**神经网络报告**

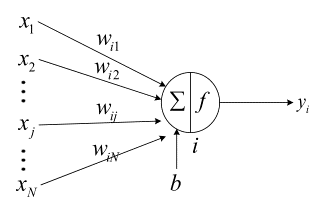
1. **神经网络概述**

人工神经网络（Artificial Neural Network，即ANN ），是20世纪80 年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象， 模拟动物中枢神经系统的计算模型，使机器像人脑一样具有学习感知能力。神经网络是一种运算模型，由大量的节点（或称神经元）之间相互联接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重。网络的输出则依网络的连接方式，权重值和激励函数的不同而不同。

1. **人工神经元**

人工神经元模型是由人量处理单元厂泛互连而成的网络，是人脑的抽象、简化、模拟，反映人脑的基本特性。图1为一个典型的人工神经元模型。

图1 人工神经元模型



+1





模型的输入信号的各个分量跟相应的权值的线性和作为神经元的最终输入后经非线性函数变换后输出信息，满足公式如下：

 （1）

其中，为神经元的输入信号；为神经元的输出；为输入信号对神经元的权重；为神经元受到的外部刺激的权重；表示激励函数，为非线性映射函数。激励函数可取Sigmod函数，tan函数以及Relu线性纠正函数，函数曲线如图2所示。神经元取不同的激活函数相当于对输入信息进行不同的非线性变换，输出不同的结果，从而使人工神经元具有不同的表现能力。

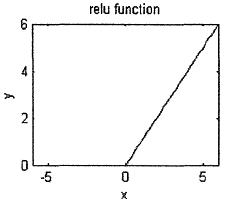
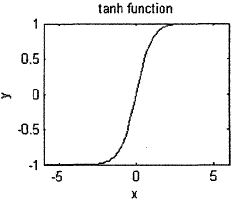
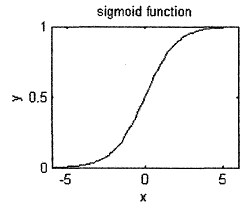


图2 非线性激励函数

1. **人工神经网络**

人工神经网络是多个神经元连接起来的网络模型，典型的神经网络模型包括感知器、BP神经网络、反馈神经网络等。当神经元模型确定后，神经网络的特性与功能主要取决于网络的拓扑结构及学习方法。

图3为一个简单的神经网络，圆形代表神经网络的各个神经元节点，Layer1为该神经网络的输入层，Layer2为神经网络的隐藏层，Layer3为神经网络的输出层。可以观察到，在不考虑偏置节点的情况下，图中的神经网络包含三个输入节点，三个隐藏节点和一个输出节点。

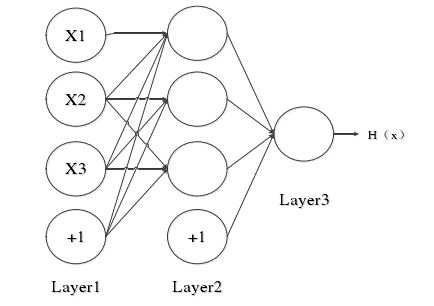


图3 简单的神经网络结构图[7]

设为神经网络的层数；为Layer；为层神经元和层神经元之间的权重；为第层神经元的偏置；表示第层神经元的输出值。以图3的神经网络为例，神经网络的前向传播算法( Forwardpropagation Algorithm )计算过程为：

(2)

令为网络第2层第个神经元的加权和，即满足且则

 (3)

由此式(2)的矩阵形式为

 (4)

1. **卷积神经网络**

4.1卷积神经网络的架构

卷积神经网络是一种主要用于二维数据处理的深度神经网络模型。其可视为一个多层的神经网络，每层由多个二维平面构成，每个平面由多个独立的神经元组成。卷积神经网络一般包含以下几个层：

1. 输入层：用于数据的输入。
2. 卷积层：使用卷积核进行特征提取。
3. 激励层：非线性映射，由于卷积是线性运算，需进行非线性处理。
4. 池化层：进行下采样，对特征图进行稀疏处理，减少数据运算量
5. 全连接层：位于卷积神经网络尾部以进行重新拟合，减少特征信息的损失。
6. 输出层：用于结果的输出。

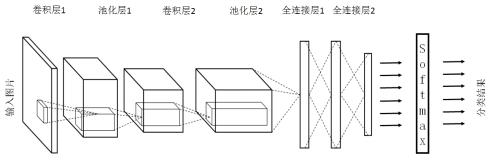


图5 卷积神经网络架构图

卷积层与池化层联系较为紧密，一般情况下信息经过卷积层进行处理后，可以提取图像特征，再通过池化层将提取的特征进行降维处理，从而降低整个神经网络参数的数量级。卷积层与池化层提取信息特征后，通过全连接层连接至传统神经网络以完成图像分类等任务。图5为用于图像分类问题的一类卷积神经网络架构图。

