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="tanh"),
layers.Dense(32, activation="tanh"),
layers.Dense(32, activation="tanh"),
layers.Dropout(0.5),
layers.Dense(1, activation="sigmoid")
])
```

```
model.compile(optimizer="rmsprop",
loss="mean_squared_error",
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=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

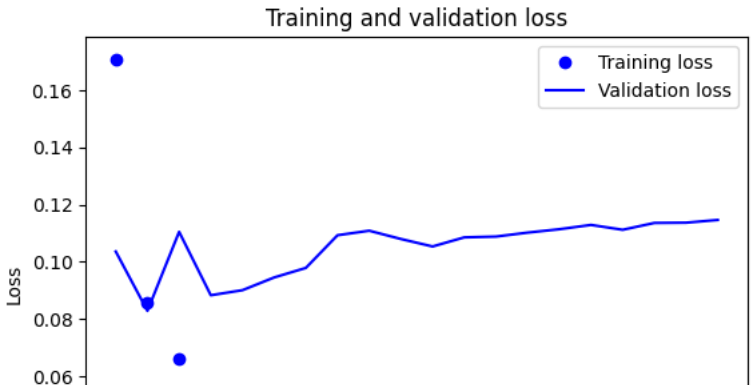
```
Epoch 1/20
30/30 [==============================] - 8s 69ms/step - loss: 0.1707 - accuracy: 0.7537 - val_loss: 0.1036 - val_accuracy: 0.8679
Epoch 2/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0856 - accuracy: 0.8922 - val_loss: 0.0828 - val_accuracy: 0.8865
Epoch 3/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0660 - accuracy: 0.9158 - val_loss: 0.1105 - val_accuracy: 0.8543
Epoch 4/20
30/30 [==============================] - 1s 24ms/step - loss: 0.0534 - accuracy: 0.9325 - val_loss: 0.0883 - val_accuracy: 0.8838
Epoch 5/20
30/30 [==============================] - 1s 25ms/step - loss: 0.0448 - accuracy: 0.9450 - val_loss: 0.0901 - val_accuracy: 0.8831
Epoch 6/20
30/30 [==============================] - 1s 25ms/step - loss: 0.0383 - accuracy: 0.9523 - val_loss: 0.0945 - val_accuracy: 0.8763
Epoch 7/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0339 - accuracy: 0.9599 - val_loss: 0.0978 - val_accuracy: 0.8799
Epoch 8/20
30/30 [==============================] - 1s 29ms/step - loss: 0.0296 - accuracy: 0.9637 - val_loss: 0.1093 - val_accuracy: 0.8684
Epoch 9/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0247 - accuracy: 0.9715 - val_loss: 0.1108 - val_accuracy: 0.8685
Epoch 10/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0229 - accuracy: 0.9741 - val_loss: 0.1080 - val_accuracy: 0.8744
Epoch 11/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0207 - accuracy: 0.9773 - val_loss: 0.1053 - val_accuracy: 0.8752
Epoch 12/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0190 - accuracy: 0.9793 - val_loss: 0.1085 - val_accuracy: 0.8728
Epoch 13/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0197 - accuracy: 0.9777 - val_loss: 0.1088 - val_accuracy: 0.8732
Epoch 14/20
30/30 [==============================] - 1s 25ms/step - loss: 0.0194 - accuracy: 0.9786 - val_loss: 0.1102 - val_accuracy: 0.8714
Epoch 15/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0163 - accuracy: 0.9825 - val_loss: 0.1114 - val_accuracy: 0.8727
Epoch 16/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0159 - accuracy: 0.9824 - val_loss: 0.1129 - val_accuracy: 0.8702
Epoch 17/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0148 - accuracy: 0.9841 - val_loss: 0.1112 - val_accuracy: 0.8731
Epoch 18/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0158 - accuracy: 0.9829 - val_loss: 0.1136 - val_accuracy: 0.8716
Epoch 19/20
30/30 [==============================] - 1s 24ms/step - loss: 0.0153 - accuracy: 0.9833 - val_loss: 0.1137 - val_accuracy: 0.8723
Epoch 20/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0138 - accuracy: 0.9851 - val_loss: 0.1146 - val_accuracy: 0.8695
```

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

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

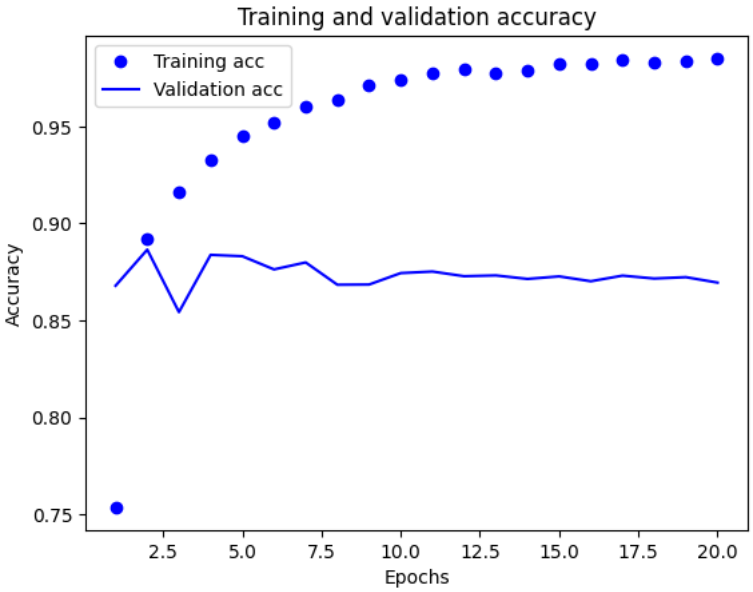
Training and validation loss

```
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="tanh"),
layers.Dense(32, activation="tanh"),
layers.Dense(32, activation="tanh"),
layers.Dropout(0.5),
layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
loss="mean_squared_error",
metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 2s 14ms/step - loss: 0.1435 - accuracy: 0.7996
Epoch 2/4
49/49 [==============================] - 1s 14ms/step - loss: 0.0763 - accuracy: 0.9014
```

```
Epoch 3/4
49/49 [==============================] - 1s 15ms/step - loss: 0.0623 - accuracy: 0.9195
Epoch 4/4
49/49 [==============================] - 1s 13ms/step - loss: 0.0535 - accuracy: 0.9330
782/782 [==============================] - 3s 4ms/step - loss: 0.0965 - accuracy: 0.8733
```

```
results
```

```
[0.09654825180768967, 0.8733199834823608]
```