

欢迎来到第四课

[Video\(YouTube\)](#) / [Video\(bilibili\)](#) / [Lesson Forum](#)

欢迎来到第四课！我们继续学习这些关键的应用来完成学习旅程。我们已经看了一系列的视觉应用。我们学了分类、定位、图片回归。我们初步接触了NLP。今天我们要深入学习NLP迁移学习。然后我们会学习表格数据和协同过滤，这些都是非常有用的应用。

然后我们会再反着讲一遍。我们会深入学习协同过滤，理解在数学层面究竟发生了什么，或者说在计算机里发生了什么。我们会用这种方式用相反的顺序来再讲一遍这些应用，来理解所有这些应用底层的原理。

更正CamVid结果

开始之前，先讲下，论坛上一些热心的人指出，在我们拿我们的结果和那个我们认为是最近的CamVid最佳成绩比较时，是做了不公平的比较，因为这个论文用的是这些分类里较小的一个子集，但我们用了所有的分类。学习小组里的Jason很热心，他用相同的子集做了实验，我们的正确率达到了94%，论文里取到的成绩是91.5%。我认为这是很酷的成绩，是仅仅使用现在的默认模型可以超过一两年前的最优成绩多么多的一个范例。去年我们开这个课程时它是最佳的成绩。这非常让人振奋。

自然语言处理 Natural Language Processing (NLP) [2:00]

我们回来看下NLP，理解究竟发生了什么。

快速回顾

首先，快速回顾下。NLP代表natural language processing自然语言处理。它处理文本，用文本做一些事情。文本分类是特别特别有用的应用。我们从它开始讲起。分类文本或者分类文档可以用来做所有这些事情：

- 垃圾邮件筛选
- 识别假新闻
- 从医学报告中生成诊断
- 识别Twitter推文的内容

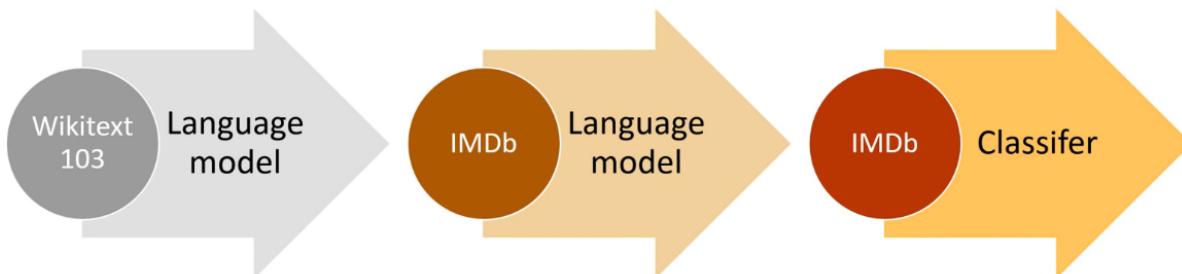
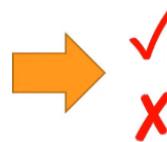
这很有趣。事实上这周有一个很好的例子，[@howkhang](#) 是我们的一个学员，他是一个律师，他在[论坛](#)里用NLP方法分类法律文本取得了很好的结果。我觉得这是一个很棒的例子。这是他们在这周的一个学术会议介绍这个方法的文章：



这里是我们要逐步深入学习的三个步骤

I'd like to eat a hot It was a hot

'This is a extremely well-made film. The acting, script and camera-work are all first-rate. The music is good, too, though it is mostly early in the film, when things are still relatively cheery. There are no really superstars in the cast, though several faces will be familiar. The entire cast does an excellent job with the script.'



我们要从这样的电影评论开始，判断它对电影的态度是正面的还是负面的。这就是我们要处理的问题。在训练集里有25,000个电影评论，里面每一个评论里，都有这样的信息：作者喜欢这个电影，或者作者不喜欢这个电影。今天这节课我们要学很多关于这个的细节。我们的神经网络（记住，它们只是一批矩阵相乘和把负值替换成0这样的非线性部分）里，这些权重矩阵是随机生成的。如果你用随机数初始化参数，然后训练这些参数来学习怎样识别电影评论是正面的还是负面的，你有25,000个0或者1，它们代表作者喜欢这个电影，不喜欢那个电影。通过这些信息来学习，显然不是不够的。懂得怎样说英语，并且要非常懂，才能完全识别他们是不是喜欢这个电影。有时它们是有细微差别的。尤其是对电影评论，像IMDB上的评论，人们会经常用讽刺。这会很难识别。

直到最近，事实上直到今年，神经网络在处理这种分类问题上表现都不够好。原因就是没有足够的信息。解决的方法，我们猜，是使用迁移学习。迁移学习经常是有效的方法。

去年的课程里我做了一些疯狂的尝试，我想试试迁移学习是不是在NLP上也是有效的。我成功了，效果非常好。现在，一年后的今天，迁移学习在NLP上绝对是个热点。我来讲下发生了什么。

在NLP上使用迁移学习 [6:04]

关键的部分是我们要使用一个预训练模型，就像在计算机视觉里做的那样，这个模型用来做的事情和我们现在想要做的事情不太一样。对于ImageNet，它原本是这样一个模型：用来预测图片是属于上千个类别中的哪一个类别。像你看到的那样，人们在它的基础上做微调，来处理各种不同的东西。我们会先从一个做不同事情的模型开始，它不是用来做电影评论分类，我们会从一个叫语言模型（language model）的预训练模型开始。

在NLP里语言模型有专门的含义。它是用来学习预测句子里的下一个单词的模型。要预测句子里的下一个单词，你要知道很多关于英语的知识（假设你在处理英语）和很多现实世界的知识。对于现实世界的知识，我举个例子：

这是你的语言模型，它读到了：

- "I'd like to eat a hot __": 显然是填 "dog"，对吗？
- "It was a hot __": 可能是 "day"

以前的NLP处理使用叫n-grams的方法，它基本上是讲这两个词或三个词可能一起出现的频率有多高。n-grams对这个问题的效果并不理想。你可以看到，没有足够的信息来决定下一个单词可能是什。但是用神经网络，你绝对可以做到。

这是很好的。如果你创建了一个神经网络来预测句子的下一个单词，你会有很多信息。不像对于每个2,000字的电影信息，只有“喜欢”或者“不喜欢”这些信息，对每个单词，你都可以预测下一个单词。所以在一个2,000字的评论里，有1,999个机会来预测下一个单词。更好的是，你能不只看电影评论。因为真正困难的不是“这个人喜不喜欢这个电影”，而是“怎样使用英语”。所以你可以从更大的文档集里学习

“怎样使用英语”。我们是从Wikipedia开始。

Wikitext 103 [8:30]

Stephen Merity和他的一些同事创建了一个叫做Wikitext 103的数据集，它是Wikipedia上一部分比较长的、做过预处理的文章。它可以被下载下来。这相当于从Wikipedia抓取数据。我用这个创建了一个语言模型。我只是创建了一个可以预测Wikipedia文章里下一个单词是什么的神经网络。这里有很多信息。如果我记得没错的话，它有十亿个token。所以我们有十亿个单独的东西要预测。每次我们预测错了，我们得到一个损失，我们得到梯度，然后更新权重，这样模型可以越来越好，直到能很好预测出Wikipedia的下一个单词。

为什么这很有用？因为我现在有了一个如何完成一个句子的模型，它知道很多关于英语和现实世界的知识，比如它知道hot在不同场景下的含义。它可以学习这样的东西"in 1996 in a speech to the United Nations, United States president _ said "... 现在它会是一个很好的语言模型，因为它需要知道在这一年谁是美国总统。所以能很好地训练语言模型是教会一个神经网络大量关于我们的世界是怎样的、世界上有什么、它们是怎样运作的这样的知识的很好的方法。这是一个吸引人的主题，它实际上是哲学家研究了几百年的问题。有一整套哲学理论研究我们可以从单单研究语言中学到什么。显然，能学到很多。

这很有趣。我们可以先用Wikipedia训练一个语言模型，你们可以用我们已经训练好的这个模型。就像用预训练的ImageNet模型做计算机视觉一样，我们现在可以用预训练的Wikitext模型做NLP，它本身很有用（预测句子里的下一个单词是有用的，但我们不是每次都做这个），更重要的是，它能理解很多语言和语言描述的事物。所以我们可以用它来做迁移学习，创建一个新的、可以预测影评里下一个单词的模型。

微调Wikitext来创建一个新的语言模型 [11:10]

如果我们可以用预训练的Wikitext模型创建一个能预测影评里下一个单词的语言模型，它会理解很多关于"my favorite actor is Tom ." 或者"I thought the photography was fantastic but I wasn't really so happy about the _ (director)." 它会学会很多怎样写影评的知识。它甚至会学会一些流行的电影名称之类的东西。

这意味着我们可以使用一个包含大量影评的语料库来学习怎样写影评，即使我们不知道这些影评是正面的还是负面的。所以预训练和微调语言模型时，我们不需要任何标签。这就是Yann LeCun说的**自监督学习 (self supervised learning)** 的。换句话说，它是经典监督模型 (classic supervised model)，我们有标签，但标签不是人工创建的。是模型自己添加到数据集里的。这非常好。因为现在我们得到了能很好地理解影评的东西，我们可以用迁移学习来微调它，来做我们想做的事情，这个例子是分类影评是正面还是负面。去年做这个任务时，我希望25,000个0或者1对于微调模型来说是足够的样本，最终确实是这样。

提问: 语言模型对口语、拼写错误的单词、俚语或者缩写（比如s6代表Samsung S6）这样东西有效吗？ [12:47]

是的，绝对有效。如果你在wikitext模型基础上，让它适配你的目标语料库。语料库 (Corpus) 就是一批文档（邮件、推文、医疗报告，或者其他文档）。你可以微调它，它就能学习俚语、缩写，或者语料库里没有出现的东西。有意思的是，这是去年我们做这项研究时，震惊到我们的一件事。人们认为从Wikipedia学习不会很有用，因为它不能代表所有人的写作方式。但是结果非常有效，因为Wikipedia和随机生成的单词之间的差别比Wikipedia和reddit之间的差别大得多。所以在99%的场景中，它都是有效的。

语言模型本身很有用。比如这篇SwiftKey（做手机输入法的团队）的**博客**讲了它们怎样用神经网络重写了他们的底层模型。这是一两年前的事。现在大部分手机输入法都这样做。你在手机上打字时，会有你想输入的下一个单词的提示，这就是你手机里的语言模型。

另外一个例子是研究者Andrey Karpathy，他现在特斯拉工作，在他读博士时，他创建了[一个关于LaTeX文档中文本的语言模型](#)，他创建了这个LaTeX文档自动生成工具，自动生成了这些论文。这很厉害。

我们不是特别关心语言模型的输出，我们是因为他对这个过程有帮助才对它感兴趣。

回顾基本过程 [15:14]

上周我们简单看过这个过程。先取到某种格式的数据。比如，我们准备了IMDB sample数据集，它们是在CSV文件里的。你可以用Pandas读取它，这里有正面或者负面的标签、每个影评的文本、代表是否属于验证集的布尔值。

```
1 | path = untar_data(URLs.IMDB_SAMPLE)
2 | path.ls()
```

```
1 | [PosixPath('/home/jhoward/.fastai/data/imdb_sample/texts.csv'),
2 | PosixPath('/home/jhoward/.fastai/data/imdb_sample/models')]
```

```
1 | df = pd.read_csv(path/'texts.csv')
2 | df.head()
```

	label	text	is_valid
0	negative	Un-bleeping-believable! Meg Ryan doesn't even ...	False
1	positive	This is a extremely well-made film. The acting...	False
2	negative	Every once in a long while a movie will come a...	False
3	positive	Name just says it all. I watched this movie wi...	False
4	negative	This movie succeeds at being one of the most u...	False

