

Lesson 6

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

欢迎来到第六课，我们将深入学习计算机视觉、卷积神经网络、什么是卷积，我们还会学习最后的正则化技巧，上周已经学了权重衰减和L2正则化。

Platform.ai

首先，我想给你们看一个让人振奋的东西，我参与创建了它。如果你们看过我在ted.com的演讲，你们可能注意到了这个有意思的demo，这是我们4年前做的，它展示了一种用没有标签的数据快速构建模型的方法。现在，我们准备好了把这个发布出来，让大家使用它。你们是第一批使用它的人。

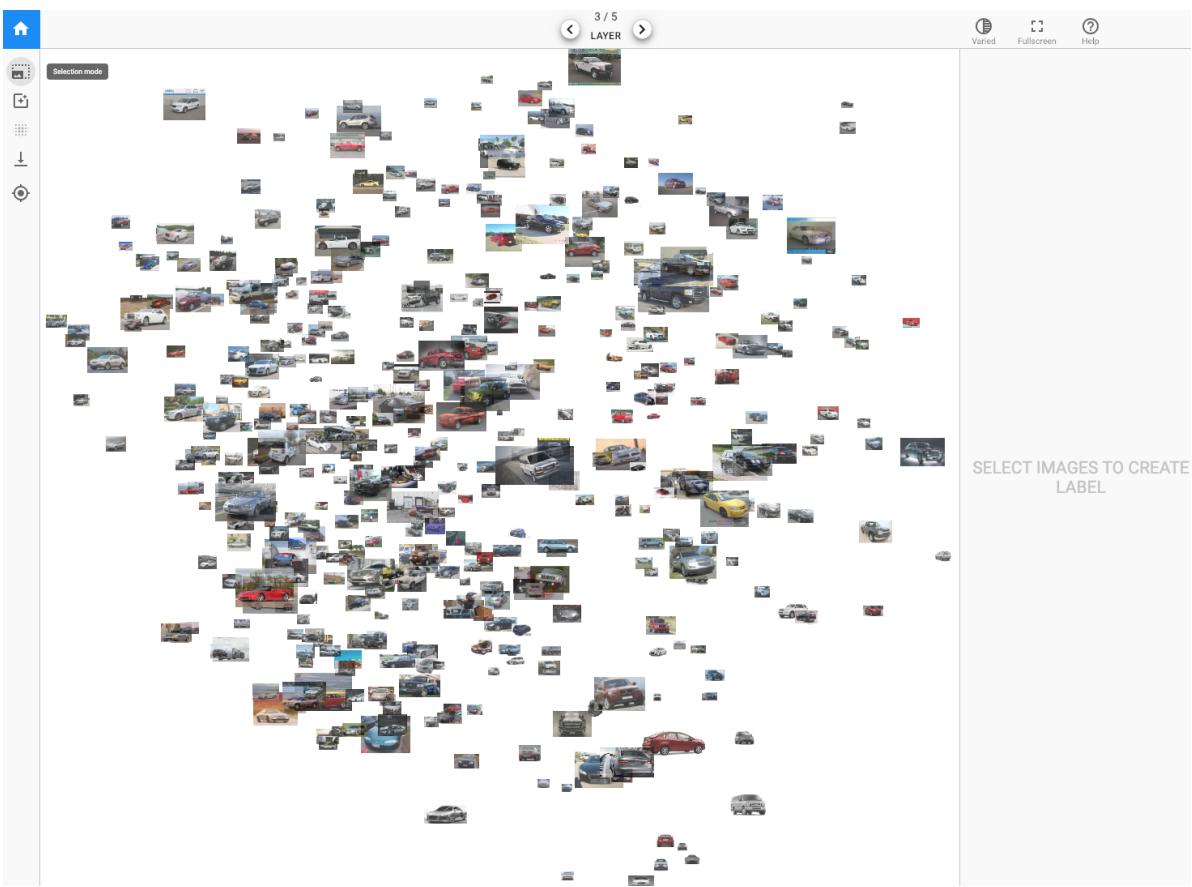
这个公司叫[platform.ai](#)，我介绍它的原因是，它可以在另外一种数据集上创建模型，就是没有标签的数据集。我们会帮助你打标签。这是我们第一次展示它，所以我很兴奋。来快速看一个demo。

如果你访问platform.ai，选择“get started”，你可以创建一个新项目。创建一个新项目后，你可以上传你自己的图片。上传500个左右，它就可以运行地很好。你可以上传几千个，但是开始时，上传500个左右来试下。它们需要在同一个文件夹里。我们假设你有一堆图片，没有任何标签，或者如果你想尝试下，你可以选择一个已有的数据。我用的是汽车数据，和四年前用的一样。

当你第一次进入platform.ai，查看你上传的图片时，随机的样本会显示在屏幕上。可以看出，它们被一个预训练模型从深度学习空间投影到2D空间。这个初始的版本，我们用的是ImageNet模型。我们会添加越来越多的预训练模型。我想做的是为数据集添加标签，区分这些汽车照片是在什么角度拍摄的，这是ImageNet不擅长的，因为ImageNet只学习了区分汽车和自行车之间的不同，这和知道拍摄照片的角度没什么关系。我们想创建的标签是ImageNet特意忽略的东西。

可以看到，我们可以点击顶部的layer按钮来切换，用神经网络里不同的层做投影。这是最后一层，对我们来说，它没什么用，因为这个投影是根据物品的种类来做的。第一层大概也没什么意义，因为它没什么有用的语义信息。但如果我看中间的，在第三层，我们大概可以看到一些不同。

然后，我们可以点击projection按钮（不仅可以点击顶部的按钮，还可以按键盘上下键）来切换投影，也可以按左右键来切换层。你可以四处看下，直到发现按你想要的区分图片的方式的投影。用这个投影，我发现它让一组右前方拍摄的汽车显示在这。如果放大些，我们可以再检查下。“这看起来很好，它们全部是右前方的”。我们可以点这里，进入选择模式，我们可以选择一些，然后检查下：



我们做的是尝试同时利用人和机器。机器很擅长快速计算，人很擅长一次性看一大堆东西，挑出其中那个特别的。这里，我要寻找不是右前方拍摄的汽车，把它们摆在我面前，我可以很快找出来。我会说“好，就是那个”，然后只要点击你不想要的那个。好了，这就完成了。

然后，你可以回去。你可以在“create a new label”里输入名称，把它们放到一个新的分类里，或者在已有的分类里选择一个。之前，我已经创建了一些。这个是右前方，所以我只要点击它就好。

基本上就是你不断浏览不同的层或者投影，尝试找到你感兴趣的一组东西。过段时间，你会发现有些东西有点难。比如，我很难找到侧边，我能做的是，在这里看下，有几个侧边的图片，我可以放大这里，选择它们中的几个。然后，我说“find similar (查找相似)”，这会在投影空间里查找，不仅是在当前显示的图片里查找，而是在所有你上传的图片里找，希望我可以找到多一点侧面照片。它遍历检查所有你上传的照片，看看哪个有和选中照片有相似的投影。但愿，我能找到更多我感兴趣的图片。

现在如果我想找出可以区分侧面和右前方的投影，我可以点击这两个，然后这个按钮现在叫“切换到能最大化这两个标签距离的投影”。它会尝试找出区分这两个类别的最好的投影。它的目的是帮助我用眼睛检查，快速找出可以用来标记的一批东西。

这是关键的功能，它很有效。你可以看下面这里，我们现在得到了一堆侧面照片，之前，我很难找出它们。通常，需要再次检查。观察神经网络的行为很有意思，和一般情况相比，这里看起来有更多的跑车。它大概找到了跑车的侧面这个角度，这很有意思。我找到了这些，现在我点击“side”分类，把它们放进去。

这样做几次之后，如果你得到了一百多个标签，你可以点击训练模型按钮，这会花几分钟，然后你的模型会显示出来。在完成用一个少量标签做的训练后，你可以切换这个改变透明度按钮，它会渐渐隐藏掉那些预测得很好的图片。它还会给出一个这个模型估计的准确度。我提这个的原因是，这样你们可以点击下载按钮，下载这些预测值，我们希望这个对很多人有用。我认为对你们这些深度学习学生来说，能下载标签是有用的。这样你们可以用这个已经有标签的数据子集和没有标签的子集，看看能不能构建出一个比platform.ai更好的模型。看看你们能不能用这个之前没有标注初始集创建模型。

显然，这个系统有些它不擅长的东西。对于需要放大仔细检查的东西，它的效果不会很好。它是被设计用来处理人类眼睛可以快速准确识别出来的东西。我们很乐意得到反馈，你们可以点击帮助按钮，给我们反馈。论坛上也有一个[platform.ai 讨论主题](#)。Arshak你可以站起来吗？Arshak是这个公司的CEO。他会在这里帮助解答问题。我希望这个对大家有用。到现在花了很多年，很高兴我们最后能把它做成这样。

完成表格Learner的正则化[9:48]

今天讲这个的一个原因是，这节课的后面一点，我们会深入学习卷积。我会回过头来，试着解释更多关于它内部是怎么工作的知识。先对这个有些了解。在这之前，我们要先完成上周的正则化，我们讲了tabular learner上下文里的正则化，这是tabular learner的init方法：

```
ps = ifnone(ps, [0]*len(layers))
ps = listify(ps, layers)
self.embeds = nn.ModuleList([embedding(ni, nf) for ni,nf in emb_szs])
self.emb_drop = nn.Dropout(emb_drop)
self.bn_cont = nn.BatchNorm1d(n_cont)
n_emb = sum(e.embedding_dim for e in self.embeds)
self.n_emb, self.n_cont, self.y_range = n_emb, n_cont, y_range
sizes = self.get_sizes(layers, out_sz)
actns = [nn.ReLU(inplace=True)] * (len(sizes)-2) + [None]
layers = []
for i,(n_in,n_out,dp,act) in enumerate(zip(sizes[:-1],sizes[1:],[0.]+ps,actns)):
    layers += bn_drop_lin(n_in, n_out, bn=use_bn and i!=0, p=dp, actn=act)
if bn_final: layers.append(nn.BatchNorm1d(sizes[-1]))
self.layers = nn.Sequential(*layers)
```

我们的目标是理解这里所有的东西，现在还没有做到。上周我们学了adult数据集，这是一个很简单的（有点太简单了）数据集，只能被当作一个玩具。这周，我们要用一个更有意义的数据集，一个Kaggle竞赛数据集，这样我们可以知道世界上最好的成绩，打破Kaggle竞赛的最佳成绩记录要比学术最佳成绩记录难得多，因为很多人在做Kaggle竞赛，参加比赛的人比大多数学术数据集更多。使用一个Kaggle竞赛数据集，并且把它做好，这是一个很好的挑战。

Rossmann数据集是他们在欧洲有3000家药店，你要尝试预测在接下来的几周里，它们会卖出多少商品。这个数据集里，要注意的一点是，测试集所在的时间段比训练集更晚些。这很常见。如果你要预测一些东西，要预测的内容不会在你的训练集里。你要预测未来的东西。

另外一个值得注意的是，他们给的度量方法是根均方百分比误差（root mean squared percent error）。

$$\text{RMSPE} = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{y_i} \right)^2}$$

这是一个普通的根均方误差（root mean squared error），只是我们要用实际值减去预测值再除以实际值外。也就是说，我们做均方根的是一个百分比误差。有些值得注意的特性。

看下排行榜。冠军的成绩是0.1。我们复现的论文的成绩大概是0.105 ~ 0.106，第10名比0.11少一些（0.10839）。

我们要跳过一点东西。他们提供的数据是一些文件，但他们还要求竞赛者在和其他人分享时提供额外的数据。所以我们要处理的数据集实际上有六七个表。关联数据表这些东西的方法不是深度学习课程的内容。我会跳过它，你们可以参考[给程序员的机器学习介绍](#)，里面会带你一步步做数据预处理。我们在[rossman_data_clean.ipynb](#)里提供了这个，你可以在里面看到整个的过程。你需要运行这个notebook来创建我们在这里（[lesson6-rossmann.ipynb](#)）用到的pickle文件：

```
1 | %reload_ext autoreload
2 | %autoreload 2
```

```
1 | from fastai.tabular import *
2 |
3 | path = Path('data/rossmann/')
4 | train_df = pd.read_pickle(path/'train_clean')
```

时间序列和add_datepart [13:21]

Rossmann data clean notebook里有一个地方要讲一下，你可以看到里面有个叫 add_datepart 的方法，我想解释一下这个是做什么的。

```
1 | add_datepart(train, "Date", drop=False)
2 | add_datepart(test, "Date", drop=False)
```

我一直说我们会学习时间序列，很多听我讲过这个的人以为我会做一些递归神经网络 (recurrent neural network)。有意思的是，研究时间序列的主要学术团体是计量经济学家，它们主要研究一种特别的时间序列，就是你拥有的所有数据只有某种东西的时间点序列。现实生活中，基本没有这样的情况。一般我们会有一些关于它代表的商店的信息，或者关于它代表的人的信息。我们有元数据，我们有相近时间段或者不同时间段的其它东西的序列。所以大多数时候，我发现在实际中，使用来自现实世界的数据集的竞赛里，取得最好成绩的结果都没有用递归神经网络。它们使用了time piece，它是一个添加了数据的日期，添加了一整组元数据。在这个例子里，比如说，我们添加了星期。我们有一个日期，我们添加了星期、年、月、第几周、这个月里的第几天、这一年里的第几天、是否是月初/月末的布尔值、是否是季度初/季度末的布尔值、从1970年计算经历的时间，等等。

如果你运行这个函数 add_datepart，传入一个日期，它会添加所有这些列到数据集里。我们看个合理的例子，购买行为可能会受发薪日影响。发薪日可能是每月15号。如果你有“这个月里第几天”这样一个数据，就能识别出时间是不是15号，或和它相关的、更晚的时间。我们不能期望神经网络做所有的特征工程。我们可以期待它能找出非线性关系和相互影响和类似的东西。对于这样格式的时间 (2015-07-31 00:00:00)，要判断出它是不是第15号，是不太容易的。如果我们把这个信息显式提供出来，就会好很多。

这是一个很有用的函数。你做了这个之后，可以像普通的表格问题一样处理很多种时间序列问题。我说的是“很多”，不是“所有”。如果有时间序列里很复杂的和状态相关的东西，比如股票交易或类似的东西，这可能不适用，或者这不是你需要的全部东西。但在这个例子里，它可以给出很好的结果，在实践中，大多数时候，我发现它的效果很好。

表格数据一般在Pandas处理，我们把它们存储成标准的Python pickle文件。我们可以读取它们。我们可以看看前五行记录。

```
1 | train_df.head().T
```

	0	1	2	3	4
index	0	1	2	3	4
Store	1	2	3	4	5
DayOfWeek	5	5	5	5	5
Date	2015-07-31 00:00:00	2015-07-31 00:00:00	2015-07-31 00:00:00	2015-07-31 00:00:00	2015-07-31 00:00:00
Sales	5263	6064	8314	13995	4822
Customers	555	625	821	1498	559
Open	1	1	1	1	1
Promo	1	1	1	1	1
StateHoliday	False	False	False	False	False
SchoolHoliday	1	1	1	1	1
Year	2015	2015	2015	2015	2015
Month	7	7	7	7	7
Week	31	31	31	31	31
Day	31	31	31	31	31
Dayofweek	4	4	4	4	4
Dayofyear	212	212	212	212	212
Is_month_end	True	True	True	True	True
Is_month_start	False	False	False	False	False
Is_quarter_end	False	False	False	False	False
Is_quarter_start	False	False	False	False	False
Is_year_end	False	False	False	False	False
Is_year_start	False	False	False	False	False
Elapsed	1438300800	1438300800	1438300800	1438300800	1438300800
StoreType	c	a	a	c	a
Assortment	a	a	a	c	a
CompetitionDistance	1270	570	14130	620	29910
CompetitionOpenSinceMonth	9	11	12	9	4
CompetitionOpenSinceYear	2008	2007	2006	2009	2015
Promo2	0	1	1	0	0
Promo2SinceWeek	1	13	14	1	1
...
Min_Sea_Level_PressurehPa	1015	1017	1017	1014	1016
Max_VisibilityKm	31	10	31	10	10
Mean_VisibilityKm	15	10	14	10	10
Min_VisibilitykM	10	10	10	10	10
Max_Wind_SpeedKm_h	24	14	14	23	14
Mean_Wind_SpeedKm_h	11	11	5	16	11
Max_Gust_SpeedKm_h	NaN	NaN	NaN	NaN	NaN
Precipitationmm	0	0	0	0	0
CloudCover	1	4	2	6	4
Events	Fog	Fog	Fog	NaN	NaN
WindDirDegrees	13	309	354	282	290
StateName	Hessen	Thueringen	NordrheinWestfalen	Berlin	Sachsen

	0	1	2	3	4
CompetitionOpenSince	2008-09-15 00:00:00	2007-11-15 00:00:00	2006-12-15 00:00:00	2009-09-15 00:00:00	2015-04-15 00:00:00
CompetitionDaysOpen	2510	2815	3150	2145	107
CompetitionMonthsOpen	24	24	24	24	3
Promo2Since	1900-01-01 00:00:00	2010-03-29 00:00:00	2011-04-04 00:00:00	1900-01-01 00:00:00	1900-01-01 00:00:00
Promo2Days	0	1950	1579	0	0
Promo2Weeks	0	25	25	0	0
AfterSchoolHoliday	0	0	0	0	0
BeforeSchoolHoliday	0	0	0	0	0
AfterStateHoliday	57	67	57	67	57
BeforeStateHoliday	0	0	0	0	0
AfterPromo	0	0	0	0	0
BeforePromo	0	0	0	0	0
SchoolHoliday_bw	5	5	5	5	5
StateHoliday_bw	0	0	0	0	0
Promo_bw	5	5	5	5	5
SchoolHoliday_fw	7	1	5	1	1
StateHoliday_fw	0	0	0	0	0
Promo_fw	5	1	5	1	1

93 rows × 5 columns

这里的关键是，我们要对一个特定的时间和特定的商店id，预测销售额（sales）的值。sales是因变量。

预处理 [16:52]

首先要讲的是预处理（pre-processes）。你们已经学过了变形（transform）。**变形是对从数据集中取出的东西每次都运行的一段代码**，它很适合做我们今天要学习的数据增强，它每次会生成一个不同的随机数。**预处理**和**变形**很像，但有一点区别，就是它是在你训练之前一次性完成的。这很重要，它们在训练集上运行一次，然后生成的各种状态或者元数据会和验证集、测试集共享。

我来给你们看个例子。当我们做图像识别时，我们有代表所有不同宠物品种的一个类型集合，它们被转换成数字。做这个的东西就是在后台创建的一个预处理函数（preprocessor）。这保证训练集的类别和验证集、测试集的类别是一样的。我们这里也会做一些类似的事情。比如，如果我们从数据里取了一个小的子集做尝试。这是在处理一个新的数据集时很好的做法。

```

1 idx = np.random.permutation(range(n))[:2000]
2 idx.sort()
3 small_train_df = train_df.iloc[idx[:1000]]
4 small_test_df = train_df.iloc[idx[1000:]]
5 small_cont_vars = ['CompetitionDistance', 'Mean_Humidity']
6 small_cat_vars = ['Store', 'DayOfWeek', 'PromoInterval']
7 small_train_df = small_train_df[small_cat_vars + small_cont_vars + ['Sales']]
8 small_test_df = small_test_df[small_cat_vars + small_cont_vars + ['Sales']]
```

我随机取了2000个ID。然后分成一个小的训练集和小的测试集，每个一半。然后取出5列。然后用这个做些尝试。简单好用。这是训练集的前几行：

```
1 | small_train_df.head()
```

	Store	DayOfWeek	PromoInterval	CompetitionDistance	Mean_Humidity	Sales
280	281	5	NaN	6970.0	61	8053
584	586	5	NaN	250.0	61	17879
588	590	5	Jan,Apr,Jul,Oct	4520.0	51	7250
847	849	5	NaN	5000.0	67	10829
896	899	5	Jan,Apr,Jul,Oct	2590.0	55	5952

可以看到，有一列叫“promo interval（促销时间）”，里面有这些字符串，有的是缺失的。在 Pandas里，缺失的会显示成 NaN。

Preprocessor: Categorify [18:39]

