计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190800159

基于差分进化的推断任务卸载策略

王 瑄 毛莺池 谢在鹏 黄 倩
河海大学计算机与信息学院 南京 211100
(xuanwang@hhu.edu.cn)

摘 要 卷积神经网络(Convolutional Neural Network,CNN)作为深度学习的重要技术,已被广泛应用在移动智能应用中。针对CNN推断任务高内存、高计算量的需求,现有解决方案多将任务卸载到云上执行,难以适应时延敏感的移动应用程序。为解决上述问题,提出了一种基于改进差分进化算法的CNN推断任务卸载策略,它采用端云协作模式将计算任务部署在云和边缘设备之间。该策略研究了成本约束下最小化时延的任务卸载方案,将CNN推断过程转化为任务图并将其构建为0-1整数规划问题,利用改进二进制差分进化算法高效求解最佳卸载决策。实验结果表明,在给定费用约束下,与移动端推断和云推断方案相比,所提策略将任务响应时间平均缩短了33.60%和6.06%。

关键词:卷积神经网络;移动云计算;计算卸载;协同推断;差分进化算法
中图法分类号 TP393

Inference Task Offloading Strategy Based on Differential Evolution

WANG Xuan,MAO Ying-chi,XIE Zai-peng and HUANG Qian
School of Computer and Information,Hohai University,Nanjing 211100,China

Abstract As an important technology of deep learning,convolutional Neural Network (CNN) has been widely used in intelligence applications. Due to the demand of CNN inference task for high computer memories and computation,most of the existing solutions are to offload tasks to the cloud for execution,which are hard to adapt to the time-delay sensitive mobile applications. To solve the above problem,this paper proposes a CNN inference task offloading strategy based on improved differential evolution algorithm,which can efficiently deploy computing tasks between cloud and edge devices using end-cloud collaboration mode. This strategy studies the task unloading scheme that minimizes the time delay under cost constraint. transforms the CNN inference process into a task graph and constructs it into a 0-1 integer programming problem,and finally uses the improved binary differential evolution algorithm to solve the problem so as to infer the optimal offloading policy. The experimental results show that,compared with mobile inference and cloud inference schemes,averagely,the proposed strategy can reduce the task response time by 33.60% and 6.06% respectively with cost constraints.

Keywords Convolutional neural network,Mobile cloud computing,Computing offloading,Collaborative inference,Differential evolution algorithm

1 引言

随着移动设备、嵌入式设备的发展,深度学习计算被广泛应用,移动云计算(Mobile Cloud Computing,MCC)利用云存储和计算资源的优势,将移动设备的计算任务卸载到云中,提升了用户体验。卷积神经网络架构是深度学习的重要分支,已被广泛应用于语音识别、文档分析、语言检测和图像识别等领域[1]。CNN是计算密集型网络,一般采用云计算解决方案,这不能满足需要实时响应的应用程序的要求[2]。例如,无人驾驶汽车需要将大量的图像数据传到云进行处理以获取路况,这需要很高的网络带宽,无法满足实时性需求。另一方面,把云作为中央处理服务器也增加了隐私泄露的风险。以语音识别应用为例,它可能生成有关个人隐私的信息数据,用户不希望这些数据传到服务器端进行处理;同时,将数据上传到云还会加剧能耗开销,对以低功耗运行的设备并不友好。

在移动端执行所有的CNN计算处理,是最简单的避免

到稿日期:2019-08-30 返修日期:2019-11-21 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发课题(2018YFC0407105);国家自然科学基金重点项目(61832005);中央高校科研业务费(2017B20914);华能集团重点研发课题(HNKJ17-21)
This work was supported by the National Key Technology Research and Development Program of the Ministry of Science and Technology of China (2018YFC0407105),Key Project of National Natural Science Foundation of China (61832005),Fundamental Research Funds for the Central Universities (2017B20914) and Key Technology Project of China Huaneng Group (HNKJ17-21).
通信作者:毛莺池(yingchimao@hhu.edu.cn)

云平台处理数据弊端的方法。然而，近年来 CNN 层数不断加深，使得 CNN 模型的推断计算任务面临着巨大的计算资源压力。存储空间较小、计算能力有限的移动端设备无法满足 CNN 模型的存储和计算需求[3]。

采用端云协作方法将计算任务动态地部署在云和边缘之间，是解决上述问题的常用方法。基于协同智能(Collaborative Intelligence)的深度学习推断模式被提出，其核心思想是以层为粒度对神经网络进行切分，部分层在移动端进行推断计算，而另一部分则卸载到云中。Kang 等[4]基于协同推断模式提出的名为 Neurosurgeon 的轻量级调度器，可以根据不同因素选择最佳切分点，将神经网络的计算任务自动划分到移动端和云，以降低端到端延迟和移动设备的能耗。Huang 等[5]考虑了终端设备-边缘服务器-云的三层边缘学习框架，对异构网络进行分层处理，以达到缩短运行时间和降低网络流量的目的。Li 等[6]基于优化框架 Edgent，研究了在给定时延要求下如何联合优化模型切分和模型精简来最大化准确性。

