第3章

协作复合神经网络模型的基础架构

前面两章介绍了利用空间信息的大数据分析预测，接下来将介绍利用相关性特征的大数据分析预测过程。本章将主要介绍利用相关性特征进行大数据分析预测时所使用的基本模型——协作复合神经网络模型(Collaborative Compound Neural Network Model，CCNNM)，分别从协作复合神经网络模型概述、自适应动态灰狼优化算法、小波神经网络模型、协作复合神经网络模型的构建以及知识扩展五部分进行介绍。

3.1 协作复合神经网络模型概述

人工神经网络(Artificial Neural Network，ANN)最早于1943年被心理学家W. S. McCulloch和数理逻辑学家W. Pitts提出，当时他们建立了神经网络和数学模型，并称之为M-P模型，然后通过该模型提出了神经元的形式化数学描述和网络结构方法，证明了单个神经执行逻辑功能过程的存在，由此拉开了研究者对人工神经网络进行研究的序幕。图3.1为M-P模型示意图。

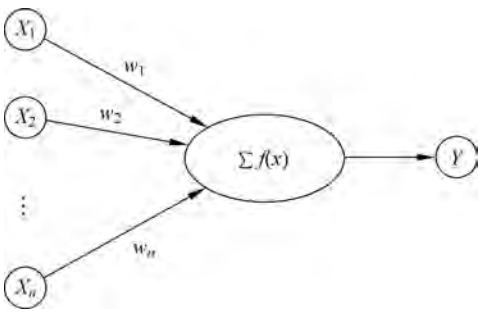

图3.1 M-P模型示意图

人工神经网络从信息处理的方向出发，仿照人脑神经元网络对信息的处理机制和对信息处理的过程进行抽象化，之后通过不同的神经元之间的连接方式组合成不同的神经网络，进而建立起某种神经网络模型。由这一过程可知，神经网络模型的主要特征体现在两方面：一方面是大量的神经元(或称节点)；另一方面则是不同的神经元之间的连接方式。人工神经网络的每个节点通常代表

某种特定输出函数，研究者一般将这种输出称为激励函数（Activation Function），而每两个节点之间都通过权重进行信号连接，当对神经网络设置不同的权重、激励函数时，神经网络的输出也会不同。人工神经网络对数据的处理过程主要为将数据输入神经网络后，通过对不同层级的不同节点、激励函数进行计算，最后由输出层进行输出，这一过程可以视作对某种算法或某种函数的逼近过程，也可以视作对某种逻辑策略的表达。

按照拓扑结构可以将神经网络分为前向神经网络以及反馈神经网络。前向神经网络是指数据输入网络后经由神经元传入下一级，每个神经元仅接收由前一级传入的输入，不存在反馈过程，整个网络可以用一个有向无环路图表示。前向神经网络主要进行数据由输入传递至输出的这一变换过程，且这一过程主要由多种非线性函数的复合来完成，因此它通常结构简单、易于实现。常见的前向神经网络有自适应线性神经网络、单层感知机、多层感知机以及 BP 神经网络。BP 神经网络结构如图 3.2 所示。

反馈神经网络与前向神经网络完全相反，当数据传入网络后，神经元不仅仅是只接收前一级传入的输入，而是整个网络的传递过程中存在神经元与神经元之间的反馈过程，这一过程可以用一个无向的完备图来表示。反馈神经网络是一种反馈动力学系统。在这种网络中，每个神经元同时将自身的输出信号作为输入信号反馈给其他神经元，它需要工作一段时间才能达到稳定。常见的反馈神经网络有 Hopfield 网络、波耳兹曼机。Hopfield 网络结构如图 3.3 所示。

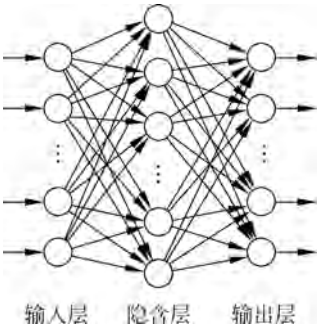

图 3.2　BP 神经网络结构

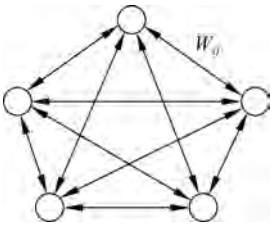

图 3.3　Hopfield 网络结构

目前使用人工神经网络进行大数据预测时通常使用前向神经网络进行，因此本章将围绕建立前向神经网络进行数据预测展开，同时选用前向神经网络的小波神经网络作为研究对象。小波神经网络用于大数据预测时，其性能主要受到隐含层、输出层参数的影响，其训练过程主要也是对参数的优化过程，一般其参数初始值主要通过随机初始化的方式获得，因此参数初始值的好坏直接影响模型训练后的性能，好的参数初始值能够有效缩短网络的训练时间、提高收敛精度以及避免陷入局部最优。为了使所选用的小波神经网络获得更好的预测效果，本书选用群智能优化算法中的灰狼优化算法对其进行参数初始值的优化，同时为了克服灰狼优化算法自身存在的问题，对其进行改进并提出自适应动态灰狼优化算法。最后通过将自适应动态灰狼优化算法与小波神经网络相结合，组成协作复合神经网络模型并实现大数据的预测过程。

3.2 自适应动态灰狼优化算法

群智能优化算法是一种通过模仿自然界中种群生物的捕食、迁移等群体行为，从而在目标范围内搜寻目标问题的最优解的优化算法，通过将其与神经网络结合，可以有效帮助神经网络获得较优的参数初始值，进而获得更好的模型训练结果。

灰狼优化(Grey Wolf Optimizer，GWO)算法是由 Seyedali Mirjalili 等人于 2014 年提出的一种群智能优化算法，这一算法主要由自然界中的灰狼群体的捕食行为启发而来。灰狼是一种群居动物，一般群体中由 5～12 个个体构成。与一般动物群体不同的是，这一群体中存在十分严格的社会主导阶层，如图 3.4 所示。这一社会层级结构与金字塔结构十分相似，主要由四个层级构成。

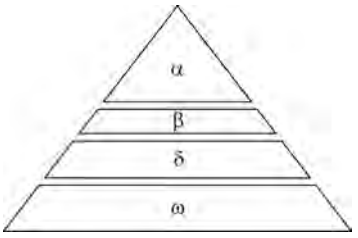

图 3.4 灰狼群体社会层级结构

首先最高的层级为一头公狼与一头母狼，可以被称为 α，它们主要对种群中的各种规则进行制定，如狩猎地点、休息地点等，整个种群都会听从它们的决定，且这种服从是具有绝对性的。特殊的是，α 可以不是种群中体能最强的个体，但它一定是管理能力最好的个体。

然后第二个层级被称为 β，这一层级的灰狼个体主要帮助 α 制定相关决策，同时将各种决策上出现的问题反馈给 α，它在整个种群中的地位仅次于 α 狼，因此低等级的灰狼个体也必须听命于 β。另外，β 同时也扮演着 α 的继承者的角色，当 α 衰老或死亡时，将由 β 来承担 α 的职责。

紧接着的一个阶级为 δ，这一个层级的灰狼个体在过去曾经是 α 或者 β，但它们现在扮演着执行者的角色并将 α 与 β 制定的规则与命令付诸行动。它们可以是哨兵、侦查者、猎人，甚至是种群中受伤狼群的看护者。

最后一个层级为 ω，这个层级的灰狼个体为最弱势的个体，它们一般为种群中年迈或残疾的个体，因此它们只能服从前面每个层级的灰狼个体。

除了上述灰狼的社会层级结构外，灰狼群体的捕食行为也存在一定的规律，根据 Muro 等人的研究，灰狼进行狩猎的过程如下：

(1) 跟踪、追逐并靠近猎物；

(2) 慢慢骚扰、影响猎物的行动，直到猎物停止行动；

(3) 以包围圈的形式攻击猎物。

根据上述灰狼种群的社会层级结构以及其捕食行为的相关规律，灰狼优化算法的整个算法过程得以设计出来。

3.2.1 灰狼优化算法原理

灰狼优化算法的原理主要依托于四个基本行为，它们分别为社会等级结构分级、包围猎物、攻击猎物以及搜索猎物。下面将先分别从这四个基本行为进行介绍。

1. 社会等级结构分级

为了更好地使设计的算法符合灰狼群体的社会等级结构，将候选解决方案的优劣性作为评判的标准，另外由于解决方案的独特性，因此将前三个等级 α、β 以及 δ 的数量设定为一个，即将候选解决方案中表现最优的方案设为 α，第二个与第三个较优解决方案分别设定为 β 与 δ，其余的解决方案则均为 ω。按照等级较低的灰狼个体跟随等级较高的灰狼个体规则，ω 解决方案将不断学习 α、β 以及 δ 解决方案以获得更好的表现。

2. 包围猎物

针对灰狼群体包围猎物的特性，使用下列公式对其行为进行描述：

$$D = | C * X_p(t) - X(t) | \tag{3.1}$$

$$X(t+1) = X_p(t) - A * D \tag{3.2}$$

其中，t 为当前迭代次数；A 和 C 为相关系数；X_p 为猎物的位置信息；X 为灰狼个体的位置信息。

A 和 C 将分别通过下面两个公式计算得出：

$$A = 2a * r_1 - a \tag{3.3}$$

$$C = 2r_2 \tag{3.4}$$

其中，a 将随着迭代的次数由 2 到 0 线性递减，r_1 与 r_2 均为 0～1 的随机数。

3. 攻击猎物

通过包围行为，所有灰狼个体将猎物控制在一个包围圈内，之后 ω 狼将在 α、β 以及 δ 狼的引导下进行捕猎，由于目前猎物的位置是未知的，而代表最优解决方案的灰狼 α、β 以及 δ 的位置信息是已知的，因此 ω 狼将通过学习灰狼 α、β 以及 δ 的位置信息来进行移动以完成对猎物的捕食。下面几个公式代表了灰狼个体的捕食行为：

$$D_\alpha = | C_1 * X_\alpha - X | \tag{3.5}$$

$$D_\beta = | C_2 * X_\beta - X | \tag{3.6}$$

$$D_\delta = | C_3 * X_\delta - X | \tag{3.7}$$

$$X_1 = X_\alpha - A_1 * D_\alpha \tag{3.8}$$

$$X_2 = X_\beta - A_2 * D_\beta \tag{3.9}$$

$$X_3 = X_\delta - A_3 * D_\delta \tag{3.10}$$

$$X(t+1) = \frac{X_1 + X_2 + X_3}{3} \tag{3.11}$$

由上述公式可以了解到，猎物的位置是随机的，灰狼个体将通过学习灰狼 α、β 以及 δ 的位置信息在猎物附近进行随机移动，以此来估计猎物的具体位置。

另外可以了解到，通过减小 a 的值，使得灰狼种群能够模拟一个不断接近猎物的过程，在这个过程中 A 是随着 a 的减小在不断变化的，它的迭代范围为$[-a, a]$，当 A 的绝对值小于 1 时，灰狼移动后的位置将会向包围圈中间收缩以攻击猎物。

4. 搜索猎物

在灰狼种群开始对猎物的随机包围时，对猎物的搜索过程也随之展开。由攻击猎物的原理过程可以了解到，A 这一系数的大小将会直接影响灰狼个体位置的移动，在整个迭代过程中，除了 A 的绝对值小于 1 外，还存在 A 的绝对值大于 1 的情况，在这种情况下，灰狼个体将向包围圈周围扩张，以此发现更多猎物可能存在的位置。即当 $|A|\geqslant 1$ 时，候选解决方案倾向于偏离当前猎物；当 $|A|<1$ 时，候选解决方案逐渐收敛于猎物的位置。

除系数 A 之外，还存在一个系数 C，C 通常是 0 到 2 之间的一个随机值，这一系数的角色类似于为猎物位置信息新添一个随机权重。在自然界中，灰狼种群对猎物的捕食通常不会是顺利的，有时会出现一定的障碍对整个搜索行为产生影响，使得灰狼种群无法直接快速地接近猎物，系数 C 则可以在为整个搜索过程增加一个随机性的同时使整个灰狼种群在优化过程中表现出更随机的行为，以此来探索更多区域并避免陷入局部最优。

结合上述四种行为的基本原理可以了解使用灰狼优化算法进行优化问题的求解时主要具有以下七种特性：

(1) 灰狼种群的社会等级结构可以使算法在迭代过程中保存每次迭代时种群的最优解；

(2) 灰狼种群对猎物的包围机制不仅适用于二维，还可以扩展到多维；

(3) 系数 A 和 C 可以帮助个体获得不同的候选解；

(4) 种群对猎物的捕食过程可以利用候选解来定位猎物的位置；

(5) 自适应变化的 a 和 A 有利于种群个体在位置变化的过程中进行向外的探索过程，同时还可以帮助算法平衡外部信息的探索与内部信息的利用这两个过程；

(6) 在 A 的变化过程中，当 $|A|\geqslant 1$ 时处于对外部信息的探索阶段，当 $|A|<1$ 时处于对内部信息的利用阶段；

(7) 灰狼优化算法仅有两个主要的参数需要进行调整，它们分别为 a 和 C。

使用灰狼优化算法对优化问题进行求解时的具体步骤可以归纳如下：

(1) 以种群个体的位置信息作为待优化问题的解，根据待优化问题的解的范围，随机初始化种群所有个体的位置信息；

(2) 初始化参数 a、A 和 C；

(3) 根据待优化问题，计算每个种群个体的适应度值，并对其进行排序，适应度值越高，则个体的位置信息越接近最优解，将适应度值排在前三的个体分别设定为灰狼 α、β 以及 δ，并保存当前最优的位置信息；

(4) 依次对种群中每个个体的位置信息进行更新，具体的更新公式如公式(3.5)～公式(3.11)所示；

(5) 针对每个个体更新后的位置信息，重新进行适应度值的计算，根据新的适应度值的大小更新灰狼 α、β 与 δ 的位置信息以及历史最优的位置信息，更新参数 a、A 和 C；

(6) 根据迭代的次数重复步骤(3)～步骤(5)，当达到最大迭代次数时停止迭代过程，输出历史最优的位置信息，该位置信息即为算法优化后获得的最优解。

灰狼优化算法如算法 3.1 所示。

算法 3.1　灰狼优化算法

```
初始化灰狼种群个体
初始化 a、A 和 C
计算搜索范围内每个个体的适应度值
将 X_α、X_β 与 X_δ 分别赋值种群中适应度值排名前三的个体位置信息
while(未满足停止迭代条件)do
    for 每个个体
        更新位置信息
    end for
    更新 a、A 和 C
    计算个体新的适应度值
    更新 X_α、X_β 与 X_δ
end while
输出 X_α
```

3.2.2　非线性余弦收敛因子

在灰狼优化算法中，参数 A 随着迭代次数的增加而线性减少，这一变化过程影响着灰狼种群中每个个体的移动距离和移动速度，该算法也以此来平衡种群个体的外部信息探索与内部信息利用这两个过程。从公式(3.3)可以看到，参数 A 的大小由 a 来决定，a 随着迭代次数的增加从 2 到 0 不断减小，当迭代次数未到达一半时，$a \geqslant 1$ 使得 $|A| \geqslant 1$，此时灰狼个体在移动时具有较快的移动速度以及较大的移动距离，这使得灰狼个体远离当前的猎物并试图在目标范围内搜索更多可能存在的猎物。当迭代次数达到一半时，$a < 1$ 使得 $|A| < 1$，此时个体的移动速度较之前开始降低，移动距离也相对有所减小，因此此时个体慢慢朝向包围的猎物靠近并攻击猎物。

然而，上述状态属于一种理想状态，在实际的优化过程中，个体位置的移动有时不一定朝向目标范围内的最优解方向，种群个体中仍然存在寻优速度较慢或陷入局部最优的情况。另外，在整个迭代过程中，a 为线性下降，因此该值保持一个相同的减小速度，这可能导致种群个体位置的变化速度在每次迭代过程中相对较为接近从而增加上述状况出现的可能性。为了解决上述问题，提出一种非线性余弦收敛因子来代替之前的线性变化的参数 a，下面分别给出了该非线性余弦收敛因子的计算公式以及其随迭代次数的变化过程图。

$$a = \cos\left(\pi * \frac{i}{i_{\max}}\right) + 1 \tag{3.12}$$

其中，i 为当前迭代的次数；$i_{\max}$ 为总的迭代次数。

图 3.5 为参数 a 随迭代次数的变化过程，可以看到，在迭代前期，a 的减小速度由 0 开始慢慢增加，此时参数 A 可以获得一个较大值，且该较大值可以维持在一个较长的迭代周期内，由参数 A 的特性可以得知这一情况能够使得种群个体更多地向包围圈周围探索以发现更多可能存在的较优解。在迭代后期 a 的减小速度由中期的一个较大值逐渐

变小，此时对应的种群个体向较优解的位置进行收敛的速度也在逐渐变慢，如此设置的原因主要是在这个阶段个体的位置在移动过程中容易陷入局部最优，通过减缓收敛速度可以帮助其增加对移动区域进行搜索的概率，以此来避免陷入局部最优。

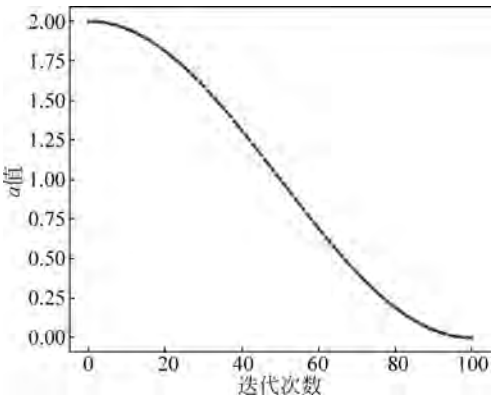

图 3.5　参数 a 随迭代次数的变化过程

3.2.3　加权位置更新

在使用灰狼优化算法进行目标的搜索过程中，种群中的所有灰狼个体将以灰狼 α、β 以及 δ 的位置信息作为移动的方向，以朝向这三者移动距离的平均值作为具体的移动距离，但是由于这三者的适应度之间存在一定的差值，因此它们距离目标猎物的距离也会随之存在一定的差距，若适应度差值较大时，该差距也会处于一个较大的状态，此时若按照公式(3.11)对个体位置进行更新，则灰狼个体向目标猎物移动的速度也会随之降低，这将直接影响最后的优化结果。

为了使种群中每个灰狼个体向灰狼 α、β 以及 δ 的位置方向的移动更加合理，在此将基于适应度值比的加权位置更新公式引入灰狼优化算法。这个更新方式主要使用进行位置移动的灰狼个体的适应度值与灰狼 α、β 以及 δ 的适应度值之间的比值作为位置更新的主要参考依据，当灰狼 α、β 以及 δ 的适应度值越大时，灰狼个体向它们移动的距离也会越大，反之亦然。此更新方式的具体更新公式如下：

$$r_\alpha = \frac{f}{f_\alpha + \varepsilon} \tag{3.13}$$

$$r_1 = \frac{r_\alpha}{r_\alpha + r_\beta + r_\delta} \tag{3.14}$$

$$X(t+1) = \frac{r_1 X_1 + r_2 X_2 + r_3 X_3}{r_1 + r_2 + r_3} \tag{3.15}$$

其中，f 为需要进行位置移动的灰狼个体的适应度值；f_α 为灰狼 α 的适应度值；ε 是为了避免分母为 0 而设置的一个极小值；r_β 与 r_δ 的计算方式与 r_α 相同，均为基于灰狼 β 与 δ 的适应度值进行计算；r_1、r_2 以及 r_3 分别为灰狼个体向灰狼 α、β 以及 δ 的位置方向移动的比例并且 r_2 与 r_3 的计算方式与 r_1 相同。

在上述移动过程中，每个灰狼个体向灰狼 α、β 以及 δ 移动的距离与其适应度值占灰狼 α、β 以及 δ 适应度值的比例直接相关。当待优化的问题为求解目标范围内的最小值时，个体的适应度值越小，个体的位置信息越好，则此时将采用公式(3.11)进行适应度值比的计算，即适应度值越小，适应度值比越大，向目标灰狼个体移动的距离也会越大。当待优化的问题为求解目标范围内的最大值时，个体的适应度值越大对应个体的位置信息越好，则此时公式(3.11)中的分子与分母需要进行对调，即较大的适应度值对应较大的适应度值比，个体向目标灰狼个体移动的距离也较大。

3.2.4　中心扰动准则

在灰狼优化算法的整个迭代过程中，个体会出现在位置更新之后适应度值较更新之前保持不变或降低的情况，这属于正常现象，但是一旦这种情况连续多次出现时，可能是因为个体陷入了局部最优，这将直接影响算法的最终性能。为了避免这种情况对算法性能的影响，在此使用中心扰动准则来对个体每次更新后的位置进行干扰。

