

目录

1 前言	1
2 花苗分类问题	2
3 数据预处理	4
3.1 掩模构建与形态学去噪	4
3.2 形态学去噪	5
3.3 边框裁剪与尺寸统一化	5
4 BP 神经网络	6
4.1 神经网络的构造与前向传播	6
4.2 神经网络的反向传播	8
4.3 激活函数	10
4.4 传统 BP 网络的应用	12
4.5 梯度下降方法	16
4.6 正则化与 dropout	17
4.7 BP 神经网络 +	18
5 卷积神经网络	18
5.1 卷积神经网络概述	18
5.2 AlexNet	18
5.3 VGGNet	18
5.4 Inception	18
5.5 ResNet	18
5.6 迁移学习	18
5.7 CNN+	18
5.8 Batch Normalization	18
6 参考文献	18

1 前言

大量数据代表了价值。数据背后通常隐含着客观规律，如果数据量足够大的话，其规律是可以被认知和学习的，其催生了机器学习的研究方向，研究如何用数据进行建模与变现。然而，由于数据量极大，而且所涉及的算法会很复杂，通常不可能进行人为的计算，即使是用计算机进行计算，也对计算机的处理速度，内存，储存空间提出了一定的要求。另一方面，如若要进行机器学习，除了计算机硬件的要求之外，还需要软件与算法的支持，其中，算法是机器学习的核心。历史发展来看，计算机硬件，用于机器学习的软件与算法的发展是相辅相成的。

在 20 世纪 40 年代，人们开始研究人工智能，由于生物学的发展，人们模仿人类的神经元运作而提出了神经网络的原型：M-P 神经元模型，并提出了激活函数的概念。在 20 世纪 50 年代到 60 年代，

感知器算法、梯度下降法、最小二乘法等求解算法面世，而且提出了感知器，并开始应用在文字、语音、信号等领域。在 20 世纪 60 年代到 70 年代，神经网络算法因感知器的缺陷而衰落。在 70 年代到 80 年代，神经网络的种类变得丰富起来，涌现出 BP 神经网络，RBF 神经网络等各种网络，并提出了深度学习的概念与卷积神经网络（CNN）和循环神经网络（RNN）的结构。90 年代后，一些有别于神经网络的算法面世，如 SVM，决策树，boosting 与随机森林等方法，从不同的角度对机器学习算法进行丰富。在 2006 年，Hinton 提出了解决深度学习中梯度消失问题的解决方法之后，深度学习开始爆发。2012 年，ReLU 激活函数的提出，进一步抑制了梯度消失的问题，并且深度学习在语音和图像方面开始有惊人的表现。2012 年，在 ImageNet 图像识别比赛上，AlexNet 通过构建一定深度的 CNN 夺得冠军，其性能彻底击败了 SVM。需注意的是，AlexNet 首次使用了 ReLU 激活函数，Dropout 防止过拟合方法，以及 GPU 加速。之后，在 AlexNet 的结构上做优化，又提出了其他更强大的模型，如 VGGNet，Inception 系列，ResNet 等。强化学习和迁移学习的提出，进一步增强了模型的性能。

本论文基于 kaggle（全球数据科学平台）的花苗分类竞赛（Plant Seedlings Classification¹）中的数据集，探究传统机器学习算法（SVM，决策树，随机森林与 boosting 等）、深度学习算法（AlexNet，VGGNet，InceptionV3）的原理与性能，并对其尝试做优化与结合（如 AlexNet+SVM 等）。

2 花苗分类问题

该问题来自于 kaggle 的 Plant Seedlings Classification 竞赛。给定的 13 类花苗（有枝干，树叶，无花）的四千余张彩色图片用于训练、构建模型。其数据基本情况如下

序号	种类	样本数量
1	Black-grass	263
2	Charlock	390
3	Cleavers	287
4	Common Chickweed	611
5	Common wheat	221
6	Fat Hen	475
7	Loose Silky-bent	654
8	Maize	221
9	Scentless Mayweed	516
10	Shepherds Purse	231
11	Small-flowered Cranesbill	496
12	Sugar beet	385
总和		4750

而且每一张的图片大小都有可能不同。每一类的图像的例子如下

¹ <https://www.kaggle.com/c/plant-seedlings-classification>



Black-grass,Charlock,Cleavers



Common Chickweed,Common wheat,Fat Hen



Loose Silky-bent,Maize,Scentless Mayweed



Shepherds Purse,Small-flowered Cranesbill,Sugar beet

由于所涉及到的数据为彩色图像，而花苗的特征为绿色，故考虑使用 opencv 的方法，通过建立掩膜筛选出花苗的图像，进而将彩色图像转化为灰度图像或二值图像，从而达到降维的目的。而即使降维之后，为了确保图像失真不大，所以至少将图像转化为 64×64 的灰度图像矩阵。若考虑直接采用 Logistic、SVM 或决策树方法，则需要将 64×64 的灰度图像矩阵拉伸为 4096×1 的图像，然而假设用全部的数据进行训练，也只有 4750 个样本，训练时极容易导致过拟合，但是，考虑集成学习的方法或是带 dropout 的 BP 神经网络可以一定程度上防止过拟合。考虑到为涉及图像的分类问题，可以采用卷积神经网络（CNN）。由于是 13 类的分类问题，且样本数较少，可以进一步考虑在用 SVM、Logistic 或决策树方法来替代神经网络最后的 softmax 层，或许能起到更好的效果。

