t81_558_class_09_3_transfer_nlp

May 24, 2025

1 T81-558: Applications of Deep Neural Networks

Module 9: Transfer Learning * Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis * For more information visit the class website.

2 Module 9 Material

- Part 9.1: Introduction to Keras Transfer Learning [Video] [Notebook]
- Part 9.2: Keras Transfer Learning for Computer Vision [Video] [Notebook]
- **Part 9.3: Transfer Learning for NLP with Keras** [Video] [Notebook]
- Part 9.4: Transfer Learning for Facial Feature Recognition [Video] [Notebook]
- Part 9.5: Transfer Learning for Style Transfer [Video] [Notebook]

3 Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

[ ]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

4 Part 9.3: Transfer Learning for NLP with Keras

You will commonly use transfer learning with Natural Language Processing (NLP). Word embeddings are a common means of transfer learning in NLP where network layers map words to vectors. Third parties trained neural networks on a large corpus of text to learn these embeddings. We will use these vectors as the input to the neural network rather than the actual characters of words.

This course has an entire module covering NLP; however, we use word embeddings to perform sentiment analysis in this module. We will specifically attempt to classify if a text sample is speaking in a positive or negative tone.

The following three sources were helpful for the creation of this section.

- Universal sentence encoder [Cite:cer2018universal]. arXiv preprint arXiv:1803.11175)
- Deep Transfer Learning for Natural Language Processing: Text Classification with Universal Embeddings [Cite:howard2018universal]
- Keras Tutorial: How to Use Google's Universal Sentence Encoder for Spam Classification

These examples use TensorFlow Hub, which allows pretrained models to be loaded into TensorFlow easily. To install TensorHub use the following commands.

[ ]:
```
# HIDE OUTPUT
!pip install tensorflow_hub
```

```
Requirement already satisfied: tensorflow_hub in /usr/local/lib/python3.7/dist-
packages (0.12.0)
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_hub) (1.19.5)
Requirement already satisfied: protobuf>=3.8.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_hub) (3.17.3)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-
packages (from protobuf>=3.8.0->tensorflow_hub) (1.15.0)
```

It is also necessary to install TensorFlow Datasets, which you can install with the following command.

[ ]:
```
# HIDE OUTPUT
!pip install tensorflow_datasets
```

```
Requirement already satisfied: tensorflow_datasets in
/usr/local/lib/python3.7/dist-packages (4.0.1)
Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.7/dist-packages (from tensorflow_datasets) (5.4.0)
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_datasets) (3.17.3)
Requirement already satisfied: attrs>=18.1.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_datasets) (21.4.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.7/dist-
packages (from tensorflow_datasets) (1.1.0)
Requirement already satisfied: promise in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (2.3)
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (0.3.4)
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (1.0.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (1.15.0)
Requirement already satisfied: tensorflow-metadata in
/usr/local/lib/python3.7/dist-packages (from tensorflow_datasets) (1.6.0)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (0.16.0)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow_datasets) (2.23.0)
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (4.62.3)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (0.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from tensorflow_datasets) (1.19.5)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests>=2.19.0->tensorflow_datasets) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (2021.10.8)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.7/dist-
packages (from importlib-resources->tensorflow_datasets) (3.7.0)
Requirement already satisfied: googleapis-common-protos<2,>=1.52.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-
metadata->tensorflow_datasets) (1.54.0)
```

Movie reviews are a good source of training data for sentiment analysis. These reviews are textual, and users give them a star rating which indicates if the viewer had a positive or negative experience with the movie. Load the Internet Movie DataBase (IMDB) reviews data set. This example is based on a TensorFlow example that you can find here.