中心扰动准则主要使用 T 值来统计迭代过程中每个个体出现位置更新后适应度值连续保持不变或降低的次数，在每次迭代之前将会为每个个体初始化一个 T 值，该值的初始值均为 0，当开始出现个体位置更新后适应度值不变或降低的情况时，将用下面的公式对 T 值进行更新：

$$T = T + 1 \tag{3.16}$$

当个体位置更新后再次出现适应度值增加的情况时，T 值将被重新初始化为 0。

除了使用 T 值来统计迭代过程中每个个体出现位置更新后适应度值连续保持不变或降低的次数外，T 值还用于计算是否对更新后的个体位置信息进行中心扰动的概率，下面给出这一概率的计算公式：

$$p = 1 - \exp\left(-\frac{T}{10}\right) \tag{3.17}$$

图 3.6 给出了概率 p 随 T 值的变化情况，在 T 值初始化为 0 时，对应的概率 p 也为 0，即此时未出现个体位置更新后适应度值连续保持不变或降低的情况，个体并未陷入局部最优，不需要对其位置信息进行中心扰动。当 T 值逐渐增大时，代表着个体位置更新后适应度值连续保持不变或降低的次数在不断增多，此时个体陷入局部最优的概率也在随之增大，因此将会有较大的概率对个体的位置信息进行中心扰动。

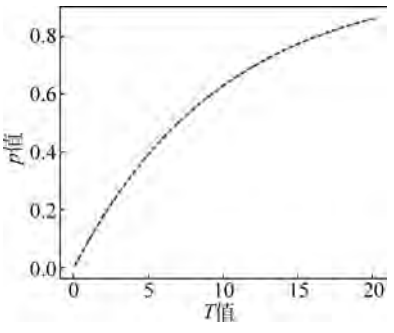

图 3.6　概率 p 随 T 值的变化过程

中心扰动准则主要使用种群的中心点位置对个体的位置信息进行扰动，由前面对灰狼优化算法的原理介绍可以得知，灰狼种群的所有个体主要以一个包围圈的形式围住猎物，因此中心点的位置往往更接近猎物的位置，个体朝着这个位置移动可以帮助其获得更好的位置信息。下面给出中心扰动准则的位置扰动公式：

$$D_{\text{mean}} = | C * X_{\text{mean}}(t) - X(t) | \tag{3.18}$$

$$X(t+1)=|X_{\text{mean}}(t)-A*D_{\text{mean}}| \tag{3.19}$$

其中，$X_{\text{mean}}(t)$代表当前迭代次数中种群中心点的位置信息。由于种群中的所有个体主要通过学习灰狼 α、β 以及 δ 的位置信息进行移动，因此灰狼 α、β 以及 δ 的位置信息的平均值将作为种群中心点的位置信息。

上述种群中心点的位置信息是由灰狼 α、β 以及 δ 的位置信息平均获得，因此这一位置信息相较于其他现有位置信息更接近全局最优解，当种群个体陷入局部最优时，通过对这一位置信息的学习可以帮助其迅速调整移动的方向与距离以跳出局部最优，同时在整个迭代过程中也更容易发现全局最优值。

中心扰动准则如算法 3.2 所示。

算法 3.2　中心扰动准则

```
for 每个灰狼个体 do
    计算适应度值
    if 适应度值较位置变化前不变或减小 then
        使用公式(3.14)与公式(3.15)分别更新 T 值与 p 值
        在 0 到 1 之间随机生成 P 值
        if p>P then
            使用公式(3.16)与公式(3.17)对个体位置信息进行扰动
        else
            个体位置信息不变
        end if
    else
        T=0
    end if
end for
```

3.2.5　自适应动态灰狼优化算法运行机制

将上述非线性余弦收敛因子、加权位置更新方式以及中心扰动准则与灰狼优化算法相结合，即为自适应动态灰狼优化(Adaptive Dynamic Grey Wolf Optimizer，ADGWO)算法。由前面对这三种改进的介绍可以得知，除灰狼优化算法的七个特性以外，自适应动态灰狼优化算法还具备下列六个特性。

(1) 参数 a 采用了随迭代次数的余弦递减变化趋势来代替传统的线性递减方式，有利于平衡种群个体的外部信息探索与内部信息利用这两个过程；

(2) 加权的位置更新方式从灰狼个体自身的适应度值出发，以个体适应度值与目标个体适应度值的比值作为自变量来决定个体位置更新的大小；

(3) 加权的位置更新方式更符合实际情况，即在位置的更新过程中应该向适应度值较低的个体学习较少的位置信息，向适应度值更高的个体学习更多的位置信息；

(4) 中心扰动准则可以帮助个体避免陷入局部最优，同时快速向全局最优位置移动；

(5) 采用灰狼 α、β 以及 δ 位置信息的平均值作为种群中心点的位置信息有助于迭代

过程中对最优解的搜索；

(6) 除参数 a 和 C 需要进行调整外，新增了参数 T 和 p，但这两个参数无须提前设置，通过在算法迭代过程中计算获得。

将灰狼优化算法的特性与上述六个特性相结合，可以将自适应动态灰狼优化算法的算法步骤归纳如下。

(1) 以种群个体的位置信息作为待优化问题的解，根据待优化问题的解的范围，随机初始化种群所有个体的位置信息；

(2) 根据待优化问题，计算每个种群个体的适应度值，并对其进行排序，适应度值越高，则个体的位置信息越接近最优解，将适应度值排在前三的个体分别设定为灰狼 α、β 以及 δ，并对当前最优的位置信息进行保存；

(3) 对参数 a，A 和 C 进行初始化；

(4) 依次对种群中每个个体的位置信息进行更新，具体的更新公式如公式(3.5)～公式(3.10)、公式(3.14)～公式(3.15)所示；

(5) 针对每个个体更新后的位置信息，重新进行适应度值的计算，根据适应度值大小的比较对 T 值与 p 值进行更新；

(6) 根据 p 值确定是否对更新后的位置进行中心扰动，若进行则采用公式(3.18)与公式(3.19)对个体位置信息进行更新，并计算更新后的适应度值；

(7) 通过适应度值的大小的比较更新灰狼 α、β 与 δ 的位置信息以及历史最优的位置信息，更新参数 a、A 和 C；

(8) 按照迭代的次数重复步骤(4)～步骤(7)，当达到最大迭代次数时停止迭代过程，输出历史最优的位置信息，该位置信息即为算法优化后获得的最优解。

自适应动态灰狼优化算法如算法 3.3 所示。

算法 3.3　自适应动态灰狼优化算法

```
初始化灰狼种群个体
初始化 a、A 和 C
计算搜索范围内每个个体的适应度值
将 X_α、X_β 与 X_δ 分别赋值种群中适应度值排名前三的个体位置信息
while(未满足停止迭代条件)do
    for 每个个体
        更新位置信息
        更新个体适应度值
        比较适应度值,更新 T 值与 p 值
        使用中心扰动准则对个体位置信息进行扰动
    end for
    更新 a、A 和 C
    计算个体新的适应度值
    更新 X_α、X_β 与 X_δ
end while
输出 X_α
```

3.2.6 对比实验

在算法的研究过程中，研究者一般使用基准函数对算法的有效性进行验证，因此为了验证自适应动态灰狼优化算法的有效性，选用十种不同的基准函数对其进行测试。在这十种测试函数中，前五种为单峰函数，即在所考虑的范围区间内只有一个严格局部极值（峰值）；后五种为多峰函数，即在所考虑的范围区间内有多个局部极值（峰值），它们在目标范围内搜索最小值，且目标最小值为 0。下面分别给出这十种基准函数的公式及范围区间。

(1) Schwefels P2.22，[−10，10]：

$$f_1(x)=\sum_{j=1}^{n}|x_i|+\prod_{i=1}^{n}|x_i| \tag{3.20}$$

(2) Rosenbrock's，[−10，10]：

$$f_2(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2] \tag{3.21}$$

(3) Step，[−100，100]：

$$f_3(x)=\sum_{i=1}^{n}(x_i+0.5)^2 \tag{3.22}$$

(4) Sum of Different Powers，[−1，1]：

$$f_4(x)=\sum_{i=1}^{n}|x_i|^{i+1} \tag{3.23}$$

(5) Zaharov，[−5，10]：

$$f_5(x)=\sum_{i=1}^{n}x_i^2+\left(\sum_{i=1}^{n}0.5ix_i\right)^2+\left(\sum_{i=1}^{n}0.5ix_i\right)^4 \tag{3.24}$$

(6) Schwefels，[−500，500]：

$$f_6(x)=418.9829n-\sum_{j=1}^{n}x_i\sin(\sqrt{|x_i|}) \tag{3.25}$$

(7) Rastrigin，[−5.12，5.12]：

$$f_7(x)=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10] \tag{3.26}$$

(8) Ackley，[−32，32]：

$$f_8(x)=20+\mathrm{e}-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) \tag{3.27}$$

(9) Dixon-Price，[−10，10]：

$$f_9(x)=(x_1-1)^2+\sum_{i=2}^{n}i(2x_i^2-x_{i-1})^2 \tag{3.28}$$

(10) Levy，[−10，10]：

$$f_{10}(x)=\sin^2(\pi\omega_1)+\sum_{i=1}^{n-1}(\omega_1-1)^2[1+10\sin^2(\pi\omega_i+1)]+$$
$$(\omega_n-1)^2[1+\sin^2(2\pi\omega_n)],$$

$$\omega_i = 1 + \frac{x_i - 1}{4}, \quad i = 1, \cdots, n \tag{3.29}$$

另外，将灰狼优化算法、遗传算法(Genetic Algorithm，GA)、粒子群优化(Particle Swarm Optimization，PSO)算法作为自适应动态灰狼优化算法的对比算法，进行性能比较。将测试函数的维度设定为 20，每种算法的种群数量和迭代次数分别设为 30 和 100。在具体的参数设置方面，遗传算法中，交叉算子与变异算子分别设为 0.6 与 0.05；粒子群优化算法中，将惯性因子设为 0.9，将两个加速系数均设置为 2；灰狼优化算法与自适应动态灰狼优化算法中主要需要进行设置的参数为 a，将该值的最大值设为 2。

首先比较的是在迭代过程中每种算法的种群最优适应度值的变化情况。由于进行实验的四种算法在每次迭代后均会保存当前迭代时种群个体的最优适应度值，因此将这四种算法在每种基准函数上运行时的最优适应度值进行保存并绘制变化情况，如图 3.7 所示，依

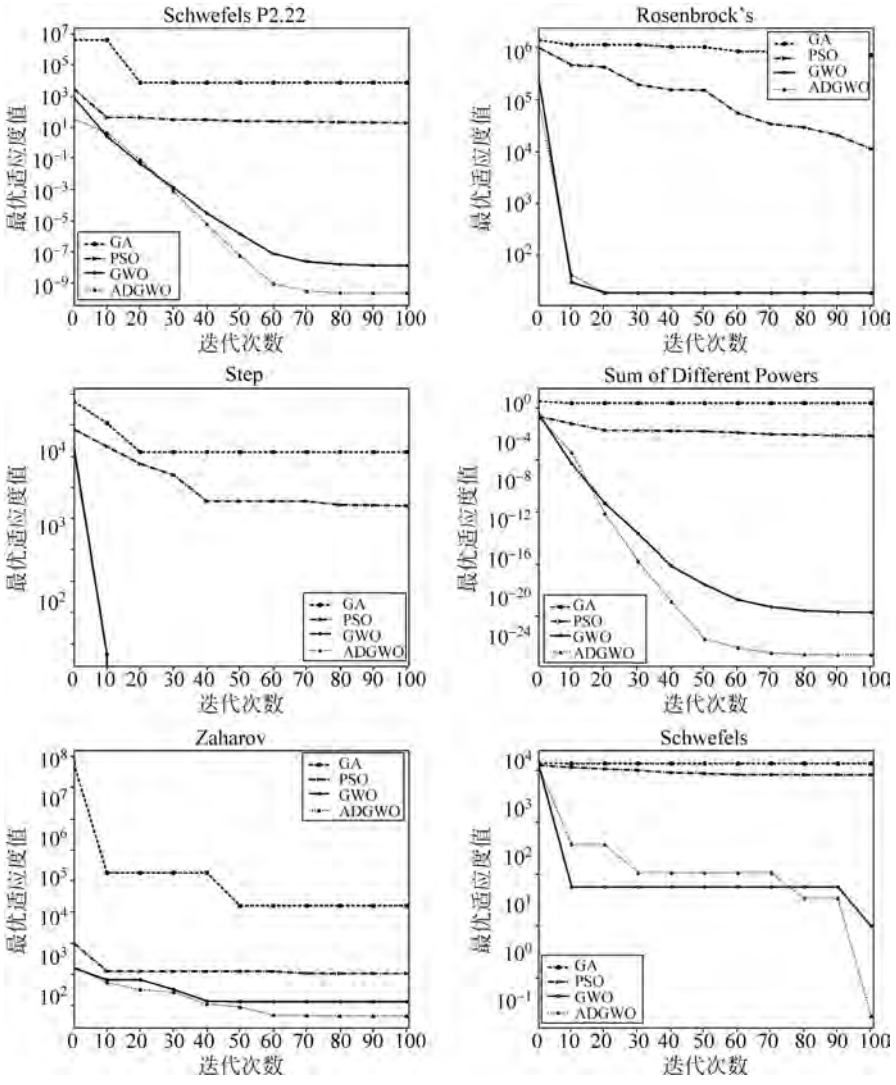

图 3.7　四种算法在十种基准函数上的最优适应度值的变化情况

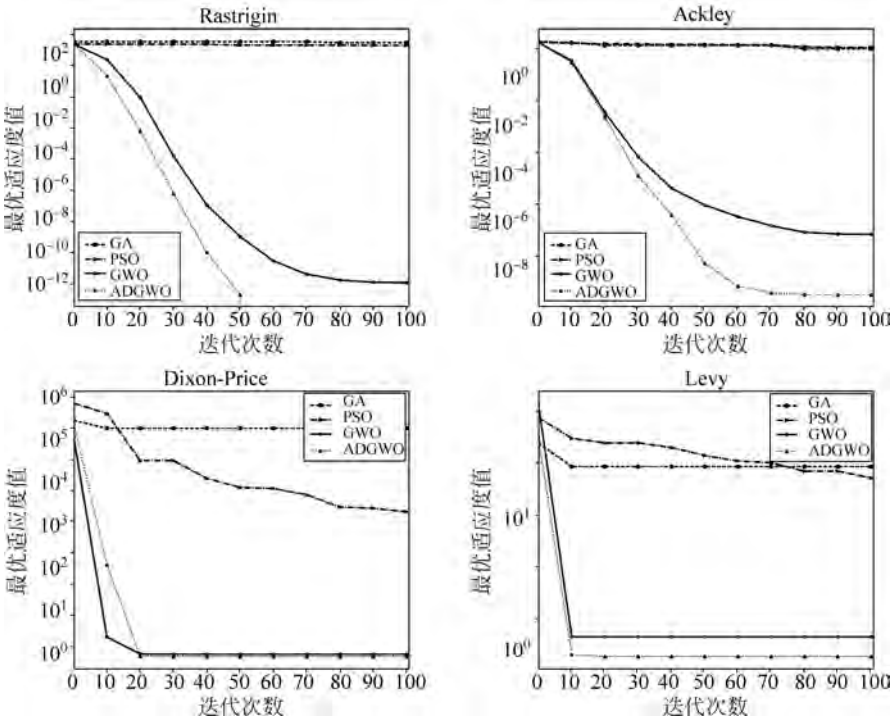

图 3.7 （续）

据适应度值的差别来比较这四种算法在十种基准函数上的收敛速度与收敛精度。

在图 3.7 中，不同的算法在不同的基准函数上的表现存在很大的差异性，但是无论在哪种基准函数下，灰狼优化算法与自适应动态灰狼优化算法的最优适应度值曲线均在遗传算法与粒子群优化算法的最优适应度值曲线之下。遗传算法与粒子群优化算法在前五种单峰函数上最优适应度值的下降幅度要明显大于后五种多峰函数，这主要是由于这两种算法的提出时间相对较早，因此当待优化问题的复杂程度提高时，其性能也会有所降低。另外，从整体上来看，粒子群优化算法最后均能够收敛到一个低于遗传算法的适应度值。对比灰狼优化算法与自适应动态灰狼优化算法的适应度值变化曲线可以看到，这两种算法在所有基准函数上均能够在迭代前期获得一个较低的适应度值，其中在部分基准函数上的最优适应度值较为接近，但在大部分基准函数上，自适应动态灰狼优化算法收敛到最优适应度值的速度要明显大于灰狼优化算法，且其迭代结束后的最优适应度值也要大于自适应动态灰狼优化算法，证明其收敛速度与收敛精度较灰狼优化算法均有所提高。

然后从迭代之后种群的所有候选解的情况对四种算法进行对比，分别统计迭代之后四种算法种群个体候选解的最优适应度值、适应度值的平均值以及适应度值的标准差，统计结果如表 3.1 所示。对比表 3.1 中不同算法的最优适应度值可以看到，四种算法的对比结果与图 3.7 中的最终比较情况相一致，而在比较适应度平均值和适应度标准差时可以看到遗传算法与粒子群优化算法的种群优化结果相较于灰狼算法与自适应动态灰狼算法更差一些，在迭代结束后这两种算法的最终的适应度平均值与适应度标准差要远大于

灰狼算法与自适应动态灰狼算法，即不同的候选解的优化结果差距较大，这可能是由于这两种算法的原理中所有个体的位置移动倾向于随机搜索。而灰狼优化算法与自适应动态灰狼优化算法在迭代结束后种群个体的优化结果要更加平均一些，这主要是因为在这两种算法中所有个体均朝向当前最优的位置方向移动，因此适应度平均值与适应度标准差均能够处于一个较低水平。最后对比灰狼算法与自适应动态灰狼算法的优化结果，这两种算法在十种基准函数上的表现均较为优秀，其中在基准函数 Step 上，灰狼优化算法达到了目标范围内的全局最优值，在基准函数 Step 与 Rastrigin 上自适应动态灰狼优化算法达到了目标范围内的全局最优值，但从整体上来看，无论是最优适应度值、适应度平均值还是适应度标准差，自适应动态灰狼优化算法的优化结果都要低于灰狼优化算法，说明迭代结束后，自适应动态灰狼优化算法的优化情况要优于灰狼优化算法。结合前面的四种算法在十种基准函数上的最优适应度值的变化情况可以得知，在这十种基准函数上，自适应动态灰狼优化算法具有更快的收敛速度与更高的收敛精度，同时最终的优化结果更好。

表 3.1　四种算法在十种基准函数上的优化结果对比

基准函数	算　法	最优适应度值	适应度平均值	适应度标准差
Schwefels P2.22	GA	7.67E+03	2.03E+15	2.11E+14
	PSO	1.92E+01	2.40E+16	7.71E+16
	GWO	1.22E−08	1.23E−08	1.16E−12
	ADGWO	2.14E−10	2.14E−10	5.80E−18
Rosenbrock's	GA	7.25E+05	7.49E+06	1.01E+06
	PSO	1.12E+04	3.66E+06	4.06E+06
	GWO	1.89E+01	1.89E+01	3.34E−06
	ADGWO	1.88E+01	1.88E+01	6.96E−15
Step	GA	8.70E+03	9.01E+04	7.25E+03
	PSO	1.71E+03	7.05E+04	5.06E+04
	GWO	0.00E+00	0.00E+00	0.00E+00
	ADGWO	0.00E+00	0.00E+00	0.00E+00
Sum of Different Powers	GA	6.28E−01	1.76E+01	6.45E−01
	PSO	2.91E−04	3.86E+00	4.55E+00
	GWO	3.68E−22	3.68E−22	1.06E−25
	ADGWO	1.73E−26	1.73E−26	4.19E−34
Zaharov	GA	1.58E+04	4.53E+10	7.59E+09
	PSO	2.88E+02	3.39E+08	1.70E+09
	GWO	5.50E+01	6.14E+01	2.11E+00
	ADGWO	2.33E+01	2.33E+01	1.35E−07
Schwefels	GA	6.90E+03	8.53E+03	2.25E+02
	PSO	4.06E+03	6.88E+03	1.34E+03
	GWO	3.47E+00	6.42E+02	4.70E+02
	ADGWO	5.20E−02	1.62E+00	1.09E+00

