In [1]:
```
import numpy as np
import pandas as pd
import os
import re

#from kinnara.gather.twitter import TwitterApiWrapper
```

In [2]:
```
import logging
logging.basicConfig(format='%(asctime)s - %(levelname)s - %(message)s',
        level=logging.INFO)
logger = logging.getLogger(__name__)
```

read in api keys and access tokens

In [3]:
```
api_key = os.getenv('API_KEY')
api_secret = os.getenv('API_SECRET')

access_token = os.getenv('ACCESS_TOKEN')
access_token_secret = os.getenv('ACCESS_TOKEN_SECRET')
```

Gather public figure tweets

In [4]:
```
# create kinnara gatherer
twitter_wrapper = TwitterApiWrapper(api_key=api_key, api_secret=api_secret,
                        access_token=access_token, access_token_secret=access_
token_secret)
```

In [5]:
```
screen_names = [
    'BarackObama',
    'realDonaldTrump',
    'KimKardashian',
    'BillGates',
    'Oprah',
    'justinbieber',
    'TheRock',
    'elonmusk',
    'JeffBezos',
    'katyperry'
]
```

In [6]:
```
users = []
for screen_name in screen_names:
    users.append(twitter_wrapper.get_user(screen_name))
```

```
2018-08-25 18:02:23,745 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:23,927 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:24,103 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:24,213 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:24,359 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:24,502 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:24,645 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:24,829 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:24,960 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:25,137 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
```

In [7]:
```
# lets grab 400 tweets from each twitter users timeline
for user in users:
    user['tweets'] = twitter_wrapper.get_tweets(user['id_str'], max_tweets_ret
urned=400)
```

```
2018-08-25 18:02:36,392 - INFO - getting tweets for 813286
2018-08-25 18:02:36,397 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:36,801 - INFO - getting tweets for 813286
2018-08-25 18:02:36,804 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:37,186 - INFO - getting tweets for 25073877
2018-08-25 18:02:37,189 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:37,566 - INFO - getting tweets for 25073877
2018-08-25 18:02:37,569 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:37,988 - INFO - getting tweets for 25365536
2018-08-25 18:02:37,991 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:38,488 - INFO - getting tweets for 25365536
2018-08-25 18:02:38,491 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:39,123 - INFO - getting tweets for 50393960
2018-08-25 18:02:39,126 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:39,573 - INFO - getting tweets for 50393960
2018-08-25 18:02:39,575 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:40,256 - INFO - getting tweets for 19397785
2018-08-25 18:02:40,259 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:40,867 - INFO - getting tweets for 19397785
2018-08-25 18:02:40,869 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:41,235 - INFO - getting tweets for 27260086
2018-08-25 18:02:41,239 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:41,779 - INFO - getting tweets for 27260086
2018-08-25 18:02:41,782 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:42,175 - INFO - getting tweets for 250831586
2018-08-25 18:02:42,178 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:42,596 - INFO - getting tweets for 250831586
2018-08-25 18:02:42,599 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:43,156 - INFO - getting tweets for 44196397
2018-08-25 18:02:43,159 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:43,519 - INFO - getting tweets for 44196397
2018-08-25 18:02:43,522 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:43,786 - INFO - getting tweets for 15506669
2018-08-25 18:02:43,788 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:44,134 - INFO - getting tweets for 21447363
2018-08-25 18:02:44,136 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
2018-08-25 18:02:44,688 - INFO - getting tweets for 21447363
2018-08-25 18:02:44,691 - INFO - Starting new HTTPS connection (1): api.twitt
er.com
```

In [8]:
```
def clean_tweet(tweet_text):
    '''clean up tweet text'''
    return re.sub(r' ?https[^ ]*', r'', tweet_text)
```

In [9]:
```
screen_name_to_tweets = {}
for user in users:
    screen_name_to_tweets[user['screen_name']] = [clean_tweet(tweet.get('full_
text', ''))
                                                   for tweet in user['tweets']
                                                   if re.match(r'^RT @', tweet[
'full_text']) is None]
```

In [55]:
```
screen_names = []
tweets = []

for k, ts in screen_name_to_tweets.items():
    for tweet in ts:
        screen_names.append(k)
        tweets.append(tweet)

celebrity_df = pd.DataFrame.from_dict({'screen_names': screen_names, 'tweets':
 tweets})
```

In [13]:
```
# save it for later
celebrity_df.to_csv('/home/ubuntu/data/twitter/celebrity_tweets.csv', header=F
alse, index=False, encoding='utf-8')
```

Model

This portion of the notebook borrows extensively from fast.ai's notebook for lesson 10 in their deep learning course part two.

