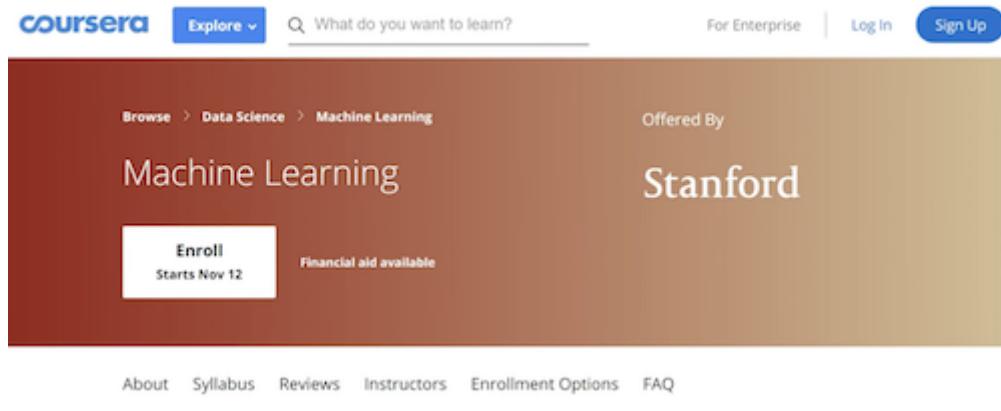


Lesson 3

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

更正一个引用。这个图最初来自Coursera上吴恩达的优秀的机器学习课程。抱歉引用搞错了。



About this Course

★★★★★ 4.9 85,364 ratings • 21,950 reviews

Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. Machine learning is so pervasive today that you probably use it dozens of times a day without knowing it. Many

[SHOW ALL](#)

Coursera上[吴恩达的机器学习课程](#)是很棒的。某些方面，它有点过时，但是很多概念是永远不会改变的。它是自下而上来讲解的。把它和我们的自上而下的方式结合起来是很好的，它们在某些地方是互补的。

如果你对机器学习基础感兴趣，你应该学习我们的[机器学习课程](#)。它大概比这个深度学习课程长一倍，它会带你更渐进地了解一些基础概念，比如验证集、模型解释、PyTorch工作原理等等。这些课程是一起的，如果你想深入学习，就学习两门课。我认识很多这样的人，他们最后说：作为一个整体来学两门课比单独学一个学到的更多。或者你可以把两门课都尝试下，看哪个更适合你。

上周我们讲过部署你的web应用。<https://course-v3.fast.ai/> 有一个“production”章节，里面有展示怎样简单地部署web应用的内容。举个例子，看看多简单，这是Navjot写的[怎样在Zeit部署的指南](#)。

Getting started ▾

Server setup ▾

Returning to work ▾

Production ▾

Zeil

fastai v1 ▾

Deploying on Zeit

Table of Contents

- One-time setup
 - Install Now's CLI
 - Grab starter pack for model deployment
- Per-project setup
 - Upload your trained model file
 - Customize the app for your model
 - Deploy
 - Scaling
 - Test the URL of your working app
- Local testing

Now – Global Serverless Deployments

就像你看到的，只有一页。几乎没有什么东西，并且都是免费的。它不能同时响应1万个并发请求。但是它可以让你顺利起步，我觉得它工作得很好、很快。部署一个模型，不再是一件很麻烦复杂的事。它的好处是，你可以用它做一个最小可用产品。如果你真的发现，它会接收到1000个并发，这时你就不能再继续用它了，你需要升级一下你的主机类型或者增加一些传统的大型工程方法。你可以使用这个起步工具箱创建泰迪熊识别器，这个模版非常简单。你可以使用你自己的样式文件、你自己的定制逻辑和它的的东西。它被设计得非常简单，这样你可以看到它是怎样运行的。后端是一个简单的REST风格接口，它返回JSON，前端是一个超级简单的JavaScript。这能很好地了解怎样构建一个和PyTorch交互的web应用。

这周里大家构建的web apps 的例子 [3:36](#)

Edward Ross 构建了这个程序“这是一辆什么汽车？”

WHATCAR

WHAT CAR IS THAT?

Upload a photo by clicking below and we'll tell you WhatCar it is

"Building an app is a great experience; being able to test my model live with photos from my phone is amazing and helped me understand my model a lot more"

It currently classifies around 400 models of the most popular cars.

It has 88% accuracy on a balanced validation set (although there are some duplicates in my dataset; if any leaked into the validation set this is optimistic). This is pretty impressive since some of the models are really similar.

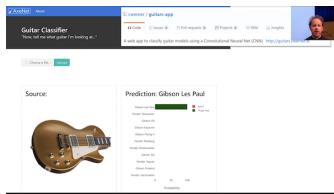
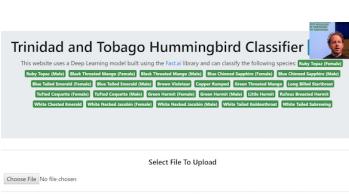
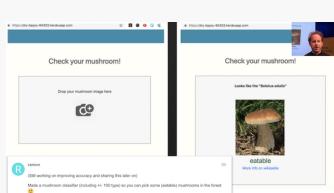
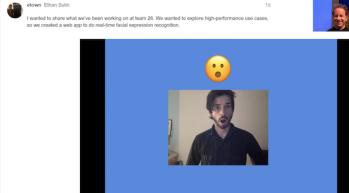
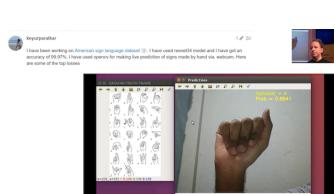
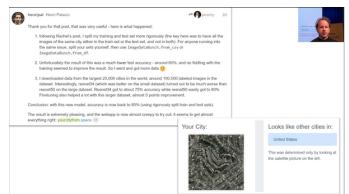
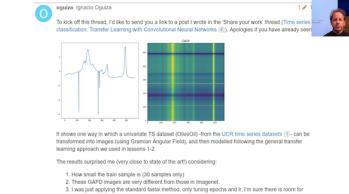
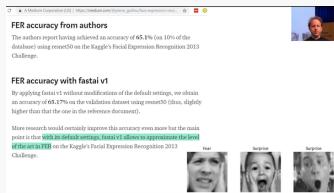
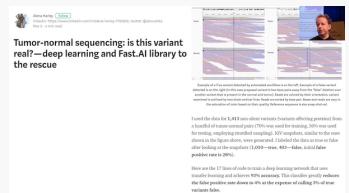
Made by Edward Ross

Mercedes Benz C200 Kompressor

Mercedes Benz C180 Kompressor

他在论坛上说，构建这个APP能很好地来理解模型是怎样变得越来越好的，这很有意义。他说他正在尝试，在手机上运行它。很多人会想，如果我要在手机上运行，应该创建一些mobile TensorFlow、ONNX之类的东西。你不需要这样做。你可以在云端运行所有的东西，然后用web APP，或者一些简单的轻量的前端，通过rest和后端通信。大多数时候你不需要在手机上运行模型，这是一个

很好的例子。

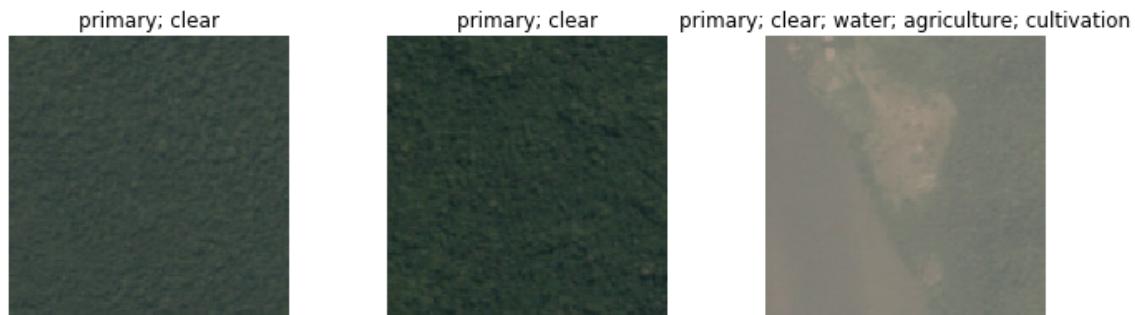
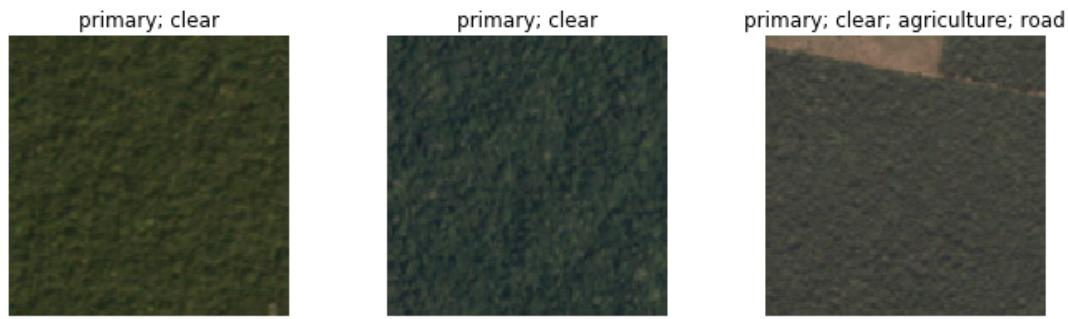
 <p>吉他分类 by Christian Werner</p>	 <p>健康诊断 by Nikhil Utane</p>	 <p>蜂鸟分类 by Nissan Dookeran</p>
 <p>食用菌识别 by Ramon</p>	 <p>表亲识别 by Charlie Harrington</p>	 <p>情感分类 by Ethan Sutin and Team 26</p>
 <p>手语 by Keyur Paralkar</p>	 <p>卫星城市图像 by Henri Palacci</p>	 <p>单变量时间序列分析 by Ignacio Oguiza</p>
 <p>表情识别 by Pierre Guillou</p>	 <p>癌症测序 by Alena Harley</p>	

很高兴看到大家创建的web app和分类器。今天要学一些不同类型的模型。我们先快速浏览一下，然后回过头来看看，它们是怎样工作的、它们的共同点是什么。你可以用这些来创建web app，也要想想怎样在上面做些修改来用到不同应用上。我觉得这是让你理解这些内容的一个很好的方法。

使用Planet Amazon数据集做多标签分类 9:51

[lesson3-planet.ipynb](#)

首先我们要看的是一个卫星图的数据集。卫星图是深度学习应用很多的一个领域。很多人都在用深度学习处理卫星图。我们要看的卫星图是这样的：



每个卫星都有一些不同的标签。通常会有一个标签表示天气，比如多云、局部多云。其他的标签会表示出所有能被看到的有意义的特征。primary代表雨林、agriculture代表有一些农田、road表示道路，等等。你一定看到了，这和我们之前看到的所有分类器有些不同，它有不止一个标签，它可能会有多个标签。可以用和之前类似的方法来做多标签分类。首先我们需要下载数据。

下载数据 11:02

这个数据来自Kaggle。Kaggle是一个广为人知的竞赛网站。在学习时，用Kaggle上的数据集是很好的，因为你可以看到，如果你参加比赛你的成绩会是怎样的。这是一个很好的方式来检查你做得怎么样。我一般把目标设成，进入前10%。根据我的经验，竞赛里前10%的人，真正知道他们在做什么。如果你可以达到前10%，那这是一个很好的标志。

几乎所有的Kaggle数据，在Kaggle之外是下载不到的，至少竞赛数据是这样的，所以你需要在上面Kaggle下载它。好消息是，Kaggle提供了一个基于python的下载工具。在这里我们简单介绍一下，怎样在Kaggle上面下载东西。

首先你需要通过 `pip` 安装Kaggle下载工具

```
1 | #! pip install kaggle --upgrade
```

在我们提供的notebook里，对于只做一次的事情，会把这部分代码注释掉。你可以在使用时，取消掉这个注释。你选中一些行，然后点击 `ctrl`+`/` 就会取消掉所有的注释。如果你再次选中它们，再点击 `ctrl`+`/`，会重新添加上注释。这一行会为你安装 `kaggle`。你可能需要使用 `sudo` 或者 `/something/pip`，这取决于你的系统。你可能需要使用 `source activate`，你可以在课程的网站上查看一下做 `conda install` 时要用的 `setup` 部分或者 `returning to work` 部分，当你使用 `pip install` 时，你也需要做相同的基本步骤。

当你安装完这个模块后。你就可以下载数据了。基本上就是使用 `kaggle competitions download -c competition_name -f file_name`。唯一需要说明的步骤是你需要通过权限检查。这里有一些关于怎样下载你的API授权信息的内容。按照这个步骤做就可以。

然后你需要上传你的Kaggle认证文件到你的实例。登录kaggle，点击右上角的你的头像图片，然后点击“My account”。下滑滚动条，找到叫做“Create New API Token”的按钮，点击它，这会下载一个叫做“kaggle.json”的文件。

通过点击Jupyter页面的“Upload”按钮，把这个文件上传到notebook运行的目录。然后取消掉后面两行的注释，执行它们（或者在命令行里运行它们）。

```
1 #! mkdir -p ~/.kaggle/
2 #! mv kaggle.json ~/.kaggle/
```

设置从[planet competition](#)下载数据。**首先要到竞赛主页接受竞赛规则**，运行下面的两个单元格（取消掉shell命令的注释来下载解压数据）。如果你收到 403 forbidden 错误，代表你没有接受竞赛规则（你需要在竞赛页面，点击Rules选项卡，滚动到底部，找到Accept按钮）。

```
1 path = Config.data_path()/'planet'
2 path.mkdir(exist_ok=True)
3 path
```

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

```
1 # ! kaggle competitions download -c planet-understanding-the-amazon-from-space -f train-jpg.tar.7z -p {path}
2 # ! kaggle competitions download -c planet-understanding-the-amazon-from-space -f train_v2.csv -p {path}
3 # ! unzip -q -n {path}/train_v2.csv.zip -d {path}
```

有时Kaggle上的东西不是zip或者tar，而是用叫做7zip的程序压缩的，后缀是.7z。这种情况，如果你的电脑没有安装过这个软件，你需要用`apt install p7zip`或者用conda 7zip来安装。有些人创建了conda 7zip安装包，它可以在所有平台使用。你可以只运行`conda install`，不需要`sudo`或其他类似的东西。这是证明 conda 超级好用的一个例子。你可以安装二进制文件或者库或者其他东西，它支持跨平台。如果你没有安装过7zip，这是一个好的安装方式。

我们7zip把文件里的内容解压出来，如果你需要安装它的话，去除下面一行代码的注释并执行（或者在命令行执行`sudo apt install p7zip`）

```
1 # ! conda install -y -c haasad eidl7zip
```

这是解压7zip文件的方法。在这个数据集里，它是用tar和7zip压缩的，你可以一步完成解压。`7za` 是你要运行的7zip解压程序名称。

如果你对命令行不熟悉，这是你要了解的所有基本知识。你可能需要实验几次来让它跑通。遇到问题时，如果在论坛搜索不到，大胆在论坛提问。

```
1 # ! 7za -bd -y -so x {path}/train-jpg.tar.7z | tar xf - -c {path}
```

多标签分类 14:49

下载解压完数据后，你可以查看一下。在这个例子里，因为每个图片有多个标签，我们不能用不同的文件夹来表示标签。我们需要不同的方式来标识。Kaggle的标识方式是提供一个CSV文件，里面的每一个文件名有对应的所有标签。我们可以使用Pandas库来读取CSV文件，看看里面的内容。如果你之前没有用过pandas，它是Python里处理表格数据的标准方法。它经常出现在 `pd` 命名空间里。这里我们只是用它来查看文件的内容。我们读取它，看看最开始的几行，就像这样：

```
1 df = pd.read_csv(path/'train_v2.csv')
2 df.head()
```

	image_name	tags
0	train_0	haze primary
1	train_1	agriculture clear primary water
2	train_2	clear primary
3	train_3	clear primary
4	train_4	agriculture clear habitation primary road

我们要把这个转换成可以用来建模的格式。我们用DataBunch类的一个对象来建模。我们用它创建一个data bunch。有了data bunch，我们就能用 `.show_batch()` 来查看它。然后我们可以用它来执行 `create_cnn`，开始训练。

开始深度学习前最麻烦的步骤是把数据转换成可以放入模型的格式。我们已经演示过怎样使用多种“工厂方法”来做这个，这些方法让你能根据这些选项、从这些种类的源数据创建出这些种类的目标数据。有时这些方法工作得很好，前两周我们展示了做这个的一些方法。但是有时你想要能更灵活，因为要做很多关于以下这些的选择：

- 这些文件在哪里
- 它们的结构是怎样的
- 标签是怎样标记的
- 怎样分离出验证集
- 怎样变形它

所以我们做了这个独特的API [data block API](#)。这个data block API让每一个选择相互独立。对创建、设置数据的每个选择都提供了独立的方法，这些方法都有各自的参数。

```
1 tfms = get_transforms(flip_vert=True, max_lighting=0.1, max_zoom=1.05,
max_warp=0.)
```

```
1 np.random.seed(42)
2 src = (ImageFileList.from_folder(path)
3         .label_from_csv('train_v2.csv', sep=' ', folder='train-
jpg', suffix='.jpg')
4         .random_split_by_pct(0.2))
```

```
1 | data = (src.datasets()  
2 |         .transform(tfms, size=128)  
3 |         .databunch().normalize(imagenet_stats))
```

比如，要取卫星数据，我们会这样写：

- 我们从文件夹取ImageFileList
- 它们的标签放在一个CSV文件里，文件叫做 (`train_v2.csv`)
 - 标签有这个分隔符 ()，我展示过它们中间有个空格。通过传入分隔符，让它创建多标签。
 - 图片在这个文件夹里 (`train-jpg`)
 - 它们有这个后缀 (`.jpg`)
- 它们会随机分出20%的数据作为验证集
- 我们会用这个 (`.datasets()`) 创建dataset，然后用这个对象 (`tfms`) 做变形 (transform)
- 然后我们会用这个 (`.databunch()`) 创建一个data bunch，我们会用这个统计分布方法 (`imagenet_stats`)做标准化

这是所有的步骤。为了让你们知道这些是什么，我先回过头来解释所有在这个过程里出现的、你需要了解的这些PyTorch类和fastai类。你会经常在fastai文档和PyTorch文档里看到它们。

Dataset (PyTorch) [18:30](#)

首先要了解的是Dataset类。Dataset类是PyTorch的一部分，这是Dataset的源代码。

```
class Dataset(object):  
    """An abstract class representing a Dataset.  
  
    All other datasets should subclass it. All subclasses should override  
    ``__len__``, that provides the size of the dataset, and ``__getitem__``,
    supporting integer indexing in range from 0 to len(self) exclusive.  
    """  
  
    def __getitem__(self, index):  
        raise NotImplementedError  
  
    def __len__(self):  
        raise NotImplementedError
```

可以看到，它实际上没有做任何事情。PyTorch里的Dataset定义了两个东西：`__getitem__` 和 `__len__`。在Python里这种“下划线+下划线+某个方法+下划线+下划线”的方法被叫做“dunder”某个方法。这两个方法叫“dunder get items”和“dunder len”。它们是有特殊行为的魔法方法。这个方法代表如果你有一个叫做 `o` 的对象，它可以用方括号索引（比如 `o[3]`）。它会调用 `__getitem__`，把3作为索引。

这个叫做 `__len__` 的方法代表你可以执行 `len(o)`，它会调用这个方法。在这个例子里，它们都没有实现。也就是说，尽管PyTorch说“为了把你的数据告诉PyTorch，你需要创建一个dataset”，但它并不会帮助你做任何事来创建dataset。它只是定义了dataset需要做什么。换句话说，你可以说：

- 我的dataset的第3个数据是什么（这是 `__getitem__` 做的）
- 我的dataset有多大（这是 `__len__` 做的）

Fastai有很多Dataset子类来处理各种不同的东西。目前为止你已经看到了图像分类dataset。对这种dataset，`__getitem__` 会返回一个图片和一个标识这个图片是什么的标签。这就是dataset。

DataLoader (PyTorch) [20:37](#)

要训练模型，dataset还不够。如果你回想上周讲的梯度下降，我们需要做的第一件事是，一次性取到一些图片/数据，这样GPU可以并行处理。记得“mini-batch”吗？mini-batch是我们在同一批里传给模型的一组数据，它们可以并行训练。要创建一个mini-batch，我们使用另外一个叫做DataLoader的PyTorch类。

```
class torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False,
sampler=None, batch_sampler=None, num_workers=0, collate_fn=<function
default_collate>, pin_memory=False, drop_last=False, timeout=0, worker_init_fn=None)
\[source\]
```

Data loader. Combines a dataset and a sampler, and provides single- or multi-process iterators over the dataset.