续表

基准函数	算法	最优适应度值	适应度平均值	适应度标准差
Rastrigin	GA	2.50E+02	4.07E+02	2.86E+01
	PSO	1.73E+02	3.66E+02	6.72E+01
	GWO	1.06E−12	1.06E−12	1.35E−15
	ADGWO	0.00E+00	0.00E+00	0.00E+00
Ackley	GA	1.01E+01	1.97E+01	4.90E−01
	PSO	1.11E+01	1.92E+01	3.02E+00
	GWO	1.83E−06	1.83E−06	2.11E−10
	ADGWO	1.09E−08	1.09E−08	0.00E+00
Dixon-Price	GA	1.45E+05	8.84E+05	7.84E+04
	PSO	1.76E+03	2.88E+04	2.34E+04
	GWO	9.48E−01	9.51E−01	6.26E−07
	ADGWO	8.45E−01	8.45E−01	9.04E−14
Levy	GA	2.25E+01	7.34E+01	1.01E+01
	PSO	1.86E+01	2.93E+02	2.37E+02
	GWO	1.29E+00	1.29E+00	6.36E−05
	ADGWO	9.26E−01	9.64E−01	9.48E−11

最后在算法的优化过程中对每种算法在每种基准函数上所需的运行时间进行统计，统计结果如表 3.2 所示。由表 3.2 可以看到，粒子群优化算法与遗传算法的运行时间整体上看是比较接近的，而灰狼优化算法的在每种基准函数上运行时间均低于粒子群优化算法和遗传算法，即从算法效率上来看灰狼优化算法要优于粒子群优化算法与遗传算法。对比灰狼优化算法与自适应动态灰狼优化算法的运行时间可以看到，自适应动态灰狼优化算法在十种基准函数上的运行时间均有所增加，这可能是因为与灰狼优化算法相比，自适应动态灰狼优化算法中增加了中心扰动准则，这一准则让个体在位置更新之后增加了适应度比较环节以及新的位置更新，而灰狼优化算法的所有步骤均保留了下来，因此，算法步骤的增加导致运行时间的增加。但相对于整体消耗的时间而言，增加的部分是合理的。因此，从运行效率上来看，自适应动态灰狼优化算法的运行效率是较好的。

表 3.2　四种算法在十种基准函数上的运行时间　　单位：s

基准函数	算法	运行时间	基准函数	算法	运行时间
Schwefels P2.22	GA	0.1993	Step	GA	0.1711
	PSO	0.1878		PSO	0.1939
	GWO	0.1154		GWO	0.1291
	ADGWO	0.2585		ADGWO	0.1891
Rosenbrock's	GA	0.3042	Sum of Different Powers	GA	0.1129
	PSO	0.1831		PSO	0.3517
	GWO	0.2454		GWO	0.2025
	ADGWO	0.3361		ADGWO	0.3505

续表

基准函数	算法	运行时间	基准函数	算法	运行时间
Zaharov	GA	0.1697	Ackley	GA	0.2514
	PSO	0.2228		PSO	0.2849
	GWO	0.1295		GWO	0.2561
	ADGWO	0.2438		ADGWO	0.4497
Schwefels	GA	0.2533	Dixon-Price	GA	0.2505
	PSO	0.3946		PSO	0.2573
	GWO	0.2515		GWO	0.1794
	ADGWO	0.3701		ADGWO	0.2761
Rastrigin	GA	0.2773	Levy	GA	0.2603
	PSO	0.4464		PSO	0.5358
	GWO	0.3322		GWO	0.3689
	ADGWO	0.4434		ADGWO	0.6573

3.3　小波神经网络模型

小波神经网络(Wavelet Neural Network,WNN)的概念是于1992年正式提出的,它是小波分析与神经网络相结合的产物,是一种基于小波分析理论以及小波变换理论而构建的多层人工神经网络。

小波分析最初是在进行地震信号分析时提出的,随后在20世纪80年代中期得到了迅速发展。小波分析被研究者视作傅里叶分析的突破性进展,它的出现以及使用使得研究者在计算机视觉、模式识别、图像处理等多个领域取得了突破性的进展。

要明白小波分析的原理首先要对傅里叶分析做一定的了解,傅里叶分析研究并扩展傅里叶级数和傅里叶变换的概念,主要研究如何将函数或信号表达为基本波形的叠加,其基本波形被称为调和函数。传统的信号理论是以傅里叶分析为基础的,但由于傅里叶变换中全局性变化的特点使其具有一定的局限性,例如无法进行局部分析。小波变换则是对傅里叶变换中局部化思想的一种继承与发展,作为一种新的变换分析方式,它的主要特点在于能够在变换的过程中突出某些局部问题的特征,同时对时间或空间频率的局部进行分析,而其特有的伸缩平移运算过程可以对信号逐步进行多尺度细化分析,最终可以聚焦到信号的任意细节,以此解决了傅里叶变换存在的局限性问题。

人工神经网络本身对数据信息能够进行并行处理,同时在处理过程中具有较强的自学习能力以及非线性逼近能力,而激活函数作为人工神经网络中神经元的核心,其作用在于将非线性因素引入神经元,它在输入传递至输出的过程中进行函数转换,以此将无限范围内的输入非线性变换为有限范围内的输出,一旦人工神经网络缺少激活函数,那么它每一层的数据传递过程就变成了单纯的矩阵计算过程,无论数据传递了多少层,最后的输出都是输入的线性组合。常用的激活函数有Sigmod()函数、Tanh()函数以及ReLU()函数。

Sigmod()函数是一种常见的S型函数,它能够将输入变量映射到0到1之间,具体

的函数公式如下：

$$f(x)=\frac{1}{1+e^{-x}} \tag{3.30}$$

Tanh()函数是一种双曲正切函数，它是由双曲正弦函数与双曲余弦函数推导而来的，其计算公式如下：

$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.31}$$

ReLU()函数是目前应用较多的一种激活函数，其计算公式如下：

$$f(x)=\max(0,x) \tag{3.32}$$

上述三种激活函数均存在一定的问题，当使用 Sigmod()函数时，若输入较大或较小，则函数的梯度值将会很小，而神经网络的反向传播过程都会利用梯度值对权重进行微分，这样就使得该微分也变得很小，不利于对权重进行优化。另外，相对于另外两种激活函数而言，Sigmod()函数中的指数运算会具有较大的运算量；当使用 Tanh()函数时，由于其函数曲线与 Sigmod()函数曲线较为相似，因此也会存在不利于权重优化的问题，但由于 Tanh()函数的值域为[−1,1]，因此相较于 Sigmod()函数其优化效果有所增强；ReLU()函数与 Sigmod()函数、Tanh()函数相比，当输入为正数时，不会出现梯度值较小的问题，同时由于其计算过程中只存在线性计算，因此其计算速度会快于另外两种激活函数，但 ReLU()函数自身也存在一定的缺陷，那就是当输入为负数时，ReLU()函数是无法进行激活的，若是在反向传递过程中，梯度将直接变成 0。由此可见，无论是哪种激活函数，均存在一定的问题，对激活函数进行研究改进也是优化人工神经网络性能的一个重要途径。

小波神经网络最显著的特征在于其激活函数使用的是小波基函数，该小波基函数一般为高斯函数，这一结合能够将小波变化的强大局部分析特性融入神经网络的非线性处理能力之中，以此在获得较强非线性逼近能力以及较快收敛速度的同时避免陷入局部最优值。

小波神经网络作为一种前向网络，主要分为前向传播过程与反向参数优化过程两部分，其中反向参数优化过程主要通过损失函数计算梯度值，从而利用梯度值对网络的参数进行微分以达到参数优化的效果。本节内容将上述原理过程进行介绍。

3.3.1 前向传播过程

小波神经网络的前向传播过程是指数据从输入层传入神经网络后依次通过隐含层、输出层，并在这两层进行线性计算与非线性转换后从输出层输出最终结果的过程。由此可以得出，小波神经网络的结构为三层神经网络结构，这三层分别为输入层、隐含层以及输出层，其结构图如图 3.8 所示。

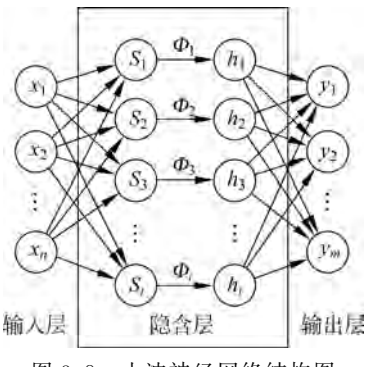

图 3.8 小波神经网络结构图

在图 3.8 中可以看到，输入层的节点数为 n，该数值的大小与输入数据的特征维度相同；隐含层的节点数为 i，这一数值大小的设置一般通过输入层与输出层

的节点数确定具体的数值范围，然后从该范围中进行选择；输出层的节点数为 m，这一数值要根据具体的求解问题来进行设定。将一组数据从输入层输入神经网络后，数据首先被传递至隐含层，在此传递过程中需要对数据进行加权线性计算，具体的计算公式如下：

$$S_j = \sum_{k=1}^{n} w_{kj} x_k, \quad j = 1,2,\cdots,i \tag{3.33}$$

其中，x_k 为输入数据中第 k 个样本数据；w_{kj} 为隐含层节点处的连接权值，S_j 为隐含层第 j 个节点处的输入数据。

当上述输入数据到达隐含层节点时，需要被激活函数进行非线性变换，在这里将被小波基函数即高斯函数进行非线性变换，这一变换过程也可以被称为伸缩变换，具体的变换公式如下：

$$\phi(x) = \cos(1.75x) * e^{-\frac{x^2}{2}} \tag{3.34}$$

$$h_j = \phi\left[\frac{S_j - b_j}{a_j}\right] \tag{3.35}$$

其中，$\phi(x)$为小波基函数；b_j 为该小波基函数的平滑因子；a_j 为基函数的伸缩因子；h_j 为隐含层第 j 个节点的输出数据。

最后隐含层第 j 个节点的输出数据进入输出层，经过计算后从输出层的 t 个节点输出，此节点上的计算公式如下：

$$y_t = \varphi\left(\sum_{j=1}^{i} w_{jt} h_j\right), \quad t = 1,2,\cdots,m \tag{3.36}$$

其中，w_{jt} 为输出层的连接权值；φ 为激活函数。小波神经网络中输出层的激活函数一般选择 Sigmod()函数。

由传递过程可以了解到，数据在神经元与神经元之间的传递是单向的，每个神经元只接收上一层神经元传递过来的数据并对其处理。在这个处理过程中，小波神经网络主要有四个参数参与计算，这四个参数分别是小波基函数的平滑因子 b_j、伸缩因子 a_j 以及隐含层与输出层的两个连接权值，这四个参数值的大小将直接影响网络的性能。

在对网络进行训练之前首先需要对这四个参数进行初始化，常用的神经网络的参数初始化方法大致可以分为三种，它们分别为预训练初始化、随机初始化以及固定值初始化。

预训练初始化是指对于在同一个数据集上已经完成训练后的神经网络模型可以获得一个较好的参数值，以此作为参数初始值来进行新的网络训练；随机初始化是指按照某种公式或规则确定神经网络参数的范围，然后在这个范围内随机生成初始化参数；固定值初始化相对前两种方法使用较少，这一方法采用固定值作为神经网络的参数初始值，例如当使用 ReLU()作为激活函数时，将偏置的初始值设为 0.01 可以使训练初期更容易激活。

上面三种初始化方法中，预训练初始化是基于已训练好的模型来生成初始参数，当使用此方法初始化参数时，训练初期会取得较好的收敛效果，但后期的收敛效果则无法得到保证，相对来说不够灵活，而部分三层神经网络一般不考虑进行偏置的设定，此时使用固

定值初始化方法无法达到设定偏置的效果，因此这一方法的可适用面相对另外两种方法来说更小。综上所述，研究者更多地使用随机参数初始化方法，下面将简要介绍几种参数随机初始化方法。

(1) 随机初始化(Random Initialization)，主要包括高斯分布初始化以及均匀分布初始化两种。高斯分布初始化是指使用高斯分布 $N(0,\sigma^2)$ 对每个参数进行随机初始化；均匀分布初始化是指在一个给定区间 $[-r,r]$ 内采用均匀分布来初始化参数。

(2) 标准初始化(Standard Initialization)，主要包括正态分布初始化以及均匀分布初始化两种。这一初始化方法更适用于 Sigmod()激活函数。假设神经网络的输入特征维度为 n，则正态分布初始化指的是随机初始化生成的每个参数满足正态分布 $N\left(0,\sqrt{\frac{1}{n}}\right)$，均匀分布初始化指的是随机初始化生成的每个参数满足均匀分布 $U\left(-\sqrt{\frac{1}{n}},\sqrt{\frac{1}{n}}\right)$。

(3) Xavier 初始化(Xavier Initialization)，这一初始化方法也分为正态分布初始化以及均匀分布初始化两种，且适用于 Losgistic()和 Sigmod()激活函数。假设神经网络的输入特征维度为 n，输出特征维度为 m，则正态分布初始化指的是随机初始化生成的每个参数满足正态分布 $N\left(0,\sqrt{\frac{2}{n+m}}\right)$，均匀分布初始化指的是随机初始化生成的每个参数满足均匀分布 $U\left(-\sqrt{\frac{6}{n+m}},\sqrt{\frac{6}{n+m}}\right)$。

(4) He 初始化(He Initialization)，与标准初始化与 Xavier 初始化相同，此方法分为正态分布初始化以及均匀分布初始化，更适用于 ReLU()激活函数。假设神经网络的输入特征维度为 n，则正态分布初始化指的是随机初始化生成的每个参数满足正态分布 $N\left(0,\sqrt{\frac{2}{n}}\right)$，均匀分布初始化指的是随机初始化生成的每个参数满足均匀分布 $U\left(-\sqrt{\frac{6}{n}},\sqrt{\frac{6}{n}}\right)$。

通常来说，在对神经网络参数初始化的方式进行选择时需要考虑到神经网络中使用的激活函数，按照具体的激活函数选择更为适用的参数初始化方式有利于网络获得更好的训练结果。

3.3.2 损失函数

在最常见的定义中，损失函数(Loss Function)或代价函数(Cost Function)被定义为一种将随机事件或与该事件有关的随机变量取值映射为某个非负实数来表示该随机事件存在的“风险”或“损失”的函数。这一函数在大多数时候都与优化问题相联系起来，例如在机器学习算法中，损失函数被用于模型的参数估计；在控制论中，损失函数被用作最优控制理论。从随机事件的类型来看，可以将损失函数应用到两类问题上，它们分别为回归问题与分类问题。下面将分别对这两种问题下的损失函数进行说明。

(1) 回归问题。回归问题是机器学习三大基本模型中的重要一环,它主要通过建立模型来分析模型的输入变量与输出变量之间的关系,这个过程一般是通过模型对输入变量的计算输出一个具体的数值。回归问题主要需对预测值与真实值进行分析,因此它的损失函数中的变量为模型的预测值与实际的真实值。回归问题大致上可分为两种损失函数,这两种损失函数的公式分别如公式(3.37)与公式(3.38)所示。

$$\text{Loss}=\omega(y_p-y_t)^2 \tag{3.37}$$

$$\text{Loss}=\omega\mid y_p-y_t\mid \tag{3.38}$$

式中,ω 代表实际的权值,y_t 为真实值,y_p 为模型的计算输出。上述两种计算方式十分相似,均为通过最小化损失值来对参数进行估计,但在实际的应用中,这两种计算方式有很大的区别。公式(3.37)中对真实值与模型计算输出之间的差值进行了平方输出,这一方式使得这两者的差值得到了放大,因此当真实值与模型计算输出之间存在差值,哪怕该差值很小时,也会给予惩罚,因此这一计算方式有利于误差梯度的计算,使模型的计算输出与真实值之间的差值达到最小;公式(3.38)中对真实值与模型计算输出之间的差值进行了绝对值输出,因此相较于公式(3.37)来说对真实值与模型计算输出之间的误差值不够敏感,但这一现象的优点在于当训练过程中存在异常值时,有利于使模型保持更加稳定的状态。

(2) 分类问题。分类问题从其本质上来说与回归问题是相似的,都可以被划分到预测这一问题领域中,但是回归问题的输出是一种连续值,而分类问题则是离散值。分类问题可以被定义为通过建立模型来判断输入数据属于哪一个类别,因此它主要需对模型的输出类别以及实际的类别进行分析。在对分类问题进行判定时,通常将分类正确的估计值取值为 0,将分类错误的估计值取值为 1,其损失函数如公式(3.39)所示。

$$\text{Loss}=\begin{cases}0, & y_p=y_t\\ 1, & y_p\neq y_t\end{cases} \tag{3.39}$$

可以看到上述公式为一个不连续的分段函数,因此在求解最小化问题时,这一公式不利于进行求解,因此研究者采用代理损失来进行模型参数的估计。代理损失一般是与原损失函数具有相合性的损失函数,常用的代理损失函数有铰链损失函数、交叉熵损失函数以及指数损失函数。

铰链损失函数是一种分段损失函数,与损失函数相同,当分类正确时取值为 0,而当分类正确但概率不足 1 或分类错误时,该样本将被用于参与模型求解。这种损失函数常用于支持向量机的分类过程,其具体的公式如(3.40)所示。

$$\text{Loss}=\max(0,1-y_py_t) \tag{3.40}$$

交叉熵损失函数是一种平滑函数,由交叉熵的定义可知,对交叉熵进行最小化的过程相当于最小化实际值与估计值的相对熵,即该函数可以视作提供一个无偏估计的代理损失函数。这一函数的公式如(3.41)所示。

$$\text{Loss}=-y_t\log(y_p)-(1-y_t)\log(1-y_p) \tag{3.41}$$

指数损失函数相较于上述两种代理函数,其最大的特点在于对错误分类施加的惩罚最大,即它的误差梯度大。当误差梯度大时,对其值求解能够获得更快的求解速度。此函

数的计算公式如(3.42)所示。

$$\text{Loss}=\exp(-y_p y_t) \tag{3.42}$$

上述几种损失函数各自均存在优点与缺点,同时对它们也提出了许多新的变形。如回归问题的损失函数方面有平均绝对误差损失函数、均方误差损失函数、Huber 损失函数以及 Log-Cosh 损失函数等,如分类问题的损失函数方面有 0-1 损失函数、log 对数损失函数、指数损失函数以及 Hinge 损失函数等。下面将选取部分原损失函数及其变形形式并进行具体的介绍。

(1) 平均绝对误差损失函数。它是公式(3.37)的变形,主要用于求取目标值与预测值之差的绝对值之和,以此来表达预测值的平均误差幅度,在使用此损失函数时通常不需要考虑误差的方向,其具体公式如(3.43)所示。

$$\text{Loss}=\frac{\sum_{i=1}^{n}|y_p^i-y_t^i|}{n} \tag{3.43}$$

其中,n 为输入样本的数量。

(2) 均方误差损失函数。它是回归损失函数中最常用的一种损失函数,是公式(3.38)的变形,主要用于求取预测值与目标值之间差值的平方和,其具体公式如(3.44)所示。

$$\text{Loss}=\frac{\sum_{i=1}^{n}(y_p^i-y_t^i)^2}{n} \tag{3.44}$$

平均绝对误差损失函数与均方误差损失函数在使用的过程中通常会拿来进行比较,当使用的训练数据中含有较多的异常值时,使用平均绝对误差损失函数会更加有效,因此这一损失函数会给异常值赋予更大的权重,以此来减小异常值所带来的误差,但它主要的问题在于当在神经网络的训练过程中误差值处于极大值点或极小值点时,此损失函数会使梯度值发生一个较大的变化,由此而导致较大的误差,从而影响到参数的优化结果。而当所处理的问题容易受到异常值的影响,需要减小异常值带来的影响时,对异常值更加敏感的均方误差损失函数则是更好的选择,它可以帮助网络的优化过程更加稳定和准确。

(3) Huber 损失函数。这一损失函数类似于平均绝对误差损失函数与均方误差损失函数的结合,这一特性使得它在对异常值不够敏感的同时保持了可微的特性。在其计算公式中,它使用了一个超参数来进行误差阈值的调节:当该超参数趋向于 0 时,其计算原理与平均绝对误差损失函数相同;当该参数趋向于无穷时,其计算原理与均方误差损失函数相同。该函数为一个连续可微的分段函数,其计算公式如下:

$$\text{Loss}=\begin{cases}\dfrac{1}{2}(y_p-y_t)^2, & |y_p-y_t|\leqslant\delta\\ \delta|y_p-y_t|-\dfrac{1}{2}\delta^2, & \text{其他}\end{cases} \tag{3.45}$$

(4) Log-Cosh 损失函数。它是一种较公式(3.38)更为平滑的损失函数,其主要特点为计算公式为双曲余弦函数,因此当误差较小时,其函数曲线与均方误差损失函数较为相似。该函数具有对异常值更加敏感的特点,而当误差值较大时,其函数曲线与平均绝对误