[ ]:
```python
# HIDE OUTPUT
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds

train_data, test_data = tfds.load(name="imdb_reviews",
                                  split=["train", "test"],
                                  batch_size=-1, as_supervised=True)

train_examples, train_labels = tfds.as_numpy(train_data)
test_examples, test_labels = tfds.as_numpy(test_data)

# /Users/jheaton/tensorflow_datasets/imdb_reviews/plain_text/0.1.0
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download:
80.23 MiB, generated: Unknown size, total: 80.23 MiB) to
/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0…
Dl Completed…: 0 url [00:00, ? url/s]
```

```
Dl Size…: 0 MiB [00:00, ? MiB/s]

0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete0GRP97/imdb_reviews-train.tfrecord

  0%|          | 0/25000 [00:00<?, ? examples/s]

0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete0GRP97/imdb_reviews-test.tfrecord

  0%|          | 0/25000 [00:00<?, ? examples/s]

0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_t
ext/1.0.0.incomplete0GRP97/imdb_reviews-unsupervised.tfrecord

  0%|          | 0/50000 [00:00<?, ? examples/s]

WARNING:absl:Dataset is using deprecated text encoder API which will be removed
soon. Please use the plain_text version of the dataset and migrate to
`tensorflow_text`.
```

`Dataset imdb_reviews downloaded and prepared to`

`/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0. Subsequent calls will`

`reuse this data.`

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-
packages/tensorflow_datasets/core/dataset_builder.py:598: get_single_element
(from tensorflow.python.data.experimental.ops.get_single_element) is deprecated
and will be removed in a future version.
Instructions for updating:
Use `tf.data.Dataset.get_single_element()`.

WARNING:tensorflow:From /usr/local/lib/python3.7/dist-
packages/tensorflow_datasets/core/dataset_builder.py:598: get_single_element
(from tensorflow.python.data.experimental.ops.get_single_element) is deprecated
and will be removed in a future version.
Instructions for updating:
Use `tf.data.Dataset.get_single_element()`.
```

Load a pretrained embedding model called gnews-swivel-20dim. Google trained this network on GNEWS data and can convert raw text into vectors.

[ ]:
```
model = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"
hub_layer = hub.KerasLayer(model, output_shape=[20], input_shape=[],
                           dtype=tf.string, trainable=True)
```

The following code displays three movie reviews. This display allows you to see the actual data.

```
[ ]: train_examples[:3]
```

```
[ ]: array([b"This was an absolutely terrible movie. Don't be lured in by Christopher
     Walken or Michael Ironside. Both are great actors, but this must simply be their
     worst role in history. Even their great acting could not redeem this movie's
     ridiculous storyline. This movie is an early nineties US propaganda piece. The
     most pathetic scenes were those when the Columbian rebels were making their
     cases for revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love
     affair with Walken was nothing but a pathetic emotional plug in a movie that was
     devoid of any real meaning. I am disappointed that there are movies like this,
     ruining actor's like Christopher Walken's good name. I could barely sit through
     it.",
            b'I have been known to fall asleep during films, but this is usually due
     to a combination of things including, really tired, being warm and comfortable
     on the sette and having just eaten a lot. However on this occasion I fell asleep
     because the film was rubbish. The plot development was constant. Constantly slow
     and boring. Things seemed to happen, but with no explanation of what was causing
     them or why. I admit, I may have missed part of the film, but i watched the
     majority of it and everything just seemed to happen of its own accord without
     any real concern for anything else. I cant recommend this film at all.',
            b'Mann photographs the Alberta Rocky Mountains in a superb fashion, and
     Jimmy Stewart and Walter Brennan give enjoyable performances as they always seem
     to do. <br /><br />But come on Hollywood - a Mountie telling the people of
     Dawson City, Yukon to elect themselves a marshal (yes a marshal!) and to enforce
     the law themselves, then gunfighters battling it out on the streets for control
     of the town? <br /><br />Nothing even remotely resembling that happened on the
     Canadian side of the border during the Klondike gold rush. Mr. Mann and company
     appear to have mistaken Dawson City for Deadwood, the Canadian North for the
     American Wild West.<br /><br />Canadian viewers be prepared for a Reefer Madness
     type of enjoyable howl with this ludicrous plot, or, to shake your head in
     disgust.'],
           dtype=object)
```

The embedding layer can convert each to 20-number vectors, which the neural network receives as input in place of the actual words.