这是一个影评的例子：

```
1 | df['text'][1]
```

```
1 | 'This is a extremely well-made film. The acting, script and camera-work are all first-rate. The music is good, too, though it is mostly early in the film, when things are still relatively cheery. There are no really superstars in the cast, though several faces will be familiar. The entire cast does an excellent job with the script.<br /><br />But it is hard to watch, because there is no good end to a situation like the one presented. It is now fashionable to blame the British for setting Hindus and Muslims against each other, and then cruelly separating them into two countries. There is some merit in this view, but it\'s also true that no one forced Hindus and Muslims in the region to mistreat each other as they did around the time of partition. It seems more likely that the British simply saw the tensions between the religions and were clever enough to exploit them to their own ends.<br /><br />The result is that there is much cruelty and inhumanity in the situation and this is very unpleasant to remember and to see on the screen. But it is never painted as a black-and-white case. There is baseness and nobility on both sides, and also the hope for change in the younger generation.<br /><br />There is redemption of a sort, in the end, when Puro has to make a hard choice between a man who has ruined her life, but also truly loved her, and her family which has disowned her, then later come looking for her. But by that point, she has no option that is without great pain for her.<br /><br />This film carries the message that both Muslims and Hindus have their grave faults, and also that both can be dignified and caring people. The reality of partition makes that realisation all the more wrenching, since there can never be real reconciliation across the India/Pakistan border. In that sense, it is similar to "Mr & Mrs Iyer".<br /><br />In the end, we were glad to have seen the film, even though the resolution was heartbreaking. If the UK and US could deal with their own histories of racism with this kind of frankness, they would certainly be better off.'
```

你可以用 `TextDataBunch.from_csv` 来取一个语言模型data bunch:

```
1 | data_lm = TextDataBunch.from_csv(path, 'texts.csv')
```

然后使用和以前一样的方式创建一个learner，然后使用fit方法。

```
1 | data_lm.save()
```

你可以保存data bunch。下次就不用再做预处理了，只加载它就行。

```
1 | data = TextDataBunch.load(path)
```

如果我们把它作为一个分类data bunch来加载，会发生什么（我们还能看到标签）？

```
1 | data = TextClasDataBunch.load(path)
2 | data.show_batch()
```

text	label
xxbos xxfld 1 raising victor vargas : a review \n\n you know , raising victor vargas is like sticking your hands into a big , xxunk bowl of xxunk . it 's warm and gooey , but you 're not sure if it feels right . try as i might	negative
xxbos xxfld 1 now that che(2008) has finished its relatively short australian cinema run (extremely limited xxunk screen in xxunk , after xxunk) , i can xxunk join both xxunk of " at the movies " in taking steven soderbergh to task . \n\n it 's usually	negative
xxbos xxfld 1 many xxunk that this is n't just a classic due to the fact that it 's the first xxup 3d game , or even the first xxunk - up . it 's also one of the first xxunk games , one of the xxunk definitely the first	positive
xxbos xxfld 1 i really wanted to love this show . i truly , honestly did . \n\n for the first time , gay viewers get their own version of the " the bachelor " . with the help of his obligatory " hag " xxunk , james , a	negative
xxbos xxfld 1 this film sat on my xxunk for weeks before i watched it . i xxunk a self - indulgent xxunk flick about relationships gone bad . i was wrong ; this was an xxunk xxunk into the xxunk - up xxunk of new xxunk . \n\n the	positive

就像我们讲过的，它为单词的每个单独部分创建了一个单独的单元（“分词token”）。大部分还是单词，但是有时，如果是 it's 里的 's 这种东西，它就是一个单独的分词。每一个标点都是一个分词（逗号、句号，等等）。

我们做的下一步是编号（numericalization），我们找出所有出现过的分词，去除重复的，这是一个很大的列表。这是里面出现次数最多的10个：

```
1 | data.vocab.itos[:10]
```

```
1 | ['xxunk', 'xxpad', 'the', ',', '.', 'and', 'a', 'of', 'to', 'is']
```

这个去重过的唯一分词的巨大列表被叫做 vocabulary（单词表），我们叫它“vocab”。然后我要做的是用它在vocab里的ID替代这个分词。

```
1 | data.train_ds[0][0]
```

```
1 | Text xxbos xxfld 1 he now has a name , an identity , some memories and a a lost girlfriend . all he wanted was to disappear , but still , they xxunk him and destroyed the world he hardly built . now he wants some explanation , and to get ride of the people how made him what he is . yeah , jason bourne is back , and this time , he 's here with a vengeance .
```

```
1 | data.train_ds[0][0].data[:10]
```

```
1 | array([ 43,  44,  40,  34, 171,  62,    6, 352,    3,  47])
```

这就是编号。你会学到，vocab里的每个单词在神经网络的权重矩阵里都会占用一行。为了避免矩阵变得太大，我们限制单词表不要超过60,000（默认值）。如果一个单词出现次数不超过两次，我们就不把它放进vocab。这样我们把vocab控制在一个合理的大小。当你看到这些xxunk时，它代表一个未知的token。它是一个不常用的token，所以没有进入我们的单词表。

我们还有另外两个的特殊token（访问`fastai.text.transform.py`来查看最新内容）：

- `xxfld`: 每个标题、总结、摘要、正文（文档中的单独部分）会有一个单独的字段，它们会被编号（例如`xxfld_2`）。
- `xxup`: 如果有大写单词，它会被转成小写，并把一个`xxup`加到后面。

使用data block API [18:31]

我个人更喜欢使用data block API，因为这样创建data bunch时，要记住的东西更少，用起来更灵活。

```
1 data = (TextList.from_csv(path, 'texts.csv', cols='text')
2         .split_from_df(col=2)
3         .label_from_df(cols=0)
4         .databunch())
```

另外一个要生成data bunch的方法是决定这些：

- 你在创建什么类型的list（你的自变量是什么）？这个例子里，自变量是文本
- 数据来自什么地方？一个CSV文件
- 你想怎样区分验证集和训练集？这个例子里，用第二列`is_valid`标记
- 你要怎样标注它？第0列有positive或者negative
- 然后把它转成一个data bunch

还是在做相同的事情。

```
1 path = untar_data(URLs.IMDB)
2 path.ls()
```

```
1 [PosixPath('/home/jhoward/.fastai/data/imdb/imdb.vocab'),
2  PosixPath('/home/jhoward/.fastai/data/imdb/models'),
3  PosixPath('/home/jhoward/.fastai/data/imdb/tmp_lm'),
4  PosixPath('/home/jhoward/.fastai/data/imdb/train'),
5  PosixPath('/home/jhoward/.fastai/data/imdb/test'),
6  PosixPath('/home/jhoward/.fastai/data/imdb/README'),
7  PosixPath('/home/jhoward/.fastai/data/imdb/tmp_clas')]
```

```
1 (path/'train').ls()
```

```
1 [PosixPath('/home/jhoward/.fastai/data/imdb/train/pos'),
2  PosixPath('/home/jhoward/.fastai/data/imdb/train/unsup'),
3  PosixPath('/home/jhoward/.fastai/data/imdb/train/unsupBow.feat'),
4  PosixPath('/home/jhoward/.fastai/data/imdb/train/labeledBow.feat'),
5  PosixPath('/home/jhoward/.fastai/data/imdb/train/neg')]
```

现在我们取出整个数据集，里面包含：

- 训练集的25,000个评论
- 验证集的25,000个评论

- 50,000个没有标签的评论 (unsupervised movie reviews)

语言模型 [19:44]

我们要用语言模型开始。现在，好消息是，我们不用再创建Wikitext 103语言模型了。这并不难，你可以下载wikitext 103语料 (corpus)，运行相同的代码。但是在一个不错的GPU上，它要花费两三天时间，所以你们不必再做，可以用我们之前训练好的。即使你有医学文档或者法律文档之类的大语料库，你还是应该先使用Wikitext 103。没有理由使用随机权重。如果可能的话，尽量使用迁移学习。

然后我们开始微调我们的IMDB语言模型。

```
1 | bs=48
```

```
1 | data_lm = (TextList.from_folder(path)
2 |         #输入: path目录的所有文本文件
3 |         .filter_by_folder(include=['train', 'test'])
4 |         #目录下可能有其他的包含文本文件的临时文件夹。我们只使用train和test文件夹下的
5 |         .random_split_by_pct(0.1)
6 |         #随机分割，使用10%的数据（10,000个评论）作为验证集
7 |         .label_for_lm()
8 |         #我们要做语言模型，所以我们用相应的标签
9 |         .databunch(bs=bs))
10 | data_lm.save('tmp_lm')
```

我们这样写：

- 这是一个TextList。完整的IMDB实际上不是一个CSV。每个文档是一个单独的文本文件
- 告诉它数据在哪里，这个例子里我们要包含 train 和 test 文件夹
- 我们用0.1的比例随机分割它

这个10%要注意下。为什么用10%随机分割，而不是用他们提供的已经定义好的训练集和验证集呢？这是迁移学习里一个很酷的地方。尽管我们需要把验证集分离出来，实质上，只是需要分离出标签。所以我们不能使用验证集里的标签。想想在Kaggle竞赛里，你当然不能使用标签，他们也不会给你。但是你可以使用自变量。这个例子里，你可以完全可以使用验证集里的文本来训练语言模型。这是一个好方法：当你做语言模型时，把验证集和训练集连在一起，然后仅仅分离出一个小的验证集，这样你可以得到更多的数据来训练你的语言模型。这是一个小trick。

如果你在Kaggle上做NLP，或者你得到了一个标记过的数据的一个小的子集，要在你的语言模型里使用你拥有的所有文本，没有不使用的理由。

- 我们要怎样标注它？记住，语言模型有它自己的标签。文本本身就是标签，所以label for a language model 方法 (label_for_lm) 可以为我们生成标签
- 创建data bunch然后保存它。做分词和编号要花几分钟

几分钟之后，我们保存它。以后你可以直接加载它，不用再运行它。

```
1 | data_lm = TextLMDataBunch.load(path, 'tmp_lm', bs=bs)
```

```
1 | data_lm.show_batch()
```

idx	text
0	xxbos after seeing the truman show , i wanted to see the other films by weir . i would say this is a good one to start with . the plot : \n\n the wife of a doctor (who is trying to impress his bosses , so he can get xxunk trying to finish her written course , while he s at work . but one day a strange man , who says that he s a plumber , tells her he s been called out to repair some pipes in there flat .
1	and turn to the wisdom of homeless people & ghosts . that 's a good plan . i would never recommend this movie ; partly because the sexual content is unnecessarily graphic , but also because it really does n't offer any valuable insight . check out " yentl " if you want to see a much more useful treatment of jewish tradition at odds with society . xxbos creep is the story of kate (potente) , an intensely unlikeable bourgeois bitch that finds herself somehow sleeping through the noise of the last
2	been done before but there is something about the way its done here that lifts it up from the rest of the pack . \n\n 8 out of 10 for dinosaur / monster lovers . xxbos i rented this movie to see how the sony xxunk camera shoots , (i recently purchased the same camera) and was blown away by the story and the acting . the directing , acting , editing was all above what i expected from what appeared at first glance to be a " low budget " type of
3	troubles . nigel and xxunk are the perfect team , i 'd watch their show any day ! i was so crushed when they removed it , and anytime they had it on xxup tv after that i was over the moon ! they put it on on demand one summer (only the first eight episodes or so) and i 'd spend whole afternoons watching them one after the other ... but the worst part ? it is now back on a channel called xxunk - and xxup it xxup 's xxup on
4	movie !) the movie is about edward , a obsessive - compulsive , nice guy , who happens to be a film editor . he is then lent to another department in the building , and he is sent to the posh yet violent world of sam campbell , the splatter and gore department . sam campbell , eddy 's new boss , is telling eddy about the big break on his movies , the gruesome loose limbs series , and he needs eddy to make the movie somewhat less violent so they can

训练 [22:29]

剩下的东西就看起来比较熟悉了。我们创建了一个learner:

```
1 | learn = language_model_learner(data_lm, pretrained_model=URLs.WT103,
drop_mult=0.3)
```

不再是创建一个CNN learner, 我们要创建一个语言模型learner。实质上, 它不再是创建一个CNN (convolutional neural network卷积神经网络), 它这次是要创建一个RNN (recurrent neural network递归神经网络)。后面的课程里我们要完整地学习它们是怎样创建的。但是简单来说, 它们的结构是相同的。输入进入一个权重矩阵 (做矩阵乘法), 然后你用0替换负数。然后进入另一个矩阵乘法, 这样做很多次。还是相同的基本结构。