第一个要介绍的预处理方法（preprocessor）是Categorify（分类）。

```
1 | categorify = Categorify(small_cat_vars, small_cont_vars)
2 | categorify(small_train_df)
3 | categorify(small_test_df, test=True)
```

Categorify做的事情和图片识别里 .classes 对因变量做的事情一样。它取出这些字符串，找出所有可能的唯一的值，对它们创建一个列表，然后把字符串转成数字。如果我在训练集调用，它会在这里创建分类（small_train_df），然后我在测试集上调用，传入 test=true，它会用和之前相同的类别。现在，当我运行 .head 时，它看起来是完全一样的：

```
1 | small_test_df.head()
```

	Store	DayOfWeek	PromoInterval	CompetitionDistance	Mean_Humidity	Sales
428412	NaN	2	NaN	840.0	89	8343
428541	1050.0	2	Mar,Jun,Sept,Dec	13170.0	78	4945
428813	NaN	1	Jan,Apr,Jul,Oct	11680.0	85	4946
430157	414.0	6	Jan,Apr,Jul,Oct	6210.0	88	6952
431137	285.0	5	NaN	2410.0	57	5377

这是因为Pandas把这个转成了一个类别变量，里面存储了数字，但对外还是显示字符串。我可以看下 promo interval内部，看看它的 cat.categories，这是标准的Pandas方法，它显示了一个列表，这在 fastai里叫“classes”，在Pandas叫“categories”。

```
1 | small_train_df.PromoInterval.cat.categories
```

```
1 | Index(['Feb', 'May', 'Aug', 'Nov', 'Jan', 'Apr', 'Jul', 'Oct', 'Mar', 'Jun', 'Sept', 'Dec'],
       dtype='object')
```

```
1 | small_train_df['PromoInterval'].cat.codes[:5]
```

```
1 280 -1
2 584 -1
3 588 1
4 847 -1
5 896 1
6 dtype: int8
```

如果我查看 `cat.codes`，你可以看到这里的list存的是数字，值是 $(-1, -1, 1, -1, 1)$ 。这些-1代表什么？代表 `NaN`，它们代表“missing，缺失（没有数据）”。Pandas用这个特殊的标志-1表示没有数据。

你们知道，这些最终会放进一个embedding矩阵。我们不能在embedding矩阵里查找-1。所以在 fastai内部，我们对所有这些都加了1。

Preprocessor: Fill Missing (填充缺失) [20:18]

另外一个有用的预处理方法是 `FillMissing`。还是一样，你可以在data frame上调用它，在测试集调用时加上 `test=True`。

```
1 fill_missing = FillMissing(small_cat_vars, small_cont_vars)
2 fill_missing(small_train_df)
3 fill_missing(small_test_df, test=True)
```

```
1 small_train_df[small_train_df['CompetitionDistance_na'] == True]
```

	Store	DayOfWeek	PromoInterval	CompetitionDistance	Mean_Humidity	Sales	CompetitionDistance_na
78375	622	5	NaN	2380.0	71	5390	True
161185	622	6	NaN	2380.0	91	2659	True
363369	879	4	Feb,May,Aug,Nov	2380.0	73	4788	True

这会对所有有缺失值的列创建一个额外的列，列名是在原来的列名后添加下划线加na（例如 `CompetitionDistance_na`），当它的值是缺失的时候，对应的值被设成true。然后我们用中位数替换掉 competition distance。为什么这样做？通常，缺失数据这个信息本身就很有价值（这个缺失可以帮助预测收入）。我们把这个信息放在一个布尔列里，这样深度学习模型可以用它来做预测。

但是，我们需要让 competition distance 是一个连续变量，这样我们可以在模型的连续变量部分使用它。我们可以用任何数替代它，因为是否缺失是更重要的变量，它可以用 `CompetitionDistance_na` 和 `CompetitionDistance` 的组合来做预测。这就是 `FillMissing` 做的。

[21:31]

你不需要手工调用预处理方法。当你调用任何item列表创建方法时，你可以传入一个预处理方法的列表，就像这样：

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

```
1 data = (TabularList.from_df(df, path=path, cat_names=cat_vars,
2                             cont_names=cont_vars, procs=procs)
3         .split_by_idx(valid_idx)
4         .label_from_df(cols=dep_var, label_cls=FloatList,
5                      log=True)
6         .databunch())
```

这就是说“好，我想fill missing，我想categorify，我想normalize（对连续变量，减去平均值，除以标准差，来帮助训练模型）”，你只要说，这是我要做的预处理，然后你只要把它们传到这里就可以了。

然后，你可以执行 `data.export`，它可以保存所有的这个data bunch的元数据，这样，你可以直接加载它，它已经知道你的类别编码是什么、用来替代缺失值的中位数是什么、用来做标准化的均值和标准差是什么。

类别变量，连续变量 (Categorical and Continuous Variables) [22:23]

要创建一个表格数据的data bunch，你要告诉它你的类别变量是什么、你的连续变量是什么。就像我们上周简要讲过的，你的类别不仅是字符串，也包括星期、日期之类的。尽管它们是数字，它们也是类别变量。因为，比如说，日期（一个月里第几天），我不认为它有一个光滑的曲线。我认为一个月的15号、1号和30号和其它日期有不同的消费行为。因此，如果我把它作为一个类别变量，它最终会创建一个embedding矩阵，这些不同的日子可以得到不同的行为。

你已经仔细思考过什么应该是类别变量了。总的来说，当有疑惑时，如果你的类别里没有太多等级（这被叫做基数（cardinality）），如果基数不是很高，我会把它当成类别变量。你可以每种都试下，看看哪个效果更好。

```
1 cat_vars = ['Store', 'DayOfweek', 'Year', 'Month', 'Day', 'StateHoliday',
2             'CompetitionMonthsOpen', 'Promo2Weeks', 'StoreType',
3             'Assortment',
4             'PromoInterval', 'CompetitionOpenSinceYear', 'Promo2SinceYear',
5             'State',
6             'Week', 'Events', 'Promo_fw', 'Promo_bw', 'StateHoliday_fw',
7             'StateHoliday_bw', 'SchoolHoliday_fw', 'SchoolHoliday_bw']
8
9 cont_vars = ['CompetitionDistance', 'Max_TemperatureC', 'Mean_TemperatureC',
10            'Min_TemperatureC', 'Max_Humidity', 'Mean_Humidity',
11            'Min_Humidity',
12            'Max_Wind_SpeedKm_h', 'Mean_Wind_SpeedKm_h', 'CloudCover',
13            'trend',
14            'trend_DE', 'AfterStateHoliday', 'BeforeStateHoliday', 'Promo',
15            'SchoolHoliday']
```

我们最终要传入的data frame，是一个包含类别变量、连续变量、因变量、时间的训练集。这个时间，我们要用它创建一个验证集，验证集的记录数量和测试集的一样，时间从测试集的结束时间开始。这样，我们可以更好地验证我们的模型。

```
1 dep_var = 'Sales'
2 df = train_df[cat_vars + cont_vars + [dep_var, 'Date']].copy()
```

```
1 test_df['Date'].min(), test_df['Date'].max()
```

```
1 (Timestamp('2015-08-01 00:00:00'), Timestamp('2015-09-17 00:00:00'))
```

```
1 cut = train_df['Date'][((train_df['Date'] == train_df['Date']
2 [len(test_df)])].index.max()
2 cut
```

```
1 41395
```

```
1 valid_idx = range(cut)
```

```
1 | df[dep_var].head()
```

```
1 | 0      5263  
2 | 1      6064  
3 | 2      8314  
4 | 3      13995  
5 | 4      4822  
6 | Name: Sales, dtype: int64
```

现在，我们可以创建一个表格列表 (tabular list)。

```
1 | data = (TabularList.from_df(df, path=path, cat_names=cat_vars,  
2 |     cont_names=cont_vars, procs=procs)  
3 |             .split_by_idx(valid_idx)  
4 |             .label_from_df(cols=dep_var, label_cls=FloatList,  
|     log=True)  
4 |             .databunch()
```

这是我们标准的data block API，你们已经见过几次了：

- 从一个data frame取数据，传入所有这些信息
- 把它拆分成训练集、验证集
- 用因变量标注数据

这个东西你可能没有见过，label class (`label_cls=FloatList`)。这是我们的因变量（上面的 `df[dep_var].head()`），可以看到，这是sales。它不是浮点型。它是int64。**如果它是浮点型，fastai会自动认为你想做回归。但这不是浮点，它是一个整型。所以fastai会认为你想做分类。所以我们标注时，我们要告诉它我们想要的标签的类型是一个浮点列表，不是一个类别列表（默认值）。所以这样可以自动把它转成一个回归问题。**然后我们创建了一个data bunch。

关于Doc [25:09]

```
1 | doc(FloatList)
```

我想再提一下 `doc`，用它你可以找到更多关于这个的信息。data blocks API里的标注函数会把所有它们不认识的参数关键词传递到label类。我传入的一个参数是 `log`，它实际上会被传到 `FloatList`，如果你运行 `doc(FloatList)`，可以看到一个摘要：

```
In [31]: doc(FloatList)
```

▼ Model

```
In [29]: max_log_y = np.log(np.max(train_df['Sales'])*1.2)
y_range = torch.tensor([0, max_log_y], device=defaults.device)
```

```
class FloatList [source]
```

```
FloatList(items: Iterator, log: bool = False, kwargs):: ItemList
```

ItemList suitable for storing the floats in items for regression. Will add a log if True
[Show in docs](#)

我还可以跳到完整的文档，里面讲，如果 `log` 是 true，它会取因变量的对数。为什么这样做？这实际上会自动取 `y` 的对数，这样做是因为，之前提过，评价度量是根均方百分比误差。

$$\text{RMSPE} = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{y_i} \right)^2}$$

fastai和PyTorch都没有内置的根均方百分比误差。我甚至不知道这样一个损失函数会不会工作得很好。如果你花些时间计算下，你会发现如果你取 y 和 \hat{y} 的对数，它就是一个差值，不再是计算比例了。换句话说，如果你取 y 的对数，RMSPE就会变成根均方差（root mean squared error）。这正是我们想要的。我们取 y 的对数，然后使用根均方差，这是回归问题的默认损失度函数。

```
In [24]: data = (TabularList.from_df(df, path=path, cat_names=cat_vars, cont_names=cont_vars, procs=procs)
    .split_by_idx(valid_idx)
    .label_from_df(cols=dep_var, label_cls=FloatList, log=True)
    .databunch())
```

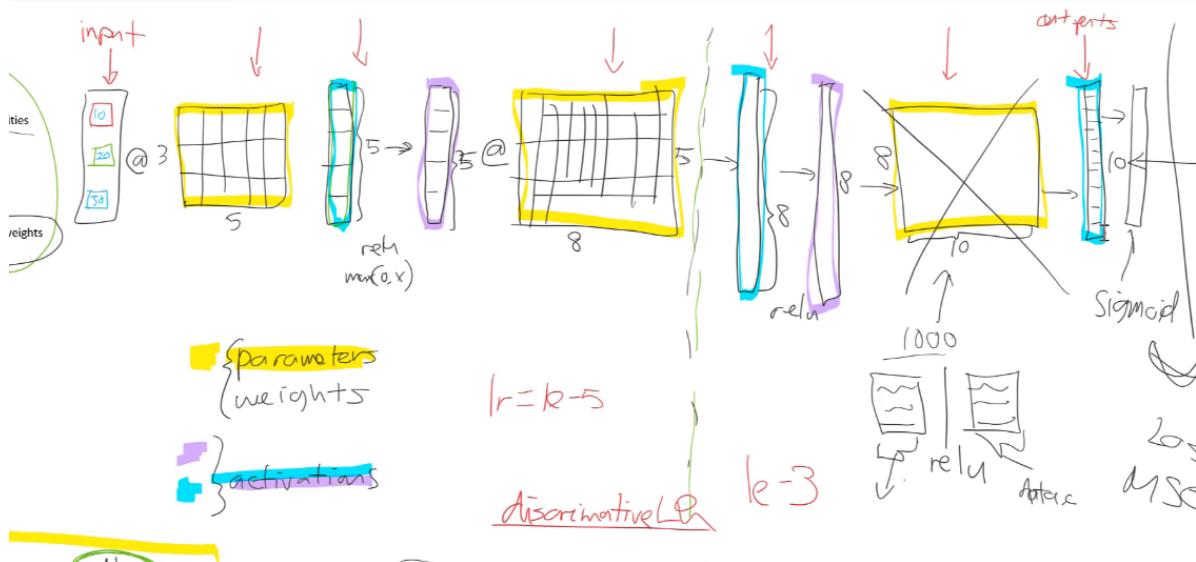
我们这里有这个`log=True`是因为它很常见。基本上每次要预测人口、销售金额之类东西时，这种数据会有长尾分布，你更关心差值百分比而不是准确的绝对差值。所以你需要用`log=True`来计算根均方百分比误差。

y_range [27:12]

```
1 max_log_y = np.log(np.max(train_df['sales'])*1.2)
2 y_range = torch.tensor([0, max_log_y], device=default.device)
```

我们之前学过了`y_range`，它用sigmoid帮助我们得到正确的区间。但这次 y 的值会被取对数，我们要让`y_range`也取对数。所以我取`sales`列的最大值。我会用比1大一点的数乘以它，记得吗，我们讲过你的range最好比数据的range宽一些。然后，我们取对数。这是我们的最大值。这样，我们的`y_range`会是从0到比这个最大值大一点的数。

现在我们得到了data bunch，我们可以用它创建一个表格learner。然后，我们要传入我们的架构。就像我们简单讲过的，对于一个表格模型，我们的架构是最基本的全连接网络，就像我们在这个图画里画的一样：



输入=>矩阵相乘=>非线性层=>矩阵相乘=>非线性层=>矩阵相乘=>非线性层，就这样。值得注意的是，这个竞赛是三年前的，我不清楚对这个排名第三的方案在架构上做些改变会不会带来显著的提升。我们还是直接使用这个简单的全连接网络处理这个问题。

```
1 learn = tabular_learner(data, layers=[1000, 500], ps=[0.001, 0.01],
2 emb_drop=0.04,
3 y_range=y_range, metrics=exp_rmspe)
```

中间的权重矩阵会把1000个激活输入转成500个激活输出，这意味着这个矩阵会有500,000个元素。对以一个只有几十万行的数据集来说这是非常大的。这会过拟合。我们需要避免这个。避免它发生的方法是**使用正则化，而不是减少参数数量**。一种方式是用权重衰减，fastai会自动使用它，如果有需要的话，你可以修改它。这个例子里，我们要用更多的正则化。我们会传入叫做`ps`的东西。这会做dropout。还有这个`embb_drop`，这会做embedding dropout。

Dropout [29:47]

我们来学习下什么是dropout。dropout是一种正则化。这是Nitish Srivastava的[dropout论文](#)，它是Srivastava在Geoffrey Hinton指导下做的硕士论文。

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava
Geoffrey Hinton
Alex Krizhevsky
Ilya Sutskever
Ruslan Salakhutdinov

NITISH@CS.TORONTO.EDU
HINTON@CS.TORONTO.EDU
KRIZ@CS.TORONTO.EDU
ILYA@CS.TORONTO.EDU
RSALAKHU@CS.TORONTO.EDU

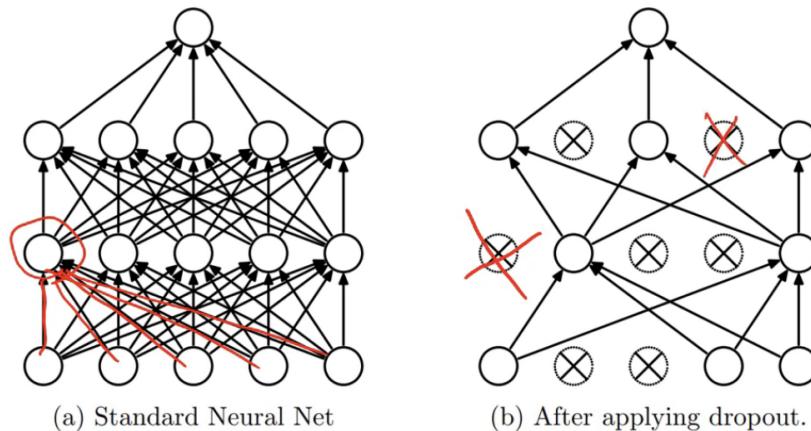


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

这张来自原始论文的图片很好地说明了它做了什么。第一张图是一个标准的全连接网络，每条线代表激活值和一个权重的乘法运算。当有多个箭头进入，就代表一次相加。所以这个激活值（红圈里的），是所有这些输入和这些激活值乘积的和。这是一个普通的全连接神经网络的样子。

对于dropout，我们把这些去掉。我们随机地**去掉一些比例的激活值**，不是权重，不是参数。记住，**神经网络里只有两种数字：参数**（也叫权重），和**激活值**。我们要去掉一些激活值。

可以看到，我们去掉激活值后，连接着它的这些东西也都被去掉了。对每个mini batch，我们去掉不同的激活值。要去掉多少呢？我们按概率 p 做删除。常用的 p 的值是0.5。这意味着什么？这个例子里，可以看到，不仅对隐藏层做了随机删除，也对输入做了删除。删除输入不常用。通常，我们只删除隐藏层里的激活值。这会做什么呢？每次处理一个mini batch时，我随机去掉一些激活值。然后，在下一个mini batch中，我把它们放回来，去掉另外一些。

这意味着没有一个激活值可以记住输入的部分信息，这正是过拟合时出现的情况。如果过拟合了，模型里的一部分参数学会识别特定的图片，而不是通用的特征或者特定的物品。使用dropout，就很难出现这样的情况。Geoffrey Hinton这样描述这个背后的思想：

在银行里，柜员不停地更换，我问他们为什么。他说他不知道，他们经常变动。我认为，这是因为要想骗过银行的话，需要员工们的配合。这让我意识到，在每个样本里随机移动神经元的不同部分会阻止阴谋，这会减少过拟合。

[Hinton: Reddit AMA](#)

他发现每次他去银行，所有的柜员和员工都变了，他意识到这样做的原因一定是他们在预防欺诈。如果他们不停变动，没人可以在一件事情上专注太久，这样他们可以解决欺诈银行的问题。现在，当然，取决于你问Hinton的时间。有时，他说这样做的原因是因为他考虑了尖刺神经元的工作方式，他是一个受过训练的神经学家：

我们不清楚为什么神经元有尖刺。一种理论是它们想要更多噪音，来适应它们，因为我们想有更多的参数来处理数据。这个dropout的主意就是如果你有噪音激活值，你就可以使用一个大得多的模型。

[Hinton: O'Reilly](#)

有观点认为尖刺神经元可以帮助正则化，dropout是一种符合尖刺神经元思想的方法。这很有趣。当你问一些人他们算法的思想是从哪里来时，它从不会是来自数学，总是来自于直觉和物理模拟或类似的东西。

无论如何，我猜很多现象的背后思想和dropout的思路都是相通的。重要的是，它的效果很好。它可以让我们模型更有泛化性，不用付出什么代价。

当然，太多dropout会减少模型的容量，这样会欠拟合。你需要对每一层尝试不同的dropout值，再做决定。

基本在每个fastai learner里，有一个叫 `ps` 的参数，它是每一层的dropout p 的值（概率值）。你可以直接传入一个list，或者你可以传入一个整数，它会创建一个list，list里每个值都是这个数。有时有些不同。比如，对CNN，如果你传入一个整数，它会在最后一层使用这个数，在前面的层使用它的 $1/2$ 。我们尝试按照最佳实践来做。你可以传入你自己的list来设置你想要的准确的dropout。

Dropout和测试时间 [34:47]

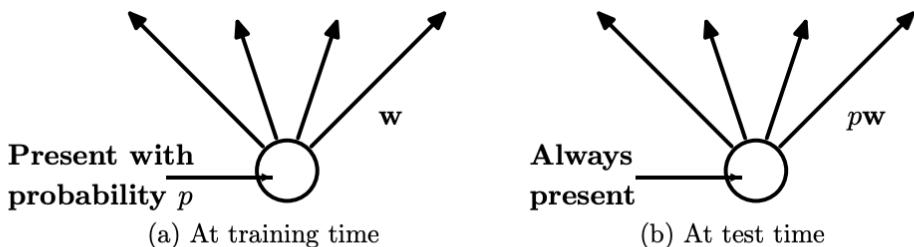


Figure 2: **Left:** A unit at training time that is present with probability p and is connected to units in the next layer with weights w . **Right:** At test time, the unit is always present and the weights are multiplied by p . The output at test time is same as the expected output at training time.

