Notebook credit: based on the F. Chollet's original notebook here.

Fundamentals of machine learning

Improving model fit

Many DL projects will go through the following progression (also listed are ideas that help you make progress if you're stuck at a particular stage):

1. Training loss goes down over time. (如果失败则 try changing gradient descent parameters)
2. Model meaningfully generalizes: you can beat a common-sense baseline you set. (如果失败则 leverage better architecture priors)
3. Your model is able to *overfit* (low training loss, high validation loss). (如果失败则 increase model capacity.)

Now——4. refine generalization by fighting overfitting.

Tuning key gradient descent parameters

Training a MNIST model with an incorrectly high learning rate

An inappropriately large learning rate(of value 1) can cause training to fail even on a simple problem like MNIST.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

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

```
(train_images, train_labels), _ = mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255

model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
model.compile(optimizer=keras.optimizers.RMSprop(1.), # use a learning rate of 1.0 w
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
model.fit(train_images, train_labels,
          epochs=10,
          batch_size=128,
          validation_split=0.2)
```

```
Epoch 1/10
375/375 [==============================] - 6s 14ms/step - loss: 544.6664 - accur
Epoch 2/10
375/375 [==============================] - 4s 11ms/step - loss: 2.5336 - accurac
Epoch 3/10
375/375 [==============================] - 4s 11ms/step - loss: 2.7800 - accurac
Epoch 4/10
375/375 [==============================] - 5s 13ms/step - loss: 2.5513 - accurac
Epoch 5/10
375/375 [==============================] - 4s 10ms/step - loss: 2.5712 - accurac
Epoch 6/10
375/375 [==============================] - 4s 10ms/step - loss: 3.0108 - accurac
Epoch 7/10
375/375 [==============================] - 5s 13ms/step - loss: 2.4447 - accurac
Epoch 8/10
375/375 [==============================] - 4s 11ms/step - loss: 2.5193 - accurac
Epoch 9/10
375/375 [==============================] - 4s 10ms/step - loss: 2.3188 - accurac
Epoch 10/10
375/375 [==============================] - 5s 13ms/step - loss: 2.5309 - accurac
<keras.src.callbacks.History at 0x7bc7292a7bb0>
```

The same model with a more appropriate learning rate

```
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
model.compile(optimizer=keras.optimizers.RMSprop(1e-2), # use a smaller learning ra
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
model.fit(train_images, train_labels,
          epochs=10,
          batch_size=128,
          validation_split=0.2)
```

```
Epoch 1/10
375/375 [==============================] - 5s 12ms/step - loss: 0.3439 - accurac
Epoch 2/10
375/375 [==============================] - 4s 11ms/step - loss: 0.1282 - accurac
Epoch 3/10
375/375 [==============================] - 5s 13ms/step - loss: 0.0973 - accurac
Epoch 4/10
375/375 [==============================] - 4s 10ms/step - loss: 0.0838 - accurac
Epoch 5/10
375/375 [==============================] - 4s 10ms/step - loss: 0.0686 - accurac
Epoch 6/10
375/375 [==============================] - 5s 13ms/step - loss: 0.0598 - accurac
Epoch 7/10
375/375 [==============================] - 4s 10ms/step - loss: 0.0584 - accurac
Epoch 8/10
375/375 [==============================] - 4s 11ms/step - loss: 0.0499 - accurac
Epoch 9/10
375/375 [==============================] - 5s 14ms/step - loss: 0.0435 - accurac
Epoch 10/10
375/375 [==============================] - 4s 10ms/step - loss: 0.0430 - accurac
<keras.src.callbacks.History at 0x7bc71b34be80>
```

If training appears to get stuck, you can try the following:

- Lowering or increasing the learning rate. A learning rate that is too high may lead to updates that vastly overshoot a proper fit, like in the preceding example, and a learning rate that is too low may make training so slow that it appears to stall.
- Increasing the batch size. A batch with more samples will lead to gradients that are more informative and less noisy (lower variance).

Leveraging better architecture priors

You are able to get training started, but for some reason your validation metrics aren't improving at all. They remain no better than what a random classifier would achieve: your model trains but doesn't generalize. There might be several reasons why this might be happening. Two common ones are:

Input data simply doesn't contain sufficient information to predict your targets

- what happened when we tried to fit an MNIST model where the labels were shuffled
- the model would train just fine, but validation accuracy would stay stuck at 10%
- it was plainly impossible to generalize with such a dataset

The kind of model you're using is not suited for the problem at hand

- In a timeseries prediction problem, a densely connected architecture may be less appropriate: a *recurrent* architecture might generalize better
- Using a model that makes the right assumptions about the problem is essential to achieve generalization: you should leverage the right architecture priors
- We willl learn about the best architectures to use for a variety of data modalities—images, text, timeseries, and so on

