Name: Tarun Tanmay

Class: MBATech CE

Semester: 6

Roll No: N049

Sap ID: 70471018055

In [59]:

```
# experiment 4
# Regularization
# Regularization is the change in weight. We do it to remove over fitting.
from keras.datasets import imdb
# The dataset has various attributes about the movies. Contains rating of movies.
```

In [60]:

```
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
#loading the dataset and giving it to the testing and training data.
# num_word is the minimum number of occurances of a word in the dataset.
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (
which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes)
is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the
ndarray
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Visi
bleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-
or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated.
If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Visi
bleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-
or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated.
If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

In [61]:

```
len(train_data[0])
# number of occurance of zeroth sample.
# 218 words are there which are the most commonly occuring.
```

Out[61]:

```
218
```

In [62]:

```
train_data[0]
# seeing the zeroth sample, the words are represented as numbers
```

Out[62]:

```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
```

```
3941,
4,
173,
36,
256,
5,
25,
100,
43,
838,
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,
13,
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,
13,
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,
3785,
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,
 1029,
 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,
 103,
 32,
 15,
 16,
 5345,
 19,
 178,
 32]
```

In [63]:

```
len(train_data[10])
```

Out[63]:

```
450
```

In [64]:

```
print(train_data[10])
```

```
[1, 785, 189, 438, 47, 110, 142, 7, 6, 7475, 120, 4, 236, 378, 7, 153, 19, 87, 108, 141,
17, 1004, 5, 2, 883, 2, 23, 8, 4, 136, 2, 2, 4, 7475, 43, 1076, 21, 1407, 419, 5, 5202, 12
0, 91, 682, 189, 2818, 5, 9, 1348, 31, 7, 4, 118, 785, 189, 108, 126, 93, 2, 16, 540, 324
, 23, 6, 364, 352, 21, 14, 9, 93, 56, 18, 11, 230, 53, 771, 74, 31, 34, 4, 2834, 7, 4, 22,
5, 14, 11, 471, 9, 2, 34, 4, 321, 487, 5, 116, 15, 6584, 4, 22, 9, 6, 2286, 4, 114, 2679,
23, 107, 293, 1008, 1172, 5, 328, 1236, 4, 1375, 109, 9, 6, 132, 773, 2, 1412, 8, 1172, 1
8, 7865, 29, 9, 276, 11, 6, 2768, 19, 289, 409, 4, 5341, 2140, 2, 648, 1430, 2, 8914, 5,
27, 3000, 1432, 7130, 103, 6, 346, 137, 11, 4, 2768, 295, 36, 7740, 725, 6, 3208, 273, 11
, 4, 1513, 15, 1367, 35, 154, 2, 103, 2, 173, 7, 12, 36, 515, 3547, 94, 2547, 1722, 5, 35
47, 36, 203, 30, 502, 8, 361, 12, 8, 989, 143, 4, 1172, 3404, 10, 10, 328, 1236, 9, 6, 55
, 221, 2989, 5, 146, 165, 179, 770, 15, 50, 713, 53, 108, 448, 23, 12, 17, 225, 38, 76, 4
397, 18, 183, 8, 81, 19, 12, 45, 1257, 8, 135, 15, 2, 166, 4, 118, 7, 45, 2, 17, 466, 45,
2, 4, 22, 115, 165, 764, 6075, 5, 1030, 8, 2973, 73, 469, 167, 2127, 2, 1568, 6, 87, 841,
18, 4, 22, 4, 192, 15, 91, 7, 12, 304, 273, 1004, 4, 1375, 1172, 2768, 2, 15, 4, 22, 764,
55, 5773, 5, 14, 4233, 7444, 4, 1375, 326, 7, 4, 4760, 1786, 8, 361, 1236, 8, 989, 46, 7,
4, 2768, 45, 55, 776, 8, 79, 496, 98, 45, 400, 301, 15, 4, 1859, 9, 4, 155, 15, 66, 2, 84
, 5, 14, 22, 1534, 15, 17, 4, 167, 2, 15, 75, 70, 115, 66, 30, 252, 7, 618, 51, 9, 2161,
4, 3130, 5, 14, 1525, 8, 6584, 15, 2, 165, 127, 1921, 8, 30, 179, 2532, 4, 22, 9, 906, 18
, 6, 176, 7, 1007, 1005, 4, 1375, 114, 4, 105, 26, 32, 55, 221, 11, 68, 205, 96, 5, 4, 19
2, 15, 4, 274, 410, 220, 304, 23, 94, 205, 109, 9, 55, 73, 224, 259, 3786, 15, 4, 22, 528
, 1645, 34, 4, 130, 528, 30, 685, 345, 17, 4, 277, 199, 166, 281, 5, 1030, 8, 30, 179, 44
42, 444, 2, 9, 6, 371, 87, 189, 22, 5, 31, 7, 4, 118, 7, 4, 2068, 545, 1178, 829]
```

In [65]:

```
train_labels[10]
```

Out[65]:

```
1
```

In [66]:

```
# every entry in the dataset has different number of attributes in the row.
# to solve this we need to prepare our dataset.
# to solve this we make those attributes as 1 which are used and rest all are made zero.
import numpy as np
```

In [67]:

```
# Preparing the dataset
 def vect_seq(sequence, dimension=10000):
  mat = np.zeros((len(sequence), dimension)) # creating a numpy array with all zeros of
shape len(sequence), 10000
  for i, s in enumerate(sequence):
    mat[i, s] = 1 # for every occurance of a word we make the zero to 1
  return mat
```

Converting our data to the desired form using the vect_seq function, thereafter checking thier shape.

In [68]:

```
x_train = vect_seq(train_data)
x_train.shape
```

Out[68]:

```
(25000, 10000)
```

In [69]:

```
x_test = vect_seq(test_data)
x_test.shape
```

Out[69]:

```
(25000, 10000)
```

Converting the labels to numpy array.

In [70]:

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

In [71]:

```
from keras import models
from keras import layers
```

Creating a model with 4 layers. Input layer with 10000 neurons. Followed by 2 layers with 16 neurons. And an output layer with 1 neuron and sigmoid activation function.

In [72]:

```
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'))
```

In [73]:

```
from keras import losses
from keras import metrics
from keras import optimizers
```

In [74]:

```
model.compile(optimizer=optimizers.RMSprop(lr=0.01), loss = losses.binary_crossentropy,
metrics=metrics.binary_accuracy)
# compiling the model using RMSprop optimizer, binary_crossentropy as loss function, and
binary_accuracy as metrics
```

In [75]:

```
# saving first 10000 rows of trainig dataset for validation
x_val = x_train[:10000]
new_x_train = x_train[10000:]
y_val = y_train[:10000]
new_y_train = y_train[10000:]
```

In [76]:

```
# checing the shape of training and validation samples
print(x_val.shape)
print(new_x_train.shape)
```

```
(10000, 10000)
(15000, 10000)
```

In [77]:

```
# the compiled model is trained and validated using training and validation data
history = model.fit(new_x_train, new_y_train, batch_size=512, epochs=20, validation_data
=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 2s 39ms/step - loss: 0.6325 - binary_accuracy: 0
.6589 - val_loss: 0.4529 - val_binary_accuracy: 0.8019
Epoch 2/20
30/30 [==============================] - 1s 22ms/step - loss: 0.2850 - binary_accuracy: 0
.8863 - val_loss: 0.2862 - val_binary_accuracy: 0.8843
Epoch 3/20
30/30 [==============================] - 1s 21ms/step - loss: 0.1913 - binary_accuracy: 0
.9267 - val_loss: 0.2821 - val_binary_accuracy: 0.8904
Epoch 4/20
30/30 [==============================] - 1s 21ms/step - loss: 0.1661 - binary_accuracy: 0
.9399 - val_loss: 0.2957 - val_binary_accuracy: 0.8859
Epoch 5/20
30/30 [==============================] - 1s 21ms/step - loss: 0.1072 - binary_accuracy: 0
```

