Experiment No. 2

Name : Tike Sahil Yogesh
Class : BE-(B3)
Roll No : B212066

In [44]:
```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all fi

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets prese
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
#/kaggle/input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset.csv
```

In [4]:
```
# Basic packages
import pandas as pd
import numpy as np
import re
import collections
import matplotlib.pyplot as plt

# Packages for data preparation
from sklearn.model_selection import train_test_split
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from keras.preprocessing.text import Tokenizer
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Packages for modeling
from keras import models
from keras import layers
from keras import regularizers
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\D_COMP_RSL-14\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [5]:
```
import os
#print(os.listdir("../input"))
import warnings
warnings.filterwarnings('ignore')
```

In [7]:
```
#importing the training data
from sklearn.datasets import load_boston
boston= load_boston();
df=pd.DataFrame(data=boston.data,columns=boston.feature_names)
df['target']=boston.target
df
```

```
df=pd.read_csv('IMDB Dataset.csv')
print(df.shape)
df.head(10)
```

```
(50000, 2)
```

Out[7]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

In [8]:
```
# Then we set some parameters that will be used throughout the notebook.

NB_WORDS = 10000  # Parameter indicating the number of words we'll put in the dictionc
VAL_SIZE = 1000  # Size of the validation set
NB_START_EPOCHS = 20  # Number of epochs we usually start to train with
BATCH_SIZE = 512  # Size of the batches used in the mini-batch gradient descent
```

Data preparation

Data cleaning

The first thing we'll do is removing stopwords. These words do not have any value for predicting the sentiment. Furthermore, as we want to build a model that can be used for other airline companies as well, we remove the mentions.

In [9]:
```
stopword_list = nltk.corpus.stopwords.words('english')
# Some words which might indicate a certain sentiment are 'no','not'
stopword_list.remove('no')
stopword_list.remove('not')
#def remove_stopwords(input_text,is_lower_case =False):
#        words = input_text.split()
#        clean_words = [word for word in words if (word not in stopword_list) and len(
#        return " ".join(clean_words)
```

In [10]:
```
def remove_stopwords(input_text,is_lower_case =False):
  words = input_text.split()
  if is_lower_case:
    filtered_words = [word for word in words if word not in stopword_list]
  else:
    filtered_words = [word for word in words if word.lower() not in stopword_list]
```

```
  filtered_text = ' '.join(filtered_words)
  return filtered_text

def remove_mentions(input_text):
        return re.sub(r'@\w+', '', input_text)

df.review = df.review.apply(remove_stopwords).apply(remove_mentions)
df.head()
```

Out[10]:

	review	sentiment
0	One reviewers mentioned watching 1 Oz episode ...	positive
1	wonderful little production. The f...	positive
2	thought wonderful way spend time hot summer we...	positive
3	Basically there's family little boy (Jake) thi...	negative
4	Petter Mattei's "Love Time Money" visually stu...	positive

Train-Test split

The evaluation of the model performance needs to be done on a separate test set. As such, we can estimate how well the model generalizes. This is done with the train_test_split method of scikit-learn.

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(df.review, df.sentiment, test_size
print('# Train data samples:', X_train.shape[0])
print('# Test data samples:', X_test.shape[0])
assert X_train.shape[0] == y_train.shape[0]
assert X_test.shape[0] == y_test.shape[0]
```

```
# Train data samples: 45000
# Test data samples: 5000
```

In [12]:
```
# to remove  escape sequence and blank values
X_train.replace(to_replace = [r"\\t|\\n|\\r","\t|\n|\r"], value=["",""] , regex=True,
X_train.replace(r'^\s*$', np.nan, regex=True, inplace=True)
X_train.dropna(axis = 0, how = 'any', inplace = True)
```

In [13]:
```
# removing the non ascii words
X_train = X_train.str.encode('ascii','ignore').str.decode('ascii')
```

In [14]:
```
# removing punctuations
import string
def remove_punctuation(text):
  for punctuation in string.punctuation:
    text = text.replace(punctuation,'')
    return text
```