Increasing model capacity

A simple logistic regression on MNIST

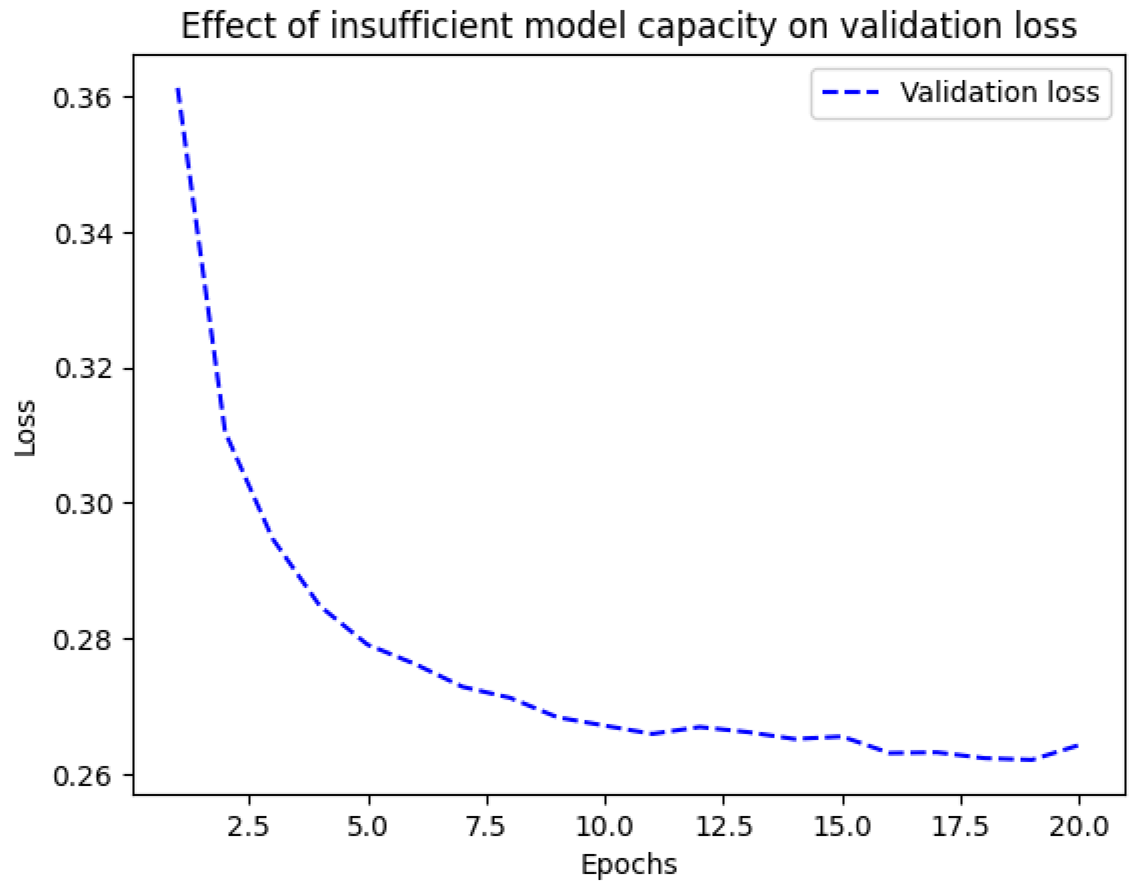
```
model = keras.Sequential([layers.Dense(10, activation="softmax")])
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
history_small_model = model.fit(
    train_images, train_labels,
    epochs=20,
    batch_size=128,
    validation_split=0.2)
```

```
Epoch 1/20
375/375 [==============================] - 2s 5ms/step - loss: 0.6747 - accuracy
Epoch 2/20
375/375 [==============================] - 1s 3ms/step - loss: 0.3535 - accuracy
Epoch 3/20
375/375 [==============================] - 1s 3ms/step - loss: 0.3182 - accuracy
Epoch 4/20
375/375 [==============================] - 1s 3ms/step - loss: 0.3023 - accuracy
Epoch 5/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2926 - accuracy
Epoch 6/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2861 - accuracy
Epoch 7/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2809 - accuracy
Epoch 8/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2769 - accuracy
```

```
Epoch 9/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2738 - accuracy
Epoch 10/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2711 - accuracy
Epoch 11/20
375/375 [==============================] - 2s 4ms/step - loss: 0.2683 - accuracy
Epoch 12/20
375/375 [==============================] - 2s 5ms/step - loss: 0.2666 - accuracy
Epoch 13/20
375/375 [==============================] - 1s 4ms/step - loss: 0.2649 - accuracy
Epoch 14/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2635 - accuracy
Epoch 15/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2623 - accuracy
Epoch 16/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2607 - accuracy
Epoch 17/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2594 - accuracy
Epoch 18/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2584 - accuracy
Epoch 19/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2574 - accuracy
Epoch 20/20
375/375 [==============================] - 1s 3ms/step - loss: 0.2565 - accuracy
```

```
val_loss = history_small_model.history["val_loss"]
epochs = range(1, 21)
plt.plot(epochs, val_loss, "b--",
         label="Validation loss")
plt.title("Effect of insufficient model capacity on validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bc73c20da20>
```



- Validation metrics seem to stall, or to improve very slowly, instead of peaking and reversing course.
- You can fit, but you can’t clearly overfit, even after many iterations over the training data.

It should always be possible to overfit

- If you can’t seem to be able to overfit, it’s likely a problem with the representational power of your model
- You’re going to need a bigger model, one with more capacity, that is to say, one able to store more information
- You can increase representational power by
  - adding more layers
  - using bigger layers (layers with more parameters)
  - using kinds of layers that are more appropriate for the problem at hand (better architecture priors).

