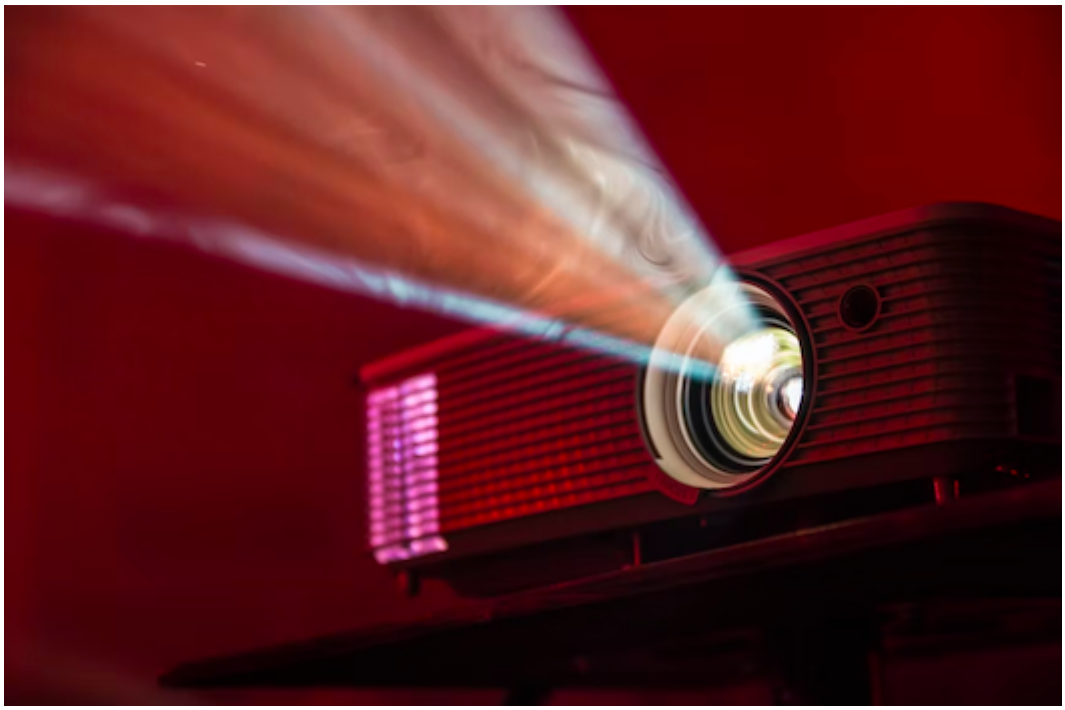
Binary Classification, Multiclass Classification, and Regression

CAP 6619 - Deep Learning | Project 2 | Summer 2022 - Dr. Marques | Matthew Acs



This project will create deep learning solutions for:

- *Binary classification of movie reviews using the IMDB dataset*
- *wires using the Reuters dataset*
- *Regression for house price estimation using the Boston Housing Price dataset*

References and Sources

The following links contain useful resources that provide information relating to this project. Some of the sources provide background information while others answer questions specifically relating to the implementation of the deep learning solutions.

- https://keras.io/api/datasets/imdb/

  Explores the IMDB movie review dataset

- https://www.tensorflow.org/datasets/catalog/imdb_reviews

Explores the IMDB movie review dataset

- https://www.tensorflow.org/tutorials/keras/text_classification_with_hub

  Example IMDB movie review sentiment classifier

- https://colab.research.google.com/github/fchollet/deep-learning-with-python-notebooks/blob/master/chapter04_getting-started-with-neural-networks.ipynb

  Jupyter notebook exploring the use of neural networks to classify the IMDB and Reuters datasets and regression for the Boston Housing Price dataset

---

- https://developers.google.com/machine-learning/guides/text-classification/

  Guide to creating text classifiers

- https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

  Example IMDB movie review sentiment classifier

- https://elitedatascience.com/overfitting-in-machine-learning

  What overfitting is and how to prevent it

- https://keras.io/api/layers/activations/

  Different Keras activation functions

- https://machinelearningmastery.com/k-fold-cross-validation/

  Introduction to k-fold validation

- /8-simple-techniques-to-prevent-overfitting-4d443da2ef7d

  How to prevent overfitting

Saved successfully!

Setup

```
from tensorflow import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras import layers
from sklearn import metrics
```

```
from matplotlib import pyplot as plt
import numpy as np
import textwrap
```

Part 1 - Binary classification of movie reviews using the IMDB dataset

We will start with a simple solution using a fully-connected neural network architecture.

Load and prepare the data

```
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-dataset
17465344/17464789 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
```

```
train_data[0]
```

```
1029,
13,
104,
88,
4,
381,
15,
297,
```

Saved successfully!

```
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
```

```
train_labels[0]
```

```
1
```

Saved successfully!

```
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-dataset
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 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")
```

Examples of Decoded Reviews

The code below shows two examples of reviews in the dataset that have been decoded to plain text. The first one is labeled as a positive review and the second one is labeled as a negative review.