In [2]:
```
import collections
import html

from fastai.text import *
```

```
/home/paperspace/anaconda3/envs/fastai/lib/python3.6/site-packages/sklearn/en
semble/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is a
n internal NumPy module and should not be imported. It will be removed in a f
uture NumPy release.
  from numpy.core.umath_tests import inner1d
```

Define the directories we'll be using

You'll need to download tweet sentiment data from kaggle - [https://www.kaggle.com/c/twitter-sentiment-analysis2/data (https://www.kaggle.com/c/twitter-sentiment-analysis2/data)](https://www.kaggle.com/c/twitter-sentiment-analysis2/data) - and put it in the twitter/orignal directory

In [3]:
```
BOS = 'xbos'  # beginning-of-sentence tag
FLD = 'xfld'  # data field tag

# path to our original tweet data from kaggle
DATA_PATH = Path("data/")
```

In [4]:
```python
# path to our classification data
CLASSIFICATION_PATH = Path('data/twitter_classification/')
CLASSIFICATION_PATH.mkdir(exist_ok = True)

# path to our language model
LANGUAGE_MODEL_PATH = Path('data/twitter_language_model/')
LANGUAGE_MODEL_PATH.mkdir(exist_ok = True)
```

start prepping our tweet data from kaggle

In [5]:
```python
CLASSES = ['neg', 'pos']
```

In [6]:
```python
df = pd.read_csv(DATA_PATH/'celebrity_train.csv', encoding='latin1')

# shuffle rows and throw out item id column
df = df.drop('ItemID', axis=1)
df = df.sample(frac=1)

# rename columns
df.columns = ['labels', 'text']

# split datafram into train and validation sets for later
split_index = int(df.shape[0] * .9)
train_df, test_df = np.split(df, [split_index], axis=0)

train_df.shape
```

Out[6]:
```
(89990, 2)
```

saving our current progress

In [7]:
```python
# save to classification directory
train_df.to_csv(CLASSIFICATION_PATH/ 'train.csv', header=False, index=False, e
ncoding='utf-8')
test_df.to_csv(CLASSIFICATION_PATH/ 'test.csv', header=False, index=False, enc
oding='utf-8')

f = open(CLASSIFICATION_PATH/'classes.txt', 'w', encoding='utf-8')
f.writelines(f'{c}\n' for c in CLASSES)
f.close()

# save to language model directory
# we should be adding in the test.csv from the kaggle competition here because
 the language model doesn't care about labels
# but in the interest of time I'm opting to just use the training set for the
 language model
train_df.to_csv(LANGUAGE_MODEL_PATH/'train.csv', header=False, index=False, en
coding='utf-8')
test_df.to_csv(LANGUAGE_MODEL_PATH/ 'test.csv', header=False, index=False, enc
oding='utf-8')
```

lets start cleaning some text

In [8]:
```python
# chunksize for pandas so it doesn't run into any memory limits
chunksize=24000
```

In [9]:
```
df.shape
```

Out[9]:
```
(99989, 2)
```

In [10]:
```
## functions pulled from the fast.ai notebook for text tokenization

re1 = re.compile(r'  +')

def fixup(x):
    '''some patterns identified by the fast.ai folks that spacy doesn't accoun
t for'''
    x = x.replace('#39;', "'").replace('amp;', '&').replace('#146;', "'").repl
ace(
        'nbsp;', ' ').replace('#36;', '$').replace('\\n', "\n").replace('quo
t;', "'").replace(
        '<br />', "\n").replace('\\"', '"').replace('<unk>','u_n').replace('
 @.@ ','.').replace(
        ' @-@ ','-').replace('\\', ' \\ ')
    return re1.sub(' ', html.unescape(x))

def get_texts(df, n_lbls=1):
    labels = df.iloc[:,range(n_lbls)].values.astype(np.int64)
    texts = f'\n{BOS} {FLD} 1 ' + df[n_lbls].astype(str)
    for i in range(n_lbls+1, len(df.columns)): texts += f' {FLD} {i-n_lbls} '
+ df[i].astype(str)
    texts = list(texts.apply(fixup).values)

    tok = Tokenizer().proc_all_mp(partition_by_cores(texts))
    return tok, list(labels)

def get_all(df, n_lbls):
    '''tokenize the text'''
    tok, labels = [], []
    for i, r in enumerate(df):
        tok_, labels_ = get_texts(r, n_lbls)
        tok += tok_;
        labels += labels_
    return tok, labels
```

In [11]:
```
# grab our dataframes from earlier
train_df = pd.read_csv(LANGUAGE_MODEL_PATH/'train.csv', header=None, chunksize
=chunksize)
val_df = pd.read_csv(LANGUAGE_MODEL_PATH/ 'test.csv', header=None, chunksize=c
hunksize)
```

