# 一篇文章带你了解深度神经网络

[](https://www.zhihu.com/people/zhang-si-jun-52)

[张斯俊](https://www.zhihu.com/people/zhang-si-jun-52" \t "/home/admin/文档\\x/_blank)

愿成为一把梯子，助你跨过无数的坑。

从这篇开始，将会进入**深度**强化学习。所谓**深度**强化学习，其实就是用**深度神经网络**辅助强化学习。

但有同学可能会对深度神经网络还不是很了解。这里强烈推荐同学看一下吴恩达老师的课程。对于入门同学非常友好。

但如果同学希望对神经网络有一个梗概的了解，那可以看看本篇。本篇将以**手写数字识别**作为示例，看一下如何直观理解深度神经网络。

### 神奇的深度神经网络

不知道大家有没有发现，在日常生活中，明明知道一些东西和另外一些东西是有关联的，但却无法描述。

举个例子： 手写数字的辨识：明明知道一个手写数字是什么，但却没有办法用语言描述。

数字8



不同人写的数字8，长得都很像，所以大家能够一眼分辨出来8长的是什么样子。 但具体来说，每个8的样子都长的不太一样：虽然知道8由两个圈圈组成，一上一下，但精确到某个像素应该白色还是黑色？圈圈的大小如何？一定要完美的圈圈才是数字8吗？这些问题却很难回答。

再举个例子，语音翻译： 希望记录人们的语音，然后翻译为对应的文字。 其实明知道语音和文字之间是有关系的，不同人对同样的文字发音不完全相同，但人类是能识别并翻译出来。 确没有办法精确描述某个发音（声纹）准确获得对应的文字。

不能描述的后果很严重，这以为着没有办法用代码的形式把规则写下来。也就意味着，不能把这部分工作**自动化**。

这就是为什么的机器能够替代体力劳动的工作，但不能替代脑力劳动。这是因为脑力劳动的工作，相对体力劳动要更复杂。

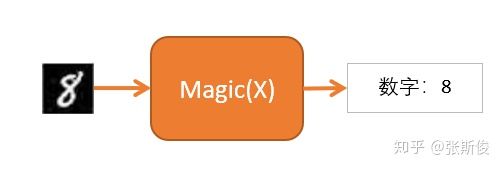
但深度学习出现后，改变了这个想法。

### 超简深度神经网原理

假如，明明知道X，y有关系，那么不妨先设这个关系可以通过函数Magic(X)获得。也就是说Magic(X)=y。

这在手写数字识别中，X就是需要识别的图片，y就是识别出来的数字分类。

的任务就是需要求这个Magic函数。



在传统方法，需要写一些系列规则来表达这个函数。但在深度神经网络，会改变一下思路。

现在假设有另外一个函数Magic'(),这个函数是由深度神经网络构成。至于是怎样构成，后面再说。

在刚开始的时候，很明显Magic'(X) 并不等于y，例如输入手写图片8，Magic'()计算后，认为数字8只有20%，但数字9有40%。

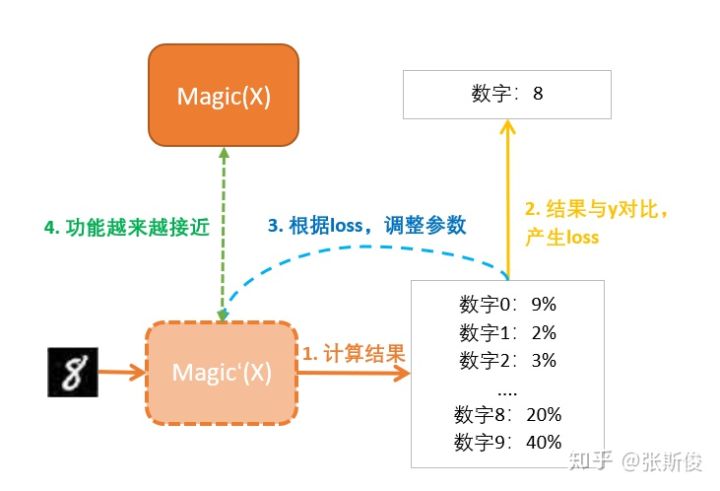
...但这没所谓，因为这是刚开始。的任务是让Magic'(X)产生的结果y' 和 y尽量接近。

y'和真实y之间的差距，叫损失，也就是loss。有时候也会把y称为目标(target)，因为的任务就是让Magic'(X)越来越靠近这个目标。

衡量loss的方法有很多，定义不同loss对神经网络学习有着重大差别，这个话题太大，暂时不展开。

loss越大，表示和目标差距越远；loss越小，表示和目标越近，当小到一定值，那么就可以认为Magic'(X)和要的magic(X)函数非常接近，因为可以通过Magic'(X)计算出y。

