aml-assignment-1

February 20, 2024

1 ** Enhancing Neural Network Performance on the IMDb Dataset**

Abstract: This project is dedicated to investigating diverse methods for enhancing the efficacy of a neural network model when applied to the IMDb dataset. We aim to refine an existing neural network model and contrast the outcomes of various strategies, including adjustments to the number of hidden layers, units, loss function, activation function, and the integration of regularization techniques such as dropout.

Dataset: The IMDb dataset, comprising movie reviews tagged as positive or negative, was utilized for this study. With 25,000 movie reviews designated for training and an additional 25,000 for testing, the dataset served as a robust foundation for our experiments.

[ ]:
```
from google.colab import drive
drive.mount("/content/drive")
```

[83]:
```
from numpy.random import seed
seed(151)
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

[84]:
```
train_data
```

[84]:
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941,
4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150,
4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4,
192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12,
16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5,
62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130,
12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28,
77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5,
723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381,
15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476,
26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38,
1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26,
4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103,
```

```
21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647,
4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002,
5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26,
6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148,
605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9,
43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228,
8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212,
14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4,
1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]),
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13,
71, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86,
320, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33, 89, 78, 12, 66, 16, 4, 360, 7,
4, 58, 316, 334, 11, 4, 1716, 43, 645, 662, 8, 257, 85, 1200, 42, 1228, 2578,
83, 68, 3912, 15, 36, 165, 1539, 278, 36, 69, 2, 780, 8, 106, 14, 6905, 1338,
18, 6, 22, 12, 215, 28, 610, 40, 6, 87, 326, 23, 2300, 21, 23, 22, 12, 272, 40,
57, 31, 11, 4, 22, 47, 6, 2307, 51, 9, 170, 23, 595, 116, 595, 1352, 13, 191,
79, 638, 89, 2, 14, 9, 8, 106, 607, 624, 35, 534, 6, 227, 7, 129, 113]),
       …,
       list([1, 11, 6, 230, 245, 6401, 9, 6, 1225, 446, 2, 45, 2174, 84, 8322,
4007, 21, 4, 912, 84, 2, 325, 725, 134, 2, 1715, 84, 5, 36, 28, 57, 1099, 21, 8,
140, 8, 703, 5, 2, 84, 56, 18, 1644, 14, 9, 31, 7, 4, 9406, 1209, 2295, 2, 1008,
18, 6, 20, 207, 110, 563, 12, 8, 2901, 2, 8, 97, 6, 20, 53, 4767, 74, 4, 460,
364, 1273, 29, 270, 11, 960, 108, 45, 40, 29, 2961, 395, 11, 6, 4065, 500, 7, 2,
89, 364, 70, 29, 140, 4, 64, 4780, 11, 4, 2678, 26, 178, 4, 529, 443, 2, 5, 27,
710, 117, 2, 8123, 165, 47, 84, 37, 131, 818, 14, 595, 10, 10, 61, 1242, 1209,
10, 10, 288, 2260, 1702, 34, 2901, 2, 4, 65, 496, 4, 231, 7, 790, 5, 6, 320,
234, 2766, 234, 1119, 1574, 7, 496, 4, 139, 929, 2901, 2, 7750, 5, 4241, 18, 4,
8497, 2, 250, 11, 1818, 7561, 4, 4217, 5408, 747, 1115, 372, 1890, 1006, 541,
9303, 7, 4, 59, 2, 4, 3586, 2]),
       list([1, 1446, 7079, 69, 72, 3305, 13, 610, 930, 8, 12, 582, 23, 5, 16,
484, 685, 54, 349, 11, 4120, 2959, 45, 58, 1466, 13, 197, 12, 16, 43, 23, 2, 5,
62, 30, 145, 402, 11, 4131, 51, 575, 32, 61, 369, 71, 66, 770, 12, 1054, 75,
100, 2198, 8, 4, 105, 37, 69, 147, 712, 75, 3543, 44, 257, 390, 5, 69, 263, 514,
105, 50, 286, 1814, 23, 4, 123, 13, 161, 40, 5, 421, 4, 116, 16, 897, 13, 2, 40,
319, 5872, 112, 6700, 11, 4803, 121, 25, 70, 3468, 4, 719, 3798, 13, 18, 31, 62,
40, 8, 7200, 4, 2, 7, 14, 123, 5, 942, 25, 8, 721, 12, 145, 5, 202, 12, 160,
580, 202, 12, 6, 52, 58, 2, 92, 401, 728, 12, 39, 14, 251, 8, 15, 251, 5, 2, 12,
38, 84, 80, 124, 12, 9, 23]),
       list([1, 17, 6, 194, 337, 7, 4, 204, 22, 45, 254, 8, 106, 14, 123, 4, 2,
270, 2, 5, 2, 2, 732, 2098, 101, 405, 39, 14, 1034, 4, 1310, 9, 115, 50, 305,
12, 47, 4, 168, 5, 235, 7, 38, 111, 699, 102, 7, 4, 4039, 9245, 9, 24, 6, 78,
1099, 17, 2345, 2, 21, 27, 9685, 6139, 5, 2, 1603, 92, 1183, 4, 1310, 7, 4, 204,
42, 97, 90, 35, 221, 109, 29, 127, 27, 118, 8, 97, 12, 157, 21, 6789, 2, 9, 6,
66, 78, 1099, 4, 631, 1191, 5, 2642, 272, 191, 1070, 6, 7585, 8, 2197, 2, 2,
544, 5, 383, 1271, 848, 1468, 2, 497, 2, 8, 1597, 8778, 2, 21, 60, 27, 239, 9,
43, 8368, 209, 405, 10, 10, 12, 764, 40, 4, 248, 20, 12, 16, 5, 174, 1791, 72,
7, 51, 6, 1739, 22, 4, 204, 131, 9])],
```

```
      dtype=object)
```

[85]:
```python
train_labels[0]
```

[85]:
```
1
```

[86]:
```python
len(train_labels)
```

[86]:
```
25000
```

[87]:
```python
len(train_labels)
```

[87]:
```
25000
```

[88]:
```python
test_data
```

[88]:
```
array([list([1, 591, 202, 14, 31, 6, 717, 10, 10, 2, 2, 5, 4, 360, 7, 4, 177,
5760, 394, 354, 4, 123, 9, 1035, 1035, 1035, 10, 10, 13, 92, 124, 89, 488, 7944,
100, 28, 1668, 14, 31, 23, 27, 7479, 29, 220, 468, 8, 124, 14, 286, 170, 8, 157,
46, 5, 27, 239, 16, 179, 2, 38, 32, 25, 7944, 451, 202, 14, 6, 717]),
       list([1, 14, 22, 3443, 6, 176, 7, 5063, 88, 12, 2679, 23, 1310, 5, 109,
943, 4, 114, 9, 55, 606, 5, 111, 7, 4, 139, 193, 273, 23, 4, 172, 270, 11, 7216,
2, 4, 8463, 2801, 109, 1603, 21, 4, 22, 3861, 8, 6, 1193, 1330, 10, 10, 4, 105,
987, 35, 841, 2, 19, 861, 1074, 5, 1987, 2, 45, 55, 221, 15, 670, 5304, 526, 14,
1069, 4, 405, 5, 2438, 7, 27, 85, 108, 131, 4, 5045, 5304, 3884, 405, 9, 3523,
133, 5, 50, 13, 104, 51, 66, 166, 14, 22, 157, 9, 4, 530, 239, 34, 8463, 2801,
45, 407, 31, 7, 41, 3778, 105, 21, 59, 299, 12, 38, 950, 5, 4521, 15, 45, 629,
488, 2733, 127, 6, 52, 292, 17, 4, 6936, 185, 132, 1988, 5304, 1799, 488, 2693,
47, 6, 392, 173, 4, 2, 4378, 270, 2352, 4, 1500, 7, 4, 65, 55, 73, 11, 346, 14,
20, 9, 6, 976, 2078, 7, 5293, 861, 2, 5, 4182, 30, 3127, 2, 56, 4, 841, 5, 990,
692, 8, 4, 1669, 398, 229, 10, 10, 13, 2822, 670, 5304, 14, 9, 31, 7, 27, 111,
108, 15, 2033, 19, 7836, 1429, 875, 551, 14, 22, 9, 1193, 21, 45, 4829, 5, 45,
252, 8, 2, 6, 565, 921, 3639, 39, 4, 529, 48, 25, 181, 8, 67, 35, 1732, 22, 49,
238, 60, 135, 1162, 14, 9, 290, 4, 58, 10, 10, 472, 45, 55, 878, 8, 169, 11,
374, 5687, 25, 203, 28, 8, 818, 12, 125, 4, 3077]),
       list([1, 111, 748, 4368, 1133, 2, 2, 4, 87, 1551, 1262, 7, 31, 318, 9459,
7, 4, 498, 5076, 748, 63, 29, 5161, 220, 686, 2, 5, 17, 12, 575, 220, 2507, 17,
6, 185, 132, 2, 16, 53, 928, 11, 2, 74, 4, 438, 21, 27, 2, 589, 8, 22, 107, 2,
2, 997, 1638, 8, 35, 2076, 9019, 11, 22, 231, 54, 29, 1706, 29, 100, 2, 2425,
34, 2, 8738, 2, 5, 2, 98, 31, 2122, 33, 6, 58, 14, 3808, 1638, 8, 4, 365, 7,
2789, 3761, 356, 346, 4, 2, 1060, 63, 29, 93, 11, 5421, 11, 2, 33, 6, 58, 54,
1270, 431, 748, 7, 32, 2580, 16, 11, 94, 2, 10, 10, 4, 993, 2, 7, 4, 1766, 2634,
2164, 2, 8, 847, 8, 1450, 121, 31, 7, 27, 86, 2663, 2, 16, 6, 465, 993, 2006, 2,
573, 17, 2, 42, 4, 2, 37, 473, 6, 711, 6, 8869, 7, 328, 212, 70, 30, 258, 11,
220, 32, 7, 108, 21, 133, 12, 9, 55, 465, 849, 3711, 53, 33, 2071, 1969, 37, 70,
1144, 4, 5940, 1409, 74, 476, 37, 62, 91, 1329, 169, 4, 1330, 2, 146, 655, 2212,
5, 258, 12, 184, 2, 546, 5, 849, 2, 7, 4, 22, 1436, 18, 631, 1386, 797, 7, 4,
```

```
8712, 71, 348, 425, 4320, 1061, 19, 2, 5, 2, 11, 661, 8, 339, 2, 4, 2455, 2, 7,
4, 1962, 10, 10, 263, 787, 9, 270, 11, 6, 9466, 4, 2, 2, 121, 4, 5437, 26, 4434,
19, 68, 1372, 5, 28, 446, 6, 318, 7149, 8, 67, 51, 36, 70, 81, 8, 4392, 2294,
36, 1197, 8, 2, 2, 18, 6, 711, 4, 9909, 26, 2, 1125, 11, 14, 636, 720, 12, 426,
28, 77, 776, 8, 97, 38, 111, 7489, 6175, 168, 1239, 5189, 137, 2, 18, 27, 173,
9, 2399, 17, 6, 2, 428, 2, 232, 11, 4, 8014, 37, 272, 40, 2708, 247, 30, 656, 6,
2, 54, 2, 3292, 98, 6, 2840, 40, 558, 37, 6093, 98, 4, 2, 1197, 15, 14, 9, 57,
4893, 5, 4659, 6, 275, 711, 7937, 2, 3292, 98, 6, 2, 10, 10, 6639, 19, 14, 2,
267, 162, 711, 37, 5900, 752, 98, 4, 2, 2378, 90, 19, 6, 2, 7, 2, 1810, 2, 4,
4770, 3183, 930, 8, 508, 90, 4, 1317, 8, 4, 2, 17, 2, 3965, 1853, 4, 1494, 8,
4468, 189, 4, 2, 6287, 5774, 4, 4770, 5, 95, 271, 23, 6, 7742, 6063, 2, 5437,
33, 1526, 6, 425, 3155, 2, 4535, 1636, 7, 4, 4669, 2, 469, 4, 4552, 54, 4, 150,
5664, 2, 280, 53, 2, 2, 18, 339, 29, 1978, 27, 7885, 5, 2, 68, 1830, 19, 6571,
2, 4, 1515, 7, 263, 65, 2132, 34, 6, 5680, 7489, 43, 159, 29, 9, 4706, 9, 387,
73, 195, 584, 10, 10, 1069, 4, 58, 810, 54, 14, 6078, 117, 22, 16, 93, 5, 1069,
4, 192, 15, 12, 16, 93, 34, 6, 1766, 2, 33, 4, 5673, 7, 15, 2, 9252, 3286, 325,
12, 62, 30, 776, 8, 67, 14, 17, 6, 2, 44, 148, 687, 2, 203, 42, 203, 24, 28, 69,
2, 6676, 11, 330, 54, 29, 93, 2, 21, 845, 2, 27, 1099, 7, 819, 4, 22, 1407, 17,
6, 2, 787, 7, 2460, 2, 2, 100, 30, 4, 3737, 3617, 3169, 2321, 42, 1898, 11, 4,
3814, 42, 101, 704, 7, 101, 999, 15, 1625, 94, 2926, 180, 5, 9, 9101, 34, 2, 45,
6, 1429, 22, 60, 6, 1220, 31, 11, 94, 6408, 96, 21, 94, 749, 9, 57, 975]),
       …,
       list([1, 13, 1408, 15, 8, 135, 14, 9, 35, 32, 46, 394, 20, 62, 30, 5093,
21, 45, 184, 78, 4, 1492, 910, 769, 2290, 2515, 395, 4257, 5, 1454, 11, 119, 2,
89, 1036, 4, 116, 218, 78, 21, 407, 100, 30, 128, 262, 15, 7, 185, 2280, 284,
1842, 2, 37, 315, 4, 226, 20, 272, 2942, 40, 29, 152, 60, 181, 8, 30, 50, 553,
362, 80, 119, 12, 21, 846, 5518]),
       list([1, 11, 119, 241, 9, 4, 840, 20, 12, 468, 15, 94, 3684, 562, 791,
39, 4, 86, 107, 8, 97, 14, 31, 33, 4, 2960, 7, 743, 46, 1028, 9, 3531, 5, 4,
768, 47, 8, 79, 90, 145, 164, 162, 50, 6, 501, 119, 7, 9, 4, 78, 232, 15, 16,
224, 11, 4, 333, 20, 4, 985, 200, 5, 2, 5, 9, 1861, 8, 79, 357, 4, 20, 47, 220,
57, 206, 139, 11, 12, 5, 55, 117, 212, 13, 1276, 92, 124, 51, 45, 1188, 71, 536,
13, 520, 14, 20, 6, 2302, 7, 470]),
       list([1, 6, 52, 7465, 430, 22, 9, 220, 2594, 8, 28, 2, 519, 3227, 6, 769,
15, 47, 6, 3482, 4067, 8, 114, 5, 33, 222, 31, 55, 184, 704, 5586, 2, 19, 346,
3153, 5, 6, 364, 350, 4, 184, 5586, 9, 133, 1810, 11, 5417, 2, 21, 4, 7298, 2,
570, 50, 2005, 2643, 9, 6, 1249, 17, 6, 2, 2, 21, 17, 6, 1211, 232, 1138, 2249,
29, 266, 56, 96, 346, 194, 308, 9, 194, 21, 29, 218, 1078, 19, 4, 78, 173, 7,
27, 2, 5698, 3406, 718, 2, 9, 6, 6907, 17, 210, 5, 3281, 5677, 47, 77, 395, 14,
172, 173, 18, 2740, 2931, 4517, 82, 127, 27, 173, 11, 6, 392, 217, 21, 50, 9,
57, 65, 12, 2, 53, 40, 35, 390, 7, 11, 4, 3567, 7, 4, 314, 74, 6, 792, 22, 2,
19, 714, 727, 5205, 382, 4, 91, 6533, 439, 19, 14, 20, 9, 1441, 5805, 1118, 4,
756, 25, 124, 4, 31, 12, 16, 93, 804, 34, 2005, 2643])],
      dtype=object)
```

[89]:
```
test_labels[0]
```

[89]: 0

```
[90]: max([max(sequence) for sequence in test_data])
```

[90]: 9999

Deciphering Textual Reviews

```
[91]: word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = " ".join(
    [reverse_word_index.get(i - 2, "?") for i in train_data[0]])
```

```
[92]: decoded_review
```

```
[92]: "? that on as about parts admit ready speaking really care boot see holy and
again who each a are any about brought life what power ? br they sound
everything a though and part life look ? fan recommend like and part elegant
successful for feeling from this based and take what as of those core movie that
on and manage airplane 4 and on me because i as about parts from been was this
military and on for kill for i as cinematography with ? a which let i is left is
two a and seat raises as sound see worried by and still i as from running a are
off good who scene some are church by of on i come he bad more a that gives as
into ? is and films best commenting was each and ? to rid a beyond who me about
parts final his keep special has to and ? manages this characters how and
perhaps was american too at references no his something of enough russ with and
bit on film say final his sound a back one jews with good who he there's made
are characters and bit really as from harry how i as actor a as transfer plot
think at was as inexplicably movie quite at"
```

Data preparation

```
[93]: import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 2.
    return results
```

Data Vectorization

```
[94]: x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

```
[95]: x_train[0]
```

[95]: array([0., 2., 2., …, 0., 0., 0.])

[96]:
```
x_test[0]
```

[96]: array([0., 2., 2., …, 0., 0., 0.])

Label Vectorization

[97]:
```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

Constructing a model utilizing the rectified linear unit (ReLU) and then compiling it.

