Lesson 10

NLP Classification

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

概述

No parallel processing

Hard to do simple things (like multi-label classification)

No obvious way to save intermediate calcs

Somewhat convoluted API

由于torchtext有很多的缺点，这次实验以fastai.text为基础，大部分类和函数与lesson 4类似，以IMDB影评作为分类对象

目录 Contents

1 Standardize format

In [4]:
```
CLASSES = ['neg', 'pos', 'unsup']

def get_texts(path):
    texts,labels = [],[]
    for idx,label in enumerate(CLASSES):
        for fname in (path/label).glob('*.*'):
            texts.append(fname.open('r').read())
            labels.append(idx)
    return np.array(texts),np.array(labels)

trn_texts,trn_labels = get_texts(PATH/'train')
val_texts,val_labels = get_texts(PATH/'test')
```

In [5]:
```
len(trn_texts),len(val_texts)
```

Out[5]:
```
(75000, 25000)
```

读取IMDB影评的内容和标签，包含75000个训练集，25000个测试集，在训练集中有50000个没有标签

1 Standardize format

In [6]:
```
col_names = ['labels','text']
```

得到随机排列的数组

In [7]:
```
np.random.seed(42)
trn_idx = np.random.permutation(len(trn_texts))
val_idx = np.random.permutation(len(val_texts))
```

In [8]:
```
trn_texts = trn_texts[trn_idx]
val_texts = val_texts[val_idx]

trn_labels = trn_labels[trn_idx]
val_labels = val_labels[val_idx]
```

打乱排列，permutation按给定的长度生成一个随机排列的数组，打乱数据集能取得更好的训练效果

1 Standardize format

```
In  [3]: CLAS_PATH=Path('data/imdb_clas/')
         CLAS_PATH.mkdir(exist_ok=True)

         LM_PATH=Path('data/imdb_lm/')
         LM_PATH.mkdir(exist_ok=True)
```

两个路径：classification path和language model path。创建语言模型的时候用LM，情感分类的时候用classification path。

有两个区别， classification path的train.csv没有unsupervised数据集；LM的标签全是0

1 Standardize format

```
In [9]: df_trn = pd.DataFrame({'text':trn_texts, 'labels':trn_labels}, columns=col_names)
        df_val = pd.DataFrame({'text':val_texts, 'labels':val_labels}, columns=col_names)
```

去除标签为2的

```
In [10]: df_trn[df_trn['labels']!=2].to_csv(CLAS_PATH/'train.csv', header=False, index=False)
         df_val.to_csv(CLAS_PATH/'test.csv', header=False, index=False)

         (CLAS_PATH/'classes.txt').open('w').writelines(f'{o}\n' for o in CLASSES)
```

把训练集文件和测试集保存为CSV文件

1 Standardize format

In [11]:
```
trn_texts,val_texts = sklearn.model_selection.train_test_split(
    np.concatenate([trn_texts,val_texts]), test_size=0.1)
```

In [12]:
```
len(trn_texts), len(val_texts)
```

Out[12]:
```
(90000, 10000)
```

In [13]:
```
df_trn = pd.DataFrame({'text':trn_texts, 'labels':[0]*len(trn_texts)}, columns=col_names)
df_val = pd.DataFrame({'text':val_texts, 'labels':[0]*len(val_texts)}, columns=col_names)

df_trn.to_csv(LM_PATH/'train.csv', header=False, index=False)
df_val.to_csv(LM_PATH/'test.csv', header=False, index=False)
```

	labels	text
0	2	Directed by Leslie Howard(who also played Mitc...
1	0	God, I was bored out of my head as I watched t...
2	1	I just viewed this great good-natured parody o...
3	2	Being a horror movie buff, I tried really hard...
4	2	Sometimes the planets align and everything jus...

