神经网络

lypto

2018,June

摘要

现如今，机器学习技术越来越普及，特别是深度学习的概念也越来越深入人心。同样的，深度学习的相关理论和思想等在数据挖掘的领域也有起着举足轻重的作用。通过模拟人类的神经网络系统，深度学习一次又一次的创造着机器学习相关领域的奇迹。其在图像视觉，语义文本分析等的研究方向上更是大展身手。本文着眼于神经网络相关知识体系，从神经网络相关概述，神经网络的历史发展，常用神经网络模型概述等几个层面来展开叙述，在本文的最后会有一个运用keras以及tensorflow来实现一个关于图像识别领域的神经网络简单实践。

目录

1 神经网络的概述

1.1 神经网络的发展历史

二十世纪40年代后期，心理学家唐纳德·赫布根据神经可塑性的机制创造了一种对学习的假说，现在称作赫布型学习。赫布型学习被认为是一种典型的非监督式学习规则，它后来的变种是长期增强作用的早期模型。从1948年开始，研究人员将这种计算模型的思想应用到B型图灵机上。

法利和韦斯利·A·克拉克（1954）首次使用计算机，当时称作计算器，在MIT模拟了一个赫布网络。

弗兰克·罗森布拉特（1956）创造了感知机。这是一种模式识别算法，用简单的加减法实现了两层的计算机学习网络。罗森布拉特也用数学符号描述了基本感知机里没有的回路，例如异或回路。这种回路一直无法被神经网络处理，直到Paul Werbos(1975)创造了反向传播算法。

在马文·明斯基和西摩·帕尔特（1969）发表了一项关于机器学习的研究以后，神经网络的研究停滞不前。他们发现了神经网络的两个关键问题。第一是基本感知机无法处理异或回路。第二个重要的问题是计算机没有足够的能力来处理大型神经网络所需要的很长的计算时间。直到计算机具有更强的计算能力之前，神经网络的研究进展缓慢。

后来出现的一个关键的进展是反向传播算法（Werbos 1975）。这个算法有效地解决了异或的问题，还有更普遍的训练多层神经网络的问题。

在二十世纪80年代中期，分布式并行处理（当时称作联结主义）流行起来。David E. Rumelhart和James McClelland（1986）的教材对于联结主义在计算机模拟神经活动中的应用提供了全面的论述。神经网络传统上被认为是大脑中的神经活动的简化模型，虽然这个模型和大脑的生理结构之间的关联存在争议。人们不清楚人工神经网络能多大程度地反映大脑的功能。

在20世纪剩下的时间里，支持向量机和其它更简单的算法（如线性分类器）的流行程度逐步超过了神经网络。

1989年，Yann LeCun提出了一种用反向传导进行更新的卷积神经网络，称为LeNet。

1997年，Sepp Hochreiter和Jürgen Schmidhuber提出了长短期记忆网络。

1998年，以Yann LeCun为首的研究人员实现了一个七层的卷积神经网络LeNet-5以识别手写数字。

21世纪初，借助GPU和分布式计算，计算机的计算能力大大提升。

2006年，Geoffrey Hinton用贪婪逐层预训练（greedy layer-wise pretraining)有效训练了一个深度信念网络。这一技巧随后被研究人员推广到了许多不同的神经网络上，大大提高了模型在测试集上的泛化效果。以Geoffrey Hinton为代表的加拿大高等研究院附属机构的研究人员开始将人工神经网络/联结主义重新包装为了深度学习并进行推广。

2009-2012年，瑞士人工智能实验室IDSIA的Jürgen Schmidhuber带领研究小组发展了递归神经网络和深前馈神经网络。2012年，Geoffrey Hinton组的研究人员在ImageNet 2012上夺冠，他们图像分类的效果远远超过了第二名，深度学习的热潮由此开始并一直持续到现在。

与此同时，深蓝和AlphaGo的问世，更是一举震惊了世人，深度学习从此进入大众视野。

1.2 梯度下降

在线性回归算法中，我们对梯度下降算法不陌生，他就是一个通过按照规定的步长来逼近局部最优点的一种代价函数收敛优化算法。如果实值函数 F(x)在点 a处可微且有定义，那么函数F(x)在 a点沿着梯度相反的方向 $-\gamma\nabla F(x)$下降最快。其数学的公式形式其实十分的简单。

$$b = a - \gamma\nabla F(x)$$

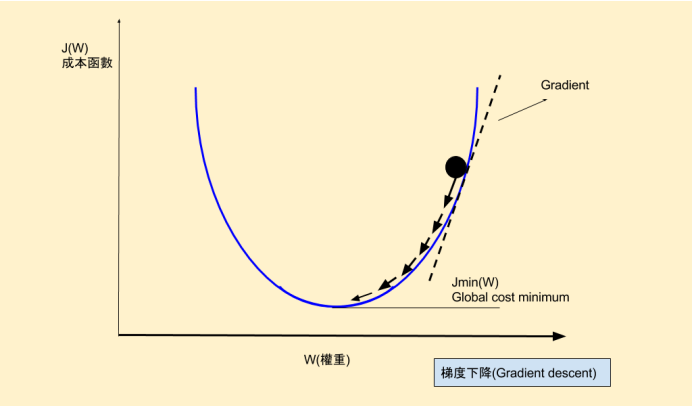

图 1: 梯度下降图解

1.3 机器学习的两种模式

监督学习 通过已有的一部分输入数据与输出数据之间的对应关系，生成一个函数，将输入映射到合适的输出，例如分类算法。数据集样式:$(x^{(1)}, y^{(1)}), (x^{(2)}, y^2)$

非监督学习 直接对输入数据集进行建模，例如聚类算法。数据集样式:$(x^{(1)}), (x^{(2)}), (x^{(3)})$

1.4 神经网络层次

人工神经网络（ANN：Artificial Neural Network），简称神经网络（NN：Neural Network）。迄今为止，人工神经网络尚无统一定义，其实一种模拟了人体神经元构成的数学模型，依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。

输入层(layer1)：输入层接收特征向量 x 。

输出层(layer2)：输出层产出最终的预测 h 。

隐含层(layer3)：隐含层介于输入层与输出层之间，之所以称之为隐含层，是因为当中产生的值并不像输入层使用的样本矩阵 X 或者输出层用到的标签矩阵 y 那样直接可见。