In [12]:
```
# tokenize the tweets
train_tokens, train_labels = get_all(train_df, 1)
val_tokens, val_labels = get_all(val_df, 1)
```

In [21]:
```
#make temporary directory
os.mkdir(LANGUAGE_MODEL_PATH/'tmp')
```

In [22]:
```
# save our tokens
np.save(LANGUAGE_MODEL_PATH/'tmp/tok_trn.npy', train_tokens)
np.save(LANGUAGE_MODEL_PATH/'tmp/tok_val.npy', val_tokens)
```

In [23]:
```
# load back in
train_tokens = np.load(LANGUAGE_MODEL_PATH/'tmp/tok_trn.npy')
val_tokens = np.load(LANGUAGE_MODEL_PATH/'tmp/tok_val.npy')
```

In [24]:
```
# lets take a look at our most common tokens
freq = Counter(token for tokens in train_tokens for token in tokens)
freq.most_common(25)
```

Out[24]:
```
[('1', 90578),
 ('\n', 90001),
 ('xbos', 89990),
 ('xfld', 89990),
 ('i', 59573),
 ('.', 48209),
 ('!', 47040),
 (',', 29222),
 ('you', 27076),
 ('the', 26915),
 ('to', 26704),
 ('a', 20717),
 ('it', 20012),
 ('t_up', 19450),
 ('?', 17512),
 ('and', 14741),
 ('that', 13064),
 ('&', 12953),
 ('...', 12921),
 ('my', 12449),
 ('is', 11447),
 ('for', 11277),
 ('/', 10948),
 ('in', 10717),
 ("'s", 10717)]
```

looks about right. quick note to keep in mind - the "t_up" token isn't in the text its self, it is a marker indicating the following token is all uppercase.

fast.ai recommends that you only keep the 60,000 or so most common tokens. Reason being low frequency tokens don't help you learn a lot about a language.

We don't have that many tokens for our tweets, but it makes me feel good to put in anyways ...

In [25]:
```
max_vocab = 60000
min_freq = 2

int_to_token = [o for o, c in freq.most_common(max_vocab) if c > min_freq]
int_to_token.insert(0, '_pad_')
int_to_token.insert(0, '_unk_')
```

In [26]:
```
token_to_int = collections.defaultdict(lambda: 0, {v: k for k, v in enumerate(
int_to_token)})
len(int_to_token)
```

Out[26]: 20133

In [27]:
```
train_lm = np.array([[token_to_int[o] for o in p] for p in train_tokens])
val_lm = np.array([[token_to_int[o] for o in p] for p in val_tokens])
```

In [28]:
```
# saving our progress
np.save(LANGUAGE_MODEL_PATH/'tmp/trn_ids.npy', train_lm)
np.save(LANGUAGE_MODEL_PATH/'tmp/val_ids.npy', val_lm)

pickle.dump(int_to_token, open(LANGUAGE_MODEL_PATH/'tmp/itos.pkl', 'wb'))
```

```
In [29]: # loading back in
         train_lm = np.load(LANGUAGE_MODEL_PATH/'tmp/trn_ids.npy')
         val_lm = np.load(LANGUAGE_MODEL_PATH/'tmp/val_ids.npy')
         int_to_token = pickle.load(open(LANGUAGE_MODEL_PATH/'tmp/itos.pkl', 'rb'))
```

```
In [30]: num_twitter_tokens = len(int_to_token)
         num_twitter_tokens, len(train_lm)
```

```
Out[30]: (20133, 89990)
```

load in a pretrained language model trained on wikipedia text

run this line to download wikipedia model

```
In [23]: # ! wget -nH -r -np -P {PATH} http://files.fast.ai/models/wt103/
```

```
In [31]: # some stats from the wikepedia model
         embedding_size, num_hidden, num_layers = 400,1150,3
```

```
In [35]: PRE_PATH = Path("/home/paperspace/data/twitter/lm/models")
         PRE_LM_PATH = Path(PRE_PATH/"lm_tt.h5")

         wgts = torch.load(PRE_LM_PATH, map_location = lambda storage, loc: storage)

         enc_wgts = to_np(wgts["0.encoder.weight"])
         row_m = enc_wgts.mean(0)

         itos2 = pickle.load((PRE_PATH/"itos_tt.pkl").open("rb"))
         stoi2 = collections.defaultdict(lambda:-1, {v:k for k,v in enumerate(itos2)})


         new_w = np.zeros((num_twitter_tokens, embedding_size), dtype=np.float32)
         for i,w in enumerate(int_to_token):
             r = stoi2[w]
             new_w[i] = enc_wgts[r] if r>=0 else row_m


         wgts['0.encoder.weight'] = T(new_w)
         wgts['0.encoder_with_dropout.embed.weight'] = T(np.copy(new_w))
         wgts['1.decoder.weight'] = T(np.copy(new_w))
```

```
In [25]: # PRE_PATH = Path('data/aclImdb/models/wt103')
         # # PRE_LM_PATH = PRE_PATH/'fwd_wt103.h5'
```

```
In [26]: # grab the weights from the encoder
         # weights = torch.load(PRE_LM_PATH, map_location=lambda storage, loc: storage)
```

