**YOU CAN FIND GOOGLE COLAB HERE:**

<https://colab.research.google.com/drive/1MzDb5oSDE4o4RsSedCZVDSYLghN7xRII>

#Just using imports from colab tutorial

from keras.preprocessing import sequence

from keras.models import Sequential

from keras.layers import Dense, Embedding, Dropout, Flatten

from keras.layers import LSTM, CuDNNLSTM

from keras.datasets import imdb

from distutils.version import LooseVersion as LV

from keras import \_\_version\_\_

from keras import backend as K

from IPython.display import SVG

from keras.utils.vis\_utils import model\_to\_dot

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

print('Using Keras version:', \_\_version\_\_, 'backend:', K.backend())

assert(LV(\_\_version\_\_) >= LV("2.0.0"))

Using TensorFlow backend.

Using Keras version: 2.2.4 backend: tensorflow

# using tutorial code to just grab the data and shape it

# determines how many words to sample

number\_words = 10000

max\_len = 80

print('Loading data...')

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=number\_words)

print('x\_train:', x\_train.shape)

print('x\_test:', x\_test.shape)

print()

x\_train = sequence.pad\_sequences(x\_train, maxlen=max\_len)

x\_test = sequence.pad\_sequences(x\_test, maxlen=max\_len)

print('x\_train shape:', x\_train.shape)

print('x\_test shape:', x\_test.shape)

Loading data...

x\_train: (25000,)

x\_test: (25000,)

x\_train shape: (25000, 80)

x\_test shape: (25000, 80)

#create base model object, build it, and look at summary

base\_model = Sequential()

base\_model.add(Embedding(10000, 8, input\_length=max\_len)) #using the embedding layer to make word embeddings instead of vectorizing the words

base\_model.add(Flatten())

base\_model.add(Dense(1, activation='sigmoid'))

base\_model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

base\_model.summary()

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op\_def\_library.py:263: colocate\_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.

Instructions for updating:

Colocations handled automatically by placer.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_1 (Embedding) (None, 20, 8) 80000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 160) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 1) 161

=================================================================

Total params: 80,161

Trainable params: 80,161

Non-trainable params: 0

#run it!

base\_history = base\_model.fit(x\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math\_ops.py:3066: to\_int32 (from tensorflow.python.ops.math\_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

Train on 20000 samples, validate on 5000 samples

Epoch 1/10

20000/20000 [==============================] - 5s 250us/step - loss: 0.6759 - acc: 0.6050 - val\_loss: 0.6398 - val\_acc: 0.6814

Epoch 2/10

20000/20000 [==============================] - 3s 128us/step - loss: 0.5657 - acc: 0.7427 - val\_loss: 0.5467 - val\_acc: 0.7206

Epoch 3/10

20000/20000 [==============================] - 3s 127us/step - loss: 0.4752 - acc: 0.7808 - val\_loss: 0.5113 - val\_acc: 0.7384

Epoch 4/10

20000/20000 [==============================] - 3s 127us/step - loss: 0.4263 - acc: 0.8077 - val\_loss: 0.5008 - val\_acc: 0.7452

Epoch 5/10

20000/20000 [==============================] - 3s 131us/step - loss: 0.3930 - acc: 0.8258 - val\_loss: 0.4981 - val\_acc: 0.7538

Epoch 6/10

20000/20000 [==============================] - 3s 126us/step - loss: 0.3668 - acc: 0.8395 - val\_loss: 0.5014 - val\_acc: 0.7530

Epoch 7/10

20000/20000 [==============================] - 3s 128us/step - loss: 0.3435 - acc: 0.8533 - val\_loss: 0.5052 - val\_acc: 0.7520

Epoch 8/10

20000/20000 [==============================] - 3s 127us/step - loss: 0.3223 - acc: 0.8657 - val\_loss: 0.5132 - val\_acc: 0.7486

Epoch 9/10

20000/20000 [==============================] - 3s 126us/step - loss: 0.3022 - acc: 0.8766 - val\_loss: 0.5213 - val\_acc: 0.7490

Epoch 10/10

20000/20000 [==============================] - 3s 127us/step - loss: 0.2839 - acc: 0.8860 - val\_loss: 0.5303 - val\_acc: 0.7466

We ended up getting validation accuracy of .74 just like in the book! But now its time to increase performance by using a RNN instead of a Feed-forward network.

