

一些被人判断是感觉是同一个人的图片，DCGAN 会产生很像的图片，把这个图片拿去 database，里面找一张最像的图，会发现最像的图跟这个图没有完全一样，代表 DCGAN 没有真的硬背 training data 里面的图。不知道为什么它会产生很像的图，但这个图并不是从 database 里面背出来的。

它要衡量 DCGAN 到底可以产生多少不一样的 image，它发现如果 sample 四百张 image 的时候，有大于 50% 的机率，可以从四百张 image 里面，找到两张人觉得是一样的人脸，借由这个机率就可以反推到底整个 database 里面，整个 DCGAN 可以产生的人脸里面，有多少不同的人脸。

详细反推的细节，你再 check 一下 paper，有一个 database 有面有 M 张 image，M 到底应该多大才会让你 sample 四百张 image 的时候，有大于 50% 的机率 sample 到重复的。总之反推出 DCGAN 它可以产生各种不同的 image，其实只有 0.16 个 million 而已，只有十六万张图而已。

有另外一个做法叫 ALI，它比较强，反推出来可以产生一百万张各种不同的人脸。ALI 看起来可以产生的人脸多样性是比较高的。

但是不论是哪些方法都觉得它们产生的人脸的多样性，跟真实的人脸比起来，还是有一定程度的差距。感觉 GAN 没有办法真的产生人脸的 distribution，这些都是尚待研究的问题。

GAN 的一个 issue 就是它产生的 distribution 不够大，它产生的 distribution 太 narrow，有一些 solution，比如说有一个方法，现在比较少人用，因为它 implement 起来很复杂，运算量很大，叫做 Unroll GAN。

Mini-batch Discrimination

有另外一个方法叫做 Mini-batch Discrimination，一般在 train discriminator 的时候，discriminator 只看一张 image，决定它是好的还是不好的。

Mini-batch Discriminator 是让 discriminator 看一把 image，决定它是好的还是不好，看一把 image 跟看一张 image 有什么不同？看一把 image 的时候不只要 check 每一张 image 是不是好的，还要 check 这些 image 它们看起来像不像。

discriminator 会从 database 里面 sample 一把 image 出来，会让 generator sample 一把 image 出来，如果 generator 每次 sample 都是一样的 image，发生 Mode collapse 的情形，discriminator 就会抓到这件事，因为在 training data 里面每张图都差很多，如果 generator 产生的图都很像，discriminator 因为它不是只看一张图，它是看一把图，它就会抓到这把图看起来不像是 realistic。

还有另外一个也是看一把图的方法，叫做 OTGAN，Optimal Transport GAN。

Transfer Learning

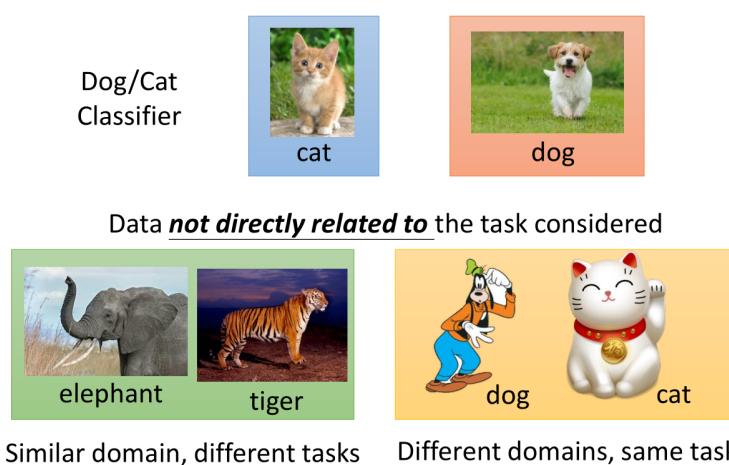
Transfer Learning

迁移学习，主要介绍共享layer的方法以及属性降维对比的方法

迁移学习，transfer learning，旨在利用一些不直接相关的数据对完成目标任务做出贡献

以猫狗识别为例，解释“不直接相关”的含义：

- input domain是类似的，但task是无关的
比如输入都是动物的图像，但这些data是属于另一组有关大象和老虎识别的task
- input domain是不同的，但task是一样的
比如task同样是做猫狗识别，但输入的是卡通图



迁移学习问的问题是：我们能不能再有一些不相关data的情况下，然后帮助我们现在要做的task。

为什么要考虑迁移学习这样的task呢？

举例来说：在speech recognition里面(台语的语音辨识)，台语的数据是很少的(但是语音的数据是很好收集的，中文，英文等)。那我们能不能用其他语音的数据来做台语这件事情。

或者在image recognition里面有一些是medical images，你想要让机器自动诊断说，有没有 tumor 之类的，这种medical image其实是很小的，但是image data是很不缺的。

或者是在文件的分析上，你现在要分析的文件是某个很 specific 的 domain，比如说你想要分析的是，某种特别的法律的文件，那这种法律的文件或许 data 很少，但是假设你可以从网络上 collect 一大堆的 data，那这些 data，有可能是有帮助的。

用不相干的数据来做domain其他的data，来帮助现在的task，是有可能的。事实上，我们在日常生活中经常会使用迁移学习，比如我们会把漫画家的生活自动迁移类比到研究生的生活。

迁移学习有很多的方法，它是很多方法的集合。下面你有可能会看到我说的terminology可能跟其他的有点不一样，不同的文献用的词汇其实是不一样的，有些人说算是迁移学习，有些人说不算是迁移学习，所以这个地方比较混乱，你只需要知道那个方法是什么就好了。

我们现在有一个我们想要做的task，有一些跟这个task有关的数据叫做**target data**，有一些跟这个task无关的数据，这个data叫做**source data**。这个target data有可能是有label的，也有可能是没有label的，这个source data有可能是有label的，也有可能是没有label的，所以现在我们就有四种可能，所以之后我们会分这四种来讨论。

Case 1

这里target data和source data都是带有标签的：

- target data: (x^t, y^t) , 作为有效数据，通常量是很少的
如果target data量非常少，则被称为**one-shot learning**
- source data: (x^s, y^s) , 作为不直接相关数据，通常量是很多的

Model Fine-tuning

One-shot learning: only a few examples in target domain

- Task description
 - Source data: (x^s, y^s) ← A large amount
 - Target data: (x^t, y^t) ← Very little
- Example: (supervised) speaker adaption
 - Source data: audio data and transcriptions from many speakers
 - Target data: audio data and its transcriptions of specific user
- Idea: training a model by source data, then fine-tune the model by target data
 - Challenge: only limited target data, so be careful about overfitting