像往常一样, 当我们创建learner时, 你要传入两个东西。

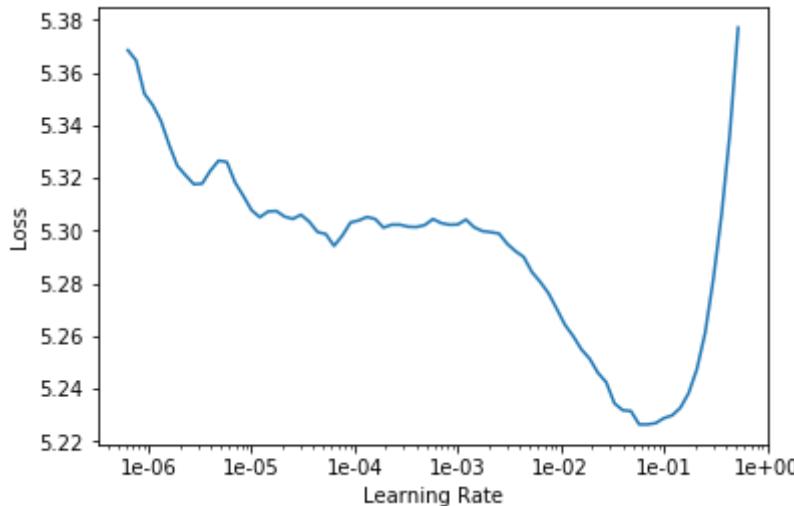
- 数据: 这就是我们的语言模型数据
- 我们要使用什么预训练模型: 这里, 预训练模型是Wikitext 103, 如果你第一次使用它的话, 它会从fastai上被下载下来。就和ImageNet预训练模型一样

这个 (`drop_mult=0.3`) 设置了dropout的数量。我们还没有讲过这个。我们简单讲过，有个叫正则化 (regularization) 的东西，你可以减少正则化来避免欠拟合。现在，只需要知道，使用一个小于一的数，因为当我第一次运行它时，我欠拟合了。如果你减小这个值，会避免欠拟合。

好了，我们得到了一个learner。我们可以用`lr_find`，看起来很标准。

```
1 | learn.lr_find()
```

```
1 | learn.recorder.plot(skip_end=15)
```



然后我们可以用fit one cycle。

```
1 | learn.fit_one_cycle(1, 1e-2, mom=[0.8, 0.7])
```

```
1 | Total time: 12:42
2 | epoch  train_loss  valid_loss  accuracy
3 | 1      4.591534    4.429290    0.251909  (12:42)
```

这里我们仅仅微调最后一层。通常，我们微调过最后一层后，下一件事是执行`unfreeze`来训练整个模型。就是这样：

```
1 | learn.unfreeze()
2 | learn.fit_one_cycle(10, 1e-3, mom=[0.8, 0.7])
```

```
1 | Total time: 2:22:17
2 | epoch  train_loss  valid_loss  accuracy
3 | 1      4.307920    4.245430    0.271067  (14:14)
4 | 2      4.253745    4.162714    0.281017  (14:13)
5 | 3      4.166390    4.114120    0.287092  (14:14)
6 | 4      4.099329    4.068735    0.292060  (14:10)
7 | 5      4.048801    4.035339    0.295645  (14:12)
8 | 6      3.980410    4.009860    0.298551  (14:12)
9 | 7      3.947437    3.991286    0.300850  (14:14)
10 | 8     3.897383    3.977569    0.302463  (14:15)
11 | 9     3.866736    3.972447    0.303147  (14:14)
12 | 10    3.847952    3.972852    0.303105  (14:15)
```

像你看到的那样，即使是在强大的GPU上，这也花了两三个小时。事实上，我还是欠拟合的。可能今天晚上，我还会整夜训练它，看看能不能好些。我猜我可以训练久些，因为你可以看到准确率还没有下降。所以我还会训练更久些。但是这个准确率值得注意，0.3代表我们在三分之一的预测里都可以猜出影评里的下一个单词。这听起来非常高，这代表你可以猜出下一个单词很多次。对于范围更有限的领域文档（比如医学和法律文本），你会发现准确率高得多。有时可以达到50%或者更高。但是0.3也已经很好了。

使用语言模型预测 [25:43]

现在你可以执行`learn.predict`，传入一个句子的前面部分，它会尝试为你补全这个句子。

```
1 | learn.predict('I liked this movie because ', 100, temperature=1.1,  
min_p=0.001)
```

```
1 | Total time: 00:10
```

```
1 | 'I liked this movie because of course after yeah funny later that the world  
reason settings - the movie that perfect the kill of the same plot - a  
mention of the most of course . do xxup diamonds and the " xxup disappeared  
kill of course and the movie niece , from the care more the story of the let  
character , " i was a lot \'s the little performance is not only . the  
excellent for the most of course , with the minutes night on the into movies  
( ! , in the movie its the first ever ! \n\n a'
```

需要讲下，它不是按照文本生成系统的目标来设计的。它是用来检查模型能不能生成一些看起来合理的句子的。要生成高质量的文本，有很多很多技巧，我们这里没有用这些。你可以看出它生成的不是随机的单词。看起来像是含糊的英语，没有表达出什么实际的含义。

现在我们有了一个影评模型。现在我们要保存它，这样才能把它加载到我们的分类器里（让它作为分类器的预训练模型）。但是我不想保存整个模型。语言模型的后半部分都是用来预测下一个单词的，而不是用来理解句子的。专门用来理解句子的部分被叫作**encoder**，我只保存这一部分（理解句子的部分，而不是生成下一个单词的部分）。

```
1 | learn.save_encoder('fine_tuned_enc')
```

分类 [27:18]

现在我们可以创建分类器了。第一步，像以前一样，是创建一个data bunch，我们要做的是完全相同的事情：

```

1 data_clas = (TextList.from_folder(path, vocab=data_lm.vocab)
2     #grab all the text files in path
3     .split_by_folder(valid='test')
4     #split by train and valid folder (that only keeps 'train' and
5     #'test' so no need to filter)
6     .label_from_folder(classes=['neg', 'pos'])
7     #remove docs with labels not in above list (i.e. 'unsup')
8     .filter_missing_y()
9     #label them all with their folders
10    .databunch(bs=50))
11 data_clas.save('tmp_clas')

```

我们要保证它使用和语言模型一样的单词表。如果语言模型里第10个单词是 `the`，我们要保证分类器里第10个单词也是 `the`。不然的话，预训练模型就没有意义了。这就是为什么我们传入语言模型的单词表来保证data bunch用相同的单词表，这是一个重要的步骤。

`split_by_folder`，记的吗，上次我们随机分割的，但是这次我们需要保证不使用测试集的标签。所以我们按文件夹分割。

这次我们不是把它作为一个语言模型来标注，而是要标注这些分类(`['neg', 'pos']`)。最后，创建一个data bunch。

有时你可能会遇到GPU内存溢出。我在一个11G的机器上运行它。如果你遇到了内存溢出，需要把这个值(`bs`)调小一点。你可能也需要重启notebook，然后从这里开始运行(classifier section)。Batch size设成50可以在11G的GPU上运行。如果你使用AWS的p2或者p3，或者Google的K80，我想你可能会有16G内存，这样你可以把它调大一点，设成64。你需要找到适合你的显卡的batch size。

这就是我们的data bunch：

```

1 data_clas = TextClasDataBunch.load(path, 'tmp_clas', bs=bs)
2 data_clas.show_batch()

```

text	label
xxfld 1 match 1 : tag team table match bubba ray and spike dudley vs eddie guerrero and chris benoit bubba ray and spike dudley started things off with a tag team table match against eddie guerrero and chris benoit . according to the rules of the match , both	pos
xxfld 1 i have never seen any of spike lee 's prior films , as their trailers never caught my interest . i have seen , and admire denzel washington , and jodie foster 's work , and have several of their dvds . i was , however , entirely	neg
xxfld 1 pier paolo pasolini , or pee - pee - pee as i prefer to call him (due to his love of showing male genitals) , is perhaps xxup the most overrated european marxist director - and they are thick on the ground . how anyone can	neg
xxfld 1 chris rock deserves better than he gives himself in " down to earth . " as directed by brothers chris & paul weitz of " american pie " fame , this uninspired remake of warren beatty 's 1978 fantasy " heaven can wait , " itself a rehash	neg
xxfld 1 yesterday , i went to the monthly antique flea market that comes to town . i really have no interest in such things , but i went for the fellowship of friends who do have such an interest . looking over the hundreds of vendor , passing many	pos

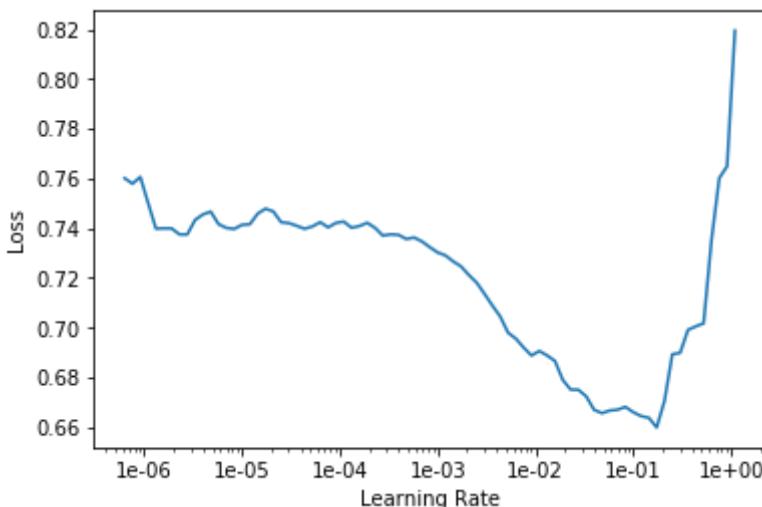
```
1 | learn = text_classifier_learner(data_clas, drop_mult=0.5)
2 | learn.load_encoder('fine_tuned_enc')
3 | learn.freeze()
```

这次，我们不再是创建一个语言模型learner，我们是在创建一个文本分类模型learner。后面的东西还是一样的，传入我们需要处理的数据，设定正则化的程度。如果你遇到了过拟合，你可以减小这个值(drop_mult)。如果你遇到了欠拟合，你可以增大这个值。最重要的是，加载我们的预训练模型。记住，这只是模型中叫作encoder的那一半，这才是我们需要加载的。

然后调用freeze，运行lr_find，找出学习率，训练一会儿。

```
1 | learn.lr_find()
```

```
1 | learn.recorder.plot()
```



```
1 | learn.fit_one_cycle(1, 2e-2, mom=(0.8,0.7))
```

```
1 | Total time: 02:46
2 | epoch  train_loss  valid_loss  accuracy
3 |   1      0.294225     0.210385    0.918960  (02:46)
```

在不到三分钟的训练之后，我们得到了92%的准确率。这很好。在你的特定领域里（无论是法律、医药、新闻、政务等等），你只需要训练领域语言模型一次。这可能要花一晚上的时间来训练。但是一旦你训练好了，你就可以用它快速地创建各种分类器和模型了。这里，我们用了三分钟就得到了一个很好地模型。当你第一次做这个的时候，可能会觉得花四个小时甚至更久来创建一个语言模型有点无聊。但是对一个你关注的领域，你只需要做一次这样的事。然后，你只要花几分钟，就可以在它的基础上构建很多不同的分类器和其它模型。

```
1 | learn.save('first')
```

```
1 | learn.load('first');
```

```
1 | learn.freeze_to(-2)
2 | learn.fit_one_cycle(1, slice(1e-2/(2.6**4), 1e-2), mom=(0.8,0.7))
```

```
1 | Total time: 03:03
2 | epoch  train_loss  valid_loss  accuracy
3 | 1       0.268781    0.180993    0.930760  (03:03)
```

我们把它保存下来，这样下次就不用再从头训练了。

这里需要注意下。我没有用 `unfreeze`，而是用了 `freeze_to`。它是说只解冻最后两层，不要解冻整个模型。我们发现不解冻整个模型，一次只解冻一层对文本分类更有效。

- 解冻最后两层
- 训练一下
- 再解冻下一层
- 训练一下
- 解冻整个模型
- 训练一下

```
1 | learn.save('second')
```

```
1 | learn.load('second');
```

```
1 | learn.freeze_to(-3)
2 | learn.fit_one_cycle(1, slice(5e-3/(2.6**4), 5e-3), mom=(0.8, 0.7))
```

```
1 | Total time: 04:06
2 | epoch  train_loss  valid_loss  accuracy
3 | 1       0.211133    0.161494    0.941280  (04:06)
```

```
1 | learn.save('third')
```

```
1 | learn.load('third');
```

```
1 | learn.unfreeze()
2 | learn.fit_one_cycle(2, slice(1e-3/(2.6**4), 1e-3), mom=(0.8, 0.7))
```

```
1 | Total time: 10:01
2 | epoch  train_loss  valid_loss  accuracy
3 | 1       0.188145    0.155038    0.942480  (05:00)
4 | 2       0.159475    0.153531    0.944040  (05:01)
```