#update max\_len for the RNN example

max\_len = 80

embedding\_dims = 50

lstm\_units = 32

#recurrent imdb model

r\_i\_model = Sequential()

r\_i\_model.add(Embedding(number\_words, embedding\_dims, input\_length=max\_len))

r\_i\_model.add(Dropout(0.2)) #stop overfitting NOW!

r\_i\_model.add(LSTM(lstm\_units)) #this is what makes it an RNN

r\_i\_model.add(Dense(1, activation='sigmoid'))

r\_i\_model.compile(loss='binary\_crossentropy',optimizer='rmsprop',metrics=['accuracy'])

print(r\_i\_model.summary())

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow\_backend.py:3445: calling dropout (from tensorflow.python.ops.nn\_ops) with keep\_prob is deprecated and will be removed in a future version.

Instructions for updating:

Please use `rate` instead of `keep\_prob`. Rate should be set to `rate = 1 - keep\_prob`.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_2 (Embedding) (None, 80, 50) 500000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 80, 50) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_1 (LSTM) (None, 32) 10624

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 33

=================================================================

Total params: 510,657

Trainable params: 510,657

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

#LETS GO

epochs = 5

validation\_split = 0.2

r\_i\_history = r\_i\_model.fit(x\_train, y\_train, batch\_size=128,epochs=epochs,validation\_split=validation\_split)

Train on 20000 samples, validate on 5000 samples

Epoch 1/5

20000/20000 [==============================] - 29s 1ms/step - loss: 0.5158 - acc: 0.7449 - val\_loss: 0.4521 - val\_acc: 0.7898

Epoch 2/5

20000/20000 [==============================] - 28s 1ms/step - loss: 0.3294 - acc: 0.8621 - val\_loss: 0.3521 - val\_acc: 0.8420

Epoch 3/5

20000/20000 [==============================] - 29s 1ms/step - loss: 0.2766 - acc: 0.8885 - val\_loss: 0.3765 - val\_acc: 0.8308

Epoch 4/5

20000/20000 [==============================] - 29s 1ms/step - loss: 0.2500 - acc: 0.8996 - val\_loss: 0.3558 - val\_acc: 0.8424

Epoch 5/5

20000/20000 [==============================] - 28s 1ms/step - loss: 0.2298 - acc: 0.9095 - val\_loss: 0.3885 - val\_acc: 0.8438

plt.figure(figsize=(5,3))

plt.plot(r\_i\_history.epoch,r\_i\_history.history['loss'], label='training')

plt.plot(r\_i\_history.epoch,r\_i\_history.history['val\_loss'], label='validation')

plt.title('loss')

plt.legend(loc='best')

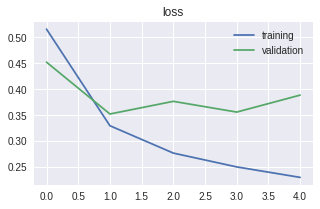
plt.figure(figsize=(5,3))

plt.plot(r\_i\_history.epoch,r\_i\_history.history['acc'], label='training')

plt.plot(r\_i\_history.epoch,r\_i\_history.history['val\_acc'], label='validation')

plt.title('accuracy')

plt.legend(loc='best');



A close up of a piece of paper

Description automatically generated

#stacking LSTM layers for the fun of it

r\_model2 = Sequential()

r\_model2.add(Embedding(number\_words, embedding\_dims, input\_length=max\_len))

r\_model2.add(Dropout(0.2))

r\_model2.add(LSTM(lstm\_units, return\_sequences=True))

r\_model2.add(LSTM(lstm\_units)) #and here... we... go.

r\_model2.add(Dense(1, activation='sigmoid'))

r\_model2.compile(loss='binary\_crossentropy',optimizer='rmsprop',metrics=['accuracy'])

print(r\_model2.summary())

#even with TPU acceleration this takes a while

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_3 (Embedding) (None, 80, 50) 500000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 80, 50) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_2 (LSTM) (None, 80, 32) 10624

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_3 (LSTM) (None, 32) 8320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 1) 33

=================================================================

Total params: 518,977

Trainable params: 518,977

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

#training the stacked RNN

r\_history2 = r\_model2.fit(x\_train, y\_train, batch\_size=128,epochs=epochs,validation\_split=validation\_split)

Train on 20000 samples, validate on 5000 samples

Epoch 1/5

20000/20000 [==============================] - 54s 3ms/step - loss: 0.4868 - acc: 0.7552 - val\_loss: 0.4297 - val\_acc: 0.7994

Epoch 2/5

20000/20000 [==============================] - 53s 3ms/step - loss: 0.3289 - acc: 0.8636 - val\_loss: 0.3665 - val\_acc: 0.8360

Epoch 3/5

20000/20000 [==============================] - 53s 3ms/step - loss: 0.2792 - acc: 0.8877 - val\_loss: 0.3591 - val\_acc: 0.8420

Epoch 4/5

20000/20000 [==============================] - 54s 3ms/step - loss: 0.2484 - acc: 0.9018 - val\_loss: 0.3657 - val\_acc: 0.8354