[98]:
```
from tensorflow import keras
from tensorflow.keras import layers
seed(151)
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

[99]:
```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

[100]:
```
seed(151)
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[101]:
```
seed(151)
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))

hist_dict = history.history
hist_dict.keys()
```

```
Epoch 1/20
29/29 [==============================] - 2s 33ms/step - loss: 0.5034 - accuracy:
0.7654 - val_loss: 0.3910 - val_accuracy: 0.8471
Epoch 2/20
29/29 [==============================] - 0s 8ms/step - loss: 0.2972 - accuracy:
0.8969 - val_loss: 0.2967 - val_accuracy: 0.8865
```

```
Epoch 3/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2236 - accuracy:
0.9252 - val_loss: 0.2781 - val_accuracy: 0.8899
Epoch 4/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1779 - accuracy:
0.9397 - val_loss: 0.2846 - val_accuracy: 0.8843
Epoch 5/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1492 - accuracy:
0.9521 - val_loss: 0.3078 - val_accuracy: 0.8805
Epoch 6/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1249 - accuracy:
0.9591 - val_loss: 0.3343 - val_accuracy: 0.8747
Epoch 7/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1007 - accuracy:
0.9702 - val_loss: 0.3225 - val_accuracy: 0.8829
Epoch 8/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0884 - accuracy:
0.9749 - val_loss: 0.3332 - val_accuracy: 0.8775
Epoch 9/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0728 - accuracy:
0.9800 - val_loss: 0.3956 - val_accuracy: 0.8705
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0655 - accuracy:
0.9819 - val_loss: 0.3643 - val_accuracy: 0.8809
Epoch 11/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0481 - accuracy:
0.9896 - val_loss: 0.3926 - val_accuracy: 0.8783
Epoch 12/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0434 - accuracy:
0.9895 - val_loss: 0.4114 - val_accuracy: 0.8720
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0385 - accuracy:
0.9903 - val_loss: 0.4243 - val_accuracy: 0.8770
Epoch 14/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0286 - accuracy:
0.9947 - val_loss: 0.4478 - val_accuracy: 0.8762
Epoch 15/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0248 - accuracy:
0.9952 - val_loss: 0.4703 - val_accuracy: 0.8740
Epoch 16/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0183 - accuracy:
0.9977 - val_loss: 0.5039 - val_accuracy: 0.8698
Epoch 17/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0213 - accuracy:
0.9961 - val_loss: 0.5190 - val_accuracy: 0.8753
Epoch 18/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0127 - accuracy:
0.9985 - val_loss: 0.7494 - val_accuracy: 0.8494
```

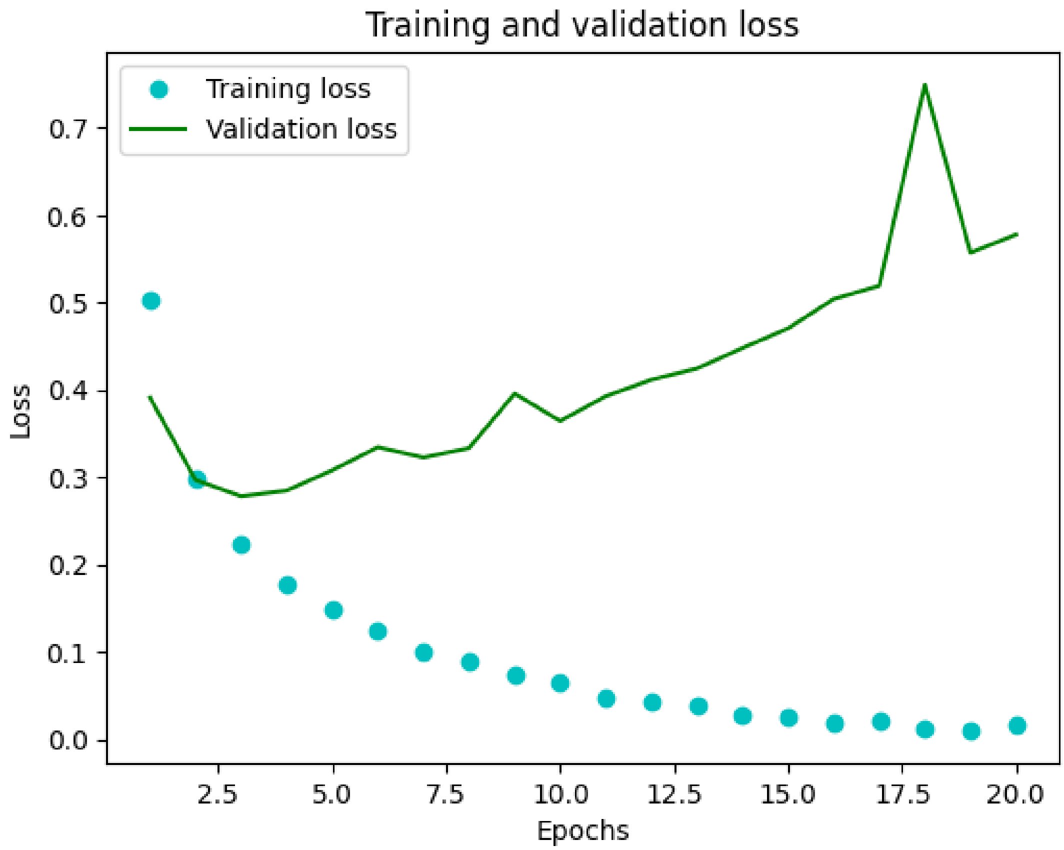
```
Epoch 19/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0101 - accuracy:
0.9989 - val_loss: 0.5570 - val_accuracy: 0.8725
Epoch 20/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0155 - accuracy:
0.9959 - val_loss: 0.5775 - val_accuracy: 0.8723
```

[101]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

1. The training began with a loss of 0.5034 and an accuracy of 0.7654 on the training set, and a validation loss of 0.1504 with a validation accuracy of 0.8471.
2. As the training progressed, the model's performance on the training set continued to improve, reaching a loss of 0.0155 and an accuracy of 0.9959 by the 20th epoch. However, on the validation set, the model achieved a loss of 0.5775 and an accuracy of 0.8723, indicating signs of overfitting to the training data.

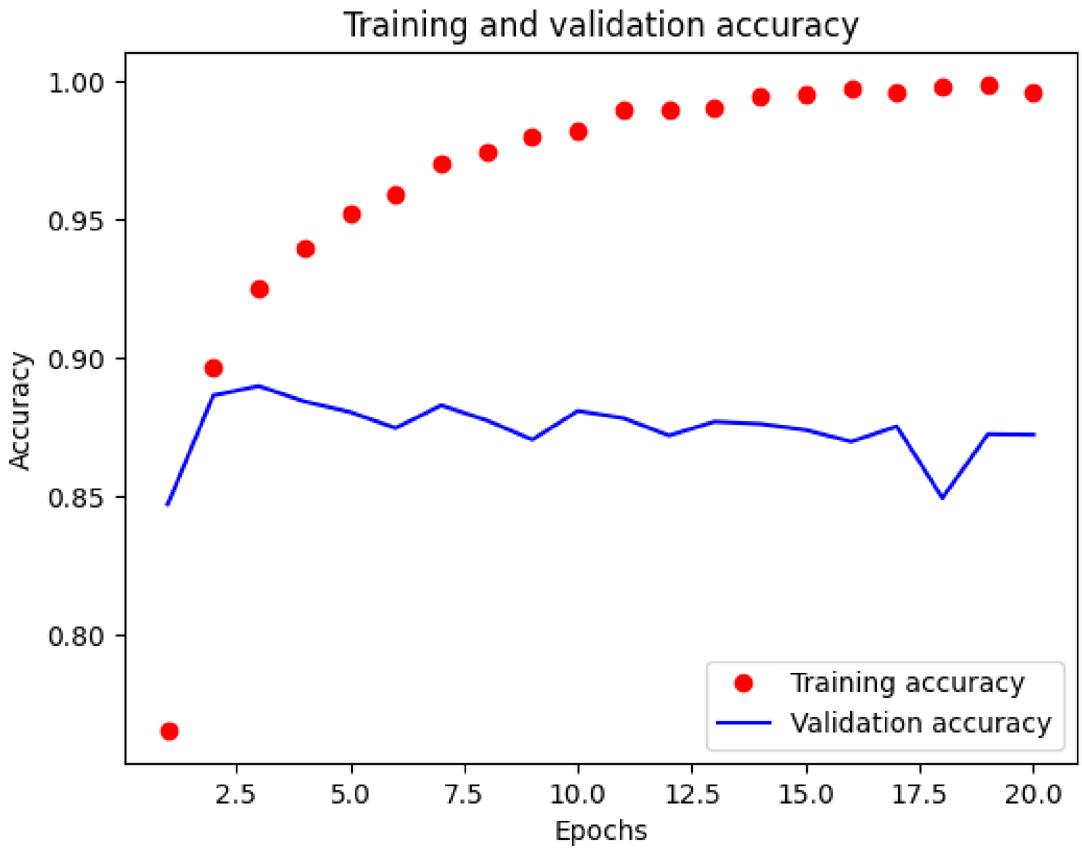
Plotting the training and validation loss

[102]:
```python
import matplotlib.pyplot as plt
hist_dict = history.history
loss_values = hist_dict["loss"]
val_loss_values = hist_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "co", label="Training loss")
plt.plot(epochs, val_loss_values, "g", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[103]:

```
plt.clf()
acc = hist_dict["accuracy"]
val_acc = hist_dict["val_accuracy"]
plt.plot(epochs, acc, "ro", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



- The visual representations indicate that the model's ability to accurately predict new data diminishes after a certain number of epochs, indicating overfitting to the training data. It may be beneficial to conduct additional analysis, such as modifying the model's hyperparameters or implementing regularization techniques, to enhance its overall performance.

Retraining the model

[104]:
```
np.random.seed(151)
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=525)
results = model.evaluate(x_test, y_test)
results
```

```
Epoch 1/4
```

```
48/48 [==============================] - 1s 6ms/step - loss: 0.4582 - accuracy:
0.8061
Epoch 2/4
48/48 [==============================] - 0s 5ms/step - loss: 0.2557 - accuracy:
0.9093
Epoch 3/4
48/48 [==============================] - 0s 6ms/step - loss: 0.1992 - accuracy:
0.9266
Epoch 4/4
48/48 [==============================] - 0s 5ms/step - loss: 0.1653 - accuracy:
0.9410
782/782 [==============================] - 1s 2ms/step - loss: 0.3663 -
accuracy: 0.8534
```

[104]:
```
[0.36632782220840454, 0.85343998670578]
```

- The neural network model has obtained an 85.34% accuracy on the test dataset, with a corresponding loss value of 0.3663.

[105]:
```python
model.predict(x_test)
```

```
782/782 [==============================] - 1s 1ms/step
```

[105]:
```
array([[0.13846359],
       [0.99989885],
       [0.5332834 ],
       …,
       [0.09695407],
       [0.05167303],
       [0.448192  ]], dtype=float32)
```

Building a neural network with 1 hidden layer

[106]:
```python
seed(151)
model1 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model1.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
history1 = model1.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
29/29 [==============================] - 1s 32ms/step - loss: 0.4822 - accuracy:
0.7842 - val_loss: 0.3757 - val_accuracy: 0.8570
Epoch 2/20
29/29 [==============================] - 0s 8ms/step - loss: 0.2913 - accuracy:
0.9026 - val_loss: 0.3032 - val_accuracy: 0.8855
Epoch 3/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2274 - accuracy:
0.9238 - val_loss: 0.3562 - val_accuracy: 0.8508
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1884 - accuracy:
0.9379 - val_loss: 0.2823 - val_accuracy: 0.8855
Epoch 5/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1588 - accuracy:
0.9509 - val_loss: 0.2876 - val_accuracy: 0.8855
Epoch 6/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1423 - accuracy:
0.9547 - val_loss: 0.2831 - val_accuracy: 0.8857
Epoch 7/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1208 - accuracy:
0.9655 - val_loss: 0.3050 - val_accuracy: 0.8774
Epoch 8/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1108 - accuracy:
0.9677 - val_loss: 0.2982 - val_accuracy: 0.8816
Epoch 9/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0979 - accuracy:
0.9729 - val_loss: 0.3156 - val_accuracy: 0.8773
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0878 - accuracy:
0.9784 - val_loss: 0.3412 - val_accuracy: 0.8709
Epoch 11/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0792 - accuracy:
0.9809 - val_loss: 0.3327 - val_accuracy: 0.8770
Epoch 12/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0714 - accuracy:
0.9829 - val_loss: 0.3626 - val_accuracy: 0.8774
Epoch 13/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0631 - accuracy:
0.9861 - val_loss: 0.4039 - val_accuracy: 0.8692
```

```
Epoch 14/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0560 - accuracy:
0.9898 - val_loss: 0.3687 - val_accuracy: 0.8743
Epoch 15/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0535 - accuracy:
0.9895 - val_loss: 0.3997 - val_accuracy: 0.8741
Epoch 16/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0463 - accuracy:
0.9920 - val_loss: 0.3913 - val_accuracy: 0.8737
Epoch 17/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0426 - accuracy:
0.9927 - val_loss: 0.4082 - val_accuracy: 0.8723
Epoch 18/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0391 - accuracy:
0.9936 - val_loss: 0.4237 - val_accuracy: 0.8719
Epoch 19/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0341 - accuracy:
0.9955 - val_loss: 0.4426 - val_accuracy: 0.8720
Epoch 20/20
29/29 [==============================] - 0s 7ms/step - loss: 0.0315 - accuracy:
0.9962 - val_loss: 0.4476 - val_accuracy: 0.8729
```

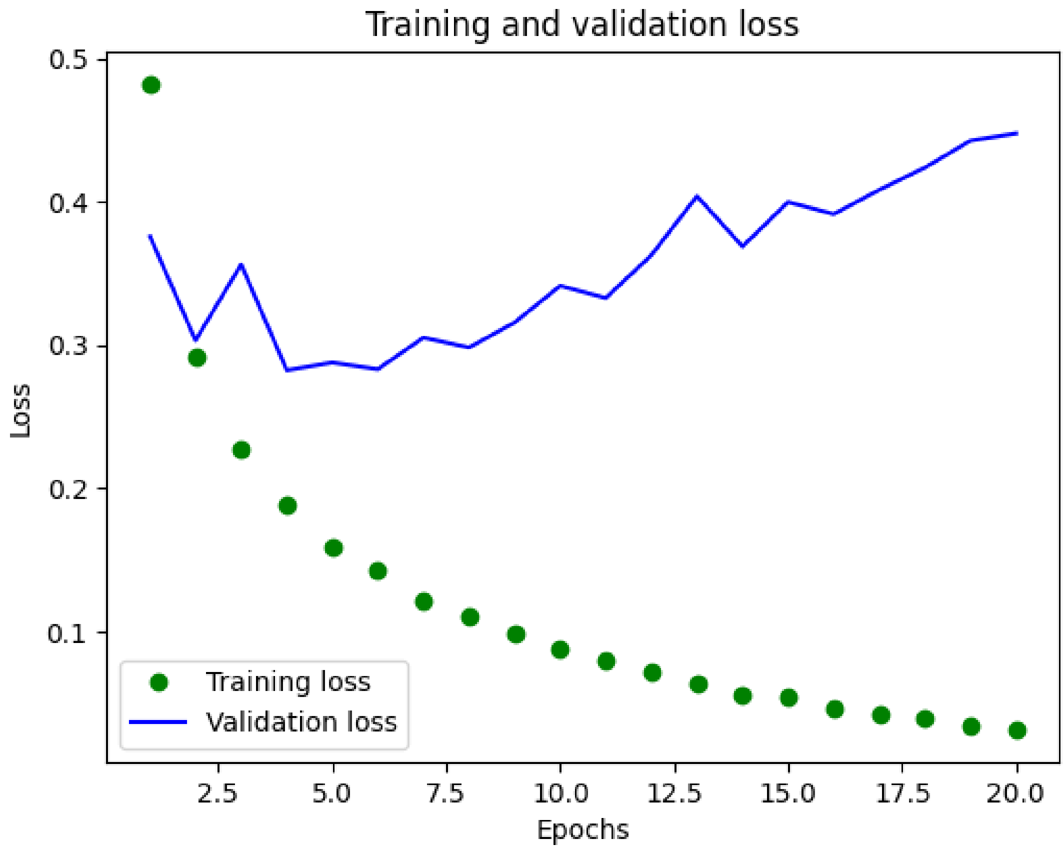
[107]:
```
hist_dict = history1.history
hist_dict.keys()
```

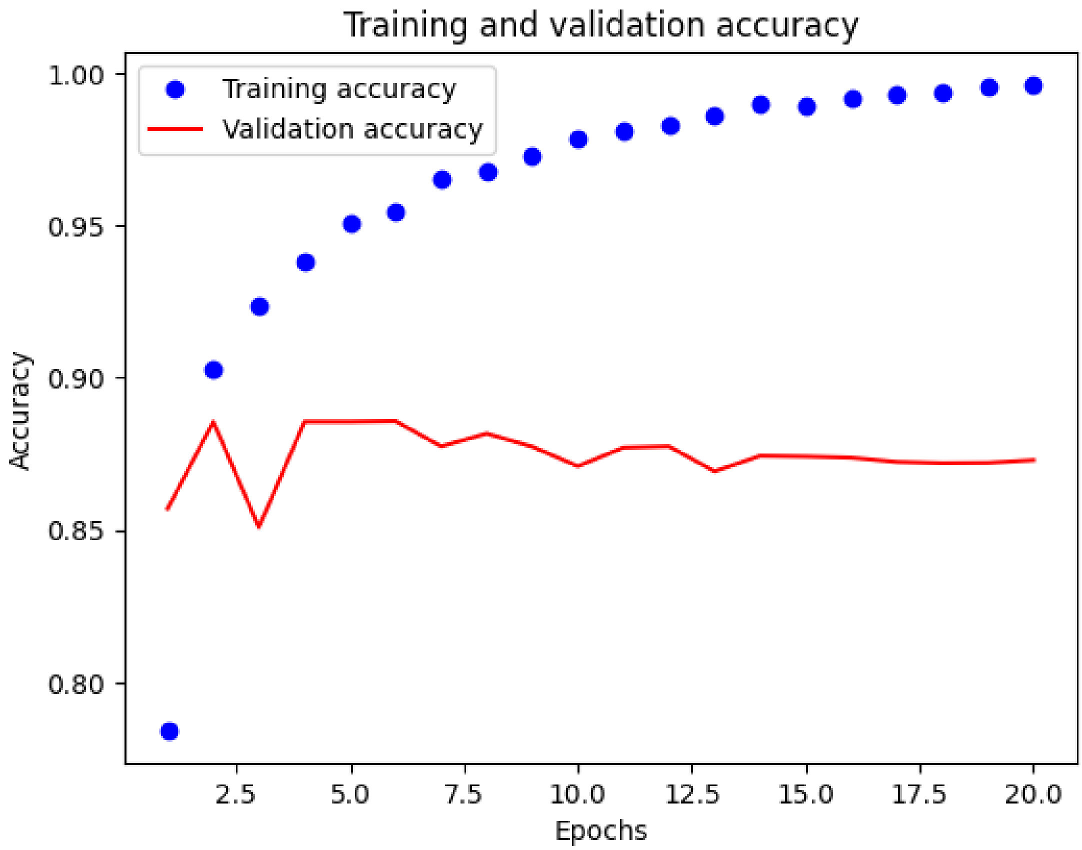
[107]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[108]:
```
import matplotlib.pyplot as plt
hist_dict = history1.history
loss_values = hist_dict["loss"]
val_loss_values = hist_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
#Plotting graph between Training and Validation loss
plt.plot(epochs, loss_values, "go", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

