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:**

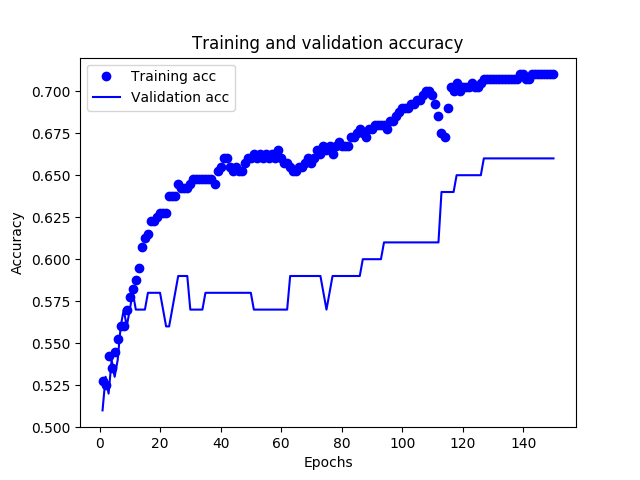
Жукова Наталия Александровна

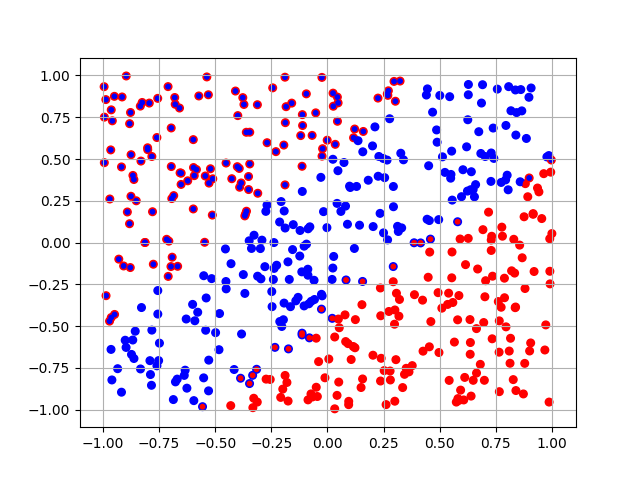
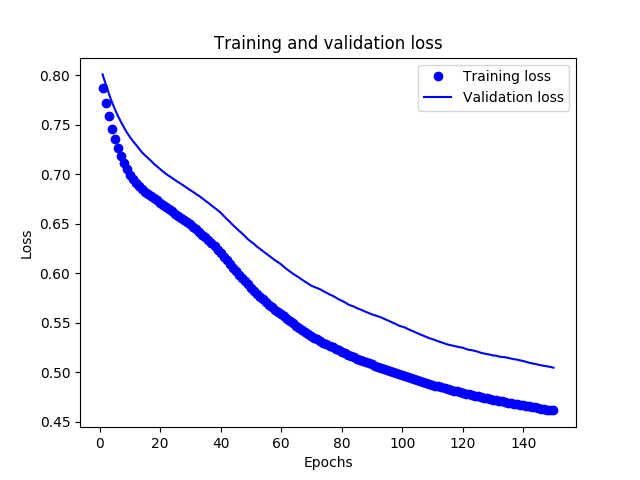
Санкт-Петербург

2019

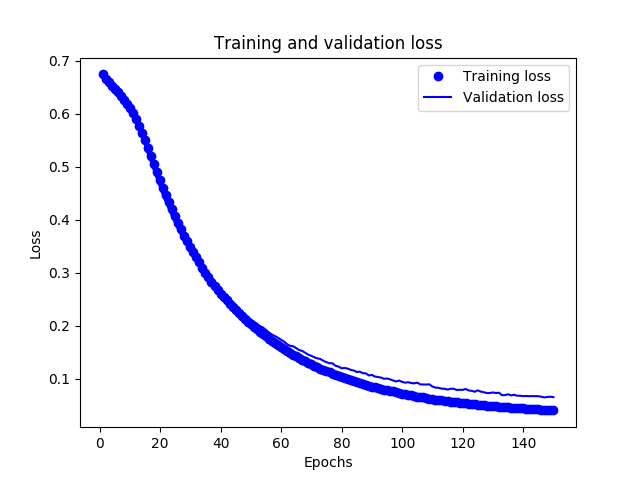
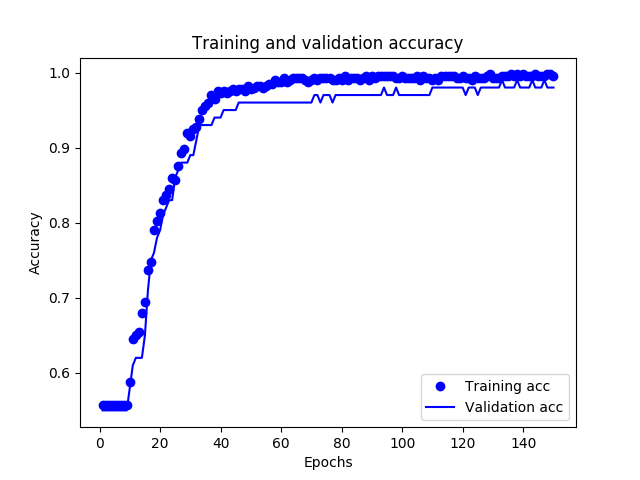
**Листинг программы**

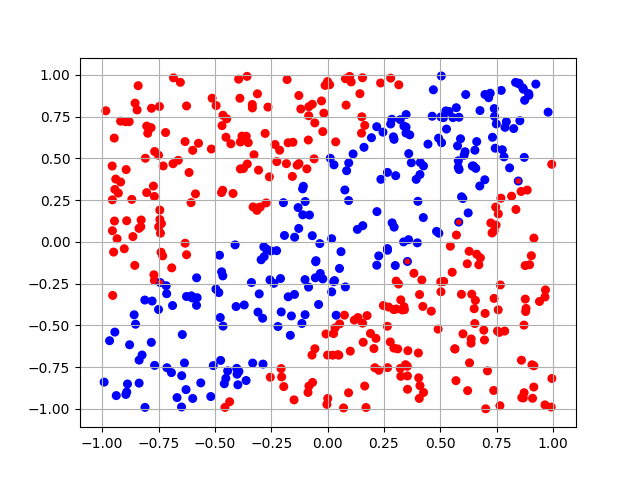
|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import matplotlib.colors as mclr  from tensorflow.keras import layers  from tensorflow.keras import models    def genData(size=500):     data = np.random.rand(size, 2)\*2 - 1     label = np.zeros([size, 1])      for i, p in enumerate(data):         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**



**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.