Epoch 5/5

20000/20000 [==============================] - 53s 3ms/step - loss: 0.2285 - acc: 0.9116 - val\_loss: 0.3889 - val\_acc: 0.8366

Looks like stacking LSTM layers didn't have a very big effect on performance...

plt.clf()

plt.figure(figsize=(5,3))

plt.plot(r\_history2.epoch,r\_history2.history['loss'], label='training')

plt.plot(r\_history2.epoch,r\_history2.history['val\_loss'], label='validation')

plt.title('loss')

plt.legend(loc='best')

plt.figure(figsize=(5,3))

plt.plot(r\_history2.epoch,r\_history2.history['acc'], label='training')

plt.plot(r\_history2.epoch,r\_history2.history['val\_acc'], label='validation')

plt.title('accuracy')

plt.legend(loc='best');

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

#Moving onward to the JENA climate dataset! It took a while to get this to work,

#but I eventually had to settle on using my google drive instead of uploading the

#file itself. If i uploaded the file, it would eventually go through, but whenever

#I would go to open it, the runtime would crash. Using google drive seems to alleviate

#that issue

from google.colab import drive

drive.mount('/content/gdrive')

#need to have file locally downloaded... this takes a while!

f = open('/content/gdrive/My Drive/jena\_climate\_2009\_2016.csv')

data = f.read()

f.close()

lines = data.split('\n')

header = lines[0].split(',')

lines = lines[1:]

print(header)

print(len(lines))

['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh (%)"', '"VPmax (mbar)"', '"VPact (mbar)"', '"VPdef (mbar)"', '"sh (g/kg)"', '"H2OC (mmol/mol)"', '"rho (g/m\*\*3)"', '"wv (m/s)"', '"max. wv (m/s)"', '"wd (deg)"']

420551

import numpy as np

#create those 420000 or so lines into a numpy array

float\_data = np.zeros((len(lines), len(header) - 1))

for i, line in enumerate(lines):

values = [float(x) for x in line.split(',')[1:]]

float\_data[i, :] = values

#this cell normalizes (standardizes?) the data and creates variables

lookback = 720 # 5 days

steps = 6 # sampling 1 point per hour

delay = 144 # predicting 24 hours in the future

mean = float\_data[:200000].mean(axis=0)

float\_data -= mean

std = float\_data[:200000].std(axis=0)

float\_data /= std

My lack of compsci knowledge is showing in the next block. There are a few keywords that I don't know what they do, like yield. Apparently its like return, only it returns a generator instead of whatever it is that return returns. The rest of it seems to make sense, but I don't have the programming chops to write something like this from scratch. As an aside, I've never used numpy.randint before (this is my first time seeing it) but its pretty easy to guess what it does.

def generator(data, lookback, delay, min\_index, max\_index,

shuffle=False, batch\_size=128, step=6):

if max\_index is None:

max\_index = len(data) - delay - 1

i = min\_index + lookback

while 1:

if shuffle:

rows = np.random.randint(

min\_index + lookback, max\_index, size=batch\_size)

else:

if i + batch\_size >= max\_index:

i = min\_index + lookback

rows = np.arange(i, min(i + batch\_size, max\_index))

i += len(rows)

samples = np.zeros((len(rows),

lookback // step,

data.shape[-1]))

targets = np.zeros((len(rows),))

for j, row in enumerate(rows):

indices = range(rows[j] - lookback, rows[j], step)

samples[j] = data[indices]

targets[j] = data[rows[j] + delay][1]

yield samples, targets

#this block is essentially our train/validation/test split.

lookback = 1440

step = 6

delay = 144

batch\_size = 128

train\_gen = generator(float\_data,lookback=lookback,delay=delay,min\_index=0,max\_index=200000,shuffle=True,step=step,batch\_size=batch\_size)

val\_gen = generator(float\_data,lookback=lookback,delay=delay,min\_index=200001,max\_index=300000,step=step,batch\_size=batch\_size)

test\_gen = generator(float\_data,lookback=lookback,delay=delay,min\_index=300001,max\_index=None,step=step,batch\_size=batch\_size)

val\_steps = (300000 - 200001 - lookback) // batch\_size

test\_steps = (len(float\_data) - 300001 - lookback) // batch\_size

#adding baseline because its there so why not

def evaluate\_naive\_method():

batch\_maes = []

for step in range(val\_steps):

samples, targets = next(val\_gen)

preds = samples[:, -1, 1]

mae = np.mean(np.abs(preds - targets))

batch\_maes.append(mae)

print(np.mean(batch\_maes))

evaluate\_naive\_method()

