浅 谈 人 工 智 能 及 其 应 用

Hollow Man

目录

摘要

人工智能与深度学习是现在的热门领域，其在游戏、自然语言处理、语音识别、专家系题、视觉系统识别、无人驾驶、智能机器人应用已经十分广泛。本文将介绍这些应用并且讲解其背后实现的机理。

关键词

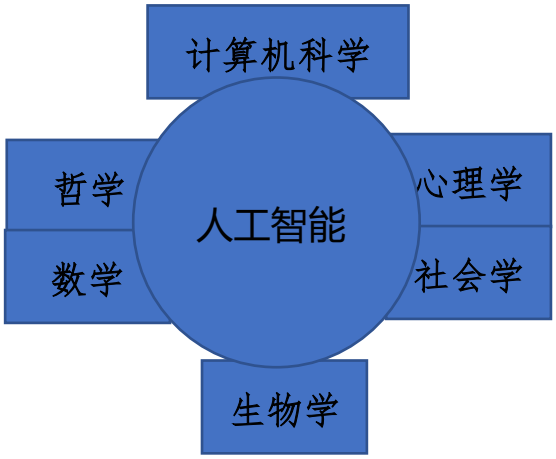
人工智能、深度学习、RNN、CNN、AlphaGo、语音识别、视觉识别、无人驾驶。

一、人工智能简介

2017 年 5 月，在中国乌镇围棋峰会上，谷歌（Google）旗下 DeepMind 公司开发的 AlphaGo 人工智能机器人与排名世界第一的世界围棋冠军柯洁对战，以 3 比 0 的总比分获胜。这标志了人工智能技术已经能在某些人类自认为无法超越的领域超越了人类，一个新时代从此开始了。

自从艾伦·图灵提出图灵测试，1956 年夏季达特茅斯会议首次提出“人工智能”这一术语以来，人工智能三起三落，直到近十年，由于计算机性能的大幅度提高，以大数据为驱动的深度学习算法被广泛应用于人工智能领域。

人工智能是一门包括十分广泛的科学，它由不同的领域组成，许多学科都和它有交集。



二、人工智能应用

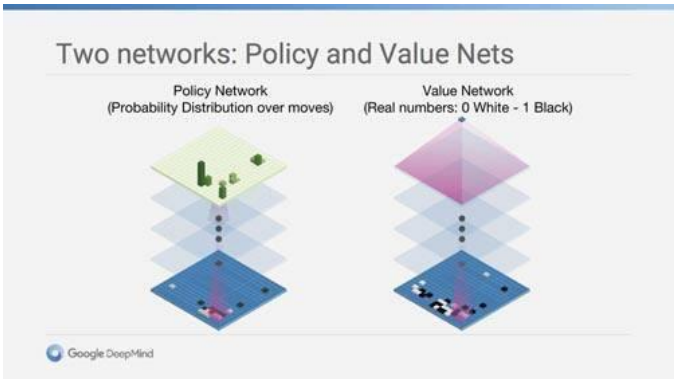
人工智能在以下各个领域占据主导地位

2.1、游戏

人工智能在国际象棋，扑克，围棋等游戏中起着至关重要的作用，机器可以根据启发式知识来思考大量可能的位置并计算出最优的下棋落子。

上世纪 90 年代末，IBM 的智能机器人“深蓝”(Deep Blue)与世界冠军加里卡斯帕罗夫(Garry Kasparov)展开了一系列棋类游戏。1997 年，深蓝最终击败了卡斯帕罗夫，这是人类历史上，机器人首次击败世界冠军。

AlphaGo 将深度学习方法引入到蒙特卡洛树搜索中，主要设计了两个深度学习网络，一个为策略网络，用于评估可能的下子点。另一个为估值网络，可以对给定的棋局进行估值，在模拟过程中，不需要模拟到棋局结束就可以利用估值网络判断棋局是否有利。AlphaGo 采用的是一种革命性的算法——让 AI 在与自己的对抗中学习，即“**对抗学习**”。在这个系统中，AlphaGo Zero 只是简单地与自己对抗，不断地学习如何掌握人类给它编程的任何游戏。在 21 天的学习之后，AlphaGo Zero 就能达到大师级水平，到第 40 天，它就能超越所有之前出现过的版本。



