```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "code",
      "execution_count": 5,
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "8ScIf544EhWe",
        "outputId": "1f200689-f886-4345-a19c-2243fade03a1"
      },
      "outputs": [
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "9999"
            ]
          },
          "metadata": {},
          "execution_count": 5
        }
      ],
      "source": [
        "#Classifying movie reviews: A binary classification example\n",
        "#The IMDB dataset\n",
        "#Loading the IMDB dataset\n",
        "\n",
        "from tensorflow.keras.datasets import imdb\n",
        "(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(\n",
        "    num_words=10000)\n",
        "\n",
        "train_data[0]\n",
        "train_labels[0]\n",
        "max([max(sequence) for sequence in train_data])\n",
        "\n"
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Decoding reviews back to text\n",
```

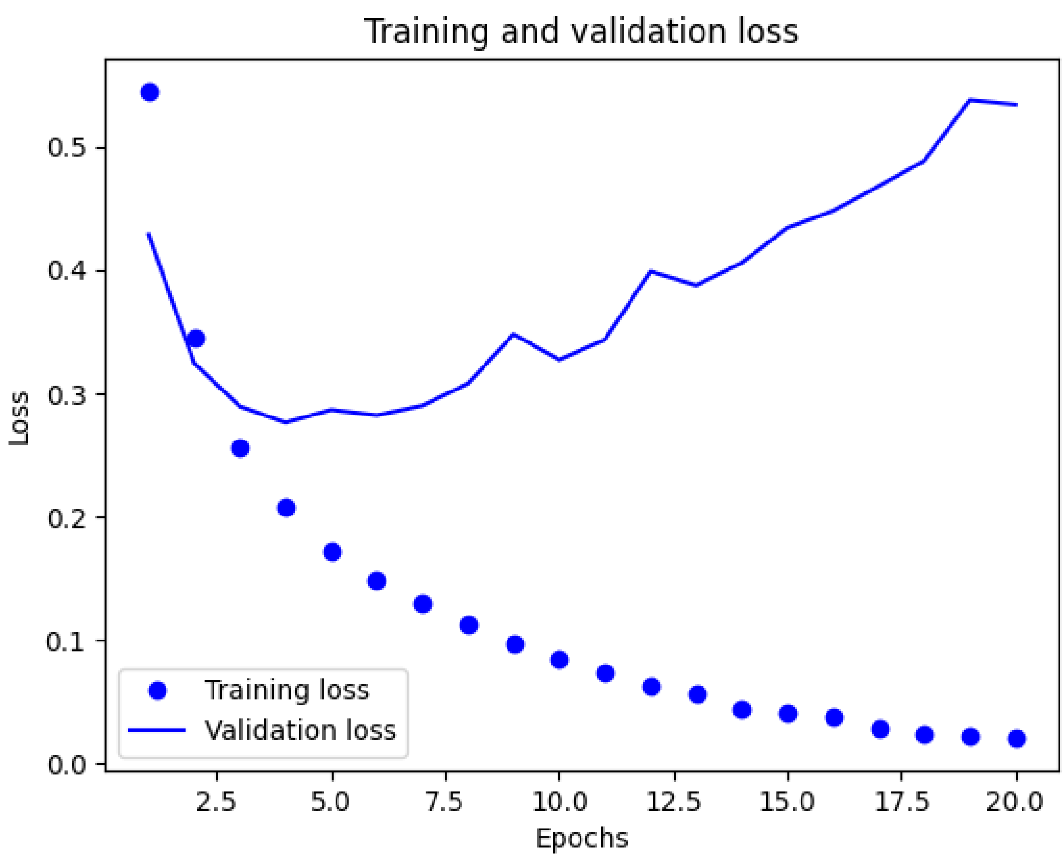
```
        "\n",
        "word_index = imdb.get_word_index()\n",
        "reverse_word_index = dict(\n",
        "    [(value, key) for (key, value) in word_index.items()])\n",
        "decoded_review = \" \".join(\n",
        "    [reverse_word_index.get(i - 3, \"?\") for i in
train_data[0]])\n",
        "\n"
      ],
      "metadata": {
        "id": "Nn86fCFoG54N"
      },
      "execution_count": 8,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#Preparing the data\n",
        "#Encoding the integer sequences via multi-hot encoding\n",
        "\n",
        "import numpy as np\n",
        "def vectorize_sequences(sequences, dimension=10000):\n",
        "    results = np.zeros((len(sequences), dimension))\n",
        "    for i, sequence in enumerate(sequences):\n",
        "        for j in sequence:\n",
        "            results[i, j] = 1.\n",
        "    return results\n",
        "x_train = vectorize_sequences(train_data)\n",
        "x_test = vectorize_sequences(test_data)\n",
        "x_train[0]\n",
        "y_train = np.asarray(train_labels).astype(\"float32\")\n",
        "y_test = np.asarray(test_labels).astype(\"float32\")\n",
        "\n"
      ],
      "metadata": {
        "id": "WRxlqY1THNhV"
      },
      "execution_count": 9,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#Building your model\n",
        "#Model definition\n",
        "\n",
        "from tensorflow import keras\n",
        "from tensorflow.keras import layers\n",
        "\n",
        "model = keras.Sequential([\n",
        "    layers.Dense(16, activation=\"relu\"),\n",
        "    layers.Dense(16, activation=\"relu\"),\n",
        "    layers.Dense(1, activation=\"sigmoid\")\n",
        "])"
      ],
      "metadata": {
        "id": "lxhv9FOeHZcP"
```

```
    },
    "execution_count": 10,
    "outputs": []
  },
  {
    "cell_type": "code",
    "source": [
      "#Compiling the model\n",
      "\n",
      "model.compile(optimizer=\"rmsprop\",\n",
      "              loss=\"binary_crossentropy\",\n",
      "              metrics=[\"accuracy\"])"
    ],
    "metadata": {
      "id": "v_hg597WHh1-"
    },
    "execution_count": 11,
    "outputs": []
  },
  {
    "cell_type": "code",
    "source": [
      "#Validating your approach\n",
      "#Setting aside a validation set\n",
      "\n",
      "x_val = x_train[:10000]\n",
      "partial_x_train = x_train[10000:]\n",
      "y_val = y_train[:10000]\n",
      "partial_y_train = y_train[10000:]"
    ],
    "metadata": {
      "id": "2SQduCS-HpXq"
    },
    "execution_count": 12,
    "outputs": []
  },
  {
    "cell_type": "code",
    "source": [
      "#Training your model\n",
      "\n",
      "history = model.fit(partial_x_train,\n",
      "                    partial_y_train,\n",
      "                    epochs=20,\n",
      "                    batch_size=512,\n",
      "                    validation_data=(x_val, y_val))\n",
      "history_dict = history.history\n",
      "history_dict.keys()"
    ],
    "metadata": {
      "colab": {
        "base_uri": "https://localhost:8080/"
      },
      "id": "_CcyklDsHwRG",
      "outputId": "c339e3ee-fcd1-493b-8810-f7a6752f37d5"
    },
    "execution_count": 13,
    "outputs": [
```

{
  "output_type": "stream",
  "name": "stdout",
  "text": [
    "Epoch 1/20\n",
    "30/30 [==============================] - 6s 189ms/step - loss: 0.5449 - accuracy: 0.7689 - val_loss: 0.4285 - val_accuracy: 0.8573\n",
    "Epoch 2/20\n",
    "30/30 [==============================] - 3s 101ms/step - loss: 0.3452 - accuracy: 0.8911 - val_loss: 0.3242 - val_accuracy: 0.8833\n",
    "Epoch 3/20\n",
    "30/30 [==============================] - 5s 160ms/step - loss: 0.2553 - accuracy: 0.9157 - val_loss: 0.2892 - val_accuracy: 0.8879\n",
    "Epoch 4/20\n",
    "30/30 [==============================] - 1s 39ms/step - loss: 0.2071 - accuracy: 0.9319 - val_loss: 0.2760 - val_accuracy: 0.8889\n",
    "Epoch 5/20\n",
    "30/30 [==============================] - 1s 49ms/step - loss: 0.1721 - accuracy: 0.9443 - val_loss: 0.2864 - val_accuracy: 0.8840\n",
    "Epoch 6/20\n",
    "30/30 [==============================] - 1s 48ms/step - loss: 0.1490 - accuracy: 0.9534 - val_loss: 0.2819 - val_accuracy: 0.8848\n",
    "Epoch 7/20\n",
    "30/30 [==============================] - 1s 40ms/step - loss: 0.1304 - accuracy: 0.9595 - val_loss: 0.2898 - val_accuracy: 0.8857\n",
    "Epoch 8/20\n",
    "30/30 [==============================] - 1s 41ms/step - loss: 0.1123 - accuracy: 0.9662 - val_loss: 0.3076 - val_accuracy: 0.8794\n",
    "Epoch 9/20\n",
    "30/30 [==============================] - 1s 37ms/step - loss: 0.0972 - accuracy: 0.9722 - val_loss: 0.3477 - val_accuracy: 0.8752\n",