In [15]:
```
X_train = X_train.apply(remove_punctuation)
# To remove special characters, html text and url
def remove_sub(text):
  text = re.sub('[^a-zA-Z0-9\s]','',text)
  html_pattern = re.compile('<.*?>')
```

```
  text = html_pattern.sub(r' ',text)
  url = re.compile(r'https?://\S+|www\.\S+')
  return url.sub(r' ',text)
```

In [16]:
```
X_train = X_train.apply(remove_sub)
tk = Tokenizer(num_words=NB_WORDS,
               filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
               lower=True,
               split=" ")
tk.fit_on_texts(X_train)
```

In [17]:
```
print('Fitted tokenizer on {} documents'.format(tk.document_count))
print('{} words in dictionary'.format(tk.num_words))
print('Top 5 most common words are:', collections.Counter(tk.word_counts).most_common(
```

```
Fitted tokenizer on 45000 documents
10000 words in dictionary
Top 5 most common words are: [('br', 102265), ('movie', 74995), ('film', 66899), ('no
t', 53704), ('one', 45776)]
```

After having created the dictionary we can convert the text to a list of integer indexes. This is done with the text_to_sequences method of the Tokenizer.

In [19]:
```
X_train_seq = tk.texts_to_sequences(X_train)
X_test_seq = tk.texts_to_sequences(X_test)

print('"{}" is converted into {}'.format(X_train[0], X_train_seq[0]))
```

```
"One reviewers mentioned watching 1 Oz episode hooked right exactly happened mebr br
The first thing struck Oz brutality unflinching scenes violence set right word GO Tru
st me not show faint hearted timid show pulls no punches regards drugs sex violence h
ardcore classic use wordbr br It called OZ nickname given Oswald Maximum Security Sta
te Penitentary focuses mainly Emerald City experimental section prison cells glass fr
onts face inwards privacy not high agenda Em City home manyAryans Muslims gangstas La
tinos Christians Italians Irish moreso scuffles death stares dodgy dealings shady agr
eements never far awaybr br I would say main appeal show due fact goes shows dare For
get pretty pictures painted mainstream audiences forget charm forget romanceOZ mess a
round first episode ever saw struck nasty surreal say ready it watched more developed
taste Oz got accustomed high levels graphic violence Not violence injustice crooked g
uards wholl sold nickel inmates wholl kill order get away it well mannered middle cla
ss inmates turned prison bitches due lack street skills prison experience Watching Oz
may become comfortable uncomfortable viewingthats get touch darker side" is converted
into [41, 1431, 989, 5947, 740, 2, 343, 157, 59, 1148, 59, 4, 862, 63, 746, 762, 543
3, 2882, 27, 6, 722, 170, 125, 214, 42, 7946, 4416, 120, 850, 1156, 798, 8313, 325, 3
063, 11, 1308, 3583, 1, 1, 584, 7946, 3805, 1317, 902, 351, 200, 11, 561, 478, 143, 9
99, 2180, 203, 238, 9394, 227, 9, 748, 227, 42, 7, 5, 456, 227, 470, 1737, 203, 238,
431, 1504, 5399, 19, 309, 1071, 40, 541, 134, 32, 30, 19, 1699, 1780, 1529, 1090, 29
0, 367, 9533, 584, 104, 270, 367, 1, 574, 43, 1407, 7433, 8645, 40, 2198, 497, 73, 39
0, 12, 13, 141, 207, 61]
```

These integers should now be converted into a one-hot encoded features.

In [20]:
```
def one_hot_seq(seqs, nb_features = NB_WORDS):
    ohs = np.zeros((len(seqs), nb_features))
    for i, s in enumerate(seqs):
        ohs[i, s] = 1.
    return ohs
```

