```
https://github.com/polyaxon/deep-learning-with-python-notebooks-on-
polyaxon/blob/master/3.5-classifying-movie-reviews.ipynb

from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)

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

```
9999
```

```
# word_index is a dictionary mapping words to an integer index
word_index = imdb.get_word_index()
# We reverse it, mapping integer indices to words
reverse_word_index = dict([(value, key) for (key, value) in
word_index.items()])
# We decode the review; note that our indices were offset by 3
# because 0, 1 and 2 are reserved indices for "padding", "start of
sequence", and "unknown".
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in
train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of
results[i] to 1s
    return results

# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)

x_train[0]
```

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

```
# Our vectorized labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

from keras import models
from keras import layers
```

```python
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
C:\Users\STES\anaconda3\Lib\site-packages\keras\src\layers\core\
dense.py:88: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

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

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

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

```python
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 120ms/step - accuracy: 0.6903 - loss:
0.6158 - val_accuracy: 0.8676 - val_loss: 0.3998
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8879 - loss:
0.3449 - val_accuracy: 0.8823 - val_loss: 0.3161
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9170 - loss:
0.2499 - val_accuracy: 0.8696 - val_loss: 0.3135
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9330 - loss:
0.1973 - val_accuracy: 0.8896 - val_loss: 0.2742
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9499 - loss:
0.1594 - val_accuracy: 0.8865 - val_loss: 0.2822
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9577 - loss:
0.1349 - val_accuracy: 0.8841 - val_loss: 0.2877
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9628 - loss:
0.1185 - val_accuracy: 0.8798 - val_loss: 0.3038
Epoch 8/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9702 - loss:
0.1014 - val_accuracy: 0.8744 - val_loss: 0.3320
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9753 - loss:
0.0880 - val_accuracy: 0.8762 - val_loss: 0.3341
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9808 - loss:
0.0723 - val_accuracy: 0.8751 - val_loss: 0.3490
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9852 - loss:
0.0603 - val_accuracy: 0.8732 - val_loss: 0.3724
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9884 - loss:
0.0529 - val_accuracy: 0.8773 - val_loss: 0.3878
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9908 - loss:
0.0428 - val_accuracy: 0.8741 - val_loss: 0.4092
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9928 - loss:
0.0365 - val_accuracy: 0.8710 - val_loss: 0.4641
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9951 - loss:
0.0310 - val_accuracy: 0.8706 - val_loss: 0.4647
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9961 - loss:
0.0275 - val_accuracy: 0.8714 - val_loss: 0.4847
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9947 - loss:
0.0244 - val_accuracy: 0.8722 - val_loss: 0.5106
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9953 - loss:
0.0225 - val_accuracy: 0.8708 - val_loss: 0.5414
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9988 - loss:
0.0154 - val_accuracy: 0.8664 - val_loss: 0.5612
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9995 - loss:
0.0122 - val_accuracy: 0.8654 - val_loss: 0.5891
```

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

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8511 - loss:
0.6522
```

```
results
```

```
[0.6396853923797607, 0.854640007019043]
```