#Plotting graph between Training and Validation Accuracy
plt.clf()
acc = hist_dict["accuracy"]
val_acc = hist_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "r", label="Validation accuracy")
```

```
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```





[109]:

```
np.random.seed(151)
model1 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model1.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model1.fit(x_train, y_train, epochs=5, batch_size=525)
results1 = model1.evaluate(x_test, y_test)
```

```
Epoch 1/5
48/48 [==============================] - 1s 5ms/step - loss: 0.4264 - accuracy:
0.8143
Epoch 2/5
48/48 [==============================] - 0s 6ms/step - loss: 0.2541 - accuracy:
0.9061
Epoch 3/5
48/48 [==============================] - 0s 5ms/step - loss: 0.2002 - accuracy:
```

```
0.9291
Epoch 4/5
48/48 [==============================] - 0s 5ms/step - loss: 0.1771 - accuracy:
0.9357
Epoch 5/5
48/48 [==============================] - 0s 5ms/step - loss: 0.1547 - accuracy:
0.9466
782/782 [==============================] - 1s 1ms/step - loss: 0.3046 -
accuracy: 0.8797
```

[110]:
```
results1
```

[110]:
```
[0.30455276370048523, 0.8797199726104736]
```

- The loss on the test set is 0.3045, and the accuracy is 87.97%.

[111]:
```
model1.predict(x_test)
```

```
782/782 [==============================] - 1s 1ms/step
```

[111]:
```
array([[0.22547168],
       [0.9999687 ],
       [0.9002679 ],
       …,
       [0.21891184],
       [0.07732808],
       [0.6713103 ]], dtype=float32)
```

Building a neural network with 3 hidden layers

[112]:
```
np.random.seed(151)
model_3 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_3.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]

history3 = model_3.fit(partial_x_train,
                    partial_y_train,
```

```
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
```

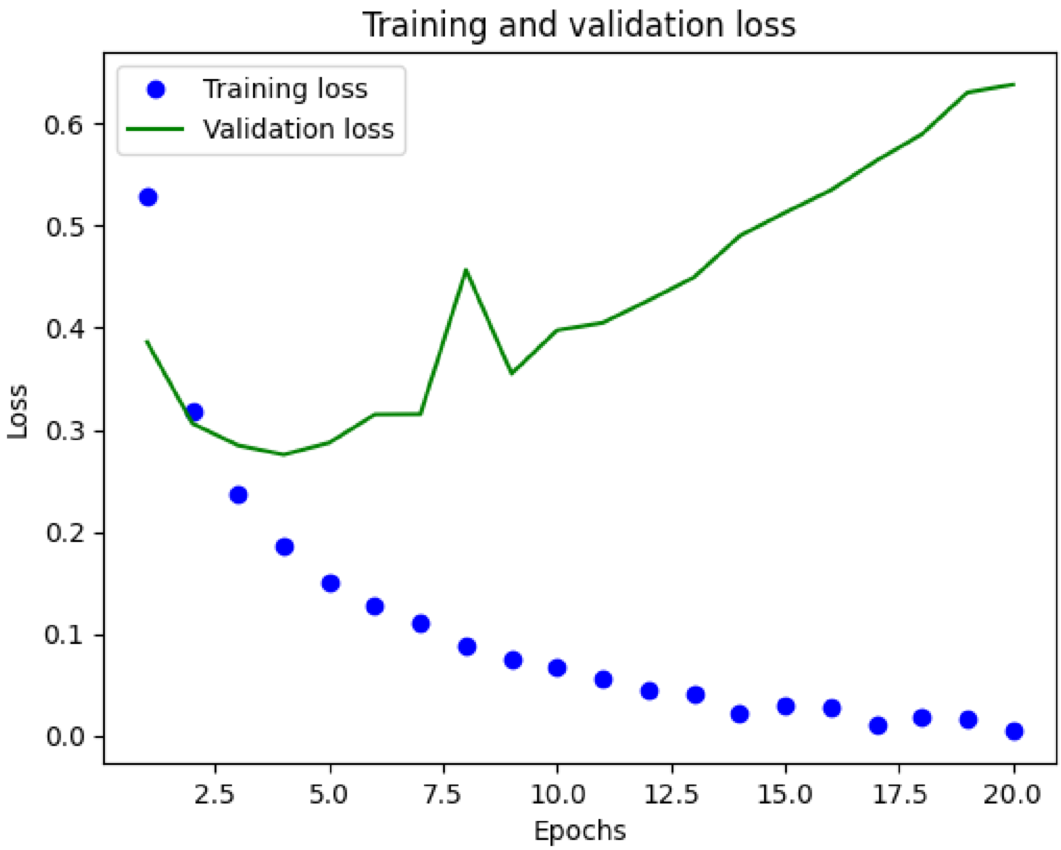
```
Epoch 1/20
29/29 [==============================] - 2s 31ms/step - loss: 0.5299 - accuracy:
0.7615 - val_loss: 0.3862 - val_accuracy: 0.8673
Epoch 2/20
29/29 [==============================] - 0s 9ms/step - loss: 0.3187 - accuracy:
0.8903 - val_loss: 0.3061 - val_accuracy: 0.8864
Epoch 3/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2373 - accuracy:
0.9198 - val_loss: 0.2847 - val_accuracy: 0.8889
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1867 - accuracy:
0.9367 - val_loss: 0.2757 - val_accuracy: 0.8895
Epoch 5/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1510 - accuracy:
0.9489 - val_loss: 0.2873 - val_accuracy: 0.8830
Epoch 6/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1272 - accuracy:
0.9575 - val_loss: 0.3151 - val_accuracy: 0.8777
Epoch 7/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1106 - accuracy:
0.9631 - val_loss: 0.3154 - val_accuracy: 0.8843
Epoch 8/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0875 - accuracy:
0.9736 - val_loss: 0.4568 - val_accuracy: 0.8570
Epoch 9/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0758 - accuracy:
0.9768 - val_loss: 0.3551 - val_accuracy: 0.8778
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0673 - accuracy:
0.9801 - val_loss: 0.3977 - val_accuracy: 0.8785
Epoch 11/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0557 - accuracy:
0.9844 - val_loss: 0.4049 - val_accuracy: 0.8751
Epoch 12/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0446 - accuracy:
0.9885 - val_loss: 0.4268 - val_accuracy: 0.8763
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0407 - accuracy:
0.9899 - val_loss: 0.4495 - val_accuracy: 0.8760
Epoch 14/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0231 - accuracy:
0.9967 - val_loss: 0.4900 - val_accuracy: 0.8690
Epoch 15/20
```

```
29/29 [==============================] - 0s 9ms/step - loss: 0.0306 - accuracy:
0.9922 - val_loss: 0.5130 - val_accuracy: 0.8736
Epoch 16/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0285 - accuracy:
0.9921 - val_loss: 0.5348 - val_accuracy: 0.8714
Epoch 17/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0113 - accuracy:
0.9993 - val_loss: 0.5638 - val_accuracy: 0.8718
Epoch 18/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0192 - accuracy:
0.9953 - val_loss: 0.5897 - val_accuracy: 0.8711
Epoch 19/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0171 - accuracy:
0.9952 - val_loss: 0.6308 - val_accuracy: 0.8717
Epoch 20/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0054 - accuracy:
0.9998 - val_loss: 0.6385 - val_accuracy: 0.8708
```

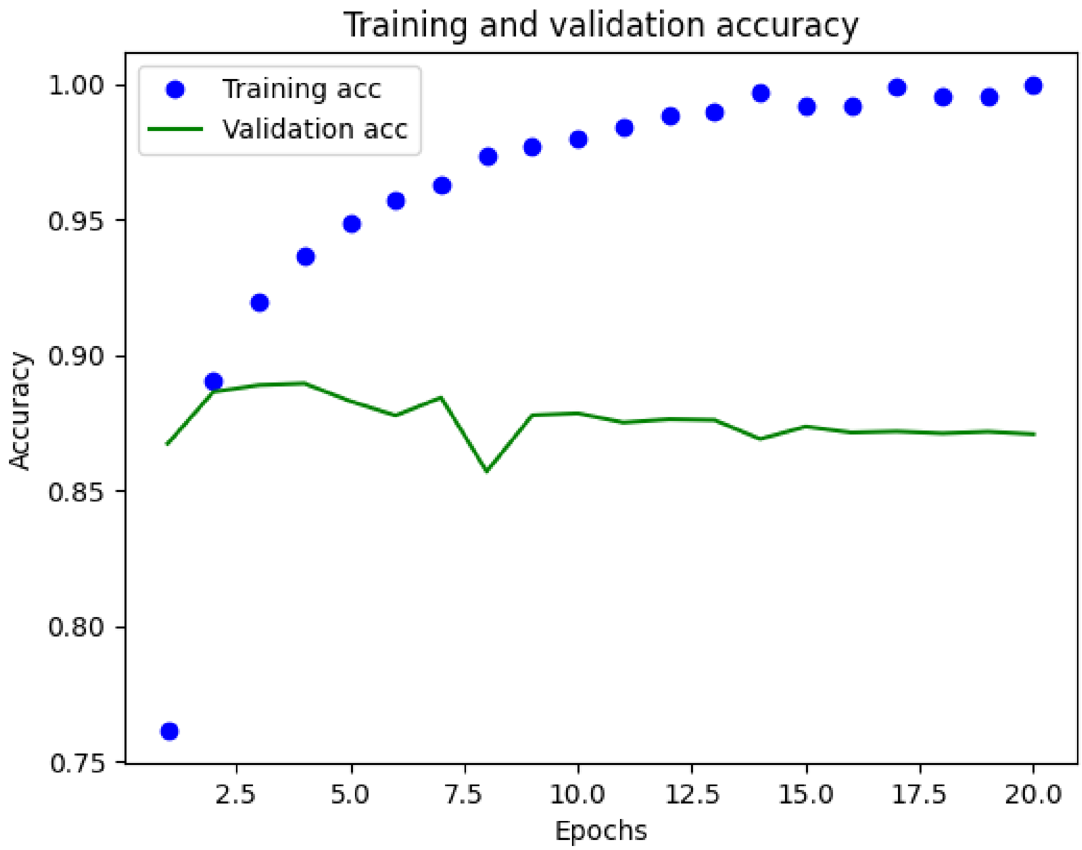
[113]:
```python
hist_dict3 = history3.history
hist_dict3.keys()
```

[113]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[114]:
```python
loss_values = hist_dict3["loss"]
val_loss_values = hist_dict3["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "g", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[115]:
```
plt.clf()
acc = hist_dict3["accuracy"]
val_acc = hist_dict3["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "g", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[116]:
```
np.random.seed(151)
model_3 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])


model_3.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model_3.fit(x_train, y_train, epochs=3, batch_size=525)
results_3 = model_3.evaluate(x_test, y_test)
```

```
Epoch 1/3
48/48 [==============================] - 1s 6ms/step - loss: 0.4684 - accuracy:
0.7813
Epoch 2/3
```

```
48/48 [==============================] - 0s 6ms/step - loss: 0.2642 - accuracy:
0.9000
Epoch 3/3
48/48 [==============================] - 0s 5ms/step - loss: 0.2042 - accuracy:
0.9245
782/782 [==============================] - 1s 1ms/step - loss: 0.3319 -
accuracy: 0.8668
```

- The loss on the test set is 0.3318, and the accuracy is 86.67%.

[117]:
```python
results_3
```

[117]:
```
[0.3318694829940796, 0.8667600154876709]
```

[118]:
```python
model_3.predict(x_test)
```

```
782/782 [==============================] - 1s 1ms/step
```

[118]:
```
array([[0.15202591],
       [0.9992959 ],
       [0.35136637],
       …,
       [0.10837806],
       [0.05962013],
       [0.13206384]], dtype=float32)
```

- Changing the number of layers does not notably enhance the model's accuracy, although the three-layer model demonstrates superior accuracy compared to the others. When determining the overall structure of your neural network, the number of units in the hidden layers must be carefully selected. Despite not directly interfacing with the external environment, these layers significantly influence the final outcome.

Building Neural Network with 32 units.

