中南林业科技大学

毕业设计说明书

学生姓名：尹建　　学　号：20153780

学　　院：计算机与信息工程学院

专业年级：2015 级计算机科学与技术 1 班

题　　目：基于深度强化学习的边缘任务迁移能效优化

指导教师：邝祝芳　教授

评阅教师：杨卫民　副教授

2019 年 5 月

摘 要

移动边缘计算（MEC）在5G网络中具有解决密集型计算需求的潜力。MEC将计算密集型任务迁移到MEC服务器，从而扩展边缘无线网络的计算能力。这里考虑一个多用户MEC系统，多用户设备可以通过无线信道将计算卸载到MEC服务器上。我们将所有用户的延迟和能耗之和作为优化目标。为了最大限度地降低所考虑的MEC系统的总成本，优化卸载决策和计算资源的分配。然而，在这样一个系统中，找到最优的策略是一个挑战。强化学习（RL）考虑的一个长期目标，这对于我们考虑的多用户无线MEC系统的动态系统非常重要。为此，提出了基于RL的无线MEC资源分配优化框架。具体地，分别提出了基于Q学习和基于深度强化学习的方案，将Q学习和深度强化学习应用与该框架上，并在不同的系统参数下，进行了实验，仿真结果表明，与其他资源分配方案相比，本论文提出的基于Q学习和基于深度强化学习的方案能显著降低总成本。

关键字：移动边缘计算；能效优化；深度增强学习

Title Deep Reinforcement Learning based Task migration and Energy efficiency optimization for MEC

Abstract:

Mobile Edge Computing (MEC) has the potential to meet the needs of intensive computing in 5G networks. MEC migrates computationally intensive tasks to MEC servers, thus expanding the computing power of edge wireless networks. In this paper, we consider a multi-user MEC system, where multiple user equipments (UEs) can perform computation offloading via wireless channels to an MEC server. We formulate the sum cost of delay and energy consumptions for all UEs as our optimization objective. In order to minimize the sum cost of the considered MEC system, we jointly optimize the offloading decision and computational resource allocation. However, it is challenging to obtain an optimal policy in such a dynamic system. Besides immediate reward, Reinforcement Learning (RL) also takes a long-term goal into consideration, which is very important to a time-variant dynamic systems, such as our considered multi-user wireless MEC system. To this end, we propose RL-based optimization framework to tackle the resource allocation in wireless MEC. Specifically, the Q- learning based and Deep Reinforcement Learning (DRL) based schemes are proposed, respectively. Q-learning and deep reinforcement learning are applied to the framework, and experiments are carried out under different system parameters. Simulation results show that the proposed scheme achieves significant reduction on the sum cost compared to other resource allocation schemes.

Keyword：Mobile edge computing; Energy efficiency optimization; Deep Reinforcement Learning

目　录

1. 绪论

1.1 研究背景及意义

5G 网络正在兴起，给予人们、机器和具有各种服务的事物之间的大规模连接。云无线接入网（C-RAN）被认为是一种核心技术，能够在 5G 网络中实现这些服务，具有前所未有的频谱效率和能源效率[1]。

同时，5G 网络中越来越多的计算密集型应用需要低延迟，如交互式游戏，再比如一些需要实时感知的应用、摄像头感知等，这些都是耗能的应用。由于电池功率和计算能力有限，移动用户设备很难满足这些要求。移动边缘计算(Mobile edge computing 以下简称 MEC）旨在解决 UES 与应用程序之间的矛盾。

MEC 通过部署高性能服务器来增强移动网络边缘的计算能力[2]。MEC 服务器密集分布在移动用户附近，用户设备可以通过无线信道将任务卸载到 MEC 服务器上,这样可以显著减少应用程序的延迟，提高服务质量（QoS）。因此，计算卸载和计算资源分配的研究成为 MEC 系统的一个关键[3] 。

1.2 国内外研究现状

近年来，研究人员针对不同的设计目标提出了 MEC 计算卸载和资源分配的一些方案[4]。部分调查可以追溯到传统移动云计算（MCC）系统。在论文[5]中的工作提出了一般指导原则，以作出能源消耗最小化的卸载决策，其中通信链路很容易被假定为具有固定速率。由于无线通信的数据速率不是恒定的，在论文[6]中提出了利用凸优化的最优二进制计算卸载决策，其中著名的香农-哈特利公式给出了功率速率函数，然而，需要准确的信道状态信息。考虑到无线信道在时间域上的随机变化，这些主题不太适合动态系统。

为了解决这个问题，强化学习（RL）被用作一个有效的解决方案。考虑到未来来自环境的奖励反馈，RL 代理可以调整其策略以实现最佳的长期目标，这在我们考虑的多用户无线 MEC 系统等时变系统中非常重要。

马尔可夫决策过程（MDP）理论是 RL 的一个基本理论，他需要一个固定的状态转移概率矩阵 P，论文 [8] 的工作就是基于 MDP 理论设计了单用户延迟最优

任务调度策略，该策略控制了本地处理和传输单元的状态，并且是在任务缓冲队列长度在通道状态下。为了进一步优化计算延迟和能耗，在论文［9］中考虑了将单个用户的长期平均执行成本最小化的问题，提出了一个半 MDP 框架来共同控制本地 CPU 频率、调制方案和数据速率。然而，这些方案很大的挑战是，很难得到矩阵 P 的实际概率分布，提出的基于 MDP 的主题过于依赖于环境信息。更重要的是，MDP 主要用于一个单元和一个用户进行任务调度的场景。考虑到实际的成本和效益，MEC 服务器需要更好地服务于更多的 UE。

1.3 研究目的及内容

为此,这里致力于为一个多用户 MEC 计算卸载系统设计一个基于 RL 的方案。我们首先提出一个基于 Q 学习的方案来取代 MDP。但随着 Q-Learning 用户数量的增加，可能发生的动作数量可能会迅速增加[10]，那么计算和存储动作值 Q 就比较复杂了。为了解决这个问题，我们使用了深度强化学习（DRL）[7] 作为增强版的 RL。提出了一种基于 DRL 的深度 Q 网络（DQN），利用深度学习网络代替 Q-learning 中的的 Q 表，并将 Q-learning 算法和 DQN 算法应用与多用户的移动边缘计算的场景，对比了在不同环境下，移动边缘计算任务的迁移决策的方案，并分析各个方案的优劣情况。

1.4 论文结构

本文的组织结构如下：

第一章：绪论部分。主要介绍了相关的研究背景和国内外研究现状，然后介绍了本论文所做的一些研究工作。

第二章：相关理论知识部分。本章节简要介绍了机器学习，及其依据其技术的分类情况，包括分类技术，回归技术，聚类技术等；介绍了深度学习，卷积神经网络的诞生即应用，以及循环神经网络的处理情景，随后介绍了强化学习中有模型学习，和免模式学习的两种强化学习模式，并通过 T 步累积奖励和γ折扣累积奖励分别阐述其相关原理。最后介绍了移动边缘计算的诞生，应用场景，以及移动边缘计算的计算卸载问题。

第三章：系统模型。在本章提出了在移动边缘计算下的一个场景，并且量化

了移动边缘计算的计算卸载模型，提出了移动边缘计算的任务迁移能效优化问题。

第四章：解决方案，介绍了如何将强化学习应用到移动边缘计算的场景下。介绍了 Q-learning 算法，和 DQN 算法相关原理，以及在 Q-learning 算法，DQN 算法应用于移动边缘计算时，应考虑的相关细节。包括 Q-learning 算法的高纬度 Q 表，DQN 算法在移动边缘计算场景下的损失函数构造，以及本文用到的 DQN 算法的优化函数。

第五章：仿真实验结果，模拟移动边缘计算的使用场景，在不同环境下，移动边缘计算任务的迁移决策方案对比，并分析各个方案的优劣情况。

2 相关理论知识

2.1 机器学习

2.1.1 机器学习概述

计算机已经深刻的进入的人们的日常生活。人们的生活已经彻底地和计算机融合。智能化是近年来计算机领域研究和发展的一个目标。

例如推荐系统，商家总是能够根据您以前的浏览、消费情况来给您推荐您可能的商品，例如医疗诊断，智能医疗能够帮助人们更早的发现疾病的隐患，再比如计算机视觉，它能够帮助人们更好的感知世界。

机器学习的产生就是为了解决两大类问题，一是希望能够预测，二是能够找出数据的内在联系。依照此依据，机器学习算法分类如下图 2.1 所示，机器学习可以分为监督学习和无监督学习。监督学习依据已知的标注信息，让模型能够预测未来的情况。无监督学习则是找到数据的内在联系。