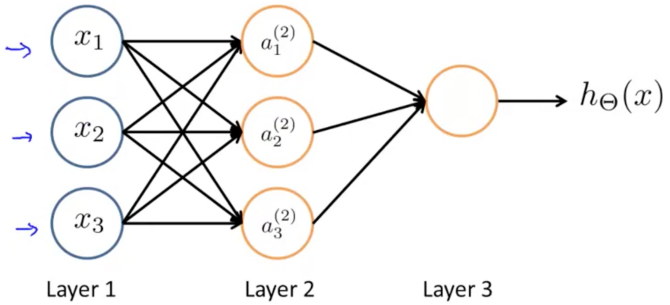

图 2: 神经网络分层模型

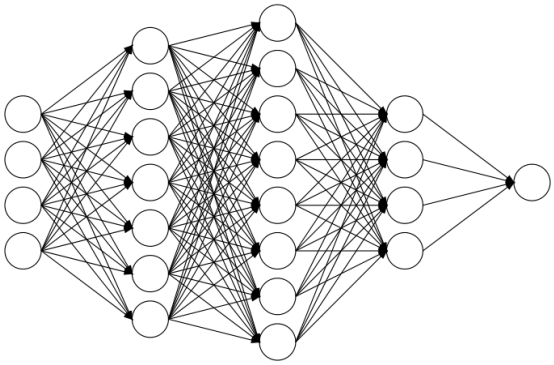

图 3: 多输入单一输出神经网络示例

1.5 感知器

感知器可以看成是最简单的神经网络，没有隐藏层。也可以看成是一个最简单的神经元，是神经网络里面最基本的单元。边上有权重（weight）,最后神经元需要经过一个激活函数f输出到下一个神经元上。

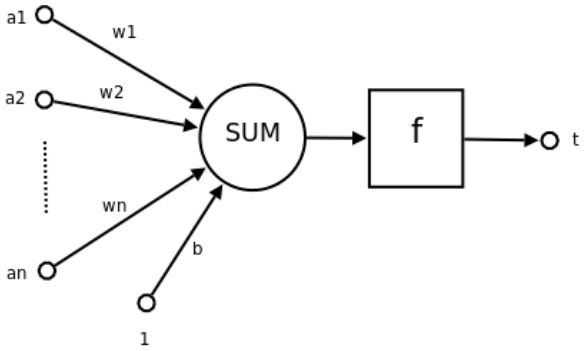

图 4: 感知器图示

1.6 激活函数

在神经元中，输入的 inputs 通过加权，求和后，还被作用了一个函数，这个函数就是激活函数（Activation Function）。激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。但是有一点需要注意的是，复杂的激活函数也许产生一些梯度消失或爆炸的问题，所以正确的选择激活函数是构建神经网络比较关键的一步。

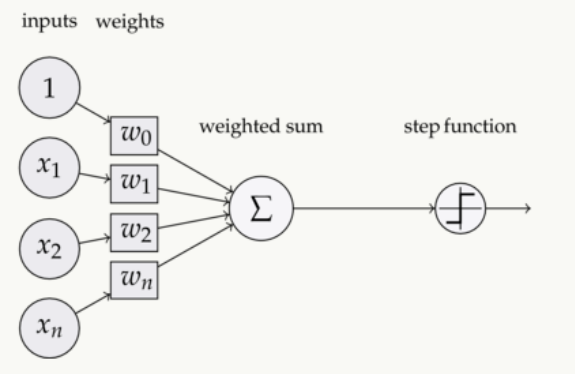

图 5: active fuction示例

sigmoid 也被称作S 形函数，其经常被使用在机器学习的逻辑回归中，取值范围为 (0,1)。Sigmoid 将一个实数映射到 (0,1) 的区间，可以用来做二分类。Sigmoid 在特征相差比较复杂或是相差不是特别大时效果比较好。

$$f(x) = \frac{1}{1 + e^{-z}}$$

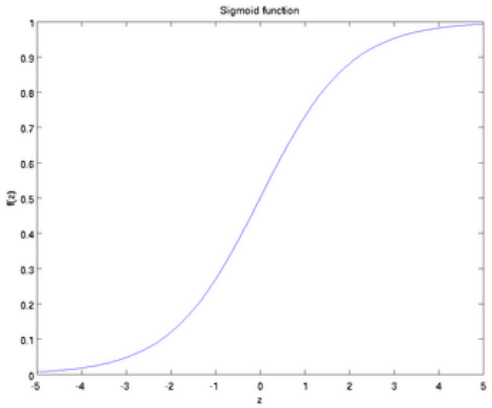

图 6: sigmoid函数

Softmax softmax 函数将 K 维的实数向量压缩（映射）成另一个 K 维的实数向量，其中向量中的每个元素取值都介于 (0，1) 之间。常用于多分类问题。注意到softmax可以实现多分

类任务，而sigmoid原则上是基于阀值来判断的0-1概率的二分类任务。当然当softmax的类别数为2的时候，在数学形式上与sigmoid一致。为方便理解，可以认为sigmoid激活函数softmax激活函数的一个特例。

$$\sigma(z)_j = \frac{e^{z_k}}{\sum_{k=1}^{k} * e^{z_k}}$$

Tanh 也称为双切正切函数，取值范围为 [-1,1]。tanh 在特征相差明显时的效果会很好，在循环过程中会不断扩大特征效果。

$$f(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

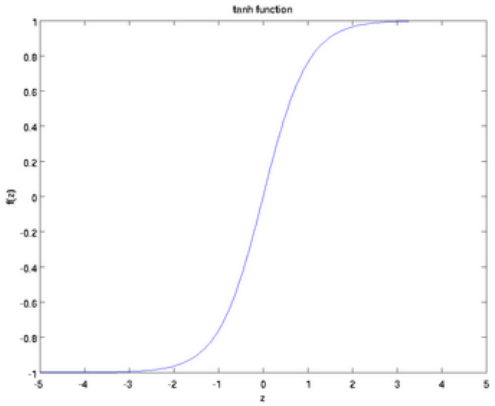

图 7: Tank函数

更多的激活函数相关知识可以参阅本章脚注网址[1]

1.7 代价函数

$$L = \text{神经网络总共包含的层数}$$
$$s_l = \text{第}l\text{层的神经元数目}$$
$$K = \text{输出层的神经元数，亦即分类的数目}$$

代价函数的一般式：

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

矩阵表示形式

[1] https://www.jiqizhixin.com/articles/2017-10-10-3

$$J(\Theta) = -\frac{1}{m}\sum(Y^T .* log(\Theta A)) + log(1 - \Theta A) .* (1 - Y^T))$$

