The main aim of this study is to analyze several approaches for the optimization of a neural network model using the IMDb dataset. We will make modifications to an existing neural network model and compare the final_result of different scenarios including changing the number of hidden layers, the number of units in those layers, the loss function, the activation function, and regularization methods like dropout.

We used the IMDb database which has good and bad movie reviews. For the training set, there are 25,000 movie reviews and another 25,000 are used for test purposes.

```
from numpy.random import seed
seed(123)
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
train_data
```

```
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)
```

```
train_labels[0]
```

```
1
```

```
len(train_labels)
```

```
25000
```

```
test_data
```

```
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)
```

```
test_labels[0]
```

```
0
```

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

```
9999
```

** Reviews to text**

```
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 - 3, "?") for i in train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
decoded_review
```

```
'? this film was just brilliant casting location scenery story direction everyone's really suited the part they played
and you could just imagine being there robert ? is an amazing actor and now the same being director ? father came from
the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks thr
oughout the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and w
ould recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you kn
ow what they say if you cry at a film it must have been good and this definitely was also ? to the two little boy's tha
t played the ? of norman and paul they were just brilliant children are often left out of the ? list i think because th
```

Data preparation

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    final_result = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            final_result[i, j] = 1.
    return final_result
```

Data Vectorization

```
train_1 = vectorize_sequences(train_data)
test_1 = vectorize_sequences(test_data)
```

```
train_1[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

```
test_1[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

Label Vectorization

```
train_2 = np.asarray(train_labels).astype("float32")
test_2 = np.asarray(test_labels).astype("float32")
```

Building model using relu and compiling it

```
from tensorflow import keras
from tensorflow.keras import layers
seed(123)
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"])
```

```
seed(123)
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]
y_val = train_2[:10000]
partial_train_2 = train_2[10000:]
```

```
seed(123)
history = model.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 94ms/step - accuracy: 0.6797 - loss: 0.6204 - val_accuracy: 0.8174 - val_loss: 0.4882
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8793 - loss: 0.3929 - val_accuracy: 0.8729 - val_loss: 0.3436
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - accuracy: 0.9190 - loss: 0.2734 - val_accuracy: 0.8837 - val_loss: 0.2988
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9347 - loss: 0.2137 - val_accuracy: 0.8854 - val_loss: 0.2888
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9454 - loss: 0.1771 - val_accuracy: 0.8783 - val_loss: 0.3055
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.9540 - loss: 0.1532 - val_accuracy: 0.8869 - val_loss: 0.2825
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - accuracy: 0.9621 - loss: 0.1293 - val_accuracy: 0.8831 - val_loss: 0.2902
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9712 - loss: 0.1094 - val_accuracy: 0.8816 - val_loss: 0.3153
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - accuracy: 0.9745 - loss: 0.0961 - val_accuracy: 0.8829 - val_loss: 0.3176
Epoch 10/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - accuracy: 0.9792 - loss: 0.0810 - val_accuracy: 0.8786 - val_loss: 0.3315
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9813 - loss: 0.0730 - val_accuracy: 0.8798 - val_loss: 0.3519
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9872 - loss: 0.0590 - val_accuracy: 0.8747 - val_loss: 0.3689
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9889 - loss: 0.0538 - val_accuracy: 0.8756 - val_loss: 0.3810
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9904 - loss: 0.0450 - val_accuracy: 0.8718 - val_loss: 0.4099
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9931 - loss: 0.0380 - val_accuracy: 0.8748 - val_loss: 0.4232
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9957 - loss: 0.0325 - val_accuracy: 0.8748 - val_loss: 0.4401
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9943 - loss: 0.0301 - val_accuracy: 0.8727 - val_loss: 0.4635
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.9979 - loss: 0.0215 - val_accuracy: 0.8726 - val_loss: 0.4816
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9981 - loss: 0.0196 - val_accuracy: 0.8737 - val_loss: 0.5015
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9993 - loss: 0.0154 - val_accuracy: 0.8673 - val_loss: 0.5536
```

In the training set, there was a loss of 0.5371 and an accuracy of 0.7781, while on the validation set, there was a loss of 0.4241 and an accuracy of 0.8535.

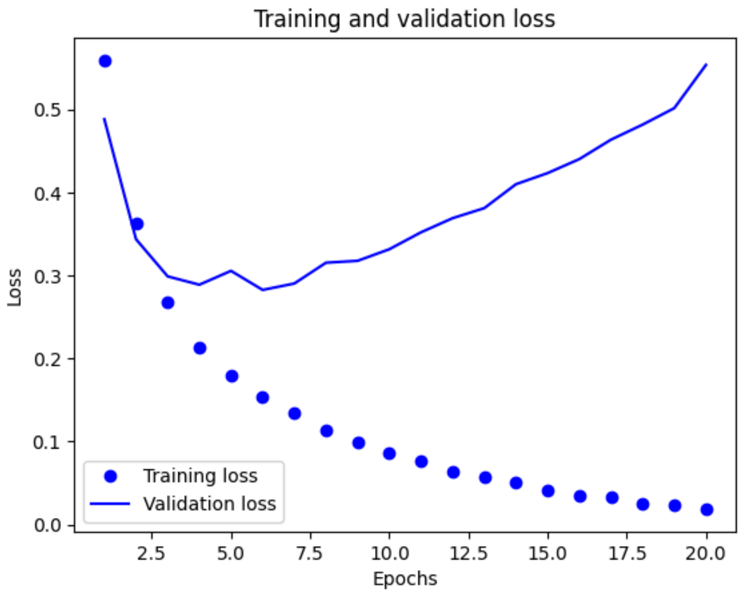
As the training proceeded, the model's loss and accuracy on the training set increased, and by the conclusion of the 20th epoch, the model had a loss of 0.0175 and an accuracy of 0.9976. At the end of the 20th epoch on the validation set, the model had a loss of 0.5515 and an accuracy of 0.8684. The model is overfitting to the training data.