In [21]:
```
X_train_oh = one_hot_seq(X_train_seq)
X_test_oh = one_hot_seq(X_test_seq)

print('"{}" is converted into {}'.format(X_train_seq[0], X_train_oh[0]))
print('For this example we have {} features with a value of 1.'.format(X_train_oh[0].s
```

```
"[41, 1431, 989, 5947, 740, 2, 343, 157, 59, 1148, 59, 4, 862, 63, 746, 762, 5433, 28
82, 27, 6, 722, 170, 125, 214, 42, 7946, 4416, 120, 850, 1156, 798, 8313, 325, 3063,
11, 1308, 3583, 1, 1, 584, 7946, 3805, 1317, 902, 351, 200, 11, 561, 478, 143, 999, 2
180, 203, 238, 9394, 227, 9, 748, 227, 42, 7, 5, 456, 227, 470, 1737, 203, 238, 431,
1504, 5399, 19, 309, 1071, 40, 541, 134, 32, 30, 19, 1699, 1780, 1529, 1090, 290, 36
7, 9533, 584, 104, 270, 367, 1, 574, 43, 1407, 7433, 8645, 40, 2198, 497, 73, 390, 1
2, 13, 141, 207, 61]" is converted into [0. 1. 1. ... 0. 0. 0.]
For this example we have 93.0 features with a value of 1.
```

Converting the target classes to numbers We need to convert the target classes to numbers as well, which in turn are one-hot-encoded with the to_categorical method in keras

In [22]:
```
le = LabelEncoder()
y_train_le = le.fit_transform(y_train)
y_test_le = le.transform(y_test)
y_train_oh = to_categorical(y_train_le)
y_test_oh = to_categorical(y_test_le)

print('"{}" is converted into {}'.format(y_train[0], y_train_le[0]))
print('"{}" is converted into {}'.format(y_train_le[0], y_train_oh[0]))
```

```
"positive" is converted into 0
"0" is converted into [1. 0.]
```

Splitting of a validation set

Now that our data is ready, we split of a validation set. This validation set will be used to evaluate the model performance when we tune the parameters of the model.

In [23]:
```
X_train_rest, X_valid, y_train_rest, y_valid = train_test_split(X_train_oh, y_train_oh

assert X_valid.shape[0] == y_valid.shape[0]
assert X_train_rest.shape[0] == y_train_rest.shape[0]

print('Shape of validation set:',X_valid.shape)
```

```
Shape of validation set: (4500, 10000)
```

Deep learning

Baseline model We start with a model with 2 densely connected layers of 64 hidden elements. The input_shape for the first layer is equal to the number of words we allowed in the dictionary and for which we created one-hot-encoded features.

As we need to predict 3 different sentiment classes, the last layer has 3 hidden elements. The softmax activation function makes sure the three probabilities sum up to 1.

In the first layer we need to estimate 640064 weights. This is determined by (nb inputs * nb hidden elements) + nb bias terms, or (10000 x 64) + 64 = 640064 In the second layer we estimate (64 x 64) + 64 = 4160 weights In the last layer we estimate (64 x 2) + 2 = 130 weights

In [24]:
```
base_model = models.Sequential()
base_model.add(layers.Dense(64, activation='relu', input_shape=(NB_WORDS,)))
base_model.add(layers.Dense(64, activation='relu'))
base_model.add(layers.Dense(2, activation='softmax'))
base_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                640064

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 2)                 130

=================================================================
Total params: 644,354
Trainable params: 644,354
Non-trainable params: 0
_________________________________________________________________
```

2023-02-09 16:36:40.934374: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance. Because this project is a multi-class, single-label prediction, we use categorical_crossentropy as the loss function and softmax as the final activation function. We fit the model on the remaining train data and validate on the validation set. We run for a predetermined number of epochs and will see when the model starts to overfit.