1.8 前向传播（FP）与反向传播（BP）

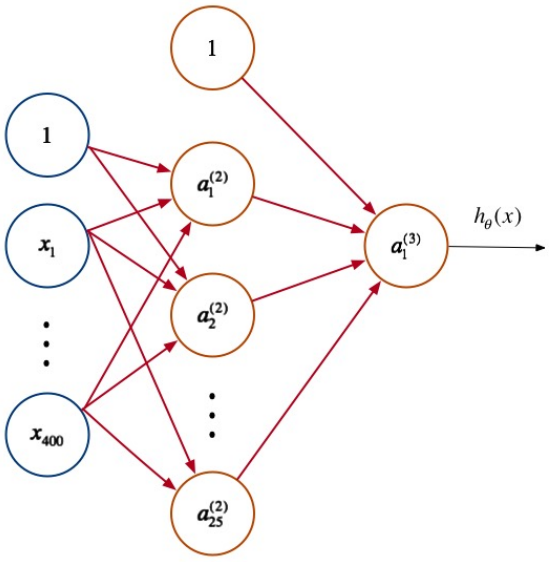

图 8: FP

1.8.1 前向传播(FP)

前向传播的过程可表示为如下：

$$\begin{aligned} a^{(1)} &= x \\ z^{(2)} &= \Theta^{(1)} a^{(1)} \\ a^{(2)} &= g(z^{(2)}) \\ z^{(3)} &= \Theta^{(2)} a^{(2)} \\ a^{(3)} &= g(z^{(3)}) \\ h_\Theta(x) &= a^{(3)} \end{aligned}$$

1.8.2 反向传播(BP)

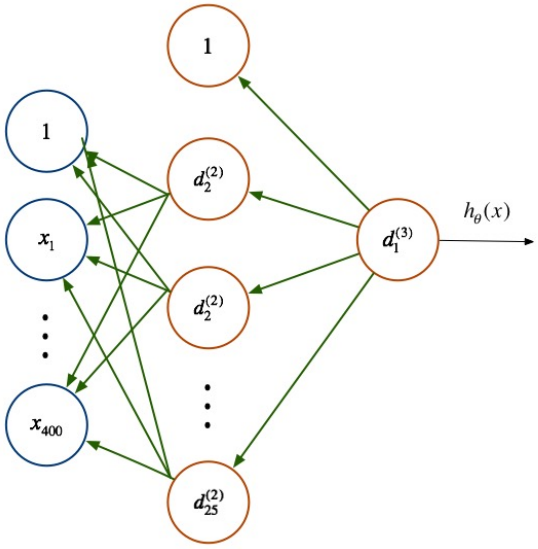

图 9: FP

反向传播的神经网络训练过程如下：

1. for all l,i,j，初始化权值梯度 $\Delta(l)$:

$$\Delta_{ij}^{(l)} = 0$$

2. for i=1 to m:

令 $a^{(1)} = x^i$

执行前向传播算法，计算各层的激活向量：$a^{(l)}$

通过标签向量 $y^{(i)}$，计算输出层的误差向量：$\delta^{(L)} = a^{(L)} - y^{(i)}$

反向依次计算其他层误差向量：$\delta^{(L-1)}, \delta^{(L-2)}, ..., \delta^{(2)}$

求 $\Delta_{ij}^{(l)} = a_j^{(l)} \delta_i^{(l+1)}$，即：$\Delta^{(l)} = \delta^{(l+1)} (a^{(l)})^T$

3.求各层权值的更新增量 D(l) ，连接偏置的权值不进行正规化：

$$D_{i,j}^{(l)} = \begin{cases} \frac{1}{m}(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)}), \text{if } j \neq 0 \\ \frac{1}{m}\Delta_{ij}^{(l)}, \text{if } j = 0 \end{cases} \tag{1}$$

4.更新各层的权值矩阵 Θ (l) ，其中 α 为学习率：

$$\Theta^{(l)} = \Theta^{(l)} + \alpha D^{(l)}$$

反向传播在一定程度上优化了梯度下降算法的原因在于，它是针对每一层的误差进行调整，相比起前向传播到末尾后再对全部的权值调整和梯度下降来说，可以提速不少。

1.9 模型的评估

1.9.1 交叉验证

在机器学习领域，我们对一个算法的拟合之初，通常需要将数集分为训练集和测试集。在两个数据集的拆分过程中，有一种比较科学的分离验证方案——交叉验证。交叉验证根据实际需要可以分为一下几个不同的实现方法：

Hold-Out Method 将原始数据随机分为两组，一组做为训练集，一组做为验证集，利用训练集训练分类器，然后利用验证集验证模型，记录最后的分类准确率为此分类器的性能指标。此种方法的好处的处理简单，只需随机把原始数据分为两组即可，其实严格意义来说Hold-Out Method并不能算是CV，因为这种方法没有达到交叉的思想，由于是随机的将原始数据分组，所以最后验证集分类准确率的高低与原始数据的分组有很大的关系，所以这种方法得到的结果其实并不具有说服性。

Double Cross Validation（2-CV） 做法是将数据集分成两个相等大小的子集，进行两回合的分类器训练。在第一回合中，一个子集作为training set，另一个便作为testing set；在第二回合中，则将training set与testing set对换后，再次训练分类器，而其中我们比较关心的是两次testing sets的辨识率。不过在实务上2-CV并不常用，主要原因是training set样本数太少，通常不足以代表母体样本的分布，导致testing阶段辨识率容易出现明显落差。此外，2-CV中分子集的变异度大，往往无法达到“实验过程必须可以被复制”的要求。

K-fold Cross Validation（K-CV） 将原始数据分成K组（一般是均分），将每个子集数据分别做一次验证集，其余的K-1组子集数据作为训练集，这样会得到K个模型，用这K个模型最终的验证集的分类准确率的平均数作为此K-CV下分类器的性能指标。K一般大于等于2，实际操作时一般从3开始取，只有在原始数据集合数据量小的时候才会尝试取2。K-CV可以有效的避免过学习以及欠学习状态的发生，最后得到的结果也比较具有说服性。

Leave-One-Out Cross Validation（LOO-CV） 如果设原始数据有N个样本，那么LOO-CV就是N-CV，即每个样本单独作为验证集，其余的N-1个样本作为训练集，所以LOO-CV会得到N个模型，用这N个模型最终的验证集的分类准确率的平均数作为此LOO-CV分类器的性能指标。