Saved successfully!

```
for i in range(len(train_labels)):
  if train_labels[i] == 1:
    positive_index.append(i)
  else:
    negative_index.append(i)

positive_decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[positive_index[0]]])

negative_decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[negative_index[0]]])

wrapper = textwrap.TextWrapper(width=100)

print("Positive Review")
```

```
print("_________________")
print()

positive_paragraph = wrapper.wrap(text=positive_decoded_review)
for x in positive_paragraph:
  print(x)

print()
print("-----------------------------------------------------------------------------------")
print()

print("Negative Review")
print("_________________")
print()
negative_paragraph = wrapper.wrap(text=negative_decoded_review)
for x in negative_paragraph:
  print(x)
```

```
Positive Review
_________________

? this film was just brilliant casting location scenery story direction everyone
part they played and you could just imagine being there robert ? is an amazing a
same being director ? father came from the same scottish island as myself so i l
was a real connection with this film the witty remarks throughout the film were
brilliant so much that i bought the film as soon as it was released for ? and wo
everyone to watch and the fly fishing was amazing really cried at the end it was
what they say if you cry at a film it must have been good and this definitely wa
little boy's that played the ? of norman and paul they were just brilliant child
out of the ? list i think because the stars that play them all grown up are such
the whole film but these children are amazing and should be praised for what the
you think the whole story was so lovely because it was true and was someone's li
was shared with us all

-----------------------------------------------------------------------------------

Saved successfully!

_________________

? big hair big boobs bad music and a giant safety pin these are the words to bes
terrible movie i love cheesy horror movies and i've seen hundreds but this had g
worst ever made the plot is paper thin and ridiculous the acting is an abominati
completely laughable the best is the end showdown with the cop and how he worked
is it's just so damn terribly written the clothes are sickening and funny in equ
lots of boobs ? men wear those cut ? shirts that show off their ? sickening that
them and the music is just ? trash that plays over and over again in almost ever
trashy music boobs and ? taking away bodies and the gym still doesn't close for
this is a truly bad film whose only charm is to look back on the disaster that w
a good old laugh at how bad everything was back then
```

Building your model

Model definition

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

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

Compiling the model

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

Validating your approach

Setting aside a validation set

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

Saved successfully!

```
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 [==============================] - 5s 41ms/step - loss: 0.5247 - accuracy:
Epoch 2/20
30/30 [==============================] - 1s 19ms/step - loss: 0.3206 - accuracy:
Epoch 3/20
30/30 [==============================] - 1s 19ms/step - loss: 0.2343 - accuracy:
Epoch 4/20
30/30 [==============================] - 1s 19ms/step - loss: 0.1837 - accuracy:
Epoch 5/20
```

```
30/30 [==============================] - 1s 23ms/step - loss: 0.1488 - accuracy:
Epoch 6/20
30/30 [==============================] - 1s 19ms/step - loss: 0.1195 - accuracy:
Epoch 7/20
30/30 [==============================] - 1s 19ms/step - loss: 0.1014 - accuracy:
Epoch 8/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0822 - accuracy:
Epoch 9/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0683 - accuracy:
Epoch 10/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0548 - accuracy:
Epoch 11/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0453 - accuracy:
Epoch 12/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0351 - accuracy:
Epoch 13/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0267 - accuracy:
Epoch 14/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0216 - accuracy:
Epoch 15/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0171 - accuracy:
Epoch 16/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0116 - accuracy:
Epoch 17/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0100 - accuracy:
Epoch 18/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0093 - accuracy:
Epoch 19/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0049 - accuracy:
Epoch 20/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0053 - accuracy:
```

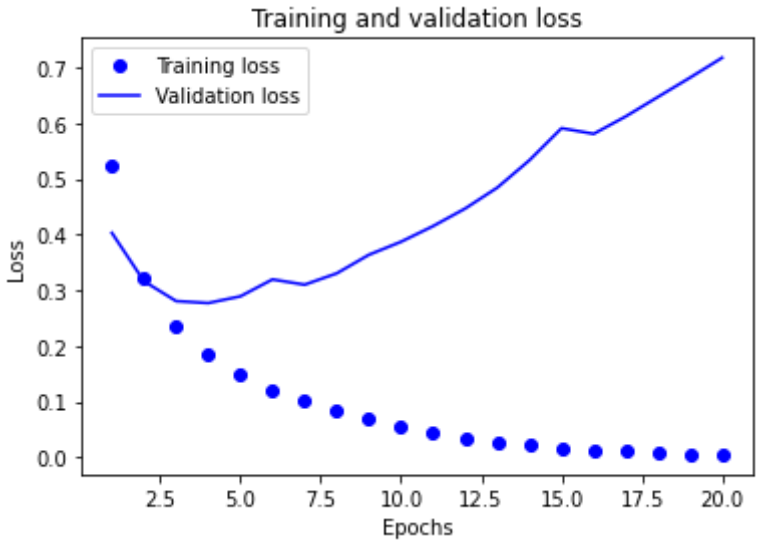
```
history_dict = history.history
history_dict.keys()
```

```
', 'val_loss', 'val_accuracy'])
```

Saved successfully!

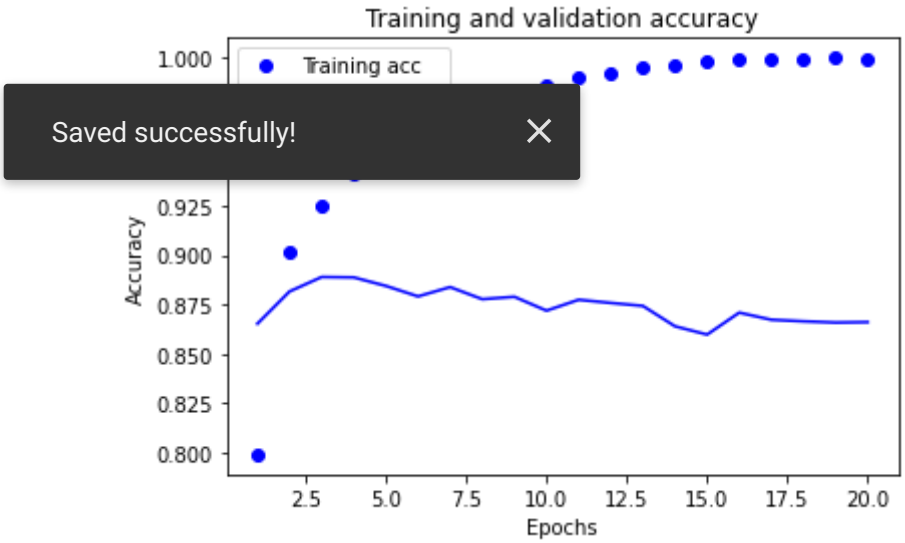
Plotting the training and validation loss

```
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()
```