0.28973597299053566

#simple machine learning approach

from keras.models import Sequential

from keras import layers

from keras.optimizers import RMSprop

model = Sequential()

model.add(layers.Flatten(input\_shape=(lookback // step, float\_data.shape[-1])))

model.add(layers.Dense(32, activation='relu'))

model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit\_generator(train\_gen,steps\_per\_epoch=500,epochs=20,validation\_data=val\_gen, validation\_steps=val\_steps)

Epoch 1/20

500/500 [==============================] - 14s 28ms/step - loss: 1.4619 - val\_loss: 0.6840

Epoch 2/20

500/500 [==============================] - 14s 28ms/step - loss: 0.4792 - val\_loss: 0.4415

Epoch 3/20

500/500 [==============================] - 14s 28ms/step - loss: 0.3071 - val\_loss: 0.3026

Epoch 4/20

500/500 [==============================] - 14s 27ms/step - loss: 0.2702 - val\_loss: 0.3893

Epoch 5/20

500/500 [==============================] - 14s 27ms/step - loss: 0.2563 - val\_loss: 0.3201

Epoch 6/20

500/500 [==============================] - 14s 27ms/step - loss: 0.2470 - val\_loss: 0.3234

Epoch 7/20

500/500 [==============================] - 14s 27ms/step - loss: 0.2409 - val\_loss: 0.3135

Epoch 8/20

500/500 [==============================] - 14s 27ms/step - loss: 0.2336 - val\_loss: 0.3218

Epoch 9/20

500/500 [==============================] - 13s 27ms/step - loss: 0.2283 - val\_loss: 0.3385

Epoch 10/20

500/500 [==============================] - 14s 27ms/step - loss: 0.2244 - val\_loss: 0.3215

Epoch 11/20

194/500 [==========>...................] - ETA: 4s - loss: 0.2229

---------------------------------------------------------------------------

KeyboardInterrupt Traceback (most recent call last)

I stopped the last model because it was taking a while and looked like it started overfitting very early

#now we are doing our first recurrent version

model = Sequential()

model.add(layers.GRU(32, input\_shape=(None, float\_data.shape[-1])))

model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit\_generator(train\_gen, steps\_per\_epoch=500,epochs=6,validation\_data=val\_gen,validation\_steps=val\_steps)

Epoch 1/6

500/500 [==============================] - 203s 406ms/step - loss: 0.3052 - val\_loss: 0.2703

Epoch 2/6

500/500 [==============================] - 200s 399ms/step - loss: 0.2861 - val\_loss: 0.2737

Epoch 3/6

500/500 [==============================] - 200s 400ms/step - loss: 0.2807 - val\_loss: 0.2647

Epoch 4/6

500/500 [==============================] - 199s 398ms/step - loss: 0.2746 - val\_loss: 0.2706

Epoch 5/6

500/500 [==============================] - 199s 398ms/step - loss: 0.2693 - val\_loss: 0.2670

Epoch 6/6

500/500 [==============================] - 198s 397ms/step - loss: 0.2623 - val\_loss: 0.2701

Wow. Even a mere 6 epochs took like half an hour to train. The 199s measurement of time that it took is a lie, it sits on 0 seconds 499/500 steps for at least a good 30 seconds before it goes on to the next epoch.

#setting up a plot of training vs validation loss

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

A picture containing indoor, photo, sky

Description automatically generated

You can see it starts to overfit after 6 epochs.

#A lot of epochs, a lot of time, and recurrent dropout.

model = Sequential()

model.add(layers.GRU(32,dropout=0.2,recurrent\_dropout=0.2,input\_shape=(None, float\_data.shape[-1])))

model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit\_generator(train\_gen,steps\_per\_epoch=500,epochs=40,validation\_data=val\_gen,validation\_steps=val\_steps)

Epoch 1/40

500/500 [==============================] - 229s 459ms/step - loss: 0.3394 - val\_loss: 0.2741

Epoch 2/40

500/500 [==============================] - 226s 452ms/step - loss: 0.3143 - val\_loss: 0.2715

Epoch 3/40

500/500 [==============================] - 225s 451ms/step - loss: 0.3101 - val\_loss: 0.2734

Epoch 4/40

500/500 [==============================] - 225s 450ms/step - loss: 0.3046 - val\_loss: 0.2728

Epoch 5/40

500/500 [==============================] - 225s 450ms/step - loss: 0.3029 - val\_loss: 0.2665

Epoch 6/40

500/500 [==============================] - 225s 450ms/step - loss: 0.2996 - val\_loss: 0.2641

Epoch 7/40

500/500 [==============================] - 225s 449ms/step - loss: 0.2974 - val\_loss: 0.2691

Epoch 8/40

500/500 [==============================] - 224s 449ms/step - loss: 0.2957 - val\_loss: 0.2657

Epoch 9/40

500/500 [==============================] - 224s 448ms/step - loss: 0.2943 - val\_loss: 0.2651