Parameters:

- **dataset** ([Dataset](#)) – dataset from which to load the data.
- **batch_size** ([int, optional](#)) – how many samples per batch to load (default: 1).

DataLoader对象在构造函数里接收一个dataset。它现在说“噢，用这个，我可以取到第三个数据、第五个数据、第九个数据”。它会：

- 随机抓取数据
- 按照你要求的大小创建一个batch
- 把它输入进GPU
- 把它发送到你的模型里

所以DataLoader是一个可以抓取独立数据，把它们组成一个mini-batch，输入GPU来建模的东西。所以它被叫做DataLoader。它由一个Dataset生成。

你可以看到，这些就是你需要做的选择：创建怎样的dataset、它的数据是什么、它来自哪里。然后我们创建DataLoader时要选择：我要用多大的批次尺寸 (batch size) 。

DataBunch (fastai) [21:59](#)

要训练模型只有这些还不够，因为我们没办法验证模型。如果我们只有训练集，我们没办法知道我们模型怎么样，我们需要分离出一个验证集，来检查模型的效果。

```
class DataBunch
```

[\[source\]](#)

```
DataBunch ( train_dl: DataLoader , valid_dl: DataLoader ,
test_dl: Optional [ DataLoader ]=None , device: device =None ,
tfms: Optional [ Collection [ Callable ]]=None ,
path: PathOrStr ='.' , collate_fn: Callable ='data_collate' )
```

Bind together a `train_dl`, a `valid_dl` and optionally a `test_dl`, ensures they are on `device` and apply to them `tfms` as batch are drawn. `path` is used internally to store temporary files, `collate_fn` is passed to the pytorch `Dataloader` (replacing the one there) to explain how to collate the samples picked for a batch. By default, it applies data to the object sent (see in `vision.image` why this can be important).

我们用一个叫DataBunch的fastai类来做这个。DataBunch把训练数据的数据 loader (train_dl) 和验证数据的数据 loader (valid_dl) 绑定在一起。在fastai文档，当你看到这些mono spaced字体的文字时，这代表它们是可以进一步查看的。在这里，train_dl 是DataBunch的第一个参数。除非你了解这个参数，你没办法仅通过名字知道这个参数是做什么的。你需要查看冒号后面的内容，来知道它是一个DataLoader。当你创建一个DataBunch时，你传入一个训练集DataLoader和一个验证集DataLoader。现在可以把它传入一个learner来做训练了。

这是基本的部分。回到这里，这是创建dataset的代码：

```
1 np.random.seed(42)
2 src = (ImageFileList.from_folder(path)
3         .label_from_csv('train_v2.csv', sep=' ', folder='train-jpg', suffix='.jpg')
4         .random_split_by_pct(0.2))
5
6 data = (src.datasets()
7         .tfms(tfms, size=128)
8         .databunch().normalize(imagenet_stats))
```

对这个dataset，这个indexer返回了两个东西：图片和标签（假设它是一个图片dataset）

- 图片来自哪里？
- 标签来自哪里？
- 然后我把数据分成训练dataset和验证dataset
- .datasets() 把它们转成PyTorch dataset
- .transform() 用来变形
- .databunch() 会一步创建DataLoader和DataBunch

Data block API例子 [23:56](#)

我们来看下data block API的例子，只要你理解了data block API，把数据转换成模型需要的形式就没有什么问题。

[data_block.ipynb](#)

MNIST

这是使用data block API的一些例子。如果你在做MNIST（手写数字分类数据集），你可以这样：

```
1 path = untar_data(URLs.MNIST_TINY)
2 tfms = get_transforms(do_flip=False)
3 path.ls()
4
5 [PosixPath('/home/jhoward/.fastai/data/mnist_tiny/valid'),
6  PosixPath('/home/jhoward/.fastai/data/mnist_tiny/models'),
7  PosixPath('/home/jhoward/.fastai/data/mnist_tiny/train'),
8  PosixPath('/home/jhoward/.fastai/data/mnist_tiny/test'),
9  PosixPath('/home/jhoward/.fastai/data/mnist_tiny/labels.csv')]
```

```
1 (path/'train').ls()
2
3 [PosixPath('/home/jhoward/.fastai/data/mnist_tiny/train/3'),
4  PosixPath('/home/jhoward/.fastai/data/mnist_tiny/train/7')]
```

```

1 | data = (ImageFileList.from_folder(path) #Where to find the data? -> in path
2 |     .label_from_folder()                 #How to label? -> depending on the
3 |     .split_by_folder()                  #How to split in train/valid? -> use
4 |     .add_test_folder()                 #Optionally add a test set
5 |     .datasets()                      #How to convert to datasets?
6 |     .transform(tfms, size=224)        #Data augmentation? -> use tfms with
7 |     a size of 224                    #Finally? -> use the defaults for
                                         conversion to ImageDataBunch

```

- 这是怎样的数据集？

- 它来自于一些文件夹里的一个图片列表
- 所在的文件夹的名字是它们的标签
- 我们按照它们所在的文件夹（train 和 valid）把它们分成训练集和验证集
- 你可以选择加入一个测试集。晚些时候我们会讲测试集
- 我们把这些转成 PyTorch dataset
- 我们用这个变形对象（tfms）做变形，我们把它变成这个尺寸（224）
- 然后把它转成data bunch

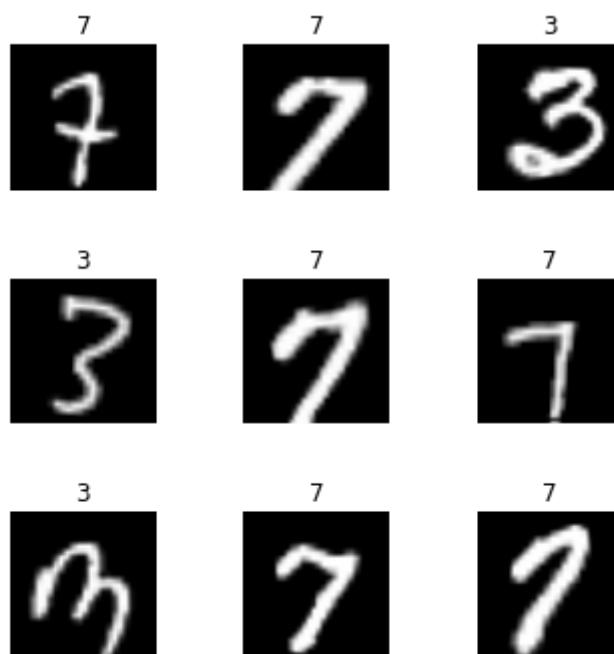
每一步你都可以传入各种参数，来控制处理过程。对于MNIST这样的数据集，默认参数就可以运行得很好。

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

```
1 | (Image (3, 224, 224), 0)
```

`data.train_ds` 是dataset（不是data loader），所以我可以用数字来索引。这是训练集里第0个数据：一个图片和一个标签。

```
1 | data.show_batch(rows=3, figsize=(5,5))
```



我们可以用show_batch看看里面的图片。然后就可以开始训练。

```
1 | data.valid_ds.classes
```

```
1 | ['3', '7']
```

这是dataset里的类别。这个被裁剪的MNIST数据集里的类别是3和7。

Planet 26:01

这是一个planet数据集的例子。这实际上是数据集的一小部分，方便我们做实验。

```
1 | planet = untar_data(URLs.PLANET_TINY)
2 | planet_tfms = get_transforms(flip_vert=True, max_lighting=0.1, max_zoom=1.05,
3 |     max_warp=0.)
4 | data = ImageDataBunch.from_csv(planet, folder='train', size=128,
5 |     suffix='.jpg', sep=' ', ds_tfms=planet_tfms)
```

使用data block API，我们可以这样重写：

```
1 | data = (ImageFileList.from_folder(planet)
2 |         #Where to find the data? -> in planet and its subfolders
3 |         .label_from_csv('labels.csv', sep=' ', folder='train',
4 |             suffix='.jpg')
5 |         #How to label? -> use the csv file labels.csv in path,
6 |         #add .jpg to the names and take them in the folder train
7 |         .random_split_by_pct()
8 |         #How to split in train/valid? -> randomly with the default 20% in
9 |         #valid
10 |         .datasets()
11 |         #How to convert to datasets? -> use ImageMultiDataset
12 |         .transform(planet_tfms, size=128)
13 |         #Data augmentation? -> use tfms with a size of 128
14 |         .databunch())
15 |         #Finally? -> use the defaults for conversion to databunch
```

这个例子里：

- 还是一样，它是一个ImageFileList
- 我们从一个文件夹里取出它
- 这次我们使用CSV文件来生成标签
- 我们可以随机分割它（默认比例是20%）
- 创建dataset
- 使用变形对象（planet_tfms）做变形，使用一个比较小的尺寸（128）
- 创建一个data bunch

```
1 | data.show_batch(rows=3, figsize=(10,8))
```

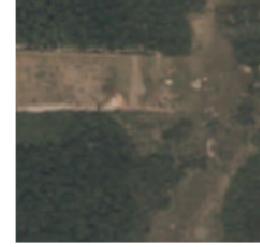
partly_cloudy; primary



primary; agriculture; clear; road; habitation



primary; agriculture; clear



primary; clear



partly_cloudy; primary; water



primary; clear



primary; clear



partly_cloudy; primary



primary; clear



完成了。Data bunch可以把它的内容画出来。

CAMVID 26:38

这是我们晚些会看到的另外一个例子。

```
1 | camvid = untar_data(URLs.CAMVID_TINY)
2 | path_lbl = camvid/'labels'
3 | path_img = camvid/'images'
```

```
1 | codes = np.loadtxt(camvid/'codes.txt', dtype=str); codes
```

```
1 | array(['Animal', 'Archway', 'Bicyclist', 'Bridge', 'Building', 'car',
2 | 'CartLuggagePram', 'Child', 'Column_Pole',
3 | 'Fence', 'LaneMkgsDriv', 'LaneMkgsNonDriv', 'Misc_Text',
4 | 'MotorcyclesScooter', 'OtherMoving', 'ParkingBlock',
5 | 'Pedestrian', 'Road', 'Roadshoulder', 'Sidewalk', 'SignSymbol', 'sky',
6 | 'SUVPickupTruck', 'TrafficCone',
7 | 'TrafficLight', 'Train', 'Tree', 'Truck_Bus', 'Tunnel',
8 | 'VegetationMisc', 'Void', 'Wall'], dtype='<U17')
```

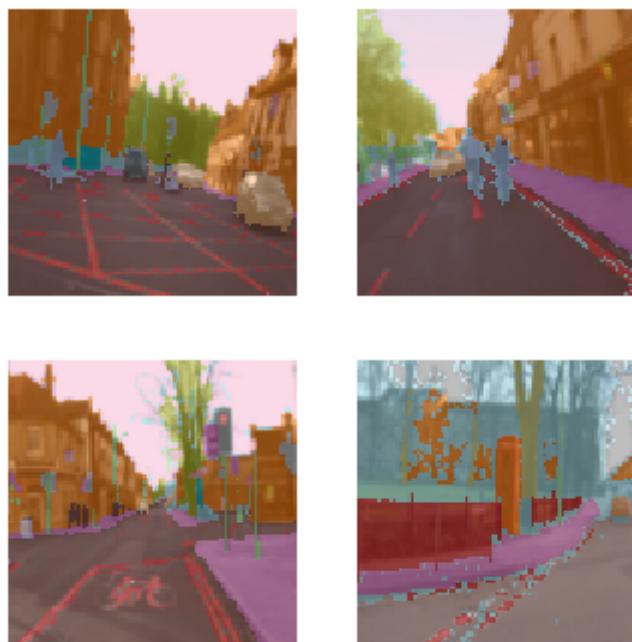
```
1 | get_y_fn = lambda x: path_lbl/f'{x.stem}_P{x.suffix}'
```

```

1 | data = (ImageFileList.from_folder(path_img)                      #Where are the
   |     .label_from_func(get_y_fn)                                     #How to label? ->
   |     use get_y_fn
   |     .random_split_by_pct()                                       #How to split
   | between train and valid? -> randomly
   |     .datasets(SegmentationDataset, classes=codes)                #How to create a
   | dataset? -> use SegmentationDataset
   |     .transform(get_transforms(), size=96, tfm_y=True)           #Data aug -> Use
   | standard tfms with tfm_y=True
   |     .databunch(bs=64))                                         #Lastly convert in
   | a databunch.

```

```
1 | data.show_batch(rows=2, figsize=(5,5))
```



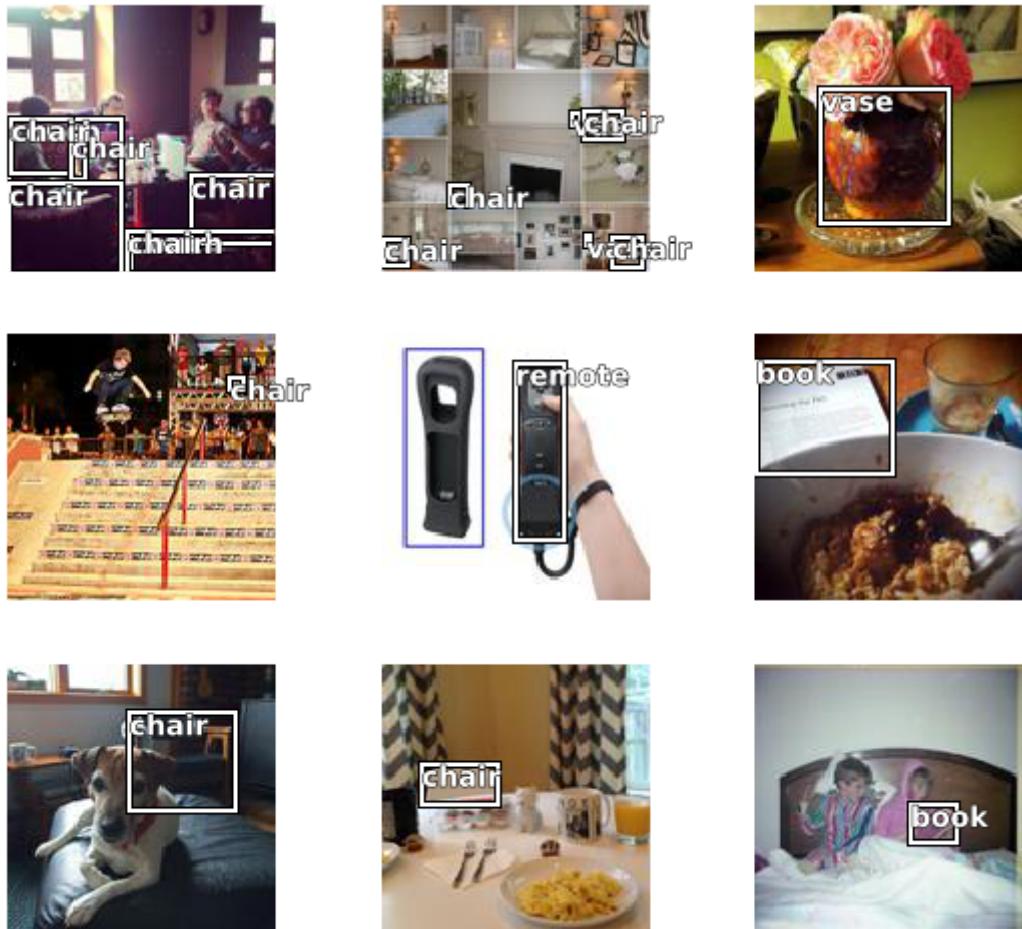
在这个CAMVID数据集里有什么？CAMVID看起来是这样的。它包含这些图片，图片的每个像素都被做了颜色标注。这个例子里：

- 我们有一个文件list，从文件夹里取到的
- 我们用一个函数来标记它们，这个函数（get_y_fn）用来告诉我们不同位置颜色标注的内容
- 随机分割它
- 创建dataset，我们可以输入记录类别的list，它们存储了1和2这样的标签值代表了什么含义，这可以从文件（codes.txt）里读取
- 做一些变形
- 创建一个data bunch，可以选择传入batch size之类的

