Open in Colab

Overfitting and Underfitting

This notebook explores overfitting and underfitting scenarios based on
https://www.tensorflow.org/tutorials/keras/overfit_and_underfit

Setup and initialization

```
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import regularizers

print(tf.__version__)

# let's set the random seed to make the results reproducible
tf.random.set_seed(74)
```

```
2.9.2
```

Load dataset

```
#!pip install git+https://github.com/tensorflow/docs

import tensorflow_docs as tfdocs
import tensorflow_docs.modeling
import tensorflow_docs.plots
```

```
from  IPython import display
from matplotlib import pyplot as plt

import numpy as np

import pathlib
import shutil
import tempfile
```

```
logdir = pathlib.Path(tempfile.mkdtemp())/"tensorboard_logs"
shutil.rmtree(logdir, ignore_errors=True)
```

The Higgs Dataset

```
gz = tf.keras.utils.get_file('HIGGS.csv.gz', 'http://mlphysics.ics.uci.edu/data/higgs/HIGGS.c
```

```
FEATURES = 28
```

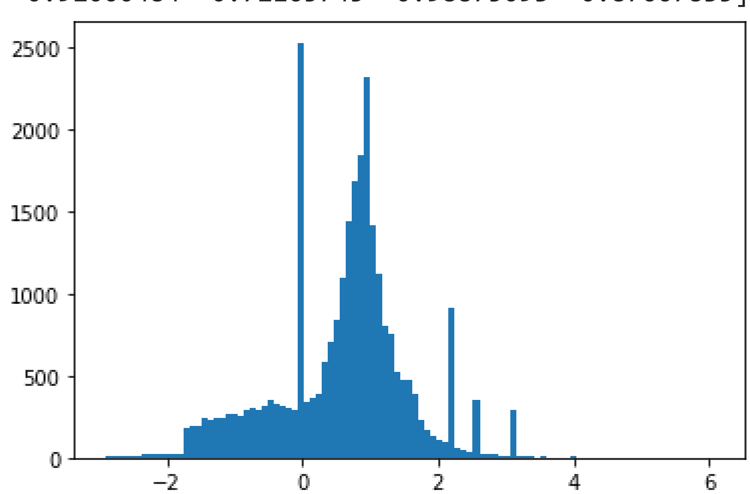
```
ds = tf.data.experimental.CsvDataset(gz,[float(),]*(FEATURES+1), compression_type="GZIP")
```

```
def pack_row(*row):
  label = row[0]
  features = tf.stack(row[1:],1)
  return features, label
```

```
packed_ds = ds.batch(10000).map(pack_row).unbatch()
```

```
for features,label in packed_ds.batch(1000).take(1):
  print(features[0])
  plt.hist(features.numpy().flatten(), bins = 101)
```

```
tf.Tensor(
[ 0.8692932  -0.6350818   0.22569026  0.32747006 -0.6899932   0.75420225
 -0.24857314 -1.0920639   0.          1.3749921  -0.6536742   0.9303491
  1.1074361   1.1389043  -1.5781983  -1.0469854   0.          0.65792954
 -0.01045457 -0.04576717  3.1019614   1.35376     0.9795631   0.97807616
  0.92000484  0.72165745  0.98875093  0.87667835], shape=(28,), dtype=float32)
```




```
N_VALIDATION = int(1e3)
```

```
N_TRAIN = int(1e4)
BUFFER_SIZE = int(1e4)
BATCH_SIZE = 500
STEPS_PER_EPOCH = N_TRAIN//BATCH_SIZE
```

```
validate_ds = packed_ds.take(N_VALIDATION).cache()
train_ds = packed_ds.skip(N_VALIDATION).take(N_TRAIN).cache()
```

```
train_ds
```

```
<CacheDataset element_spec=(TensorSpec(shape=(28,), dtype=tf.float32, name=None),
TensorSpec(shape=(), dtype=tf.float32, name=None))>
```

```
validate_ds = validate_ds.batch(BATCH_SIZE)
train_ds = train_ds.shuffle(BUFFER_SIZE).repeat().batch(BATCH_SIZE)
```

Demonstrating Overfitting

In deep learning, the number of learnable parameters in a model is often referred to as the model's "capacity".