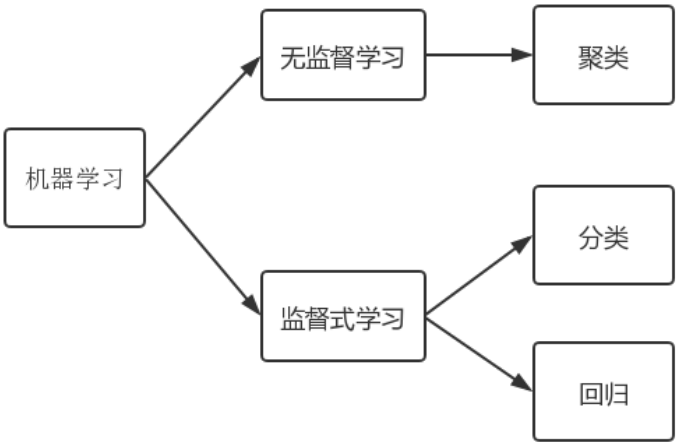

图 2.1 机器学习技术分类

2.1.2 监督学习

监督学习主要采用分类和回归两种技术方案。

例如，分类技术可以预测离散情况。例如，图像是猫还是狗，电影是爱情片

还是恐怖片，邮件是不是垃圾邮件等等。分类技术分辨输入数据的类别。

例如，回归技术可以预测连续情况。例如，房价的预测，股市的预测等等。

2.1.3 无监督学习

聚类是最常用的无监督学习方法。这种方法可以发现数据内在的联系，如因子分析。常常应用于基因序列分析、市场调查等等。

经过多年努力，人们围绕特征，提出了一个一个优秀的思想，例如知识发现和数据挖掘会议提出的“十大算法”等等，他们在很多领域取得了不少的成绩。

这些算法依照上面监督学习和无监督学习的方法的分类情况如下图 2.2 所示：

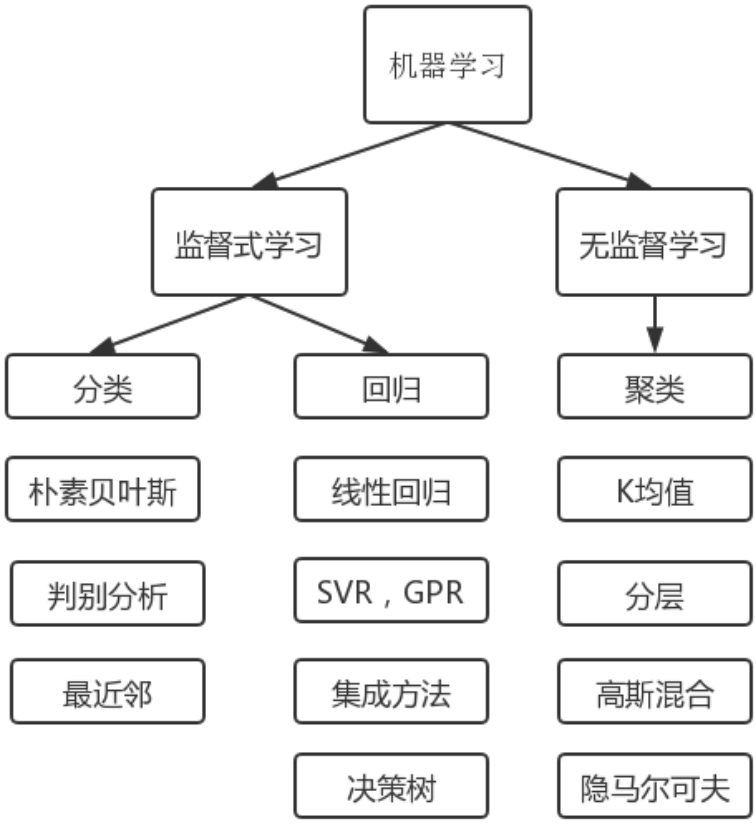

图 2.2 机器学习具体算法分类情况

2.2 深度学习

2.2.1 深度学习概述

近年来，深度学习受到极大的关注，事实上它的基于神经网络模型，和以数据为驱动的核心思想已经被研究将近百年。

我们现在正处于一个程序设计得到深度学习的帮助越来越多的时代。这可以说是计算机科学历史上的一个分水岭。举个例子，深度学习已经在你的手机里：拼写校正、语音识别、认出社交媒体照片里的好友们等。得益于优秀的算法、快速而廉价的算力、前所未有的大量数据以及强大的软件工具，如今大多数软件工程师都有能力建立复杂的模型来解决十年前连最优秀的科学家都觉得棘手的问题。

在相当一段时间里，深度学习一直被其他机器学习的算法所超越，例如 SVM，当时的深度学习的计算过程是：

1. 获取图像数据集；
2. 使用已有的特征提取函数生成图像的特征；
3. 使用机器学习模型对图像的特征分类。

当时机器学习领域生机勃勃、严谨而且极其有用。然而，如果跟计算机视觉研究者交谈，则是另外一幅景象。他们会告诉你图像识别里“不可告人”的现实是：计算机视觉流程中真正重要的是数据和特征。也就是说，使用较干净的数据集和较有效的特征甚至比机器学习模型的选择对图像分类结果的影响更大。

近年来，仰仗着大数据集和强大的硬件，深度学习已逐渐成为处理图像、文本语料和声音信号等复杂高维度数据的主要方法。

2.2.2 卷积神经网络

卷积神经网络出现以前，我们就已经出现神经网络了，此时是以单一的全联接层的形式。多层感知机（MLP）就是其中的代表，其网络结构如下图 2.3 所示：

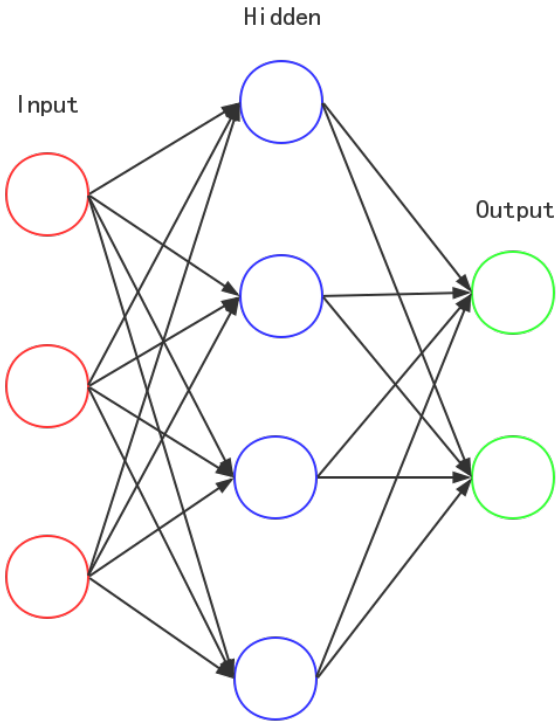

图 2.3 感知机网络结构

这样的神经网络在图像领域是不适合的，例如我们 Fashion-MNIST 这个小型的数据集，每张图像高宽 28 个像素，将像素逐行展开，得到 784 的向量，将它作为一个输入。此时有两个局限性：

1. 图像在同一列的信息会丢失。
2. 图像如果较大 1000*1000 的彩色图片，如果输出是 256 个单元，则需要 3000000*256 个单元，则将是近 3GB 的内存或显存。

于是，卷积神经网络诞生；

如图 2.4 所示的卷积神经网络中二位互相关运算，输入部分可以看作是感受野，感受野与卷积和进行互相关运算，如图 2.4 阴影部分所示，阴影部分为第一个输出元素及其计算所使用的输入和核数组元素：$0\times0+1\times1+3\times2+4\times3=190\times0+1\times1+3\times2+4\times3=19$。

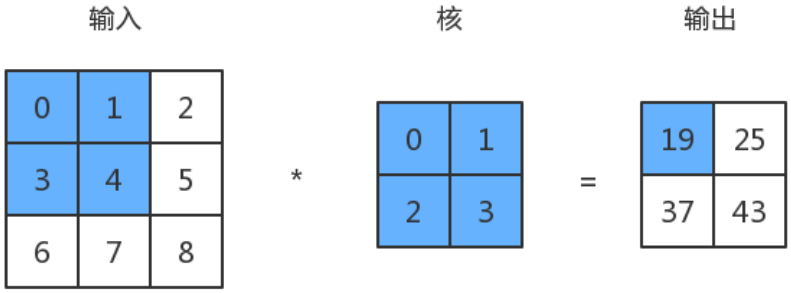

图 2.4 卷积神经网络中的二维互相关运算

在二维互相关运算中，卷积核从感受野的最左上方开始，按从左往右、从上往下的顺序，依次在输入数组上滑动。当卷积窗口滑动到某一位置时，窗口中的输入子数组与核数组按元素相乘并求和，得到输出数组中相应位置的元素。图 2.4 中的输出数组高和宽分别为 2，其中的 4 个元素由二维互相关运算得出：

0*0+1*1+3*2+4*3=19

1*0+2*1+4*2+5*3=25

3*0+4*1+6*2+7*3=37

4*0+5*1+7*2+8*3=43

这样定义的二维相关互相关运算。在 AlexNet 网络中，一举赢得 ImageNet 2012 图像识别大赛。它首次证明了这样的通过学习的得到的特征是可以超越手工设计的特征。从此深度卷积神经网络发展越来越快。VGG（使用重复元素的网络），GoogleNet（含并行连接的网络），ResNet（残差网络），DenseNet（稠密网络）等等相继诞生。