4.2 卷积层

卷积层也叫特征提取层，假设卷积层的输入为一个的二维神经元，定义为一个感受视野。一个感受视野带有一个卷积核，将感受视野中的权重称为卷积核，感受视野对输入的扫描间隔称为步长。计算过程如图6所示，黄色部分为感受视野，当步长为1时，卷积核以一个神经元为间隔，从左到右从上到下对输入的二维神经元做卷积运算，计算出的值即粉色框中的数据就是提取出的特征，该过程为线性运算过程，满足公式如下：

 (5)

卷积核上的参数大小决定了卷积核覆盖的区域对应的神经元对最终卷积结果的重视度，权重越大，重视度越高。一个带有卷积核的感受视野扫描生成的下一层神经元称为特征映射图。由此同一个特征映射图上的神经元使用的卷积核是相同的。若使用三个卷积核则可以输出三个特征映射图。

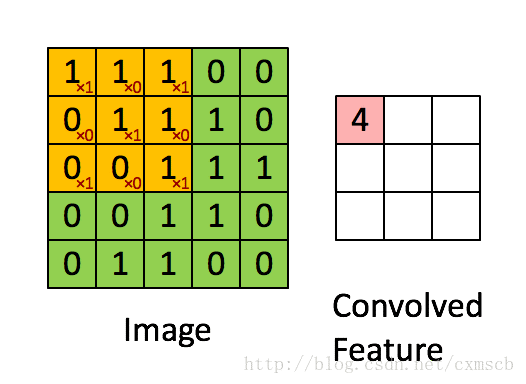


图6 卷积处理示意图

Input neurons

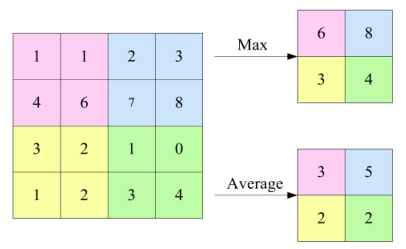
First hidden layer

4.3 池化层

理论上，卷积神经网络通过卷积层提取输入图像特征后就可以使用特征训练网络。而实际中，训练神经网络面临的重要问题是庞大的计算量，当输入数据尺寸较大时网络的训练速度进行十分缓慢。为了进一步减少运算数据量，降低网络的训练时间，采用池化处理来减少数据量。

池化层也有一个池化视野，其包含最大池化和平均池化两种计算方式，扫描过程类似卷积过程，如图7所示，其中扫描步长为2。

图7 池化处理示意图



1. **关于卷积神经网络的仿真**

本次仿真基于Tensorflow框架，对MNIST数据集进行分类。MNIST是应用最广泛的手写数字数据集，拥有60000个训练样本集和10000个测试样本集。这些数字样本都需经过归一化处理，每个数字样本图像大小为，数据库包括0到9共10个阿拉伯数字，每个数字样本都有对应的数字标签如图8所示。

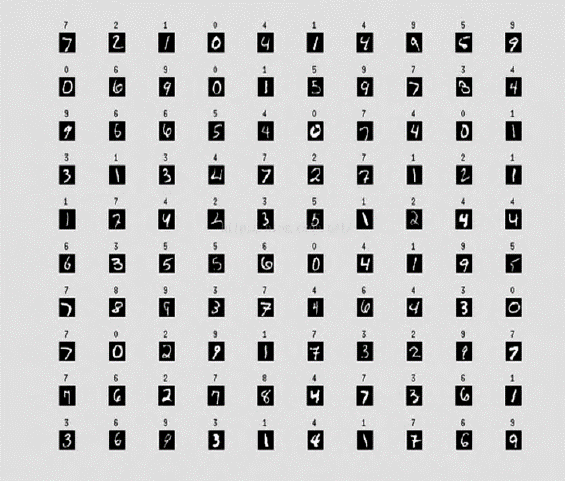


图8 MNIST数据集部分可视化数字

不同参数设置对卷积神经网络性能有不同的影响，为研究卷积核大小对卷积神经网络性能的影响，保持其他参数不变，选取不同大小的卷积核，其分类结果如下：

|  |  |  |
| --- | --- | --- |
| 表1 卷积神经网络不同卷积核大小性能对比 | | |
| ksize | correctness | loss |
|  | 0.9629 | 0.130016 |
|  | 0.9723 | 0.087545 |
|  | 0.9670 | 0.103182 |

从表1中可以观察到，卷积核并非越大越好，也并非越小越好，本例中取的卷积核性能最优。

选择两种池化操作进行对比，获得的结果如下表所示：

|  |  |  |
| --- | --- | --- |
| 表2 卷积神经网络三种池化操作性能对比 | | |
| Method | correctness | loss |
| avg | 0.9689 | 0.107492 |
| max | 0.9723 | 0.087545 |