3 数据预处理

3.1 掩模构建与形态学去噪

对于花苗图像，我们可以看到，花苗的背景通常为黄土、砂砾或塑料箱等，而绿色的花苗则显得非常好辨认。而且我们面对的花苗是绿色的，因而考虑设置一个 hsv 范围，将绿色的部分从图像中剥离出来。于是我们首先将花苗图像进行颜色空间的转换，从 rgb 颜色空间转化为 hsv 颜色空间，之后设定 hsv 颜色空间为 [26,43,46] 和 [99,255,255]，在原图中过滤出在这个 hsv 颜色空间的图像得到掩膜，若在这个区间中，则为白色，否则为黑色。之后可以通过该掩膜对原图进行位运算，则可得到原图的图像。其部分代码如下

```

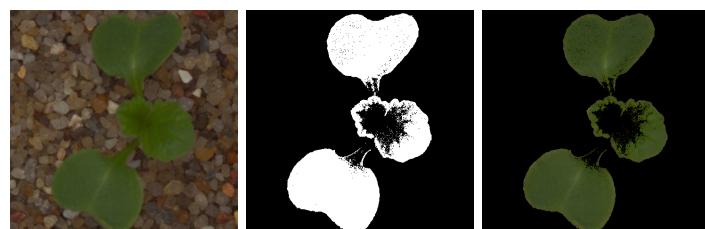
1 import cv2
2 # 假设待处理的图像为img
3 # rgb图像转化为hsv图像
4 hsv = hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
5 # 绿色的阈值(HSV)
6 lower_green = np.array([26,43,46])
7 upper_green = np.array([99,255,255])
8 # 根据阈值构建掩膜
9 mask = cv2.inRange(hsv,lower_green,upper_green)
10 # 对原图像和掩膜进行位运算
11 res = cv2.bitwise_and(img,img,mask=mask)

```

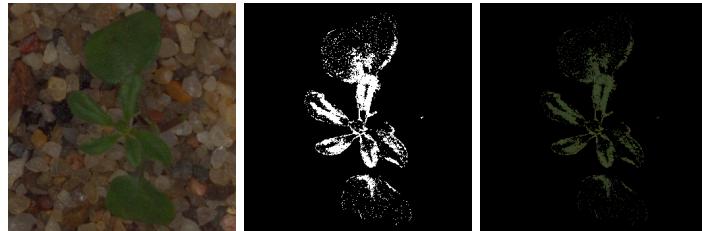
得到的结果如下，以 Black-grass, Charlock, Cleavers 种类的各一个图像为例



Black-grass 的一个图像，其大小为 370×368 ，从左到右依次为原图，掩膜图像，通过掩膜对原图过滤的图像



Charlock 的一个图像，其大小为 484×484 ，从左到右依次为原图，掩膜图像，通过掩膜对原图过滤的图像



Cleavers 的一个图像，其大小为 346×346 ，从左到右依次为原图，掩膜图像，通过掩膜对原图过滤的图像

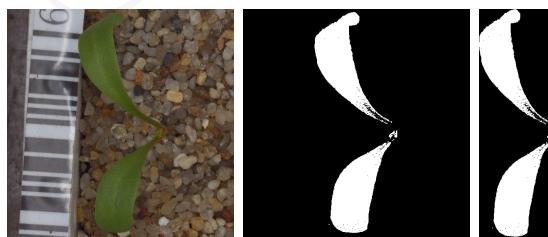
3.2 形态学去噪

对于用掩膜处理后的花苗二值图像，考虑到在花苗所在盆栽可能会有一些小草，通过掩膜处理后会有噪声。因而考虑用形态学方法去噪。

对于一个二值图像，比较常用的去噪方法是形态学去噪，而这通常涉及两种形态学转换，分别为腐蚀和膨胀，其涉及的原理较简单。对于腐蚀，先定义一个窗口，窗口将沿着图像滑动，以遍历整个图像。滑动过程中，窗口内所有像素不全为 1 时，则令窗口中的所有像素等于 0；若窗口内所有像素全为 1 时，则不做操作。选用一个合适尺寸的窗口，对于腐蚀之后的图片，其白噪声点可以消除，但也会对物体的边缘进行腐蚀。膨胀则与腐蚀相反，区别在于滑动过程中，窗口内元素只要有 1，则整个窗口元素都令为 1，这样会增大物体的尺寸。通常对于有白噪声的图片，先腐蚀再膨胀可以消除白噪声，但一定程度会导致物体失真。但由于用掩膜处理后的图像，其物体十分明显，用形态学方法去噪后失真的可能性不大。因而考虑用形态学方法去噪。

3.3 边框裁剪与尺寸统一化

我们从掩膜图像可以看出，目标图像（白色部分）只是占所有图像的很小一部分，而其余其余部分为黑色，而这其余的部分往往是无效的。现在考虑用一个矩形边框去提取出图像的有效区域，而将无效区域剔除。方法是访问图像中有效区域的宽度最小值和最大值，以及高度最小值和最大值，从而确定矩形边框区域。其效果如下图所示