Model Fine-tuning

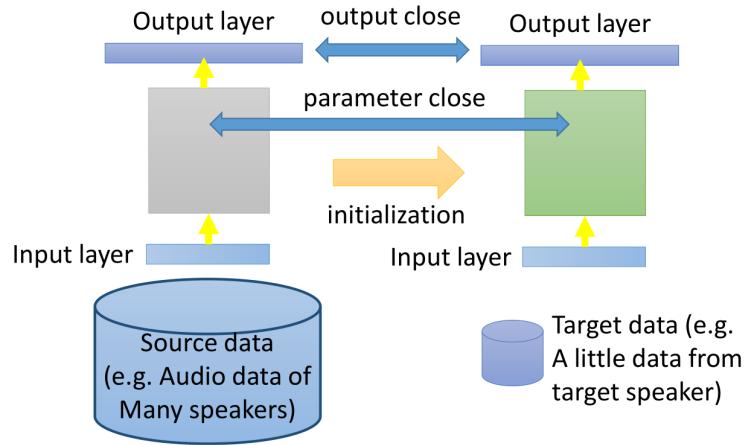
模型微调的基本思想：用source data去训练一个model，再用target data对model进行微调(fine tune)

所谓“微调”，类似于pre-training，就是把用source data训练出的model参数当做是参数的初始值，再用target data继续训练下去即可，但当target data非常少时，可能会遇到的challenge是，你在source data train出一个好的model，然后在target data上做train，可能就坏掉了。

所以训练的时候要小心，有许多技巧值得注意

Conservation Training

如果现在有大量的source data，比如在语音识别中有大量不同人的声音数据，可以拿它去训练一个语音识别的神经网络，而现在你拥有的target data，即特定某个人的声音数据，可能只有十几条左右，如果直接拿这些数据去再训练，肯定得不到好的结果。



此时我们就需要在训练的时候加一些限制，让用target data训练前后的model不要相差太多：

- 我们可以让新旧两个model在看到同一笔data的时候，output越接近越好
- 或者让新旧两个model的L2 norm越小越好，参数尽可能接近
- 总之让两个model不要相差太多，防止由于target data的训练导致过拟合

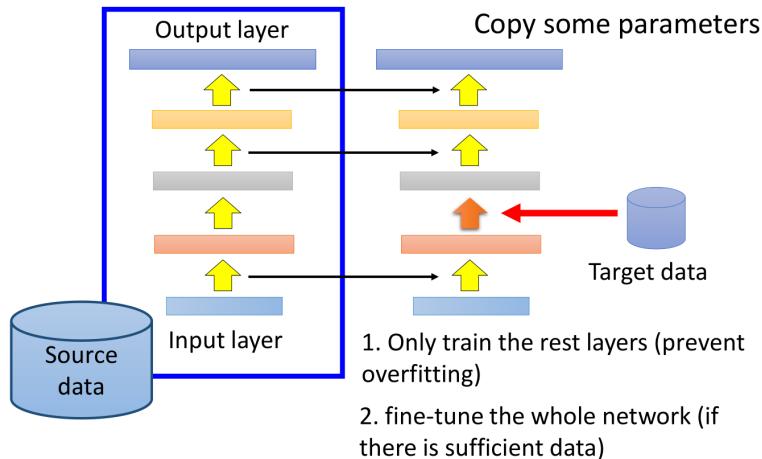
注：这里的限制就类似于regularization

Layer Transfer

现在我们已经有一个用source data训练好的model，此时把该model的某几个layer拿出来复制到同样大小的新model里，接下来只用target data去训练余下的没有被复制到的layer

这样做的好处是target data只需要考虑model中非常少的参数，这样就可以避免过拟合。

如果target data足够多，fine-tune 整个model也是可以的。

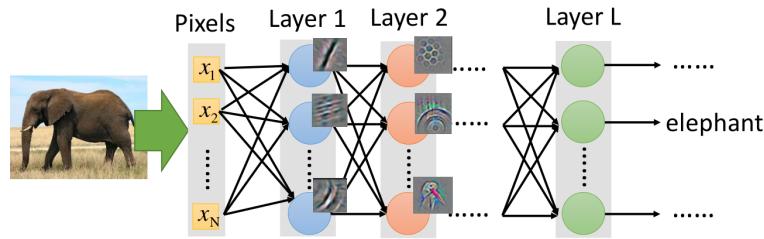


Layer Transfer是个非常常见的技巧，接下来要面对的问题是，哪些layer应该被transfer，哪些layer不应该去transfer呢？

有趣的是在不同的task上面需要被transfer的layer往往是不一样的：

- 在语音识别中，往往迁移的是最后几层layer，再重新训练与输入端相邻的那几层
由于口腔结构不同，同样的发音方式得到的发音是不一样的，NN的前几层会从声音信号里提取出发音方式，再用后几层判断对应的词汇，从这个角度看，NN的后几层是跟特定的人没有关系的，因此可做迁移
- 在图像处理中，往往迁移的是前面几层layer，再重新训练后面的layer
CNN在前几层通常是做最简单的识别，比如识别是否有直线斜线、是否有简单的几何图形等，这些layer的功能是可以被迁移到其它task上通用的
- case by case，运用之妙，存乎一心

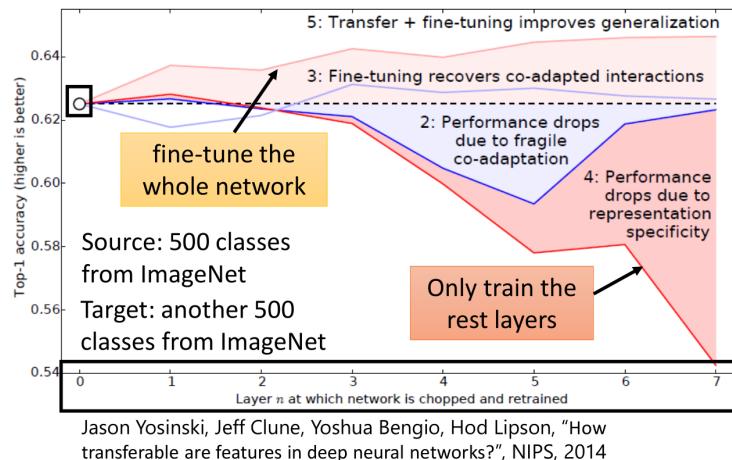
- Which layer can be transferred (copied)?
 - Speech: usually copy the last few layers
 - Image: usually copy the first few layers