Always keep this in mind: deep learning models tend to be good at fitting to the training data, but the real challenge is generalization, not fitting.

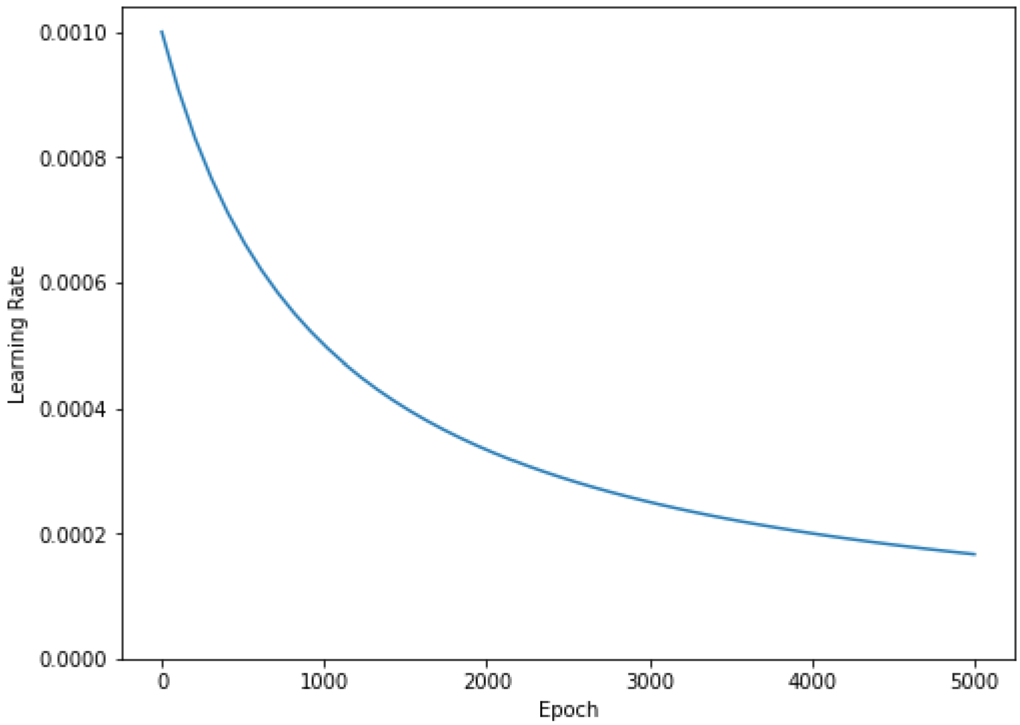
Training procedure

Many models train better if you gradually reduce the learning rate during training. Use `tf.keras.optimizers.schedules` to reduce the learning rate over

```
lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
  0.001,
  decay_steps=STEPS_PER_EPOCH*1000,
  decay_rate=1,
  staircase=False)

def get_optimizer():
  return tf.keras.optimizers.Adam(lr_schedule)
```

```
step = np.linspace(0,100000)
lr = lr_schedule(step)
plt.figure(figsize = (8,6))
plt.plot(step/STEPS_PER_EPOCH, lr)
plt.ylim([0,max(plt.ylim())])
```

```
plt.xlabel('Epoch')
_ = plt.ylabel('Learning Rate')
```



Use callbacks.TensorBoard to generate TensorBoard logs for the training.

```
def get_callbacks(name):
  return [
    tfdocs.modeling.EpochDots(),
    tf.keras.callbacks.EarlyStopping(monitor='val_binary_crossentropy', patience=200),
    tf.keras.callbacks.TensorBoard(logdir/name),
  ]
```