1.9.2 过拟合与欠拟合

过拟合 模型在测试集上预测准确率很高，但是在测试集合上准确率相差很远。往往在这种时候可以采用增加数据集或者正则化等方法进一步试探或者优化。

欠拟合 模型对数据的拟合效果欠佳，体现在测试集与训练集的数据预测正确率都比较底下。但是出现这种情况的一般性措施是增加模型的特征系数，或者跟换训练模型等。

1.9.3 精确率和召回率

假如把正类预测为正类称作TP，负类预测为正类称作FP，正类预测为负类称作FN，则二者的数学表达式如下：

$$P_{\text{精确率}} = \frac{TP}{TP + FP}$$
$$P_{\text{召回率}} = \frac{TP}{TP + FN}$$

2 神经网络模型

现如进，神经网络的模型随着研究的深入，从上世纪末到现在，出现了诸如前馈神经网络，霍普菲尔网络，玻尔兹曼机，自编码机等一系列的神经网络系统模型。本节主要选取了其中三个现如今在图像，语义等领域大放异彩的三个模型进行阐述。在本节的开始还归纳了一下制约神经网络领域发展的三个突出的问题。

2.1 神经网络必须直面的三个问题

随着神经网络层数的加深，有三个重大问题：一是非凸优化问题，即优化函数越来越容易陷入局部最优解；二是梯度消失问题；三是过拟合问题。

凸优化问题 线性回归，本质是一个多元一次函数的优化问题，设f(x,y)=x+y 多层神经网络，本质是一个多元K次函数优化问题，设f(x,y)=xy 在线性回归当中，从任意一个点出发搜索，最终必然是下降到全局最小值附近的。所以置0也无妨（这也是为什么我们往往解线性回归方程时初值为0）。而在多层神经网络中，从不同点出发，可能最终困在局部最小值。局部最小值是神经网络结构带来的挥之不去的阴影，随着隐层层数的增加，非凸的目标函数越来越复杂，局部最小值点成倍增长，利用有限数据训练的深层网络，性能还不如较浅层网络。。避免的方法一般是权值初始化。为了统一初始化方案，通常将输入缩放到[1,1]，但是仍然无法保证能够达到全局最优，其实这也是科学家们一直在研究而未解决的问题。所以，从本质上来看，深度结构带来的非凸优化仍然不能解决（包括现在的各类深度学习算法和其他非凸优化问题都是如此），这限制着深度结构的发展。

梯度消失 这个问题实际上是由激活函数不当引起的，多层使用Sigmoid系函数，会使得误差从输出层开始呈指数衰减。“梯度消失”现象具体来说，我们常常使用sigmoid作为神经元的输入输出函数。对于幅度为1的信号，在BP反向传播梯度时，每传递一层，梯度衰减为原来的0.25。层数一多，梯度指数衰减后低层基本上接受不到有效的训练信号。

幸运的是，这个问题已经被Hinton在2006年提出的逐层贪心预训练权值矩阵变向减轻，最近提出的ReLu则从根本上提出了解决方案。

2012年，Hinton组的Alex Krizhevsky率先将受到Gradient Vanish影响较小的CNN中大规模使用新提出的ReLu函数。

2014年，Google研究员贾扬清则利用ReLu这个神器，成功将CNN扩展到了22层巨型深度网络对于深受Gradient Vanish困扰的RNN，其变种LSTM也克服了这个问题。

过拟合 在机器学习，深度学习领域，我们往往不太希望我们的模型对数据的拟合效果过好，因为太好的拟合效果达不到泛化误差，也就导致模型的一般性不强，不能抽象出更具一般适用性的规律。过拟合的概念其实还是比较好理解，随着神经网络隐藏层的增多，随着而来的过拟合问题就成为了深度学习领域上的一道阻碍。当今研究领域的学者也在着力于研究这方面的课题。

2.2 神经网络的三种粗略分类

2.2.1 前馈网络

网络中各个神经元按接受信息的先后分为不同的组。每一组可以看作一个神经层。每一层中的神经元接受前一层神经元的输出，并输出到下一层神经元。整个网络中的信息是朝一个方向传播，没有反向的信息传播。前馈网络可以用一个有向无环路图表示。前馈网络可以看作一个函数，通过简单非线性函数的多次复合，实现输入空间到输出空间的复杂映射。这种网络结构简单，易于实现。前馈网络包括全连接前馈网络和卷积神经网络等。如前馈神经网络和深度信念网络等。

2.2.2 反馈网络

网络中神经元不但可以接收其它神经元的信号，也可以接收自己的反馈信号。和前馈网络相比，反馈网络在不同的时刻具有不同的状态，具有记忆功能，因此反馈网络可以看作一个程序，也具有更强的计算能力。反馈神经网络可用一个完备的无向图来表示。如循环神经网络。

2.2.3 记忆网络

记忆网络在前馈网络或反馈网络的基础上，引入一组记忆单元(门结构等)，用来保存中间状态。同时，根据一定的取址、读写机制，来增强网络能力。和反馈网络相比，忆网络具有更强的记忆功能。如Dynamic Memory Networks。

2.3 前馈神经网络（FNN）

给定一组神经元，我们可以以神经元为节点来构建一个网络。不同的神经网络模型有着不同网络连接的拓扑结构。一种比较直接的拓扑结构是前馈网络。前馈神经网络（Feedforward Neural Network，FNN）是最早发明的简单人工神经网络。

在前馈神经网络中，各神经元分别属于不同的层。每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。第一层叫输入层，最后一层叫输出层，其它中间层叫做隐藏层。整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图表示。

前馈神经网络也经常称为多层感知器（multi-layer perceptron，MLP）。但多层感知器的叫法并不是十分合理，因为前馈神经网络其实是由多层的logistic回归模型（连续的非线性函数）组成，而不是由多层的感知器。

前馈神经网络是最直接的可以理解的一种神经网络，贴切实际的一种模型。它的相关概念其实上一节我们有相对详细的概述，这里就不再啰嗦。很多后来的神经网络模型都是直接或间接的在前馈神经网络的概念引导下产生的。后来的反向传播算法在一定程度上缩小了前馈神经网络的计算代价，让其更加的迷人和贴近生产实际。

2.4 深度信念网络(DBN)

深度信念网络 (Deep Belief Network, DBN) 由 Geoffrey Hinton 在 2006 年提出。它是一种生成模型，通过训练其神经元间的权重，我们可以让整个神经网络按照最大概率来生成训练数据。我们不仅可以使用 DBN 识别特征、分类数据，还可以用它来生成数据[2]。还有一种比较偏重于从原理上的定义：使用层叠波尔兹曼机[3]组成深度神经网络的方法，在深度学习里被称作深度信念网络DBN。

训练过程 DBN 在训练模型的过程中主要分为两步:

第 1 步：分别单独无监督地训练每一层 RBM 网络,确保特征向量映射到不同特征空间时,都尽可能多地保留特征信息;

第 2 步：在 DBN 的最后一层设置 BP 网络,接收 RBM 的输出特征向量作为它的输入特征向量,有监督地训练实体关系分类器.而且每一层 RBM 网络只能确保自身层内的权值对该层特征向量映射达到最优,并不是对整个 DBN 的特征向量映射达到最优,所以反向传播网络还将错误信息自顶向下传播至每一层 RBM,微调整个 DBN 网络.RBM 网络训练模型的过程可以看作对一个深层 BP 网络权值参数的初始化,使DBN 克服了 BP 网络因随机初始化权值参数而容易陷入局部最优和训练时间长的缺点.