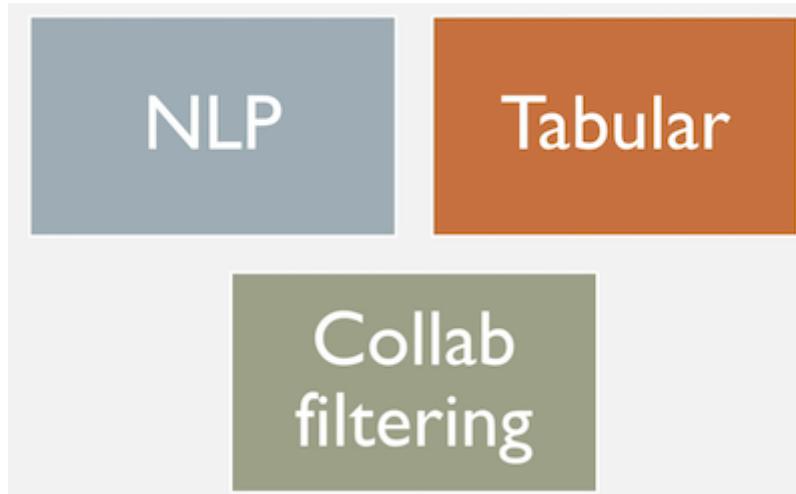
可以看到我传入这个 (`mom=(0.8, 0.7)`)，把momentum (动量) 设成了0.8、0.7。下周我们会学习它代表什么。我们可以自动生成它。可能在你看到这个视频的时候，我们不再需要传入这个值了。我们发现对训练递归神经网络来说，把momentum减少一点会有帮助。就是这样。

在接近半小时的训练后，它得到了94.4%的准确率。训练分类器的时间少很多。如果使用一些trick (技巧) 的话，效果还会更好些。我不确定我们会不会在课程第一部分讲所有这些trick。可能会是在下一个部分。不过，单单是这个标准的方法取到的结果也非常不错。

IMDb	<i>BCN+Char+CoVe [Ours]</i>	91.8
	SA-LSTM [Dai and Le, 2015]	92.8
	bmLSTM [Radford et al., 2017]	92.9
	TRNN [Dieng et al., 2016]	93.8
	oh-LSTM [Johnson and Zhang, 2016]	94.1
	Virtual [Miyato et al., 2017]	94.1

如果我们和去年的IMDb最佳成绩比较，这个成绩出自Salesforce Research的McCann et al的[CoVe论文](#)。他们的论文取到了91.8%的准确率。他们找到了一篇2017年的特定领域的情绪分析论文，里面得到了94.1%的正确率。现在，我们得到了94.4%。我能够构建的最好模型的成绩达到了95.1%。如果你要做文本分类，这个标准的迁移学习方法效果很好。

表格数据 (tabular data) [\[33:10\]](#)



这就是NLP。这个课程里我们会学习更多NLP的内容。现在我要换个主题，看看表格数据。表格数据是很值得关注的，因为，对于大部分人，它是你们每天都在电子表格，关系型数据库里处理的东西。

提问: 学习率里的这个魔法数字 2.6^4 是怎么来的? [\[33:38\]](#)

```
1 | learn.fit_one_cycle(2, slice(1e-3/(2.6**4), 1e-3), mom=(0.8, 0.7))
```

好问题。这里的学习率是不同的值除以 2.6 的4次方。今天最后的部分，我们会讲为什么要用4次方。现在我们看下 2.6 。为什么是 2.6 ? 今天晚些时候，我们会看到更多细节，基本上，slice的第一个参数和第二个参数的差值就是第一层学习的速度和最后一层学习的速度的差值。这被叫作判别学习率 (discriminative learning rate)。这个问题实际上是，当你一层层训练时，学习率要减少多少。我们发现对NLP RNN来说，答案是 2.6 。

怎样找出它是 2.6 的呢? 我用很多不同的各种类型 (dropout、学习率、判别学习率，等等) 的超参数运行了很多不同的模型，然后我创建了一个随机森林，这是一个用来预测用这些参数时我的NLP分类器的准确率的模型。然后我用了随机森林解释函数来找出最优的参数，我找出的答案就是这个 2.6 。这个没有发表过，之前可能也没有提过，这是一个新的技巧。事实上，在我做了这件事几个月后，Stephen Merity和其他的一些人发表了一篇描述类似方法的论文，里面可以看到实现的思路。

其中一些思路来自一个叫Frank Hutter的研究者和他的同事。他们做了一些有意思的工作，展示了怎样使用随机森林来找到最优的超参数。这是一个很好的技巧。很多人对Auto ML很感兴趣，它是用模型来找出训练模型的方法。我们不痴迷于这个。但是我们发现可以构建模型来更好地理解超参数的作用，然后可以找出经验法则，比如 2.6 总是很好用。总之，这是我们之前尝试出的值。

回到表格数据 [36:41]

我们讲下表格数据。表格数据是你可能在电子表格、关系型数据库、财务报告里看到的东西，它可以包含很多不同类型的东西。我试着做了一个表格数据分析应用领域的列表。



我们第一次提出使用神经网络分析表格数据时，人们非常怀疑，他们认为这是一个糟糕的想法，因为每个人都知道应该用逻辑回归、随机森林、gradient boosting（每个方法都有擅长的领域）。但是后来，很明显，这种普遍的观点被证明是错的。神经网络对表格数据没有用的想法是不对的，事实上非常有用。我们在我们的少数课程上演示了这个，但引起真正改变的是一些非常有影响力的组织开始发表论文和文章讲述他们用神经网络处理表格数据时做到出的成果。

一个反复出现的重要情况是，尽管特征工程没有消失，但已经变得简单了。比如，Pinterest取代了gradient boosting，它们是使用神经网络来做首页内容推荐的。他们在一次会议上展示了这个，讲解了这让工程变得多么简单，因为很多人工创建的特征不再有用了。你还是需要用到一些，但是变得简单了。最后他们得到了更准确结果，更重要的是，这需要更少的维护。所以我要说，这不是用来分析表格数据的唯一的工具。但是，之前我在用机器学习处理表格数据时，99%的任务里，我使用随机森林，现在90%的任务里，我使用神经网络。现在这是我每次首选的方法，这很可靠、很高效。

困难的一个地方在于，直到现在，没有一个创建和训练表格神经网络的简单方法。没有人把它做到一个库里。所以我们创建了 `fastai.tabular`，我觉得这是用神经网络处理表格数据第一次变得这么简单。让我们看下，这有多简单。

表格示例 [39:51]

[lesson4-tabular.ipynb](#)

这实际上是直接用的fastai仓库example文件夹里的例子。我没有做一点变动。像往常一样，引用fastai，引用应用，这回是用tabular。

```
1 | from fastai import *
2 | from fastai.tabular import *
```

我们假设你的数据是在一个Pandas DataFrame里。Pandas DataFrame是Python里表格数据的标准格式。有很多方式能取到它，但是最常用的是 `pd.read_csv`。不管你的数据在哪里，你都可以很方便地放进Pandas data frame里。

```
1 path = untar_data(URLs.ADULT_SAMPLE)
2 df = pd.read_csv(path/'adult.csv')
```

提问: 不使用神经网络的10%的情况是怎样的? [\[40:41\]](#)

好问题。我猜我还会尝试下，但是还不确定。基本是这样，你做了一阵，你开始有了一些了解，在一些情况下它们的效果不是很好。我这周会想一想。我没有总结好的经验。但是我要说，你要两种方式都试下。试下随机森林和神经网络。它们都很快，很容易运行。看看结果怎样。如果结果很接近，我会每个都深入一下，看看能不能让结果更好。但是如果随机森林更好，我大概会直接选它。用有效的那个。

我们使用data frame里的数据，我们有一个adult sample数据集，它是一个经典的老数据集。它很小很简单，很适合做实验。它在一个CSV文件里，你可以用 `pd.read_csv` 把它读到data frame。如果你的数据在一个关系数据库里，Pandas也可以从里面读数据。如果在spark或者Hadoop里，Pandas也可以从里面读数据。Pandas基本上可以从所有你丢给它的東西里读数据。所以我们默认使用它。

```
1 dep_var = '>=50k'
2 cat_names = ['workclass', 'education', 'marital-status', 'occupation',
   'relationship', 'race']
3 cont_names = ['age', 'fnlwgt', 'education-num']
4 procs = [FillMissing, Categorify, Normalize]
```

```
1 test = TabularList.from_df(df.iloc[800:1000].copy(), path=path,
   cat_names=cat_names, cont_names=cont_names)
```

```
1 data = (TabularList.from_df(df, path=path, cat_names=cat_names,
   cont_names=cont_names, procs=procs)
   .split_by_idx(List(range(800,1000)))
   .label_from_df(cols=dep_var)
   .add_test(test, Label=0)
   .databunch())
```

像以前一样，我觉得用data block API很好。这个例子里，我们要创建的list是一个tabular list，我们要从data frame创建它。你可以告诉它：

- data frame是什么
- 你要保存模型和做中间步骤的路径是什么
- 然后你要告诉它，你的类别变量和连续变量是什么

连续 vs. 类别 [\[43:07\]](#)

我们下周会讲很多关于这对神经网络来讲意味着什么的内容，现在先做快速介绍一下。自变量是你用来做预测的东西。像教育经历、婚姻状况、年龄等等。一些变量是数字，比如年龄。它们可以是任何数字，你的年龄可以是13.36或者19.4，或者其他值。另外，像婚姻状况这样的东西是一些选项：已婚，单身，离异。有时这些选项可能会很多，比如职业。有很多可能的职业。有时，可能是一个布尔值（真或假）。所有你可以从可能选项中做选择的被叫做**类别变量**。所以我们在神经网络里用不同的方法来对类别变量和连续变量建模。对于类别变量，我们会使用叫 **embeddings** 的方法，今天晚些时候我们

会讲到。对于连续变量，它们可以像像素值那样被直接传入神经网络。神经网络里的像素是数字，这些连续变量也是数字。这很简单。

这就是为什么你需要告诉tabular list哪一列是连续的、哪一列是类别的。在Pandas里有一些其他方法，可以做预处理，把输入变成类别变量，但是用这个fastai里的API更简单，你不用再自己处理太多东西。

处理器 (Processor) [45:04]

然后我们创建一些和计算机视觉的transforms很像的东西。计算机视觉的transforms做按照轴线反转、旋转、提高亮度或者标准化这些事情。对于表格数据，没有这些变形，我们有个叫作processes的东西。它和变形很相似，但是最关键的不同是processor是在前面使用的，这很重要。变形是用来做数据增强 (data augmentation) 的，我们希望随机处理，每次都是不同的。而我们是在前面做processes，只做一次。

```
1 | procs = [FillMissing, Categorify, Normalize]
```

在fastai库里我们有几个processes。这次要用的几个是：

- `FillMissing`: 找出缺失的值，用某些方式处理它们
- `Categorify`: 找出类别变量，把它们转成Pandas类别
- `Normalize`: 做标准化是取到连续变量，减去它们的均值，除以它们的标准差，所以它们是0到1之间的变量

我们下周会讲更多关于我们处理缺失数据的方式。简单讲，我们用中位数替代它，并且加上一列来标记它是不是缺失的。

对于所有这些，你对训练集做了什么，你就需要用同样的方法处理验证集和测试集。你用什么替换了缺失值，你就要用相同的值在验证集做替换。fastai会为你处理所有这些细节。如果你手工去做的话，会像我一样，在最终完全做好之前，出错错很多次。所以库里提供了这些processes。

然后我们要分割训练集和验证集。这个例子里，我们用一个索引的列表来分割，这些索引是从800到1000。这很常见。我不太记得这个数据集的细节，但是在希望验证集是连续的时候，这种方法很常见。如果是地图切片，它们应该是连在一起的一批切片。如果是视频截图，它们应该是连在一起的一段视频的截图。否则，某种意义上讲，就是作弊了。所以如果你的数据有这样的结构的话，通常使用`split_by_idx`来取出连续的数据是一个好方法，也可以使用一些其他的能达到相同效果的方法。

好了，现在有了一个训练集和验证集。我们现在需要添加标签。这个例子里，标签直接来自我们之前取到的data frame，我们可以直接告诉它哪一列是标签。因变量是它们的薪水是否高于\$50,000。这是我们要预测的东西。

