深度学习概述：从感知机到深度网络

（注：本文译自一篇博客，作者行文较随意，我尽量按原意翻译，但作者所介绍的知识还是非常好的，包括例子的选择、理论的介绍都很到位，由浅入深，源文地址）

近些年来，人工智能领域又活跃起来，除了传统了学术圈外，Google、Microsoft、facebook 等工业界优秀企业也纷纷成立相关 研究团队，并取得了很多令人瞩目的成果。这要归功于社交网络用户产生的大量数据，这些数据大都是原始数据，需要被进一步分析处理；还要归功于廉价而又强大 的计算资源的出现，比如 GPGPU 的快速发展。

除去这些因素，AI 尤其是机器学习领域出现的一股新潮流很大程度上推动了这次复兴——深度学习。本文中我将介绍深度学习背后的关键概念及算法，从最简单的元素开始并以此为基础进行下一步构建。

（本文作者也是 Java deep learning library 的作者，可以从此处获得，本文中的例子就是使用这个库实现的。如果你喜欢，可以在 Github 上给个星~。用法介绍也可以从此处获得）

机器学习基础

如果你不太熟悉相关知识，通常的机器学习过程如下：

1、机器学习算法需要输入少量标记好的样本，比如 10 张小狗的照片，其中 1 张标记为 1（意为狗）其它的标记为 0（意为不是狗）——本文主要使用监督式、二叉分类。

2、这些算法“学习”怎么样正确将狗的图片分类，然后再输入一个新的图片时，可以期望算法输出正确的图片标记（如输入一张小狗图片，输出 1；否则输出 0）。

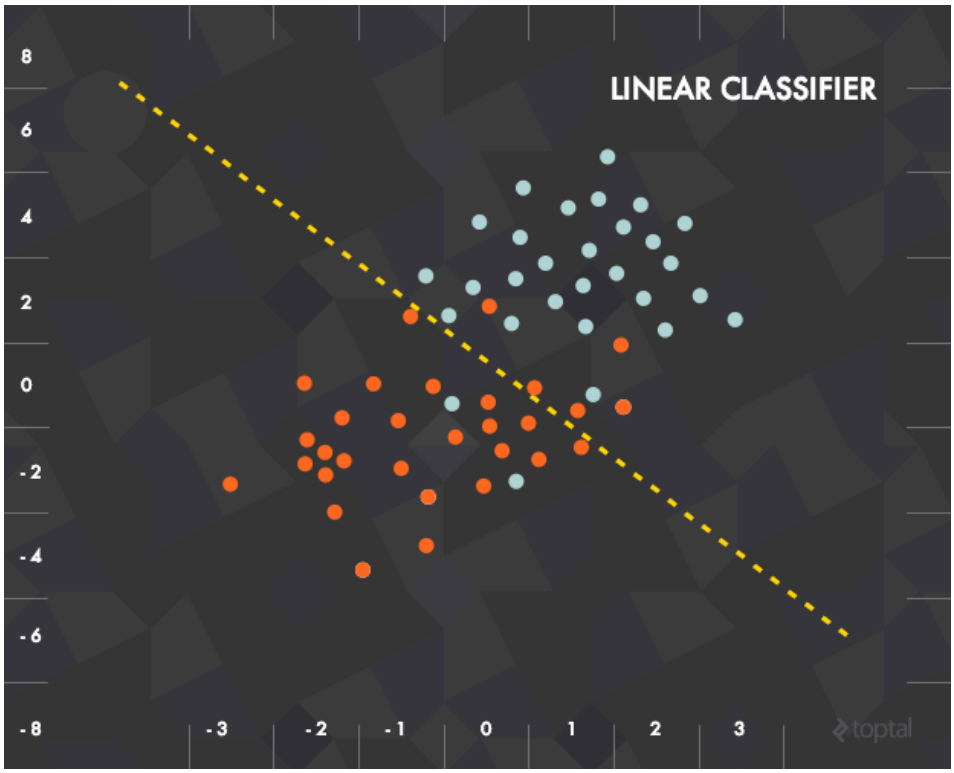
这通常是难以置信的：你的数据可能是模糊的，标记也可能出错；或者你的数据是手写字母的图片，用其实际表示的字母来标记它。

感知机

感知机是最早的监督式训练算法，是神经网络构建的基础。

假如平面中存在 *n* 个点，并被分别标记为“0”和“1”。此时加入一个新的点，如果我们想知道这个点的标记是什么（和之前提到的小狗图片的辨别同理），我们要怎么做呢？

一种很简单的方法是查找离这个点最近的点是什么，然后返回和这个点一样的标记。而一种稍微“智能”的办法则是去找出平面上的一条线来将不同标记的数据点分开，并用这条线作为“分类器”来区分新数据点的标记。