当有许许多多这样的y，经过许许多多轮后。Magic'就越来越贴近Magic。也就是说X和y之间的关系就能越来越好地表达出来。



最后，就说这个Magic函数，学习到某种能力。

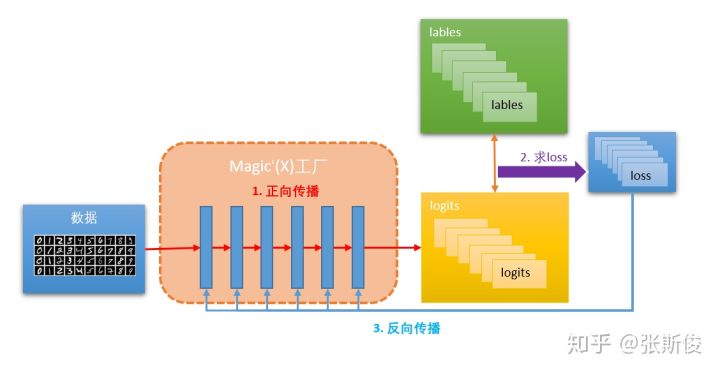
当然，上面的描述实在太简化，这个Magic函数能否训练出来，选取的loss函数，神经网络的结构，数据的大小，质量等很多很多学问。如果想入门，再次强烈推荐吴恩达老师的课程。

但目前把关注点放在强化学习的算法，不妨先把神经网络看成一个万能的Magic函数。后续在分析代码的时候，会把一些简单的原理介绍给大家。

### 放大镜下深度神经网络

为了用代码构建深度神经网络，有必要解剖一下，深度神经网络到底是怎样构造的。

现在可以把深度神经网络的Magic函数，看成是一个数据加工厂。而X就是要进行加工的数据。



为了让这个数据加工厂运行得更快，通常需要把要加工的数据X变得更‘标准’一些。

例如图片的尺寸大小，有多少通道的颜色等等，然后分批(一般称为batch)，输入工厂。

在输入工厂的时候，会有一个‘大门’，称为输入层，去检查数据是否已经按照工厂的标准整理好。

数据工厂里有很多车间，按照流水线排列。和一般的自动化车间一样，需要定义好这个车间的操作标准。

一般称这些车间叫**层**。这些层都已经封装好在tensorflow、tensorlayer、pytorch等里面了。常用的层包括：Dense、Conv2D、LSTM、Reshape、Flatten等。

最终，数据工厂会把原数据X，加工成产品y'(也叫做：logits)。从源数据加工成产品的过程，叫**正向传播**。

但产品y'是否是一个合格的产品，还需要真正的y(也叫做:lables)作为标准去鉴定。把鉴定出来的差距就是loss。

工厂根据鉴定结果，以**梯度下降**的方式，**反向传递**给每个车间，告诉车间要如何调整各自的参数，让源数据和产出y'能够对应起来。

经过N个批次（batch）的数据输入，然后鉴别，工厂调整。最后工厂就能达到的生产标准了。也就是说magic函数已经被训练好了。

### 构建深层神经网络

以fashion\_mnist为例，看看应该怎样构建深层的神经网络。

其实构建深层神经网络的方法是多种多样的，有许多更为简便的方式，例如keras。 在这里，以tensorlayer作为例子，主要有两个考量：

1. 在往后的示例代码，基本上都用tensorlayer，可以借此熟悉一下；
2. 这里用上了自动求导的方式，会更加灵活，也让大家看清楚这个基本流程。

### Fahsion\_MNIST 数据介绍



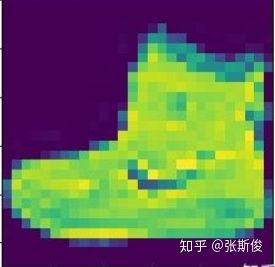
Fashion MNIST是一个非常适合入门，又好玩的数据集，做完之后保证成就感满满。