```
[ ]: hub_layer(train_examples[:3])
```

```
[ ]: <tf.Tensor: shape=(3, 20), dtype=float32, numpy=
     array([[ 1.7657859 , -3.882232  ,  3.913424  , -1.5557289 , -3.3362343 ,
             -1.7357956 , -1.9954445 ,  1.298955  ,  5.081597  , -1.1041285 ,
             -2.0503852 , -0.7267516 , -0.6567596 ,  0.24436145, -3.7208388 ,
              2.0954835 ,  2.2969332 , -2.0689783 , -2.9489715 , -1.1315986 ],
            [ 1.8804485 , -2.5852385 ,  3.4066994 ,  1.0982676 , -4.056685  ,
             -4.891284  , -2.7855542 ,  1.3874227 ,  3.8476458 , -0.9256539 ,
             -1.896706  ,  1.2113281 ,  0.11474716,  0.76209456, -4.8791065 ,
```

```
    2.906149  ,  4.7087674 , -2.3652055 , -3.5015903 , -1.6390051 ],
  [ 0.71152216, -0.63532174,  1.7385626 , -1.1168287 , -0.54515934,
   -1.1808155 ,  0.09504453,  1.4653089 ,  0.66059506,  0.79308075,
   -2.2268343 ,  0.07446616, -1.4075902 , -0.706454  , -1.907037  ,
    1.4419788 ,  1.9551864 , -0.42660046, -2.8022065 ,  0.43727067]],
dtype=float32)>
```

We add additional layers to classify the movie reviews as either positive or negative.

[ ]:
```python
model = tf.keras.Sequential()
model.add(hub_layer)
model.add(tf.keras.layers.Dense(16, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 20)                400020

 dense (Dense)               (None, 16)                336

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 400,373
Trainable params: 400,373
Non-trainable params: 0
_________________________________________________________________
```

We are now ready to compile the neural network. For this application, we use the adam training method for binary classification. We also save the initial random weights for later to start over easily.

[ ]:
```python
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
init_weights = model.get_weights()
```

Before fitting, we split the training data into the train and validation sets.

[ ]:
```python
x_val = train_examples[:10000]
partial_x_train = train_examples[10000:]

y_val = train_labels[:10000]
partial_y_train = train_labels[10000:]
```

We can now fit the neural network. This fitting will run for 40 epochs and allow us to evaluate the effectiveness of the neural network, as measured by the training set.

```
[ ]: history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=40,
                         batch_size=512,
                         validation_data=(x_val, y_val),
                         verbose=1)
```

```
Epoch 1/40
30/30 [==============================] - 4s 80ms/step - loss: 1.5554 - accuracy:
0.5515 - val_loss: 0.8048 - val_accuracy: 0.5865
Epoch 2/40
30/30 [==============================] - 2s 73ms/step - loss: 0.7600 - accuracy:
0.6011 - val_loss: 0.7107 - val_accuracy: 0.6230
Epoch 3/40
30/30 [==============================] - 2s 82ms/step - loss: 0.6541 - accuracy:
0.6561 - val_loss: 0.6263 - val_accuracy: 0.6662
Epoch 4/40
30/30 [==============================] - 2s 70ms/step - loss: 0.5864 - accuracy:
0.6953 - val_loss: 0.5818 - val_accuracy: 0.6978
Epoch 5/40
30/30 [==============================] - 2s 77ms/step - loss: 0.5493 - accuracy:
0.7248 - val_loss: 0.5551 - val_accuracy: 0.7190
Epoch 6/40
30/30 [==============================] - 2s 77ms/step - loss: 0.5222 - accuracy:
0.7452 - val_loss: 0.5336 - val_accuracy: 0.7338
Epoch 7/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4990 - accuracy:
0.7618 - val_loss: 0.5146 - val_accuracy: 0.7477
Epoch 8/40
30/30 [==============================] - 1s 36ms/step - loss: 0.4765 - accuracy:
0.7768 - val_loss: 0.4967 - val_accuracy: 0.7637
Epoch 9/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4551 - accuracy:
0.7925 - val_loss: 0.4798 - val_accuracy: 0.7739
Epoch 10/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4335 - accuracy:
0.8062 - val_loss: 0.4629 - val_accuracy: 0.7864
Epoch 11/40
30/30 [==============================] - 1s 36ms/step - loss: 0.4129 - accuracy:
0.8191 - val_loss: 0.4466 - val_accuracy: 0.7971
Epoch 12/40
30/30 [==============================] - 1s 36ms/step - loss: 0.3915 - accuracy:
0.8315 - val_loss: 0.4309 - val_accuracy: 0.8086
Epoch 13/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3710 - accuracy:
0.8431 - val_loss: 0.4159 - val_accuracy: 0.8180
```