```
30/30 [==============================] - 1s 21ms/step - loss: 0.1072 - binary_accuracy: 0
.9587 - val_loss: 0.4847 - val_binary_accuracy: 0.8613
Epoch 6/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0895 - binary_accuracy: 0
.9665 - val_loss: 0.4235 - val_binary_accuracy: 0.8799
Epoch 7/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0750 - binary_accuracy: 0
.9779 - val_loss: 0.5109 - val_binary_accuracy: 0.8765
Epoch 8/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0338 - binary_accuracy: 0
.9894 - val_loss: 0.5046 - val_binary_accuracy: 0.8760
Epoch 9/20
30/30 [==============================] - 1s 20ms/step - loss: 0.0131 - binary_accuracy: 0
.9966 - val_loss: 0.7559 - val_binary_accuracy: 0.8715
Epoch 10/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0808 - binary_accuracy: 0
.9867 - val_loss: 0.6861 - val_binary_accuracy: 0.8734
Epoch 11/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0061 - binary_accuracy: 0
.9979 - val_loss: 0.8191 - val_binary_accuracy: 0.8738
Epoch 12/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0039 - binary_accuracy: 0
.9981 - val_loss: 1.0008 - val_binary_accuracy: 0.8708
Epoch 13/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0493 - binary_accuracy: 0
.9927 - val_loss: 0.8059 - val_binary_accuracy: 0.8674
Epoch 14/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0047 - binary_accuracy: 0
.9984 - val_loss: 1.0309 - val_binary_accuracy: 0.8702
Epoch 15/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0018 - binary_accuracy: 0
.9990 - val_loss: 1.2379 - val_binary_accuracy: 0.8701
Epoch 16/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0261 - binary_accuracy: 0
.9978 - val_loss: 1.0025 - val_binary_accuracy: 0.8657
Epoch 17/20
30/30 [==============================] - 1s 23ms/step - loss: 0.0019 - binary_accuracy: 0
.9998 - val_loss: 1.1431 - val_binary_accuracy: 0.8671
Epoch 18/20
30/30 [==============================] - 1s 21ms/step - loss: 7.5881e-04 - binary_accurac
y: 0.9999 - val_loss: 1.2668 - val_binary_accuracy: 0.8684
Epoch 19/20
30/30 [==============================] - 1s 21ms/step - loss: 6.0423e-04 - binary_accurac
y: 1.0000 - val_loss: 1.4062 - val_binary_accuracy: 0.8686
Epoch 20/20
30/30 [==============================] - 1s 21ms/step - loss: 3.3831e-04 - binary_accurac
y: 1.0000 - val_loss: 1.6091 - val_binary_accuracy: 0.8669
```

In [78]:

```python
# creating a dictionary of the result parameters of the trained model
history_dict = history.history
history_dict.keys()
```

Out[78]:

```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

In [79]:

```python
import matplotlib.pyplot as plt
# calling matplotlib to visualize the results
```

