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

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]:
```
# 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]:
```
CLASSES = ['neg', 'pos']
```

```
In [6]: 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]: # 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]: # 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 [ ]:
```
learner.fit(lrs/2, 1, wds=wd, use_clr=(32,2), cycle_len=2)
```

```
 8%|█          | 43/515 [00:05<00:45, 10.37it/s, loss=5.32]
```

In [38]:
```
# save our progress
learner.save('lm_last_ft')
```

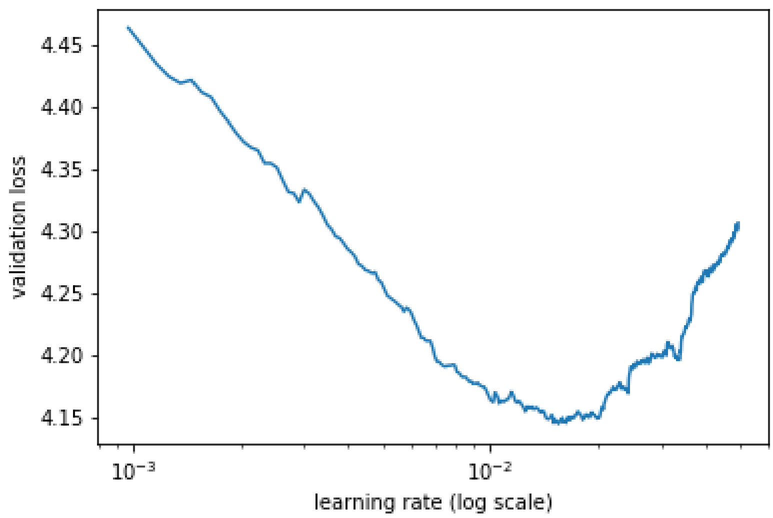
In [39]:
```
# load back in
learner.load('lm_last_ft')
```

In [40]:
```
# now with our new embedding weights trained up, we can unfreeze and train all
 layers
learner.unfreeze()
```

In [41]:
```
# 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      4.307766   4.150575   0.321169
```

In [42]:
```
learner.sched.plot()
```



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

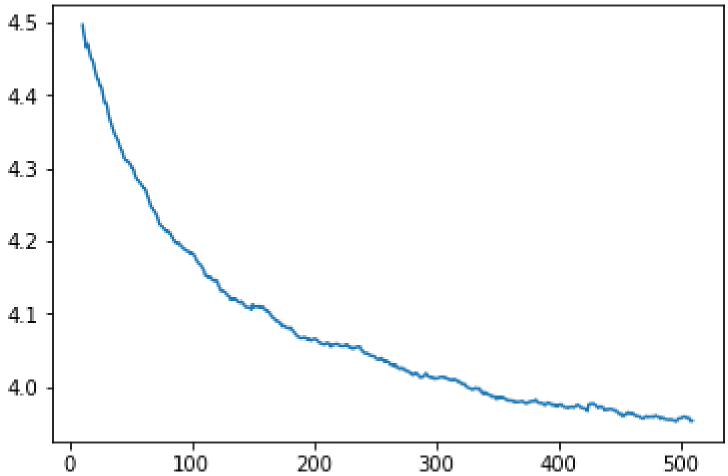
In [44]:
```
learner.fit(lrs, 1, wds=wd, use_clr=(20,10), cycle_len=1)
```

```
epoch      trn_loss   val_loss   accuracy
    0      3.955422   3.76257    0.360825
```

Out[44]: [array([3.76257]), 0.36082455994827406]

In [45]:
```
# save our progress
learner.save('lm1')
learner.save_encoder('lm1_enc')
```

In [46]:
```
# 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 [47]:
```
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 [48]:
```
# 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 [ ]:
```
# 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 [ ]:
```
# load back in
train_tokens = np.load(CLASSIFICATION_PATH/'tmp/tok_trn.npy')
val_tokens = np.load(CLASSIFICATION_PATH/'tmp/tok_val.npy')
```

In [ ]:
```
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)
```

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

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

In [89]:
```
min_label = train_labels.min()
train_labels -= min_label
val_labels -= min_label
c = int(train_labels.max()) + 1
```