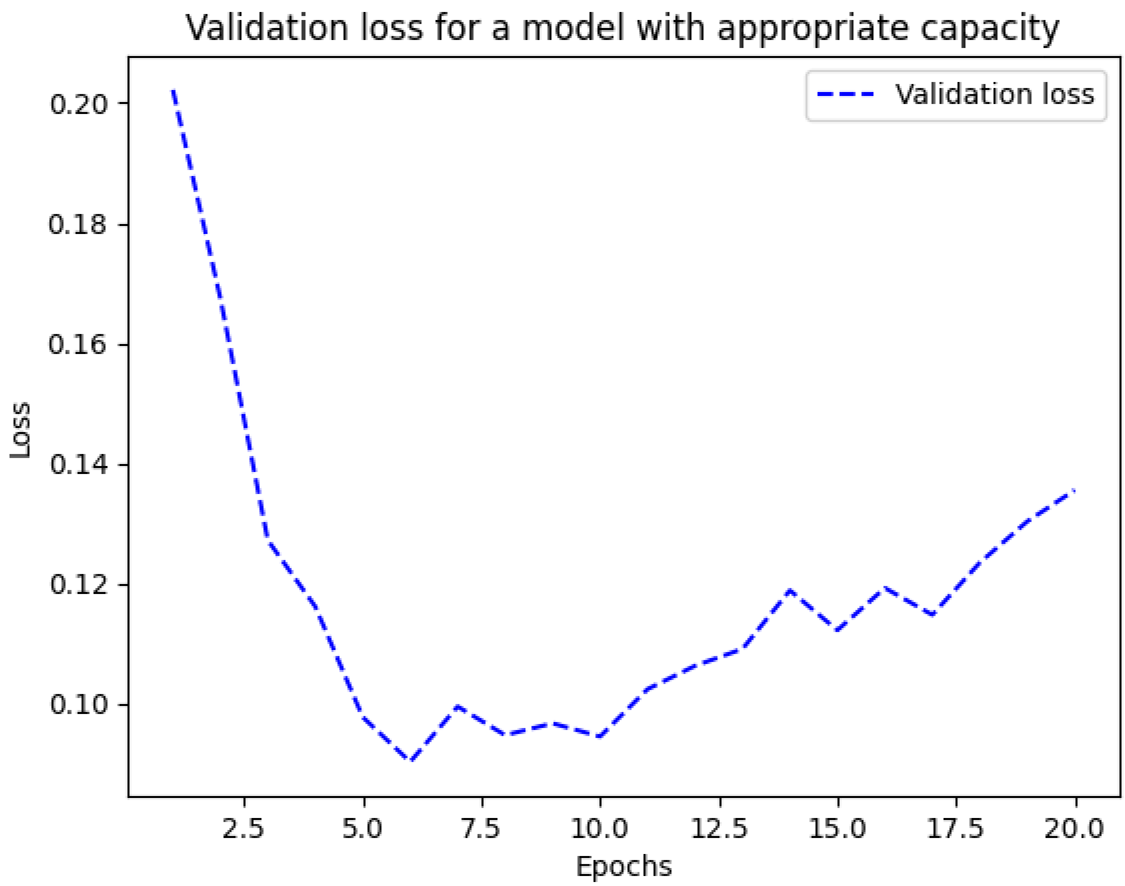
Let’s try training a bigger model, one with two intermediate layers with 96 units each:

```
model = keras.Sequential([
    layers.Dense(96, activation="relu"),
    layers.Dense(96, activation="relu"),
    layers.Dense(10, activation="softmax"),
])
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
history_large_model = model.fit(
    train_images, train_labels,
    epochs=20,
    batch_size=128,
    validation_split=0.2)
```

```
Epoch 1/20
375/375 [==============================] - 3s 7ms/step - loss: 0.3663 - accuracy
Epoch 2/20
375/375 [==============================] - 2s 5ms/step - loss: 0.1646 - accuracy
Epoch 3/20
375/375 [==============================] - 2s 4ms/step - loss: 0.1165 - accuracy
Epoch 4/20
375/375 [==============================] - 2s 5ms/step - loss: 0.0897 - accuracy
Epoch 5/20
375/375 [==============================] - 1s 4ms/step - loss: 0.0724 - accuracy
Epoch 6/20
375/375 [==============================] - 2s 4ms/step - loss: 0.0603 - accuracy
Epoch 7/20
375/375 [==============================] - 2s 4ms/step - loss: 0.0494 - accuracy
Epoch 8/20
375/375 [==============================] - 2s 5ms/step - loss: 0.0418 - accuracy
Epoch 9/20
375/375 [==============================] - 2s 6ms/step - loss: 0.0354 - accuracy
Epoch 10/20
375/375 [==============================] - 2s 5ms/step - loss: 0.0299 - accuracy
Epoch 11/20
375/375 [==============================] - 2s 4ms/step - loss: 0.0251 - accuracy
Epoch 12/20
375/375 [==============================] - 2s 4ms/step - loss: 0.0212 - accuracy
Epoch 13/20
375/375 [==============================] - 1s 4ms/step - loss: 0.0176 - accuracy
Epoch 14/20
375/375 [==============================] - 2s 4ms/step - loss: 0.0153 - accuracy
Epoch 15/20
375/375 [==============================] - 2s 5ms/step - loss: 0.0131 - accuracy
Epoch 16/20
375/375 [==============================] - 3s 7ms/step - loss: 0.0111 - accuracy
Epoch 17/20
375/375 [==============================] - 2s 4ms/step - loss: 0.0098 - accuracy
Epoch 18/20
375/375 [==============================] - 2s 4ms/step - loss: 0.0086 - accuracy
Epoch 19/20
375/375 [==============================] - 2s 5ms/step - loss: 0.0068 - accuracy
Epoch 20/20
375/375 [==============================] - 2s 5ms/step - loss: 0.0061 - accuracy
```

```
val_loss = history_large_model.history["val_loss"]
epochs = range(1, 21)
plt.plot(epochs, val_loss, "b--",
         label="Validation loss")
plt.title("Validation loss for a model with appropriate capacity")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bc71a79a470>
```



Improving generalization

Regularizing your model

Regularization: actively impeding the model's ability to fit perfectly to the training data, with the goal of making the model perform better during validation.