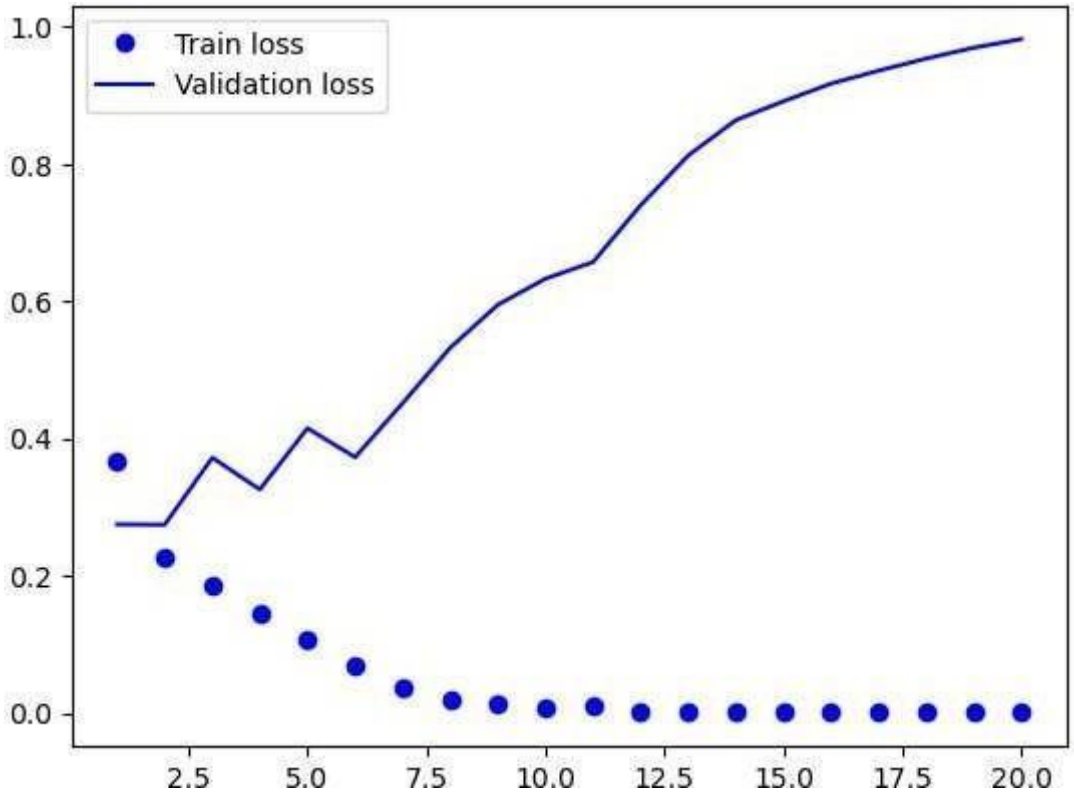
In [26]:
```
def deep_model(model):
    model.compile(optimizer='rmsprop'
                  , loss='categorical_crossentropy'
                  , metrics=['accuracy'])

    history = model.fit(X_train_rest
                        , y_train_rest
                        , epochs=NB_START_EPOCHS
                        , batch_size=BATCH_SIZE
                        , validation_data=(X_valid, y_valid)
                        , verbose=0)

    return history
base_history = deep_model(base_model)
```

2023-02-09 16:36:44.001876: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2) We can see here that the validation loss starts to increase as from epoch 4. The training loss continues to lower, which is normal as the model is trained to fit the train data as good as possible.

In [27]:
```
def eval_metric(history, metric_name):
    metric = history.history[metric_name]
    val_metric = history.history['val_' + metric_name]

    e = range(1, NB_START_EPOCHS + 1)

    plt.plot(e, metric, 'bo', label='Train ' + metric_name)
    plt.plot(e, val_metric, 'b', label='Validation ' + metric_name)
    plt.legend()
    plt.show()
```