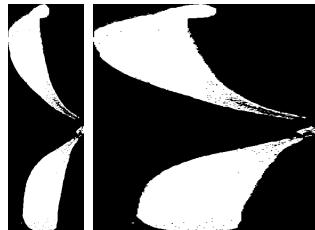
Sugar beet 的一个图像，其大小为 546×546 ，从左到右依次为原图，掩膜图像，对掩膜图像矩形裁剪之后的图像，其大小为 199×530

实现这个效果的代码如下

```
1 import cv2
2 # 获得矩形边框的最小宽高以及宽度和高度
3 x,y,w,h = cv2.boundingRect(mask_temp)
```

```
4 # 在原掩膜图像中选取矩形边框
5 mask_tg = mask[y:(y+h),x:(x+w)]
```

由于卷积神经网络需要同样大小尺寸的输入，所以考虑将图像统一尺寸归一化为统一的大小。内插的方法是 INTER-CUBIC，其结果如下图



Sugar beet 的一个图像，左边为矩形裁剪之后的图像，大小为 199×530 ，右边为尺寸统一化后的图像，大小为 96×96 实现这个效果的代码如下

```
1 import cv2
2 # 尺寸变换
3 mask_tg_rs = cv2.resize(mask_tg,[96,96],interpolation=cv2.INTER_CUBIC)
```

最后，对所有的图片都做上述操作。每张图片的掩膜的尺寸归一后的图像作为输入，需要注意的是，图片格式为 uint8，需要转化为 float 才不会出问题，用一下代码可以解决该问题。

```
1 from skimage import img_as_float
2 mask_tg_rs = img_as_float(mask_tg_rs)
```

输出则采用将名字用 one-hot 编码作为标签，采用自助法分割训练集和验证集。

4 BP 神经网络

4.1 神经网络的构造与前向传播

神经网络是由单个或多个神经元组成。下面是单个神经元的构造。

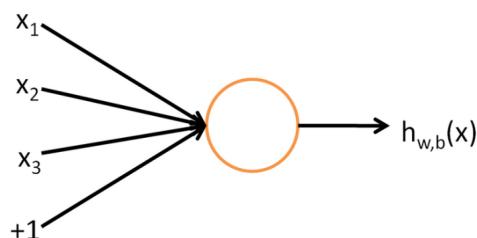


图 1：神经元

该神经元的输入由三个数据 x_1, x_2, x_3 以及偏置项 (bias)+1 组成，通过神经元后输出的表达式为

$$h_{W,b}(x) = f(W^T x + b) = f\left(\sum_{i=1}^3 W_i x_i + b\right) \quad (1)$$

其中 f 为激活函数。激活函数是为了将线性项 $W^T x$ 变换为非线性。在 BP 中，较常用的激活函数为 sigmoid 函数，其表达式如下

$$f(z) = \frac{1}{1 + \exp(-z)} \quad (2)$$

另外，令 $b = w_0$ ，则可重新定义 $W = (w_0, w_1, w_2, w_3)^T$, $x = (1, x_1, x_2, x_3)$ ，于是可将上式写为

$$h_{W,b}(x) = f(W^T x) \quad (3)$$

下面讨论神经网络。多个神经元可以组成一个层，多个层互相连接可以组成神经网络。其中，接受数据输入的层为输入层，数据计算后的数据的输出层，中间的层则称为隐含层。下图是含有两个隐含层的神经网络。

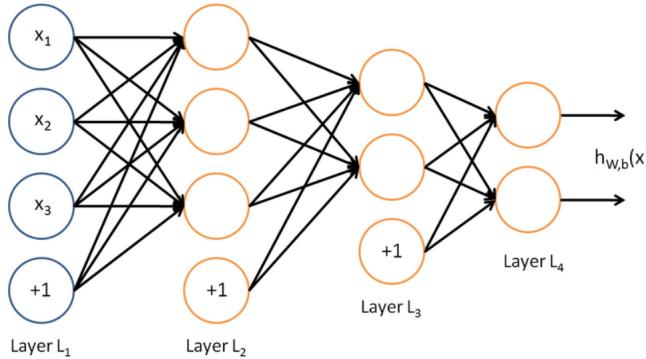


图 2：含有两个隐含层的神经网络

如图，最左边的为输入层，即图中的 Layer L1，最右边的为输出层，即图中的 Layer L4，中间的所有层，即图中的 Layer L2, Layer L3 为隐含层。

我们用 n_l 来表示网络的层数，记第 i 层为 L_i ，于是输入层为 L_1 ，输出层为 L_{n_l} 。由于神经网络可以有任意多的隐层以及隐藏神经元，则我们记 $W_{ij}^{(l)}$ 为第 l 层第 j 单元以及第 $l+1$ 层第 i 单元之间的连接权重， $b_i^{(l)}$ 为第 $l+1$ 层第 i 单元的偏置。我们用 $a_i^{(l)}$ 表示第 l 层第 i 单元的激活值（输出值），则有

$$a_i^{(l+1)} = f\left(\sum_{j=1}^{S_l} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)}\right) \quad (4)$$