```
Epoch 14/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3510 - accuracy:
0.8544 - val_loss: 0.4017 - val_accuracy: 0.8262
Epoch 15/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3310 - accuracy:
0.8643 - val_loss: 0.3883 - val_accuracy: 0.8315
Epoch 16/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3118 - accuracy:
0.8740 - val_loss: 0.3754 - val_accuracy: 0.8385
Epoch 17/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2932 - accuracy:
0.8846 - val_loss: 0.3638 - val_accuracy: 0.8454
Epoch 18/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2747 - accuracy:
0.8930 - val_loss: 0.3533 - val_accuracy: 0.8495
Epoch 19/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2568 - accuracy:
0.9030 - val_loss: 0.3434 - val_accuracy: 0.8539
Epoch 20/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2396 - accuracy:
0.9094 - val_loss: 0.3360 - val_accuracy: 0.8567
Epoch 21/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2246 - accuracy:
0.9183 - val_loss: 0.3298 - val_accuracy: 0.8605
Epoch 22/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2104 - accuracy:
0.9248 - val_loss: 0.3234 - val_accuracy: 0.8633
Epoch 23/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1971 - accuracy:
0.9295 - val_loss: 0.3192 - val_accuracy: 0.8663
Epoch 24/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1856 - accuracy:
0.9359 - val_loss: 0.3173 - val_accuracy: 0.8678
Epoch 25/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1739 - accuracy:
0.9417 - val_loss: 0.3147 - val_accuracy: 0.8704
Epoch 26/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1631 - accuracy:
0.9464 - val_loss: 0.3144 - val_accuracy: 0.8713
Epoch 27/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1538 - accuracy:
0.9508 - val_loss: 0.3134 - val_accuracy: 0.8725
Epoch 28/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1454 - accuracy:
0.9540 - val_loss: 0.3158 - val_accuracy: 0.8723
Epoch 29/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1372 - accuracy:
0.9573 - val_loss: 0.3174 - val_accuracy: 0.8739
```

