилучшую модель

Ход работы

Данные для анализа в программу загружаются из файла iris.csv. Данные разделяются на входные и выходные – параметры цветка и соответствующий им сорт цветка. Затем выходные данные переводятся от текстовых меток к категориальному вектору.

```
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]

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

Затем создается модель, состоящая из двух слоев: входного с 4мя нейронами и функцией активации relu, выходного с 3мя нейронами и функцией активации softmax, который возвращает массив с 3мя оценками вероятности принадлежности случайного цветка какому-то классу, в сумме дают 1. После

этого инициализируются параметры обучения сети – функция потерь, метрика, оптимизатор.

```
model = Sequential()
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))

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

Полученная сеть проходит обучение в течение 75 эпох, с размером батча – 10 и 10% данных отданных на валидацию.

```
history = model.fit(X, dummy_y, epochs=75, batch_size=10, validation_split=0.1)
```

Графики ошибок и точности изначальной сети представлены на рисунке 1

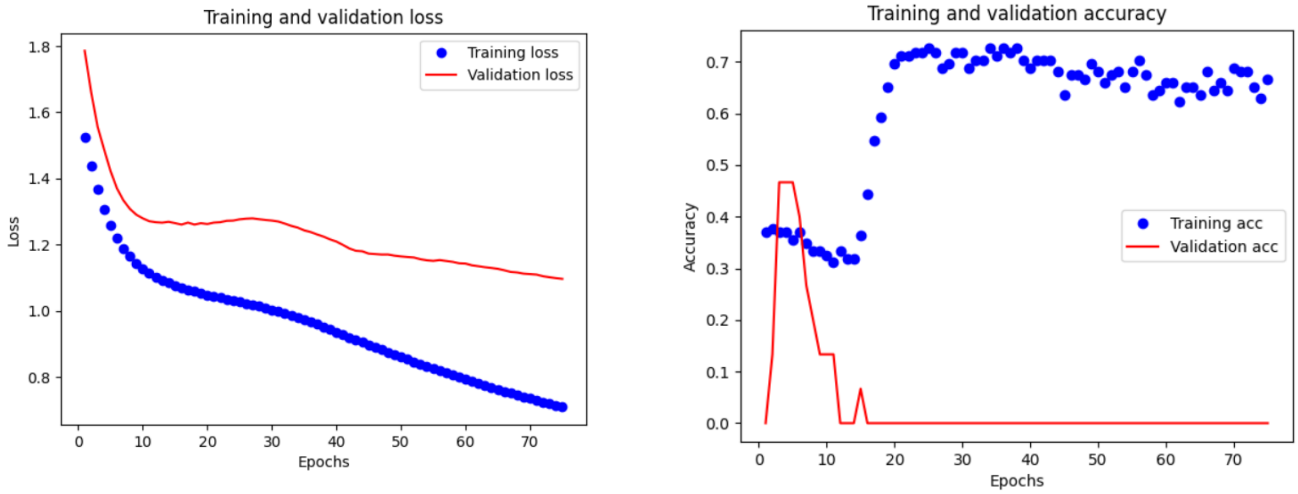


Рисунок 1 - Ошибки и точность изначальной сети

Затем количество эпох было увеличено до 300. При нескольких программы запусках на такой конфигурации сети – получаемые графики каждый раз значительно изменялись – в некоторых вариантах уже на 20й эпохе точность доходила до 1 и не падала ниже 0.9, в некоторых не поднималась выше 0.7 за все 300 эпох.

В модели был модифицирован входной слой – количество нейронов увеличено с 4 до 40. На рисунке 2 представлены графики точности и ошибок сети.

