1 用自己的话叙述从逻辑回归到神经元工作原理

逻辑回归是一种用于解决二分类（0 or 1）问题的机器学习方法，其假设函数是 sigmoid 函数，形式如下：

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

其中 x 是我们的输入，Θ 为我们要求取的参数, sigmoid 函数能将任意的实数值映射到（0，1）区间，因此逻辑回归以假设函数值为 0.5 作为阈值，限定在给定 x 和 Θ 条件下二分类问题的分类结果。

相对于生物学角度的神经元，神经网络中的神经元可以看作是神经网络中的对应一个特定数学计算模型的计算单位。

下面图展示了一个常用的数学模型。

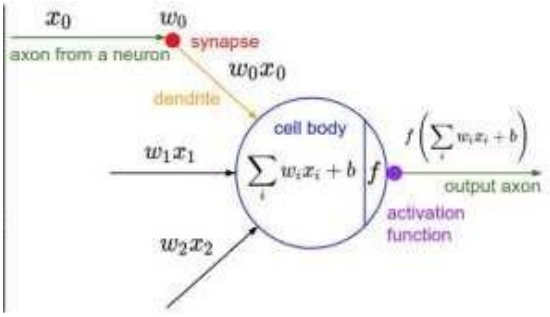

图 1. 1

图中x_i是输入数据里的自变量，w_i是神经网络中的权重项,b 表示偏差，f是一个非线性的激活函数，它将在网络中引入非线性的因素。

首先将输入进行加权求和加上偏执，得到待激励值，然后将该值作为输入，输入到激活函数中，最后输出的是一个激励后的值，这里的激活函数可以看成对生物中神经元的激活率建模。

一个神经元可以看成包含两个部分，一个是对输入的加权求和加上偏置，一个是激活函数对求和后的激活或者抑制。

逻辑回归就可以看做是仅含有一个神经元的单层的神经网络。

2 给出至少 2 种常用的激活函数

（1） Sigmoid 函数

Sigmoid 函数如图 2.1 所示，它无论输入值是无穷大还是无穷小，输出值范围均为(0,1)，很适合二分类问题。在神经元模型里使用 sigmoid 函数，就相当于给神经元的输出赋予了概率意义，当 sigmoid 神经元的输出大于 0.5 时，则预测类别为 1，否则预测类别为

0。但它也具有会引起梯度消失、在深度神经网络时候相比其他运算就比较慢等问题。

(2)Relu 函数

Relu 函数的函数图如下所示，当输入小于 0 时，输出为 0，当输入大于 0 时，输出等于输入。其函数形式为：

$$Relu=max(0,x)$$

Relu 函数可以避免梯度爆炸和梯度消失等问题，计算过程简单，但是当 x 是小于 0 的时候，那么从此所以流过这个神经元的梯度将都变成 0；这个时候这个 ReLU 单元在训练中将死亡（也就是参数无法更新），这也导致了数据多样化的丢失（因为数据一旦使得梯度为 0，也就说明这些数据已不起作用），而 MaxOut 函数可以改善这个问题。

（3） SoftMax 函数

softmax 经常用在神经网络的最后一层，它将多个神经元的输出，映射到（0,1）区间内，进行多分类。假设输入向量为$Z = (z_1, z_2, \dots, z_n)$,SoftMax 函数输出公式如下：