在本例中，每一个输入数据都可以表示为一个向量 **x** = (*x_1, x_2*) ，而我们的函数则是要实现“如果线以下，输出 0；线以上，输出 1”。

用数学方法表示，定义一个表示权重的向量 **w** 和一个垂直偏移量 *b*。然后，我们将输入、权重和偏移结合可以得到如下传递函数：

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b$$

这个传递函数的结果将被输入到一个激活函数中以产生标记。在上面的例子中，我们的激活函数是一个门限截止函数（即大于某个阈值后输出 1）：

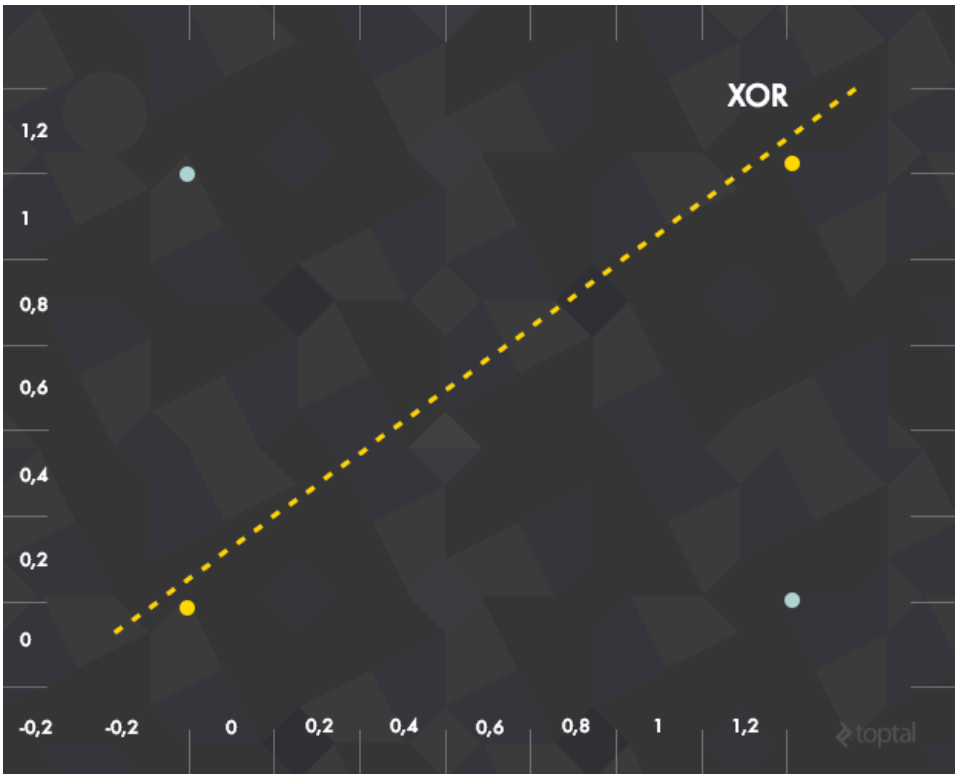
$$h(\mathbf{x}) = \begin{cases} 1 : & \text{if } f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 : & \text{otherwise} \end{cases}$$

训练

感知机的训练包括多训练样本的输入及计算每个样本的输出。在每一次计算以后，权重 *w* 都要调整以最小化输出误差，这个误差由输入样本的标记值与实际计算得出值的差得出。还有其它的误差计算方法，如均方差等，但基本的原则是一样的。

缺陷

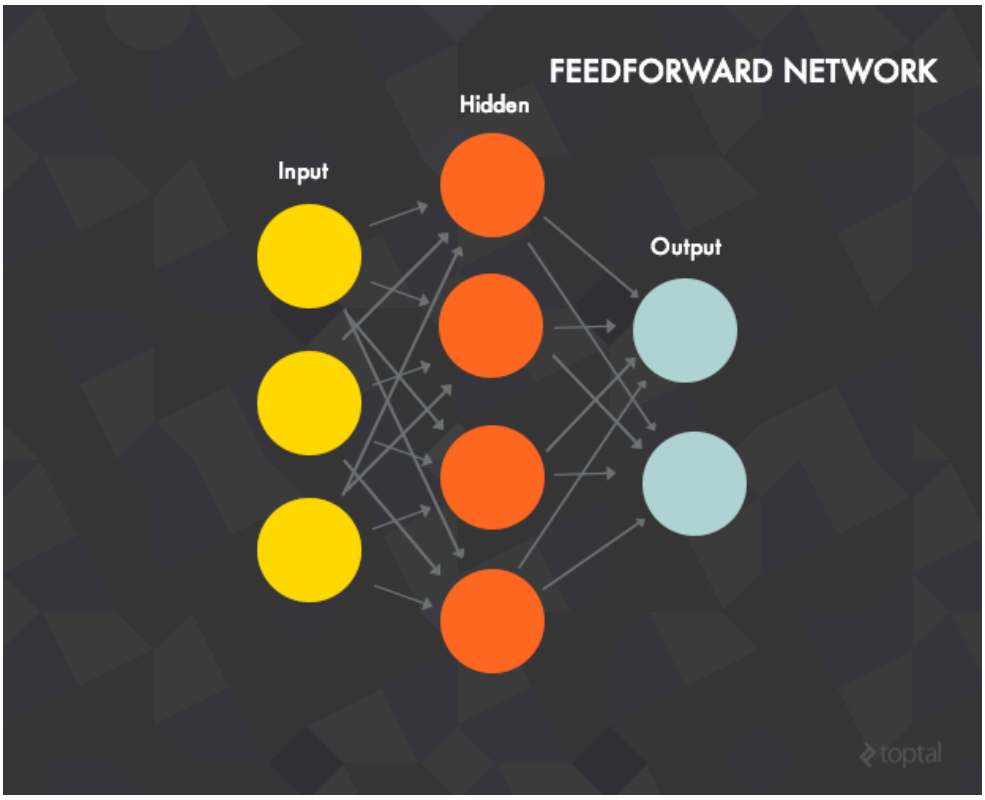
这种简单的感知机有一个明显缺陷：只能学习线性可分函数。这个缺陷重要吗？比如 XOR，这么简单的函数，都不能被线性分类器分类（如下图所示，分隔两类点失败）：



为了解决这个问题，我们要使用一种多层感知机，也就是——前馈神经网络：事实上，我们将要组合一群这样的感知机来创建出一个更强大的学习机器。

前馈神经网络

神经网络实际上就是将大量之前讲到的感知机进行组合，用不同的方法进行连接并作用在不同的激活函数上。



我们简单介绍下前向神经网络，其具有以下属性：

- 一个输入层，一个输出层，一个或多个隐含层。上图所示的神经网络中有一个三神经元的输入层、一个四神经元的隐含层、一个二神经元的输出层。
- 每一个神经元都是一个上文提到的感知机。
- 输入层的神经元作为隐含层的输入，同时隐含层的神经元也是输出层神经元的输入。
- 每条建立在神经元之间的连接都有一个权重 *w*（与感知机中提到的权重类似）。
- 在 *t* 层的每个神经元通常与前一层（*t - 1 层*）中的每个神经元都有连接（但你可以通过将这条连接的权重设为 0 来断开这条连接）。
- 为了处理输入数据，将输入向量赋到输入层中。在上例中，这个网络可以计算一个 3 维输入向量（由于只有 3 个输入层神经元）。假如输入向量是 [7, 1, 2]，你将第一个输入神经元输入 7，中间的输入 1，第三个输入 2。这些值将被传播到隐含层，通过加权传递函数传给每一个隐含层神经元（这就是前向传播），隐含层神经元再计算输出（激活函数）。
- 输出层和隐含层一样进行计算，输出层的计算结果就是整个神经网络的输出。