dropout有一个需要注意的特性。就是训练阶段和测试阶段（也叫推理阶段）。训练阶段是我们做这些权重更新（反向传播）的时间段。训练时，dropout按我们刚刚看过的方式工作。测试时，我们关掉dropout，我们不再用dropout，因为我们希望尽可能的精确。我们不是在训练，所以不会引起过拟合。所以我们去掉dropout。这意味着，如果之前的 p 的值是0.5，有一半的激活值被删除了。当它们都存在时，整体的激活值会是之前用的两倍。因此，这个论文里，他们建议测试时在所有权重上乘以 p 。

有意思的是，你可以深入PyTorch源代码，看下实现dropout的C语言代码。

```
noise.bernoulli_(1 - p);
noise.div_(1 - p);
return multiply<inplace>(input, noise);
```

可以看到它们做的事情很有意思，它们首先做了一个概率 $1-p$ 的伯努利试验，满足时返回1，否则返回0。这里 p 时dropout的概率， $1-p$ 就是保留激活值的概率。所以最终得到1或0。然后原地（记住在PyTorch里下划线代表“原地”）除以 $1-p$ 。如果它是0，那值没什么变化，还是0。如果它是1， p 是0.5，那么它的值就变成了2。最后，我们把输入原地乘以这个noise（它是dropout的mask）。

也就是说，在PyTorch里，我们在测试时没有改变什么。我们只在训练阶段做了处理，这意味着使用PyTorch时你不需要在测试时做什么特别的事情。不仅是在PyTorch，这是一个很常见的模式。看下PyTorch的代码很有用，它难以置信的酷，非常有价值，只有3行代码，它们用的是C语言，我猜这是因为最终一起执行时这样会快一点。很多库用Python做这个，效果也很好。你可以写一个自己的dropout层，它应该得到和这个一样的结果。这是一个很好的可以自己尝试的实验。看下你能不能用Python创建你自己的dropout层，看下能不能复现和原来的dropout层一样的结果。

[37:38]

```
1 | learn = tabular_learner(data, layers=[1000, 500], ps=[0.001, 0.01],
2 |     emb_drop=0.04,
2 |     y_range=y_range, metrics=exp_rmspe)
```

这就是dropout。这里我们会在第一层用一个非常小的dropout (0.001)，在下一层用一个比较小的dropout (0.01)，然后会在embedding层用特定的dropout。为什么在embedding层用特定的dropout？如果你看下fastai内部的代码，这是我们的tabular模型：

```
class TabularModel(nn.Module):
    "Basic model for tabular data."
    def __init__(self, emb_szs:ListSizes, n_cont:int, out_sz:int, layers:Collection[int], ps:Collection[float]=None,
                 emb_drop:float=0., y_range:OptRange=None, use_bn:bool=True, bn_final:bool=False):
        super().__init__()
        ps = ifnone(ps, [0]*len(layers))
        ps = listify(ps, layers)
        self.embeds = nn.ModuleList([embedding(ni, nf) for ni,nf in emb_szs])
        self.emb_drop = nn.Dropout(emb_drop)
        self.bn_cont = nn.BatchNorm1d(n_cont)
        n_emb = sum(e.embedding_dim for e in self.embeds)
        self.n_emb, self.n_cont, self.y_range = n_emb, n_cont, y_range
        sizes = self.get_sizes(layers, out_sz)
        actns = [nn.ReLU(inplace=True)] * (len(sizes)-2) + [None]
        layers = []
        for i,(n_in,n_out,dp,act) in enumerate(zip(sizes[:-1],sizes[1:],[0.]+ps,actns)):
            layers += bn_drop_lin(n_in, n_out, bn=use_bn and i!=0, p=dp, actn=act)
        if bn_final: layers.append(nn.BatchNorm1d(sizes[-1]))
        self.layers = nn.Sequential(*layers)

    def get_sizes(self, layers, out_sz):
        return [self.n_emb + self.n_cont] + layers + [out_sz]

    def forward(self, x_cat:Tensor, x_cont:Tensor) -> Tensor:
        if self.n_emb != 0:
            x = [e(x_cat[:,i]) for i,e in enumerate(self.embeds)]
            x = torch.cat(x, 1)
            x = self.emb_drop(x)
        if self.n_cont != 0:
            x_cont = self.bn_cont(x_cont)
            x = torch.cat([x, x_cont], 1) if self.n_emb != 0 else x_cont
        x = self.layers(x)
        if self.y_range is not None:
            x = (self.y_range[1]-self.y_range[0]) * torch.sigmoid(x) + self.y_range[0]
        return x
```

你可以看到检查是否有embedding的这一部分 (`forward`方法里 `if self.n_emb != 0:`)

- 我们调用每个embedding
- 我们把embedding放进一个矩阵
- 然后我们调用embedding dropout

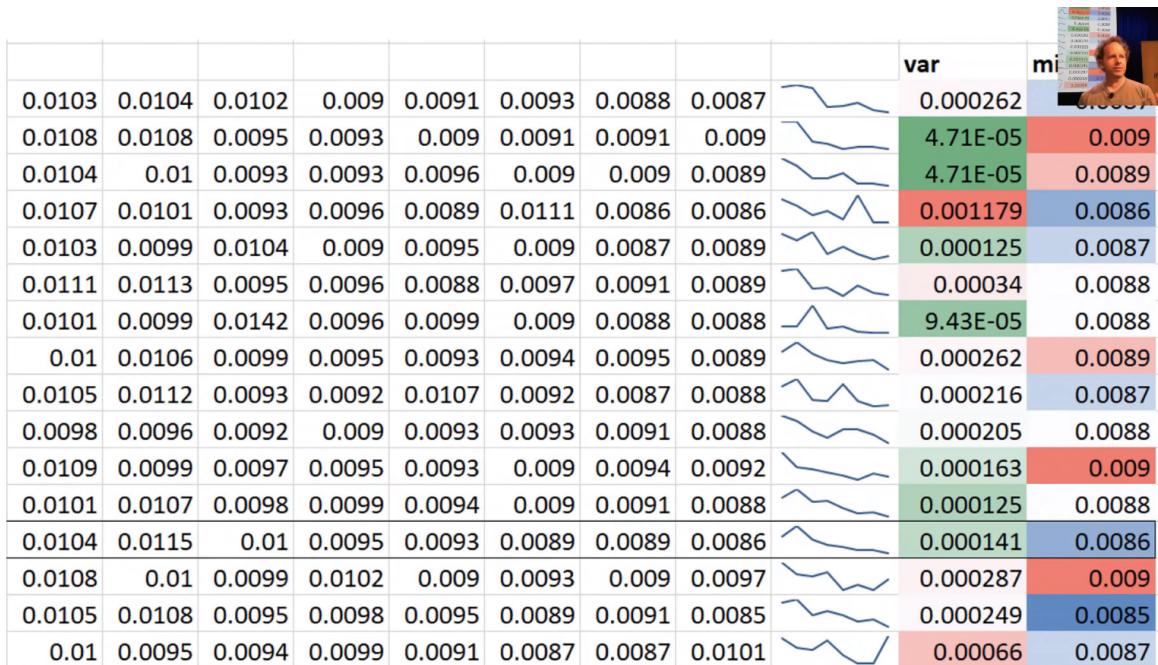
embedding dropout只是一个dropout。它只是dropout module的一个实例。这是合理的，对吗？对于连续变量，连续变量只在一列里。你不会想在上面做dropout，因为这会删除整个输入，你肯定不想这样做。对embedding来说，embedding只是一个矩阵和一个one hot编码矩阵相乘，它只是一个层。所以在embedding的输出上做dropout很有意义。这就是说让我们随机删掉这些embedding里的

一些结果（一些激活值）。所以这是合理的。

我们这样做的另外一个原因是一年前我在这个数据集上做了很多实验。我尝试了很多不同的方法，做各种事情。你在这里可以看到。

Row Labels	Avg Score	Column L		No scale Total		Scaled Total		Grand Total	
		No scale	Dict	Eq	Dict	Scaled	Eq	Dict	
No init	0.0088	0.0087		0.0087	0.0089	0.0089	0.0089	0.0088	0.0088
Dense	0.0087	0.0088		0.0087	0.0090	0.0088	0.0089	0.0088	0.0088
Split	0.0089	0.0086		0.0087	0.0088	0.0090	0.0089	0.0088	0.0088
Init	0.0086	0.0090		0.0088	0.0086	0.0087	0.0087	0.0087	0.0087
Dense	0.0085	0.0090		0.0087	0.0086	0.0087	0.0086	0.0087	0.0087
Split	0.0088	0.0089		0.0089	0.0087	0.0087	0.0087	0.0087	0.0088
Grand Total	0.0087	0.0088		0.0088	0.0088	0.0088	0.0088	0.0088	0.0088

我把它们都放在一个电子表格里（当然，是Microsoft Excel），把它们放进一个透视表（pivot table）来把它们放在一起，来找出不同的选择下，效果更好和效果更差的超参数、架构。然后我做了这样一些图：



这是不同参数、架构的组合的训练图形总结。我发现有一个组合能一直得到很好的预测准确度，训练的波动很低，可以看到，这是一个很光滑的曲线。

这是我用fastai库做的实验一个例子。我发现embedding dropout效果很好。为什么我做这些特别的实验？这是因为我看Kaggle获奖者论文里的“什么更有效”这一部分，对论文里的一些内容，我觉得，“他们有其他的选择，为什么他们不这样”，我试着这样做，找出什么是有效的、什么没有效果，找到一些能改进的方法。你也可以做这样的实验，尝试不同的模型和架构、不同的dropout、层数、激活值数量，等等。

[41:02]

创建完learner后，我们可以输入`learn.model`看下它：

```
1 | learn.model
```

```
1 | TabularModel(  
2 |     (embeds): ModuleList(  
3 |         [
```

```

3     (0): Embedding(1116, 50)
4     (1): Embedding(8, 5)
5     (2): Embedding(4, 3)
6     (3): Embedding(13, 7)
7     (4): Embedding(32, 17)
8     (5): Embedding(3, 2)
9     (6): Embedding(26, 14)
10    (7): Embedding(27, 14)
11    (8): Embedding(5, 3)
12    (9): Embedding(4, 3)
13    (10): Embedding(4, 3)
14    (11): Embedding(24, 13)
15    (12): Embedding(9, 5)
16    (13): Embedding(13, 7)
17    (14): Embedding(53, 27)
18    (15): Embedding(22, 12)
19    (16): Embedding(7, 4)
20    (17): Embedding(7, 4)
21    (18): Embedding(4, 3)
22    (19): Embedding(4, 3)
23    (20): Embedding(9, 5)
24    (21): Embedding(9, 5)
25    (22): Embedding(3, 2)
26    (23): Embedding(3, 2)
27 )
28 (emb_drop): Dropout(p=0.04)
29 (bn_cont): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
30 (layers): Sequential(
31     (0): Linear(in_features=229, out_features=1000, bias=True)
32     (1): ReLU(inplace)
33     (2): BatchNorm1d(1000, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
34     (3): Dropout(p=0.001)
35     (4): Linear(in_features=1000, out_features=500, bias=True)
36     (5): ReLU(inplace)
37     (6): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
38     (7): Dropout(p=0.01)
39     (8): Linear(in_features=500, out_features=1, bias=True)
40   )
41 )

```

像期望的一样，这里有一整堆embedding。每个embedding矩阵显示了每个输入的数量（第一个数）。和 cat_vars list 对照，第一个是 store，这里有1,116个商店。第二个数字，是embedding的大小。这个数字你可以自己选择。

fastai有一些默认值，它们几乎所有时间都可以工作得很好。我几乎不改变它们。当你创建你的 tabular_learner 时，你可以传入一个设置embedding大小的 dictionary，字典里存上要修改默认值的变量名和对应的embedding大小。

然后是embedding dropout层，再后面是一个有16个输入的batch norm层，这16个输入是合理的，因为我们有16个连续变量。

```
1 | len(data.train_ds.cont_names)
```

`cont_names` 的长度是16。这是用于连续变量用的东西。我们用 `bn_cont` 处理连续变量：

```
def forward(self, x_cat:Tensor, x_cont:Tensor) -> Tensor:
    if self.n_emb != 0:
        x = [e(x_cat[:,i]) for i,e in enumerate(self embeds)]
        x = torch.cat(x, 1)
        x = self.emb_drop(x)
    if self.n_cont != 0:
        x_cont = self.bn_cont(x_cont)
        x = torch.cat([x, x_cont], 1) if self.n_emb != 0 else x_cont
    x = self.layers(x)
    if self.y_range is not None:
        x = (self.y_range[1]-self.y_range[0]) * torch.sigmoid(x) + self.y_range[0]
    return x
```

`bn_cont` 是一个 `nn.BatchNorm1d`。这是什么？简单说，这是一个我实验过的做batch norm的东西，我发现它效果很好。我来讲给你们，它具体是什么。

- 它是一个正则化 (regularization)
- 它是一个训练辅助方法 (training helper)

它叫batch normalization (批次标准化)，它来自[这篇论文](#)。

[43:06]

在做这个之前，我想讲一个关于dropout的有趣的事情，我讲过，它是一个硕士论文。但它不是一个普通的硕士论文，它是过去十年最有影响力的论文之一。



Chris Gorgolewski

@ChrisFiloG

Follow

v

Did you know that Dropout was originally introduced in a Master's thesis and was rejected from NIPS? Was disseminated via #arxiv! #OHB2018

Dropout

Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

- Srivastava's Master's(!) thesis.
- Training scheme that randomly masks neurons at every step.
- Usually gives a small performance boost.
- Mysterious.

This paper was rejected from NIPS in 2012, and propagated solely as a preprint on arxiv.

5:30 PM - 20 Jun 2018

它之前被神经网络会议NIPS (现在叫NeurIPS) 拒绝了。我认为这是一个有意思的事情，这说明我们的学术社区不擅长识别最终很重要的东西。一般，人们只接受他们从事的、他们能理解的范围里的东西。dropout不在这个范围。很难理解它是什么。这值得关注。

这提醒我们，如果你从一个实践者成长为一个真正的研究者，不要只关注那些每个人都谈论的东西，要关注那些你认为可能有意义的事。因为每个人都谈论的东西最终一般不会特别有意义。在有巨大影响力的论文刚出现时，学术圈不太善于识别它们。

Batch Normalization [44:28]

相反，Batch normalization被立即认为是有巨大影响的。我清楚地记得在2015，当它刚出现时，每个人都在谈论它。这是因为这太明显了，它们展示了这个图片：

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe

Google Inc., sioffe@google.com

Christian Szegedy

Google Inc., szegedy@google.com

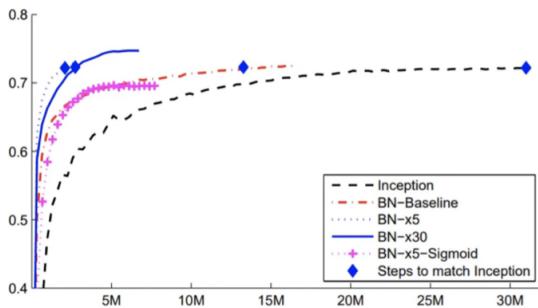


Figure 2: Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

它展示当时的最佳ImageNet模型Inception。这是它得到一个很好的结果花的时间，然后他们用这个叫batch norm的新东西做了同样的事情，它们做得非常非常快。这足够让很多人说“哇，这很有用”

具体说，这个东西叫batch normalization，它通过减少内部协变量移位 (internal covariate shift) 来加速训练。什么是内部协变量移位？这不重要。因为batch norm是一些研究者靠直觉提出来的、要试试看的东西。他们成功了，它的效果很好，他们事后补充了一些数学分析来试着说明为什么它是有效的。最终，这些数学分析完全错了。

[45:29]

How Does Batch Normalization Help Optimization?

Shibani Santurkar*

MIT

shibani@mit.edu

Dimitris Tsipras*

MIT

tsipras@mit.edu

Andrew Ilyas*

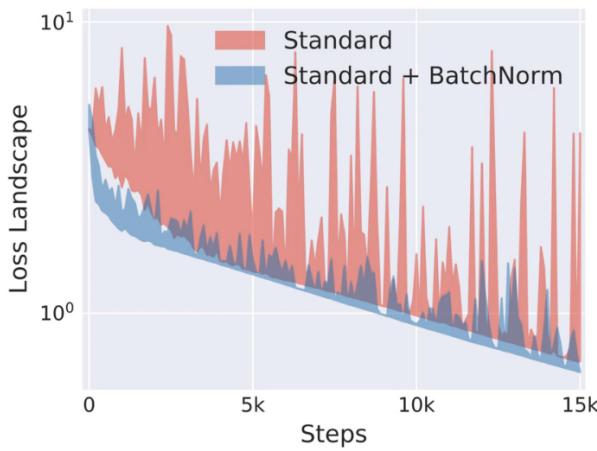
MIT

ailyas@mit.edu

Aleksander Madry

MIT

madry@mit.edu



“the positive impact of BatchNorm on training might be somewhat serendipitous”

最近两个月，有两篇论文（人们花了三年指出这个问题）指出batch normalization根本没有减少内部协变量移位。并且，就算真的减少了，也对模型的效果没什么帮助。我认为这是一个有意义的检查，也说明为什么我们要专注于做实践者和实验主义者，提升直觉。

batch norm做的是你看到的这个图片上的东西。这是步数 (steps) 或者batch (x-axis)，这是损失 (y-axis)。红线是没有batch norm时训练的情况，波动很大。蓝线是用了batch norm后，训练的情况，波动不大了。这意味着，使用batch norm你可以提高你的学习率。因为这些大的波动代表你的权重跳出到权重空间里可怕的、再也没办法回来的地方，这很危险，波动越多，越多遇到危险的次数就越多。如果它没那么多波动，你可以用更高的学习率训练它。这就是它的原理。

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

这是算法，它很简单。算法会取一个mini batch。我们有了一个mini batch，记住，这是一个层，进入这个层的是激活值。batch norm是一个层，它会接收激活值。激活值被叫做 x_1, x_2, x_3 等等。

1. 首先，我们要找到这些激活值的平均值 (mean)，和除以数量就是平均值。
2. 第二件事，我们找到激活值的方差 (variance)，差值的平方求和，再除以数量是就方差。
3. 然后做标准化 (normalize)，激活值减去平均指除以标准差，就是标准化。这实际上不是很重要。我们曾经以为很重要，但后来发现不是。真正重要的部分是下面的东西。
4. 我们取这些值，加上一个bias的向量 (这里把它叫做beta)。我们之前已经看过了。我们用过这个了。我们像之前一样，加上bias。然后，我们要用另外一个和bias很像的东西，但不是加上它，我们会乘以它。这些参数gamma γ 和 beta β 是要学习的参数。

记住，在神经网络里，只有两种数字：激活值和参数。这些是参数。是用梯度下降学习到的东西。 β 只是一个普通的bias层， γ 是一个做乘法的bias层。没有人这样叫它，但它就是这样的。它就像bias一样，但我们乘以它，而不是加上它。这就是batch norm。这就是这一层做的事。

为什么这可以实现了不起的结果？我不清楚有没有人之前准确地把这写下来。如果有，抱歉这里没有引用它，因为我没有看过。让我解释下。究竟发生了什么。我们的预测值 y -hat是权重的函数，参数数量可以达到上百万，它也是一个关于输入的函数。

$$\hat{y} = f(w_1, w_2 \dots w_{1000000}, \vec{x})$$

这个函数 f 是我们的神经网络函数，无论神经网络里是什么。然后，我们的损失函数，假设是均方差，只是实际值减去预测值的平方。

$$L = \sum (y - \hat{y})^2$$

假设我们要预测电影评分，它们在1到5之间。我们已经训练过模型，最后的激活值在-1到1之间。这和它们应该是的值不一致。区间不一致、平均值不一致。我们应该怎么做？一个方法是用一个新的权重集合，让区间增长，让平均值增长。但是这很难，因为所有这些参数有很密切复杂的相互作用。所有这些非线性单元，是组合在一起的。想让它们变大，要穿过复杂的参数空间，我们用像动量、Adam之类的东西来帮助我们，但还要做很多事情才能达到目的。这会花很长时间，波动会很大。