同样的，它可以画出里面的内容。你可以用它开始训练

COCO 27:41

另外一个例子。如果要创建这样的东西要怎样做：



这是一个目标检测数据集。同样的，我们用的是裁剪过的缩小版COCO数据集。COCO是最著名的
目标检测学术数据集。

```

1 | coco = untar_data(URLs.COCO_TINY)
2 | images, lbl_bbox = get_annotations(coco/'train.json')
3 | img2bbox = {img:bb for img, bb in zip(images, lbl_bbox)}
4 | get_y_func = lambda o:img2bbox[o.name]

```

```

1 | data = (ImageFileList.from_folder(coco)
2 |     #where are the images? -> in coco
3 |     .label_from_func(get_y_func)
4 |     #How to find the labels? -> use get_y_func
5 |     .random_split_by_pct()
6 |     #How to split in train/valid? -> randomly with the default 20% in
7 |     # valid
8 |     .datasets(ObjectDetectDataset)
9 |     #How to create datasets? -> with ObjectDetectDataset
10 |    #Data augmentation? -> Standard transforms with tfm_y=True
11 |    .databunch(bs=16, collate_fn=bb_pad_collate))
12 |    #Finally we convert to a DataBunch and we use bb_pad_collate

```

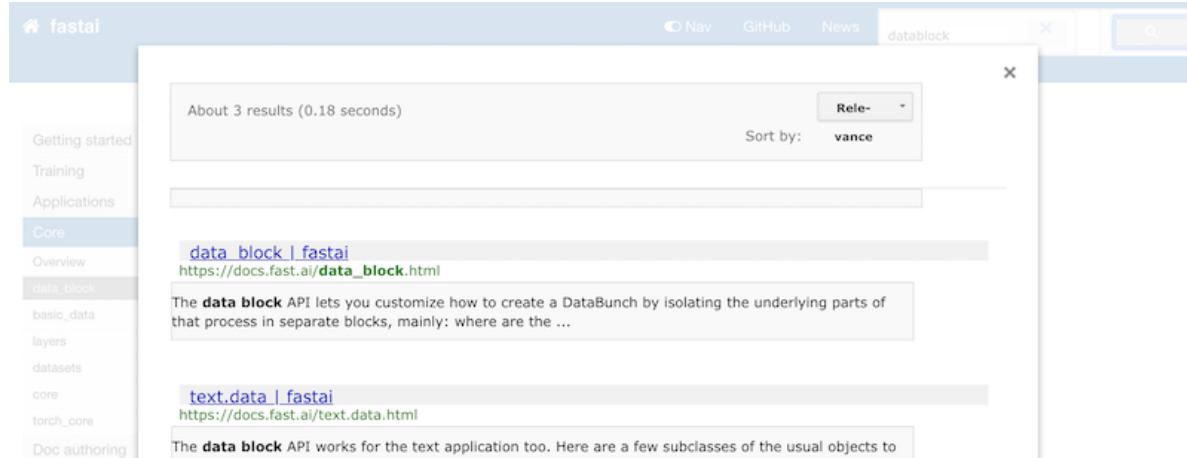
我们可以用相同的流程创建它：

- 从文件夹里取文件列表
- 使用这个函数做标记 (`get_y_func`)
- 随机分割它
- 创建目标检测数据集

- 创建 data bunch。这个例子里你需要使用较小的batch size，否则你会内存溢出。你需要使用一个collation函数

完成后，我们可以画出它来，这就是我们的目标检测数据集。这是一个很方便的notebook，哪里可以找到它？这个notebook就是这个文档。记得我说过我们所有的文档都是notebook生成的吗？你可以在[fastai仓库docs_src目录](#)找到它们。我们可以使用不同输入来做实验。在repo里你可以找到[data block API 使用实例 \(notebook\)](#)，如果你访问文档，它在这里：[data block API 使用实例 \(doc\)](#)。

所有你想在fastai里使用的东西，你都可以在文档里找到它。这里还有搜索功能：



如果你想尝试一下的文档里的内容，直接搜索名字（比如 `data_block.html`），你就可以打开 fastai仓库里一个相同名字的notebook(比如 `data_block.ipynb`)来做实验了。

创建卫星图data bunch 29:35

这是对data block API的简单介绍，它是一个很好的API。有很多关于标记输入、分割数据、创建dataset的不同方法的文档。这是我们在planet用到的。

在这个文档里，这两个步骤合在一起了：

```

1 np.random.seed(42)
2 src = (ImageFileList.from_folder(path)
3     .label_from_csv('train_v2.csv', sep=' ', folder='train-jpg',
4     suffix='.jpg')
5     .random_split_by_pct(0.2))

1 data = (src.datasets()
2     .transform(tfms, size=128)
3     .databunch().normalize(imagenet_stats))

```

我们也可以在这里做，但是稍后你会学到为什么我们把它分成两个步骤比较好。

一些关于这个的有意思的细节。

- **变形**：默认的变形是随机反转每个图片，但是只会水平随机反转。如果你要识别猫和狗，朝左朝右没有关系，但是你不会希望它上下反转。对卫星图来说，是否多云，有雾，是否有道路显然可以上下反转。对空间来说没有必须的方向。所以我们不用 `flip_vert` 的 `False` 的默认值，要把它改成 `True`，表示可以上下反转。它不仅会上下反转，还会尝试每种90度旋转（会尝试8种可能的旋转）
- **Warp**: 很少库会提供Perspective warping功能，即使提供了也会很慢。我觉得fastai是第一个提供快速perspective warping的库。这个比较有意义的原因是，如果我在上方或者下方看

你，你的形状是不一样的。所以如果你对猫和狗拍照，有时会比较高，有时会比较矮，这种情况导致的形状的改变，是你在创建batch时希望处理的，每次都修改一些。对卫星图来说不是这样，卫星都是垂直向下拍摄。如果你加入perspective warping，你将加入现实中不存在的变形。所以要关闭它。

这些都属于数据增强（data augmentation）。后续我们还会讲很多关于这个的内容。现在你可以开始了解下，用它来增强你的数据。通常，或许最重要的是，如果你的数据是天文数据，医学影像或者卫星图这种没有上下之分的图像，就把垂直反转设成true，这通常会让你的模型更好。

创建多标签分类器 35:59

接下来要创建一个可以识别出卫星图上天气或者其他东西的多标签分类器，基本上没有新东西要学，之前学过的可以基本可以照样用。

```
1 | arch = models.resnet50  
  
1 | acc_02 = partial(accuracy_thresh, thresh=0.2)  
2 | f_score = partial(fbeta, thresh=0.2)  
3 | learn = create_cnn(data, arch, metrics=[acc_02, f_score])
```

第一次做这个notebook时，我用的是通常的 resnet34。然后我像往常一样尝试了 resnet50。我发现 resnet50 效果更好，所以在这个例子里我使用 resnet50。

另外一个修改是metrics。提醒一下，metrics与模型训练无关。改变它不会改变模型结果。它只是在训练时打印出来的东西。

```
1 | lr = 0.01  
  
1 | learn.fit_one_cycle(5, slice(lr))
```

```
1 | Total time: 04:17  
2 | epoch train_loss valid_loss accuracy_thresh fbeta  
3 | 1     0.115247  0.103319  0.950703    0.910291 (00:52)  
4 | 2     0.108289  0.099074  0.953239    0.911656 (00:50)  
5 | 3     0.102342  0.092710  0.953348    0.917987 (00:51)  
6 | 4     0.095571  0.085736  0.957258    0.926540 (00:51)  
7 | 5     0.091275  0.085441  0.958006    0.926234 (00:51)
```

这里它打印出了准确度，和另外一个叫 fbeta 的度量标准。如果你希望改进模型，改变metrics没有什么帮助。它只是显示出训练效果。

你可以使用一个metric（度量），也可以不用，也可以使用很多个。这个例子里，我想看到两个指标：

1. 准确率
2. 在Kaggle上的表现怎样

Kaggle会用一个叫F score的指标来评价模型。我不会花时间解释F score，它没什么意思。它基本是这样的，你的分类器会得到一些假正例（false positives，预测正，这是错的）和假负例（false negatives，预测负，这是错的）。怎样为这两个指标分配权重来得到一个整体的指标？有很多不同的方式把它们组合成一个值，F score是一个好方法。有多种F score：F1, F2等等。Kaggle在竞赛规则里说，要用F2做度量指标。

```
In [23]: arch = models.resnet50
In [ ]: fbeta()
In [24]: Signature: fbeta(y_pred:torch.Tensor, y_true:torch.Tensor, thresh:float=0.2, beta:float=2, eps:float=1e-09, sigmoid:bool=True) -> <function NewType.<locals>.new_type at 0x7f6c9d6c9b70>
Docstring: Computes the f_beta between preds and targets
```

We use the LR Finder to pick a good learning rate.

我们有一个叫 `fbeta` 的度量指标。也就是说它是以beta值（1、2或其他）做参数的F值。我们可以看下它的签名，它有一个阈值（threshold）和一个beta值，beta值默认是2，Kaggle会使用F2，所以我们不用修改它。但是我们要修改阈值。

它代表什么？你记得我们看过准确度的代码吗？我们看到它用了 `argmax`。我们输入图片，它经过我们的模型，输出十个数字。如果我们在做MNIST数字识别，这十个数字是十个可能结果的概率。然后我们会检查所有的输出，找出最大的。在Numpy、PyTorch、math这些库里，找出最大值返回对应索引的函数都叫 `argmax`。

为了得到我们的宠物识别程序的准确率，我们使用了 `argmax` 找出我们看到的类别ID。然后和真实值做对比，再取平均值，这就是准确率。

37:23

对卫星图识别来说，我们不能这样做，它有不止一个标签，而是有很多个。`data bunch`有一个特别的属性 `c`，`c` 是我们希望模型输出的结果数量。对所有分类器，我们都想看到每个可能类别的概率。换句话说，`data.c`通常和 `data.classes` 的长度是相等的。

```
1 data.c
17
1 len(data.classes)
17
1 data.classes
['road',
 'slash_burn',
 'cultivation',
 'water',
 'selective_logging',
 'bare_ground',
 'artisinal_mine',
 'blow_down',
 'partly_cloudy',
 'clear',
 'conventional_mine',
 'cloudy',
 'blooming',
 'agriculture',
 'haze',
 'habitation',
 'primary']
```

有17个可能结果。我们会得到每个可能结果的概率。但是我们不会只从中选择一个，而是会从这17个中选择 n 个。我们要做的是把每一个概率和阈值比较。如果概率高于阈值，我们就认为存在这个特征。所以我们要选择这个阈值。

```
1 arch = models.resnet50
2
3 acc_02 = partial(accuracy_thresh, thresh=0.2)
4 f_score = partial(fbeta, thresh=0.2)
5 learn = create_cnn(data, arch, metrics=[acc_02, f_score])
```

我发现对于这个数据集来说，把0.2作为阈值，模型表现不错。对这类问题，我们可以简单通过实验来找到一个好的阈值。我打算打印出把0.2作为阈值时的准确率。

但是不能直接使用普通的准确率函数。它不能用 `argmax`。我们需要用另一个准确率函数 `accuracy_thresh`。它会把每个概率和阈值对比，返回所有高于阈值的结果，用这个计算准确率。

accuracy_thresh()

Signature: accuracy_thresh(y_pred:torch.Tensor, y_true:torch.Tensor, thresh:float, gmoid:bool=True) -> <function NewType.<locals>.new_type at 0x7f6c9d6c9b70>

Docstring: Compute accuracy when `y_pred` and `y_true` are the same size.

File: /data1/jhoward/git/fastai/fastai/metrics.py

Python3 `partial` [39:17]

我们传入的一个参数是 `thresh`。现在我们的度量会调用我们的函数，我们不想每次回调时都告诉它阈值是什么。我们想创建一个这个函数的特别版本，让它每次都使用0.2做阈值。一个实现方式是定义一个 `acc_02` 这样的函数。

```
1 def acc_02(inp, targ): return accuracy_thresh(inp, targ, thresh=0.2)
```

我们可以这样做。但这是一个常见的问题，计算机科学里有一个描述这个的术语，叫“partial”/“partial function application”（创建一个和另一个函数类似的新函数，但是只用一个特定的参数调用它）。

Python3 支持 `partial`，它接收一些函数和一些参数名和值的列表，创建一个新的和 (`accuracy_thresh`)一样的函数，但它会一直使用这个参数调用(`thresh=0.2`)。

```
1 acc_02 = partial(accuracy_thresh, thresh=0.2)
```

用fastai库时会经常用到这个，因为有很多地方需要传入函数，你经常会需要传入一个稍微修改过的函数，这时你就可以使用它。

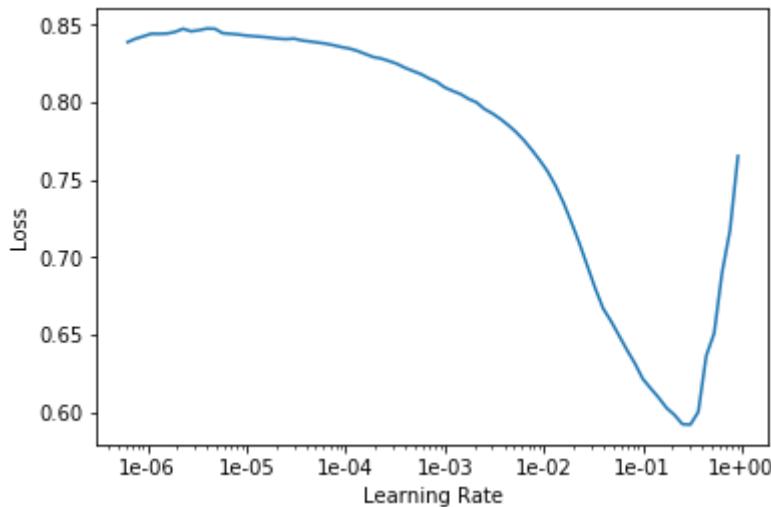
同样的，可以这样使用 `thresh=0.2` 来执行 `fbeta`：

```
1 acc_02 = partial(accuracy_thresh, thresh=0.2)
2 f_score = partial(fbeta, thresh=0.2)
3 learn = create_cnn(data, arch, metrics=[acc_02, f_score])
```

我可以把两个都作为度量传入，然后照常做其他的事情。

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

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



找到最陡的坡：是在 $1e-2$ 附近，把它作为我们的学习率。

```
1 | lr = 0.01
```

然后使用`5, slice(lr)`来拟合，看看结果。

```
1 | learn.fit_one_cycle(5, slice(lr))
```

```
1 | Total time: 04:17
2 | epoch  train_loss  valid_loss  accuracy_thresh  fbeta
3 | 1      0.115247   0.103319   0.950703       0.910291 (00:52)
4 | 2      0.108289   0.099074   0.953239       0.911656 (00:50)
5 | 3      0.102342   0.092710   0.953348       0.917987 (00:51)
6 | 4      0.095571   0.085736   0.957258       0.926540 (00:51)
7 | 5      0.091275   0.085441   0.958006       0.926234 (00:51)
```

我们得到了接近96%的准确率，F beta大概是0.926，你可以看下[Planet排行榜](#)。第50名在0.93左右，我们做对了。就像你可以看到的那样，只要把数据集准备好，训练过程基本上没有什么需要特别处理的。

提问：部署好的app如果做了错误的预测，有没有好的方法来记录错误，针对性地改进模型？[42:01]

很好的问题。首先是有没有办法记录。当然有。这取决于你自己。可能你们中的一些人这周就可以尝试一下。预测错时，你需要让用户告诉你。比如你的模型认为这个汽车是Holden，但实际上他是Falcon。所以首先，你需要收集这些反馈，唯一的方法是让用户告诉你有没有出错。然后你需要记录在日志里，一个记录预测值和用户说的实际值的文件。一天或者一周结束时，你可以用定时任务处理或者手工处理一下。怎样处理？做一些微调。怎样微调？像这样：

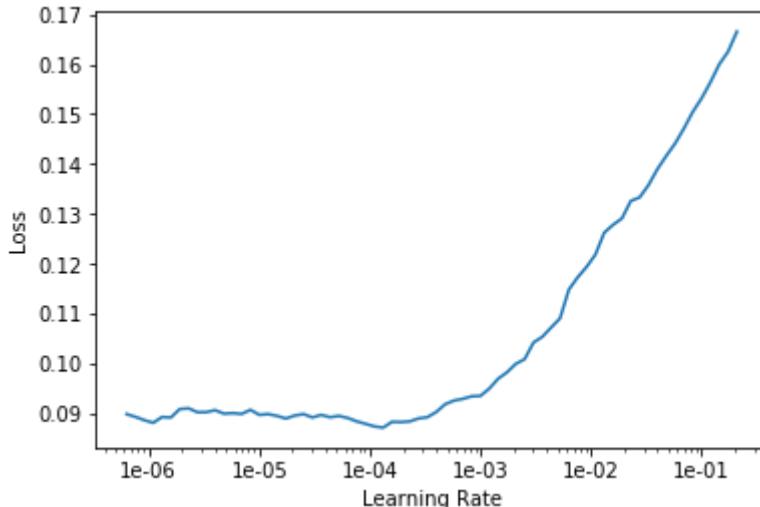
假如这是你保存的模型：

```
1 | learn.save('stage-1-rn50')
```

我们解冻它：

```
1 | learn.unfreeze()
```

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



然后再训练一次。这里，我使用的是原来的数据。但是你可以使用判断错误的数据创建一个新的 data bunch，用它来执行 fit。判断错误的数据是需要特别关注的。所以你可能需要用稍微高些的学习率，或者多训练几轮。你只需要用出错图片和修正的分类来训练。这能有效改进模型。

有很多其它改进技巧，但这是基本的思路。

```
1 | learn.fit_one_cycle(5, slice(1e-5, lr/5))
```

```
1 | Total time: 05:48  
2 | epoch  train_loss  valid_loss  accuracy_thresh  fbeta  
3 | 1      0.096917   0.089857   0.964909      0.923028 (01:09)  
4 | 2      0.095722   0.087677   0.966341      0.924712 (01:09)  
5 | 3      0.088859   0.085950   0.966813      0.926390 (01:09)  
6 | 4      0.085320   0.083416   0.967663      0.927521 (01:09)  
7 | 5      0.081530   0.082129   0.968121      0.928895 (01:09)
```

```
1 | learn.save('stage-2-rn50')
```

提问：能不能多讲讲data block的理念？我不明白它是用来做什么的。需要按固定的顺序使用吗？有没有其他使用这种方法的库能让我看下？ [\[44:01\]](#)

需要按照固定顺序使用它们。你可以在[使用示例](#)里看到顺序。

```

1 data = (ImageItemList.from_folder(path) #Where to find the data? -> in path
2     .split_by_folder()                  #How to split in train/valid? -> use
3     .label_from_folder()                #How to label? -> depending on the
4     .add_test_folder()                 #Optionally add a test set (here
5     .transform(tfms, size=64)          #Data augmentation? -> use tfms with
6     a size of 64                      #Finally? -> use the defaults for
7     .databunch())

```

- 你的数据是什么类型的?
- 它来自哪里?
- 怎样分割它?
- 怎样找到标签?
- 你需要哪种数据集?
- 怎样做变形, 这是可选的?
- 怎样创建一个data bunch?

