Dropout Iteration

Importing IMDB Dataset

```
import keras
```

```
from keras.datasets import imdb

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

```
train_data[0]
```

```
train_labels[0]
```

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

```
# 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]])
```

```
decoded_review
```

Preparing the Data

```
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]
```

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

Building the Network

```
from keras import models
from keras import layers

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'))
```

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

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

Validating the Apporach

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

```
partial_x_train = x_train[10000:]

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


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

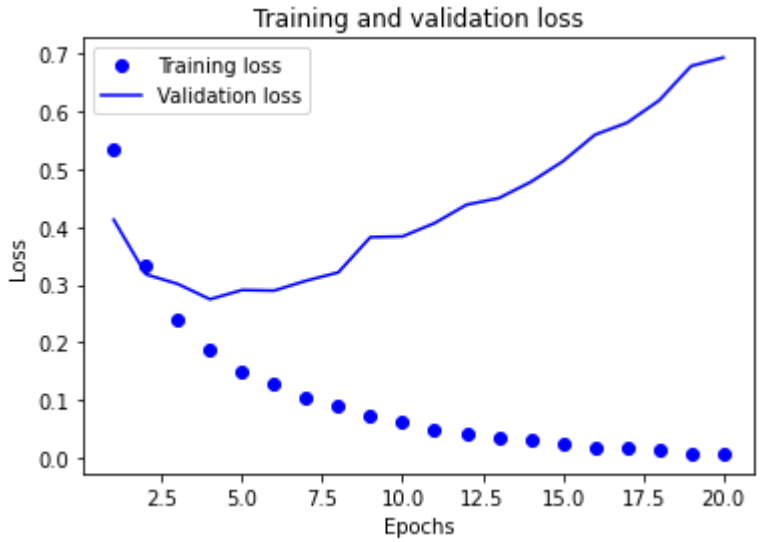
```
Epoch 1/20
30/30 [==============================] - 2s 54ms/step - loss: 0.6032 - binary_accuracy:
Epoch 2/20
30/30 [==============================] - 1s 34ms/step - loss: 0.3501 - binary_accuracy:
Epoch 3/20
30/30 [==============================] - 1s 35ms/step - loss: 0.2461 - binary_accuracy:
Epoch 4/20
30/30 [==============================] - 1s 34ms/step - loss: 0.1914 - binary_accuracy:
Epoch 5/20
30/30 [==============================] - 1s 43ms/step - loss: 0.1497 - binary_accuracy:
Epoch 6/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1260 - binary_accuracy:
Epoch 7/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1046 - binary_accuracy:
Epoch 8/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0866 - binary_accuracy:
Epoch 9/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0695 - binary_accuracy:
Epoch 10/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0583 - binary_accuracy:
Epoch 11/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0465 - binary_accuracy:
Epoch 12/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0388 - binary_accuracy:
Epoch 13/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0342 - binary_accuracy:
Epoch 14/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0258 - binary_accuracy:
Epoch 15/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0200 - binary_accuracy:
Epoch 16/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0169 - binary_accuracy:
Epoch 17/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0142 - binary_accuracy:
Epoch 18/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0092 - binary_accuracy:
Epoch 19/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0068 - binary_accuracy:
Epoch 20/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0063 - binary_accuracy:
```

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

```
history_dict.keys()
```

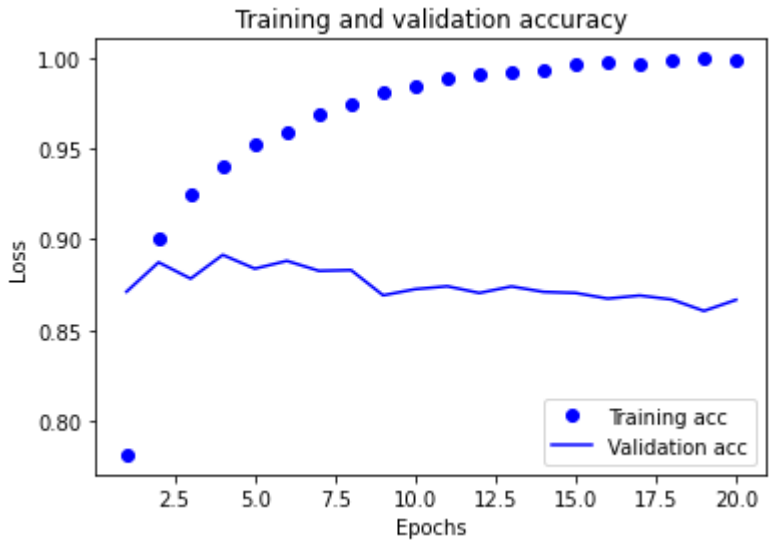
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

```
import matplotlib.pyplot as plt

binary_accuracy = history.history['binary_accuracy']
val_binary_accuracy = history.history['val_binary_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```




```
plt.clf()   # clear figure
binary_accuracy_values = history_dict['binary_accuracy']
val_binary_accuracy_values = history_dict['val_binary_accuracy']

plt.plot(epochs, binary_accuracy, 'bo', label='Training acc')
plt.plot(epochs, val_binary_accuracy, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
plt.show()
```




```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

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

model.fit(x_val, y_val, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
20/20 [==============================] - 1s 27ms/step - loss: 0.6697 - accuracy: 0.5879
Epoch 2/4
20/20 [==============================] - 1s 27ms/step - loss: 0.5493 - accuracy: 0.7484
Epoch 3/4
20/20 [==============================] - 1s 27ms/step - loss: 0.4661 - accuracy: 0.8057
Epoch 4/4
20/20 [==============================] - 1s 26ms/step - loss: 0.4001 - accuracy: 0.8474
782/782 [==============================] - 1s 2ms/step - loss: 0.3475 - accuracy: 0.8756
```

```
results
```

```
[0.34754160046577454, 0.8755999803543091]
```

Predictions on New Data

+ Code + Text

```
model.predict(x_test)
```

```
array([[0.36082226],
       [0.9654    ],
       [0.7100765 ],
       ...,
       [0.22045922],
       [0.22716764],
       [0.47536862]], dtype=float32)
```