```
Epoch 30/40
30/30 [==============================] - 1s 38ms/step - loss: 0.1287 - accuracy:
0.9605 - val_loss: 0.3174 - val_accuracy: 0.8748
Epoch 31/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1211 - accuracy:
0.9634 - val_loss: 0.3202 - val_accuracy: 0.8752
Epoch 32/40
30/30 [==============================] - 1s 38ms/step - loss: 0.1140 - accuracy:
0.9657 - val_loss: 0.3226 - val_accuracy: 0.8745
Epoch 33/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1067 - accuracy:
0.9683 - val_loss: 0.3272 - val_accuracy: 0.8738
Epoch 34/40
30/30 [==============================] - 1s 38ms/step - loss: 0.1001 - accuracy:
0.9707 - val_loss: 0.3323 - val_accuracy: 0.8737
Epoch 35/40
30/30 [==============================] - 1s 38ms/step - loss: 0.0934 - accuracy:
0.9734 - val_loss: 0.3352 - val_accuracy: 0.8737
Epoch 36/40
30/30 [==============================] - 1s 38ms/step - loss: 0.0871 - accuracy:
0.9761 - val_loss: 0.3403 - val_accuracy: 0.8740
Epoch 37/40
30/30 [==============================] - 1s 38ms/step - loss: 0.0817 - accuracy:
0.9788 - val_loss: 0.3449 - val_accuracy: 0.8729
Epoch 38/40
30/30 [==============================] - 1s 36ms/step - loss: 0.0765 - accuracy:
0.9805 - val_loss: 0.3508 - val_accuracy: 0.8739
Epoch 39/40
30/30 [==============================] - 1s 37ms/step - loss: 0.0711 - accuracy:
0.9820 - val_loss: 0.3562 - val_accuracy: 0.8738
Epoch 40/40
30/30 [==============================] - 1s 37ms/step - loss: 0.0661 - accuracy:
0.9847 - val_loss: 0.3626 - val_accuracy: 0.8728
```

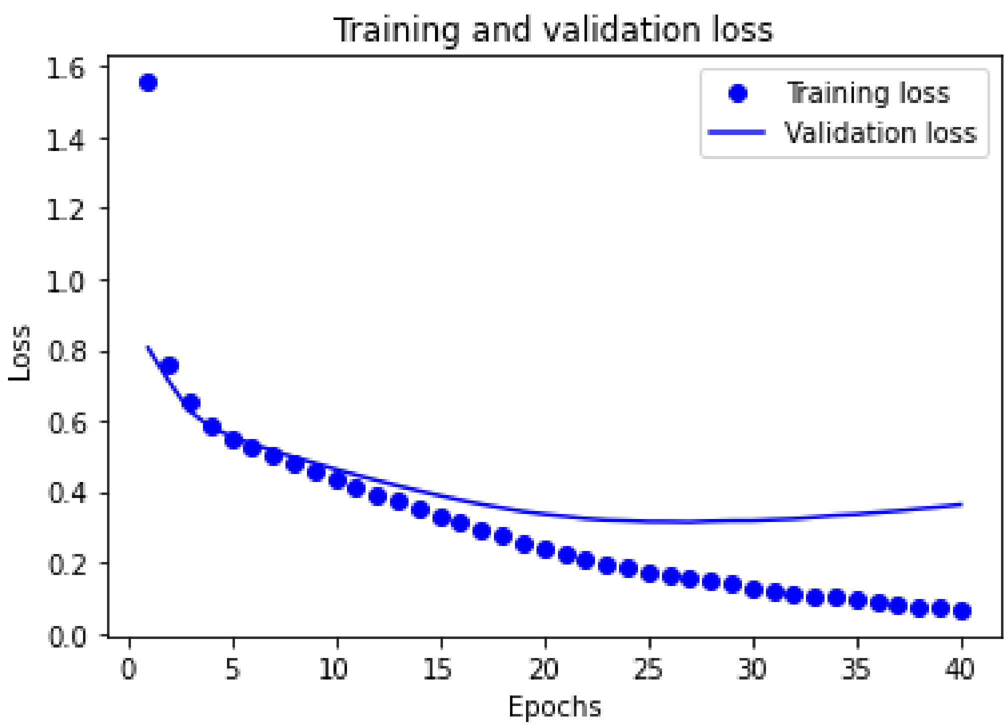
4.1 Benefits of Early Stopping

While we used a validation set, we fit the neural network without early stopping. This dataset is complex enough to allow us to see the benefit of early stopping. We will examine how accuracy and loss progressed for training and validation sets. Loss measures the degree to which the neural network was confident in incorrect answers. Accuracy is the percentage of correct classifications, regardless of the neural network's confidence.

We begin by looking at the loss as we fit the neural network.

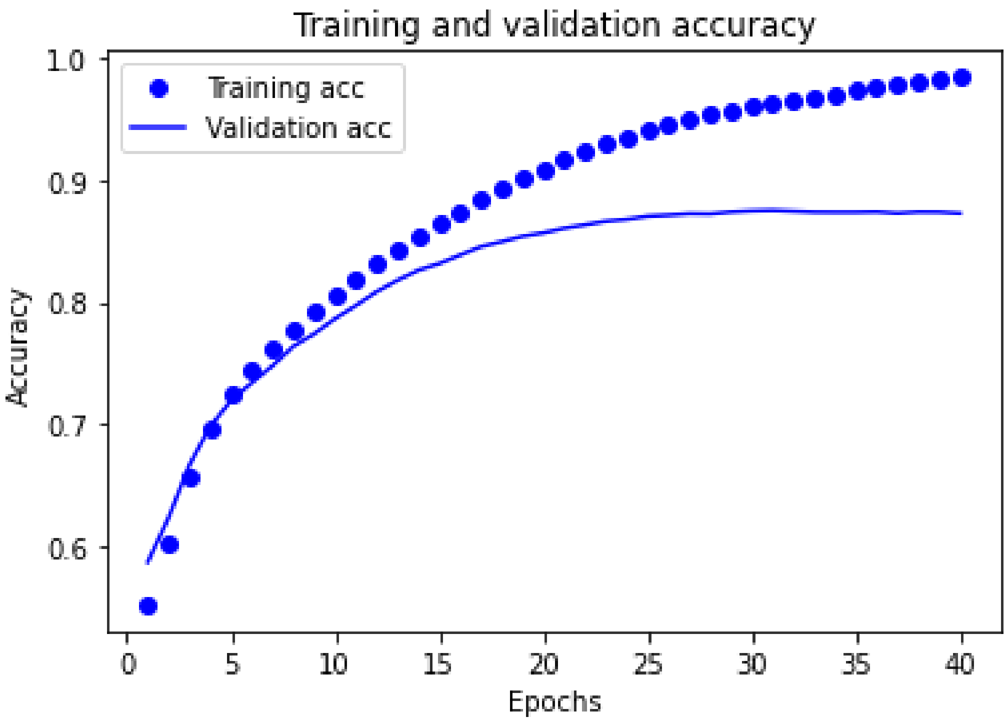
[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

history_dict = history.history
```

```
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 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()
```