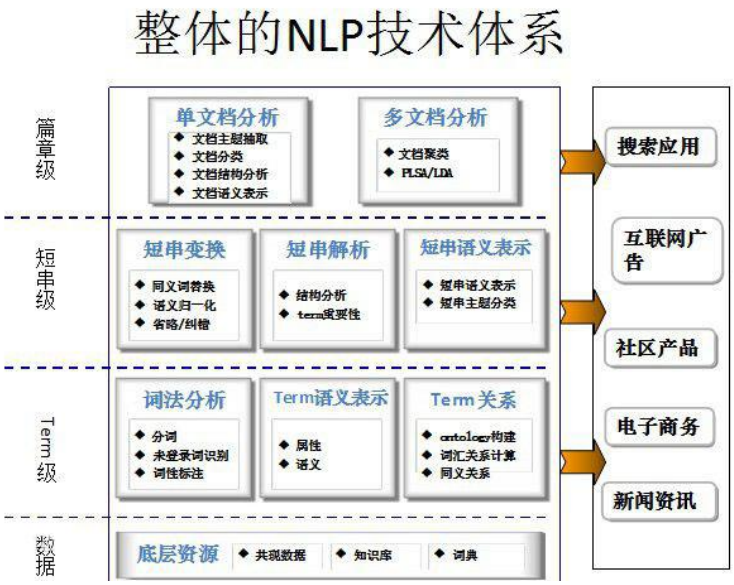
尽管围棋非常复杂，但对于人工智能来说，掌握扑克技术仍是另一个完全不同的挑战性命题。要想在扑克中取胜，人工智能必须掌握欺骗的艺术。在这个“臭

名昭著”的纸牌游戏中，必须具备一定的识别技术和诡骗技术，这也就意味着人工智能必须具有至关重要的动态能力。2017 年 1 月，来自卡内基梅隆大学的一个团队举办了一场公开活动，当时，这个名为 Libratus 的 AI 系统花了 20 天的时间，玩了 12 万把自由德州扑克。尽管专业人士们每天晚上都在讨论在人工智能中发现的其可能存在的弱点，但这台机器每天都在进行自我校正，修补游戏中的漏洞，并改进其牌技和策略。

最近，法尔茅斯大学的一名研究人员开发了一种新的机器深度学习算法，他声称，这种算法可以为我们设计出自己的游戏，让我们从游戏中失败的地方重新开始玩。这个人工智能系统被称为“Angelina”，它每天都在不断改进，现在，它可以利用从维基共享网站到在线报纸和社交媒体等多种来源的数据来制作游戏。游戏开发可以完全由人工智能自身来完成!

2.2、自然语言处理

自然语言处理（NLP）是信息时代最重要的技术之一。理解复杂的语言也是人工智能的重要组成部分。NLP 的应用无处不在，因为人们用语言进行大部分沟通：网络搜索，广告，电子邮件，客户服务，语言翻译，发布学报告等等。NLP 应用背后有大量的基础任务和机器学习模型。常见应用有常见机器翻译系统、人机对话系统。

