Deep Learning models
for classification

CLASSIFICATION

Classification and regression glossary

- *Binary classification*—A classification task where each input sample should be categorized into two exclusive categories.
- *Multiclass classification*—A classification task where each input sample should be categorized into more than two categories: for instance, classifying handwritten digits.
- *Multilabel classification*—A classification task where each input sample can be assigned multiple labels. For instance, a given image may contain both a cat and a dog and should be annotated both with the "cat" label and the "dog" label. The number of labels per image is usually variable.

Binary classification

- IMDB Dataset -

OVERVIEW

The IMDB (Internet Movie Database) dataset has reviews (positive and negative) for about 50000 movies.

Half the reviews for training and half the reviews for testing

Each set of 25000 reviews has 50% positive and 50% negative

The data has already been pre-processed.

Each review is a paragraph (a sequence of words)

Each word in the reviews has been transformed into an integer (each one stands for a specific word in a dictionary).

NEURAL NETWORK FOR IMDB Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from tensorflow import keras
```

```
from tensorflow.keras import layers
from tensorflow.keras.datasets import imdb
```

Load the IMDB Dataset

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

- Ignore rare words
- Keep the top 10000 most frequently occurring words in the train data set

NEURAL NETWORK FOR IMDB Dataset

```
# number of words in first 4 reviews

print(len(train_data[0]),
    len(train_data[1]),
    len(train_data[2]),
    len(train_data[3]))
```

```
218 189 141 550
```

- Each word in the reviews has been transformed into an integer
- Each one stands for a specific word in a dictionary

```
# show integer-encoded words of 1st review
train_data[:1]
```

```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 83
8, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 1
3, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 1
3, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 378
5, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407,
16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1
029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480,
5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 10
3, 32, 15, 16, 5345, 19, 178, 32])],
```

```
# show class of 1st review (0 is positive, 1 is negative)
test_labels[:1]
```

```
array([0])
```

NEURAL NETWORK FOR IMDB Dataset

.

```
string1 = decode_review1[:2000]
string1
```

```
". this film was just brilliant casting location scenery story direction everyone's really suited the part they pla
yed and you could just imagine being there robert . is an amazing actor and now the same being director . father ca
me from the same scottish island as myself so i loved the fact there was a real connection with this film the witty
remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was relea
sed for . and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it wa
s so sad and you know what they say if you cry at a film it must have been good and this definitely was also . to t
he two little boy's that played the . of norman and paul they were just brilliant children are often left out of th
e . list i think because the stars that play them all grown up are such a big profile for the whole film but these
children are amazing and should be praised for what they have done don't you think the whole story was so lovely be
cause it was true and was someone's life after all that was shared with us all"
```

```
# number of letters in 1st review
len(string1)
```

```
1113
```

```
# number of words in 1st review
len(string1.split())
```

```
218
```

ENCODE THE SEQUENCE OF INTEGERS INTO VECTORS OF 0s AND 1s

```
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.shape
```

```
(25000, 10000)
```

```
# transform labels from binary int64 to as float32
```

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

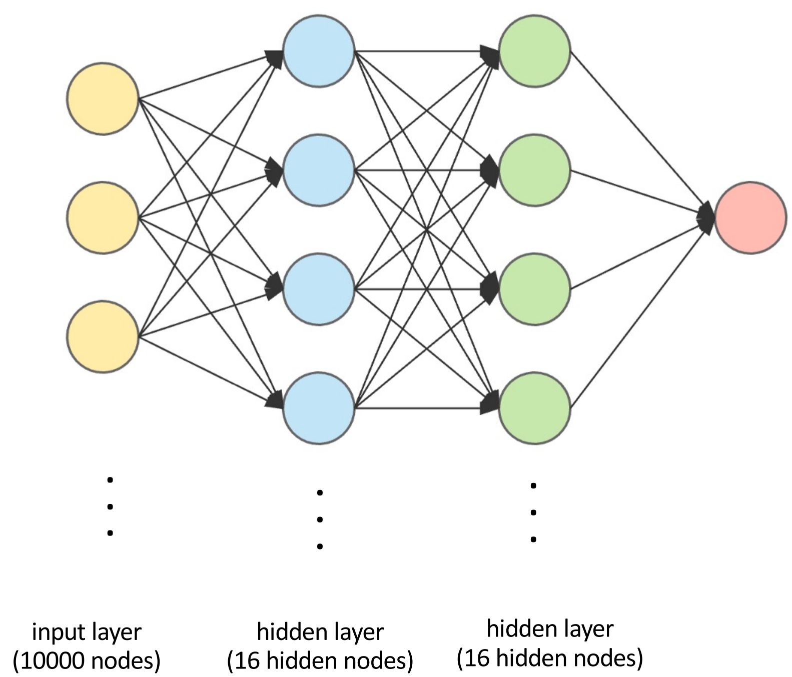
- Reviews have different lengths
- Store each review into a binary vector of length 10000
- For example the vector [8, 5] would be in a vector filled with 0s, except for entries 8 and 5 which would be filled with 1s
- There are 25000 reviews in the train set

BUILD THE NEURAL NETWORK

3. Build Network