Plotting the 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()
```



Saved successfully!

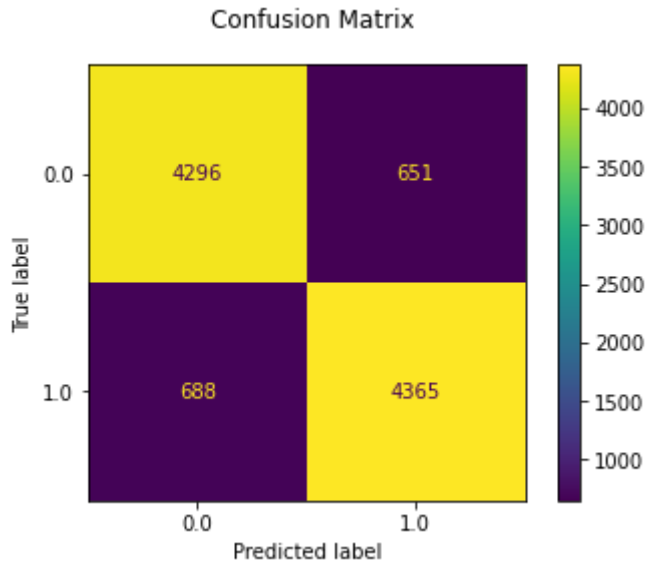
Mistakes and Confusion Matrix

The code below displays the confusion matrix for the classifier. The confusion matrix shows how many reviews were incorrectly and correctly classified and what the true and predicted classes of those reviews were.

```
prediction = model.predict(x_val)
prediction_classes = prediction.round()

disp = metrics.ConfusionMatrixDisplay.from_predictions(y_val, prediction_classes)
disp.figure_.suptitle("Confusion Matrix")

plt.show()
```



The code below displays two examples where the classifier made mistakes. The first review is a false positive, which means it was predicted to be positive while the actual label was negative. By the classifier may have made a mistake. The review is not more neutral. The review discussed both positive and negative aspects of the movie but overall lacked a clear definitive conclusion. In fact, the reviewer stated that they would like to hear it in either French or English to give a definitive review. This ambiguity and lack of clear negative or positive indicators may have confused the classifier. The second review is a false negative, which means that it was predicted to be negative while it was actually positive. The review contains many negative indicators such as "hate", "waste", "stinks" and "lacking" that likely caused the classifier to predict this review to be negative. However, the reviewer is just stating what other reviews may say about the movie before then discussing their opinion and finally recommending the movie. This is a somewhat unusual format for a review and probably caused the review to be misclassified.


```
false_positive = []
```

```
false_negative = []
for i in range(len(prediction_classes)):
  if prediction_classes[i] != y_val[i] and prediction_classes[i] == 1:
    false_positive.append(i)
  elif prediction_classes[i] != y_val[i] and prediction_classes[i] == 0:
    false_negative.append(i)

print("Number of mistakes made: " + str(len(false_positive) + len(false_negative)))

print("----------------")
false_positive_decoded_review = " ".join(
  [reverse_word_index.get(i - 3, "?") for i in train_data[false_positive[0]]])

wrapper = textwrap.TextWrapper(width=100)
false_positive_paragraph = wrapper.wrap(text=false_positive_decoded_review)
for x in false_positive_paragraph:
  print(x)

print()
print("Actual class: Negative")
print("Predicted class: Positive")
print("----------------")
print()

false_negative_decoded_review = " ".join(
  [reverse_word_index.get(i - 3, "?") for i in train_data[false_negative[0]]])

wrapper = textwrap.TextWrapper(width=100)
false_negative_paragraph = wrapper.wrap(text=false_negative_decoded_review)
for x in false_negative_paragraph:
  print(x)

print()
```

Saved successfully!

```
Number of mistakes made: 1339
----------------
? the dvd version consists of 2 episodes the ? of ? being the ? in addition the
without subtitles hence it's hard for me to review in depth this movie because b
understand what was said br br ? being an historic icon the part is very difficu
for a newcomer ? ? just plays fine she is strong ? but also a very supportive te
dalton as ? is perfect and their romance is the main thing of the first episode
a documentary nor a ? but a great love story br br after the ? a new lover comes
flavor is gone we remember always our first love so i found the second episode d
fate isn't told ? br br nonetheless the production is ? the sets are big ? ? the
exotic and the ? splendid the producers have a lot of money for sure but they sp
special effects they are so poor blue screens ships ? that it's funny br br fina
very much to hear it in french or english to make a definitive opinion about thi