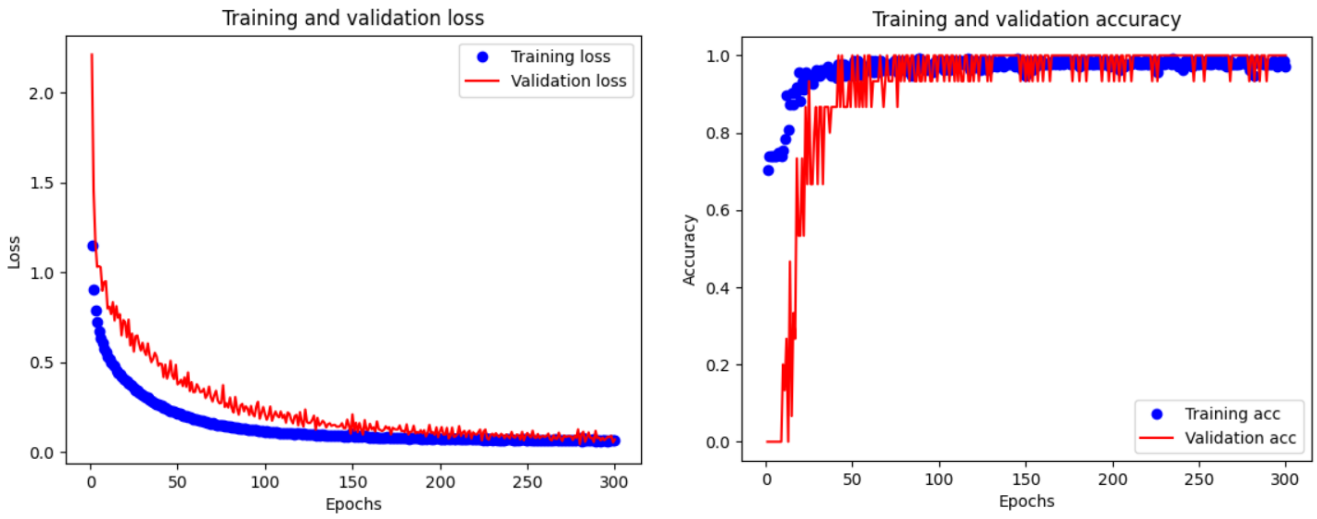


Рисунок 2 – Ошибки и точность сети, на входном слое 40 нейронов

При нескольких запусках графики были визуально близки между собой, что свидетельствует о большей стабильности обучения нейронной сети. Чаще всего уже после 50й эпохи сеть показывала стабильно хорошие результаты – боле 80% точности.

Затем количество нейронов было изменено с 40 на 100 на входном слое, а количество эпох уменьшено до 200. На рисунке 3 представлены графики точности и ошибок сети с данной конфигурацией.

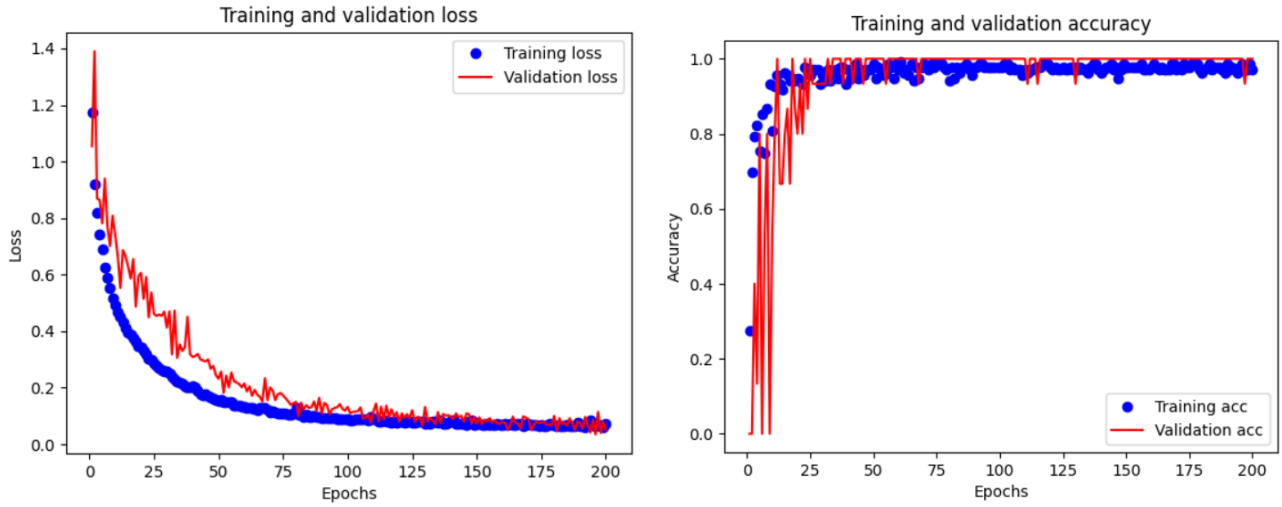
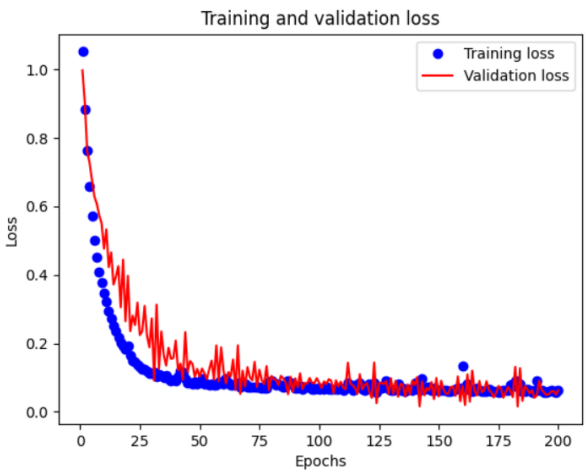