其中当 $l = 1$ 时， $a^{(l)} = x$ ， x 为输入向量 $(x_1, x_2, \dots, x_{S_l})$ ， S_l 指第 l 层的神经元个数，我们用 $z_i^{(l+1)}$ 表示第 $l+1$ 层第 i 单元输入加权和（包括偏置），即

$$z_i^{(l+1)} = \sum_{j=1}^{S_l} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)} \quad (5)$$

则有

$$a_i^{(l+1)} = f(z_i^{(l+1)}) \quad (6)$$

$$h_{W,b}(x) = a^{(n_l)} = f(z^{(n_l)}) \quad (7)$$

上述过程称为神经网络的前向传播。

4.2 神经网络的反向传播

根据上面的前向传播，我们设神经网络的各层表示为 L_1, L_2, \dots, L_{n_l} ，其中， L_{n_l} 为输出层，对于输出层，假设输出层输出为 $t = a^{(n_l)}$ ， y 为标签，则若为回归问题，则代价函数使用 MSE，即

$$J(W, b; x, y) = \frac{1}{2} \|t - y\|^2 \quad (8)$$

接下来计算输出层的残差

$$\begin{aligned} \delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{(n_l)}} J(W, b; x, y) \\ &= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 \\ &= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \sum_{j=1}^S (y_j - a_j^{(n_l)})^2 \\ &= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \sum_{j=1}^S (y_j - f(z_i^{(n_l)}))^2 \\ &= -(y_i - f(z_i^{(n_l)})) \cdot f'(z_i^{(n_l)}) \\ &= -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)}) \end{aligned} \quad (9)$$

下面考虑残差的递推算法，以输出层前一层为例。由前向传播我们可以推导出

$$z_i^{(l+1)} = \sum_{j=1}^{S_l} W_{ij}^{(l)} f(z_j^{(l)}) + b_i^{(l)} \quad (10)$$

则有

$$z_i^{(n_l)} = \sum_{j=1}^{S_l} W_{ij}^{(n_l-1)} f(z_j^{(n_l-1)}) + b_i^{(n_l-1)} \quad (11)$$

于是有

$$\frac{\partial z_i^{(n_l)}}{\partial z_i^{(n_l-1)}} = \sum_{j=1}^{S_l} W_{ij}^{(n_l-1)} f'(z_j^{(n_l-1)}) \quad (12)$$

则可以得到输出层前一层的残差

$$\begin{aligned} \delta_i^{(n_l-1)} &= \frac{\partial}{\partial z_i^{(n_l-1)}} J(W, b; x, y) \\ &= \frac{\partial J(W, b; x, y)}{\partial z_i^{(n_l)}} \cdot \frac{\partial z_i^{(n_l)}}{\partial z_i^{(n_l-1)}} \\ &= \sum_{j=1}^{S_l} \delta_j^{(n_l)} W_{ij}^{(n_l-1)} f'(z_j^{(n_l-1)}) \end{aligned} \quad (13)$$

将 $n_l - 1$ 与 n_l 的关系替换为 l 与 $l + 1$ 的关系，则可得到

$$\delta_i^{(l)} = \frac{\partial}{\partial z_i^{(l)}} J(W, b; x, y) = \left(\sum_{j=1}^{S_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \quad (14)$$

若取函数 f 为 sigmoid 函数, 则有

$$f'(z_i^{(l)}) = f(z_i^{(l)}) \circ (1 - f(z_i^{(l)})) = a_i^{(l)} \circ (1 - a_i^{(l)}) \quad (15)$$

其中 \circ 代表点乘。于是可得到 $\sigma_j^{(l+1)}$ 到 $\sigma_j^{(l)}$ 的递推式:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{S_{l+1}} W_{ij}^{(l)} \delta_j^{(l+1)} \right) (a_i^{(l)} \circ (1 - a_i^{(l)})) \quad (16)$$

反向传播, 一般采用梯度下降法对每一层的权重进行调整, 即

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) \quad (17)$$

其中, α 是学习率。因而需要求权重 $W_{ij}^{(l)}$ 对于代价函数的偏导, 此时可使用当前层的残差来进行计算, 即

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = \frac{\partial J(W, b; x, y)}{\partial z_i^{(l+1)}} \frac{z_i^{(l+1)}}{W_{ij}^{(l)}} \quad (18)$$

又有

$$\frac{z_i^{(l+1)}}{W_{ij}^{(l)}} = \frac{\left(\sum_{j=1}^{S_l} W_{ij}^{(l)} f(z_j^{(l)}) \right)}{W_{ij}^{(l)}} = f(z_i^{(l)}) = a_i^{(l)} \quad (19)$$

于是可得

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)} \quad (20)$$

综上, 可以总结 BP 神经网络算法

BP 神经网络算法

- 1 输入: 训练输入 x , 训练输出 y , 学习率 α
- 2 **while** 未达到收敛条件
- 3 输入训练输入, 训练输出, 学习率
 1. 初始化神经网络的权重与偏置
 2. 对输入进行前向传播, 得到除输入层外每一层 (L_2, \dots, L_{n_l}) 的激活值 $a^{(2)}, \dots, a^{(n_l)}$
 3. 计算各层残差:
 - (1) 对输出层 (第 n_l 层)

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \cdot (a^{(l)} \circ (1 - a^{(l)})) \quad (21)$$
 - (2) 对于 $l = n_l - 1, \dots, 2$ 各层, 可递推出残差值