我们晚些会讲测试集。这个例子里我们可以添加一个测试集。最后取到data bunch。现在我们得到了这样的东西。

```
1 | data.show_batch(rows=10)
```

workclass	education	marital-status	occupation	relationship	race	education-num_na	age	fnlwgt	education-num	target
Private	Prof-school	Married-civ-spouse	Prof-specialty	Husband	White	False	0.1036	0.9224	1.9245	1
Self-emp-inc	Bachelors	Married-civ-spouse	Farming-fishing	Husband	White	False	1.7161	-1.2654	1.1422	1
Private	HS-grad	Never-married	Adm-clerical	Other-relative	Black	False	-0.7760	1.1905	-0.4224	0
Private	10th	Married-civ-spouse	Sales	Own-child	White	False	-1.5823	-0.0268	-1.5958	0
Private	Some-college	Never-married	Handlers-cleaners	Own-child	White	False	-1.3624	0.0284	-0.0312	0
Private	Some-college	Married-civ-spouse	Prof-specialty	Husband	White	False	0.3968	0.4367	-0.0312	1
?	Some-college	Never-married	?	Own-child	White	False	-1.4357	-0.7295	-0.0312	0
Self-emp-not-inc	5th-6th	Married-civ-spouse	Sales	Husband	White	False	0.6166	-0.6503	-2.7692	1
Private	Some-college	Married-civ-spouse	Sales	Husband	White	False	1.5695	-0.8876	-0.0312	1
Local-gov	Some-college	Never-married	Handlers-cleaners	Own-child	White	False	-0.6294	-1.5422	-0.0312	0

这就是我们的数据。然后使用它。它看起来很熟悉。你得到一个learner，这个例子里它是一个tabular learner，传入数据、架构的信息，和一些度量 (metrics)。然后可以调用fit。

```
1 | learn = tabular_learner(data, layers=[200,100], metrics=accuracy)
```

```
1 | learn.fit(1, 1e-2)
```

```
1 | Total time: 00:03
2 | epoch  train_loss  valid_loss  accuracy
3 | 1      0.362837    0.413169    0.785000  (00:03)
```

提问: 怎样用fastai把(做过分词的)NLP数据和元数据(表格数据)结合起来?比如,对于IMDb分类,怎样使用演员、拍摄时间、题材等等这些信息。[\[49:14\]](#)

我们没讲到这个。我们需要学习更多关于神经网络架构是怎样运作的内容。理论上,这和把连续变量和类别变量结合在一起是一样的。在神经网络里,你可以把两组不同输入合在一起输入到一个层里。它可以进入前面的层,也可以进入后面的层,视情况而定。如果是文本和图片和一些元数据,你可能希望文本进入RNN,图片进入CNN,元数据进入表格模型。然后你需要把它们合并,然后经过全连接层,这样端到端地训练它们。我们可能会在课程第二部分讲这些,有可能会全部放在第二部分讲。理论上,这是接下来三周要学的东西的简单延伸。

提问: 你觉得scikit-learn和xgboost这样的东西会不会过时?是不是未来所有人都会使用深度学习工具?除了比较小的数据集?[\[50:36\]](#)

我不知道。我不善于做预测。我不是一个机器学习模型。我觉得xgboost是一个很好的软件。有不少做gradient boosting的软件。事实上,尤其是随机森林有很好的可解释性。我认为未来神经网络上也会有类似的东西,但现在没有。所以我不知道未来如何。现在,它们都是有用的工具。scikit-learn是一个经常用来做预处理和运行模型的的库。再说一遍,很难预测未来会怎样。某种意义上,它更聚焦于一些老的建模方法。它们也在不停地加入新东西。我一直尝试在fastai里加入更多scikit-learn里的东西,但我总是会发现更好的方法,就放弃了。所以这是为什么在fastai里没有依赖scikit-learn的原因。我在寻找其他的方式。

[52:12]

```
learn = tabular_learner(data, layers=[200,100], metrics=accuracy)
```

这节课的最后或者下节课的开始我们会学习 `layers` 是什么意思。这是我们定义模型架构的地方，就像我们为卷积网络选择ResNet34或者其它架构。晚些我们会学习更多关于metrics的内容，先提醒一下，metrics只是用来打印出来结果的。它完全不会改变模型。这个例子里，我们希望它打印出准确率，看看效果怎样。

这就是怎样处理表格数据。进度刚好，因为我们就快要到一个节点了。我们计划在3个半课时后，要完成对所有应用的初步概览，然后再做深入的学习。我觉得我们刚刚好，就快完成了。下一个应用是协同过滤。

协同过滤 [53:08]

协同过滤是说你有关于谁买了什么东西、或者谁喜欢什么东西的信息，基本上是你有一个用户、或者评论者或其他的人关于它们买了什么、他们写过什么、或者他们评论过什么的信息。在协同过滤最基本的版本里，你只有两列，用户id之类的东西和电影id，这代表用户看了这个电影。例如，Amazon有一个很大的用户id和买过的产品的id的列表。然后你可以添加额外的信息到这个表里，比如，他们做了评价，评价内容是什么？现在这个表大概是这样，包含用户id、电影id、评价中的星星数量。你可以添加时间，这样就知道这个用户在这个时间买了这个产品，给出了这个评价。但是它们的结构是相同的。

有两种构造协同过滤结构的方法。一个是两列的方式，表格里面有用户和电影。你得到用户id、电影id，每对代表用户看了这个电影，或者代表评价的星星数量。另外一种方式是，你可以把所有用户纵向排列，所有电影横向排列。然后你可以找到一个特定的单元格，它可能代表用户给电影的评分、它可能是一个1代表用户看过这个电影，或者其他内容。



这样两种不同的方法，它们都表示相同的信息。理论上，用右边这种方式去理解它会比较简单。但是大多数情况下你不能这样存储它。因为，大多数情况，你得到一个非常稀疏的矩阵，也就是说，大部分用户没有看过大部分电影，或者大部分客户没有买过大部分产品。所以如果你把它们存储在这样一个每个客户商品组合都占有一个单元格的矩阵里，这个矩阵会非常大。你会用左边这种方式存储它。或者你可以用一些特殊的稀疏矩阵格式来把它存储成一个矩阵。如果感兴趣，你可以学习fastai上的[Rachel的计算机线性代数课程](#)，上面有很多很多关于稀疏矩阵存储方式的内容。现在，我们只用左边这种格式。

[56:38]

[lesson4-collab.ipynb](#)

对于协同过滤，有一个很好的数据集，叫做MovieLens，它是GroupLens创建的，你可以下载不同大小的版本（2000万影评，10万影评）。我们已经创建了一个更小的版本来做实验，今天我们会用这个版本。大概下周，我们会用更大的版本。

```
1 | from fastai import *
2 | from fastai.collab import *
3 | from fastai.tabular import *
```

你可以用 `URLS.ML_SAMPLE` 取到这个较小的版本:

```
1 | user,item,title = 'userId','movieId','title'
```

```
1 | path = untar_data(URLS.ML_SAMPLE)
2 | path
```

```
1 | PosixPath('/home/jhoward/.fastai/data/movie_lens_sample')
```

```
1 | ratings = pd.read_csv(path/'ratings.csv')
2 | ratings.head()
```

	userId	movieId	rating	timestamp
0	73	1097	4.0	1255504951
1	561	924	3.5	1172695223
2	157	260	3.5	1291598691
3	358	1210	5.0	957481884
4	130	316	2.0	1138999234

这是一个CSV，你可以用Pandas读取它，这是里面的数据。它里面一个用户id列表，我们不知道这个用户是谁。有一些电影id。有一些电影这是什么电影的信息，下周我们才会使用它。有评分和时间戳。我们这次忽略时间戳。这是我们数据集的一个子集。Pandas里 `head` 只返回前面几行。

现在我们得到了一个data frame，协同过滤的好处是它非常简单。

```
1 | data = CollabDataBunch.from_df(ratings, seed=42)
```

```
1 | y_range = [0,5.5]
```

```
1 | learn = collab_learner(data, n_factors=50, y_range=y_range)
```

这是我们需要的所有数据，所以现在你可以执行 `collab_learner`，你可以传入data bunch。对于架构 (`n_factors`) 你需要告诉它，需要多少个factor，后面我们会学习这是什么意思。然后可能有用的是告诉它分数的范围是什么。我们后面会学习它有什么用。这里，最小分数是0，最大分数是5。

```
1 | learn.fit_one_cycle(3, 5e-3)
```

```
1 Total time: 00:04
2 epoch train_loss valid_loss
3 1      1.600185   0.962681   (00:01)
4 2      0.851333   0.678732   (00:01)
5 3      0.660136   0.666290   (00:01)
```

现在你有了一个learner，你可以继续，调用`fit_one_cycle`训练几次，就是这样。最后，你得到这样的模型，你可以选一个用户id和一个电影id，它可以猜出这个用户会不会喜欢这个电影。

冷启动问题 [58:55]

这显然是一个非常有用的应用，你们大部分人这周都要尝试下。之前的课程里，很多人把协同过滤用在他们的工作里，发现在实践中使用它会更难些。因为在实践中，你会遇到冷启动的问题。冷启动是指你想推荐电影时，遇到新用户、或者遇到新电影的情况。这时，在你的系统过滤系统里，没有任何数据，这就很难。

现在，在fastai里没有内置的处理冷启动问题的功能，这是因为，我所知道的解决这个问题的唯一方式（事实上，我认为是理论上唯一的方式）是加入另外一个模型，它不是协同过滤模型，而是一个新用户和新电影的元数据驱动模型。

我不知道Netflix现在还有没有这样做，在我注册时他们确实是这样做的，他们展示了很多电影，询问“你看过这些吗？”，“你喜欢这些吗？”，这样他们通过用户交互解决了冷启动问题，这样就没有冷启动问题了。他们找了20个常见的电影，问我是否喜欢它们，他们根据我对这20部电影的反馈，显示另外20部我可能看过的电影，在询问了60部电影后，就不再有冷启动问题了。

对新电影，这不是问题，先询问100个人他们是不是喜欢这个电影，然后对后面的10万人、100万人来说，就没有冷启动问题了。

如果你不能通过用户交互询问他们是否喜欢这些东西（比如，如果你是在卖东西，你不会想显示很多产品，询问用户是不是喜欢，因为你只是希望他们买东西），你可以使用一个基于元数据的表格模型，你可能知道他们来自什么地方、他们的年龄和性别，用这些你可以猜测做些初始推荐。

协同过滤在你有些关于用户和电影的信息、或者客户和产品的信息时才是适用的。

[1:01:37]

提问：如果包含不同的语言时（英语单词中包含印地语，或者包含很多emoji的文本）怎样训练语言模型？

包含emoji的文本比较简单。Wikipedia中没有很多emoji，它们基本上都是在关于emoji的维基页面，而不是在普通的语句里。但是你可以（也应该）用语言模型做微调，传入人们日常使用emoji的语料库，这样你可以把Wikitext语言模型微调成reddit或者Twitter或者其他的语言模型。emoji的数量并不多。人们可能使用的单词有几十万个，但可能使用的emoji只有一点点。所以模型很容易学会这些emoji是怎样用的，小菜一碟。

我对印地语不熟悉，但是很熟悉中文普通话，可以用这个做例子。对中文来说，你可以得到一个用中文汉字训练的模型。常用的中文汉字有五六千个，这些汉字都有拼音，拼音是罗马字符。尽管可以把汉字映射成拼音，但不能把拼音映射成汉字，因为同一个拼音可能对应着多个汉字。这需要特别处理。

如果你想在中文上用这个方法，首先你要有一个中文的语言模型。

事实上，fastai里叫做[Language Model Zoo](#)的东西，我们在向里面加入越来越多的不同语言的语言模型，也在加入各种不同领域的语言模型，比如英文医学文本，甚至不是NLP里的语言模型，比如说基因序列、分子数据、MIDI音符等等。所以，显然，你应该使用它来做其他语言的语言模型。

把它（简体或者繁体中文）转换成拼音，你可以直接映射到单词表，或者，因为这些多层模型里，只有第一层把分词转成一组向量，你可以仅仅微调第一层。你将在课程第二部分学到这些，还要再过几周，你才能学到怎样做，如果你对这个有兴趣，我们可以在论坛上讨论，因为这是对理解程度的一个很好的检验。

提问：把时间序列用在表格数据上怎样？ [tabular.models](#) 里有没有涉及到RNN？ [1:05:09]