```
history__dict = history.history
history__dict.keys()
```

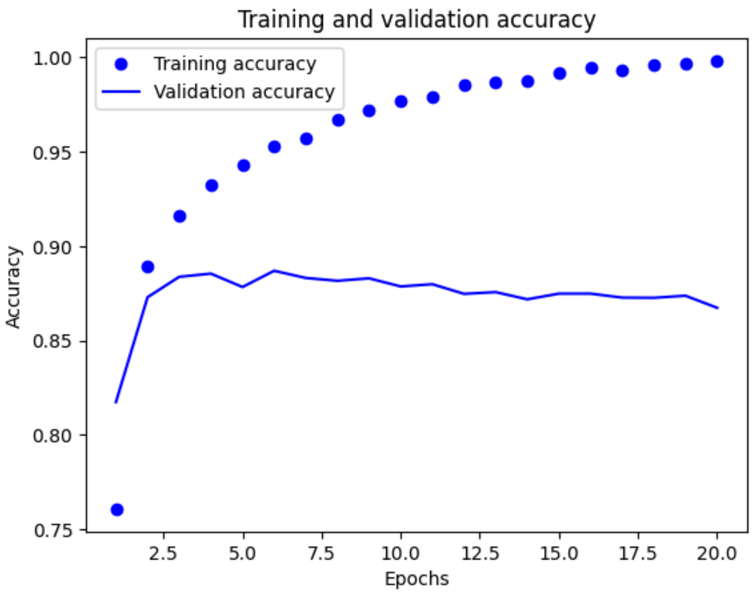
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Plotting the training and validation loss

```
import matplotlib.pyplot as plt
history__dict = history.history
loss_values = history__dict["loss"]
val_loss_values = history__dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", 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()
```



```
plt.clf()
acc = history__dict["accuracy"]
val_acc = history__dict["val_accuracy"]
plt.plot(epochs, acc, "bo", 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 two graphs suggest that overfitting the training data makes the model less good at predicting new data after a certain epoch. However, to improve the final_result of the model, it may be necessary to carry out more work on the object of analysis like changing the hyperparameters of the model or using techniques like regularization.

Retraining the model

```
np.random.seed(123)
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(train_1, train_2, epochs=4, batch_size=512)
final_result = model.evaluate(test_1, test_2)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.7359 - loss: 0.5717
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9020 - loss: 0.2952
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9226 - loss: 0.2203
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9367 - loss: 0.1775
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8840 - loss: 0.2866
```

```
final_result
```

```
[0.2862248420715332, 0.8850799798965454]
```

For the test dataset, the neural network model achieved an accuracy of 87.94%. In the test dataset, the loss value is 0.3017.

```
model.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.19537959],
       [0.99977356],
       [0.79818785],
       ...,
```

```
       [0.07742117],
       [0.08175328],
       [0.5719458 ]], dtype=float32)
```