这就是使用的步骤。我们发明了这个API。我不清楚没有其他人也发明了这个。这种把一串方法用点连接成流水线的方法是很常见的。在Python里没有那么多, 在JavaScript里会看到很多。这种方法里每一步产生了不同的东西。你可以在ETL (提取extraction、变形transformation、加载loading) 软件里看到这种用法, 这里几个步骤组成了流水线。设计它的灵感来自几个地方。但是你只需要按照示例去使用, 在文档里看看需要用哪个具体的方法。在里, 我们用的是 `ImageItemList`。你在datablock文档里找不到 `ImageItemList` 的内容, 它是视觉 (visoin) 应用部分的内容。对于其它方面的应用, 你需要用到text、vision等其它应用里对应的对象。在对应的应用文档里, 你可以找到相关的datablock API内容。

当然, 如果你要做全新类型的应用, 你可以查看源代码。对所有这些步骤, 自己写一个方法。基本所有的这类函数都只有几行代码。可能我们需要看一个例子。

```

In [44]: 1 ImageItemList.from_csv??

```

```

Signature: ImageItemList.from_csv(path:Union[pathlib.Path, str], csv_name:str, cols:Union[int, Collection[int], str, Collection[str]]=0, header:str='infer', folder:Union[pathlib.Path, str]='.suffix:str='') -> 'ItemList'
Docstring: Create an 'ItemList' in `path` from the inputs in the `cols` of `path/csv_name` opened with `header`.
Source:
@classmethod
def from_csv(cls, path:PathOrStr, csv_name:str, cols:IntsOrStrs=0, header:str='infer',
             folder:PathOrStr='.', suffix:str='')->'ItemList':
    df = pd.read_csv(Path(path)/csv_name, header=header)
    return cls.from_df(df, path=Path(path), cols=cols, folder=folder, suffix=suffix)
File:      ~/git/fastai/fastai/vision/data.py
Type:      method

```

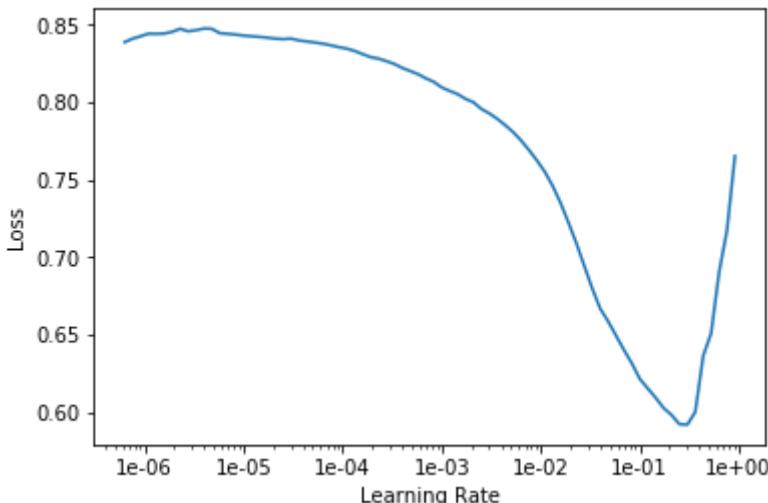
你可以查看文档看看它具体做了什么。就像你看到的那样, 大部分fastai函数都只有几行代码。可以直接看到全部内容, 了解怎样使用它们。多尝试使用, 你就会熟悉怎样把它们组织在一起。如果你有东西不明白, 请让我们知道, 我们会想办法帮助你。

提问: 有没有推荐的处理视频的工具? 比如抓取图片并传入模型[47:39]

看情况。如果你使用web, 我觉得大部分人是用web, 有web API可以做这个。你可以使用web API抓取图片, 然后直接把图片传给模型。如果你用的是客户端, 我觉得大部分人会使用OpenCV。这周, 做视频app的人可以告诉我们你们用过什么觉得很有用的工具, 我们可以在课程wiki里准备一个资源列表, 听起来好像很多人对这个感兴趣。

怎样选择好的学习率 [48:50]

这里需要注意的一件事情是，在我们解冻前，你可能会每次都得到这种形状。



如果你在解冻前使用slope的话。这很简单——找最陡的斜坡，**不是最底部**。记住，我们要寻找快速下滑的位置。所以，如果你从底部开始，损失度会上升。

解冻后再调用一次，你一般会得到一个非常不同的形状。

49:24

要找出合适的值有点难，因为它可能是这种形状，有点向上，然后快速下降，然后再向上。我一般会找最低点，然后缩小10倍，这是一个经验规则。所以用1e-5做区间的开始值，然后找区间的结束值，我一般参考解冻前的值，在它的基础上除以5或者10，基本是这个位置。这是我的经验：

- 找到接近底部的位置，除以10，作为区间的最小值
- `lr/5` 或者 `lr/10` 作为区间的最大值

这叫discriminative learning rates（差别学习率），后续课程里还会介绍。

改进模型 50:30

怎样让结果更好些？如果我们希望进入前10%，基本要达到0.929，现在我们的结果还达不到(0.9288)。

这是要用的技巧[51:01]。当我创建dataset时，我传入了`size=128`，事实上，Kaggle给我们的尺寸是256。我使用128的一个原因是想要快速实验。使用小图片来实验会更快更简单。第二个原因是，现在我们有了一个能很好地识别128 x 128卫星图里内容的模型。要想创建一个能识别256 x 256卫星图的模型应该怎样做？为什么不用迁移学习？为什么不在能识别128 x 128图片的模型上做微调？不再从头开始。这很有意义，因为如果我们训练太多次，会有过拟合的风险。我把图片长宽翻倍，面积变成4倍，这样创建一个新的dataset。对CNN来说这是完全不同的数据。继续用原来的程序，只是换成新的256 x 256的数据bunch。这就是之前为什么要在创建dataset前停下来：

```
1 np.random.seed(42)
2 src = (ImageItemList.from_csv(path, 'train_v2.csv', folder='train-jpg'
3     .random_split_by_pct(0.2)
4     .label_from_df(sep=' '))

1 data = (src.transform(tfms, size=128)
2     .databunch().normalize(imagenet_stats))
```

因为我要用这个数据源 (src) , 使用256的图片创建一个新的data bunch。看看怎样做。

```
1 | data = (src.transform(tfms, size=256)
2 |         .databunch().normalize(imagenet_stats))
```

就是这样。使用原来的数据 (src) , 使用原来的变形对象, 但是把size设成256, 这样模型效果会更好, 因为图片分辨率更大。我要在预训练模型基础上训练 (这还是原来的learner) 。

我要使用新的data bunch替换掉learner里的数据。

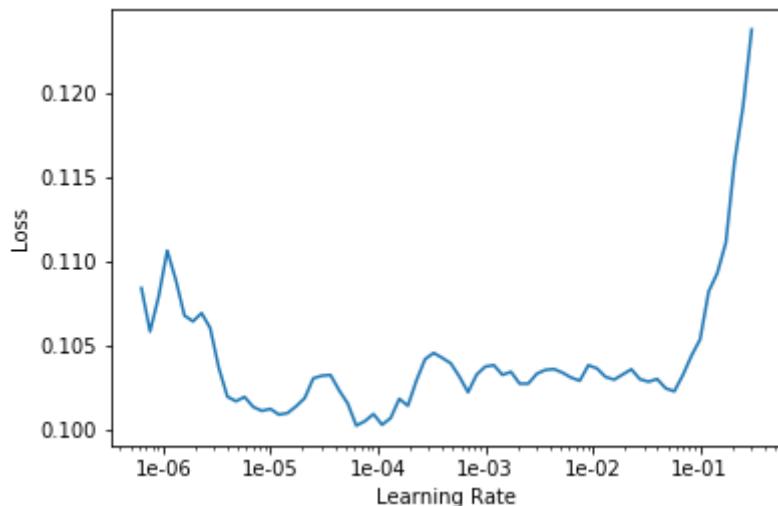
```
1 | learn.data = data
2 | data.train_ds[0][0].shape
```

```
1 | torch.size([3, 256, 256])
```

```
1 | learn.freeze()
```

然后我要再冻结 (仅训练最后几层) , 再运行 lr_find()

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



因为已经有了一个很好的模型 (它对 128×128 图片来说表现的很好, 所以它对 256×256 图片来说效果应该也不错) , 没有再遇到之前的特别倾斜的形状了。但是我还是会看到一段很高的线。所以我会找到它上升之前的位置。然后除以10。我用 $1e-2/2$, 这看起来是在上升前比较远的一个位置。

```
1 | lr=1e-2/2
```

再训练一下。

```
1 | learn.fit_one_cycle(5, slice(lr))
```

```
1 | Total time: 14:21
2 | epoch  train_loss  valid_loss  accuracy_thresh  fbeta
3 | 1      0.088628   0.085883   0.966523       0.924035 (02:53)
4 | 2      0.089855   0.085019   0.967126       0.926822 (02:51)
5 | 3      0.083646   0.083374   0.967583       0.927510 (02:51)
6 | 4      0.084014   0.081384   0.968405       0.931110 (02:51)
7 | 5      0.083445   0.081085   0.968659       0.930647 (02:52)
```

因为做了冻结，我们只训练最后几层，多训练几次。可以看到，之前训练几次后得到的成绩是0.928。我们直接达到了这里，并且很快超过了0.93。现在我们进入了前10%。我们达到了我们的第一个目标。我们至少可以自信地处理卫星图识别问题了。

```
1 | learn.save('stage-1-256-rn50')
```

当然现在我们可以像之前一样，我们可以解冻再多训练几次。

```
1 | learn.unfreeze()
```

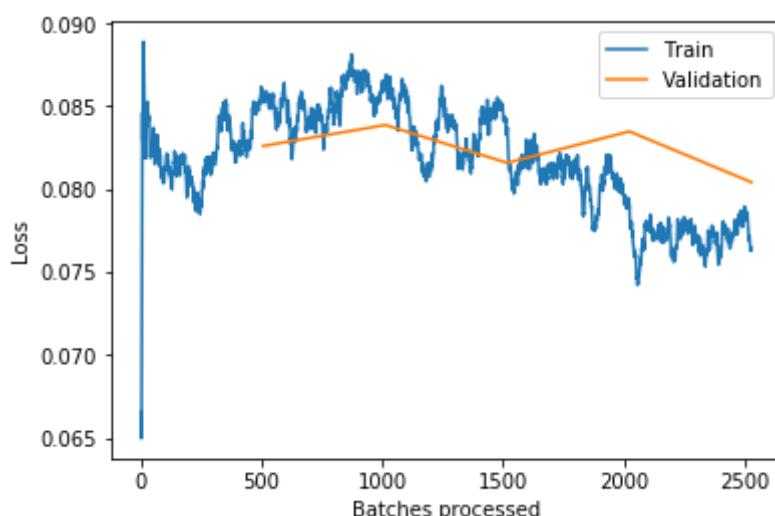
仍然使用我之前描述的方法，lr/5作为区间最大值，底部的十分之一作为区间最小值。

```
1 | learn.fit_one_cycle(5, slice(1e-5, lr/5))
```

```
1 | Total time: 18:23
2 | epoch  train_loss  valid_loss  accuracy_thresh  fbeta
3 | 1      0.083591   0.082895   0.968310       0.928210 (03:41)
4 | 2      0.088286   0.083184   0.967424       0.928812 (03:40)
5 | 3      0.083495   0.083084   0.967998       0.929224 (03:40)
6 | 4      0.080143   0.081338   0.968564       0.931363 (03:40)
7 | 5      0.074927   0.080691   0.968819       0.931414 (03:41)
```

再训练几次。成绩是0.9314，这很好，基本是前25名。一年前，我和我的朋友Brendan参加了这个竞赛，用0.9315的成绩得到了第22名，我们花费了几个月。现在使用很多默认值和一个改变尺寸的技巧，你就可以进入这个富有挑战的竞赛的排行榜前列。我不知道现在这个成绩能排多少名，我们需要用Kaggle提供的测试集测试结果，然后提交到竞赛里。你可以使用late submission。后面的课程里会学习怎样提交。但现在我们知道我们做的不错。

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



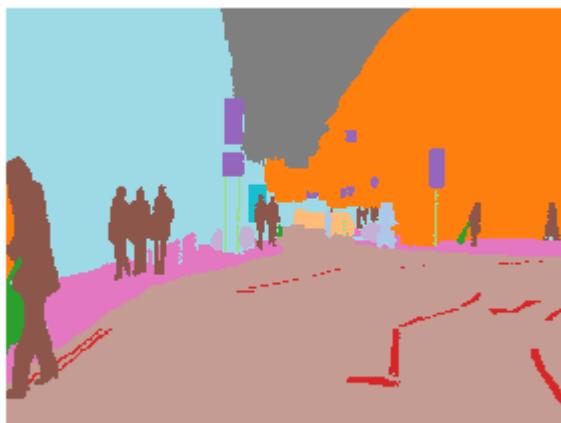
```
1 | learn.save('stage-2-256-rn50')
```

可以看到我经常保存模型。你可以随意命名，但是我会在名字里标明是否解冻、图片尺寸、模型架构。这样可以经常回来重新使用它，做实验更方便。这就是planet多标签分类。

图像分割示例: CamVid [56:31]

[Notebook](#)

下一个要讲的数据集是CamVid。它是用来做图像分割的。开始时我们有一张左边这样的图片：



我们要创建一张右边这样的用颜色标注过的图片，里面自行车是一种颜色、道路标线是一种颜色、树是一种颜色、建筑是一种颜色、天空是一种颜色，等等。

实际上我们不会给他们标颜色。我们会给每个像素一个代表类别的数值。在这个例子里，比如左上角是建筑，我猜建筑是4，右上角是树，是26，等等。

换句话说，我们会对右上角的图像做分类问题，就像处理宠物分类的问题一样。右上角的图像是什么？是一个自行车、道路标线、人行道、还是建筑？然后是下一个图像是什么？我们会对每个图片里的每个区域，当做一个小的分类问题处理。这就是图像分割。

要创建一个分割模型，你需要下载或者创建一个标注了每个区域的数据集。想象的出，这个工作量很大，你不用自己创建分割数据集，你可以下载或者在一些地方找到它。

这在医学和生命科学里很常见。如果你浏览细胞核的图片，很可能看到很多标注过的细胞和原子核。如果你在做放射工作，你可能有很多标注过的病变影像的例子，等等。所以，有很多不同的领域，有专门的工具创建这样的分割图片。从这个例子里可以猜到，它在自动驾驶，或者其它的需要看出什么东西在附近、它们在哪里，这样的类似领域里非常普遍。

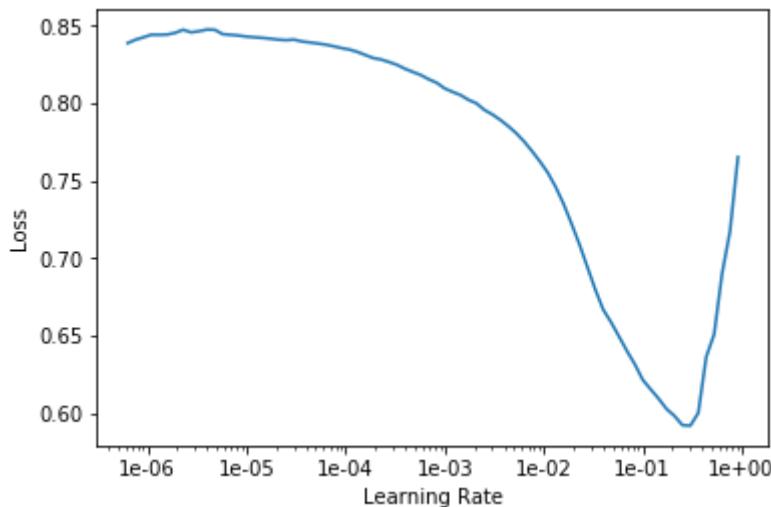
这里有一个很好的叫做CamVid的数据集。里面有很多标注好的图片。记住，所有的图片我们都提供了内置的URL。你可以在这里看到细节<https://course.fast.ai/datasets>。他们基本上都是学术数据集，有很多好心人帮我们处理了麻烦，让我们可以直接使用它。如果你在任何项目里使用这些数据集时，你能够添加数据引用，并且注明感谢作者，就太好了。因为他们提供这些数据，想要的只是能得到这样的荣誉。这就是CamVid数据集和他的引用（在我们的数据集页面，它会链接到发表这个数据集的学术论文）。

IMAGE LOCALIZATION

Source	Citation	Download	Description
Camvid: Motion-based Segmentation and Recognition Dataset	Brostow et al., 2008	download	Segmentation dataset with per-pixel semantic segmentation of over 700 images, each inspected and confirmed by a second person for accuracy.