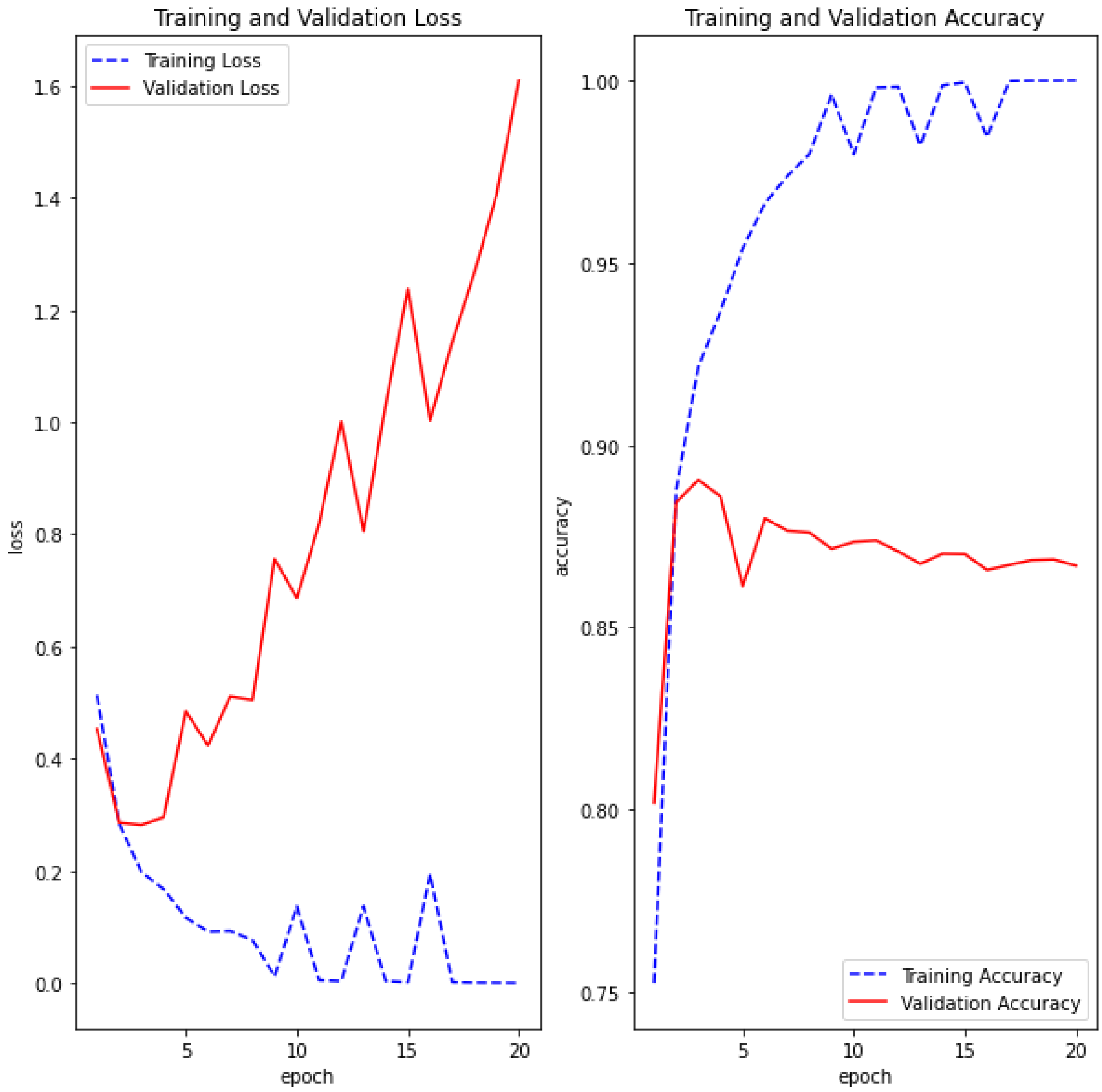
In [80]:

```python
# visualising the loss and accuracy of the training and valication phase
loss = history_dict['loss']
val_loss = history_dict['val_loss']
acc = history_dict['binary_accuracy']
val_acc = history_dict['val_binary_accuracy']
epoch = range(1, len(acc)+1)
fig , ax = plt.subplots(1,2)
```

```
fig.set_size_inches(10,10)

ax[0].plot(epoch, loss, 'b--', label='Training Loss')
ax[0].plot(epoch, val_loss, 'r', label='Validation Loss')
ax[0].set_title('Training and Validation Loss')
ax[0].legend()
ax[0].set_xlabel('epoch')
ax[0].set_ylabel('loss')

ax[1].plot(epoch, acc, 'b--', label='Training Accuracy')
ax[1].plot(epoch, val_acc, 'r', label='Validation Accuracy')
ax[1].set_title('Training and Validation Accuracy')
ax[1].legend()
ax[1].set_xlabel('epoch')
ax[1].set_ylabel('accuracy')
plt.show()
```



From the above plots we can say that training process is good, but the validation gives us poor results. Thus we can say that the model has over fit.

In [81]:

```
from keras import regularizers
```

In [97]:

```
# adding regularizers to the model to prevent over fitting
model_1 = models.Sequential()
model_1.add(layers.Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.001
), input_shape=(10000,)))
model_1.add(layers.Dense(16, activation='relu'))
model_1.add(layers.Dense(1, activation='sigmoid'))
```

In [98]:

```
model_1.compile(optimizer=optimizers.RMSprop(lr=0.01), loss = losses.binary_crossentropy
, metrics=metrics.binary_accuracy)
# compiling the updated model using RMSprop optimizer, binary_crossentropy as loss functi
on, and binary_accuracy as metrics
```