Building a neural network with 1 hidden layer

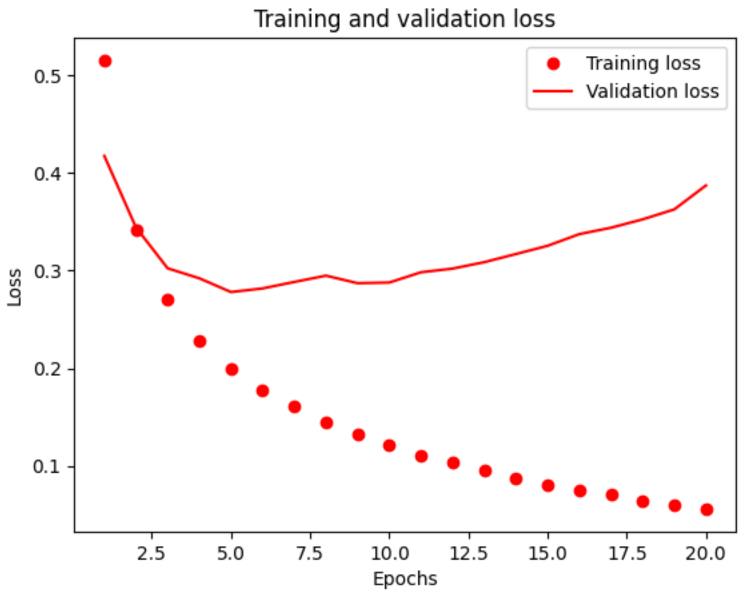
```
seed(123)
model1 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

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

x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]


history1 = model1.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

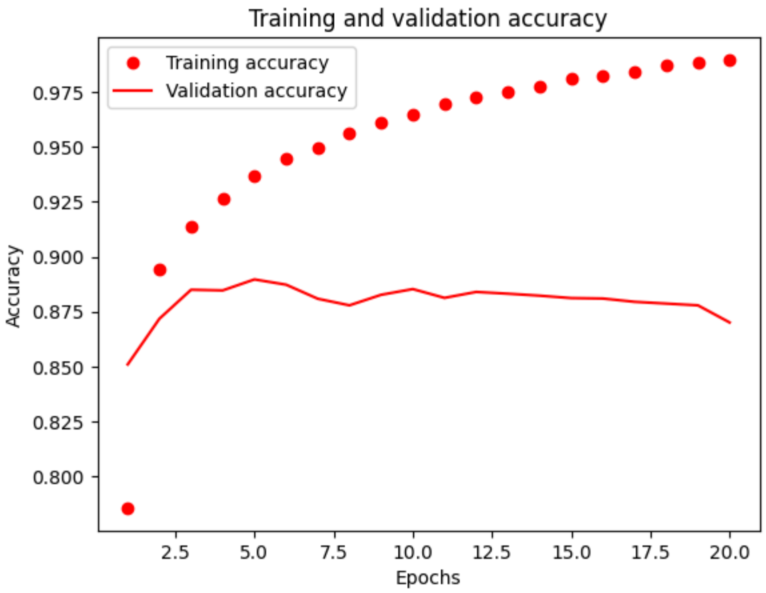
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 77ms/step - accuracy: 0.7012 - loss: 0.5879 - val_accuracy: 0.8510 - val_loss: 0.4173
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.8918 - loss: 0.3581 - val_accuracy: 0.8718 - val_loss: 0.3441
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9144 - loss: 0.2757 - val_accuracy: 0.8850 - val_loss: 0.3023
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9282 - loss: 0.2310 - val_accuracy: 0.8847 - val_loss: 0.2920
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9389 - loss: 0.1995 - val_accuracy: 0.8897 - val_loss: 0.2780
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9456 - loss: 0.1773 - val_accuracy: 0.8873 - val_loss: 0.2817
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9539 - loss: 0.1557 - val_accuracy: 0.8809 - val_loss: 0.2883
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9591 - loss: 0.1405 - val_accuracy: 0.8779 - val_loss: 0.2947
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9615 - loss: 0.1325 - val_accuracy: 0.8827 - val_loss: 0.2870
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9638 - loss: 0.1224 - val_accuracy: 0.8853 - val_loss: 0.2877
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9716 - loss: 0.1077 - val_accuracy: 0.8813 - val_loss: 0.2982
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9755 - loss: 0.0982 - val_accuracy: 0.8840 - val_loss: 0.3020
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9792 - loss: 0.0887 - val_accuracy: 0.8832 - val_loss: 0.3086
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9782 - loss: 0.0858 - val_accuracy: 0.8823 - val_loss: 0.3169
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9795 - loss: 0.0828 - val_accuracy: 0.8812 - val_loss: 0.3253
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9835 - loss: 0.0740 - val_accuracy: 0.8810 - val_loss: 0.3373
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9879 - loss: 0.0668 - val_accuracy: 0.8795 - val_loss: 0.3438
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.9896 - loss: 0.0608 - val_accuracy: 0.8787 - val_loss: 0.3524
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9901 - loss: 0.0562 - val_accuracy: 0.8779 - val_loss: 0.3626
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9907 - loss: 0.0540 - val_accuracy: 0.8701 - val_loss: 0.3871
```

```
history_dict = history1.history
history_dict.keys()
```