[119]:
```python
np.random.seed(151)
model_32 = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
#model compilation
model_32.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
#model validation
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
```

```python
partial_y_train = y_train[10000:]

np.random.seed(123)
history32 = model_32.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
```

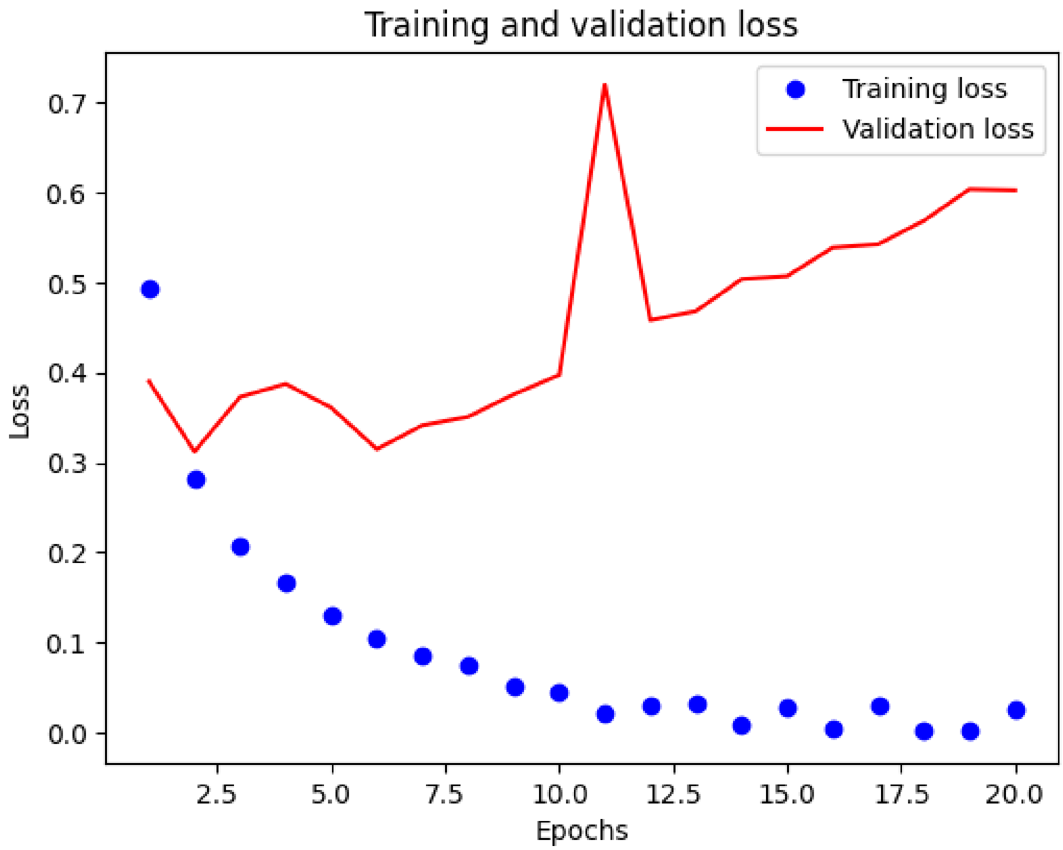
```
Epoch 1/20
29/29 [==============================] - 1s 31ms/step - loss: 0.4934 - accuracy:
0.7719 - val_loss: 0.3900 - val_accuracy: 0.8394
Epoch 2/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2825 - accuracy:
0.8953 - val_loss: 0.3120 - val_accuracy: 0.8732
Epoch 3/20
29/29 [==============================] - 0s 10ms/step - loss: 0.2079 - accuracy:
0.9265 - val_loss: 0.3728 - val_accuracy: 0.8486
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1660 - accuracy:
0.9399 - val_loss: 0.3872 - val_accuracy: 0.8509
Epoch 5/20
29/29 [==============================] - 0s 11ms/step - loss: 0.1309 - accuracy:
0.9568 - val_loss: 0.3608 - val_accuracy: 0.8670
Epoch 6/20
29/29 [==============================] - 0s 11ms/step - loss: 0.1036 - accuracy:
0.9665 - val_loss: 0.3148 - val_accuracy: 0.8840
Epoch 7/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0850 - accuracy:
0.9740 - val_loss: 0.3412 - val_accuracy: 0.8818
Epoch 8/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0752 - accuracy:
0.9772 - val_loss: 0.3507 - val_accuracy: 0.8824
Epoch 9/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0511 - accuracy:
0.9863 - val_loss: 0.3756 - val_accuracy: 0.8792
Epoch 10/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0449 - accuracy:
0.9869 - val_loss: 0.3972 - val_accuracy: 0.8780
Epoch 11/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0205 - accuracy:
0.9973 - val_loss: 0.7201 - val_accuracy: 0.8270
Epoch 12/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0307 - accuracy:
0.9917 - val_loss: 0.4581 - val_accuracy: 0.8740
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0328 - accuracy:
```

```
0.9899 - val_loss: 0.4682 - val_accuracy: 0.8770
Epoch 14/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0075 - accuracy:
0.9997 - val_loss: 0.5037 - val_accuracy: 0.8735
Epoch 15/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0276 - accuracy:
0.9913 - val_loss: 0.5071 - val_accuracy: 0.8788
Epoch 16/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0043 - accuracy:
0.9999 - val_loss: 0.5388 - val_accuracy: 0.8761
Epoch 17/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0307 - accuracy:
0.9917 - val_loss: 0.5426 - val_accuracy: 0.8776
Epoch 18/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0027 - accuracy:
1.0000 - val_loss: 0.5687 - val_accuracy: 0.8773
Epoch 19/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 0.6037 - val_accuracy: 0.8748
Epoch 20/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0262 - accuracy:
0.9916 - val_loss: 0.6027 - val_accuracy: 0.8748
```

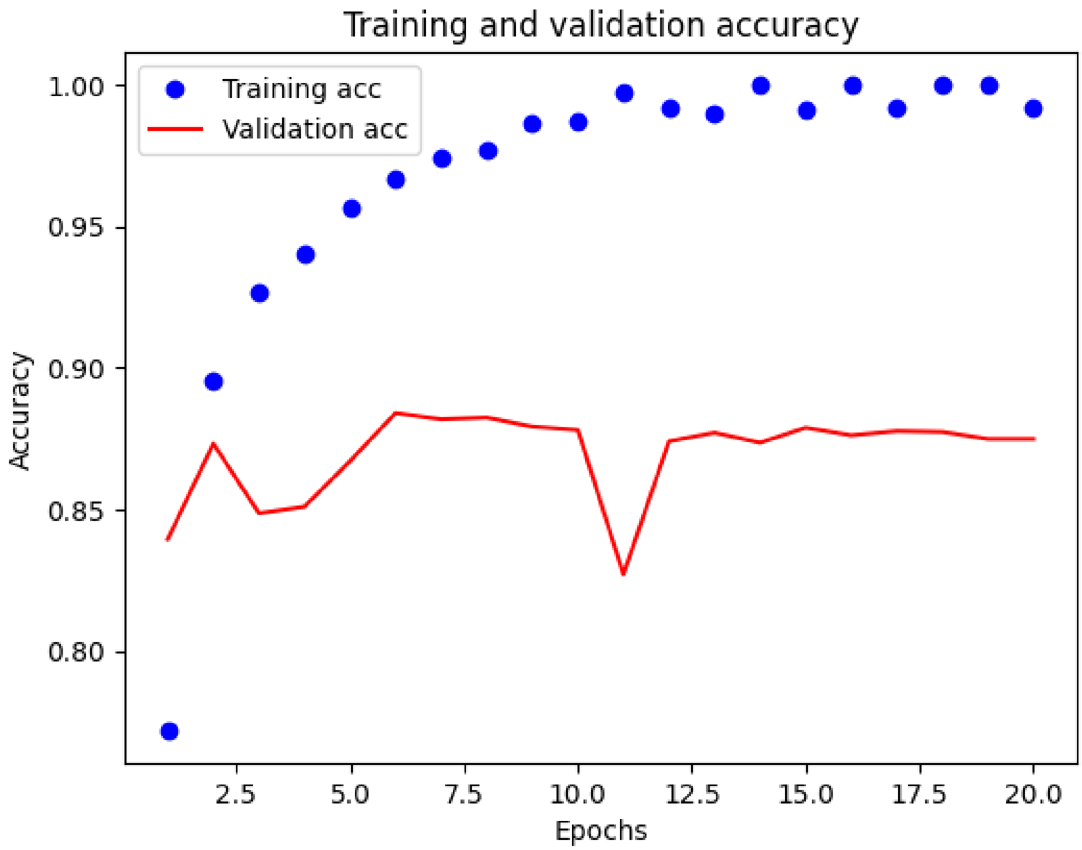
[120]:
```python
hist_dict32 = history32.history
hist_dict32.keys()
```

[120]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[121]:
```python
loss_values = hist_dict32["loss"]
val_loss_values = hist_dict32["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[122]:
```
plt.clf()
acc = hist_dict32["accuracy"]
val_acc = hist_dict32["val_accuracy"]
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()
```



[123]:
```
history_32 = model_32.fit(x_train, y_train, epochs=3, batch_size=525)
results_32 = model_32.evaluate(x_test, y_test)
results_32
```

```
Epoch 1/3
48/48 [==============================] - 0s 7ms/step - loss: 0.1897 - accuracy:
0.9466
Epoch 2/3
48/48 [==============================] - 0s 7ms/step - loss: 0.1004 - accuracy:
0.9692
Epoch 3/3
48/48 [==============================] - 0s 6ms/step - loss: 0.0675 - accuracy:
0.9788
782/782 [==============================] - 1s 2ms/step - loss: 0.4369 -
accuracy: 0.8666
```

[123]:
```
[0.4368942081928253, 0.8666399717330933]
```

[124]:
```
model_32.predict(x_test)
```

```
782/782 [==============================] - 1s 1ms/step
```

[124]:
```
array([[0.02879145],
       [0.99999994],
       [0.8684288 ],
       …,
       [0.0552201 ],
       [0.0410776 ],
       [0.9360502 ]], dtype=float32)
```

- The accuracy on the validation set is 86.66%

Traing the model with 64 units

[125]:
```
np.random.seed(151)
model_64 = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_64.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]

np.random.seed(151)
history64 = model_64.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
```

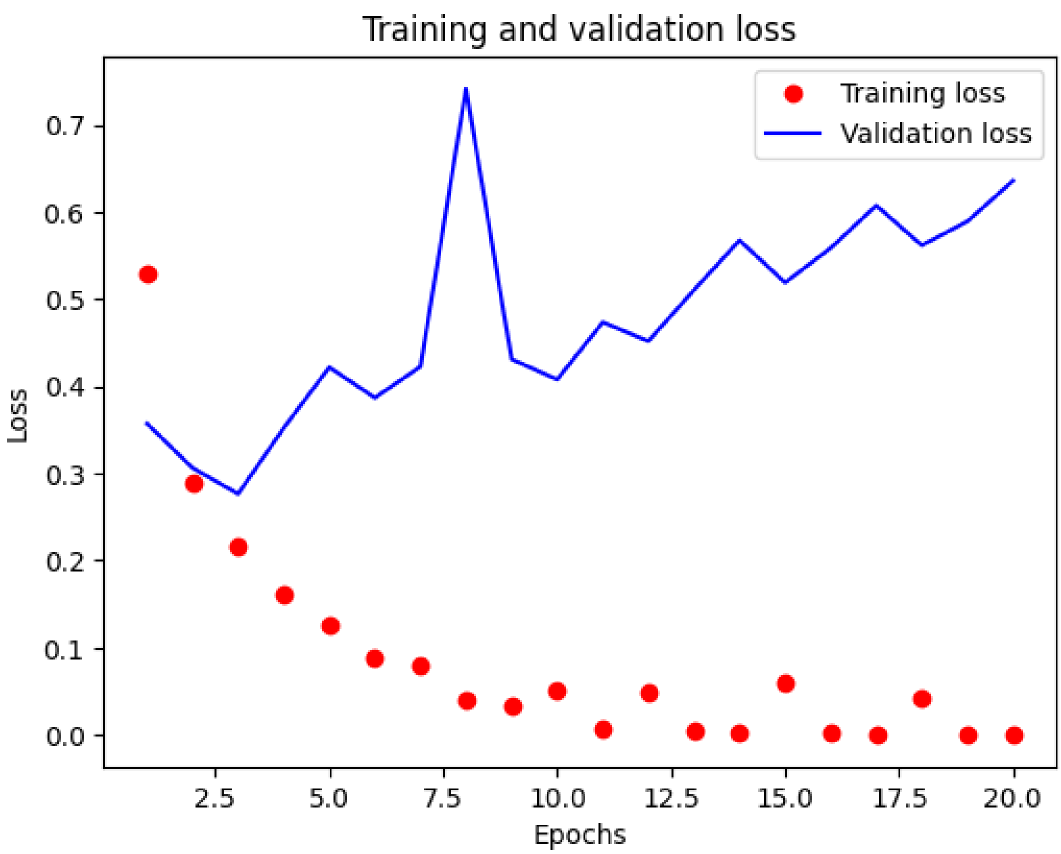
```
Epoch 1/20
29/29 [==============================] - 2s 34ms/step - loss: 0.5301 - accuracy:
0.7361 - val_loss: 0.3572 - val_accuracy: 0.8602
Epoch 2/20
29/29 [==============================] - 0s 13ms/step - loss: 0.2896 - accuracy:
0.8881 - val_loss: 0.3063 - val_accuracy: 0.8745
Epoch 3/20
29/29 [==============================] - 0s 14ms/step - loss: 0.2174 - accuracy:
0.9179 - val_loss: 0.2766 - val_accuracy: 0.8846
Epoch 4/20
29/29 [==============================] - 0s 13ms/step - loss: 0.1623 - accuracy:
0.9405 - val_loss: 0.3522 - val_accuracy: 0.8685
```

```
Epoch 5/20
29/29 [==============================] - 0s 13ms/step - loss: 0.1265 - accuracy:
0.9531 - val_loss: 0.4218 - val_accuracy: 0.8368
Epoch 6/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0890 - accuracy:
0.9705 - val_loss: 0.3869 - val_accuracy: 0.8748
Epoch 7/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0803 - accuracy:
0.9717 - val_loss: 0.4225 - val_accuracy: 0.8644
Epoch 8/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0409 - accuracy:
0.9906 - val_loss: 0.7409 - val_accuracy: 0.7775
Epoch 9/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0343 - accuracy:
0.9907 - val_loss: 0.4310 - val_accuracy: 0.8677
Epoch 10/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0503 - accuracy:
0.9855 - val_loss: 0.4078 - val_accuracy: 0.8804
Epoch 11/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0082 - accuracy:
0.9995 - val_loss: 0.4734 - val_accuracy: 0.8802
Epoch 12/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0499 - accuracy:
0.9861 - val_loss: 0.4517 - val_accuracy: 0.8782
Epoch 13/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0041 - accuracy:
0.9999 - val_loss: 0.5103 - val_accuracy: 0.8777
Epoch 14/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0025 - accuracy:
1.0000 - val_loss: 0.5673 - val_accuracy: 0.8784
Epoch 15/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0605 - accuracy:
0.9865 - val_loss: 0.5189 - val_accuracy: 0.8779
Epoch 16/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0018 - accuracy:
1.0000 - val_loss: 0.5585 - val_accuracy: 0.8793
Epoch 17/20
29/29 [==============================] - 0s 14ms/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 0.6073 - val_accuracy: 0.8801
Epoch 18/20
29/29 [==============================] - 0s 15ms/step - loss: 0.0430 - accuracy:
0.9879 - val_loss: 0.5616 - val_accuracy: 0.8737
Epoch 19/20
29/29 [==============================] - 0s 13ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 0.5891 - val_accuracy: 0.8778
Epoch 20/20
29/29 [==============================] - 0s 13ms/step - loss: 8.0046e-04 -
accuracy: 1.0000 - val_loss: 0.6357 - val_accuracy: 0.8778
```

[126]:
```python
hist_dict64 = history64.history
hist_dict64.keys()
```

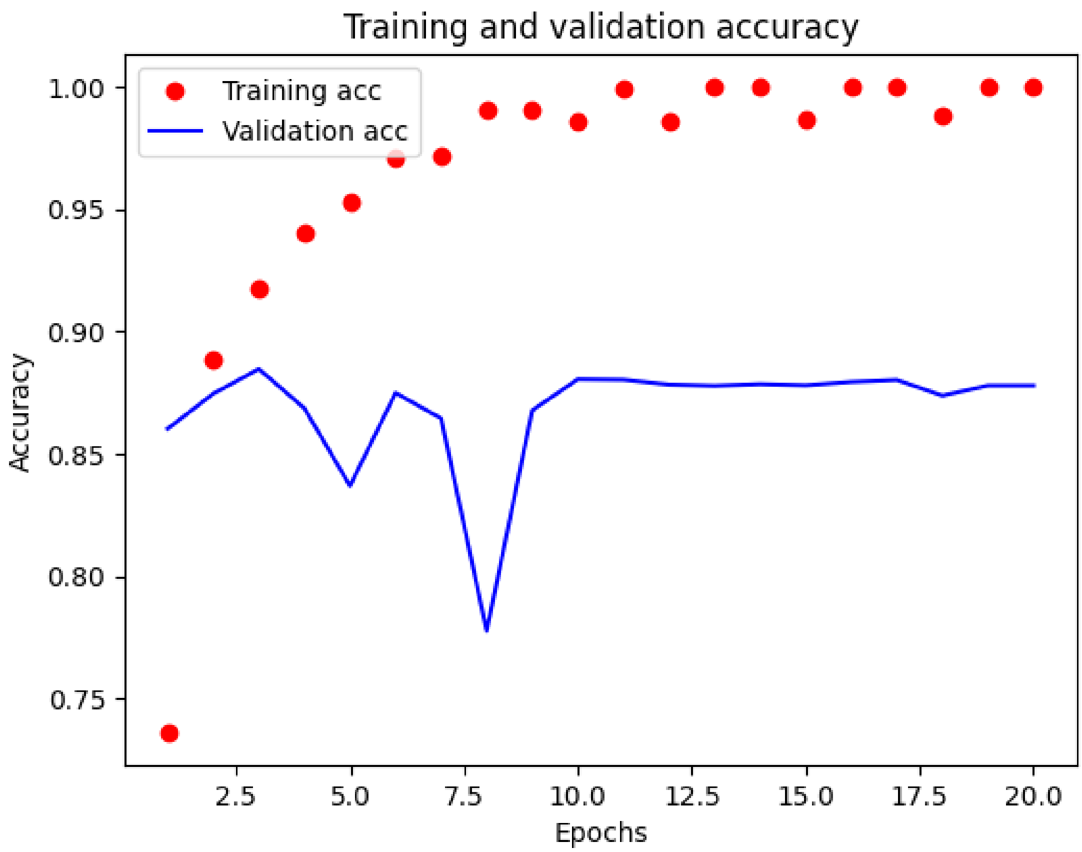
[126]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

[127]:
```python
loss_values = hist_dict64["loss"]
val_loss_values = hist_dict64["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "ro", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[128]:
```python
plt.clf()
acc = hist_dict64["accuracy"]
val_acc = hist_dict64["val_accuracy"]
plt.plot(epochs, acc, "ro", 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()
```