Similarly each model will use the same Model.compile and Model.fit settings:

```
def compile_and_fit(model, name, optimizer=None, max_epochs=10000):
  if optimizer is None:
    optimizer = get_optimizer()
  model.compile(optimizer=optimizer,
                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=[
                  tf.keras.losses.BinaryCrossentropy(
                      from_logits=True, name='binary_crossentropy'),
                  'accuracy'])

  model.summary()

  history = model.fit(
    train_ds,
    steps_per_epoch = STEPS_PER_EPOCH,
```

```
    epochs=max_epochs,
    validation_data=validate_ds,
    callbacks=get_callbacks(name),
    verbose=0)
  return history
```

Tiny Model

```
tiny_model = tf.keras.Sequential([
    layers.Dense(16, activation='elu', input_shape=(FEATURES,)),
    layers.Dense(1)
])
```

```
size_histories = {}
```

```
size_histories['Tiny'] = compile_and_fit(tiny_model, 'sizes/Tiny')
```

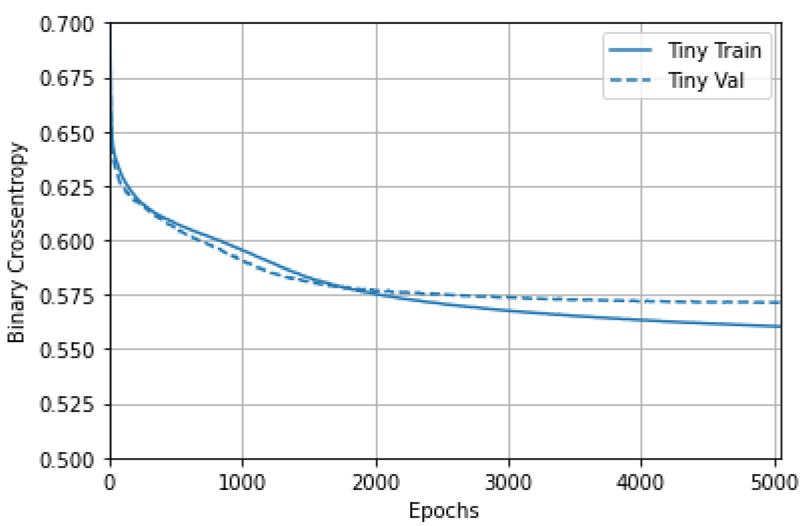
```
....................................................................................................
Epoch: 2200, accuracy:0.6772,  binary_crossentropy:0.5731,  loss:0.5731,  val_accurac
....................................................................................................
Epoch: 2300, accuracy:0.6838,  binary_crossentropy:0.5725,  loss:0.5725,  val_accurac
....................................................................................................
Epoch: 2400, accuracy:0.6763,  binary_crossentropy:0.5716,  loss:0.5716,  val_accurac
....................................................................................................
Epoch: 2500, accuracy:0.6780,  binary_crossentropy:0.5708,  loss:0.5708,  val_accurac
....................................................................................................
Epoch: 2600, accuracy:0.6812,  binary_crossentropy:0.5699,  loss:0.5699,  val_accurac
....................................................................................................
Epoch: 2700, accuracy:0.6840,  binary_crossentropy:0.5695,  loss:0.5695,  val_accurac
....................................................................................................
Epoch: 2800, accuracy:0.6830,  binary_crossentropy:0.5686,  loss:0.5686,  val_accurac
....................................................................................................
Epoch: 2900, accuracy:0.6872,  binary_crossentropy:0.5683,  loss:0.5683,  val_accurac
....................................................................................................
Epoch: 3000, accuracy:0.6856,  binary_crossentropy:0.5675,  loss:0.5675,  val_accurac
....................................................................................................
Epoch: 3100, accuracy:0.6858,  binary_crossentropy:0.5671,  loss:0.5671,  val_accurac
....................................................................................................
Epoch: 3200, accuracy:0.6880,  binary_crossentropy:0.5665,  loss:0.5665,  val_accurac
....................................................................................................