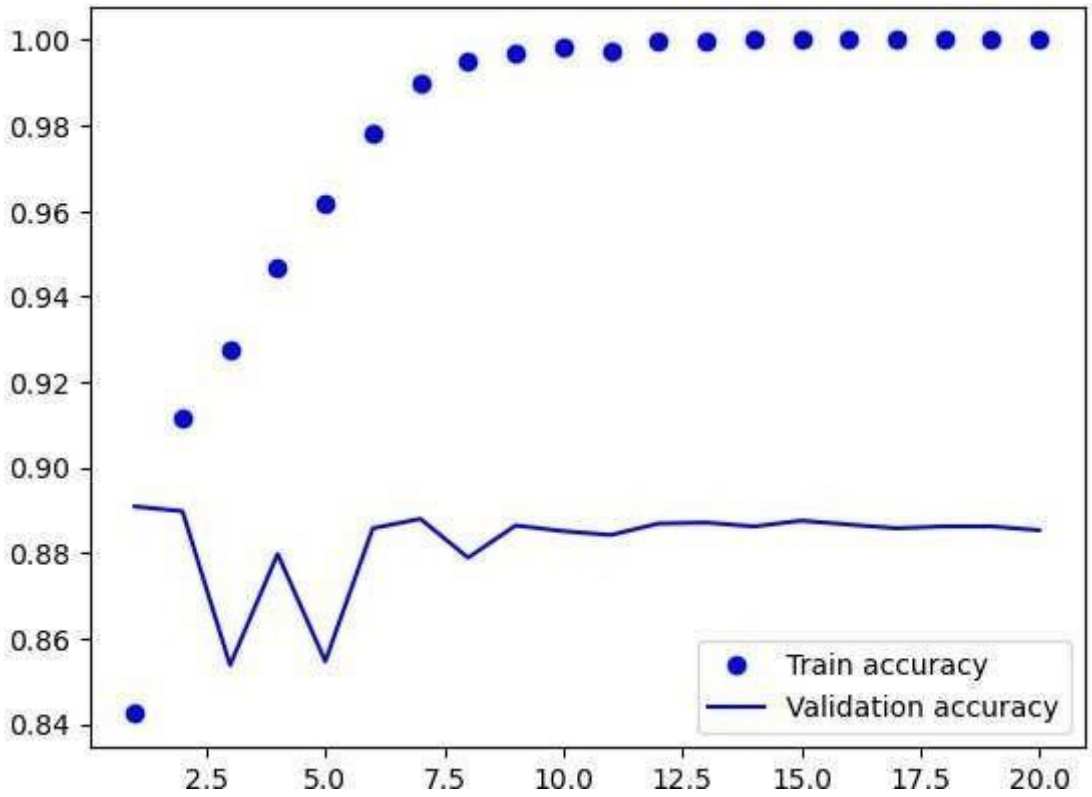
We can see here that the validation loss starts to increase as from epoch 4. The training loss continues to lower, which is normal as the model is trained to fit the train data as good as possible.

In [28]:
```
eval_metric(base_history, 'loss')
```



Just as with the validation loss, the validation accuracy peaks at an early epoch. After that, it goes down slightly. So to conclude, we can say that the model starts overfitting as from epoch 4.

In [29]:
```
eval_metric(base_history, 'accuracy')
```



Handling overfitting Now, we can try to do something about the overfitting. There are different options to do that.

Option 1: reduce the network's size by removing layers or reducing the number of hidden elements in the layers. Option 2: add regularization, which comes down to adding a cost to the loss function for large weights. Option 3: adding dropout layers, which will randomly remove certain features by setting them to zero,

Reducing the network's size We reduce the network's size by removing one layer and lowering the number of hidden elements in the remaining layer to 32.

