Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Лабораторная работа №2
Дисциплина «Разработка интеллектуальных систем»
Вариант 1

Выполнил:
Съестов Дмитрий Вячеславович
Группа P3417

Преподаватель:
Жукова Наталия Александровна

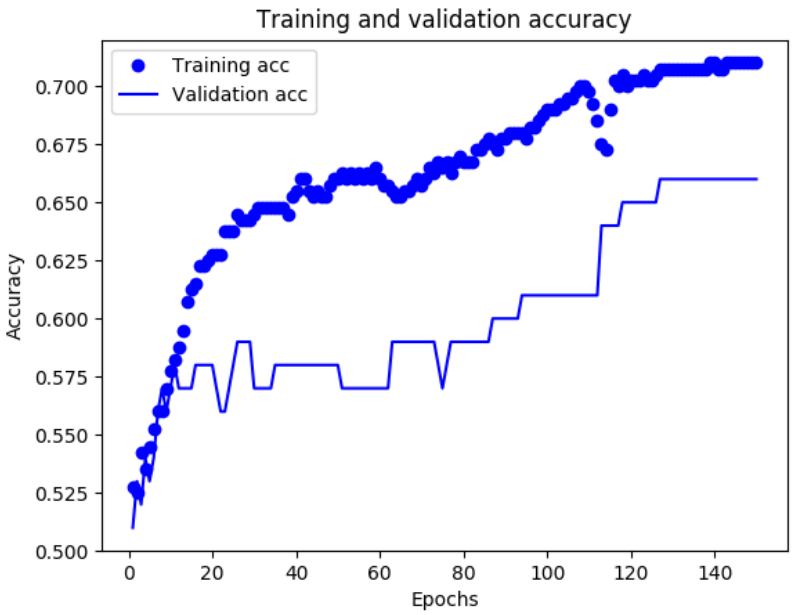
Санкт-Петербург
2019

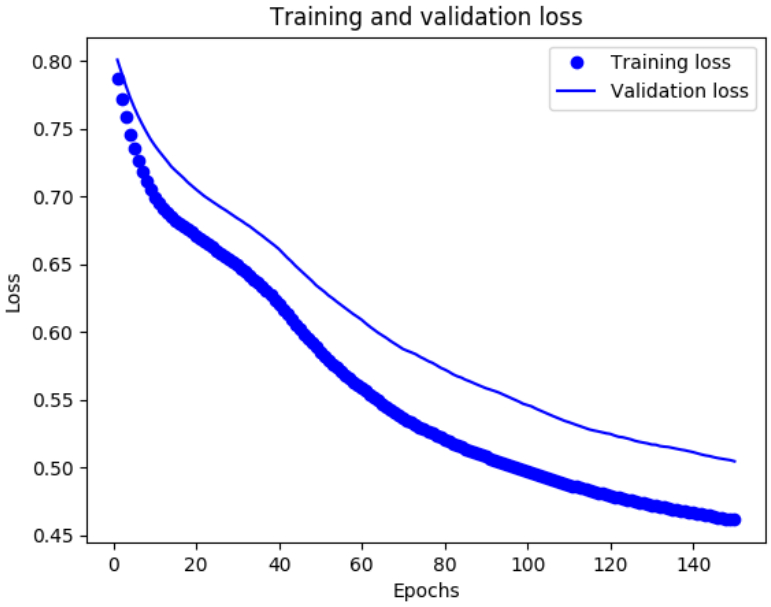
Листинг программы

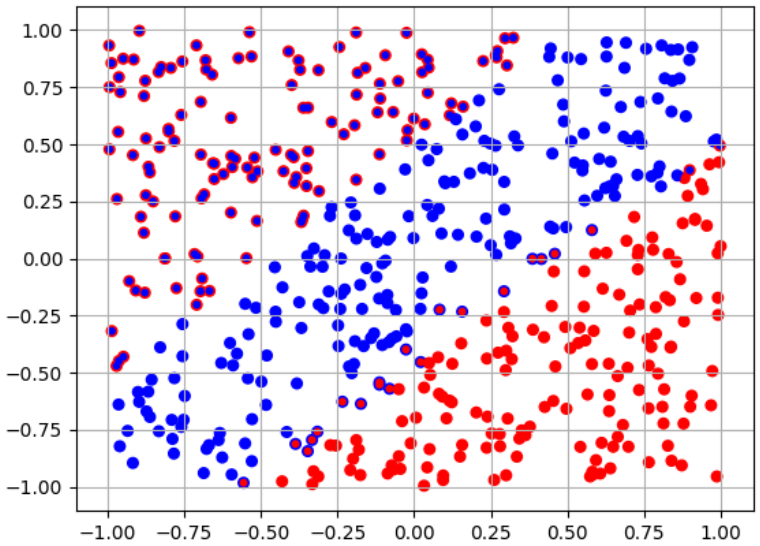
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mclr
from tensorflow.keras import layers
from tensorflow.keras import models