Epoch: 3300, accuracy:0.6899,  binary_crossentropy:0.5661,  loss:0.5661,  val_accurac
....................................................................................................
Epoch: 3400, accuracy:0.6927,  binary_crossentropy:0.5660,  loss:0.5660,  val_accurac
....................................................................................................
Epoch: 3500, accuracy:0.6863,  binary_crossentropy:0.5653,  loss:0.5653,  val_accurac
....................................................................................................
Epoch: 3600, accuracy:0.6871,  binary_crossentropy:0.5647,  loss:0.5647,  val_accurac
....................................................................................................
Epoch: 3700, accuracy:0.6897,  binary_crossentropy:0.5645,  loss:0.5645,  val_accurac
```

```
....................................................................................
Epoch: 3800, accuracy:0.6919,  binary_crossentropy:0.5641,  loss:0.5641,  val_accurac
....................................................................................
Epoch: 3900, accuracy:0.6894,  binary_crossentropy:0.5637,  loss:0.5637,  val_accurac
....................................................................................
Epoch: 4000, accuracy:0.6900,  binary_crossentropy:0.5633,  loss:0.5633,  val_accurac
....................................................................................
Epoch: 4100, accuracy:0.6887,  binary_crossentropy:0.5631,  loss:0.5631,  val_accurac
....................................................................................
Epoch: 4200, accuracy:0.6929,  binary_crossentropy:0.5626,  loss:0.5626,  val_accurac
....................................................................................
Epoch: 4300, accuracy:0.6930,  binary_crossentropy:0.5621,  loss:0.5621,  val_accurac
....................................................................................
Epoch: 4400, accuracy:0.6920,  binary_crossentropy:0.5619,  loss:0.5619,  val_accurac
....................................................................................
Epoch: 4500, accuracy:0.6913,  binary_crossentropy:0.5617,  loss:0.5617,  val_accurac
....................................................................................
Epoch: 4600, accuracy:0.6945,  binary_crossentropy:0.5619,  loss:0.5619,  val_accurac
....................................................................................
Epoch: 4700, accuracy:0.6934,  binary_crossentropy:0.5611,  loss:0.5611,  val_accurac
....................................................................................
Epoch: 4800, accuracy:0.6966,  binary_crossentropy:0.5609,  loss:0.5609,  val_accurac
....................................................................................
Epoch: 4900, accuracy:0.6963,  binary_crossentropy:0.5607,  loss:0.5607,  val_accurac
....................................................................................
Epoch: 5000, accuracy:0.6898,  binary_crossentropy:0.5605,  loss:0.5605,  val_accurac
```