    "Epoch 10/20\n",
    "30/30 [==============================] - 1s 37ms/step - loss: 0.0842 - accuracy: 0.9775 - val_loss: 0.3269 - val_accuracy: 0.8811\n",
    "Epoch 11/20\n",
    "30/30 [==============================] - 1s 45ms/step - loss: 0.0727 - accuracy: 0.9803 - val_loss: 0.3434 - val_accuracy: 0.8806\n",
    "Epoch 12/20\n",
    "30/30 [==============================] - 2s 61ms/step - loss: 0.0626 - accuracy: 0.9835 - val_loss: 0.3986 - val_accuracy: 0.8726\n",
    "Epoch 13/20\n",
    "30/30 [==============================] - 1s 35ms/step - loss: 0.0555 - accuracy: 0.9871 - val_loss: 0.3874 - val_accuracy: 0.8759\n",
    "Epoch 14/20\n",

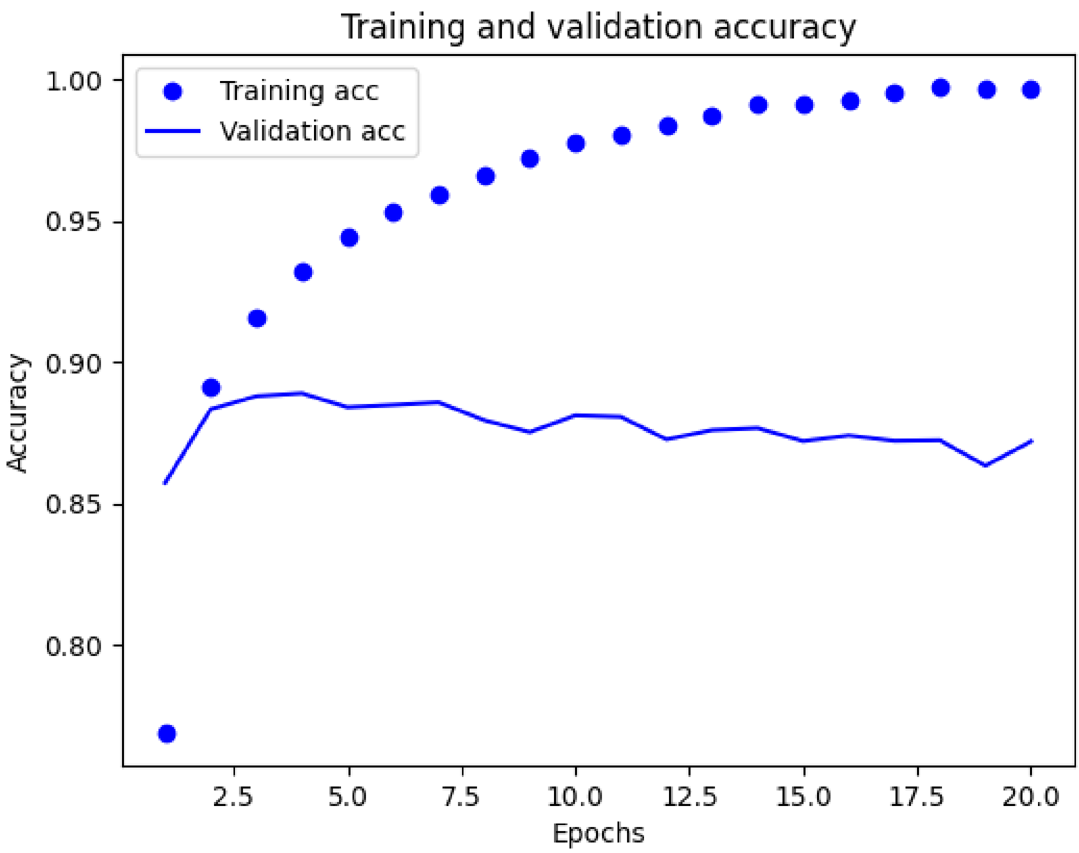
```
            "30/30 [==============================] - 1s 37ms/step -
loss: 0.0444 - accuracy: 0.9915 - val_loss: 0.4056 - val_accuracy:
0.8766\n",
            "Epoch 15/20\n",
            "30/30 [==============================] - 1s 36ms/step -
loss: 0.0410 - accuracy: 0.9913 - val_loss: 0.4340 - val_accuracy:
0.8720\n",
            "Epoch 16/20\n",
            "30/30 [==============================] - 1s 48ms/step -
loss: 0.0370 - accuracy: 0.9923 - val_loss: 0.4477 - val_accuracy:
0.8740\n",
            "Epoch 17/20\n",
            "30/30 [==============================] - 1s 37ms/step -
loss: 0.0280 - accuracy: 0.9951 - val_loss: 0.4677 - val_accuracy:
0.8721\n",
            "Epoch 18/20\n",
            "30/30 [==============================] - 1s 49ms/step -
loss: 0.0234 - accuracy: 0.9974 - val_loss: 0.4883 - val_accuracy:
0.8723\n",
            "Epoch 19/20\n",
            "30/30 [==============================] - 1s 44ms/step -
loss: 0.0219 - accuracy: 0.9968 - val_loss: 0.5377 - val_accuracy:
0.8633\n",
            "Epoch 20/20\n",
            "30/30 [==============================] - 1s 36ms/step -
loss: 0.0205 - accuracy: 0.9969 - val_loss: 0.5339 - val_accuracy:
0.8719\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "dict_keys(['loss', 'accuracy', 'val_loss',
'val_accuracy'])"
            ]
          },
          "metadata": {},
          "execution_count": 13
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Plotting the training and validation loss\n",
        "\n",
        "import matplotlib.pyplot as plt\n",
        "history_dict = history.history\n",
        "loss_values = history_dict[\"loss\"]\n",
        "val_loss_values = history_dict[\"val_loss\"]\n",
        "epochs = range(1, len(loss_values) + 1)\n",
        "plt.plot(epochs, loss_values, \"bo\", label=\"Training
loss\")\n",
        "plt.plot(epochs, val_loss_values, \"b\", label=\"Validation
loss\")\n",
        "plt.title(\"Training and validation loss\")\n",
        "plt.xlabel(\"Epochs\")\n",
```

```
      "plt.ylabel(\"Loss\")\n",
      "plt.legend()\n",
      "plt.show()"
    ],
    "metadata": {
      "colab": {
        "base_uri": "https://localhost:8080/",
        "height": 472
      },
      "id": "1m4UZIBkIAu-",
      "outputId": "8bf2c0c0-28fb-4a7a-8e1b-3d7214b6df56"
    },
    "execution_count": 14,
    "outputs": [
      {
        "output_type": "display_data",
        "data": {
          "text/plain": [
            "<Figure size 640x480 with 1 Axes>"
          ],
          "image/png":
```


```
        },
        "metadata": {}
      }
    ]
  },
  {
    "cell_type": "code",
    "source": [
      "#Plotting the training and validation accuracy\n",
```

```
    "\n",
    "plt.clf()\n",
    "acc = history_dict[\"accuracy\"]\n",
    "val_acc = history_dict[\"val_accuracy\"]\n",
    "plt.plot(epochs, acc, \"bo\", label=\"Training acc\")\n",
    "plt.plot(epochs, val_acc, \"b\", label=\"Validation acc\")\n",
    "plt.title(\"Training and validation accuracy\")\n",
    "plt.xlabel(\"Epochs\")\n",
    "plt.ylabel(\"Accuracy\")\n",
    "plt.legend()\n",
    "plt.show()"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 472
    },
    "id": "5cKQhfjfIHQP",
    "outputId": "cede21b0-a47b-4fc5-9346-10cc359e7ac8"
  },
  "execution_count": 15,
  "outputs": [
    {
      "output_type": "display_data",
      "data": {
        "text/plain": [
          "<Figure size 640x480 with 1 Axes>"
        ],
        "image/png":
```


```
      },
```

```
          "metadata": {}
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Retraining a model from scratch\n",
        "\n",
        "model = keras.Sequential([\n",
        "    layers.Dense(16, activation=\"relu\"),\n",
        "    layers.Dense(16, activation=\"relu\"),\n",
        "    layers.Dense(1, activation=\"sigmoid\")\n",
        "])\n",
        "model.compile(optimizer=\"rmsprop\",\n",
        "              loss=\"binary_crossentropy\",\n",
        "              metrics=[\"accuracy\"])\n",
        "model.fit(x_train, y_train, epochs=4, batch_size=512)\n",
        "results = model.evaluate(x_test, y_test)\n",
        "results"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "O0ICaPBOINZB",
        "outputId": "f6c621dd-821e-4ddd-fe00-e5b3796a5d31"
      },
      "execution_count": 16,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Epoch 1/4\n",
            "49/49 [==============================] - 3s 28ms/step -
loss: 0.4970 - accuracy: 0.8015\n",
            "Epoch 2/4\n",
            "49/49 [==============================] - 2s 31ms/step -
loss: 0.2880 - accuracy: 0.8977\n",
            "Epoch 3/4\n",
            "49/49 [==============================] - 2s 39ms/step -
loss: 0.2231 - accuracy: 0.9191\n",
            "Epoch 4/4\n",
            "49/49 [==============================] - 1s 29ms/step -
loss: 0.1879 - accuracy: 0.9328\n",
            "782/782 [==============================] - 2s 3ms/step -