本文针对以最小化时延为单一目标的卸载策略难以满足用户 QoS 需求的问题，提出了移动云环境中 CNN 推断任务细粒度卸载策略；在综合考虑时延与云端费用的基础上建立任务卸载模型，将费用约束下最小化时延的任务卸载问题构建成 0-1 整数规划问题；并提出了改进二进制差分进化算法(Improved Binary Differential Evolution，IBDE)来求解任务卸载问题，弥补了标准差分进化算法不能有效求解 0-1 整数规划以及容易陷入局部最优的缺点。实验结果表明，与移动端推断策略和云端推断策略相比，基于改进二进制差分进化的 CNN 推断任务卸载策略可以缩短响应时间。

本文第 2 节介绍任务卸载的相关工作；第 3 节描述基于改进差分进化算法的 CNN 推断任务卸载策略的流程框架；第 4 节和第 5 节分别给出建立卸载模型过程及改进差分进化算法的具体实现细节；第 6 节通过仿真实验评估该卸载策略的有效性；最后总结全文。

2　相关工作

移动云计算是基于手机等终端设备的云计算技术，它利用云的存储和计算资源突破了移动终端的资源限制。移动云计算主要通过任务卸载来增强移动设备的数据处理能力，减小手机能耗[7]。任务卸载指将移动设备上的任务部分或全部发送到云平台进行处理，实现缩短计算延迟、节约能耗、保护数据隐私性等目的。

根据卸载的粒度，任务卸载可以分为粗粒度卸载和细粒度卸载。粗粒度卸载将整个程序卸载至云，如 Cyber Foraging 系统[8]基于虚拟机技术，将移动设备上的计算任务卸载到代理服务器上，以减少细粒度划分和决策带来的计算开销。细粒度卸载将应用划分为多个可卸载部分，用有向无环图表示各部分之间的关系，构建任务卸载模型进行决策[7]。Huang 等[9]将应用程序建模为带权的有向无环图，提出基于李雅普诺夫优化的卸载算法。Barrameda 等[10]提出了细粒度应用程序模型和快速优化的卸载决策算法。Eshratifar 等[11]将深度学习任务卸载问题建模为整数线性规划问题(Integer Linear Programming，ILP)进行求解。根据给定的约束条件，可以得到不同场景下的最优调度。Zannat 等[12]将粗粒度卸载与细粒度卸载结合，使用马尔可夫决策过程得到卸载策略。CNN 模型是多层网络结构，本文将 CNN 的每一层视作单独的子任务，通过细粒度方式进行任务卸载研究。

根据卸载地点，任务卸载可分为单站点卸载和多站点卸载。多站点卸载涉及卸载策略求解等问题，相比传统的数学方法，启发式算法通过自组织和群体协同能完成复杂的计算任务。Enzai 等[13]提出一种启发式算法来求解多目标组合优化的多站点计算卸载问题。遗传算法(Genetic Algorithm，GA)模拟达尔文生物进化论，包括复制、选择、交叉、变异等遗传操作，是求解卸载策略常见的启发式算法之一。

根据卸载目标，任务卸载分为面向时延的卸载、面向能耗的卸载、权衡时延和能耗的卸载。前两者分别以降低任务响应时间和移动端能耗为目标。Jeong 等[14]通过考虑时延性能，提出了一种在 Web 应用程序的上下文环境中将 DNN 计算卸载到通用边缘服务器的新方法。Guo 等[15]在边缘网络中以最小化平均任务完成时间为目的，将任务卸载问题建模为混合整数规划问题。Jade 系统[16]实现了建立在 Android 操作系统上的细粒度计算卸载，降低了移动设备的能耗。更多研究权衡了时延和能耗两方面的性能，Eshratifar 等[17]将 DNN 中的计算卸载划分问题简化为最短路径问题，提出了一种高效的、自适应的、实用的引擎 JointDNN，来优化时延和能耗。

现有基于协同推断的研究多以最小化时延或能耗为单一卸载目标，忽视了用户 QoS 需求对任务卸载决策的影响，往往不能满足用户的需求。本文综合考虑时延和云平台费用两方面因素，研究了费用约束下最小化时延的 CNN 推断任务卸载问题，在标准差分进化算法的基础上进行改进，以求解近似最优的卸载决策解。

3　整体框架

3.1　问题描述

考虑一个线性拓扑结构的 CNN 模型，假设该模型有 n 个不同类型层 $C=\{c_1,c_2,\cdots,c_n\}$ 和 n 个切分点，其基本结构如图 1 所示。为充分利用移动端设备的计算能力，采用细粒度卸载方式，以层为粒度对 CNN 模型进行切分。在推断计算时，每一层作为一个子任务，在切分点处可以进行任务卸载。子任务按照顺序执行，每个子任务的输入由上一个子任务提供。d_k 和 d_k' 表示第 k 层输入和输出数据的大小，有 d_k' 和 d_k。图 2 为将推断计算过程转换后的任务图。