把数据集分成90000个训练集，10000个验证集，保存LM的CSV文件，路径是language model path

2 Language model tokens

```
In [14]: chunksize=24000
```

```
In [15]: re1 = re.compile(r'  +')

def fixup(x):
    x = x.replace('#39;', "'").replace('amp;', '&').replace('#146;', "'").replace(
        'nbsp;', ' ').replace('#36;', '$').replace('\\n', "\n").replace('quot;', "'").replace(
        '<br />', "\n").replace('\\"', '"').replace('<unk>','u_n').replace(' @.@ ','.').replace(
        ' @-@ ','-').replace('\\', ' \\ ')
    return re1.sub(' ', html.unescape(x))
```

```
In [16]: 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)
```

切分用的函数

```
In [17]: def get_all(df, n_lbls):
    tok, labels = [], []
    for i, r in enumerate(df):
        print(i)
        tok_, labels_ = get_texts(r, n_lbls)
        tok += tok_;
        labels += labels_
    return tok, labels
```

分词切片：把一长串字符变成类似单词的格式，比如把don’t变成do和n’t。并对句子中的特殊符号进行替换

2 Language model tokens

```
In [18]: df_trn = pd.read_csv(LM_PATH/'train.csv', header=None, chunksize=chunksize)
         df_val = pd.read_csv(LM_PATH/'test.csv', header=None, chunksize=chunksize)

In [19]: tok_trn, trn_labels = get_all(df_trn, 1)
         tok_val, val_labels = get_all(df_val, 1)

0
1
2
3
0

In [20]: (LM_PATH/'tmp').mkdir(exist_ok=True)

In [21]: np.save(LM_PATH/'tmp'/'tok_trn.npy', tok_trn)
         np.save(LM_PATH/'tmp'/'tok_val.npy', tok_val)

In [22]: tok_trn = np.load(LM_PATH/'tmp'/'tok_trn.npy')
         tok_val = np.load(LM_PATH/'tmp'/'tok_val.npy')
```

利用pandas读取文件时，chunksize能提高效率，get_texts把标签变成整数，切分是利用Spacy，调用了proc_all_mp进行多核切分，大大提高效率

2 Language model tokens

In [24]:
```
' '.join(tok_trn[0])
```

Out[24]:
```
'\n xbos xfld 1 yes hi " kill or be killed . " - sorry to only answer your posting now , but i \'ve only recently become aware of all this \' stuff \' on " kill and kill again , " on which i was the cinematographer and helped get the martial artists for the movie , since i knew them all tk_rep 4 . you are right . a lot of people preferred the first film also with james ryan as steve chase in " kill and kill again " , but i must tell you that the box - office figures tell a very different story ... in fact , depending on how old you are , you might remember a film made by bo derek \'s husband john derek called " 10 " ... a \' tits and bums \' vehicle to show off the many charms of his lovely wife that did very well at the box - office and was released the same year as " kill and kill again \' . well , believe it or not , " kill and kill again " made more money than " 10 " . much more money tk_rep 4 . i have the figures somewhere tk_rep 6 . anyhow both these movies have gone into the memories and archives of " chop suie \' or karate pot - boilers tk_rep 5 . one of the many things i admire about james ryan is that after these two big successes for him in the t_up usa , he was offered many more roles like that and could of happily gone on making them ad nauseum , going for the money and fame . but he declined tk_rep 4 . went back to south africa to become one of sa \'s best male leads in theater and tv / cinema ... a \' big up \' for steve chase tk_rep 8 . tai krige sasc .'
```

可以看到tok_trn被切分，并且句子开头是xbos，这是经过统计很少出现的字母组合，在此用作区分不同文章的标志。此处的t_up表示把大写转换成了小写，tk_rep是把一行相同的字符进行简写，因此，如果一行有29个！，就用tk_rep 29！表示。

2 Language model tokens

进行切分后下一个步骤是进行保存，把训练集合验证集都进行保存，具体代码如下：

```
In  [21]:  np.save(LM_PATH/'tmp'/'tok_trn.npy', tok_trn)
           np.save(LM_PATH/'tmp'/'tok_val.npy', tok_val)

In  [22]:  tok_trn = np.load(LM_PATH/'tmp'/'tok_trn.npy')
           tok_val = np.load(LM_PATH/'tmp'/'tok_val.npy')
```

2 Language model tokens

```
In [23]: freq = Counter(p for o in tok_trn for p in o)
         freq.most_common(25)

Out[23]: [('the', 1207090),
          ('.', 992211),
          (',', 985167),
          ('and', 586916),
          ('a', 583091),
          ('of', 524334),
          ('to', 484279),
          ('is', 393069),
          ('it', 341489),
          ('in', 337312),
          ('i', 308628),
          ('this', 270368),
          ('that', 261488),
          ('"', 235932),
          ("'s", 221186),
          ('-', 187912),
          ('was', 180721),
          ('\n\n', 178970),
          ('as', 165574),
          ('with', 158996),
          ('for', 158658),
          ('movie', 157452),
          ('but', 150260),
          ('film', 143987),
          ('you', 124188)]
```