超线性

如果每一个感知机都只能使用一个线性激活函数会怎么样？整个网络的最终输出也仍然是将输入数据通过一些线性函数计算过一遍，只是用一些在网络中 收集的不同权值调整了一下。换名话说，再多线性函数的组合还是线性函数。如果我们限定只能使用线性激活函数的话，前馈神经网络其实比一个感知机强大不到哪 里去，无论网络有多少层。

正是这个原因，大多数神经网络都是使用的非线性激活函数，如对数函数、双曲正切函数、阶跃函数、整流函数等。不用这些非线性函数的神经网络只能学习输入数据的线性组合。

训练

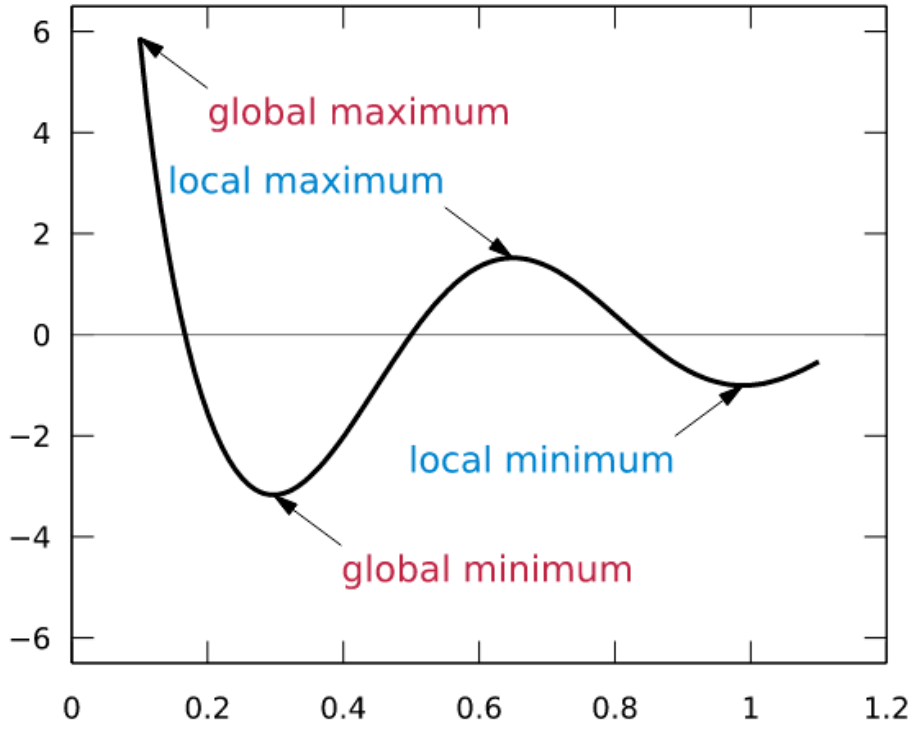
大多数常见的应用在多层感知机的监督式训练的算法都是反向传播算法。基本的流程如下：

1、将训练样本通过神经网络进行前向传播计算。

2、计算输出误差，常用均方差：

$$E = \frac{1}{2}(t - y)^2$$

其中 *t* 是目标值， *y* 是实际的神经网络计算输出。其它的误差计算方法也可以，但 MSE（均方差）通常是一种较好的选择。



3、网络误差通过随机梯度下降的方法来最小化。

梯度下降很常用，但在神经网络中，输入参数是一个训练误差的曲线。每个权重的最佳值应该是误差曲线中的全局最小值（上图中的 *global minimum*）。在训练过程中，权重以非常小的步幅改变（在每个样本或每小组样本训练完成后）以找到全局最小值，但这可不容易，训练通常会结束在局部最小值上（上图中的 local minima）。如例子中的，如果当前权重值为 0.6，那么要向 0.4 方向移动。

这个图表示的是最简单的情况，误差只依赖于单个参数。但是，网络误差依赖于每一个网络权重，误差函数非常、非常复杂。

好消息是反向传播算法提供了一种通过利用输出误差来修正两个神经元之间权重的方法。关系本身十分复杂，但对于一个给定结点的权重修正按如下方法（简单）：

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}$$

其中 *E* 是输出误差， *w_i* 是输入 *i* 的权重。

实质上这么做的目的是利用权重 *i* 来修正梯度的方向。关键的地方在于误差的导数的使用，这可不一定好计算：你怎么样能给一个大型网络中随机一个结点中的随机一个权重求导数呢？

答案是：通过反向传播。误差的首次计算很简单（只要对预期值和实际值做差即可），然后通过一种巧妙的方法反向传回网络，让我们有效的在训练过程中修正权重并（期望）达到一个最小值。

隐含层

隐含层十分有趣。根据普适逼近原理，一个具有有限数目神经元的隐含层可以被训练成可逼近任意随机函数。换句话说，一层隐含层就强大到可以学习任何函数了。这说明我们在多隐含层（如深度网络）的实践中可以得到更好的结果。

隐含层存储了训练数据的内在抽象表示，和人类大脑（简化的类比）保存有对真实世界的抽象一样。接下来，我们将用各种方法来搞一下这个隐含层。

一个网络的例子

可以看一下这个通过 *testMLPSigmoidBP* 方法用 Java 实现的简单(4-2-3)前馈神经网络，它将 IRIS 数据集进行了分类。这个数据集中包含了三类鸢尾属植物，特征包括花萼长度，花瓣长度等等。每一类提供 50 个样本给这个神经网络训练。特征被赋给输入神经元， 每一个输出神经元代表一类数据集（“1/0/0” 表示这个植物是 Setosa，“0/1/0”表示 Versicolour，而“0/0/1”表示 Virginica）。分类的错误率是 2/150（即每分类 150 个，错 2 个）。

大规模网络中的难题

神经网络中可以有多个隐含层：这样，在更高的隐含层里可以对其之前的隐含层构建新的抽象。而且像之前也提到的，这样可以更好的学习大规模网络。增加隐含层的层数通常会导致两个问题：