Demo

这边是 image 在 layer transfer 上的实验，这个实验做在 ImageNet 上，把 ImageNet 的 corpus，一百二十万张 image 分成 source 跟 target。这个分法是按照 class 来分的，我们知道 ImageNet 的 image 一个 typical 的 setup 是有一千个 class，把其中五百个 class 归为 source data，把另外五百个 class 归为 target data。

横轴是我们在做 transfer learning 的时候，copy 了几个 layer。copy 0 个 layer 就代表完全没有做 transfer learning。这是一个 baseline，就直接在 target data 上面 train 下去。纵轴是 top-1 accuracy，所以是越高越好。没有做 transfer learning 是白色这个点



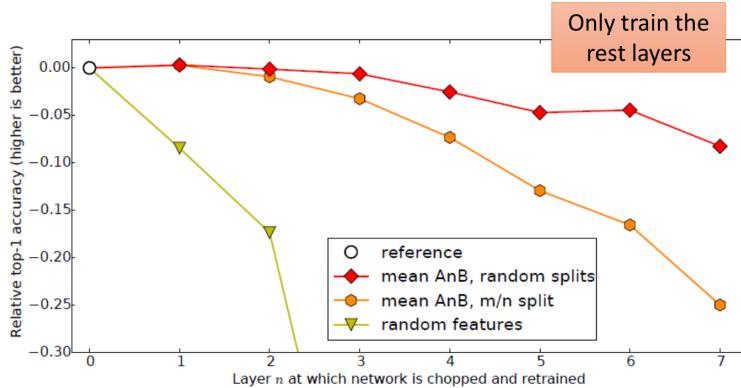
只有 copy 第一个 layer 的时候，performance 稍微有点进步，copy 前面两个 layer，performance 几乎是持平的，但是 copy 的 layer 太多，结果是会坏掉。

这个实验显示说在不同的 data 上面，train 出来的 neural network，前面几个 layer 是可以共享的，后面几个可能是没有办法共享的。如果 copy 完以后，还有 fine-tune 整个 model 的话，把第一个 layer，在 source domain 上 train 一个 model，然后把第一个 layer copy 过去以后，再用 target domain fine-tune 整个 model，包括前面 copy 过的 layer 的话，那得到 performance 是橙色这条线，在所有的 case 上面都是有进步的。

其实这个结果很 surprised，不要忘了，这可是 ImageNet 的 corpus，一般在做 transfer learning 的时候，都是假设 target domain 的 data 非常少，这边 target domain 可是有六十万张，这 target domain 的 data 是非常多的。但是就算在这个情况下，再多加了另外六十张 image 做 transfer learning，其实还是有帮助的。

这两条蓝色的线跟 transfer learning 没有关系，不过是这篇 paper 里面发现一个有趣的现象。他想要做一个对照组，在 target domain 上面 learn 一个 model，把前几个 layer copy 过来，再用一次 target domain 的 data train 剩下几个 layer。前面几个 layer 就 fix 住，只 train 后面几个 layer，直觉上这样做应该跟直接 train 整个 model 没有太大差别，先 train 好一个 model，fix 前面几个 layer，接下来只 train 后面几个 layer，结果有些时候是会坏掉的。他的理由是 training 的时候，前面的 layer 跟后面的 layer 他们其实是要互相搭配，所以如果只 copy 前面的 layer，然后只 train 后面的 layer，后面的 layer 就没有办法跟前面的 layer 互相搭配，结果有点差。如果可以 fine-tune 整个 model 的话，performance 就跟没有 transfer learning 是一样的。这是另一个有趣的发现，作者自己对这件事情是很 surprised。

这是另外一个实验结果，红色这条线是前一页看到的红色的这条线。这边假设 source 跟 target 是比较没有关系的，把 ImageNet 的 corpus 分成 source data 跟 target data 的时候把自然界的东西，通通当作 source，target 通通是人造的东西，桌子、椅子等等。这样 transfer learning 会有什么样的影响？



Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson, "How transferable are features in deep neural networks?", NIPS, 2014

如果 source 跟 target 的 data 是差很多的，在做 transfer learning 的时候，performance 会掉的比较多，前面几个 layer 影响还是比较小的，如果只 copy 前面几个 layer，仍然跟没有 copy 是持平的，这意味着，就算是 source domain 跟 target domain 是非常不一样的，一边是自然的东西，一边是人造的东西，在 neural network 第一个 layer，他们仍然做的事情很有可能是一样的。黄色的这条线，烂掉的这条线是假设前面几个 layer 的参数是 random 的。

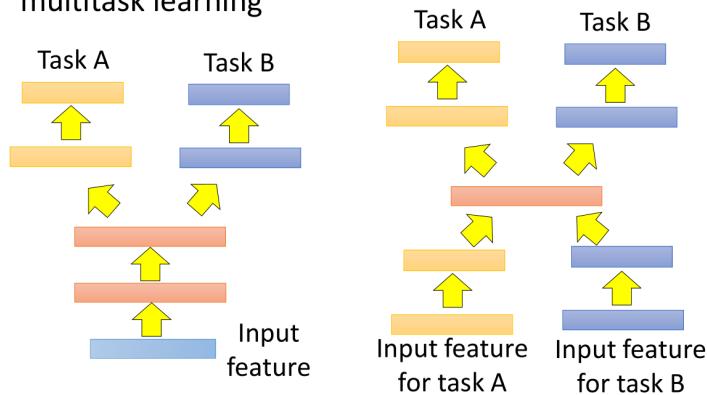
Multitask Learning

fine-tune 仅考虑在 target data 上的表现，而多任务学习，则是同时考虑 model 在 source data 和 target data 上的表现

如果两个 task 的输入特征类似，则可以用同一个神经网络的前几层 layer 做相同的工作，到后几层再分方向到不同的 task 上，这样做的好处是前几层得到的 data 比较多，可以被训练得更充分。这样做的前提是：这两个 task 有共通性，可以共用前面几个 layer。

有时候 task A 和 task B 的输入输出都不相同，两个不同 task 的 input 都用不同的 neural network 把它 transform 到同一个 domain 上去，中间可能有某几个 layer 是 share 的，也可以达到类似的效果