差损失函数较为相似，使得此时函数不会受到异常值的影响。该函数的计算公式如(3.46)所示。

$$\text{Loss}=\sum_{i=1}^{n}\log(\cosh(y_p^i-y_t^i)) \tag{3.46}$$

(5) 0-1 损失函数。它一般是对预测值与目标值之间的差值进行判断，当存在区别时取值为 1，当相同时取值为 0，其特点在于在分类问题中可以直接判断错误的个数。但是这一函数直接以预测值与目标值是否相同作为评判标准，在某些情况下此标准过于严苛，因此对其进行一定的调整，使得当预测值与目标值之间的差值小于某个数时，将其取值为 0。该函数的原公式如公式(3.39)所示，其调整后如公式(3.47)所示。

$$\text{Loss}=\begin{cases}1, & |y_p-y_t|\geqslant a\\ 0, & |y_p-y_t|<a\end{cases} \tag{3.47}$$

(6) log 对数损失函数。它特别适用于多分类问题，在其计算公式的结果中具有较好的表征概率分布，但其对异常值的敏感度相对较低，逻辑回归问题中常用此损失函数。其计算公式如下：

$$\text{Loss}(y_p,P(y_p\mid x))=-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}y_{ij}\log(P_{ij}) \tag{3.48}$$

其中，m 为分类存在的类别数；y_{ij} 为二值指标，表示类别 j 是否为输入 x_i 的真实类别；P_{ij} 表示模型预测的输入 x_i 属于类别 j 的概率。

(7) 指数损失函数。相较于 log 对数损失函数而言，它对异常值的敏感度更高，其常见的应用是与 AdaBoost 算法的结合，其计算公式如下：

$$\text{Loss}=\exp(-y_py_t) \tag{3.49}$$

(8) Hinge 损失函数。其计算公式如公式(3.50)所示。由此公式可以看到，当被分类正确时，损失值为 0；当被分类错误时，损失值为 $1-yf(x)$。通常 $f(x)$ 为处于 -1 到 1 之间的预测值；y 为 -1 到 1 之间的目标值，这一范围的设定使得正确分类的样本距离分割线能够不超过 1，即让分类器能够更专注于整体的误差值。

$$\text{Loss}=\max(0,1-y_py_t) \tag{3.50}$$

(9) 感知损失函数。它是 Hinge 损失函数的一个变形。与 Hinge 损失函数不同的是该函数不关注判定边界的距离，只关注样本类别的判定是否正确，因此该函数较 Hinge 损失函数也更加简单，同时其泛化能力较 Hinge 损失函数要差一些。其计算公式如公式(3.51)所示。

$$\text{Loss}=\max(0,-y_p) \tag{3.51}$$

(10) 交叉熵损失函数。其标准形式在公式(3.41)中已进行了具体的介绍，该标准形式常用于二分类问题，同时适用于激活函数 Sigmod()和 Softmax()的输出，当处理多分类问题时，使用公式(3.52)中的变形形式作为损失函数。交叉熵损失函数最大的特点在于它能够完美解决平方损失函数权重更新过慢的问题，因此当误差较大的时候，其权重更新速度较快，当误差较小时，其权重更新速度较慢。

$$\text{Loss}=-\frac{1}{n}\sum_{i=1}^{n}y_t^i\ln y_p^i \tag{3.52}$$

使用人工神经网络进行大数据预测时，关键的步骤在于判断输入变量输入神经网络后得到的输出变量与目标变量之间是否接近，当它们的差值较大时需要通过对网络进行训练来使此差距达到最小，因此大数据预测问题本质上属于回归问题。而在处理回归问题时，当在同一个人工神经网络模型上使用不同的损失函数时，模型的性能也会有较大的区别，因此即使使用损失函数也需要视实际情况而定，研究者通常选用均方误差损失函数作为使用人工神经网络进行大数据预测时的损失函数。

3.3.3 RMSProp

由前面的介绍可以得知，在使用人工神经网络进行预测时，初始化生成的参数在使用时往往难以使网络获得最好的回归效果，因此除了前向传播过程外，需要通过反向传播过程对相关参数进行优化，从而使得回归效果达到最好，而神经网络的训练过程即为前向传播过程与反向传播过程的循环重复。由 3.3.2 节对损失函数的相关介绍可以得知，使用损失函数的目的主要有两个：一个是用来衡量网络的回归效果是否优秀；另一个则是通过损失函数来获取输出层参数的梯度，从而对输出层的参数进行优化。

梯度可以被理解为一个矢量，它表示某个函数在这一点的方向导数在该方向上可以取的最大值，即函数在该点处沿着这个梯度方向变化最快、变化率最大。假设某一连续可微的多元函数为 $f(x_1,x_2,\cdots,x_n)$，则在点 $(x_1,x_2,\cdots,x_n)$ 处，可以将其梯度表示为 $\left(\frac{\partial f}{\partial x_1},\frac{\partial f}{\partial x_2},\cdots,\frac{\partial f}{\partial x_n}\right)$，此时函数 $f(x_1,x_2,\cdots,x_n)$ 在此方向上函数值增长的速度最快，而 $\left(-\frac{\partial f}{\partial x_1},-\frac{\partial f}{\partial x_2},\cdots,-\frac{\partial f}{\partial x_n}\right)$ 为该处的负梯度，即函数 $f(x_1,x_2,\cdots,x_n)$ 在此方向上函数值减小的速度最快。

根据上述梯度的定义可以将损失函数作为一个连续可微的多元函数，而由于输入数据为固定值，则由输出层的计算公式可以得知不同的参数值可以视作该多元函数上的不同的点，为了使该函数值能够达到目标值（最大或最小），则需要通过梯度值的计算，使得函数在梯度的方向上具有较快的增长（减小）速度。同理，隐含层的参数优化过程也是如此，由损失函数可计算得出隐含层输出数据，而隐含层的梯度值可以通过该数据计算，因此隐含层的参数可在此梯度方向上进行变化以获得最优值。

反向传播过程中最常用的算法即为误差反向传播算法（Error Back Propagation，EBP），接下来将以均方误差损失函数作为神经网络的损失函数，对误差反向传播算法的原理进行介绍。

假设神经网络输入层、隐含层、输出层的节点数分别为 n、i、m，其隐含层的权值为 w，阈值为 v，输出层的权值为 u，阈值为 γ，则其隐含层中第 j 个节点的输入的计算公式如下：

$$\alpha_j=\sum_{l=1}^{n}w_{lj}x_l \tag{3.53}$$

其输出层第 t 个节点的输入的计算公式如下：

$$\beta_t=\sum_{j=1}^{i}u_{jt}b_j \tag{3.54}$$

其中，b_j 表示隐含层第 j 个节点的输出。

假设该神经网络最后的输出为 $y_p=(y_p^1,y_p^2,\cdots,y_p^m)$，则第 t 个节点的输出为

$$y_p^t=f(\beta_t-\gamma_t) \tag{3.55}$$

其中，f 为激活函数。

使用均方误差损失函数计算后，获得的均方误差为

$$E=\frac{1}{2}\sum_{t=1}^{m}(y^t-y_p^t)^2 \tag{3.56}$$

使用神经网络进行预测时要求预测值与目标值之间的误差达到最小，因此上述计算过程需要朝负梯度的方向进行减小，此减小过程的计算公式如下：

$$\Delta u_{jt}=-\mu\frac{\partial E}{\partial u_{jt}} \tag{3.57}$$

其中，μ 为给定的学习率，而等号右边的梯度值可以通过以下公式进行计算：

$$\frac{\partial E}{\partial u_{jt}}=\frac{\partial E}{\partial y_p^t}\cdot\frac{\partial y_p^t}{\partial \beta_t}\cdot\frac{\partial \beta_t}{\partial u_{jt}} \tag{3.58}$$

而由 β_t 的计算公式可以得知

$$\frac{\partial \beta_t}{\partial u_{jt}}=b_j \tag{3.59}$$

假设输出层使用的激活函数为 Sigmod()函数，则对激活函数求导后可以求得

$$f=f(1-f) \tag{3.60}$$

由上述神经网络输出以及均方误差的计算公式可以求得

$$e_t=-\frac{\partial E}{\partial y_p^t}\cdot\frac{\partial y_p^t}{\partial \beta_t}=-(y^t-y_p^t)f'(\beta_t-\gamma_t)=y_p^t(1-y_p^t)(y^t-y_p^t) \tag{3.61}$$

最后可以得出参数 u_{jt} 的更新公式为

$$\Delta u_{jt}=\mu e_t b_j \tag{3.62}$$

同理，可分别计算得出参数 γ_t、w_{lj}、v_j 的更新公式，分别如下：

$$\Delta\gamma_t=-\mu e_t \tag{3.63}$$

$$\Delta w_{lj}=\mu g_j x_l \tag{3.64}$$

$$\Delta v_j=-\mu g_j \tag{3.65}$$

在公式(3.64)与公式(3.65)中

$$g_j=-\sum_{t=1}^{m}\frac{\partial E}{\partial \beta_t}\cdot\frac{\partial \beta_t}{\partial b_j}f'(\alpha_j-v_j) \tag{3.66}$$

其中

$$-\frac{\partial E}{\partial \beta_t}\cdot\frac{\partial \beta_t}{\partial b_j}=u_{jt}e_t \tag{3.67}$$

则可以得出

$$g_j=b_j(1-b_j)\sum_{t=1}^{m}u_{jt}e_t \tag{3.68}$$

上述计算过程即为误差反向传播算法的主要原理。在实际的应用过程中，首先将输

入数据输入神经网络，输入数据依次经过神经网络的隐含层、输出层，在每一层分别经由线性计算与非线性转换最后获得输出层的输出结果，然后使用损失函数计算出误差值，通过使用输出层的误差梯度来对输出层的权值、阈值进行调整，最后将误差反向传递至隐含层神经元并根据隐含层的误差梯度来对隐含层的权值、阈值进行调整。

同理，当神经网络为小波神经网络时，需要调整的参数为隐含层的平滑因子、伸缩因子、连接权值以及输出层的连接权值，因此需要利用输出层的误差来对输出层的连接权值进行调整，使用隐含层的误差对隐含层的平滑因子、伸缩因子、连接权值进行调整。

需要注意的是，当激活函数使用 Sigmod()函数或 Tanh()函数时，由这两个函数的图像可以得知在 x 趋近于正负无穷大时，偏导数趋近于 0，则此时对连接权值的调整值也会趋近于 0，因此无法进行调整，这就是误差反向传播算法中容易出现的梯度消失问题，因此在训练的过程中需要对学习率进行合理调整以防止梯度消失的现象发生。

将误差反向传播算法应用于神经网络参数优化时的算法（即 BP 算法）如算法 3.4 所示。

算法 3.4　BP 算法

```
随机初始化神经网络中所有的连接权值、阈值
while(未满足停止迭代条件)do
  for 所有输入数据
    按照前向传播过程计算网络的输出
    使用损失函数计算误差值
    根据误差值计算出隐含层、输出层每个参数的梯度项
    利用梯度项对所有参数进行更新
  end for
end while
输出隐含层与输出层参数确定的多层前馈神经网络
```

误差反向传播算法属于反向传播过程中的提出时间较早的基础算法，而目前研究者在该算法的基础上提出了许多新的算法，并且相较于该算法均取得了较好的优化效果，在使用小波神经网络进行大数据预测时，将采用均方根反向传播（Root Mean Square Back Propagation，RMSProp）算法作为误差反向传播算法对参数进行优化。

均方根反向传播算法通常会被视作自适应梯度（Adaptive Gradient，AdaGrad）算法的一种改进，因此在了解均方根反向传播算法前需要对自适应梯度算法的原理进行理解。自适应梯度算法与传统的误差反向传播算法不同的地方在于，在该算法中使用累积平方梯度代替迭代过程中每一次计算的梯度值。假设对某个参数计算的梯度值为 g，则自适应梯度算法需要通过下述公式计算出其梯度值的累积平方：

$$r = r + g^2 \tag{3.69}$$

其中，r 的初始值为 0。

之后通过累积平方对每次更新中的学习率进行调整，并使用更新后的学习率计算出参数的更新值。更新值的计算公式如式(3.70)所示。

$$\Delta = -\frac{\mu}{\delta + \sqrt{r}} * g \tag{3.70}$$

其中，μ 为全局学习率；δ 为避免分母为 0 而设置的极小值。

由前面的梯度值计算的相关原理可以得知，参数每次迭代更新的梯度与数据特征的稀疏性关联较大，当数据特征较为稀疏时，参数每次更新的梯度相对较小，此时累积梯度将长时间处于一个较小值，使得学习率的下降速度也会较慢，因此对参数的更新值会随之变大；当数据特征较为稠密时，参数每次更新的梯度相对较大，那么累积梯度将会较大，使得学习率加快下降速度，因此更新值会随之降低。由此带来的最直接的效果就是参数的调整速度随之加快进而使得神经网络的训练速度随之加快。

将自适应梯度算法应用于神经网络参数优化时的算法（即 AdaGrad 算法）如算法 3.5 所示。

算法 3.5　AdaGrad 算法

```
随机初始化神经网络中所有的连接权值、阈值
设置全局学习率 μ 及参数 δ，初始化梯度的累积平方
while(未满足停止迭代条件)do
    for 所有输入数据
        按照前向传播过程计算网络的输出
        使用损失函数计算误差值
        根据误差值计算出隐含层、输出层每个参数的梯度项
        计算梯度累积平方
        利用梯度累积平方更新参数
    end for
end while
输出隐含层与输出层参数确定的多层前馈神经网络
```

自适应梯度算法让每个参数在迭代过程中按照不同的学习率进行自适应调整，但其本身仍存在问题，由于该算法对梯度值不断地累积平方，因此无论数据特征如何，在达到一定的迭代次数后，累积平方会累加到一个较大值，此时计算得到的学习率将会变得极小，并会导致每次参数的更新量变得极小，最后更新速度停滞，训练也会随之结束。

为解决上述问题，研究者在该算法的基础上提出了均方根反向传播算法。均方根反向传播的主要改进点为对累积梯度的计算方式进行了调整，具体的计算公式如下：

$$r = \rho r + (1-\rho) g^2 \tag{3.71}$$

其中，ρ 为衰减系数。

对梯度平方进行累积，主要目的在于对过去历史梯度信息的获取，而公式(3.71)中衰减系数的加入可以控制历史梯度信息获取量的多少，它相当于采用了一个变量均方根来记录历史更新次数中的梯度平方的平均值，并以此作为对学习率进行调整的主要依据。采用此方法最大的一个优点在于无论迭代多少次，参数的调整量是以历史梯度信息的平均值作为参考依据，因此其参数的更新量相对来说更加缓和，也不会出现更新速度停滞、

训练提前结束的情况。

综上所述，均方根反向传播算法的算法核心思想可以视作给每一个参数设置一个均方值来计算历史梯度平方的平均值，之后再以全局学习率除以该均方值来获得此迭代次数下的学习率，最后以此学习率与当下梯度值相乘获得参数的更新量。

将均方根反向传播算法应用于参数优化时的算法（即 RMSProp 算法）如算法 3.6 所示。

算法 3.6　RMSProp 算法

```
随机初始化神经网络中所有的连接权值、阈值
设置全局学习率 μ、参数 δ、衰减系数 ρ，初始化梯度的均方值
while(未满足停止迭代条件)do
    for 所有输入数据
        按照前向传播过程计算网络的输出
        使用损失函数计算误差值
        根据误差值计算出隐含层、输出层每个参数对应的均方值
        根据每个参数对应均方值计算其学习率
        利用学习率对参数进行更新
    end for
end while
输出隐含层与输出层参数确定的多层前馈神经网络
```

3.3.4　Nesterov 动量

在普通的梯度下降法中，一般将梯度值定义为对参数进行优化时的调整方向，将学习率定义为朝调整方向上的移动步长，参数的梯度值的变化曲线通常是波动的不规则曲线，其中存在多个较低的梯度值，若此时使用学习率对参数进行调整，容易出现陷入局部最优，但如果结合历史梯度值对参数进行调整，则可以对参数的优化方向进行调整，以此加快收敛速度并避免陷入局部最优，深度学习的动量（Momentum）算法便是以此为目的而提出的。

“动量”这个概念起源于牛顿运动定律。在该定律中，动量是与物体的质量和速度相关的物理量，而一个物体的动量指的是这个物体在它运动方向上保持运动的趋势。深度学习中的动量则是引入了“速度”这一概念，以变量 v 作为参数在调整过程中的调整方向和速率，将变量 v 与学习率相乘并作为每次迭代中参数的调整量，其计算公式为

$$\Delta = -\mu v \tag{3.72}$$

其中，μ 为学习率；而变量 v 是通过对梯度值进行指数加权平均计算而来的，因此在理解变量 v 之前首先需要对指数加权平均的原理进行了解。

通常通过计算 n 个数的和之后再除以 n，以获得这些数的平均值，这个平均值为它们的算术平均值，而在对许多实际问题进行求平均值时，还需要考虑每个数据的重要程度，即需要根据这些数据的重要程度来获取它们的加权值，进而通过求和、相除以获得加权平

均值，算术平均值一般可以视作加权平均值的一种特殊形式，即此时每个数据的重要程度是一模一样的。

在上述描述中，变量 v 是随着迭代的次数而在不断变化的，属于时间序列数据，它反映了变量 v 随着时间而不断变化的趋势，因此倘若对此类数据进行求平均值需要使用加权移动平均来描述它变化的趋势。指数加权平均又被称为指数加权移动平均，可以被视作加权移动平均的改进，是一种常见的序列处理方式。下面结合一个降水量的例子对此方法的原理进行说明。

图 3.9 为一个月的降水量分布情况，图中横轴表示一个月为 30 天，纵轴表示每天的降水量。

下面使用指数加权平均来描述降水量的变化趋势，即使用下述公式来计算局部平均值：

$$v_t = \rho v_{t-1} + (1-\rho)\sigma_t \tag{3.73}$$

其中，v_t 表示局部平均值；σ_t 表示当天的降水量。

当衰减率 ρ 设为 0.9 时，计算出来的降水量趋势如图 3.10 中的深色曲线所示；当衰减率 ρ 设为 0.95 时，计算出来的降水量趋势如图 3.10 中的浅色曲线所示。对比这两条曲线可以看到浅色曲线相对于深色曲线而言整体向右平移，但是相对于深色曲线更加稳定，这主要是因为当衰减率 ρ 设为 0.9 时计算的是 10 天降水量的指数加权平均值，而当 ρ 设为 0.95 时计算的是 20 天降水量的指数加权平均值，因此后者所反映的趋势要更加平稳。

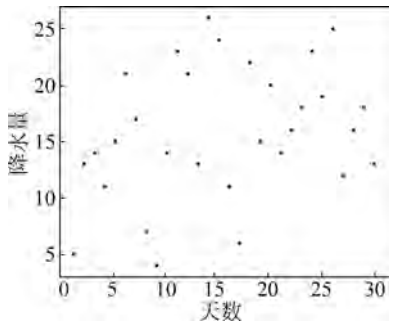

图 3.9　一个月的降水量分布情况

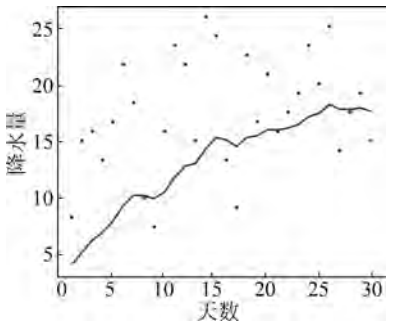

图 3.10　不同衰减率下的降水量变化趋势

上述指数加权平均的天数主要通过下述公式计算而来：

$$v_t = (1-\rho)(\sigma_t + \rho\sigma_{t-1} + \cdots + \rho^{t-1}\sigma_1) \tag{3.74}$$

式中，数值的加权系数会随着时间呈指数下降，通常数学中以 $\frac{1}{e}$ 作为一个临界值。对于任意 ρ，存在

$$\rho^{\frac{1}{1-\rho}} \approx \frac{1}{e} \tag{3.75}$$

则指数加权平均的平均数量计算公式为

$$N = \frac{1}{1-\rho} \tag{3.76}$$

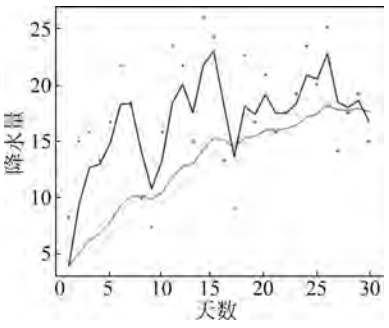

图 3.11 不同衰减率下的降水量变化趋势

图 3.11 展示了当衰减率 ρ 分别设为 0.9 与 0.5 时的降水量趋势，其中上方曲线代表的 ρ 值为 0.5，下方曲线代表的 ρ 值为 0.9。由于当 ρ 值为 0.5 时计算的是两天的平均降水量，因此它的曲线相较于下方曲线的波动性要大，但是此时曲线更能够拟合原始数据。