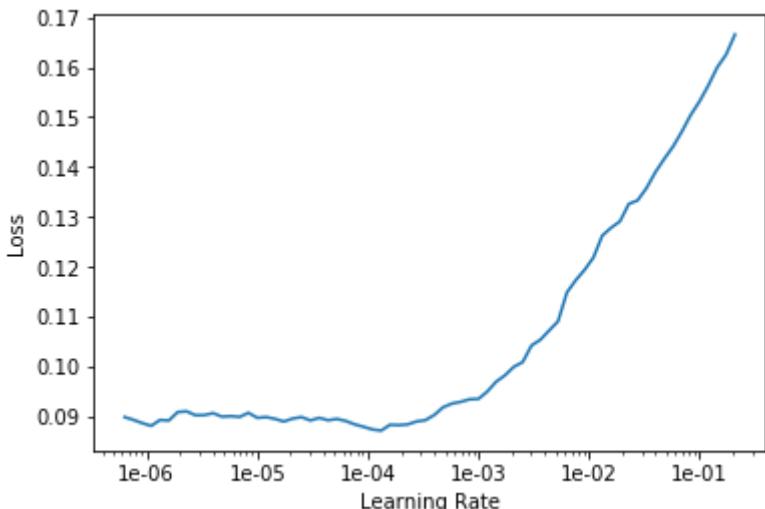
提问: 有没有什么方式使用 `learn.lr_find()` 这个方法，让它返回一个建议的值，而不是仅仅画出一张图，然后通过观察这张图来选择学习率？(还有一些关于解读学习率寻找器图像的其他问题)
[1:00:26]

答案是没有这样的方法，为什么说没有？是因为这更取决于经验。就像你能看到的，我说过，如何解读学习率图像，取决于现在是第几阶段，和它的形状是什么样的。我猜当只训练最后的几层时（在你解冻前），它的图像是这样的：



你确实可以创建一些方法，找出最陡的向下的斜坡来找出学习率。这基本上，在任何时候都是可行的。

但是对于这种，就需要很多的实验：



但好消息是你可以做些实验。显然，如果线是向上的，我们不想要它。几乎可以确定，你不希望正好在最底部的点这里，因为你需要它继续向下。你可以从这个位置的 $1/10$ 的地方开始尝试，然后再缩小 $1/10$ 。多试几次，找出最好的那个。

几周之内，你会发现，你可以每次都能找出最好的学习率。但在这个阶段，你仍然需要多做一些尝试，来培养对不同形状的感觉，培养如何处理它们的感觉。可能在这个视频发布的时候，有些人可以做出好用的自动learning rate finder，但现在还没有。这可能不是一个非常难的工作。这会是一个有意思项目：收集所有不同的数据集，可能要抓去我们数据集页面的所有数据集。尝试找出一些简单的方法，和我所有讲过的方法做对比。这会是一个很有趣的项目，我确信这是可能的，但现在我们还没有这样一个工具。

图像分割 [1:03:05]

怎样做图像分割呢？和其他处理类型问题的方式是一样的。首先，构建存放数据文件的路径变量。

```
1 %reload_ext autoreload
2 %autoreload 2
3 %matplotlib inline
```

```
1 from fastai import *
2 from fastai.vision import *
```

首先，我总是会解压文件，执行`ls`，看看里面有什么。这个例子里，有一个叫`labels`的标签文件夹，和一个`images`文件夹。我会为这两个文件夹创建path对象

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

```
1 [PosixPath('/home/ubuntu/course-v3/nbs/dl1/data/camvid/images'),
2 PosixPath('/home/ubuntu/course-v3/nbs/dl1/data/camvid/codes.txt'),
3 PosixPath('/home/ubuntu/course-v3/nbs/dl1/data/camvid/valid.txt'),
4 PosixPath('/home/ubuntu/course-v3/nbs/dl1/data/camvid/labels')]
```

```
1 path_lbl = path/'labels'
2 path_img = path/'images'
```

看看里面的内容

```
1 | fnames = get_image_files(path_img)
2 | fnames[:3]
```

```
1 | [PosixPath('/home/ubuntu/course-
v3/nbs/dl1/data/camvid/images/0016E5_08370.png'),
2 | PosixPath('/home/ubuntu/course-
v3/nbs/dl1/data/camvid/images/seq05vd_f04110.png'),
3 | PosixPath('/home/ubuntu/course-
v3/nbs/dl1/data/camvid/images/0001TP_010170.png')]
```

```
1 | lbl_names = get_image_files(path_lbl)
2 | lbl_names[:3]
```

```
1 | [PosixPath('/home/ubuntu/course-
v3/nbs/dl1/data/camvid/labels/0016E5_01890_P.png'),
2 | PosixPath('/home/ubuntu/course-
v3/nbs/dl1/data/camvid/labels/seq05vd_f00330_P.png'),
3 | PosixPath('/home/ubuntu/course-
v3/nbs/dl1/data/camvid/labels/seq05vd_f01140_P.png')]
```

你可以看到有一些文件名编码过的文件，和一些文件名编码过的标注。你要找出它们是怎样对应的。通常，这种数据集会有一个README文件，或者你可以查看它们的网页。

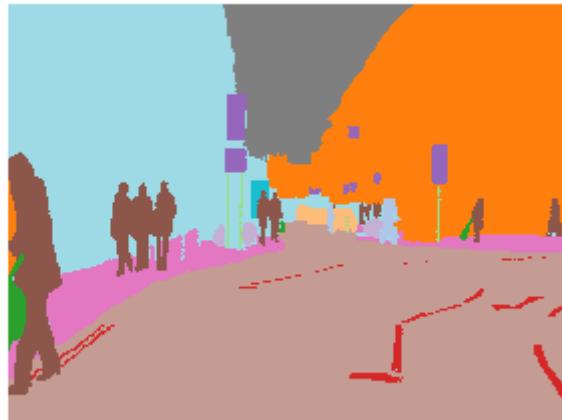
```
1 | img_f = fnames[0]
2 | img = open_image(img_f)
3 | img.show(figsize=(5,5))
```



```
1 | get_y_fn = lambda x: path_lbl/f'{x.stem}_P{x.suffix}'
```

通常很明显。这里，我猜是文件名 + `_P`，我创建了一个简单的函数，取文件名，然后加上 `_P`，放到另一个目录里 (`path_lbl`)，然后尝试打开它，成功了。

```
1 | mask = open_mask(get_y_fn(img_f))
2 | mask.show(figsize=(5,5), alpha=1)
```



我创建了这个把图片文件名转成对应的标签文件名的简单函数。把它打开，确认它可以用。通常，我们用 `open_image` 来打开文件，然后你可以用 `.show` 来查看它。但是，就像我们讲过的，这不是普通的图片文件，它包含的是数字。所以你要用 `open_masks` 而不是 `open_image`，因为我们要返回整数而不是浮点数。fastai知道怎样处理mask（遮罩），所以你执行 `mask.show` 时，它会自动添加颜色标注。这就是为什么我们使用 `open_masks`。

```
1 | src_size = np.array(mask.shape[1:])
2 | src_size,mask.data
```

```
1 | (array([720, 960]), tensor([[ [30, 30, 30, ..., 4, 4, 4],
2 |           [30, 30, 30, ..., 4, 4, 4],
3 |           [30, 30, 30, ..., 4, 4, 4],
4 |           ...,
5 |           [17, 17, 17, ..., 17, 17, 17],
6 |           [17, 17, 17, ..., 17, 17, 17],
7 |           [17, 17, 17, ..., 17, 17, 17]]]))
```

我们可以看看里面，看看`mask.data`，看下它的大小。是720*960。另外一个你可能已经注意到的东西是 `codes.txt` 文件和 `valid.txt` 文件。

```
1 | codes = np.loadtxt(path/'codes.txt', dtype=str); codes
```

```
1 | array(['Animal', 'Archway', 'Bicyclist', 'Bridge', 'Building', 'Car',
2 |        'CartLuggagePram', 'child', 'Column_Pole',
3 |        'Fence', 'LaneMkgsDriv', 'LaneMkgsNonDriv', 'Misc_Text',
4 |        'MotorcycleScooter', 'OtherMoving', 'ParkingBlock',
5 |        'Pedestrian', 'Road', 'RoadShoulder', 'Sidewalk', 'SignSymbol', 'sky',
6 |        'SUVPickupTruck', 'TrafficCone',
7 |        'TrafficLight', 'Train', 'Tree', 'Truck_Bus', 'Tunnel',
8 |        'VegetationMisc', 'Void', 'Wall'], dtype='<U17')
```

`code.txt` 包含一个列表，其中4是 `building`。就像我们做过的灰熊、黑熊、泰迪熊，这里我们可以看到每个像素编号的含义。

创建data bunch [1:05:53]

要创建一个data bunch，我们可以用data block API这样写：

- 我们从文件夹得到了一个图片文件的列表
- 然后我们要分割出训练集和验证集。这个例子里，我不会随机分，因为这些图片是从视频里截取出来的。如果随机选择，有可能会把两个相邻的截图一个放到训练集，一个放到验证集。这就太简单了，结果是不准确的。所以创建数据集的人提供了一个验证集文件列表（`valid.txt`），它们在视频里和训练集是不相连的。所以这里用一个文件名文件来区分验证集和训练集
- 我们要用 `get_y_fn` (get Y file name function) 函数创建标签

```
1 | size = src_size//2
2 | bs=8
```

```
1 | src = (SegmentationItemList.from_folder(path_img)
2 |         .split_by_fname_file('../valid.txt')
3 |         .label_from_func(get_y_fn, classes=classes))
```

这样，我可以创建dataset。

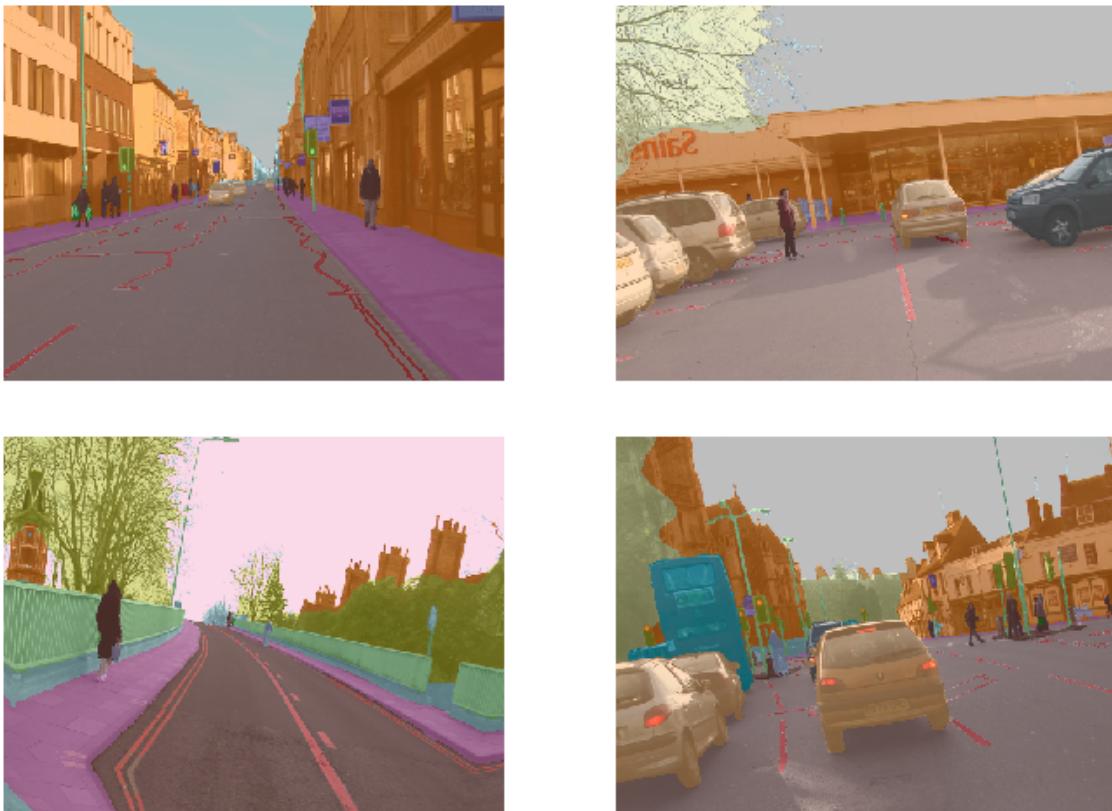
我有了一个类别名的列表。对于planet或者宠物数据集，我们会得到一个字符串来表示这是哈巴狗、这是布娃娃猫、这是巴曼猫或者这是多云等等。但是对这个数据集来说，不能每一个像素都用一个完整的字符串来标记，这会非常低效。每个像素是用一个数字来标记的，有一个文件来存储每个数字的含义。这里我们告诉data block API这是代表数字含义的列表。

```
1 | data = (src.transform(get_transforms(), size=size, tfm_y=True)
2 |         .databunch(bs=bs)
3 |         .normalize(imagenet_stats))
```

这是变形对象。这是一个有意思的地方。记得我讲过有时我们随机反转一个图片吗？如果我们仅随机反转自变量图片、而不同时反转目标遮罩会怎样？这样就没法再对应上。所以我们需要让程序对Y做变形（X是自变量，Y是因变量）。所以我们对X做了什么，也要对Y做什么（`tfm_y=True`）。这就是我们需要设置的参数。

我可以创建我们的data bunch。我使用了一个比较小的batch size (`bs=8`)。因为，你可以想象的到，我们在创建一个对每个像素的分类的程序，这会使用非常多的GPU。8是能处理的最大值。然后像以前一样做标准化。

```
1 | data.show_batch(2, figsize=(10,7))
```



这很好。因为fastai知道你给了它一个分割问题，在你调用show batch时，它把两部分结合起来，把图片标上颜色。很不错吧？这就真实数据的样子。

训练 [1:09:00]

得到这个后，我们可以继续。

- 创建一个learner。晚些我会演示更多细节。
- 调用`lr_find`，找到最陡的部分，看起来大概是 $1e-2$ 。
- 调用`fit`，传入`slice(lr)`，看看准确率。
- 保存模型。
- 解冻，再训练一次。

这就是基本的思路。

```

1 name2id = {v:k for k,v in enumerate(codes)}
2 void_code = name2id['Void']
3
4 def acc_camvid(input, target):
5     target = target.squeeze(1)
6     mask = target != void_code
7     return (input.argmax(dim=1)[mask]==target[mask]).float().mean()
```

```

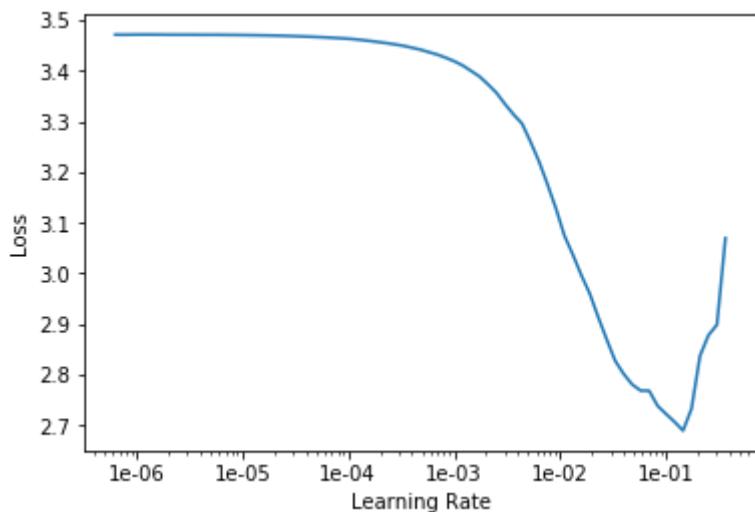
1 metrics=acc_camvid
2 # metrics=accuracy
```

```

1 learn = Learner.create_unet(data, models.resnet34, metrics=metrics)
```

```

1 lr_find(learn)
2 learn.recorder.plot()
```



```
1 | lr=1e-2
```

```
1 | learn.fit_one_cycle(10, slice(lr))
```

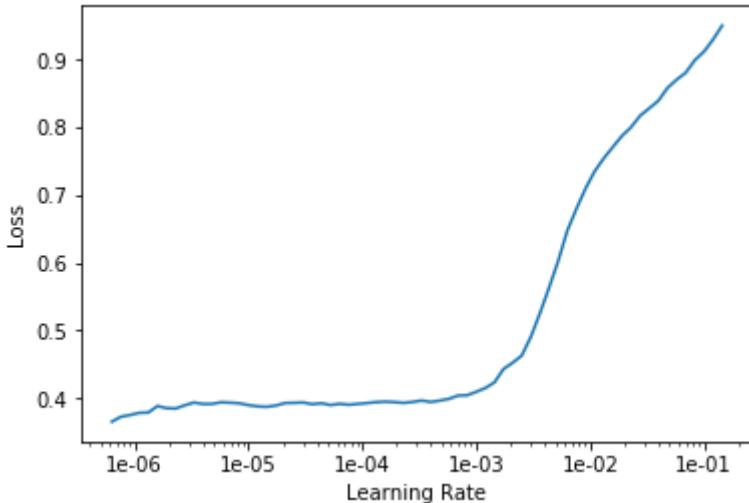
```
1 | Total time: 02:46
2 | epoch  train_loss  valid_loss  acc_camvid
3 | 1      1.537235   0.785360   0.832015   (00:20)
4 | 2      0.905632   0.677888   0.842743   (00:15)
5 | 3      0.755041   0.759045   0.844444   (00:16)
6 | 4      0.673628   0.522713   0.854023   (00:16)
7 | 5      0.603915   0.495224   0.864088   (00:16)
8 | 6      0.557424   0.433317   0.879087   (00:16)
9 | 7      0.504053   0.419078   0.878530   (00:16)
10 | 8     0.457378   0.371296   0.889752   (00:16)
11 | 9     0.428532   0.347722   0.898966   (00:16)
12 | 10    0.409673   0.341935   0.901897   (00:16)
```

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

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

```
1 | learn.unfreeze()
```

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



```
1 | lrs = slice(1e-5, lr/5)
```

```
1 | learn.fit_one_cycle(12, lrs)
```

```

1 | Total time: 03:36
2 | epoch  train_loss  valid_loss  acc_camvid
3 | 1      0.399582   0.338697   0.901930   (00:18)
4 | 2      0.406091   0.351272   0.897183   (00:18)
5 | 3      0.415589   0.357046   0.894615   (00:17)
6 | 4      0.407372   0.337691   0.904101   (00:18)
7 | 5      0.402764   0.340527   0.900326   (00:17)
8 | 6      0.381159   0.317680   0.910552   (00:18)
9 | 7      0.368179   0.312087   0.910121   (00:18)
10 | 8     0.358906   0.310293   0.911405   (00:18)
11 | 9     0.343944   0.299595   0.912654   (00:18)
12 | 10    0.332852   0.305770   0.911666   (00:18)
13 | 11    0.325537   0.294337   0.916766   (00:18)
14 | 12    0.320488   0.295004   0.916064   (00:18)

```

提问: 这里能不能用无监督学习来避免人工标注很多图片? [\[1:10:03\]](#)

不能完全用无监督学习,但是你能看出,这些标签不是必需的。时间允许的话,我们会尝试演示几个不需要标注来做分割的例子。不用人工的话,得不到这样高质量的标注。如果你想要有这样水平的分割遮罩,你需要有很好的分割标注技能。

提问: 是否可以不先用小尺寸 (64x64) 的图片训练,再在这个基础上微调,来适应大尺寸 (128x128, 256x256) 图片? [\[1:10:51\]](#)

你应该先用小尺寸训练再在这个基础上训练大图片。这个方法非常有效。这个方法是我在几年前的课程上首先提出的,我觉得这很明显,仅仅是把它当作一个好的方法,后来我发现没有人在我之前发表过这个方法。然后我们对这个方法做了实验。它基本上是我们帮赢得DAWN Bench ImageNet训练竞赛的主要trick。

这不是标准方法,而且没有人听说过这种方法。现在有一些论文在很多课题上使用这个trick,但是仍然很少有人知道它。用它你可以训练得更快,泛化效果更好。现在仍然不太清楚怎样是最好的,比如用多大的图片、在每阶段用多少张等等。我们把它叫做“渐进式调整尺寸”(progressive resizing)。我发现小于64x64时帮助不大。然而,这是一个很棒的技术,我一定会尝试几个不同的尺寸。

提问: [\[1:12:35\]](#) 对像素分割来说准确率代表什么? 是不是

#正确分类的像素数量 / #总的像素数量?