```
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'] )
```



TRAIN THE NEURAL NETWORK

4. Train the NN

```
# Split the train data set
# into validation and "actual train" sets
```

```
# set aside the validation set
x_val = x_train[:10000]
y_val = y_train[:10000]

# define the actual train set
partial_x_train = x_train[10000:]
partial_y_train = y_train[10000:]
```

- Split train subset into validation and actual train portions
- There are 25000 reviews in the train set
- Use first 10000 reviews (from the train set) for validation
- Use remaining 15000 reviews for training

- Train (fit) the model for 20 epochs

TRAIN THE NEURAL NETWORK

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

- We do not update the gradient vector with each observation to reduce computer time. Instead we do it in batches of 512 observations
- Split the train data set into batches of 512 observations
- After the batch gives the new gradient we move in that direction and update the loss value
- After all batches are processed we get the mimized loss
- We repeat the process 20 times (each iteration over all the training data is called an *epoch*)

TRAIN THE NEURAL NETWORK

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

- After calling `fit` the model will start to iterate on the training data in batches of 512 observations, 20 times over (each iteration over all the training data is called an *epoch*).
- For each batch, the model will compute the gradient of the loss and update the weights (in the gradient direction) reducing the value of the loss for the batch.
- There will be 15000/512 = 29 gradient updates per epoch.
- After 20 epochs, the model will have performed 29 x 20 = 580 gradient updates.
- We expect that the loss will be sufficiently low that the model is capable of classifying the newswires with high accuracy

TRAIN THE NEURAL NETWORK

4. Train the NN

```
# set aside the validation set
partial_x_train = x_train[10000:]
partial_y_train = y_train[10000:]

# test set
x_val = x_train[:10000]
y_val = y_train[:10000]
```

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

The call to model.fit() returns a History object.
This object has a member .history, which is a dictionary with the loss and accuracy after each epoch

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

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

```
# see the metrics in a dataframe format
```

```
df9 = pd.DataFrame(history_dict)
df9[:5]
```

	loss	accuracy	val_loss	val_accuracy
0	0.537925	0.776267	0.432548	0.8275
1	0.339545	0.890933	0.375541	0.8479
2	0.257223	0.914667	0.291091	0.8882
3	0.204734	0.933933	0.278373	0.8890
4	0.175646	0.944067	0.279144	0.8869

TRAIN THE NEURAL NETWORK

4. Train the NN

```
# set aside the validation set
partial_x_train = x_train[10000:]
partial_y_train = y_train[10000:]

# test set
x_val = x_train[:10000]
y_val = y_train[:10000]
```

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

The call to model.fit() returns a History object.
This object has a member .history, which is a dictionary with the loss and accuracy after each epoch

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

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

Plot Validation loss to prevent overfitting