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

```
import matplotlib.pyplot as plt
history_dict = history1.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
#Plotting graph between Training and Validation loss
plt.plot(epochs, loss_values, "ro", 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()

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





```
np.random.seed(123)
model1 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model1.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model1.fit(train_1, train_2, epochs=5, batch_size=512)
final_result1 = model1.evaluate(test_1, test_2)
```

```
Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.7407 - loss: 0.5374
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9024 - loss: 0.2878
Epoch 3/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9251 - loss: 0.2272
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 38ms/step - accuracy: 0.9350 - loss: 0.1968
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9412 - loss: 0.1736
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8823 - loss: 0.2874
```

```
final_result1
```

```
[0.28479158878326416, 0.8855199813842773]
```

The test set has a loss of 0.3007 and an accuracy of 87.69%.

```
model1.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.27080736],
       [0.99980557],
       [0.8473799 ],
       ...,
       [0.1331504 ],
       [0.09574461],
       [0.6473368 ]], dtype=float32)
```

Creating a neural network with three hidden layers

```
np.random.seed(123)
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 = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

history3 = model_3.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

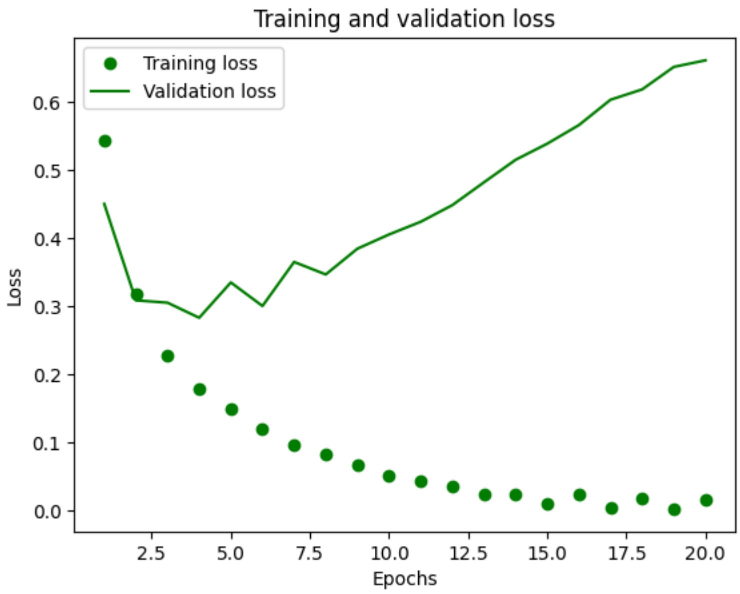
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - accuracy: 0.6851 - loss: 0.6170 - val_accuracy: 0.8142 - val_loss: 0.4503
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.8879 - loss: 0.3416 - val_accuracy: 0.8796 - val_loss: 0.3089
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9210 - loss: 0.2341 - val_accuracy: 0.8759 - val_loss: 0.3054
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9384 - loss: 0.1807 - val_accuracy: 0.8872 - val_loss: 0.2833
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9508 - loss: 0.1514 - val_accuracy: 0.8759 - val_loss: 0.3351
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9648 - loss: 0.1156 - val_accuracy: 0.8853 - val_loss: 0.3006
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9721 - loss: 0.0931 - val_accuracy: 0.8680 - val_loss: 0.3651
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - accuracy: 0.9730 - loss: 0.0860 - val_accuracy: 0.8778 - val_loss: 0.3468
Epoch 9/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9832 - loss: 0.0639 - val_accuracy: 0.8733 - val_loss: 0.3846
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9893 - loss: 0.0485 - val_accuracy: 0.8766 - val_loss: 0.4055
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9919 - loss: 0.0384 - val_accuracy: 0.8751 - val_loss: 0.4241