Counter()把tok_trn的不同词进行统计，并标明出现的次数

2 Language model tokens

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

In [26]: itos = [o for o,c in freq.most_common(max_vocab) if c>min_freq]
         itos.insert(0, '_pad_')
         itos.insert(0, '_unk_')

In [27]: stoi = collections.defaultdict(lambda:0, {v:k for k,v in enumerate(itos)})
         len(itos)

Out[27]: 60002

In [28]: trn_lm = np.array([[stoi[o] for o in p] for p in tok_trn])
         val_lm = np.array([[stoi[o] for o in p] for p in tok_val])

In [29]: np.save(LM_PATH/'tmp'/'trn_ids.npy', trn_lm)
         np.save(LM_PATH/'tmp'/'val_ids.npy', val_lm)
         pickle.dump(itos, open(LM_PATH/'tmp'/'itos.pkl', 'wb'))

In [30]: trn_lm = np.load(LM_PATH/'tmp'/'trn_ids.npy')
         val_lm = np.load(LM_PATH/'tmp'/'val_ids.npy')
         itos = pickle.load(open(LM_PATH/'tmp'/'itos.pkl', 'rb'))

In [31]: ' '.join([str(val) for val in trn_lm[0]])

Out[31]: '40 41 42 39 441 5347 15 515 54 38 535 3 15 17 780 8 80 1557 146 11646 165 4 24 12 158 80
         28 15 515 5 515 192 4 15 28 78 12 18 2 4391 5 1663 98 2 1611 2783 22 2 23 4 253 12 699 11
         7 99 5490 2 105 25 102 21 623 1758 20 1336 1252 11 15 515 5 515 192 15 4 24 12 224 399 26
         285 81 91 11 212 4 5641 28 107 175 26 33 4 26 251 416 6 25 112 46 5035 3886 16 599 320 38
         13528 61 2249 8 140 142 2 128 6447 7 35 1300 334 14 86 69 88 44 2 884 17 1053 5 18 650 2
         88 4 283 10 54 32 4 15 515 5 515 192 15 112 68 307 92 15 183 15 3 93 68 307 206 225 3 12
         8 150 117 36 800 103 2 1891 5 14352 7 15 7493 0 61 54 4946 4206 17 48215 206 374 3 37 7 2
         4 118 150 126 219 10107 22 108 11 2 31 2477 4 34 18 2849 128 68 581 52 14 5 95 7 3169 800
         2 307 5 2012 3 24 34 14630 206 225 3 432 161 8 1238 2728 8 446 37 7 17987 16 138 895 813
         71 61 22 1336 1252 206 739 3 16049 18218 0 3'

In [32]: vs=len(itos)
         vs,len(trn_lm)

Out[32]: (60002, 90000)
```

把文本中出现次数少于2的去除，取文本中出现频率最高的60000个词保存在itos中，并插入_pad_，_unk_，便于后续保存没出现的词

2 Language model tokens

```
In [34]: em_sz,nh,nl = 400,1150,3

In [35]: PRE_PATH = PATH/'models'/'wt103'
         PRE_LM_PATH = PRE_PATH/'fwd_wt103.h5'

In [36]: wgts = torch.load(PRE_LM_PATH, map_location=lambda storage, loc: storage)

In [37]: enc_wgts = to_np(wgts['0.encoder.weight'])
         row_m = enc_wgts.mean(0)

In [38]: itos2 = pickle.load((PRE_PATH/'itos_wt103.pkl').open('rb'))
         stoi2 = collections.defaultdict(lambda:-1, {v:k for k,v in enumerate(itos2)})

In [39]: new_w = np.zeros((vs, em_sz), dtype=np.float32)
         for i,w in enumerate(itos):
             r = stoi2[w]
             new_w[i] = enc_wgts[r] if r>=0 else row_m