Now check how the model did:

```
plotter = tfdocs.plots.HistoryPlotter(metric = 'binary_crossentropy', smoothing_std=10)
plotter.plot(size_histories)
plt.ylim([0.5, 0.7])
```

```
(0.5, 0.7)
```



Small model

To check if you can beat the performance of the small model, progressively train some larger models.

Try two hidden layers with 16 units each:

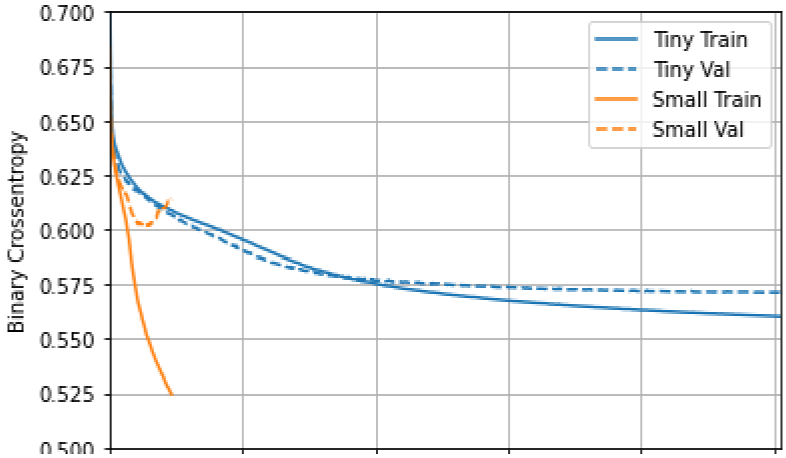
```
small_model = tf.keras.Sequential([
    # `input_shape` is only required here so that `.summary` works.
    layers.Dense(32, activation='elu', input_shape=(FEATURES,)),
    layers.Dense(32, activation='elu'),
    layers.Dense(1)
])
```

```
size_histories['Small'] = compile_and_fit(small_model, 'sizes/Small')
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 32)                928

 dense_22 (Dense)            (None, 32)                1056

 dense_23 (Dense)            (None, 1)                 33

=================================================================
Total params: 2,017
Trainable params: 2,017
Non-trainable params: 0
_________________________________________________________________

Epoch: 0, accuracy:0.4793,  binary_crossentropy:0.7203,  loss:0.7203,  val_accuracy:0.4
....................................................................................
Epoch: 100, accuracy:0.6156,  binary_crossentropy:0.6113,  loss:0.6113,  val_accuracy:0
....................................................................................
Epoch: 200, accuracy:0.6791,  binary_crossentropy:0.5718,  loss:0.5718,  val_accuracy:0
....................................................................................
Epoch: 300, accuracy:0.6984,  binary_crossentropy:0.5479,  loss:0.5479,  val_accuracy:0
....................................................................................
Epoch: 400, accuracy:0.7145,  binary_crossentropy:0.5349,  loss:0.5349,  val_accuracy:0
....................................................................
```

```
# plot
#plotter = tfdocs.plots.HistoryPlotter(metric = 'binary_crossentropy', smoothing_std=10)
plotter.plot(size_histories)
plt.ylim([0.5, 0.7])
```

(0.5, 0.7)



Medium model

```
medium_model = tf.keras.Sequential([
    layers.Dense(64, activation='elu', input_shape=(FEATURES,)),
    layers.Dense(64, activation='elu'),
    layers.Dense(64, activation='elu'),
    layers.Dense(1)
])
```

And train the model using the same data:

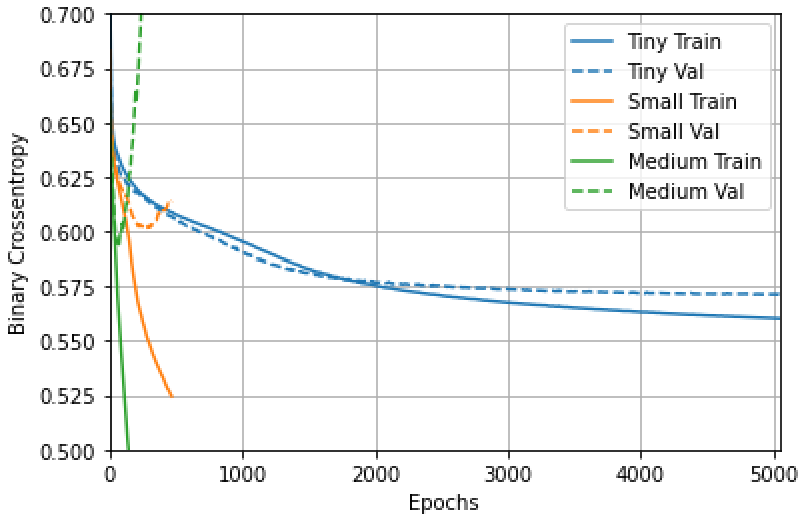
```
size_histories['Medium']  = compile_and_fit(medium_model, "sizes/Medium")
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_24 (Dense)            (None, 64)                1856      
                                                                 
 dense_25 (Dense)            (None, 64)                4160      
                                                                 
 dense_26 (Dense)            (None, 64)                4160      
                                                                 
 dense_27 (Dense)            (None, 1)                 65        
                                                                 
=================================================================
Total params: 10,241
Trainable params: 10,241
Non-trainable params: 0
_________________________________________________________________

Epoch: 0, accuracy:0.4973,  binary_crossentropy:0.7047,  loss:0.7047,  val_accuracy:0.4
....................................................................................
Epoch: 100, accuracy:0.7171,  binary_crossentropy:0.5324,  loss:0.5324,  val_accuracy:0
....................................................................................
```

```
Epoch: 200, accuracy:0.7767,  binary_crossentropy:0.4458,  loss:0.4458,  val_accuracy:0
....................................................................
```

```
plotter.plot(size_histories)
plt.ylim([0.5, 0.7])
```

```
(0.5, 0.7)
```