我们这样做怎么样？乘以 g ，再加上 b 会怎么样？

$$\hat{y} = f(w_1, w_2 \dots w_{1000000}, \vec{x}) \times g + b$$

我们多加了两个参数向量。现在它很简单。这个数 g 直接增大区间。这个数 b 直接改变平均值。没有相互作用和复杂性，都是直来直去的，这就是 batch norm 做的事。batch norm 让使输出变大变小这个重要工作更容易做到。这就是为什么我们能得到这样的结果。

这些细节，在某种意义上，不是特别重要。真正重要的是你肯定需要使用它。如果不使用它，也会用类似的东西。现在，有很多其它类型的标准化方法，但 batch norm 效果很好。我们在 fastai 里用的其它的标准化方法主要是 weight norm，这是最近几个月新开发的。

[51:50]

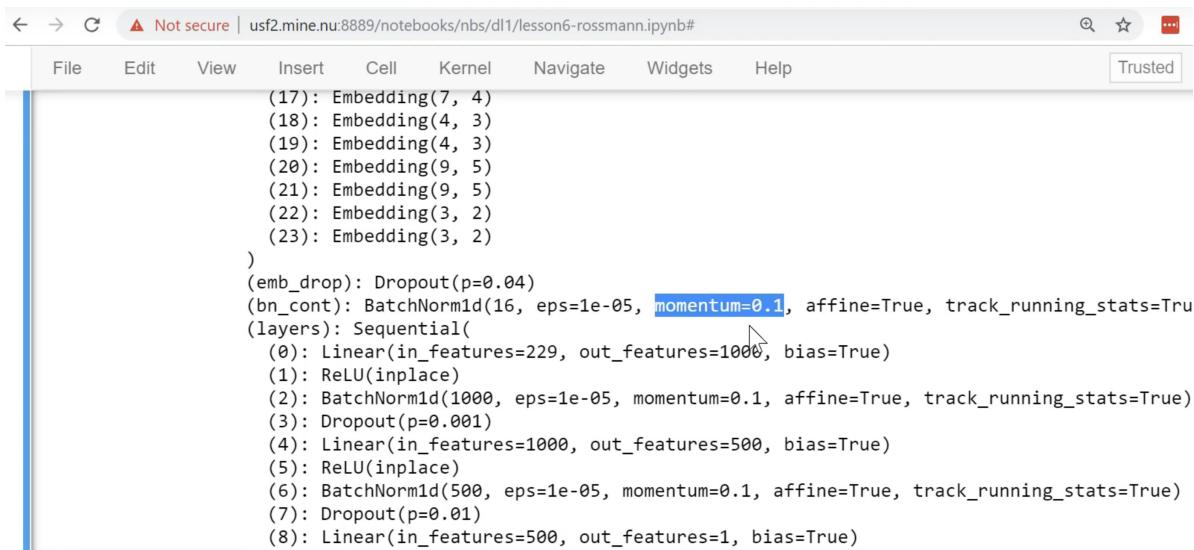
```
class TabularModel(nn.Module):
    "Basic model for tabular data."
    def __init__(self, emb_szs:ListSizes, n_cont:int, out_sz:int, layers:Collection[int], ps:Collection[float]=None,
                 emb_drop:float=0., y_range:OptRange=None, use_bn:bool=True, bn_final:bool=False):
        super().__init__()
        ps = ifnone(ps, [0]*len(layers))
        ps = listify(ps, layers)
        self.embeds = nn.ModuleList([embedding(ni, nf) for ni,nf in emb_szs])
        self.emb_drop = nn.Dropout(emb_drop)
        self.bn_cont = nn.BatchNorm1d(n_cont)
        n_emb = sum(e.embedding_dim for e in self.embeds)
        self.n_emb, self.n_cont, self.y_range = n_emb, n_cont, y_range
        sizes = self.get_sizes(layers, out_sz)
        actns = [nn.ReLU(inplace=True)] * (len(sizes)-2) + [None]
        layers = []
        for i,(n_in,n_out,dp,act) in enumerate(zip(sizes[:-1],sizes[1:],ps,actns)):
            layers += [nn.Linear(n_in, n_out, bias=False), nn.BatchNorm1d(n_out, momentum=0.1), nn.ReLU(inplace=True)]
            if dp > 0:
                layers += [nn.Dropout(p=dp)]
        if bn_final: layers.append(nn.BatchNorm1d(sizes[-1]))
        self.layers = nn.Sequential(*layers)

    def get_sizes(self, layers, out_sz):
        return [self.n_emb + self.n_cont] + layers + [out_sz]

    def forward(self, x_cat:Tensor, x_cont:Tensor) -> Tensor:
        if self.n_emb != 0:
            x = [e(x_cat[:,i]) for i,e in enumerate(self.embeds)]
            x = torch.cat(x, 1)
            x = self.emb_drop(x)
        if self.n_cont != 0:
            x_cont = self.bn_cont(x_cont)
            x = torch.cat([x, x_cont], 1) if self.n_emb != 0 else x_cont
        x = self.layers(x)
        if self.y_range is not None:
            x = (self.y_range[1]-self.y_range[0]) * torch.sigmoid(x) + self.y_range[0]
        return x
```

这就是 batch norm，我们为每一个连续变量创建了一个 batch norm 层。`n_cont` 是连续变量的数量。在 fastai 里，`n_something` 通常代表这个东西的数量。`cont` 通常代表连续（continuous）。这里是我们使用它的地方。我们取到连续变量，让它们通过一个 batch norm 层。

你可以在模型里的这个地方看到它。



```
File Edit View Insert Cell Kernel Navigate Widgets Help Trusted
(17): Embedding(7, 4)
(18): Embedding(4, 3)
(19): Embedding(4, 3)
(20): Embedding(9, 5)
(21): Embedding(9, 5)
(22): Embedding(3, 2)
(23): Embedding(3, 2)
)
(emb_drop): Dropout(p=0.04)
(bn_cont): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(layers): Sequential(
    (0): Linear(in_features=229, out_features=1000, bias=True)
    (1): ReLU(inplace)
    (2): BatchNorm1d(1000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.001)
    (4): Linear(in_features=1000, out_features=500, bias=True)
    (5): ReLU(inplace)
    (6): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): Dropout(p=0.01)
    (8): Linear(in_features=500, out_features=1, bias=True)
```

一个值得注意的东西是这里的动量 (momentum)。这不是优化里的动量，是指数加权移动平均里的动量。具体来说，这个 (batch norm算法里的) 平均值和标准差，我们没有为每一个mini batch用不同的平均值和标准差。如果这样着，它会改变非常多，非常难训练。我们用平均值和标准差的指数加权移动平均值。如果你不记得这是什么意思，就回去看下上周的课程，复习下指数加权移动平均，上节课我们在excel里实现动量和Adam时讲的。

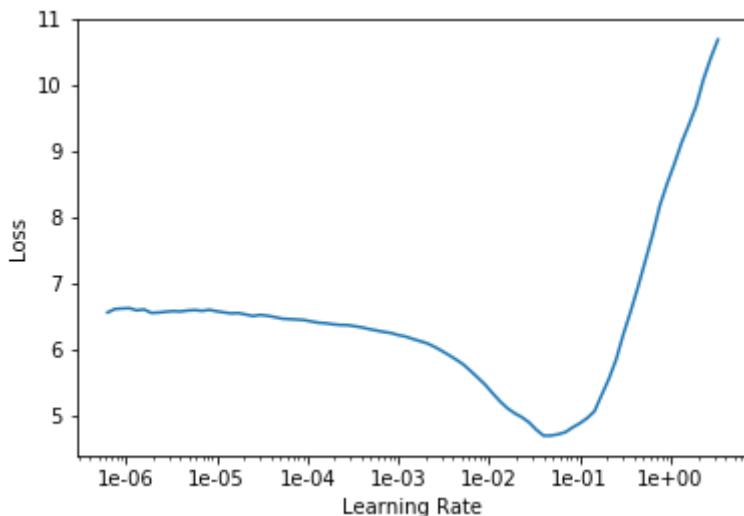
[53:10]

你可以通过传入一个不同的值来改变一个batch norm层里动量的数量。如果你用了一个较小的数，这意味着从一个mini batch到一个mini batch平均值和标准差会变小，它们受正则化的影响也会变小。一个较大的数，意味着方差会变大，受正则化的影响也会变大。如果这个参数设得好，训练会更好。平均值和标准差的动量增加了正则化的效果。

当你用了batch norm，你需要用更大的学习率。这是我们的模型。你可以运行 `lr_find`，你可以看下：

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

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

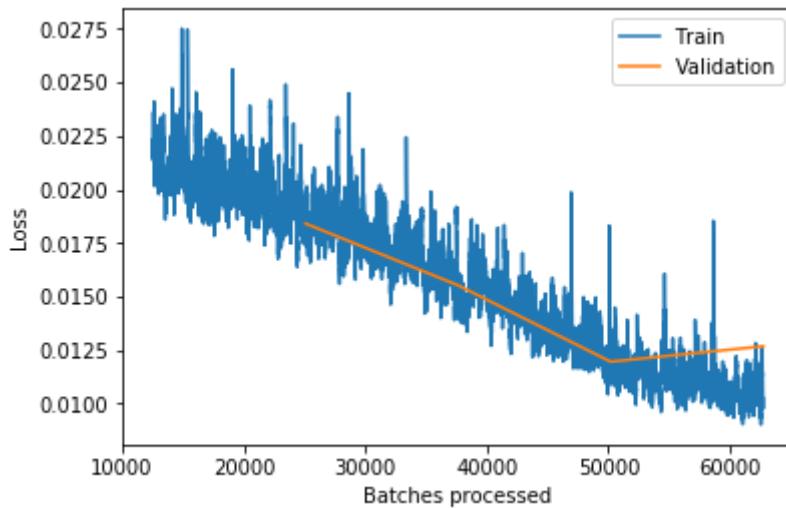


```
1 | learn.fit_one_cycle(5, 1e-3, wd=0.2)
```

```
1 | Total time: 14:18
2 | epoch  train_loss  valid_loss  exp_rmspe
3 | 1      0.021467    0.023627    0.149858  (02:49)
4 | 2      0.017700    0.018403    0.128610  (02:52)
5 | 3      0.014242    0.015516    0.116233  (02:51)
6 | 4      0.012754    0.011944    0.108742  (02:53)
7 | 5      0.010238    0.012665    0.105895  (02:52)
```

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

```
1 | learn.recorder.plot_losses(last=-1)
```



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

```
1 | learn.fit_one_cycle(5, 3e-4)
```

```
1 | Total time: 13:52
2 | epoch  train_loss  valid_loss  exp_rmspe
3 | 1      0.018280   0.021080   0.118549   (02:49)
4 | 2      0.018260   0.015992   0.121107   (02:50)
5 | 3      0.015710   0.015826   0.113787   (02:44)
6 | 4      0.011987   0.013806   0.109169   (02:43)
7 | 5      0.011023   0.011944   0.104263   (02:42)
```

```
1 | learn.fit_one_cycle(5, 3e-4)
```

Total time: 14:41

epoch	train_loss	valid_loss	exp_rmspe
1	0.012831	0.012518	0.106848
2	0.011145	0.013722	0.109208
3	0.011676	0.015752	0.115598
4	0.009419	0.012901	0.107179
5	0.009156	0.011122	0.103746

最终得到0.103。竞赛的第十名时0.108，这看起来很好。但不要太当回事，因为你还要用真实的训练集，把它提交到Kaggle，但可以看到，我们至少是2015年的顶尖水平了。我说过，它们至今没有做过架构上的改进。当时没有batch norm，我们添加了batch norm应该能得到更好的结果，并且运行得更快。他们的模型用一个比较低的学习率，训练了很久。可以看到，这个用了不到45分钟。这很好很快。

提问: 你使用dropout和其他正则化方法比如权重衰减 (weight decay)、L2正则化等等的比例是怎样的？[\[54:49\]](#)

记得吗，L2正则化和权重衰减是做同一件事的两种方式，我们应该总是用权重衰减，不用L2正则化。所以权重衰减是一个选择。batch norm有正则化的效果。我们马上要学习的数据增强（data augmentation）是一个选择。还有dropout。我们应该总是用batch norm。它很简单。我们等下会学数据增强。剩下就是比较dropout和权重衰减。我不知道。我没有看到过有人做过令人信服的关于如何结合这两个东西的研究。可以总是用其中一个，而不用另一个吗？为什么能？为什么不能？我想没人可以解答。在实践中，看起来，通常你需要同时用这两个。你通常需要用权重衰减，但也经常需要用dropout。说实话，我不知道为什么。我没有见过有人解释为什么，如何做选择。这个需要你去尝试，来获得这样一种感觉，知道对你的问题，哪个方法是有效的。我认为我们在每个learner里提供的默认方法可以在大多数场景工作得很好。但是，肯定要试下。

数据增强 (Data augmentation) [56:45]

我们要学的另外一种正则化是数据增强。数据增强是一个最不受重视、没有深入研究的正则化方法。但我认为这是一个最让我兴奋的方法。原因是，它几乎没有成本。你用数据增强可以得到更好的泛化性，不需要花更多的训练时间，不会欠拟合。我来解释下。

[lesson6-pets-more.ipynb](#)

我们现在要回到计算机视觉，回到宠物数据集。我们加载进它。在宠物数据集里，图片在 `images` 目录：

```
1 %reload_ext autoreload
2 %autoreload 2
3 %matplotlib inline
4
5 from fastai.vision import *
```

```
1 bs = 64
```

```
1 path = untar_data(URLS.PETS) / 'images'
```

我像之前一样调用 `get_transforms`，但调用 `get_transforms` 时，有很多参数可以提供：

```
1 tfms = get_transforms(max_rotate=20, max_zoom=1.3, max_lighting=0.4,
2                         max_warp=0.4, p_affine=1., p_lighting=1.)
```

目前，我们还没有改变太多。为了真正理解数据增强，我要写出所有这些默认值，这里有代表仿射变形（affine transform）发生概率的参数、亮度改变概率的参数，我把它们都设成1。它们都会被变形，我会做更多的旋转、更多的缩放、更多的亮度改变、更多的扭曲变形。

这像代表什么呢？你需要查看文档，输入doc，这是简要的文档：

The screenshot shows a Jupyter Notebook interface. In the top left, it says "In [5]: doc(get_transforms)". Below it, the code `get_transforms [source]` is shown. The output block contains the function definition for `get_transforms` and a detailed description: "Utility func to easily create a list of flip, rotate, zoom, warp, lighting transforms." At the bottom of the output block, there is a link "Show in docs".

```
get_transforms (do_flip: bool =`True`, `flip_vert`: bool =`False`, `max_rotate`: float =`10.0`,
`max_zoom`: float =`1.1`, `max_lighting`: float =`0.2`, `max_warp`: float =`0.2`, `p_affine`: float =`0.75`,
`p_lighting`: float =`0.75`, `xtra_tfms`: Optional [Collection [ Transform]] =`None`) →
Collection [ Transform]
```

Utility func to easily create a list of flip, rotate, zoom, warp, lighting transforms.
Show in docs

完整的文档在docs里，我点击[Show in docs](#)就可以看到。文档里写了这些是做什么的，一般docs里最有用的部分是在顶部，你可以看到概要的说明。

这里，有一个叫[List of transforms](#)的东西，你可以看到每一个变形方法都有一些不同值的例子。

List of transforms

Here is the list of all the deterministic functions on which the transforms are built. As explained before, each of those can have a probability `p` of being executed, and any time an argument is type-annotated with a random function, it's possible to randomize it via that function.

brightness

[source]

```
brightness(`x`, `change`: uniform) → Image :: TfmLighting
```

Apply `change` in brightness of image `x`.

This transform adjusts the brightness of the image depending on the value in `change`. A `change` of 0 will transform the image to black and a `change` of 1 will transform the image to white. `change = 0.5` doesn't do adjust the brightness.

```
fig, axs = plt.subplots(1,5, figsize=(12,4))
for change, ax in zip(np.linspace(0.1,0.9,5), axs):
    brightness(get_ex(), change).show(ax=ax, title=f'change={change:.1f}')
```



这是亮度（brightness）。你要读下它们，记住这些notebook，你可以打开它们，运行里面的代码，得到输出。所有这HTML文件都是从fastai repo的 `doc_source` 目录里的notebook自动生成的。如果你运行它们，你可以看到相同的猫的图片。Sylvain很喜欢猫，所以这个文档里有很多猫，他创建了很棒的文档，他选择了他喜欢的素材。

对不同的亮度值来说，我会检查两件事。一个是变形的程度，看下还能不能看出图片原来的内容。最左边这个有点变的不清楚了，最右边的也不清楚了。第二件事是，看下要建模的数据集里或者验证集里的图片，看下亮度的方差大概是多少。

[1:00:12]

如果这些图片都被拍得很专业，我希望它们的亮度都比较居中。如果这些照片是不太专业的人拍的，里面的一些会是过度曝光的，一些是曝光不足的。所以你需要选择一个合适的亮度增强的值，让图片保持清晰，让它还是原来的那件东西。



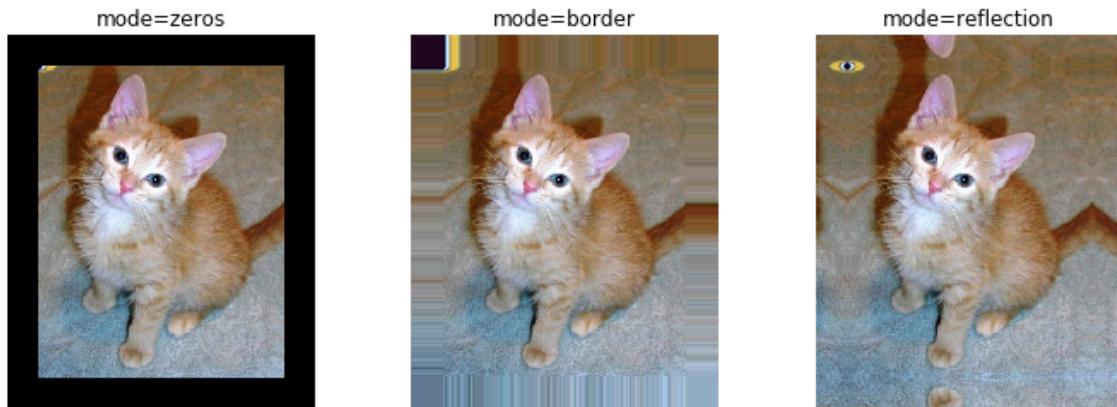
对于对比度，也是相同的。在一个数据集里，有这样荒唐的对比度并不常见，但你有可能会遇到，这时，你应该调整数据增强的取值范围。如果你没有遇到这样的数据，就不需要了。



这个 `dicedral` 是用来做所有可能的旋转和翻转。很明显，你的大部分图片不会是头朝下的猫。你可能会说“嘿，这没有意义，我不会在这个数据集里使用它”。如果你在处理卫星图片，你就会用它了。

另外，`flip` 非常有用，你应该使用它。

fastai里的很多方法，有一个padding（边距）模式的选项，这是padding模式的样子：



你可以选择zeros，你可以选择border，这只是简单的重复，你可以选择reflection，你可以看到，它用边上的一些像素对称填充了边框。**reflection通常是最好的**。我不知道有没有人研究过这个，但是我们在一定程度上研究过它。我们没有写论文，但是对证明reflection大多数时候是最好的已经足够了。所以我们把它作为默认值。

还有一组很酷的透视扭曲变形方法，我用对称扭曲（symmetric warp）演示下。



我们为图片添加了黑边，这样会看起来更明显。可以看到对称扭曲做的就像是把相机移动到物体的上边或者侧边，然后扭曲这个东西。比较酷的是，你可以看到，每张图片看起来就像是在不同的角度拍摄的，都是合理的。这是一种很棒的数据增强。我不知道有哪个其他的库做了这个功能，至少没有像 fastai 这样又快、又能保持图片的效果。如果你希望赢得一场 Kaggle 竞赛，这个东西会帮你超过那些不用 fastai 的人。