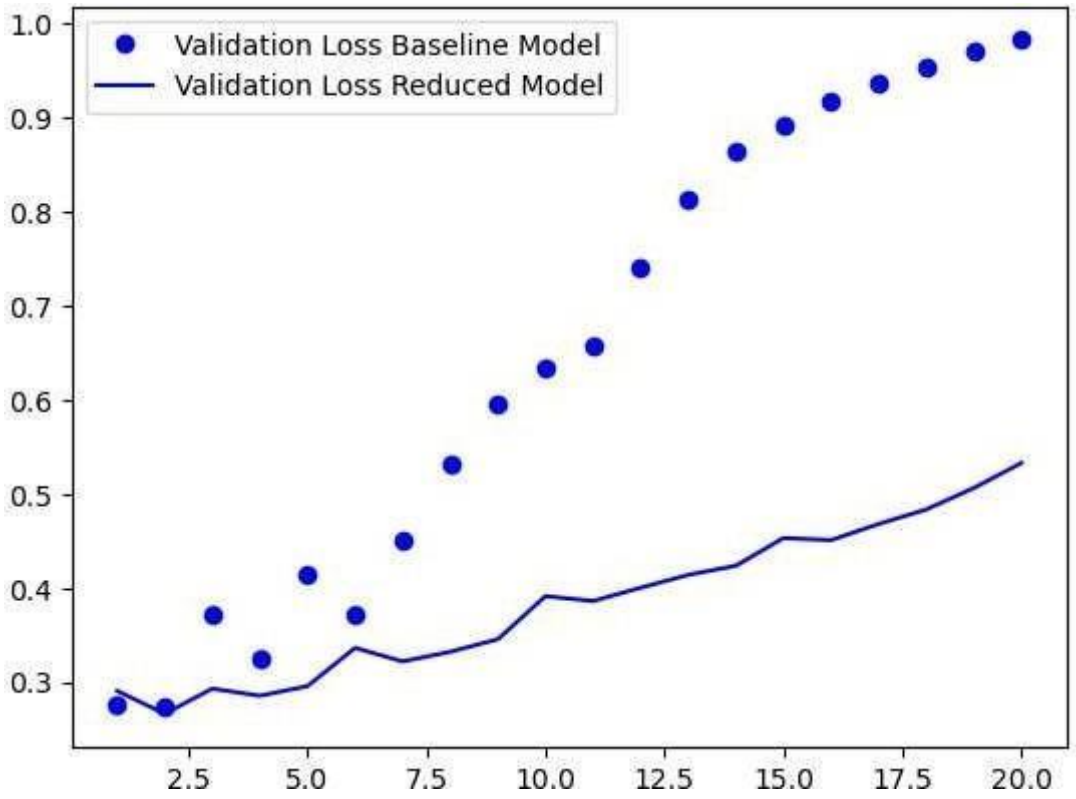
In [30]:
```
reduced_model = models.Sequential()
reduced_model.add(layers.Dense(32, activation='relu', input_shape=(NB_WORDS,)))
reduced_model.add(layers.Dense(2, activation='softmax'))
reduced_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 32)                320032

 dense_4 (Dense)             (None, 2)                 66

=================================================================
Total params: 320,098
Trainable params: 320,098
Non-trainable params: 0
_________________________________________________________________
```

In [31]:
```
reduced_history = deep_model(reduced_model)
def compare_loss_with_baseline(h, model_name):
    loss_base_model = base_history.history['val_loss']
    loss_model = h.history['val_loss']

    e = range(1, NB_START_EPOCHS + 1)

    plt.plot(e, loss_base_model, 'bo', label='Validation Loss Baseline Model')
    plt.plot(e, loss_model, 'b', label='Validation Loss ' + model_name)
    plt.legend()
    plt.show()
```

We can see that it takes more epochs before the reduced model starts overfitting (around epoch 10). Moreover, the loss increases much slower after that epoch compared to the baseline model.