In [40]: 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))
```

调用已经训练好的wikitext103的权值，对itos中保存的词，如果在wikitext103中出现就使用该权重，如果没出现就使用所有权重的均值

Embedding matrix为60002x400

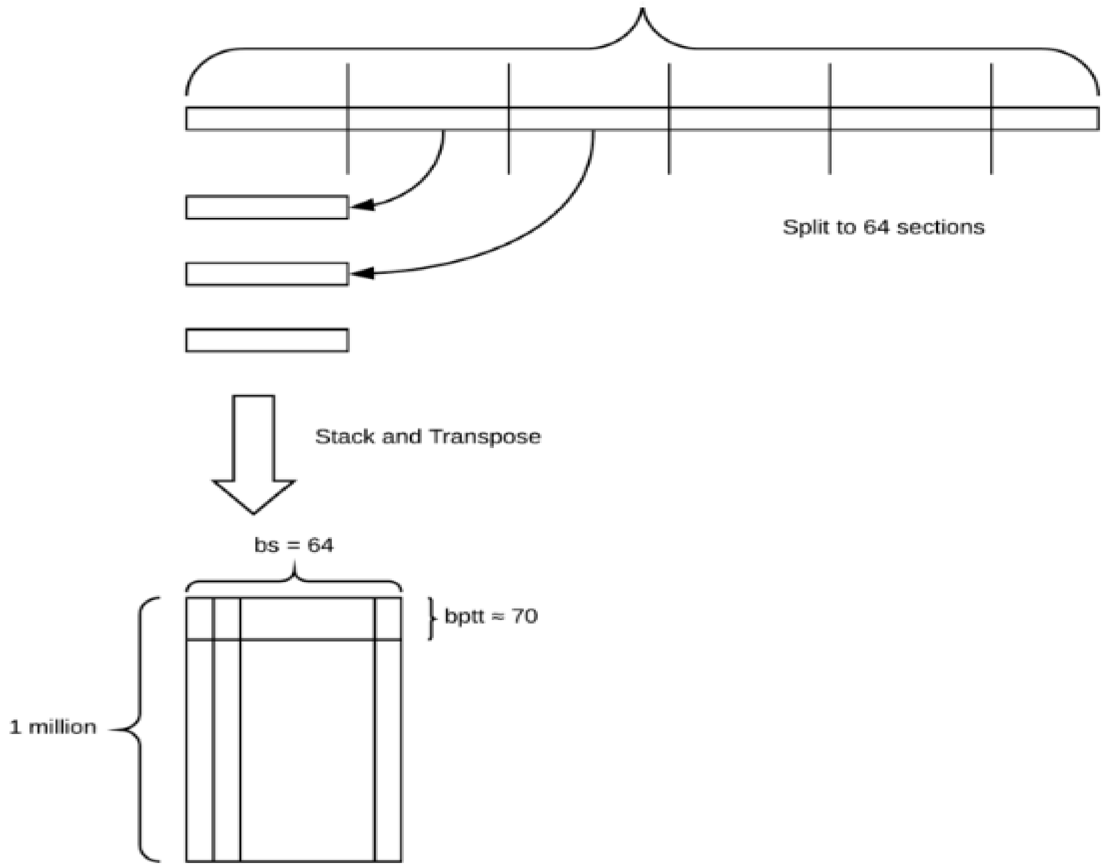
nl是隐藏层的数目，nh是每一层activation的个数

3 Language model

RNN → A linear layer with dropout

- 把所有文本连起来
- 设置dropout
- 创建learner
- 对单词进行预测

3 Language model



与lesson 4一样，把文本分成64个batch，取bptt为70

3 Language model

```
In [42]: trn_dl = LanguageModelLoader(np.concatenate(trn_lm), bs, bptt)
         val_dl = LanguageModelLoader(np.concatenate(val_lm), bs, bptt)
         md = LanguageModelData(PATH, 1, vs, trn_dl, val_dl, bs=bs, bptt=bptt)
```

```
In [43]: drops = np.array([0.25, 0.1, 0.2, 0.02, 0.15])*0.7
```

```
In [44]: learner= md.get_model(opt_fn, em_sz, nh, nl,
             dropouti=drops[0], dropout=drops[1], wdrop=drops[2], dropoute=drops[3], dropouth=drops[4])

         learner.metrics = [accuracy]
         learner.freeze_to(-1)
```

```
In [45]: learner.model.load_state_dict(wgts)
```

首先创建数据加载器，设置dropout,然后通过get_model建立learner，get_model主要调用get_language_model实现功能

3 Language model