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \cdot (a^{(l)}) \quad (22)$$
 - (3) 计算损失函数对每一层权重的偏导数值

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T \quad (23)$$

(4) 更新参数

$$W^{(l)} = W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x, y) \quad (24)$$

4 end

若为多分类问题，先对 y 进行 one-hot 处理得到 p 维向量 (y_1, y_2, \dots, y_p) （假设 y 有 p 种取值），并将输出层的激活函数选为 softmax，即

$$a_i^{(n_l)} = f_s(z_i^{(n_l)}) = \frac{e^{z_i^{(n_l)}}}{\sum_j e^{z_j^{(n_l)}}} \quad (25)$$

并且代价函数使用交叉熵损失函数

$$J(W, b; x, y) = - \sum_i y_i \log a_i^{(n_l)} \quad (26)$$

则输出层残差为

$$\begin{aligned} \delta_i^{(n_l)} &= \frac{\partial J}{\partial z_i^{(n_l)}} \\ &= \sum_i \frac{\partial J}{a_i^{(n_l)}} \cdot \frac{\partial a_i^{(n_l)}}{\partial z_i^{(n_l)}} \\ &= \sum_i \frac{\partial -\sum_i y_i \log a_i^{(n_l)}}{a_i^{(n_l)}} \cdot \frac{\partial a_i^{(n_l)}}{\partial z_i^{(n_l)}} \\ &= -\sum_i \frac{y_i}{a_i^{(n_l)}} \frac{\partial a_i^{(n_l)}}{\partial z_j^{(n_l)}} \end{aligned} \quad (27)$$

当 $i = j$ 时，记 $e^{z_j^{(n_l)}} = e^A$ ， $\sum_{k \neq j} e^{z_k^{(n_l)}} = e^B$ ，显然有 $e^A + e^B = \sum_i e^{z_i^{(n_l)}}$ ，于是

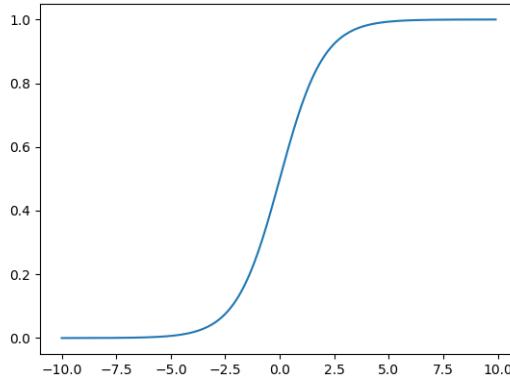
$$\begin{aligned} \frac{\partial a_i^{(n_l)}}{\partial z_j^{(n_l)}} &= \frac{\partial a_j^{(n_l)}}{\partial z_j^{(n_l)}} \\ &= \frac{\partial \frac{e^A}{e^A + e^B}}{\partial A} \\ &= \frac{e^A(e^B + e^A) - e^{2A}}{(e^A + e^B)^2} \\ &= \frac{e^A e^B}{(e^A + e^B)^2} \\ &= \frac{e^A}{e^A + e^B} \frac{e^B}{e^A + e^B} \\ &= \frac{e^A}{e^A + e^B} \left(1 - \frac{e^A}{e^A + e^B}\right) \\ &= a_j^{(n_l)} (1 - a_j^{(n_l)}) \end{aligned} \quad (28)$$

4.3 激活函数

sigmoid sigmoid 函数表达式如下

$$f(x) = \frac{1}{1 + e^{-x}} \quad (29)$$

其图像如下图所示



sigmoid 激活函数考虑将输入值映射到 $(0, 1)$ 的区间中，该函数在定义域内连续，且导数大于 0。它也有较为简单的求导结果

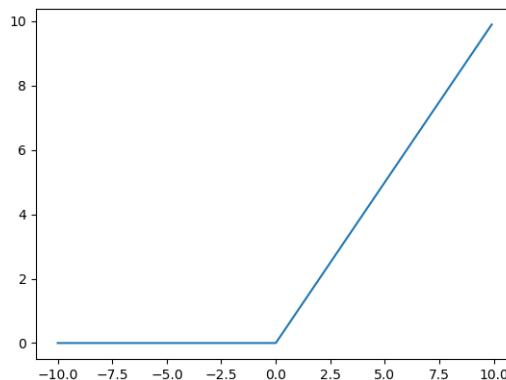
$$f'(x) = f(x)(1 - f(x)) \quad (30)$$

但是在神经网络中，特别是对于层数较多的网络，通常不采用 sigmoid 作为激活函数，主要是因为其容易产生梯度消失的情况。当输入非常大或非常小的时候，其梯度趋近于 0，反向传播的过程中直接导致梯度无法传播，无法有效地调整权重。虽然做标准化可以让数据近似服从正态分布，但梯度消失仍有可能产生，在学习过程中可能会产生输入较大或较小的情况。或许这个问题可以用 batch-normalization 来缓解，但明显采取一种更佳的激活函数是较为可取的做法。