看完这个后，我们有一个 `get_data` 函数，它里面做了像以前一样的用 block API 的东西，我们这次显式添加了 padding 模式，这样，我们可以把 padding 模式设成 zeros，来看看哪个更好。

```
1 | src = ImageItemList.from_folder(path).random_split_by_pct(0.2, seed=2)
```

```
1 | def get_data(size, bs, padding_mode='reflection'):
2 |     return (src.label_from_re(r'([^\.]+)\.\d+.jpg$')
3 |             .transform(tfms, size=size, padding_mode=padding_mode)
4 |             .databunch(bs=bs).normalize(imagenet_stats))
```

```
1 | data = get_data(224, bs, 'zeros')
```

```
1 | def _plot(i,j,ax):
2 |     x,y = data.train_ds[3]
3 |     x.show(ax, y=y)
4 |
5 | plot_multi(_plot, 3, 3, figsize=(8,8))
```

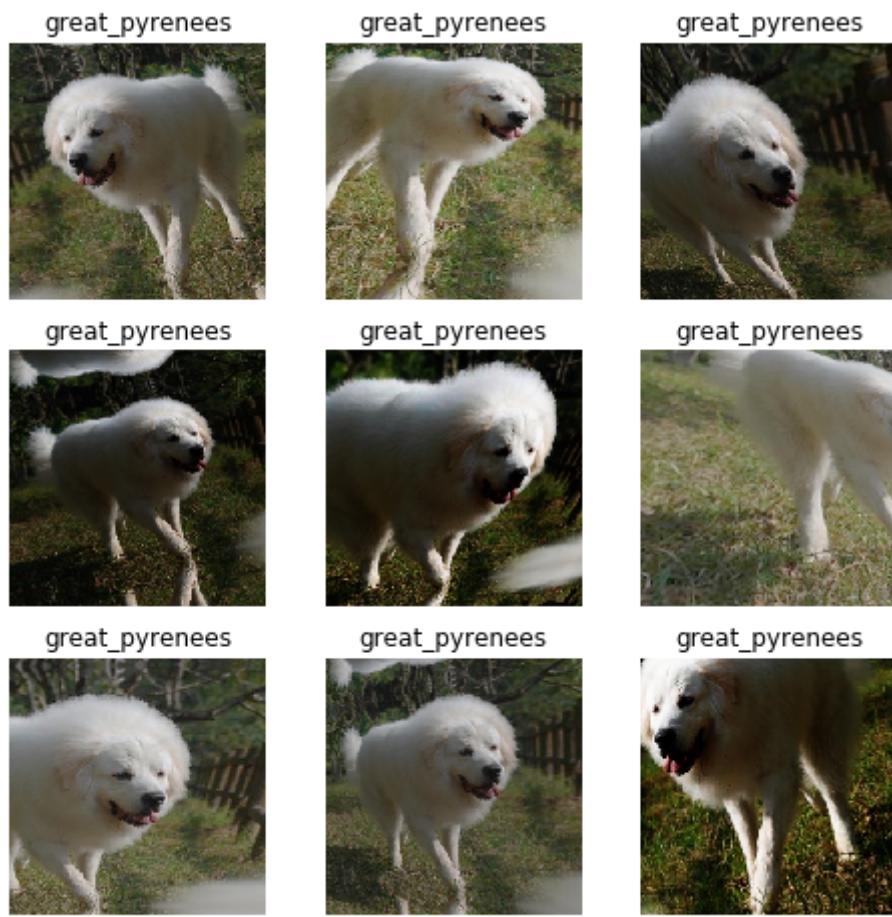


fastai里有个简短的 `plot_multi` 函数，它会创建一个 3×3 个图片网格，每个都存放了调用 `_plot` 的结果，这个函数的参数是坐标和轴。我会在每个格子里画相同的东西，但因为这是训练集，它会使用数据增强。你可以看到用了各种数据增强的同一个狗的照片。你可以看到为什么它会很有效，因为这些图片看起来很不同。但我们没有做任何手工标注或其它的事情。它们就像是额外赠送的数据。所以数据增强非常棒。

一个很大的研究机会是找出其它领域做数据增强的方法。怎样对文本数据、基因数据、医学数据等等做数据增强。几乎没有人在研究这个，对我来说，这是最大的机会之一，它能让你的数据需求减少到原来的1/5到1/10。

```
1 | data = get_data(224,bs)
```

```
1 | plot_multi(_plot, 3, 3, figsize=(8,8))
```



还是一样的，只是使用reflection padding替代zero padding。你可以看到在底部这个狗真的被反射了。reflection padding会生成更自然合理的图片。在真实世界里，你不会遇到黑边。它看起来更有效。

卷积神经网络 (Convolutional Neural Network) [1:05:14]

我们要学习卷积神经网络，我们先来创建一个卷积神经网络。你知道怎样创建它们，我先创建一个。我训练它一下，解冻它。然后，我要创建一个更大的 352×352 的数据集，再训练它，然后保存它。

```
1 | gc.collect()
2 | learn = create_cnn(data, models.resnet34, metrics=error_rate, bn_final=True)
```

```
1 | learn.fit_one_cycle(3, slice(1e-2), pct_start=0.8)
```

```
1 | Total time: 00:52
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      2.413196    1.091087    0.191475    (00:18)
4 | 2      1.397552    0.331309    0.081867    (00:17)
5 | 3      0.889401    0.269724    0.068336    (00:17)
```

```
1 | learn.unfreeze()
2 | learn.fit_one_cycle(2, max_lr=slice(1e-6,1e-3), pct_start=0.8)
```

```
1 | Total time: 00:44
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      0.695697    0.286645    0.064276    (00:22)
4 | 2      0.636241    0.295290    0.066982    (00:21)
```

```
1 | data = get_data(352,bs)
2 | learn.data = data
```

```
1 | learn.fit_one_cycle(2, max_lr=slice(1e-6,1e-4))
```

```
1 | Total time: 01:32
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      0.626780    0.264292    0.056834    (00:47)
4 | 2      0.585733    0.261575    0.048038    (00:45)
```

```
1 | learn.save('352')
```

我们得到了一个CNN。现在我们要弄明白这里面做了什么。要弄明白的方式就是学习怎样生成这个图片：



这是一个热度图 (heat map)。这个图片显示了当CNN判断这张图片是什么时，它关注图片里的哪个部分。我们会从头做这个热力图。

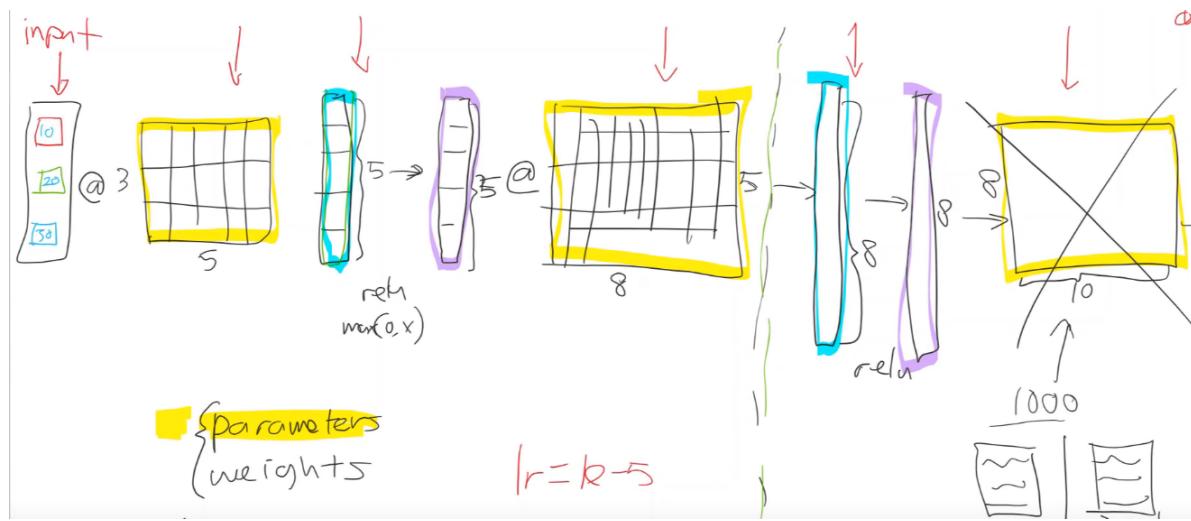
现在我们处在课程的这样一个阶段，我假设你们到达了这个阶段，并且还处在这个阶段，就是你们对深入这些细节很有兴趣，并且对深入它们已经做好了准备。我们要学习怎样创建这个热度图，不用 fastai 库里的东西。我们会用 PyTorch 里的纯粹的张量计算，我们用这些来真正理解它是怎样运行的。

提醒下，这不是火箭科学，但很多内容看起来很新，不要期待一次就能弄明白，而是要听课，进入 notebook，做些尝试，检查结果，查看张量形状和输入输出，检查你理解了没，然后回去再听一遍。试几次，你就会明白了。这里会有很多概念，因为我们没有在纯粹的 PyTorch 里做过这么多东西。

[1:07:32]

让我们开始学习卷积神经网络。有趣的是在课程快结束的时候才学习卷积，这是很少见的。知道 batch norm 怎样工作、dropout 怎样工作、卷积怎样工作，不如知道它们是怎样一起运行的、用它们能做什么、怎样能做得更好。现在我们处于这样一个阶段，我们想做热度图这样的东西。尽管我们把这个方法直接添加到了我们的库里，你可以运行一个函数来做。但是你做的越多，你越能找到一些做这些事情的不同的方法，在你的领域里可能有一些东西让你觉得“噢，我可以做这个的一个简单变种”。你会有更多的经验，这对让你知道怎样自己做更多的东西有帮助，这意味着你需要理解现象背后发生了什么。

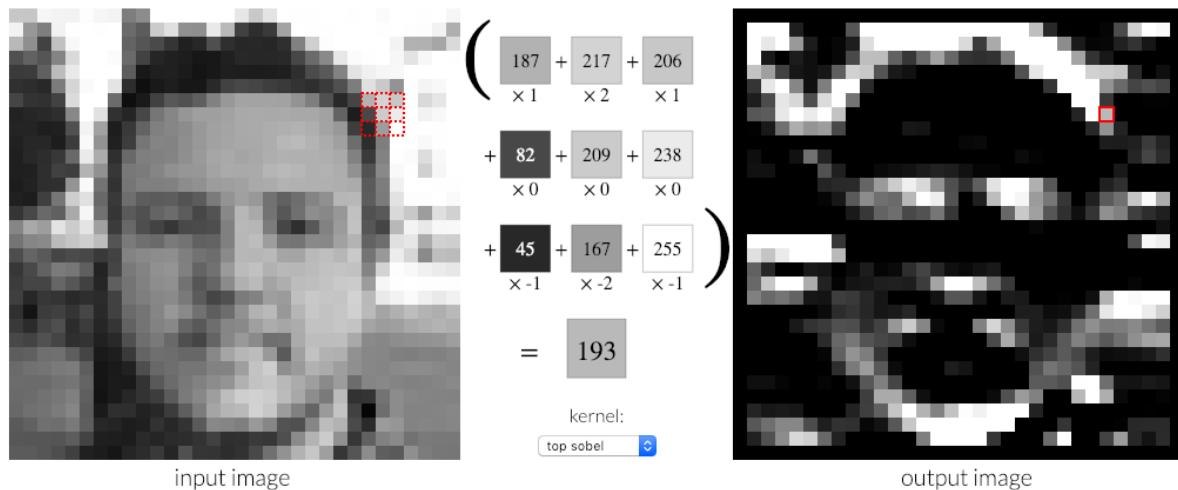
现象背后是我们在创建一个像这样的神经网络：



但是，不再是做矩阵乘法，我们会做一个卷积。卷积是一个有一些特殊性质的矩阵乘法。

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



你一定要访问下这个网站<http://setosa.io/ev/image-kernels/> (ev 代表可视化解释explain visually) , 我们借用这个美妙的动画, 它实际上是一个JavaScript程序, 你可以自己尝试, 来看看卷积是怎样工作的。当我们移动这个小红方框时, 它真的显示给你一个卷积。

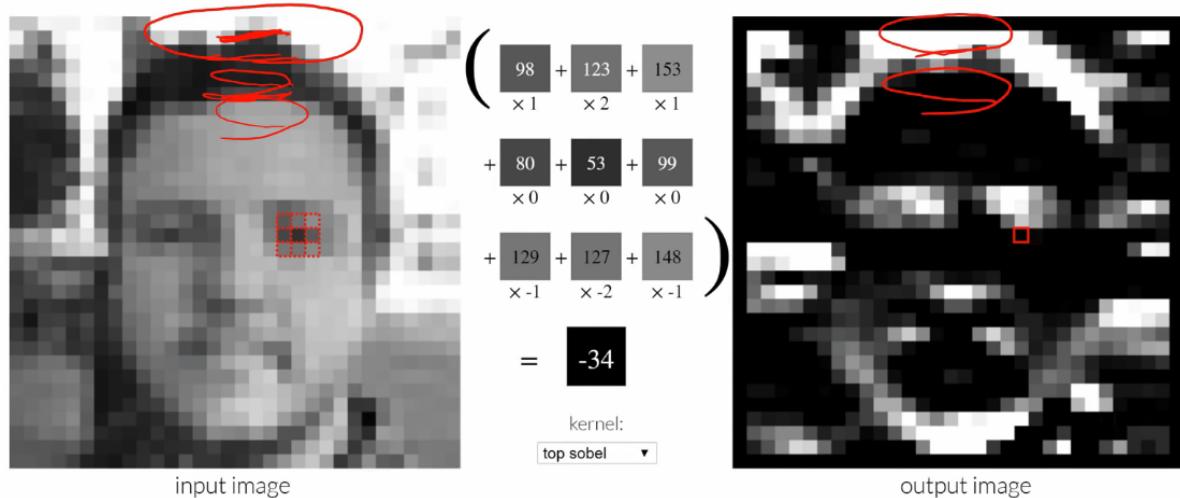
这是一个图片, 一个黑白灰的图片。3x3的红框移动时, 底部有个的3x3的矩阵。它显示了像素的值。在fastai里, 像素值是从0到1, 这里它们是从0到255。这是9个像素值(底部矩阵)。这个区域是白色的, 所以它们是很大的数字。

当我们移动时, 你可以看到这个9个比较大的数字变了, 你可以看到它们的颜色也变了。这里还有9个数字, 你可以看到这里的 $\times 1 \times 2 \times 1$ 。当我们移动这个红方框时, 这些数字在变化, 我们用对应的数字乘以它们。让我们学些新名称。

这里的东西, 我们叫kernel (核) , 卷积核。我们取图片上每个3x3的部分, 对鼠标悬浮的9个像素和卷积核里的9个数字做元素相乘。把每个组相乘, 然后我们可以把它们加起来。这就是右面显示的东西。随着左边红色的东西移动, 你可以看到右边出现一个红色的东西。右边有一个红色东西的原因是, 每9个像素, 在经过和kernel元素相乘, 再相加后, 生成了一个输出。所以可以看到, 右边图片每条边都

比原来的少一个像素。看到这里的黑边了吗？这是因为在边上，这个 3×3 的核，不能走得更远了。能走到的最远的地方是中间的格走到角落旁边这个位置。

为什么我们做这个？可能你能看出发生了什么。这个脸变成了用白色画出水平的边。怎样做到的，就是用这个把9个像素和这个核做元素相乘，把它们加起来，把对应的点的结果画在这。为什么在水平的边上生成了白点？我们来想一下。看这里（头的顶部）：

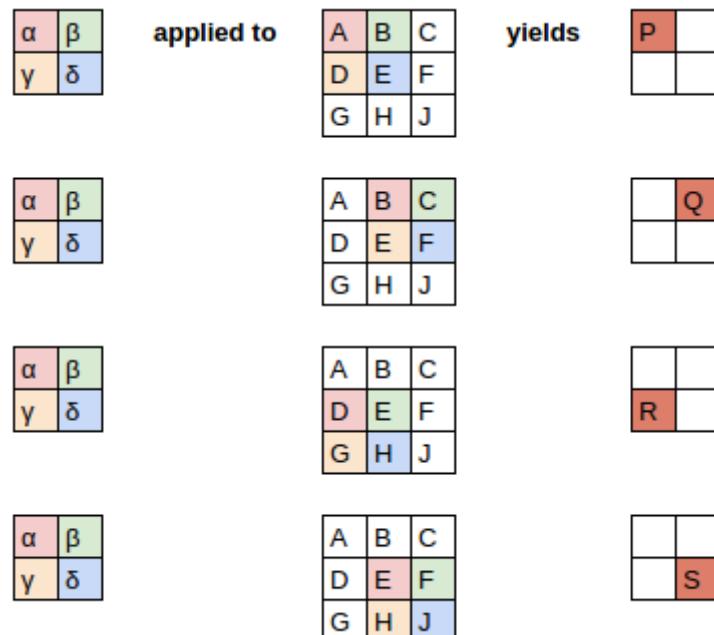


如果我们只在这一小块，它上面的点都是白色，它们有很高的值。这些上面的数乘以了(1 2 1)。所以这会生成一个大数字，中间的数都是0，不用管它们。然后下面的数都是小数字，因为它们接近0，所以不会有太大影响。所以这一小块最终是明亮的白色。另一方面，下面这里（发际线），下面是浅色像素，就会得到很多负值，上面的深色像素很小，没什么影响。因此，我们会得到比较大的负值。

我们取每个 3×3 的区域，用核做元素相乘，把它们加起来，生成一个输出，这叫一次卷积。就是这样，这就是卷积。这对你可能看起来很熟悉，因为不久前我们看了Zeiler和Fergus的论文，看到了每个不同的层，我们可视化了权重在做什么。记得第一层怎样找到对角线和斜线吗？这是卷积可以做到的事情。我们的每一个层是一个卷积。第一层只能做这样的事情（找到顶部边）。但是好消息是，下一层可以得到这样的结果，它可以组合一个通道（channel，一个卷积过滤器的输出被叫做一个channel），它可以组合一个能找到顶部边的channel和一个能找到左侧边的通道，然后上面的层可以把这两个作为输入，生成能找到左上角的东西，就像我们在Zeiler和Fergus的文章里看到的那样。

[1:15:02]

让我们再从其它角度看下这个。我们看下这一篇[Matt Kleinsmith的精彩文章](#)，他是我们这个课程第一年的学生。他写了这个作为他的课程项目的一部分。



这里是我们的图片。它是一个 3×3 的图片（中间列），我们的核是一个 2×2 的矩阵（左边列）。我们要做的是把这个核应用在这个普通的左上角 2×2 的部分。这一部分和这部分对应地相乘，绿色对应绿色，红色对应红色。把它们加在一起，生成输出里的左上角的值。也就是说：

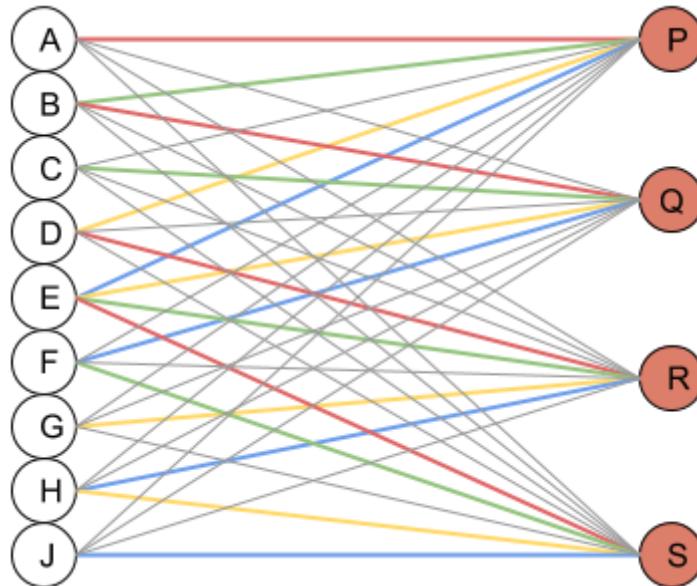
$$\alpha A + \beta B + \gamma D + \delta E + b = P$$

$$\alpha B + \beta C + \gamma E + \delta F + b = Q$$

$$\alpha D + \beta E + \gamma G + \delta H + b = R$$

$$\alpha E + \beta F + \gamma H + \delta J + b = S$$

b 是一个bias，这没什么问题。它只是一个普通的bias。你可以看到每个输出像素是不同线性等式的结果。可以看到这相同的四个权重被移动了，因为它们是我们的卷积核。