```
class LanguageModelLoader():
    def __init__(self, nums, bs, bptt, backwards=False):
        self.bs,self.bptt,self.backwards = bs,bptt,backwards
        self.data = self.batchify(nums)
        self.i,self.iter = 0,0
        self.n = len(self.data)

    def __iter__(self):
        self.i,self.iter = 0,0
        while self.i < self.n-1 and self.iter<len(self):
            bptt = self.bptt if np.random.random() < 0.95 else self.bptt / 2.
            seq_len = max(5, int(np.random.normal(bptt, 5)))
            res = self.get_batch(self.i, seq_len)
            self.i += seq_len
            self.iter += 1
            yield res

    def __len__(self): return self.n // self.bptt - 1

    def batchify(self, data):
        nb = data.shape[0] // self.bs
        data = np.array(data[:nb*self.bs])
        data = data.reshape(self.bs, -1).T
        if self.backwards: data=data[::-1]
        return T(data)

    def get_batch(self, i, seq_len):
        source = self.data
        seq_len = min(seq_len, len(source) - 1 - i)
        return source[i:i+seq_len], source[i+1:i+1+seq_len].view(-1)
```

LanguageModelLoader先调用batchify()把整个文本分成64份，每份长度约为390k，然后通过迭代每次取长度为70的mini batch

3 Language model

```
bptt = self.bptt if np.random.random() < 0.95 else self.bptt / 2.
seq_len = max(5, int(np.random.normal(bptt, 5)))
res = self.get_batch(self.i, seq_len)
```

为了引入随机性，bptt的值不是固定的，这里95%的可能为70，5%的可能为35

3 Language model

然后建立了LanguageModelData类，传递了训练集，验证集，bptt以及标记的总数。

```
class LanguageModel(BasicModel):
    def get_layer_groups(self):
        m = self.model[0]
        return [*zip(m.rnns, m.dropouths), (self.model[1], m.dropouti)]


class LanguageModelData():
    def __init__(self, path, pad_idx, nt, trn_dl, val_dl, test_dl=None, bptt=70, backwards=False, **kwargs):
        self.path,self.pad_idx,self.nt = path,pad_idx,nt
        self.trn_dl,self.val_dl,self.test_dl = trn_dl,val_dl,test_dl

    def get_model(self, opt_fn, emb_sz, n_hid, n_layers, **kwargs):
        m = get_language_model(self.nt, emb_sz, n_hid, n_layers, self.pad_idx, **kwargs)
        model = LanguageModel(to_gpu(m))
        return RNN_Learner(self, model, opt_fn=opt_fn)
```

get_language_model()源码如下，其中RNN_Encoder创建embedding层以及三层LSTM，返回一个linear decoder，tie_weights=True表示encoder和decoder公用相同的权重

```
def get_language_model(n_tok, emb_sz, nhid, nlayers, pad_token,
                       dropout=0.4, dropouth=0.3, dropouti=0.5, dropoute=0.1, wdrop=0.5, tie_weights=True):
    """Returns a SequentialRNN model...."""

    rnn_enc = RNN_Encoder(n_tok, emb_sz, nhid=nhid, nlayers=nlayers, pad_token=pad_token,
                       dropouth=dropouth, dropouti=dropouti, dropoute=dropoute, wdrop=wdrop)
    enc = rnn_enc.encoder if tie_weights else None
    return SequentialRNN(rnn_enc, LinearDecoder(n_tok, emb_sz, dropout, tie_encoder=enc))
```

3 Language model

```
In [46]: lr=1e-3
         lrs = lr

In [47]: learner.fit(lrs/2, 1, wds=wd, use_clr=(32,2), cycle_len=1)
```

Epoch 100% 1/1 [20:21<00:00, 1221.24s/it]

epoch	trn_loss	val_loss	accuracy
0	4.683687	4.440973	0.258392

```
Out[47]: [array([4.44097]), 0.2583918860182166]

In [48]: learner.save('lm_last_ft')

In [49]: learner.load('lm_last_ft')

In [50]: learner.unfreeze()