```
# see the metrics in a dataframe format
```

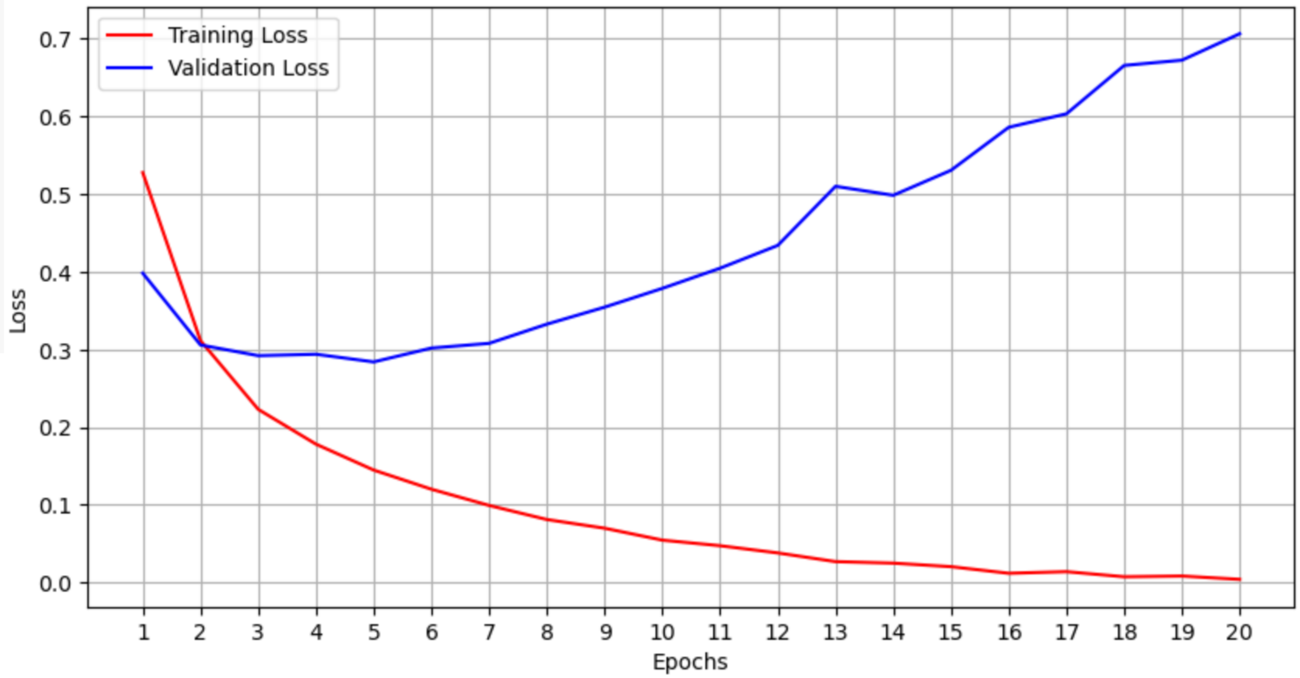
```
df9 = pd.DataFrame(history_dict)
df9[:5]
```

	Train loss	accuracy	Validation val_loss	val_accuracy
0	0.537925	0.776267	0.432548	0.8275
1	0.339545	0.890933	0.375541	0.8479
2	0.257223	0.914667	0.291091	0.8882
3	0.204734	0.933933	0.278373	0.8890
4	0.175646	0.944067	0.279144	0.8869

Plot Train and Validation loss

```
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,21)
```

```
plt.figure(figsize=(10,5))
plt.plot(epochs,loss_values,'r',
         label='Training Loss')
plt.plot(epochs,val_loss_values,'b',
         label='Validation Loss')
plt.xticks(epochs)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
```



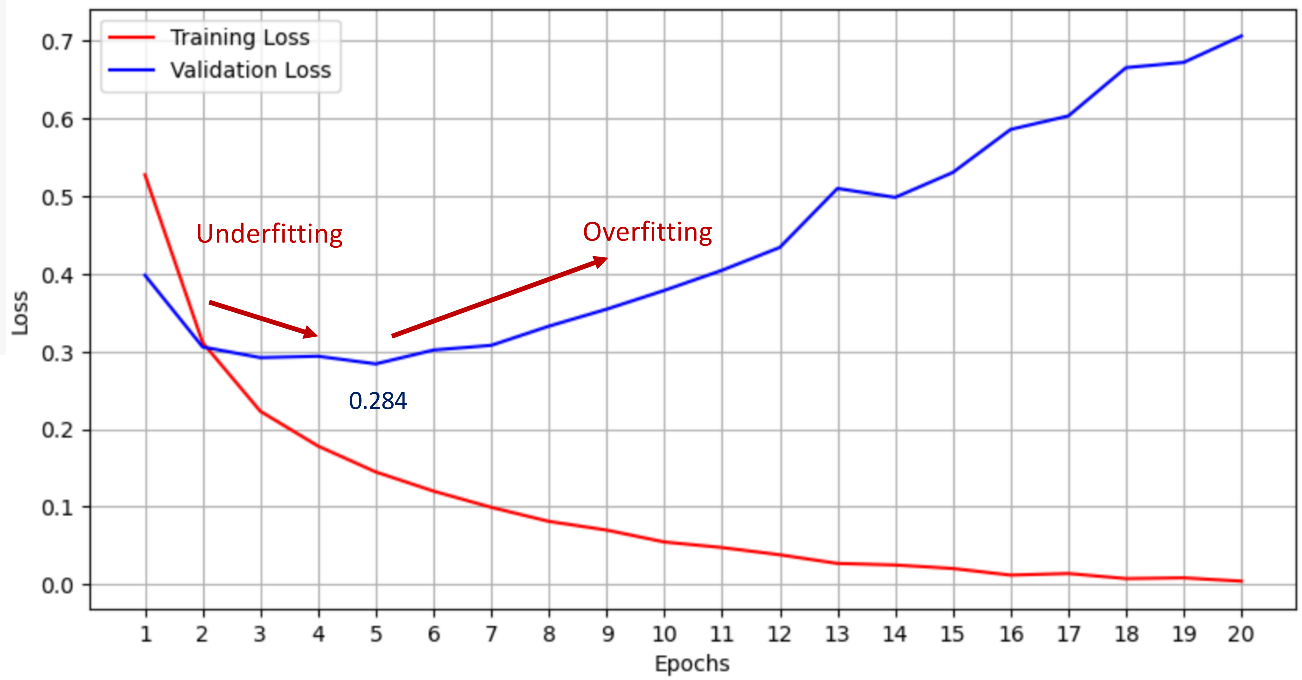
Plot Train and Validation loss