def genData(size=500):
   data = np.random.rand(size, 2)*2 - 1
   label = np.zeros([size, 1])

   for i, p in enumerate(data):
       if (p[0] + 0.5 >= p[1]) and (p[0] - 0.5 <= p[1]):
           label[i] = 1.0
       else:
           label[i] = 0.0

   div = round(size*0.8)
   train_data = data[:div, :]
   test_data = data[div:, :]
   train_label = label[:div, :]
   test_label = label[div:, :]

   return (train_data, train_label), (test_data, test_label)


def drawResults(data, label, prediction):
   p_label = np.array([round(x[0]) for x in prediction])
   plt.clf()
   plt.scatter(data[:, 0], data[:, 1], s=30, c=label[:, 0],
cmap=mclr.ListedColormap(['red', 'blue']))
   plt.scatter(data[:, 0], data[:, 1], s=10, c=p_label,
cmap=mclr.ListedColormap(['red', 'blue']))
   plt.grid()
   plt.show()
plt.savefig('plot.png')


(train_data, train_label), (test_data, test_label) = genData()

model = models.Sequential()
model.add(layers.Dense(2, activation='relu'))
#model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

H = model.fit(train_data, train_label, epochs=150, batch_size=10,
validation_data=(test_data, test_label))

loss = H.history['loss']
val_loss = H.history['val_loss']
acc = H.history['accuracy']
val_acc = H.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, 'b', label='Validation loss')

plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()
plt.savefig('err.png')

#Построение графика точности
plt.clf()
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.legend()
plt.show()
plt.savefig('acc.png')

#Получение и вывод результатов на тестовом наборе
results = model.evaluate(test_data, test_label)
print(results)

#Вывод результатов бинарной классификации
all_data = np.vstack((train_data, test_data))
all_label = np.vstack((train_label, test_label))
pred = model.predict(all_data)