2.2.3 循环神经网络

在处理语言模型时，循环神经网络是最常见的一种处理方案。

语言可以看作一段长度为 T 的文本，依次出现了$w_1, w_2, \ldots, w_T$,在时间 t 序列上，我们可以把$w_t(1 \le t \le T)$看作在时间步 t 的标签。一个长度为 T 的文本$w_1, w_2, \ldots, w_T$所出现的概率为：$p(w_1, w_2, \ldots, w_T)$

例如在机器翻译时，首先采用逐词翻译”you go first”，逐词翻译的结果是“你走先”，事实上，“你先走”出现的概率比“你走先”出现的概率要高，于是把”you go first”翻译为“你先走”。

依据概率论的知识，我们知道文本$(w_1, w_2, w_3, \ldots w_T)$出现的概率为公式（2-1）所示：

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | W_1 \ldots W_{t-1}) \quad (2\text{-}1)$$

例如：T=4 时，文本(w_1, w_2, w_3, w_4)出现的概率为公式（2-2）所示：

$$p(w_1, w_2, w_3, w_4) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)p(w_4|w_1, w_2, w_3) \quad (2\text{-}2)$$

此概率，我们可以通过词频来估计，例如$p(w_1)$是w_1占训练集的总词数的比值，$p(w_2|w_1)$ 可以通过w_1, w_2另个相邻词语的频数和w_1的频数的比值计算。但是同样，会带来一个问题，就是需要存储很多单词同时出现的概率，且呈指数增长。虽然，我们可以通过 N 阶马尔可夫链来减少一些，但是 N 较小时，概率不精确，N 较大时，同样存储的概率会很多。于是循环神经网络就诞生了。

对于一个单层的感知机，一个样本数为N，特征数为d，隐藏层激活函数为$\emptyset$，隐藏层输出为$H \in R^{n*h}$，假设分类问题中的输出为q,则计算公式如（2-3），（2-4）所示：

$$H = \emptyset(X_{N*D} W_{D*H} + B_{1*H}) \quad (2\text{-}3)$$

$$O = HW_{HQ} + B_{1*Q} \quad (2\text{-}4)$$

然后针对输出变量O，采用$softmax(O)$计算输出类别的概率分布。

考虑到输入数据具有时序性。假设X_t是时间 t 的一个小批量输入，Ht 是该时间的隐藏变量，在多层感知机的基础上，我们保存了上一步时间的隐藏变量 Ht-1，隐藏层输出改变为如公式（2-5）所示：

$$H_t = \emptyset(X_{n*d} W_{d*h} + H_{t-1} W_{h*h} + b_{1*h}) \quad (2\text{-}5)$$

图 2.5 展示了包含隐藏层的循环神经网络。

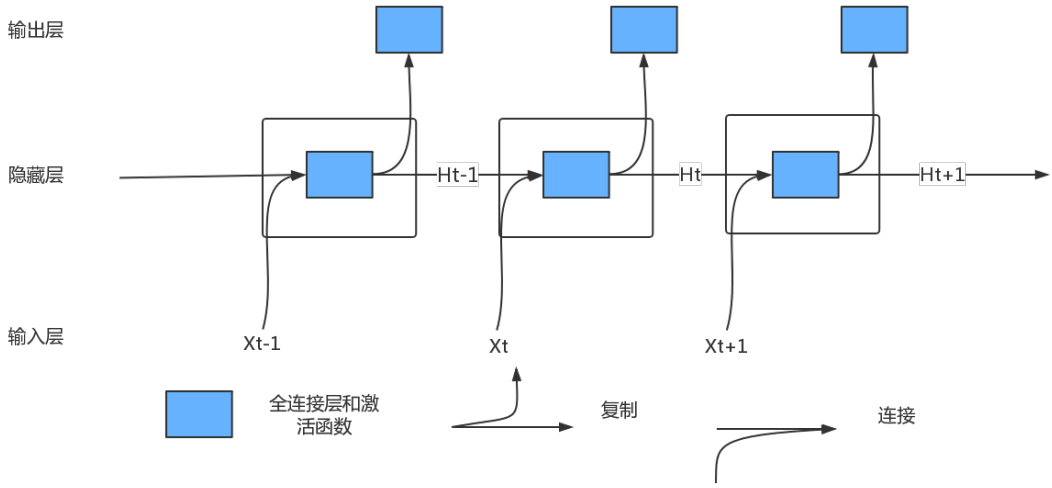

图 2.5 包含隐藏层的循环神经网络

于是，我们就可以利用这样的循环神经网络，训练语言模型，例如我们有一个批量数目为 1 的文本序列，“想学深度学习”，那么循环神经网络的模型建立如下图 2.6 所示：

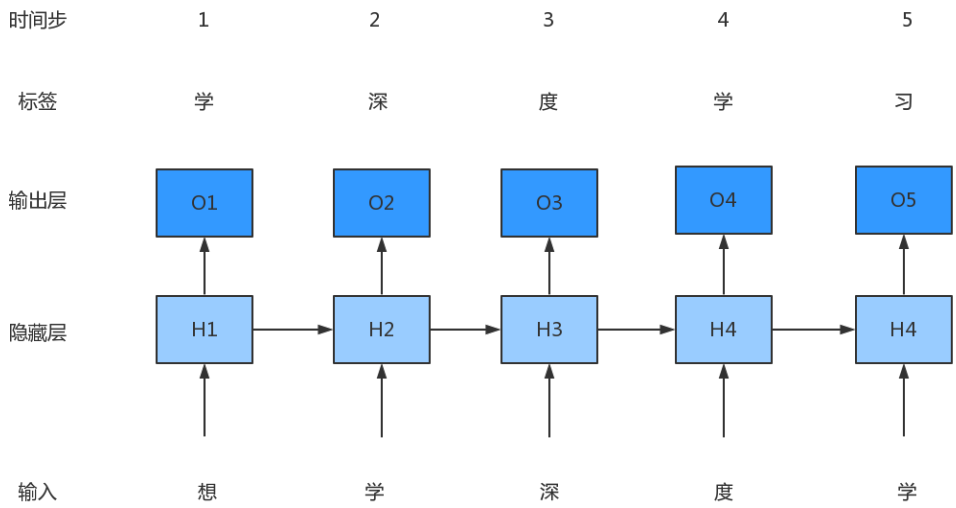

图 2.6 多层感知机的循环神经网络案例

每个输出层使用$softmax$计算，用交叉熵损失计算与标签的误差，由于在隐藏层存在循环，也就是说，某个输出节点，需要考虑到前面的文本序列。例如输出层O_3的输出就要考虑到前面的序列“想学深”，输出的概率分布与标签“度”损失。

2.3 强化学习

2.3.1 强化学习概述

如图 2.7 给出了强化学习的一个图示。可以通过马尔可夫决策(MDP)来描述一个强化学习的任务：神经网络处于环境E中，状态空间为X，其中每个状态x属于X是神经网络感知到的环境的描述；若某个动作a属于A作用于状态x上，则潜在的转移函数将状态s_tt转移到s_{t+1}，并且返回此次转移的奖励。

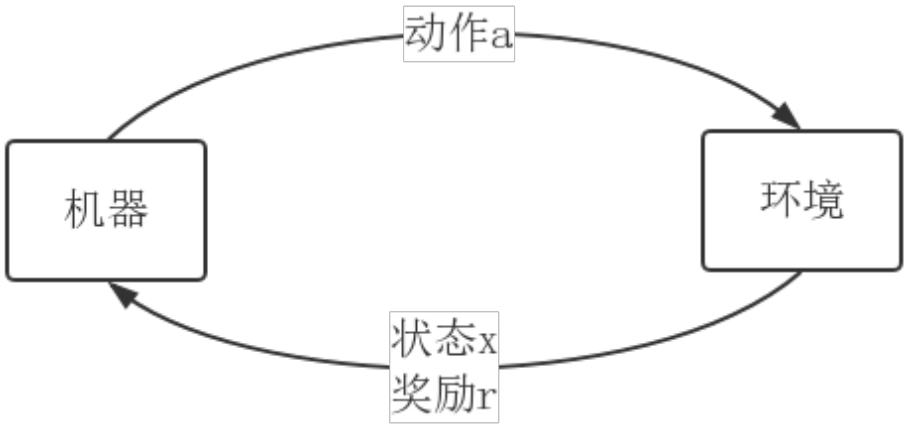

图 2.7 强化学习图示

这里的神经网络与环境的区别在于，在环境中的状态转移、奖赏是不受神经网络（或者机器）的控制的，神经网络只能通过选择要执行的动作来影响环境，也只能通过观察转移后的状态和返回的奖励来感知环境。

强化学习的就是要通过在环境中不断尝试而学习到一个策略（policy）π，根据这个策略，在状态 x 下就知道下一步执行的动作$\mathrm{a} = \pi(x)$；这里的决策有两种表现形式：

1. 确定性决策：$X \to \mathrm{A}$ ，输入一个状态，有确定的一个动作输出

2. 概率性决策：$X \times A \to R$ ，$\pi(x,a)$ 为状态 x 下选择动作 a 的概率，这里有 $\sum_a \pi(x,a) = 1$

决策的优劣不能通过一次测试来评价，而是需要看在环境中不断尝试得到的累积奖励。只要在环境中，长期表现优异的策略才是人们所真正需要的。其中长期累积奖赏有多种计算方法，常用的有两种：