Epoch 10/40

500/500 [==============================] - 222s 445ms/step - loss: 0.2932 - val\_loss: 0.2640

Epoch 11/40

500/500 [==============================] - 222s 445ms/step - loss: 0.2890 - val\_loss: 0.2702

Epoch 12/40

500/500 [==============================] - 222s 444ms/step - loss: 0.2902 - val\_loss: 0.2637

Epoch 13/40

500/500 [==============================] - 225s 450ms/step - loss: 0.2883 - val\_loss: 0.2677

Epoch 14/40

500/500 [==============================] - 227s 454ms/step - loss: 0.2873 - val\_loss: 0.2619

Epoch 15/40

500/500 [==============================] - 227s 453ms/step - loss: 0.2842 - val\_loss: 0.2699

Epoch 16/40

500/500 [==============================] - 226s 452ms/step - loss: 0.2843 - val\_loss: 0.2606

Epoch 17/40

500/500 [==============================] - 227s 455ms/step - loss: 0.2837 - val\_loss: 0.2639

Epoch 18/40

500/500 [==============================] - 227s 453ms/step - loss: 0.2830 - val\_loss: 0.2621

Epoch 19/40

500/500 [==============================] - 227s 454ms/step - loss: 0.2816 - val\_loss: 0.2616

Epoch 20/40

500/500 [==============================] - 227s 454ms/step - loss: 0.2814 - val\_loss: 0.2640

Epoch 21/40

500/500 [==============================] - 230s 460ms/step - loss: 0.2802 - val\_loss: 0.2645

Epoch 22/40

500/500 [==============================] - 228s 456ms/step - loss: 0.2788 - val\_loss: 0.2644

Epoch 23/40

500/500 [==============================] - 227s 455ms/step - loss: 0.2785 - val\_loss: 0.2653

Epoch 24/40

500/500 [==============================] - 230s 460ms/step - loss: 0.2779 - val\_loss: 0.2615

Epoch 25/40

500/500 [==============================] - 229s 458ms/step - loss: 0.2756 - val\_loss: 0.2639

Epoch 26/40

500/500 [==============================] - 229s 458ms/step - loss: 0.2771 - val\_loss: 0.2684

Epoch 27/40

500/500 [==============================] - 229s 457ms/step - loss: 0.2758 - val\_loss: 0.2662

Epoch 28/40

500/500 [==============================] - 229s 458ms/step - loss: 0.2741 - val\_loss: 0.2653

Epoch 29/40

500/500 [==============================] - 229s 458ms/step - loss: 0.2750 - val\_loss: 0.2667

Epoch 30/40

500/500 [==============================] - 228s 457ms/step - loss: 0.2736 - val\_loss: 0.2666

Epoch 31/40

500/500 [==============================] - 228s 457ms/step - loss: 0.2730 - val\_loss: 0.2608

Epoch 32/40

500/500 [==============================] - 228s 456ms/step - loss: 0.2715 - val\_loss: 0.2671

Epoch 33/40

500/500 [==============================] - 228s 457ms/step - loss: 0.2718 - val\_loss: 0.2701

Epoch 34/40

500/500 [==============================] - 229s 458ms/step - loss: 0.2699 - val\_loss: 0.2625

Epoch 35/40

500/500 [==============================] - 228s 457ms/step - loss: 0.2700 - val\_loss: 0.2627

Epoch 36/40

500/500 [==============================] - 229s 458ms/step - loss: 0.2713 - val\_loss: 0.2670

Epoch 37/40

500/500 [==============================] - 230s 459ms/step - loss: 0.2693 - val\_loss: 0.2636

Epoch 38/40

500/500 [==============================] - 229s 457ms/step - loss: 0.2693 - val\_loss: 0.2627

Epoch 39/40

500/500 [==============================] - 229s 458ms/step - loss: 0.2687 - val\_loss: 0.2666

Epoch 40/40

500/500 [==============================] - 228s 456ms/step - loss: 0.2686 - val\_loss: 0.2651

I was using a GPU instance instead of a TPU instance, but it still took like 3 hours...

#another train v val loss plot

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

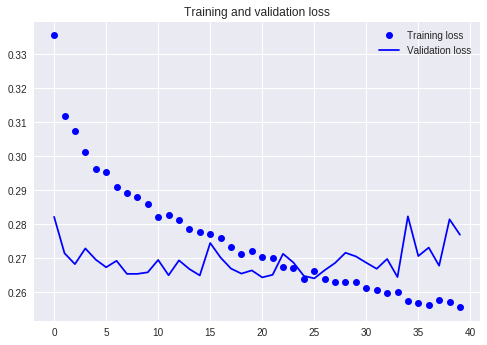
plt.show()

#Yep, doesn't look like overfitting is as much of a problem. Global minumum looks

#like 15 epochs though.

model.summary() #the summary was added and run after the next block.

