Embedding Layer

```
import keras
keras.__version__
```

```
'2.4.3'
```

```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
```

```
from keras.datasets import imdb
from keras import preprocessing

# Number of words to consider as features
max_features = 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen = 150

# Load the data as lists of integers.
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.n
17465344/17464789 [==============================] - 0s 0us/step
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
print(len(x_train))
```

```
25000
```

```
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.layers import Flatten, Dense

model = Sequential()
# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
_________________________________
put Shape                Param #
=================================
ne, 150, 8)              80000
_________________________________
ne, 1200)                0
_________________________________
ne, 1)                   1201
=================================




_________________________________

=======] - 2s 3ms/step - loss: 0.6708 - acc: 0.5891 - val_loss: 0.4483 - val_acc: 0.8198

=======] - 1s 2ms/step - loss: 0.3713 - acc: 0.8647 - val_loss: 0.3214 - val_acc: 0.8672

=======] - 1s 2ms/step - loss: 0.2578 - acc: 0.8980 - val_loss: 0.3004 - val_acc: 0.8736

=======] - 1s 2ms/step - loss: 0.2105 - acc: 0.9189 - val_loss: 0.2930 - val_acc: 0.8748

=======] - 1s 2ms/step - loss: 0.1822 - acc: 0.9309 - val_loss: 0.2953 - val_acc: 0.8778

=======] - 1s 2ms/step - loss: 0.1550 - acc: 0.9438 - val_loss: 0.3054 - val_acc: 0.8728

=======] - 1s 2ms/step - loss: 0.1326 - acc: 0.9531 - val_loss: 0.3164 - val_acc: 0.8718

=======] - 1s 2ms/step - loss: 0.1147 - acc: 0.9619 - val_loss: 0.3220 - val_acc: 0.8694

=======] - 1s 2ms/step - loss: 0.1015 - acc: 0.9664 - val_loss: 0.3351 - val_acc: 0.8708
```

```
=======] - 1s 2ms/step - loss: 0.0831 - acc: 0.9745 - val_loss: 0.3495 - val_acc: 0.8636
```

Pre-trained Embedding Layer

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mou
```

```
import os
import shutil

imdb_dir = '/content/gdrive/MyDrive/ML Assignment 3/aclImdb'
train_dir = '/content/gdrive/MyDrive/ML Assignment 3/aclImdb/train'
#train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname))
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

maxlen = 150  # We will cut reviews after 150 words
training_samples = 100  # We will be training on 100 samples
validation_samples = 10000  # We will be validating on 10000 samples
max_words = 10000  # We will only consider the top 10,000 words in the dataset

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
```

```
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# Split the data into a training set and a validation set
# But first, shuffle the data, since we started from data
# where sample are ordered (all negative first, then all positive).
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
     Found 88582 unique tokens.
     Shape of data tensor: (25000, 150)
     Shape of label tensor: (25000,)
```

```
glove_dir = '/content/gdrive/MyDrive/ML Assignment 3/glove6B'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

```
     Found 400001 word vectors.
```

```
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
```