上述训练模型中第一步在深度学习的术语叫做预训练，第二步叫做微调。最上面有监督学习的那一层，根据具体的应用领域可以换成任何分类器模型，而不必是BP网络。

[2]这里的生成数据主要指的是可以在维度上对数据进行重构。胡邵华等，他们用一种自编码网络实现了对经典的"瑞士卷"数据的重构。

[3]所谓的受限玻尔兹曼机（一种随机动力系统网络，具体的可以参看本文相关参考文献）指的就是在原始的三层玻尔兹曼机的基础上，去输出层后的仅有输入和中间层的一种结构。

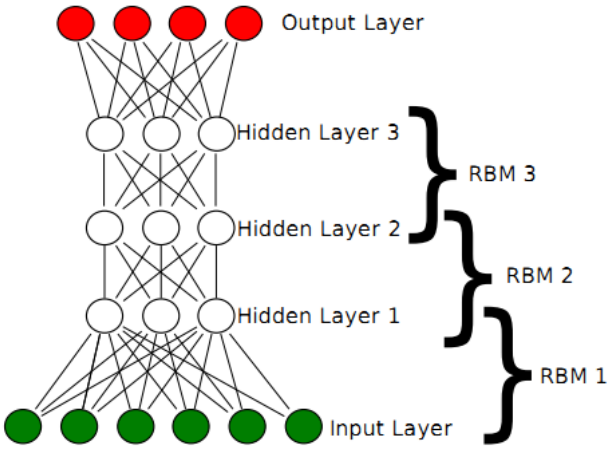

图 10: DBN图示

2.5 卷积神经网络(CNN)

1989年LeCun提出第一个真正意义上的卷积神经网络。随着时代的发展，卷积神经网络不断的被注入新鲜的血液。卷积神经网络卷积神经网络(Convolutional Neural Network, CNN)是深度学习技术中极具代表的网络结构之一，在图像处理领域取得了很大的成功，在国际标准的ImageNet数据集上，许多成功的模型都是基于CNN的。那么什么是卷积神经网络？很多的专家学者已经尝试对卷积神经网络的原理进行直观的解释。本文并不打算就卷积神经网络的具体实现细节上做过多的阐述，主要抓住一些卷积神经网络的特性来进行论述，如果读者想要了解更多的卷积网络原理细节可以参阅文末的参考文献。这里先给出一张CNN网络背后的实现原理图：

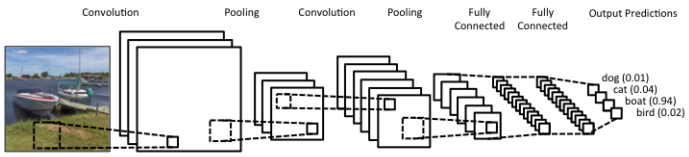

图 11: CNN图示

CNN的主要应用领域是图像领域，CNN的出现，可以很好的解决图像领域位图空间排布的特性识别，同时根据其权重共享和局部连接机制，可以很好解决图像识别过程中的参数爆炸，然后经过池化，有点类似与PCA的目的，卷积后的特征图片在保证特征最大化前提下进行维度约减，从而随着层次的深入，特征识别可以逐渐变得专一的同时，计算代价也在递减。

2.5.1 卷积神经网络的层次

INPUT（输入层） 输入层通常是一张张平面展开的图片矩阵，其实说白了就是普通的输入层而已，只不过CNN主要是用来处理图片。每个输入结点变成了多维的图片矩阵。

CONV（卷积层） 卷积层是CNN的最核心思想所在。每个卷积层可以根据需要定义N(N是根据算法的需要，自己设定的)个filter。每个filter是一个n*m的小矩阵，这些小矩阵将要去对前一层的输入进行按行步进式扫描，steplen是可以根据需要自行定义的。每次扫描的过程，其实就是一个前层图像相应的n*m大小的矩阵和filter的内积和，然后根据计算出来的结果可以得到一张新的特性图(feature map)，这张图保留了前层输入的关键特性（当然是在权重优化后），然后以这幅特性图作为后一层的输入，依次类推，然后根据标签值计算误差，运用反向传播算法来优化权重。注意这里的权重就是filter里面的每一个数值。为了方便理解，我们可以把一幅图看成X,把一个filter看成w，那么在形式上就和我们之前提到的前馈神经网络差不多了。只不过在前馈神经网络的基础上把每个w也单独展开成了向量而已。然后之前的wight*x,变成了filter与x的按行内积和。这就是卷积名称的由来。

权重共享 每个feature map使用同一个filter进行卷积，也就是说，前一个feature map在该层根据不同的filter,可以生成多个filter map，但是每个feature map的生成不是按照每移动一段距离就换一个权重(filter)的方式，而是整个图的卷积使用同一套卷积和激活函数，这就是所谓的权重共享。

POOL（池化层） 通过卷积层获得了图像的特征之后，理论上我们可以直接使用这些特征训练分类器（如softmax），但是这样做将面临巨大的计算量的挑战，而且容易产生过拟合的现象。为了进一步降低网络训练参数及模型的过拟合程度，对卷积层进行池化/采样(Pooling)处理。池化/采样的方式通常有以下两种：

1. 最大池化（Max Pooling: 选择Pooling窗口中的最大值作为采样值；

2. 均值池化（Mean Pooling）: 将Pooling窗口中的所有值相加取平均，以平均值作为采样值

FC（全连接层） 全连接层是传统的多层感知器，在输出层使用的是 softmax 激活函数（也可以使用其他像 SVM 的分类器，但在本文中只使用 softmax）。“全连接（Fully Connected）”这个词表明前面层的所有神经元都与下一层的所有神经元连接。这里需要注意到：除了全连接层是每个feature map都与后一个神经元连接，之前的所有卷积层的连接都是局部连接的形式，局部连接的产生是由于权重共享机制产生的，这个概念可以通过以下两张图表展示。