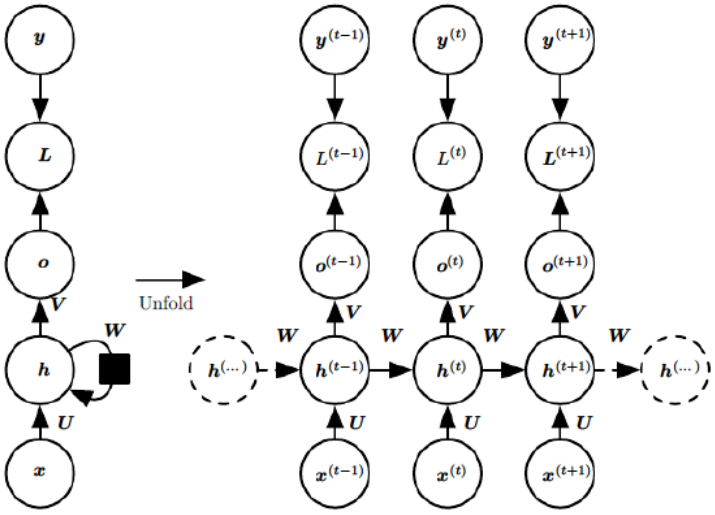


2.3、语音识别

语音识别属于自然语言处理的一部分。智能系统能够与人类对话，通过句子及其含义来听取和理解人的语言。它可以处理不同的重音，俚语，背景噪音，不同人的的声调变化等。

现在的苹果 Siri，微软的小冰、小娜，以及 MIUI 系统上的小爱同学，百度搜索引擎语音识别，都是比较成熟的自然语言处理交互系统。

一种常见的语音识别神经网络——循环神经网络（RNN），其结构如下：



2.4、专家系统

专家系统是一个具有的专门知识与经验的程序系统，它应用人工智能技术和计算机技术，根据某领域一个或多个专家提供的知识和经验，进行推理和判断，模拟人类专家的决策过程，以便解决那些需要人类专家处理的复杂问题。有一些应用程序集成了机器，软件和特殊信息，以传授推理和建议。它们为用户提供解释和建议。比如分析股票行情，进行量化交易。

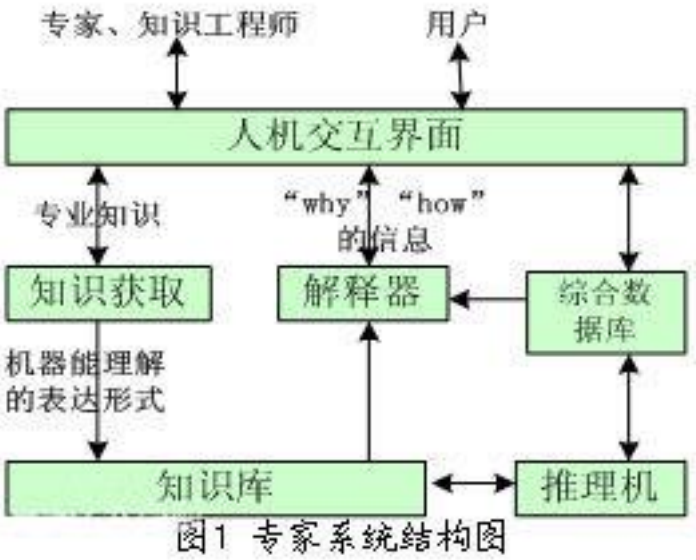


图1 专家系统结构图

20 世纪 60 年代初，出现了运用逻辑学和模拟心理活动的一些原始的通用问题求解程序，它们可以证明定理和进行逻辑推理。但是这些通用方法无法解决大的实际问题，很难把实际问题改造成适合于计算机解决的形式，并且对于解题所需的巨大的搜索空间也难于处理。1965 年，可以推断化学分子结构的世界上第一个专家系统 dendral 从此诞生。