We can see that training and validation loss are similar early in the fitting. However, as fitting continues and overfitting sets in, training and validation loss diverge from each other. Training loss continues to fall consistently. However, once overfitting happens, the validation loss no longer falls and eventually begins to increase a bit. Early stopping, which we saw earlier in this course, can prevent some overfitting.

[ ]:
```python
plt.clf()   # clear figure

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()
```



The accuracy graph tells a similar story. Now let's repeat the fitting with early stopping. We begin by creating an early stopping monitor and restoring the network's weights to random. Once this is complete, we can fit the neural network with the early stopping monitor enabled.

[ ]:
```python
from tensorflow.keras.callbacks import EarlyStopping

monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=5, verbose=1, mode='auto',
        restore_best_weights=True)

model.set_weights(init_weights)

history = model.fit(partial_x_train,
```

```
                    partial_y_train,
                    epochs=40,
                    batch_size=512,
                    callbacks=[monitor],
                    validation_data=(x_val, y_val),
                    verbose=1)
```

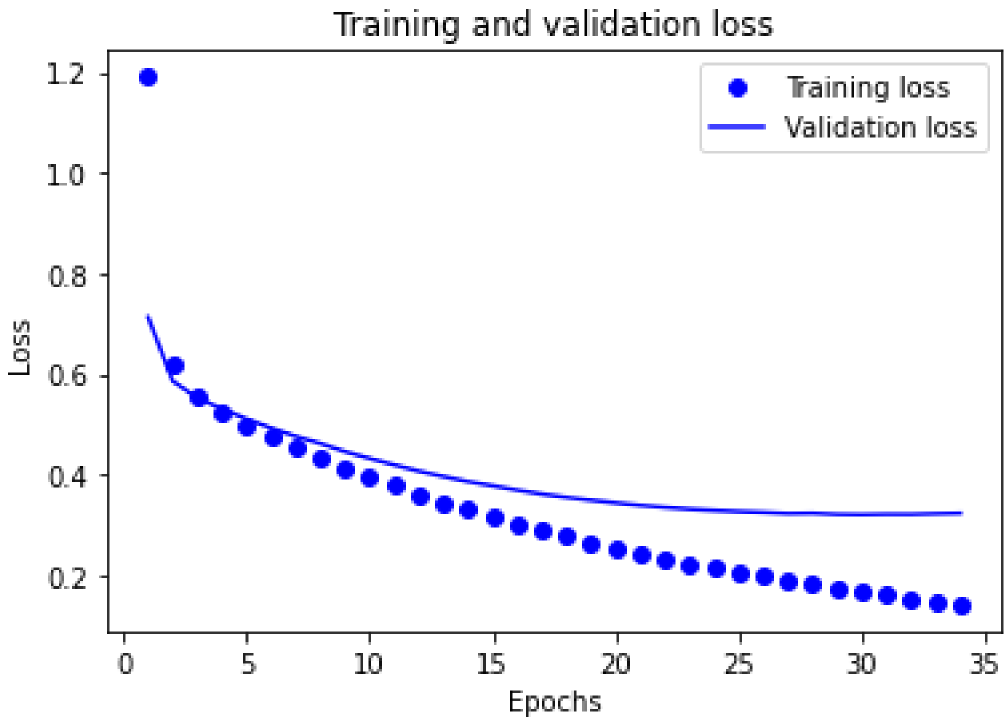
```
Epoch 1/40
30/30 [==============================] - 1s 38ms/step - loss: 1.1912 - accuracy:
0.5643 - val_loss: 0.7129 - val_accuracy: 0.6332
Epoch 2/40
30/30 [==============================] - 1s 37ms/step - loss: 0.6202 - accuracy:
0.6749 - val_loss: 0.5862 - val_accuracy: 0.6922
Epoch 3/40
30/30 [==============================] - 1s 37ms/step - loss: 0.5531 - accuracy:
0.7251 - val_loss: 0.5525 - val_accuracy: 0.7208
Epoch 4/40
30/30 [==============================] - 1s 36ms/step - loss: 0.5228 - accuracy:
0.7483 - val_loss: 0.5308 - val_accuracy: 0.7403
Epoch 5/40
30/30 [==============================] - 1s 38ms/step - loss: 0.4981 - accuracy:
0.7671 - val_loss: 0.5106 - val_accuracy: 0.7569
Epoch 6/40
30/30 [==============================] - 1s 38ms/step - loss: 0.4754 - accuracy:
0.7823 - val_loss: 0.4925 - val_accuracy: 0.7690
Epoch 7/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4542 - accuracy:
0.7959 - val_loss: 0.4759 - val_accuracy: 0.7803
Epoch 8/40
30/30 [==============================] - 1s 37ms/step - loss: 0.4342 - accuracy:
0.8089 - val_loss: 0.4619 - val_accuracy: 0.7883
Epoch 9/40
30/30 [==============================] - 1s 38ms/step - loss: 0.4143 - accuracy:
0.8215 - val_loss: 0.4455 - val_accuracy: 0.7998
Epoch 10/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3960 - accuracy:
0.8318 - val_loss: 0.4319 - val_accuracy: 0.8090
Epoch 11/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3784 - accuracy:
0.8390 - val_loss: 0.4190 - val_accuracy: 0.8166
Epoch 12/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3613 - accuracy:
0.8483 - val_loss: 0.4068 - val_accuracy: 0.8237
Epoch 13/40
30/30 [==============================] - 1s 37ms/step - loss: 0.3453 - accuracy:
0.8570 - val_loss: 0.3962 - val_accuracy: 0.8291
Epoch 14/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3301 - accuracy:
```

```
0.8644 - val_loss: 0.3861 - val_accuracy: 0.8360
Epoch 15/40
30/30 [==============================] - 1s 39ms/step - loss: 0.3157 - accuracy:
0.8735 - val_loss: 0.3778 - val_accuracy: 0.8366