loss: 0.2890 - accuracy: 0.8852\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "[0.2890424132347107, 0.8851600289344788]"
            ]
          },
          "metadata": {},
```

```
          "execution_count": 16
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Using a trained model to generate predictions on new data\n",
        "\n",
        "model.predict(x_test)"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "yOQyA22aIZHc",
        "outputId": "8ca9b679-299e-43ab-ff30-339ff397e78d"
      },
      "execution_count": 17,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "782/782 [==============================] - 3s 2ms/step\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "array([[0.22125062],\n",
              "       [0.9995276 ],\n",
              "       [0.8962404 ],\n",
              "       ...,\n",
              "       [0.09870885],\n",
              "       [0.11608151],\n",
              "       [0.667172  ]], dtype=float32)"
            ]
          },
          "metadata": {},
          "execution_count": 17
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Further experiments\n",
        "#Wrapping up\n",
        "#Classifying newswires: A multiclass classification example\n",
        "#The Reuters dataset\n",
        "#Loading the Reuters dataset\n",
        "\n",
        "from tensorflow.keras.datasets import reuters\n",
        "(train_data, train_labels), (test_data, test_labels) =
reuters.load_data(\n",
        "    num_words=10000)\n",
```

```
        "len(train_data)\n",
        "len(test_data)\n",
        "train_data[10]"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "cR9E18x8Iged",
        "outputId": "42d1d52e-ce20-48f7-c27b-6857c84a22fe"
      },
      "execution_count": 18,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-
datasets/reuters.npz\n",
            "2110848/2110848 [==============================] - 0s
0us/step\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "[1,\n",
              " 245,\n",
              " 273,\n",
              " 207,\n",
              " 156,\n",
              " 53,\n",
              " 74,\n",
              " 160,\n",
              " 26,\n",
              " 14,\n",
              " 46,\n",
              " 296,\n",
              " 26,\n",
              " 39,\n",
              " 74,\n",
              " 2979,\n",
              " 3554,\n",
              " 14,\n",
              " 46,\n",
              " 4689,\n",
              " 4329,\n",
              " 86,\n",
              " 61,\n",
              " 3499,\n",
              " 4795,\n",
              " 14,\n",
              " 61,\n",
              " 451,\n",
              " 4329,\n",
              " 17,\n",
```

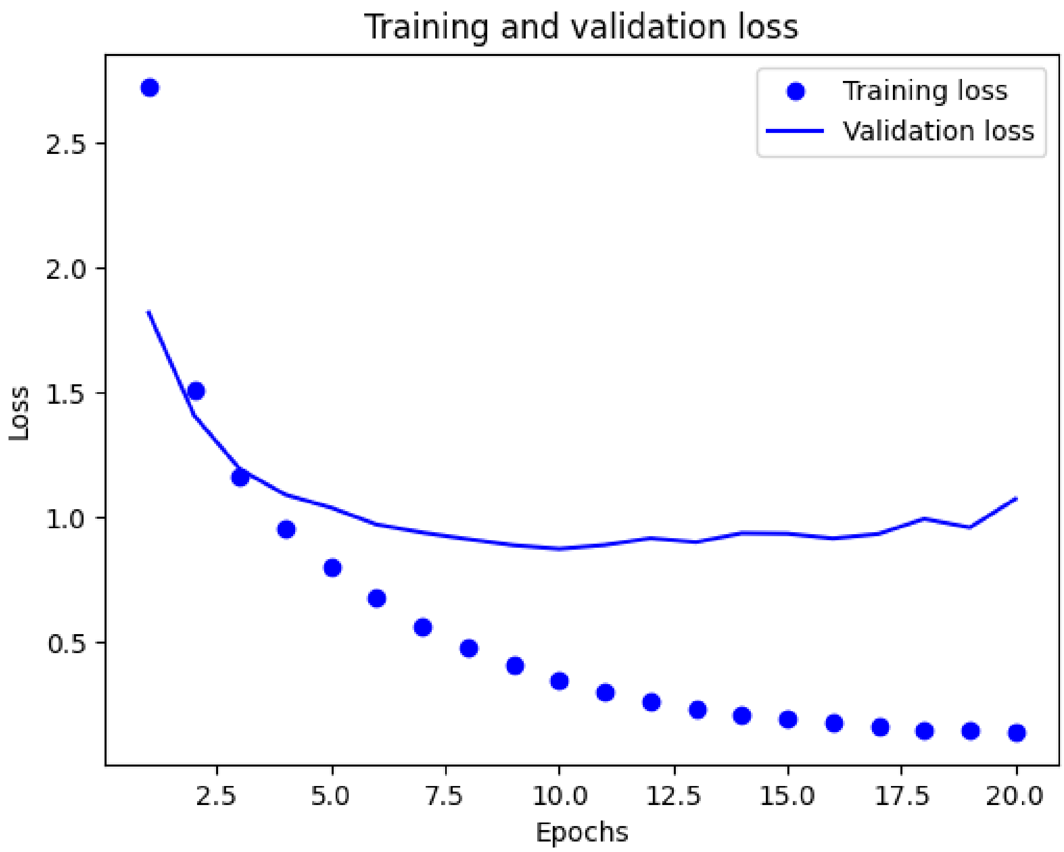
```
            " 12]"
          ]
        },
        "metadata": {},
        "execution_count": 18
      }
    ]
  },
  {
    "cell_type": "code",
    "source": [
      "#Decoding newswires back to text\n",
      "\n",
      "word_index = reuters.get_word_index()\n",
      "reverse_word_index = dict([(value, key) for (key, value) in
word_index.items()])\n",
      "decoded_newswire = \" \".join([reverse_word_index.get(i - 3,
\"?\") for i in\n",
      "    train_data[0]])\n",
      "train_labels[10]"
    ],
    "metadata": {
      "colab": {
        "base_uri": "https://localhost:8080/"
      },
      "id": "bWR-C-YyIpg3",
      "outputId": "21f4b492-a34b-4ca0-c7b4-207327538959"
    },
    "execution_count": 19,
    "outputs": [
      {
        "output_type": "stream",
        "name": "stdout",
        "text": [
          "Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-
datasets/reuters_word_index.json\n",
          "550378/550378 [==============================] - 0s
0us/step\n"
        ]
      },
      {
        "output_type": "execute_result",
        "data": {
          "text/plain": [
            "3"
          ]
        },
        "metadata": {},
        "execution_count": 19
      }
    ]
  },
  {
    "cell_type": "code",
    "source": [
      "#Preparing the data\n",
      "#Encoding the input data\n",
```

```
    "\n",
    "x_train = vectorize_sequences(train_data)\n",
    "x_test = vectorize_sequences(test_data)"
  ],
  "metadata": {
    "id": "zCJeAyNHIwgg"
  },
  "execution_count": 20,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "#Encoding the labels\n",
    "\n",
    "def to_one_hot(labels, dimension=46):\n",
    "    results = np.zeros((len(labels), dimension))\n",
    "    for i, label in enumerate(labels):\n",
    "        results[i, label] = 1.\n",
    "    return results\n",
    "y_train = to_one_hot(train_labels)\n",
    "y_test = to_one_hot(test_labels)\n",
    "from tensorflow.keras.utils import to_categorical\n",
    "y_train = to_categorical(train_labels)\n",
    "y_test = to_categorical(test_labels)"
  ],
  "metadata": {
    "id": "HbVXiIIZI3QE"
  },
  "execution_count": 21,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "#Building your model\n",
    "#Model definition\n",
    "\n",
    "model = keras.Sequential([\n",
    "    layers.Dense(64, activation=\"relu\"),\n",
    "    layers.Dense(64, activation=\"relu\"),\n",
    "    layers.Dense(46, activation=\"softmax\")\n",
    "])\n",
    "\n",
    "#Compiling the model\n",
    "\n",
    "model.compile(optimizer=\"rmsprop\",\n",
    "              loss=\"categorical_crossentropy\",\n",
    "              metrics=[\"accuracy\"])"
  ],
  "metadata": {
    "id": "4Njk2R6BJAJz"
  },
  "execution_count": 23,
  "outputs": []
},
{
  "cell_type": "code",
```

```
      "source": [
        "#Validating your approach\n",
        "#Setting aside a validation set\n",
        "\n",
        "x_val = x_train[:1000]\n",