1. “T 步累积奖赏”:$E[\frac{1}{T}\sum_{t=1}^{T} r_t]$
2. “γ折扣累积奖赏”:$E[\sum_{t=0}^{+\infty} \gamma^t r_{t+1}]$

其中r_t表示第 t 步获得的奖励值，E 表示对所有随机变量的期望。

依次看来，相比之前的机器学习，深度学习所不同的是，在使用监督学习时，事先准备了大量的样本和标签，而在强化学习中，没有人知道神经网络在什么状态下应该做什么决策，而是只有等到走到最后一步状态，才能反馈知道之前的动作是否正确。

依据马尔可夫决策过程的四元组$E =< X, A, P, R >$是否已知，如果全部已知，则为有模型学习，如果有未知的变量，则为免模型学习。

2.3.2 有模型学习

如果马尔可夫决策的四元组$E =< X, A, P, R >$均已知，是指状态在动作下进行状态转移时，转移到的下一个状态的概率，以及环境给予的奖励均已知。 对于此类情况，任意策略都能够确定的评测其优劣性。

函数 V(x)表示从状态 x 出发，使用策略π在环境下发出动作，转移状态，所带来的所有的累积奖励，这个函数也被称为状态值函数；将状态值函数更加具体一些，函数 Q(x, a) 表示在时用策略π时，从状态 x 开始，采取动作 a 后，能够得到的所有累积奖励，这个函数也被称为状态-动作值函数。根据累积奖励的两种方式，我们可以得到状态值函数如公式（2-6）所示：

$$\begin{cases} V_T^{\pi}(x) = E_{\pi}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x\right], & \text{T 步累积奖励;} \\ V_{\gamma}^{\pi}(x) = E_{\pi}[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x], & \gamma\text{折扣累积奖励.} \end{cases} \tag{2-6}$$

其中令x_0代表起始状态，a_0代表在x_0上执行的动作；对于 T 步累计奖励，用下标 t 表示时间步，可以得到状态-动作值函数如公式（2-7）所示：

$$\begin{cases} Q_T^{\pi}(x,a) = E_{\pi}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x, a_0 = a\right]; \\ Q_{\gamma}^{\pi}(x,a) = E_{\pi}[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | x_0 = x, a_0 = a. \end{cases} \tag{2-7}$$

由于马尔可夫的性质，即系统下一时刻的状态仅与当前状态有关，而不依赖于之前的任何状态，于是值函数就有了递归的形式，如公式（2-8）所示：

$$\begin{aligned} V_T^{\pi}(x) &= E_{\pi}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x\right] \\ &= E_{\pi}\left[\frac{1}{T} r1 + \frac{T-1}{T}\frac{1}{T-1}\sum_{t=2}^{T} r_t | x_0 = x\right] \\ &= \sum_{a\in A} \pi(x,a) \sum_{x'\in X} P_{x\to x'}^{a}\left(\frac{1}{T} R_{x\to x'}^{a} + \frac{T-1}{T} V_{T-1}^{\pi}(x')\right) \end{aligned} \tag{2-8}$$

同样，对于折扣累积奖励有值函数表达式，如公式（2-9）所示：

$$\mathrm{V}_{\gamma}^{\pi}(x) = \sum_{a\in A} \pi(x,a) \sum_{x'\in X} P_{x\to x'}^{a}(R_{x\to x'}^{a} + \gamma V_{\gamma}^{\pi}(x')) \tag{2-9}$$

于是，我们就可以通过这个递归，求出此策略的评估值。

状态-动作函数的计算，如公式（2-10）所示：

$$\begin{cases} Q_T^{\pi}(x,a) = \sum_{x'\in X} P_{x\to x'}^{a}\left(\frac{1}{T} R_{x\to x'}^{a} + \frac{T-1}{T} V_{T-1}^{\pi}(x')\right); \\ Q_{\gamma}^{\pi}(x,a) = \sum_{x'\in X} P_{x\to x'}^{a}\left(R_{x\to x'}^{a} + \gamma V_{\gamma}^{\pi}(x')\right). \end{cases} \tag{2-10}$$

以上，我们已经可以完成对某个策略的累积奖励进行评估，发现它不是最优的策略，那如何改进这种策略，将累积奖励最大化,理想的决策即为公式（2-11）所示：

$$\pi^* = argmax_{\pi} \sum_{x\in X} V^{\pi}(x) \tag{2-11}$$

与之对应的值函数V^*如公式（2-12）所示：

$$\forall x \in \mathrm{X}: V^*(x) = V^{\pi^*}(x) \tag{2-12}$$

由于最优值函数的累积奖赏值已经达到最大，因此可以对之前面的公式（2-8）和公式（2-9）改动，即将前面的对动作求和改为直接取最优，得到（2-13）所示：

$$\begin{cases} V_T^*(x) = max_{a\in \mathrm{A}} \sum_{x\prime\in \mathrm{X}} P_{x\to x\prime}^a \left(\frac{1}{T} R_{x\to x\prime}^a + \frac{T-1}{T} V_{T-1}^*(x')\right); \\ V_\gamma^*(x) = max_{a\in \mathrm{A}} \sum_{x'\in \mathrm{X}} P_{x\to x'}^a \left(R_{x\to x'}^a + V(x')\right). \end{cases} \quad (2\text{-}13)$$

同理，可以通过公式（2-10）推到得到Q_T^*和Q_T^*，得到公式 2.14 所示：

$$\begin{cases} Q_T^*(x,a) = \sum_{x\prime\in X} P_{x\to x'}^a \left(\frac{1}{T} R_{x\to x'}^a + \frac{T-1}{T} max_{a\prime\in A} Q_{T-1}^*(x')\right); \\ Q_\gamma^*(x,a) = \sum_{x\prime\in X} P_{x\to x'}^a \left(\frac{1}{T} R_{x\to x'}^a + \gamma max_{a\prime\in A} Q_\gamma^*(x')\right). \end{cases} \quad (2\text{-}14)$$

于是，我们可以公式(2-14)中，得到非最优策略的更新方式：**将策略选择的动作，改为当前最优的动作。**这样的改变一定能使策略更好,理由如下：

不妨改变后的策略为 π' ,改变动作的条件是$V^\pi\left(x,\pi'(x)\right) \geq Q^\pi(x)$，以 γ折扣累积奖励为例，可以得到递推不等式（2-15）：

$$\begin{aligned} \mathrm{V}(x) &\leq Q\left(x,\pi'(x)\right) \\ &= \sum_{x\prime\in X} P_{x\to x\prime}^{\pi\prime(x)} (R_{x\to x\prime}^{\pi\prime(x)} + \gamma V(x')) \\ &\leq \sum_{x\prime\in X} P_{x\to x\prime}^{\pi'(x)} \left(R_{x\to x\prime}^{\pi'(x)} + \gamma Q^\pi\left(x',\pi'(x')\right)\right) \\ &= \cdots \\ &= V^{\pi\prime}(x) \end{aligned} \quad (2\text{-}15)$$

因此，值函数对于策略的每一点改进都是向着最优策略靠拢的，因此对于当前策略π，可以放心的将其改成当前最优动作的策略，如公式（2-16）所示：

$$\pi'(x) = argmax_{a\in A} Q^\pi(x,a) \quad (2\text{-}16)$$

直到π与π'相同。于是，针对与有模型的强化学习，一定可以找到其最优策略。

2.3.3 免模型学习

在现实的强化学习任务中，环境转移的概率、奖励函数往往很难得知，针对这样的情况建模，被称为免模式学习。相比有模型学习，免模型学习具有两个困难。

1. 在不知道环境转移的概率，奖励函数的时候，就会导致模型无法按照全

概率展开，无法直接评估策略的优劣，因此，我们可以将这个决策进行多次实验，然后求取累积奖励的平均值。

2. 在此之前计算状态值函数 V 时，是通过状态-动作值函数 Q 来得到的。当模型已知使，我们可以通过计算得到 V，但是马尔可夫的四元组中有未知变量时，我们就很难求得到 V，于是转变为估计 Q，即估计每一对“状态-动作”的值函数。

想要较好的获得状态-动作值函数的估计，我们需要很多条不同的采样轨迹，来尽可能的覆盖到每个状态和动作。然而，我们的策略是确定的，即对于每个状态只会输出一个确定的动作，这样，我们动作序列以及状态转移序列都是相同的。于是，加入一个随机因子ϵ ，来解决上述问题，即以一定概率采取动作，如公式（2-17）所示：

$$\pi(x)=\begin{cases}\pi(x) & 以概率\ 1-\epsilon;\\ A中以均匀概率选取的动作， & 以概率\epsilon.\end{cases} \tag{2-17}$$

采取公式（2-17）的方案，就可以采样到不同的轨迹。

在某种未知模式下，从起始状态出发，使用某种策略，执行 T 步获得的轨迹：

$$<\mathrm{X}_0,\mathrm{A}_0,\mathrm{R}_1,\mathrm{X}_1,\mathrm{A}_1,\mathrm{R}_2,\ldots,\mathrm{X}_{T-1},\mathrm{A}_{T-1},\mathrm{R}_T,\mathrm{X}_T>$$