Epoch 16/40
30/30 [==============================] - 1s 38ms/step - loss: 0.3022 - accuracy:
0.8784 - val_loss: 0.3690 - val_accuracy: 0.8419
Epoch 17/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2888 - accuracy:
0.8866 - val_loss: 0.3612 - val_accuracy: 0.8465
Epoch 18/40
30/30 [==============================] - 1s 39ms/step - loss: 0.2765 - accuracy:
0.8917 - val_loss: 0.3546 - val_accuracy: 0.8500
Epoch 19/40
30/30 [==============================] - 1s 36ms/step - loss: 0.2645 - accuracy:
0.8961 - val_loss: 0.3490 - val_accuracy: 0.8524
Epoch 20/40
30/30 [==============================] - 1s 36ms/step - loss: 0.2533 - accuracy:
0.9024 - val_loss: 0.3436 - val_accuracy: 0.8554
Epoch 21/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2433 - accuracy:
0.9065 - val_loss: 0.3386 - val_accuracy: 0.8567
Epoch 22/40
30/30 [==============================] - 1s 37ms/step - loss: 0.2333 - accuracy:
0.9108 - val_loss: 0.3348 - val_accuracy: 0.8590
Epoch 23/40
30/30 [==============================] - 1s 38ms/step - loss: 0.2231 - accuracy:
0.9165 - val_loss: 0.3312 - val_accuracy: 0.8615
Epoch 24/40
30/30 [==============================] - 1s 50ms/step - loss: 0.2142 - accuracy:
0.9206 - val_loss: 0.3287 - val_accuracy: 0.8630
Epoch 25/40
30/30 [==============================] - 1s 48ms/step - loss: 0.2054 - accuracy:
0.9247 - val_loss: 0.3264 - val_accuracy: 0.8639
Epoch 26/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1972 - accuracy:
0.9299 - val_loss: 0.3247 - val_accuracy: 0.8660
Epoch 27/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1891 - accuracy:
0.9327 - val_loss: 0.3225 - val_accuracy: 0.8668
Epoch 28/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1818 - accuracy:
0.9354 - val_loss: 0.3231 - val_accuracy: 0.8656
Epoch 29/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1746 - accuracy:
0.9384 - val_loss: 0.3208 - val_accuracy: 0.8685
Epoch 30/40
30/30 [==============================] - 1s 36ms/step - loss: 0.1671 - accuracy:
```

```
0.9419 - val_loss: 0.3203 - val_accuracy: 0.8694
Epoch 31/40
30/30 [==============================] - 1s 36ms/step - loss: 0.1605 - accuracy:
0.9445 - val_loss: 0.3210 - val_accuracy: 0.8688
Epoch 32/40
30/30 [==============================] - 1s 37ms/step - loss: 0.1539 - accuracy:
0.9480 - val_loss: 0.3209 - val_accuracy: 0.8699
Epoch 33/40
30/30 [==============================] - 1s 39ms/step - loss: 0.1475 - accuracy:
0.9508 - val_loss: 0.3220 - val_accuracy: 0.8700
Epoch 34/40
29/30 [============================>.] - ETA: 0s - loss: 0.1419 - accuracy:
0.9528Restoring model weights from the end of the best epoch: 29.
30/30 [==============================] - 1s 38ms/step - loss: 0.1414 - accuracy:
0.9531 - val_loss: 0.3231 - val_accuracy: 0.8704
Epoch 00034: early stopping
```

The training history chart is now shorter because we stopped earlier.

[ ]:
```python
history_dict = history.history
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 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()
```



Finally, we evaluate the accuracy for the best neural network before early stopping occured.

[ ]:
```
from sklearn.metrics import accuracy_score
import numpy as np

pred = model.predict(x_val)
# Use 0.5 as the threshold
predict_classes = pred.flatten()>0.5

correct = accuracy_score(y_val,predict_classes)
print(f"Accuracy: {correct}")
```

```
Accuracy: 0.8685
```