#It was taking forever and wanted to double check if I was using the correct architecture.



#New instance, re import

from keras.models import Sequential

from keras import layers

from keras.optimizers import RMSprop

#studying the effects of stacking recurrent layers firsthand... again.

model = Sequential()

model.add(layers.GRU(32,dropout=0.1,recurrent\_dropout=0.5,return\_sequences=True,input\_shape=(None, float\_data.shape[-1])))

model.add(layers.GRU(64, activation='relu',dropout=0.1,recurrent\_dropout=0.5))

model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit\_generator(train\_gen,steps\_per\_epoch=500,epochs=40,validation\_data=val\_gen,validation\_steps=val\_steps)

#had to

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op\_def\_library.py:263: colocate\_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.

Instructions for updating:

Colocations handled automatically by placer.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow\_backend.py:3445: calling dropout (from tensorflow.python.ops.nn\_ops) with keep\_prob is deprecated and will be removed in a future version.

Instructions for updating:

Please use `rate` instead of `keep\_prob`. Rate should be set to `rate = 1 - keep\_prob`.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math\_ops.py:3066: to\_int32 (from tensorflow.python.ops.math\_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

Epoch 1/40

500/500 [==============================] - 362s 724ms/step - loss: 0.3330 - val\_loss: 0.2752

Epoch 2/40

500/500 [==============================] - 359s 718ms/step - loss: 0.3120 - val\_loss: 0.2719

Epoch 3/40

500/500 [==============================] - 359s 718ms/step - loss: 0.3055 - val\_loss: 0.2689

Epoch 4/40

500/500 [==============================] - 360s 719ms/step - loss: 0.3019 - val\_loss: 0.2715

Epoch 5/40

500/500 [==============================] - 359s 718ms/step - loss: 0.2978 - val\_loss: 0.2681

Epoch 6/40

500/500 [==============================] - 359s 718ms/step - loss: 0.2952 - val\_loss: 0.2687

Epoch 7/40

500/500 [==============================] - 359s 719ms/step - loss: 0.2931 - val\_loss: 0.2701

Epoch 8/40

500/500 [==============================] - 360s 719ms/step - loss: 0.2897 - val\_loss: 0.2645

Epoch 9/40

500/500 [==============================] - 357s 715ms/step - loss: 0.2865 - val\_loss: 0.2615

Epoch 10/40

500/500 [==============================] - 358s 717ms/step - loss: 0.2862 - val\_loss: 0.2631

Epoch 11/40

500/500 [==============================] - 358s 715ms/step - loss: 0.2813 - val\_loss: 0.2669

Epoch 12/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2807 - val\_loss: 0.2662

Epoch 13/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2789 - val\_loss: 0.2738

Epoch 14/40

500/500 [==============================] - 359s 719ms/step - loss: 0.2750 - val\_loss: 0.2647

Epoch 15/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2752 - val\_loss: 0.2643

Epoch 16/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2732 - val\_loss: 0.2664

Epoch 17/40

500/500 [==============================] - 358s 717ms/step - loss: 0.2728 - val\_loss: 0.2712

Epoch 18/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2706 - val\_loss: 0.2653

Epoch 19/40

500/500 [==============================] - 357s 714ms/step - loss: 0.2696 - val\_loss: 0.2633

Epoch 20/40

500/500 [==============================] - 358s 715ms/step - loss: 0.2676 - val\_loss: 0.2692

Epoch 21/40

500/500 [==============================] - 359s 719ms/step - loss: 0.2671 - val\_loss: 0.2675

Epoch 22/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2655 - val\_loss: 0.2629

Epoch 23/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2656 - val\_loss: 0.2632

Epoch 24/40

500/500 [==============================] - 357s 715ms/step - loss: 0.2634 - val\_loss: 0.2702

Epoch 25/40

500/500 [==============================] - 359s 719ms/step - loss: 0.2639 - val\_loss: 0.2769

Epoch 26/40

500/500 [==============================] - 360s 720ms/step - loss: 0.2614 - val\_loss: 0.2661

Epoch 27/40

500/500 [==============================] - 359s 718ms/step - loss: 0.2616 - val\_loss: 0.2643

Epoch 28/40

500/500 [==============================] - 359s 717ms/step - loss: 0.2610 - val\_loss: 0.2678

Epoch 29/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2597 - val\_loss: 0.2667

Epoch 30/40

500/500 [==============================] - 358s 717ms/step - loss: 0.2587 - val\_loss: 0.2671

Epoch 31/40

500/500 [==============================] - 358s 717ms/step - loss: 0.2572 - val\_loss: 0.2729