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_8 (Embedding)      (None, 150, 100)          1000000   
_________________________________________________________________
flatten_4 (Flatten)          (None, 15000)             0         
_________________________________________________________________
dense_4 (Dense)              (None, 32)                480032    
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33        
=================================================================
Total params: 1,480,065
Trainable params: 1,480,065
Non-trainable params: 0
_________________________________________________________________
```

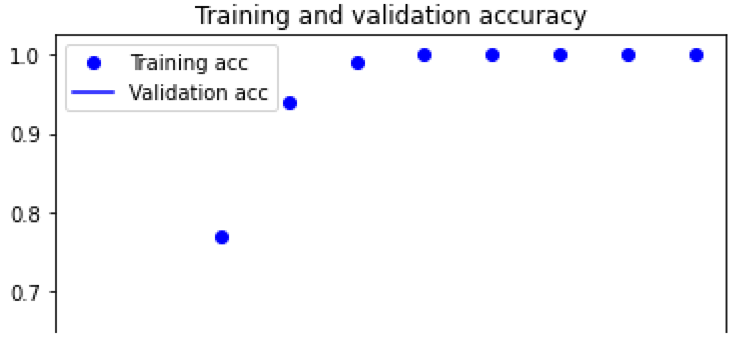
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
======] - 1s 249ms/step - loss: 2.5702 - acc: 0.5019 - val_loss: 1.8446 - val_acc: 0.4979

======] - 1s 200ms/step - loss: 1.1649 - acc: 0.5280 - val_loss: 1.3421 - val_acc: 0.4979

======] - 1s 197ms/step - loss: 0.6219 - acc: 0.6820 - val_loss: 1.1974 - val_acc: 0.4979

======] - 1s 202ms/step - loss: 0.2262 - acc: 0.9187 - val_loss: 0.8204 - val_acc: 0.5166

======] - 1s 198ms/step - loss: 0.1360 - acc: 0.9845 - val_loss: 0.7795 - val_acc: 0.5392

======] - 1s 197ms/step - loss: 0.0875 - acc: 1.0000 - val_loss: 0.7468 - val_acc: 0.5507
```

```
======] - 1s 205ms/step - loss: 0.0364 - acc: 1.0000 - val_loss: 1.0611 - val_acc: 0.5065

======] - 1s 199ms/step - loss: 0.0288 - acc: 1.0000 - val_loss: 1.0194 - val_acc: 0.5192

======] - 1s 197ms/step - loss: 0.0267 - acc: 1.0000 - val_loss: 0.9418 - val_acc: 0.5345

======] - 1s 198ms/step - loss: 0.0086 - acc: 1.0000 - val_loss: 0.8271 - val_acc: 0.5541
```

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



```
test_dir = os.path.join(imdb_dir, 'test')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(test_dir, label_type)
    for fname in sorted(os.listdir(dir_name)):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname))
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)

sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

```
model.load_weights('pre_trained_glove_model.h5')
model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.8328 - acc: 0.5549
[0.8327580690383911, 0.5549200177192688]
```

Hypertuning Embedding Layer 1 - 1000 Samples

```
import keras
keras.__version__
```

```
'2.4.3'
```

```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
```

```
from keras.datasets import imdb
from keras import preprocessing

# Number of words to consider as features
max_features = 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen = 150

# Load the data as lists of integers.
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:1000]
y_train = y_train[:1000]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
print(len(x_train))
```

```
1000
```

```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
```

```
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
__________________________________
tput Shape                Param #
==================================
one, 150, 8)              80000
__________________________________
one, 1200)                0
__________________________________
one, 1)                   1201
==================================



__________________________________

======] - 1s 11ms/step - loss: 0.6931 - acc: 0.4843 - val_loss: 0.6926 - val_acc: 0.5300

======] - 0s 4ms/step - loss: 0.6721 - acc: 0.8472 - val_loss: 0.6912 - val_acc: 0.5350

======] - 0s 4ms/step - loss: 0.6522 - acc: 0.9056 - val_loss: 0.6892 - val_acc: 0.5800

======] - 0s 4ms/step - loss: 0.6268 - acc: 0.9477 - val_loss: 0.6865 - val_acc: 0.5950

======] - 0s 4ms/step - loss: 0.5953 - acc: 0.9562 - val_loss: 0.6834 - val_acc: 0.6050

======] - 0s 4ms/step - loss: 0.5577 - acc: 0.9761 - val_loss: 0.6797 - val_acc: 0.6200

======] - 0s 3ms/step - loss: 0.5147 - acc: 0.9792 - val_loss: 0.6748 - val_acc: 0.6350

======] - 0s 3ms/step - loss: 0.4679 - acc: 0.9797 - val_loss: 0.6697 - val_acc: 0.6600

======] - 0s 4ms/step - loss: 0.4185 - acc: 0.9840 - val_loss: 0.6643 - val_acc: 0.6650

======] - 0s 4ms/step - loss: 0.3711 - acc: 0.9903 - val_loss: 0.6587 - val_acc: 0.6550
```

Hypertuning Embedding Layer 2 - 850 Samples