我们更新状态-动作值函数时，有两种方案：

1. 采集到一条完整的路径后，将最后的奖励取平均值，作为状态-动作值函数的近似。
2. 通过增量形式来更新状态-动作函数。对于一个状态-动作对(x,a),不妨假设给予 t 个采样已估计出值函数$Q_t^{\pi}(x,a)=\frac{1}{t}\sum_{i=1}^{t}r_i$,则在得到第 t+1 个采样r_{t+1}时，可以改写成如公式（2-18）所示：

$$\mathrm{Q}_{t+1}^{\pi}(x,a)=Q_t^{\pi}(x,a)+\frac{1}{t+1}\left(r_{t+1}-Q_t^{\pi}(x,a)\right) \tag{2-18}$$

对于γ折扣累积奖励来说，利用动态规划方法考虑到模型未知时使用状态-动作值函数如公式（2-19）所示：

$$\begin{aligned} \mathrm{Q(x,a)} &= \sum_{x' \in X} P^{a}_{x \to x'} \left(R^{a}_{x \to x'} + \gamma V^{\pi}(x') \right) \\ &= \sum_{x' \in X} P^{a}_{x \to x'} \left(R^{a}_{x \to x'} + \gamma \sum_{a' \in A} \pi(x', a') Q(x', a') \right). \end{aligned} \tag{2-19}$$

通过增量求和可得如公式（2-20）所示：

$$Q^{\pi}_{t+1}(x,a) = Q^{\pi}_{t}(x,a) + \alpha(R^{a}_{x \to x'} + \gamma Q^{\pi}_{t}(x',a') - Q^{\pi}_{t}(x,a)) \tag{2-20}$$

其中x'是前一次在状态x执行动作 a 转移到的状态，a' 是策略 π 在x' 上选择的动作，这也是本文 4.3 节中 Q-learning 算法更新状态-动作值函数的方式。

2.4 移动边缘计算

2.4.1 移动边缘计算发展

随着社会的发展，人们对网络传输速率和传输品质(QoS)的需求越来越高，现如今各种高要求服务和应用产生。虽然新的移动设备在中央处理器（CPU）方面计算能力越来越强大，但即使这些设备也不能在短时间内处理需要大量处理的应用程序，尤其是一些实时系统。此外，电池的消耗也是不可忽视的部分，现在的移动设备的电池，都普遍存在电池不够用，或者老化的情况，如果我们能够减少这些移动设备的计算量，降低其电池耗能，将更加便利于用户，移动云计算（MCC）为此而诞生了。可以将一些高耗能的计算需求，统一提交至强大的远程集中云去。

但是，移动云计算同样带来了其他的问题，例如网络的负担，如果用户设备距离移动云计算中心较远，则数据的不断传递，会严重影响到应用的延时，影响到用户的体验。

尽管如此，移动云计算在移动网络上的数据传递上也增加了巨大的额外负担，并且由于数据被发送到功能强大的服务器场（从网络拓扑结构来看，这些服务器远离用户），所以会带来很高的延迟。

为了解决长延迟的问题，云服务应该移动到 UE 的附近，即移动网络的边缘。

边缘计算可以作为MCC的一种特殊情况来进行。然而，在传统的MCC中，云服务是通过互联网连接访问的，而在边缘计算的情况下，计算和存储资源应该靠近UES（从网络拓扑的意义上讲）。因此，与移动云计算相比，移动边缘计算可以提供更优的通信质量。表2.1概述了MCC和边缘计算的关键技术方面的一些比较。

表2.1 移动云计算与移动边缘计算关键技术区别

指标	MCC	MEC
部署	集中式	分布式
与UE的距离	远	近
等待时间	高	低
计算能力	充足	有限
存储容量	充足	有限

2.4.2 移动边缘计算场景

移动边缘计算给很多人带来了便利，例如移动运营商，服务提供商，以及用户，依据这三类，我们把主要移动边缘计算的案例做了的一个划分。

A. 面向消费者的服务

第一个应用场景是面向消费者的，移动边缘计算直接对用户提供便利。在论文[20]-[26]中可知，其一般表现形式是，用户通过计算卸载从移动边缘计算中得到优化，这使得用户设备能够运行更多的高耗能，高运算的程序。例如Web加速浏览器，能够通过移动边缘计算卸载大多数的浏览功能到MEC服务器上；论文[18]中有关将Web加速浏览器卸载到MEC服务器的实验结果。此外，人脸识别、语音识别或图像、视频编辑也可以应用与移动边缘计算，因为这些应用需要大量的计算并且需要低延迟。还有，基于增强现实和虚拟现实的应用程序也可以利用向MEC的计算卸载得到优化，还有运行低延迟应用程序（如在线游戏或远程桌面）的用户可能会从附近的MEC中得到优化。在这种情况下，将在适当的移动边缘主

机上启动特定应用程序的新实例，以减少在 UE 上的应用程序的延迟和资源需求。

B. 运营商和第三方服务

第二类运营商和第三方可以从中受益的服务。例如从传感器收集大量数据。这些数据一方面可以通过智能终端进行预处理，然后在在发送到远程中央服务器之前，在 MEC 进行处理和分析。这可用于安全和安保目的，如监控区域（如停车场监控）。

另一个用例是利用 MEC 实现物联网。基本上，物联网设备可能使用了各种不同的通信协议，连接这些物联网设备就需要低延迟聚合点来处理各种协议。MEC 则可以充当物联网网关这一角色。

C. 网络性能和 QoE 服务

第三类用例是优化网络性能和改善 QoE(体验品质)的用例。一个这样的用例是实现无线电和回程网络之间的协调。到目前为止，总体网络性能还是会收到带宽的影响。在这方面，可以利用 MEC 服务器，可以利用网络的实时信息，来限制网络的流量。

还可以通过移动边缘的本地内容缓存来提高网络性能。MEC 服务器可以存储一些该地理区域中使用的最频繁的内容。避免不必要的网络传输。

2.4.3 计算卸载介绍

计算卸载是 MEC 的一个关键技术之一。一般来说，计算卸载的关键部分是决定是否卸载。在前一种情况下，还存在一个问题，即应卸载多少计算量以及应卸载哪些计算。基本上，对计算卸载的决定可能导致：

•本地执行——整个计算在都只能在本地执行（见图 2.8）。例如，由于 MEC 计算资源不可用，或者如果卸载没有得到优化，则不会执行到 MEC 的卸载。

•完全卸载——整个计算交付 MEC 卸载和处理。

•部分卸载——部分计算在本地处理，其余部分卸载到 MEC。

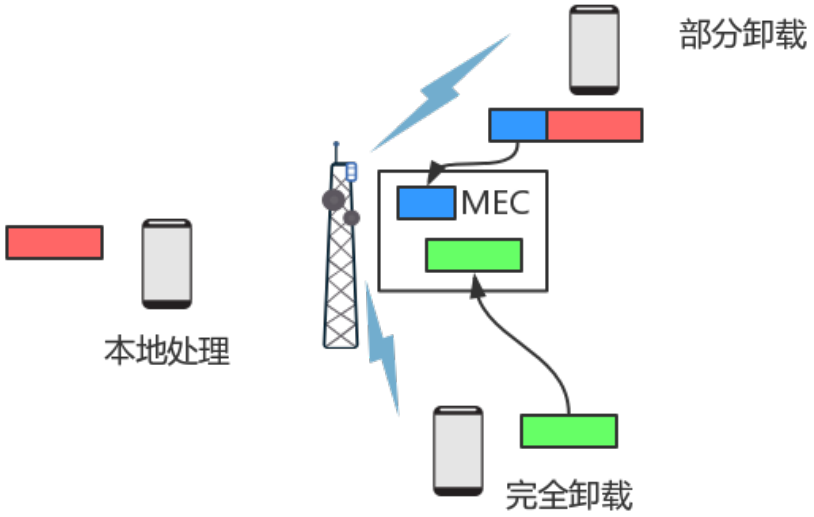

图2.8 计算卸载决策的可能结果

计算卸载和部分卸载是一个非常复杂的过程，受不同因素的影响，如用户偏好、无线链路质量、UE 功能。计算卸载中的一个重要方面也是应用程序模型/类型，因为它决定了 UE 是否能够完全卸载和部分卸载、可以卸载什么多少计算量以及如何卸载。在这方面，我们可以根据以下几个标准对应用程序进行分类：

- 是否具有可卸载性

  一个应用程序能否卸载，要看他的代码能否分区。有的应用程序能够全部卸载其计算，例如图 2.9a 所示，他的全部代码和数据都可以卸载，这里他选择了将 4，5，7，8，10 部分分区卸载，来远程处理。而有的应用程序则只能部分卸载其计算，例如他可能要处理用户输入等等，例如图 2.9b 所示。他的 1，3，4，5，8，9，10 可以通过计算卸载来远程处理，但是部分分区却不能做到。

