This is a companion notebook for the book [Deep Learning with Python, Second Edition]. For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

Getting started with neural networks: Classification and regression

Classifying movie reviews: A binary classification example

The IMDB dataset

Double-click (or enter) to edit

Double-click (or enter) to edit

Loading the IMDB dataset

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

```
train_data[0]
```

```
      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]
```

```
train_labels[0]
```

```
1
```

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

```
9999
```

Decoding reviews back to text

```
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

Preparing the data

```
import numpy as np
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)
```

Encoding the integer sequences via multi-hot encoding

```
x_train[0]
```

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

Vectorizing the model

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

1) Building network with three layers

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation="relu", input_shape=(10000,)),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
```

```
    layers.Dense(1, activation="sigmoid")
```

Compiling the model

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

Validating your approach

Setting aside a validation set

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training your model

```
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 [==============================] - 3s 59ms/step - loss: 0.5999 - accuracy: 0.7043
Epoch 2/20
30/30 [==============================] - 1s 48ms/step - loss: 0.3767 - accuracy: 0.8840
Epoch 3/20
30/30 [==============================] - 1s 44ms/step - loss: 0.2505 - accuracy: 0.9220
Epoch 4/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1827 - accuracy: 0.9448
Epoch 5/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1468 - accuracy: 0.9527
Epoch 6/20
30/30 [==============================] - 1s 43ms/step - loss: 0.1177 - accuracy: 0.9642
Epoch 7/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0964 - accuracy: 0.9707
Epoch 8/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0803 - accuracy: 0.9767
Epoch 9/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0626 - accuracy: 0.9831
Epoch 10/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0498 - accuracy: 0.9876
Epoch 11/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0400 - accuracy: 0.9903
Epoch 12/20
```

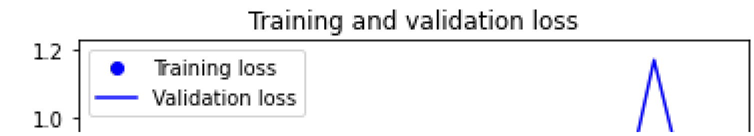
```
30/30 [==============================] - 1s 44ms/step - loss: 0.0367 - accuracy: 0.9917
Epoch 13/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0268 - accuracy: 0.9940
Epoch 14/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0170 - accuracy: 0.9978
Epoch 15/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0200 - accuracy: 0.9945
Epoch 16/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0201 - accuracy: 0.9943
Epoch 17/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0067 - accuracy: 0.9991
Epoch 18/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0150 - accuracy: 0.9959
Epoch 19/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0039 - accuracy: 0.9995
Epoch 20/20
30/30 [==============================] - 2s 75ms/step - loss: 0.0116 - accuracy: 0.9969
```

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

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

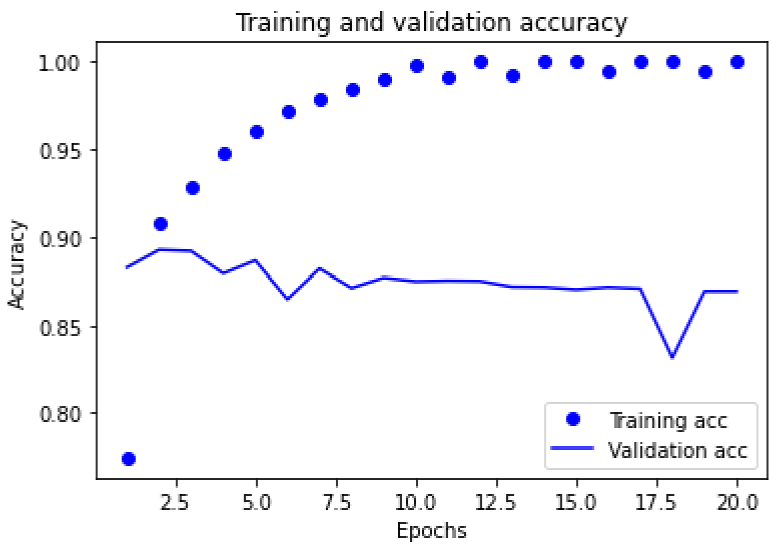
Plotting the training and validation loss