下周我们会学习时间序列表格数据，简单讲，一般不会在时间序列表格数据上使用RNN。你可以取到一列存储星期几、是否周末、是否节假日、是否商店营业之类的东西。添加这些额外的列，你可以自动获得最优的结果。在时间序列上RNN有一些好的应用，但是不是这种表格的时间序列（比如零售商店物流数据）。

提问：有没有什么资源可以学习更多关于冷启动的问题？ [\[1:06:14\]](#)

我要找找，如果你知道一些好的资源，请在论坛上告诉我们。

课程完成一半了 [1:06:34]

现在既是第四课的课间休息，也是我们课程完成一半的节点。现在我们已经学习了所有重要应用的例子。后面我们会深入学习它们究竟是怎样工作的，会讲更多的理论，介绍源代码是怎样写的，等等。现在可以好好休息下。另外，今天是我的生日，真是一个特别的时刻。

用Microsoft Excel做协同过滤 [1:07:25]

[collab_filter.xlsx](#)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
1						movield	27	49	57	72	79	89	92	99	143	179	180	197	402	417	505	
2						userId	14	3	5	1	3	4	4	5	2	5	5	4	5	5	2	5
3							29	5	5	5	4	5	4	5	4	4	5	5	3	4	5	
4							72	4	5	5	4	5	3	4.5	5	4.5	5	5	5	4.5	5	
5							211	5	4	4	3	5	3	4	4.5	4	3	3	5	3		
6							212	2.5		2	5		4	2.5		5	5	3	3	4	3	
7							293	3		4	4	4	3		3	4	4	4.5	4	4.5	4	
8							310	3	3	5	4.5	5	4.5	2	4.5	4	3	4.5	4.5	4	4	
9							379	5	5	5	4		4	5	4	4	4	3	5	4	4	
10							451	4	5	4	5	4	4	5	5	4	4	4	2	3.5	5	
11							467	3	3.5	3	2.5			3	3.5	3.5	3	3.5	3	3	4	
12							508	5	5	4	3	5	2	4	4	5	5	5	3	4.5	3	
13							546	5	2	3	5			5	5		2.5	2	3.5	3.5	5	
14							563	1	5	3	5	4	5	5		2	5	5	3	3	4	
15							579	4.5	4.5	3.5	3	4	4.5	4	4	4	4	3.5	4.5	4	4.5	
16							623	5	3	3			3	5		5	5	5	2	5	4	
17																						
18																						
19																						
20																						
21																						
22																						
23																						
24																						
25																						
26																						
27																						
28																						

Microsoft Excel是我最喜欢的探索数据和理解模型的方式。我会把这个放到repo里，大概也可以在Google sheets里做这个。但是我觉得这是一个糟糕的产品，所以请使用Microsoft Excel。这两个产品没有什么地方是一样的，我试过所有的功能。电子表格受到了那些不懂得使用它们的人的严厉批评。一些人花了很长时间在Excel上，然后他们开始使用Python，他们可能会把Excel称作愚蠢的东西。要精通电子表格要花几千小时，但是要对它有信心只需要几小时。一旦你对它有了信心，你可以看到所有的东西，全都在那里，它很强大。

Jeremy的电子表格技巧 [1:08:37]

今天讲个电子表格的小技巧。如果你按下 `ctrl` 键或者 `command` 键，然后按箭头键，比如按 `ctrl + →`，它会选到当前表格的最右面。这是移动光标的最好的方式。

现在，我想在这个表格里跳动，我可以按 `ctrl + ↓ →` 来跳到右下角，按 `ctrl + ← ↑` 到左上角。

这里有些数据，我们说过，显示表格数据的一种方式是这样：

F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
userId	movielid	27	49	57	72	79	89	92	99	143	179	180	197	402	417	505
14	3	5	1	3	4	4	5	2	5	5	4	5	5	2	5	5
29	5	5	5	4	5	4	4	5	4	4	4	5	5	3	4	5
72	4	5	5	4	5	3	4.5	5	4.5	5	5	5	5	4.5	5	4
211	5	4	4	3	5	3	4	4.5	4	3	3	3	5	3	5	3
212	2.5		2	5		4	2.5		5	5	3	3	4	3	2	
293	3		4	4	4	3		3	4	4	4.5	4	4.5	4		
310	3	3	5	4.5	5	4.5	2	4.5	4	3	4.5	4.5	4	3	4	
379	5	5	5	4		4	5	4	4	4		3	5	4	4	
451	4	5	4	5	4	4	5	5	4	4	4	4	2	3.5	5	
467	3	3.5	3	2.5			3	3.5	3.5	3	3.5	3	3	4	4	
508	5	5	4	3	5	2	4	4	5	5	5	3	4.5	3	4.5	
546		5	2	3	5		5	5		2.5	2	3.5	3.5	3.5	5	
563	1	5	3	5	4	5	5		2	5	5	3	3	4	5	
579	4.5	4.5	3.5	3	4	4.5	4	4	4	4	3.5	3	4.5	4	4.5	
623	5	3	3			3	5		5	5	5	5	2	5	4	

我们从MovieLens数据里取出看过最多电影的人，和被看过最多次的电影。只选择15条数据。可以看到，这样做的话，它不再是稀疏的了。只有少数的空格。

现在可以用它创建一个模型。怎样建模型？我们想做的是创建可以预测293号用户是不是会喜欢49号电影的东西。我们要写出能做判断的函数。

这里有个简单可行的方法。我们会用矩阵乘法。我在这里创建了一个随机矩阵。这是一个随机数的矩阵（左边）。我在这里创建了另一个随机数的矩阵(顶部)。具体来说，我为每个电影创建了五个随机数，为每个用户创建了五个随机数。

NB: These are initialized to random numbers	-1.69	1.49	-0.14	1.95	-0.09	1.80	1.74	0.68	0.22	1.92					
Then we use Solver to optimize them	1.01	0.12	1.36	1.49	1.17	0.73	-0.20	-0.01	2.06	1.40					
with gradient descent	0.82	1.48	0.02	0.53	1.07	1.24	1.64	0.95	0.43	0.82					
	1.89	0.50	1.74	0.41	1.57	0.49	0.20	1.54	0.43	-0.22					
	2.39	1.13	1.15	-0.74	1.14	-0.63	0.90	1.24	1.11	0.19					
	movield														
	userId														
	27	49	57	72	79	89	92	99	143	179					
0.21	1.61	2.89	-1.26	0.82	14	=IF(H2="",0,MMULT(\$B\$25:\$F\$25,H\$19:H\$23))		4.98	5.45						
1.55	0.75	0.22	1.62	1.26	29	4.40	4.98	5.08	3.99	4.95	3.61	4.37	5.32	4.08	4.10
1.50	1.17	0.22	1.08	1.49	72	4.43	4.94	4.98	4.13	4.86	3.42	4.30	4.74	4.96	4.75
0.47	0.89	1.32	1.13	0.77	211	5.16	4.21	4.01	2.83	5.06	3.19	3.74	4.27	3.84	0.00
0.31	2.10	1.47	-0.29	-0.15	212	1.91	0.00	2.18	4.52	0.00	3.87	2.35	0.00	4.74	4.77
1.00	1.45	0.37	0.83	0.67	293	3.24	0.00	4.05	4.14	4.05	3.28	0.00	3.12	4.46	4.19
1.16	1.16	0.19	2.16	-0.03	310	3.37	3.19	5.14	4.98	4.80	4.23	2.49	4.24	3.60	3.51
0.79	1.07	1.30	1.29	0.70	379	4.92	4.68	4.41	3.84	0.00	4.00	4.19	4.62	4.26	3.93

然后，我们可以说，对14号用户和27号电影，用户喜欢这个电影吗？我们可以把这个向量（红色的）和这个向量（紫色的）相乘，来得到评分。我们做的是点积。然后我们可以为每一个组合做同样的事。我们可以只在一个地方做，然后复制它，它可以填满整个表格。我们为什么要这样做？这是一个神经网络的最开始时的样子，不是吗？神经网络的开始时就是对两个矩阵做乘法，这就是第一层的样子。我们需要找到，可以相乘的两个矩阵是什么。显然，你需要一个用户向量（一个所有用户的矩阵）和一个电影向量（一个所有电影的矩阵），把它们乘在一起，得到一些数字。这还不够。它们是随机的。现在我们可以用梯度下降来让这些数字（顶部）和这些数字（左边）给我们接近我们想要的结果。

我们怎样做到呢？我们假设这是一个线性模型，然后我们需要的是一个损失函数。我们可以这样计算损失函数，当14号用户对27号电影的评分是3时，可以说结果正确。使用这个随机矩阵，得到的评分是0.91，所以我们可以得到差的平方是 $(3 - 0.91)^2$ ，然后我们可以把它们加在一起。Excel里已经有一个求平方的和的方法，会为X、Y的差的平方求和(SUMXY2)，所以我们可以直接用这个函数，传入这两个range，然后除以总数，得到平均值。

这是一个均方差的平方根。你有时会看到人们讨论MSE，它代表Mean Squared Error（均方差），有时你会看到RMSE，这代表Root Mean Squared Error（均方差根）。因为我在前面做了开方，这是均方差的平方根。

Excel 求解器 (Solver) [1:14:30]

我们有了损失函数，现在我们需要做的是使用梯度下降来修改权重矩阵来减小损失度。Excel可以为我们做到。

如果你没有求解器，进入Options → Add-ins, 启用“Solver Add-in”。

Excel里的梯度下降求解器叫做“Solver”，它只做普通的梯度下降。你可以这样打开Data → Solver（你需要确保在设置里启用了Excel自带的solver拓展），你只需要告诉它哪个单元格代表损失度函数。就是这个，V41单元格。哪些单元格保存了变量，是这些，H19到V23，B25到F39。然后你可以说：“通过改变这些单元格来取到损失函数的最小值”，点击Solve：

你可以看到它从2.81开始，可以看到这个数字在下降。它用的就是梯度下降，和我们之前在notebook里手工做的一样。只是它不是在Python里对 $a @ x$ 解均方差，而是解这个损失函数，这个损失函数是这些向量和这些向量的点积的均方差。

我们让它运行一会儿，看看发生了什么。基本上，这是一个在宏里创建一个神经网络的简单方法，这里，神经网络是一个只有一个线性层的、用梯度下降求解协同过滤问题的网络。

[回到 notebook \[1:17:02\]](#)

我们回来看下，在这里怎样做。

```
1 | data = collabDataBunch.from_df(ratings, seed=42)
```

```
1 | y_range = [0, 5.5]
```

```
1 | learn = collab_learner(data, n_factors=50, y_range=y_range)
```

```
1 | learn.fit_one_cycle(3, 5e-3)
```

```
1 | Total time: 00:04
2 | epoch  train_loss  valid_loss
3 | 1      1.600185   0.962681   (00:01)
4 | 2      0.851333   0.678732   (00:01)
5 | 3      0.660136   0.666290   (00:01)
```

这里我们使用 `collab_learner` 来取到一个模型。深入深度学习的一个很好的方法，就是深入 fastai 的代码，看看里面是怎么做的。如果你要这样做，你需要知道怎样熟练使用你的编辑器来查看源代码。基本上要知道这两件主要的事情：

1. 通过名字跳到一个特定的“符号（symbol）”，比如一个特定的类或者函数
2. 当你查看一个特定符号时，能跳到它的实现

比如在这个例子里，我想找到 `def collab_learner`。在大部分编辑器里，包括我用的这个vim，你可以设置它，这样你可以按tab或者其他键，它会显示所有可能的补全，你可以点击enter，它会直接调到定义。这就是 `collab_learner` 的定义。你可以看到，它很小，它做的主要的事情是创建一种 `EmbeddingDotBias` 模型，传入你指定的各种东西。你要在你的编辑器里找出怎样跳到定义，在vim里，你可以用 `Ctrl +]`，这是 `EmbeddingDotBias` 的定义。

```
class EmbeddingDotBias(nn.Module):
    "Base dot model for collaborative filtering."
    def __init__(self, n_factors:int, n_users:int, n_items:int, y_range:Tuple[float, float]=None):
        super().__init__()
        self.y_range = y_range
        (self.u_weight, self.i_weight, self.u_bias, self.i_bias) = [embedding(*o) for o in [
            (n_users, n_factors), (n_items, n_factors), (n_users,1), (n_items,1)
        ]]
    def forward(self, users:LongTensor, items:LongTensor) -> Tensor:
        dot = self.u_weight(users)* self.i_weight(items)
        res = dot.sum(1) + self.u_bias(users).squeeze() + self.i_bias(items).squeeze()
        if self.y_range is None: return res
        return torch.sigmoid(res) * (self.y_range[1]-self.y_range[0]) + self.y_range[0]
class CollabDataBunch(DataBunch):...
class CollabLearner(Learner):...
def collab_learner(data, n_factors:int=None, use_nn:bool=False, metrics=None,
                   emb_szs:Dict[str,int]=None, wd:float=0.01, **kwargs)->Learner:
    "Create a Learner for collaborative filtering on `data`."
    emb_szs = data.get_emb_szs(ifnone(emb_szs, {}))
    u,m = data.classes.values()
    if use_nn: model = EmbeddingNN(emb_szs=emb_szs, **kwargs)
    else:      model = EmbeddingDotBias(n_factors, len(u), len(m), **kwargs)
    return CollabLearner(data, model, metrics=metrics, wd=wd)
```