是的，就是这样。如果你想象每个像素都是一个要分类的对象，就是这样计算准确率。事实上，你可以直接传入accuracy作为度量，但我们在那里用了一个新的叫做acc_camvid的度量指标，因为它们在标注图片时，会把有的像素标记成void。我不太清楚为什么，但确实有些像素是void。在CamVid论文里，他们说，在报告准确率时，应该去除void像素。所以我们创建了accuracy CamVid方法。每个度量都接收神经网络的实际输出（度量的输入）和目标值（我们尝试预测的标签）。

```
name2id = {v:k for k,v in enumerate(codes)}
void_code = name2id['Void']

def acc_camvid(input, target):
    target = target.squeeze(1)
    mask = target != void_code
    return (input.argmax(dim=1)[mask]==target[mask]).float().mean()
```

我们创建了一个mask（我们寻找不是void的位置），然后取到输入，像往常一样做argmax，仅取出不是void的值。对目标值做相同的事，然后取平均值，所以这是一个标准的准确率。

这基本和我们之前看到的准确率代码很像，只是多了mask。这很常见。对Kaggle竞赛里指定的度量或者你所在的组织的评分标准，你都需要做些小的调整。就像这个一样，这很简单。做这样的东西最主要的是，要清楚地知道怎样在PyTorch里做基础的数学计算，这是你需要练习的东西。

提问：我注意到大多数例子和大多数我的模型的结果都是训练集损失度大于验证集损失度。修正这个最好的方法是什么？我尝试了很多epoch数量和学习率后，还是这样。[\[1:15:03\]](#)

如果你的训练集损失度比验证集损失度高，那就是欠拟合。它一定是代表欠拟合。你希望你的训练集损失度小于验证集损失度。如果你欠拟合了，你可以：

- 训练更久
- 用小些的学习率训练最后的部分

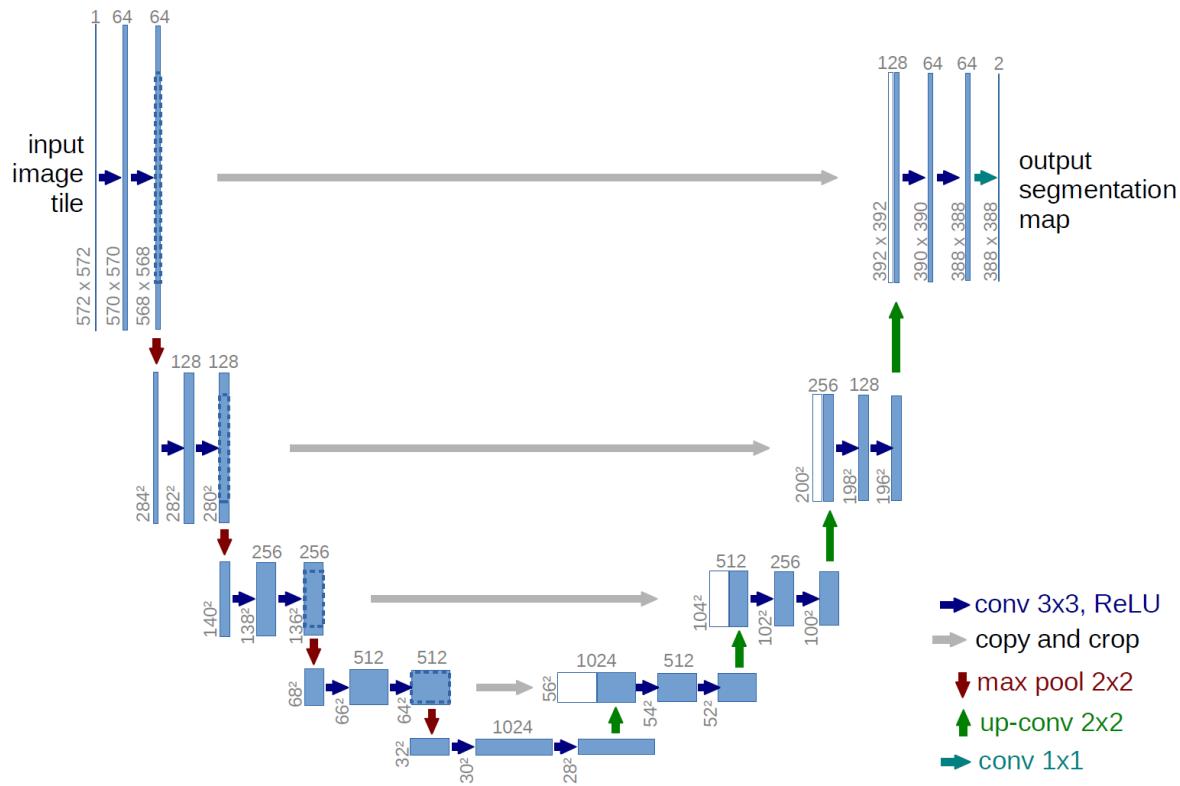
如果仍然欠拟合，你需要减少正则化。我们还没有讲过这个。在后半部分，我们会讲很多关于正则化的内容和怎样用正则化来避免过拟合和欠拟合。我们会学习：

- 权重衰减 (weight decay)
- 随机失活 (dropout)
- 数据增强 (data augmentation)

这些是我们要学习的主要内容。

U-Net [\[1:16:24\]](#)

对于分割问题，我们没有用一个卷积网络，我们可以用它，但是用叫U-Net的网络架构效果会更好。



这是U-Net的样子。它来自这个讲U-Net的[大学网站](#)。我们课程的第一部分和第二部分都会学习它。左边向下的部分是一个普通的卷积网络。它有时从一个大图片开始，然后逐步缩小，最后仅有一个预测值。U-Net做的是再把它逐步变大，把向下时每一步的值拷贝到对面，这样就出现了一个U型。

它最初是作为一个生物图片分割方法被创建和发表的。但是它后来被发现其他领域也很有效。它是在MICCAI被提出的，这是一个主要的医学图像会议，就在昨天，它成为了这个会议最多被引用的论文。它的效果难以置信——它有超过3,000个引用。

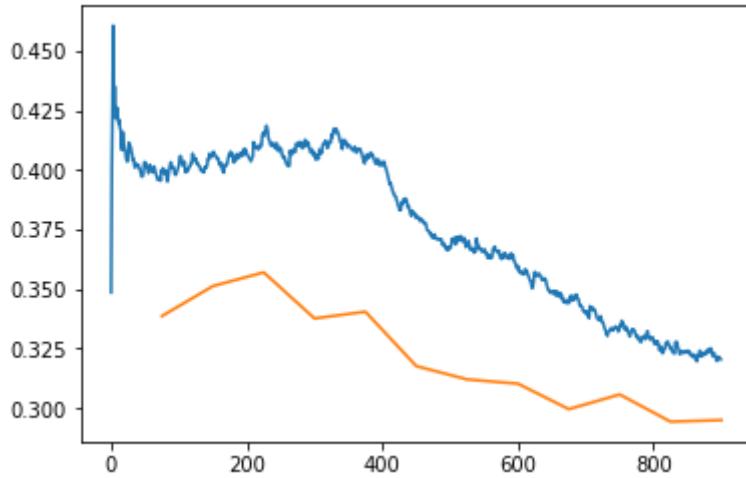
现在你不必知道这些细节。你需要知道的就是如果你想创建一个分割模型，你应该用 `Learner.create_unet`，而不是 `create_cnn`。不过，要传递的参数和以前一样：`data bunch`、架构和度量指标都是一样的。

做完这些，其他的东西都是一样的。

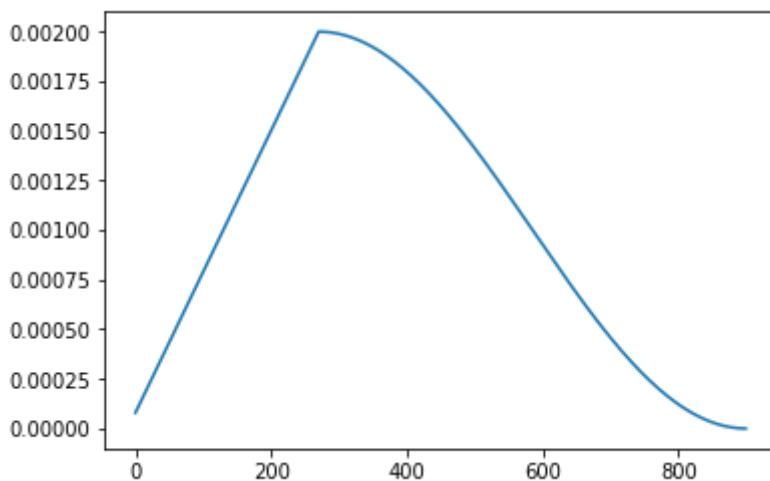
一些其它的关于 `learn.recorder` 的内容 [1:18:54]

这里有些有意思的东西。`learn.recorder` 是我们追踪训练情况的工具。它有一些很好的方法，其中一个是一个 `plot_losses`。

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



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



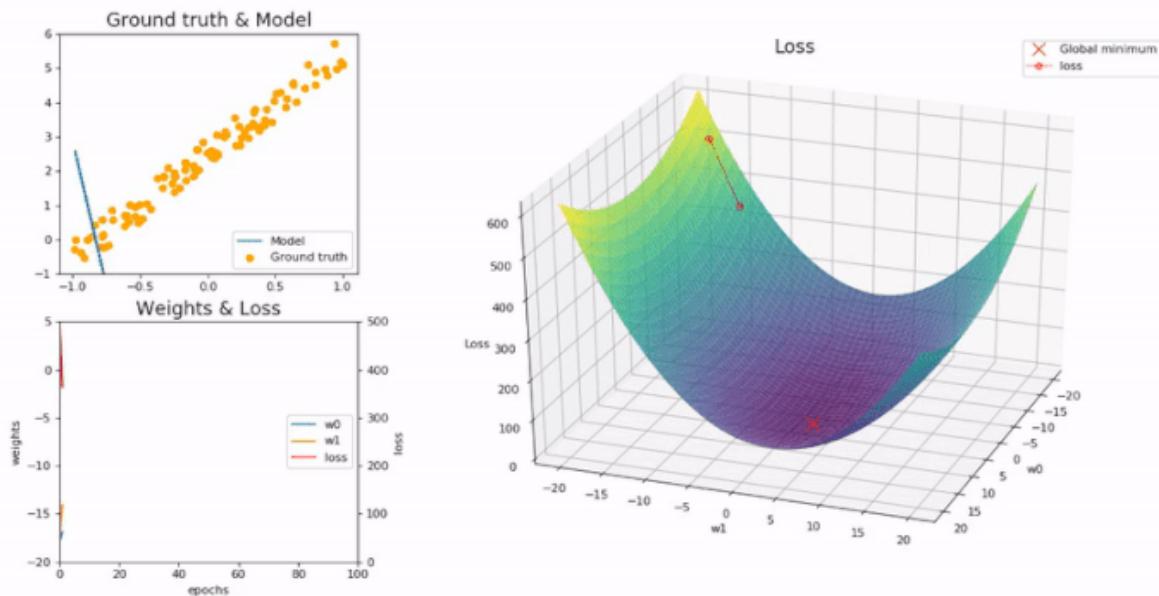
这里画出了你的训练集损失度和验证集损失度。通常，它们会先上升一下，然后再下降。为什么？这是因为学习率先上升后下降。这又是为什么？因为我们用了 `fit_one_cycle`。这就是fit one cycle方法做的事，它让学习率先缓慢上升，再下降。

为什么这是一个好方法？要知道为什么这是一个好方法，让我们先看下José Fernández Portal这周做的[一个很酷的项目](#)。他用了我们的梯度下降demo notebook，画出了真实值和模型的对照，画出了权重的变化。用了几个不同的学习率。

我们有两个参数： $y = ax + b$ ，或者按他的命名是 $y = w_0x + w_1$

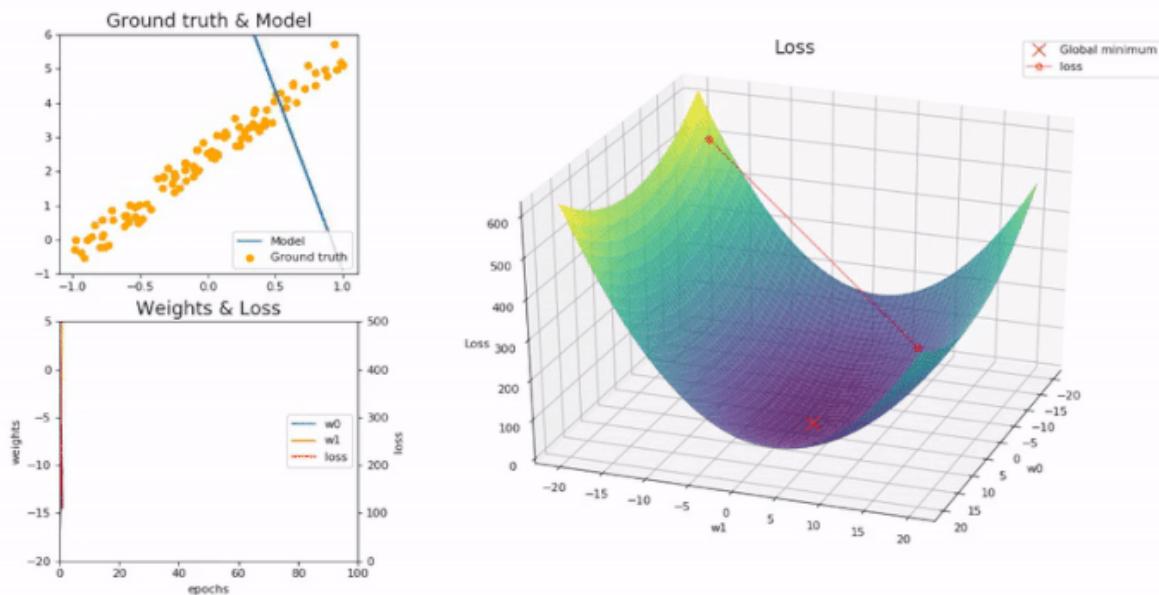
我们可以看看这些参数是怎样随着时间变化的。我们知道这是正确的答案（用红色X标记）。学习率设成0.1时，它是这样的，花了一些时间到了正确的位置。你可以看到损失度在降低。

lr: 0.1 - Epoch: 2/100



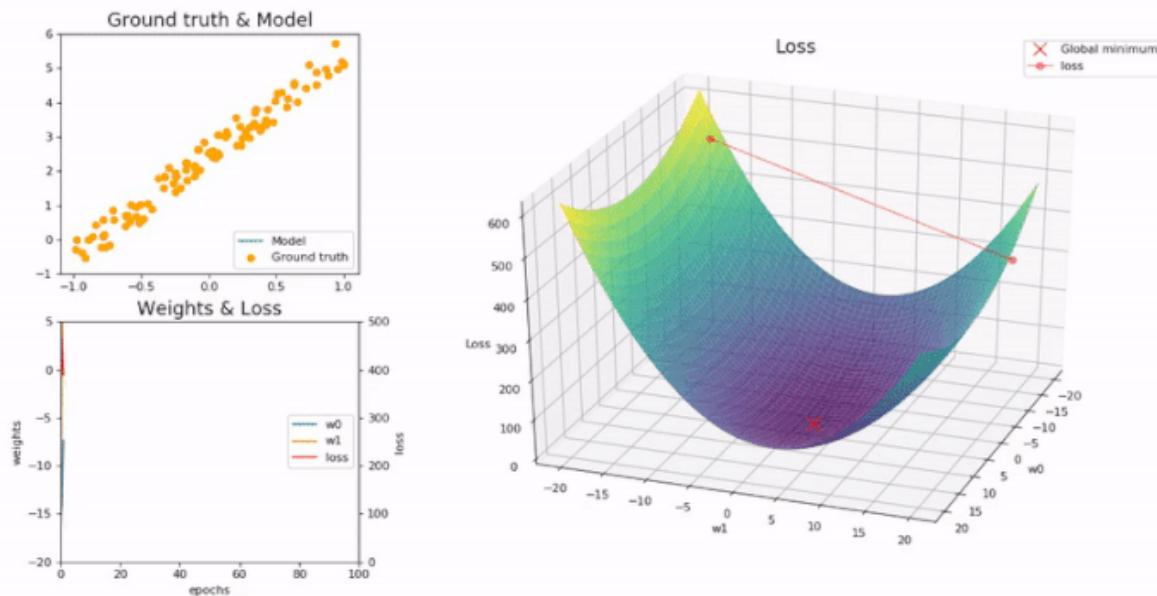
学习率是0.7时，你可以看到模型很快跳到了正确位置。可以看到参数直接跳到了正确位置。

lr: 0.7 - Epoch: 2/100



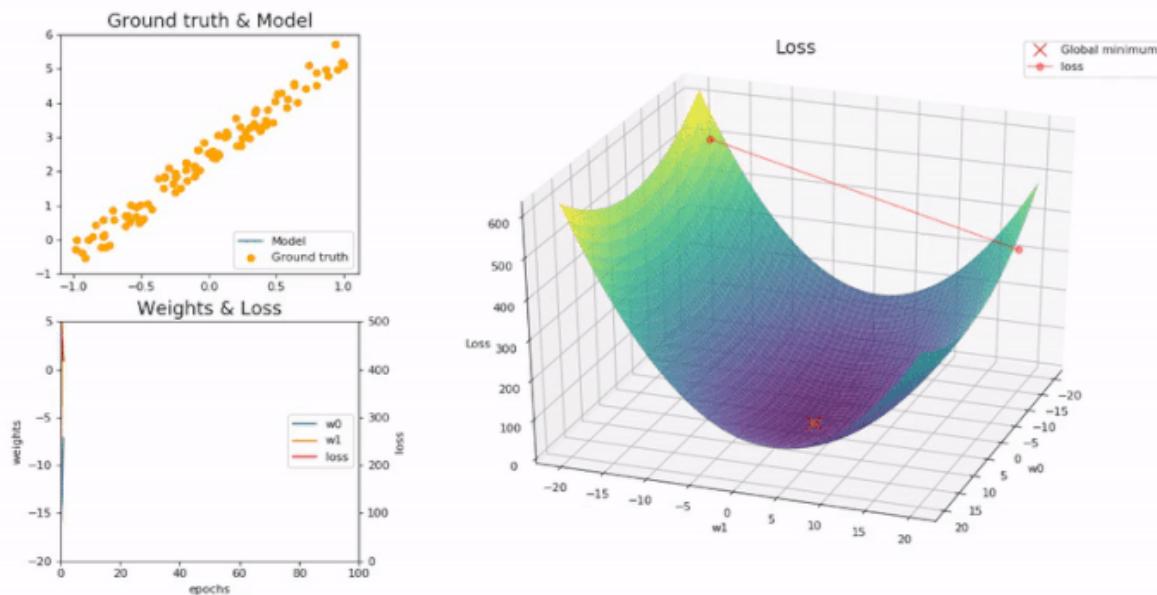
如果用一个太高的学习率会怎样？你可以看到它用了非常长时间来找到正确位置。

lr: 0.99 - Epoch: 2/100



如果学习率非常高，它会发散（diverge）。

lr: 1.01 - Epoch: 2/100



可以看到，使用正确的学习率是很重要的。如果你使用正确的学习率，它会很快找到最好位置。

随着你接近最小值，会发生有意思的事情，你希望你的学习率变小，因为你离正确位置很近了。

事实上，你通常不会有这种损失度曲线（记住，会有很多维度。但我们只能画出二维），它会像这个一样颠簸。所以你需要一个这样足够高的学习率来越过颠簸，但是一旦你接近最优值，你不再希望在颠簸间来回跳动。你希望你的学习率变小，这样随着你越来越接近最优值，你的每一步越来越小。这就是为什么我们希望学习率最后降下来。