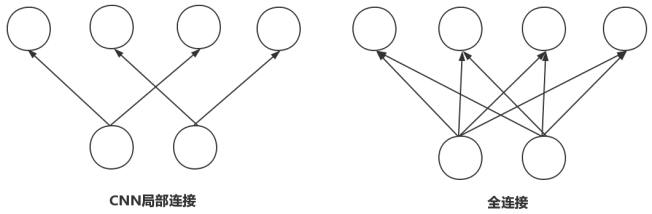

图 12: 局部连接与全连接

在全连接后，我们就可以实现对分类的目的，同样可以采用反向传播算法来优化权重。

2.6 循环神经网络(RNN)

在介绍RNN之前。我们先来思考一个问题：RNN为什么会出现？一个不变的定理：存在即合理。RNN的出现一定是为了解决实际问题的。正好比反向传播的出现是为了优化神经网络的梯度下降算法，使得更深层次的神经网络的权重优化成为可能。那么RNN是为了解决什么问题呢？RNNs的目的使用来处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNNs之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNNs能够对任何长度的序列数据进行处理。循环神经网络(Recurrent Neural Networks，RNNs)已经在众多自然语言处理(Natural Language Processing, NLP)中取得了巨大成功以及广泛应用。

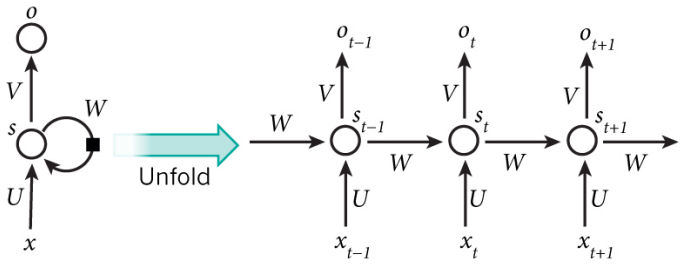

图 13: RNN图示1

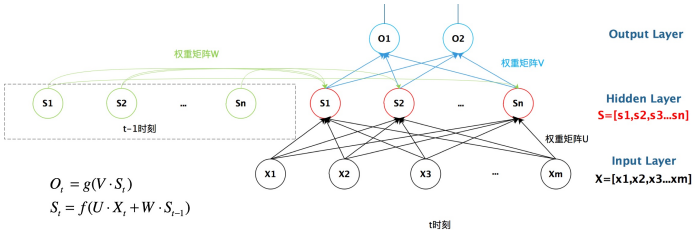

图 14: RNN图示2

单向RNN的传播缺陷 从单向的结构可以知道它的下一刻预测输出是根据前面多个时刻的输入来共同影响的，而有些时候预测可能需要由前面若干输入和后面若干输入共同决定，这样会更加准确。例如：“2020年xxx将要在东京举办，因为东京于2013年9月申奥成功。”

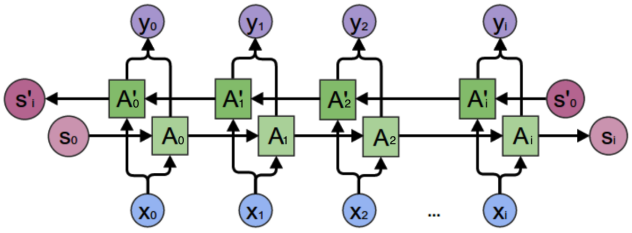

图 15: 双向RNN

2.6.1 常用领域

语言模型与文本生成 给你一个单词序列，我们需要根据前面的单词预测每一个单词的可能性。语言模型能够一个语句正确的可能性，这是机器翻译的一部分，往往可能性越大，语句越正确。另一种应用便是使用生成模型预测下一个单词的概率，从而生成新的文本根据输出概率的采样。语言模型中，典型的输入是单词序列中每个单词的词向量(如 One-hot vector)，输出时预测的单词序列。

机器翻译 机器翻译是将一种源语言语句变成意思相同的另一种源语言语句，如将英语语句变成同样意思的中文语句。与语言模型关键的区别在于，需要将源语言语句序列输入后，才进行输出，即输出第一个单词时，便需要从完整的输入序列中进行获取。

语音识别 语音识别是指给一段声波的声音信号，预测该声波对应的某种指定源语言的语句以及该语句的概率值。

图像描述生成 和卷积神经网络(convolutional Neural Networks, CNNs)一样，RNNs已经在对无标图像描述自动生成中得到应用。将CNNs与RNNs结合进行图像描述自动生成。这是一个非常神奇的研究与应用。该组合模型能够根据图像的特征生成描述。

RNNs已经被在实践中证明对NLP是非常成功的。如词向量表达、语句合法性检查、词性标注等。在RNNs中，但是RNN在处理需要长时间记忆性的问题的时候参数调整乏力。尽管从理论上来说，RNN记住很久以前的一个输入是可能的[4]，但是在实际的模型构建中几乎是不可能训练出这样对长记忆效率很优的模型。学术界为了解决这一问题，提出了长短时记忆模型(LSTM),并被业界广泛采用。同样随着时间的发展，为了解决具体的实际问题，基于LSTM的门可拓展性，长短时记忆模型也出现了很多的衍生类，同样在深度学习领域掀起了一股热潮。[5]。

2.6.2 LSTM

LSTM通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是LSTM的默认属性，而非需要付出很大的代价才能获得的能力！所有的RNN都具有一种重复神经网络模块的

[4]该问题被前人论证过Hochreiter （1991）【德国】和Bengio等人

[5]长短时记忆模型LSTM由Hochreiter和Schmidhuber在1997年提出，并在近期被Alex Graves进行了改良和推广。在很多问题上，LSTM都取得了相当巨大的成功，并得到了广泛的使用。