drawResults(all_data, all_label, pred)
```

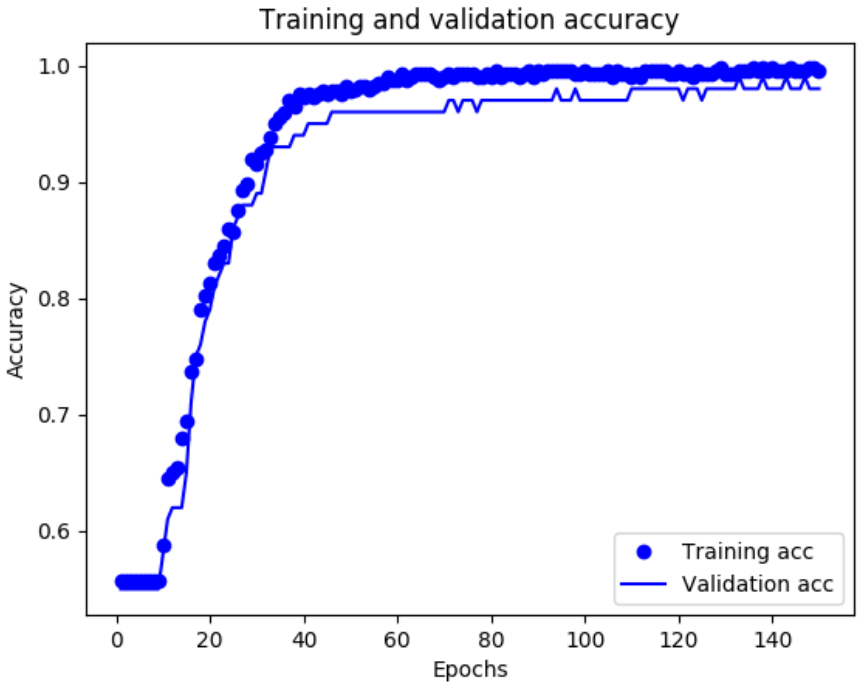
Без промежуточного слоя

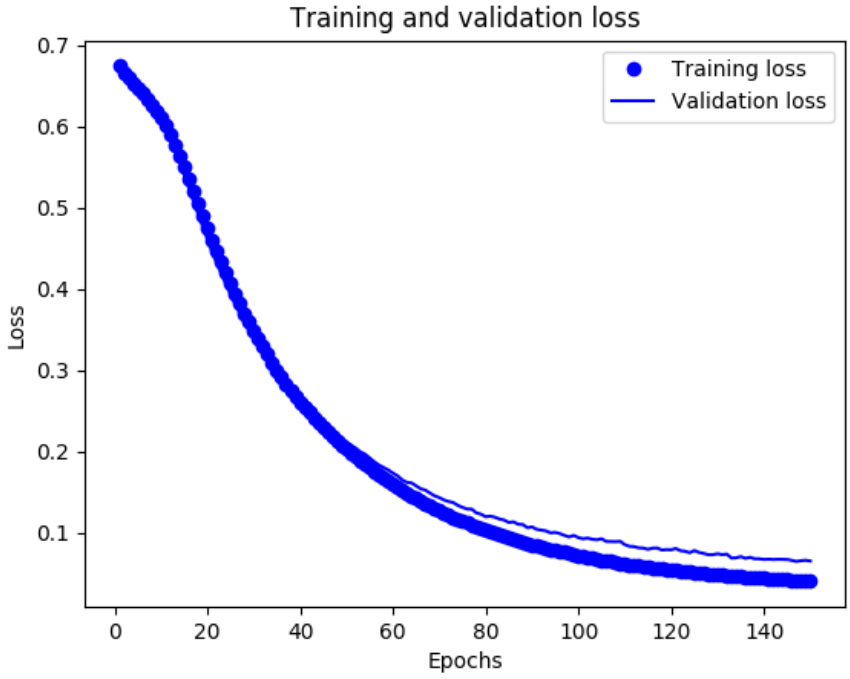


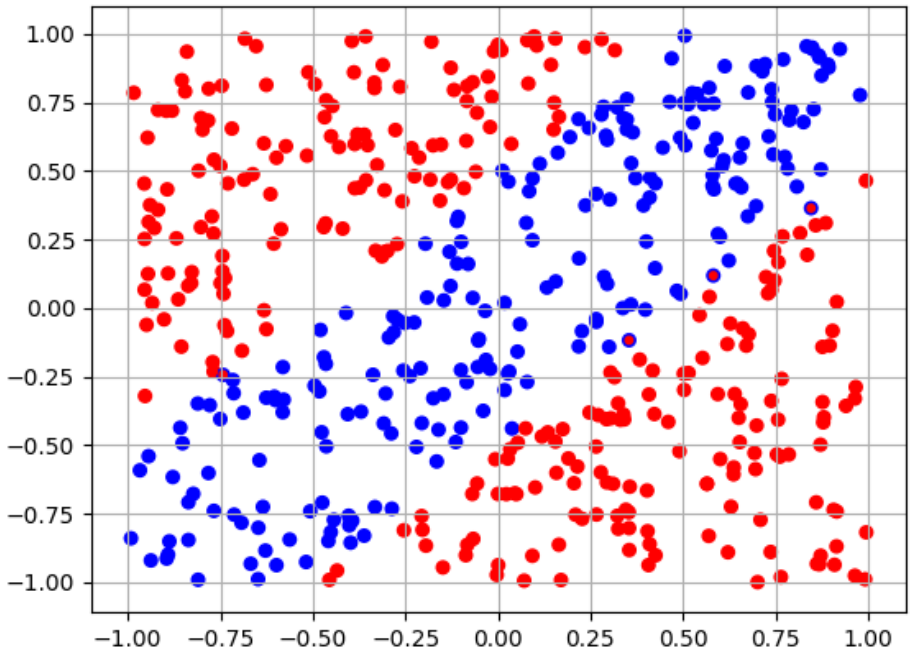

Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0
20
40
60
80
100
120
140


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

С промежуточным слоем







Вывод

Как видно по графикам, при добавлении в модель промежуточного слоя существенно возросла точность и снизились потери как на обучающих, так и на тестовых данных.