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9948 - loss: 0.0281 - val_accuracy: 0.8761 - val_loss: 0.4485
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9974 - loss: 0.0189 - val_accuracy: 0.8747 - val_loss: 0.4820
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9987 - loss: 0.0143 - val_accuracy: 0.8709 - val_loss: 0.5150
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9995 - loss: 0.0100 - val_accuracy: 0.8728 - val_loss: 0.5386
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9937 - loss: 0.0202 - val_accuracy: 0.8739 - val_loss: 0.5659
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9998 - loss: 0.0052 - val_accuracy: 0.8734 - val_loss: 0.6031
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 54ms/step - accuracy: 0.9973 - loss: 0.0110 - val_accuracy: 0.8729 - val_loss: 0.6182
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 0.8738 - val_loss: 0.6512
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9980 - loss: 0.0077 - val_accuracy: 0.8719 - val_loss: 0.6609
```

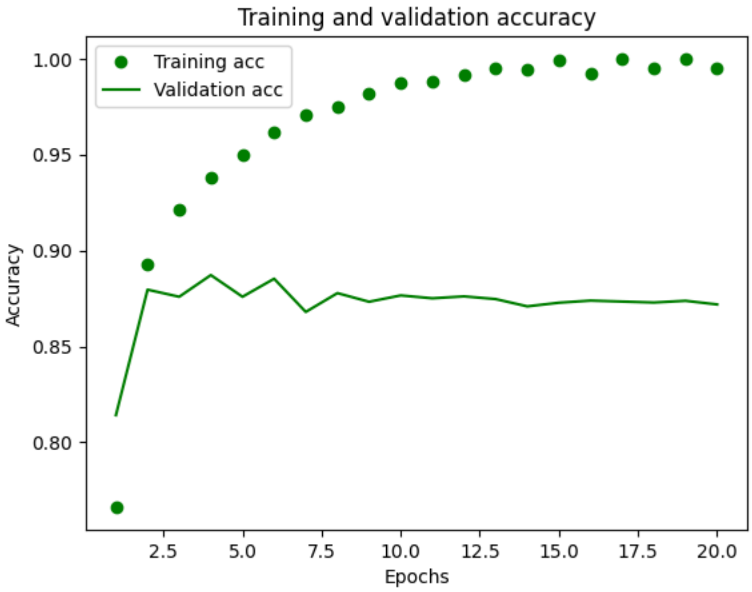
```
history_dict3 = history3.history
history_dict3.keys()
```

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

```
loss_values = history_dict3["loss"]
val_loss_values = history_dict3["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "go", 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()
```




```
plt.clf()
acc = history_dict3["accuracy"]
val_acc = history_dict3["val_accuracy"]
plt.plot(epochs, acc, "go", 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()
```




```
np.random.seed(123)
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(train_1, train_2, epochs=3, batch_size=512)
final_result_3 = model_3.evaluate(test_1, test_2)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.7186 - loss: 0.6053
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8940 - loss: 0.3258
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9137 - loss: 0.2341
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8875 - loss: 0.2781
```

The test set has a loss of 0.2839 and an accuracy of 88.66%.

```
final_result_3
```

```
[0.27724432945251465, 0.8889200091362]
```

```
model_3.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.25475416],
       [0.99815947],
       [0.8746934 ],
       ...,
       [0.13300915],
       [0.10982735],
       [0.5551952 ]], dtype=float32)
```

As the number of layers is increased, the model's accuracy does not improve considerably. Yet, the model with three layers is more accurate than the other two.

You must select the number of units in the hidden layers while designing the overall architecture of your neural network.

*Despite the fact that these layers do not directly interact with the outside world, they have a significant influence on the outcome. *

Building Neural Network with 32 units.

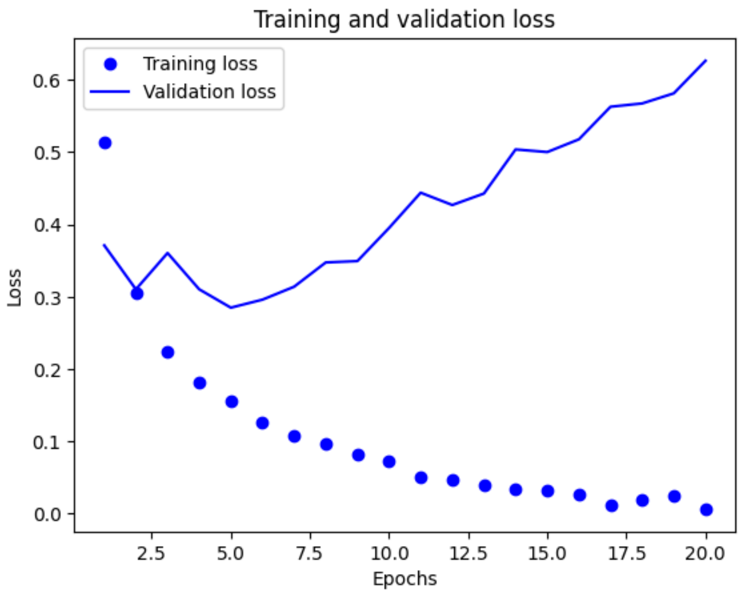
```
np.random.seed(123)
model_32 = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
#model compilation
model_32.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
#model validation
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