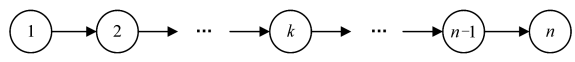

图 1　串型 CNN 模型的结构

Fig. 1　Structure of linear CNN model

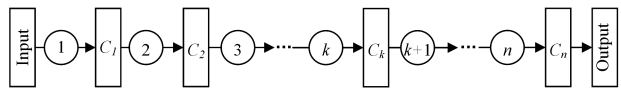

图 2　串型 CNN 模型推断计算的任务图

Fig. 2　DAG for inference computing of linear CNN model

CNN 模型除串行结构外，还存在并行结构，如 ResNet 模型[18]和 GoogleLeNet 模型[19]等。包含并行结构的 CNN 模型也可以转换为任务图，其子任务得以执行的前提条件是它所有的前驱任务执行完成。图 3 为 Inception v1 模型[19]推断过程转换后的任务图。

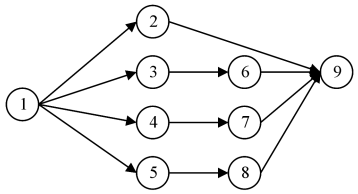

图 3 Inception v1 模型转换后的任务图

Fig. 3 DAG of Inception v1 model after transformation

任务图中的每个子任务在调度时存在两种选择，即在移动端运行或者卸载到云端中执行。设任务由 n 个子任务组成，则共有 2^n 种调度方案。Zhang 等[20]将线性链式任务的执行流程用图 4 表示。节点 k 代表子任务 k 在移动端运行，节点 k^* 代表子任务 k 在云端运行。边的权重代表子任务的执行成本以及两节点间的通信开销。若卸载地点相同，则通信开销忽略不计。为满足用户的 QoS 需求，本文综合考虑费用与时延两方面性能，云端费用约束下最小化时延的任务调度问题可以转化为有向无环图中从起始端 S 到终止端 E 的最短路径问题。

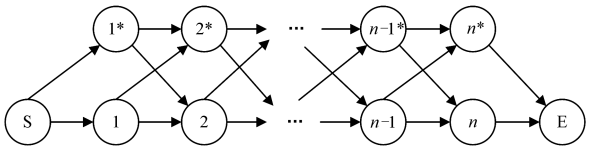

图 4 线性链式任务的执行过程

Fig. 4 Execution process of linear chain task

为了在移动云环境中综合考虑时延和费用，进行 CNN 推断任务卸载来满足用户的 QoS 需求；事先预测 CNN 不同层的运行时间，以层为粒度将 CNN 模型转化为任务图，并构建费用约束下最小化时延的任务卸载模型；提出改进差分进化算法求解任务卸载模型，得到最优卸载策略，从而达到缩短任务响应时间的目的。

3.2 具体流程及系统框架

下面给出在移动云环境中利用协同推断模式进行 CNN 推断任务卸载的具体流程。

(1)预测运行时间：收集移动端和云的训练样本，建立线性回归模型以预测 CNN 不同层如卷积层、全连接层的运行时间。

(2)构建任务卸载模型：以层为粒度将 CNN 的推断过程转化为任务图，提出费用约束下的最小时延问题。

(3)制定卸载策略：提出改进差分进化算法以高效求解任务卸载问题，利用加权自适应变异和二次变异机制防止算法陷入局部最优，得到高质量的最优卸载策略。

(4)协同推断：根据制定的最优卸载策略，移动终端与云对 CNN 模型进行协同推断。

基于以上研究内容，本文提出 CNN 推断任务的系统框架，如图 5 所示。

首先，用户将任务提交到移动端的任务管理模块，运行时间预测模块通过收集样本建立线性回归模型。其次根据网速监测模块监测到的实时网络传输速度，卸载决策模块解析 CNN 模型，将 CNN 的推断计算过程转化为任务图，建立任务卸载模型，并使用改进差分进化算法求解最优卸载策略。数据传输模块实现移动端和云端之间的数据传输。CNN 模型事先存储在两边的 CNN 模型存储模块中，由 CNN 运行环境集成深度学习框架，运行 CNN 推断任务。最后，任务管理模块将结果返回给用户。

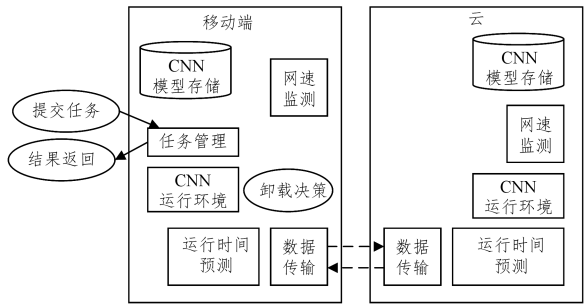

图 5 CNN 任务推断系统的框架

Fig. 5 Framework of CNN inference task system

4 面向时延和费用的 CNN 推断任务卸载模型

4.1 CNN 运行时间预测

CNN 模型层的类型有限且相同类型层结构一致，因此考虑单独对每种类型层进行运行时间的预测。不同网络层的时延取决于各自的参数，如卷积层的主要参数包括输入及输出特征图的大小，以及通道数、卷积核大小和步长；全连接层延迟由输入和输出特征数决定。表 1 列出了不同层影响时延的自变量。用户设备分别在移动端和云收集部分时延数据作为样本，为 CNN 每种类型层建立线性回归模型以预测每层的运行时间。