- regularizing a model is a process that should always be guided by an accurate evaluation procedure

- you will only achieve generalization if you can measure it!

Let’s review some of the most common regularization techniques in the context of IMDB movie reviews problem

Reducing the network's size

- a model that is too small will not overfit
- simplest way to mitigate overfitting is to reduce the size of the model
- number of learnable parameters in the model is determined by the number of layers and the number of units per layer
- however, you should use models that have enough parameters that they don’t underfit!
- compromise is to be found between too much capacity and not enough capacity
- no magical formula to determine the right number of layers or the right size for each layer
- evaluate an array of different architectures (on your validation set, not on your test set, of course) in order to find the correct model size for your data
- general workflow for finding an appropriate model size:
    - start with relatively few layers and parameters
    - increase the size of the layers or add new layers until you see diminishing returns with regard to validation loss

Original model

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

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
train_data = vectorize_sequences(train_data)

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"])
history_original = model.fit(train_data, train_labels,
                             epochs=20, batch_size=512, validation_split=0.4)
```

```
Epoch 1/20
30/30 [==============================] - 4s 121ms/step - loss: 0.5265 - accuracy
Epoch 2/20
30/30 [==============================] - 2s 55ms/step - loss: 0.3205 - accuracy:
Epoch 3/20
30/30 [==============================] - 2s 56ms/step - loss: 0.2395 - accuracy:
Epoch 4/20
30/30 [==============================] - 2s 56ms/step - loss: 0.1914 - accuracy:
Epoch 5/20
30/30 [==============================] - 1s 43ms/step - loss: 0.1571 - accuracy:
Epoch 6/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1371 - accuracy:
Epoch 7/20
30/30 [==============================] - 2s 54ms/step - loss: 0.1159 - accuracy:
Epoch 8/20
30/30 [==============================] - 2s 52ms/step - loss: 0.1014 - accuracy:
Epoch 9/20
30/30 [==============================] - 2s 60ms/step - loss: 0.0834 - accuracy:
Epoch 10/20
30/30 [==============================] - 2s 59ms/step - loss: 0.0757 - accuracy:
Epoch 11/20
30/30 [==============================] - 2s 57ms/step - loss: 0.0630 - accuracy:
Epoch 12/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0554 - accuracy:
Epoch 13/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0467 - accuracy:
Epoch 14/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0393 - accuracy:
Epoch 15/20
30/30 [==============================] - 2s 55ms/step - loss: 0.0376 - accuracy:
Epoch 16/20
30/30 [==============================] - 1s 41ms/step - loss: 0.0251 - accuracy:
Epoch 17/20
30/30 [==============================] - 2s 56ms/step - loss: 0.0242 - accuracy:
Epoch 18/20
30/30 [==============================] - 2s 67ms/step - loss: 0.0209 - accuracy:
Epoch 19/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0204 - accuracy:
Epoch 20/20
30/30 [==============================] - 1s 40ms/step - loss: 0.0170 - accuracy:
```

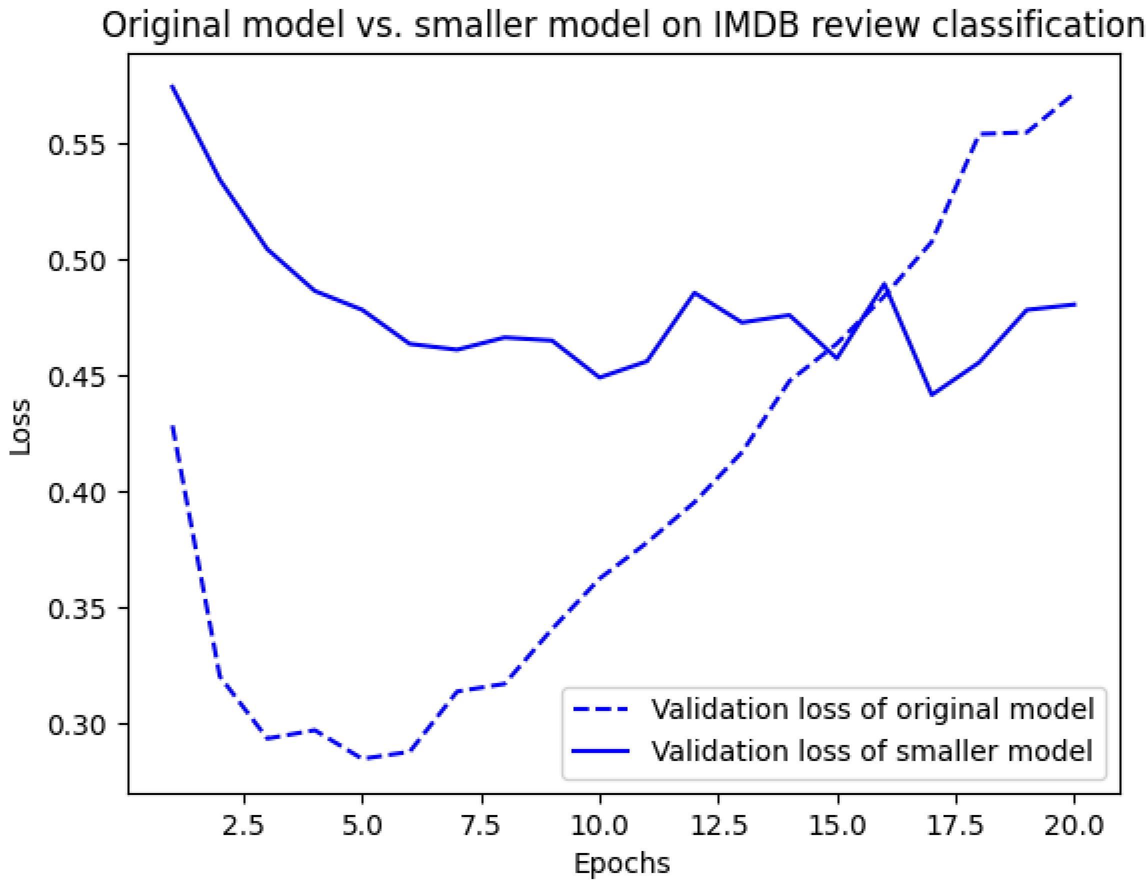
Version of the model with lower capacity