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

图 2.3 是 SoftMax 函数的一个形象图示。

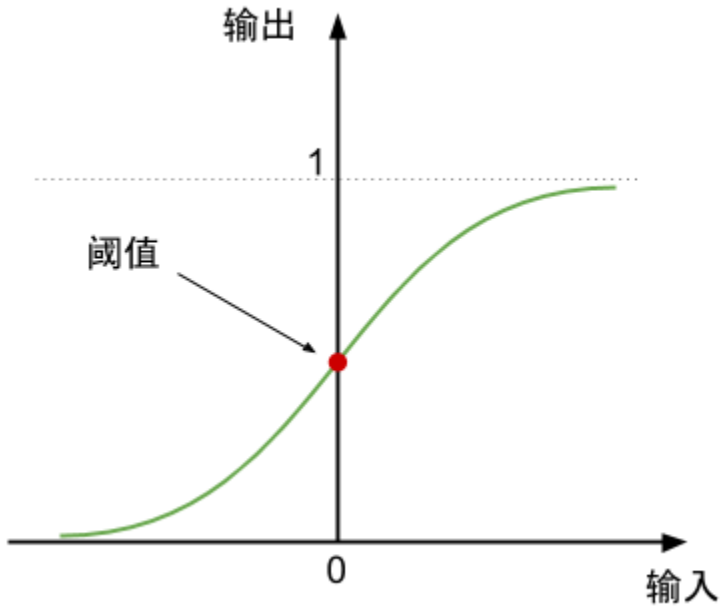

图 2. 1

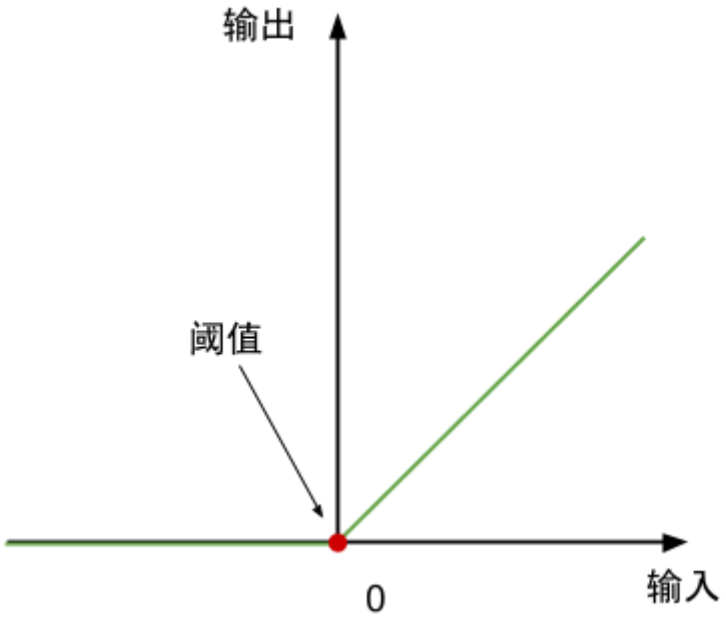

图 2.2

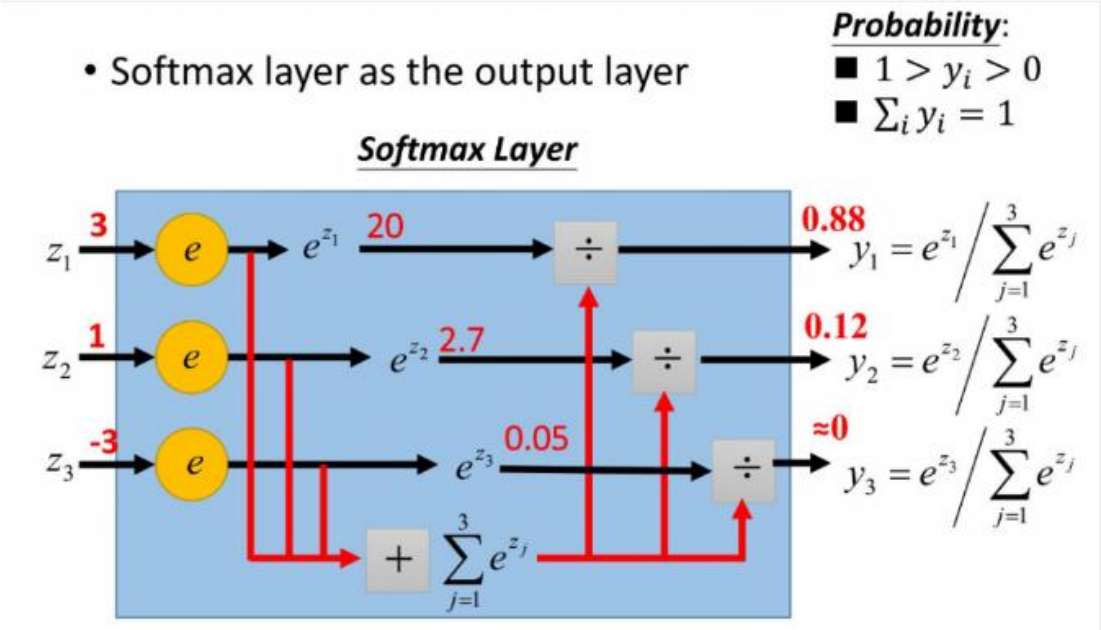

图 2.3

3 用数学方程描述 感知器模型，用到的多层神经网络模型

感知器模型：

如下图 3.1 感知器模型：

主要完成了两步：

（1）接收多个输入，并为每个输入加上一个权值 w 后累加，再加上一个偏置项 b。

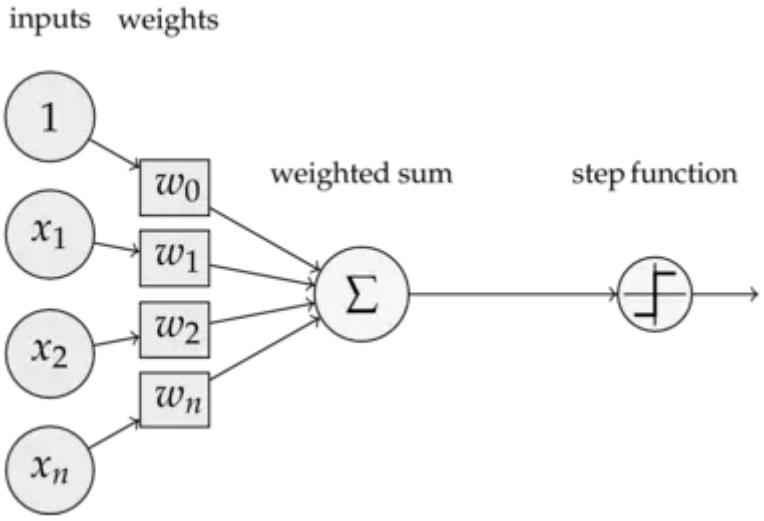

图 3.1

（2）用激活函数 f 激活（1）中的输出结果。

那么输出可以表示为以下公式：

$$y = f(w \cdot x + b)$$

多层神经网络模型：