Actual class: Negative
Predicted class: Positive
```

```
----------------

? i hate reading reviews that say something like ? waste your time this film sti
to that reviewer yet for me it may have some sort of ? charm if you like the oth
this one will be watchable if you like 40s ? films this one will be watchable br
as good in my opinion as any of the earlier series entries which starred richard
protagonist it's much slower and the plot is trite you've seen this same narrati
many other films and usually better br br but the acting is good and so is the l
dialog it's just lacking in energy and you'll likely figure out exactly what's g
all going to come out in the end not more than a quarter of the way through br b
semi noir and there character mood lighting camera movement and angles are more
story itself but this film is not noir it's too light weight and hollywood innoc
richard ? character nor those of any of his ladies in the previous films had to
you just never knew until the end br br but still i'll recommend this one for at
viewing i've watched it at least twice myself and got a reasonable amount of enj
both times

Actual class: Positive
Predicted class: Negative
----------------
```

Retraining a model from scratch

The model below is retrained from scratch using different parameters. There are three different sets of parameters to choose from below. The first one is the same as the original model, but only trains for 4 epochs. The second one adds a dense layer with a size of 8 and an activation of softmax. The last set of parameters removes one of the dense layers of size 16 from the first model. To choose between these three sets of parameters the variable x can be changed to either 0,1, or 2.

```
x = 1
```

Saved successfully!

```
  model = keras.Sequential([
      layers.Dense(16, activation="relu"),
      layers.Dense(16, activation="relu"),
      layers.Dense(1, activation="sigmoid")
  ])
  model.compile(optimizer="rmsprop",
                loss="binary_crossentropy",
                metrics=["accuracy"])
  model.fit(x_train, y_train, epochs=4, batch_size=512)
  results = model.evaluate(x_test, y_test)

elif x == 1:

  model = keras.Sequential([
      layers.Dense(16, activation="relu"),
```

```
      layers.Dense(16, activation="relu"),
      layers.Dense(8, activation="softmax"),
      layers.Dense(1, activation="sigmoid")
  ])
  model.compile(optimizer="rmsprop",
                loss="binary_crossentropy",
                metrics=["accuracy"])
  model.fit(x_train, y_train, epochs=4, batch_size=512)
  results = model.evaluate(x_test, y_test)

elif x == 2:

  model = keras.Sequential([
      layers.Dense(16, activation="relu"),
      layers.Dense(1, activation="sigmoid")
  ])
  model.compile(optimizer="rmsprop",
                loss="binary_crossentropy",
                metrics=["accuracy"])
  model.fit(x_train, y_train, epochs=4, batch_size=512)
  results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 1s 12ms/step - loss: 0.5855 - accuracy:
Epoch 2/4
49/49 [==============================] - 1s 12ms/step - loss: 0.4713 - accuracy:
Epoch 3/4
49/49 [==============================] - 1s 12ms/step - loss: 0.4307 - accuracy:
Epoch 4/4
49/49 [==============================] - 1s 12ms/step - loss: 0.4039 - accuracy:
782/782 [==============================] - 2s 3ms/step - loss: 0.4389 - accuracy
```

Using a trained model to generate predictions on new data


```
model.predict(x_test)
```

```
array([[0.30144066],
       [0.7138371 ],
       [0.7110614 ],
       ...,
       [0.3094792 ],
       [0.29584292],
       [0.3450538 ]], dtype=float32)
```

Part 2 - Multiclass classification of newswires using the Reuters dataset

Once again, we will start with a simple solution using a fully-connected neural network architecture.

The Reuters dataset

Loading the Reuters dataset

```
from tensorflow.keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
2113536/2110848 [==============================] - 0s 0us/step
2121728/2110848 [==============================] - 0s 0us/step
```

```
len(train_data)
```

```
8982
```

```
len(test_data)
```

```
2246
```

```
train_data[10]
```

```
[1,
 245,
 273,
 207,
 156,
 53,
 14,
 46,
 296,
 26,
 39,
 74,
 2979,
 3554,
 14,
 46,
 4689,
 4329,
 86,
 61,
 3499,
 4795,
 14,
```

Saved successfully!

```
 61,
 451,
 4329,
 17,
 12]
```

Decoding newswires back to text

```
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = " ".join([reverse_word_index.get(i - 3, "?") for i in
    train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
557056/550378 [==============================] - 0s 0us/step
565248/550378 [==============================] - 0s 0us/step
```

```
train_labels[10]
```

```
3
```

Preparing the data

Encoding the input data

```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Saved successfully!

```
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
y_train = to_one_hot(train_labels)
y_test = to_one_hot(test_labels)
```

```
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)
```

Building your model

Model definition

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
])
```

Compiling the model

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

Validating your approach

Setting aside a validation set

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = y_train[:1000]
partial_y_train = y_train[1000:]
```

Saved successfully!