In [91]:
```
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 [92]:
```
# part 1
dps = np.array([0.4, 0.5, 0.05, 0.3, 0.1])
```

In [93]:
```
# part 2
dps = np.array([0.4,0.5,0.05,0.3,0.4])*0.5
```

In [94]:
```
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 [95]:
```
opt_fn = partial(optim.Adam, betas=(0.7, 0.99))
```

In [96]:
```
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 [97]:
```
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 [ ]:
```
#lrs=np.array([1e-4,1e-4,1e-4,1e-3,1e-2])
```

load our encoder from our tweet language model

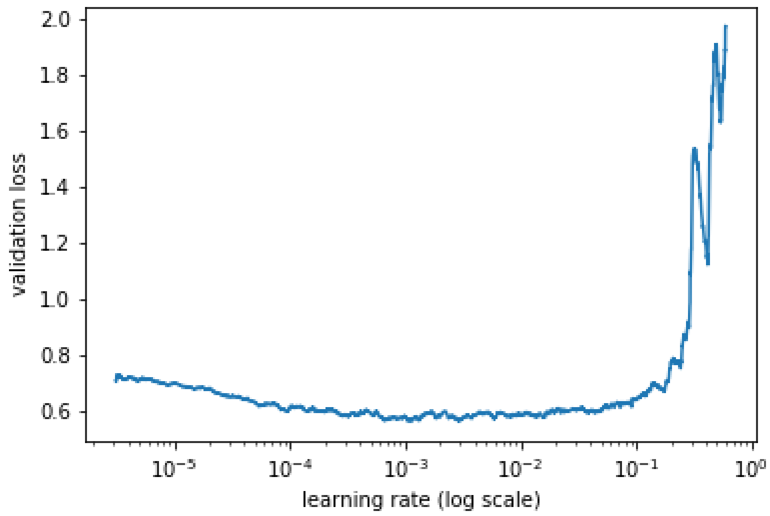
In [98]:
```
wd = 1e-7
wd = 0
learn.load_encoder('lm1_enc')
```

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

In [100]:

```
# to find learning rate
learn.lr_find(lrs/1000)
learn.sched.plot()
```

```
81%|████████    | 3050/3750 [00:40<00:09, 75.95it/s, loss=2.16]
```



In [ ]:

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

In [101]:

```
learn.fit(lrs, 1, wds=wd, cycle_len=1, use_clr=(8,3))
```

```
epoch      trn_loss   val_loss   accuracy
    0      0.565335   0.512834   0.743274
```

Out[101]:

```
[array([0.51283]), 0.7432743282047006]
```

In [1]:

```
# save our first classifier
learn.save('clas_0')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-44c81e4f52fe> in <module>
      1 # save our first classifier
----> 2 learn.save('clas_0')

NameError: name 'learn' is not defined
```

In [2]:

```
# load it back in
learn.load('clas_0')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-608fe198f397> in <module>
      1 # load it back in
----> 2 learn.load('clas_0')

NameError: name 'learn' is not defined
```

In [104]:

```
# unfreeze one more layer
learn.freeze_to(-2)
```

In [105]:
```
learn.fit(lrs, 1, wds=wd, cycle_len=1, use_clr=(8,3))
```

```
epoch      trn_loss   val_loss   accuracy
    0      0.489556   0.457601   0.783578
```

Out[105]:
```
[array([0.4576]), 0.7835783562233167]
```

In [106]:
```
# save our second classifier
learn.save('clas_1')
```

In [107]:
```
# load it back in
learn.load('clas_1')
```

In [108]:
```
# unfreeze all layers so we're training the whole network
learn.unfreeze()
```

In [111]:

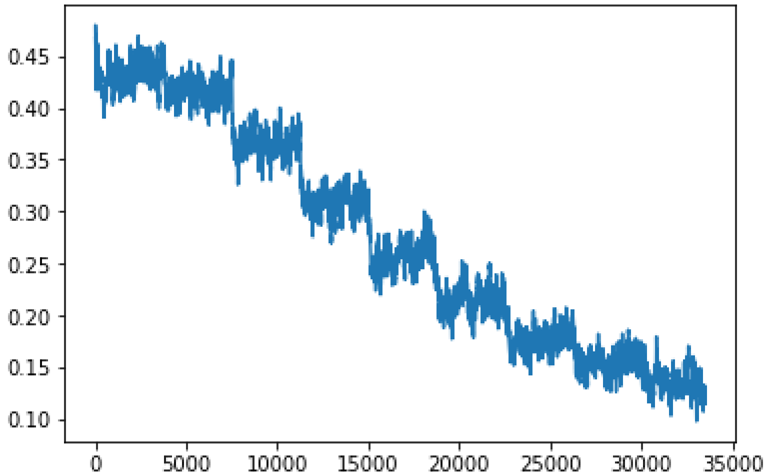
```
learn.fit(lrs, 1, wds=wd, cycle_len=10, use_clr_beta=(32,10))
```

```
epoch      trn_loss   val_loss   accuracy
    0      0.454162   0.421473   0.808981
    1      0.432943   0.411333   0.810281
    2      0.366563   0.422387   0.810081
    3      0.3215     0.449872   0.814681
    4      0.248104   0.49484    0.811581
    5      0.224862   0.553132   0.809581
    6      0.198725   0.528209   0.80478
    7      0.153145   0.568736   0.806881
 93%|█████████▎| 3490/3750 [01:52<00:08, 30.19it/s, loss=0.121]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-111-2ce4d2f4f7f2> in <module>
----> 1 learn.fit(lrs, 1, wds=wd, cycle_len=10, use_clr=(32,10))

~/fastai/courses/dl2/fastai/text.py in fit(self, *args, **kwargs)
    209
    210     def _get_crit(self, data): return F.cross_entropy
--> 211     def fit(self, *args, **kwargs): return super().fit(*args, **kwarg
s, seq_first=True)
    212
    213     def save_encoder(self, name): save_model(self.model[0], self.get_
model_path(name))

~/fastai/courses/dl2/fastai/learner.py in fit(self, lrs, n_cycle, wds, **kwar
gs)
    300         self.sched = None
    301         layer_opt = self.get_layer_opt(lrs, wds)
--> 302         return self.fit_gen(self.model, self.data, layer_opt, n_cycle
, **kwargs)
    303
    304     def warm_up(self, lr, wds=None):

~/fastai/courses/dl2/fastai/learner.py in fit_gen(self, model, data, layer_op
t, n_cycle, cycle_len, cycle_mult, cycle_save_name, best_save_name, use_clr,
 use_clr_beta, metrics, callbacks, use_wd_sched, norm_wds, wds_sched_mult, us
e_swa, swa_start, swa_eval_freq, **kwargs)
    247             metrics=metrics, callbacks=callbacks, reg_fn=self.reg_fn,
 clip=self.clip, fp16=self.fp16,
    248             swa_model=self.swa_model if use_swa else None, swa_start=
swa_start,
--> 249             swa_eval_freq=swa_eval_freq, **kwargs)
    250
    251     def get_layer_groups(self): return self.models.get_layer_groups()

~/fastai/courses/dl2/fastai/model.py in fit(model, data, n_epochs, opt, crit,
 metrics, callbacks, stepper, swa_model, swa_start, swa_eval_freq, visualize,
 **kwargs)
    139             batch_num += 1
    140             for cb in callbacks: cb.on_batch_begin()
--> 141             loss = model_stepper.step(V(x),V(y), epoch)
    142             avg_loss = avg_loss * avg_mom + loss * (1-avg_mom)
    143             debias_loss = avg_loss / (1 - avg_mom**batch_num)

~/fastai/courses/dl2/fastai/model.py in step(self, xs, y, epoch)
     55         if self.loss_scale != 1: assert(self.fp16); loss = loss*self.
loss_scale
     56         if self.reg_fn: loss = self.reg_fn(output, xtra, raw_loss)
---> 57         loss.backward()
     58         if self.fp16: update_fp32_grads(self.fp32_params, self.m)
     59         if self.loss_scale != 1:

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/autograd/variable.p
y in backward(self, gradient, retain_graph, create_graph, retain_variables)
    165                 Variable.
    166         """
--> 167         torch.autograd.backward(self, gradient, retain_graph, create_
graph, retain_variables)
    168
    169     def register_hook(self, hook):

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/autograd/__init__.p
y in backward(variables, grad_variables, retain_graph, create_graph, retain_v
```

```
ariables)
     97
     98         Variable._execution_engine.run_backward(
---> 99             variables, grad_variables, retain_graph)
    100
    101

KeyboardInterrupt:
```

In [112]:
```
# plot out our loss
learn.sched.plot_loss()
```



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