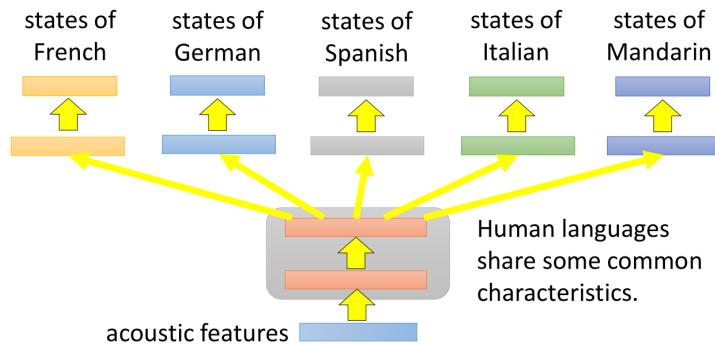
- The multi-layer structure makes NN suitable for multitask learning



以上方法要求不同的 task 之间要有一定的共性，这样才有共用一部分 layer 的可能性

Multilingual Speech Recognition

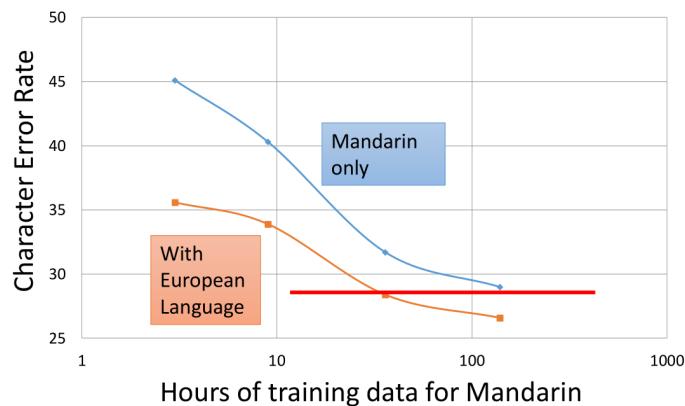
多任务学习一个很成功的例子就是多语言的语音辨识，假设你现在手上有一大堆不同语言的 data (法文，中文，英文等)，那你在 train 你的 model 的时候，同时可以辨识这五种不同的语言。这个 model 前面几个 layer 他们会共用参数，后面几个 layer 每一个语言可能会有自己的参数，这样做是合理的。虽然是不同的语言，但是都是人类所说的，所以前面几个 layer 它们可能是 share 同样的咨询，共用同样的参数。



Similar idea in translation: Daxiang Dong, Hua Wu, Wei He, Dianhai Yu and Haifeng Wang, "Multi-task learning for multiple language translation.", ACL 2015

在translation上，你也可以做同样的事情，假设你今天要做中翻英，也要做中翻日，你也把这两个model一起train。在一起train的时候无论是中翻英还是中翻日，你都要把中文的data先做process，那一部分neural network就可以是两种不同语言的data共同使用。

在过去收集了十几种语言，把它们两两之间互相做transfer，做了一个很大的 $N \times N$ 的table，每一个 case 都有进步。所以目前发现大部分 case，不同人类的语言就算你觉得它们不是非常像，但是它们之间都是可以transfer的。



Huang, Jui-Ting, et al. "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers." ICASSP, 2013

上图为从欧洲语言去transfer中文，横轴是中文的数据，纵轴是character error rate。假设你一开始用中文train一个model，data很少，error rate很大，随着data越来越多，error rate就可以压到30以下。但是今天如果你有一大堆的欧洲语言，你把这些欧洲语言跟中文一起去做multi-task training，用这个欧洲语言的数据来帮助中文model前面几层让它train的更好。你会发现：在中文data很少的情况下，你有做迁移学习，你就得到比较好的性能。随着中文data越多的时候，中文本身performance越好，就算是中文有一百小时的数据，借用一些从欧洲语言来的knowledge，对这个辨识也是有帮助的。

所以这边的好处是：假设你做多任务学习的时候，你会发现你有100多个小时跟50小时对比，如果你有做迁移学习的话，你只需要1/2的数据就可以跟有两倍的数据做的一样好。

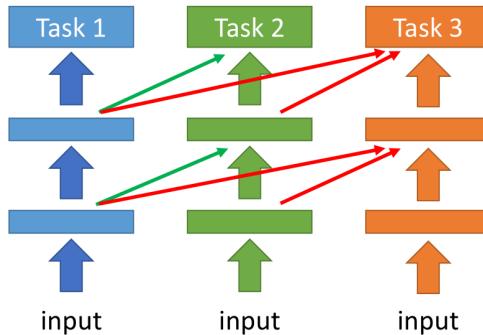
常常有人会担心说：迁移学习会不会有负面的效应，这是有可能的，如果两个task不像的话，你的transfer就是negative的。但是有人说：总是思考两个task到底之间能不能transfer，这样很浪费时间。所以有人 propose 了 progressive neural networks

Progressive Neural Network

如果两个task完全不相关，硬是把它们拿来一起训练反而会起到负面效果

而在Progressive Neural Network中，每个task对应model的hidden layer的输出都会被接到后续model的hidden layer的输入上，这样做的好处是：

- task 2的数据并不会影响到task 1的model，因此task 1一定不会比原来更差
- task 2虽然可以借用task 1的参数，但可以将之直接设为0，最糟的情况下就等于没有这些参数，也不会对本身的performance产生影响
- task 3也做一样的事情，同时从task 1和task 2的hidden layer中得到信息



Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, Raia Hadsell, "Progressive Neural Networks", arXiv preprint 2016

Case 2

这里target data不带标签，而source data带标签：

- target data: (x^t)
- source data: (x^s, y^s)

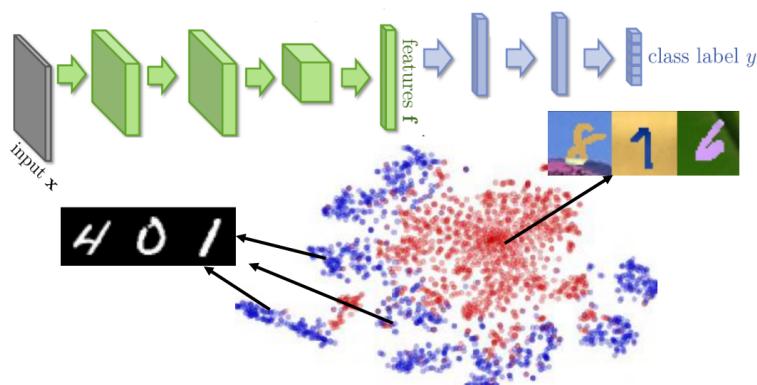
举例来说：我们可以说：source data是MNIST image，target data是MNIST-M image(MNIST image加上一些奇怪的背景)。MNIST是有label的，MNIST-M是没有label的，在这种情况下我们通常是把source data视作training data，target data视作testing data。产生的问题是：training data跟testing data是非常mismatch的。