Рисунок 3 – Ошибки и точность сети, на входном слое 100 нейронов

Несмотря на то, что количество эпох, за которые сеть достигает стабильно хороших результатов уменьшилось примерно в два раза – с 50 до 25.

Конфигурация была изменена на следующую: три слоя по 40, 40 и 3 нейронов соответственно, два с функцией активации relu, один softmax. На рисунке 4 представлены графики точности и ошибок сети.



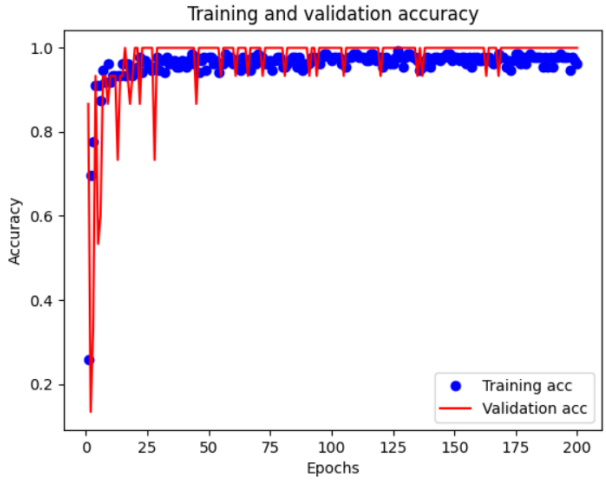


Рисунок 4 – Ошибки и точность сети, два слоя по 40 нейронов

Можно заметить, что в сравнении с предыдущей конфигурацией в данном случае ошибки уменьшаются стремительнее.

Затем текущая конфигурация была изменена так, что на входном и скрытом слое по 30 нейронов. В данном варианте заметны меньше колебания между эпохами – не наблюдается таких резких перепадов между соседними эпохами. Данная конфигурация слоев была выбрана финальной.

Далее были рассмотрены параметры обучения сети такие, как размер батча и процент данных валидации.

При повышении размера батча понижалась точность и возрастали ошибки, но при этом обучение происходило заметно быстрее. Данное значение отвечает за то, через какое количество обработанных элементов будут обновляться веса. Поэтому при небольших значениях параметра постоянное обновление весов занимает большее время. В итоге было выбрано значение параметра равное 8ми. На рисунке 5 представлены графики точности и ошибок сети.