链式的形式。在标准的RNN中，这个重复的模块只有一个非常简单的结构，例如一个tanh层。

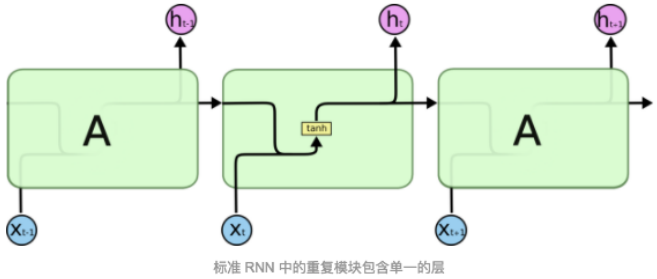

图 16: 普通的RNN单个循环体内部

但是LSTM通过加入一些记忆保留删选sigmoid门以及执行门单元，让记住长时效性历史成为了可能。

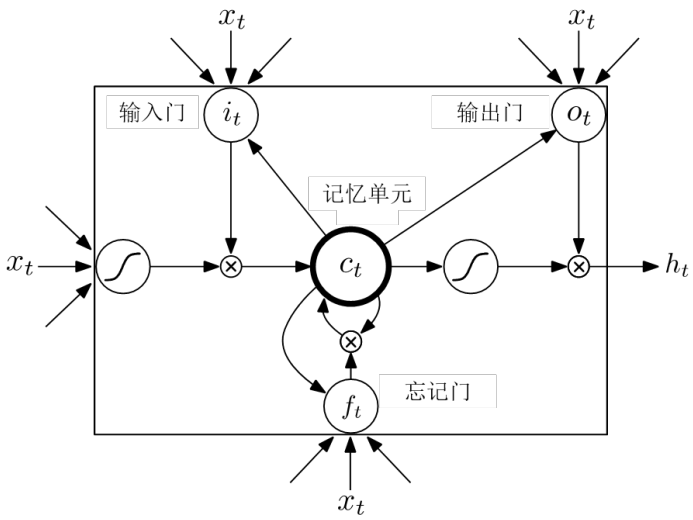

图 17: LSTM单个循环体内部

3 运用工具实现神经网络

神经网络的理论研究基本在上世纪末就已经基本成体系，然而其热潮却在cpu,gpu的摩尔定律发展中延迟到来。站在前人的肩膀上，我们可以运用的理论以及比较完善，我们可以使用的神经网络构建工具也是十分的丰富。基于python在数据科学领域的优势，本节将利用python以及其丰富的库来实现一些简单的神经网络的应用，相关代码托管在github[6]上。

[6] https://github.com/IversionBY/sklearn_machine_learning

3.1 实验环境概览

人生苦短，我用python!python因为其丰富的库，让其在各个领域都有用武之地，特别是在数据科学，其更是占据了半片江山。本节将使用tensorflow,keras，numpy，matploitlib等库函数，运用python3.6环境，构建简单的神经网络应用。

3.1.1 tensorflow

tensorflow是深度学习领域最流行的一个开源框架，因其基于张量，图的流式计算，十分符合神经网络的思想。同时因为其对python友好，继承了python简约大气的思想，用户使用的接口上也是相当的友好。特别是其使用基于session的机制，将整个计算过程完全放在python环境外使用C++来执行，最后通过session接口机制来实现交互。这个设计理念很好的解决了python计算慢的不足。同时，它天生支持cpu,gpu以及并行计算，可以说对深度学习相当的友好。

3.1.2 keras

keras是一个深度学习应用顶层框架。引用其官网的话：Keras 是一个用 Python 编写的高级神经网络 API，它能够以 TensorFlow, CNTK, 或者 Theano 作为后端运行。没错，他是对CNTK,Tensorflow,Theano这些深度学习大咖的更顶层封装，并且使用的是sklearn的一些API机制，让熟悉sklearn[7]的编程人员可以比较平缓的过渡到深度学习的应用开发上。

Keras 被认为是构建神经网络的未来，以下是一些它流行的原因：

1.轻量级和快速开发：Keras 的目的是在消除样板代码。几行 Keras 代码就能比原生的 TensorFlow 代码实现更多的功能。你也可以很轻松的实现 CNN 和 RNN，并且让它们运行在 CPU 或者 GPU 上面。

2.框架的“赢者”：Keras 是一个API，运行在别的深度学习框架上面。这个框架可以是 TensorFlow 或者 Theano。Microsoft 也计划让 CNTK 作为 Keras 的一个后端。目前，神经网络框架世界是非常分散的，并且发展非常快。

3.2 实践项目描述

在这个实践项目中，我们将会运用神经网络的相关知识，结合现在流行的神经网络架构keras,实现一个网络安全项目——webshell的检测算法。

3.2.1 什么是webshell

webshell是一种在web攻击中根据web应用的漏洞，通过传入的webshell，实现与服务器后端交互，并且任意命令执行的一种技术。下面列举一些常用的php一句话shell:

```
1 php:
2
3 <?php system($_GET["CMD"]);?>
4 <?php @eval($_POST[sb]);?>
```

[7]scikit-learn：一个机器学习领域十分全面的库，但是目前在深度学习的支持上还存在不足

```
<?php assert($_POST[sb]);?>
......
```

3.2.2 数据集

在这个实验中所使用的数据集是根据国外的信息安全教育机构ADFA收集整理的一些数据集ADFA-LD[8]。里面囊括了很多类型的攻击向量，其中就包括webshell，同样里面也包括一些常规的数据。这个数据样本的好处是：它已经帮我们实现了对文本信息的ascll码特征转换，这样我们就省区了数据处理的一部分。可以专注于模型的构建。在本实验中我们最后用到的数据集的大小一共有700个左右，每个数据项表示的是一个ascii码特征化的一维矢量，数据样本可以分为两类：正常文件，webshell文件。所以我们最后要构建的模型是一个可以识别一些文本语义的有一定时间记忆的神经网络模型。

3.2.3 抽象的神经网络模型构建

首先分析数据样本，我们发现数据是一个二元分类的模型，然后是一个需要基于一定的语义，也就是说数据是具有先后输入效应的神经网络模型，这样我们就基本可以确定这里可以使用的是RNN类型的神经网络的二元分类。但是由于样本数大概只有700左右，那么过多层次的模型有可能是会造成过拟合的，所以初步决定层数不超过4层，然后就是每一层里面的神经元数量也不宜过大，所以最后我们决定构造的模型如下：

```
model = Sequential()
model.add(Embedding(input_dim=max_sys_call+1,output_dim=128, input_length=max_s
model.add(Dense(2,activation="softmax"))
model.summary()
```

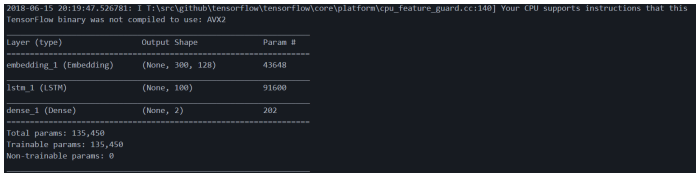

图 18: keras模型输出展示

3.2.4 参数调整与模型优化

在这一步里面，我们可以针对训练好的模型进行一些参数的细调整，如神经元个数，层数，以及代价函数的选取等等，但是本实践的样本数量相对较少，参数优化上效果不是很明显，准确率可以达到90%左右，在这样少的样本容量下，可以达到这样一个训练效果还是相对不错的。