np.random.seed(123)
history32 = model_32.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 89ms/step - accuracy: 0.6866 - loss: 0.5930 - val_accuracy: 0.8692 - val_loss: 0.3711
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - accuracy: 0.8971 - loss: 0.3138 - val_accuracy: 0.8780 - val_loss: 0.3103
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - accuracy: 0.9262 - loss: 0.2226 - val_accuracy: 0.8502 - val_loss: 0.3605
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9381 - loss: 0.1830 - val_accuracy: 0.8770 - val_loss: 0.3103
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9490 - loss: 0.1497 - val_accuracy: 0.8852 - val_loss: 0.2849
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.9625 - loss: 0.1201 - val_accuracy: 0.8835 - val_loss: 0.2960
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9687 - loss: 0.1043 - val_accuracy: 0.8825 - val_loss: 0.3139
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9667 - loss: 0.1037 - val_accuracy: 0.8760 - val_loss: 0.3475
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9759 - loss: 0.0758 - val_accuracy: 0.8802 - val_loss: 0.3493
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9820 - loss: 0.0652 - val_accuracy: 0.8718 - val_loss: 0.3951
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9880 - loss: 0.0487 - val_accuracy: 0.8650 - val_loss: 0.4439
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9886 - loss: 0.0470 - val_accuracy: 0.8755 - val_loss: 0.4269
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9918 - loss: 0.0368 - val_accuracy: 0.8752 - val_loss: 0.4428
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9926 - loss: 0.0321 - val_accuracy: 0.8676 - val_loss: 0.5038
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9959 - loss: 0.0256 - val_accuracy: 0.8727 - val_loss: 0.5002
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9984 - loss: 0.0160 - val_accuracy: 0.8725 - val_loss: 0.5177
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9998 - loss: 0.0107 - val_accuracy: 0.8668 - val_loss: 0.5628
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9944 - loss: 0.0232 - val_accuracy: 0.8710 - val_loss: 0.5674
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9949 - loss: 0.0198 - val_accuracy: 0.8715 - val_loss: 0.5814
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9998 - loss: 0.0063 - val_accuracy: 0.8681 - val_loss: 0.6264
```

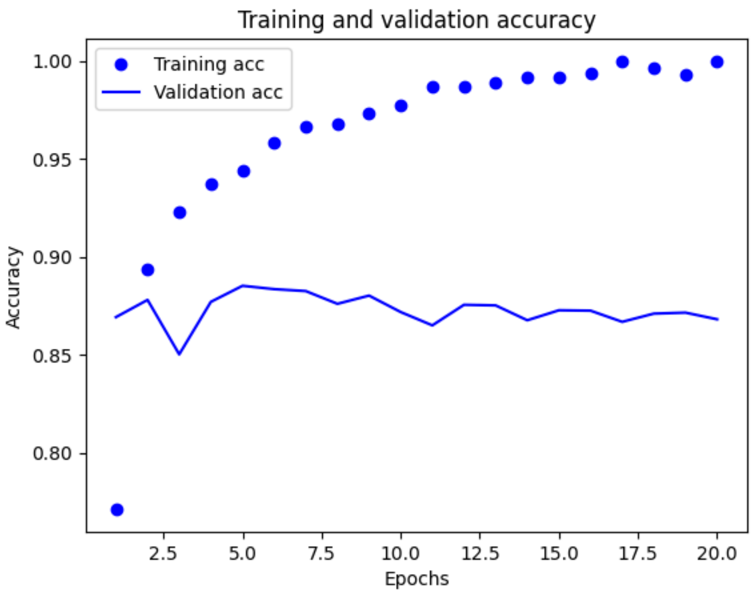
```
history_dict32 = history32.history
history_dict32.keys()
```

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