表 1 线性回归模型的自变量

Table 1 Independent variables of linear regression model

层类型	自变量
卷积层	输入特征图数量，滤波器数量，滤波器尺寸，步长
池化层	输入特征数，输出特征数
全连接层	输入特征数，输出特征数
Relu 层	输入特征数
Dropout 层	输入特征数

4.2 费用约束下最小化时延的任务卸载模型

为了构建任务卸载模型，将 CNN 模型的计算过程转换为任务图，每个子任务存在两种调度选择。将集合 $X=\{x_1, x_2, \cdots, x_n\}$ $(x_i \in X)$ 定义为任务图中子任务的卸载决策，$x_i=0$ 代表子任务 i 在移动端执行，$x_i=1$ 代表任务在云执行。

响应时间包括节点运行时间和数据在移动端和云之间的传输时间。由线性回归模型得到子任务在移动端和云上的执行时间，分别为 t_i^l 和 t_i^c。设子任务 i 和 j 之间的数据传输量为 d_{ij}，数据传输速度为 v，若子任务 i 和 j 在不同端执行，则传输时间为 $t_{ij}=d_{ij}/v$。此外，设输入数据从移动端传输到云需要的时间为 d_o/v，输出数据从云传输到移动端需要的时间为 d_n/v。

相邻两个节点的任务调度存在 4 种可能：1）从移动端到移动端，这种情况只包括移动端执行时间（t_i^l）；2）从云到云，它只包括云执行时间（t_c^i）；3）从移动端到云，它包括移动端执行时间以及数据传输到下一节点的上传时间（$t_i^l+d_{ij}/v$）；4）从云到移动端，它包括云执行时间和数据传输到下一节点的下载成本（$t_c^i+d_{ij}/v$）。基于以上 4 种情况，云费用约束下最小化时延的任务调度问题可以转化为寻求从起始端 S 到终止端 E 的最短路径问题[17]。

若 t_{i2} 表示子任务 i 的开始时间，t_{i1} 表示其完成时间。设子任务 i 的前驱任务集合为 $J=\{j_1,j_2,\cdots,j_{m_i}\}$，$m_i$ 为前驱任务的数量。子任务 i 只有等到所有前驱任务完成之后才能开始运行，因此子任务 i 的开始时间为：

$$t_{i2}=\max\{t_{k1}+t_{ki}(x_i-x_k)^2,k\in J_i\} \tag{1}$$

其中，t_{ki} 表示子任务 k 和 i 之间的传输时间。子任务 i 的完成时间为：

$$t_{i1}=t_{i2}+x_i\cdot t_c^i+(1-x_i)\cdot t_i^l \tag{2}$$

为了计算整个任务图的完成时间，可以从起始子任务开始计算。起始子任务的开始时间为：

$$t_{i2}=x_1\frac{d_0}{v} \tag{3}$$

其他子任务根据式(1)和式(2)迭代计算。按子任务执行顺序得到最终的响应时间为：

$$T(X)=t_{n1}+x_n\frac{d_n}{v} \tag{4}$$

考虑到用户的 QoS 需求，本文研究在满足云执行费用约束的情况下最小化响应时间的问题。设在云上执行任务单位时间所需的计算费用为 p，则总费用 $C=\sum_{i=1}^{n}x_i\cdot(t_{i1}-t_{i2})\cdot p$。用户给定费用限制 $C_{\max}$，则 CNN 推断任务卸载问题转化为在 $C\leqslant C_{\max}$ 的情况下求解 $\min\ T(X)$ 的卸载策略的问题。从目标函数看出，这是一个 0-1 整数规划问题。该问题是 NP-hard 问题，传统方法难以高效求解，因此本文使用改进差分进化算法来得到近似最优解。

5　基于差分进化算法的 CNN 推断任务卸载策略

5.1　基于加权海明距离的种群初始化

差分进化（Differential Evolution，DE）算法的思想来源于遗传算法，但遗传操作的复杂度更低。该算法的初始个体由随机函数产生，为防止初始种群中个体的相似度过高，本文使用加权海明距离[21]度量种群中个体的相似性。设初始种群规模为 N，个体维度为 D，最大进化代数为 $L_{\max}$，第 t 代种群中的第 $i(i=1,2,\cdots,N)$ 个个体表示为 $X_i(t)=[x_i^1(t),x_i^2(t),\cdots,x_i^D(t)]$，其中 $x_i^j(t)$ 表示第 i 个体的第 j 维分量，可以取值为 0 或 1。将个体第 k 维分量的权重设定为 a_k，则个体 i 与 j 之间的加权海明距离为：

$$WHD_{ij}=\sum_{k=1}^{D}a_k|x_i^k-x_j^k| \tag{5}$$

初始化过程中会随机生成 $3N$ 个个体，随机选择其中一个加入初始种群集合 G；之后根据格式计算剩余个体与 G 中个体的加权海明距离之和，选择和值最大的个体加入集合 G。重复以上步骤，直至集合 G 中有 N 个个体。改进后的初始化方案可以最大化种群的差异性，使个体均匀分布在解空间中。