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the training and validation accuracy

```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Conclusion - From the above Matblot graphs we can see that training loss decreases with every epoch and the training accuracy increases with every epoch. Whereas, our validation loss has minimum value at 5 epoch and validation accuracy has peak at 5 Epoch. So, Ideally we should stop after 5 epoch.

2) Training a new network layers with more hidden units: 32,64 etc

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
```

```
              metrics=["accuracy"])
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 [==============================] - 4s 117ms/step - loss: 0.4745 - accuracy: 0.797
Epoch 2/20
30/30 [==============================] - 1s 49ms/step - loss: 0.2872 - accuracy: 0.9061
Epoch 3/20
30/30 [==============================] - 1s 50ms/step - loss: 0.2185 - accuracy: 0.9287
Epoch 4/20
30/30 [==============================] - 1s 49ms/step - loss: 0.1795 - accuracy: 0.9429
Epoch 5/20
30/30 [==============================] - 1s 50ms/step - loss: 0.1482 - accuracy: 0.9553
Epoch 6/20
30/30 [==============================] - 1s 48ms/step - loss: 0.1276 - accuracy: 0.9628
Epoch 7/20
30/30 [==============================] - 1s 48ms/step - loss: 0.1084 - accuracy: 0.9685
Epoch 8/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0941 - accuracy: 0.9727
Epoch 9/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0815 - accuracy: 0.9779
Epoch 10/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0689 - accuracy: 0.9839
Epoch 11/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0594 - accuracy: 0.9865
Epoch 12/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0540 - accuracy: 0.9877
Epoch 13/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0428 - accuracy: 0.9917
Epoch 14/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0380 - accuracy: 0.9931
Epoch 15/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0332 - accuracy: 0.9944
Epoch 16/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0270 - accuracy: 0.9966
Epoch 17/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0242 - accuracy: 0.9968
Epoch 18/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0183 - accuracy: 0.9983
Epoch 19/20
30/30 [==============================] - 1s 47ms/step - loss: 0.0170 - accuracy: 0.9984
Epoch 20/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0140 - accuracy: 0.9987
```

Generate Predictions

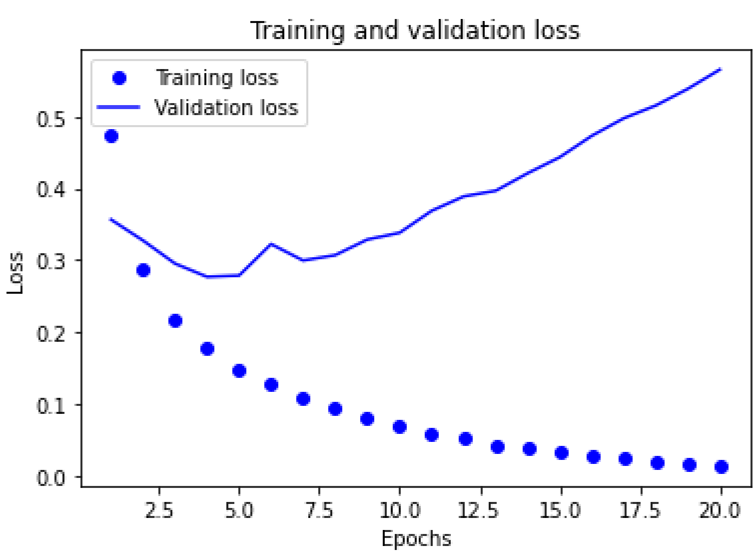
```
model.predict(x_test)
```

```
array([[0.01173723],
       [0.99999946],
       [0.68324226],
       ...,
       [0.00676459],
       [0.02534658],
       [0.7311161 ]], dtype=float32)
```

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

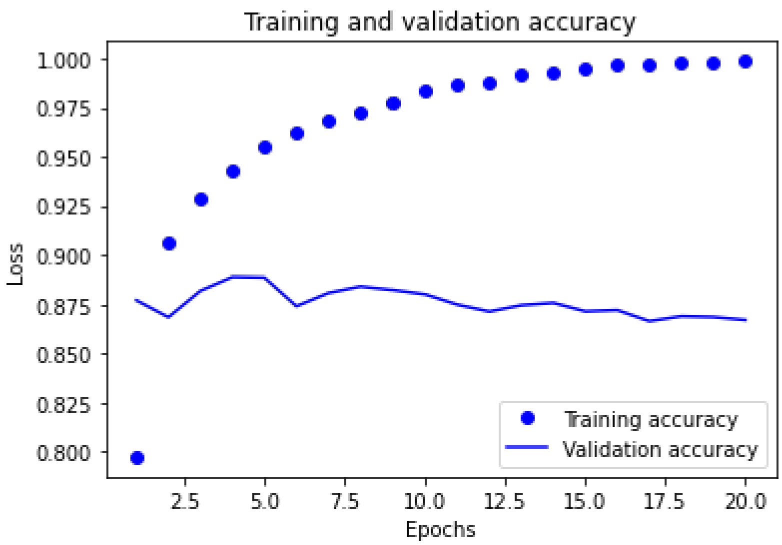
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
import matplotlib.pyplot as plt

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(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
```

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

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

plt.show()
```



Conclusion - From the above Matblot graphs we can see that training loss decreases with every epoch and the training accuracy increases with every epoch. Whereas, our validation loss has minimum value at 2 epochs and validation accuracy has a peak at 2 Epoch. So, Ideally we should stop after 2 epochs.

