In [22]:
```
from fastai import *        # Quick access to most common functionality
from fastai.text import *
```

In [2]:
```
torch.cuda.set_device(0)
```

In [23]:
```
path = untar_data(URLs.IMDB_SAMPLE)
path
```

Out[23]:
```
PosixPath('/home/annaanisienia/.fastai/data/imdb_sample')
```

In [24]:
```
df_train = pd.read_csv(path/'train.csv', header=None)
df_val = pd.read_csv(path/'valid.csv',header=None)
df_val.head(1)
```

Out[24]:

	0	1
0	1	This very funny British comedy shows what migh...

In [5]:
```
## Making a small example
df_train = df_train.iloc[:80,:]
df_val = df_val.iloc[:20,:]
# Add new column to simulate multi-label/multi-class
df_train['new1'] = 1
df_train['new2'] = 0
df_val['new1'] = 1
df_val['new2'] = 0

df_train = df_train[[0,'new1','new2',1]]
df_val = df_val[[0,'new1','new2',1]]
df_train.head(1)
```

Out[5]:

	0	new1	new2	1
0	0	1	0	Un-bleeping-believable! Meg Ryan doesn't even ...

In [21]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 4 columns):
0       80 non-null int64
new1    80 non-null int64
new2    80 non-null int64
1       80 non-null object
dtypes: int64(3), object(1)
memory usage: 2.6+ KB
```

In [19]:
```
data_path = Path('./data')
data_path
```

Out[19]:
```
PosixPath('data')
```

In [7]:
```
import os
if not os.path.exists(data_path):
    os.makedirs(data_path)
```

In [8]:
```
df_train.to_csv(data_path/'train.csv',header=None,index=None)
df_val.to_csv(data_path/'valid.csv',header=None,index=None)
```

In [9]:
```
data_lm = TextLMDataBunch.from_csv(data_path)
data_clas = TextClasDataBunch.from_csv(data_path,
                vocab=data_lm.train_ds.vocab, n_labels=3)
```

In [10]:
```
data_clas.train_ds.classes
```

Out[10]:
```
array([0., 1.], dtype=float32)
```

In [11]:
```
is_listy(data_clas.train_ds.labels)
```

Out[11]:
```
False
```

In [12]:
```
learn = RNNLearner.language_model(data_lm,
                pretrained_model=URLs.WT103, drop_mult=0.5)
learn.fit_one_cycle(1, 1e-2)
```

```
Total time: 00:01
epoch  train_loss  valid_loss  accuracy
1      4.157425    3.564368    0.270833  (00:01)
```

In [13]:
```
learn.save_encoder("lm")
```

In [14]:
```
learn = RNNLearner.classifier(data_clas)
learn.metrics = []
learn.load_encoder("lm")
learn.fit_one_cycle(1, 1e-3)
```

```
Total time: 00:03
epoch  train_loss  valid_loss
1      0.700342    0.682231    (00:03)
```

In [15]:
```
learn.fit_one_cycle(1)
```

```
Total time: 00:03
epoch  train_loss  valid_loss
1      0.690605    0.679217    (00:03)
```

In [16]:
```
1 learn.model.eval()
```

Out[16]:
```
SequentialRNN(
  (0): MultiBatchRNNCore(
    (encoder): Embedding(935, 400, padding_idx=1)
    (encoder_dp): EmbeddingDropout(
      (emb): Embedding(935, 400, padding_idx=1)
    )
    (rnns): ModuleList(
      (0): WeightDropout(
        (module): LSTM(400, 1150)
      )
      (1): WeightDropout(
        (module): LSTM(1150, 1150)
      )
      (2): WeightDropout(
        (module): LSTM(1150, 400)
      )
    )
    (input_dp): RNNDropout()
    (hidden_dps): ModuleList(
      (0): RNNDropout()
      (1): RNNDropout()
      (2): RNNDropout()
    )
  )
  (1): PoolingLinearClassifier(
    (layers): Sequential(
      (0): BatchNorm1d(1200, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (1): Dropout(p=0.4)
      (2): Linear(in_features=1200, out_features=50, bias=True)
      (3): ReLU(inplace)
      (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      (5): Dropout(p=0.1)
      (6): Linear(in_features=50, out_features=3, bias=True)
    )
  )
)
```

In [17]:

```
preds = learn.get_preds(is_test=False)
preds[1]
```

```
/home/annaanisienia/.local/lib/python3.6/site-packages/torch/nn/functional.py:1120:
UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

Out[17]:

```
tensor([[1., 1., 0.],
        [1., 1., 0.],
        [1., 1., 0.],
        [0., 1., 0.],
        [0., 1., 0.],
        [0., 1., 0.],
        [1., 1., 0.],
        [0., 1., 0.],
        [1., 1., 0.],
        [0., 1., 0.],
        [1., 1., 0.],
        [1., 1., 0.],
        [1., 1., 0.],
        [1., 1., 0.],
        [0., 1., 0.],
        [0., 1., 0.],
        [0., 1., 0.],
        [0., 1., 0.],
        [1., 1., 0.],
        [0., 1., 0.]])
```

In [18]:

```
len(df_val), len(preds[1])
```

Out[18]:

```
(20, 20)
```