1、梯度消失：随着我们添加越来越多的隐含层，反向传播传递给较低层的信息会越来越少。实际上，由于信息向前反馈，不同层次间的梯度开始消失，对网络中权重的影响也会变小。

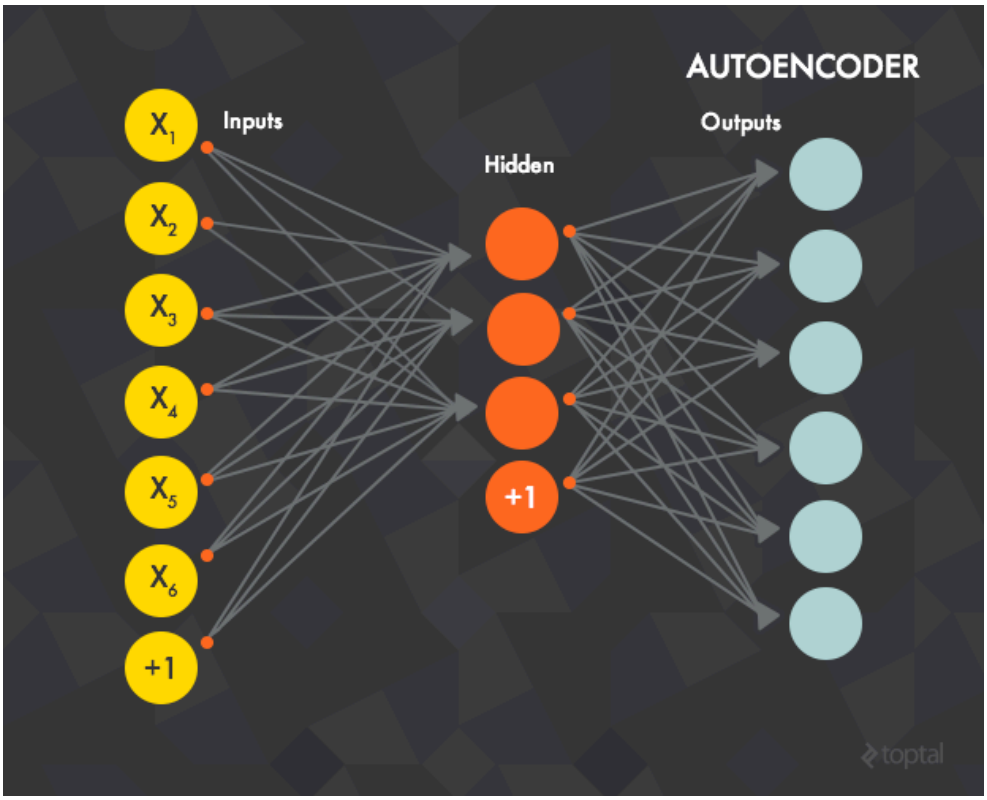
2、过度拟合：也许这是机器学习的核心难题。简要来说，过度拟合指的是对训练数据有着过于好的识别效果，这时导至模型非常复杂。这样的结果会导致对训练数据有非常好的识别较果，而对真实样本的识别效果非常差。

下面我们来看看一些深度学习的算法是如何面对这些难题的。

自编码器

大多数的机器学习入门课程都会让你放弃前馈神经网络。但是实际上这里面大有可为——请接着看。

自编码器就是一个典型的前馈神经网络，它的目标就是学习一种对数据集的压缩且分布式的表示方法（编码思想）。



从概念上讲，神经网络的目的是要训练去“重新建立”输入数据，好像输入和目标输出数据是一样的。换句话说：你正在让神经网络的输出与输入是同一样东西，只是经过了压缩。这还是不好理解，先来看一个例子。

压缩输入数据：灰度图像

这里有一个由 28x28 像素的灰度图像组成的训练集，且每一个像素的值都作为一个输入层神经元的输入（这时输入层就会有 784 个神经元）。输出层神经元要有相同的数目（784），且每一个输出神经元的输出值和输入图像的对应像素灰度值相同。

在这样的算法架构背后，神经网络学习到的实际上并不是一个训练数据到标记的“映射”，而是去学习数据本身的内在结构和特征（也正是因为这，隐含 层也被称作特征探测器(feature detector)）。通常隐含层中的神经元数目要比输入/输入层的少，这是为了使神经网络只去学习最重要的特征并实现特征的降维。

我们想在中间层用很少的结点去在概念层上学习数据、产生一个紧致的表示方法。

流行感冒

为了更好的描述自编码器，再看一个应用。

这次我们使用一个简单的数据集，其中包括一些感冒的症状。如果感兴趣，这个例子的源码发布在这里。

数据结构如下：

-

- 输入数据一共六个二进制位
- 前三位是病的证状。例如，*1 0 0 0 0 0*代表病人发烧；*0 1 0 0 0 0*代表咳嗽；*1 1 0 0 0 0*代表即咳嗽又发烧等等。
- 后三位表示抵抗能力，如果一个病人有这个，代表他/她不太可能患此病。例如，*0 0 0 1 0 0*代表病人接种过流感疫苗。一个可能的组合是：*0 1 0 1 0 0*，这代表着一个接种过流感疫苗的咳嗽病人，等等。

当一个病人同时拥用前三位中的两位时，我们认为他生病了；如果至少拥用后三位中的两位，那么他是健康的，如：

-

  - *111000, 101000, 110000, 011000, 011100* = 生病
  - *000111, 001110, 000101, 000011, 000110* = 健康

我们来训练一个自编码器（使用反向传播），六个输入、六个输出神经元，而只有两个隐含神经元。

在经过几百次迭代以后，我们发现，每当一个“生病”的样本输入时，两个隐含层神经元中的一个（对于生病的样本总是这个）总是显示出更高的激活值。而如果输入一个“健康”样本时，另一个隐含层则会显示更高的激活值。