[129]:

```
history_64 = model_64.fit(x_train, y_train, epochs=3, batch_size=525)
results_64 = model_64.evaluate(x_test, y_test)
results_64
```

```
Epoch 1/3
48/48 [==============================] - 0s 9ms/step - loss: 0.1833 - accuracy:
0.9458
Epoch 2/3
48/48 [==============================] - 0s 9ms/step - loss: 0.0875 - accuracy:
0.9711
Epoch 3/3
48/48 [==============================] - 0s 9ms/step - loss: 0.0441 - accuracy:
0.9872
```

```
782/782 [==============================] - 1s 2ms/step - loss: 0.4238 -
accuracy: 0.8712
```

[129]:
```
[0.42381522059440613, 0.8712000250816345]
```

[130]:
```
model_64.predict(x_test)
```

```
782/782 [==============================] - 1s 2ms/step
```

[130]:
```
array([[0.02735085],
       [1.        ],
       [0.8862462 ],
       …,
       [0.01080568],
       [0.01272339],
       [0.96593916]], dtype=float32)
```

- The accuracy on the validation set is 87.12%

MSE Loss Function

[131]:
```
np.random.seed(151)
model_MSE = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
#Model compilation
model_MSE.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
# validation
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
# Model Fit
np.random.seed(151)
history_model_MSE = model_MSE.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
29/29 [==============================] - 2s 27ms/step - loss: 0.1682 - accuracy:
0.7578 - val_loss: 0.1100 - val_accuracy: 0.8718
Epoch 2/20
```

```
29/29 [==============================] - 0s 9ms/step - loss: 0.0903 - accuracy:
0.8916 - val_loss: 0.0916 - val_accuracy: 0.8835
Epoch 3/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0658 - accuracy:
0.9237 - val_loss: 0.0935 - val_accuracy: 0.8721
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0527 - accuracy:
0.9414 - val_loss: 0.0832 - val_accuracy: 0.8889
Epoch 5/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0457 - accuracy:
0.9488 - val_loss: 0.0893 - val_accuracy: 0.8793
Epoch 6/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0358 - accuracy:
0.9619 - val_loss: 0.0906 - val_accuracy: 0.8764
Epoch 7/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0322 - accuracy:
0.9667 - val_loss: 0.1002 - val_accuracy: 0.8656
Epoch 8/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0269 - accuracy:
0.9736 - val_loss: 0.0993 - val_accuracy: 0.8685
Epoch 9/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0229 - accuracy:
0.9784 - val_loss: 0.0878 - val_accuracy: 0.8818
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0194 - accuracy:
0.9827 - val_loss: 0.0892 - val_accuracy: 0.8806
Epoch 11/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0196 - accuracy:
0.9813 - val_loss: 0.0902 - val_accuracy: 0.8808
Epoch 12/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0143 - accuracy:
0.9887 - val_loss: 0.0953 - val_accuracy: 0.8739
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0142 - accuracy:
0.9871 - val_loss: 0.0929 - val_accuracy: 0.8793
Epoch 14/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0097 - accuracy:
0.9926 - val_loss: 0.1209 - val_accuracy: 0.8464
Epoch 15/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0089 - accuracy:
0.9929 - val_loss: 0.0953 - val_accuracy: 0.8783
Epoch 16/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0106 - accuracy:
0.9906 - val_loss: 0.0954 - val_accuracy: 0.8784
Epoch 17/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0104 - accuracy:
0.9897 - val_loss: 0.0972 - val_accuracy: 0.8761
Epoch 18/20
```

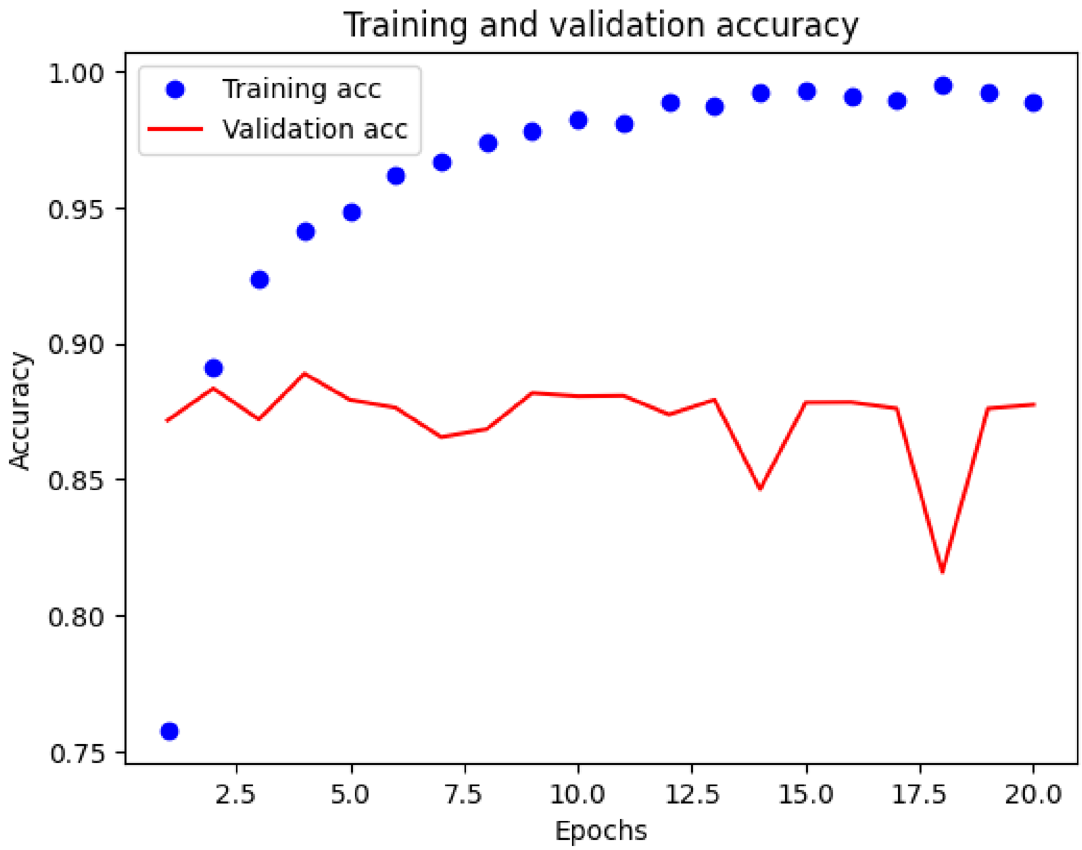
```
29/29 [==============================] - 0s 9ms/step - loss: 0.0061 - accuracy:
0.9952 - val_loss: 0.1530 - val_accuracy: 0.8160
Epoch 19/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0076 - accuracy:
0.9922 - val_loss: 0.0987 - val_accuracy: 0.8761
Epoch 20/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0098 - accuracy:
0.9889 - val_loss: 0.0984 - val_accuracy: 0.8775
```

[132]:
```
hist_dict_MSE = history_model_MSE.history
hist_dict_MSE.keys()
```

[132]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[133]:
```
import matplotlib.pyplot as plt
loss_values = hist_dict_MSE["loss"]
val_loss_values = hist_dict_MSE["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

[134]:
```
plt.clf()
acc = hist_dict_MSE["accuracy"]
val_acc = hist_dict_MSE["val_accuracy"]
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()
```



[135]:

```
model_MSE.fit(x_train, y_train, epochs=8, batch_size=525)
results_MSE = model_MSE.evaluate(x_test, y_test)
results_MSE
```

```
Epoch 1/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0433 - accuracy:
0.9482
Epoch 2/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0332 - accuracy:
0.9623
Epoch 3/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0261 - accuracy:
0.9708
Epoch 4/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0225 - accuracy:
0.9763
Epoch 5/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0197 - accuracy:
0.9793
Epoch 6/8
```

```
48/48 [==============================] - 0s 6ms/step - loss: 0.0179 - accuracy:
0.9818
Epoch 7/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0153 - accuracy:
0.9848
Epoch 8/8
48/48 [==============================] - 0s 5ms/step - loss: 0.0148 - accuracy:
0.9850
782/782 [==============================] - 1s 1ms/step - loss: 0.1079 -
accuracy: 0.8679
```

[135]:
```
[0.10791987925767899, 0.8679199814796448]
```

[136]:
```python
model_MSE.predict(x_test)
```

```
782/782 [==============================] - 1s 1ms/step
```

[136]:
```
array([[0.00888386],
       [0.99999994],
       [0.9346486 ],
       …,
       [0.04037911],
       [0.00962277],
       [0.9049492 ]], dtype=float32)
```

Tanh Activation Function

[137]:
```python
np.random.seed(151)
model_tanh = keras.Sequential([
    layers.Dense(16, activation="tanh"),
    layers.Dense(16, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])

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

x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]

np.random.seed(151)

history_tanh = model_tanh.fit(partial_x_train,
                    partial_y_train,
```

```
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
```

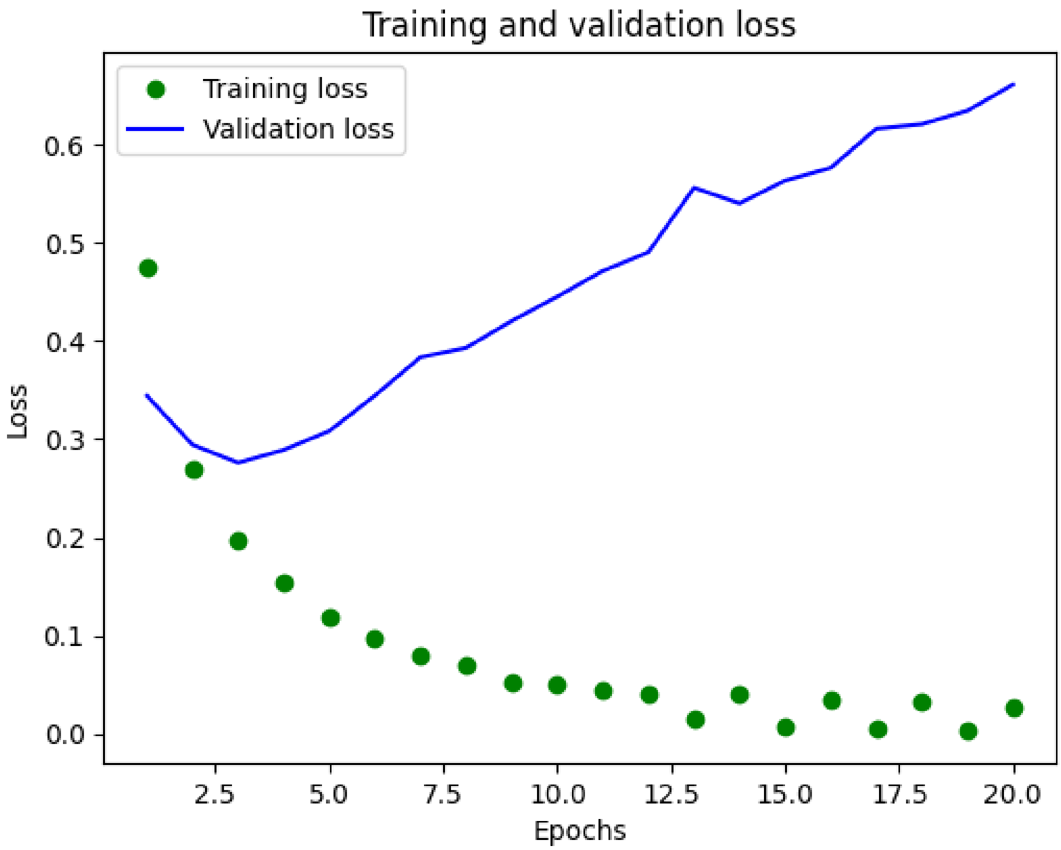
```
Epoch 1/20
29/29 [==============================] - 3s 31ms/step - loss: 0.4753 - accuracy:
0.7795 - val_loss: 0.3441 - val_accuracy: 0.8663
Epoch 2/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2704 - accuracy:
0.9021 - val_loss: 0.2942 - val_accuracy: 0.8791
Epoch 3/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1965 - accuracy:
0.9308 - val_loss: 0.2761 - val_accuracy: 0.8850
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1547 - accuracy:
0.9450 - val_loss: 0.2890 - val_accuracy: 0.8820
Epoch 5/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1180 - accuracy:
0.9598 - val_loss: 0.3083 - val_accuracy: 0.8811
Epoch 6/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0966 - accuracy:
0.9679 - val_loss: 0.3443 - val_accuracy: 0.8737
Epoch 7/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0795 - accuracy:
0.9741 - val_loss: 0.3834 - val_accuracy: 0.8746
Epoch 8/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0697 - accuracy:
0.9773 - val_loss: 0.3930 - val_accuracy: 0.8763
Epoch 9/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0532 - accuracy:
0.9840 - val_loss: 0.4203 - val_accuracy: 0.8738
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0504 - accuracy:
0.9853 - val_loss: 0.4451 - val_accuracy: 0.8714
Epoch 11/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0441 - accuracy:
0.9866 - val_loss: 0.4713 - val_accuracy: 0.8701
Epoch 12/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0414 - accuracy:
0.9876 - val_loss: 0.4903 - val_accuracy: 0.8692
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0147 - accuracy:
0.9982 - val_loss: 0.5559 - val_accuracy: 0.8651
Epoch 14/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0403 - accuracy:
0.9889 - val_loss: 0.5401 - val_accuracy: 0.8667
Epoch 15/20
```

```
29/29 [==============================] - 0s 9ms/step - loss: 0.0082 - accuracy:
0.9993 - val_loss: 0.5632 - val_accuracy: 0.8685
Epoch 16/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0352 - accuracy:
0.9906 - val_loss: 0.5761 - val_accuracy: 0.8677
Epoch 17/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0054 - accuracy:
0.9995 - val_loss: 0.6161 - val_accuracy: 0.8634
Epoch 18/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0337 - accuracy:
0.9907 - val_loss: 0.6207 - val_accuracy: 0.8665
Epoch 19/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0036 - accuracy:
0.9998 - val_loss: 0.6343 - val_accuracy: 0.8656
Epoch 20/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0278 - accuracy:
0.9928 - val_loss: 0.6610 - val_accuracy: 0.8646
```

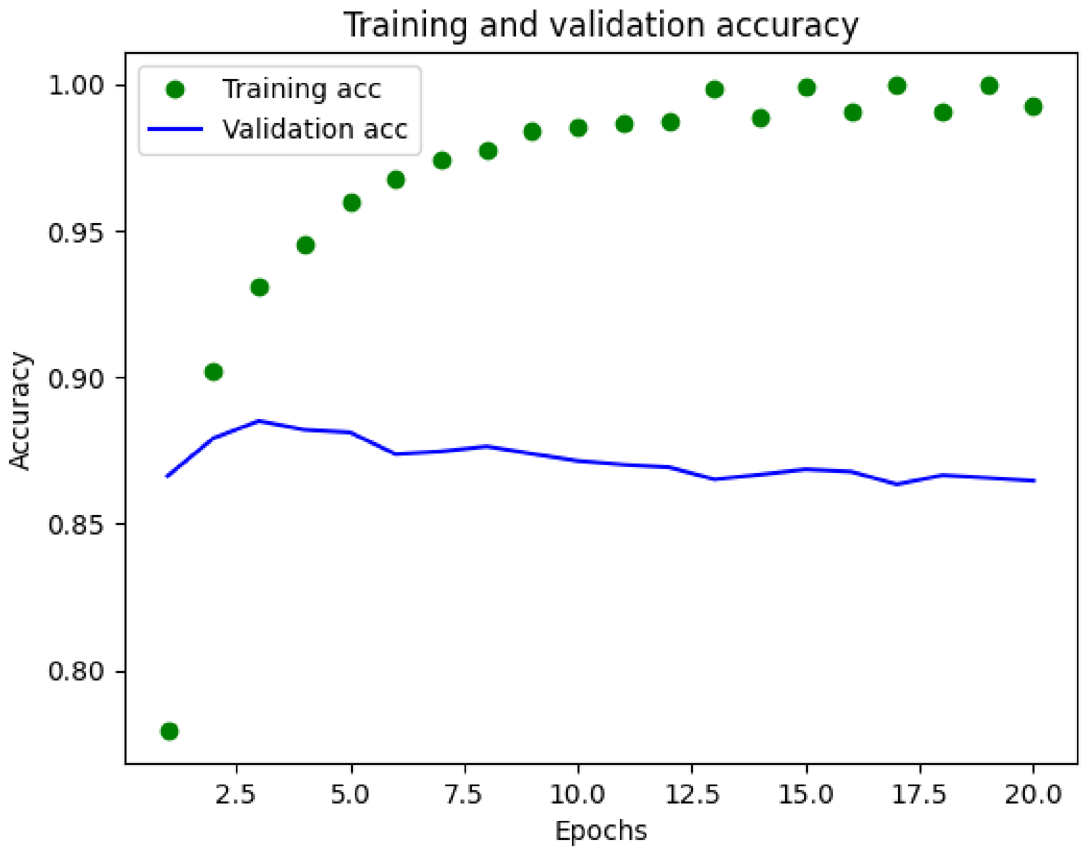
[138]:
```
hist_dict_tanh = history_tanh.history
hist_dict_tanh.keys()
```

[138]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[139]:
```
loss_values = hist_dict_tanh["loss"]
val_loss_values = hist_dict_tanh["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "go", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[140]:
```
plt.clf()
acc = hist_dict_tanh["accuracy"]
val_acc = hist_dict_tanh["val_accuracy"]
plt.plot(epochs, acc, "go", 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()
```