Part 3) and 4) Train a new Network using the mse loss function instead of binary_crossentropy and tanh activation

```
model = keras.Sequential([
    layers.Dense(16, activation="tanh", input_shape=(10000,)),
    layers.Dense(1, activation="sigmoid")])
model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
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 52ms/step - loss: 0.1612 - accuracy: 0.7959
Epoch 2/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0936 - accuracy: 0.9046
Epoch 3/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0696 - accuracy: 0.9295
Epoch 4/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0550 - accuracy: 0.9456
Epoch 5/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0456 - accuracy: 0.9550
Epoch 6/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0382 - accuracy: 0.9642
Epoch 7/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0319 - accuracy: 0.9710
Epoch 8/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0271 - accuracy: 0.9765
Epoch 9/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0234 - accuracy: 0.9800
Epoch 10/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0199 - accuracy: 0.9839
Epoch 11/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0171 - accuracy: 0.9868
Epoch 12/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0147 - accuracy: 0.9886
Epoch 13/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0126 - accuracy: 0.9913
Epoch 14/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0106 - accuracy: 0.9932
Epoch 15/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0091 - accuracy: 0.9939
Epoch 16/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0080 - accuracy: 0.9943
Epoch 17/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0064 - accuracy: 0.9958
Epoch 18/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0061 - accuracy: 0.9959
Epoch 19/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0045 - accuracy: 0.9968
Epoch 20/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0050 - accuracy: 0.9961
```

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

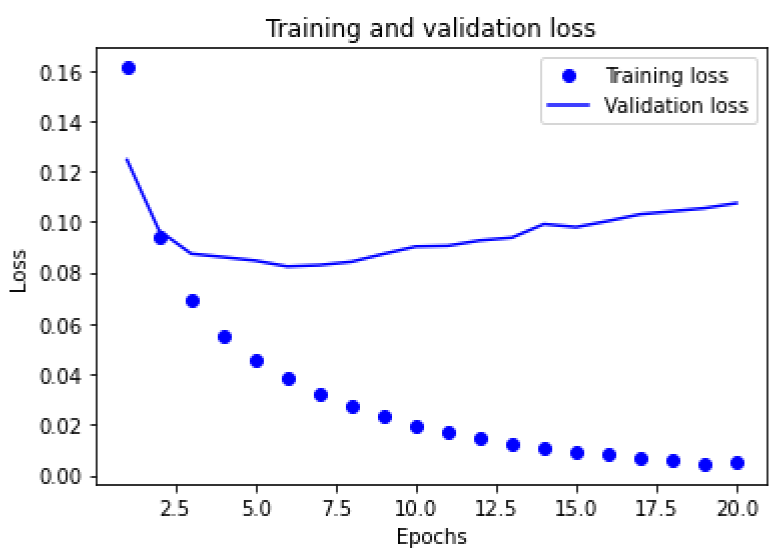
```
array([[0.0230056 ],
       [0.99998844],
       [0.8306817 ],
       ...,
       [0.02304727],
       [0.09159389],
       [0.8514501 ]], dtype=float32)
```

```
history_dict = history.history
```

```
history_dict.keys()
```

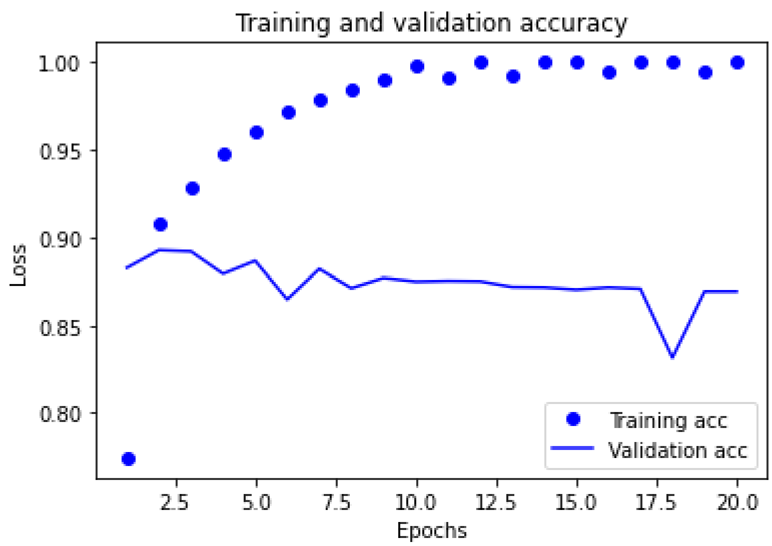
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
import matplotlib.pyplot as plt