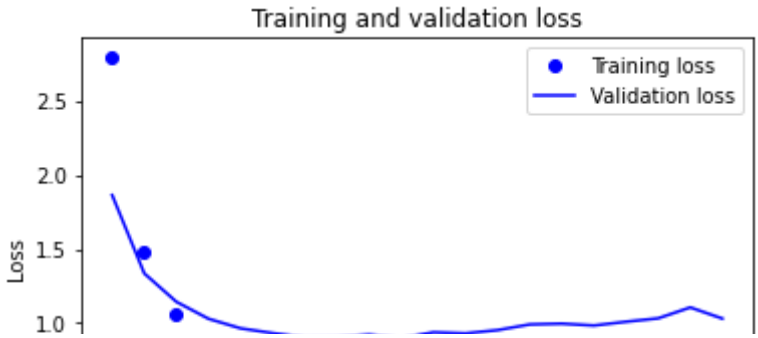
```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
16/16 [==============================] - 1s 28ms/step - loss: 2.7879 - accuracy:
Epoch 2/20
16/16 [==============================] - 0s 15ms/step - loss: 1.4849 - accuracy:
Epoch 3/20
16/16 [==============================] - 0s 15ms/step - loss: 1.0623 - accuracy:
Epoch 4/20
16/16 [==============================] - 0s 15ms/step - loss: 0.8206 - accuracy:
```

```
Epoch 5/20
16/16 [==============================] - 0s 16ms/step - loss: 0.6562 - accuracy:
Epoch 6/20
16/16 [==============================] - 0s 16ms/step - loss: 0.5200 - accuracy:
Epoch 7/20
16/16 [==============================] - 0s 15ms/step - loss: 0.4192 - accuracy:
Epoch 8/20
16/16 [==============================] - 0s 15ms/step - loss: 0.3423 - accuracy:
Epoch 9/20
16/16 [==============================] - 0s 15ms/step - loss: 0.2845 - accuracy:
Epoch 10/20
16/16 [==============================] - 0s 15ms/step - loss: 0.2403 - accuracy:
Epoch 11/20
16/16 [==============================] - 0s 15ms/step - loss: 0.2123 - accuracy:
Epoch 12/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1839 - accuracy:
Epoch 13/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1665 - accuracy:
Epoch 14/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1558 - accuracy:
Epoch 15/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1438 - accuracy:
Epoch 16/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1328 - accuracy:
Epoch 17/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1229 - accuracy:
Epoch 18/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1227 - accuracy:
Epoch 19/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1163 - accuracy:
Epoch 20/20
16/16 [==============================] - 0s 15ms/step - loss: 0.1156 - accuracy:
```

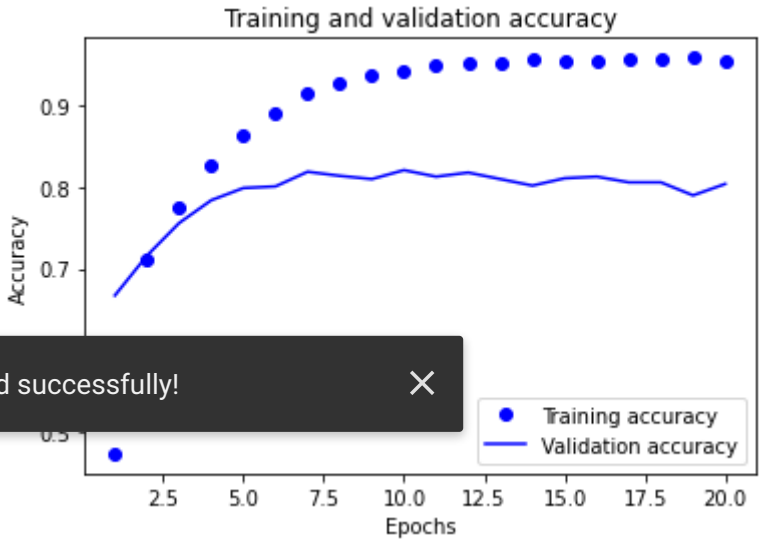
Plotting the training and validation loss


```
val_loss = history.history["val_loss"]
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the training and validation accuracy

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



Saved successfully!

Retraining a model from scratch

The model below is retrained from scratch using different parameters. There are three different sets of parameters to choose from below. The first one is the same as the original model, but only trains for 9 epochs. The second one adds another dense layer with a size of 64. The last set of parameters removes one of the dense layers of size 64 from the first model. To choose between these three sets of parameters the variable x can be changed to either 0,1, or 2.

```
x = 0

if x == 0:

  model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
  ])
  model.compile(optimizer="rmsprop",
                loss="categorical_crossentropy",
                metrics=["accuracy"])
  model.fit(x_train,
            y_train,
            epochs=9,
            batch_size=512)
  results = model.evaluate(x_test, y_test)

elif x == 1:

  model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
  ])
  model.compile(optimizer="rmsprop",
                loss="categorical_crossentropy",
                metrics=["accuracy"])
  model.fit(x_train,
            y_train,
            epochs=9,
            batch_size=512)
                                  t, y_test)
elif x == 2:

  model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
  ])
  model.compile(optimizer="rmsprop",
                loss="categorical_crossentropy",
                metrics=["accuracy"])
  model.fit(x_train,
            y_train,
            epochs=9,
            batch_size=512)
  results = model.evaluate(x_test, y_test)
```

Saved successfully!

```
Epoch 1/9
```

```
18/18 [==============================] - 1s 13ms/step - loss: 2.5293 - accuracy:
Epoch 2/9
18/18 [==============================] - 0s 12ms/step - loss: 1.3928 - accuracy:
Epoch 3/9
18/18 [==============================] - 0s 12ms/step - loss: 1.0261 - accuracy:
Epoch 4/9
18/18 [==============================] - 0s 13ms/step - loss: 0.7934 - accuracy:
Epoch 5/9
18/18 [==============================] - 0s 12ms/step - loss: 0.6211 - accuracy:
Epoch 6/9
18/18 [==============================] - 0s 12ms/step - loss: 0.4981 - accuracy:
Epoch 7/9
18/18 [==============================] - 0s 13ms/step - loss: 0.3913 - accuracy:
Epoch 8/9
18/18 [==============================] - 0s 12ms/step - loss: 0.3186 - accuracy:
Epoch 9/9
18/18 [==============================] - 0s 12ms/step - loss: 0.2664 - accuracy:
71/71 [==============================] - 0s 3ms/step - loss: 0.9351 - accuracy:
```

```
results
```

```
[0.9350989460945129, 0.800979495048523]
```

```
import copy
test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
hits_array = np.array(test_labels) == np.array(test_labels_copy)
hits_array.mean()
```

```
0.18210151380231523
```

Generating predictions on new data

Saved successfully!