Large model

```
large_model = tf.keras.Sequential([
    layers.Dense(512, activation='elu', input_shape=(FEATURES,)),
    layers.Dense(512, activation='elu'),
    layers.Dense(512, activation='elu'),
    layers.Dense(512, activation='elu'),
    layers.Dense(1)
])
```

And, again, train the model using the same data:

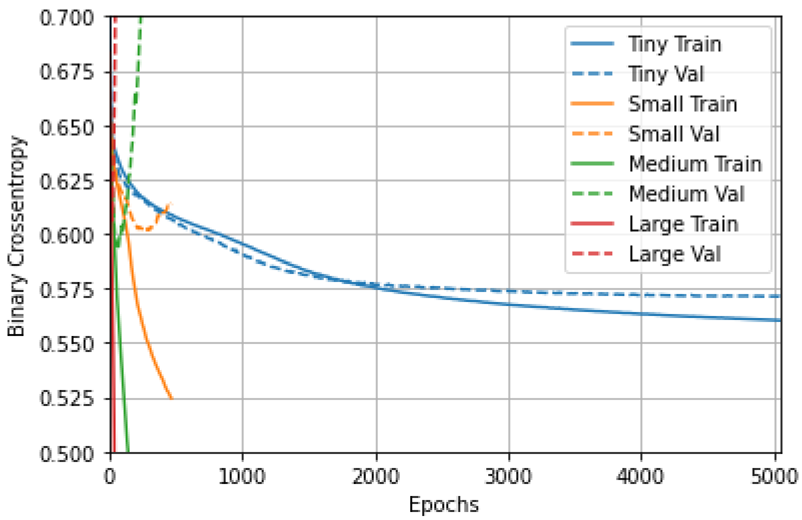
```
size_histories['large'] = compile_and_fit(large_model, "sizes/large")
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_28 (Dense)            (None, 512)               14848

 dense_29 (Dense)            (None, 512)               262656

 dense_30 (Dense)            (None, 512)               262656
```

```
 dense_31 (Dense)            (None, 512)               262656

 dense_32 (Dense)            (None, 1)                 513

=================================================================
Total params: 803,329
Trainable params: 803,329
Non-trainable params: 0
_________________________________________________________________

Epoch: 0, accuracy:0.5066,  binary_crossentropy:0.8190,  loss:0.8190,  val_accuracy:0.4
....................................................................................................
Epoch: 100, accuracy:1.0000,  binary_crossentropy:0.0020,  loss:0.0020,  val_accuracy:0
....................................................................................................
Epoch: 200, accuracy:1.0000,  binary_crossentropy:0.0001,  loss:0.0001,  val_accuracy:0
.........................
```

```
plotter.plot(size_histories)
plt.ylim([0.5, 0.7])
```

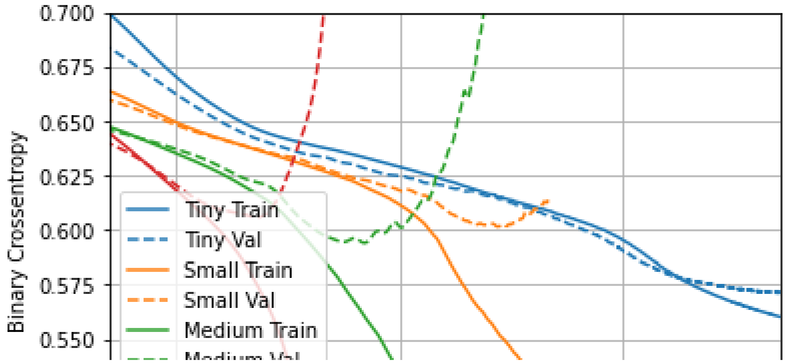
```
(0.5, 0.7)
```



Plot the training and validation losses

```
plotter.plot(size_histories)
a = plt.xscale('log')
plt.xlim([5, max(plt.xlim())])
plt.ylim([0.5, 0.7])
plt.xlabel("Epochs [Log Scale]")
```

```
Text(0.5, 0, 'Epochs [Log Scale]')
```