- Source data: $(x^s, y^s) \rightarrow$ Training data
 - Target data: $(x^t) \rightarrow$ Testing data
- } Same task, mismatch



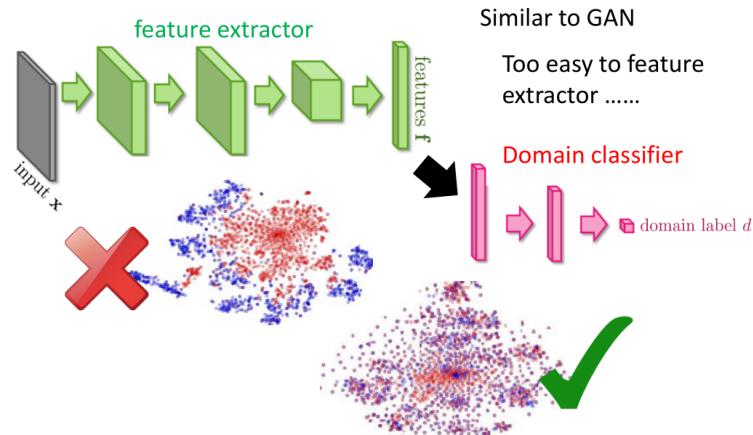
这个时候一般会把source data当做训练集，而target data当做测试集，如果不管训练集和测试集之间的差异，直接训练一个普通的model，得到的结果准确率会相当低。

实际上，神经网络的前几层可以被看作是在抽取feature，后几层则是在做classification，如果把用MNIST训练好的model所提取出的feature做t-SNE降维后的可视化，可以发现MNIST的数据特征明显分为紫色的十团，分别代表10个数字，而作为测试集的数据却是挤成一团的红色点，因此它们的特征提取方式根本不匹配。



Domain-adversarial Training

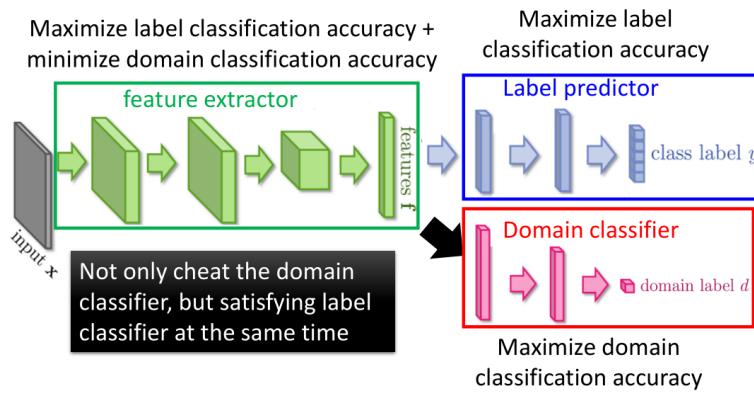
所以该怎么办呢？希望做的事情是：前面的feature extract 它可以把domain的特性去除掉，这一招叫做Domain-adversarial training。也就是feature extract output不应该是红色跟蓝色的点分成两群，而是不同的domain应该混在一起。



那如何learn这样的feature extract呢？这边的做法是在后面接一下domain classifier。把feature extract output丢给domain classifier，domain classifier它是一个classification task，它要做的事情就是：根据feature extract给它的feature，判断这个feature来自于哪个domain，在这个task里面，要分辨这些feature是来自MNIST还是来自与MNIST-M。

有一个generator 的output，然后又有discriminator，让它的架构非常像GAN。但是跟GAN不一样的事情是：之前在GAN那个task里面，你的generator要做的事情是产生一个image，然后骗过discriminator，这件事很难。

但是在这个Domain-adversarial training里面，要骗过domain classifier太简单了。有一个solution是：不管看到什么东西，output都是0，这样就骗过了classifier。

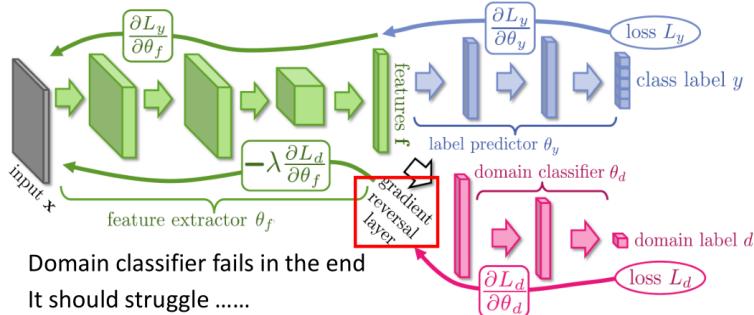


所以你要在feature extract增加它任务的难度，所以feature extract它output feature不仅要骗过domain classifier还要同时让label predictor做好。这个label predictor它就吃feature extract output，然后它的output就是10个class。

所以今天你的feature extract 不只要骗过domain classifier，还要满足label predictor的需求。抽出的feature不仅要把domain的特性消掉，同时还要保留原来feature的特性。

那我们把这三个neural放在一起的话。实际上就是一个大型的neural network，是一个各怀鬼胎的neural network(一般的neural network整个参数想要做的事情都是一样的，要minimize loss)，在这个neural network里面参数的目标是不同的。label predictor做的事情是把class分类做的正确率越高越好，domain classifier做的事情是想正确predict image是属于哪个domain。feature extractor想要做的事情是：要同时improve label predictor，同时想要minimize domain classifier accuracy，所以feature extractor 其实是在做陷害队友这件事的。

feature extractor 怎样陷害队友呢(domain classifier)？这件事情是很容易的，你只要加一个gradient reversal layer就行了。也就是你在做back-propagation(Domain classifier 计算 back propagation 有 forward 跟 backward 两个 path)，在做backward task的时候你的domain classifier传给feature extractor什么样的value，feature extractor就把它乘上一个负号。也就是domain classifier 告诉你说某个value要上升，它就故意下降。



Yaroslav Ganin, Victor Lempitsky, Unsupervised Domain Adaptation by Backpropagation, ICML, 2015

Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, Domain-Adversarial Training of Neural Networks, JMLR, 2016

domain classifier因为看不到真正的image，所以它最后一定fail掉。因为它所能看到的东西都是feature extractor告诉它的，所以它最后一定会无法分辨feature extractor所抽出来的feature是来自哪个domain。