In [51]: learner.lr_find(start_lr=lrs/10, end_lr=lrs*10, linear=True)
```

Epoch 100% 1/1 [23:32<00:00, 1412.74s/it]

epoch	trn_loss	val_loss	accuracy
0	4.761771	4.581846	0.2479

由于开始的encoder和decoder层的权值用的平均值，需要进行更新，解冻最后一层embedding decoder，进行训练后把更新的权值保存下来

3 Language model

In [53]:
```
learner.fit(lrs, 1, wds=wd, use_clr=(20,10), cycle_len=15)
```

Epoch 100% 15/15 [5:41:07<00:00, 1364.51s/it]

epoch	trn_loss	val_loss	accuracy
0	4.358673	4.16161	0.285604
1	4.25144	4.081562	0.29331
2	4.185964	4.034758	0.297899
3	4.138488	4.008867	0.300584
4	4.11093	3.990038	0.302639
5	4.079962	3.976144	0.304147
6	4.082877	3.961384	0.305554
7	4.03841	3.955175	0.306595
8	4.008811	3.946932	0.307697
9	4.00662	3.939671	0.308497
10	4.02867	3.928717	0.309533
11	3.980618	3.921993	0.31044
12	3.9583	3.918905	0.310883
13	3.924396	3.915851	0.311596
14	3.912441	3.911088	0.312248

Out[53]:
```
[array([3.91109]), 0.3122478389414027]
```

We save the trained model weights and separately save the encoder part of the LM model as we
classification task model.

In [54]:
```
learner.save('lm1')
```

In [55]:
```
learner.save_encoder('lm1_enc')
```

解冻所有层，设置epoch为15

保存encoder的权重

save()保存所有的权值，

save_encoder（）值保存encoder的权值

4 Classifier

In [57]:
```
df_trn = pd.read_csv(CLAS_PATH/'train.csv', header=None, chunksize=chunksize)
df_val = pd.read_csv(CLAS_PATH/'test.csv', header=None, chunksize=chunksize)
```

In [58]:
```
tok_trn, trn_labels = get_all(df_trn, 1)
tok_val, val_labels = get_all(df_val, 1)
```

```
0
1
0
1
```

In [59]:
```
(CLAS_PATH/'tmp').mkdir(exist_ok=True)

np.save(CLAS_PATH/'tmp'/'tok_trn.npy', tok_trn)
np.save(CLAS_PATH/'tmp'/'tok_val.npy', tok_val)

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

In [60]:
```
tok_trn = np.load(CLAS_PATH/'tmp'/'tok_trn.npy')
tok_val = np.load(CLAS_PATH/'tmp'/'tok_val.npy')
```

In [61]:
```
itos = pickle.load((LM_PATH/'tmp'/'itos.pkl').open('rb'))
stoi = collections.defaultdict(lambda:0, {v:k for k,v in enumerate(itos)})
len(itos)
```

Out[61]:
```
60002
```

In [62]:
```
trn_clas = np.array([[stoi[o] for o in p] for p in tok_trn])
val_clas = np.array([[stoi[o] for o in p] for p in tok_val])
```

In [63]:
```
np.save(CLAS_PATH/'tmp'/'trn_ids.npy', trn_clas)
np.save(CLAS_PATH/'tmp'/'val_ids.npy', val_clas)
```

读取CLAS_PATH下的数据，使用LM的60002个词的数据，因为要使用相同的encoder。

4 Classifier

首先导入数据，设置em_sz等，c是分类的个数

In [64]:
```
trn_clas = np.load(CLAS_PATH/'tmp'/'trn_ids.npy')
val_clas = np.load(CLAS_PATH/'tmp'/'val_ids.npy')
```

In [65]:
```
trn_labels = np.squeeze(np.load(CLAS_PATH/'tmp'/'trn_labels.npy'))
val_labels = np.squeeze(np.load(CLAS_PATH/'tmp'/'val_labels.npy'))
```

In [66]:
```
bptt,em_sz,nh,nl = 70,400,1150,3
vs = len(itos)
opt_fn = partial(optim.Adam, betas=(0.8, 0.99))
bs = 48
```

In [67]:
```
min_lbl = trn_labels.min()
trn_labels -= min_lbl
val_labels -= min_lbl
c=int(trn_labels.max())+1
```