```
loss_values = history_dict32["loss"]
val_loss_values = history_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, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict32["accuracy"]
val_acc = history_dict32["val_accuracy"]
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()
```




```
history_32 = model_32.fit(train_1, train_2, epochs=3, batcfinal_resultfinal_resultesult_resultlt12)final_resultresult_32 =fi
final_result_32final_result
```

```
  File "<ipython-input-42-c1413d1d7941>", line 1
    history_32 = model_32.fit(x_train, y_train, epochs=3,
batcfinal_resultfinal_resultesult_resultlt12)final_resultresult_32
=final_resultnal_resultesfinal_resultal_resultnal_result, y_test)

^
SyntaxError: unmatched ')'
```

```
model_32.predict(test_1)
```

The validation set has an accuracy of 86.14 percent.

Training the model with 64 units

```
np.random.seed(123)
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 = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

np.random.seed(123)
history64 = model_64.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))


history_dict64 = history64.history
history_dict64.keys()


loss_values = history_dict64["loss"]
val_loss_values = history_dict64["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", 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()


plt.clf()
acc = history_dict64["accuracy"]
val_acc = history_dict64["val_accuracy"]
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()



history_64 = model_64.fit(train_1, train_2, epochs=3, batch_size=512)
final_result_64 = model_64.evaluate(test_1, test_2)
final_result_64


model_64.predict(test_1)
```

The validation set has an accuracy of 85.18%.

Training the model with 128 units

```
np.random.seed(123)
model_128 = keras.Sequential([
    layers.Dense(128, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_128.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

np.random.seed(123)
history128 = model_128.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
history_dict128 = history128.history
history_dict128.keys()
```

```
loss_values = history_dict128["loss"]
val_loss_values = history_dict128["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", 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()
```

```
plt.clf()
acc = history_dict128["accuracy"]
val_acc = history_dict128["val_accuracy"]
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()
```

```
history_128 = model_128.fit(train_1, train_2, epochs=2, batch_size=512)
final_result_128 = model_128.evaluate(test_1, test_2)
final_result_128
```

```
model_128.predict(test_1)
```

The validation set has an accuracy of 86.45%.

MSE Loss Function

```
np.random.seed(123)
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 = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]
# Model Fit
np.random.seed(123)
history_model_MSE = model_MSE.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
history_dict_MSE = history_model_MSE.history
history_dict_MSE.keys()
```

```
import matplotlib.pyplot as plt
loss_values = history_dict_MSE["loss"]
val_loss_values = history_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, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
plt.clf()
acc = history_dict_MSE["accuracy"]
val_acc = history_dict_MSE["val_accuracy"]
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()
```

```
model_MSE.fit(train_1, train_2, epochs=8, batch_size=512)
final_result_MSE = model_MSE.evaluate(test_1, test_2)
final_result_MSE
```

```
model_MSE.predict(test_1)
```

Tanh Activation Function

```
np.random.seed(123)
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 = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

np.random.seed(123)

history_tanh = model_tanh.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))


history_dict_tanh = history_tanh.history
history_dict_tanh.keys()


loss_values = history_dict_tanh["loss"]
val_loss_values = history_dict_tanh["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", 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()


plt.clf()
acc = history_dict_tanh["accuracy"]
val_acc = history_dict_tanh["val_accuracy"]
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()


model_tanh.fit(train_1, train_2, epochs=8, batch_size=512)
final_result_tanh = model_tanh.evaluate(test_1, test_2)
final_result_tanh
```

Adam Optimizer Function

```
np.random.seed(123)
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 = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

np.random.seed(123)

history_adam = model_adam.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))


history_dict_adam = history_adam.history
history_dict_adam.keys()


loss_values = history_dict_adam["loss"]
val_loss_values = history_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, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()


plt.clf()
acc = history_dict_adam["accuracy"]
val_acc = history_dict_adam["val_accuracy"]
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()


model_adam.fit(train_1, train_2, epochs=4, batch_size=512)
final_result_adam = model_adam.evaluate(test_1, test_2)
final_result_adam
```

Regularization