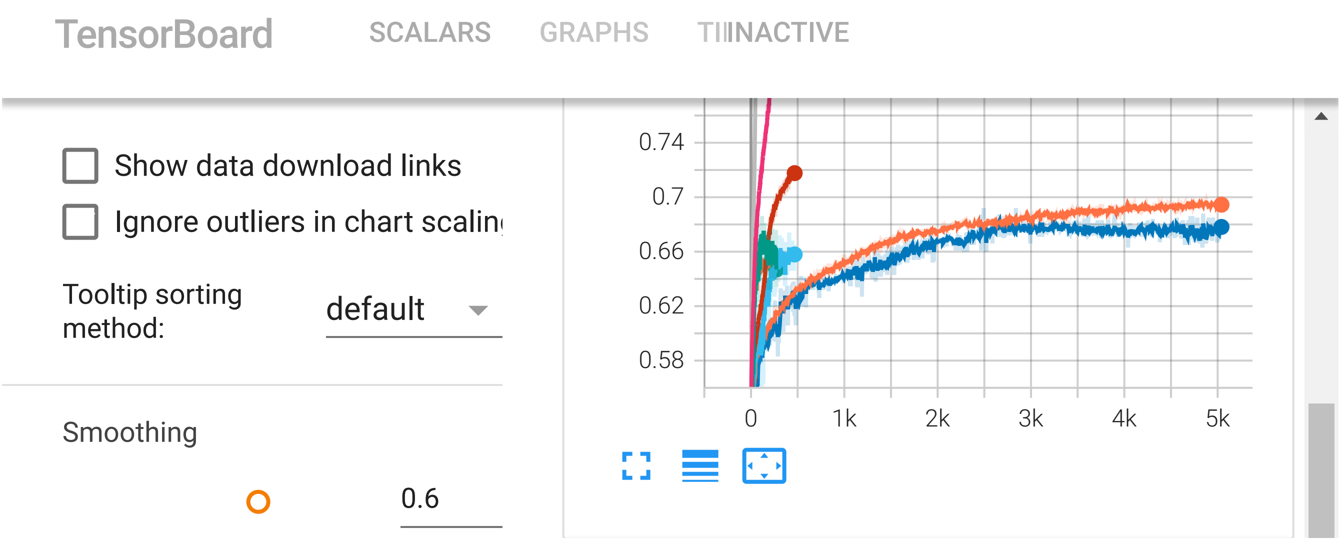


View in TensorBoard

Epochs [Log Scale]

```
#docs_infra: no_execute

# Load the TensorBoard notebook extension
%load_ext tensorboard

# Open an embedded TensorBoard viewer
%tensorboard --logdir {logdir}/sizes
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```




```
#tensorboard dev upload --logdir  {logdir}/sizes
```

```
shutil.rmtree(logdir/'regularizers/Tiny', ignore_errors=True)
shutil.copytree(logdir/'sizes/Tiny', logdir/'regularizers/Tiny')
```