2.5、视觉系统识别

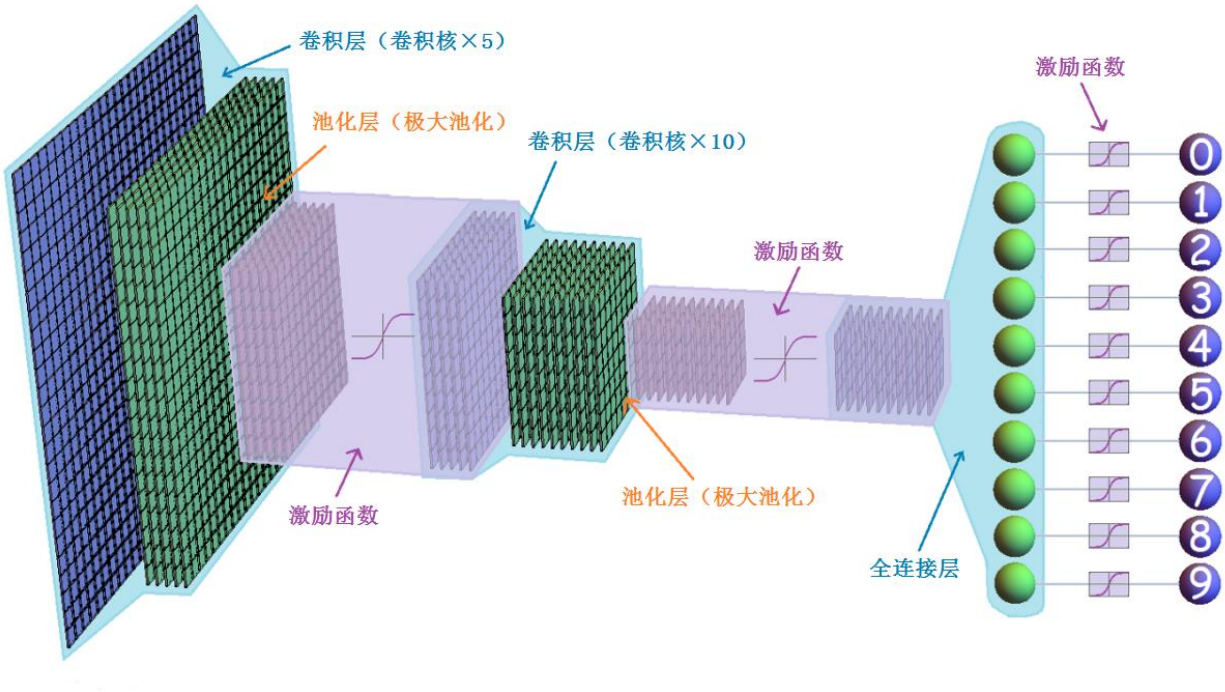
这需要系统理解，解释计算机上的视觉输入。例如，间谍飞机拍摄照片，用于计算空间信息或区域地图。医生使用临床专家系统来诊断患者。警方使用的计

算机软件可以识别数据库里面存储的肖像，从而识别犯罪者的脸部。还有我们最常用的车牌识别等。

视觉感知问题的要点是形成一个精练的表示来取代极其庞大的未经加工的输入信息，把庞大的视觉输入信息转化为一种易于处理和有感知意义的描述。机器视觉可分为低层视觉和高后视觉两个层次，低层视觉主要是对视觉团像执行预处理，例如，边缘检测、运动目标检测、纹理分析等，另外还有立体造型、曲面色彩等，其目的是使对象凸现出来，这时还谈不上对它的理解。高层视觉主要是理解对象，显然，实现高层视觉需要掌捏与对象相关的知识。 机器视觉的前沿研究课题包括：实时图像的并行处理，实时图像的压缩、传输与复原，三绍景物的建模识别，动态和时变视觉等。

2018 年的秋天，在第五届世界互联网大会上中国推出全球首个“人工智能主持人”。这位“主持人”并不简单，它可以随意模仿 99%的主持人，随意切换，不仅模仿他们的声音还能克隆他们的相貌。

视觉识别常采用卷积神经网络（CNN），其结构如下：（另附实现代码见附录）



2.6、无人驾驶

无人驾驶汽车是一种通过车载传感系统感知道路环境，自动规划行车路线并控制车辆到达预定目标的智能汽车。

它是利用车载传感器来感知车辆周围环境，并根据感知所获得的道路、车辆位置和障碍物信息，控制车辆的转向和速度，从而使车辆能够安全、可靠地在道路上行驶。

它集自动控制、体系结构、人工智能、视觉计算等众多技术于一体，是计算机科学、模式识别和智能控制技术高度发展的产物。

2.7、智能机器人

机器人能够执行人类给出的任务。它们具有传感器，检测到来自现实世界的

光，热，温度，运动，声音，碰撞和压力等数据。他拥有高效的处理器，多个传感器和巨大的内存，以展示它的智能，并且能够从错误中吸取教训来适应新的环境。

智能机器人能够理解人类语言，用人类语言同操作者对话，在它自身的“意识”中单独形成了一种使它得以“生存”的外界环境——实际情况的详尽模式。它能分析出现的情况，能调整自己的动作以达到操作者所提出的全部要求，能拟定所希望的动作，并在信息不充分的情况下和环境迅速变化的条件下完成这些动作。