这是另外一种看它的方式，这是一个传统的神经网络视图。现在P是每个这些输入和权重的乘积的和，这些灰色的值会是0。因为，要记得P只连接了A、B、D和E。换句话说，这代表一个矩阵乘法，所以我们可以用一个矩阵乘法表示它。

α	β	0	γ	δ	0	0	0	0
0	α	β	0	γ	δ	0	0	0
0	0	0	α	β	0	γ	δ	0
0	0	0	0	α	β	0	γ	δ

$$* \begin{matrix} \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \\ \text{E} \\ \text{F} \\ \text{G} \\ \text{H} \\ \text{J} \end{matrix}$$

$$\begin{matrix} \text{b} \\ \text{b} \\ \text{b} \\ \text{b} \end{matrix} = \begin{matrix} \alpha A + \beta B + 0C + \gamma D + \delta E + 0F + 0G + 0H + 0J + b \\ 0A + \alpha B + \beta C + 0D + \gamma E + \delta F + 0G + 0H + 0J + b \\ 0A + 0B + 0C + \alpha D + \beta E + 0F + \gamma G + \delta H + 0J + b \\ 0A + 0B + 0C + 0D + \alpha E + \beta F + 0G + \gamma H + \delta J + b \end{matrix} = \begin{matrix} \alpha A + \beta B + \gamma D + \delta E + b \\ \alpha B + \beta C + \gamma E + \delta F + b \\ \alpha D + \beta E + \gamma G + \delta H + b \\ \alpha E + \beta F + \gamma H + \delta J + b \end{matrix} = \begin{matrix} P \\ Q \\ R \\ S \end{matrix}$$

这是我们 3×3 的图片拉伸后放到一个向量后的像素列表，这是一个矩阵和向量相乘加上bias。然后，所有这些，我们要设成0。所以你可以看到，我们得到了：

$$\alpha \quad \beta \quad 0 \quad \gamma \quad \delta \quad 0 \quad 0 \quad 0 \quad 0$$

它和下面这个对应

A	B	C
D	E	F
G	H	J

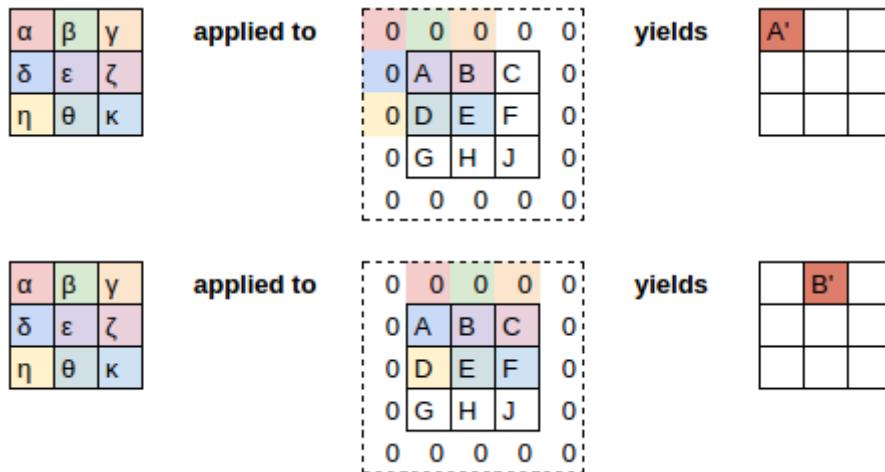
换句话说，卷积只是一个矩阵乘法，有两个特性：

- 有一些元素的值被一直设成了0

- 所有这些有相同颜色的元素，一直有相同的权重值

当你得到用相同权重乘的东西时，这叫做**weight tying (权重捆绑)**

很明显，我们可以用矩阵乘法实现一个卷积，但我们不这样做，因为它太慢了。所以，在实践中，我们的库有专门的卷积函数。它们基本上就是做[这个](#)，也就是[这个](#)，也就是[这个等式](#)，也就是和[这个矩阵乘法](#)相同。



像我们讨论过的，我们需要考虑padding，因为如果你有一个 3×3 的核，和一个 3×3 的图片，这只能生成一个像素的输出。这个 3×3 只有一个地方可以去。如果我们要生成一个多于1个像素的输出。我们需要用这个叫padding的东西，就是在外面添加一圈额外的数字。大多数库做的只是添加一圈0。所以对于 3×3 的核，每个边都是0。当你这样做padding时，你可以用 3×3 的核遍历所有像素，得到和你原来图片相同尺寸的输出。

我们讲过，在fastai里，我们通常不用zero padding。如果可行，我们会尽量用reflection padding，不过对这些简单的卷积，我们经常用zero padding，因为它在大图片里影响不大。不会有太大差别。

这就是卷积。一个卷积神经网络如果只能生成顶部的边时不是很有意义，所以我们要做更多。

如果我们有一个输入，它可能是一个标准的红绿蓝图片。我们可以创建一个 3×3 的核，用这个核遍历所有不同的像素。但是想想看，我们的输入不再是2D的了，我们的输入是3D的（rank是3的张量）。我们可能不希望对红绿蓝都用相同的核，比如，如果我们在创建一个绿色青蛙识别程序，我们想让绿色比蓝色有更高的激活值。如果我们想找到一个可以识别从绿色到蓝色的变化的东西，对不同channel的核需要有不同的值。所以，我们需要创建一个 $3 \times 3 \times 3$ 的核。这还是我们的核，它可以在行和列上移动核。不再是做9个数的元素乘法，我们会做27个数的元素乘法，我们还会把它们加成一个数。当我们在这个上面移动这个立方体时，会完全对应。当我们做这部分卷积时，它还是会创建一个数字，因为我们是做所有27个数字的元素乘法，把它们加在一起。

我们可以遍历这个添加了一个像素padding的输入。我们开始时是 5×5 ，我们最终的输出也是 5×5 。但现在我们的输入是3通道的，我们的输出是1通道的。只用一个通道我们不能做很多，因为我们现在已经做的只是找到顶部的边。我们要怎样找到一个侧边？和一个斜边？一个白色区域？我们要创建另外一个核，我们在上面再做卷积，我们会生成另外两个 5×5 。然后，我们可以把它们叠在一起，作为另外一个轴（axis）遍历它，我们可以做很多很多次，这会给我们输出一个秩（rank）是3的张量。

这就是它实际做的事情。实践中，我们开始时有一个 $H \times W \times 3$ 的输入。我们让它通过一批卷积核，我们设置我们需要的数量，它会给我们返回一个输出，它的尺寸是height \times width \times 核的数量。通常在第一层，它会用16。现在我们有了16个通道，代表在这个像素有多少个左侧边、多少个顶部边、在这个9个RGB像素的集合上有多少个由蓝到红的斜线。

然后你可以再做相同的事情。你可以用另外一批核，这会生成另外两个秩是3的张量（尺寸还是height, weight, 核的数量（可能还是16））。现在，随着我们进到网络比较深的地方，我们想要越来越多的通道。我们希望能找到越来越多的特征，就像在Zeiler和Fergus的论文里看到的那样，所以你确实需要很多通道。

为了避免内存超出控制，我们有时用这样的卷积，它不对每个 3×3 的集合做处理，每次会跳过来两个像素。开始时这个 3×3 的矩阵的中心在(2, 2)，然后我们跳到(2, 4)、(2, 6)、(2, 8)等等。这叫做**stride 2 convolution**（步长是2的卷积），它做的事，看起来还一样的，还是一组核，但是我们只是一次跳过了2个。我们跳过了每个可选的输入像素，所以，对应的输出的尺寸会变成 $H/2$ 和 $W/2$ 。当我们这样做时，我们通常创建两倍的核，所以我们在每个点有32个激活值。这就是现代卷积神经网络的样子。

[1:26:26]

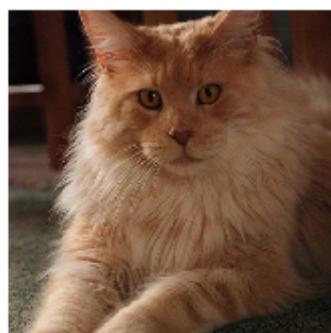
如果我们打开我们的pet notebook，可以看到这个。我们取到我们的CNN，我们看下这只猫：

```
1 | data = get_data(352, 16)
```

```
1 | learn = create_cnn(data, models.resnet34, metrics=error_rate,
1 | bn_final=True).load('352')
```

```
1 | idx=0
2 | x,y = data.valid_ds[idx]
3 | x.show()
4 | data.valid_ds.y[idx]
```

```
1 | Category Maine_Coon
```



如果我们运行 `x,y = data.valid_ds[idx]`，它会取第0个图片。我们运行 `.show()`，我们打印出 `y` 的值。看起来这只猫的品种是Maine Coon（缅因猫）。在一周以前，我都不知道有猫的品种叫Maine Coon。在花了一周时间在这只猫上后，我现在对这个缅因猫熟悉多了。

```
1 | learn.summary()
```

```
1 Input Size override by Learner.data.train_dl
2 Input Size passed in: 16
3
4 =====
5 Layer (type)          output Shape        Param #
6 =====
7 Conv2d                 [16, 64, 176, 176]    9408
8
9 BatchNorm2d            [16, 64, 176, 176]    128
10
11 ReLU                  [16, 64, 176, 176]    0
12
13 MaxPool2d              [16, 64, 88, 88]    0
14
15 Conv2d                 [16, 64, 88, 88]    36864
16
17 BatchNorm2d            [16, 64, 88, 88]    128
18
19 ReLU                  [16, 64, 88, 88]    0
20
21 ...
```

如果我运行 `learn.summary()`，记住，我们的输入是352x352像素的，一般第一个卷积的步长会是2。所以在经过第一层后，它是176x176。`learn.summary()` 会打印出每层的输出的形状。

第一个卷积集合有64个激活值。如果输入 `learn.model`，我们可以看到这个：

```
1 | learn.model
```

```
1 Sequential(
2   (0): Sequential(
3     (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
4       bias=False)
5     (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
6       track_running_stats=True)
7     (2): ReLU(inplace)
8     (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
9       ceil_mode=False)
10    (4): Sequential(
11      (0): BasicBlock(
12        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
13          (1, 1), bias=False)
14        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
15          track_running_stats=True)
```

```

11         (relu): ReLU(inplace)
12         (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
13             (1, 1), bias=False)
14             (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
15                 track_running_stats=True)
16     )
17     (1): BasicBlock(
18         (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
19             (1, 1), bias=False)
20             (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
21                 track_running_stats=True)
22         (relu): ReLU(inplace)
23         (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
24             (1, 1), bias=False)
25             (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
26                 track_running_stats=True)
27     )

```

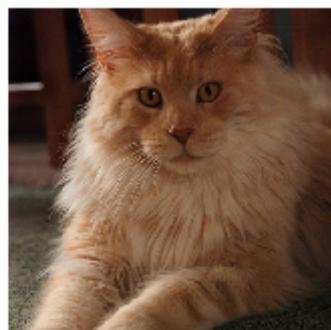
这里你可以看到它是一个2D的、有3个输入通道、64个输出通道的卷积、步长是2。值得注意的是，它在第一层用了一个 7×7 的核。几乎所有的卷积是 3×3 的。看到它们都是 3×3 了吗？我们经常在第一层使用比较大的核，原因会在课程的第二部分讲到。如果你用一个比较大的核，你需要用更多的padding，所以你要用核的尺寸除以2取整做padding，来做到不丢失像素。

我们现在有了64个输出通道，因为步长是2，它现在是 176×176 的。然后，随着我们继续，你可以看到过一段时间，尺寸会减半（尺寸从 88×88 到 44×44 ，这是一个2D卷积），然后我们这样做时，我们通常把通道数量加倍。

我们继续经过几个卷积层，可以看到，它们做了batch norm和ReLU，这很标准。最终，我们再做一次，又一个步长是2的卷积，通道数量加倍。我们现在的尺寸是 $512 \times 11 \times 11$ 。这里我们网络的主要部分就结束了。我们最终的尺寸是 $512 \times 11 \times 11$ 。

手工卷积 [1:29:24]

现在我们能开始做这个热度图了。让我们试着完成它。开始之前，我想演示下怎样自己手工做卷积，这很有趣。



```

1 k = tensor([
2     [0. , -5/3, 1],
3     [-5/3, -5/3, 1],
4     [1. , 1 , 1],
5 ]) .expand(1,3,3,3)/6

```

我们使用这缅因猫的图片，我已经创建了一个卷积核。可以看到它有一个右侧边和一个底部边，这两个边都是正数。在它的内部，都是负数。这样，我觉得它可以过滤出右下方的边。这就是我的张量。

一个 3×3 的核不能实现这个目标，我需要再多两个维度。我需要3个维度来处理红绿蓝。所以我写了`.expand`，这是我的 3×3 的张量，我又在前面添加了一个3。`.expand`做的是，把原来的张量复制三次，来创建一个 $3 \times 3 \times 3$ 的张量。其实它并没有真的复制，它假装复制了，但其实只是引用了相同的内存位置，这是一种高效的内存复制方式。现在，这里是前面张量的3个拷贝：

```
1 | k  
  
1 | tensor([[[[ 0.0000, -0.2778,  0.1667],  
2 |           [-0.2778, -0.2778,  0.1667],  
3 |           [ 0.1667,  0.1667,  0.1667]],  
4 |  
5 |           [[ 0.0000, -0.2778,  0.1667],  
6 |           [-0.2778, -0.2778,  0.1667],  
7 |           [ 0.1667,  0.1667,  0.1667]],  
8 |  
9 |           [[ 0.0000, -0.2778,  0.1667],  
10 |          [-0.2778, -0.2778,  0.1667],  
11 |          [ 0.1667,  0.1667,  0.1667]]])
```

这样做的原因是我想用这个核用相同的方式处理红绿蓝。然后我们需要再多一个轴，因为我们实际上不是用一个像刚刚打印出来的那样的独立的核，我们是要用一个秩是4的张量。最前面的维度，用来区分每个不同的核。

[1:31:30]

这个例子里，我只会创建一个核。要做一个卷积，我还要把这个维度放在前面。现在你可以看到`k.shape`是`[1, 3, 3, 3]`：

```
1 | k.shape
```

```
1 | torch.size([1, 3, 3, 3])
```

这是一个 3×3 的核。有3个这样的核。然后它们组成了一个核。要理解这些高维的向量会花一些时间，因为我们不习惯写4D的张量，但是这样想就好了，4D的张量就是一组叠在一起的3D的张量。

这就是我们的4D张量，然后你可以调用`conv2d`，传入一些图片，我要用的图片是验证集里的第一个。

```
1 | t = data.valid_ds[0][0].data; t.shape
```

```
1 | torch.size([3, 352, 352])
```

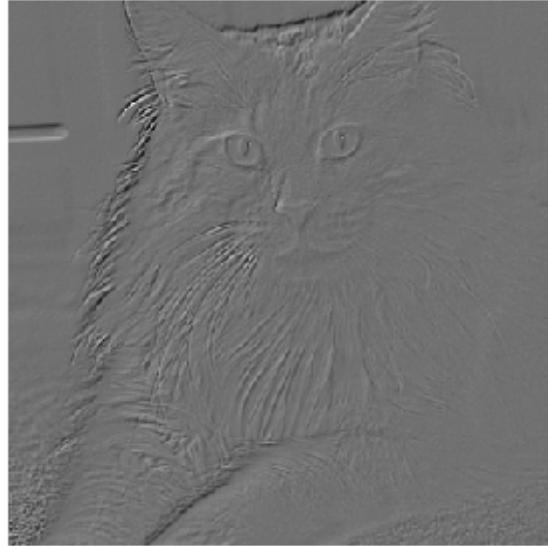
```
1 | t[None].shape
```

```
1 | torch.size([1, 3, 352, 352])
```

```
1 | edge = F.conv2d(t[None], k)
```

在PyTorch里，几乎所有东西都是处理一个mini-batch，不是一条单独的数据。所以，在这里，我们要创建一个大小是1的mini-batch。我们原始的图片是3通道乘以352x352（高x宽）。记住，PyTorch是通道x高x宽。我要创建一个秩是4的张量，第一个维度的值是1。换句话说，这是一个大小是1的mini batch，这是PyTorch想要的。在PyTorch和numpy里，你可以用一个特殊的值 `None` 对数组或张量做索引，这会创建一个新的值是1的轴。`t[None]` 是一个秩是4的张量，一个存放了一张图片的 $1 \times 3 \times 352 \times 352$ 的mini batch。

```
1 | show_image(edge[0], figsize=(5,5));
```



现在，你可以运行 `conv2d` 来得到一个猫，就是我们的缅因猫。你可以自己这样尝试下使用卷积。怎样生成热度图呢？

生成热度图 [1:33:50]

这里比较有趣。我有红绿蓝的输入，它经过一组卷积层来生成激活值，有了越来越多的通道和越来越小的宽和高。最终，记得我们在summary里看到的吗，我们最后得到了 $11 \times 11 \times 512$ 的张量。中间我们经过了很多层。

现在这里有37个类别，`data.c` 是类别的数量。我们在这个的最后可以看到，我们的模型最后有37个特征。这意味着我们最终有37个概率，对应37个猫和狗的品种，这就是我们最后需要的输出，我们拿它和one hot编码矩阵比较，里面代表缅因猫的位置的值是1。

我们需要把这个 $11 \times 11 \times 512$ 变成37。我们的实现方式是取这些 11×11 图片里每一个的平均数，我们只是取平均数。先取第一个图片，计算平均数，得到一个值。然后取这512个图中的第二个，算平均数，又得到一个值。我们对每一个图片这样做，得到一个长度是512的向量。

现在，我们要做的是让它乘以一个 512×37 的矩阵，这会给我们一个长度是37的输出。我们为每个图片取平均值的步骤叫做**平均值池化 (average pooling)**。

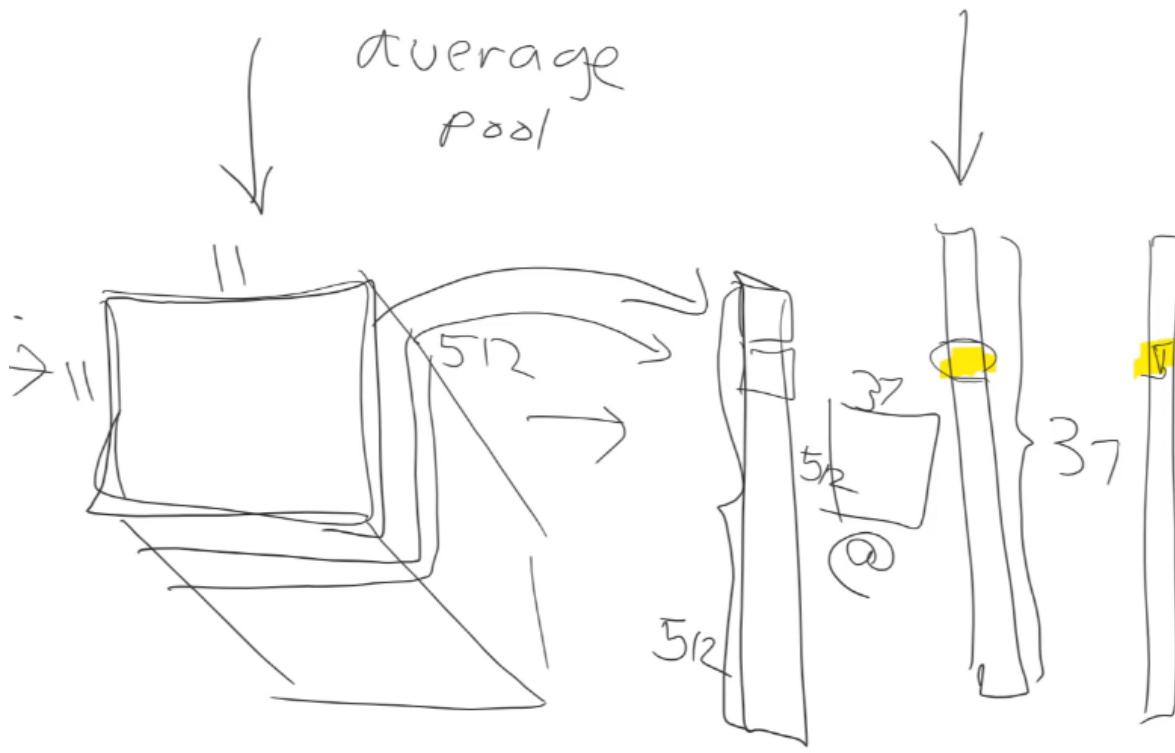
[1:36:52]