In [32]:
```
compare_loss_with_baseline(reduced_history, 'Reduced Model')
```

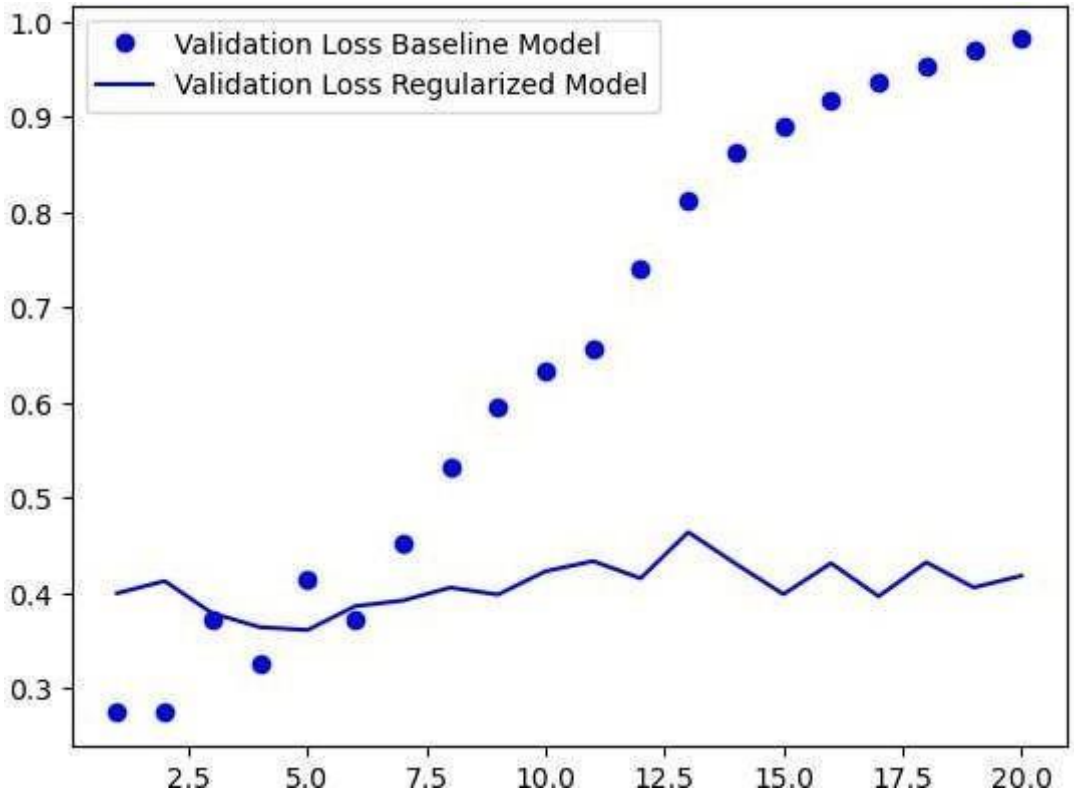


Adding regularization To address overfitting, we can also add regularization to the model. Let's try with L2 regularization.

In [33]:
```
reg_model = models.Sequential()
reg_model.add(layers.Dense(64, kernel_regularizer=regularizers.l2(0.001), activation='
reg_model.add(layers.Dense(64, kernel_regularizer=regularizers.l2(0.001), activation='
reg_model.add(layers.Dense(2, activation='softmax'))
reg_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 64)                640064

 dense_6 (Dense)             (None, 64)                4160

 dense_7 (Dense)             (None, 2)                 130

=================================================================
Total params: 644,354
Trainable params: 644,354
Non-trainable params: 0
_________________________________________________________________
```

In [34]:
```
reg_history = deep_model(reg_model)

#For the regularized model we notice that it starts overfitting earlier than the basel
#the loss increases much slower afterwards.

compare_loss_with_baseline(reg_history, 'Regularized Model')
```



Adding dropout layers The last option we'll try is to add dropout layers.

In [35]:
```
drop_model = models.Sequential()
drop_model.add(layers.Dense(64, activation='relu', input_shape=(NB_WORDS,)))
drop_model.add(layers.Dropout(0.5))
drop_model.add(layers.Dense(64, activation='relu'))
drop_model.add(layers.Dropout(0.5))
drop_model.add(layers.Dense(2, activation='softmax'))
drop_model.summary()
```