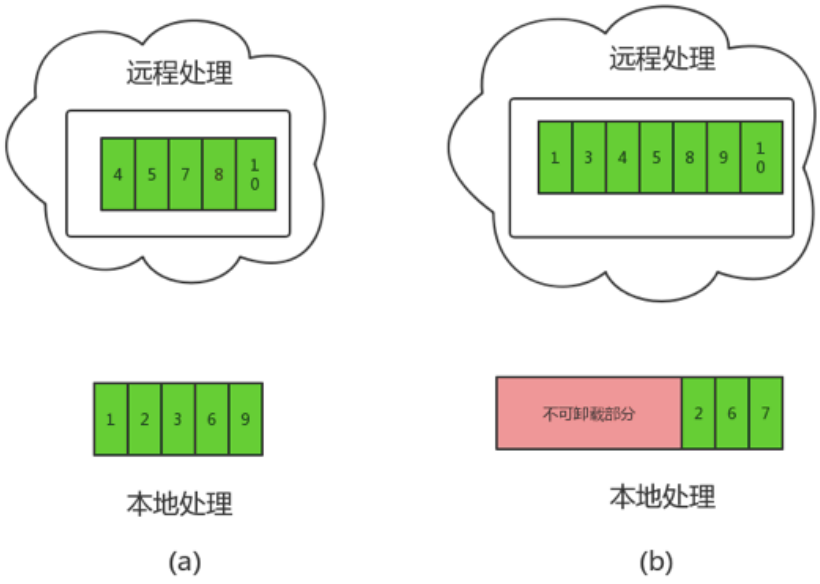

图2.9 可完全卸载的应用a ,不可完全卸载的应用b

- 是否能预测待处理数据量情况

  可以根据应用程序待处理的数据有效时间进行分类。例如有些应用要处理的数据量是预先知道的，比如一些确定的计算需求。但是有的应用程序去无法估计要处理的数据量，因为这些是需要连续执行应用程序，无法预测它们将运行多长时间（例如，在线互动游戏）。显然，对于连续执行应用程序来说，计算卸载的决策可能非常困难。

- 可卸载部分与不可卸载部分之间的依赖性

  如果应用程序同样可以分为多个代码，数据分区，但是却不能像第一点中所提到的任意的卸载，例如图 2.10 所示，整个图包括可卸载部分（2，3，5）和不可卸载部分（1，4，6），其中的箭头就是应用程序各部分之间的依赖关系，例如不可卸载部分 1，与可卸载部分 3，由于 3 的完成需要等待不可卸载部分的 1 的完成，于是，如果将可卸载部分的 3，通过计算卸载到 MEC 服务器上，则会造成 MEC 服务器资源的浪费。

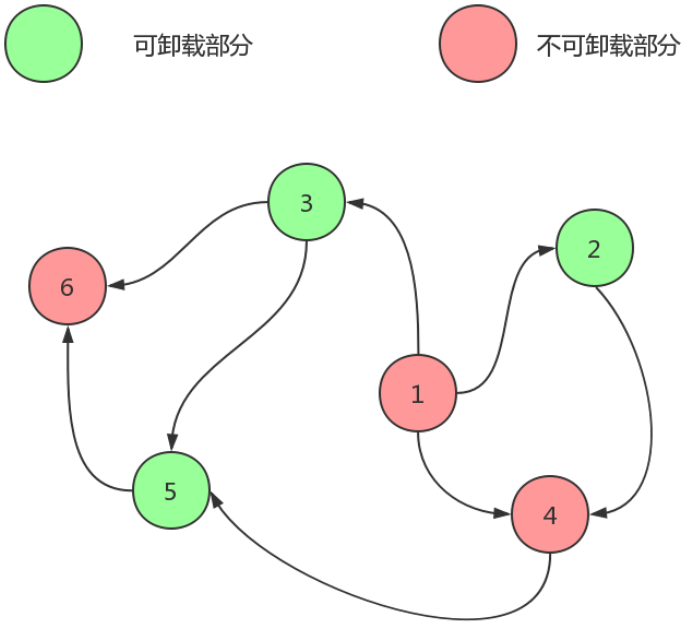

图2.10 可卸载部分与不可卸载部分的依赖性

计算卸载是移动边缘计算中十分重要的研究方向，计算卸载决策的优劣直接影响到整个移动边缘计算系统的性能，如何管理与决策计算卸载是至关重要的。对于用户部分，UE 的应用程序也十分重要，一般情况下，需要有代码探查器、系统探查器和决策引擎组成[19]。其功能分别是，探测代码是否可以卸载，已经哪些是可以卸载的；监控各种网络参数；最后决定是否卸载。

3　系统模型

3.1 网络模型

图 3.1 展示了本文讨论的网络模型，我们考虑了一个小单元的场景。

这里有一个基站(eNB)和 N 个用户设备（UEs），一个 MEC 服务器与eNB一起部署。

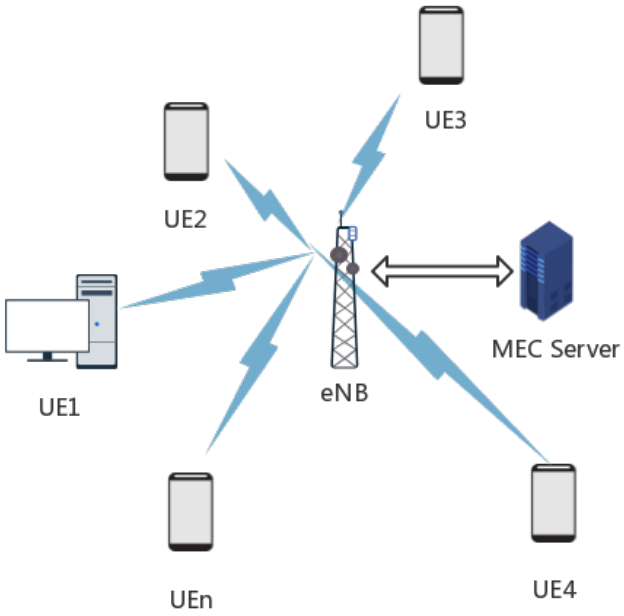

图 3.1 网络模型

UE 的集合表示为$N = \{1,2,\dots,N\}$。我们假设每个 UE 都有一个计算密集型任务要完成。每个 UE 都可以通过无线方式将任务完全卸载到 MEC 服务器或在本地执行任务。但是 MEC 服务器的容量有限，可能不足以让所有 UE 卸载任务。

我们将 W 定义为无线信道的带宽。一个小单元内只有一个 eNB，因此忽略了间隔干扰。假设多个 UE 同时选择卸载任务，无线带宽将平均分配给卸载计算的 UE 以上载数据。根据论文[11]，第 n 个用户设备 UE n 的可实现上传数据率如公式（3-1）所示：

$$r_n = \frac{W}{K} log\ (1 + \frac{P_n h_n}{\frac{W}{K} N_0}) \tag{3-1}$$

其中：各个参数的含义如表 3.1 所示。

表 3.1 UEn 可实现上传速率公式各参数含义

参数	含义
W	带宽
K	K 为计算卸载 UE 的数量
P_n	上传数据时 UE n 的传输功率
h_n	无线信道中 UE n 的信道增益
N_0	复杂高斯信道噪声的方差

3.2 任务模型

我们假设每一个用户设备UE_n都有一个计算密集型任务$R_n \triangleq (\mathrm{B}_n, \mathrm{D}_n, \tau_n)$，在本模型中，假设每一个任务都是可以完全卸载的[12]，这里的Bn是完成该任务所需要的数据量，包括一些要处理的数据，以及处理这些数据的代码。Dn表示完成计算任务所需的 CPU 周期总数，任务R_n和D_n与B_n的大小呈正相关。Dn反映完成任务R_n所需的计算资源量。我们假设无论是在本地还是在 MEC 服务器上执行，D_n的大小都保持不变。τ_n表示任务R_n的最大容许延迟，即每个用户的体验延迟不应超过τ_n，这将是我们优化问题的一个重要约束。这三个参数都与应用程序的特性有关，可以通过任务配置文件进行估计，因此不同类型的应用程序之间可能存在很大的差异。

我们假设任务不能划分为不同设备上要处理的分区（仅考虑不可卸载计算，和完全卸载计算），这意味着每个 UE 应该通过本地计算或者上传至 MEC 服务器计算，来执行其任务。我们将 $\alpha_n \in \{0,1\}$表示为UE_n 的计算卸载决策，并将计算卸载决策向量定义为$a = [\alpha_1, \alpha_2, \dots, \alpha_n]$。如果在本地计算来执行任务，则$\alpha_n = 0$，否则$\alpha_n = 1$。

3.3 计算模型

3.3.1 本地计算模型

如果计算设备UE_n选择在本地执行其任务R_n，那么我们将T_n^l定义为计算设备UE_n的本地执行延迟，它只包括本地 CPU 的处理延迟。然后我们将 f_n^l 表示为计算设备UE_n的计算能力（单位为每秒 CPU 周期数），不同的UE之间的计算能力可能不同。任务R_n的本地计算延迟如公式（3-2）所示：

$$\mathrm{T}_n^l = \frac{\mathrm{D}_n}{f_n^l} \tag{3-2}$$

我们将E_n^l作为任务R_n的相应能量消耗，表示为公式（3-3）：

$$E_n^l = z_n D_n \tag{3-3}$$

其中，z_n 表示完成任务R_n的每个 CPU 周期的能耗。根据文献[13]中的实际测量，我们设置公式（3-4）计算z_n：

$$z_n = 10^{-27}(f_n^l)^2 \tag{3-4}$$

结合时间成本公式（3-2）和能量成本公式（3-3），本地计算的总成本可以表示为公式（3-5）：

$$C_n^l = I_n^t \mathrm{T}_n^l + I_n^e E_n^l \tag{3-5}$$

其中，I_n^t和I_n^e表示任务R_n的时间和能源成本的权重。权重满足$0 \le I_n^t \le 1$、$0 \le I_n^e \le 1$、和 $I_n^t + I_n^e = 1$。