4 Classifier

TextDataset把文本和标签放在一起

```
trn_ds = TextDataset(trn_clas, trn_labels)
val_ds = TextDataset(val_clas, val_labels)
```

```
class TextDataset(Dataset):
    def __init__(self, x, y, backwards=False, sos=None, eos=None):
        self.x,self.y,self.backwards,self.sos,self.eos = x,y,backwards,sos,eos

    def __getitem__(self, idx):
        x = self.x[idx]
        if self.backwards: x = list(reversed(x))
        if self.eos is not None: x = x + [self.eos]
        if self.sos is not None: x = [self.sos]+x
        return np.array(x),self.y[idx]

    def __len__(self): return len(self.x)
```

4 Classifier

DataLoader通过输入的取样器参数把验证集按长度从短到长排列，把训练集进行同样操作并加入了随机性，由于文本长度不一样，对文本进行填充。

```
trn_samp = SortishSampler(trn_clas, key=lambda x: len(trn_clas[x]), bs=bs//2)
val_samp = SortSampler(val_clas, key=lambda x: len(val_clas[x]))
trn_dl = DataLoader(trn_ds, bs//2, transpose=True, num_workers=1, pad_idx=1, sampler=trn_samp)
val_dl = DataLoader(val_ds, bs, transpose=True, num_workers=1, pad_idx=1, sampler=val_samp)
md = ModelData(PATH, trn_dl, val_dl)
```