The mean of the weights from layer 0 can be used to assign weights to tokens that exist in the wikipedia dataset but not in the twitter dataset

```
In [27]: # encoder_weights = to_np(weights['0.encoder.weight'])
         # # row_m = enc_wgts.mean(0)
         # encoder_mean = encoder_weights.mean(0)
```

In [28]:
```
# wiki_int_to_token = pickle.load(open(PRE_PATH/'itos_wt103.pkl', 'rb'))
# wiki_token_to_int = collections.defaultdict(lambda: -1, {v:k for k, v in enu
merate(wiki_int_to_token)})
```

We need to assign mean weights to tokens that exist in our twitter dataset that dont in the wikipedia dataset the pretrained model was trained on.

In [29]:
```
# new_weights = np.zeros((num_twitter_tokens, embedding_size), dtype=np.float3
2)
# for i, w in enumerate(int_to_token):
#     r = wiki_token_to_int[w]
#     new_weights[i] = encoder_weights[r] if r >= 0 else encoder_mean
```

We now need to put the new weights into the pretrained model

The weights between the encoder and decoder also need to be tied together

In [30]:
```
# weights['0.encoder.weight'] = T(new_weights)
# weights['0.encoder_with_dropout.embed.weight'] = T(np.copy(new_weights))
# weights['1.decoder.weight'] = T(np.copy(new_weights))
```

Retraining the wikipedia language model

In [36]:
```
wd=1e-7 # weight decay
bptt=70 # ngram size.  i.e. the model sees ~70 tokens and then tries to predic
t the 71st
bs=52 # batch size
opt_fn = partial(optim.Adam, betas=(0.8, 0.99)) # optimazation function
```

Here we define a special fastai data loader, the `LanguageModelLoader`, to feed the training data into the model whilst training.

We can then use those to instanciate a `LanguageModelData` class that returns a fastai model we can train

In [37]:
```
train_dl = LanguageModelLoader(np.concatenate(train_lm), bs, bptt)
val_dl = LanguageModelLoader(np.concatenate(val_lm), bs, bptt)

md = LanguageModelData(DATA_PATH, 1, num_twitter_tokens, train_dl, val_dl, bs=
bs, bptt=bptt)
```

In [38]:
```
# the dropouts for each layer.
drops = np.array([0.25, 0.1, 0.2, 0.02, 0.15])*0.7
```

The last embedding layer needs to be tuned first so the new weights we set for the pretrained model get tuned properly.

fastai allows you to freeze and unfreeze model layers. So here we freeze everything but the weights in the last embedding layer

In [39]:
```
learner = md.get_model(
    opt_fn, embedding_size, num_hidden, num_layers, dropouti=drops[0], dropout
=drops[1],
    wdrop=drops[2], dropoute=drops[3], dropouth=drops[4]
)

learner.metrics = [accuracy]

# freeze everything except last layer
learner.freeze_to(-1)
```

In [41]:
```
# load the weights
learner.model.load_state_dict(wgts)
```

In [42]:
```
lr = 1e-3 # learning rate
lrs = lr
```

In [43]:
```
learner.fit(lrs/2, 1, wds=wd, use_clr=(32,2), cycle_len=2)
```

```
epoch      trn_loss   val_loss   accuracy
    0      4.926055   4.753983   0.299884
    1      4.473818   4.339071   0.302607
```