在梯度下降法中引入动量算法的重要步骤之一就是对参数的梯度值进行指数加权平均，这也是变量 v 的计算方式，其公式为

$$v_t = \rho v_{t-1} + (1-\rho)\mathrm{d}w_t \tag{3.77}$$

其中，变量 v_t 的初始值为 0；g 为参数的梯度值。另外由于 v_t 的初始值为 0，则其值将在迭代初值过小，此时容易导致参数的调整速率较小而影响优化速度，因此可以使用下列公式对 v_t 的取值进行调整：

$$v_t = \frac{v_t}{1-\rho^t} \tag{3.78}$$

当迭代次数较小时，上述公式的分母部分小于 1，此时将适当增大 v_t 的取值从而加快参数的调整速率；当迭代次数逐渐增大时，上述公式中的分母部分的计算结果将接近 1，则不会对动量项的大小造成影响。

由此可得动量算法的主要步骤为在每次通过损失函数计算得到参数的梯度值后，使用指数加权平均计算出动量项，最后使用公式(3.72)对参数进行更新。

另外结合公式(3.72)与公式(3.77)可以看出，在迭代过程中，由于动量项主要是对历史梯度信息的加权平均，因此梯度值将主要受到前一段时刻的影响，当梯度值前一段时刻保持相同方向时，动量项将在此方向上逐渐增大；当梯度值前一段时刻方向发生改变时，动量项将会随之减小。这一现象可以在梯度值出现反复波动时帮助其跳出局部最优，达到更好的收敛效果。

当将动量法与随机梯度下降相结合时，对动量法中变量 v_t 的计算方式与参数的调整公式进行了合并，采用动量参数 α 来参与计算动量项，具体的计算公式如下：

$$v_t = \alpha v_{t-1} - \mu g_t \tag{3.79}$$

之后对参数值进行如下更新：

$$w_t = w_{t-1} + v_t \tag{3.80}$$

将加入动量法的随机梯度下降法应用于神经网络参数优化时的算法如算法 3.7 所示。

算法 3.7 加入动量法的随机梯度下降算法

```
1:  随机初始化神经网络中所有的连接权值、阈值
2:  设置学习率 μ、动量参数 α
3:  while(未满足停止迭代条件)do
4:    for 所有输入数据
5:      按照前向传播过程计算网络的输出
```

```
6：        使用损失函数计算出每个参数的梯度值
7：        利用梯度值对每个参数的动量项进行计算
8：        对参数进行更新
9：      end for
10：  end while
11：  输出隐含层与输出层参数确定的多层前馈神经网络
```

在动量法提出后不久，Nesterov 动量法也随之被提出，此方法属于动量法的进一步发展。二者不同的是，在动量法中会用每一次迭代中计算得出的动量项作为参数调整的方向和速率，而在 Nesterov 动量法中，在计算新的动量项之前，会用当前的动量项进行一次参数的调整，然后用调整的临时参数计算梯度值，然后再利用该梯度值进行新的动量项的计算，最后进行参数的更新，此过程可以看作在动量法中添加了一个校正因子。Nesterov 动量法的核心思想在于在每次计算梯度时使用下一次的权重，然后将下一次的变化情况反映在这次计算的梯度上。

Nesterov 动量法的具体步骤如下。

首先使用当前的动量项进行参数的临时调整，调整公式如下：

$$w_t = w_{t-1} + \alpha v_{t-1} \tag{3.81}$$

然后使用调整后的临时参数计算出临时梯度值 g'，具体计算方式已在 3.2.3 节中进行详细说明。使用临时梯度值计算获得新的动量项，动量项的更新如公式(3.80)所示，最后再使用公式(3.81)对参数进行调整以获得新的参数值。

将加入 Nesterov 动量法的随机梯度下降法应用于神经网络参数优化时的算法如算法 3.8 所示。

算法 3.8　加入 Nesterov 动量法的随机梯度下降算法

```
1：  随机初始化神经网络中所有的连接权值、阈值
2：  设置学习率 μ、动量参数 α
3：  while(未满足停止迭代条件)do
4：    for 所有输入数据
5：      利用当前动量项对参数进行临时更新
6：      使用临时更新的参数按照前向传播过程计算网络的输出
7：      使用损失函数计算出每个参数的临时梯度值
8：      利用梯度值对每个参数的新动量项进行计算
9：      利用新动量项对参数进行更新
10：   end for
11：  end while
12：  输出隐含层与输出层参数确定的多层前馈神经网络
```

由于动量法与 Nesterov 动量法可以有效提高参数的优化速度，避免陷入局部最优，因此通常在应用时会将它们与各种不同的梯度下降法相结合。以均方根反向传播为例，加入 Nesterov 动量法后，该算法会在求累积梯度平方之前对参数进行临时更新，然后以

通过此临时参数计算获得的梯度值进行梯度平方累积，最后再对参数进行更新。

将加入 Nesterov 动量法的均方根反向传播算法应用于神经网络参数优化时的算法如算法 3.9 所示。

算法 3.9　加入 Nesterov 动量法的 RMSProp 算法

随机初始化神经网络中所有的连接权值、阈值
设置全局学习率 μ、参数 δ、衰减系数 ρ 以及动量参数 α，初始化梯度的均方根值
while(未满足停止迭代条件)do
  for 所有输入数据
    对参数进行临时更新
    按照前向传播过程计算网络的输出
    使用损失函数计算误差值
    根据误差值计算出隐含层、输出层每个参数对应的均方根值
    根据每个参数对应均方根值计算其学习率
    利用学习率对参数进行更新
  end for
end while
输出隐含层与输出层参数确定的多层前馈神经网络

3.4　协作复合神经网络模型的构建

通过 3.3 节对小波神经网络中相关原理的介绍，可以将小波神经网络的建模步骤总结如下：

(1) 根据输入数据的相关特征确定小波神经网络输入层、隐含层以及输出层的节点数；

(2) 使用 Xavier 均匀分布初始化方法随机初始化小波神经网络隐含层的连接权值、伸缩因子、平滑因子以及输出层的连接权值；

(3) 数据由输入层输入小波神经网络，传递至隐含层后经小波基函数对数据进行非线性转换；

(4) 数据在隐含层输出后传递至输出层，在与输出层的连接权值进行线性计算后由激活函数进行非线性转换，另外激活函数选择 Tanh()函数，最后得到网络的前向传播输出；

(5) 使用均方误差作为小波神经网络的损失函数，以网络输出值与目标值为参数计算得到损失值；

(6) 以输出层的损失值计算得到输出层连接权值的梯度，使用加入 Nesterov 动量法的均方根反向传播算法对此连接权值进行调整；

(7) 损失值传递至隐含层，以隐含层的损失值为参考依据，对隐含层的伸缩因子、平滑因子以及连接权值进行调整；

(8) 获得一个参数的更新后的小波神经网络；

(9) 在达到最大迭代次数之前，重复步骤(3)～步骤(8)，在达到最大迭代次数后，输出由隐含层与输出层参数确定的多层前馈神经网络。

小波神经网络模型的算法如算法 3.10 所示。

算法 3.10　小波神经网络模型的算法

```
确定神经网络的节点数，随机初始化神经网络中所有的参数
设置全局学习率 μ、参数 δ、衰减系数 ρ 以及动量参数 α，初始化梯度的均方根值
while(未达到最大迭代次数)do
    计算神经网络前向传播的输出
    计算每个参数的梯度值
    使用加入 Nesterov 动量法的均方根反向传播算法对每个参数进行更新
end while
输出由隐含层与输出层参数确定的多层前馈神经网络
```

使用小波神经网络进行大数据预测时，由于其参数的初始值采用随机初始化的方法，因此若初始化的参数值更加接近目标参数值，则网络的训练速度与训练效果也会相对较好，但一旦初始化生成的参数值偏离目标参数值，则会直接影响网络的训练速度以及最后的训练结果，同时可以得知在使用反向传播算法对参数进行调整时，依旧可能出现陷入局部最优的情况，因此小波神经网络的大数据预测性能也会出现一定的随机性。

通过前面群智能优化算法的介绍可以得知，若使用群智能优化算法对神经网络的初始参数进行优化，则可以使神经网络获得一个较好的初始参数值，进而使其获得更好的训练结果，而提出的自适应动态灰狼优化算法有效性通过对比实验也已得到了证明，因此为解决神经网络参数初始化随机性较高的问题，将自适应动态灰狼优化算法与小波神经网络相结合，以此构建协作复合神经网络模型用于大数据预测。

在协作复合神经网络模型中，自适应动态灰狼优化算法主要用于初始化小波神经网络的参数，在之后的训练过程中，仍需要使用加入 Nesterov 动量法的均方根反向传播算法对参数进行调整，但该算法并未考虑迭代过程中梯度值发生正负变化这一情况，而是使用固定的学习率作为学习步长，因此在面对较大的梯度变化时，容易导致优化结果的波动性大，降低收敛速度。为解决此问题，提出一种加入 Nesterov 动量法的自适应均方根反向传播(Adaptive Root Mean Square Back Propagation，ARMSprop)算法，该算法考虑到迭代过程中梯度发生正负值变化时，已经跳过了梯度值为 0 的点，因此在下一次调整过程中，不仅需要对学习方向进行调整，同时学习步长也应有所降低；而当前后两次迭代梯度的正负值未发生变化时，说明距离极值点较远，因此学习步长需要有所增加。

加入 Nesterov 动量法的自适应均方根反向传播算法基本步骤如下。以第 t 次迭代中权值 w 的更新为例，首先对权值 w 进行临时更新，并计算更新后的梯度 g_t。

$$w' = w_t + \alpha * v_{t-1} \tag{3.82}$$

其中，α 为动量系数；v_{t-1} 为上一次迭代更新值，初始值为 0。

计算累积梯度平方 r。

$$r_t = \rho * r_{t-1} + (1-\rho) * g_t \tag{3.83}$$

之后计算此时的学习率。

$$\mu = \begin{cases} \mu_0 * (s_t + \text{abs}(g))/s_t, & g_t * g_{t-1} \geqslant 0 \\ \mu_0 * (s_t - \text{abs}(g))/s_t, & g_t * g_{t-1} < 0 \end{cases} \tag{3.84}$$

其中，s_t 为累积梯度值，初值为 0；μ_0 为固定值。即若迭代前后梯度正负值发生变化，则学习率减小，反之则学习率增大；另外随着迭代次数的增大，s_t 不断增大，则学习率的变化不断减小。

最后对权值 w 进行更新。

$$v_t = \alpha * v_{t-1} - \frac{\mu}{\sqrt{r_t}} * g_t \tag{3.85}$$

$$w_{t+1} = w_t + v_t \tag{3.86}$$

结合上述不同的方法，协作复合神经网络模型的具体建模步骤如下。

(1) 根据输入数据的相关特征确定小波神经网络输入层、隐含层以及输出层的节点数。

(2) 对自适应动态灰狼优化算法的所有参数进行设定。

(3) 使用 Xavier 均匀分布初始化方法随机初始化小波神经网络隐含层的连接权值、伸缩因子、平滑因子以及输出层的连接权值，然后将四个参数组合成一个数组形式，例如，由小波神经网络的计算特点可知每个参数的存在形式均为数组形式，因此假设四个参数的数组形状分别为 8×4、8×1、8×1 以及 8×1，则将这四个数组组合成一个并列的大数组。

(4) 以上述步骤中的大数组作为自适应动态灰狼优化算法的种群个体，按照统一方式随机生成整个种群。

(5) 以使用每个种群个体作为小波神经网络的参数，使用均方误差作为小波神经网络的损失函数，计算得出每个个体对应的损失值，将此损失值作为每个个体对应的适应度值，适应度值越低则该个体的位置信息越好。

(6) 对适应度值按从小到大的顺序排序，适应度值越小则个体的位置信息越接近最优解，将适应度值排在前三的个体分别设定为灰狼 α、β 以及 δ，并对当前最优的位置信息进行保存。

(7) 依次对种群中每个个体的位置信息进行更新，针对更新后的信息重新计算其适应度值，然后用中心扰动准则来判断是否对个体的位置信息进行更新。

(8) 重新比较整个种群中个体的适应度值，保存灰狼 α、β 与 δ 的位置信息以及历史最优的位置信息。

(9) 按照设定的迭代次数重复步骤(5)～步骤(8)，当达到最大迭代次数时停止迭代过程，输出历史最优的位置信息，将该位置信息作为小波神经网络中参数的初始值。

(10) 使用该网络中参数的初始值确定新的小波神经网络，数据由输入层输入小波神经网络，传递至隐含层后经小波基函数对数据进行非线性转换。

(11) 数据在隐含层输出后传递至输出层，在与输出层的连接权值进行线性计算后由

激活函数进行非线性转换，另外激活函数选择 Tanh()函数，最后得到网络的前向传播输出。

(12) 使用均方误差作为小波神经网络的损失函数，以网络输出值与目标值为参数计算得到损失值。

(13) 以输出层的损失值计算得到输出层连接权值的梯度，使用加入 Nesterov 动量法的自适应均方根反向传播算法对此连接权值进行调整。

(14) 损失值传递至隐含层，以隐含层的损失值为参考依据，对隐含层的伸缩因子、平滑因子以及连接权值进行调整。

(15) 获得一个参数的更新后的小波神经网络。

(16) 在达到最大迭代次数之前，重复步骤(11)～步骤(15)，在达到最大迭代次数后，输出隐含层与输出层参数确定的多层前馈神经网络。

(17) 将待预测数据的输入特征数据输入此多层前馈神经网络，经过网络的前向计算获得最后的预测值。

协作复合神经网络模型的流程图如图 3.12 所示。

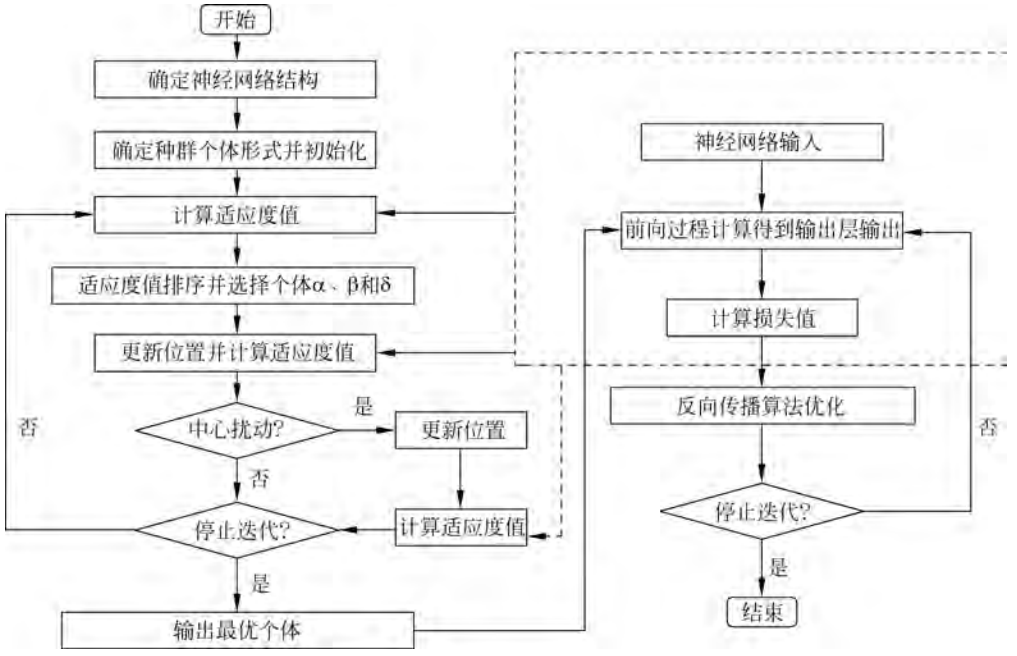

图 3.12　协作复合神经网络模型的流程图

结合协作复合神经网络模型的建模步骤，可以将其特性归纳总结如下。

(1) 该模型对神经网络结构的设置以实验数据的数据特征为主要依据，当对实验数据进行更改时，需要重新对神经网络结构进行调整。

(2) 该模型对参数的优化主要分为两部分：一部分是使用自适应动态灰狼优化算法对参数进行初始化生成；另一部分是使用反向传播算法对参数进行优化调整。

(3) 相较于传统生成方式，自适应动态灰狼优化算法可以使参数更加接近目标值，从

而进一步提高训练速度，获得更好的优化结果。

(4) 反向传播算法选择的是加入 Nesterov 动量法的自适应均方根反向传播算法，此算法引入 Nesterov 动量的思想，结合历史梯度值对参数进行调整，则可以对参数的优化方向进行调整，以此加快收敛速度并避免陷入局部最优。

(5) 加入 Nesterov 动量法的自适应均方根反向传播算法将根据梯度值正负变化而自适应调整的学习率引入算法步骤中，降低了优化结果的波动性，加快收敛速度。

(6) 该模型虽然将自适应动态灰狼优化算法与小波神经网络相结合，但以小波神经网络输出的损失值作为适应度值这一设置只利用了小波神经网络的前向计算过程，主要涉及基本线性计算与非线性转换，而后续的参数调整部分与小波神经网络反向传播过程完全一致，因此算法复杂度并未随之提高。

(7) 在参数设置方面，除自适应动态灰狼优化算法与小波神经网络中的原参数外并未新增其他参数。

3.5 知识扩展

在对自适应动态灰狼优化算法进行性能比较时，对比算法采用的是遗传算法与粒子群优化算法，当对这两种算法的基本原理不够了解时，则无法更加深入地理解它们与自适应动态灰狼优化算法之间的区别。而在对反向传播的介绍中也只涉及了自适应梯度算法与均方根反向传播算法，但实际上这一类算法有许多种，并且它们之间均存在一定的关联性。因此，为了帮助更好地理解前面所介绍的内容，本节将对前面提到相关算法进行补充说明。

3.5.1 遗传算法

遗传算法(Genetic Algorithm，GA)这一概念最早于 1967 年在 Bagley 的博士论文中被提到，该论文主要讨论了遗传算法在博弈论中的应用。之后 Bagley 的老师 J. Holland 教授于 1975 年提出了遗传算法的相关基础理论和方法。在 20 世纪 80 年代之后研究者对遗传算法的研究开始增加，遗传算法的发展进入兴盛时期，同时其被广泛应用于自动控制、图像处理等研究领域。

遗传算法通常被定义为是根据达尔文生物进化理论中的自然选择以及进化过程而提出的一种计算模型，它符合自然界的生物进化规律。从算法方面来说，遗传算法是一种将待求解问题看作自然环境，将问题的每一个候选解决方案作为环境中的个体，通过仿照自然界的进化过程而进一步发现环境中的最优个体即表现最优的候选解决方案。

遗传算法主要以种群中的所有个体作为对象，然后以待解决问题的空间范围对所有个体进行随机编码，最后以选择、交叉、变异这三个操作来实现目标范围内的搜索过程。遗传算法直接对所有的候选解按照设定的步骤进行相关操作，不存在进行求导和函数连续性的限定，无须确定的规则就能自适应地在目标空间调整搜索方向，确定搜索目标。

在对遗传算法的相关术语进行了解之前，首先需要对遗传算法中涉及的相关生物术语的原理进行了解，下面将分别对这些术语展开相关介绍。

(1) 基因型。染色体决定的性状的内部表现。

(2) 表现型。由基因型决定的个体的外部表现。

(3) 编码。遗传信息在染色体上按照一定的顺序进行排列，可以看作由表现型到基因型的映射。

(4) 解码。可以看作由基因型到表现型的映射。

(5) 进化。生物为了在环境中获得更好的生存状态而不断调整自身品质以适应环境的过程，此过程通常以一个种群为单位进行。

(6) 适应度。生物对所生活环境的适应程度。

(7) 选择。通常指个体按照自身是否更适合在环境中生存而被环境进行选择的过程。此过程按照一定的概率进行，个体的适应度越大越容易被选择。

(8) 交叉。也可称为基因重组，指两条染色体在同一位置被分割成两段，然后进行交叉重组从而形成新的两条染色体。

(9) 复制。在细胞分裂时，染色体会被复制进入新生成的细胞中，新细胞与旧细胞的基因信息完全一致。

(10) 变异。在染色体复制的过程中，部分染色体的某一小段基因信息发生改变，从而生成了新的染色体。发生染色体变异的概率通常很小。

与上述生物术语相对应，下面将对它们在遗传算法中的意义进行解释说明。

(1) 编码。遗传算法在对待求解问题的候选解进行生成时，主要以遗传空间中的染色体或个体的形式进行表现，这个随机生成过程被称为编码。编码首先需要可以对应遗传空间中所有问题的候选解，其次一个染色体或个体将与一个候选解相对应，最后所有的候选解将可以作为遗传空间中染色体的直接表现。编码的方法有许多种，如二进制编码法、浮点编码法等。由于遗传算法所求解的问题解部分为十进制数，而二进制数、浮点数等更适合算法的交叉、变异等过程，因此在进行遗传算法的相关操作前需要进行十进制数与二进制数等形式之间的转换过程。