```
PosixPath('/tmp/tmpoc2olwle/tensorboard_logs/regularizers/Tiny')
```

```
regularizer_histories = {}
regularizer_histories['Tiny'] = size_histories['Tiny']
```

Add weight regularization

L1 regularization, where the cost added is proportional to the absolute value of the weights coefficients (i.e. to what is called the "L1 norm" of the weights).

L2 regularization, where the cost added is proportional to the square of the value of the weights coefficients (i.e. to what is called the squared "L2 norm" of the weights). L2 regularization is also called weight decay in the context of neural networks. Don't let the different name confuse you: weight decay is mathematically the exact same as L2 regularization.

L1 regularization pushes weights towards exactly zero, encouraging a sparse model. L2 regularization will penalize the weights parameters without making them sparse since the penalty goes to zero for small weights—one reason why L2 is more common.

In tf.keras, weight regularization is added by passing weight regularizer instances to layers as keyword arguments. Add L2 weight regularization:

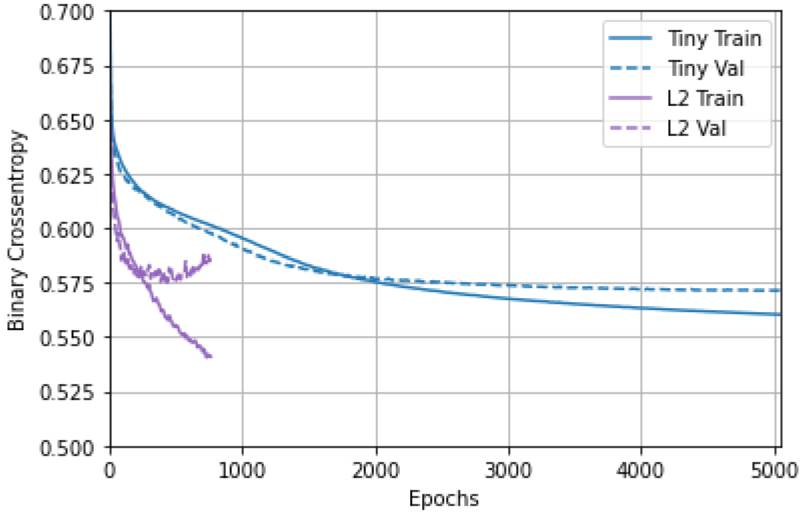
```
l2_model = tf.keras.Sequential([
```

```
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001),
                 input_shape=(FEATURES,)),
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1)
])

regularizer_histories['l2'] = compile_and_fit(l2_model, "regularizers/l2")
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_33 (Dense)            (None, 512)               14848

 dense_34 (Dense)            (None, 512)               262656

 dense_35 (Dense)            (None, 512)               262656

 dense_36 (Dense)            (None, 512)               262656

 dense_37 (Dense)            (None, 1)                 513

=================================================================
Total params: 803,329
Trainable params: 803,329
Non-trainable params: 0
_________________________________________________________________

Epoch: 0, accuracy:0.5080,  binary_crossentropy:0.7454,  loss:2.2338,  val_accuracy:0.4
....................................................................................
Epoch: 100, accuracy:0.6524,  binary_crossentropy:0.5976,  loss:0.6200,  val_accuracy:0
....................................................................................
Epoch: 200, accuracy:0.6715,  binary_crossentropy:0.5849,  loss:0.6074,  val_accuracy:0
....................................................................................
Epoch: 300, accuracy:0.6794,  binary_crossentropy:0.5823,  loss:0.6087,  val_accuracy:0
....................................................................................
Epoch: 400, accuracy:0.6945,  binary_crossentropy:0.5616,  loss:0.5878,  val_accuracy:0
....................................................................................
Epoch: 500, accuracy:0.6944,  binary_crossentropy:0.5574,  loss:0.5842,  val_accuracy:0
....................................................................................
Epoch: 600, accuracy:0.7053,  binary_crossentropy:0.5484,  loss:0.5758,  val_accuracy:0
....................................................................................
Epoch: 700, accuracy:0.7098,  binary_crossentropy:0.5411,  loss:0.5687,  val_accuracy:0
.................................................................
```

```
plotter.plot(regularizer_histories)
plt.ylim([0.5, 0.7])
```

```
(0.5, 0.7)
```



Tiny Train
Tiny Val
L2 Train
L2 Val
Binary Crossentropy
Epochs
0.700
0.675
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0
1000
2000
3000
4000
5000