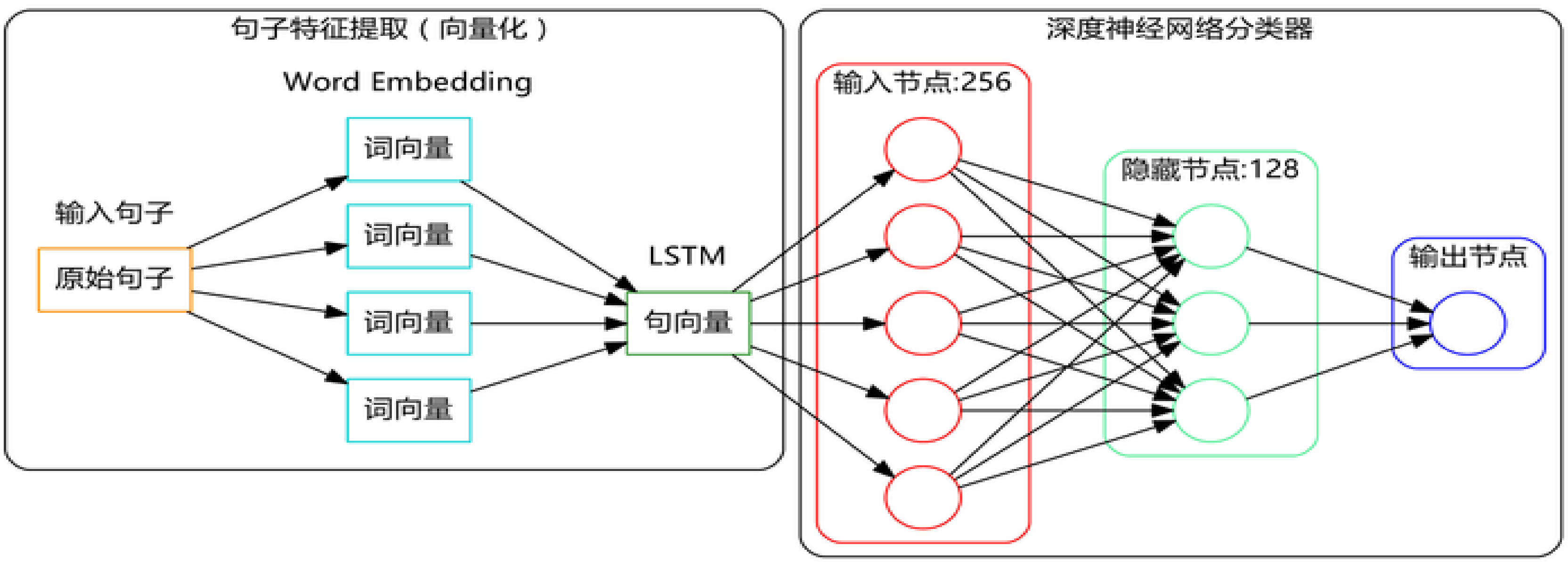
4 Classifier

get_rnn_classifier创建一个RNNencoder，并返回一个线性分类器

```
In [71]: m = get_rnn_classifier(bptt, 20*70, c, vs, emb_sz=em_sz, n_hid=nh, n_layers=nl, pad_token=1,
                   layers=[em_sz*3, 50, c], drops=[dps[4], 0.1],
                   dropouti=dps[0], wdrop=dps[1], dropoute=dps[2], dropouth=dps[3])
```

```
def get_rnn_classifer(bptt, max_seq, n_class, n_tok, emb_sz, n_hid, n_layers, pad_token, layers, drops, bidir=False,
                      dropouth=0.3, dropouti=0.5, dropoute=0.1, wdrop=0.5):
    rnn_enc = MultiBatchRNN(bptt, max_seq, n_tok, emb_sz, n_hid, n_layers, pad_token=pad_token, bidir=bidir,
                      dropouth=dropouth, dropouti=dropouti, dropoute=dropoute, wdrop=wdrop)
    return SequentialRNN(rnn_enc, PoolingLinearClassifier(layers, drops))
```

4 Classifier



layers=[em_sz*3, 50, c]
em_sz * 3：线性分类器的输入，联合池化，包括activations的平均值、最大值、最终值
50：第一层的输出
c：第二层的输出

4 Classifier

调用RNN_Learner建立学习器，RNN_Learner的loss function是cross_entropy

```
In  [73]: 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]
```

```
class RNN_Learner(Learner):
    def __init__(self, data, models, **kwargs):
        super().__init__(data, models, **kwargs)
        self.crit = F.cross_entropy
```

4 Classifier

In [74]:
```
lr=3e-3
lrm = 2.6
lrs = np.array([lr/(lrm**4), lr/(lrm**3), lr/(lrm**2), lr/lrm, lr])
```

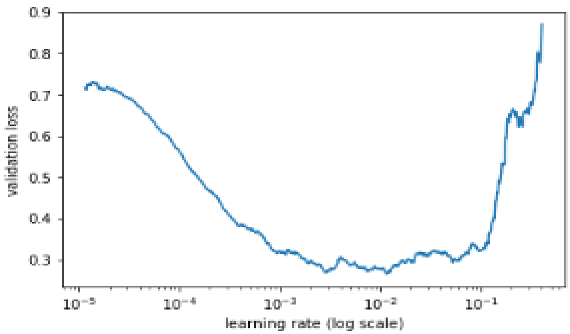
In [75]:
```
lrs=np.array([1e-4,1e-4,1e-4,1e-3,1e-2])
```

In [77]:
```
learn.freeze_to(-1)
```

In [78]:
```
learn.lr_find(lrs/1000)
learn.sched.plot()
```

Epoch 0% 0/1 [00:00<?, ?it/s]

77%|██████████ | 807/1042 [01:37<00:28, 8.30it/s, loss=1.06]



In [79]:
```
learn.fit(lrs, 1, wds=wd, cycle_len=1, use_clr=(8,3))
```

Epoch 100% 1/1 [03:36<00:00, 216.37s/it]

epoch	trn_loss	val_loss	accuracy
0	0.288433	0.18446	0.93184

Out[79]: [array([0.18446]), 0.9318400001525879]

各层设置不同的学习率，训练最后一层

4 Classifier

```
learn.save('clas_1')
```

```
learn.load('clas_1')
```

```
learn.unfreeze()
```

```
learn.fit(lrs, 1, wds=wd, cycle_len=14, use_clr=(32,10))
```

A Jupyter Widget

epoch	trn_loss	val_loss	accuracy
0	0.337347	0.186812	0.930782
1	0.284065	0.318038	0.932062
2	0.246721	0.156018	0.941747
3	0.252745	0.157223	0.944106
4	0.24023	0.159444	0.945393
5	0.210046	0.202856	0.942858
6	0.212139	0.149009	0.943746
7	0.21163	0.186739	0.946553
8	0.186233	0.1508	0.945218
9	0.176225	0.150472	0.947985
10	0.198024	0.146215	0.948345
11	0.20324	0.189206	0.948145
12	0.165159	0.151402	0.947745
13	0.165997	0.146615	0.947905

[0.14661488, 0.9479046703071374]

保存各层参数后解冻所有层进行训练

谢 谢！