(2) 解码。在对待求解问题的候选解编码并进行选择、交叉、变异操作后，需要重新计算该候选解的适应度，因此需要将候选解的数的形式转换为原形式，此转换过程即为解码过程。

(3) 种群初始化。种群中的所有个体均需通过实际问题随机生成，根据实际待求解问题确定最优解存在的空间范围，然后在该空间范围内按照候选解的存在形式随机生成初始群体。

(4) 适应度值。适应度值是用来评价候选解的好坏的数值，该数值与目标求解的问题直接相关。以求解目标范围最大(最小)问题为例，可以将该求解问题相关函数视作适应度函数，候选解经适应度函数计算后获得的值为适应度值。对适应度值好坏的判断通常以待求解问题为依据，求解最大(最小)值时，适应度值越大，候选解越好。

(5) 选择。按照种群中个体的适应度值大小，选择出一定数量的个体进行之后的交叉、变异操作。一般使用轮盘赌方法进行选择。轮盘赌方法首先需要通过个体适应度值

计算出每个个体遗传到下一代的概率。此概率的计算公式如下：

$$p(x_i)=\frac{f(x_i)}{\sum_{i=1}^{n}f(x_i)} \tag{3.87}$$

其中，$f(x_i)$为个体 x_i 的适应度值；n 为种群数量。个体的适应度值越高，则被遗传到下一代的概率越大。

之后计算个体的累积概率，以累积概率为依据判断是否选择该个体。累积概率的计算公式如下：

$$P(x_i)=\sum_{j=1}^{i}p(x_j) \tag{3.88}$$

之后随机生成 0～1 的实数 p'，然后将该实数与每个个体的累积概率进行比较，若累积概率大于此实数 p'，则将此个体选择遗传至下一代或进行之后的交叉、变异操作。

(6) 交叉。此过程主要使用交叉算子对被选择的个体进行交叉操作。以二进制编码为例，每个个体为一串较长的二进制编码数，以交叉算子作为对个体进行交叉操作的概率，每两个个体为一组进行操作，针对每个被选择的个体随机生成一个 0～1 的数，若该数小于变异算子，则对此组个体进行交叉操作，这是以随机算子个体中的某个数作为交叉点，将两个个体交叉点后的部分进行交换，以此组成两个新的个体来完成交叉操作。

(7) 变异。此过程主要使用变异算子对被选择的个体进行变异操作。以二进制编码为例，以变异算子作为对每个选择的个体进行变异操作的概率，针对每个被选择的个体随机生成一个 0～1 的数，若该数小于变异算子，则对该个体进行变异操作，随机选择个体中的某个或多个数进行变异，即重新随机生成该数值。

在上述遗传算法的相关术语介绍中提到了个体的编码过程，个体的编码方式有许多，大致上可以分为三类：二进制编码法、浮点编码法以及符号编码法。下面将针对这三种方法分别进行介绍。

(1) 二进制编码法主要使用 0 和 1 两个数来对种群的个体进行表示，类似染色体中使用两种碱基来串联成一条染色体，虽然只有 0 和 1 两种状态，但是足够长的二进制编码数可以表达所有的特征，二进制编码的表达类型为“110101011”。

二进制编码法操作简单易行，当使用此种编码方式生成个体后，个体的形态更接近染色体的形态，因此进行交叉、变异操作更加符合生物学原理。但是这种方法在对连续函数进行优化处理时，由于其随机性使得局部搜索能力较差，另外变异过程虽然是对部分二进制编码数位进行操作，但实际上对原数值的影响较大，容易使得个体的适应度值相较于变异前降低。

(2) 浮点编码法相对于二进制编码法来说最大的区别在于每个基因值采用某个范围内的一个浮点数来表示，此范围值必须保证在给定的区间范围内，因此在继承了二进制编码优点的同时，能够在个体编码长度较长时获得更高的精度。浮点编码法的表达类型为“2.3-4.1-2.6-5.2-6.3”。

由于浮点编码法使用浮点数来表示基因值使得其精度得到了提高，因此该方法更适合表示范围较大的数，同时也更适合在较大的空间内进行搜索。通常遗传算法在应用过程中会与其他方法相结合，浮点编码法的编码方式相较于二进制编码法而言也更适合与其他方法混合。

(3) 符号编码法在对个体染色体进行编码时通常将其基因值取自一个无数值含义而只有代码含义的符号。这种编码方式更加符合有意义积木块编码原则，同时倘若需要用到求解问题的专门知识时，该算法也会相对来说更加方便。

以使用遗传算法优化人工神经网络的参数为例，人工神经网络的参数一般为数组形式，为了使遗传算法的优化过程更加简单，则可以将种群的个体以多个神经网络参数数组组成大数组，之后的交叉操作将以不同小数组之间的连接点作为交叉点，变异操作以小数组为单位进行取值范围内的随机变异。

在遗传算法中，主要需要设置的参数为种群的数量、交叉算子、变异算子以及迭代次数。因此，结合上述内容，可以将遗传算法的步骤总结如下。

(1) 根据目标求解问题，对遗传算法的相关参数进行设置。

(2) 按照问题的候选解形式，在目标范围内随机生成多个种群个体，然后计算每个个体的适应度值，将当前适应度值最高的个体信息作为历史最优个体信息。

(3) 对每个个体进行编码。

(4) 对每个个体进行选择操作，使用轮盘赌方法选择遗传到下一代或进行交叉、变异操作的个体，一般种群中适应度值排在前几名的个体可直接遗传到下一代。

(5) 对选择的个体进行交叉操作，根据交叉概率对经过选择后的种群个体进行个体间基因片段的交换。

(6) 对选择的个体进行变异操作，根据变异概率对经过选择后的种群个体进行基因片段的改变。

(7) 将选择直接遗传到下一代的个体、交叉操作的个体以及变异操作的个体，总数量设置为种群的数量，并以它们作为新的种群个体，然后对每个个体进行解码，之后计算每个个体的适应度值，对历史最优个体信息进行更新。

(8) 重复步骤(3)～步骤(7)，当达到最大迭代次数或满足停止迭代条件后停止迭代，然后输出此时种群中的全局历史最优个体的位置信息。

遗传算法的流程图如图 3.13 所示。

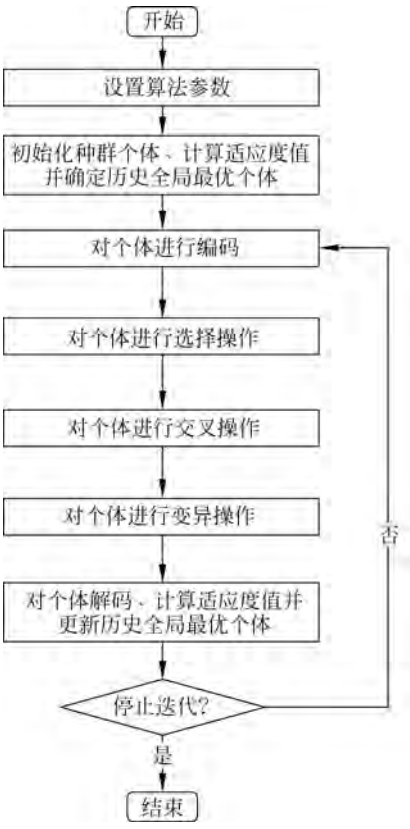

图 3.13　遗传算法流程图

遗传算法如算法 3.11 所示。

算法 3.11　遗传算法

```
1:  确定遗传算法中的相关参数
2:  初始化生成种群中的所有个体,计算每个个体的适应度值,将当前适应度值最高的个体信息
    作为历史最优个体信息
3:  while(未达到最大迭代次数)do
4:      对所有个体进行编码
5:      选择操作
6:      交叉操作
7:      变异操作
8:      对个体进行解码并计算其适应度值,对历史最优个体信息进行更新
9:  end while
10:     输出历史最优个体信息
```

下面以遗传算法对 Zaharov 测试函数进行目标范围内的最小值寻优为例对此算法进行说明,该测试函数的计算公式在 3.2.6 节中已介绍,其取值范围为[−5,10],取值范围内的理想最优解为 0,将其搜索的空间维度设为 20。

在参数设置方面,将遗传算法的种群数量设为 30,交叉算子设为 0.8,变异算子设为 0.05,选择直接遗传至下一代的个体数量为 10,进行遗传操作的个体数量为 10,进行变异操作的个体数量为 10,迭代次数设为 1000。

主要的寻优过程如下:首先在此函数的取值范围[−5,10]内随机生成 30 个 20 维数组作为种群个体的位置信息,对每个数组依次使用测试函数公式计算适应度值,确定适应度值最高的个体作为历史最优个体;然后对每个数组进行二进制编码,按照适应度值大小进行轮盘赌法选择操作,选出 10 个个体直接遗传至下一代,之后从剩下的个体中选出 10 个进行交叉操作,其余个体进行变异操作;之后对新生成的种群个体进行解码,然后计算它们新的适应度值,更新历史最优个体;最后重复相关遗传操作,当达到最大迭代次数 1000 时停止迭代,输出此时的种群历史最优个体适应度值及其位置信息。

在优化过程中,种群个体的历史最优适应度值的大小随迭代次数的变化情况如图 3.14 所示。

由图 3.14 可以看出,在迭代过程中,种群个体朝向最优适应度值的收敛速度一直发生变化,从迭代开始到迭代 150 次左右,算法都具有一个较快的收敛速度,其中历史最优适应度值由超过 10^6 迅速降低到低于 10^3,说明算法在迭代前期能够在目标范围内进行较为全面的搜索,以此发现更多可能存在的较多解。另外观察这一阶段的适应度值曲线可以看到,在此下降过程中有三段较短的直线存在,这可能是因为在此期间算法收敛至局部最优区域导致历史最优适应度值无较大变化,而这三段直线之后均出现了下降的曲线,即适应度值均有所下降,说明算法跳出了此局部最优。最后算法在迭代 150 次之后最优适应度值未发生大的变化,此时可能存在两种情况:一种是此时已陷入了局部最优区域且难以跳出此状态;另一种则是算法收敛效果较好,已经找得了目标范围内的全局最优解。

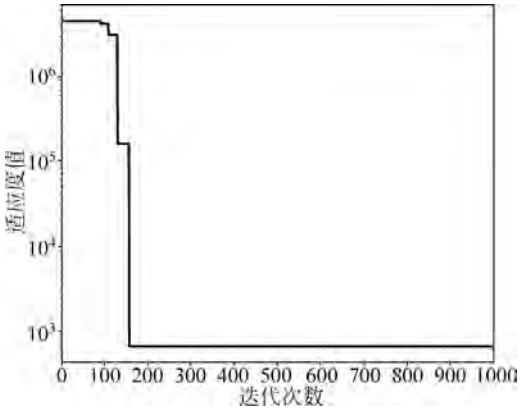

图 3.14　历史最优适应度值的变化情况

3.5.2　粒子群优化算法

粒子群优化(Particle Swarm Optimization,PSO)算法是 1995 年被 Kennedy 等人提出的一种模拟自然界中鸟群进行觅食过程的群智能优化算法。由于此算法在操作简单的同时具有较快的收敛速度,因此它具有较大的发展价值和发展空间。

目前,粒子群优化算法被应用到多种不同的领域中,例如将粒子群优化算法用于人工神经网络中的参数初始值寻优或参数优化过程以获得更好的神经网络性能;将电力领域中的某些问题转换为函数的最小化问题,进而使用粒子群优化算法进行寻优求解;除此之外还在系统识别、信号识别、目标检测等多个方面得到了较好的应用。

粒子群优化算法的核心思想灵感来源于自然界中的鸟群觅食行为,在自然界中鸟群之间均存在信息的相互交流,包括能够搜索到食物的鸟群个体向种群中其他个体传递食物具体的位置信息等。而粒子群优化算法则是将待求解问题的每一个候选解视作鸟群中的每一个个体的具体位置信息,每个候选解对应的最优适应度值作为每个个体在该位置处所能搜索到的食物的量,通过个体间位置信息的相互交流来发现目标范围内的最优适应度值对应的最优候选解。

粒子群优化算法以种群中存在的每个个体作为实施对象,让每个个体对种群中历史的最优位置信息以及目前该个体所到达的最优位置信息进行学习,通过此学习来确定自身接下来的移动方向和移动距离。此算法的特点在于它对全局历史最优位置信息与个体历史最优位置信息进行了保存。这一特点可以有效帮助种群在获得较快的收敛速度的同时避免过早陷入局部最优。

在使用粒子群优化算法进行优化问题的求解时,需要进行理解的概念主要有两个:一个是粒子的位置信息;另一个是粒子的速度信息。

粒子的位置信息通常对应待优化问题的候选解,最初需要对此位置信息在目标范围内进行初始化,然后通过此位置信息计算获得此粒子的适应度值。与遗传算法一样,粒子群优化算法也使用适应度值来衡量候选解的好坏。在求解目标范围最大(最小)问题时,将该求解问题相关函数视作适应度函数,候选解经适应度函数计算后获得的值为适应度

值。在待求解问题为求解最大(最小)值时,适应度值越大(越小),候选解越好。在每次迭代的过程中粒子需要对自己的位置信息进行更新,具体的更新公式如下:

$$X_i^{t+1} = X_i^t + V_i^{t+1} \tag{3.89}$$

其中,V_i^{t+1} 为个体 i 在第 t 次迭代后的速度;X_i^t 个体 i 在第 t 次迭代前的位置信息;X_i^{t+1} 个体 i 在第 t 次迭代后的位置信息。

由公式(3.89)可以了解到,在对个体的位置信息进行更新前,需要通过计算获得个体新的速度信息,这一速度信息一般包括个体在接下来的一次迭代过程中的移动方向和移动距离,其具体的计算公式如下:

$$V_i^{t+1} = wV_i^t + c_1 r_1(\text{pbest}_i^t - X_i^t) + c_2 r_2(\text{gbest}^t - X_i^t) \tag{3.90}$$

其中,V_i^t 为个体 i 在第 t 次迭代前的速度,速度的初始值为 0;w 为惯性因子;c_1、c_2 为加速因子,其中前者为每个粒子的个体学习加速因子,后者为每个粒子的全局学习加速因子,通常这两个数被设置为常数 2,也可设为其他的常数,其取值范围处于[0,4)]区间,r_1 与 r_2 均为 0～1 的随机数;pbest_i^t 为个体 i 在第 t 次迭代之前搜索到的最优位置信息;gbest^t 为第 t 次迭代时该种群搜索到的最优位置信息。

上述公式等号右边主要分成三部分。第一部分为记忆项,表示粒子按上次的速度大小与方向移动的趋势。其中,惯性因子通常被设为非负数,当此值较大时,表示个体对过去移动信息的继承量较大,此时个体全局寻优能力较强,局部寻优能力较弱;当此值较小时,个体对过去移动信息的继承量较小,则个体的全局寻优能力较小,局部寻优能力较强。第二部分为自身认知项,表示粒子根据自身过去最好的经验进行位置移动。第三部分为群体认知项,表示粒子根据全局历史最好的经验进行位置移动。

在粒子群优化算法中,需要提前设置的参数分别为粒子群的种群数量、惯性因子 w、加速因子 c_1 与 c_2。因此结合上述内容,可以将粒子群优化算法的算法步骤总结如下。

(1) 根据目标求解问题,对粒子群优化算法的相关参数进行设置。

(2) 按照问题的候选解形式,在目标范围内随机生成多个种群个体,然后分别计算出每个个体的适应度值。

(3) 将每个个体适应度值作为其历史最优适应度值,对应的位置信息作为历史最优位置信息;比较种群中所有个体的适应度值,将最好的适应度值作为全局历史最优适应度值,将对应的位置信息作为全局历史最优位置信息。

(4) 对每个个体依次使用公式(3.90)对其速度信息进行更新,之后使用公式(3.89)对其位置信息进行更新。

(5) 对更新位置信息后的个体计算它们的适应度值,将此适应度值与个体历史最优适应度值进行比较,若更新后的适应度值更优,则对个体历史最优适应度值以及对应的个体历史最优位置信息进行更新,反之则不更新。

(6) 选出此时种群中适应度值最优的个体,将其适应度值与全局历史最优适应度值进行比较,若最优的适应度值较全局历史最优适应度值更好,则对全局历史最优适应度值以及对应的全局历史最优位置信息进行更新,反之则不更新。

(7) 重复步骤(4)～步骤(6),当达到最大迭代次数或满足停止迭代条件时停止迭代,

然后输出此时种群中的全局历史最优个体的位置信息。

粒子群优化算法的流程如图 3.15 所示。

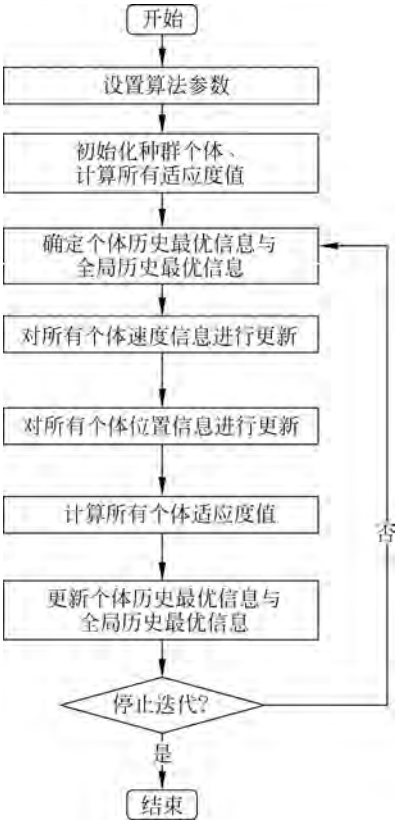

图 3.15　粒子群优化算法的流程

粒子群优化算法如算法 3.12 所示。

算法 3.12　粒子群优化算法

```
1:  确定粒子群优化算法中的相关参数
2:  初始化生成种群中的所有个体,计算每个个体的适应度值,将当前所有个体的信息作为个体
    历史最优信息,将适应度值最高的个体信息作为全局历史最优信息
3:  while(未达到最大迭代次数)do
4:      对所有个体速度信息进行更新
5:      对所有个体位置信息进行更新
6:      计算更新后的个体适应度值
7:      对个体历史最优信息与全局历史最优信息进行更新
8:  end while
9:  输出全局历史最优个体信息
```

将粒子群优化算法的原理与遗传算法相比,它们存在许多相同点与不同点。相同点

主要表现在以下方面。

(1) 两种算法都是在目标范围内对种群个体进行随机初始化。

(2) 使用适应度值衡量个体位置信息(候选解)的优劣性。

(3) 都是以达到最大迭代次数或满足停止迭代条件时停止迭代。

(4) 两种算法都是进行随机搜索,都不能够保证一定搜索到目标范围内的最优解。

两种算法的不同点则相对来说较为明确,那就是在遗传算法中对个体位置信息进行更新时主要是通过个体之间的交叉以及个体自身的变异来实现的,这两个过程表现为不同的个体之间进行信息的交换以及个体自身的随机转换。而在粒子群优化算法中对个体位置信息进行更新时主要通过对自身上一次迭代时速度信息、自身历史最优位置信息以及全局历史最优位置信息进行学习,这三个过程表现为对自身信息的学习以及对全局最优信息的学习。