```
model = keras.Sequential([
    layers.Dense(4, activation="relu"),
    layers.Dense(4, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_smaller_model = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

```
Epoch 1/20
30/30 [==============================] - 3s 94ms/step - loss: 0.6234 - accuracy:
Epoch 2/20
30/30 [==============================] - 1s 50ms/step - loss: 0.5377 - accuracy:
Epoch 3/20
30/30 [==============================] - 2s 52ms/step - loss: 0.4921 - accuracy:
Epoch 4/20
30/30 [==============================] - 2s 57ms/step - loss: 0.4597 - accuracy:
Epoch 5/20
30/30 [==============================] - 1s 43ms/step - loss: 0.4343 - accuracy:
Epoch 6/20
30/30 [==============================] - 1s 35ms/step - loss: 0.4129 - accuracy:
Epoch 7/20
30/30 [==============================] - 1s 33ms/step - loss: 0.3946 - accuracy:
Epoch 8/20
30/30 [==============================] - 1s 34ms/step - loss: 0.3783 - accuracy:
Epoch 9/20
30/30 [==============================] - 1s 36ms/step - loss: 0.3629 - accuracy:
Epoch 10/20
30/30 [==============================] - 1s 35ms/step - loss: 0.3494 - accuracy:
Epoch 11/20
30/30 [==============================] - 1s 38ms/step - loss: 0.3364 - accuracy:
Epoch 12/20
30/30 [==============================] - 1s 34ms/step - loss: 0.3248 - accuracy:
Epoch 13/20
30/30 [==============================] - 1s 51ms/step - loss: 0.3138 - accuracy:
Epoch 14/20
30/30 [==============================] - 2s 59ms/step - loss: 0.3032 - accuracy:
Epoch 15/20
30/30 [==============================] - 2s 57ms/step - loss: 0.2933 - accuracy:
Epoch 16/20
30/30 [==============================] - 2s 54ms/step - loss: 0.2839 - accuracy:
Epoch 17/20
30/30 [==============================] - 1s 41ms/step - loss: 0.2751 - accuracy:
Epoch 18/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2669 - accuracy:
Epoch 19/20
30/30 [==============================] - 1s 32ms/step - loss: 0.2587 - accuracy:
Epoch 20/20
30/30 [==============================] - 1s 31ms/step - loss: 0.2502 - accuracy:
```

```
val_loss_original = history_original.history["val_loss"]
val_loss_smaller = history_smaller_model.history["val_loss"]
epochs = range(1, 21)
plt.plot(epochs, val_loss_original, "b--",
         label="Validation loss of original model")
plt.plot(epochs, val_loss_smaller, "b-",
         label="Validation loss of smaller model")
plt.title("Original model vs. smaller model on IMDB review classification")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bc720ce2710>
```



- the smaller model starts overfitting later than the reference model
- its performance degrades more slowly once it starts overfitting