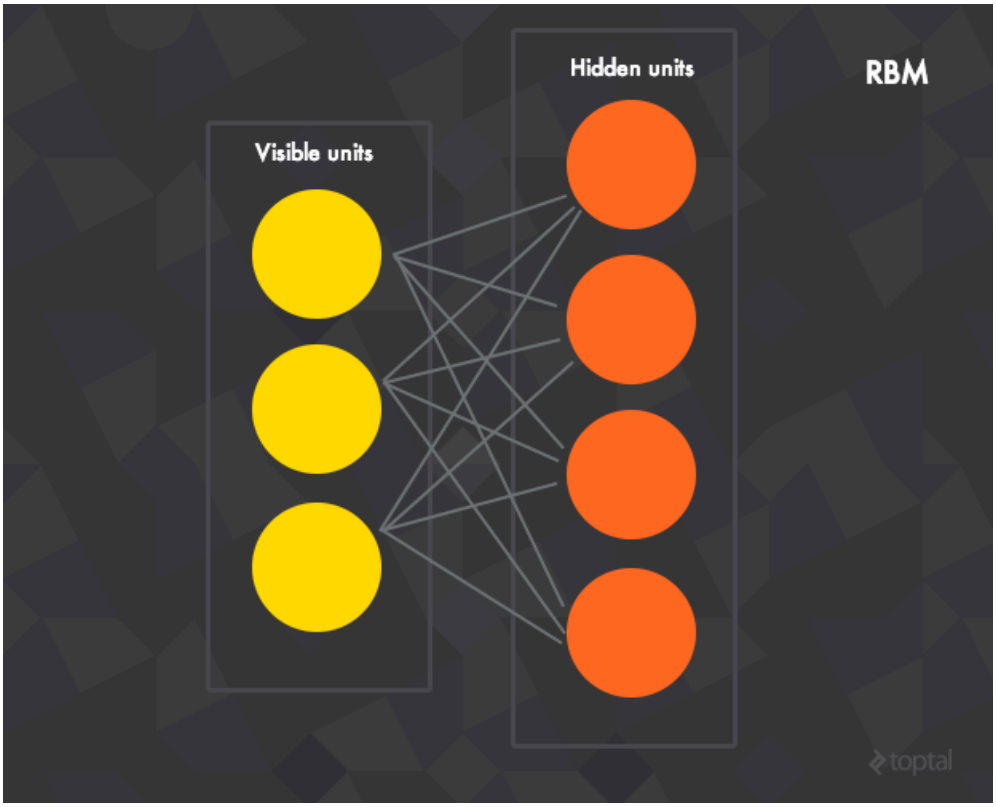
再看学习

本质上来说，这两个隐含神经元从数据集中学习到了流感症状的一种紧致表示方法。为了检验它是不是真的实现了学习，我们再看下过度拟合的问题。通过训练我们的神经网络学习到的是一个紧致的简单的，而不是一个高度复杂且对数据集过度拟合的表示方法。

某种程度上来讲，与其说在找一种简单的表示方法，我们更是在尝试从“感觉”上去学习数据。

受限波尔兹曼机

下一步来看下受限波尔兹曼机（Restricted Boltzmann machines RBM），一种可以在输入数据集上学习概率分布的生成随机神经网络。



RBM 由隐含层、可见层、偏置层组成。和前馈神经网络不同，可见层和隐含层之间的连接是无方向性（值可以从可见层->隐含层或隐含层 ->可见层任意传输）且全连接的（每一个当前层的神经元与下一层的每个神经元都有连接——如果允许任意层的任意神经元连接到任意层去，我们就得到了一个波尔兹曼机（非受限的））。

标准的 RBM 中，隐含和可见层的神经元都是二态的（即神经元的激活值只能是服从伯努力分布的 0 或 1），不过也存在其它非线性的变种。

虽然学者们已经研究 RBM 很长时间了，最近出现的对比差异无监督训练算法使这个领域复兴。

对比差异

单步对比差异算法原理：

1、正向过程：

-
  - 输入样本 *v* 输入至输入层中。
  - *v* 通过一种与前馈网络相似的方法传播到隐含层中，隐含层的激活值为 *h*。

2、反向过程：

-
  - 将 *h* 传回可见层得到 *v′*（可见层和隐含层的连接是无方向的，可以这样传）。
  - 再将 *v′* 传到隐含层中，得到 *h′*。

3、权重更新：

$$w(t+1) = w(t) + a\left(\mathbf{v}\mathbf{h}^{\intercal} - \mathbf{v}'\mathbf{h}'^{\intercal}\right)$$

其中 *a* 是学习速率， *v*, *v'*, *h*, *h'* 和 *w* 都是向量。

算法的思想就是在正向过程中影响了网络的内部对于真实数据的表示。同时，反向过程中尝试通过这个被影响过的表示方法重建数据。主要目的是可以使生成的数据与原数据尽可能相似，这个差异影响了权重更新。

换句话说，这样的网络具有了感知对输入数据表示的程度的能力，而且尝试通过这个感知能力重建数据。如果重建出来的数据与原数据差异很大，那么进行调整并再次重建。

再看流行感冒的例子

为了说明对比差异，我们使用与上例相同的流感症状的数据集。测试网络是一个包含 6 个可见层神经元、2 个隐含层神经元的 RBM。我们用对比差异的方法对网络进行训练，将症状 *v* 赋到可见层中。在测试中，这些症状值被重新传到可见层；然后再被传到隐含层。隐含层的神经元表示健康/生病的状态，与自编码器相似。

在进行过几百次迭代后，我们得到了与自编码器相同的结果：输入一个生病样本，其中一个隐含层神经元具有更高激活值；输入健康的样本，则另一个神经元更兴奋。

例子的代码在这里。

深度网络

到现在为止，我们已经学习了隐含层中强大的特征探测器——自编码器和 RBM，但现在还没有办法有效的去利用这些功能。实际上，上面所用到的这些数据集都是特定的。而我们要找到一些方法来间接的使用这些探测出的特征。