对粒子群优化算法的优缺点进行总结,其优点主要表现为以下方面。

(1) 此算法为不确定性算法,因此在用于进行全局寻优时算法有更大的概率能够搜索到全局最优解。

(2) 该算法为一种群智能优化算法,种群的各个个体之间通过相互协作来更好地调整自身的位置信息以获得对环境更好的适应度。

(3) 具有并行性,所有个体同时进行位置信息的调整以搜索目标范围内的最优解。

(4) 具有记忆功能,所有粒子均保存自身的历史最优位置信息与对应的历史最优适应度值,除此之外还保存整个种群的历史最优位置信息以及对应的历史最优适应度值。

(5) 具有一定的稳定性,此算法在多种不同的环境下依旧适用。

而粒子群优化算法的主要缺点则表现在以下方面。

(1) 种群个体位置信息更新只局限在种群内部的信息交流,因此容易陷入局部最优。

(2) 惯性因子 w 的设置通常会直接影响到算法的性能,但是其设置方式却没有一个较好的方案,若该值设置不当则会直接影响算法的局部搜索能力。

(3) 当待求解问题的复杂性变高或求解的空间范围变大时,算法的搜索精度相对来说会出现降低的情况。

为了克服上述缺点,获得更好的粒子群优化算法性能,研究者主要通过以下几种方式对粒子群优化算法进行改进。

(1) 改变种群内粒子之间的拓扑结构。传统粒子群优化算法中一个较大的优点在于保存了全局历史最优信息与粒子的历史最优信息,但是由于粒子群的相互关系为并列存在,因此可供学习的全局历史最优信息只有一个,同时搜索范围较小。将粒子之间的拓扑结构进行调整,例如将其相互之间改成环形拓扑结构并增加此结构的数量,则整个种群内粒子可进行学习较优位置信息相对有所增加,在某些拓扑结构的粒子陷入局部最优时,其他较优的位置信息可以帮助其迅速跳出局部最优,一定程度上保证了解的最优性。

(2) 引入新的算法机制。目前所使用的粒子群优化算法步骤相对较为简单,同时在某些步骤上的合理性仍然存在一定的欠缺,例如传统的粒子群优化算法为无论更新后的粒子位置信息如何变化均会接受此次更新,但是在对粒子每次的位置进行更新后有很大概率出现粒子的适应度值降低的情况,因此将直接影响算法的收敛速度。在这里可以考

虑设置一个新的算法机制，此算法机制在每次粒子进行位置更新后对其适应度值进行比较，若粒子适应度值出现降低或保持不变的情况，则可以考虑按照一定的概率不接受此次位置更新或重新对位置进行更新。

(3) 调整惯性因子 w 的设置方式。在传统方法中，惯性因子通常被设为一个固定的非负数，当此值较大时，表示个体对过去移动信息的继承量较大，此时个体全局寻优能力较强，局部寻优能力较弱；当此值较小时，个体对过去移动信息的继承量较小，则个体的全局寻优能力较小，局部寻优能力较强。部分研究者采用线性下降或非线性下降的更新方式让惯性因子 w 随着迭代的次数进行变化，通常在迭代前期保持一个较大的惯性因子以获得较好的全局寻优能力，在迭代后期保持一个较低的惯性因子以在局部进行更好的搜索。

(4) 与其他算法的结合。不同的算法自身均存在不同的优点与缺点，将某些算法与粒子群优化算法相结合可以利用这些算法的优点来克服粒子群优化算法自身的缺点，以此获得更好的算法性能。目前尝试较多的算法有模拟退火算法、遗传算法等。

下面以粒子群优化算法对 Rosenbrock's 测试函数进行目标范围内的最小值寻优为例对粒子群优化算法进行说明，该测试函数的计算公式在 3.2.6 节中已介绍，其取值范围为[−10,10]，取值范围内的理想最优解为 0，将其搜索的空间维度设为 20。

在参数设置方面，将粒子群优化算法的种群数量设为 30，惯性因子 w 设为 0.9，两个加速因子 c_1、c_2 均设为 2，迭代次数设为 1000。除此之外采用基于线性下降惯性因子的粒子群优化算法(Particle Swarm Optimization Algorithm with Linear Descent of Inertia Factor，IPSO)与动态多种群粒子群优化算法(Dynamic Multi-Swarm Particle Swarm Optimizer，DMSPSO)进行对比实验。

随迭代次数线性下降，基于线性下降惯性因子的粒子群优化算法与传统粒子群优化算法的唯一区别在于该算法对其惯性因子采用对迭代次数的增加而不断线性下降的方式进行设置，惯性因子的初始值与传统粒子群优化算法一样，设为 0.9，当迭代次数结束时，惯性因子的大小减小至 0。

动态多种群粒子群优化算法主要将多种群的思想引入粒子群优化算法中，在最初对种群进行设置时，将所有粒子分成多个种群，在对粒子位置更新时，更新方式不变，但公式(3.90)中需将对全局历史最优位置信息改成对小种群中的全局历史最优位置进行学习，而全局历史最优位置信息通过对不同的种群之间进行比较而获得，另外会设定一个固定的迭代周期，当达到此迭代次数时会对每个种群中的个体进行重组，即将所有个体进行混合，然后随机分成之前数量的种群，在接下来的迭代周期中以新的种群进行位置的学习移动。在参数设置方面，除与传统粒子群优化算法相同的参数设置外，将种群的数量设为 5，对种群进行重组的更新周期设为 10。

主要的寻优过程如下：首先在此函数的取值范围[−10,10]内随机生成 30 个 20 维数组作为种群个体的位置信息，在动态多种群粒子群优化算法上将其分成 5 个种群，其他算法为 1 个，然后根据测试函数计算公式计算个体的适应度值，将适应度值由小到大的顺序进行排序，从而确定种群历史最优信息以及个体历史最优信息，之后再对个体位置信息进行更新并计算更新后的适应度，同时需对种群历史最优信息以及个体历史最优信息进

行更新，最后重复相关操作，当达到最大停止迭代次数 1000 时停止迭代，输出此时的种群历史最优个体适应度值及其位置信息。

在优化过程中，种群的历史最优适应度值的变化情况如图 3.16 所示。

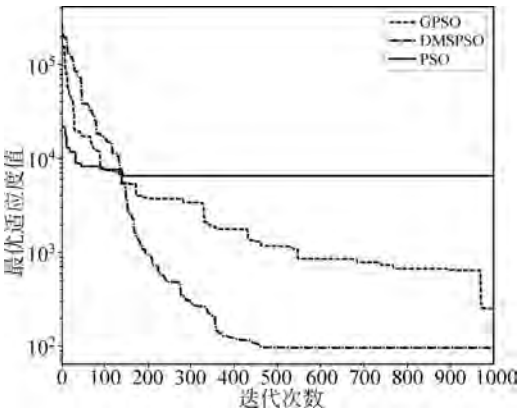

图 3.16　历史最优适应度值的变化情况

由图 3.16 可以看到，种群历史最优适应度值的变化速度一直在发生变化，而不是以相同的速度在减小。在迭代次数低于 100 时，传统粒子群优化算法相较于另外两种算法能够以更快的速度到达一个较优值，这可能是由于传统粒子群优化算法相较于另外两种算法步骤更加简单，同时采用固定值惯性因子，因此前期的全局寻优能力更强。当迭代次数为 100～500 次时，传统粒子群优化算法的种群历史最优适应度值没有大的变化趋势，说明此时可能已经陷入局部最优，采用线性下降惯性因子的粒子群优化算法在这个迭代周期之间的种群历史最优适应度值的下降速度相对较为稳定，而动态多种群粒子群优化算法在这个迭代周期内以一个较快的下降速度迅速将种群历史最优适应度值下降到三种算法中的最低值。最后在迭代 500 次到迭代结束之间，传统粒子群优化算法的历史最优适应度值基本保持不变，采用线性下降惯性因子的粒子群优化算法的种群历史最优适应度值继续下降，但下降速度开始减小，虽然在迭代次数为 950 次左右出现了一次大的下降幅度，但其种群历史最优适应度值依旧低于动态多种群粒子群优化算法，直至迭代结束，动态多种群粒子群优化算法具有最低的种群历史最优适应度值。综上所述，无论是从算法的收敛速度还是从算法的收敛精度来比较，动态多种群粒子群优化算法要比另外两种对比算法更好，说明动态多种群粒子群优化算法相较于传统粒子群优化算法所做的改进是合理且有效的。

3.5.3　模拟退火算法

模拟退火（Simulated Annealing，SA）算法是一种基于蒙特卡洛迭代求解策略的随机优化算法，其思想起源是 1953 年由 Metropolis 等人提出的。直到 1983 年，由于软硬件条件的成熟，Kirkpatrick 等人才成功地将模拟退火思想引入组合优化领域。由于模拟退火算法在理论上具有概率全局最优化的能力，因此它在工程控制、生产调度、信号处理、机器学习等诸多领域都得到了广泛的应用。

模拟退火算法的中心思想与现实生活中的金属固体退火降温的原理直接相关。在现实生活中对金属固体进行加热时，金属固体的内能会随着温度的升高而不断增大，当达到一定温度时，金属开始熔解，而内部的粒子则会表现得极其活跃，同时呈现出无序的状态，当不再进行加热时，温度逐渐降低，金属开始冷却变成固体，粒子则会由之前无序的状态逐渐恢复到排列有序的状态，此时系统的能量也会逐渐减少，直至达到最低值。在此降温过程中，温度越高，出现降温的概率则越大，随着温度的不断降低，出现降温情况的概率也会随之不断减小。

模拟退火算法中涉及的概念主要有三个，分别为目标函数、接受概率以及冷却进度表。下面将分别针对这三个概念展开介绍。

1. 目标函数

由上述思想可以得知，模拟退火算法的整个降温过程可以类比于在目标范围内求取最优解的过程，将金属固体的内能类比于目标函数值，当温度这个控制参数变化时，金属固体的内能也在不断改变，此时通过一定概率判断是否接受这个新的内能，即最优解。因此模拟退火算法可以广泛应用于优化问题的求解，通过目标范围内不同的解之间的比较而不断搜索最优解的具体信息。

2. 接受概率

接受概率即为判断是否接受新的内能的概率。模拟退火算法中通常使用 Metropolis 接受准则进行接受概率的计算。在此准则中，倘若内能朝着想要的趋势发生变化(若为求解最小值，则 $\Delta E<0$；若为求解最大值，则 $\Delta E>0$)，则接受新的状态，否则将使用概率 $\exp\left(-\frac{\Delta E}{kT}\right)$ 来判断是否接受新的状态，其中 k 为 Boltzmann 常数，通常取值为 1.3806×10^{-23}。

3. 冷却进度表

从前面两个概念的介绍中可以得知，在每个状态的变化过程中，温度值起到至关重要的作用，尤其是 Metropolis 接受准则中是否接受新的状态的概率直接与温度值相关，而由前面的算法思想介绍可以得知，温度应当处于一个不断降低的状态，即温度越低时，出现内能改变的可能性也会降低，因此在这里采用冷却进度表来控制温度的变化。冷却进度表是指用来控制从某高温状态 T_0 开始进行降温的幅度和趋势的管理表。假设 t 时刻的温度为 T_0，那么在经典的模拟退火算法中，温度随着时间的变化趋势可以表示为 $\frac{T_0}{\lg(1+t)}$，而在快速模拟退火算法中，温度随着时间的变化趋势可以表示为 $T_0/(1+t)$，二者的区别主要表现在变化的速度上，但它们都可以帮助算法有效收敛到全局最优值处。另外在实际的应用中，时刻 t 可以使用迭代次数来代替。

在上述模拟退火算法原理中主要涉及的参数分别为初始温度 T 和温度下限 $T_{\min}$(此值主要控制温度的最低值，当达到此值时将不再进行状态的改变)。

结合上述相关算法原理，可以将使用模拟退火算法进行优化问题求解时的特征总

结如下。

(1) 模拟退火算法在对目标函数进行求解时,其搜索的范围可以是多维的。

(2) 模拟退火算法中内能的最理想状态即为目标范围内的最优解对应的目标函数值。

(3) 在进行内能的变化时,算法的搜索采用的是随机搜索,即具有更大搜索到较优解的可能性。

(4) 使用接受概率来判断是否接受新的状态时,大部分情况下获得的新状态较前一状态都更优一些,即状态在朝着更好的方向不断变化。

(5) Metropolis 接受准则可以帮助算法加快收敛速度,获得更高的收敛精度。

(6) 随着温度的减小,即迭代次数的增加,在新状态较差的情况下接受新状态的概率不断减小。

(7) 前期温度较高,较大的接受较差状态的概率可以帮助算法扩大搜索范围;后期温度较低,较小的接受较差状态的概率有利于提高算法的收敛速率。

(8) 该算法主要进行调节的参数为初始温度 T 以及温度下限 $T_{\min}$,算法简单易行。

使用模拟退火算法对优化问题进行求解时的具体步骤可以归纳如下。

(1) 设置初始温度 T、温度下限 $T_{\min}$、初始的解状态、每个温度 T 值下的迭代次数 L 以及算法停止迭代条件(一般设为当连续多个新解都没有被接受时停止算法)。

(2) 根据目标函数,对初始解对应的目标函数值进行计算。

(3) 对初始的解进行变换,产生一个目标范围内的新解,通常采用简单的变换产生新解,例如对旧解中的全部或部分元素进行置换、互换与随机数相加等。

(4) 根据目标函数,对新解对应的目标函数值进行计算。

(5) 通过新解与旧解的目标函数值计算它们之间的差值,然后使用 Metropolis 接受准则判断是否接受新解,若接受,则舍弃旧解,使用新解替代旧解。

(6) 重复上述步骤(2)~步骤(5),直至达到迭代次数 L。

(7) 判断此时是否满足停止迭代条件,若满足则输出此时的解;若不满足,则使用冷却进度表对温度进行更新。

(8) 对温度进行更新后判断是否达到温度下限值,若达到则输出此时的解,若未达到,则重复步骤(6)~步骤(7)。

(9) 当满足停止迭代条件或温度达到温度下限值时,停止算法寻优,输出此时的解状态及其对应的目标函数值。

以求解最小值问题为例,按照上述过程可将模拟退火算法归纳为如算法 3.13 所示。

算法 3.13 模拟退火算法

```
1:  初始化解状态、温度 T 以及温度下限 T_min
2:  根据待求解问题初始化解状态对应的目标函数值
3:  while(未满足停止迭代条件)do
4:    while(温度 T 大于温度下限 T_min)do
5:      for i=1: L
```

```
        对旧解进行更新获得新解
        计算新解对应的目标函数值
        if ΔE<0
          接受新解
        else
          以 Metropolis 接受准则判断是否接受信息
        end if
      end for
      对温度进行更新
    end while
end while
输出此时的解状态及对应的目标函数值
```

在模拟退火算法中主要存在两个问题容易影响算法的有效性，分别为温度初始值 T 的设定以及温度降低的速度。在温度初始值 T 的设定方面，一般来说温度是由较高值开始逐渐降低到较低值，当初始值设置较高时，解状态更新后接受新解的概率将相对较高，此时有助于对目标范围进行全局搜索，但与此同时，算法的整个迭代次数将会随着增加，算法的效率也会随着降低，而当初始值设置较低时，搜索到全局最优解的可能性也会随之降低。温度降低的速度问题从根本上来说与温度初始值 T 的设定这一问题是存在关联的，它同样是对算法的全局搜索效率造成直接影响。温度降低的速度将直接影响不同温度状态的持续时长，由前面的算法原理可以得知在温度较高时，接受新解的概率会处于一个较大值，此时有助于全局搜索。因此，当温度降低较快时，高温度持续的时间较短，进行全局搜索的时长也会相对较短；而当温度降低较慢时，算法的迭代时长也会随之增加，将花费大量的计算时间进而降低算法效率。因此，如何选择一个最优的初始温度值以及合适的温度降低速度是目前研究者对模拟退火算法进行改进的重点。

3.5.4　蚁群优化算法

蚁群优化(Ant Colony Optimization，ACO)算法的基本理论和设计思想可以简单描述为：在整个自然界中，蚂蚁搜寻某种食物时并非仅仅是单只蚂蚁的主动搜寻，而是一种非常具有高度群体性的搜索行为，当使用蚁群觅食时，每只蚂蚁之间都可能存在着互相传递食物信息的行为。

假定各地的蚂蚁都在没有事先通过信息素告知任何食物来源所在处的情况下，从零开始寻找新的食物。当其中的一只蚂蚁搜寻到一个新的食物源之后，这只蚂蚁就会向它的周围环境不断释放一种具有一定挥发性的分泌物质，这种物质被人们称为信息素，该种挥发性分泌物质随着周围空气的流通和时间的推进而逐渐挥发并最终逐渐消失，信息素气体浓度的水平高低分别代表了当前位置与食物源位置距离的远近，通过这种方式可以吸引其他蚂蚁沿着它的原始觅食路径方向前进，这样会导致越来越多的蚂蚁依次聚集在同一条通往食物的路线上，最终依次沿着此条行进路线前进，最终可以找到这个食物源。然而，万事都可能存在差异，总会有一些蚂蚁并不会按照其他蚂蚁的觅食路径前进，它们

有可能会另辟蹊径，按照自己的方式进行食物的搜寻，与此同时自己也会分泌并释放出大量的信息素，经过很长一段距离后，该蚂蚁找到之前蚂蚁搜寻到的食物。假如这条觅食路径和原来的觅食路径相比要短，则越来越多的蚂蚁被食物吸引而走到了这条较短的觅食路径上去。最后，经过了长时间的演练，大多数蚂蚁都会在最短的觅食路径上，因此，这样的过程就是正反馈的过程。

在优化问题中，首先构造解空间，其中将蚂蚁的行走路径表示为待优化问题的可行解，那么，整个蚂蚁群体的所有路径构成待优化问题的解空间。其次，蚂蚁依照信息素进行路径搜索，随着时间的推进，大多数蚂蚁会沿着信息素较多的路径前进，这条路径也许是最短的路径。在正反馈的作用下，最终，这条最短路径便是待优化问题的最优解。

蚁群优化算法中的相关规则可以总结归纳如下。

1. 视野范围

通常蚂蚁感受的范围很小，一般假设蚂蚁能感受的范围为边长为 1cm 的方格。

2. 环境

蚂蚁所处的环境是随机的、多变的。比如环境中有障碍物、蚂蚁、信息素等，蚂蚁只能感受在它范围内的环境信息，蚂蚁在寻找食物的过程中，主要依据信息素分泌进行前进，然而信息素又以一定的概率消失。

3. 觅食规则

蚂蚁在觅食的过程中能感受到在其范围内的环境，如果食物在范围内，则蚂蚁就直接找到了食物，否则，蚂蚁依据信息素前进。蚂蚁在寻找食物的过程中都是往信息素较多的方向前进，但是，有些蚂蚁并没有按照此前进，如果这些蚂蚁搜索的路径更短，则会吸引其他蚂蚁过来，最终，形成一条最短路径。总之，觅食过程是一个正反馈的过程。

4. 移动规则

蚂蚁移动时是按照信息素较多的方向前进的，如果在其范围内，无信息素分泌，则按照原来的方向继续前进，一旦改变方向，则选取此时能察觉的信息素较多的方向前进。

5. 避障规则

蚂蚁会按照觅食规则进行前进，一旦前方有障碍物，则按照信息素的多少，改变方向，避开障碍物。

6. 信息素规则

在寻找食物之前，蚂蚁之间并没有直接的关系，但是每只蚂蚁都和环境发生交互，蚂蚁在搜索的过程中就会分泌信息素，进而蚂蚁之间就根据信息素关联了起来。

下面以经典的旅行商问题为例对蚁群优化算法的基本算法流程总结如下。

(1) 初始化所有路径上的信息素浓度为 0；

(2) 在 n 个城市中随机选取其中 m 个城市作为 m 只蚂蚁的出发城市,并把每只蚂蚁当前所在的城市 x 写入该蚂蚁的已访问城市列表;

(3) 如果第 $k(k=1,2,\cdots,m)$ 只蚂蚁还有未被访问的城市,则该只蚂蚁根据概率 p_{ij} (p_{ij} 表示城市 i 到城市 j 的信息素浓度值与城市间距离生成的函数)选出下一个将要访问的城市 j,并把 j 写入该蚂蚁的已访问城市列表,直至所有城市都被访问过一次;

(4) 计算第 k 只蚂蚁爬过的路径的长度,找出其中的最短路径;

(5) 根据当前最短路径中的城市访问情况更新路径上每条线段的信息素浓度,并清空该蚂蚁的已访问城市列表;

