Notebook found here:

<https://colab.research.google.com/drive/1akgDG7Etnf-VFbh2v2dpMLTQfNtfb2zY?usp=sharing>

## Define the learning rate scheduler

lr\_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 \* 10\*\*(epoch/20))

* Define the scheduler as such. It takes the current epoch and will use that to adjust the current learning rate
* e here doesn’t mean epsilon. It means 10 to the power of something.
* 1e-8 means 10-8
* In this case, the learning rate will be 10-8 x 10(Epoch/20)

## Plot Graph of loss over epoch

#Plot graph with log scaling

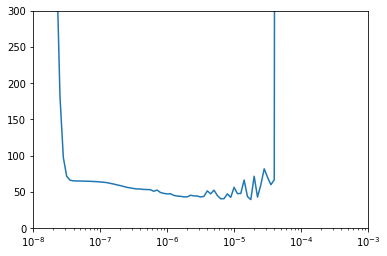
plt.semilogx(history.history['lr'] ,history.history['loss'])

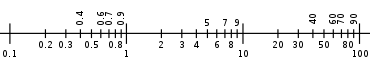
# [xmin, xmax, ymin, ymax]

plt.axis([1e-8, 1e-3, 0, 300])

* History.history[‘lr’] is the learning rate of the model
* Use semilogx which means that the x axis will be log scaled (This allows for a huge range to be represented in less space)
* Define the axis, with the xmin value being the lr where epoch = 0 and xmax being the lr where epoch = 100

## Getting the Optimum Learning Rate



* The x axis is not linear (The spacing is not equal). The distance between 1,2,3,4 are all diff (Same goes to the other values)
* Only numbers with an increase in power have equal spacing
* 10-7 ,10-8 and 10-9 have equal spacing
* The loss seems to be the lowest at 8 x 10-6 . This can be written as 8e-6

## Define the optimizer using the new learning rate

optimizer = tf.keras.optimizers.SGD(lr=8e-6, momentum=0.9)

* Knowing that the learning rate 8e-6 gave the lowest loss, use it to train the model.