Epoch 32/40

500/500 [==============================] - 357s 713ms/step - loss: 0.2574 - val\_loss: 0.2713

Epoch 33/40

500/500 [==============================] - 359s 718ms/step - loss: 0.2571 - val\_loss: 0.2747

Epoch 34/40

500/500 [==============================] - 360s 719ms/step - loss: 0.2558 - val\_loss: 0.2725

Epoch 35/40

500/500 [==============================] - 358s 716ms/step - loss: 0.2544 - val\_loss: 0.2700

Epoch 36/40

500/500 [==============================] - 359s 717ms/step - loss: 0.2536 - val\_loss: 0.2705

Epoch 37/40

500/500 [==============================] - 358s 715ms/step - loss: 0.2537 - val\_loss: 0.2723

Epoch 38/40

500/500 [==============================] - 357s 714ms/step - loss: 0.2522 - val\_loss: 0.2829

Epoch 39/40

500/500 [==============================] - 357s 714ms/step - loss: 0.2513 - val\_loss: 0.2687

Epoch 40/40

500/500 [==============================] - 357s 715ms/step - loss: 0.2520 - val\_loss: 0.2706

#I HAD TO RUN THIS LAST BLOCK TWICE BECAUSE WINDOWS UPDATE

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

#moment of truth after like 8 hours

A screenshot of a cell phone

Description automatically generated

#This is a new instance (computer turned off again), going to reimport rather than run previous cells again

from keras.models import Sequential

from keras import layers

from keras.optimizers import RMSprop

#features again

max\_features = 10000

maxlen = 500

# Need to get the IMDB data back from the beginning, I reload it here.

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

x\_train = sequence.pad\_sequences(x\_train, maxlen=maxlen)

x\_test = sequence.pad\_sequences(x\_test, maxlen=maxlen)

#Create bidirectional RNN to analyze the IMDB dataset

model = Sequential()

model.add(layers.Embedding(max\_features, 32))

model.add(layers.Bidirectional(layers.LSTM(32)))

model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_split=0.2)

Train on 20000 samples, validate on 5000 samples

Epoch 1/10

20000/20000 [==============================] - 155s 8ms/step - loss: 0.5575 - acc: 0.7221 - val\_loss: 0.4312 - val\_acc: 0.8340

Epoch 2/10

20000/20000 [==============================] - 152s 8ms/step - loss: 0.3539 - acc: 0.8667 - val\_loss: 0.3235 - val\_acc: 0.8820

Epoch 3/10

20000/20000 [==============================] - 153s 8ms/step - loss: 0.2873 - acc: 0.8982 - val\_loss: 0.3033 - val\_acc: 0.8806

Epoch 4/10

20000/20000 [==============================] - 152s 8ms/step - loss: 0.2337 - acc: 0.9148 - val\_loss: 0.3405 - val\_acc: 0.8716

Epoch 5/10

20000/20000 [==============================] - 152s 8ms/step - loss: 0.2045 - acc: 0.9271 - val\_loss: 0.3227 - val\_acc: 0.8880

Epoch 6/10

20000/20000 [==============================] - 151s 8ms/step - loss: 0.1813 - acc: 0.9375 - val\_loss: 0.4227 - val\_acc: 0.8508

Epoch 7/10

20000/20000 [==============================] - 152s 8ms/step - loss: 0.1720 - acc: 0.9402 - val\_loss: 0.3461 - val\_acc: 0.8798

Epoch 8/10

20000/20000 [==============================] - 151s 8ms/step - loss: 0.1548 - acc: 0.9463 - val\_loss: 0.5604 - val\_acc: 0.8388

Epoch 9/10

20000/20000 [==============================] - 151s 8ms/step - loss: 0.1421 - acc: 0.9516 - val\_loss: 0.4121 - val\_acc: 0.8552

Epoch 10/10

20000/20000 [==============================] - 151s 8ms/step - loss: 0.1356 - acc: 0.9550 - val\_loss: 0.3769 - val\_acc: 0.8746

10 epochs of 1 bidirectional RNN layer outperforms 5 epochs of stacked RNNs from before!

CODETEXT

model = Sequential()

model.add(layers.Bidirectional(layers.GRU(32), input\_shape=(None, float\_data.shape[-1])))

model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit\_generator(train\_gen,steps\_per\_epoch=500,epochs=40,validation\_data=val\_gen,validation\_steps=val\_steps)

#Applying the bidirectional model to the temperature prediction problem

#Performs slightly worse than the stacked monodirectional RNNS

#Chronological order is therefore important!