[141]:
```
model_tanh.fit(x_train, y_train, epochs=8, batch_size=525)
results_tanh = model_tanh.evaluate(x_test, y_test)
results_tanh
```

```
Epoch 1/8
48/48 [==============================] - 0s 6ms/step - loss: 0.2124 - accuracy:
0.9456
Epoch 2/8
48/48 [==============================] - 0s 6ms/step - loss: 0.1392 - accuracy:
0.9556
Epoch 3/8
48/48 [==============================] - 0s 6ms/step - loss: 0.1105 - accuracy:
0.9629
Epoch 4/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0876 - accuracy:
0.9710
Epoch 5/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0762 - accuracy:
0.9752
Epoch 6/8
```

```
48/48 [==============================] - 0s 6ms/step - loss: 0.0646 - accuracy:
0.9787
Epoch 7/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0537 - accuracy:
0.9831
Epoch 8/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0536 - accuracy:
0.9832
782/782 [==============================] - 1s 2ms/step - loss: 0.5871 -
accuracy: 0.8540
```

[141]:
```
[0.5871069431304932, 0.8539599776268005]
```

Adam Optimizer Function

[142]:
```
np.random.seed(151)
model_adam = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model_adam.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]

np.random.seed(151)

history_adam = model_adam.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
29/29 [==============================] - 2s 31ms/step - loss: 0.4884 - accuracy:
0.7734 - val_loss: 0.3212 - val_accuracy: 0.8714
Epoch 2/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2212 - accuracy:
0.9205 - val_loss: 0.2834 - val_accuracy: 0.8844
Epoch 3/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1436 - accuracy:
```

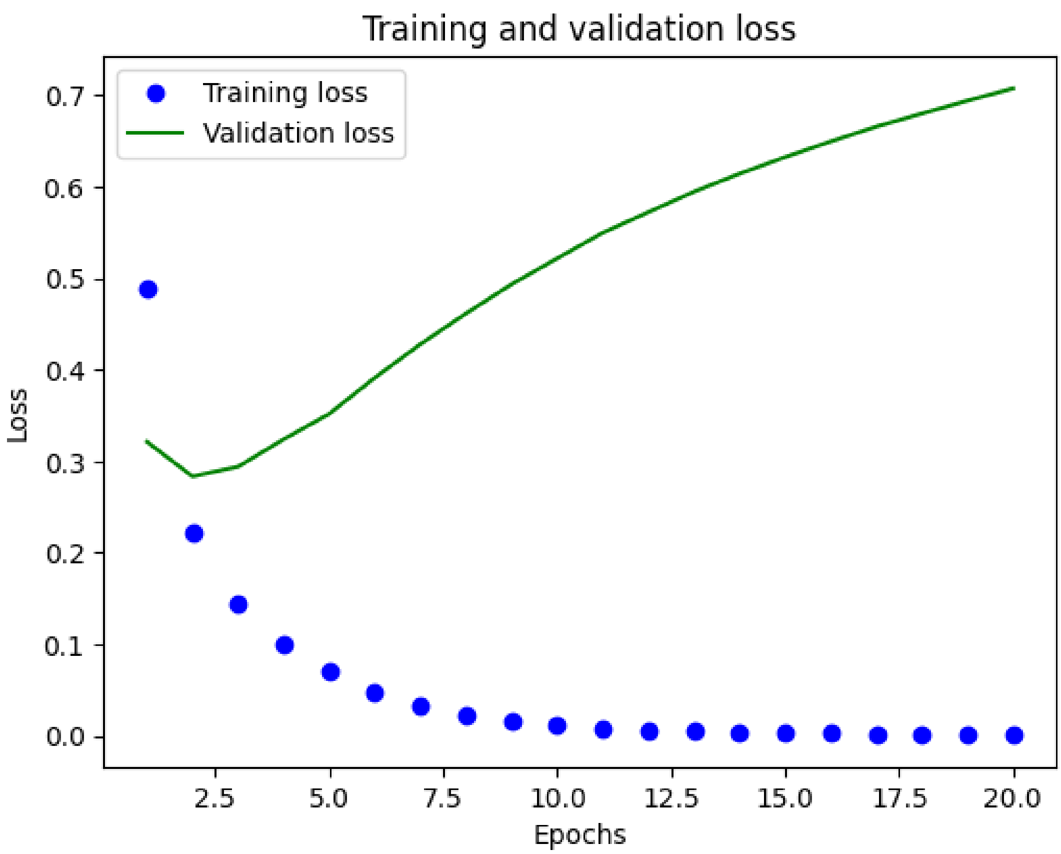
```
0.9525 - val_loss: 0.2941 - val_accuracy: 0.8838
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0996 - accuracy:
0.9721 - val_loss: 0.3240 - val_accuracy: 0.8808
Epoch 5/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0709 - accuracy:
0.9826 - val_loss: 0.3519 - val_accuracy: 0.8802
Epoch 6/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0477 - accuracy:
0.9914 - val_loss: 0.3914 - val_accuracy: 0.8748
Epoch 7/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0332 - accuracy:
0.9957 - val_loss: 0.4280 - val_accuracy: 0.8724
Epoch 8/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0224 - accuracy:
0.9983 - val_loss: 0.4615 - val_accuracy: 0.8724
Epoch 9/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0154 - accuracy:
0.9995 - val_loss: 0.4935 - val_accuracy: 0.8720
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0109 - accuracy:
0.9998 - val_loss: 0.5217 - val_accuracy: 0.8707
Epoch 11/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0082 - accuracy:
0.9999 - val_loss: 0.5497 - val_accuracy: 0.8701
Epoch 12/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0064 - accuracy:
0.9999 - val_loss: 0.5722 - val_accuracy: 0.8695
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0050 - accuracy:
0.9999 - val_loss: 0.5946 - val_accuracy: 0.8679
Epoch 14/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0041 - accuracy:
0.9999 - val_loss: 0.6142 - val_accuracy: 0.8677
Epoch 15/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0034 - accuracy:
1.0000 - val_loss: 0.6324 - val_accuracy: 0.8678
Epoch 16/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0029 - accuracy:
1.0000 - val_loss: 0.6494 - val_accuracy: 0.8669
Epoch 17/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0024 - accuracy:
1.0000 - val_loss: 0.6657 - val_accuracy: 0.8669
Epoch 18/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 0.6800 - val_accuracy: 0.8665
Epoch 19/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0018 - accuracy:
```

```
1.0000 - val_loss: 0.6940 - val_accuracy: 0.8658
Epoch 20/20
29/29 [==============================] - 0s 8ms/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 0.7074 - val_accuracy: 0.8659
```

[143]:
```
hist_dict_adam = history_adam.history
hist_dict_adam.keys()
```

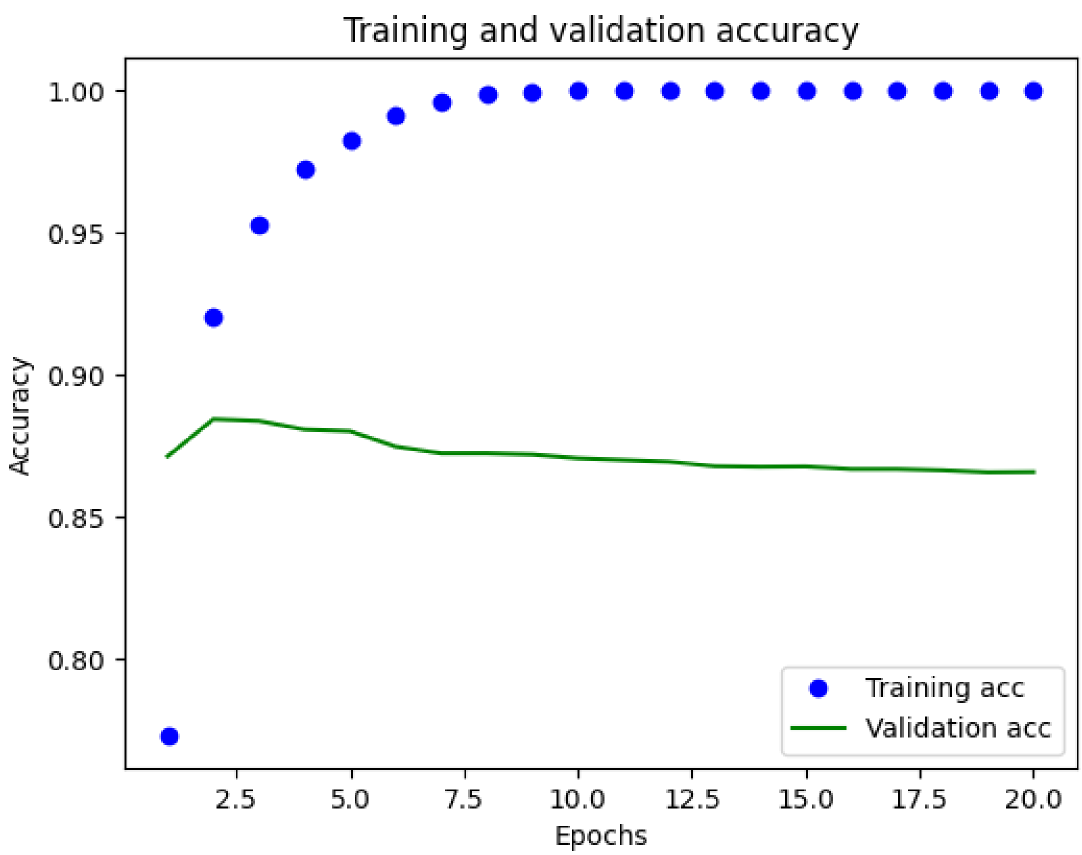
[143]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[144]:
```
loss_values = hist_dict_adam["loss"]
val_loss_values = hist_dict_adam["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "g", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[145]:

```
plt.clf()
acc = hist_dict_adam["accuracy"]
val_acc = hist_dict_adam["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "g", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[146]:

```
model_adam.fit(x_train, y_train, epochs=8, batch_size=525)
results_adam = model_adam.evaluate(x_test, y_test)
results_adam
```

```
Epoch 1/8
48/48 [==============================] - 0s 6ms/step - loss: 0.2416 - accuracy:
0.9298
Epoch 2/8
48/48 [==============================] - 0s 5ms/step - loss: 0.1116 - accuracy:
```

```
0.9628
Epoch 3/8
48/48 [==============================] - 0s 5ms/step - loss: 0.0691 - accuracy:
0.9818
Epoch 4/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0467 - accuracy:
0.9892
Epoch 5/8
48/48 [==============================] - 0s 5ms/step - loss: 0.0318 - accuracy:
0.9943
Epoch 6/8
48/48 [==============================] - 0s 5ms/step - loss: 0.0207 - accuracy:
0.9972
Epoch 7/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0134 - accuracy:
0.9987
Epoch 8/8
48/48 [==============================] - 0s 5ms/step - loss: 0.0089 - accuracy:
0.9992
782/782 [==============================] - 1s 2ms/step - loss: 0.7068 -
accuracy: 0.8551
```

[146]: [0.7067854404449463, 0.8551200032234192]

Regularization

[147]:
```
from tensorflow.keras import regularizers
np.random.seed(151)
model_regularization = keras.Sequential([
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.
 ↪001)),
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.
 ↪001)),
    layers.Dense(1, activation="sigmoid")
])
model_regularization.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
np.random.seed(151)
history_model_regularization = model_regularization.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
hist_dict_regularization = history_model_regularization.history
hist_dict_regularization.keys()
```

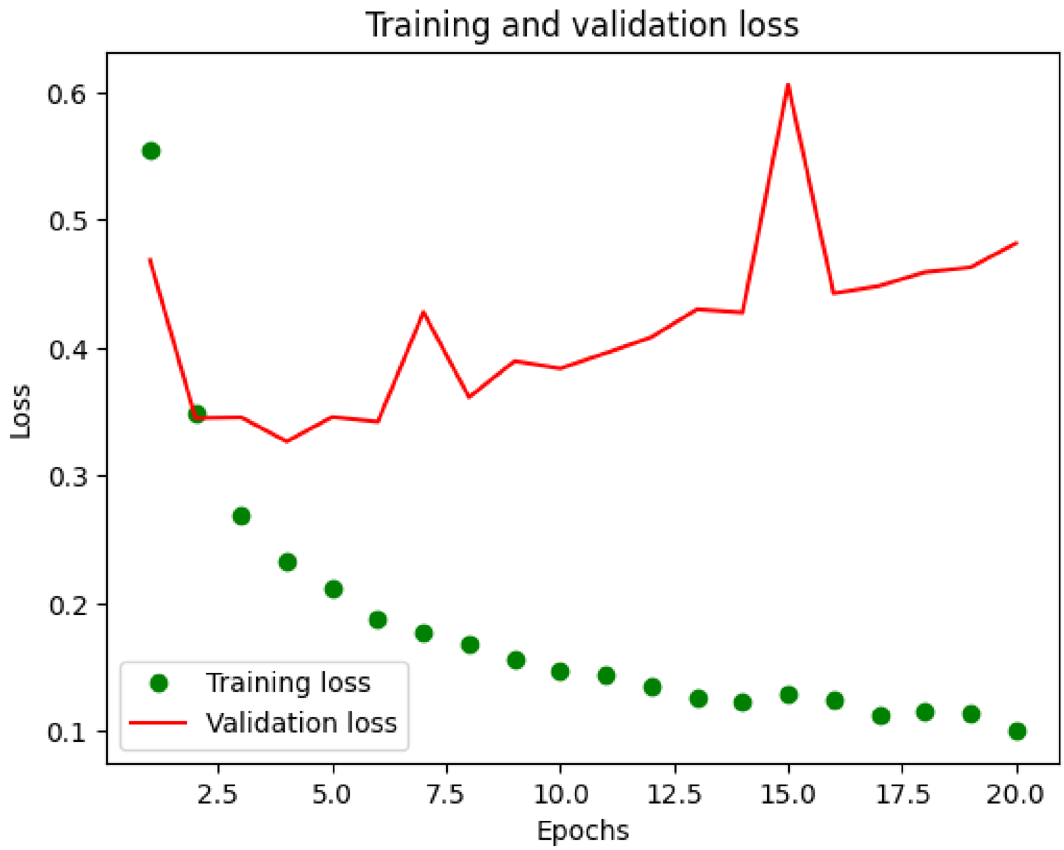
```
Epoch 1/20
```

```
29/29 [==============================] - 2s 31ms/step - loss: 0.5553 - accuracy:
0.7621 - val_loss: 0.4685 - val_accuracy: 0.8180
Epoch 2/20
29/29 [==============================] - 0s 9ms/step - loss: 0.3487 - accuracy:
0.8929 - val_loss: 0.3451 - val_accuracy: 0.8857
Epoch 3/20
29/29 [==============================] - 0s 8ms/step - loss: 0.2686 - accuracy:
0.9255 - val_loss: 0.3456 - val_accuracy: 0.8802
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2333 - accuracy:
0.9365 - val_loss: 0.3266 - val_accuracy: 0.8884
Epoch 5/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2116 - accuracy:
0.9445 - val_loss: 0.3460 - val_accuracy: 0.8833
Epoch 6/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1882 - accuracy:
0.9557 - val_loss: 0.3421 - val_accuracy: 0.8840
Epoch 7/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1768 - accuracy:
0.9601 - val_loss: 0.4279 - val_accuracy: 0.8583
Epoch 8/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1683 - accuracy:
0.9629 - val_loss: 0.3613 - val_accuracy: 0.8820
Epoch 9/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1563 - accuracy:
0.9689 - val_loss: 0.3896 - val_accuracy: 0.8754
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1471 - accuracy:
0.9729 - val_loss: 0.3839 - val_accuracy: 0.8806
Epoch 11/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1439 - accuracy:
0.9731 - val_loss: 0.3959 - val_accuracy: 0.8787
Epoch 12/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1354 - accuracy:
0.9787 - val_loss: 0.4083 - val_accuracy: 0.8741
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1259 - accuracy:
0.9803 - val_loss: 0.4301 - val_accuracy: 0.8714
Epoch 14/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1236 - accuracy:
0.9815 - val_loss: 0.4276 - val_accuracy: 0.8767
Epoch 15/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1297 - accuracy:
0.9771 - val_loss: 0.6059 - val_accuracy: 0.8350
Epoch 16/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1253 - accuracy:
0.9794 - val_loss: 0.4426 - val_accuracy: 0.8760
Epoch 17/20
```

```
29/29 [==============================] - 0s 9ms/step - loss: 0.1129 - accuracy:
0.9847 - val_loss: 0.4484 - val_accuracy: 0.8743
Epoch 18/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1155 - accuracy:
0.9835 - val_loss: 0.4591 - val_accuracy: 0.8752
Epoch 19/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1137 - accuracy:
0.9846 - val_loss: 0.4630 - val_accuracy: 0.8758
Epoch 20/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1008 - accuracy:
0.9892 - val_loss: 0.4817 - val_accuracy: 0.8706
```