一般来说，数据集包含了训练集和测试集。如图：



在有监督学习下，一般会把数据分为训练集和测试集。就如名字一样，训练集就是用来训练模型的，测试集就是为了测试模型是否真的管用的。

其中每种数据集合中包含了图片和标签，例如：



图片，对应的标签是[鞋子]分类。

当然并不是所有的数据集，都是图片，也有可能是数据。例如天气预测，数据可能是以往的天气数据，例如前一天的气温，气压，湿度等。标签就是当天的天气。

在Fashion MNIST里，训练集和测试集里面的数据，都已经为大家对应好了。

现在要训练一个神经网络，用训练集的数据，学习根据图像，去判别这是什么服装类型。然后把这个神经网络用在测试集上，看看正确率有多少。

**加载数据集**

现在，可以用这么一个语句，来加载Fashion MNIST数据集

(train\_images, train\_labels), (test\_images, test\_labels) = keras.datasets.fashion\_mnist.load\_data()

train\_images：训练集图片 train\_labels：训练集标签 test\_images：测试集图片 test\_labels：测试集标签

print(train\_images.shape)

print(train\_labels.shape)

print(test\_images.shape)

print(test\_labels.shape)

打印结果：

(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)

也就是说 训练集含有：60,000张图像，对应60,000个标签； 测试集含有：10,000张图像对应10,000个标签。

而每张图的大小为 28 x 28。

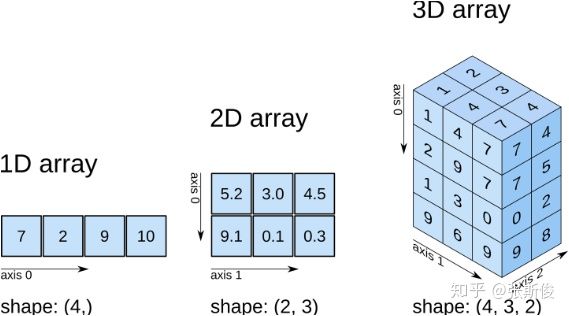
### 数据预处理

注意：在神经网络的相关编程里面，有两个东西大家必须要留意，一个是数据的shape，一个是数据的type。

输入神经网络的数据，往往不是一个单一的数值。而是很多数值组成的形状。

例如一张图28x28大小灰度图。从数据层面来看它是由28x28个数值表示表示出来的。那么说这个数据的shape就是28x28。

说一张灰度图是二维的，也有三维数据，例如彩色图片，是由RGB三层数据表示。



当然，也有更高纬度的数据。数据在输入到神经网络之前，都要整理好形状。

而说的数据的type，是指数据的每一个数值的类型。一般来说，会用32位浮点数(float32)。

在Fashion\_Mnist每张图的大小为28x28，每个像素值处于0到255之间。

因为的像素，除以0到255之间，因此需要除以255，让数据保持在0-1之间。

### 建立深度神经网络的一般步骤

1、**创建一个深度神经网络和优化器**

def get\_model(inputs\_shape):

inputs = tl.layers.Input(inputs\_shape)

cnn = tl.layers.Conv2d(n\_filter=32,filter\_size=(3,3),strides=(1,1),act='relu',padding='SAME')(inputs)

flatten = tl.layers.Flatten()(cnn)

fc = tl.layers.Dense(n\_units=128, act='relu')(flatten)

outputs = tl.layers.Dense(n\_units=10,act=tf.nn.softmax)(fc)

return tl.models.Model(inputs=inputs, outputs=outputs)