这个model原理讲起来很简单，但可能实际上的training可能跟GAN一样是没有那么好train的，问题就是domain classifier一定要奋力的挣扎，因为它要努力去判断现在的feature是来自哪个domain。如果domain classifier他比较弱、懒惰，他一下就放弃不想做了，就没有办法把前面的feature extractor逼到极限，就没有办法让feature extractor真的把domain information remove掉。如果domain classifier很weak，他一开始就不想做了，他output永远都是0的话，那feature extractor胡乱弄什么feature都可以骗过classifier的话，那就达不到把domain特性remove掉的效果，这个task一定要让domain classifier奋力挣扎然后才死掉，这样才能把feature extractor的潜能逼到极限。

SOURCE	MNIST	SYN NUMBERS	SVHN	SYN SIGNS	
	MNIST-M	SVHN	MNIST	GTSRB	
TARGET					
METHOD	SOURCE	MNIST	SYN NUMBERS	SVHN	SYN SIGNS
	TARGET	MNIST-M	SVHN	MNIST	GTSRB
SOURCE ONLY		.5749	.8665	.5919	.7400
SA (FERNANDO ET AL., 2013)		.6078 (7.9%)	.8672 (1.3%)	.6157 (5.9%)	.7635 (9.1%)
PROPOSED APPROACH		.8149 (57.9%)	.9048 (66.1%)	.7107 (29.3%)	.8866 (56.7%)
TRAIN ON TARGET		.9891	.9244	.9951	.9987

Yaroslav Ganin, Victor Lempitsky, Unsupervised Domain Adaptation by Backpropagation, ICML, 2015

Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, Domain-Adversarial Training of Neural Networks, JMLR, 2016

这是paper一些实验的结果，做不同domain的transfer。如果看实验结果的话，纵轴代表用不同的方法，这边有一个source only的方法，直接在source domain上train一个model，然后test在target domain上，如果只用source only的话，Performance是比较差的。这边比较另一个transfer learning的方法，大家可以自己去看参考文献。这篇paper proposed的方法是刚刚讲的domain-adversarial training。

直接拿target domain的数据去做training，会得到performance是最下面这个row，这其实是performance的upper bound。用source data跟target data train出来的结果是天差地远的，这中间有一个很大的gap。

如果用domain-adversarial training可以在不同的case上面都有很好的improvement。

Zero-shot Learning

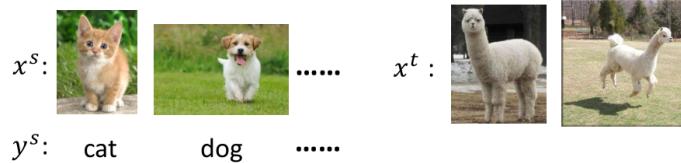
在zero-shot-learning里面跟刚才讲的task是一样的，source data有label，target data没有label。

在刚才task里面可以把source data当做training data，把target data当做testing data，但是实际上在zero-shot learning里面，它的define又更加严格一点。它的define是：今天在source data和target data里面，它的task是不一样的。

比如说在影像上面(你可能要分辨猫跟狗)，你的source data可能有猫的class，也有狗的class。但是你的target data里面image是草泥马，在source data里面是从来没有出现过草泥马的，如果machine看到草泥马，就未免有点强人所难了。但是这个task在语音上很早就有solution了，其实语音是常常会遇到zero-shot learning的问题。

- Source data: $(x^s, y^s) \rightarrow$ Training data
- Target data: $(x^t) \rightarrow$ Testing data

Different tasks



In speech recognition, we can not have all possible words in the source (training) data.

How we solve this problem in speech recognition?

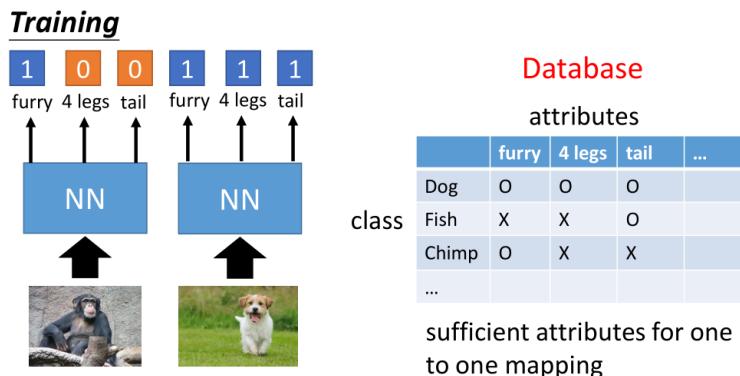
假如我们把不同的word都当做一个class的话，那本来在training的时候跟testing的时候就有可能看到不同的词汇。你的testing data本来就有一些词汇是在training的时候是没有看过的。

那在语音上我们如何来解决这个问题呢？不要直接去辨识一段声音是属于哪一个word，我们辨识的是一段声音是属于哪一个phoneme。然后我们在做一个phoneme跟table对应关系的表，这个东西也就是lexicon(词典)。在辨识的时候只要辨识出phoneme就好，再去查表说：这个phoneme对应到哪一个word。

这样就算有一些word是没有在training data里面的，它只要在lexicon里面出现过，你的model可以正确辨识出声音是属于哪一个phoneme的话，你就可以处理这个问题。

Attribute embedding

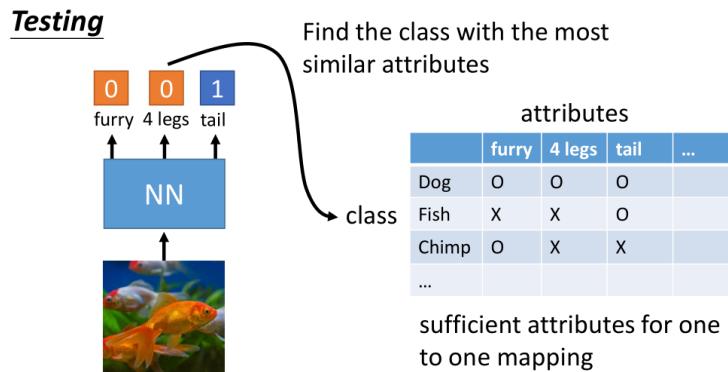
- Representing each class by its attributes



在影像上我们可以把每一个class用它的attribute来表示，也就是说：你有一个database，这个database里面有所以不同可能的object跟它的特性。假设你要辨识的是动物，但是你training data跟testing data他们的动物是不一样的。但是你有一个database，这个database告诉你说：每一种动物它是有什么样的特性。比如狗就是毛茸茸，四只脚，有尾巴；鱼是有尾巴但不是毛茸茸，没有脚。