与任务模型中提到的三个符号一样，每种类型的任务的权重可能不同。为了简单起见，我们假设任务R_n的权重在无论是在本地计算还是在计算卸载过程中保持不变。

3.3.2 计算卸载模型

如果用户设备UE_n选择通过计算卸载来执行任务R_n，则整个卸载方法将分为

三个步骤。首先，UE_n需要通过无线接入网络向基站eNB上传输入数据（即程序代码和参数等），基站eNB将数据转发给 MEC 服务器。然后由 MEC 服务器分配部分计算资源来完成UE_n的计算任务，最后由 MEC 服务器将执行结果返回给计算设备UE_n。

A. 根据上述步骤，卸载计算的第一步所需的时间是传输延迟如公式（3-6）所示来计算得到：

$$T_{n,t}^{o} = \frac{B_n}{r_n} \tag{3-6}$$

其中，r_n 表示网络模型中提到的无线信道中的上传速率。

此时的第一步能量消耗如公式（3-7）所示来计算得到：

$$E_{n,t}^{o} = P_n T_{n,t}^{o} = \frac{P_n B_n}{r_n} \tag{3-7}$$

B. 对于计算卸载的第二步，所需时间是 MEC 服务器的处理延迟，可以给出公式（3-8） 所示来计算得到：

$$T_{n,t}^{o} = \frac{D_n}{f_n} \tag{3-8}$$

其中，f_n定义为分配的计算资源（单位为每秒 CPU 周期数），由 MEC 服务器完成任务R_n，F 定义为 MEC 服务器的整个资源。因为分配的资源总量不能超过整个计算量，那么它遵循$\sum_{n=1}^{N} \alpha_n f_n \le F$，在这一步中，我们假设用户设备$UE_n$保持空闲状态，并将空闲状态的功耗定义为$P_n^i$，相应的能耗可以由公式（3-9）计算得到：

$$E_{n,p}^{o} = P_n^i T_{n,t}^{o} = \frac{P_n^i D_n}{f_n} \tag{3-9}$$

C. 对于计算卸载的最后一步，所需的时间是返回处理后的结果的下载延迟，可以通过公式（3-10）计算得到：

$$T_{n,b}^{o} = \frac{B_b}{r_b} \tag{3-10}$$

其中B_b为处理结果返回值的大小，r_b为用户设备UE_n的下载数据的速率，但根据[14]、[15]，下载数据率一般很高，结果的数据量相比输入数据的数据量往往小得多，因此本文在此部分忽略了这一步的延迟和能耗。

根据公式（3-6） ，公式（3-7），公式（3-8）及公式（3-9），可以得到通过

计算卸载的方法用户设备UE_n的执行延迟为公式（3-11）所示：

$$T_n^o = \frac{B_n}{r_n} + \frac{D_n}{f_n} \tag{3-11}$$

和能源消费总和为公式（3-12）所示：

$$E_n^o = \frac{P_n B_n}{r_n} + \frac{P_n^i D_n}{f_n} \tag{3-12}$$

结合公式（3.11）时间成本和公式（3.12）能量成本，和本地计算一样，考虑每个任务的时间花费权重，和能量成本权重，于是可以计算得到总成本，计算卸载方案总成本可给出如下公式（3-13）：

$$\mathrm{C}_n^o = I_n^t \mathrm{T}_n^o + I_n^e E_n^o \tag{3-13}$$

于是可以求和，计算出所有用户设备的消耗，MEC 计算卸载系统中所有用户的总成本表示为公式（3-14）：

$$\mathrm{C}_{all} = \sum_{n=1}^{N} (1-\alpha_n) C_n^l + \alpha_n C_n^o \tag{3-14}$$

其中α_n表示UE_n的卸载决策，如果UE_n选择通过本地计算执行其任务，那么$\alpha_n = 0$，如果UE_n选择通过计算卸载，利用 MEC 服务器计算执行其任务，那么决策$\alpha_n = 1$。

3.4 问题模型

在本节中，我们将 MEC 系统的卸载和资源分配作为一个优化问题，本文的目标是在最大容许延误和计算能力的约束下，将 MEC 系统中所有用户的执行延迟和能耗相结合的总成本最小化，问题如公式（3-15）所示：

$$\begin{aligned}
&\mathrm{MIN}_{A.f} \sum_{n=1}^{N} \alpha_n C_n^o + (1-\alpha_n) C_n^l \\
&C1: \ \alpha_n \in \{0,1\}, \forall \mathrm{n} \in N \\
&C2: (1-\alpha_n) T_n^l \ \le \tau_n, \forall \mathrm{n} \in N \\
&C3: 0 \le f_n \le \alpha_n F, \forall \mathrm{n} \in N \\
&C4: \ \sum_{n=1}^{N} \alpha_n f_n \le \ F, \forall \mathrm{N} \in N
\end{aligned} \tag{3-15}$$

在公式（3-15）中$\mathrm{A} = [\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n]$向量是计算卸载的决策向量，$f = [f_1, f_2, f_3, \ldots, f_n]$是计算资源的分配向量。该优化问题的目标就是最小化整个系统的成本。

C1：代表每个 UE 选择本地执行或计算卸载该任务。

C2：代表该任务本地执行计算或计算卸载，时间成本不应超过该任务的最大延迟。

C3：代表 MEC 服务器分配的计算资源，若在本地执行则不予以分配计算资源，若计算卸载，则确保不超过总计算资源 F。

C4： 保证所有的选择计算卸载的任务，他们需要的计算资源不超过 MEC 的全部计算资源。

问题（3-15）可以通过寻找最优决策向量和计算卸载的资源分配来解决。然而，我们可以知道决策向量 A 是二元变量的可行集并且目标函数是非凸问题。此外，当用户数量增加时，问题(3-15)的大小会呈指数规模增加，因此，从论文[16]可以知道他是一个从背包问题扩展而来的非凸问题，它是 NP 难问题。因此这里不采用传统的优化方法求解这个 NP 困问题（3.15），而是采用强化学习法来寻找最优的 A 和 F。

4 解决方案

在本节中，我们首先详细定义了 DRL 代理的具体状态、行为和奖励。然后介绍了基于 Q 学习的主题。为了解决 Q-learning 维度灾难的困扰，在这里提出了一种 DQN 方法，该方法利用神经网络估计 Q-learning 的状态-动作值函数。

4.1 强化学习的要素

从第二章强化学习方法中，我们知道强化学习方法有几个关键要素，即状态、动作、奖励，具体到本文的模型：

1. 状态：系统状态由两个部分组成$s = (tc, ac)$。我们将tc定义为整个系统的总成本，$tc = C_{all}$；ac 是 MEC 服务器的可用计算容量，可以计算为$\mathrm{ac} = \mathrm{F} - \sum_{n=0}^{N} f_n$.

2. 动作：在我们的系统模型中，动作由两部分组成，分别是 n 个用户设备 UEs 的决策向量 $a = [\alpha_1, \alpha_2, \ldots, \alpha_n]$和资源分配向量$f = [f_1, f_2, \ldots, f_n]$。因此，动作向量可以用动作向量和资源分配向量的组合 $[\alpha_1, \alpha_2, \ldots, \alpha_n, f_1, f_2, \ldots, f_n]$来表示。

3. 奖励：对于每一步，代理环境在执行每一个可能的动作A后，都会在一定的状态S下，得到一个奖励$R(S, A)$，一般来说，奖励功能应该与目标功能相关。因为，我们的优化问题的目标是获得最小的成本，而强化学习的目标是获得最大长远报酬，因此奖励的价值应该与总成本的大小呈负相关，我们将奖励定义为归一化的如公式（4-1）所示：

$$\frac{tc_{local} - tc(s,a)}{tc_{local}} \tag{4-1}$$

其中tc_{local}是指所有UE_n均在本地执行任务时的总成本，$tc(s, a)$是在当前状态下的实际总成本。

然而，如果有越来越多的用户设备 UEs，动作空间将迅速增加，为了限制动作空间的大小，我们在学习过程之前提出了一个预分类步骤。对于UE_n，如果约束$T_n^l \leq \tau_n$不能满足，即在本地计算不能满足延迟，只能将这个任务上传至 MEC 服务器计算，可以直接得到这个决策期间α_n固定为 1，并且分配的资源应满足$f_n \geq \frac{Dn}{\tau_n - \frac{Bn}{rn}}$，这样，我们可以减少 a 和 f 的可能值，以限制强化学习代理的决策空间。

4.2 代理环境

为使用强化学习解决 MEC 计算卸载的问题，这里构造了一个 MEC 服务器的代理环境；包含几项重要参数和操作，如表 4.1 所示：

表 4.1. MEC 代理环境参数和功能表

参数和功能	含义
F	MEC 总计算资源
Get_Init_State()	随机决策计算卸载，以及资源分配。
Step()	根据指定卸载决策和资源分配决策，返回转移后的 MEC 状态，以及奖励。

这里代理环境 Step 函数的伪代码如下所示：

The environment accepts each UE's decision a, and resource application f, as well as the relevant parameters of these tasks.