```
predictions = model.predict(x_test)
```

```
predictions[0].shape
```

```
(46,)
```

```
np.sum(predictions[0])
```

```
1.0000001
```

```
np.argmax(predictions[0])
```

```
3
```

Part 3 - Regression for house price estimation using the Boston Housing Price dataset

The Boston Housing Price dataset

Loading the Boston housing dataset

```
from tensorflow.keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
57344/57026 [==============================] - 0s 0us/step
65536/57026 [==============================] - 0s 0us/step
```

```
train_data.shape
```

```
(404, 13)
```

```
test_data.shape
```

```
(102, 13)
```

```
train_targets
```

```
1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4, 12.1,
7,  8.8, 50. , 22.5, 24.1, 27.5, 10.9, 30.8,
3, 22.9, 34.7, 16.6, 17.5, 22.3, 16.1, 14.9,
23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2, 24.7, 22.2, 16.7,
12.7, 15.6, 18.4, 21. , 30.1, 15.1, 18.7,  9.6, 31.5, 24.8, 19.1,
22. , 14.5, 11. , 32. , 29.4, 20.3, 24.4, 14.6, 19.5, 14.1, 14.3,
15.6, 10.5,  6.3, 19.3, 19.3, 13.4, 36.4, 17.8, 13.5, 16.5,  8.3,
14.3, 16. , 13.4, 28.6, 43.5, 20.2, 22. , 23. , 20.7, 12.5, 48.5,
14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2, 34.9, 22.5, 31.1,
28.7, 46. , 41.7, 21. , 26.6, 15. , 24.4, 13.3, 21.2, 11.7, 21.7,
19.4, 50. , 22.8, 19.7, 24.7, 36.2, 14.2, 18.9, 18.3, 20.6, 24.6,
18.2,  8.7, 44. , 10.4, 13.2, 21.2, 37. , 30.7, 22.9, 20. , 19.3,
31.7, 32. , 23.1, 18.8, 10.9, 50. , 19.6,  5. , 14.4, 19.8, 13.8,
19.6, 23.9, 24.5, 25. , 19.9, 17.2, 24.6, 13.5, 26.6, 21.4, 11.9,
22.6, 19.6,  8.5, 23.7, 23.1, 22.4, 20.5, 23.6, 18.4, 35.2, 23.1,
27.9, 20.6, 23.7, 28. , 13.6, 27.1, 23.6, 20.6, 18.2, 21.7, 17.1,
 8.4, 25.3, 13.8, 22.2, 18.4, 20.7, 31.6, 30.5, 20.3,  8.8, 19.2,
19.4, 23.1, 23. , 14.8, 48.8, 22.6, 33.4, 21.1, 13.6, 32.2, 13.1,
23.4, 18.9, 23.9, 11.8, 23.3, 22.8, 19.6, 16.7, 13.4, 22.2, 20.4,
21.8, 26.4, 14.9, 24.1, 23.8, 12.3, 29.1, 21. , 19.5, 23.3, 23.8,
```

Saved successfully!

```
       17.8, 11.5, 21.7, 19.9, 25. , 33.4, 28.5, 21.4, 24.3, 27.5, 33.1,
       16.2, 23.3, 48.3, 22.9, 22.8, 13.1, 12.7, 22.6, 15. , 15.3, 10.5,
       24. , 18.5, 21.7, 19.5, 33.2, 23.2,  5. , 19.1, 12.7, 22.3, 10.2,
       13.9, 16.3, 17. , 20.1, 29.9, 17.2, 37.3, 45.4, 17.8, 23.2, 29. ,
       22. , 18. , 17.4, 34.6, 20.1, 25. , 15.6, 24.8, 28.2, 21.2, 21.4,
       23.8, 31. , 26.2, 17.4, 37.9, 17.5, 20. ,  8.3, 23.9,  8.4, 13.8,
        7.2, 11.7, 17.1, 21.6, 50. , 16.1, 20.4, 20.6, 21.4, 20.6, 36.5,
        8.5, 24.8, 10.8, 21.9, 17.3, 18.9, 36.2, 14.9, 18.2, 33.3, 21.8,
       19.7, 31.6, 24.8, 19.4, 22.8,  7.5, 44.8, 16.8, 18.7, 50. , 50. ,
       19.5, 20.1, 50. , 17.2, 20.8, 19.3, 41.3, 20.4, 20.5, 13.8, 16.5,
       23.9, 20.6, 31.5, 23.3, 16.8, 14. , 33.8, 36.1, 12.8, 18.3, 18.7,
       19.1, 29. , 30.1, 50. , 50. , 22. , 11.9, 37.6, 50. , 22.7, 20.8,
       23.5, 27.9, 50. , 19.3, 23.9, 22.6, 15.2, 21.7, 19.2, 43.8, 20.3,
       33.2, 19.9, 22.5, 32.7, 22. , 17.1, 19. , 15. , 16.1, 25.1, 23.7,
       28.7, 37.2, 22.6, 16.4, 25. , 29.8, 22.1, 17.4, 18.1, 30.3, 17.5,
       24.7, 12.6, 26.5, 28.7, 13.3, 10.4, 24.4, 23. , 20. , 17.8,  7. ,
       11.8, 24.4, 13.8, 19.4, 25.2, 19.4, 19.4, 29.1])
```

Preparing the data

Normalizing the data

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```

Building your model

Saved successfully!

Model definition

3 model definitions have been created in order to experiment with different architectures.