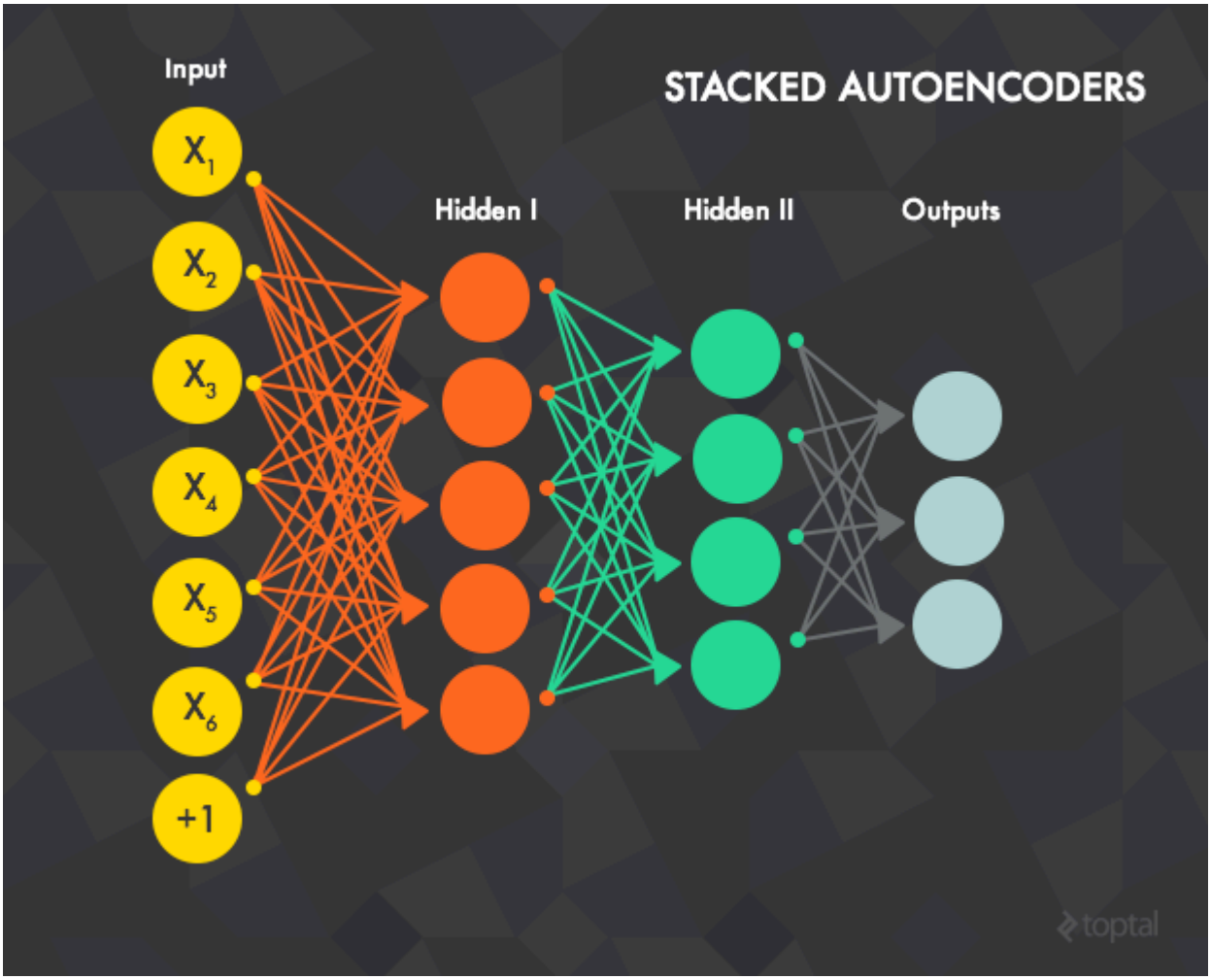
好消息是，已经发现这些结构可以通过栈式叠加来实现深度网络。这些网络可以通过贪心法的思想训练，每次训练一层，以克服之前提到在反向传播中梯度消失及过度拟合的问题。

这样的算法架构十分强大，可以产生很好的结果。如 Google 著名的“猫”识别，在实验中通过使用特定的深度自编码器，在无标记的图片库中学习到人和猫脸的识别。

下面我们将更深入。

栈式自编码器

和名字一样，这种网络由多个栈式结合的自编码器组成。



自编码器的隐含层 *t* 会作为 *t + 1* 层的输入层。第一个输入层就是整个网络的输入层。利用贪心法训练每一层的步骤如下：

1、通过反向传播的方法利用所有数据对第一层的自编码器进行训练（*t=1*，上图中的红色连接部分）。

2、训练第二层的自编码器 *t=2*（绿色连接部分）。由于 *t=2* 的输入层是 *t=1* 的隐含层，我们已经不再关心 *t=1* 的输入层，可以从整个网络中移除。整个训练开始于将输入样本数据赋到 *t=1* 的输入层，通过前向传播至 t = 2 的输出层。下面 t = 2 的权重（输入->隐含和隐含->输出）使用反向传播的方法进行更新。t = 2 的层和 t=1 的层一样，都要通过所有样本的训练。