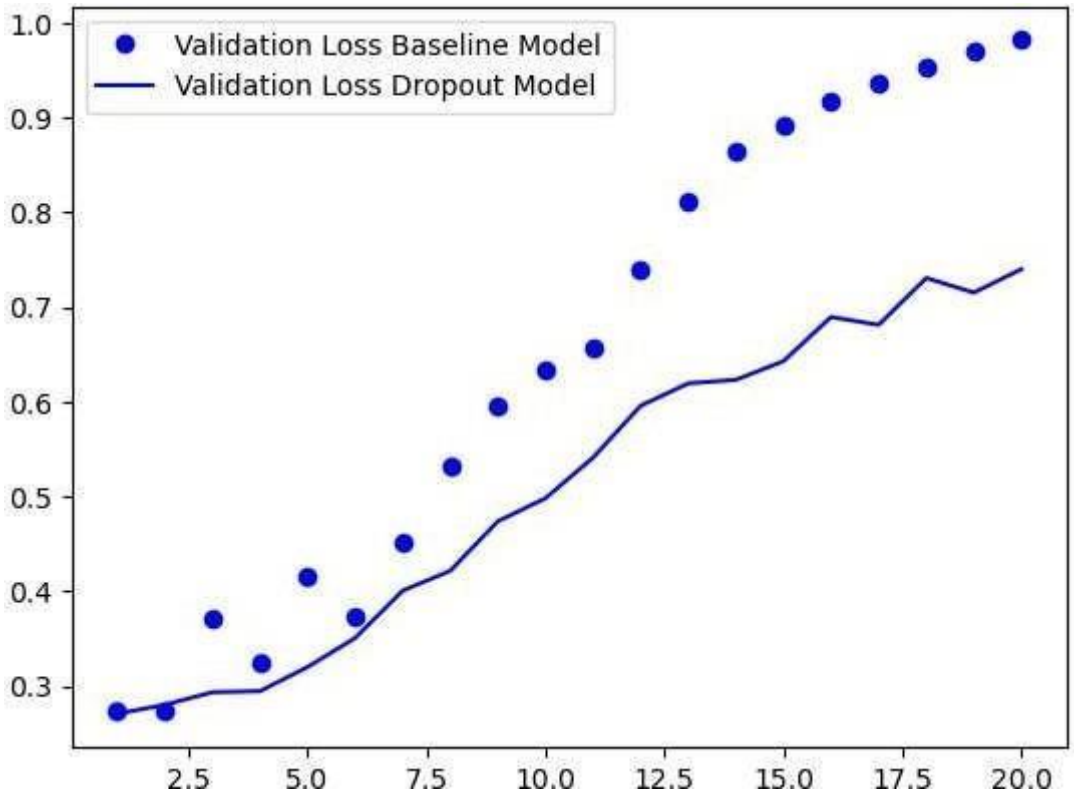
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 64)                640064

 dropout (Dropout)           (None, 64)                0

 dense_9 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_10 (Dense)            (None, 2)                 130

=================================================================
Total params: 644,354
Trainable params: 644,354
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
drop_history = deep_model(drop_model)
```

The model with dropout layers starts overfitting a bit later than the baseline model.

The loss also increases slower than the baseline model.

In [37]:
```
compare_loss_with_baseline(drop_history, 'Dropout Model')
```



Training on the full train data and evaluation on test data At first sight the reduced model seems to be the best model for generalization. But let's check that on the test set.

In [43]:
```
def test_model(model, epoch_stop):
    model.fit(X_train_oh
              , y_train_oh
              , epochs=epoch_stop
              , batch_size=BATCH_SIZE
              , verbose=0)
    results = model.evaluate(X_test_oh, y_test_oh)

    return results
base_results = test_model(base_model, 8)
print('/n')
print('Test accuracy of baseline model: {0:.2f}%'.format(base_results[1]*100))
```

```
157/157 [==============================] - 1s 4ms/step - loss: 1.0987 - accuracy: 0.8
804
/n
Test accuracy of baseline model: 88.04%
```

In [40]:
```
reduced_results = test_model(reduced_model, 10)
print('/n')
print('Test accuracy of reduced model: {0:.2f}%'.format(reduced_results[1]*100))
```

```
157/157 [==============================] - 1s 5ms/step - loss: 0.6837 - accuracy: 0.8
674
/n
Test accuracy of reduced model: 86.74%
```

In [41]:
```
reg_results = test_model(reg_model, 5)
print('/n')
print('Test accuracy of regularized model: {0:.2f}%'.format(reg_results[1]*100))
```

```
157/157 [==============================] - 0s 2ms/step - loss: 0.5452 - accuracy: 0.8
540
/n
Test accuracy of regularized model: 85.40%
```

In [42]:
```
drop_results = test_model(drop_model, 6)
print('/n')
print('Test accuracy of dropout model: {0:.2f}%'.format(drop_results[1]*100))
```

```
157/157 [==============================] - 0s 2ms/step - loss: 0.5962 - accuracy: 0.8
864
/n
Test accuracy of dropout model: 88.64%
```

Conclusion

As we can see above, the model with the dropout layers performs the best on the test data.

In [ ]:
```

```