```
import keras
keras.__version__
```

```
'2.4.3'
```

```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
```

```
from keras.datasets import imdb
from keras import preprocessing

# Number of words to consider as features
max_features = 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen = 150

# Load the data as lists of integers.
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:850]
y_train = y_train[:850]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
print(len(x_train))
```

```
850
```

```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
# into a 2D tensor of shape  (samples, maxlen * 8)
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
__________________________________
tput Shape              Param #
==================================
one, 150, 8)            80000
__________________________________
one, 1200)              0
__________________________________
one, 1)                 1201
==================================



__________________________________

======] - 1s 11ms/step - loss: 0.6923 - acc: 0.5447 - val_loss: 0.6940 - val_acc: 0.4882

======] - 0s 4ms/step - loss: 0.6696 - acc: 0.7993 - val_loss: 0.6941 - val_acc: 0.4941

======] - 0s 4ms/step - loss: 0.6500 - acc: 0.9138 - val_loss: 0.6942 - val_acc: 0.5059

======] - 0s 4ms/step - loss: 0.6301 - acc: 0.9438 - val_loss: 0.6950 - val_acc: 0.4882

======] - 0s 4ms/step - loss: 0.6010 - acc: 0.9460 - val_loss: 0.6960 - val_acc: 0.4765

======] - 0s 4ms/step - loss: 0.5670 - acc: 0.9516 - val_loss: 0.6963 - val_acc: 0.4765

======] - 0s 4ms/step - loss: 0.5332 - acc: 0.9584 - val_loss: 0.6971 - val_acc: 0.4647

======] - 0s 4ms/step - loss: 0.4872 - acc: 0.9794 - val_loss: 0.6976 - val_acc: 0.4765

======] - 0s 4ms/step - loss: 0.4432 - acc: 0.9855 - val_loss: 0.6999 - val_acc: 0.4824

======] - 0s 4ms/step - loss: 0.4004 - acc: 0.9867 - val_loss: 0.7018 - val_acc: 0.4824
```

Hypertuning Embedding Layer 3 - 750 Samples

```
import keras
keras.__version__
```

```
'2.4.3'
```

```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
```

```
from keras.datasets import imdb
from keras import preprocessing

# Number of words to consider as features
max_features = 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen = 150

# Load the data as lists of integers.
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:750]
y_train = y_train[:750]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
print(len(x_train))
```

```
750
```

```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
__________________________________
tput Shape                Param #
==================================
one, 150, 8)              80000
__________________________________
one, 1200)                0
__________________________________
one, 1)                   1201
==================================




__________________________________

======] - 1s 11ms/step - loss: 0.6919 - acc: 0.5258 - val_loss: 0.6952 - val_acc: 0.4533

======] - 0s 4ms/step - loss: 0.6710 - acc: 0.7942 - val_loss: 0.6960 - val_acc: 0.4600

======] - 0s 4ms/step - loss: 0.6523 - acc: 0.8755 - val_loss: 0.6969 - val_acc: 0.4733

======] - 0s 5ms/step - loss: 0.6250 - acc: 0.9316 - val_loss: 0.6974 - val_acc: 0.4733

======] - 0s 4ms/step - loss: 0.5980 - acc: 0.9518 - val_loss: 0.6981 - val_acc: 0.4733

======] - 0s 5ms/step - loss: 0.5673 - acc: 0.9539 - val_loss: 0.6988 - val_acc: 0.4733

======] - 0s 5ms/step - loss: 0.5310 - acc: 0.9697 - val_loss: 0.6998 - val_acc: 0.4600

======] - 0s 5ms/step - loss: 0.4921 - acc: 0.9677 - val_loss: 0.7005 - val_acc: 0.4467

======] - 0s 4ms/step - loss: 0.4517 - acc: 0.9696 - val_loss: 0.7014 - val_acc: 0.4667

======] - 0s 4ms/step - loss: 0.4029 - acc: 0.9647 - val_loss: 0.7020 - val_acc: 0.4733
```

Hypertuning Embedding Layer 4 - 900 Samples

```
import keras
keras.__version__
```

```
'2.4.3'
```

```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
```

```
from keras.datasets import imdb
from keras import preprocessing

# Number of words to consider as features
max_features = 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen = 150

# Load the data as lists of integers.
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:900]
y_train = y_train[:900]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vi
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vi
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
print(len(x_train))
```