以图 3.2 多层神经网络为例：

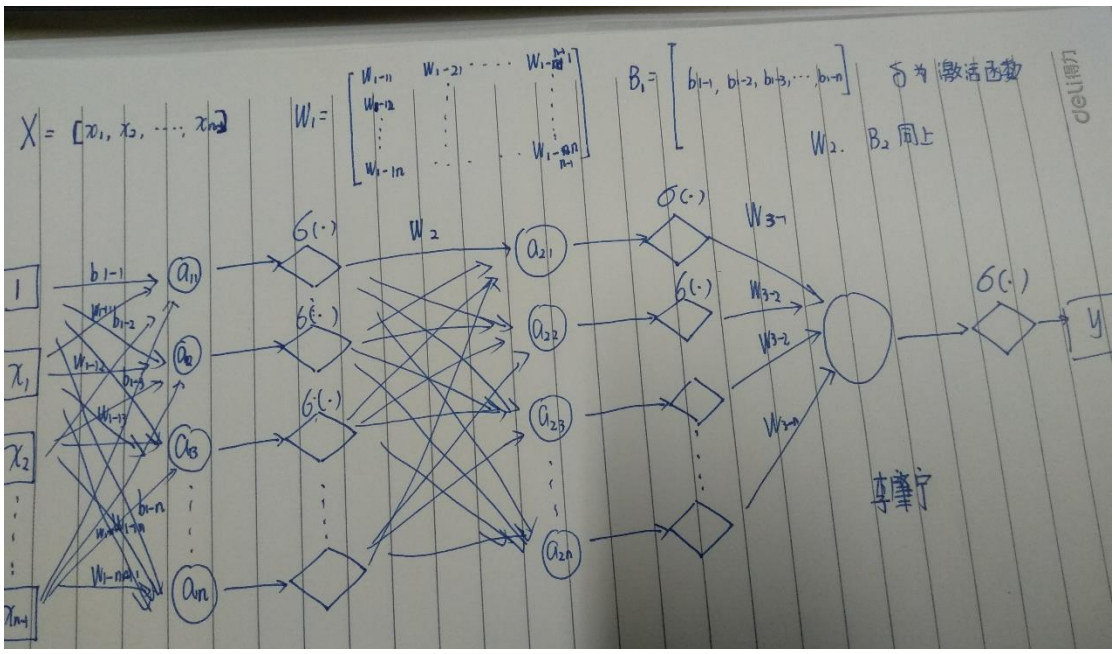

图 3.2

公式表示为：

$$a_1 = X \cdot W_1 + B_1$$

$$a_2 = W_2 \cdot \sigma(a_1) + B_2$$

$$y = \sigma(W_3 \cdot \sigma(a_2) + B_3)$$

4 简述卷积神经网络要素：卷积核，滤波器，池化，特征图

卷积核：卷积操作是通过将一个滤波矩阵与图像中不同数据窗口的数据作内积提取图像不同频段的特征的操作。卷积核就是这个滤波矩阵，带着一组固定权重的神经元，通常是 n*m 二维的矩阵，n 和 m 也是神经元的感受野。n*m 矩阵中存的是对感受野中数据处理的系数。通过卷积层从原始数据中提取出新的特征的过程又成为 feature map(特征映射)。filter_size 是指 filter 的大小，例如 3*3；filter_num 是指每种 filter_size 的 filter 个数，通常是通道个数。