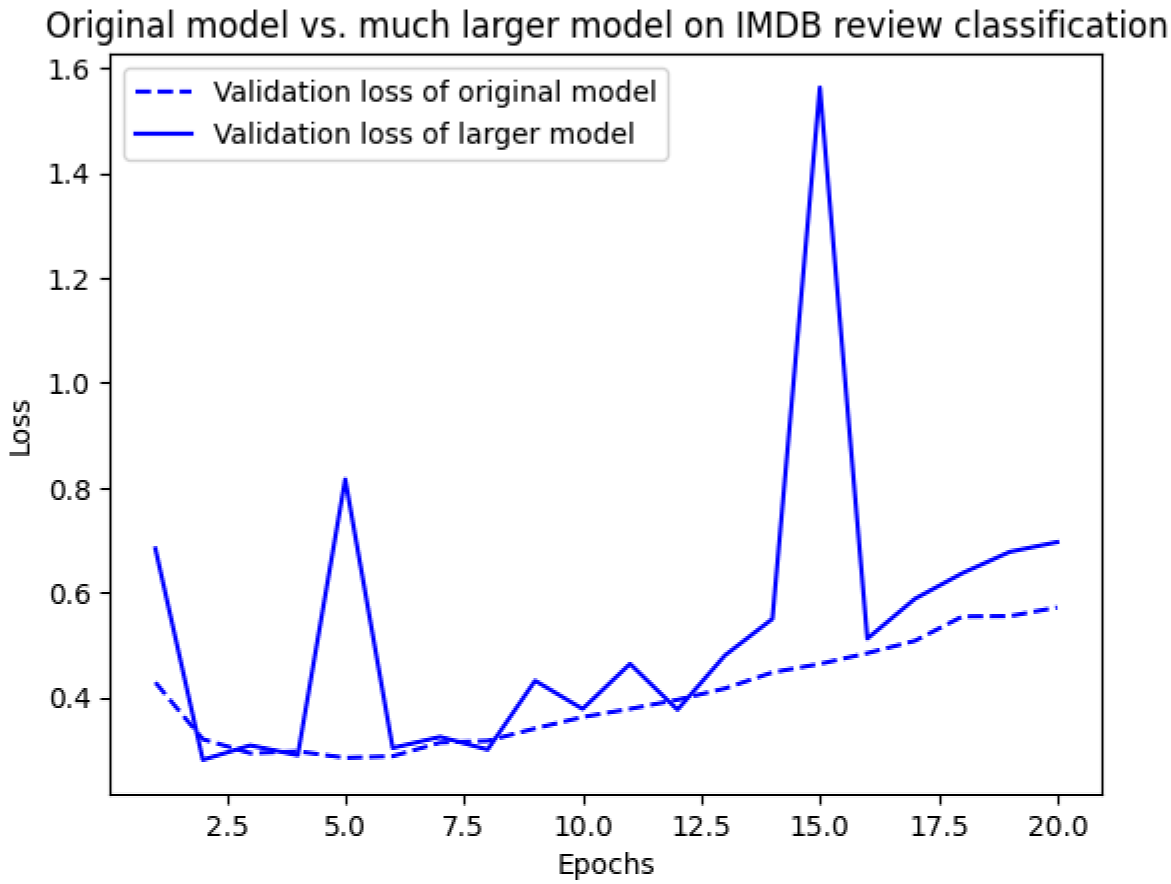
Version of the model with higher capacity

```
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(512, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_larger_model = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

```
Epoch 1/20
30/30 [==============================] - 13s 400ms/step - loss: 0.5620 - accurac
Epoch 2/20
30/30 [==============================] - 11s 367ms/step - loss: 0.3395 - accurac
Epoch 3/20
30/30 [==============================] - 11s 360ms/step - loss: 0.2385 - accurac
Epoch 4/20
30/30 [==============================] - 12s 416ms/step - loss: 0.1811 - accurac
Epoch 5/20
30/30 [==============================] - 12s 417ms/step - loss: 0.1378 - accurac
Epoch 6/20
30/30 [==============================] - 9s 308ms/step - loss: 0.1225 - accuracy
Epoch 7/20
30/30 [==============================] - 9s 311ms/step - loss: 0.0876 - accuracy
Epoch 8/20
30/30 [==============================] - 10s 352ms/step - loss: 0.0759 - accurac
Epoch 9/20
30/30 [==============================] - 10s 346ms/step - loss: 0.0189 - accurac
Epoch 10/20
30/30 [==============================] - 10s 324ms/step - loss: 0.0759 - accurac
Epoch 11/20
30/30 [==============================] - 10s 320ms/step - loss: 0.0072 - accurac
Epoch 12/20
30/30 [==============================] - 10s 327ms/step - loss: 0.0777 - accurac
Epoch 13/20
30/30 [==============================] - 12s 417ms/step - loss: 0.0054 - accurac
Epoch 14/20
30/30 [==============================] - 13s 430ms/step - loss: 0.0019 - accurac
Epoch 15/20
30/30 [==============================] - 11s 356ms/step - loss: 0.0109 - accurac
Epoch 16/20
30/30 [==============================] - 10s 331ms/step - loss: 0.0229 - accurac
Epoch 17/20
30/30 [==============================] - 10s 340ms/step - loss: 0.0010 - accurac
Epoch 18/20
30/30 [==============================] - 10s 341ms/step - loss: 6.1109e-04 - acc
Epoch 19/20
30/30 [==============================] - 11s 356ms/step - loss: 4.0700e-04 - acc
Epoch 20/20
30/30 [==============================] - 10s 331ms/step - loss: 3.0132e-04 - acc
```