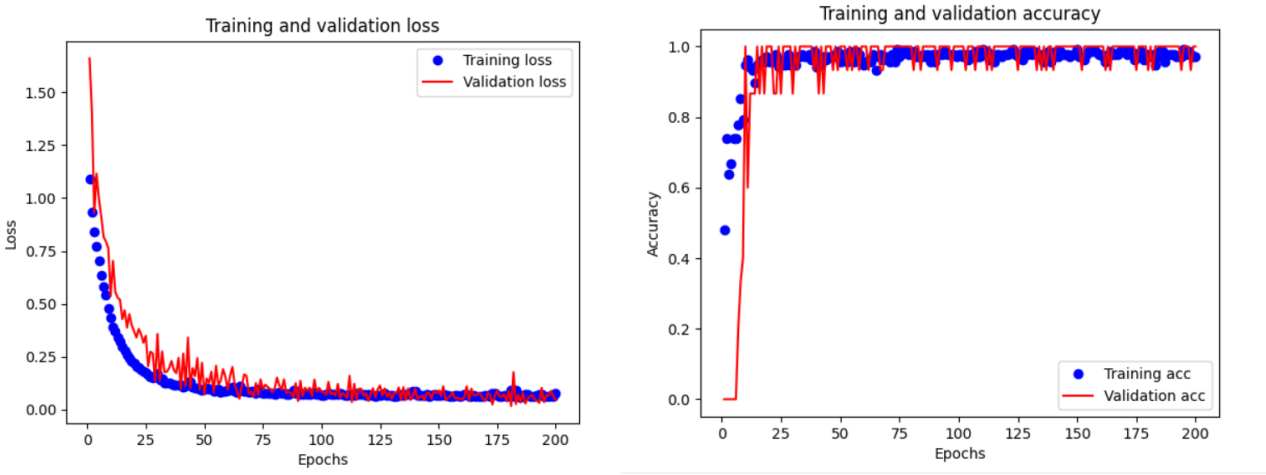


Рисунок 5 – Ошибки и точность сети с размеров батча 8

Затем был изменен параметр, отвечающий за долю валидационных данных. При больших значениях параметра сети оставалось мало данных для обучения и точность обучения резко падает. При совсем маленьких значениях увеличивается качество обучения сети, но при этом данных на валидацию остается очень мало.

Итоговая конфигурация нейронной сети следующая:

```
model = Sequential()
model.add(Dense(30, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(3, activation='softmax'))

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

history = model.fit(X, dummy_y, epochs=200, batch_size=8, validation_split=0.1)
```

Данная модель обеспечивает наибольшую точность и наименьшие потери среди всех рассмотренных вариантов. На рисунке 6 представлены графики точности и ошибок итоговой сети.

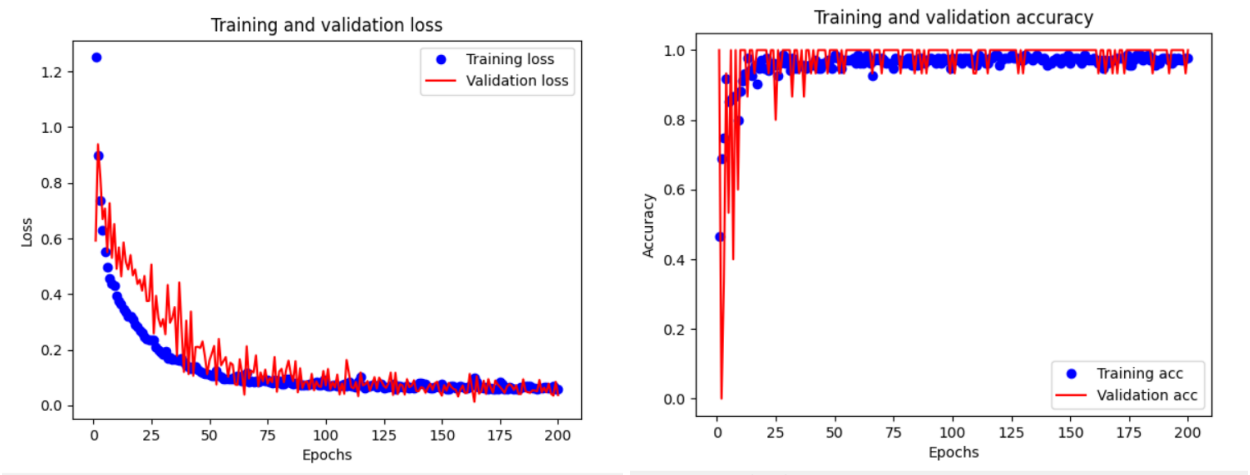


Рисунок 5 – Ошибки и точность итоговой сети

В большинстве случаев такая модель обеспечивает точность свыше 85% уже после 30й эпохи. Полный код программы, реализующий данную нейронную сеть представлен в приложении А.

Выводы

В ходе лабораторной работы было изучено влияние количества слоев и нейронов на скорость и качество обучения, также были исследованы параметры обучения. Была создана, настроена и обучена нейронная сеть, обеспечивающая многоклассовую классификацию цветов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

# Загрузка данных
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]

# Переход от текстовых меток к категориальному вектору
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
dummy_y = to_categorical(encoded_Y)

# Задание архитектуры сети
model = Sequential()
model.add(Dense(30, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Инициализация параметров обучения
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])

# Обучение
history = model.fit(X, dummy_y, epochs=200, batch_size=8, validation_split=0.1)

#Получение ошибки и точности в процессе обучения
loss = history.history['loss']
val_loss = history.history['val_loss']
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(loss) + 1)

#Построение графика ошибки
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#Построение графика точности
plt.clf()
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```