Out[43]: `[array([4.33907]), 0.30260673484631945]`

In [44]:
```
# save our progress
learner.save('lm_last_ft')
```

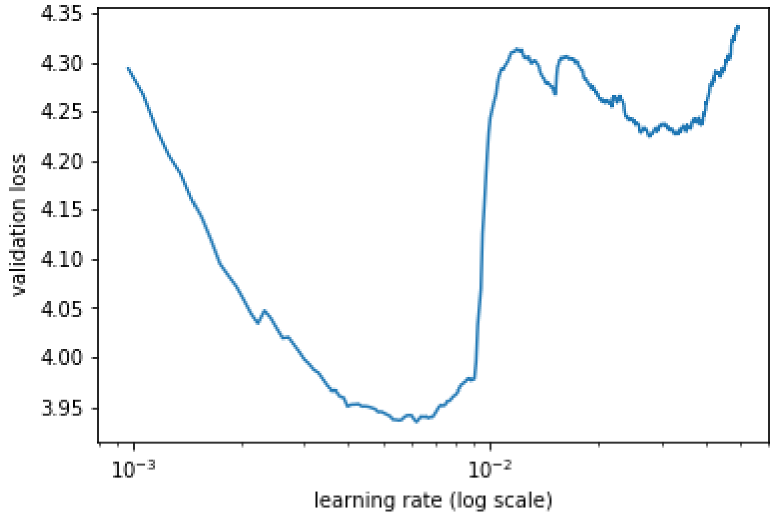
In [45]:
```
# load back in
learner.load('lm_last_ft')
```

In [46]:
```
# now with our new embedding weights trained up, we can unfreeze and train all
 layers
learner.unfreeze()
```

In [47]:
```
# to find our learning rate
learner.lr_find(start_lr=lrs/10, end_lr=lrs*50, linear=True)
```

```
epoch      trn_loss   val_loss   accuracy
    0      4.339951   4.196579   0.316808
```

In [48]:
```
learner.sched.plot()
```



In [43]:
```
# looks like 10-2 or 10-3 or so could be a good learning rate for us
```

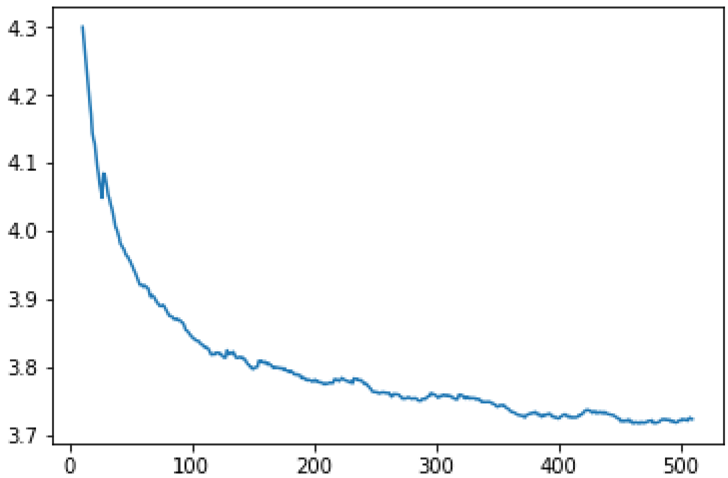
In [49]:
```
learner.fit(lrs, 1, wds=wd, use_clr=(20,10), cycle_len=1)
```

```
epoch      trn_loss   val_loss   accuracy
    0      3.735618   3.564491   0.37966
```

Out[49]:
```
[array([3.56449]), 0.37966035678982735]
```

In [50]:
```
# save our progress
learner.save('lm1')
learner.save_encoder('lm1_enc')
```

In [51]:
```
# taking a look at our loss
learner.sched.plot_loss()
```



Tweet Sentiment Classifier

Now that we have our language model trained on tweets, we can start training our tweet sentiment classifier

To do this all we have to do is tack on a layer to our trained encoder.

In [52]:
```
train_df = pd.read_csv(CLASSIFICATION_PATH/'train.csv', header=None, chunksize
=chunksize)
val_df = pd.read_csv(CLASSIFICATION_PATH/'test.csv', header=None, chunksize=ch
unksize)
```

In [53]:
```
# do the same cleaning we did for the language model
train_tokens, train_labels = get_all(train_df, 1)
val_tokens, val_labels = get_all(val_df, 1)
```

In [55]:
```
# make temp directory in classifier directory
os.mkdir(CLASSIFICATION_PATH/'tmp')

# save tokens
np.save(CLASSIFICATION_PATH/'tmp/tok_trn.npy', train_tokens)
np.save(CLASSIFICATION_PATH/'tmp/tok_val.npy', val_tokens)

np.save(CLASSIFICATION_PATH/'tmp/trn_labels.npy', train_labels)
np.save(CLASSIFICATION_PATH/'tmp/val_labels.npy', val_labels)
```

In [56]:
```
# load back in
train_tokens = np.load(CLASSIFICATION_PATH/'tmp/tok_trn.npy')
val_tokens = np.load(CLASSIFICATION_PATH/'tmp/tok_val.npy')
```

In [57]:
```
int_to_token = pickle.load(open(LANGUAGE_MODEL_PATH/'tmp/itos.pkl', 'rb'))
token_to_int = collections.defaultdict(lambda: 0, {v:k for k, v in enumerate(i
nt_to_token)})
len(int_to_token)
```