滤波器：滤波器和卷积核意义相同，就是在卷积和池化操作中在输入上移动进行卷积或取最大值（平均值）的滤波操作的小矩阵窗口。

池化：池化操作通常在卷积操作之后，它能够保留主要特征的同时减少参数和计算量，防止过拟合。在经过卷积层提取特征之后，得到的特征图代表了 比 像素 更高级的特征，已经可以交给 分类器 进行训练分类了。但是我们 每一组卷积核 都生成 一副与原图像素相同大小的 卷积图，节点数一点没少。如果使用了 多个卷积核 还会使得通道数比之前更多，所以卷积之后我们需要进行池化，也就是进行降维。

池化操作也是利用一个滤波矩阵在特征图上进行扫描，通过取最大值或者平均值等来减少元素的个数实现降维。一个值得记住的知识点是，最大值和平均值的方法可以使得特征提取拥有“平移不变性”，也就说图像有了几个像素的位移情况下，依然可以获得稳定的特征组合，平移不变形对于识别十分重要。

如下图 4.1 是我们本次 MINIST 数据集实验所使用的池化函数。

```
l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
```

图 4. 1

l1a 就是来自卷积操作的输出，是本次池化操作的输入，ksize 指的是池化窗口的大小，长宽分别为 2，因为我们不想在 batch 和 channels 上做池化，所以这两个维度设为了 1。Strides 是窗口在每一个维度上滑动的步长。Padding 为 same 指的是在对输入进行补边使得池化后输出的特征图尺寸不变。

特征图：我对于特征图的理解是这样的，以第一个卷积层为例，假设输入的为一

个 32x32 像素的图片，第一个卷积层共有 8 个卷积核，那么就会以 8 种通道对原始输入的图片进行提取 8 种不同特征的卷积滤波操作，那么输出的就是就分别对应 8 种通道的特征的特征图。