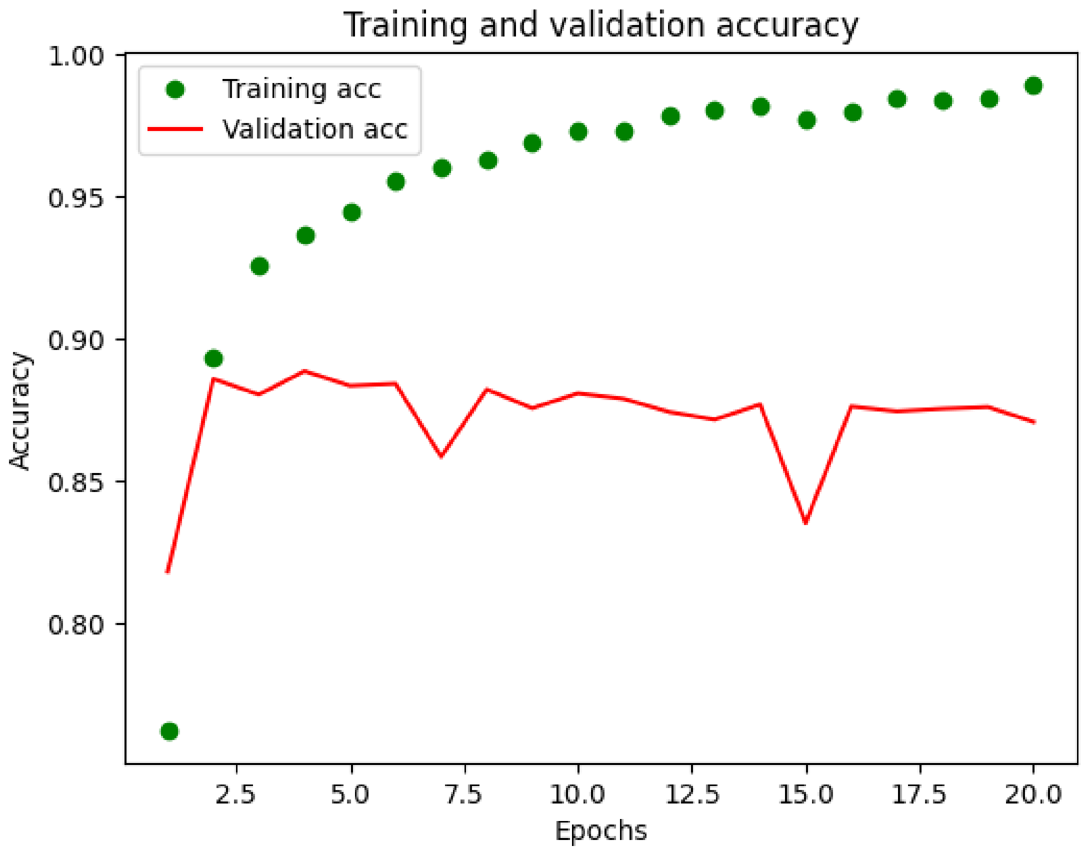
[147]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[148]:
```
loss_values = hist_dict_regularization["loss"]
val_loss_values = hist_dict_regularization["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "go", label="Training loss")
plt.plot(epochs, val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[149]:

```
plt.clf()
acc = hist_dict_regularization["accuracy"]
val_acc = hist_dict_regularization["val_accuracy"]
plt.plot(epochs, acc, "go", 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()
```



[150]:
```
model_regularization.fit(x_train, y_train, epochs=4, batch_size=525)
results_regularization = model_regularization.evaluate(x_test, y_test)
results_regularization
```

```
Epoch 1/4
48/48 [==============================] - 0s 6ms/step - loss: 0.2444 - accuracy:
0.9386
Epoch 2/4
48/48 [==============================] - 0s 6ms/step - loss: 0.1900 - accuracy:
0.9524
Epoch 3/4
48/48 [==============================] - 0s 6ms/step - loss: 0.1656 - accuracy:
0.9623
Epoch 4/4
48/48 [==============================] - 0s 5ms/step - loss: 0.1582 - accuracy:
0.9641
782/782 [==============================] - 1s 2ms/step - loss: 0.4355 -
accuracy: 0.8667
```

[150]: [0.43545255064964294, 0.8666800260543823]

- The loss on test set is 0.4217 and accuracy is 86.92%.

Dropout

[151]:
```python
from tensorflow.keras import regularizers
np.random.seed(151)
model_Dropout = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model_Dropout.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
np.random.seed(151)
history_model_Dropout = model_Dropout.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
hist_dict_Dropout = history_model_Dropout.history
hist_dict_Dropout.keys()
```

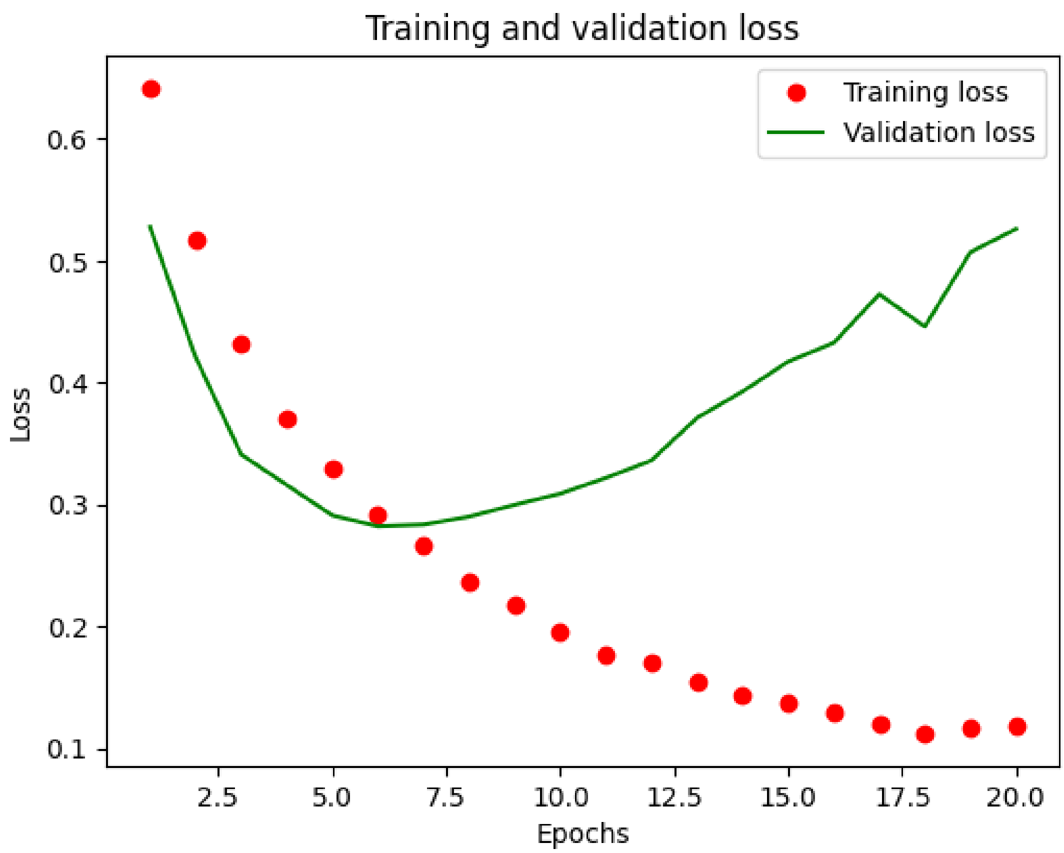
```
Epoch 1/20
29/29 [==============================] - 2s 30ms/step - loss: 0.6417 - accuracy:
0.6181 - val_loss: 0.5277 - val_accuracy: 0.8421
Epoch 2/20
29/29 [==============================] - 0s 9ms/step - loss: 0.5170 - accuracy:
0.7645 - val_loss: 0.4210 - val_accuracy: 0.8620
Epoch 3/20
29/29 [==============================] - 0s 9ms/step - loss: 0.4317 - accuracy:
0.8234 - val_loss: 0.3411 - val_accuracy: 0.8819
Epoch 4/20
29/29 [==============================] - 0s 9ms/step - loss: 0.3702 - accuracy:
0.8560 - val_loss: 0.3160 - val_accuracy: 0.8807
Epoch 5/20
29/29 [==============================] - 0s 8ms/step - loss: 0.3289 - accuracy:
0.8796 - val_loss: 0.2911 - val_accuracy: 0.8831
Epoch 6/20
29/29 [==============================] - 0s 8ms/step - loss: 0.2924 - accuracy:
0.8957 - val_loss: 0.2821 - val_accuracy: 0.8878
Epoch 7/20
29/29 [==============================] - 0s 8ms/step - loss: 0.2658 - accuracy:
0.9043 - val_loss: 0.2834 - val_accuracy: 0.8893
Epoch 8/20
29/29 [==============================] - 0s 9ms/step - loss: 0.2370 - accuracy:
```

```
0.9128 - val_loss: 0.2900 - val_accuracy: 0.8872
Epoch 9/20
29/29 [==============================] - 0s 8ms/step - loss: 0.2170 - accuracy:
0.9217 - val_loss: 0.2997 - val_accuracy: 0.8875
Epoch 10/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1956 - accuracy:
0.9339 - val_loss: 0.3089 - val_accuracy: 0.8878
Epoch 11/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1769 - accuracy:
0.9368 - val_loss: 0.3218 - val_accuracy: 0.8857
Epoch 12/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1701 - accuracy:
0.9393 - val_loss: 0.3361 - val_accuracy: 0.8850
Epoch 13/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1553 - accuracy:
0.9415 - val_loss: 0.3710 - val_accuracy: 0.8869
Epoch 14/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1439 - accuracy:
0.9458 - val_loss: 0.3927 - val_accuracy: 0.8844
Epoch 15/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1372 - accuracy:
0.9449 - val_loss: 0.4172 - val_accuracy: 0.8857
Epoch 16/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1296 - accuracy:
0.9497 - val_loss: 0.4326 - val_accuracy: 0.8823
Epoch 17/20
29/29 [==============================] - 0s 9ms/step - loss: 0.1192 - accuracy:
0.9535 - val_loss: 0.4725 - val_accuracy: 0.8849
Epoch 18/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1122 - accuracy:
0.9550 - val_loss: 0.4459 - val_accuracy: 0.8833
Epoch 19/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1174 - accuracy:
0.9555 - val_loss: 0.5070 - val_accuracy: 0.8845
Epoch 20/20
29/29 [==============================] - 0s 8ms/step - loss: 0.1185 - accuracy:
0.9537 - val_loss: 0.5261 - val_accuracy: 0.8833
```

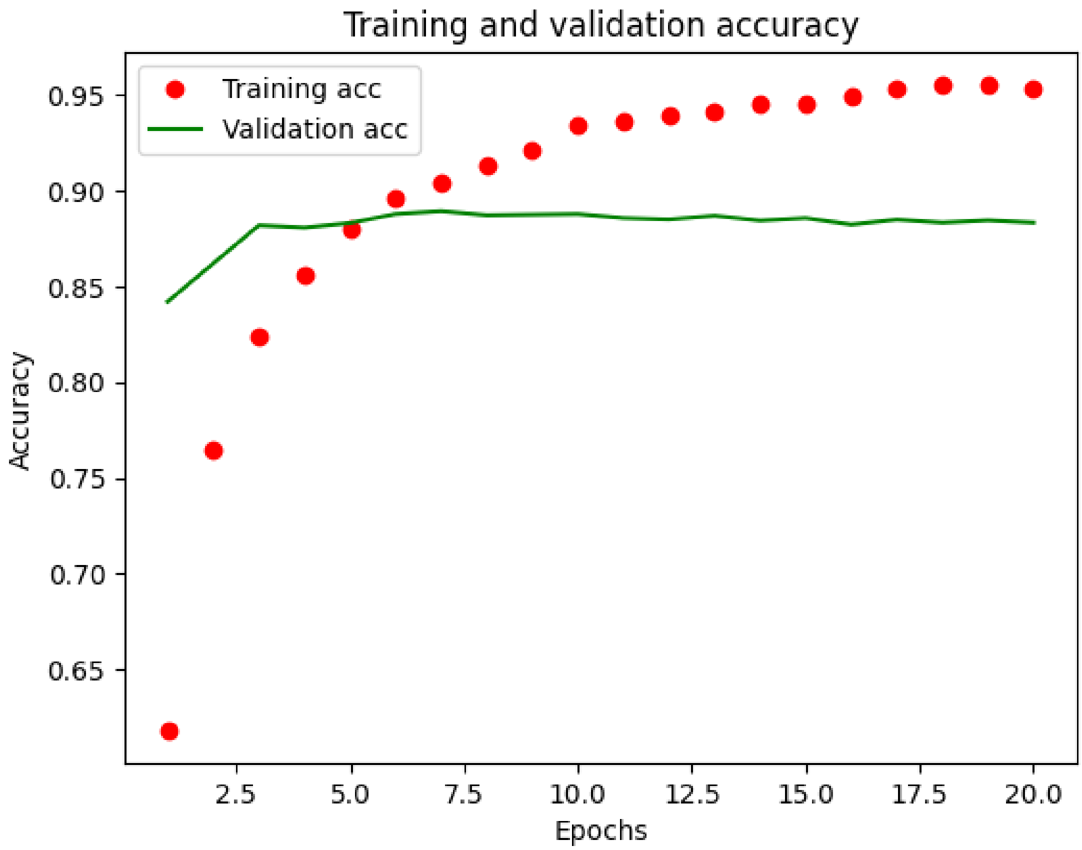
[151]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[152]:
```
loss_values = hist_dict_Dropout["loss"]
val_loss_values = hist_dict_Dropout["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "ro", label="Training loss")
plt.plot(epochs, val_loss_values, "g", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
```

```
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[153]:
```
plt.clf()
acc = hist_dict_Dropout["accuracy"]
val_acc = hist_dict_Dropout["val_accuracy"]
plt.plot(epochs, acc, "ro", label="Training acc")
plt.plot(epochs, val_acc, "g", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[154]:
```
model_Dropout.fit(x_train, y_train, epochs=8, batch_size=525)
results_Dropout = model_Dropout.evaluate(x_test, y_test)
results_Dropout
```

```
Epoch 1/8
48/48 [==============================] - 0s 6ms/step - loss: 0.2810 - accuracy:
0.9066
Epoch 2/8
48/48 [==============================] - 0s 6ms/step - loss: 0.2419 - accuracy:
0.9150
Epoch 3/8
48/48 [==============================] - 0s 6ms/step - loss: 0.2143 - accuracy:
0.9207
Epoch 4/8
48/48 [==============================] - 0s 6ms/step - loss: 0.1940 - accuracy:
0.9251
Epoch 5/8
48/48 [==============================] - 0s 6ms/step - loss: 0.1784 - accuracy:
0.9325
Epoch 6/8
```