这里，采用的是函数式的方式建立深度神经网络。

之前讲过，神经网络的层，就相当于一个流水线工厂的车间，每个车间独立承担了对数据的一个操作。

inputs = tl.layers.Input(inputs\_shape) 输入层相当于工厂的大门，检测输入数据是否符合工厂定义的标准。在Fashion MNIST中，的输入层需要接受一个batch大小的数据，每个数据是28 x 28 大小。batch大小暂时不设定，可以预留，用None表示。所以，的imputs\_shapes=[None,28,28]

flatten = tl.layers.Flatten()(inputs) flatten层，需要把这28 x 28像素的图，变成一个长条形的数据，以方便后面接入全连接层。所以用上Flatten层，把数据变成[None,784]

fc = tl.layers.Dense(n\_units=128, act='relu')(flatten) 现在加上全连接层，全连接层有128个单元，和原来的784个单元连接。在前面inputs和flatten两层，并没有对数据内容进行加工。而Dense是一种对数据内容进行计算的操作。

outputs = tl.layers.Dense(n\_units=10,act=tf.nn.softmax)(fc) 在输出层，同样用上全连接层，输出10个单元，对应需要输出的10个分类，并用上softmax作为激活函数。

细心读代码的同学可能已经明白，以fc层为例：代码前一段：tl.layers.Dense(n\_units=128, act='relu')是相当于车间。后面(flatten)是上一层车间产出的结果。函数式的模型定义方式，就是把上一层的输出，放入到输入，从而连成一条数据的流水线。

tl.models.Model(inputs=inputs, outputs=outputs) 流水线都定义好了，还需要一行代码，定义好工厂。工厂的定义很简单，只需要把工厂的大门和输入定义好就可以了。

2、开始训练

在训练的时候，会用到tensorflow2.0的**自动求导机制**。

先定义梯度带，并在梯度带中，描述数据从源数据开始到产品logits的，然后通过loss函数，求loss的全过程。

with tf.GradientTape() as tape:

logits = network(images)

loss = tf.losses.sparse\_categorical\_crossentropy(labels,logits)

loss函数的定义是非常高深的学问，幸好，常用的loss函数已经被提取成工具，对于新手的只需要按需使用即可。

常用的loss函数有4个，在使用之前需要先考虑，在解决的是分类问题还是回归问题。 **分类问题**就是把原数据按照标签分类的任务： 一种是像正在解决的Fashion MNIST，是要分成多个类别的。这时候一般sparse\_categorical\_crossentropy 如果只需要分两类，那么可以用。 sigmoid\_cross\_entropy\_with\_logits

**回归问题**是试图从凌乱的数据中寻着规律的问题。 虽然不太准确，但大家可以想象一下：的神经网络在产生一个曲线，神经网络在训练的时候扭动曲线（或者是曲面或者是更高纬度的东西），去连接每一个数据点。

如果你不清楚，很简单，你发现不是分类问题，那么一般来说就是回归问题。

对于回归问题，可以MSE均值方差。 loss = tf.reduce\_mean(tf.square(logits, labels))

都定义loss的计算过程，就可以利用梯度带自动求导了。

grads = tape.gradient(loss, network.trainable\_weights)

optimizer.apply\_gradients(zip(grads,network.trainable\_weights))

第一行，是用梯度带求导。获得对网络中每个参数的导数。 第二行，用优化器，对参数进行求导。

optimizer = tf.optimizers.Adam()

而优化器（optimizer），在之前定义了使用Adam()，

这行代码可以变化的不多，如果大家不明白，可以先记起来。

### 展示训练效果

def predict():

acc = 0

for i in range(test\_lables.shape[0]):

if np.argmax(predictions[i])==test\_lables[i]:

acc += 1

print('accurcy:%f'%(acc/test\_lables.shape[0]))

最终，测试集准确率约90%。事实上这是个很一般的成绩，也有很多可以优化的地方。

这篇文章旨在给新手同学一些关于深度神经网络的指引，有兴趣的同学可以优化模型。

如果感觉到困难，也不用慌，将会在每个算法的实战篇中详细介绍，不断加深大家对**深度**强化学习的理解。

下一篇，将会回到DQN。

===========你的支持，就是在下努力的原动力===========

如果专栏对你有用，请点赞并关注在下喔。如果发现有问题，也可以在文章下留言。

你的每一点关注，都是在下的继续努力的动力来源！感激！

[白话强化学习​](https://zhuanlan.zhihu.com/c_1215667894253830144" \t "/home/admin/文档\\x/_blank)

zhuanlan.zhihu.com[](https://zhuanlan.zhihu.com/c_1215667894253830144)