现在我们马上可以看到所有内容，你可以看到，里面内容不多。fastai要为你创建的模型实际上是PyTorch模型。PyTorch模型被叫做 `nn.Module`，这是PyTorch里模型的名字。它们有些细微的差别，但现在来说，开始时可以这样理解。当一个PyTorch `nn.Module` 运行时（当你计算这一层或者神经网络的结果时），它总是在为你调用一个叫 `forward` 的方法。所以在这里，你可以找出这些是怎样计算出来的。

开始，当模型被创建时，它调用这个叫 `__init__` 的东西，我们之前简单提过，在Python里，人们叫它 "dunder init"。dunder init就是我们创建模型的地方，`forward`是我们运行模型的地方。

如果你看得仔细，一个你可能会注意的东西是，没有地方说怎样计算模型梯度，这是因为PyTorch为我们做了这个。所以你只需要告诉它怎样计算你模型的输出，PyTorch会继续为你计算出梯度。

这个例子里，模型包括：

- 对一个用户 (user) 的一组权重 (weights)
- 对一个条目 (item) 的一组权重
- 对一个用户的一组偏移 (biases)
- 对一个条目的一组偏移

这里的每一个都来自这个叫 `embedding` 的东西。这是 `embedding` 的定义：

```
def embedding(ni:int,nf:int) -> nn.Module:
    "Create an embedding layer."
    emb = nn.Embedding(ni, nf)
    # See https://arxiv.org/abs/1711.09160
    with torch.no_grad(): trunc_normal_(emb.weight, std=0.01)
    return emb
```

它做的所有事情就是调用PyTorch里叫做 `nn.Embedding` 的东西。PyTorch里，有很多设置好的标准神经网络。它创建一个embedding，然后这个东西（`trunc_normal_`）只是随机初始化它。它为 embedding 创建随机数。

Embedding [1:21:41]

什么是embedding? 一个embedding是一个权重矩阵。具体来讲，一个embedding是一个这样的权重矩阵：

NB: These are initialized to random numbers	0.71	0.92	0.68	0.83	0.60	0.18	0.26	0.91	0.99
Then we use Solver to optimize them	0.81	0.55	0.28	0.88	0.50	0.31	0.08	0.47	0.94
with gradient descent	0.74	0.86	0.53	0.33	0.81	0.68	0.92	0.61	0.46
	0.04	0.44	0.16	0.41	0.73	0.39	0.29	0.94	0.12
	movield	0.04	0.80	0.94	0.24	0.53	0.09	0.74	0.13
	userId	27	49	57	72	79	89	92	99
0.19	0.63	0.31	0.44	0.51	14	0.91	1.40	1.02	1.12
0.25	0.83	0.71	0.96	0.59	29	1.44	2.20	1.49	1.71
0.30	0.44	0.19	0.00	0.72	72	0.73	1.26	1.10	0.87
0.02	0.72	0.69	0.35	0.25	211	1.12	1.36	0.86	1.08
0.60	0.87	0.76	0.30	0.04	212	1.71	0.00	1.14	1.65
0.73	0.70	0.44	0.47	0.29	293	1.44	0.00	1.27	1.63
0.23	0.81	0.36	0.47	0.12	310	1.10	1.27	0.76	1.24
0.68	0.90	0.20	0.92	0.74	379	1.43	2.30	1.67	1.98
0.81	0.41	0.81	0.15	0.17	451	1.52	1.88	1.28	1.41
0.70	0.61	0.90	0.89	0.24	467	1.70	2.35	1.49	1.84
0.50	0.27	0.73	0.44	0.83	483	1.16	2.10	1.65	1.28

你可以查看它内部，取出它的一个item。基本上讲，一个embedding矩阵就是一个权重矩阵，它被设计成，你可以用索引取到一个数组，取出一个向量。这就是embedding矩阵。在这个例子里，我们有一个用户的embedding矩阵，和一个电影的embedding矩阵。这里我们算它们的点积。

NB: These are initialized to random numbers					0.71	0.92	0.68	0.83	0.60	0.18	0.26	0.91	0.99	0.52	
Then we use Solver to optimize them with gradient descent					0.81	0.55	0.28	0.88	0.50	0.31	0.08	0.47	0.94	0.70	
					0.74	0.86	0.53	0.33	0.81	0.68	0.92	0.61	0.46	0.64	
					0.04	0.44	0.16	0.41	0.73	0.39	0.29	0.94	0.12	0.67	
					0.04	0.80	0.94	0.24	0.53	0.09	0.74	0.13	0.39	0.44	
					27	49	57	72	79	89	92	99	143	179	
0.19	0.63	0.31	0.44	0.51	14	0.91	1.40	1.02	1.12	1.27	0.66	0.89	1.13	1.18	1.27
0.25	0.83	0.71	0.96	0.59	29	1.44	2.20	1.49	1.71	2.16	1.22	1.49	2.04	1.71	2.08
0.30	0.44	0.19	0.00	0.72	72	0.73	1.26	1.10	0.87	0.93	0.38	0.82	0.68	1.07	0.90
0.02	0.72	0.69	0.35	0.25	211	1.12	1.36	0.86	1.08	1.31	0.85	0.97	1.14	1.15	0.00
0.60	0.87	0.76	0.30	0.04	212	1.71	0.00	1.14	1.65	0.00	1.02	1.03	0.00	1.82	1.64
0.73	0.70	0.44	0.47	0.29	293	1.44	0.00	1.27	$=IF(K7="",0,MMULT($B$0:$F$0,K$19:K$23))$						
0.23	0.81	0.36	0.47	0.12	310	1.10	1.27	0.76	1.24	1.24	0.73	0.67	1.26	1.26	1.29
0.68	0.90	0.20	0.92	0.74	379	1.43	2.30	1.67	1.98	0.00	0.96	1.24	2.13	2.02	2.07

如果你仔细想想，会发现这还不够。因为我们忽略了这个：可能有些电影每个人都喜欢，可能有些人很喜欢电影。所以，我不能仅仅把这两个向量乘起来，而是要加上一个数，表示这个电影多么受欢迎，添加一个数表示这个用户多么喜欢看电影。这叫做偏移（bias）。记住我讲的，有个偏移概念，我们在梯度下降notebook里的处理方式是添加一列。但实际实现里，我们实际上是说我要加一项偏移。我们不是仅仅想让预测值等于这两个矩阵的点积，我们想让它等于点积加上电影偏移和用户偏移。

回到代码 [[1:23:55](#)]

就是这样。我们设置模型，设置用户embedding矩阵和电影embedding矩阵。然后我们设置用户偏移向量和电影偏移向量。

```

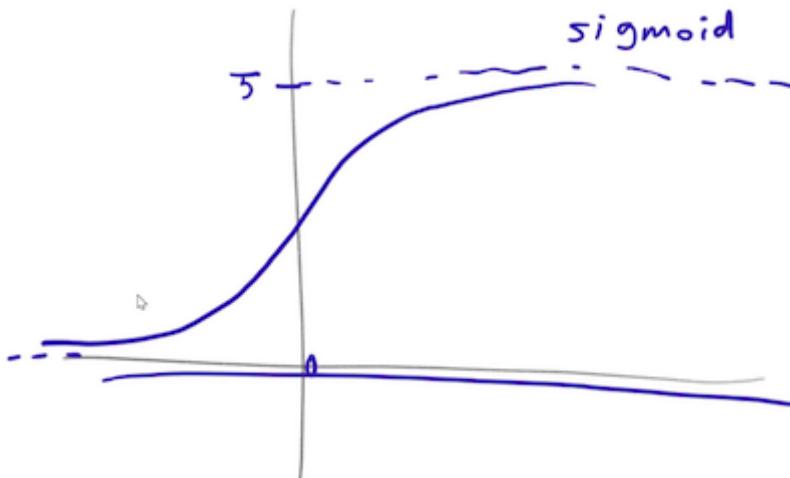
class EmbeddingDotBias(nn.Module):
    "Base dot model for collaborative filtering."
    def __init__(self, n_factors:int, n_users:int, n_items:int, y_range:Tuple[float,float]=None):
        super().__init__()
        self.y_range = y_range
        (self.u_weight, self.i_weight, self.u_bias, self.i_bias) = [embedding(*o) for o in [
            (n_users, n_factors), (n_items, n_factors), (n_users,1), (n_items,1)
        ]]

    def forward(self, users:LongTensor, items:LongTensor) -> Tensor:
        dot = self.u_weight(users)* self.i_weight(items)
        res = dot.sum(1) + self.u_bias(users).squeeze() + self.i_bias(items).squeeze()
        if self.y_range is None: return res
        return torch.sigmoid(res) * (self.y_range[1]-self.y_range[0]) + self.y_range[0]

```

然后，我们计算模型时，我们仅仅是把这两个乘起来。就像我们做的，我们就是做乘积，我们把它叫做 `dot`。然后我们添加偏移（等会再看 `y_range`），这就是我们返回的值。你可以看到这个模型做的事情和我们在电子表格里做的事情一样，只是我们添加了偏移。这是一个非常简单的线性模型。对这种协同过滤问题，这种简单线性模型可以运行得很好。

在最后这里，有一个处理。这个例子，我们设定这里有一个y的范围是0到5.5。所以有些东西要指出来。你做了点积，然后加上两个偏移，这会输出数轴上的任意数字，从无穷小到无穷大。但我们知道，我们只想要0到5之间的数。我们可以用这个函数转换，这个函数的形状是S形的。这里接近5，这里接近0。



这样，无论我们点积加上偏移的结果是什么数字，如果我们把它输入这个函数，它都不会大于5，不会小于0。严格讲，这不是必须的。因为我们的参数可以学会找出能得到正确结果的参数。如果不是必须的，我们为什么要做这样额外的事情呢？原因是，我们想让模型处理的事情尽量简单。如果我们这样设置了，它就不会预测出很大或者很小的值，然后它可以把时间花在参数预测上，也就是预测谁会喜欢哪个电影，这是我们更关心的事情。这个方法，我们还会再讲到，它会让神经网络运行得更好。我们用的这些小技巧，让网络能更容易学到正确的东西。这是最后一个技巧：

```

1 |     return torch.sigmoid(res) * (self.y_range[1]-self.y_range[0]) +
|         self.y_range[0]

```

我们取到点积加上偏移，把它输入到一个sigmoid函数。sigmoid函数（S形函数）是 $\frac{1}{1+e^{-x}}$ ，但这个定义不重要。重要的是，它的这个形状，它介于0和1之间。如果你把它乘以 `y_range[1]` 减 `y_range[0]` 的差，把乘积加上 `y_range[0]`，它就会输出介于 `y_range[0]` 和 `y_range[1]` 之间的数。

这是个小小的神经网络，说它是神经网络有些勉强，它是一个只有一个权重向量的、没有非线性函数的神经网络，它是最无聊的、末尾有一个S形函数的神经网络。我觉得它有一个非线性部分。最后的S形函数是非线性的，它只有一层权重。它实际上能得出接近最佳成绩（state-of-the-art）的结果。我在网上找过人们在这个MovieLens 100k数据集的最好成绩，我用这个小程序取到的结果好于我能找到的标准商业产品的结果，你可以下载它们，它们是专门用于这个的。窍门可能是添加了这个小小的S形函数，它产生了很大影响。

[1:29:09]