在训练时降低学习率的方法存在很久了。它被称作**学习率退火 (learning rate annealing)**。但是在训练开始时逐渐增大学习率是最近被提出的，它来自于Leslie Smith([看看Leslie Smith](#))。

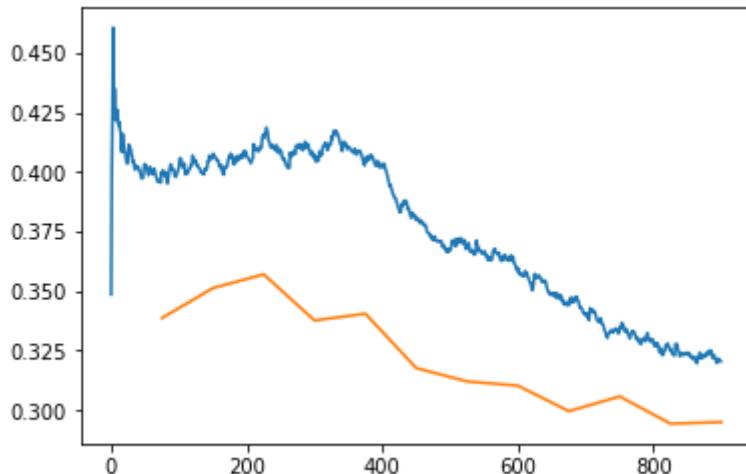
损失度曲线会有平坦的部分和曲折的部分。如果你在曲折部分的底部停下来，这个模型不会泛化得很好，因为你找到了一个在这个地方比较好但在其他地方不够好的方案。如果你在平坦的部分找到了解，它可能会泛化的比较好，因为它不仅在一个点表现好，也会在附近表现比较好。

如果你用一个很小的学习率，它会掉进来被困在这里。如果你逐渐增大学习率，虽然它会掉进去，但随着学习率变大，它会像这样再回到上面。然后学习率变成这样，它会来回跳动。最终学习率渐渐变小，它会找到一条来到这个平坦部分的路。

逐渐增大学习率是一个帮助模型探索整个函数曲线，找到平坦的、损失度低的区域的很好的方法。如果它是曲折的，它会被再弹出来。这让我们能用比较高的学习率训练，这意味着我们求解问题的速度能快很多，并且能找到泛化性更好的解。

怎样使用 plot_losses [1:25:01]

如果你调用 `plot_losses`，发现损失度先是变差了一点，然后变好了很多，这说明你找到了一个很好的学习率最大值。



当你调用fit one cycle时，你传入的其实不是一个学习率，而是一个学习率的最大值。如果它一直在降低，尤其是在你解冻之后，这表明你大概要把学习率调大一些，因为你需要看到这种形状。它会训练地更快、泛化能力更好。你大概会在验证集上看到这个（橙色的是验证集）。从知道理论到能够应用它，要大量地看这些图片。所以在你训练完后，获得比较好的结果时，输入 `learn.recorder.`，点击tab键，看看这里有什么，来培养对这些图形的感觉。然后试下把学习率调高，试下把学习率调低，用更多的迭代，用更少的迭代，感受下这时它们的图形是什么样的。

放大 [1:26:16]

```
size = src_size//2  
bs=8
```

这个例子里，我们在变形里用了这个尺寸 `原始图片尺寸/2`。两个斜杠在Python里表示整除，显然，尺寸里不会出现半个像素。我把batch size设成8。现在我发现它在我的CPU可以运行，但它在你的GPU上不一定适用。如果在你的GPU上运行不了，你可以把batch size减到4。

这没有完全解决问题，我们的问题是分割所有的像素，不是一半的像素。我们使用和上次相同的方法，现在我要把尺寸放大到完整图片的尺寸，这意味着我把batch size减半，不然会导致内存溢出。

```
1 | size = src_size  
2 | bs=4
```

```
1 | data = (src.transform(get_transforms(), size=size, tfm_y=True)  
2 |         .databunch(bs=bs)  
3 |         .normalize(imagenet_stats))
```

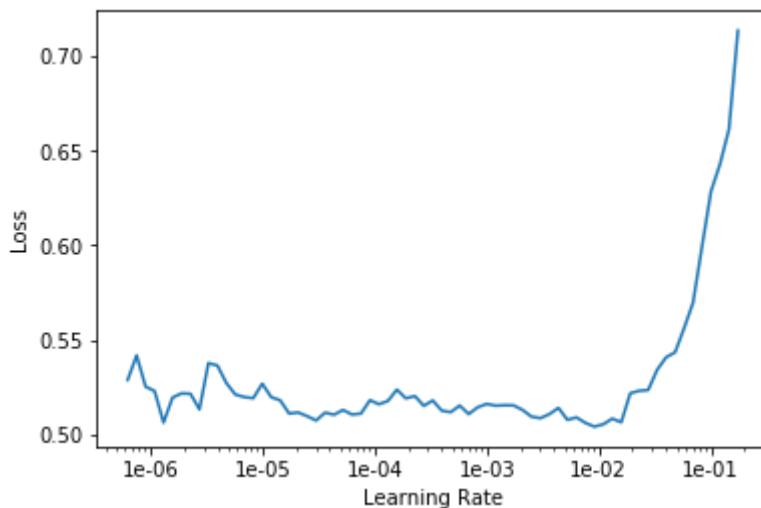
```
1 | learn = Learner.create_unet(data, models.resnet34, metrics=metrics)
```

```
1 | learn.load('stage-2');
```

我也可以用 `learn.data = data`，但是我发现这会遇到很多GPU内存问题，所以我重启notebook内核，回到这里，创建一个新的learner，加载上次保存的权重。

关键的地方是，现在这个learner的权重和我们之前保存的是一样的，但数据是完整尺寸的图片。

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



```
1 | lr=1e-3
```

```
1 | learn.fit_one_cycle(10, slice(lr))
```

```
1 | Total time: 08:44
2 | epoch  train_loss  valid_loss  acc_camvid
3 | 1      0.454597   0.349557   0.900428   (01:02)
4 | 2      0.418897   0.351502   0.897495   (00:51)
5 | 3      0.402104   0.330255   0.906775   (00:50)
6 | 4      0.385497   0.313330   0.911832   (00:51)
7 | 5      0.359252   0.297264   0.916108   (00:52)
8 | 6      0.335910   0.297875   0.917553   (00:50)
9 | 7      0.336133   0.305602   0.913439   (00:51)
10 | 8     0.321016   0.305374   0.914063   (00:51)
11 | 9     0.311554   0.299226   0.915997   (00:51)
12 | 10    0.308389   0.301060   0.915253   (00:51)
```

```
1 | learn.save('stage-1-big')
```

```
1 | learn.load('stage-1-big');
```

```
1 | learn.unfreeze()
```

```
1 | lrs = slice(1e-6,lr)
```

```
1 | learn.fit_one_cycle(10, lrs, wd=1e-3)
```

```
1 | Total time: 09:30
2 | epoch  train_loss  valid_loss  acc_camvid
3 | 1      0.323283   0.300749   0.915948   (00:56)
4 | 2      0.329482   0.290447   0.918337   (00:56)
5 | 3      0.324378   0.298494   0.920271   (00:57)
6 | 4      0.316414   0.296469   0.918053   (00:56)
7 | 5      0.305226   0.284694   0.920893   (00:57)
8 | 6      0.301774   0.306676   0.914202   (00:57)
9 | 7      0.279722   0.285487   0.919991   (00:57)
10 | 8     0.269306   0.285219   0.920963   (00:57)
11 | 9     0.260325   0.284758   0.922026   (00:57)
12 | 10    0.251017   0.285375   0.921562   (00:57)
```

```
1 | learn.save('stage-2-big')
```

```
1 | learn.load('stage-2-big')
```

```
1 | learn.show_results()
```



你可以执行 `learn.show_results()` 看看预测结果，和真实值对比，看起来相当不错。

好到什么程度呢？准确率是92.15%，我知道的研究分割问题的最好的论文是 [The One Hundred Layers Tiramisu](#)，它在大概两年前提出了一种卷积dense net。在我今天训练完这个之后，我回去看了这个论文，发现它们的最高准确率是91.5%，我们得到了92.1%。我不知道在这篇论文发表后有没有更好的结果，但我记得这篇论文在发表时，产生了很大影响。这是一个非常好的分割成绩。如果你拿它和之前的最好成绩比较，它取得了很大的进步。

在去年的课程里，我们花了很多时间复现那篇论文。现在，我们只使用fastai里默认的类，轻松地打败了91.5%的成绩。我还记得之前要几个小时几个小时地训练。今天，我只训练了几分钟。所以这对分割问题是一个很强大的架构。

我不是说这就是现在的最好水平，我没有做过一个彻底的学术搜索来查查过去两年有没有新的记录。但它确实超过了之前的记录。这就是我们用到的改进模型的一些trick：预训练和one cycle convergence。这些trick都很有效。

我们还没有发表关于这个U-Net变种的细节的论文，我们用了一些小技巧。如果你参加part 2的课程，你会学到所有的使它取得这么好成绩的技术细节。现在你只需要调用 `learner.create_unet`，也可以取得很好的结果。

另一个trick：混合精度训练（Mixed precision training） [1:30:59]

如果你经常遇到内存溢出问题，有一个技巧来处理。你可以使用混合精度训练。混合精度训练是指，你可以使用半个精度浮点数来做模型计算，而不是使用单精度浮点数，就是使用16位，而不是32位。这个方法是最近两年才提出的。一些硬件可以用它执行得很快。我觉得fastai库第一个，也可能仍然是唯一一个能很方便地使用这个方法的库。

```
learn = Learner.create_unet(data, models.resnet34, metrics=metrics).to_fp16()
```

如果你把 `to_fp16()` 加在任何learner调用的方法的后面，你可以得到一个用16位精度来训练的模型。因为这是一个新技术，你需要使用最新的CUDA驱动和一些其它东西才能使用它。今天上午我在一些平台上执行它时，它会让kernel无法工作。所以你需要用最新的驱动。如果你用最新的GPU，比如说2080Ti，就可以使用16位精度，能比以前快上一倍。它会使用更少的GPU内存。即使你没有2080Ti，你也可能会发现，原来在你的GPU上运行不了的代码，在使用这个方法后，可以运行了。

我没看到过有人在分割问题上用混合精度浮点。只是想随便试下，结果得到了更好的成绩。我今天早上才发现这个，如果使用更小的浮点精度，它可以更好地泛化。我还没有见过在CamVid数据集能得到92.5%的准确率。它不仅会变快，你还能使用更大的batch size，你可能还能像我一样得到更好的结果。这是一个很酷的小trick。

你只需要保证每次创建learner后加上 `to_fp16()`。如果你的kernel被卡死了，这大概表明，你用的CUDA驱动版本太低了，或者你的显卡太旧了。我也不是特别清楚哪些显卡支持FP16。

BIWI头部姿势数据集回归 [1:34:03]

[lesson3-head-pose.ipynb](#)

再讲两个内容。第一个要演示的是一个有趣的数据集，它是 [BIWI头部姿势数据集](#)。Gabriele Fanelli很友好，授权我们在课程里使用这个数据集。他的团队创建了这个很酷的数据集。



这是数据集里的内容。里面有几项内容。我们只做简化的版本，他们在脸部中心标记一个点。所以我们要尝试创建一个在脸上找到这个点的模型。

```
1 %reload_ext autoreload  
2 %autoreload 2  
3 %matplotlib inline
```

```
1 from fastai import *  
2 from fastai.vision import *
```

对这个数据集，有些东西要处理，我也不是特别清楚其中的细节，仅仅是在readme文件里了解到这些内容，你也需要读下这个。他们使用了深度摄像机，我猜他们用的是Xbox Kinect。

```
1 path = untar_data(URLs.BIWI_HEAD_POSE)
```

他们在一个文件里提供了一些测量数据，我们需要读取出来：

```
1 cal = np.genfromtxt(path/'01'/'rgb.cal', skip_footer=6); cal
```

```
1 array([[517.679,    0.,   320.],
2        [    0., 517.679, 240.5],
3        [    0.,    0.,    1.]])
```

```
1 fname = '09/frame_00667_rgb.jpg'
```

```
1 def img2txt_name(f): return path/f'{str(f)[:-7]}pose.txt'
```

```
1 img = open_image(path/fname)  
2 img.show()
```



```
1 | ctr = np.genfromtxt(img2txt_name(fname), skip_header=3); ctr
```

```
1 | array([187.332 , 40.3892, 893.135 ])
```

他们提供一个函数，你要用它把相机坐标转换成图片中的坐标。

```
1 | def convert_biwi(coords):
2 |     c1 = coords[0] * cal[0][0]/coords[2] + cal[0][2]
3 |     c2 = coords[1] * cal[1][1]/coords[2] + cal[1][2]
4 |     return tensor([c2,c1])
5 |
6 | def get_ctr(f):
7 |     ctr = np.genfromtxt(img2txt_name(f), skip_header=3)
8 |     return convert_biwi(ctr)
9 |
10 | def get_ip(img,pts): return ImagePoints(FlowField(img.size, pts),
scale=True)
```

你看到的这些转换流程，都是他们要我们做的。这和我们用深度学习来找出这个点的工作没什么关系。

```
1 | get_ctr(fname)
```

```
1 | tensor([263.9104, 428.5814])
```

```
1 | ctr = get_ctr(fname)
2 | img.show(y=get_ip(img, ctr), figsize=(6, 6))
```



有趣的是，我们做的不是一个图片，不是一个对图片的分割，而是一个图上的一个点。我们会在后面学习关于这个的更多内容，图片点使用了坐标的概念。它们不是像素值，而是XY坐标(只有两个数字)。

这是一个图片文件的例子(`09/frame_00667_rgb.jpg`)。脸部中心的坐标是`[263.9104, 428.5814]`。只有两个数字就可以表示脸部中心的位置。如果我们要创建一个找到脸部中心的位置的模型，我们需要一个能找出两个数字的神经网络。注意，这不是一个分类模型。这两个数不是代表它们是一个列表中的道路、还是建筑、还是布偶猫或者其他的东西。它们是位置。

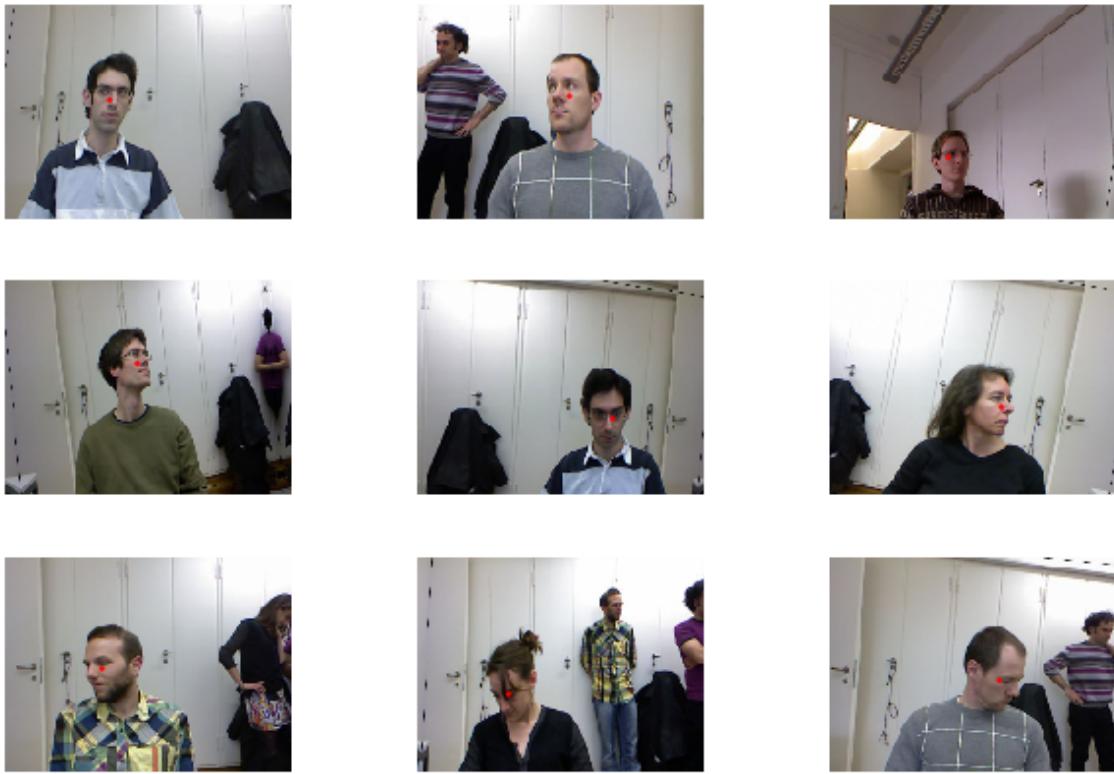
目前为止，我们做的都是分类模型，用来创建标签或者类别。这个例子是第一次创建一个回归模型。很多人认为回归代表线性回归，不是这样。回归代表你的模型的结果是一个或多个连续值。所以我们要创建一个图像回归模型。怎样做呢？和之前的一样。

```
1 | data = (ImageItemList.from_folder(path)
2 |         .split_by_valid_func(lambda o: o.parent.name=='13')
3 |         .label_from_func(get_ctr, label_cls=PointsItemList)
4 |         .transform(get_transforms(), tfm_y=True, size=(120,160))
5 |         .databunch().normalize(imagenet_stats)
6 |     )
```

