Getting started with neural networks: Classification and regression

Classifying movie reviews: A binary classification example

The IMDB dataset

Loading the IMDB dataset

```
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[0]
```

```
[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]
```

```
train_labels[0]
```

```
1
```

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

```
9999
```

Decoding reviews back 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
```

Preparing the data

Encoding the integer sequences via multi-hot encoding

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

```
x_train[0]
```

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

```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

Building your model

Model definition and compilation

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

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

Validating the data set

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training model

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=25,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

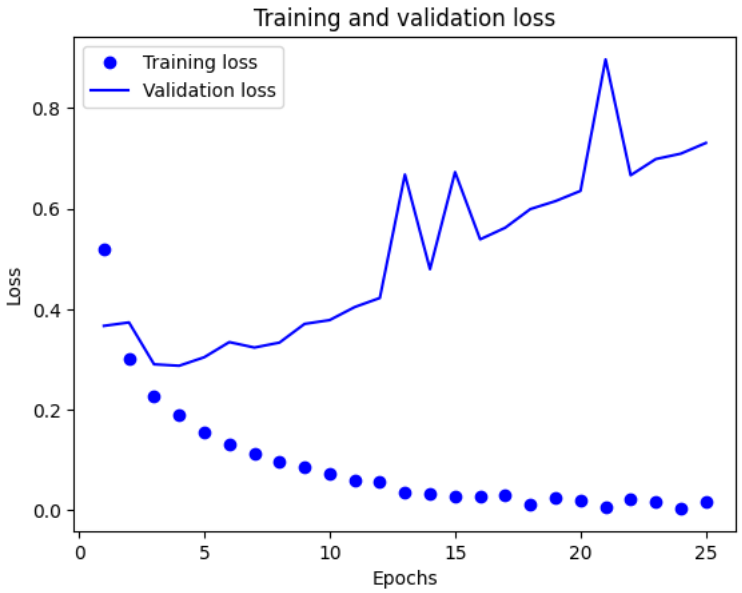
```
Epoch 1/25
30/30 [==============================] - 7s 81ms/step - loss: 0.5177 - accuracy: 0.7625 - val_loss: 0.3667 - val_accuracy: 0.8706
Epoch 2/25
30/30 [==============================] - 1s 35ms/step - loss: 0.3009 - accuracy: 0.8939 - val_loss: 0.3735 - val_accuracy: 0.8386
Epoch 3/25
30/30 [==============================] - 1s 23ms/step - loss: 0.2266 - accuracy: 0.9187 - val_loss: 0.2903 - val_accuracy: 0.8832
Epoch 4/25
30/30 [==============================] - 1s 24ms/step - loss: 0.1901 - accuracy: 0.9313 - val_loss: 0.2874 - val_accuracy: 0.8851
Epoch 5/25
30/30 [==============================] - 1s 22ms/step - loss: 0.1563 - accuracy: 0.9461 - val_loss: 0.3043 - val_accuracy: 0.8804
Epoch 6/25
30/30 [==============================] - 1s 23ms/step - loss: 0.1307 - accuracy: 0.9558 - val_loss: 0.3344 - val_accuracy: 0.8729
Epoch 7/25
30/30 [==============================] - 1s 25ms/step - loss: 0.1126 - accuracy: 0.9610 - val_loss: 0.3236 - val_accuracy: 0.8784
Epoch 8/25
30/30 [==============================] - 1s 25ms/step - loss: 0.0965 - accuracy: 0.9687 - val_loss: 0.3335 - val_accuracy: 0.8796
Epoch 9/25
30/30 [==============================] - 1s 22ms/step - loss: 0.0861 - accuracy: 0.9708 - val_loss: 0.3706 - val_accuracy: 0.8737
Epoch 10/25
30/30 [==============================] - 1s 22ms/step - loss: 0.0718 - accuracy: 0.9772 - val_loss: 0.3782 - val_accuracy: 0.8778
Epoch 11/25
30/30 [==============================] - 1s 22ms/step - loss: 0.0608 - accuracy: 0.9810 - val_loss: 0.4041 - val_accuracy: 0.8755
Epoch 12/25
30/30 [==============================] - 1s 25ms/step - loss: 0.0581 - accuracy: 0.9829 - val_loss: 0.4221 - val_accuracy: 0.8755
Epoch 13/25
30/30 [==============================] - 1s 24ms/step - loss: 0.0363 - accuracy: 0.9915 - val_loss: 0.6675 - val_accuracy: 0.8395
Epoch 14/25
30/30 [==============================] - 1s 25ms/step - loss: 0.0318 - accuracy: 0.9921 - val_loss: 0.4793 - val_accuracy: 0.8729
Epoch 15/25
30/30 [==============================] - 1s 21ms/step - loss: 0.0272 - accuracy: 0.9941 - val_loss: 0.6724 - val_accuracy: 0.8448
Epoch 16/25
30/30 [==============================] - 1s 22ms/step - loss: 0.0270 - accuracy: 0.9932 - val_loss: 0.5387 - val_accuracy: 0.8718
Epoch 17/25
30/30 [==============================] - 1s 38ms/step - loss: 0.0305 - accuracy: 0.9915 - val_loss: 0.5618 - val_accuracy: 0.8703
Epoch 18/25
30/30 [==============================] - 1s 29ms/step - loss: 0.0108 - accuracy: 0.9994 - val_loss: 0.5988 - val_accuracy: 0.8698
Epoch 19/25
30/30 [==============================] - 1s 26ms/step - loss: 0.0247 - accuracy: 0.9931 - val_loss: 0.6146 - val_accuracy: 0.8696
Epoch 20/25
30/30 [==============================] - 1s 25ms/step - loss: 0.0206 - accuracy: 0.9943 - val_loss: 0.6350 - val_accuracy: 0.8692
Epoch 21/25
30/30 [==============================] - 1s 21ms/step - loss: 0.0059 - accuracy: 0.9997 - val_loss: 0.8964 - val_accuracy: 0.8408
Epoch 22/25
30/30 [==============================] - 1s 22ms/step - loss: 0.0231 - accuracy: 0.9929 - val_loss: 0.6661 - val_accuracy: 0.8693
Epoch 23/25
30/30 [==============================] - 1s 22ms/step - loss: 0.0165 - accuracy: 0.9951 - val_loss: 0.6982 - val_accuracy: 0.8683
Epoch 24/25
30/30 [==============================] - 1s 21ms/step - loss: 0.0039 - accuracy: 0.9998 - val_loss: 0.7090 - val_accuracy: 0.8683
Epoch 25/25
30/30 [==============================] - 1s 20ms/step - loss: 0.0173 - accuracy: 0.9945 - val_loss: 0.7303 - val_accuracy: 0.8682
```