5.2　变异策略的二进制转换

DE 算法采用基于差分向量的简单变异，常用的变异策略有 DE/rand/1，DE/best/1 和 DE/rand/2 等。设缩放因子为 F，变异策略 DE/rand/1 表示为：

$$H_i(t)=X_{r_1}(t)+F\cdot(X_{r_2}(t)-X_{r_3}(t)) \tag{6}$$

其中，r_1，r_2 和 r_3 是第 t 代种群中随机选取的 3 个个体编号。DE/best/1 策略是将 $X_{r_1}(t)$ 用最优个体 $X_{best}(t)$ 代替。由于这些 DE 变异策略不能很好地应用在 0-1 整数规划问题中，本文将变异策略进行二进制转换。DE/rand/1 变异策略转换后表示为：

$$H_i(t)=X_{r_1}(t)+W\otimes(X_{r_2}(t)\oplus X_{r_3}(t)) \tag{7}$$

其中，$+$ 代表或运算，$\otimes$ 代表与运算，$\oplus$ 代表异或运算。W 是随机的二进制串，其每一位 w_j 由以下公式确定：

$$w_j=\begin{cases}1, & \text{if } rand[0,1]\leqslant F\\ 0, & \text{otherwise}\end{cases} \tag{8}$$

5.3　加权自适应变异

缩放因子 F 值改变会明显影响算法的收敛速度。当前许多研究通过自适应调整 F 值来提高 DE 算法的性能，这些方案基于两个主要因素：进化代数和适应度。

（1）基于种群进化代数的 F 自适应调整符合 Logistic 模型[22]，第 t 代时的缩放因子为：

$$F(t)=\frac{F_{\min}}{1+(F_{\min}/F_{\max}-1)e^{-\alpha t}} \tag{9}$$

其中，$F_{\min}$ 和 $F_{\max}$ 分别表示缩放因子的最小值和最大值，α 为衰减速率。

（2）个体适应度考虑个体层面 F 的调整，当个体适应度与最优适应度相差较大时应增大 F 以提高全局搜索能力，相反则应减小 F 以加速收敛。第 t 代时个体 i 的缩放因子为：

$$F_i(t)=F_{\min}+(F_{\max}-F_{\min})\cdot\frac{f_i(t)-f_b(t)}{f_w(t)-f_b(t)} \tag{10}$$

其中，$f_i(t)$ 为个体 i 的适应度值，$f_b(t)$ 和 $f_w(t)$ 分别表示第 t 代种群中个体的最优和最差适应度。综合考虑两种因素，本文引入构造权重因子 μ，构造 F_i 的动态调整公式：

$$F_i^*(t)=\mu F(t)+(1-\mu)F_i(t) \tag{11}$$

根据式(11)在种群进化过程中自适应调整缩放因子，能有效加快收敛速度且防止陷入局部最优。

5.4　二次变异

随着进化代数的增加，个体之间的差异变小，这会导致收敛速度变慢且容易陷入局部最优。本文采用基于聚集度的二次变异来增加种群多样性，防止种群进化停滞；使用种群适应度方差来反映聚集度[23]，当方差小于给定阈值 ε 时，随机选取 $N\cdot MF$（MF 为随机变异比例）个个体对每一维分量进行随机扰动。若二次变异之后个体适应度增加，则保留变异个体。计算适应度方差的公式如下：

$$\sigma^2=\frac{\sum_{i=1}^{N}(f_i(t)-f_{avg}(t))^2}{N} \tag{12}$$

基于以上改进内容，图 6 给出了 IBDE 算法的基本流程，其中交叉、选择步骤的方法与标准差分进化算法一致。

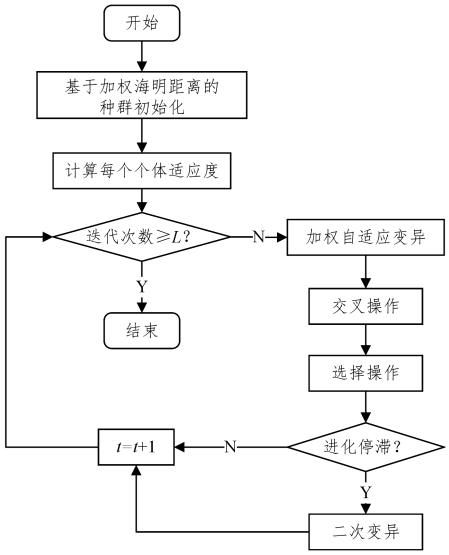

图 6 IBDE 算法的流程图

Fig. 6 Flow chart of IBDE algorithm

5.5 基于 IBDE 算法的任务卸载策略

本文提出基于 IBDE 的移动云计算 CNN 推断任务卸载策略，如图 7 所示。系统针对不同的 CNN 层预测每一层在移动端和云上的运行时间；然后基于细粒度卸载方式将 CNN 推断过程转化为任务图，构建费用约束下最小化时延的模型，将卸载问题转换为 0-1 整数规划问题，并采用 IBDE 算法解决该问题。