```
900
```

```
from keras.models import Sequential
from keras.layers import Flatten, Dense
```

```
model = Sequential()
# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
__________________________________
tput Shape                Param #
==================================
one, 150, 8)              80000
__________________________________
one, 1200)                0
__________________________________
one, 1)                   1201
==================================



__________________________________

======] - 1s 10ms/step - loss: 0.6918 - acc: 0.5087 - val_loss: 0.6922 - val_acc: 0.5167

======] - 0s 4ms/step - loss: 0.6706 - acc: 0.8263 - val_loss: 0.6913 - val_acc: 0.5556

======] - 0s 4ms/step - loss: 0.6495 - acc: 0.9258 - val_loss: 0.6903 - val_acc: 0.5389

======] - 0s 3ms/step - loss: 0.6231 - acc: 0.9547 - val_loss: 0.6895 - val_acc: 0.5500

======] - 0s 4ms/step - loss: 0.5934 - acc: 0.9712 - val_loss: 0.6884 - val_acc: 0.5444

======] - 0s 4ms/step - loss: 0.5602 - acc: 0.9673 - val_loss: 0.6867 - val_acc: 0.5444

======] - 0s 4ms/step - loss: 0.5213 - acc: 0.9777 - val_loss: 0.6852 - val_acc: 0.5500

======] - 0s 4ms/step - loss: 0.4748 - acc: 0.9718 - val_loss: 0.6839 - val_acc: 0.5611

======] - 0s 4ms/step - loss: 0.4254 - acc: 0.9823 - val_loss: 0.6830 - val_acc: 0.5722
```

```
======] - 0s 3ms/step - loss: 0.3768 - acc: 0.9888 - val_loss: 0.6818 - val_acc: 0.6056
```

Hypertuning Embedding Layer 5 - 875 Samples

```
import keras
keras.__version__
```

```
'2.4.3'
```

```
from keras.layers import Embedding

# The Embedding layer takes at least two arguments:
# the number of possible tokens, here 1000 (1 + maximum word index),
# and the dimensionality of the embeddings, here 64.
embedding_layer = Embedding(1000, 64)
```

```
from keras.datasets import imdb
from keras import preprocessing

# Number of words to consider as features
max_features = 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen = 150

# Load the data as lists of integers.
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = x_train[:875]
y_train = y_train[:875]

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
print(len(x_train))
```

```
875
```

```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
__________________________________
tput Shape                Param #
==================================
one, 150, 8)              80000
__________________________________
one, 1200)                0
__________________________________
one, 1)                   1201
==================================



__________________________________

======] - 1s 11ms/step - loss: 0.6931 - acc: 0.4920 - val_loss: 0.6941 - val_acc: 0.4857

======] - 0s 4ms/step - loss: 0.6726 - acc: 0.7996 - val_loss: 0.6937 - val_acc: 0.4971

======] - 0s 4ms/step - loss: 0.6530 - acc: 0.9294 - val_loss: 0.6933 - val_acc: 0.4857

======] - 0s 4ms/step - loss: 0.6285 - acc: 0.9506 - val_loss: 0.6927 - val_acc: 0.4857

======] - 0s 4ms/step - loss: 0.6031 - acc: 0.9466 - val_loss: 0.6919 - val_acc: 0.4971

======] - 0s 4ms/step - loss: 0.5682 - acc: 0.9560 - val_loss: 0.6911 - val_acc: 0.4971
```

```
======] - 0s 4ms/step - loss: 0.5313 - acc: 0.9629 - val_loss: 0.6902 - val_acc: 0.5143

======] - 0s 4ms/step - loss: 0.4848 - acc: 0.9716 - val_loss: 0.6889 - val_acc: 0.5257

======] - 0s 4ms/step - loss: 0.4432 - acc: 0.9643 - val_loss: 0.6877 - val_acc: 0.5429

======] - 0s 4ms/step - loss: 0.3988 - acc: 0.9789 - val_loss: 0.6862 - val_acc: 0.5543
```

✓ 1s completed at 5:22 PM ● ✕