(6) 重复上述步骤(2)～步骤(5),当算法满足停止迭代的条件时停止迭代并输出当前最优解。

蚁群优化算法是一种仿生进化算法,从生物学原理可以看出,蚂蚁搜索食物的行为是一种自催化行为,它们之间是靠彼此分泌的信息素进行关联的,初始阶段,由于每条路径上的信息素分布都是均匀的,每只蚂蚁都以相等的机会选择路径,当蚂蚁经过某个路径段时都会分泌信息素,随着信息素的增多,某个路径信息素最多的路径将会吸引更多的蚂蚁过来,然而也存在一些不按照已经分布的信息素选择路径而是选择其他的路径的情况,假如这条路径上的信息素比原来路径的信息素多,那么,经过一段时间后,该路径上的蚂蚁数量会更多,其所经过的路径就是最短的路径,因此这种蚂蚁搜索过程就是一种正反馈的过程。并且蚁群优化算法具有较强的健壮性。因为蚁群优化算法设置简单,只要对其模型略加改动,就可以用于其他问题的求解。在蚁群优化算法中,蚂蚁的位置移动完全依靠路径上的信息素浓度和路径长度(即启发信息),而这种移动规律正是一种概率性的状态转移规则,它能够提高算法的全局搜索能力,不仅如此,蚁群优化算法易与其他算法如遗传算法相结合,扬长避短,提高传统蚁群优化算法的搜索能力和收敛速度。

但是,蚁群优化算法有两个明显的缺点。第一个缺点是蚁群优化算法的搜索时间较长。蚁群中单只蚂蚁的运动是随机的,它们在路径搜索过程中释放信息素并通过这种化学物质相互沟通最终发现最短路径,但当问题的规模较大时,要想在较短时间内从大量杂乱无章的路径中找出一条较优路径是很难的,这是因为在路径搜索的初始时刻,各条路径上的信息量几乎相同,随着搜索的进行,由于正反馈作用,使得较优路径上的信息量逐渐增加;另外,由于信息素会随着时间的流逝逐渐挥发掉,使得较差路径上的信息素越来越少,经过很长一段时间,这种差距越来越明显,最终绝大多数蚂蚁都选择这条信息量大的路径,即较优路径,而这一寻优过程一般需要较长的时间。

第二个缺点是蚁群优化算法易陷入局部最优。在蚁群优化算法中,蚂蚁的方向选择是依靠各路径上的信息量和启发信息(即路径长度)的,它们的运动总是倾向于信息素多的路径。但初始时刻,由于所有的路径上都没有信息素,这样蚂蚁构建的第一条路径主要是由路径长度来引导的,同时蚂蚁会在其所走过的路径上释放出与路径长度有关的信息素,而这条路径不一定能代表全局最优路径的方向,无法保证蚂蚁构建的第一条路径能指引蚁群走向全局最优路径,但随着时间的推移,由于正反馈作用,最终所有的蚂蚁都选择了这条局部最优路径,导致算法陷入局部最优,即出现“早熟”现象。

3.5.5 常见的反向传播算法

在前面对小波神经网络的反向传播过程进行阐述时，介绍了自适应梯度算法与均方根反向传播算法，这两种算法都是反向传播算法中的经典算法。反向传播算法的提出与发展由来已久，研究者也提出了很多有效的算法，这些算法之间也存在着一定联系，下面将对部分常见的反向传播算法进行介绍，最后也通过比较在使用不同反向传播算法时小波神经网络的损失值在迭代过程中的变化情况来对比这些反向传播算法的实际性能。

1. 弹性反向传播算法

在均方根反向传播算法提出之前，曾提出过一种弹性反向传播(Resilient Back Propagation，RProp)算法，此算法主要是针对反向传播算法中学习率的选择困难这一问题而提出的。

在传统反向传播算法中，学习率通常会被设置成一个固定值，但是具体将这个值设置成多少则是难以确定的，一旦选择不合理则会导致学习效果较差；另外，一般会针对所有的参数设置同一个学习率，但是在实际问题中，不同参数的梯度值基本上不会出现相同的情况，因此采用同一个学习率对不同的参数进行优化会导致不同参数的收敛速度完全不同，进而直接影响算法的收敛效果。

为避免出现上述问题，弹性反向传播算法针对每个参数都设置了专门对应此参数的学习率，在整个迭代过程中学习率并未设为固定值，而是会针对每次迭代过程中梯度值的变化情况而不断变化，其具体的公式如下：

$$\mu_t=\begin{cases}\mu_1 * \mu_{t-1}, & g_{t-1}g_t>0\\ \mu_2 * \mu_{t-1}, & g_{t-1}g_t<0\\ \mu_{t-1}, & \text{其他}\end{cases} \tag{3.91}$$

在上述公式中，μ_t 表示算法迭代到第 t 次时参数的学习率；μ_{t-1} 表示算法迭代到第 $t-1$ 次时参数的学习率；μ_1 表示预先设置的学习率增加倍数；μ_2 表示预先设置的学习率减少倍数；g_t 表示算法迭代到第 t 次时参数的梯度值；g_{t-1} 表示算法迭代到第 $t-1$ 次时参数的梯度值。

传统反向传播算法对参数的更新量主要是通过梯度与学习率的计算获得的，但是弹性反向传播算法则是直接将学习率作为参数的更新量，而梯度值主要用于对参数变化方向的调整。正如公式(3.91)所示，当迭代前后参数的梯度相乘为正数，即梯度方向未发生变化，则说明此时距离最小点处仍有一定的距离，因此不需要改变参数的调整方向，同时需要对参数调整的大小进行增加；当迭代前后参数的梯度值相乘为负数，即梯度方向发生了变化，则说明此时有可能已经跳过了最小点处，因此需要对参数的调整方向进行改变，同时为避免再次错过最小点处，需要降低参数的调整大小。具体的调整公式如下：

$$\Delta_t=\begin{cases}-\mu_t, & g_t>0\\ \mu_t, & g_t<0\\ 0, & \text{其他}\end{cases} \tag{3.92}$$

综合上述原理，可以得知弹性反向传播算法的主要算法步骤为：首先设置每个参数对应的学习率的初始值，并计算保存此次每个参数所对应的梯度值，然后对每个参数都使用学习率初始值进行更新，同时计算保存每个参数所对应的更新后的梯度值，之后利用公式(3.91)与公式(3.92)对每个学习率及其对应的参数进行更新，最后重复上述参数更新步骤，直至达到最大迭代次数时输出此时每个参数的值。

将弹性反向传播算法应用于神经网络参数优化时的算法(即 RProp 算法)如算法 3.14 所示。

算法 3.14　RProp 算法

```
随机初始化神经网络中所有的连接权值、阈值，设置学习率初始值等参数
while(未满足停止迭代条件)do
  for 所有输入数据
    按照前向传播过程计算网络的输出
    使用损失函数计算误差值
    根据误差值计算出隐含层、输出层每个参数的梯度项
    利用梯度项对学习率进行更新
    利用学习率对参数进行更新
  end for
end while
输出隐含层与输出层参数确定的多层前馈神经网络
```

2. 自适应学习率法

自适应学习率法(Adaptive Learning Rate Method，Adadelta)可以视作自适应梯度法的一种扩展。自适应梯度法在迭代过程中累加了过去所有的梯度平方，而自适应学习率法与均方根反向传播算法类似，在这一阶段引入了衰减系数来对梯度累积平方公式进行调整。但是与均方根反向传播算法相比，这两种算法在学习率的设置上也有很大区别，均方根反向传播算法主要使用全局学习率对参数调整步长进行控制，而在自适应学习率法中，使用了专门的移动步长项来代替全局学习率。

自适应学习率法的累积梯度的计算方式与均方根反向传播算法相同，这里不再进行介绍，其移动步长项的计算公式如下：

$$\theta_{t+1}=\rho\theta_t+(1-\rho)v_t^2 \tag{3.93}$$

其中，θ_{t+1} 表示第 $t+1$ 次迭代时参数的移动步长值；θ_t 表示第 t 次迭代时参数的移动步长值，此移动步长的初始值为 0；ρ 为衰减系数；v_t 表示第 t 次迭代时参数的更新量。

对参数进行更新时，具体更新量的计算公式如下：

$$\Delta_t=\sqrt{\frac{\theta_{t+1}}{r_{t+1}+\varepsilon}}g_{t+1} \tag{3.94}$$

其中，r_{t+1} 表示第 $t+1$ 次迭代时的累积梯度值；g_{t+1} 表示第 $t+1$ 次迭代时的梯度值；ε 为避免出现分母为 0 的情况而设置的极小值。

将自适应学习率法应用于神经网络参数优化时的算法(即 Adadelta 算法)如算法 3.15 所示。

算法 3.15 Adadelta 算法

```
随机初始化神经网络中所有的连接权值、阈值,设置移动步长初始值等参数
while(未满足停止迭代条件)do
  for 所有输入数据
    按照前向传播过程计算网络的输出
    使用损失函数计算误差值
    根据误差值计算出隐含层、输出层每个参数的梯度项
    利用梯度项计算累积梯度平方
    对移动步长进行更新
    利用移动步长与累积梯度平方对每个参数的更新量进行计算
    对每个参数进行更新
  end for
end while
输出隐含层与输出层参数确定的多层前馈神经网络
```

按照上述相关说明,可以将自适应学习率法的算法步骤归纳如下:

(1) 设置初始移动步长、衰减系数等参数;

(2) 通过损失函数计算获得当前梯度值;

(3) 使用当前梯度值对累积梯度平方进行更新;

(4) 通过公式(3.93)对移动步长进行更新;

(5) 通过公式(3.94)对参数的更新量进行计算;

(6) 对参数进行更新;

(7) 重复上述步骤(2)~步骤(6),当达到最大停止迭代次数时停止迭代,输出此时的参数值。

3. 自适应矩估计算法

自适应矩估计(Adaptive Moment Estimation,Adam)算法从其本质上看可以视作带有动量项的均方根反向传播算法,一方面,它使用动量法中的方式来进行参数历史梯度的累积,从而更好地利用历史信息;另一方面,它利用梯度的一阶矩估计和二阶矩估计来动态调整每个参数的学习率,在获得更快收敛速度的同时使得波动的幅度更小。

在自适应矩估计算法中,主要涉及的参数有学习率 μ、衰减率 ρ_1 与 ρ_2、时间步长 t 以及避免出现分母为 0 的情况而设置的极小值 ε。其中,衰减率 ρ_1 与 ρ_2 为分别针对梯度的一阶矩估计和二阶矩估计而设置,其计算公式分别如动量法中的累积梯度计算方式以及均方根方向传播算法中的累积梯度平方计算方式完全一致,在算法最初将一阶矩估计和二阶矩估计均初始化为 0。

在更新对梯度进行一阶矩估计与二阶矩估计之后,需要对这两个值进行偏差修正,对

一阶矩的修正方式如下：

$$s'_t=\frac{s_t}{1-\rho_1^t} \tag{3.95}$$

其中，s'_t为修正后的一阶矩；s_t 为修正前的一阶矩；t 作为时间步长，随着迭代次数的增加而增加，一般与迭代次数相同。对二阶矩的修正方式与一阶矩完全相同，使用 r'_t表示修正后的二阶矩，r_t 表示修正前的二阶矩。

以修正后的一阶矩与二阶矩作为计算依据对参数的更新量进行计算，具体的计算方式如公式(3.96)所示。

$$\Delta_t=-\mu\frac{s'_t}{\sqrt{r'_t}+\varepsilon} \tag{3.96}$$

按照上述相关说明，可以将自适应矩估计算法的步骤归纳如下：

(1) 设置初始一阶矩、二阶矩、全局学习率以及衰减系数等参数；

(2) 通过损失函数计算获得当前梯度值；

(3) 对时间步长进行计算；

(4) 使用当前梯度值对累积梯度更新以进行一阶矩估计；

(5) 使用当前梯度值对累积梯度平方更新以进行二阶矩估计；

(6) 对一阶矩与二阶矩进行偏差修正；

(7) 通过修正后的一阶矩与二阶矩对参数的更新量进行计算；

(8) 对参数进行更新；

(9) 重复上述步骤(2)～步骤(8)，当达到最大停止迭代次数时停止迭代，输出此时的参数值。

自适应矩估计算法的主要特点在于它在善于处理稀疏梯度的同时能够很好地处理非平稳目标，而在其对参数更新量的计算过程中也可以视作对不同的参数计算了不同的自适应学习率，因此它适用于大多数非凸函数优化问题。此外，该算法对于较大的数据集以及在高维空间范围内也能够取得较好的优化效果。

自适应矩估计算法应用于神经网络参数优化时的算法(即 Adam 算法)如算法 3.16 所示。

算法 3.16　Adam 算法

```
随机初始化神经网络中所有的连接权值、阈值，设置全局学习率、初始一阶矩、二阶矩以及衰减系数等参数
while(未满足停止迭代条件)do
    for 所有输入数据
        按照前向传播过程计算网络的输出
        使用损失函数计算误差值
        根据误差值计算出隐含层、输出层每个参数的梯度项
        利用梯度项计算累积梯度平方、累积梯度
        分别进行一阶矩估计与二阶矩估计
```

```
9：        对一阶矩与二阶矩进行参数修正
10：       对每个参数的更新量进行计算
11：       对每个参数进行更新
12：    end for
13：  end while
14：  输出隐含层与输出层参数确定的多层前馈神经网络
```

另外，在自适应矩估计算法的基础上还进一步提出了两种变形：一种变形的英文名称为 Adamax；另一种变形的英文名称为 Nadam。

Adamax 主要对最后参数的更新量计算公式进行了调整，原式中的分母部分为对二阶矩估计进行开方，修改后采用了下列公式进行该部分的确定：

$$r_t = \max(\rho_2 r_{t-1}, |g_t|) \tag{3.97}$$

之后采用下列公式对参数的调整量进行计算：

$$\Delta_t = -\mu \frac{s'_t}{r_t + \varepsilon} \tag{3.98}$$

此算法的主要特点在于为学习率提供了一个更加简单的上限范围。

Nadam 相对 Adamax 来说更加简单，此算法相当于将 Nestrov 动量法的临时梯度思想引入了自适应矩估计算法中，在每次对梯度进行计算时获得一个临时梯度，以此临时梯度对一阶矩与二阶矩进行估计，并使用临时一阶矩与二阶矩来计算参数的更新量。

上述几种反向传播算法在对使用同一数据集的前向神经网络进行参数优化时优化结果也会存在一定的差异，为了更清晰地了解它们之间的区别，下面以使用不同反向传播算法的小波神经网络为例进行对比实验说明。

在本实验中选择武汉市六个新城区的农田土壤重金属数据集作为实验数据，该实验数据共有 500 组，其中数据的输入特征共有四种，输出特征为一种。对小波神经网络结构进行设置时，分别将其输入层节点数、隐含层节点数以及输出层节点数设为 4、8、1。对参数的初始化方式选择 Xavier 均匀分布初始化，损失函数选择均方误差函数，训练次数设为 100 次。对几种反向传播算法的参数设置如表 3.3 所示。

表 3.3　反向传播算法的参数设置

算　法	参 数 设 置
Adam	$\mu=0.01,\rho=0.9,\varepsilon=10^{-6}$
Adamax	$\mu=0.01,\rho=0.9,\varepsilon=10^{-6}$
Nadam	$\mu=0.01,\rho=0.9,\varepsilon=10^{-6}$
Adagrad	$\mu=0.01,\varepsilon=10^{-6}$
Adadelta	$\rho=0.9,\varepsilon=10^{-6}$
RMSProp	$\mu=0.01,\rho=0.9,\varepsilon=10^{-6}$

使用上述几种不同的反向传播算法进行小波神经网络参数优化时，神经网络损失值的变化情况如图 3.17 所示。

由图 3.17 可以看到，在这六种算法中，代表均方根反向传播算法（RMSProp）的曲线

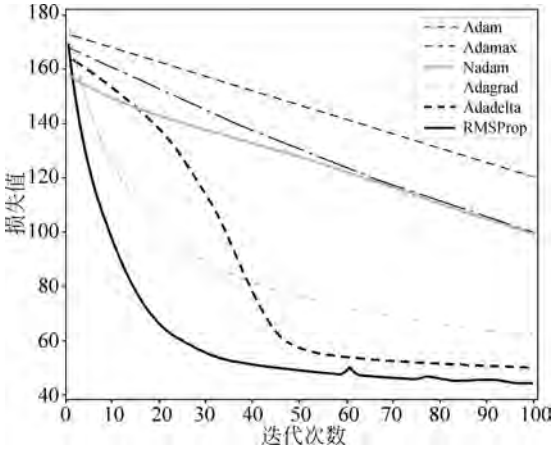

图 3.17　神经网络损失值的变化情况

相较于其他曲线是最先到达一个较低点的，即该算法的收敛速度是最快的，其次是自适应梯度算法（Adagrad）先到达一个较低点，但是其收敛速度却以较快的速度在减小，而代表自适应学习率法（Adadelta）的曲线的变化量在逐渐增大，因此在迭代 40 次左右时，自适应学习率法的收敛速度已经快于自适应梯度算法，这可能是由于自适应学习率法相较于自适应梯度算法而言在累积梯度中引入了衰减系数的原因。自适应矩估计算法（Adam）及其两种扩展算法 Adamax 以及 Nadam 在整个迭代过程中保持一个较为稳定的收敛速度，每次迭代损失值的变化量也较为平均。另外从整体的最低损失值来看，自适应矩估计算法的两种扩展算法的损失值曲线在迭代过程中始终低于它，说明这两种改进均是有效的，而自适应梯度算法与自适应学习率算法可能更适用于此数据集，因此它们的损失值曲线要低于前三种，均方根反向传播算法的损失值曲线全程明显低于另外五种算法，即其收敛效果较另外五种算法更好。综上所述，在对应用此数据集的小波神经网络进行训练时，均方根反向传播算法的收敛速度与收敛精度都要优于其他五种反向传播算法。

除了比较使用这几种算法优化参数时的收敛情况外还统计了它们在同一神经网络上迭代次数为 100 次时所需的时间，结果如表 3.4 所示。

表 3.4　算法运行时间对比

算　法	运行时间/s	算　法	运行时间/s
Adam	14.6175	Adagrad	18.1425
Adamax	18.1301	Adadelta	17.3487
Nadam	19.1805	RMSProp	13.8038

由表 3.4 可以看到，自适应矩估计算法的两种扩展算法虽然优化效果较其有所提高，但是其运行时间却相较原算法有所增加，自适应学习率法从步骤上较自适应梯度算法来看并无大的变动，因此其运行时间并无较大的变化，而这几种算法中运行时间最短的是均方根反向传播算法。所以结合前面的对比情况可以得出结论：在对应用此数据集的小波神经网络进行训练时，上述六种反向传播算法中，均方根反向传播算法是最合适的。

3.6 本章小结

本章介绍了一种用于大数据预测的协作复合神经网络模型的基础架构，此协作复合神经网络模型主要由自适应动态灰狼优化算法与小波神经网络组合而成，而本章的内容则主要分为四部分，分别为自适应动态灰狼优化算法、小波神经网络模型、协作复合神经网络模型的构建以及相关知识扩展。

在自适应动态灰狼优化算法部分，首先对传统的灰狼优化算法展开介绍，包括其自身存在的问题，同时针对这些问题分别采用了非线性余弦收敛因子、加权位置更新方式以及中心扰动准则来对原算法进行优化，进而进一步提出了自适应动态灰狼优化算法。最后将自适应动态灰狼优化算法与遗传算法、粒子群优化算法以及灰狼优化算法进行算法的性能对比实验，实验结果中无论是从收敛速度还是收敛精度来看，自适应动态灰狼优化算法的优化效果都要比另外几种对比算法要好。

在小波神经网络模型部分，按照小波神经网络模型的建模过程依次对神经元节点处的激活函数、网络的前向传播过程、参数的初始化方式、针对不同问题的损失函数以及反向传播算法进行了介绍，其中在反向传播算法部分主要介绍均方根反向传播算法、动量法以及 Nesterov 动量法。此外由于对协作复合神经网络模型的构建需要以小波神经网络模型作为基础模型，因此对小波神经网络中激活函数、参数初始化方式、损失函数以及反向传播算法的选择进行了具体的说明。

之后的协作复合神经网络模型的构建部分主要围绕协作复合神经网络模型的建模过程展开介绍，包括如何使用自适应动态灰狼优化算法对小波神经网络的初始参数进行生成、参数具体的训练过程以及此模型的具体建模步骤等，并给出了协作复合神经网络模型的流程图。

最后扩展介绍了在前面部分所提到的几种算法，包括用于和自适应动态灰狼优化算法进行对比实验的遗传算法、粒子群优化算法以及同时期提出的模拟退火算法与蚁群优化算法，反向传播算法中常用的弹性反向传播算法、自适应学习率法以及自适应矩估计算法等。

另外在对遗传算法与粒子群优化算法进行介绍时，使用了求解最大最小寻优问题的方式对其原理进行辅助说明，而在对反向传播算法进行介绍时，则是通过训练小波神经网络参数的方式对这几种算法的性能进行了对比分析，分析的过程主要围绕神经网络损失值的变化情况以及算法的运行时间两方面进行。分析结果表明，在使用小波神经网络进行此土壤重金属数据集的训练时，均方根反向传播算法具有最好的收敛效果。