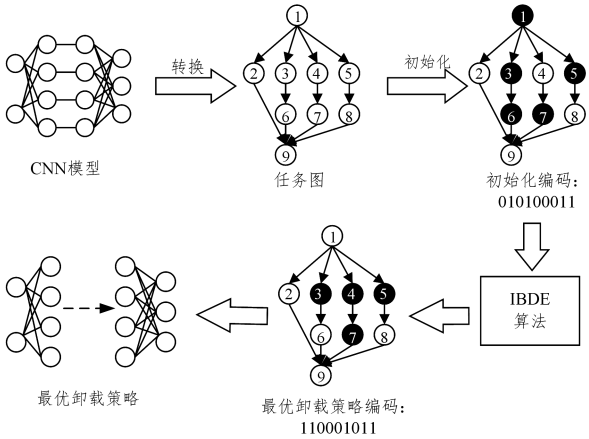

图 7 基于 IBDE 的 CNN 推断任务卸载策略

Fig. 7 CNN inference task offloading strategy based on IBDE algorithm

IBDE 算法首先使用加权海明距离初始化种群，并对变异公式进行二进制转换。在迭代过程中，根据进化代数和个体适应度调整缩放因子，加速收敛过程。计算种群聚集度，判断种群进化是否停滞，若停滞则进行二次变异以增加种群多样性。算法达到最大进化代数后终止，进而根据最优任务卸载策略实现协同推断。

6 实验验证

6.1 实验环境

(1)IBDE 算法性能的研究。利用计算机仿真实验进行评估，实验平台为 Matlab R2016a。选择二进制遗传算法(Binary Genetic Algorithm，BGA)以及二进制差分进化算法(Binary Differential Evolution，BDE)作为对比。仿真实验及各算法的部分参数分别如表 2 和表 3 所列。

表 2 仿真实验参数

Table 2 Simulation parameters

仿真参数	数值
移动端处理能力/MIPS	200
云处理能力/MIPS	1000
子任务计算量/百万指令	300～3000
子任务间数据传输量/kB	100～2000
网络传输速率/(kB/s)	1000
单位时间云计算费用/元	150

表 3 3 种算法的部分参数

Table 3 Partial parameters of three algorithms

IBDE		BGA		BDE	
参数	数值	参数	数值	参数	数值
种群规模 N	30	染色体数	30	种群规模	30
缩放因子 F	0.3～0.9	交叉概率	0.6	交叉因子	0.5
自适应调整权重	0.5	变异概率	0.1	缩放因子	0.6
适应度方差阈值	2				
随机变异率	0.1				

(2)基于 IBDE 算法的 CNN 推断任务卸载策略的有效性研究。使用 PC 机作为云端，将树莓派作为移动端。仿真系统开发基于 Python 语言，选择 PyTorch 作为深度学习框架。PC 机和树莓派通过 Wi-Fi 连接，使用 WonderShaper 软件控制网络传输速度。利用在 CIFAR-10 数据集上训练好的 AlexNet 模型[24]，比较云端方案、移动端方案以及本文提出的基于 IBDE 算法的协同推断方案。

6.2 实验结果

6.2.1 IBDE 算法的性能

为验证算法的有效性，实验随机产生任务图。实验方案选择了两种与 IBDE 算法较为相近的启发式算法(BGA 和 BDE)与之进行比较。实验比较了 3 种算法在任务量固定情况下的收敛速度以及不同任务量下任务响应时间和算法的耗时。

图 8 给出了子任务数 $N=50$ 时 3 种算法的任务响应时间与迭代次数的关系变化曲线。由图 8 可以看出，随着迭代次数的增加，3 种算法都可以缩短任务响应时间。IBDE 算法在迭代 30 次左右时就得到了最佳解，而 BDE 大约迭代了 44 次，BGA 算法则迭代了超过 70 次。由此可见，IBDE 算法的收敛速度明显快于 BDE 和 BGA 算法，其速度是 BDE 算法的约 1.41 倍，是 BGA 算法的 2.32 倍。同时，由于 IBDE 算法利用加权自适应变异和二次变异的机制增加了种群多样性，防止算法陷入局部最优，因此该算法在任务响应速度方面的表现最佳，所得最优解的质量高于另两种启发式算法。

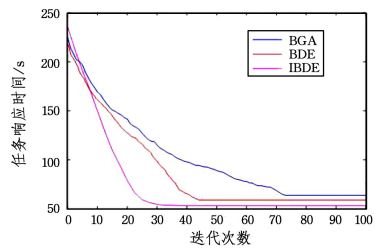

图 8 3 种算法的任务响应时间与迭代次数的变化曲线

Fig. 8 Change curves of task response time and iteration number of three algorithms

第二个实验比较不同数量子任务情况下，3 种启发式算法的性能差异。取子任务数为 25，50 和 100，随机生成任务图来比较 BGA，BDE，IBDE 的性能差异，包括算法结果是否接近最优解以及算法耗时长短。为保证实验结果的准确性，采用 10 次实验结果的平均值作为最终结果，如图 9 和图 10 所示。