```
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
    return model

def build_model_1():
    model = keras.Sequential([
```

```
        layers.Dense(64, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
    return model

def build_model_2():
    model = keras.Sequential([
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
    return model
```

Validating your approach using K-fold validation

```
kvalue = 4 ## change this value to experiment with the effect of different k values

k = kvalue
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print(f"Processing fold #{i}")
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=16, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```



```
Processing fold #0
Processing fold #1
Processing fold #2
Processing fold #3
```

```
all_scores
```

```
[2.0294394493103027, 2.3611016273498535, 2.6545917987823486, 2.454267740249634]
```

```
np.mean(all_scores)
```

```
2.3748501539230347
```

Saving the validation logs at each fold

```
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print(f"Processing fold #{i}")
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=16, verbose=0)
    mae_history = history.history["val_mae"]
    all_mae_histories.append(mae_history)
```

```
Processing fold #0
Processing fold #1
```

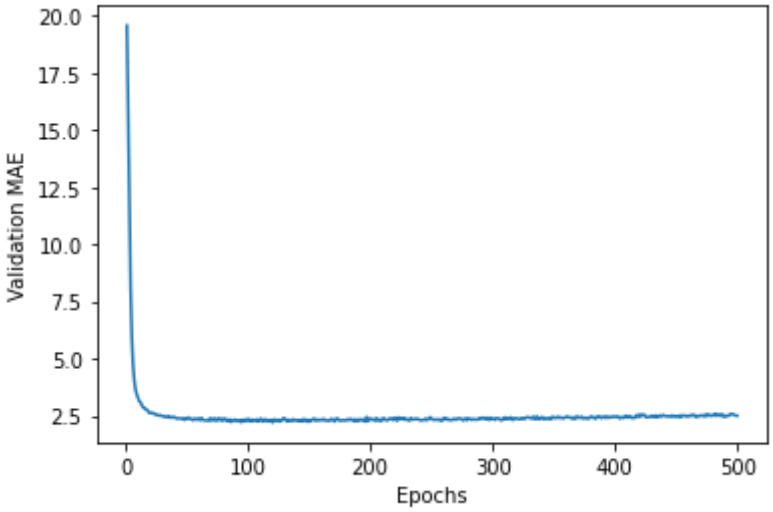
Saved successfully!

Building the history of successive mean K-fold validation scores

```
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

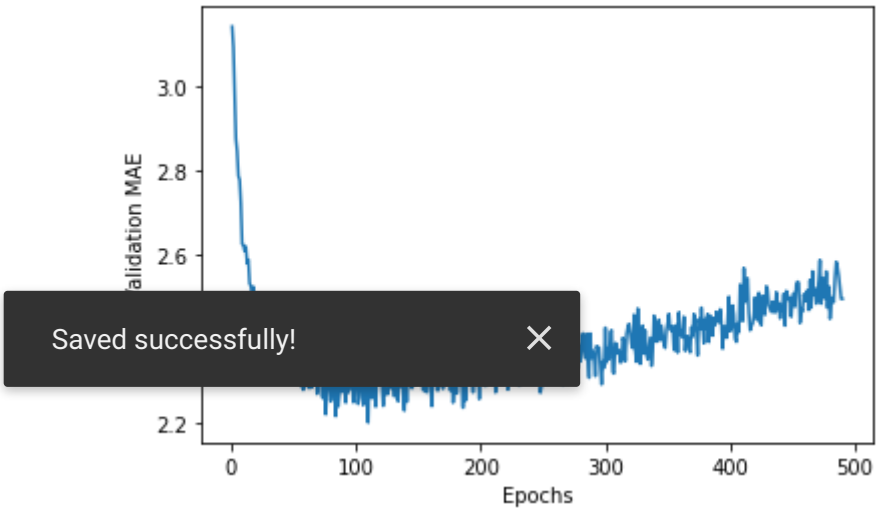
Plotting validation scores

```
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel("Epochs")
plt.ylabel("Validation MAE")
plt.show()
```



Plotting validation scores, excluding the first 10 data points

```
truncated_mae_history = average_mae_history[10:]
plt.plot(range(1, len(truncated_mae_history) + 1), truncated_mae_history)
plt.xlabel("Epochs")
plt.ylabel("Validation MAE")
plt.show()
```



Training the final model

```
epochvalue = 130 ## change this value to experiment with different numbers of epochs

model = build_model()
model.fit(train_data, train_targets,
          epochs=epochvalue, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
4/4 [==============================] - 0s 4ms/step - loss: 16.3395 - mae: 2.5752
```