```
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,21)
```

```
plt.figure(figsize=(10,5))
plt.plot(epochs,loss_values,'r',
         label='Training Loss')
plt.plot(epochs,val_loss_values,'b',
         label='Validation Loss')
plt.xticks(epochs)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
```

```
df9[df9.val_loss==df9.val_loss.min()]
```

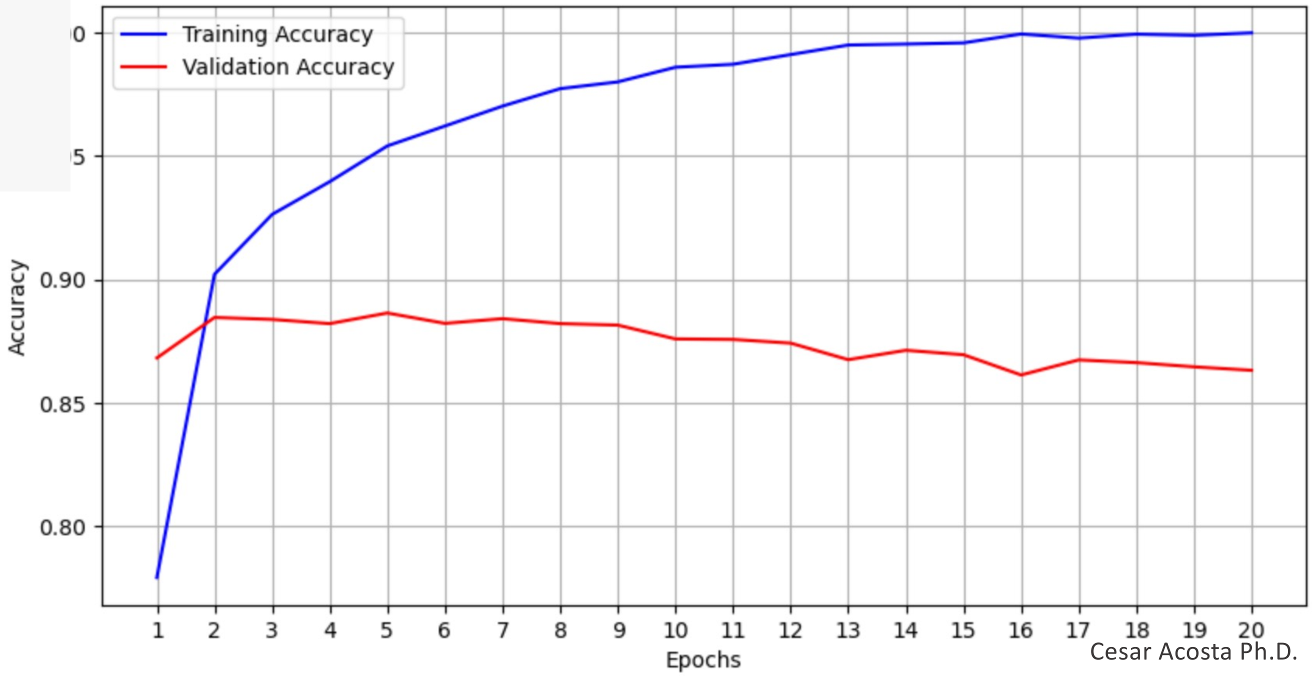
	loss	accuracy	val_loss	val_accuracy
5	0.144871	0.954067	0.28409	0.8864



Plot Train and Validation Accuracy