        "partial_x_train = x_train[1000:]\n",
        "y_val = y_train[:1000]\n",
        "partial_y_train = y_train[1000:]"
      ],
      "metadata": {
        "id": "0rAKlfmqJPKT"
      },
      "execution_count": 25,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#Training the model\n",
        "\n",
        "history = model.fit(partial_x_train,\n",
        "                    partial_y_train,\n",
        "                    epochs=20,\n",
        "                    batch_size=512,\n",
        "                    validation_data=(x_val, y_val))"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "5POTOcFJJVwb",
        "outputId": "8ce3d5c4-adeb-4906-b5f8-a06462178442"
      },
      "execution_count": 26,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Epoch 1/20\n",
            "16/16 [==============================] - 2s 71ms/step -
loss: 2.7257 - accuracy: 0.5140 - val_loss: 1.8193 - val_accuracy:
0.6220\n",
            "Epoch 2/20\n",
            "16/16 [==============================] - 1s 54ms/step -
loss: 1.5123 - accuracy: 0.6855 - val_loss: 1.4070 - val_accuracy:
0.6740\n",
            "Epoch 3/20\n",
            "16/16 [==============================] - 1s 53ms/step -
loss: 1.1652 - accuracy: 0.7451 - val_loss: 1.1919 - val_accuracy:
0.7350\n",
            "Epoch 4/20\n",
            "16/16 [==============================] - 1s 52ms/step -
loss: 0.9570 - accuracy: 0.7907 - val_loss: 1.0894 - val_accuracy:
0.7610\n",
            "Epoch 5/20\n",
```

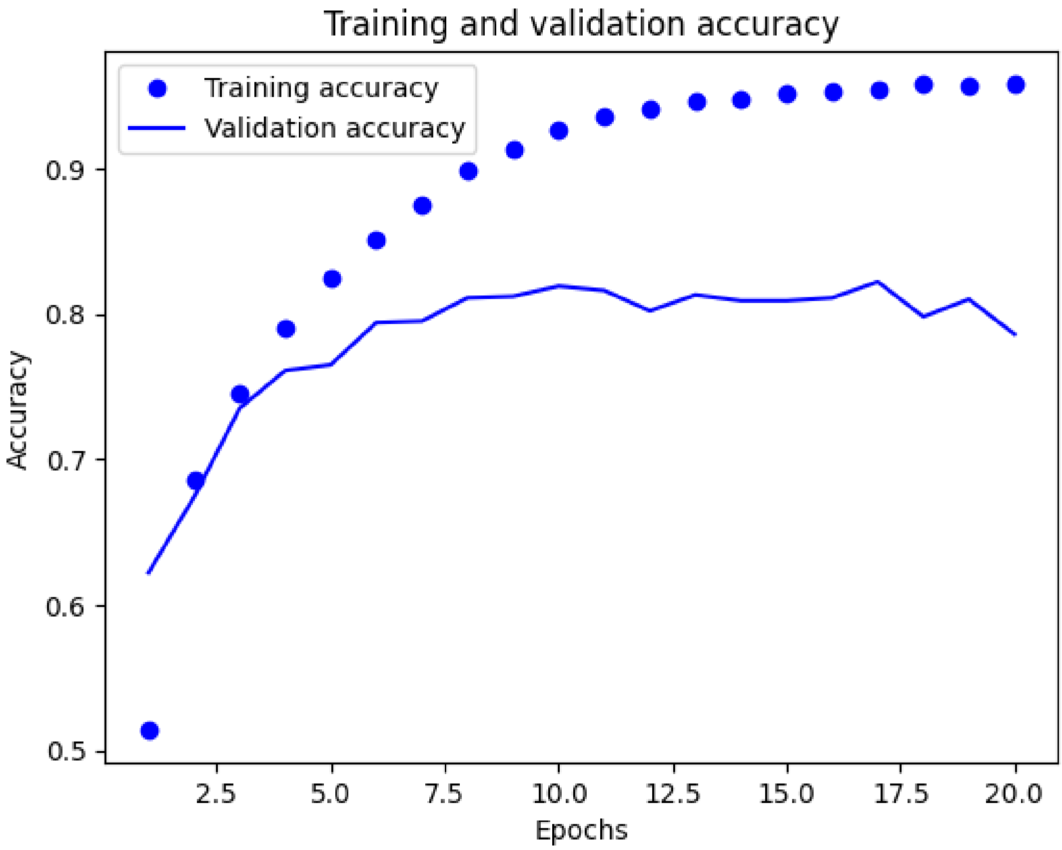
"16/16 [==============================] - 1s 57ms/step - loss: 0.8010 - accuracy: 0.8249 - val_loss: 1.0376 - val_accuracy: 0.7650\n",
"Epoch 6/20\n",
"16/16 [==============================] - 1s 83ms/step - loss: 0.6772 - accuracy: 0.8510 - val_loss: 0.9701 - val_accuracy: 0.7940\n",
"Epoch 7/20\n",
"16/16 [==============================] - 1s 86ms/step - loss: 0.5626 - accuracy: 0.8746 - val_loss: 0.9376 - val_accuracy: 0.7950\n",
"Epoch 8/20\n",
"16/16 [==============================] - 1s 64ms/step - loss: 0.4765 - accuracy: 0.8991 - val_loss: 0.9115 - val_accuracy: 0.8110\n",
"Epoch 9/20\n",
"16/16 [==============================] - 1s 53ms/step - loss: 0.4073 - accuracy: 0.9136 - val_loss: 0.8871 - val_accuracy: 0.8120\n",
"Epoch 10/20\n",
"16/16 [==============================] - 1s 50ms/step - loss: 0.3449 - accuracy: 0.9270 - val_loss: 0.8728 - val_accuracy: 0.8190\n",
"Epoch 11/20\n",
"16/16 [==============================] - 1s 52ms/step - loss: 0.2991 - accuracy: 0.9356 - val_loss: 0.8881 - val_accuracy: 0.8160\n",
"Epoch 12/20\n",
"16/16 [==============================] - 1s 50ms/step - loss: 0.2632 - accuracy: 0.9416 - val_loss: 0.9152 - val_accuracy: 0.8020\n",
"Epoch 13/20\n",
"16/16 [==============================] - 1s 50ms/step - loss: 0.2321 - accuracy: 0.9463 - val_loss: 0.8996 - val_accuracy: 0.8130\n",
"Epoch 14/20\n",
"16/16 [==============================] - 1s 51ms/step - loss: 0.2109 - accuracy: 0.9470 - val_loss: 0.9353 - val_accuracy: 0.8090\n",
"Epoch 15/20\n",
"16/16 [==============================] - 1s 51ms/step - loss: 0.1900 - accuracy: 0.9513 - val_loss: 0.9334 - val_accuracy: 0.8090\n",
"Epoch 16/20\n",
"16/16 [==============================] - 1s 49ms/step - loss: 0.1765 - accuracy: 0.9529 - val_loss: 0.9147 - val_accuracy: 0.8110\n",
"Epoch 17/20\n",
"16/16 [==============================] - 1s 51ms/step - loss: 0.1622 - accuracy: 0.9540 - val_loss: 0.9326 - val_accuracy: 0.8220\n",
"Epoch 18/20\n",
"16/16 [==============================] - 1s 51ms/step - loss: 0.1491 - accuracy: 0.9577 - val_loss: 0.9928 - val_accuracy: 0.7980\n",
"Epoch 19/20\n",

```
            "16/16 [==============================] - 1s 52ms/step - loss: 0.1458 - accuracy: 0.9563 - val_loss: 0.9580 - val_accuracy: 0.8100\n",
            "Epoch 20/20\n",
            "16/16 [==============================] - 1s 61ms/step - loss: 0.1391 - accuracy: 0.9583 - val_loss: 1.0725 - val_accuracy: 0.7860\n"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Plotting the training and validation loss\n",
        "\n",
        "loss = history.history[\"loss\"]\n",
        "val_loss = history.history[\"val_loss\"]\n",
        "epochs = range(1, len(loss) + 1)\n",
        "plt.plot(epochs, loss, \"bo\", label=\"Training loss\")\n",
        "plt.plot(epochs, val_loss, \"b\", label=\"Validation loss\")\n",
        "plt.title(\"Training and validation loss\")\n",
        "plt.xlabel(\"Epochs\")\n",
        "plt.ylabel(\"Loss\")\n",
        "plt.legend()\n",
        "plt.show()"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 472
        },
        "id": "ZymszwCaJ7TN",
        "outputId": "3b0c2539-da23-48ba-f48a-72276cd17282"
      },
      "execution_count": 29,
      "outputs": [
        {
          "output_type": "display_data",
          "data": {
            "text/plain": [
              "<Figure size 640x480 with 1 Axes>"
            ],
            "image/png":
```




```
          },
          "metadata": {}
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Plotting the training and validation accuracy\n",
        "\n",
        "plt.clf()\n",
        "acc = history.history[\"accuracy\"]\n",
        "val_acc = history.history[\"val_accuracy\"]\n",
        "plt.plot(epochs, acc, \"bo\", label=\"Training accuracy\")\n",
        "plt.plot(epochs, val_acc, \"b\", label=\"Validation
accuracy\")\n",
        "plt.title(\"Training and validation accuracy\")\n",
        "plt.xlabel(\"Epochs\")\n",
        "plt.ylabel(\"Accuracy\")\n",
        "plt.legend()\n",
        "plt.show()"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 472
        },
        "id": "h0oTBzCQLRda",