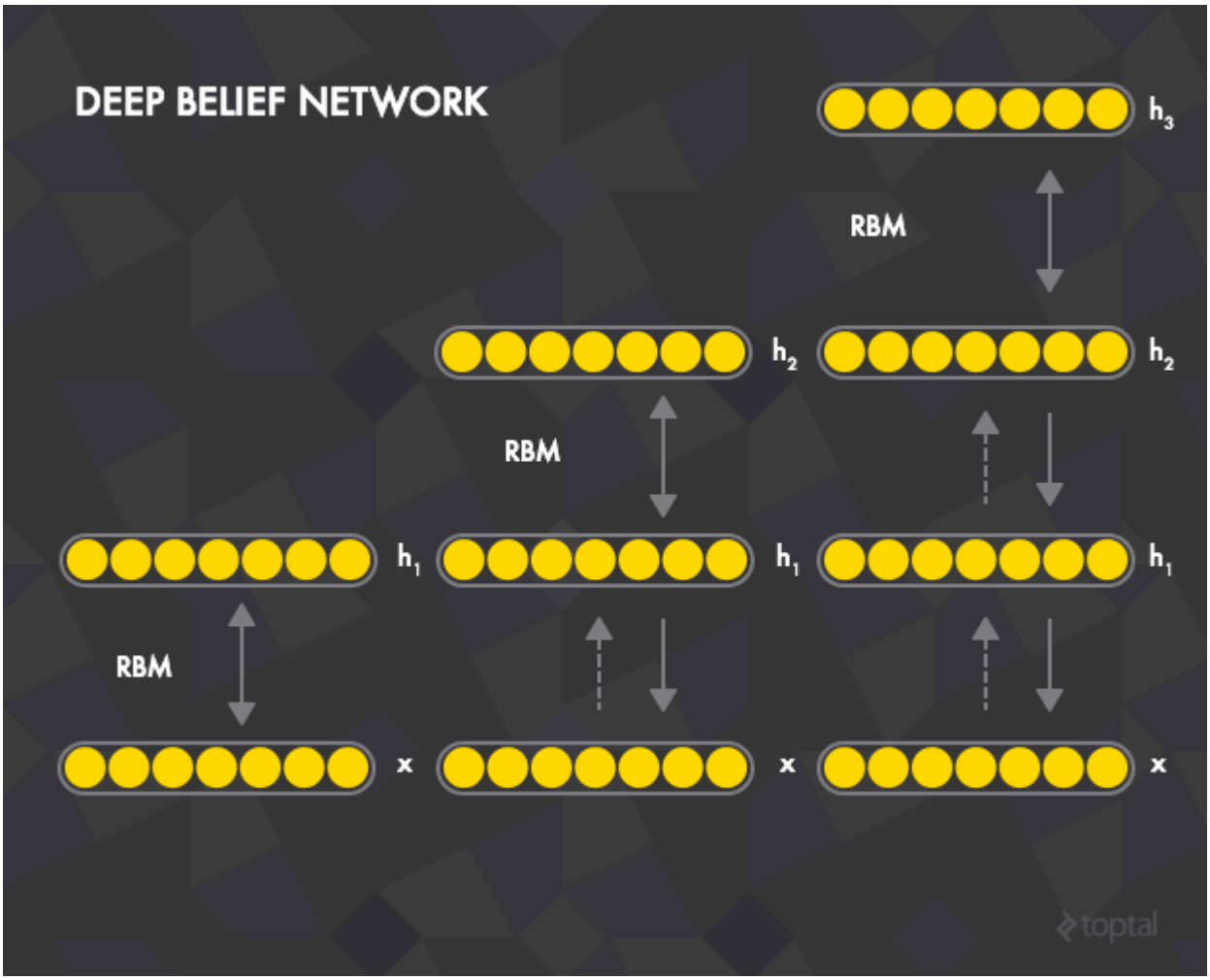
3、对所有层重复步骤 1-2（即移除前面自编码器的输出层，用另一个自编码器替代，再用反向传播进行训练）。

4、步骤 1-3 被称为预训练，这将网络里的权重值初始化至一个合适的位置。但是通过这个训练并没有得到一个输入数据到输出标记的映射。例 如，一个网络的目标是被训练用来识别手写数字，经过这样的训练后还不能将最后的特征探测器的输出（即隐含层中最后的自编码器）对应到图片的标记上去。这 样，一个通常的办法是在网络的最后一层（即蓝色连接部分）后面再加一个或多个全连接层。整个网络可以被看作是一个多层的感知机，并使用反向传播的方法进行 训练（这步也被称为微调）。

栈式自编码器，提供了一种有效的预训练方法来初始化网络的权重，这样你得到了一个可以用来训练的复杂、多层的感知机。

深度信度网络

和自编码器一样，我也可以将波尔兹曼机进行栈式叠加来构建深度信度网络（DBN）。



在本例中，隐含层 RBM *t* 可以看作是 RBM *t+1* 的可见层。第一个 RBM 的输入层即是整个网络的输入层，层间贪心式的预训练的工作模式如下：

1. 通过对比差异法对所有训练样本训练第一个 RBM *t=1*

2. 训练第二个 RBM *t=1*。由于 *t=2* 的可见层是 *t=1* 的隐含层，训练开始于将数据赋至 *t=1* 的可见层，通过前向传播的方法传至 *t=1* 的隐含层。然后作为 *t=2* 的对比差异训练的初始数据。

3. 对所有层重复前面的过程。

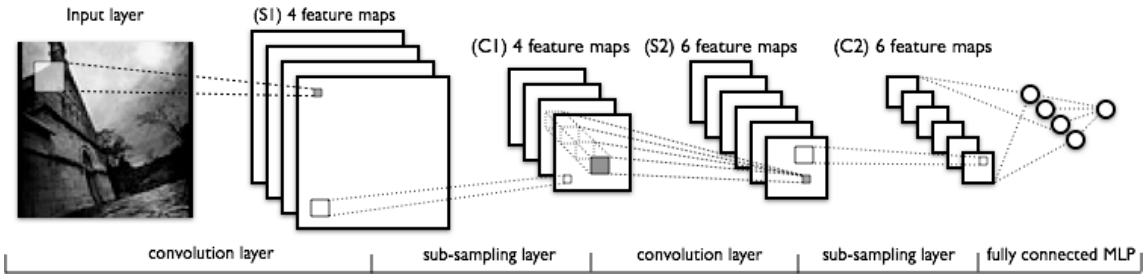
4. 和栈式自编码器一样，通过预训练后，网络可以通过连接到一个或多个层间全连接的 RBM 隐含层进行扩展。这构成了一个可以通过反向传播进行微调的多层感知机。

本过程和栈式自编码器很相似，只是用 RBM 将自编码器进行替换，并用对比差异算法将反向传播进行替换。

(注: 例中的源码可以从 此处 获得.)

卷积网络

这个是本文最后一个软件架构——卷积网络，一类特殊的对图像识别非常有效的前馈网络。



在我们深入看实际的卷积网络之臆，我们先定义一个图像滤波器，或者称为一个赋有相关权重的方阵。一个滤波器可以应用到整个图片上，通常可以应用 多个滤波器。比如，你可以应用四个 6x6 的滤波器在一张图片上。然后，输出中坐标（1,1）的像素值就是输入图像左上角一个 6x6 区域的加权和，其它像素 也是如此。