我们只需要这样：

- 取一个图片文件列表 (`ImageItemList`)
- 它存在一个文件夹里
- 用某个函数来划分训练集和验证集。这个例子里，这些文件来自于视频。我只用一个文件夹 (`13`) 作为验证集(这是另外一个人的图片)。我思考过怎样算是公正的验证，公正的验证是保证它在一个没有见过的人的图片上可以很好地运行。所以我们的验证集是一个没有出现在训练集里的人。
- 我要用这个函数来标记，它按照readme里的方法从文本文件里读取坐标。它为每个人返回两个数字(坐标)
- 创建一个dataset。这个数据集是一组代表坐标的点。
- 做变形。我需要用 `tfm_y=True`，因为如果我反转或者旋转了图片，红点的位置也会变化。
- 选择尺寸。我选了一个可以很快运行的尺寸
- 创建data bunch
- 标准化

```
1 | data.show_batch(3, figsize=(9,6))
```



我注意到他们的红点不是总是在脸部的中间。我不知道他们画点的算法是什么。有时好像是选择鼻子，有时并不是。无论如何，都是在脸部中心或者鼻子附近的区域里。

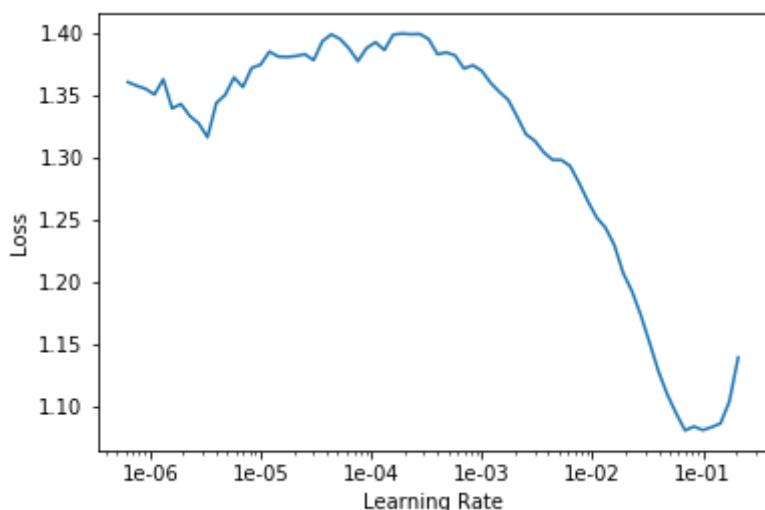
创建回归模型 [1:38:59]

怎样创建模型呢？我们创建一个CNN。后面的课程里我们会学习很多关于损失函数的内容。损失函数是衡量模型效果的函数。对于分类问题，我们使用交叉熵 (cross-entropy) 损失，它代表“你有没有预测出正确结果，你对这个有多自信？”。我们不能用它来做回归，我们要使用均方差 (mean squared error, MSE)。上节课，我们从头实现了MSE。它就是两个值的差，做平方，然后加在一起。

```
1 | learn = create_cnn(data, models.resnet34)
2 | learn.loss_func = MSELossFlat()
```

我们要告诉模型这不是在做分类，我们要使用均方误差做损失函数。

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



```
1 | lr = 2e-2
```

```
1 | learn.fit_one_cycle(5, slice(lr))
```

```
1 | Total time: 07:28
2 | epoch  train_loss  valid_loss
3 | 1      0.043327   0.010848   (01:34)
4 | 2      0.015479   0.001792   (01:27)
5 | 3      0.006021   0.001171   (01:28)
6 | 4      0.003105   0.000521   (01:27)
7 | 5      0.002425   0.000381   (01:29)
```

创建learner后，我们告诉它要用哪个损失函数，然后继续，执行`lr_find`，然后执行`hen fit`，你可以在这里看到在1.5分钟里，我们得到了0.0004的均方误差。

均方误差很好解释。我们要做的预测是在几百个点中选一个，我们得到了错误的平方值的平均数是0.0004。我们感到很自信，这是一个很好的模型。下面可以看看结果：

```
1 | learn.show_results()
```



这几乎是完美的结果。这就是做图片回归模型的方法。在你需要预测连续值时，你就可以用这样的方法。

IMDB [1:41:07]

[lesson3-imdb.ipynb](#)

在学习基础理论前的最后一个例子，是NLP的。下周，我们会学习很多关于NLP的内容。现在让我们再做一遍相同的事，不过这次不是分类图片，而是分类文档。下周我们会讲很多细节，这周先快速做一遍。

```
1 %reload_ext autoreload
2 %autoreload 2
3 %matplotlib inline
```

```
1 from fastai import *
2 from fastai.text import *
```

不再是引用 `fastai.vision`，这次我们引用 `fastai.text`。这里面你可以找到所有的专门用来分析文本文档的东西。

这个例子里，我们要用一个叫IMDB的数据集。IMDB里有很多电影评论。一般是两三千个单词。每个评论被分为正面的或者负面的。

```
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')]
```

它就是一个CSV文件。我们可以使用pandas来读取它，简单看一下。

```
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

```
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.
>
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.
>
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.
>
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.
>
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".
>
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.'

还是像之前一样，我们可以使用工厂方法或者data block API来创建data bunch。这是一个从CSV文本文件中创建data bunch的快速方法。

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

现在我可以创建一个learner，开始训练。先看一些细节，我们下周会讲更多。你创建data bunch时，它实际上做了几步：

1. **分词 (Tokenization)** : 它处理这些单词，把它们转成标准的token (分词)。基本上每个token代表一个单词。

```
data = TextClasDataBunch.load(path)
data.show_batch()
```

	text	label
xxfld 1 raising victor vargas : a review\n\nyou 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 , no matter how warm and gooey raising victor vargas became i was always aware that something didn 't quite feel right . victor vargas suffers from a certain xxunk on the director 's part . apparently , the director thought that the ethnic backdrop of a latino family on the		negative
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\nin it 's usually satisfying to watch a film director change his style / subject , but soderbergh 's most recent stinker , the girlfriend xxunk) , was also missing a story , so narrative (and editing ?) seem to suddenly be soderbergh 's main challenge . strange , after xxunk years		negative
xxfld 1 many xxunk that this is n't just a classic due to the fact that it 's the first 3d game , or even the first xxunk - up . it 's also one of the first xxunk games , one of the xxunk definitely the first) truly claustrophobic games , and just a pretty well - xxunk gaming experience in general with graphics that are terribly dated today . the game xxunk will		positive

它做了一些这样的事情，看到没有，“didn't”被转成里两个独立的单词（`did` 和 `n't`）。并且所有的内容都被转成小写了。看到“you're”被转成两个独立的单词（`you` 和 `'re`）了吗？分词（Tokenization）是要让每个“token”（每个词两边都有空白）代表一个独立的语义。它遇到特别少见的单词（比如说名字）时，会把它们用一个叫`unknown`（`xxunk`）专门的token替代。在fastai里所有以`xx`开头的都是特殊token。这就是Tokenization，所以我们最后可以得到一个被分词过的单词的列表。你也能看到标点周围也被加了空格，来保证它们是单独的token。

2. **编号 (Numericalization)**：下一步是为所有可能的token创建一个完整的唯一的列表，这被叫做 `vocab` 单词表。

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

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

这是所有电影评论里最常出现的token（前十个）。然后我们用一个数字列表来替代每个电影评论。

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

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

这个列表里的数字代表这个位置上的分词在单词表里的序号。

分词和编号，是NLP里把文档转成一个数字list的标准方式。

我们用data block API：

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

这次，不再用`ImageFilesList`，而是从CSV创建一个TextList，创建一个data bunch。这时我们可以开始创建一个模型了。

下周会讲到，做NLP分类时我们会创建两个模型：

1. 第一个模型叫 **语言模型 (language model)** 我们用普通的方式训练

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

我们创建一个language model learner，训练它，保存它，解冻，再训练。

2. 在我们创建语言模型之后，我们创建 **分类器 (classifier)**。我们创建classifier的数据bunch，创建一个learner，训练它，得到准确率。

这里快速讲下。下周我们会讲更多细节。你可以看到训练NLP分类器的基本思路和我们之前的创建其他模型的思路是很相似的。目前IMDB分类的最好成绩（state of the art）是我们和一个叫Sebastian Ruder的同事一起创造、发表的。刚刚演示的基本就是取得最好成绩的算法，里面还有些其它的小技巧。如果你努力尝试，你可以达到95%。这很接近我们得到的最好成绩。

提问: 对于不同于ImageNet的数据集，比如说卫星图或者生物图，我们应该使用我们自己的统计分布 (stats)。

Jeremy 说过：

如果你使用预训练的模型，你需要使用和预训练相同的stats。

这是为什么？是不是使用自己的stats标准化的模型有和ImageNet基本一样的分布？唯一我能想到的不同的地方是偏度 (skewness)。你这么说是因为偏度还是其他的东西？这是否代表对于差别比较大的数据集你不推荐使用预训练模型？[\[1:46:53\]](#)

不是。你可以看到，我在所有的地方都用了预训练模型。每次我用ImageNet预训练模型时，我都使用ImageNet stats。为什么呢？因为模型是用这些stats训练的。比如说，假如你在分类不同品种的绿青蛙。如果你使用自己数据集每个channel (红蓝绿) 均值，你最终会把它们转换成在每个channel上服从均值为0标准差为1的分布。这意味着它们不再像绿青蛙了。而是像灰青蛙。但是ImageNet希望青蛙是绿的。所以你需要用ImageNet训练者使用的stats做标准化。否则你的数据集的唯一特征会不见，你在做标准化时把它们丢失了。所以你需要使用和训练时相同的stats。

每个例子里我们都是用mini batch做梯度下降（随机梯度下降）来拟合模型参数。这些参数是用来做矩阵乘法的。这个课程的后半部分里，我们会学习卷积，这也是一种矩阵乘法。

尽管没有矩阵乘法运算可以创建出这样的东西：可以识别电影评论是正面还是负面、看出卫星图识别出上面有没有道路。这远远不是线性分类器可以做到的。现在我们知道这是深度神经网络。深度神经网络包含很多这样的矩阵乘法，每个矩阵乘法运算都是一个线性分类器。一个线性函数在另一个的上层。如果你回忆高中数学，你可能会记得如果你有一个 $y = ax + b$ ，然后在它上面做 $cy + d$ ，它还是一个有不同斜率和截距的线性函数。所以矩阵乘法运算不会有任何帮助。

这些模型实质上是什么呢？我们实质上在做什么？这是个有意思的事情：我们确实做了矩阵乘法运算（或者像卷积这样的变种），但每做一次矩阵乘法后，我们会做一次非线性计算或者叫**激活函数**。激活函数把矩阵乘法的结果作为输入，在上面做些处理。这是一些我们用到的激活函数 ([by Sagar Sharma](#))：

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

以前，最常用的函数是**sigmoid**。它们有特定的数学定义。现在我们几乎不用这个了。我们总是用**rectified linear unit (ReLU)**。当你做深度学习时，用很大很长的、引人注意的词很重要。否则普通人都会觉得他也可以做 😊。但是，只告诉你们，rectified linear unit是用这样的函数定义的：

```
1 | max(x, 0)
```

就是这样。如果你想高大上些，你可以用缩写ReLU，来显示你在一个高大上的团队里。这就是ReLU激活函数。

这是神奇的事情。如果你把红绿蓝像素值输入，做矩阵乘法运算，把负值用0替代，做另外一个矩阵运算，把负值用0替代，你不停地这样做，就能得到一个神经网络。就是这样而已。

通用逼近定理Universal approximation theorem [1:52:27]

这是怎样做到的呢？一个特别酷的叫Michael Nielsen的家伙展示了他的作品。他有一个很好的网站（实际上是一本书）<http://neuralnetworksanddeeplearning.com>，上面有这些美妙的JavaScript的控件，你可以在上面做操作。因为这是以前做的，以前我们一般用sigmoids激活函数。他演示了，如果你做足够的矩阵乘法运算，然后用sigmoids处理（用ReLU也是完全一样的），你可以创建出任意函数。这个线性函数和非线性函数的组合可以创建任意形状的理论被称作通用逼近定理。

它说的是，如果你有一批线性函数和非线性函数，它们最终可以无限逼近任何函数。所以你只需要保证你有足够的或者足够多的矩阵来做乘法运算。如果这个函数是由一批线性函数和非线性函数构成的，非线性函数可以是这些激活函数里的任何一个，如果它可以逼近任何函数，你需要做的就是找出这些做乘法的权重矩阵的值，这些矩阵相乘可以解决你要处理的问题。我们已经知道了怎样找到参数的值。我们可以用梯度下降。就是这样。

我发现最难向学生们解释的事情是：我们现在已经学完了。人们总是在课后来问我“其余的是什么，请解释下深度学习的其余的内容”。但是，没有其余的了。我们有一个函数可以接收像素或者其他的东西，我们用一些矩阵和输入值相乘，把负值替换成0，我们再用另一个矩阵乘以它，把负值替换成0，多做几次。我们看看它里我们的目标有多近，然后按照梯度下降方法，用导数更新参数，多做几次。最终，我们得到了可以分类电影评论或者可以识别布偶猫（ragdoll cat）的程序。就是这样。

这很难直观地理解的原因是，我们讲的矩阵是有上亿个参数的（把所有的加起来）。它们是非常大的矩阵。对于多次用一个线性模型乘以一些东西，把负值替换成零这样一个操作能做些什么，你的直觉处理不了。你只能根据经验来接受这个现实：这样的操作很有效。

在课程的第二部分，我们会从头开始实现这些。但是提前讲下，你会发现它们就是5行代码。就是一个for循环，里面运行 `t = x @ w1, t2 = max(t, 0)`，再用一个for循环执行所有的矩阵，最后计算损失函数。当然，我们不会自己计算梯度，因为PyTorch为我们做了这些。就是这样。

提问: 有一个关于分词的问题。我比较好奇分词是怎样做到处理词组的，比如说San Francisco [1:56:45]

怎样对San Francisco这样的东西做分词。San Francisco包含两个token `san` `Francisco`。就是这样。这就是怎样对 San Francisco 做分词。这个问题可能是做过传统NLP的人提的，他们经常要用叫n-grams的东西。N-grams是基于线性模型计算像San Francisco这样的特定的文本字符串出现多少次。当n等于2时，n-grams就是bi-gram。使用深度学习，我们不用再考虑这个，这很酷。像很多其它东西一样，有了深度学习后，很多复杂的特征工程用不上了。使用深度学习，每个token只是一个单词（对 `you're` 这种实际上包含两个词的情况，你把它分成两个单词），然后我们要做的就是让深度学习模型找出组合单词的最好方法。现在，当我们说让深度学习模型找出它来的时候，我们指的当然就是使用梯度下降找出能给出正确答案的权重矩阵。没有其他的内容。

另外，有些小技巧。在后半部分，我们会学习针对图像模型的技巧，使用卷积，让它成为CNN，针对语言模型的技巧是使用递归模型或者叫RNN。和刚才讲的相比较，这都是次要的技巧。使用RNN，它可以学到当 san 和 Francisco 这两个词在一起时，会有不同的含义。

提问: 一些卫星图有4个通道。怎样用预训练模型处理4通道的数据或者2通道的数据？ [1:59:09]

我想这是我们要尝试并集成到fastai里的东西。希望你们看到这个课程视频时，会有做到这个的简单方法。预训练的ImageNet模型接收红绿蓝像素。如果你只有两个通道，你可以做一些事情，基本上就是多构造一个通道。可以让这个通道里的值都是0，或者是另外两个通道的平均值。你可以使用普通的PyTorch运算来创建这样一个通道。你可以提前在循环里构造，保存一个三通道的版本，也可以根据需要创建一个定制的数据集。

对于4通道的情况，你可能不希望舍弃第四个通道。所以你需要修改模型，在后面几节课里，我们会知道怎样做。基本上是对于初始权重矩阵（权重矩阵是不准确的，应该叫权重张量weight tensors，因为有多个维度），它的一个轴上要有三个slice（切片），我们会用0或者随机数初始化它们。简单讲就是这样，但是要理解究竟这是什么意思，我们还要再多花两节课。

总结 [2:01:19]

我们今天看了什么？开始时讲了现在创建一个web app很容易，我们有给初学者的资料，里面演示了怎样创建web app，大家使用我们目前学到的单标签分类创建了很多很酷的web app。

使用和单标签分类相同的方法，我们可以做这些事情，这很酷：

- 做多标签分类，比如处理planet数据集
- 图像分割
- 各种图像回归
- NLP分类
- 很多其他的

每个任务里，我们实际上做的都是：

- 梯度下降
- 非线性处理

通用逼近定理告诉我们，它能几乎无限逼近任何给定的函数，比如这些：

- 把说话的声音转化成这个人说的内容
- 把日语的句子转换成英语的句子
- 把一个狗的图片转换成一段描述狗的文字

这些都是我们可以用这个方法学习的数学函数。

这周，看看你们是不是可以提出对你想解决的问题的一个有意思的想法，比如多标签分类、图像回归、图像分割，或者其他类型的东西，试试你是不是能解决这个问题。你可能会发现最难的部分是创建data bunch，所以你需要深入学习data block API，学会怎样用你的数据创建出data bunch。做些练习，你就会很擅长了。这不是一个很大的API，只有几个部分。要添加你自己的方法也很容易。如果在尝试时遇到了问题，就在论坛上提问。

下周，我们会回来学习更多NLP的内容。学习更多关于让SGD训练得更快的细节。我们会学习Adam， RMSProp之类的东西。希望能展示很多你们这周做出来的很酷的web app和模型。再见，谢谢！