```
48/48 [==============================] - 0s 6ms/step - loss: 0.1721 - accuracy:
0.9345
Epoch 7/8
48/48 [==============================] - 0s 6ms/step - loss: 0.1610 - accuracy:
0.9382
Epoch 8/8
48/48 [==============================] - 0s 5ms/step - loss: 0.1547 - accuracy:
0.9414
782/782 [==============================] - 1s 1ms/step - loss: 0.4943 -
accuracy: 0.8743
```

[154]:
```
[0.49430418014526367, 0.8743199706077576]
```

- The loss on the test set is 0.4943% and accuracy is 0.8743%

Training model with hyper tuned parameters

[155]:
```python
from tensorflow.keras import regularizers
np.random.seed(151)
model_Hyper = keras.Sequential([
    layers.Dense(32, activation="relu",kernel_regularizer=regularizers.l2(0.
 ↪0001)),
    layers.Dropout(0.5),
    layers.Dense(32, activation="relu",kernel_regularizer=regularizers.l2(0.
 ↪0001)),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.
 ↪0001)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model_Hyper.compile(optimizer="adam",
              loss="mse",
              metrics=["accuracy"])
np.random.seed(151)
history_model_Hyper = model_Hyper.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=525,
                    validation_data=(x_val, y_val))
hist_dict_Hyper = history_model_Hyper.history
hist_dict_Hyper.keys()
```

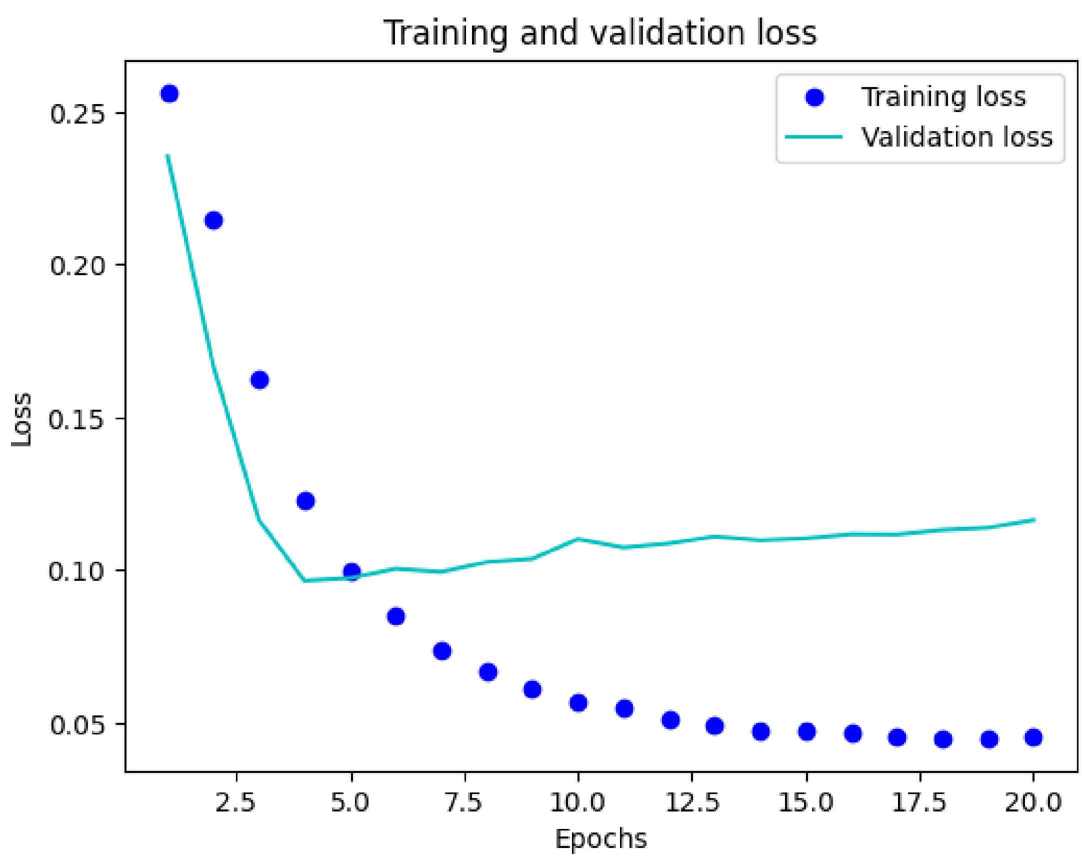
```
Epoch 1/20
29/29 [==============================] - 2s 32ms/step - loss: 0.2564 - accuracy:
0.5582 - val_loss: 0.2354 - val_accuracy: 0.7722
Epoch 2/20
29/29 [==============================] - 0s 10ms/step - loss: 0.2146 - accuracy:
```

```
0.7077 - val_loss: 0.1669 - val_accuracy: 0.8538
Epoch 3/20
29/29 [==============================] - 0s 10ms/step - loss: 0.1628 - accuracy:
0.8178 - val_loss: 0.1164 - val_accuracy: 0.8782
Epoch 4/20
29/29 [==============================] - 0s 10ms/step - loss: 0.1226 - accuracy:
0.8673 - val_loss: 0.0965 - val_accuracy: 0.8882
Epoch 5/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0998 - accuracy:
0.8951 - val_loss: 0.0974 - val_accuracy: 0.8865
Epoch 6/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0849 - accuracy:
0.9147 - val_loss: 0.1004 - val_accuracy: 0.8855
Epoch 7/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0738 - accuracy:
0.9273 - val_loss: 0.0994 - val_accuracy: 0.8891
Epoch 8/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0667 - accuracy:
0.9385 - val_loss: 0.1026 - val_accuracy: 0.8888
Epoch 9/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0615 - accuracy:
0.9435 - val_loss: 0.1037 - val_accuracy: 0.8859
Epoch 10/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0571 - accuracy:
0.9494 - val_loss: 0.1101 - val_accuracy: 0.8811
Epoch 11/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0552 - accuracy:
0.9517 - val_loss: 0.1074 - val_accuracy: 0.8862
Epoch 12/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0513 - accuracy:
0.9567 - val_loss: 0.1087 - val_accuracy: 0.8861
Epoch 13/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0493 - accuracy:
0.9587 - val_loss: 0.1110 - val_accuracy: 0.8861
Epoch 14/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0474 - accuracy:
0.9619 - val_loss: 0.1098 - val_accuracy: 0.8859
Epoch 15/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0471 - accuracy:
0.9606 - val_loss: 0.1104 - val_accuracy: 0.8831
Epoch 16/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0465 - accuracy:
0.9617 - val_loss: 0.1117 - val_accuracy: 0.8851
Epoch 17/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0453 - accuracy:
0.9646 - val_loss: 0.1116 - val_accuracy: 0.8846
Epoch 18/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0450 - accuracy:
```

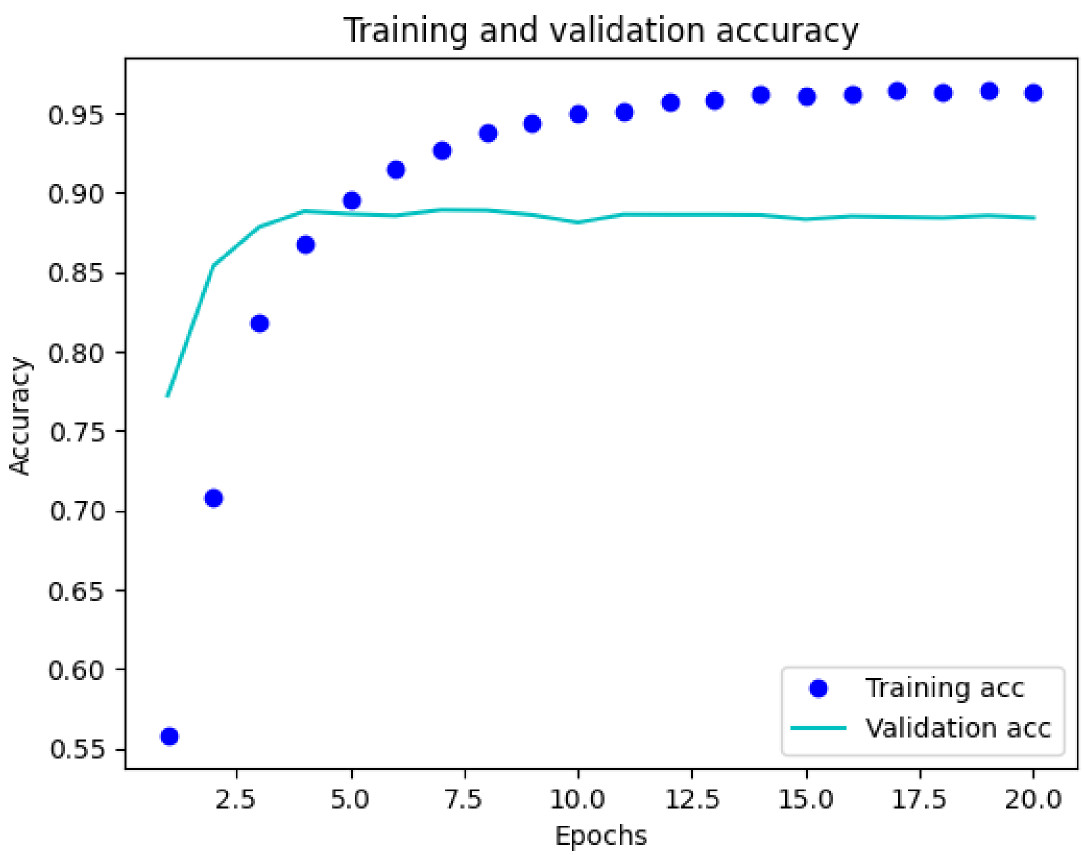
```
0.9637 - val_loss: 0.1131 - val_accuracy: 0.8841
Epoch 19/20
29/29 [==============================] - 0s 9ms/step - loss: 0.0449 - accuracy:
0.9640 - val_loss: 0.1139 - val_accuracy: 0.8855
Epoch 20/20
29/29 [==============================] - 0s 10ms/step - loss: 0.0457 - accuracy:
0.9635 - val_loss: 0.1163 - val_accuracy: 0.8841
```

[155]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[156]:
```
loss_values = hist_dict_Hyper["loss"]
val_loss_values = hist_dict_Hyper["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "c", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



[157]:
```
plt.clf()
acc = hist_dict_Hyper["accuracy"]
val_acc = hist_dict_Hyper["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "c", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[158]:
```
model_Hyper.fit(x_train, y_train, epochs=8, batch_size=525)
results_Hyper = model_Hyper.evaluate(x_test, y_test)
results_Hyper
```

```
Epoch 1/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0784 - accuracy:
0.9244
Epoch 2/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0737 - accuracy:
```

```
0.9298
Epoch 3/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0705 - accuracy:
0.9346
Epoch 4/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0652 - accuracy:
0.9418
Epoch 5/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0637 - accuracy:
0.9437
Epoch 6/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0612 - accuracy:
0.9479
Epoch 7/8
48/48 [==============================] - 0s 7ms/step - loss: 0.0583 - accuracy:
0.9522
Epoch 8/8
48/48 [==============================] - 0s 6ms/step - loss: 0.0574 - accuracy:
0.9530
782/782 [==============================] - 1s 2ms/step - loss: 0.1182 -
accuracy: 0.8774
```

[158]:
```
[0.11816161125898361, 0.8773599863052368]
```

Summary

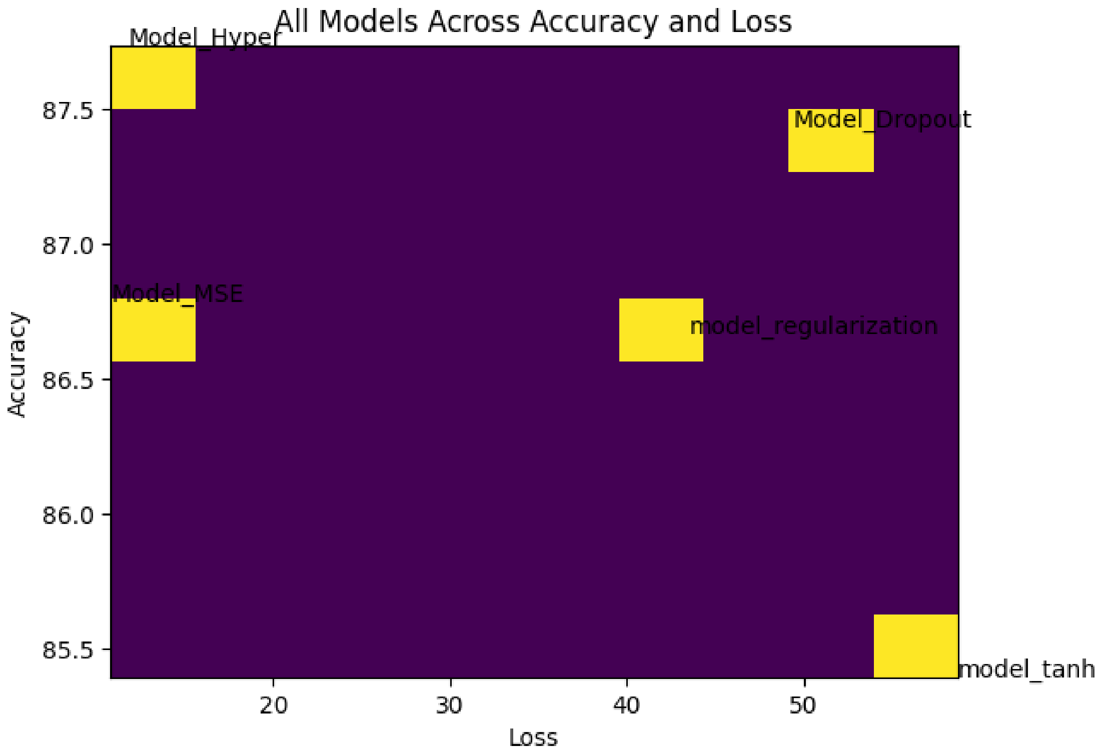
[159]:
```
model_Loss= np.
 ↪array([results_Dropout[0],results_Hyper[0],results_MSE[0],results_regularization[0],results
model_Loss
model_Accuracy= np.
 ↪array([results_Dropout[1],results_Hyper[1],results_MSE[1],results_regularization[1],results
model_Accuracy
Labels=['Model_Dropout','Model_Hyper','Model_MSE','model_regularization','model_tanh']
plt.clf()
```

```
<Figure size 640x480 with 0 Axes>
```

Compilation

[160]:
```
fig, ax = plt.subplots()
ax.hist2d(model_Loss,model_Accuracy)
for i, txt in enumerate(Labels):
    ax.annotate(txt, (model_Loss[i],model_Accuracy[i] ))
plt.title("All Models Across Accuracy and Loss")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```



Summary

- After freighting the data and setting the ultimate number of words and review extent, we constructed a baseline neural network model with a single secluded layer comprising 16 units. The activation function for the secluded layer was set to relu, andbinary_crossentropy was exercised as the loss function.

- To enhance the model's interpretation, we experimented with nonidentical approaches. originally, we varied the number of retired layers, likening models with one and three retired layers. Following training and evaluation on both the training and test datasets, we set up that the three retired layer model yielded hardly advanced confirmation and test delicacy compared to the single retired layer model.

- Afterward , we explored the jolt of conforming the number of hidden units within the layers, specially utilizing 32 and 64 units. By training and assessing models with varying figures of hidden units and conniving the confirmation delicacy for each, we observed that adding the number of hidden units usually redounded in advanced confirmation and test delicacy. still, inordinate units could conduct to overfitting.

- In extension, we researched the use of the mean squared inaccuracy(mse) loss function rather ofbinary_crossentropy. Through training and assessing the model with mse loss and likening the effects with the baseline model, we set up that the mse loss didn't significantly affect the model's interpretation.

Conclusion

- In the final phase of our trial, we enforced dropout regularization to alleviate overfitting. By incorporating dropout layers into a new model and conducting training and evaluation on the training and test datasets, we observed that the application of dropout regularization redounded in advanced confirmation delicacy assimilated to the baseline model.

- It's apparent that the colorful duplications of the neural network models displayed differing situations of delicacy and loss. The"Model_Hyper" demonstrated the loftiest delicacy and loss, indicating that employing three packed layers with a dropout rate of0.5 can yield optimal interpretation for the IMDB dataset. likewise, exercising the mean coincided inaccuracy(MSE) loss function led to the smallest loss value assimilated to doublecross-entropy. Again, the tanh activation function displayed lesser delicacy due to the evaporating grade case.

- The Adam optimizer function was linked as effective for calculating the model.likewise, regularization ways substantiated operative in reducing overfitting and performing in diminished losses, with the L2 model strutting hardly bettered delicacy. While the dropout technique downgraded the loss function, it didn't specially impact the delicacy.esteeming the vivid representation, it's apparent that the"Model_Hyper" exhibits the loftiest delicacy with a nicely low loss.
- On the other phase, the"Model_MSE" demonstrated the smallest loss value but didn't achieve the same position of delicacy as the"Model_Hyper." The"Model_tanh" displayed lesser delicacy assimilated to other models, while the"model_regularization" showcased advanced loss and lesser delicacy in comparison to the other models.

- Hence, grounded on the complete evaluation of the models, it can be concluded that the"Model_Hyper" stands out as the best- performing model among those assessed.