ReLU ReLU 函数表达式如下

$$f(x) = \max\{0, x\} \quad (31)$$

图像如下



其决定它有非常简单的求导结果

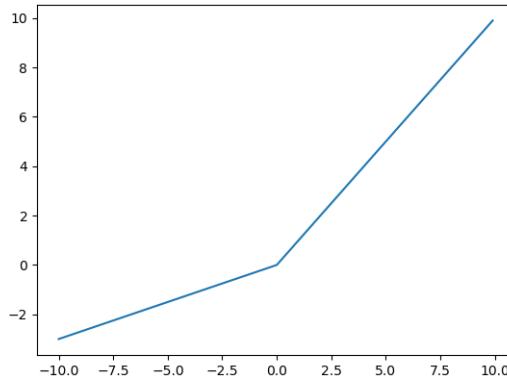
$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \quad (32)$$

RuLU 收敛能比 sigmoid 快的多，一方面其计算快，比起 sigmoid 函数的导数需要指数运算，RuLU 只需要做大小的比较。另一方面，其梯度经过多个层传播之后，多数能够保持原汁原味，比起 sigmoid 会梯度消失要好得多。然而，RuLU 也有弱点，当 $x < 0$ 时 $f(x) = 0$ ，梯度为 0，这直接导致该神经元失活。因而在训练过程中，要注意取较小的学习率。

Leaky ReLU Leaky ReLU 是针对 RuLU 的弱点而改进的，其考虑用一个比较小的数去替代 $x < 0$ 时的 $f(x) = 0$ ，即

$$f'(x) = \begin{cases} x, & x > 0 \\ ax, & x < 0 \end{cases} \quad (33)$$

图像如下



其求导结果为

$$f'(x) = \begin{cases} 1, & x > 0 \\ a, & x < 0 \end{cases} \quad (34)$$

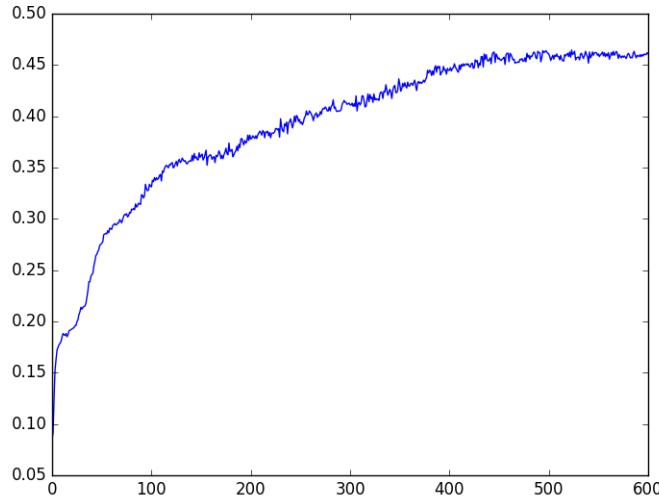
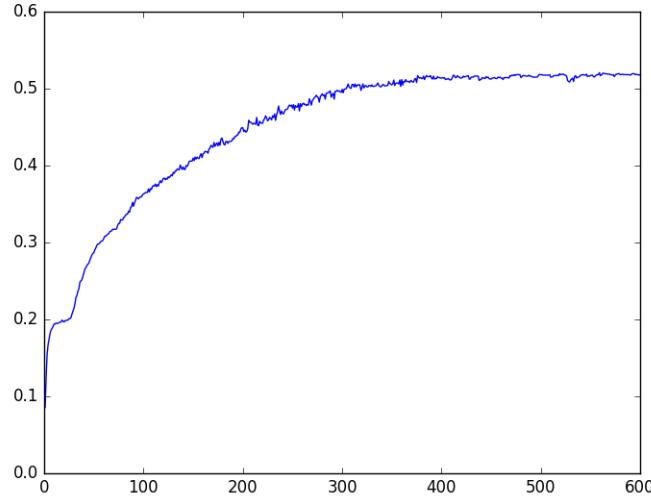
这个方法可以使 $x < 0$ 处避免失活，但是额外引入了超参数 a 。

PReLU PReLU 是针对 Leaky ReLU 的进一步优化，其考虑在反向传播过程中，也对 a 进行学习，从而避免引入超参数 a 。一些实验^[1] 证明这种优化能取到好的学习效果。

4.4 传统 BP 网络的应用

以上介绍的 BP 网络的算法以及较为传统的结构，我们想探究随着图像尺寸的变化（即输入大小）以及隐含层神经元。首先我们制备数据，通过 opencv 的方法，将输入图像归一化为同一大小，分别为