        "outputId": "bb4e6a26-e49e-413d-d58b-9a04f1af68cd"
      },
```

```
  "execution_count": 31,
  "outputs": [
    {
      "output_type": "display_data",
      "data": {
        "text/plain": [
          "<Figure size 640x480 with 1 Axes>"
        ],
        "image/png":
```


```
      },
      "metadata": {}
    }
  ]
},
{
  "cell_type": "code",
  "source": [
    "#Retraining a model from scratch\n",
    "\n",
    "model = keras.Sequential([\n",
    "   layers.Dense(64, activation=\"relu\"),\n",
    "   layers.Dense(64, activation=\"relu\"),\n",
    "   layers.Dense(46, activation=\"softmax\")\n",
    "])\n",
    "model.compile(optimizer=\"rmsprop\",\n",
    "              loss=\"categorical_crossentropy\",\n",
    "              metrics=[\"accuracy\"])\n",
    "model.fit(x_train,\n",
    "          y_train,\n",
```

```
        "          epochs=9,\n",
        "          batch_size=512)\n",
        "results = model.evaluate(x_test, y_test)\n",
        "results\n",
        "\n",
        "import copy\n",
        "test_labels_copy = copy.copy(test_labels)\n",
        "np.random.shuffle(test_labels_copy)\n",
        "hits_array = np.array(test_labels) ==
np.array(test_labels_copy)\n",
        "hits_array.mean()"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "ZMq7xWD1LXuZ",
        "outputId": "12b6d6a1-bb3f-47ab-f704-cb568a2f8486"
      },
      "execution_count": 33,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Epoch 1/9\n",
            "18/18 [==============================] - 2s 56ms/step -
loss: 2.6564 - accuracy: 0.5148\n",
            "Epoch 2/9\n",
            "18/18 [==============================] - 1s 49ms/step -
loss: 1.4806 - accuracy: 0.6819\n",
            "Epoch 3/9\n",
            "18/18 [==============================] - 1s 51ms/step -
loss: 1.1260 - accuracy: 0.7503\n",
            "Epoch 4/9\n",
            "18/18 [==============================] - 1s 60ms/step -
loss: 0.9199 - accuracy: 0.8012\n",
            "Epoch 5/9\n",
            "18/18 [==============================] - 1s 48ms/step -
loss: 0.7598 - accuracy: 0.8342\n",
            "Epoch 6/9\n",
            "18/18 [==============================] - 1s 50ms/step -
loss: 0.6312 - accuracy: 0.8642\n",
            "Epoch 7/9\n",
            "18/18 [==============================] - 1s 46ms/step -
loss: 0.5254 - accuracy: 0.8868\n",
            "Epoch 8/9\n",
            "18/18 [==============================] - 1s 62ms/step -
loss: 0.4435 - accuracy: 0.9063\n",
            "Epoch 9/9\n",
            "18/18 [==============================] - 1s 75ms/step -
loss: 0.3725 - accuracy: 0.9194\n",
            "71/71 [==============================] - 1s 5ms/step - loss:
0.9168 - accuracy: 0.7890\n"
          ]
        },
        {
          "output_type": "execute_result",
```

```
          "data": {
            "text/plain": [
              "0.19323241317898487"
            ]
          },
          "metadata": {},
          "execution_count": 33
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Generating predictions on new data\n",
        "predictions = model.predict(x_test)\n",
        "predictions[0].shape\n",
        "np.sum(predictions[0])\n",
        "np.argmax(predictions[0])"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "vTxyLfxHLu7O",
        "outputId": "eb1fbbef-3e2d-4ce9-962a-5ae4eac1827b"
      },
      "execution_count": 34,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "71/71 [==============================] - 0s 4ms/step\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "3"
            ]
          },
          "metadata": {},
          "execution_count": 34
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#A different way to handle the labels and the loss\n",
        "y_train = np.array(train_labels)\n",
        "y_test = np.array(test_labels)\n",
        "model.compile(optimizer=\"rmsprop\",\n",
        "              loss=\"sparse_categorical_crossentropy\",\n",
        "              metrics=[\"accuracy\"])"
      ],
      "metadata": {
```

```
        "id": "LiSmsJwYMRLz"
      },
      "execution_count": 35,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#The importance of having sufficiently large intermediate
layers\n",
        "#A model with an information bottleneck\n",
        "\n",
        "model = keras.Sequential([\n",
        "    layers.Dense(64, activation=\"relu\"),\n",
        "    layers.Dense(4, activation=\"relu\"),\n",
        "    layers.Dense(46, activation=\"softmax\")\n",
        "])\n",
        "model.compile(optimizer=\"rmsprop\",\n",
        "              loss=\"categorical_crossentropy\",\n",
        "              metrics=[\"accuracy\"])\n",
        "model.fit(partial_x_train,\n",
        "          partial_y_train,\n",
        "          epochs=20,\n",
        "          batch_size=128,\n",
        "          validation_data=(x_val, y_val))"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "ggwoAPLkMUKV",
        "outputId": "036e62ef-c52a-4295-81b4-0754cee9ff7c"
      },
      "execution_count": 36,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Epoch 1/20\n",
            "63/63 [==============================] - 2s 23ms/step -
loss: 2.6702 - accuracy: 0.2752 - val_loss: 2.0430 - val_accuracy:
0.5140\n",
            "Epoch 2/20\n",
            "63/63 [==============================] - 1s 19ms/step -
loss: 1.7775 - accuracy: 0.5678 - val_loss: 1.6408 - val_accuracy:
0.5790\n",
            "Epoch 3/20\n",
            "63/63 [==============================] - 1s 19ms/step -
loss: 1.4859 - accuracy: 0.5986 - val_loss: 1.5342 - val_accuracy:
0.5890\n",
            "Epoch 4/20\n",