尽管机器人人工智能取得了显著的成绩，控制论专家们认为它可以具备的智能水平的极限并未达到。问题不光在于计算机的运算速度不够和感觉传感器种类少，而且在于其他方面，如缺乏编制机器人理智行为程序的设计思想。你想，现在甚至连人在解决最普通的问题时的思维过程都没有破译，人类的智能会如何呢——这种认识过程进展十分缓慢，又怎能掌握规律让计算机“思维”速度快点呢？因此，没有认识人类自己这个问题成了机器人发展道路上的绊脚石。制造“生活”在具有不固定性环境中的智能机器人这一课题，近年来使人们对发生在生物系统、动物和人类大脑中的认识和自我认识过程进行了深刻研究。结果就出现了等级自适应系统说，这种学说正在有效地发展着。作为组织智能机器人进行符合目的的行为的理论基础，我们的大脑是怎样控制我们的身体呢？纯粹从机械学观点来粗略估算，我们的身体也具有两百多个自由度。当我们在进行写字、走路、跑步、游泳、弹钢琴这些复杂动作的时候，大脑究竟是怎样对每一块肌肉发号施令的呢？大脑怎么能在最短的时间内处理完这么多的信息呢？我们的大脑根本没有参与

这些活动。大脑——我们的中心信息处理机“不屑于”去管这个。它根本不去监督我们身体的各个运动部位，动作的详细设计是在比大脑皮层低得多的水平上进行的。这很像用高级语言进行程序设计一样，只要指出“间隔为一的从1～20的一组数字”,机器人自己会将这组指令输入详细规定的操作系统。最明显的就是，“一接触到热的物体就把手缩回来”这类最明显的指令甚至在大脑还没有意识到的时候就已经发出了。

把一个大任务在几个皮层之间进行分配，这比控制器官给构成系统的每个要素规定必要动作的严格集中的分配合算、经济、有效。在解决重大问题的时候，这样集中化的大脑就会显得过于复杂，不仅脑颅，甚至连人的整个身体都容纳不下。在完成这样或那样的一些复杂动作时，我们通常将其分解成一系列的普遍的小动作 （如起来、坐下、迈右脚、迈左脚)。教给小孩各种各样的动作可归结为在小孩的“存储器”中形成并巩固相应的小动作。同样的道理，知觉过程也是如此组织起来的。感性形象——这是听觉、视觉或触觉脉冲的固定序列或组合（马、人)，或者是序列和组合二者兼而有之。学习能力是复杂生物系统中组织控制的另一个普遍原则，是对先前并不知道、在相当广泛范围内发生变化的生活环境的适应能力。这种适应能力不仅是整个机体所固有的，而且是机体的单个器官、甚至功能所固有的，这种能力在同一个问题应该解决多次的情况下是不可替代的。可见，适应能力这种现象，在整个生物界的合乎目的的行为中起着极其重要的作用。