Epoch 1/40

500/500 [==============================] - 197s 393ms/step - loss: 0.2996 - val\_loss: 0.2753

Epoch 2/40

500/500 [==============================] - 196s 393ms/step - loss: 0.2758 - val\_loss: 0.2655

Epoch 3/40

500/500 [==============================] - 197s 394ms/step - loss: 0.2686 - val\_loss: 0.2719

Epoch 4/40

500/500 [==============================] - 197s 395ms/step - loss: 0.2612 - val\_loss: 0.2640

Epoch 5/40

500/500 [==============================] - 195s 389ms/step - loss: 0.2563 - val\_loss: 0.2679

Epoch 6/40

500/500 [==============================] - 194s 388ms/step - loss: 0.2502 - val\_loss: 0.2692

Epoch 7/40

500/500 [==============================] - 193s 387ms/step - loss: 0.2450 - val\_loss: 0.2837

Epoch 8/40

500/500 [==============================] - 194s 387ms/step - loss: 0.2393 - val\_loss: 0.2746

Epoch 9/40

500/500 [==============================] - 192s 384ms/step - loss: 0.2324 - val\_loss: 0.2796

Epoch 10/40

500/500 [==============================] - 193s 386ms/step - loss: 0.2252 - val\_loss: 0.2837

Epoch 11/40

500/500 [==============================] - 193s 387ms/step - loss: 0.2193 - val\_loss: 0.2826

Epoch 12/40

500/500 [==============================] - 198s 396ms/step - loss: 0.2147 - val\_loss: 0.2902

Epoch 13/40

500/500 [==============================] - 198s 396ms/step - loss: 0.2086 - val\_loss: 0.2927

Epoch 14/40

500/500 [==============================] - 198s 395ms/step - loss: 0.2039 - val\_loss: 0.2954

Epoch 15/40

500/500 [==============================] - 198s 395ms/step - loss: 0.1993 - val\_loss: 0.2958

Epoch 16/40

500/500 [==============================] - 198s 397ms/step - loss: 0.1937 - val\_loss: 0.3051

Epoch 17/40

500/500 [==============================] - 199s 397ms/step - loss: 0.1897 - val\_loss: 0.3037

Epoch 18/40

500/500 [==============================] - 200s 399ms/step - loss: 0.1879 - val\_loss: 0.3085

Epoch 19/40

500/500 [==============================] - 198s 395ms/step - loss: 0.1828 - val\_loss: 0.3034

Epoch 20/40

500/500 [==============================] - 198s 396ms/step - loss: 0.1800 - val\_loss: 0.3103

Epoch 21/40

500/500 [==============================] - 201s 402ms/step - loss: 0.1768 - val\_loss: 0.3071

Epoch 22/40

500/500 [==============================] - 200s 399ms/step - loss: 0.1729 - val\_loss: 0.3190

Epoch 23/40

500/500 [==============================] - 199s 399ms/step - loss: 0.1702 - val\_loss: 0.3175

Epoch 24/40

500/500 [==============================] - 200s 400ms/step - loss: 0.1681 - val\_loss: 0.3161

Epoch 25/40

500/500 [==============================] - 200s 400ms/step - loss: 0.1654 - val\_loss: 0.3176

Epoch 26/40

500/500 [==============================] - 200s 400ms/step - loss: 0.1620 - val\_loss: 0.3235

Epoch 27/40

500/500 [==============================] - 196s 392ms/step - loss: 0.1605 - val\_loss: 0.3202

Epoch 28/40

500/500 [==============================] - 194s 388ms/step - loss: 0.1582 - val\_loss: 0.3158

Epoch 29/40

500/500 [==============================] - 195s 390ms/step - loss: 0.1564 - val\_loss: 0.3197

Epoch 30/40

500/500 [==============================] - 194s 389ms/step - loss: 0.1540 - val\_loss: 0.3198

Epoch 31/40

500/500 [==============================] - 195s 390ms/step - loss: 0.1518 - val\_loss: 0.3273

Epoch 32/40

500/500 [==============================] - 198s 396ms/step - loss: 0.1502 - val\_loss: 0.3241

Epoch 33/40

500/500 [==============================] - 198s 397ms/step - loss: 0.1489 - val\_loss: 0.3197

Epoch 34/40

500/500 [==============================] - 199s 398ms/step - loss: 0.1467 - val\_loss: 0.3203

Epoch 35/40

500/500 [==============================] - 197s 394ms/step - loss: 0.1456 - val\_loss: 0.3277

Epoch 36/40

500/500 [==============================] - 196s 393ms/step - loss: 0.1441 - val\_loss: 0.3248

Epoch 37/40

500/500 [==============================] - 196s 392ms/step - loss: 0.1424 - val\_loss: 0.3309

Epoch 38/40

499/500 [============================>.] - ETA: 0s - loss: 0.1413

I don’t know why it stopped at 38 epochs, I didn’t do a keyboard interrupt. Maybe it lost connection to the colab kernel.