            "63/63 [==============================] - 1s 18ms/step -
loss: 1.3407 - accuracy: 0.6250 - val_loss: 1.4276 - val_accuracy:
0.6150\n",
            "Epoch 5/20\n",
```

```
            "63/63 [==============================] - 1s 17ms/step - 
loss: 1.2326 - accuracy: 0.6555 - val_loss: 1.3840 - val_accuracy: 
0.6370\n",
            "Epoch 6/20\n",
            "63/63 [==============================] - 1s 23ms/step - 
loss: 1.1496 - accuracy: 0.6788 - val_loss: 1.3693 - val_accuracy: 
0.6650\n",
            "Epoch 7/20\n",
            "63/63 [==============================] - 2s 27ms/step - 
loss: 1.0819 - accuracy: 0.6956 - val_loss: 1.3408 - val_accuracy: 
0.6520\n",
            "Epoch 8/20\n",
            "63/63 [==============================] - 1s 19ms/step - 
loss: 1.0185 - accuracy: 0.7209 - val_loss: 1.3346 - val_accuracy: 
0.6840\n",
            "Epoch 9/20\n",
            "63/63 [==============================] - 1s 19ms/step - 
loss: 0.9660 - accuracy: 0.7453 - val_loss: 1.3311 - val_accuracy: 
0.6870\n",
            "Epoch 10/20\n",
            "63/63 [==============================] - 1s 17ms/step - 
loss: 0.9163 - accuracy: 0.7615 - val_loss: 1.3240 - val_accuracy: 
0.6880\n",
            "Epoch 11/20\n",
            "63/63 [==============================] - 1s 17ms/step - 
loss: 0.8759 - accuracy: 0.7667 - val_loss: 1.3629 - val_accuracy: 
0.6850\n",
            "Epoch 12/20\n",
            "63/63 [==============================] - 1s 18ms/step - 
loss: 0.8401 - accuracy: 0.7762 - val_loss: 1.3709 - val_accuracy: 
0.6860\n",
            "Epoch 13/20\n",
            "63/63 [==============================] - 1s 17ms/step - 
loss: 0.8076 - accuracy: 0.7849 - val_loss: 1.3758 - val_accuracy: 
0.6860\n",
            "Epoch 14/20\n",
            "63/63 [==============================] - 1s 17ms/step - 
loss: 0.7794 - accuracy: 0.7883 - val_loss: 1.4144 - val_accuracy: 
0.6850\n",
            "Epoch 15/20\n",
            "63/63 [==============================] - 1s 18ms/step - 
loss: 0.7538 - accuracy: 0.7929 - val_loss: 1.4538 - val_accuracy: 
0.6800\n",
            "Epoch 16/20\n",
            "63/63 [==============================] - 1s 18ms/step - 
loss: 0.7292 - accuracy: 0.7977 - val_loss: 1.5199 - val_accuracy: 
0.6740\n",
            "Epoch 17/20\n",
            "63/63 [==============================] - 2s 26ms/step - 
loss: 0.7077 - accuracy: 0.7993 - val_loss: 1.4769 - val_accuracy: 
0.6880\n",
            "Epoch 18/20\n",
            "63/63 [==============================] - 2s 27ms/step - 
loss: 0.6892 - accuracy: 0.8013 - val_loss: 1.5256 - val_accuracy: 
0.6750\n",
            "Epoch 19/20\n",
```

```
            "63/63 [==============================] - 1s 17ms/step -
loss: 0.6699 - accuracy: 0.8028 - val_loss: 1.5625 - val_accuracy:
0.6880\n",
            "Epoch 20/20\n",
            "63/63 [==============================] - 1s 17ms/step -
loss: 0.6560 - accuracy: 0.8091 - val_loss: 1.6277 - val_accuracy:
0.6910\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "<keras.src.callbacks.History at 0x7ec1ae173c70>"
            ]
          },
          "metadata": {},
          "execution_count": 36
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Further experiments\n",
        "#Wrapping up\n",
        "#Predicting house prices: A regression example\n",
        "#The Boston Housing Price dataset\n",
        "#Loading the Boston housing dataset\n",
        "\n",
        "from tensorflow.keras.datasets import boston_housing\n",
        "(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()\n",
        "train_data.shape\n",
        "test_data.shape\n",
        "train_targets"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "-w3coJ60kF0U",
        "outputId": "7c0724b5-1ffc-4cbf-8e13-df9badc80cc6"
      },
      "execution_count": 1,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-
datasets/boston_housing.npz\n",
            "57026/57026 [==============================] - 0s
0us/step\n"
          ]
        },
        {
```

"output_type": "execute_result",
"data": {
"text/plain": [
"array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6,
15.6, 14.4, 12.1,\n",
"   17.9, 23.1, 19.9, 15.7, 8.8, 50. , 22.5, 24.1,
27.5, 10.9, 30.8,\n",
"   32.9, 24. , 18.5, 13.3, 22.9, 34.7, 16.6, 17.5,
22.3, 16.1, 14.9,\n",
"   23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2,
24.7, 22.2, 16.7,\n",
"   12.7, 15.6, 18.4, 21. , 30.1, 15.1, 18.7, 9.6,
31.5, 24.8, 19.1,\n",
"   22. , 14.5, 11. , 32. , 29.4, 20.3, 24.4, 14.6,
19.5, 14.1, 14.3,\n",
"   15.6, 10.5, 6.3, 19.3, 19.3, 13.4, 36.4, 17.8,
13.5, 16.5, 8.3,\n",
"   14.3, 16. , 13.4, 28.6, 43.5, 20.2, 22. , 23. ,
20.7, 12.5, 48.5,\n",
"   14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2,
34.9, 22.5, 31.1,\n",
"   28.7, 46. , 41.7, 21. , 26.6, 15. , 24.4, 13.3,
21.2, 11.7, 21.7,\n",
"   19.4, 50. , 22.8, 19.7, 24.7, 36.2, 14.2, 18.9,
18.3, 20.6, 24.6,\n",
"   18.2, 8.7, 44. , 10.4, 13.2, 21.2, 37. , 30.7,
22.9, 20. , 19.3,\n",
"   31.7, 32. , 23.1, 18.8, 10.9, 50. , 19.6, 5. ,
14.4, 19.8, 13.8,\n",
"   19.6, 23.9, 24.5, 25. , 19.9, 17.2, 24.6, 13.5,
26.6, 21.4, 11.9,\n",
"   22.6, 19.6, 8.5, 23.7, 23.1, 22.4, 20.5, 23.6,
18.4, 35.2, 23.1,\n",
"   27.9, 20.6, 23.7, 28. , 13.6, 27.1, 23.6, 20.6,
18.2, 21.7, 17.1,\n",
"   8.4, 25.3, 13.8, 22.2, 18.4, 20.7, 31.6, 30.5,
20.3, 8.8, 19.2,\n",
"   19.4, 23.1, 23. , 14.8, 48.8, 22.6, 33.4, 21.1,
13.6, 32.2, 13.1,\n",
"   23.4, 18.9, 23.9, 11.8, 23.3, 22.8, 19.6, 16.7,
13.4, 22.2, 20.4,\n",
"   21.8, 26.4, 14.9, 24.1, 23.8, 12.3, 29.1, 21. ,
19.5, 23.3, 23.8,\n",
"   17.8, 11.5, 21.7, 19.9, 25. , 33.4, 28.5, 21.4,
24.3, 27.5, 33.1,\n",
"   16.2, 23.3, 48.3, 22.9, 22.8, 13.1, 12.7, 22.6, 15.