```
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
```

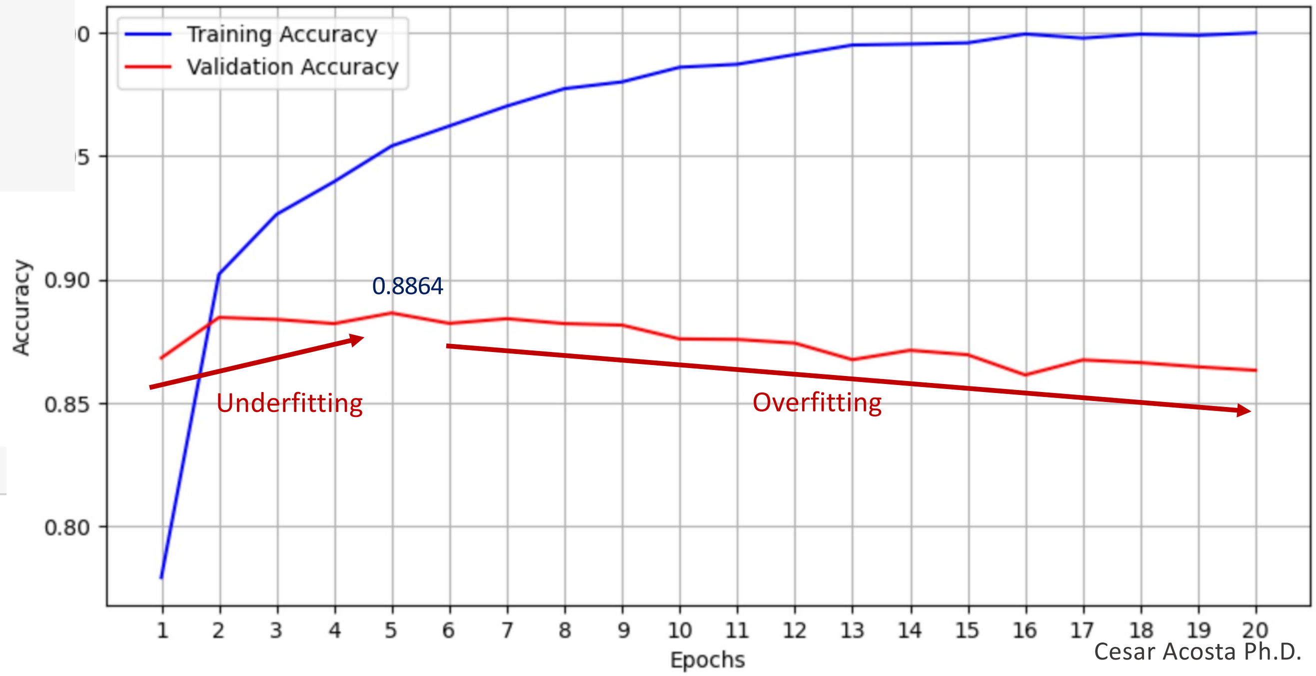
```
plt.figure(figsize=(10,5))
plt.plot(epochs,df9.accuracy,'b',
         label='Training Accuracy')
plt.plot(epochs,df9.val_accuracy,'r',
         label='Validation Accuracy')
plt.xticks(epochs)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```



Plot Train and Validation Accuracy

```python
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
```

```python
plt.figure(figsize=(10,5))
plt.plot(epochs,df9.accuracy,'b',
         label='Training Accuracy')
plt.plot(epochs,df9.val_accuracy,'r',
         label='Validation Accuracy')
plt.xticks(epochs)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```




```python
df9[df9.val_accuracy==df9.val_accuracy.max()]
```

	loss	accuracy	val_loss	val_accuracy
5	0.144871	0.954067	0.28409	0.8864

Retrain the model from scratch (Use all the train set)