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

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

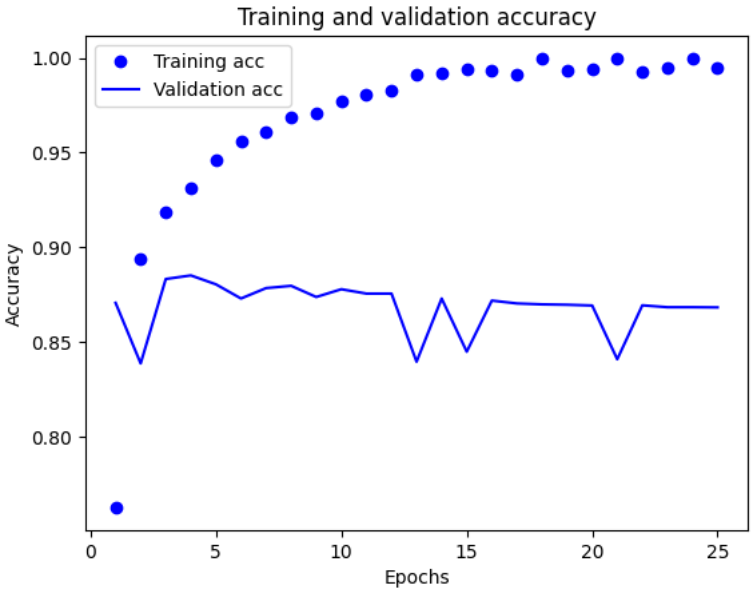
Training and validation loss

```
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()
```



Training and validation accuracy

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



Retraining a model from scratch

```
model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=10, batch_size=512)
```

```
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/10
49/49 [==============================] - 2s 19ms/step - loss: 0.4425 - accuracy: 0.8115
Epoch 2/10
49/49 [==============================] - 1s 16ms/step - loss: 0.2547 - accuracy: 0.9066
Epoch 3/10
49/49 [==============================] - 1s 18ms/step - loss: 0.2070 - accuracy: 0.9219
Epoch 4/10
49/49 [==============================] - 1s 14ms/step - loss: 0.1760 - accuracy: 0.9350
Epoch 5/10
49/49 [==============================] - 1s 14ms/step - loss: 0.1546 - accuracy: 0.9437
Epoch 6/10
49/49 [==============================] - 1s 14ms/step - loss: 0.1344 - accuracy: 0.9518
Epoch 7/10
49/49 [==============================] - 1s 14ms/step - loss: 0.1189 - accuracy: 0.9572
Epoch 8/10
49/49 [==============================] - 1s 13ms/step - loss: 0.1016 - accuracy: 0.9640
Epoch 9/10
49/49 [==============================] - 1s 14ms/step - loss: 0.0885 - accuracy: 0.9696
Epoch 10/10
49/49 [==============================] - 1s 12ms/step - loss: 0.0753 - accuracy: 0.9750
782/782 [==============================] - 3s 3ms/step - loss: 0.4232 - accuracy: 0.8688
```

```
results
```

```
[0.42323464155197144, 0.8688399791717529]
```