让我们回头看下我们的模型。

```
(2): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
)
(1): Sequential(
    (0): AdaptiveConcatPool2d(
        (ap): AdaptiveAvgPool2d(output_size=1)
        (mp): AdaptiveMaxPool2d(output_size=1)
    )
    (1): Lambda()
    (2): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.25)
    (4): Linear(in_features=1024, out_features=512, bias=True)
    (5): ReLU(inplace)
    (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): Dropout(p=0.5)
    (8): Linear(in_features=512, out_features=37, bias=True)
    (9): BatchNorm1d(37, eps=1e-05, momentum=0.01, affine=True, track_running_stats=True)
)
)
```

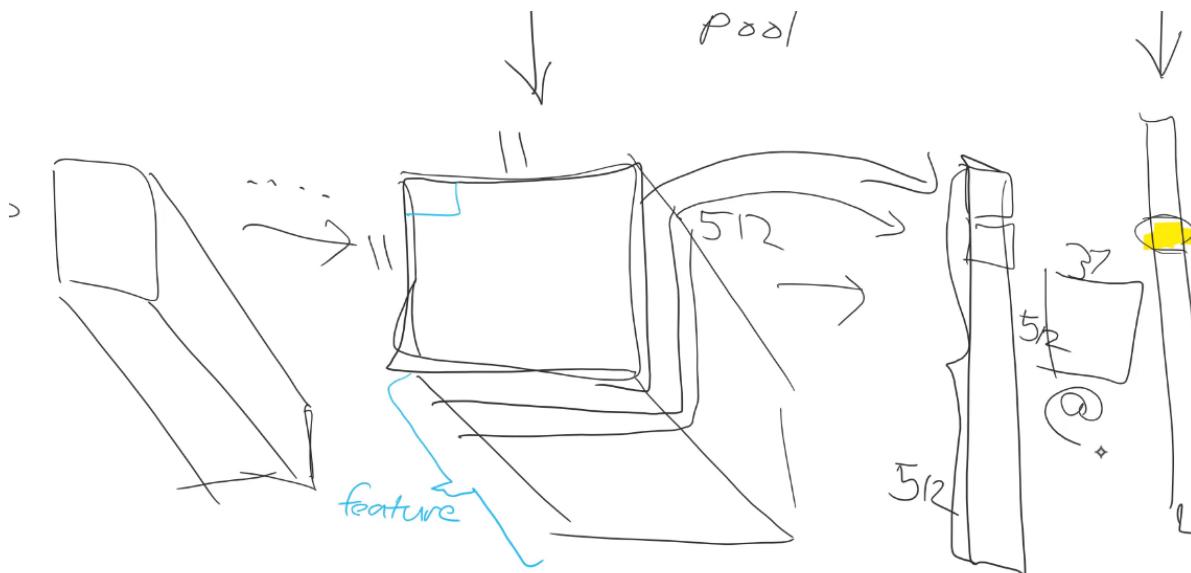
这是我们最后的512。我们会在课程第二部分讲什么是concat pooling。现在，我们只要知道它是 fastai里特有的东西。其他人都只使用这个 AdaptiveAvgPool2d，它的输出的尺寸是1。

这里有一些fastai里特有的东西，我们这里有两个层，通常只有一个Linear层，它的输入是512，输出是37。



这意味着，对于这个方框（输出层）我们希望得到代表缅因猫的1，我们得到了一个这里的方框（输出层前的一层），我们希望在这个位置有一个比较高的值，这样损失度会比较小。如果我们想让这里有一个高的值，唯一的方法就是用这个矩阵相乘，它代表这512个值的简单的权重线性组合。如果我们要能很确信地说这是一个缅因猫，只要取这些输入的加权求和，这些输入代表这样的特征，比如是不是毛茸茸的、鼻子是什么颜色、腿有多长、耳朵有多尖，所有这些可以用到的东西。要想识别出图片里是不是一个斗牛狗，也要使用这些一样的512个输入，只是权重不同。这只是一个矩阵相乘。只是一组加权求和，每个输出有不同的加权和。

所以，我们知道这些数十个甚至数百个卷积层最后输出 11×11 的平面，每个都对应这样一个特征：识别图片上有没有尖耳朵、是不是毛茸茸的、有没有红鼻子。这就是这些东西的含义。每个平面代表一个不同的特征。我们可以把这些输出当成是不同的特征。



我们感兴趣的不是对这 11×11 个点做平均，来得到这个输出的集合。我们想知道的是在这些 11×11 的点里有什么。如果不对 11×11 做平均，对这512个数做平均会怎样。如果我们将512做平均，我们会得到一个 11×11 的矩阵，每个点代表这个区域激活程度的评价值。也就是，当它认出这是一个缅因猫时，在这个 11×11 网格里的某一部分，有多少缅因猫的特征。

这就是我们想生成一个热度图需要做的，我想大概最简单的方式是倒着做。这是我们的热度图，它来自于叫做平均激活值的东西 (avg_acts average activations)。

```
1 | show_heatmap(avg_acts)
```



它里面包含一些matplotlib代码和fastai代码。

```
1 | def show_heatmap(hm):
2 |     _,ax = plt.subplots()
3 |     xb_im.show(ax)
4 |     ax.imshow(hm, alpha=0.6, extent=(0,352,352,0),
5 |               interpolation='bilinear', cmap='magma');
```

fastai方法用来显示图片，matplotlib方法接收热度图（heat map）参数，它传入时的名字是average activations（avg_acts，平均激活值）。`hm`代表heat map热读图。`alpha=0.6`让它变透明一些，matplotlib `extent`代表把它从11x11拓展到352x352。使用bilinear（双线性）的interpolation（插值）让它不要有斑点。使用不同的颜色（`cmap='magma'`）来提高亮度。这只是matplotlib里的东西，不怎么重要。

关键的地方是，平均激活值是11x11的矩阵，这是我们期望的。它是这样的。

```
1 | avg_acts = acts.mean(0)
2 | avg_acts.shape
```

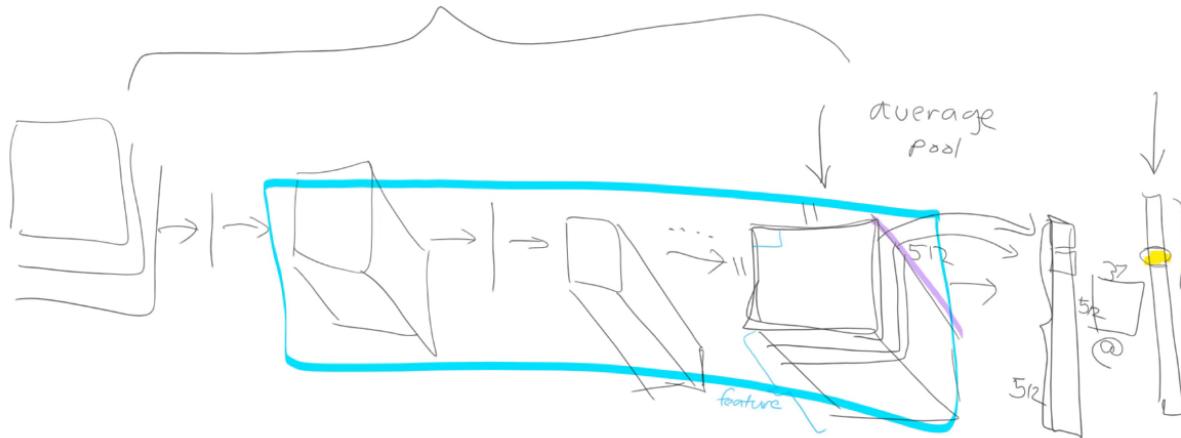
```
1 | torch.size([11, 11])
```

平均激活值的形状是11x11。要做到这个，我们取激活值在维度0上的平均数，在PyTorch里，通道维度是第一个维度，对维度0取平均值，可以把512x11x11的张量变成11x11的。所以激活值`acts`包含了我们要做平均的激活值。它们从哪来的？

```
1 | acts = hook_a.stored[0].cpu()
2 | acts.shape
```

```
1 | torch.size([512, 11, 11])
```

它们来自一个叫hook（钩子）的东西。hook是一个很酷的PyTorch高级特性，它让你能链接到PyTorch内部，运行任意的Python代码。这是一个令人惊奇的实用的功能。通常，当我们运行forward时，它计算出这个输出的集合。但我们知道，在这个过程里，它计算了这个（512x11x11的特征）。我要做的是，我想进入这个forward，告诉PyTorch“嘿，你计算这个时，能不能帮我把它保存下来”。什么是“这个”？这是模型的卷积部分的输出。卷积部分是所有平均池化层之前的东西：



回想下迁移学习，记得吗，使用迁移学习时，我们去掉所有模型卷积部分之后的东西，用新的层替代它。使用fastai，原来的模型的卷积部分会放到模型的最前面。具体讲，它在代码里是 `m[0]`。这里，我把我的模型命名为 `m`，你可以看到 `m` 里有很多东西：

```
In [34]: m = learn.model.eval();
In [35]: m
Out[35]: Sequential(
    (0): Sequential(
        (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace)
        (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (4): Sequential(
        (0): BasicBlock(
            (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (1): BasicBlock(
            (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        )
    )
)
```

通常（至少在fastai里），`m[0]` 会是模型的卷积部分。这里，我们创建了一个 `resnet34`，ResNet34（我们保存的预训练的部分）的主要部分在 `m[0]` 里。基本就是这样。这是打印出来的 ResNet34，它的最后，有512个激活值。

```
1 | m = learn.model.eval();
```

```
1 | xb,_ = data.one_item(x)
2 | xb_im = Image(data.denorm(xb)[0])
3 | xb = xb.cuda()
```

```
1 | from fastai.callbacks.hooks import *
```

```
1 | def hooked_backward(cat=y):
2 |     with hook_output(m[0]) as hook_a:
3 |         with hook_output(m[0], grad=True) as hook_g:
4 |             preds = m(xb)
5 |             preds[0,int(cat)].backward()
6 |     return hook_a, hook_g
```

```
1 | hook_a, hook_g = hooked_backward()
```

换句话说，我们想要取 `m[0]`，勾住它的输出：

```
1 | with hook_output(m[0]) as hook_a:
```

这是一个很有用的东西。fastai里为你创建了一些用来做这个的东西，你运行 `hook_output`，传入你希望勾住输出的那个PyTorch模块。你希望勾住的模块最有可能是模型的卷积部分，它通常是 `m[0]` 或者 `learn.model[0]`。

我们给这个hook命名 `hook_a`。不用管这部分 (`with hook_output(m[0], grad=True) as hook_g:`)。我们会在下周学习它。勾住了这个输出，现在我们要做forward前向传递。记住，在PyTorch里，要让它计算东西（做forward），你就把模型当成一个函数，传入我们的X mini batch。

我们已经有了一个叫 `x` 的缅因猫图片，但不能直接把它传入我们的模型。需要先标准化，转成mini batch，放到GPU。fastai的 `data` 对象里有一个创建data batch的东西，你可以用 `data.one_item(x)` 来创建一个只有一个数据的mini batch。

作为课后的练习，你可以试着创建一个mini batch，不用 `data.one_item`，来学习怎样自己做标准化和其它东西。用这个，你可以创建一个只有一个数据在里面的mini batch。然后，我可以用 `.cuda()` 把它放到GPU上。这是我传到模型上的东西。

我不太关心得到的预测值，因为预测值是这些东西（长度是37的向量）不是我想要的。我不会用预测值去做什么。我关心的是我刚刚创建的hook。

现在，一件要知道的事情是，如果你在PyTorch里勾住一些东西，每次你运行这个模型（假设你在勾住输出），它都会存下这些输出。当你得到你想要的东西后，你要把hook删除掉，不然的话，你再使用这个模型时，它会勾住越来越多的输出，这会消耗很多内存，会让程序变慢。我们创建了这个（Python把这叫做上下文管理器context manager），你可以用任何hook做上下文管理器，在 `with` 块的末尾，它会删除掉这个hook。

我们得到了我们的hook，现在fastai的hook（至少这个输出hook）会给你叫 `.stored` 的东西，它存储了你让它勾住的东西。现在激活值存放在里面。

1. 我们在勾住模型的卷积部分后，做了一个forward前向传递
2. 我们取出它存储的东西
3. 我们检查它的形状，它是 $512 \times 11 \times 11$ ，和我们预期的一样
4. 我们然后取通道这个轴的平均值，得到 11×11 的张量
5. 这就是我们想要的图片

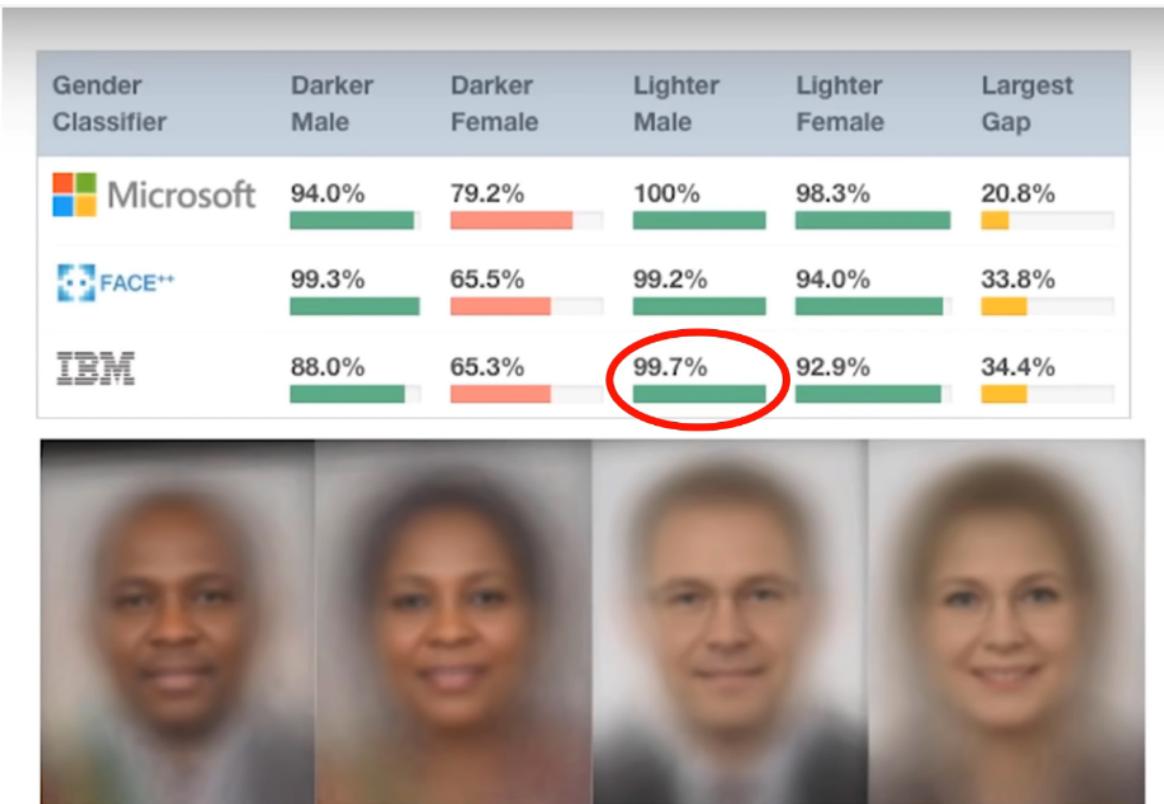
里面有很多内容。你可以运行下这两部分的代码，这个notebook里的**Convolution Kernel**部分和**Heatmap**部分，运行这些代码，改变其中一些，记住，最重要的是查看形状。你可能注意到了，当我演示这些notebook时，我总会打印出形状。当你查看形状时，你要看有多少个维度，想想为什么是有这么多维度。回过头来打印summary，看看实际的层的list，想想它们是怎么运行的。讲了很多的技术内容，现在我们不再讲技术内容，来讲一些更重要的事情。

伦理 (Ethics) 和数据科学 [1:49:10]

下节课，我们会学习生成模型（generative models），包括文本模型和图像生成模型。用生成模型，你可以创建一段新的文本或者新的图片、或者新的视频、或者新的声音。你们可能已经知道，这是过去12个月深度学习发展最快的领域。现在我们可以生成逼真的视频、图片、音频甚至文本。很多领域都涉及伦理问题，但生成模型可能是最大的一个。所以在开始学习它之前，我想专门讲下伦理和数据科学。。所以在开始学习它之前，我想专门讲下伦理和数据科学。

我演示的大部分东西实际上是Rachel做的，Rachel有一个很酷的 [TEDx San Francisco 演讲](#)，你可以在YouTube看到。这个演讲对AI的伦理原则和偏见原则的更广泛的分析，她有一个播放列表，你可以观看下。

我们已经接触过偏见的例子，这个性别预测研究里，男性白人在IBM计算机视觉系统的准确率是99.7%，女性黑人的准确率低了几百倍，惊人的差别。首先，要知道，这不是出现在单纯的技术研究上，这是在一个巨大公司的、有大量市场占有率的、有数百名质量控制人员研究过的、很多人在使用的产品上出现的。它已经在外部使用了。



Joy Buolamwini & Timnit Gebru

这看起来很疯狂，是吗？思考这个的原因很有意义。一个原因是提供给这些系统的数据。我们（包括我在内）倾向不加思考地使用很多数据集。像ImageNet是很多计算机视觉系统的基础，但是其中多半数据是来自英国和美国的。

No Classification without Representation: Assessing Geodiversity Issues in Open Data Sets for the Developing World

Shreya Shankar, Yoni Halpern, Eric Breck, James Atwood, Jimbo Wilson, D. Sculley

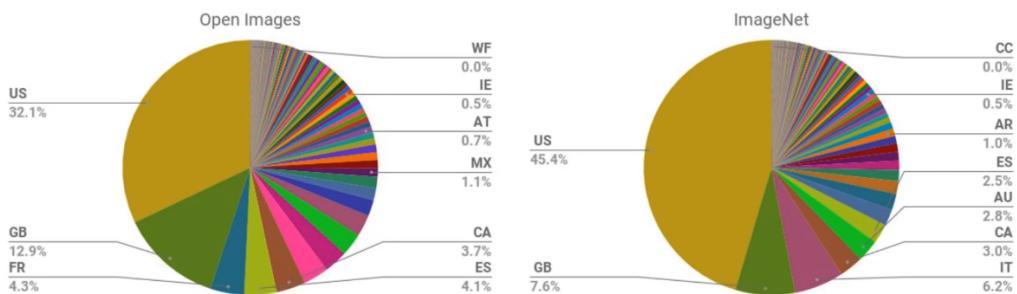


Figure 1: Fraction of Open Images and ImageNet images from each country. In both data sets, top represented locations include the US and Great Britain. Countries are represented by their two-letter ISO country codes.