```
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=5,batch_size=512);
```

```
test_loss,test_acc = model.evaluate(x_test,y_test)
```

```
test_loss
```

```
0.3159539997577667
```

```
# test accuracy rate
```

```
test_acc
```

```
0.8787999749183655
```

← Test accuracy rate

Multiclass classification

- Reuters Dataset -

REUTERS DATASET

The objective is to classify newswires into one of 46 topics (multiclass classification problem)

The Reuters dataset has 11228 newswires already split into train and test set

There are 8982 newswires for training and 2246 for testing

The data has already been pre-processed.

Each newswire is a paragraph (a sequence of words)

Each word in the newswire has been transformed into an sequence of integers (where each integer stands for a specific word)

ENCODING METHODS

- *Binary classification*—A classification task where each input sample should be categorized into two exclusive categories.
- *Multiclass classification*—A classification task where each input sample should be categorized into more than two categories: for instance, classifying handwritten digits.

- There are two ways to handle labels in multiclass classification:

One-hot encoding

– Encoding the labels via categorical encoding (also known as one-hot encoding) and using `categorical_crossentropy` as a loss function

label encoding

– Encoding the labels as integers and using the `sparse_categorical_crossentropy` loss function

NEURAL NETWORK FOR REUTERS Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from tensorflow import keras

from tensorflow.keras.utils import to_categorical

from tensorflow.keras import layers
from tensorflow.keras.datasets import reuters
```

NEURAL NETWORK FOR REUTERS Dataset

```
from tensorflow.keras.datasets import reuters

(train_data, train_labels),\
(test_data, test_labels) = reuters.load_data(num_words=10000)
```

Rare words are to be discarded.
Keep the top 10000 most frequently occurring words in the train data set.

```
print(train_data.shape,train_labels.shape)
```

```
(8982,) (8982,)
```

```
print(test_data.shape,test_labels.shape)
```

```
(2246,) (2246,)
```

```
# See 10th review topic from train set
train_labels[10]
```

```
3
```

10th review is category 3
(there are 46 categories)

NEURAL NETWORK FOR REUTERS Dataset

Encoding the input data

```
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)
```

```
print(x_train.shape,x_test.shape)
```

```
(8982, 10000) (2246, 10000)
```

NEURAL NETWORK FOR REUTERS Dataset

Encoding the labels

```
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)
```

```
print(y_train.shape,y_test.shape)
```

```
(8982, 46) (2246, 46)
```

NEURAL NETWORK FOR REUTERS Dataset

Encoding the labels

```
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)
```

```
print(y_train.shape,y_test.shape)
```

```
(8982, 46) (2246, 46)
```

```
from tensorflow.keras.utils import to_categorical
```

```
y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)
```

```
print(y_train.shape,y_test.shape)
```

```
(8982, 46) (2246, 46)
```

NEURAL NETWORK FOR REUTERS Dataset

Build the model

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

← hidden layer
← hidden layer
← output layer

Since the output layer is 46-dimensional, avoid hidden layers with less than 46 hidden units

Compiling the model

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

← One-hot encoding

Validation

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

← Train set is made with the last 7982 newswires

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

← Validation set is made with the first 1000 newswires

Train the model

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

TRAIN THE NEURAL NETWORK

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

- After calling `fit` the model will start to iterate on the training data in batches of 512 observations, 20 times over (each iteration over all the training data is called an *epoch*).
- For each batch, the model will compute the gradient of the loss and update the weights (in the gradient direction) reducing the value of the loss for the batch.
- There will be 7982/512 = 16 gradient updates per epoch.
- After 20 epochs, the model will have performed 16 x 20 = 320 gradient updates.
- We expect that the loss will be sufficiently low that the model is capable of classifying the newswires with high accuracy

TRAIN THE NEURAL NETWORK

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

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

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

```
df9 = pd.DataFrame(history_dict)
df9.index = range(1,21)
```

	Train loss	Train accuracy	val_loss	val_accuracy
1	2.545177	0.490980	1.665100	0.639
2	1.391077	0.698822	1.258056	0.723
3	1.041145	0.775119	1.112869	0.752
4	0.825949	0.824355	1.017738	0.787
5	0.657540	0.866575	0.947129	0.795
6	0.528120	0.892007	0.901879	0.812
7	0.422487	0.912177	0.884048	0.814
8	0.347410	0.926084	0.954315	0.782
9	0.286693	0.934102	0.881968	0.823
10	0.240055	0.943122	0.927826	0.810

	Train loss	Train accuracy	val_loss	val_accuracy
11	0.214139	0.946379	0.918055	0.828
12	0.182144	0.951641	0.945607	0.815
13	0.166701	0.953646	0.944044	0.816
14	0.152213	0.955024	0.986204	0.802
15	0.141612	0.955901	0.999093	0.804
16	0.130503	0.956652	1.062793	0.794
17	0.124488	0.958031	1.023027	0.807
18	0.118980	0.957780	1.039983	0.816
19	0.114638	0.958031	1.026351	0.824
20	0.110513	0.958031	1.053710	0.809