这个attribute要更丰富，每一个class都要有不一样的attribute(如果两个class有相同的attribute的话，方法会fail掉)。那在training的时候，我们不直接辨识每一张image是属于哪一个class，而是去辨识：每一张image里面它具备什么样的attribute。所以你的neural network target就是说：看到猩猩的图，就要说：这是一个毛茸茸的动物，没有四只脚，没有尾巴。看到狗的图就要说：这是毛茸茸的动物，有四只脚，有尾巴。

- Representing each class by its attributes



那在testing的时候，就算今天来了你从来没有见过的image，也是没有关系的。你今天neural network target也不是input image判断它是哪一种动物，而是input这一张image判断具有什么样的attribute。所以input你从来没有见过的动物，你只要把它的attribute找出来，然后你就查表看说：在database里面哪一种动物它的attribute跟你现在model output最接近。有时可能没有一摸一样的也是没有关系的，看谁最接近，那个动物就是你要找的。

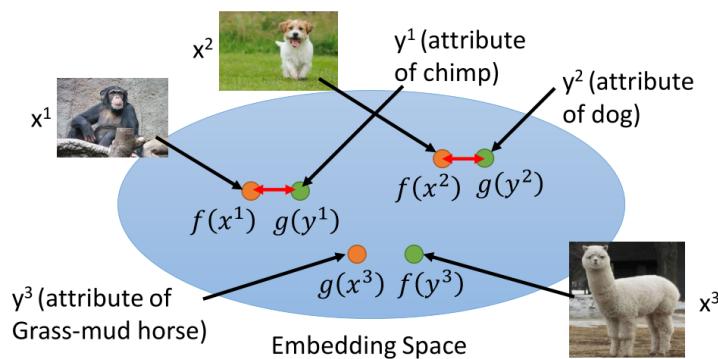
Zero-shot Learning

- Attribute embedding

$f(*)$ and $g(*)$ can be NN.

Training target:

$f(x^n)$ and $g(y^n)$ as close as possible



那有时候你的attribute可能非常的复杂(attribute dimension非常大)，你可以做attribute embedding。也就是说现在有一个embedding space，把training data每一个image都通过一个transform，变成一个embedding space上的一个点。然后把所有的attribute也都变成embedding space上的一个点，这个 $g(*)$ 跟 $f(*)$ 都可以是neural network，那training的时候希望 f 跟 g 越接近越好。那在testing的时候如果有一张没有看过的image，你就可以说这张image attribute embedding以后跟哪个attribute最像，那你就可以知道它是什么样的image。

image跟attribute都可以描述为vector，要做的事情就是把attribute跟image都投影到同一个空间里面。也就是说：你可以想象成是对image的vector，也就是图中的x，跟attribute的vector，也就是图中的y都做降维，然后都降到同一个dimension。所以你把x通过一个function f都变成embedding space上的vector，把y通过另外一个function g也都变成embedding space上的vector。

但是咋样找这个f跟g呢？你可以说f跟g就是neural network。input一张image它变成一个vector，或者input attribute 变成一个vector。training target是你希望说：假设我们已经知道 y^1 是 x^1 的attribute， y^2 是 x^2 的attribute，那你就希望说找到一个f跟g，它可以让 x^1 跟 y^1 投影到embedding space以后越接近越好， x^2 跟 y^2 投影到embedding space以后越接近越好。

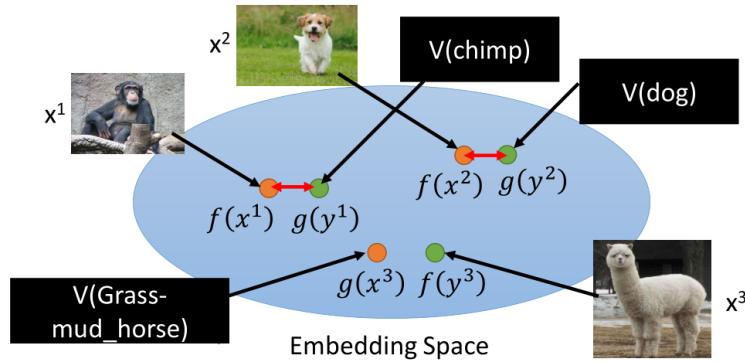
那现在把f跟g找出来了，那现在假如有一张你从来没见过的image x^3 在你的testing data里面，它也可以通过这个f变成embedding space上面的一个vector，接下来你就可以说这个embedding vector它跟 y^3 最接近，那 y^3 就是它的attribute，再来确定是哪个动物。

Attribute embedding + word embedding

Zero-shot Learning

What if we don't have database

- Attribute embedding + word embedding



又是你会遇到一个问题，如果我没有database呢？我根本不知道每一个动物的attribute是什么，怎么办呢？

可以借用word vector，word vector的每一个dimension就代表了现在这个word的某种attribute。所以不一定需要有个database去告诉你每一个动物的attribute是什么。假设有一组word vector，这组word vector

里面你知道每个动物他对应的 word 的 word vector，这 word vector 你可以拿一个很大的 corpus，比如说 Wikipedia train 出来，就可以把 attribute 直接换成 word vector，所以把 attribute 通通换成那个 word 的 word vector，再做跟刚才一样的 embedding，就结束了。

$$f^*, g^* = \arg \min_{f,g} \sum_n \|f(x^n) - g(y^n)\|_2 \quad \text{Problem?}$$

$$f^*, g^* = \arg \min_{f,g} \sum_n \max(0, k - f(x^n) \cdot g(y^n))$$

Margin you defined + $\max_{m \neq n} f(x^n) \cdot g(y^m)$

Zero loss: $k - f(x^n) \cdot g(y^n) + \max_{m \neq n} f(x^n) \cdot g(y^m) < 0$

$$\frac{f(x^n) \cdot g(y^n)}{\text{f}(x^n) \text{ and } g(y^n) \text{ as close}} - \frac{\max_{m \neq n} f(x^n) \cdot g(y^m)}{\text{f}(x^n) \text{ and } g(y^m) \text{ not as close}} > k$$