5 给出实验用到的模型图示，给出实验中的关键代码并解释其实现原理

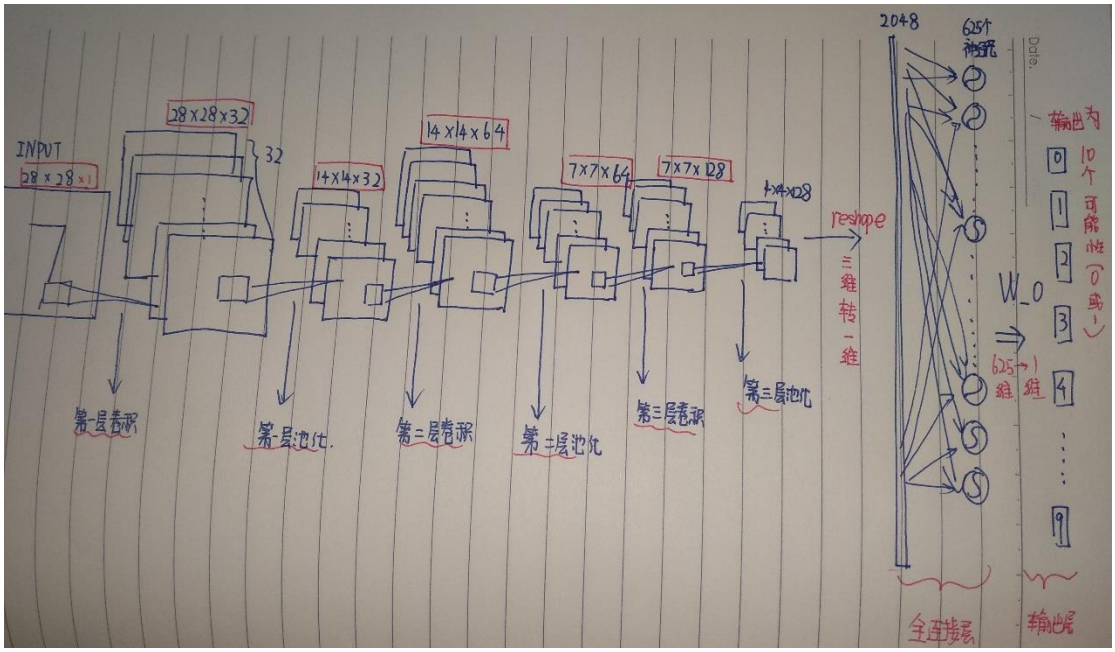

图 4. 1

如图 5.1 为实验所用到的模型图示，以下为负责定义输入数据、初始化卷积核等参数、定义网络结构以及定义损失函数、训练操作、预测操作等的关键代码及其注释解释：

定义输入数据

```
#从 MNIST 数据集中获取训练数据、测试数据和标签
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
trX, trY, teX, teY = mnist.train.images, mnist.train.labels,
mnist.test.images, mnist.test.labels
#数据预处理
trX = trX.reshape(-1, 28, 28, 1)  # 28x28x1 input img
teX = teX.reshape(-1, 28, 28, 1)  # 28x28x1 input img
```

初始化卷积核等参数

```
#第一层卷积核大小 3x3，输入维度 1，输出维度 32
w = init_weights([3, 3, 1, 32])       # 3x3x1 conv, 32 outputs
#第二层卷积核大小 3x3，输入维度 32，输出维度 64
w2 = init_weights([3, 3, 32, 64])     # 3x3x32 conv, 64 outputs
#第三层卷积核大小 3x3，输入维度 64，输出维度 128
w3 = init_weights([3, 3, 64, 128])    # 3x3x32 conv, 128 outputs
```

```
#全连接层，输入维度 128x4x4 上层输数据三维转一维，输出维度 625 第一层卷积核大小 3x3，输入维度 128，输出维度 625
w4 = init_weights([128 * 4 * 4, 625]) # FC 128 * 4 * 4 inputs, 625 outputs
# 输出层，输入维度 625，输出维度 10 代表 10 类(labels)
w_o = init_weights([625, 10])         # FC 625 inputs, 10 outputs (labels)
# 定义 dropout 占位符
p_keep_conv = tf.placeholder("float")
p_keep_hidden = tf.placeholder("float")
py_x = model(X, w, w2, w3, w4, w_o, p_keep_conv, p_keep_hidden)
```

定义网络结构

```
def model(X, w, w2, w3, w4, w_o, p_keep_conv, p_keep_hidden):
    #第一层卷积层 步长为 1
    l1a = tf.nn.relu(tf.nn.conv2d(X, w,   strides=[1, 1, 1, 1], padding='SAME'))    # l1a shape=(?, 28, 28, 32)
    #第一层池化层，窗口大小为 2X2，步长为 2，取最大值
    l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME') # l1 shape=(?, 14, 14, 32)
    #第一层 dropout
    l1 = tf.nn.dropout(l1, p_keep_conv)
    #第二层卷积层 步长为 1
    l2a = tf.nn.relu(tf.nn.conv2d(l1, w2, strides=[1, 1, 1, 1], padding='SAME')) # l2a shape=(?, 14, 14, 64)
    #第二层池化层，窗口大小为 2X2，步长为 2，取最大值
    l2 = tf.nn.max_pool(l2a, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME') # l2 shape=(?, 7, 7, 64)
    #第二层 dropout
    l2 = tf.nn.dropout(l2, p_keep_conv)
    #第二层卷积层 步长为 1
    l3a = tf.nn.relu(tf.nn.conv2d(l2, w3, strides=[1, 1, 1, 1], padding='SAME')) # l3a shape=(?, 7, 7, 128)
    #第三层池化层，窗口大小为 2X2，步长为 2，取最大值
    l3 = tf.nn.max_pool(l3a, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME') # l3 shape=(?, 4, 4, 128)
    #第三层 reshape
    l3 = tf.reshape(l3, [-1, w4.get_shape().as_list()[0]])    # reshape to (?, 2048)
    #第三层 dropout
    l3 = tf.nn.dropout(l3, p_keep_conv)
    # 全连接层，dropout 部分神经元
    l4 = tf.nn.relu(tf.matmul(l3, w4))
    l4 = tf.nn.dropout(l4, p_keep_hidden)
    #输出层
```

```
    pyx = tf.matmul(l4, w_o)
    return pyx
```

定义损失函数、训练操作、预测操作

```
# 损失函数使用的是 softmax 交叉熵
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=
py_x,
labels=Y))
# 定义训练操作
train_op = tf.train.RMSPropOptimizer(0.001,
0.9).minimize(cost)
# 定义预测操作
predict_op = tf.argmax(py_x, 1)
```