, 15.3, 10.5,\n",
"   24. , 18.5, 21.7, 19.5, 33.2, 23.2, 5. , 19.1,
12.7, 22.3, 10.2,\n",
"   13.9, 16.3, 17. , 20.1, 29.9, 17.2, 37.3, 45.4,
17.8, 23.2, 29. ,\n",
"   22. , 18. , 17.4, 34.6, 20.1, 25. , 15.6, 24.8,
28.2, 21.2, 21.4,\n",
"   23.8, 31. , 26.2, 17.4, 37.9, 17.5, 20. , 8.3,
23.9, 8.4, 13.8,\n",
"   7.2, 11.7, 17.1, 21.6, 50. , 16.1, 20.4, 20.6,
21.4, 20.6, 36.5,\n",

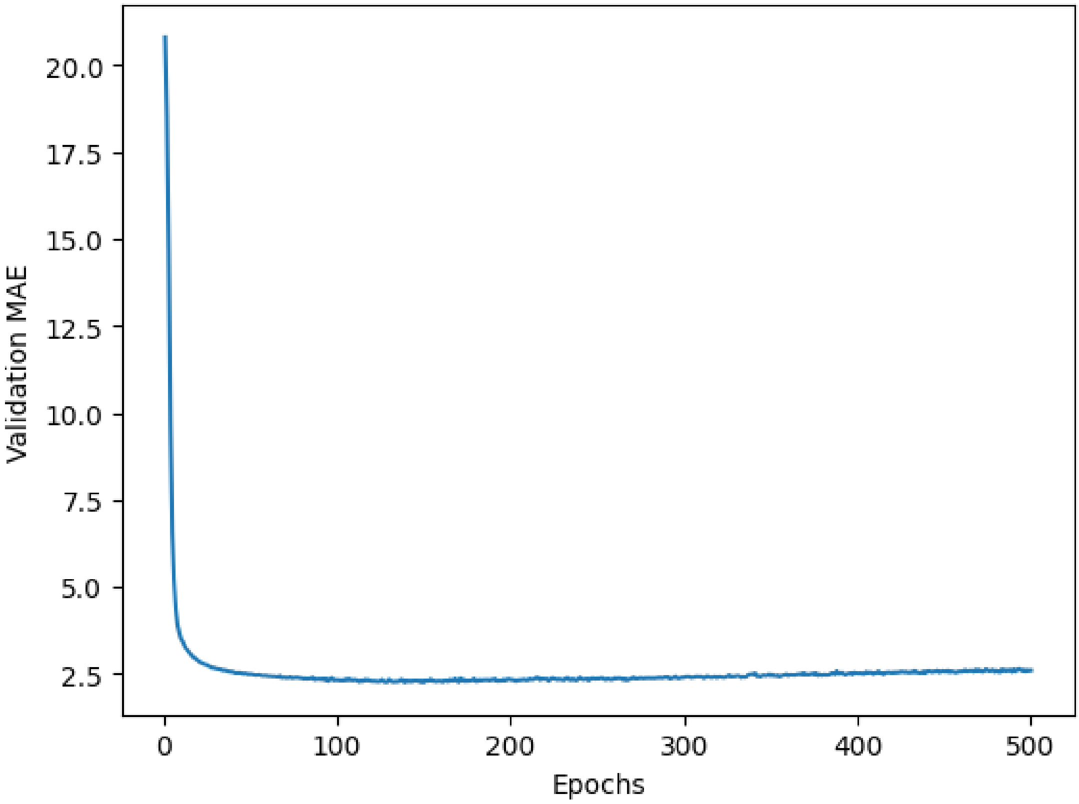
```
              "        8.5, 24.8, 10.8, 21.9, 17.3, 18.9, 36.2, 14.9,
18.2, 33.3, 21.8,\n",
              "       19.7, 31.6, 24.8, 19.4, 22.8,  7.5, 44.8, 16.8,
18.7, 50. , 50. ,\n",
              "       19.5, 20.1, 50. , 17.2, 20.8, 19.3, 41.3, 20.4,
20.5, 13.8, 16.5,\n",
              "       23.9, 20.6, 31.5, 23.3, 16.8, 14. , 33.8, 36.1,
12.8, 18.3, 18.7,\n",
              "       19.1, 29. , 30.1, 50. , 50. , 22. , 11.9, 37.6, 50.
, 22.7, 20.8,\n",
              "       23.5, 27.9, 50. , 19.3, 23.9, 22.6, 15.2, 21.7,
19.2, 43.8, 20.3,\n",
              "       33.2, 19.9, 22.5, 32.7, 22. , 17.1, 19. , 15. ,
16.1, 25.1, 23.7,\n",
              "       28.7, 37.2, 22.6, 16.4, 25. , 29.8, 22.1, 17.4,
18.1, 30.3, 17.5,\n",
              "       24.7, 12.6, 26.5, 28.7, 13.3, 10.4, 24.4, 23. , 20.