```
val_loss_larger = history_larger_model.history["val_loss"]
epochs = range(1, 21)
plt.plot(epochs, val_loss_original, "b--",
         label="Validation loss of original model")
plt.plot(epochs, val_loss_larger, "b-",
         label="Validation loss of larger model")
plt.title("Original model vs. much larger model on IMDB review classification")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bc7259e9990>
```



- bigger model starts overfitting almost immediately
- it overfits much more severely
- its validation loss is also noisier
- it gets training loss near zero very quickly
- a very high capacity model will
  - fit the the training data quickly (resulting in a low training loss)
  - but will be more susceptible it is to overfitting (resulting in a large difference between the training and validation loss)

Adding weight regularization

Adding L2 weight regularization to the model

Regularization can be applied to:

- weights using `kernel_regularizer`
- biases using `bias_regularizer`
- output of the layer using `activity_regularizer`

We will use weight regularization below.

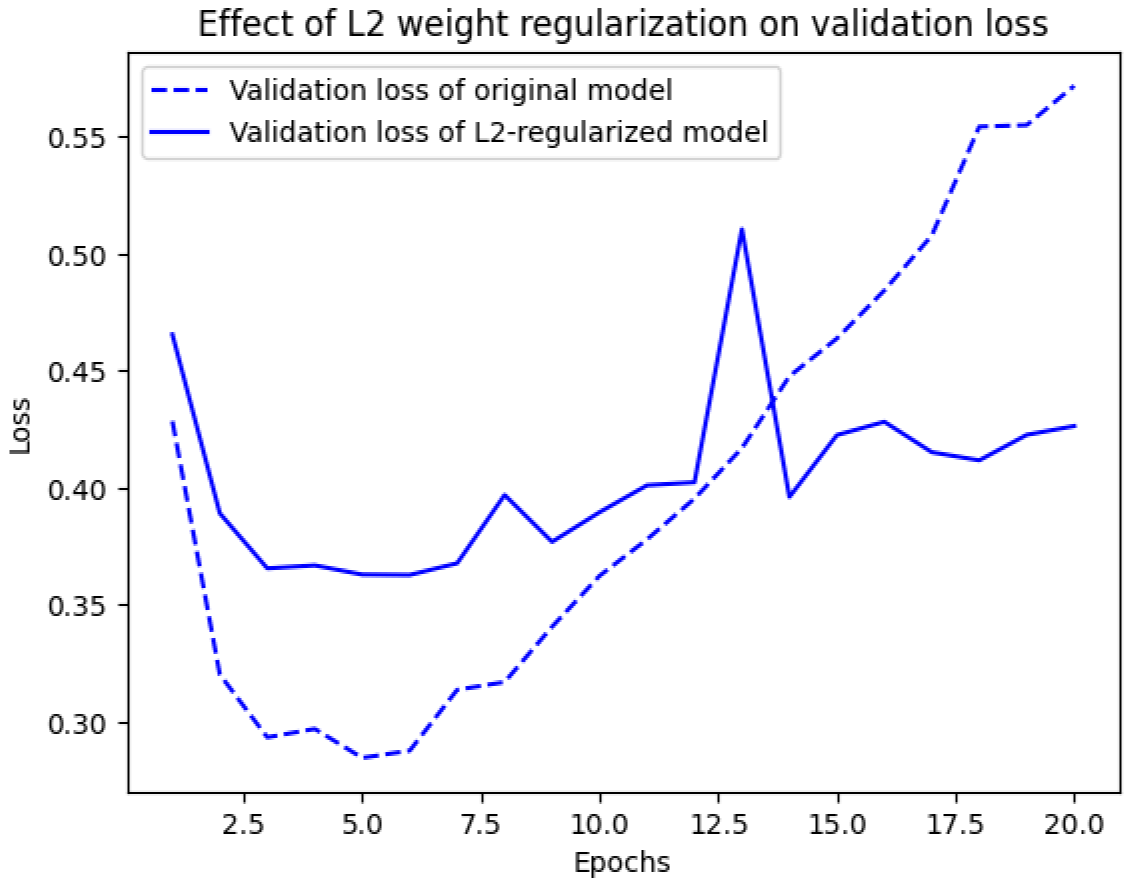
```
from tensorflow.keras import regularizers
model = keras.Sequential([
    # every coefficient in the weight matrix of the layer will add
    # 0.002 * weight_coefficient_value ** 2
    # to the total loss of the model
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_l2_reg = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

```
Epoch 1/20
30/30 [==============================] - 3s 72ms/step - loss: 0.6074 - accuracy:
Epoch 2/20
30/30 [==============================] - 1s 47ms/step - loss: 0.4009 - accuracy:
Epoch 3/20
30/30 [==============================] - 1s 36ms/step - loss: 0.3311 - accuracy:
Epoch 4/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2942 - accuracy:
Epoch 5/20
30/30 [==============================] - 1s 35ms/step - loss: 0.2740 - accuracy:
Epoch 6/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2592 - accuracy:
Epoch 7/20
30/30 [==============================] - 2s 56ms/step - loss: 0.2473 - accuracy:
Epoch 8/20
30/30 [==============================] - 2s 55ms/step - loss: 0.2378 - accuracy:
Epoch 9/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2360 - accuracy:
```

```
Epoch 10/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2248 - accuracy:
Epoch 11/20
30/30 [==============================] - 1s 37ms/step - loss: 0.2285 - accuracy:
Epoch 12/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2193 - accuracy:
Epoch 13/20
30/30 [==============================] - 1s 35ms/step - loss: 0.2105 - accuracy:
Epoch 14/20
30/30 [==============================] - 1s 38ms/step - loss: 0.2105 - accuracy:
Epoch 15/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2108 - accuracy:
Epoch 16/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2051 - accuracy:
Epoch 17/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2093 - accuracy:
Epoch 18/20
30/30 [==============================] - 2s 54ms/step - loss: 0.2018 - accuracy:
Epoch 19/20
30/30 [==============================] - 1s 50ms/step - loss: 0.1996 - accuracy:
Epoch 20/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1957 - accuracy:
```

```
val_loss_l2_reg = history_l2_reg.history["val_loss"]
epochs = range(1, 21)
plt.plot(epochs, val_loss_original, "b--",
         label="Validation loss of original model")
plt.plot(epochs, val_loss_l2_reg, "b-",
         label="Validation loss of L2-regularized model")
plt.title("Effect of L2 weight regularization on validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bc72018e1d0>
```