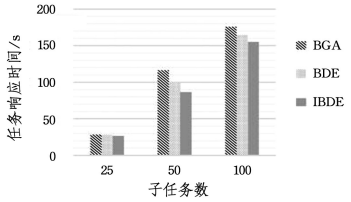

图 9　不同子任务数下的任务响应时间

Fig. 9　Task response time under different numbers of subtasks

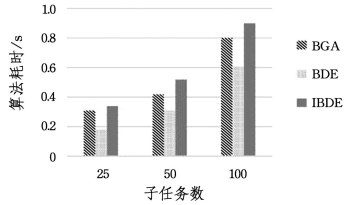

图 10　不同子任务数下算法的耗时

Fig. 10　Time consumption of algorithms with different numbers of subtasks

如图 9 所示，当子任务数量较少时，3 种算法的任务响应时间几乎相等。而随着子任务数的增加，每种算法所需的响应时间不断增长。当子任务数为 100 时，IBDE 算法的任务响应时间为 155.02 s，比 BGA 算法缩短了 11.90%，比 BDE 算法缩短了 6.11%。然而，从图 10 可以发现，随着子任务数的增加，IBDE 算法的耗时明显长于 BDE 和 BGA 两种算法的耗时，这是由于 IBDE 算法在初始化和改进的变异机制中需要额外的计算开销导致的。当 $N=100$ 时，IBDE 算法耗时约 0.91 s，BGA 算法耗时约 0.83 s，而 BDE 算法耗时仅约0.61 s，IBDE 的耗时是 BDE 耗时的近 1.36 倍。这说明，相比其他两种启发式算法，IBDE 算法可以得到更优解，缩短了任务响应时间，但算法本身需要更长的计算时间。

6.2.2　基于 IBDE 算法的协同推断性能

本文研究的目的是在费用约束下最小化时延来满足用户的 QoS 需求，因此费用约束对卸载决策有着重要的影响。此外，网络传输速度也是卸载决策的决定性因素之一。本次实验是将基于 IBDE 算法的协同推断策略与云端和移动端两种卸载策略进行对比，研究在不同费用约束以及网络传输速度情况下 3 种卸载策略的任务响应时间。

图 11 显示了当网络传输速度 $v=500$ kB/s 时，C_{max} 与任务响应时间之间的关系。可以看到，在 C_{max} 为 0 时，整个 CNN 模型在移动端执行。由于树莓派计算能力的限制，本地执行所需的响应时间很长，约为 3.93 s；当 $C_{max}<0.2$ 时，随着费用约束的提升，能卸载到云的 CNN 层数增加，任务响应时间缩短；当 $C_{max}\geqslant 0.2$ 时，整个 CNN 模型卸载到云，此时响应时间最短，大约为 1.88 s，比移动端方案降低了近 51.92%，比 $C_{max}=0.1$ 时的响应时间缩短了大约 27.80%。同时可以看出，云方案只在 $C_{max}\geqslant 0.2$ 时才能执行，不能很好地适应用户需求。

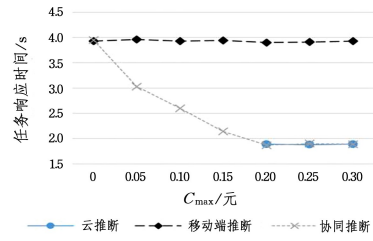

图 11　C_{max} 对任务响应时间的影响（$v=500$ kB/s）

Fig. 11　Task response time with different C_{max} ($v=500$ kB/s)

取 $C_{max}=0.3$，研究不同网络传输速度对任务响应时间的影响，如图 12 所示。移动端方案不考虑将 CNN 任务传输至云端，响应时间仍约为 3.93 s 左右。当网络传输速度为 50 kB/s 时，由于将输入数据传输到云的耗时过长，因此云方案的任务总响应时间超过移动端方案的响应时间，约达到 4.34 s。随着网速的增加，云方案的任务响应时间明显降低。而基于 IBDE 算法的协同推断方案在 $50\text{ kB/s}<v<300\text{ kB/s}$ 时，只将部分 CNN 层卸载到云，利用了云的计算优势且避免了过长的通信延迟，因此性能优于另两种方案。当网速达到 300 kB/s 时，协同推断方案将所有 CNN 层卸载到云。当 $v=400$ kB/s 时，协同推断方案比移动端方案节省了约 48.73%的耗时。

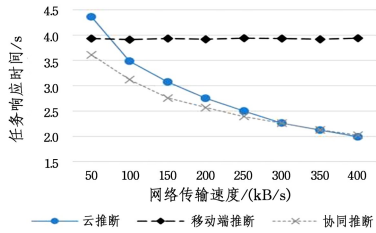

图 12　网络传输速度对任务响应时间的影响（$C_{max}=0.3$）

Fig. 12　Effect of network transmission speeds on task response time ($C_{max}=0.3$)

上述实验结果表明，本文提出的 IBDE 算法所得最优解的质量明显高于其他两种启发式算法，但算法耗时较长。相比云端和移动端方案，基于 IBDE 算法的协同推断方案能有效缩短任务响应时间，且能根据设定的费用约束和不同网速自适应地求解卸载决策。

上述实验结果表明，本文提出的 IBDE 算法所得最优解的质量明显高于其他两种启发式算法，但算法耗时较长。相比云端和移动端方案，基于 IBDE 算法的协同推断方案能有效缩短任务响应时间，且能根据设定的费用约束和不同网速自适应地求解卸载决策。