```
test_mae_score
```

```
2.5752456188201904
```

Generating predictions on new data

```
predictions = model.predict(test_data)
predictions[0]
```

```
array([9.686867], dtype=float32)
```

Parameter Experimentation

To further experiment with the results of changing different parameters, you can use the functions build_model_1 and build_model_2 instead of build_model, which change the number of layers and the size of each layer. The number of epochs and k for the k-fold validation can also be changed to see the effect.

Note: Due to the google server's RAM limits, the experiments must be run by changing the parameters and rerunning the notebook. I would have run separate code to show the experiments and the results, but RAM limits would not allow me to.

Questions for Part 1

Saved successfully!

pplied to the raw input data? Why was it necessary to do so?

What type of preprocessing was applied to the raw input data? Why was it necessary to do so?
Multi-hot encoding was applied to the raw data. This is because the data came as a sequence of integers with each integer representing a word in a dictionary of the 10,000 most common words. Each review has a different length, so the lists of integers can not be used in the neural network without preprocessing. The multi-hot encoding turned the lists of integers into 10,000-dimensional vectors with all zeros except for the indices that corresponded to the integers in the original lists, which contain ones. The labels are also vectorized.

Does your model suffer from overfitting? If so, what would you suggest doing about it?

The model suffers from overfitting because the model preforms better on the training data than the validation data. This can be seen in the loss and accuracy plots, which show that after a few epochs the training loss decreases and the training accuracy increases, but the validation loss increases and the validation accuracy decreases. One way to curb overfitting would be to use a lower number of epochs to prevent the model from overfitting to the training data. Additionally, a model with fewer layers or less units per layer could help reduce overfitting.

Is accuracy a good metric of success in this case? Why (not)?

Accuracy is a good metric of success because the classifier needs to choose between two classes. Each review is either positive or negative, so the accuracy will tell us how well the classifier did at predicting the correct class. The accuracy also tells us how useful this classifier would be in a practical setting because it indicates what percentage of the data we can expect it to correctly predict. The greater this accuracy is, the more likely the model is to correctly predict the movie review sentiment.

Questions for Part 2

What type of preprocessing was applied to the raw input data? Why was it necessary to do so?

Multi-hot encoding was applied to the raw input data. Similar to part 1, the data came as a
r representing a word in a dictionary of the 10,000 most
l of different lengths, so the raw data can not be fed into the neural network without making the data uniform. This is done by turning each list of integers into a 10,000-dimensional vector with all zeros except for the indices that correspond to the integers in the original lists, which contain ones. The labels were encoded using one-hot encoding. This took each integer label (0-45) and turns it into a 46-dimensional vector with all zeros except for the index that corresponds to the label, which contains a one.

How many categories are there in this case?

There are 46 categories in the dataset. These categories are represented by integers ranging from 0 to 45.

Does your model suffer from overfitting? If so, what would you suggest doing about it?

The model suffers from overfitting because the training data preforms better than the validation data. This can be seen in the loss and accuracy curves. As the number of epochs increases, the training loss decreases and the training accuracy increases, however, the validation loss increases and the validation accuracy decreases. To prevent overfitting, the number of epochs can be reduced to where the overfitting occurs on the plots.

Is accuracy a good metric of success in this case? Why (not)?

Accuracy is a good metric of success because it tells us how well the classifier performed on the data set. The accuracy value lets us know what percentage of the data it was able to correctly classify. This is a useful metric because it tells us how well the classifier would do on a practical dataset, which is essentially how useful and reliable the classifier is.

Questions for Part 3

What type of preprocessing was applied to the raw input data? Why was it necessary to do so?

The raw data were normalized by subtracting the mean of each feature and dividing each feature by its standard deviation. This is necessary because the original data have very different ranges between each feature, which would make learning more difficult. By normalizing the data, the ranges for each feature are more homogenous, which is easier for the neural network to deal with.

Why is this problem a case of regression (rather than classification)?

This is a case of regression because the model predicts a value on a continuous scale rather than a discrete class among a few different classes. In terms of this scenario, the model will output a number that represents the predicted house value rather than a class.

Does your model suffer from overfitting? If so, what would you suggest doing about it?

Yes, the model suffers from overfitting. This is because the validation MAE starts to increase as the number of epochs increases, which suggests that the model is overfitting to the training data rather than improving its performance on validation data as well. To curb overfitting, the number of

epochs can be reduced to the number of epochs when overfitting started to occur on the validation

Is mean absolute error (MAE) a good metric of success in this case? Why (not)?

MAE is a good metric of success in this case. This is because the problem deals with regression rather than classification. The MAE tells us how far the continuous predictions were from the true values. This is appropriate for assessing the success of a model that outputs a continuous range of values rather than classes. The MAE also tells us how far we can expect the predictions to be from the true values, or put another way, how large we can expect the error of the model to be when making predictions on new data. Accuracy is not a good metric because you can only measure how far the predicted values are from the actual values (the MAE), not how accurate those values are concerning a correct or incorrect classification. Accuracy is a better metric of success for classifiers.

Discussions and Conclusions

Overall, this assignment demonstrated the difference between binary classification, multiclass classification, and regression. It showed some of the different ways that neural networks can be applied to solve problems. Overfitting also began to appear as a major theme in deep learning that every model must contend with. It became apparent that the first challenge of deep learning is to create a model that overfits. The second challenge is then to improve the accuracy of the model by curbing overfitting and improving the parameters and architecture to achieve better results. The assignment also explored the different architectures, activations, and success metrics that are used for different types of deep learning problems.

fortable working with the different steps involved in the deep learning workflow. I have a better understanding of the different types of data preprocessing as well as the choices that are made for the architecture of the model. I also have a better understanding of the three different applications of deep learning that were explored in this assignment. One frustrating aspect of this assignment was the RAM limits that google imposes on the runtimes. As I was running the experiments for the models to see how changing parameters would impact the results, I would frequently run out of RAM and I would have to run all of the code again after reconnecting to the runtime. I may need to connect to a local runtime in the future to overcome the RAM limits of google's hosted runtimes.

✓ 0s completed at 11:42 PM

Saved successfully!