Out[57]:
```
20133
```

In [58]:
```
train_classification = np.array([[token_to_int[o] for o in p] for p in train_t
okens])
val_classification = np.array([[token_to_int[o] for o in p] for p in val_token
s])
```

In [59]:
```
np.save(CLASSIFICATION_PATH/'tmp/trn_ids.npy', train_classification)
np.save(CLASSIFICATION_PATH/'tmp/val_ids.npy', val_classification)
```

In [60]:
```
# load back in
train_classification = np.load(CLASSIFICATION_PATH/'tmp/trn_ids.npy')
val_classification = np.load(CLASSIFICATION_PATH/'tmp/val_ids.npy')

train_labels = np.squeeze(np.load(CLASSIFICATION_PATH/'tmp/trn_labels.npy'))
val_labels = np.squeeze(np.load(CLASSIFICATION_PATH/'tmp/val_labels.npy'))
```

In [61]:
```
# params
bptt, embedding_size, num_hidden, num_layers = 70, 400, 1150, 3
num_tokens = len(int_to_token)
opt_fn = partial(optim.Adam, betas=(0.8, 0.99))
bs = 48
```

In [62]:
```
train_classification[:5], train_labels[:5]
```

Out[62]:
```
(array([list([3, 4, 5, 2, 0, 295, 12013, 54, 194, 317, 206, 53, 53]),
        list([3, 4, 5, 2, 12014, 15, 2509, 46, 15, 17, 15, 2777, 60, 25, 21,
15, 492, 8, 8, 298, 58, 34, 18, 804, 16]),
        list([3, 4, 5, 2, 552, 13, 32, 1128, 193, 14870, 0]),
        list([3, 4, 5, 2, 1997, 122, 88, 43, 24, 24, 225, 24, 0, 38, 2396, 22
40]),
        list([3, 4, 5, 2, 0, 6, 154, 6, 130, 40, 6524, 8, 8, 8, 15, 53, 1153,
30, 145, 409, 28, 8, 6, 89, 404, 454, 128, 33, 476, 9, 54, 97, 103, 12, 0])],
       dtype=object), array([1, 0, 0, 1, 0]))
```

In [63]:
```
min_label = train_labels.min()
train_labels -= min_label
val_labels -= min_label
c = int(train_labels.max()) + 1
```

In [64]:
```
train_ds = TextDataset(train_classification, train_labels)
val_ds = TextDataset(val_classification, val_labels)

# the sortish sampler helps by sorting things kinda sorta by their token lengt
h so padding isn't crazy
train_sampler = SortishSampler(train_classification, key=lambda x: len(train_c
lassification[x]), bs=bs//2)
# doesn't matter so much for the validation set
val_sampler = SortSampler(val_classification, key=lambda x: len(val_classifica
tion[x]))

# get data loaders
train_dl = DataLoader(train_ds, bs//2, transpose=True, num_workers=1, pad_idx=
1, sampler=train_sampler)
val_dl = DataLoader(val_ds, bs, transpose=True, num_workers=1, pad_idx=1, samp
ler=val_sampler)

# model data
md = ModelData(DATA_PATH, train_dl, val_dl)
```

In [65]:
```
# part 1
dps = np.array([0.4, 0.5, 0.05, 0.3, 0.1])
```

In [66]:
```
# part 2
dps = np.array([0.4,0.5,0.05,0.3,0.4])*0.5
```

In [67]:
```
m = get_rnn_classifier(bptt, 20*70, c, num_tokens, emb_sz=embedding_size, n_hi
d=num_hidden, n_layers=num_layers,
                       pad_token=1, layers=[embedding_size*3, 50, c], drops=[d
ps[4], 0.1], dropouti=dps[0],
                       wdrop=dps[1], dropoute=dps[2], dropouth=dps[3])
```

In [68]:
```
opt_fn = partial(optim.Adam, betas=(0.7, 0.99))
```

In [95]:
```
learn = RNN_Learner(md, TextModel(to_gpu(m)), opt_fn=opt_fn)
learn.reg_fn = partial(seq2seq_reg, alpha=2, beta=1)
learn.clip=25.
learn.metrics = [accuracy]
```

In [96]:
```
lr=3e-3
lrm = 2.6
lrs = np.array([lr/(lrm**4), lr/(lrm**3), lr/(lrm**2), lr/lrm, lr]) # differen
tial learning rates
```