由表2可知最大池化操作的性能测试结果优于平均池化操作，这是由于平均池化操作带入过多的干扰信息造成的。

最大池化操作取不同核大小的实验结果如下：

|  |  |  |
| --- | --- | --- |
| 表3 最大池化操作不同核大小性能对比 | | |
| ksize | correctness | loss |
|  | 0.9723 | 0.087545 |
|  | 0.9737 | 0.085037 |
|  | 0.9721 | 0.089812 |
|  | 0.9720 | 0.089814 |

由表3可知，选取最大池化操作时，核大小为时性能最优。核较小时容易造成数据的过拟合，核较大时由于保留了过多的冗余信息将造成误差增大。

采用不同激活函数在MNIST数据集上进行测试，实验结果如下：

|  |  |  |
| --- | --- | --- |
| 表4 卷积神经网络不同激活函数性能对比 | | |
| Method | correctness | loss |
| Sigmoid | 0.8832 | 0.400207 |
| TanH | 0.9633 | 0.128243 |
| ReLU | 0.9723 | 0.087545 |

由表4可知，使用ReLU激活函数的卷积神经网络性能最优，其次为使用TanH激活函数的卷积神经网络。这是由于Sigmoid函数具有饱和性，易出现梯度消散且其输出不以零为中心，易造成数据分散。TanH函数解决了Sigmoid函数不以零为中心的问题，但仍无法解决神经元梯度消散问题。而ReLU函数具有单侧抑制性很好地解决了梯度消散问题。因此ReLU函数相较于传统的激活函数性能更优，也是实际中最常用到的激活函数。

综上可知，卷积神经网络在MNIST数据分类问题中，设置卷积核大小为，取最大池化操作且池化核取，激活函数应用ReLU函数可获得最优性能。

利用MNIST数据集实现卷积神经网络，结构参数设置如下表：

|  |  |  |
| --- | --- | --- |
| 表5 卷积神经网络结构参数 | | |
| Layer | Input | Parameter |
| Layer1 | conv | Output size：32；Kernel：；Pad：SAME |
| ReLU |  |
| MAX pooling | Kernel：；Pad：SAME |
| Layer2 | conv | Output size：64；Kernel：；Pad：SAME |
| ReLU |  |
| MAX pooling | Kernel：；Pad：SAME |
| Layer3 | Full connect | Output size：1024 |
| Layer4 | ReLU + Dropout |  |
| Layer4 | Full connect | Output size：10 |