In [99]:

```
# the updated compiled model is trained and validated using training and validation data
history_1 = model_1.fit(new_x_train, new_y_train, batch_size=512, epochs=20, validation_
data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 2s 32ms/step - loss: 0.7932 - binary_accuracy: 0
.6348 - val_loss: 0.4429 - val_binary_accuracy: 0.8474
Epoch 2/20
30/30 [==============================] - 1s 21ms/step - loss: 0.4697 - binary_accuracy: 0
.8276 - val_loss: 0.5341 - val_binary_accuracy: 0.7874
Epoch 3/20
30/30 [==============================] - 1s 22ms/step - loss: 0.4164 - binary_accuracy: 0
.8659 - val_loss: 0.3968 - val_binary_accuracy: 0.8841
Epoch 4/20
30/30 [==============================] - 1s 22ms/step - loss: 0.3777 - binary_accuracy: 0
.8912 - val_loss: 0.4399 - val_binary_accuracy: 0.8573
Epoch 5/20
30/30 [==============================] - 1s 23ms/step - loss: 0.3959 - binary_accuracy: 0
.8717 - val_loss: 0.3999 - val_binary_accuracy: 0.8734
Epoch 6/20
30/30 [==============================] - 1s 23ms/step - loss: 0.3658 - binary_accuracy: 0
.8921 - val_loss: 0.4312 - val_binary_accuracy: 0.8565
Epoch 7/20
30/30 [==============================] - 1s 23ms/step - loss: 0.3390 - binary_accuracy: 0
.8985 - val_loss: 0.3806 - val_binary_accuracy: 0.8829
Epoch 8/20
30/30 [==============================] - 1s 23ms/step - loss: 0.3300 - binary_accuracy: 0
.9048 - val_loss: 0.3833 - val_binary_accuracy: 0.8809
Epoch 9/20
30/30 [==============================] - 1s 21ms/step - loss: 0.3340 - binary_accuracy: 0
.9133 - val_loss: 0.3780 - val_binary_accuracy: 0.8824
Epoch 10/20
30/30 [==============================] - 1s 21ms/step - loss: 0.3219 - binary_accuracy: 0
.9141 - val_loss: 0.4278 - val_binary_accuracy: 0.8576
Epoch 11/20
30/30 [==============================] - 1s 22ms/step - loss: 0.3010 - binary_accuracy: 0
.9229 - val_loss: 0.3885 - val_binary_accuracy: 0.8815
Epoch 12/20
30/30 [==============================] - 1s 22ms/step - loss: 0.2966 - binary_accuracy: 0
.9279 - val_loss: 0.4646 - val_binary_accuracy: 0.8613
Epoch 13/20
30/30 [==============================] - 1s 32ms/step - loss: 0.3278 - binary_accuracy: 0
.9040 - val_loss: 0.6949 - val_binary_accuracy: 0.8094
Epoch 14/20
30/30 [==============================] - 1s 22ms/step - loss: 0.3581 - binary_accuracy: 0
.8930 - val_loss: 0.3987 - val_binary_accuracy: 0.8803
Epoch 15/20
30/30 [==============================] - 1s 22ms/step - loss: 0.2888 - binary_accuracy: 0
.9303 - val_loss: 0.4904 - val_binary_accuracy: 0.8505
Epoch 16/20
30/30 [==============================] - 1s 21ms/step - loss: 0.2991 - binary_accuracy: 0
.9230 - val_loss: 0.4683 - val_binary_accuracy: 0.8713
Epoch 17/20
30/30 [==============================] - 1s 23ms/step - loss: 0.3205 - binary_accuracy: 0
.9111 - val_loss: 0.4884 - val_binary_accuracy: 0.8265
Epoch 18/20
30/30 [==============================] - 1s 23ms/step - loss: 0.3046 - binary_accuracy: 0
.9138 - val_loss: 0.8273 - val_binary_accuracy: 0.7630
Epoch 19/20
30/30 [==============================] - 1s 22ms/step - loss: 0.3291 - binary_accuracy: 0
.9166 - val_loss: 0.5096 - val_binary_accuracy: 0.8472
Epoch 20/20
30/30 [==============================] - 1s 23ms/step - loss: 0.2911 - binary_accuracy: 0
.9265 - val_loss: 0.4467 - val_binary_accuracy: 0.8689
```

```
[illegible] - val_loss: [illegible] - val_binary_accuracy: [illegible]
```

In [100]:

```python
# creating a dictionary of the result parameters of the updated trained model
history_dict_1 = history_1.history
history_dict_1.keys()
```

Out[100]:

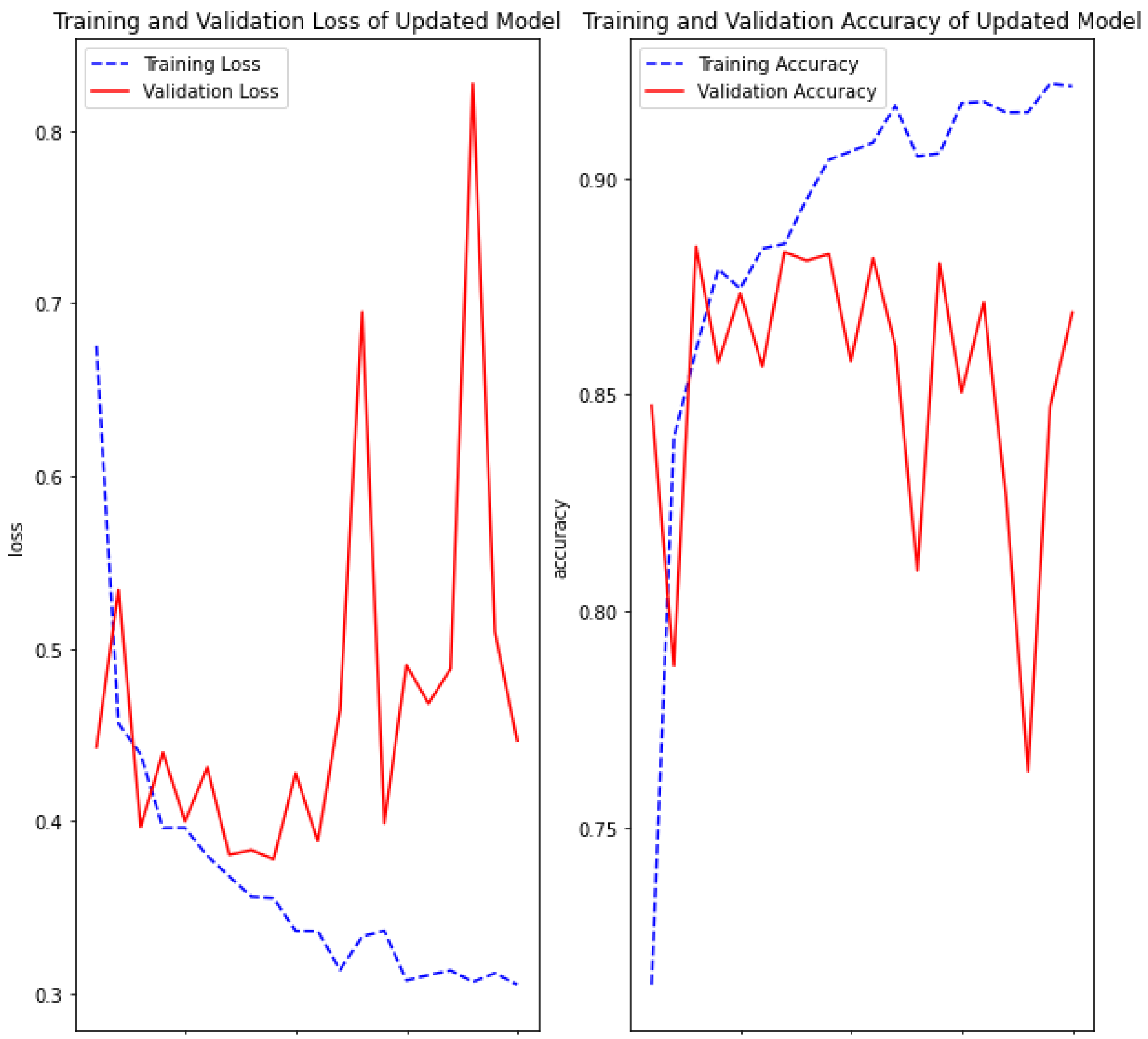
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

In [101]:

```python
# visualising the loss and accuracy of the training and valication phase of thr updated m
odel
loss_1 = history_dict_1['loss']
val_loss_1 = history_dict_1['val_loss']
acc_1 = history_dict_1['binary_accuracy']
val_acc_1 = history_dict_1['val_binary_accuracy']
epoch_1 = range(1, len(acc)+1)
fig , ax = plt.subplots(1,2)
fig.set_size_inches(10,10)

ax[0].plot(epoch_1, loss_1, 'b--', label='Training Loss')
ax[0].plot(epoch_1, val_loss_1, 'r', label='Validation Loss')
ax[0].set_title('Training and Validation Loss of Updated Model')
ax[0].legend()
ax[0].set_xlabel('epoch')
ax[0].set_ylabel('loss')

ax[1].plot(epoch_1, acc_1, 'b--', label='Training Accuracy')
ax[1].plot(epoch_1, val_acc_1, 'r', label='Validation Accuracy')
ax[1].set_title('Training and Validation Accuracy of Updated Model')
ax[1].legend()
ax[1].set_xlabel('epoch')
ax[1].set_ylabel('accuracy')
plt.show()
```



Conclusion

- **With each epoch training loss reduces but validation loss increases, as shown by the graph. Showing that the model has overfit.**
- **To avoid overfiting, the regularizer with L2 norm was used, and results show that the training loss reduces with each epoch and validation loss does not increase with each epoch, it gets stabalized, within a small range.**
- **For the given dataset validation loss varies from the range 0.3 to 0.5 with regularizer and 0.3 to 1.7 without regularizer.**