In [97]:
```
#lrs=np.array([1e-4,1e-4,1e-4,1e-3,1e-2])
```

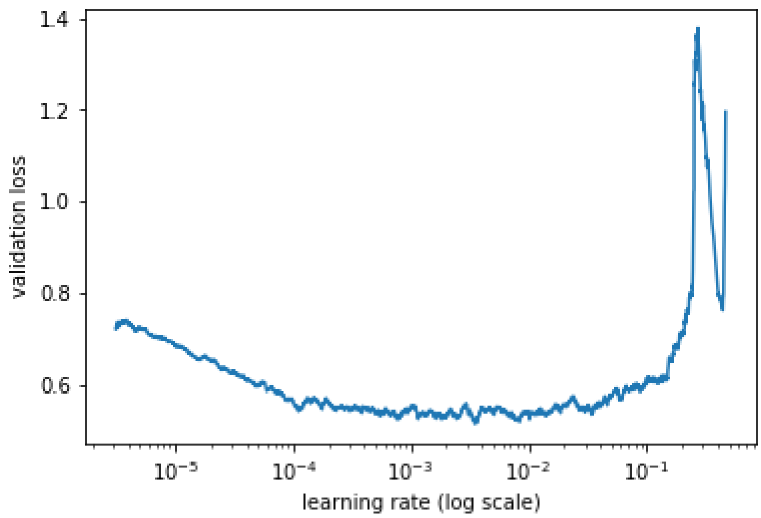
load our encoder from our tweet language model

In [98]:
```
wd = 1e-7
wd = 0
learn.load_encoder('lm1_enc')
```

In [99]:
```
# freeze all except last layer
learn.freeze_to(-1)
```

In [74]:
```
# to find learning rate
learn.lr_find(lrs/1000)
learn.sched.plot()
```

```
80%|████████      | 2991/3750 [00:44<00:11, 67.70it/s, loss=1.9]
```



In [ ]:
```
# little tough to tell here, but we'll go with what we set previously and what
 fastai used for their imdb dataset
```

In [100]:
```
learn.fit(lrs, 1, wds=wd, cycle_len=1, use_clr=(8,3))
```

```
epoch      trn_loss   val_loss   accuracy
    0      0.509631   0.47636    0.768477
```

Out[100]:
```
[array([0.47636]), 0.7684768462570944]
```

In [101]:
```
# save our first classifier
learn.save('clas_0')
```

In [102]:
```
# load it back in
learn.load('clas_0')
```

In [103]:
```
# unfreeze one more layer
learn.freeze_to(-2)
```

In [104]:
```
learn.fit(lrs, 1, wds=wd, cycle_len=1, use_clr=(8,3))
```

```
epoch      trn_loss   val_loss   accuracy
    0      0.46139    0.429541   0.79928
```

Out[104]: `[array([0.42954]), 0.7992799280911568]`

In [105]:
```
# save our second classifier
learn.save('clas_1')
```

In [106]:
```
# load it back in
learn.load('clas_1')
```

In [107]:
```
# unfreeze all layers so we're training the whole network
learn.unfreeze()
```

In [108]:
```
learn.fit(lrs, 1, wds=wd, cycle_len=1, use_clr_beta=(20,20, 0.95, 0.85))
```

```
epoch      trn_loss   val_loss   accuracy
    0      0.438546   0.408425   0.813881
```

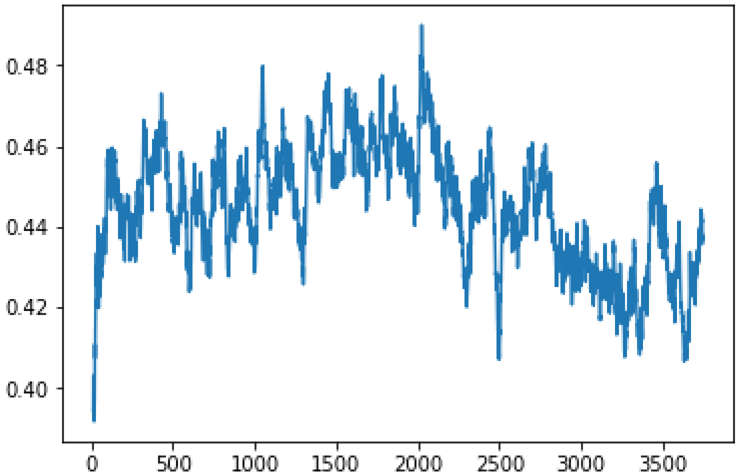
Out[108]: `[array([0.40843]), 0.8138813873221485]`

In [110]:
```
learn.fit(lrs/10, 1, wds=wd, cycle_len=1, use_clr_beta=(20,20, 0.95, 0.85))
```

```
epoch      trn_loss   val_loss   accuracy
    0      0.381378   0.402993   0.813981
```