结束语　本文针对费用约束下最小化时延的任务卸载问题，提出了一种基于差分进化的 CNN 推断任务卸载策略。该策略基于细粒度卸载的协同推断模式，利用线性回归模型预测 CNN 层的运行时间，通过把推断过程转化为任务图来构建 0-1 整数规划问题。针对 0-1 整数规划变量离散的特点，提出加权自适应变异和随机二次变异的二进制差分进化算法来高效求解任务卸载问题，弥补了标准差分进化算法不能有效求解 0-1 整数规划以及容易陷入局部最优的缺点。算法仿真实验结果显示，相比其他启发式算法，IBDE 算法能得到更优质量的卸载决策解。使用树莓派和 PC 机作为移动端和云

端的仿真实验结果表明，基于 IBDE 算法的 CNN 推断任务卸载策略能够在满足费用约束的情况下有效缩短任务响应时间。

参 考 文 献

[1] ZHOU F Y, JIN L P, DONG J. Review of Convolutional Neural Network[J]. Chinese Journal of Computers, 2017, 40(6): 1229-1251.

[2] PLASTIRAS G, TERZI M, KYRKOU C, et al. Edge Intelligence: Challenges and Opportunities of Near-Sensor Machine Learning Applications[C]// 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2018: 1-7.

[3] GUO T. Cloud-based or On-device: An Empirical Study of Mobile Deep Inference[C]// 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2018: 184-190.

[4] KANG Y, HAUSWALD J, GAO C, et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge[J]. ACM SIGplan Notices, 2017, 52(4): 615-629.

[5] HUANG Y T, MA Y Q, FAN X Y, et al. When Deep Learning Meets Edge Computing[C]// 2017 IEEE 25th International Conference on Network Protocols (ICNP). IEEE Computer Society, 2017: 1-2.

[6] LI E, ZHOU Z, CHEN X. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy[C]// Proceedings of the 2018 Workshop on Mobile Edge Communications. ACM, 2018: 31-36.

[7] ZHANG Q, ZHANG H L. Research progress of task offloading technologies in mobile cloud computing[J]. Intelligent Computer and Applications, 2016, 6(6): 1-4.

[8] GOYAL S, CARTER J. A lightweight secure cyber foraging infrastructure for resource-constrained devices[C]// Sixth IEEE Workshop on Mobile Computing Systems and Applications. IEEE, 2004: 186-195.

[9] HUANG D, WANG P, NIYATO D. A dynamic offloading algorithm for mobile computing[J]. IEEE Transactions on Wireless Communications, 2012, 11(6): 1991-1995.

[10] BARRAMEDA J, SAMAAN N. A novel application model and an offloading mechanism for efficient mobile computing[C]// 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). IEEE, 2014: 419-426.

[11] ESHRATIFAR A E, PEDRAM M. Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment[C]// Proceedings of the 2018 on Great Lakes Symposium on VLSI. ACM, 2018: 111-116.

[12] ZANNAT H, HOSSAIN M S. A hybrid framework using Markov decision process for mobile code offloading[C]// 2016 19th International Conference on Computer and Information Technology (ICCIT). IEEE, 2016: 31-35.

[13] ENZAI N I M, TANG M. A heuristic algorithm for multi-site computation offloading in mobile cloud computing[J]. Procedia Computer Science, 2016, 80: 1232-1241.

[14] JEONG H J, JEONG I C, LEE H J, et al. Computation Offloading for Machine Learning Web Apps in the Edge Server Environment[C]// 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE Computer Society, 2018: 1492-1499.

[15] GUO K, YANG M, ZHANG Y, et al. An Efficient Dynamic Offloading Approach based on Optimization Technique for Mobile Edge Computing[C]// 2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud). IEEE, 2018: 29-36.

[16] QIAN H, ANDREAEN D. Reducing mobile device energy consumption with computation offloading[C]// 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). IEEE, 2015: 1-8.

[17] ESHRATIFAR A E, ABRISHAMI M S, PEDRAM M. JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services[J/OL]. https://arxiv.org/pdf/1801.08618.

[18] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 770-778.

[19] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015: 1-9.

[20] ZHANG W, WEN Y, WU D O. Energy-efficient scheduling policy for collaborative execution in mobile cloud computing[C]// 2013 Proceedings Ieee Infocom. IEEE, 2013: 190-194.

[21] CHU C A, LIN W, XIAO H. An adaptive genetic algorithm based on weighted hamming distance[J]. Journal of South Normal University (Natural Science Edition), 2015, 47(6): 121-127.

[22] CHEN H, FAN Y R, DENG S G. Adaptive differential evolution algorithm based on logistic model[J]. Control and Decision, 2011, 26(7): 1105-1108.

[23] WU L H, WANG Y N, YUAN X F, et al. Differential Evolution Algorithm with Adaptive Second Mutation[J]. Control and Decision, 2006, 21(8): 898-902.

[24] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.

WANG Xuan, born in 1996, master candidate, is a student member of CCF. Her main research interests include edge computing and cloud computing.

MAO Ying-chi, born in 1976, Ph.D, professor, is a senior member of China Computer Federation. Her main research interests include cloud computing and edge computing, mobile sensing systems and internet of things.