[8]可以从这个地址下载数据https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-IDS-Datasets/

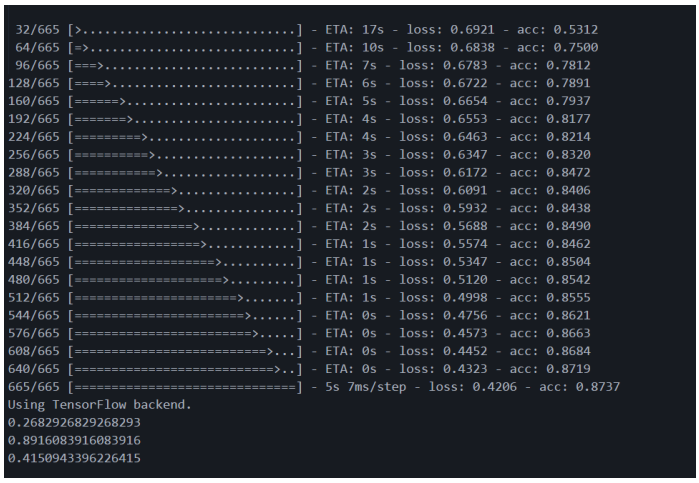

图 19: 训练的过程和最后的召回率，准确率和F1 score

3.2.5 整体的实现过程

```
import re
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn import metrics
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Input, LSTM, Dense,Embedding,Activation
from keras.models import Sequential
from keras.models import Model
from keras.utils import plot_model

max_sequences_len=300
max_sys_call=0

def load_one_flle(filename):
    global max_sys_call
    x=[]
    with open(filename) as f:
        line=f.readline()
        line=line.strip('\n')
        line=line.split(' ')
        for v in line:
            if len(v) > 0:
                x.append(int(v))
                if int(v) > max_sys_call:
                    max_sys_call=int(v)
    return x


def load_adfa_training_files(rootdir):
    x=[]
    y=[]
    list = os.listdir(rootdir)
    for i in range(0, len(list)):
        path = os.path.join(rootdir, list[i])
```

```
        if os.path.isfile(path):
            x.append(load_one_flle(path))
            y.append(0.)
    return x,y


def dirlist(path, allfile):
    filelist = os.listdir(path)
    for filename in filelist:
        filepath = os.path.join(path, filename)
        if os.path.isdir(filepath):
            dirlist(filepath, allfile)
        else:
            allfile.append(filepath)
    return allfile


def load_adfa_webshell_files(rootdir):
    x=[]
    y=[]
    allfile=dirlist(rootdir,[])
    for file in allfile:
        if re.match(r"./data/ADFA-LD/Attack_Data_Master/Web_Shell*",file):
            x.append(load_one_flle(file))
            y.append(1.)
    return x,y


def do_rnn(trainX, testX, trainY, testY):
    global max_sequences_len
    global max_sys_call

    trainX = pad_sequences(trainX, maxlen=max_sequences_len, value=0.)
    testX = pad_sequences(testX, maxlen=max_sequences_len, value=0.)
    trainY = to_categorical(trainY)
    testY_old=testY
    testY = to_categorical(testY)

    model = Sequential()
```

```
    model.add(
        Embedding(
            input_dim=max_sys_call+1,
            output_dim=128,
            input_length=max_sequences_len))
    model.add(LSTM(100,dropout=0.2,activation='tanh'))
    model.add(Dense(2,activation="softmax"))
    model.summary()
    model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])
    model.fit(trainX,trainY)
    plot_model(model, to_file='model.png')
    model.save(r"./model")
    y_predict_list=model.predict(testX)
    y_predict = []
    for i in y_predict_list:
        if i[0] > 0.5:
            y_predict.append(0)
        else:
            y_predict.append(1)
    print(metrics.recall_score(testY_old, y_predict))
    print(metrics.accuracy_score(testY_old, y_predict))
    print(metrics.f1_score(testY_old, y_predict))


if __name__ == '__main__':
    x1,y1=load_adfa_training_files(r"./data/ADFA-LD/Training_Data_Master/")
    x2,y2=load_adfa_webshell_files(r"./data/ADFA-LD/Attack_Data_Master/")
    x=x1+x2
    y=y1+y2
    x_train, x_test, y_train, y_test = train_test_split(
        x,
        y,
        test_size=0.3,
        random_state=0)
    do_rnn(x_train, x_test, y_train, y_test)
```

4 总结

本篇文章可以看到神经网络的大概知识框架以及神经网络的发展历程。学者们在深度学习领域的研究还在不断的深入着，与此同时，神经网络也在随着研究与实践的不断深入，造福着我们人类。越来越智能化的手机，甚至是家居设备都可以和你唠嗑一两句；自动驾驶的实践也即将迎来批量生产阶段，深度学习在生特征识别，如人类识别等领域的发展已经相当成熟.....然而，人工智能的脚步还在继续，我们的生活又究竟会发生怎么样的变化，我们拭目以待，翘首以盼.....

参考文献

[1] Rosenblatt, Frank. “The perceptron: a probabilistic model for information storage and organization in the brain.” Psychological review 65.6 (1958): 386.

[2] Hinton, Geoffrey E., and Terrence J. Sejnowski. “Learning and releaming in Boltzmann machines.” Parallel distributed processing: Explorations in the microstructure of cognition 1 (1986): 282-317.

[3] Smolensky, Paul. Information processing in dynamical systems: Foundations of harmony theory. No. CU-CS-321-86. COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.

[4] Bengio, Yoshua, et al. “Greedy layer-wise training of deep networks.” Advances in neural information processing systems 19 (2007): 153.

[5] LeCun, Yann, et al. “Gradient-based learning applied to document recognition.” Proceedings of the IEEE 86.11 (1998): 2278-2324.

[6] Elman, Jeffrey L. “Finding structure in time.” Cognitive science 14.2 (1990): 179-211.

[7] Hochreiter, Sepp, and Jürgen Schmidhuber. “Long short-term memory.” Neural computation 9.8 (1997): 1735-1780.

[8] 晓军.斯坦福机器学习笔记[EB/OL].https://yoyoyohamapi.gitbooks.io/mit-ml/content/, 2017.

[9] Fjodor Van Veen,The Neural Network Zoo[EB/OL].http://www.asimovinstitute.org/neural-network-zoo/, 2016.

[10] 邱锡鹏,神经网络与深度学习[EB/OL].https://nndl.github.io/, 2016.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville,Deep Learning[EB/OL].http://www.deeplearningbook.org, 2016.