Out[110]: `[array([0.40299]), 0.8139813973797787]`

In [109]:
```
# plot out our loss
learn.sched.plot_loss()
```



In [113]:
```
# save our final classifier
learn.save('clas_2')
```

Celebrity tweet sentiment

In [29]:
```
learn.load('clas_2')
```

In [30]:
```
# load our celebrity tweets
celebrity_df = pd.read_csv('/home/ubuntu/data/twitter/celebrity_tweets.csv', h
eader=None)
```

In [31]:
```
celebrity_tokens, _ = get_texts(celebrity_df, 0)
```

In [32]:
```
int_to_token = pickle.load(open(LANGUAGE_MODEL_PATH + '/tmp/itos.pkl', 'rb'))
token_to_int = collections.defaultdict(lambda: 0, {v:k for k, v in enumerate(i
nt_to_token)})
len(int_to_token)
```

Out[32]:
```
15833
```

In [33]:
```
celebrity_classification = np.array([[token_to_int[o] for o in p] for p in cel
ebrity_tokens])
```

In [34]:
```
celebrity_ds = TextDataset(celebrity_classification, np.zeros(len(celebrity_cl
assification), dtype=int))

celebrity_dl = DataLoader(celebrity_ds, bs, transpose=True, num_workers=1, pad
_idx=1)
```

In [35]:
```
log_preds = learn.predict_dl(celebrity_dl)
```

In [36]:
```
log_preds.shape
```

Out[36]:
```
(3256, 2)
```

In [37]:
```
preds = np.argmax(log_preds, axis=1)
probs = np.exp(log_preds[:,1])
```

In [38]:
```
preds
```

Out[38]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [39]:
```
celebrity_df = celebrity_df.assign(sentiment=pd.Series(preds))
celebrity_df.to_csv('/home/ubuntu/data/twitter/celebrity_tweets_results.csv',
header=None, index=None)
```

In [2]:
```
celebrity_df = pd.read_csv('results/celebrity_tweets_results.csv', header=None
)
celebrity_df.columns = [0, 1, 'sentiment']
```

In [11]:

```
celebrity_to_tweets = {}
for index, row in celebrity_df.iterrows():
    if row[0] not in celebrity_to_tweets:
        celebrity_to_tweets[row[0]] = []
    elif row[1]:
        celebrity_to_tweets[row[0]].append({
            'tweet': row[1],
            'sentiment': row['sentiment']
        })
```

In [15]:

```
results = []
for screen_name, tweets in celebrity_to_tweets.items():
    if tweets:
        avg_sentiment = np.mean([t['sentiment']
                                 for t in tweets
                                 if not pd.isnull(t['tweet'])]) # throw out th
e empty tweets
        print(screen_name, avg_sentiment)
        results.append((screen_name, avg_sentiment))
```

```
BarackObama 0.7585227272727273
realDonaldTrump 0.7068965517241379
KimKardashian 0.85
BillGates 0.772020725388601
Oprah 0.8575197889182058
justinbieber 0.9529914529914529
TheRock 0.9039039039039038
elonmusk 0.8439306358381503
JeffBezos 0.9559748427672956
katyperry 0.8891752577319587
```

The most postive celebrities on twitter

In [16]:

```
results = sorted(results, key=lambda x: x[1], reverse=True)
results
```

Out[16]:

```
[('JeffBezos', 0.9559748427672956),
 ('justinbieber', 0.9529914529914529),
 ('TheRock', 0.9039039039039038),
 ('katyperry', 0.8891752577319587),
 ('Oprah', 0.8575197889182058),
 ('KimKardashian', 0.85),
 ('elonmusk', 0.8439306358381503),
 ('BillGates', 0.772020725388601),
 ('BarackObama', 0.7585227272727273),
 ('realDonaldTrump', 0.7068965517241379)]
```

In [17]:
```
import math

from bokeh.io import show, output_file
from bokeh.models import ColumnDataSource
from bokeh.palettes import Spectral10
from bokeh.plotting import figure

output_file("celebrity_tweet_sentimate.html")

handles = ['@' + x[0] for x in results]
counts = [x[1] for x in results]
counts = [int(x) for x in np.asarray(counts) * 100]

source = ColumnDataSource(data=dict(handles=handles, counts=counts, color=Spec
tral10))

p = figure(x_range=handles, y_range=(50,100), plot_height=400, title="Who's th
e most postive public figure on Twitter?",
           toolbar_location=None, tools="")

p.vbar(x='handles', top='counts', width=0.8, color='color', source=source, )

p.xaxis.major_label_orientation = -math.pi/5
p.min_border_right = 50
p.yaxis.axis_label = "% of tweets that are positive"
p.xaxis.axis_label = "Twitter handle"


p.xgrid.grid_line_color = None
p.legend.orientation = "horizontal"
p.legend.location = "top_center"

show(p)
```