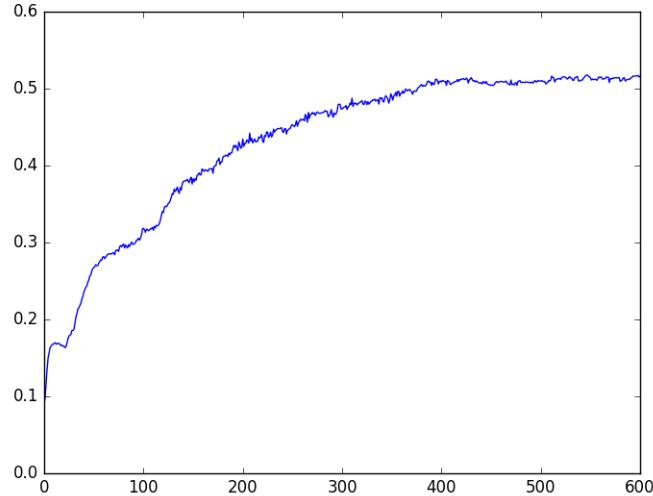
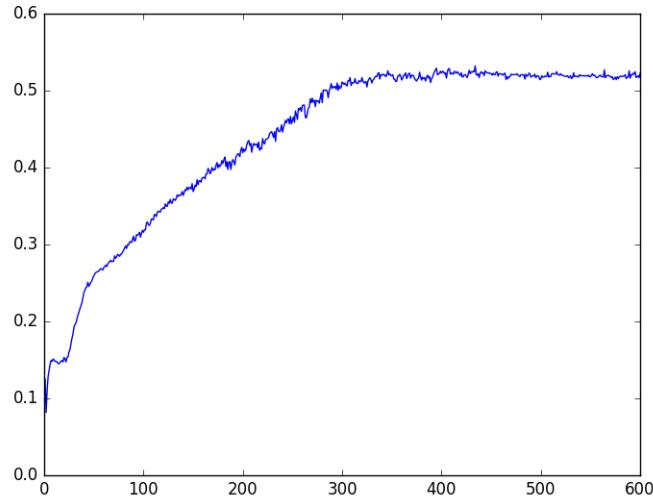
64×64 , 96×96 , 128×128 , 学习率设置 0.03, 优化函数采用 Mini-batch, 以 8 个样本作为一个 batch, epoch 设为 600。首先考虑当隐含层分别设为 1000 和 500 时, 图像大小为 64×64 时, 模型的训练准确率如下:



图像大小为 64×64 时, 隐含层为 1000 和 500 的准确率图

从图中可以看出, 隐含层为 1000 时比 500 好接近 5%, 收敛速度上, 前者在 epoch 为 300 时就趋于稳定, 后者在 epoch 为 450 时趋于稳定。其原因四隐含层 1000 时, 其自由度比 500 大, 随着参数的增加, 更有可能得到偏差小的模型。从实验可以看出, 前者相比于后者在达到较低偏差的同时, 其方差也不会很大。

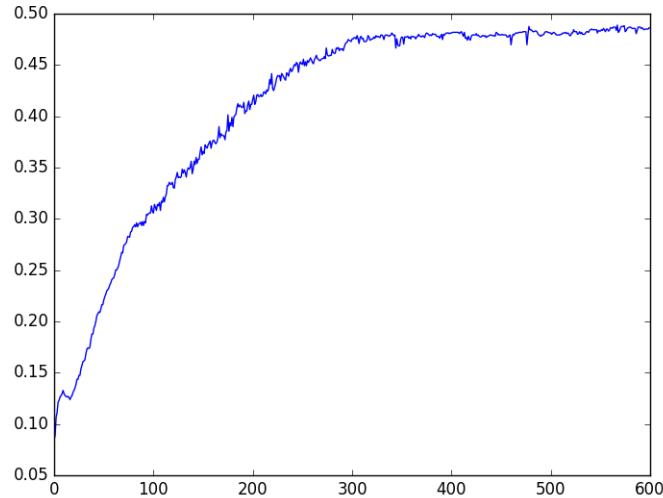
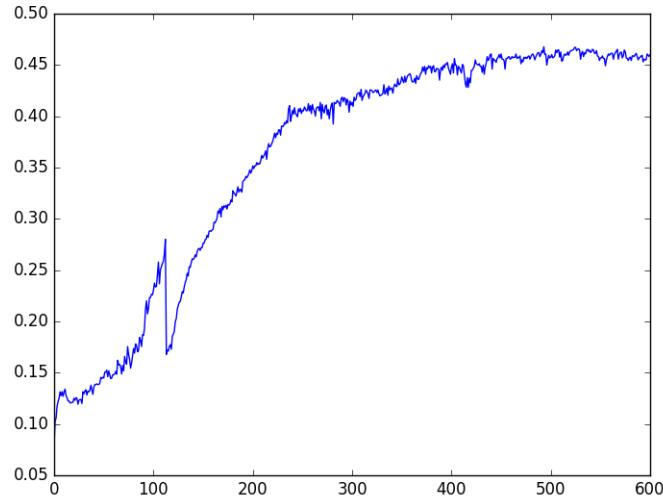
当隐含层分别设为 1000 和 500 时, 图像大小为 96×96 时, 模型的训练准确率如下:



图像大小为 96×96 时，隐含层为 1000 和 500 的准确率图

从图中可以看出，隐含层 1000 与 500 在准确率上持平，为 50% 左右。由于随着图像的尺寸增加，过拟合的风险增大。而前者相比于后者有更低的模型复杂度，一定程度上抵制了过拟合。而过拟合的风险随着图像尺寸的增大而增大的现象，我们将在下图进一步看到：

当隐含层分别设为 1000 和 500 时，图像大小为 128×128 时，模型的训练准确率如下：



图像大小为 128×128 时，隐含层为 1000 和 500 的准确率图

从图中可看出，当隐含层为 1000 时，其训练过程中准确率出现了大幅度的震荡，而且准确率收敛在了 45% 左右，而隐含层为 500 的模型相比隐含层为 1000 的模型的更加健壮，而且准确率接近 50%，比隐含层为 1000 的模型高了大概 4%。