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 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
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

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

```
plt.show()
```



Conclusion - From the above Matblot graphs we can see that training loss decreases with every epoch and the training accuracy increases with every epoch. Whereas, our validation loss has minimum value at 4 epochs and validation accuracy has a peak at 24 Epoch. So, Ideally we should stop after 4 epochs

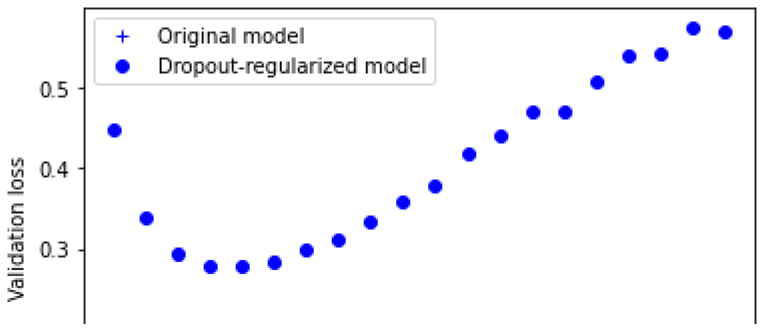
Applying Dropout is one of the most effective and most commonly used techniques for neural networks. We have added 50% dropout rate for the first two layers

```
dpt_model = keras.Sequential([
    layers.Dense(16, activation="relu", input_shape=(10000,)),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])

dpt_model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['acc'])


dpt_model_hist = dpt_model.fit(x_train, y_train,
                               epochs=20,
                               batch_size=512,
                               validation_data=(x_test, y_test))
```

```
Epoch 1/20
49/49 [==============================] - 5s 80ms/step - loss: 0.5875 - acc: 0.6951 - va
Epoch 2/20
49/49 [==============================] - 2s 45ms/step - loss: 0.4401 - acc: 0.8130 - va
```

```
Epoch 3/20
49/49 [==============================] - 2s 47ms/step - loss: 0.3558 - acc: 0.8631 - va
Epoch 4/20
49/49 [==============================] - 2s 46ms/step - loss: 0.3032 - acc: 0.8916 - va
Epoch 5/20
49/49 [==============================] - 2s 47ms/step - loss: 0.2594 - acc: 0.9093 - va
Epoch 6/20
49/49 [==============================] - 2s 45ms/step - loss: 0.2335 - acc: 0.9186 - va
Epoch 7/20
49/49 [==============================] - 2s 46ms/step - loss: 0.2059 - acc: 0.9294 - va
Epoch 8/20
49/49 [==============================] - 2s 46ms/step - loss: 0.1886 - acc: 0.9362 - va
Epoch 9/20
49/49 [==============================] - 2s 48ms/step - loss: 0.1722 - acc: 0.9415 - va
Epoch 10/20
49/49 [==============================] - 2s 46ms/step - loss: 0.1607 - acc: 0.9471 - va
Epoch 11/20
49/49 [==============================] - 2s 46ms/step - loss: 0.1458 - acc: 0.9514 - va
Epoch 12/20
49/49 [==============================] - 2s 46ms/step - loss: 0.1304 - acc: 0.9542 - va
Epoch 13/20
49/49 [==============================] - 2s 45ms/step - loss: 0.1294 - acc: 0.9569 - va
Epoch 14/20
49/49 [==============================] - 2s 47ms/step - loss: 0.1228 - acc: 0.9582 - va
Epoch 15/20
49/49 [==============================] - 2s 46ms/step - loss: 0.1158 - acc: 0.9611 - va
Epoch 16/20
49/49 [==============================] - 2s 47ms/step - loss: 0.1123 - acc: 0.9604 - va
Epoch 17/20
49/49 [==============================] - 2s 47ms/step - loss: 0.1083 - acc: 0.9646 - va
Epoch 18/20
49/49 [==============================] - 2s 46ms/step - loss: 0.1089 - acc: 0.9644 - va
Epoch 19/20
49/49 [==============================] - 2s 46ms/step - loss: 0.1042 - acc: 0.9654 - va
Epoch 20/20
49/49 [==============================] - 2s 45ms/step - loss: 0.1043 - acc: 0.9652 - va
```

```
dpt_model_val_loss = dpt_model_hist.history['val_loss']

plt.plot(epochs, val_loss, 'b+', label='Original model')
plt.plot(epochs, dpt_model_val_loss, 'bo', label='Dropout-regularized model')
plt.xlabel('Epochs')
plt.ylabel('Validation loss')
plt.legend()

plt.show()
```



Above plot shows clear improvement once dropout-regularization technique has been implied validation loss significantly reduced.