TRAIN THE NEURAL NETWORK

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

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

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

```
df9 = pd.DataFrame(history_dict)
df9.index = range(1,21)
```

	Train loss	Train accuracy	val_loss	val_accuracy
1	2.545177	0.490980	1.665100	0.639
2	1.391077	0.698822	1.258056	0.723
3	1.041145	0.775119	1.112869	0.752
4	0.825949	0.824355	1.017738	0.787
5	0.657540	0.866575	0.947129	0.795
6	0.528120	0.892007	0.901879	0.812
7	0.422487	0.912177	0.884048	0.814
8	0.347410	0.926084	0.954315	0.782
9	0.286693	0.934102	0.881968	0.823
10	0.240055	0.943122	0.927826	0.810

	Train loss	Train accuracy	val_loss	val_accuracy
11	0.214139	0.946379	0.918055	0.828
12	0.182144	0.951641	0.945607	0.815
13	0.166701	0.953646	0.944044	0.816
14	0.152213	0.955024	0.986204	0.802
15	0.141612	0.955901	0.999093	0.804
16	0.130503	0.956652	1.062793	0.794
17	0.124488	0.958031	1.023027	0.807
18	0.118980	0.957780	1.039983	0.816
19	0.114638	0.958031	1.026351	0.824
20	0.110513	0.958031	1.053710	0.809

```
df9[df9.val_loss==df9.val_loss.min()]
```

	loss	accuracy	val_loss	val_accuracy
9	0.286693	0.934102	0.881968	0.823

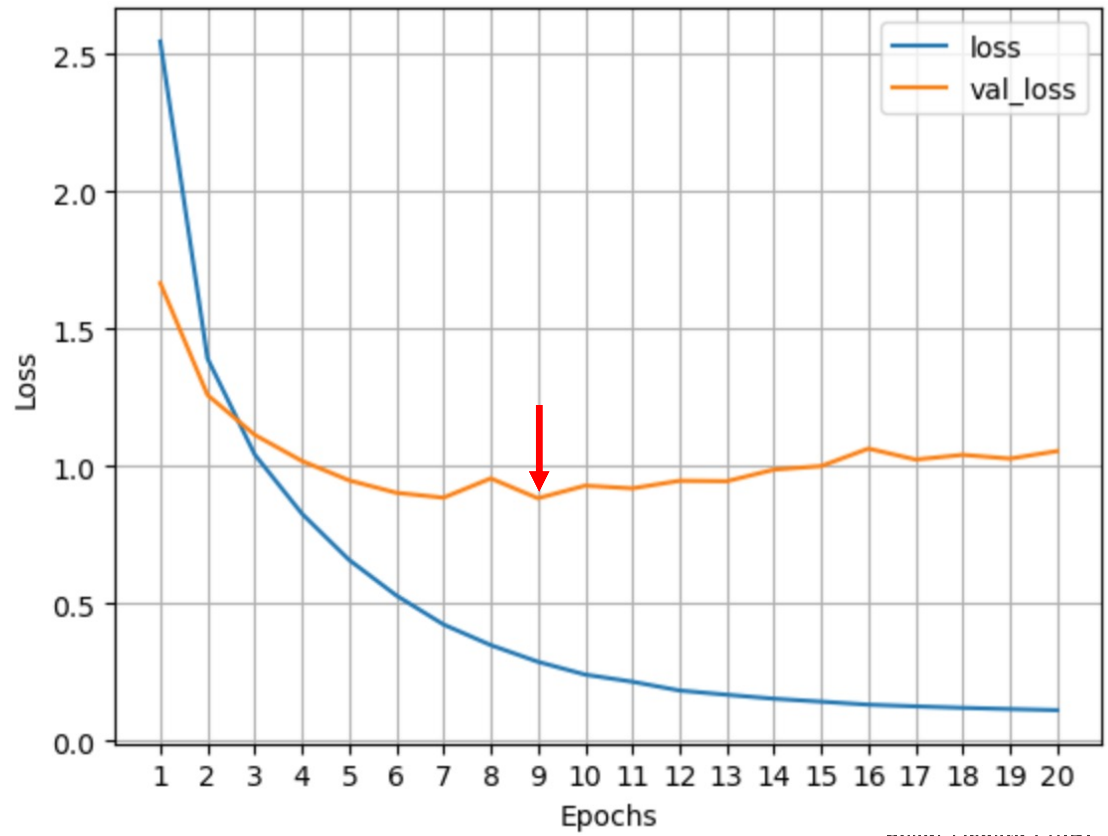
```
df9[df9.val_accuracy==df9.val_accuracy.max()]
```