- model with L2 regularization has become much more resistant to overfitting than the reference model
- both models have the same number of parameters

Total loss when using weight regularization includes prediction losses as well as layer losses

- "loss" as a metric changes meaning when you have weight regularization
- without regularization, loss is simply the average of the prediction loss function over the dataset
- with weight regularization, loss includes *both* prediction losses as well as regularization losses for regularized layers

Different weight regularizers available in Keras

```
from tensorflow.keras import regularizers
regularizers.l1(0.001)
regularizers.l1_l2(l1=0.001, l2=0.001)
```

```
<keras.src.regularizers.L1L2 at 0x7bc72b6f3640>
```

- weight regularization is more typically used for smaller deep learning models
- large deep learning models tend to be so overparameterized that imposing constraints on weight values hasn't much impact on model capacity and generalization
- in these cases, a different regularization technique is preferred: **dropout**

Adding dropout

Adding dropout to the IMDB model

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_dropout = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

```
Epoch 1/20
30/30 [==============================] - 3s 70ms/step - loss: 0.6242 - accuracy:
Epoch 2/20
30/30 [==============================] - 1s 37ms/step - loss: 0.5097 - accuracy:
Epoch 3/20
30/30 [==============================] - 1s 34ms/step - loss: 0.4326 - accuracy:
Epoch 4/20
30/30 [==============================] - 1s 33ms/step - loss: 0.3838 - accuracy:
Epoch 5/20
30/30 [==============================] - 2s 52ms/step - loss: 0.3360 - accuracy:
Epoch 6/20
30/30 [==============================] - 2s 63ms/step - loss: 0.2942 - accuracy:
Epoch 7/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2681 - accuracy:
Epoch 8/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2303 - accuracy:
Epoch 9/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2134 - accuracy:
Epoch 10/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1925 - accuracy:
Epoch 11/20
30/30 [==============================] - 1s 39ms/step - loss: 0.1731 - accuracy:
Epoch 12/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1594 - accuracy:
Epoch 13/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1489 - accuracy:
```

```
Epoch 14/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1315 - accuracy:
Epoch 15/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1247 - accuracy:
Epoch 16/20
30/30 [==============================] - 2s 56ms/step - loss: 0.1189 - accuracy:
Epoch 17/20
30/30 [==============================] - 1s 50ms/step - loss: 0.1094 - accuracy:
Epoch 18/20
30/30 [==============================] - 1s 36ms/step - loss: 0.1035 - accuracy:
Epoch 19/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0996 - accuracy:
Epoch 20/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0942 - accuracy:
```

```
val_loss_dropout = history_dropout.history["val_loss"]
epochs = range(1, 21)
plt.plot(epochs, val_loss_original, "b--",
         label="Validation loss of original model")
plt.plot(epochs, val_loss_dropout, "b-",
         label="Validation loss of dropout-regularized model")
plt.title("Effect of dropout on validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bc724d25570>
```

- dropout achieves clear improvement over the reference model
- it also seems to be working much better than L2 regularization (lowest validation loss reached has improved)

How does dropout work?

- dropout, **applied to a layer**, consists of randomly dropping out (setting to zero) a number of output features of the layer **during training**
- after applying dropout, the layer output will have a few zero entries distributed at random
- the dropout rate is the fraction of the features that are zeroed out; it's usually set between 0.2 and 0.5.

```
batch_size = 4
feature_dim = 5
layer_output = tf.random.uniform((batch_size, feature_dim)) # in reality, layer_out
layer_output.numpy()
```

```
array([[0.02471972, 0.04226112, 0.6676756 , 0.28473115, 0.10217285],
       [0.43423676, 0.8592428 , 0.5135548 , 0.03304565, 0.91238153],
       [0.41194844, 0.9791504 , 0.8356286 , 0.2270031 , 0.373057  ],
       [0.95226467, 0.92783797, 0.4603274 , 0.34082532, 0.12339067]],
      dtype=float32)
```

```
dropout = 0.2 # dropout probability
mask = tf.random.uniform(shape=layer_output.shape) < 1 - dropout # random boolean a
mask = tf.cast(mask, tf.float32) # convert True/False to 1/0
mask.numpy()
```

```
array([[1., 1., 1., 0., 1.],
       [1., 1., 1., 1., 1.],
       [0., 1., 1., 1., 1.],
       [1., 1., 0., 0., 1.]], dtype=float32)
```

```
layer_output_dropout = layer_output * mask
layer_output_dropout.numpy() # roughly half of the entries will have been zeroed ou
```

```
array([[0.02471972, 0.04226112, 0.6676756 , 0.        , 0.10217285],
       [0.43423676, 0.8592428 , 0.5135548 , 0.03304565, 0.91238153],
       [0.        , 0.9791504 , 0.8356286 , 0.2270031 , 0.373057  ],
       [0.95226467, 0.92783797, 0.        , 0.        , 0.12339067]],
```