控制机器人的问题在于模拟动物运动和人的适应能力。建立机器人控制的等级——首先是在机器人的各个等级水平上和子系统之间实行知觉功能、信息处理

功能和控制功能的分配。第三代机器人具有大规模处理能力，在这种情况下信息的处理和控制的完全统一算法，实际上是低效的，甚至是不中用的。所以，等级自适应结构的出现首先是为了提高机器人控制的质量，也就是降低不定性水平，增加动作的快速性。为了发挥各个等级和子系统的作用，必须使信息量大大减少。因此算法的各司其职使人们可以在不定性大大减少的情况下来完成任务。总之，智能的发达是第三代机器人的一个重要特征。人们根据机器人的智力水平决定其所属的机器人代别。有的人甚至依此将机器人分为以下几类：受控机器人——“零代”机器人，不具备任何智力性能，是由人来掌握操纵的机械手；可以训练的机器人——第一代机器人，拥有存储器，由人操作，动作的计划和程序由人指定，它只是记住 （接受训练的能力）和再现出来；感觉机器人——机器人记住人安排的计划后，再依据外界这样或那样的数据 （反馈）算出动作的具体程序；智能机器人——人指定目标后，机器人独自编制操作计划，依据实际情况确定动作程序，然后把动作变为操作机构的运动。因此，它有广泛的感觉系统、智能、模拟装置（周围情况及自身——机器人的意识和自我意识）。

这也是人工智能的最高级形态。

三、前景展望

在对 350 多个人工智能进行的一项大型调查中表明，人工智能将在短时间内全面超过人类。根据该调查，研究人员预测到，到 2027 年，人工智能的驾驶技术将全面超过人类，到 2049 年，它就能写出一本畅销小说，到 2053 年，它写的小说就能超过人类。事实上，根据这项调查，研究人员得出的结论是，到 2060 年，有 50%的可能性，人工智能将能够做我们能做的每一件事，甚至能做得更好！

参考文献

1. 部分内容参考自“百度百科” http://baike.baidu.com
2. “人工智能的 2017——AI 几乎在所有的游戏领域都战胜了人类玩家” https://baijiahao.baidu.com/s?id=1588090853019844023&wfr=spider&for=p
3. CNN 的 python 实现代码参考自书籍《深度学习入门——基于 Python 的理论与实现》:

附录：CNN 代码实现

```
# coding: utf-8
import pickle
import numpy as np
from collections import OrderedDict
class SimpleConvNet:
"""简单的 ConvNet
conv - relu - pool - affine - relu - affine - softmax
参数:
----------
input_size : 输入大小（MNIST 的情况下为 784）
hidden_size_list : 隐藏层的神经元数量的列表（e.g. [100, 100, 100]）
output_size : 输出大小（MNIST 的情况下为 10）
activation : 'relu' or 'sigmoid'
weight_init_std : 指定权重的标准差（e.g. 0.01）
指定'relu'或'he'的情况下设定“He 的初始值”
指定'sigmoid'或'xavier'的情况下设定“Xavier 的初始值”
"""
def __init__(self, input_dim=(1, 28, 28),
      conv_param={'filter_num':30, 'filter_size':5, 'pad':0, 'stride':1},
      hidden_size=100, output_size=10, weight_init_std=0.01):
  filter_num = conv_param['filter_num']
  filter_size = conv_param['filter_size']
```

```
23    filter_pad = conv_param['pad']
24    filter_stride = conv_param['stride']
25    input_size = input_dim[1]
26    conv_output_size = (input_size - filter_size + 2*filter_pad) / filter_stride+1
27    pool_output_size = int(filter_num * (conv_output_size/2) *
    (conv_output_size/2))
28  # 初始化权重
29  self.params = {}
30  self.params['W1'] = weight_init_std * \
31  np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
32  self.params['b1'] = np.zeros(filter_num)
33  self.params['W2'] = weight_init_std * \
34  np.random.randn(pool_output_size, hidden_size)
35  self.params['b2'] = np.zeros(hidden_size)
36  self.params['W3'] = weight_init_std * \
37  np.random.randn(hidden_size, output_size)
38  self.params['b3'] = np.zeros(output_size)
39  # 生成层
40  self.layers = OrderedDict()
41  self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],
42  conv_param['stride'], conv_param['pad'])
43  self.layers['Relu1'] = Relu()
44  self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
45  self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2'])
46  self.layers['Relu2'] = Relu()
47  self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3'])
48  self.last_layer = SoftmaxWithLoss()
49  def predict(self, x):
50    for layer in self.layers.values():
51      x = layer.forward(x)
52    return x
53  def loss(self, x, t):
54  """求损失函数
55  参数 x 是输入数据、t 是教师标签
56  """
```

```
57    y = self.predict(x)
58    return self.last_layer.forward(y, t)
59  def accuracy(self, x, t, batch_size=100):
60    if t.ndim != 1 :
61      t = np.argmax(t, axis=1)
62      acc = 0.0
63    for i in range(int(x.shape[0] / batch_size)):
64      tx = x[i*batch_size:(i+1)*batch_size]
65      tt = t[i*batch_size:(i+1)*batch_size]
66    y = self.predict(tx)
67    y = np.argmax(y, axis=1)
68    acc += np.sum(y == tt)
69    return acc / x.shape[0]
70  def numerical_gradient(self, x, t):
71  """求梯度（数值微分）
72  参数
73  ----------
74  x : 输入数据
75  t : 教师标签
76
77  返回
78  -------
79  具有各层的梯度的字典变量
80  grads['W1']、grads['W2']、...是各层的权重
81  grads['b1']、grads['b2']、...是各层的偏置
82  """
83      h = 1e-4 # 0.0001
84      grad = np.zeros_like(x)
85
86      it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
87      while not it.finished:
88          idx = it.multi_index
89          tmp_val = x[idx]
90          x[idx] = float(tmp_val) + h
91          fxh1 = f(x) # f(x+h)
```

```
92          x[idx] = tmp_val - h
93          fxh2 = f(x) # f(x-h)
94          grad[idx] = (fxh1 - fxh2) / (2*h)
95
96          x[idx] = tmp_val # 还原值
97          it.iternext()
98      return grad
99      loss_w = lambda w: self.loss(x, t)
100     grads = {}
101     for idx in (1, 2, 3):
102     grads['W' + str(idx)] = numerical_gradient(loss_w, self.params['W' +
    str(idx)])
103     grads['b' + str(idx)] = numerical_gradient(loss_w, self.params['b' +
    str(idx)])
104     return grads
    def gradient(self, x, t):
105     """求梯度（误差反向传播法）
106     参数
107     ----------
108     x : 输入数据
109     t : 教师标签
110
111     返回
112     -------
113     具有各层的梯度的字典变量
114     grads['W1']、grads['W2']、...是各层的权重
115     grads['b1']、grads['b2']、...是各层的偏置
116     """
117     # forward
118     self.loss(x, t)
119     # backward
120     dout = 1
121     dout = self.last_layer.backward(dout)
122     layers = list(self.layers.values())
123     layers.reverse()
```

```
124     for layer in layers:
125       dout = layer.backward(dout)
126     # 设定
127     grads = {}
128     grads['W1'], grads['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
129     grads['W2'], grads['b2'] = self.layers['Affine1'].dW,
    self.layers['Affine1'].db
130     grads['W3'], grads['b3'] = self.layers['Affine2'].dW,
    self.layers['Affine2'].db
131     return grads
    def save_params(self, file_name="params.pkl"):
132     params = {}
133       for key, val in self.params.items():
134       params[key] = val
135     with open(file_name, 'wb') as f:
136       pickle.dump(params, f)
    def load_params(self, file_name="params.pkl"):
137     with open(file_name, 'rb') as f:
138       params = pickle.load(f)
139     for key, val in params.items():
140       self.params[key] = val
141     for i, key in enumerate(['Conv1', 'Affine1', 'Affine2']):
142       self.layers[key].W = self.params['W' + str(i+1)]
143       self.layers[key].b = self.params['b' + str(i+1)]
```

注：

这里代码实现采用了 CN 中的卷积层和池化层的组合，进行手写数字识别 CNN。参数的梯度通过误差反向传播法求出，通过把正向传播和反向传播组装在一起来完成，最后把各层权重参数的梯度保存到 grads 字典中。

此实现代码，如果使用 MNIST 数据集训练，则训练数据的平均识别率为 99.82%，测试数据的识别率平均为 99.86%。