当对于那些世界上有很多人口的国家，我们在这里看不到它们。它们在这些非常细的线里。因为，记住，这些数据集几乎全部是被美国、英国的人创建的，现在中国人创建的比例也在增长。所以在我创建的这些内容里有大量的偏见，因为创建内容的人有偏见，甚至在理论上，它是被用一种很中立的方式创建的。但是，你不能和这些数据争辩。这显然一点也不中立。



当你用带偏见的数据创建带偏见的算法时，你需要问“我们用它做什么？”“我们已经花了大量时间讲图像识别。几年前，这个公司DeepGlint做广告说他们的图像识别系统可以用来做人群监控，在经过的人中找出关注的人。先不管“用这样一个系统是不是一个好主意”，你应该想想“用这样一个对某类人的识别错误率是另外一种的300多倍的系统，是不是一个好主意？”

PALANTIR HAS SECRETLY BEEN USING NEW ORLEANS TO TEST ITS PREDICTIVE POLICING TECHNOLOGY

Palantir deployed a predictive policing system in New Orleans that even city council members don't know about

By Ali Winston | Feb 27, 2018, 3:25pm EST



Amazon is selling police departments a real-time facial recognition system

New documents obtained by the ACLU shed light on Amazon's Rekognition project

By Russell Bandom | @russellbandom | May 22, 2018, 11:06am EDT

Amazon's Face Recognition Falsely Matched 28 Members of Congress With Mugshots

 By Jacob Snow, Technology & Civil Liberties Attorney, ACLU of Northern California
JULY 26, 2018 | 8:00 AM

然后想下这个，这是美国开始发生的事，这些系统被售出了。现在在美国有了这样可以在视频里识别关注的人、通知当地警察的系统。这些系统是非常不准确的、有非常大偏见的。如果你在一个黑人为主的街区，识别的准确率会低很多，你可能被黑人包围，突然所有的黑人被认为是受关注的人。或者，在一个有受关注的人的视频里，所有人都会被认为在受关注的人的附近。突然出现在警察局的这些提示会让警察赶到这里。因此可能会导致更多的逮捕，这会被反馈到系统里来提升它。

现在这些正在发生。好在，有少数人在关注这些。我是说，非常少，但至少好过没有。一个最好的让人们了解的方法是做“有趣的”实验，比如让我们测试那些广泛使用的疑犯图像识别系统，用议会议员测试它，发现有28个黑人议员被这个系统识别出来了（这是明显错误的）。

The screenshot shows two pairs of English-Turkish-Spanish translations. In the first pair, "She is a doctor. He is a nurse." translates to "O bir doktor. O bir hemşire." In the second pair, "O bir doktor. O bir hemşire" translates back to "He is a doctor. She is a nurse." This demonstrates that Google Translate preserves the gendered language it receives.



在很多我们使用的系统中可以看到这样的偏见，不仅仅是图像识别，还有文本翻译。当你把"She is a doctor. He is a nurse"翻译成土耳其语，你得到了正确的性别无关代词，土耳其语里是这样使用的。你可以再取出它，用这样的性别无关代词填到土耳其语输入框里，你会得到"He is a doctor. She is a nurse."偏见又出现了。这是广泛使用的、被认真研发的系统。这不像其他问题（能一次解决的问题）一样，能被很快修复。谷歌翻译已经知道这个问题很久了，但问题还是存在，没有被修复。

这个的结果，在我看来，非常可怕。因为在很多国家，包括美国，我现在在的地方，算法被越来越多的用在各种公共政策、司法和其它目的中。

The article title is "Machine Bias". Below it, a subtitle reads: "There's software used across the country to predict future criminals. And it's biased against blacks." The ProPublica logo is in the top left corner, and a photo of a man is in the top right corner.

Prediction Fails Differently for Black Defendants

	WHITE	AFRICAN AMERICAN
Labeled Higher Risk, But Didn't Re-Offend	23.5%	44.9%
Labeled Lower Risk, Yet Did Re-Offend	47.7%	28.0%

比如，有一个叫COMPAS的系统，它被广泛用于判决。它用几种方法做这个：它告诉法官对特定的案件应该用什么量刑准则，它告诉他们哪些人应该被保释。问题是，它总是说让白人保释，尽管他们是再次犯罪。反过来也一样，它错误释放的白人是黑人的两倍。这很可怕，因为，用在这个的系统里数据是这样的，它只是问人们“你的父母有没有入过狱”或者“你有没有朋友吸毒”。它们问这些关于别人的、他们无法控制的问题。不仅这些系统有偏见，它们是在超出控制的有偏见的数据下被做出来的。“你的父母离婚了吗”也是一个用来决定你是不是应该入狱的问题。

当我们在Twitter上、访谈里或其他地方提出这些问题时，总会有些人（通常是男性白人）会说“这就是这个世界本来的样子”。“这只是反映了数据的内容”。但是如果你真正检查下，会发现不是这样。这实际上是系统性的错误。系统性地歧视有色人种、少数族裔、那些较少参与创建这些产品的基础系统的人。

有时，这会产生巨大的影响。在缅甸，曾经有一场对罗兴亚族（Rohingya）的大屠杀。这场屠杀很大程度上是被Facebook引发的。并不是Facebook的人希望发生这件事，我在Facebook有很多朋友。他们确实努力做正确的事。他们努力做一个人们喜欢的产品，但是他们的做法考虑不够周全。在缅甸，这个一半的人都没有电子设备的国家里，当你发布一些东西，说“嘿，你们可以免费上网，但只能上Facebook”的时候，必须慎重考虑你们做的事情。你用算法推送他们要点击的东西。当然，人们点击的内容是有争议的、会让他们变得愤怒的东西。当他们真的开始要求缅甸军方的将军把婴儿丢到篝火里时，他们说“我们知道，它们不是人，它们是动物，我们读了新闻，我们在网上读过”。因为这是算法推送的文章。算法在推送文章，算法很好。它们知道怎样吸引眼球，吸引人们来看，让人们点击。Facebook没有人说，让我们来在缅甸引导一场大屠杀。他们只是说，让我们把我们平台在这个新市场的用户活跃度最大化。



他们成功地最大化了活跃度。要知道，在2013、2014、2015，人们警告过Facebook的管理层，这个平台被用来煽动暴力。在2015，有人甚至警告过管理层，缅甸人会像卢旺达人在卢旺达大屠杀中使用广播那样使用Facebook。在2015，Facebook只有4个说缅甸语的外包员工。尽管他们接到非常非常严重的警告，他们没有在这个问题上投入太多资源。

Ethics is complicated

- Many ethical issues are complex and don't have clear or easy answers.
- I'm not here to tell you what to do or provide answers.
- It's good to think about how you'd handle a situation BEFORE you are in it.
- Volkswagen engineer sentenced to prison for creating software to cheat on emissions tests (he was following his boss's orders)

为什么会发生这种事？部分原因是，伦理是复杂的，你不会看到Rachel或者我告诉你们怎样做伦理工作，怎样解决它，我们不知道。我们只能告诉你一些事情，让你考虑它。另外一个原因是，我们经常听到，“这不是我的问题，我只是一个研究者，我只是做技术的，我只是在创建一个数据集，这个问题和我无关。我只是做了些基础设施，和这个问题远得很”。但是，如果你在创建ImageNet，你想让它成功，你希望很多人使用它，希望很多人用它做产品，很多人基于它做研究。如果你在努力创建人们使用的东西，你希望它们被使用，那么，请试着让它不会产生大量的危害，不要有大量的偏见。

它会反过来，让你付出代价。开发尾气排放检测作弊程序的大众汽车（Volkswagen）工程师最终进了监狱。并不是他们要在检测上作弊，是他们的管理者让他们这样做，他们写了这些代码。所以，他们最终要为犯罪负责，他们进了监狱。如果你在某种程度上，做了最终引发了事故的糟糕的事，这会最终影响到你，让你陷入困境。

In the concentration camps, IBM's code for Jews was 8. Its code for Gypsies was 12. General executions were coded as 4, death in the gas chambers as 6. Only Jews and Gypsies were systematically murdered in [gas chambers](#).

The Swiss judge ruled "It does not thus seem unreasonable to deduce that IBM's technical facilities facilitated the tasks of the Nazis in the commission of their crimes against humanity, acts also involving accountancy and classification by IBM machines and utilized in the concentration camps themselves."



"To the blind technocrat, the means were more important than the ends. The destruction of the Jewish people became even less important because the invigorating nature of IBM's technical achievement was only heightened by the fantastical profits to be made at a time when bread lines stretched across the world."

The Nazi Party: IBM & "Death's Calculator"

by Edwin Black

有时，这会引发严重的问题。如果你回到二战时期，对IBM来说，这是他们展示他们令人惊奇的表格计算系统的第一个巨大机会。他们有一个纳粹德国的大客户。纳粹德国用这个系统记录这个国家所有不同类型的犹太人和所有不同类型的有问题的人。犹太人有8个，吉普赛人有12个，然后不同的结果被计算出来，4个执行死刑，6个进毒气室。

一个瑞士法官裁定IBM积极参与推动了这些反人类罪行。绝对有很多人们用数据处理技术直接导致死亡的例子，有时是上百万的死亡。我们不希望成为其中的一个。你可能会想“噢，你知道，我只是在做一些数据处理软件”，有些人在想“我只是销售员”，有些人在想“我只是开发新市场的业务开发人员”。但这些会叠加在一起。我们需要防范。

Myth of neutral platforms



- Revenue model
- Design: do friends' posts, sponsored content, etc have same format
- Controls and filters available to users, available to advertisers
- Experiments
- Whether to have human moderators & how many
- Who gets banned

Facebook fires human editors, algorithm immediately posts fake news

Facebook makes its Trending feature fully automated, with mixed results.

ANNALEE NEWITZ - 8/29/2016, 11:20 AM

我们需要关心的一件事情是，让人类回到决策流程中来。当人类被从决策流程中排除时，就会发生问题。不知道你们记不记得，我记得很清楚，我第一次听到Facebook解雇新闻编辑（他们负责策划最终显示在页面上的新闻）的时候。那时，我认为这是会引发灾难的决策。因为，我看到过很多次，人类是这个过程中能认识到问题的部分。很难创建一个能识别出“这是不对的”的算法。另一方面，人类很擅长这个。我们看到发生了什么。在Facebook解雇掉人类编辑后，Facebook上的文章的性质戏剧性地改变了。你开始看到阴谋论在扩散，算法疯狂地推荐有争议的话题。当然，这改变了人们的偏好，他们想看更多的有争议的话题。

这里有一个很值得注意的地方，Cathy O'Neil（凯西·奥尼尔，他写了一半很棒的书《算法霸权（Weapons of Math Destruction）》）和其他很多人指出过。算法最终影响了人。比如，COMPAS量刑准则被法官使用。现在你可以说算法很好。我认为在COMPAS这里，它做得不好。它实际上最终就像随机判断一样差。因为这是一个黑盒一样的东西。即使它很好，你可能会说“好，法官是在使用算法的建议，不然，他们也会使用人的建议，人也会给出差的建议。这有什么差别吗？”人和算法是不一样的。法官很可能说“哦，就用电脑说的吧”，这很平常，尤其是那些对技术不是很熟悉的人。电脑查找答案给出这样的结果。

让一个不懂技术的人看到电脑建议后改变决策过程是极困难的。我们可以看到，算法经常被用到没有交互的流程里。他们经常被用到大规模的决策系统中，因为它们很便宜。然后使用算法的人过分信任它们，因为使用者经常是没有能力自己做出判断的人。

Algorithms are used differently than human decision makers:



- Algorithms are more likely to be implemented with **no appeals process** in place.
- Algorithms are often used **at scale**.
- Algorithmic systems are **cheap**.
- People are more likely to assume algorithms are **objective or error-free** (even if they're given the option of a human override)

The privileged are processed by people; the poor are processed by algorithms. (Cathy O'Neil)

有一个有人失去健康保障的例子。因为新算法的一个错误，算法没能识别出很多人在受脑瘫和糖尿病影响。这个错误后来才被发现，但病人已经错过应得的医疗护理。这些脑瘫患者的生活基本上毁了。

当做出了有错误的算法的人被问到这个问题，尤其是被问到他们是不是应该找出能综合考虑系统的长处和短处的、更好人机交互方式时，他们说“我该要打扫我的房间了”。这就是他们关注的东西。

这非常常见，我总会听到这个。在事后更容易发现这些问题，“在问题发生之后，我才能看到它，这真是一件糟糕的事。”在开发过程中，很难发现这个问题。

[Rachel]对这个例子，我想多说一些。这件事里有很多参与者。有一些人做算法，另外一些人做软件，现在它在50个州中超过一半州中使用，这些州实施这些政策。这种情况下，没人觉得是自己的责任，因为做算法的人想“噢不，是这些州的政策太糟糕。”这些州可以说“噢不，是那些做软件的人的错”，每个人都只是指责别人，不承担责任。

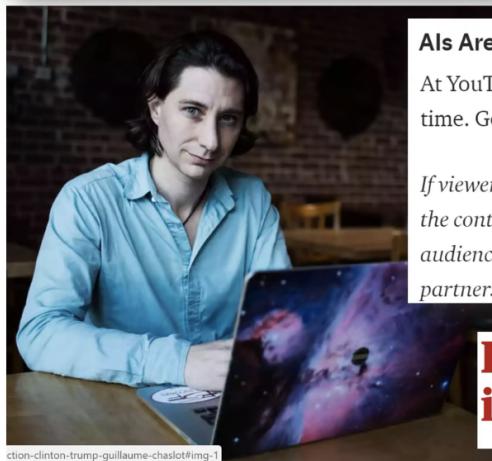
你知道，在某种意义上，这样不公平，但我可以说创建数据集的人和实现算法的人应该站出来，应该认真处理这些问题，保证他们在整个过程中的作用。

How Algorithms Can Learn to Discredit the Media

Guillaume Chaslot



Defamation is efficient, and AIs may have already figured it out



Als Are Designed to Maximize Watch Time

At YouTube, we used a complex AI to pursue a simple goal: maximize watch time. Google explains this focus in [the following statement](#):

If viewers are watching more YouTube, it signals to us that they're happier with the content they've found. It means that creators are attracting more engaged audiences. It also opens up more opportunities to generate revenue for our partners.

How an ex-YouTube insider investigated its secret algorithm

ction-clinton-trump-guillaume-chaslot#img-1

我们也在YouTube上遇到了这个。这和Facebook的事情类似，我们听过看fast.ai课程的学生说“嘿，Jeremy, Rachel，观看fast.ai课程让我们很开心，但在一个课程结束后，YouTube自动推荐了阴谋论的视频。”当系统认为你喜欢阴谋论时，它会为你推荐越来越多这样的东西。

[Rachel]简单讲。你甚至不需要喜欢阴谋论。它的目标是让尽可能多的人被阴谋论吸引，无论你有没有表现出兴趣。

还是一样，值得注意的是，我认识很多人参与做YouTube推荐系统的人。他们当中没有人想推荐阴谋论。但有人点击它们，分享它们，看过阴谋论的人会看更多的YouTube视频。所以，算法很擅长找到一个大量观看时长的主题，然后让这个主题的观看时长更多。

THE WALL STREET JOURNAL.

How YouTube Drives People to the Internet's Darkest Corners

Google's video site often recommends divisive or misleading material, despite recent changes designed to fix the problem

“YouTube may be the most powerful radicalizing instrument of the 21st century.” – Zeynep Tufekci, The New York Times

How an ex-YouTube insider investigated its secret algorithm



这是反馈循环的一个例子。纽约时报说，YouTube可能是21世纪最强大的激化工具。我在YouTube推荐系统部门工作的朋友不认为他们在建造21世纪最强大的激化工具。说实话，他们中大多数人都认为他们不是。他们认为这是胡说。不是所有人，但大多数人现在认为他们是被冤枉的，人们错怪了他们。你知道，他们错了，他们不理解我们要做的。当你处在其中时，要想做出正确的判断非常非常难。

所以，你需要在开始时，就做正确的判断。你正在从事的工作可能导致的不是你们本意的后果是什么？作为参与其中的技术人员，怎样可以事先站出来让人们意识到这个问题？

[Rachel]很多这些阴谋论在宣扬白人至上，这是极右的种族主义，是反科学的，大概5年10年前，我认为阴谋论是很边缘的东西，但是，我们现在看到相信它的人的巨大社会影响。

你总是在YouTube上看到它，就会觉得它很平常。人们在努力解决这个问题，一个方法是，和那些可能影响决策过程的人沟通。

Talk to domain experts & those impacted.



FAT* Conference 2019 ▾ 2018 ▾ Organization Resources

Tutorials - Translating to Computer Science - Vanderbilt Hall 210

FAT Breakout Room_210

The video player shows a flowchart on the left and a video of three speakers on the right. The flowchart details the process from being stopped by NYPD to trial, defense, or plea. The video shows Kristian Lum, Elizabeth Bender, and Terrence Wilkerson speaking at a conference.

▶ 34:13 / 2:06:44

Kristian Lum,
Elizabeth Bender, &
Terrence Wilkerson

比如，最近有一个很酷的事，统计学家和数据科学家和曾在司法系统里的人走到一起，和曾经共事过的律师谈话，让他们讨论，系统是怎样工作的，输入是哪里来的，人们怎样处理它们，谁会参与其中。

这很酷，这是让你作为一个数据产品开发者能真正知道你的产品将会怎样工作的唯一方式。

Choosing NOT to just maximize a metric



The video player shows a presentation slide with the title "When Recommendation Systems Go Bad" in red. Below the title, it says "Evan Estola 5/20/16". The video player interface includes a play button, a progress bar at 0:06 / 23:39, and various control icons.

Evan Estola - When Recommendations Systems Go Bad - MLconf SEA 2016

Evan Estola做了很棒的事，他在一个Meetup上说“嘿，很多人来参加我们的技术活动，如果我们用一个天真的推荐系统，它会推荐更多的技术聚会给男性，这会让更多的男性参加，当女性想参加时，她们会想‘天呀，有这么多男性在这’，这会让更多的男性参加技术聚会。只显示推荐给男性，不显示给女性”

Evan和朋友们认为这明显不代表人们真实的愿望。它会形成一个动态的反馈循环。所以，让我们在这发生之前阻止它，不再推荐更少的技术聚会给女性，更多的技术聚会给男性。我认为这很酷，这就像在说，我们不要成为算法的奴隶。我们要自己做决定。

The Nut Behind the Wheel



99% INVISIBLE



Datasheets for Datasets*

Timnit Gebru¹, Jamie Morgenstern², Briana Vecchione³,
Jennifer Wortman Vaughan¹, Hanna Wallach¹,
Hal Daumé III^{1,4}, and Kate Crawford^{1,5}

Early cars:

- sharp metal knobs on dashboard that could lodge in people's skulls in crash
- non-collapsible steering columns would frequently impale drivers
- belief that cars were dangerous because of **the people** driving them

人们可以做的另外一件事是立法监管。通常，提到监管，有人的反应是“怎样监管这些东西，这很荒唐，你没法监管AI”但当你认真研究下这个精彩的论文[Datasheets for Datasets](#)，里面有很多这样的例子。有很多工业界的例子，人们本来认为这不能被监管，人们认为它们本来就是这样。比如汽车。因为仪表盘上有尖锐的金属按钮、操纵杆不能折叠，总有人会在汽车上死亡，在社区里，人们的言论是“汽车就是这样”，人们在车上死亡，这是人的问题。但是最终监管出台。现在驾车变得非常安全，比以前安全很多很多倍。所以，我们可以在政策上有所作为。

Let's try to make the world a *better* place.



Only 0.3-0.5% of the global population knows how to code.



最后，世界上有0.3%到0.5%的人懂编程，我们是其中之一。我们有只有少数人才有的技能。不仅如此，我们还懂得这些深度学习算法，这是我所知道的最强大的代码。我希望我们可以认真地思考，至少不要让算法使世界变糟，或许能让它变好。



You know about problems no one else knows about.

为什么这对你们听讲的人有意义？这是因为fast.ai致力于让领域专家能更容易地使用深度学习。这张山羊的图片是我们以前课程的学员拍的，他是奶羊场主，他告诉我们，他们要在遥远的加拿大岛，使用深度学习帮助研究山羊乳房疾病。对我来说，这是一个领域专业问题的很好的例子，没有其他人懂这个问题，更不要说知道这是一个可以用深度学习解决的计算机视觉问题。不管你在什么领域，在你的领域里，你可能知道很多机会，能让它比以前好很多。你可能能提出各种很酷的产品设想，可能能创立一家公司，或者在你的公司里创建一个新的产品项目，等等。但是，请考虑下这在实践中会出现什么，想下在这个过程里，把人类放在什么位置、在哪里释放压力、你可以和谁探讨、谁会被影响、谁可以帮助你理解问题，让相关的人帮助你懂得历史、心理、社会等方面的影响。

这是我们的请求。你已经学了这么多，肯定已经做好了准备，能为世界做出深刻影响，希望我们能让这是积极的影响。下周见！