, 17.8,  7. ,\n",
              "       11.8, 24.4, 13.8, 19.4, 25.2, 19.4, 19.4, 29.1])"
            ]
          },
          "metadata": {},
          "execution_count": 1
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Preparing the data\n",
        "#Normalizing the data\n",
        "\n",
        "mean = train_data.mean(axis=0)\n",
        "train_data -= mean\n",
        "std = train_data.std(axis=0)\n",
        "train_data /= std\n",
        "test_data -= mean\n",
        "test_data /= std"
      ],
      "metadata": {
        "id": "ra413hlekUL_"
      },
      "execution_count": 2,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#Building your model\n",
        "#Model definition\n",
        "\n",
        "def build_model():\n",
        "    model = keras.Sequential([\n",
        "        layers.Dense(64, activation=\"relu\"),\n",
        "        layers.Dense(64, activation=\"relu\"),\n",
        "        layers.Dense(1)\n",
        "    ])\n",
```

```
        "    model.compile(optimizer=\"rmsprop\", loss=\"mse\",
metrics=[\"mae\"])\n",
        "    return model"
      ],
      "metadata": {
        "id": "OZ3AebMVkaxL"
      },
      "execution_count": 3,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#Validating your approach using K-fold validation\n",
        "#K-fold validation\n",
        "!pip install numpy\n",
        "import numpy as np\n",
        "\n",
        "\n",
        "k = 4\n",
        "num_val_samples = len(train_data) // k\n",
        "num_epochs = 100\n",
        "all_scores = []\n",
        "\n",
        "for i in range(k):\n",
        "    print(f\"Processing fold #{i}\")\n",
        "\n",
        "    val_data = train_data[i * num_val_samples: (i + 1) *
num_val_samples]\n",
        "    val_targets = train_targets[i * num_val_samples: (i + 1) *
num_val_samples]\n",
        "\n",
        "    partial_train_data = np.concatenate(\n",
        "        [train_data[:i * num_val_samples],\n",
        "         train_data[(i + 1) * num_val_samples:]],\n",
        "        axis=0)\n",
        "\n",
        "    partial_train_targets = np.concatenate(\n",
        "        [train_targets[:i * num_val_samples],\n",
        "         train_targets[(i + 1) * num_val_samples:]],\n",
        "        axis=0)\n",
        "\n",
        "import numpy as np\n",
        "from tensorflow import keras\n",
        "from tensorflow.keras import layers\n",
        "\n",
        "# Rest of your code...\n",
        "\n",
        "def build_model():\n",
        "    model = keras.Sequential([\n",
        "        layers.Dense(64, activation=\"relu\"),\n",
        "        layers.Dense(64, activation=\"relu\"),\n",
        "        layers.Dense(1)  # Adjust the number of units according
to your problem\n",
        "    ])\n",
        "\n",
        "    model.compile(optimizer='adam', loss='mse',
metrics=['mae'])\n",
```

```
        "\n",
        "    return model\n"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "bNTtW9Oxkgmp",
        "outputId": "2677e981-a151-470f-e91c-8d133ab461b0"
      },
      "execution_count": 31,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (1.25.2)\n",
            "Processing fold #0\n",
            "Processing fold #1\n",
            "Processing fold #2\n",
            "Processing fold #3\n"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Saving the validation logs at each fold\n",
        "\n",
        "num_epochs = 500\n",
        "all_mae_histories = []\n",
        "for i in range(k):\n",
        "    print(f\"Processing fold #{i}\")\n",
        "    val_data = train_data[i * num_val_samples: (i + 1) *
num_val_samples]\n",
        "    val_targets = train_targets[i * num_val_samples: (i + 1) *
num_val_samples]\n",
        "    partial_train_data = np.concatenate(\n",
        "        [train_data[:i * num_val_samples],\n",
        "         train_data[(i + 1) * num_val_samples:]],\n",
        "        axis=0)\n",
        "    partial_train_targets = np.concatenate(\n",
        "        [train_targets[:i * num_val_samples],\n",
        "         train_targets[(i + 1) * num_val_samples:]],\n",
        "        axis=0)\n",
        "    model = build_model()\n",
        "    history = model.fit(partial_train_data,
partial_train_targets,\n",
        "                        validation_data=(val_data,
val_targets),\n",
        "                        epochs=num_epochs, batch_size=16,
verbose=0)\n",
        "    mae_history = history.history[\"val_mae\"]\n",
        "    all_mae_histories.append(mae_history)"
      ],
      "metadata": {
```

```
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "xkqrCxR1vgop",
        "outputId": "73de8841-972c-4094-f983-2033d8cb87d7"
      },
      "execution_count": 33,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Processing fold #0\n",
            "Processing fold #1\n",
            "Processing fold #2\n",
            "Processing fold #3\n"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Building the history of successive mean K-fold validation
scores\n",
        "\n",
        "average_mae_history = [\n",
        "    np.mean([x[i] for x in all_mae_histories]) for i in
range(num_epochs)]"
      ],
      "metadata": {
        "id": "RwqULJblwNd5"
      },
      "execution_count": 34,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#Plotting validation scores\n",
        "\n",
        "import matplotlib.pyplot as plt\n",
        "\n",
        "# Assuming you have defined `average_mae_history` before this
point\n",
        "\n",
        "%matplotlib inline\n",
        "\n",
        "plt.plot(range(1, len(average_mae_history) + 1),
average_mae_history)\n",
        "plt.xlabel(\"Epochs\")\n",
        "plt.ylabel(\"Validation MAE\")\n",
        "plt.show()\n"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 449
```

},
"id": "NyFu6NPzwSmw",
"outputId": "fa24a7a7-3d05-4399-ae84-998503286909"
},
"execution_count": 37,
"outputs": [
{
"output_type": "display_data",
"data": {
"text/plain": [
"<Figure size 640x480 with 1 Axes>"
],
"image/png":



},
"metadata": {}
}
]
},
{
"cell_type": "code",
"source": [
"#Training the final model\n",
"\n",
"model = build_model()\n",
"model.fit(train_data, train_targets,\n",
"   epochs=130, batch_size=16, verbose=0)\n",
"test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)\n",
"test_mae_score"
],

```
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "mR_ZbbQvwkS7",
        "outputId": "df94c078-79e2-46b4-c828-6f35e192d21c"
      },
      "execution_count": 38,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "4/4 [==============================] - 0s 3ms/step - loss:
15.5534 - mae: 2.5204\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "2.520387649536133"
            ]
          },
          "metadata": {},
          "execution_count": 38
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Generating predictions on new data\n",
        "predictions = model.predict(test_data)\n",
        "predictions[0]"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "pNwDe-G7wrfJ",
        "outputId": "40a530ed-728e-48e8-c921-2a3c7380fd17"
      },
      "execution_count": 39,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "4/4 [==============================] - 0s 4ms/step\n"
          ]
        },
        {
          "output_type": "execute_result",
          "data": {
            "text/plain": [
              "array([7.7675962], dtype=float32)"
            ]
```

```
          },
          "metadata": {},
          "execution_count": 39
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "#Wrapping up\n",
        "#Summary"
      ],
      "metadata": {
        "id": "ggZUxDpkwwvM"
      },
      "execution_count": null,
      "outputs": []
    }
  ]
}
```


Summary

It is a binary classification model for movie reviews using the IMDB dataset. It loads and preprocesses the data, encoding reviews into multi-hot vectors. The neural network, built with **TensorFlow and Keras**, consists of two hidden layers with **ReLU activation** and a final layer with sigmoid activation. The model is compiled using binary **crossentropy** loss and RMSprop optimizer. Training and validation sets are prepared, and the model is trained over **20 epochs**. The code includes decoding functions for text representation. It offers a concise example of natural language processing, data preparation, model definition, and training for binary classification tasks.

The provided code compiles and trains a binary classification model for movie reviews using the IMDB dataset. It utilizes the RMSprop optimizer and binary crossentropy loss for compilation. The dataset is split into training and validation sets, and the model is trained over 20 epochs with batch **size 512.** Training history is stored, and the code generates plots for training and validation accuracy, as well as training and validation loss. The plots reveal the model's performance over epochs, showcasing potential overfitting. The code offers insights into model training and evaluation for natural language processing tasks, highlighting the importance of validation sets and performance visualization.

The code initiates a new neural network for multiclass classification using the Reuters dataset. It loads the data, consisting of newswires and their corresponding topics, encodes the sequences into vectors, and preprocesses the labels using one-hot encoding. The model architecture comprises two hidden layers with ReLU activation and a final layer with **softmax activation.** It compiles the model using categorical crossentropy loss and trains it on the training data. The code demonstrates essential steps in multiclass classification, including data loading, preprocessing, model building, and training. Further experiments and the potential for improvements are indicated, providing a foundation for exploring advanced techniques in natural language processing.

It explores the impact of intermediate layer size on a neural network's performance. It defines a model for multiclass classification with an information bottleneck, using smaller intermediate layers. The model is trained on the Reuters dataset, revealing limitations in learning complex patterns due to the constrained layer size. Further experimentation is encouraged. Additionally, the code introduces a regression example for predicting house prices using the Boston Housing dataset. The regression model is defined with two hidden layers and compiled with mean squared error loss. This code provides insights into the importance of layer size in neural networks for classification and regression tasks.