	loss	accuracy	val_loss	val_accuracy
11	0.214139	0.946379	0.918055	0.828

Plot Train and Validation loss

```
df99 = df9.iloc[:,[0,2]]
```

```
df99.plot()
plt.xticks(epochs)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```



To avoid overfitting train the NN for 9 epochs

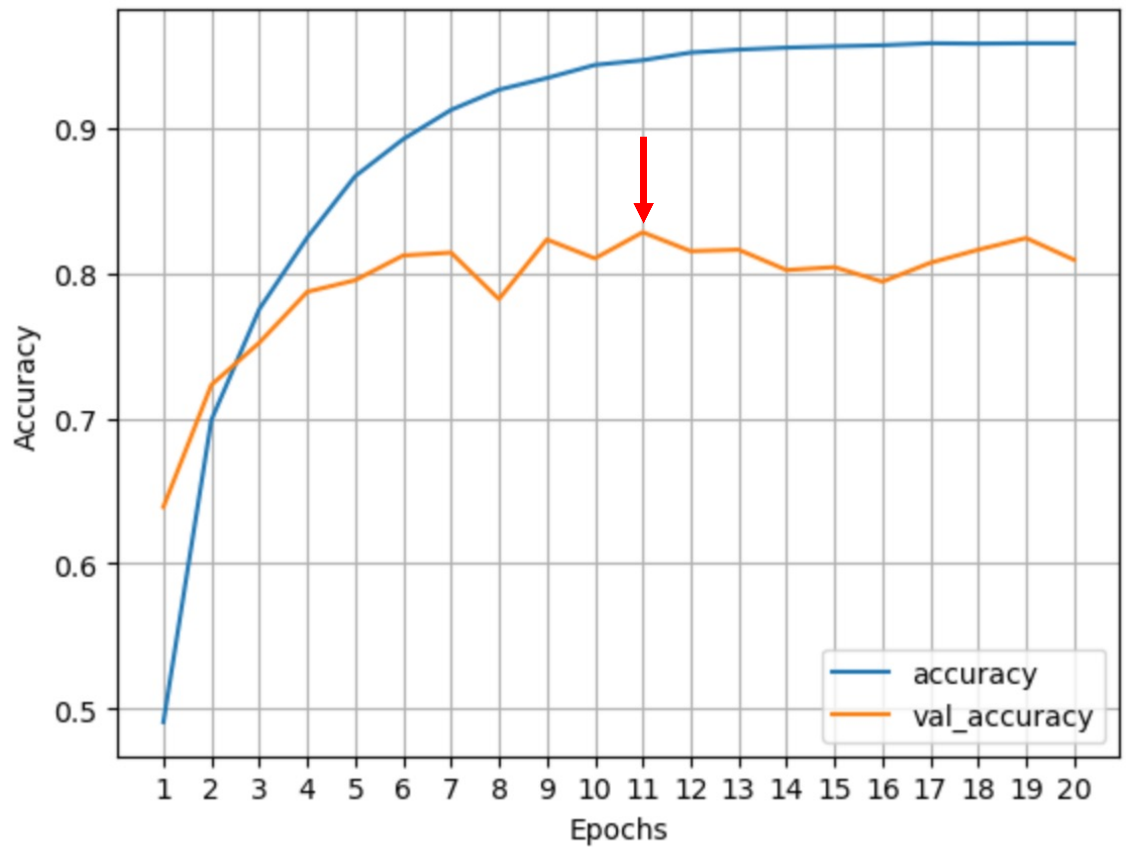
```
df9[df9.val_loss==df9.val_loss.min()]
```

	loss	accuracy	val_loss	val_accuracy
9	0.286693	0.934102	0.881968	0.823

Plot Train and Validation Accuracy

```
df99 = df9.iloc[:,[1,3]]
```

```
df99.plot()
plt.xticks(epochs)
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(loc=4)
```



To avoid overfitting train the NN for 11 epochs

```
df9[df9.val_accuracy==df9.val_accuracy.max()]
```

	loss	accuracy	val_loss	val_accuracy
11	0.214139	0.946379	0.918055	0.828

Retrain a model from scratch (11 epochs)

```
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=11,batch_size=512);
```

```
test_loss,test_acc = model.evaluate(x_test,y_test)
```

```
test_loss
```

```
0.9902769923210144
```

```
test_acc
```

```
0.7960819005966187
```

← Test accuracy rate