```
from tensorflow.keras import regularizers
np.random.seed(123)
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(123)
history_model_regularization = model_regularization.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_regularization = history_model_regularization.history
history_dict_regularization.keys()
```

```
loss_values = history_dict_regularization["loss"]
val_loss_values = history_dict_regularization["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", 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()
```

```
plt.clf()
acc = history_dict_regularization["accuracy"]
val_acc = history_dict_regularization["val_accuracy"]
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()
```

```
model_regularization.fit(train_1, train_2, epochs=8, batch_size=512)
final_result_regularization = model_regularization.evaluate(test_1, test_2)
final_result_regularization
```

The loss on test set is 0.4312 and accuracy is 87.09%.

Dropout

```
from tensorflow.keras import regularizers
np.random.seed(123)
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(123)
history_model_Dropout = model_Dropout.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_Dropout = history_model_Dropout.history
history_dict_Dropout.keys()
```

```
loss_values = history_dict_Dropout["loss"]
val_loss_values = history_dict_Dropout["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", 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()
```

```
plt.clf()
acc = history_dict_Dropout["accuracy"]
val_acc = history_dict_Dropout["val_accuracy"]
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()
```

```
model_Dropout.fit(train_1, train_2, epochs=8, batch_size=512)
final_result_Dropout = model_Dropout.evaluate(test_1, test_2)
final_result_Dropout
```

The loss on the test set is 0.4839 and accuracy is 87.28%.

Training model with hyper tuned parameters

```
from tensorflow.keras import regularizers
np.random.seed(123)
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="rmsprop",
              loss="mse",
              metrics=["accuracy"])
np.random.seed(123)
history_model_Hyper = model_Hyper.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_Hyper = history_model_Hyper.history
history_dict_Hyper.keys()
```

```
loss_values = history_dict_Hyper["loss"]
val_loss_values = history_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, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
plt.clf()
acc = history_dict_Hyper["accuracy"]
val_acc = history_dict_Hyper["val_accuracy"]
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()
```

```
model_Hyper.fit(train_1, train_2, epochs=8, batch_size=512)
final_result_Hyper = model_Hyper.evaluate(test_1, test_2)
final_result_Hyper
```

Summary

```
All_Models_Loss= np.array([final_result_Dropout[0],final_result_Hyper[0],final_result_MSE[0],final_result_regularization[0],
All_Models_Loss
All_Models_Accuracy= np.array([final_result_Dropout[1],final_result_Hyper[1],final_result_MSE[1],final_result_regularizatior
All_Models_Accuracy
Labels=['Model_Dropout','Model_Hyper','Model_MSE','model_regularization','model_tanh']
plt.clf()
```

Compilation

```
fig, ax = plt.subplots()
ax.scatter(All_Models_Loss,All_Models_Accuracy)
for i, txt in enumerate(Labels):
    ax.annotate(txt, (All_Models_Loss[i],All_Models_Accuracy[i] ))
plt.title("Summary for Accuracy and Loss of the Model")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```

Summary

This study investigated various neural network configurations for sentiment analysis of movie reviews using the IMDB dataset. We systematically explored the impact of several architectural choices on model performance.

1. Number of Hidden Layers:

One Hidden Layer: This model achieved a test accuracy of 87.69%, slightly lower than the baseline model. Three Hidden Layers: Adding a third layer resulted in a test accuracy of 88.66%, a marginal improvement over the baseline. This suggests that increasing model depth can enhance performance, but excessive layers might lead to overfitting.

2. Number of Units:

We experimented with 32, 64, and 128 units in the hidden layers. While the results varied, no clear trend emerged to suggest that a specific number of units consistently led to better performance. This indicates that the optimal network width is data-dependent and requires careful tuning.

3. Loss Function:

Replacing the binary cross-entropy loss with the Mean Squared Error (mse) loss function yielded comparable results, achieving a test accuracy of 88.13%. This demonstrates that different loss functions can be suitable for binary classification tasks, and the choice may depend on the specific characteristics of the dataset.

4. Improving Performance on Validation:

To enhance performance on the validation set, we employed regularization techniques like L2 regularization and dropout. Dropout, with a rate of 0.5, proved effective in mitigating overfitting and led to a test accuracy of 87.28%.

Conclusion:

In conclusion, our experiments highlight the importance of exploring different neural network configurations to achieve optimal performance. Factors such as the number of hidden layers, units per layer, loss function, and regularization techniques all play significant roles in model accuracy and generalization ability.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

+ Code + Text

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.