这个loss function存在些问题，它会让model把所有不同的x和y都投影到同一个点上：

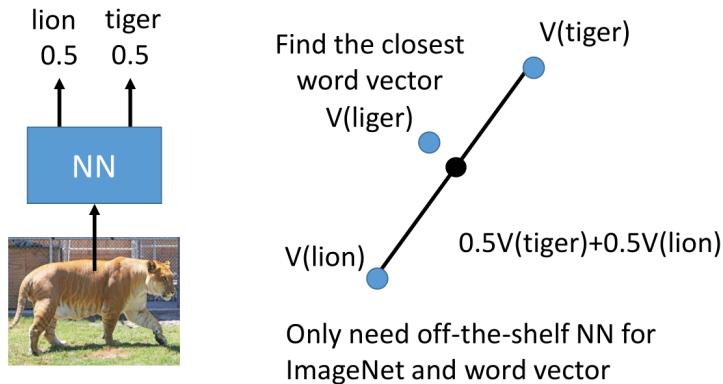
$$f^*, g^* = \arg \min_{f,g} \sum_n \|f(x^n) - g(y^n)\|_2$$

类似用t-SNE的思想，我们既要考虑同一对 x^n 和 y^n 距离要接近，又要考虑不属于同一对的 x^n 与 y^m 距离要拉大(这是前面的式子没有考虑到的)，于是有：

$$f^*, g^* = \arg \min_{f,g} \sum_n \max(0, k - f(x^n) \cdot g(y^n) + \max_{m \neq n} f(x^n) \cdot g(y^m))$$

Oloss的情况是： x^n 跟 y^n 之间的inner product大过所有其它的 y^m 跟 x^n 之间的inner product，而且要大过一个margin k

Convex Combination of Semantic Embedding



还有另外一个简单的Zero-Shot learning的方法叫做convex combination of semantic embedding。这个方法是说：在这边不需要做任何 training，就可以做 transfer learning。假设有一个 off-the-shelf 识别系统，跟一个 off-the-shelf 的 word vector。这两个可能不是自己 train，或网络上载下来的。

我把一张图丢到neural network里面去，它的output没有办法决定是哪一个class，但它觉得有0.5的机率是lion，有0.5的机率是tiger。接下来去找lion跟tiger的word vector，然后把 lion 跟 tiger 的 word vector 用 1:1 的比例混合，0.5 tiger 的 vector 加 0.5 lion 的 vector，得到另外一个新的vector。再看哪个word的vector跟这个混合之后的结果最接近。假设是liger最接近，那这个东西就是liger。

Example of Zero-shot Learning

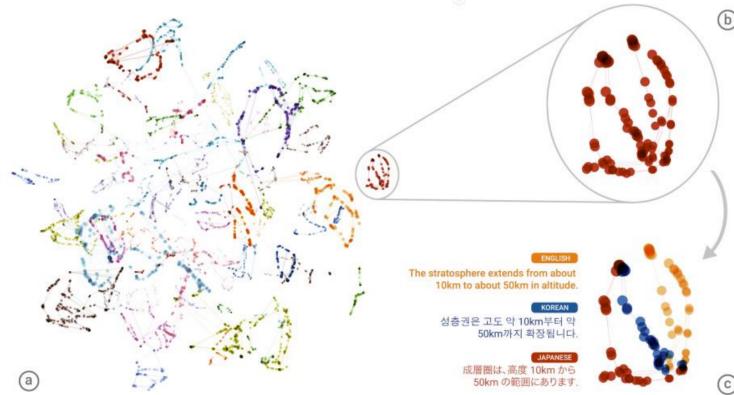
在training的时候，machine看过如何把英文翻译成韩文，知道咋样把韩文翻译为英文，知道咋样把英文翻译为日文，知道咋样把日文翻译为英文。但是它从来没有看过日文翻译韩文的数据，但是可以翻，但是它从来没有看过韩文翻译日文的数据，但是可以翻。

为什么zero-shot在这个task上是可行的呢？如果你今天用同一个model做了不同语言之间的translation以后，machine可以学到的事情是：对不同语言的input 句子都可以project到同一个space上面。

在training的时候，machine看过如何把英文翻译成韩文，知道咋样把韩文翻译为英文，知道咋样把英文翻译为日文，知道咋样把日文翻译为英文。但是它从来没有看过日文翻译韩文的数据，但是可以翻，但是它从来没有看过韩文翻译日文的数据，但是可以翻。

为什么zero-shot在这个task上是可行的呢？如果你今天用同一个model做了不同语言之间的translation以后，machine可以学到的事情是：对不同语言的input 句子都可以project到同一个space上面。

我们现在根据我们learn好的translation，那个translation有一个encoder，它会把你input的句子变成vector，decoder根据这个vector解回一个句子，就是翻译的结果。那今天我们就把不同语言都丢到这个encoder里面让它变成vector的话，那这些不同语言的不同句子在这个space上面有什么不一样的关系呢？



它发现说今天有日文、英文、韩文这三个句子，这三个句子讲的是同一件事情，通过encoder embedding以后再space上面其实是差不多的位置。在左边这个图上面不同的颜色代表说：不同语言的用一个意思的句子。所以你这样说：machine发明了一个新语言也是可以接受的，如果你把这个embedding space当做一个新的语言的话。machine做的是：发现可一个sequence language，每一种不同的语言都先要先转成它知道的sequence language，在用这个sequence language转为另外一种语言。

所以今天就算是某一个翻译task，你的input语言和output语言machine没有看过，它也可以通过过这种自己学出来的sequence language来做translation。

Case 3 & 4

刚刚讲的状况都是 source data 有 label 的状况，有时候会遇到 source data 没有 label 的状况。

target data 有 label，source data 没有 label，这种是 Self-taught learning

target data 没有 label，source data 也没有 label，这种是 Self-taught clustering

有一个要强调的是 Self-taught learning 跟 source data 是 unlabeled data，target data 是 labeled data

这也是一种 semi-supervised learning。这种 semi-supervised learning 跟一般 semi-supervised learning 有一些不一样，一般 semi-supervised learning 会假设那些 unlabeled data 至少还是跟 labeled data 是比较有关系的。但在 Self-taught learning 里面，那些 unlabeled data、那些 source data，跟 target data 关系比较远。

其实 Self-taught learning 概念很简单，假设 source data 够多，虽然它是 unlabeled，可以去 learn 一个 feature extractor，在原始的 Self-taught learning paper 里面，他的 feature extractor 是 sparse coding。因为这 paper 比较旧，大概十年前，现在也不见得要用 sparse coding 也可以 learn，比如说 auto-encoder。

总之，有大量的 data，他们没有 label，可以做的是用这些 data learn 一个好的 feature extractor，learn 一个好的 representation。用这个 feature extractor 在 target data 上面去抽 feature。

在 Self-taught learning 原始的 paper 里面，其实做了很多 task，这些 task 都显示 Self-taught learning 是可以得到显著 improvement 的。

Learning to extract better representation from the source data (unsupervised approach)

Extracting better representation for target data

Concluding Remarks