有了上面的基础，我们来介绍定义出卷积网络的属性：

- **卷积层** 对输入数据应用若干滤波器。比如图像的第一卷积层使用 4 个 6x6 滤波器。对图像应用一个滤波器之后 的得到的结果被称为特征图谱（feature map, FM），特征图谱的数目和滤波器的数目相等。如果前驱层也是一个卷积层，那么滤波器应用在 FM 上，相当于输入一个 FM，输出另外一个 FM。从直觉上来讲， 如果将一个权重分布到整个图像上后，那么这个特征就和位置无关了，同时多个滤波器可以分别探测出不同的特征。
- **下采样层** 缩减输入数据的规模。例如输入一个 32x32 的图像，并且通过一个 2x2 的下采样，那么可以得到一个 16x16 的输出图像，这意味着原图像上的四个像素合并成为输出图像中的一个像素。实现下采样的方法有很多种，最常见的是最大值合并、平均值合并以及随机合并。
- 最后一个下采样层（或卷积层）通常连接到一个或多个全连层，全连层的输出就是最终的输出。
- 训练过程通过改进的反向传播实现，将下采样层作为考虑的因素并基于所有值来更新卷积滤波器的权重。

可以在这看几个应用在 MNIST 数据集上的卷积网络的例子，在这还有一个用 JavaScript 实现的一个可视的类似网络。

实现

目前为止，我们已经学会了常见神经网络中最主要的元素了，但是我只写了很少的在实现过程中所遇到的挑战。

概括来讲，我的目标是实现一个深度学习的库，即一个基于神经网络且满足如下条件的框架：

-
    - 一个可以表示多种模型的通用架构（比如所有上文提到的神经网络中的元素）

- 可以使用多种训练算法（反向传播，对比差异等等）。
- 体面的性能

为了满足这些要求，我在软件的设计中使用了分层的思想。

结构

我们从如下的基础部分开始：

-

  - NeuralNetworkImpl 是所有神经网络模型实现的基类。
  - 每个网络都包含有一个 layer 的集合。
  - 每一层中有一个 connections 的链表， connection 指的是两个层之间的连接，将整个网络构成一个有向无环图。

这个结构对于经典的反馈网络、RBM 及更复杂的如 ImageNet 都已经足够灵活。

这个结构也允许一个 layer 成为多个网络的元素。比如，在 Deep Belief Network（深度信度网络）中的 layer 也可以用在其 RBM 中。

另外，通过这个架构可以将 DBN 的预训练阶段显示为一个栈式 RBM 的列表，微调阶段显示为一个前馈网络，这些都非常直观而且程序实现的很好。

数据流

下个部分介绍网络中的数据流，一个两步过程:

1. 定义出层间的序列。例如，为了得到一个多层感知机的结果，输入数据被赋到输入层（因此，这也是首先被计算的层），然后再将数据通过不同的方法流向输出层。为了在反向传播中更新权重，输出的误差通过广度优先的方法从输出层传回每一层。这部分通过 LayerOrderStrategy 进行实现，应用到了网络图结构的优势，使用了不同的图遍历方法。其中一些样例包含了 广度优先策略 和 定位到一个指定的层。层的序列实际上由层间的连接进行决定，所以策略部分都是返回一个连接的有序列表。
2. 计算激活值。每一层都有一个关联的 ConnectionCalculator，包含有连接的列表（从上一步得来）和输入值（从其它层得到）并计算得到结果的激活值。例如，在一个简单的 S 形前馈网络中，隐含层的 *ConnectionCalculator* 接受输入层和偏置层的值（分别为输入值和一个值全为 1 的数组）和神经元之间的权重值（如果是全连接层，权重值实际上以一个矩阵的形式存储在一个 FullyConnected 结构中，计算加权和，然后将结果传给 S 函数。*ConnectionCalculator* 中实现了一些转移函数（如加权求和、卷积）和激活函数（如对应多层感知机的对数函数和双曲正切函数，对应 RBM 的二态函数）。其中的大部分都可以通过 Aparapi 在 GPU 上进行计算，可以利用迷你批次训练。

通过 Aparapi 进行 GPU 计算

像我之前提到的，神经网络在近些年复兴的一个重要原因是其训练的方法可以高度并行化，允许我们通过 GPGPU 高效的加速训练。本文中，我选择 Aparapi 库来进行 GPU 的支持。

Aparapi 在连接计算上强加了一些重要的限制：

- 只允许使用原始数据类型的一维数组（变量）。
- 在 GPU 上运行的程序只能调用 Aparapi *Kernel* 类本身的成员函数。

这样，大部分的数据（权重、输入和输出数据）都要保存在 Matrix 实例里面，其内部是一个一维浮点数组。所有 Aparapi 连接计算都是使用 AparapiWeightedSum （应用在全连接层和加权求和函数上）、 AparapiSubsampling2D （应用在下采样层）或 AparapiConv2D （应用在卷积层）。这些限制可以通过 Heterogeneous System Architecture 里介绍的内容解决一些。而且 Aparapi 允许相同的代码运行在 CPU 和 GPU 上。

训练

training 的模块实现了多种训练算法。这个模块依赖于上文提到的两个模块。比如，BackPropagationTrainer （所有的训练算法都以 Trainer 为基类）在前馈阶段使用前馈层计算，在误差传播和权重更新时使用特殊的广度优先层计算。

我最新的工作是在 Java8 环境下开发，其它一些更新的功能可以在这个 branch 下获得，这部分的工作很快会 merge 到主干上。

结论

本文的目标是提供一个深度学习算法领域的一个简明介绍，由最基本的组成元素开始（感知机）并逐渐深入到多种当前流行且有效的架构上，比如受限波尔兹曼机。

神经网络的思想已经出现了很长时间，但是今天，你如果身处机器学习领域而不知道深度学习或其它相关知识是不应该的。不应该过度宣传，但不可否认 随着 GPGPU 提供的计算能力、包括 Geoffrey Hinton, Yoshua Bengio, Yann LeCun and Andrew Ng 在内的研究学者们提出的高效算法，这个领域已经表现出了很大的希望。现在正是最佳的时间深入这些方面的学习。

附录：相关资源

如果你想更深入的学习，下面的这些资源在我的工作当中都起过重要的作用：

- DeepLearning.net: 深度学习所有方面知识的一个门户。里面有完善的手册、软件库 和一个非常好的 阅读列表。
- 活跃的 Google+ 社区.
- 两个很好的课程: Machine Learning and Neural Networks for Machine Learning, 都在 Coursera 上。
- The Stanford neural networks tutorial，斯坦福神经网络指南。