综上，我们可以得到各个模型的准确率表格

	64×64	96×96	128×128
1000	0.518188	0.522655	0.460115
500	0.460753	0.516273	0.48628

4.5 梯度下降方法

梯度下降法的选取能影响收敛速度与质量，它也是模型构成的一部分。在应用中一般有如下的梯度下降法可供选择

批量梯度下降法 批量梯度下降法 (Batch Gradient Descent) 考虑在计算了所有样本之后再对参数进行更新，即

$$W^{(l)} = \sum_{i=1}^m W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \quad (35)$$

由于通常训练的样本非常大，若在计算所有样本之后再进行参数更新，会让更新的速度减慢。另外，模型实现一般会采用矩阵运算，BGD 占的内存会非常多，从而影响计算速度。

随机梯度下降法 随机梯度下降法 (Stochastic Gradient Descent) 的想法与 BGD 截然不同，计算每一个样本之后便进行一次反向传播，对参数进行更新，即

$$W^{(l)} = W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x, y) \quad (36)$$

相比之下，SGD 的训练速度比 BGD 快得多，在 BGD 进行一次反向传播的时间内，SGD 已经进行过多次传播。但是在梯度下降过程中，SGD 容易出现震荡，由于单个样本并不能代表梯度最大的方向，也有可能导致解非最优的情况。

小批量梯度下降法 小批量梯度下降法 (Mini-batch Gradient Descent) 考虑了批量梯度下降法和随机梯度下降法的优缺点，并进行结合，考虑将数据集划分成多个含有较小数据的 batch，然后对这些 batch 分别采用 BGD。下面给出第 i 个 batch 的训练公式

$$W^{(l)} = \sum_{(x,y) \in b_i}^m W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x, y) \quad (37)$$

其中， b_i 代表当前 batch 所包含的训练样本 (x, y) 的集合。

动量梯度下降法 无论是 SGD 还是 MGD，即便 MGD 已在 SGD 上做了优化，在训练过程中仍可能有振荡的风险。一种优化的方法是基于 SGD，在对参数 $W^{(l)}$ 进行更新时，会考虑上一次的更新幅度，若是当前的梯度方向与上一次的相同，则能够加速收敛，反之则能抑制更新，这也是采用了动量的想法。其算法如下

1 输入：学习率 ϵ ，动量参数 α

2 $t_{dW} = \alpha t_{dW} + (1 - \alpha)t_{dW}$

3 $W = W - \epsilon t_{dW}$

4.6 正则化与 dropout

机器学习中，常会发生过拟合的情况，通常引起这种情况的原因有数据量过小、维度过大、模型复杂度过大等，而此现象是方差过大且偏差太小所致。通常维度过大可采用特征选择的方法来降维，而模型复杂度可以用正则化项来限制。它是考虑在损失函数中添加能反映出模型复杂度的项。例如在神经网络中，下面的损失函数的第二项称为 L2 正则化

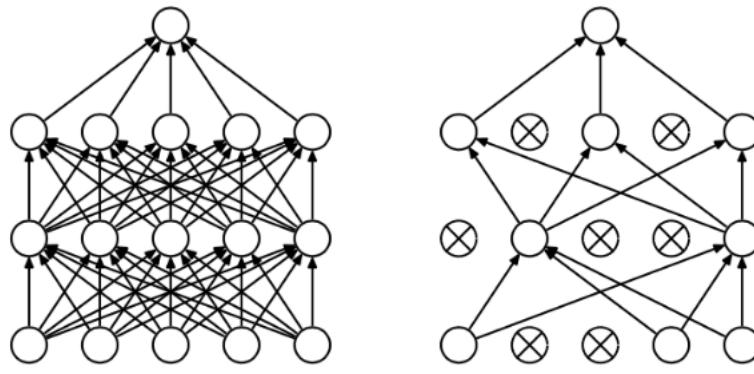
$$J(W, b; x, y) = - \sum_i y_i \log a_i^{(n)} + \lambda \sum_w w^2 \quad (38)$$

我们可以把损失函数看出是由偏差衡量项（第一项）和方差衡量项（第二项）组成，其本质是偏差、方差权衡，权衡通过 λ 来实现。

除了 L2 正则化之外，常用的还有 L1 正则化，为如下形式

$$J(W, b; x, y) = - \sum_i y_i \log a_i^{(n)} + \lambda \sum_w |w| \quad (39)$$

神经网络中，除了加入正则化项之外，还能考虑在每次训练中，让所有神经元以一定概率失活，即封闭该神经元的输出，此方法成为 dropout。因而在每次训练中，网络结构都不一样，在降低模型复杂度的同时，也是对于多个模型的集成，其示意图如下。



dropout 工作原理示意图

4.7 BP 神经网络 +

5 卷积神经网络

5.1 卷积神经网络概述

5.2 AlexNet

5.3 VGGNet

5.4 Inception

5.5 ResNet

5.6 迁移学习

5.7 CNN+

5.8 Batch Normalization

6 参考文献

[1] Kaiming He,Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,<https://arxiv.org/abs/1502.01852>