采用梯度下降法优化网络损耗，程序运行结果如下：

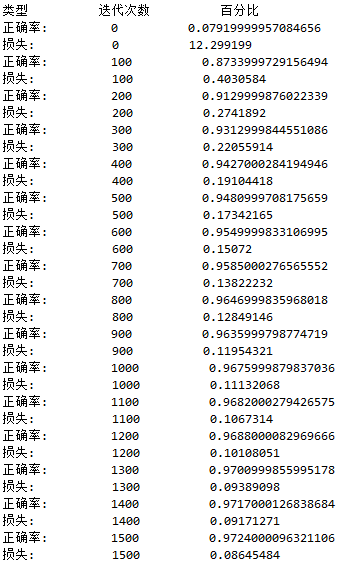


图9 CNN程序运行结果（1）

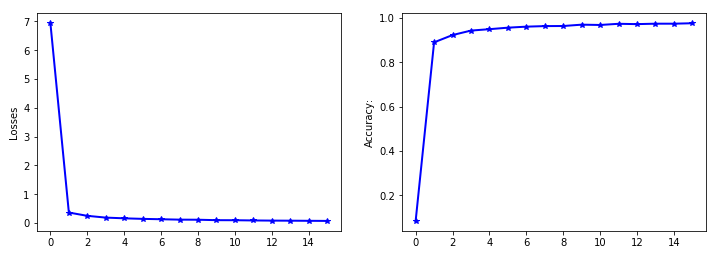


图10 CNN程序运行结果（2）

代码如下：

|  |
| --- |
| from \_\_future\_\_ import print\_function  import tensorflow as tf  from tensorflow.examples.tutorials.mnist import input\_data  import numpy as np  import matplotlib.pyplot as plt      #选去1到10的数字  mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)    def add\_layer(inputs, in\_size, out\_size, activation\_function=None):  W = tf.Variable(tf.random\_normal([in\_size, out\_size]))  b = tf.Variable(tf.zeros([1,out\_size])+0.1)  Wb = tf.matmul(inputs, W)+b  if activation\_function is None:  outputs = Wb  else:  outputs = activation\_function(Wb)  return outputs    def compute\_accuracy\_CNN(v\_xs, v\_ys):  global prediction\_CNN  y\_pre = sess.run(prediction\_CNN, feed\_dict = {xs:v\_xs, keep\_prob:1})  correct\_prediction = tf.equal(tf.argmax(y\_pre,1), tf.argmax(v\_ys,1))  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  result = sess.run(accuracy, feed\_dict = {xs: v\_xs, ys:v\_ys, keep\_prob:1})  return result    def kernel\_variable(shape):  initial = tf.truncated\_normal(shape=shape, stddev = 0.1)  return tf.Variable(initial)    def bias\_variable(shape):  initial = tf.constant(0.1, shape=shape)  return tf.Variable(initial)    def conv2d(x,W):  return tf.nn.conv2d(x, W, strides = [1,1,1,1], padding = 'SAME') #x为输入，W为卷积权重，strides为滤波器移动范围，取步长为1    def max\_pool\_2x2(x):  return tf.nn.max\_pool(x, ksize= [1,2,2,1], strides=[1,2,2,1], padding='SAME')    #占位操作  xs = tf.placeholder(tf.float32, [None, 784]) #输入信息，输入向量为1\*784的矩阵  ys = tf.placeholder(tf.float32, [None, 10]) #类别标签共有10个类别  keep\_prob = tf.placeholder(tf.float32)  x\_image = tf.reshape(xs, [-1,28,28,1]) #将xs转化为28\*28的形式    # 卷积层  w\_conv1 = kernel\_variable([7,7,1,32]) #核 5\*5, 入层大小 1, 出层大小 32  b\_conv1 = bias\_variable([32])  h\_conv1 = tf.nn.relu(conv2d(x\_image, w\_conv1)+b\_conv1) #输出大小 28\*28\*32  h\_pool1 = max\_pool\_2x2(h\_conv1) #输出大小 14\*14\*32    # 卷积层2  w\_conv2 = kernel\_variable([7,7,32,64]) #核 5\*5, 入层大小 32, 出层大小 64  b\_conv2 = bias\_variable([64])  h\_conv2 = tf.nn.relu(conv2d(h\_pool1, w\_conv2)+ b\_conv2) #输出大小 14\*14\*64  h\_pool2 = max\_pool\_2x2(h\_conv2) #输出大小 7\*7\*64    # 全连接层1  w\_fc1 = kernel\_variable([7\*7\*64, 1024])  b\_fc1 = bias\_variable([1024])  h\_pool2\_flat = tf.reshape(h\_pool2, [-1,7\*7\*64])  h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, w\_fc1)+b\_fc1)  h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)    # 全连接层2  w\_fc2 = kernel\_variable([1024,10])  b\_fc2 = bias\_variable([10])  prediction\_CNN = tf.nn.softmax(tf.matmul(h\_fc1\_drop,w\_fc2)+b\_fc2)      #对数据进行预测，784个节点10个分类  prediction = add\_layer(xs, 784, 10, activation\_function= tf.nn.softmax)    #计算预测误差  cross\_entropy\_CNN = tf.reduce\_mean(-tf.reduce\_sum(ys\* tf.log(prediction\_CNN), reduction\_indices=[1]))    train\_step\_CNN = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_CNN)  saver = tf.train.Saver()    #结果输出 ，循环1500次  Total\_test\_loss\_CNN = np.zeros((int(1501/100)+1), float)  Total\_test\_acc\_CNN = np.zeros((int(1501/100)+1), float)  count =0  with tf.Session() as sess:  if int((tf.\_\_version\_\_).split('.')[1]) <12 and int((tf.\_\_version\_\_).split('.')[0])<1:  init = tf.initialize\_all\_veriables()  else:  init = tf.global\_variables\_initializer()  print(tf.\_\_version\_\_)  sess. run(init)  print('类型 ',' 迭代次数 ',' 百分比 ')    for i in range(1501):  batch\_xs, batch\_ys = mnist.train.next\_batch(100)  sess.run(train\_step\_CNN, feed\_dict={xs: batch\_xs, ys: batch\_ys, keep\_prob: 0.5})    if i%100 ==0:  Total\_test\_acc\_CNN[count] = compute\_accuracy\_CNN(mnist.test.images, mnist.test.labels)  print('正确率: ', i,' ', Total\_test\_acc\_CNN[count])  loss\_CNN = sess.run(cross\_entropy\_CNN,  feed\_dict={xs: mnist.test.images, ys: mnist.test.labels, keep\_prob: 1.0})  print('损失: ', i,' ', loss\_CNN)  Total\_test\_loss\_CNN[count] = loss\_CNN  count += 1    #数据存储与绘图  plt.figure(1, figsize=(12, 4))  plt.subplot(121)  plt.ylabel('损失')  plt.plot(Total\_test\_loss\_CNN, 'b\*-', lw=2)    plt.subplot(122)  plt.ylabel('精度')  plt.plot(Total\_test\_acc\_CNN, 'b\*-', lw=2)  plt.show() |