提问：怎样设置vim？我已经链接到了你的 [.vimrc](#)，但是我想知道你有没有其他要讲的。现在的设置和你的一样

你的设置和我们的一样？我的设置里没什么东西，这是真的。不论你在用编辑器做什么，你可能想要它能这样：比如你有一个类，当你不处理它时，可以这样折叠起来，它折叠后，你看不到它了。你需要这样可以折叠、展开的东西。vim已经为你做了这些了。另外，像我讲过的，你还需要可以跳转到对象定义的东西，在vim里，叫using tags（比如，要跳到 Learner 的定义，光标放到 Learner 上，然后按 `Ctrl +]`）。vim已经为你做好了这些。你只需要阅读介绍文档。我的.vimrc是最简单的，我几乎不用什么拓展插件（extensions）或其他东西。另外一个很棒的编辑器是 [Visual Studio Code](#)。它是免费的，它很酷，有和vim一样的功能。VS code也为我们做好了所有事情。我更喜欢用vim是因为我可以在远程机器使用它，但是你当然可以clone下git repo到你本地电脑，使用VS Code来运行。只在github上看代码而不试一试会让你疯掉的。你需要展开它，折叠它，跳到定义，再跳回来。可能有些人在论坛上发表关于vim、VS Code、Sublime的使用方法的帖子。如果你要选择一个编辑器，如果想用在本地，我会使用VS Code，我认为这是最好的。如果你想用在远程终端，我会用VIM或者Emacs，对我来说它们明显更好。

重要术语概览 [1:31:24]

最后，要掌握这个协同过滤的例子，要在后面三节课里描述怎样创建更复杂的神经网络，我们要学习这些概念：

- 输入
- 权重/参数 (Weights/parameters)
 - 随机
- 激活
- 激活函数/ 非线性项 (Activation functions / nonlinearities)
- 输出
- 损失度
- 度量 (Metric)
- 交叉熵 (Cross-entropy)
- Softmax
- 微调 (Fine tuning)
 - 层删除和随机权重 (Layer deletion and random weights)
 - 冻结和解冻 (Freezing & unfreezing)

我们来想一想当我们用神经网络做图像识别时发生了什么。我们取一个像素，你会有很多像素，但我们只取一个。这样你得到一个红绿蓝像素。每一个都是0到255的数，或者我们做标准化（normalize）让它们服从0, 1分布。这里我们用0到255的数字。所以红通道的值是10，绿通道的值是20，蓝通道的值是30。我们用这个做什么呢？我们要把它们当做一个向量。我们用一个矩阵乘以它。让这个矩阵有3行，应该有多少列呢？这需要你来选择，就像在协同过滤程序里的一样，我选择5作为每个embedding向量的尺寸。我们用尺寸是5的embedding。你需要选择你的权重矩阵的大小。这里我们选择尺寸是5。它是 3×5 的矩阵。

开始时，权重矩阵包含的是随机数。记得我们看过 `embedding` 权重矩阵吗？

```
def embedding(ni:int,nf:int) -> nn.Module:  
    "Create an embedding layer."  
    emb = nn.Embedding(ni, nf)  
    # See https://arxiv.org/abs/1711.09160  
    with torch.no_grad(): trunc_normal_(emb.weight, std=0.01)  
    return emb
```

这里有两行，第一行是创建矩阵，第二行是用随机数填充它。这就是所有的东西。这是fastai和PyTorch里做的所有的事。所以当你初始它时，它创建了一个随机矩阵。行的数量是3，来匹配输入的矩阵的长度。列的数量可以设成你想要的。当你把输入向量和权重矩阵相乘后，你会得到一个长度是5的向量。

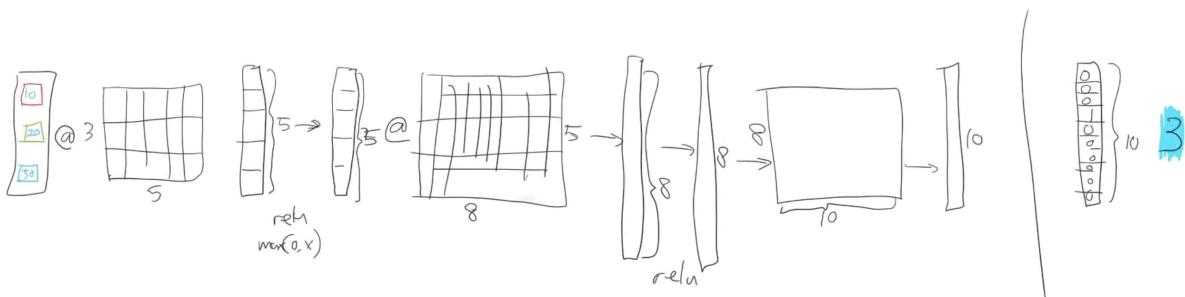
人们总是问，要做深度学习需要多少线性代数知识。这就是所有你需要的。如果你对这个不熟悉，没关系。你只需要知道矩阵乘积。你不需要知道太多关于这个的东西，你只需要知道在计算时它们是什么、它们做了什么的。你要对一个某种大小的矩阵乘以某种大小的矩阵，得到某种大小的矩阵比较熟悉（维度是怎样对应的）。如果你输入的向量长度是3，在numpy和PyTorch里，我们用@符号乘以 3×5 的矩阵，得到一个长度是5的向量。

然后会发生什么？它经过一个激活函数，例如ReLU，就是 `max(0,x)`，得到一个新的向量，尺寸当然是相同的。因为 **激活函数不会改变尺寸，它只会改变内容**。所以输出的尺寸还是5。

然后会发生什么？我们乘以另外一个矩阵。还是一样，列的数量可以是任意的。但是行的数量需要和前面的对应，它只能是5。原来列的数量是5，这次让我们设成10，这会产生相同尺寸的输出，输出的尺寸会是10，我们再让它经过ReLU一次。这还是会生成相同尺寸的输出。

然后我们可以让它经过另一矩阵。为了能看得更清楚，我这次用8，不再用10。

假设我们在做数字识别。有十个可能的数字，所以我的最后一个权重矩阵的尺寸应该是10。这意味着我的最终输出是一个尺寸是10的向量。记住，如果你在做数字识别，我们实际的尺寸是10，如果我们要预测的数字是3，这意味着在第三个位置的值是1([0,0,0,1,0,...])。

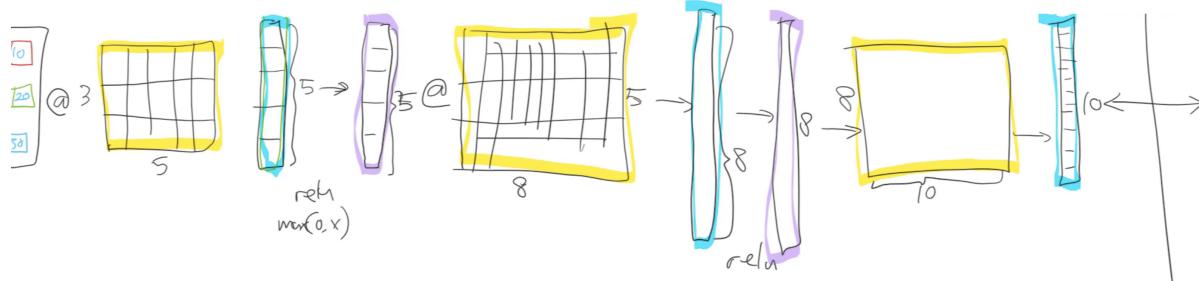


我们的神经网络从输入开始，然后是权重矩阵→ReLU→权重矩阵→ReLU→权重矩阵→最终输出。然后我们比较这两个，使用一些损失函数看看它们有多接近，下周我们会学习所有我们会使用的损失函数，现在我们只学过均方差。我们比较输出（你可以把它们看成十个结果的概率）和实际值，来得到损失度，然后我们找到每个矩阵的梯度，然后更新权重矩阵。

现在我想讲的是我们用到的术语，这很重要。

这些东西（黄色的）包含着数字。确切讲，它们初始时是包含随机数的矩阵。在PyTorch里，它们叫参数（parameter），我们可以这样称呼它。有时我们把它们称作权重（weight），尽管叫权重不太精确，因为它们也包含偏移（bias）。有时我们使用时把这两个术语当成是相同的、可以相互替换的东西。严格来说，我们应该叫它们参数。

在和矩阵做过乘积后，得到了一个数字向量。这些数字（蓝色的）是经过权重矩阵乘法计算出结果。然后，这里有一组数字（紫色），它们是ReLU作为激活函数计算出的结果。每个值被叫做激活值。

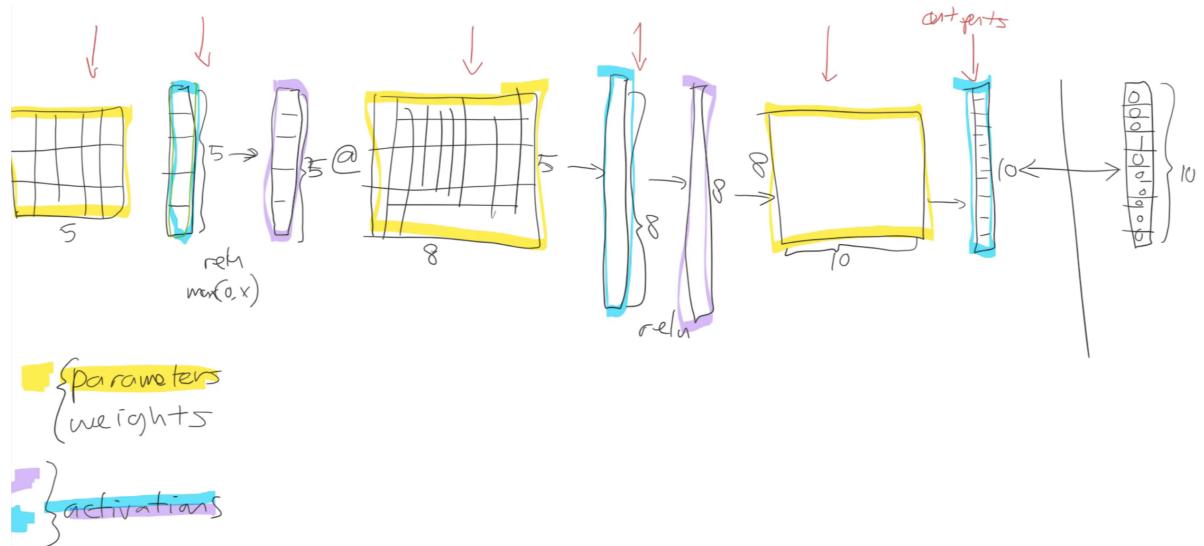


激活值和参数，指的都是数字。它们都是数字。但是**参数**是要被存起来的数字，它们被用来做计算。**激活值**是计算的结果，最后计算出的数字。这是你们要记住的两个东西。

要使用这些术语，并且正确地、精确地使用它们。如果你读到这些术语，它们指的是很特定的东西。不要把它们搞混。记住，这不是什么古怪的神奇的东西，它们很简单。

- 激活值是矩阵乘法或者激活函数的结果
- 参数是用来做乘法的矩阵里的数字

就是这样。然后有一些特别的层。每个这样的东西都做一个计算，所有这些做计算的东西（红色箭头）被叫做一个层。它们就是神经网络里的层。每个层输出一组激活值，因为有一个输出结果的计算。



最开始有一个特殊的层，叫做输入层。最后，你有一组激活值，我们把这些特殊的数字叫做输出（不是在数学上特殊的，而是在语义上）。要知道，这里重要的点是神经网络的输出在数学上没什么特殊的，它们只是一层的激活值。

我们在协同过滤的例子里，做了一些有趣的事情。我们实际上在最后添加了一个额外的激活函数。这个激活函数是一个S形函数，具体讲它是一个放大的S型函数，它的范围是0到5。使用一个激活函数作为最后一层很常见，我们几乎不会使用ReLU，因为它在0的位置被截断了，这不是我们想要的。经常会使用S形函数或类似的东西，因为你需要输出值介于两个数之间，你可以这样缩放它。

基本上就是这样。输入、权重、激活值、激活函数（有时叫非线性项nonlinearities）、输出，然后比较这两个值的东西叫损失函数，目前我们用的是MSE。

今天的内容就是这样。下周我们要做的是添加一些其他的东西，我们会用叫做**交叉熵cross-entropy**的损失度函数做分类问题、我们会用叫做**softmax**的激活函数做单标签分类、我们会学习在做微调时会发生什么、里面的层是怎样的、在做解冻时会发生什么、在做迁移学习时会发生什么。谢谢大家，下周再见。