Initialize tc, ac, tclocal

For i = 1, len(T) do

if $a_i == 0$:

$tc \mathrel{+}= The\ cost\ of\ local\ execution\ of\ task_i$

elif $a_i == 1$:

$tc \mathrel{+}= The\ cost\ of\ computing\ offload\ of\ task_i$

$tclocal \mathrel{+}= The\ cost\ of\ local\ execution\ of\ task_i$

ac = sum(f)

s’ = (tc, ac)

$reward = \frac{tclocal - tc}{tclocal}$

return s’, reward

4.3 Q-learning 算法

4.3.1 Q-learning 算法流程

依据第二章节强化学习的相关理论，我们可以知道正是要在此代理环境（并且属于免模式学习）下，做一个时序差分学习。Q-learning 正是在免模式学习模式下的一种使用γ折扣累积奖励，增量更新奖励的算法。本小节将描述 Q-learning 的详细细节，与代理环境下的结合。

为了解决时序差分决策问题，我们可以学习对每个行动的最佳价值的估计，定义为在采取该行动时,以及随后遵循最佳政策时预期的未来回报之和。在第二章节公式 2.7 中，我们可以得到在给定的策略π下，状态 s,动作 a 的真实值为公式（4-2）所示：

$$Q_{\pi}(s,a) \equiv E[R_1 + \gamma R_2 + \cdots | S_0 = s, A_0 = a] \qquad (4\text{-}2)$$

其中，$\gamma \in [0，1]$ 是一个折价因子，它权衡了即时和后期回报的重要性。

Q-learning 对每个（状态，动作）都有一个$Q(s,a)$值。算法需要计算并存储$Q(s,a)$在 Q 表中，该值可视为状态 s,在 a 动作下的长期奖励，也就是我们在第二章节中谈到的动作-状态值函数，对于 Q-learning,需要记录一个 Q 表，Q 表其结构如表 4.2 所示：

表 4.2 Q 表结构

Qtable	Action1	Action2	…	Actionn
S1	Q(s1,a1)	Q(s1,a2)	…	Q(sn,an)
S2	Q(s2,a1)	Q(s2,a2)	…	Q(sn,an)
S3	Q(s3,a1)	Q(s3,a2)	…	Q(sn,an)
...	…	…	…	…
Sn	Q(sn,a1)	Q(sn,a1)	…	Q(sn,a1)

然后关于 Q-learning 算法的状态-动作值函数$Q(s,a)$增量更新公式可以由公式（2-20）推到所得出公式（4-3）：

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}\right)}^{\text{learned value}}$$

(4-3)

带入到上述 Q 表化简得公式（4-4）：

$$qtable[s_t,a_t] += \alpha * (r_t + \gamma * max(qtable[s_{t+1}',a_{t+1}'] - qtable[s_t,a_t]))$$

(4-4)

其中，α 是学习率，代表该状态下有多少奖励会被学习。

其中，s_t，a_t 是当前状态和当前状态下执行的动作，s_{t+1}'，a_{t+1}' 是下一个状态，和下一个状态下选择的动作。

γ 作为超参数，可以用户自动调节，值得注意的是，如果 γ 趋向于 0，这意味着代理环境更加注重考虑直接奖励，如果 γ 趋向于 1，这意味着代理也非常关注未来奖励，证明如下：

我们简化一下qtable的更新公式，max $(qtable[s_{t+1}',a_{t+1}'] - aqtable[s,a])$ 部分，其实就是选择下一个状态s'下最优的动作对应的长期奖励，简化用$Q(s_{t+1})$代替，那么就有如下推导，如公式（4-5）所示：

$$\begin{aligned} Q(s_t) &= r_t + \gamma * Q(s_{t+1}) \\ &= r_t + \gamma * \left(r_{t+1} + \gamma * Q(s_{t+2})\right) \\ &= r_t + \gamma * \left(r_{t+1} + \gamma * (r_{t+2} + \gamma * Q(s_{t+3})\right) \\ &= r_t + \gamma * (r_{t+1} + \gamma * (r_{t+2} + \gamma * (r_{t+3} + Q(s_{t+4}))) \end{aligned}$$

(4-5)

将公式（4-5）展开得到公式（4-6）：

$$Q(s_t) = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \gamma^3 * r_{t+3} + \gamma^4 * r_{t+4} + \cdots \quad (4\text{-}6)$$

从公式（4-6）中得知，由于γ 参数的引入，使得算法不再只看到眼前的利益，而能看到未来的利益。注意，这里的极端情况：

当 $\gamma = 0$ 时：算法只能看到眼前的利益。

当 $\gamma = 1$ 时：算法能够清晰的看到后面状态的全部利益。

当$0 < \gamma < 1$时：算法对越靠后的利益越感到模糊。

以上，我们已经了解了 Q-learning 算法最核心的更新 Q 表的流程，下面是

Q-learning 算法和环境交互，学习经验的流程图，如图 4.1 所示：

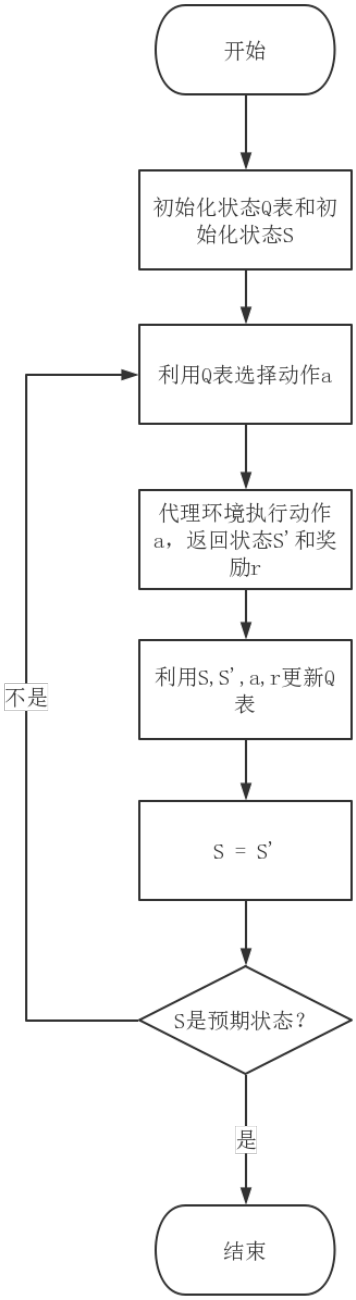

图 4.1 Q-learning 算法流程图

Q-learning 算法伪代码：

Algorithm 1 Q-learning method
Initialize Q(s,a)
for each episode:

Choose a random state s.

for each step. do

Choose an action from all possible actions of state s

Execute chosen a, observe reward and s'

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha R_{t+1} + \lambda max_a Q(s_{t+1}, a) - Q(s_t, a_t))$

Let $s_t \leftarrow s'$

Until reach expected state $s_{terminal}$

End for

4.3.2 Q-learning 与模型的结合

对应到本文的代理环境下，对于 Q 表这里需要修改两个方面的内容；

一方面：由于，我们有以下两大类决策

1. $A = [\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n]$ ——计算卸载决策向量。
2. $f = [f_1, f_2, f_3, \dots, f_n]$ ——MEC 服务器资源分配向量。

同时，对于每一个决策α_i 都有两种决策，因为$\alpha_i \in \{0,1\}$，同理，对于每一个资源分配决策，我们可以将它离散化，即$f_i \in \{0,1,2,\dots,F\}$ 其中 F 为 MEC 服务器全部可支配的资源。

另一方面：在 4.1 章节中，我们指出了，我们的状态的描述也是由两个部分组成的$s = (tc, ac)$

于是，对于我们的 Q 表，将是一个高维度的表格。于是，我们建立了两个 Q 表：

$qtable1[tc, ac, num\_ue, 2]$ 代表在状态$s = (tc, ac)$ 下，某一个 UE 选择计算卸载决策的状态-动作值函数。

$qtable2[tc, ac, num\_ue, (F+1)]$ 代表在状态$s = (tc, ac)$ 下，某一个 UE 选择计算卸载时，MEC 服务器分配多少计算资源的状态-动作值函数。

类似公式 4.3，在本环境下的状态-动作值函数的更新公式为公式(4-7)所示：

$$\begin{aligned}&\mathrm{q}table1[tc_t, ac_t, i, a]+\\&\quad=\alpha*(r_t+\gamma*\max(qtable1[tc', ac', i, a']-qtable1[tc, at, i, a]))\\&\mathrm{q}table2[tc_t, ac_t, i, \mathrm{f}]+\\&\quad=\alpha*(r_t+\gamma*\max(qtable2[tc', ac', i, f']-qtable2[tc, at, i, f]))\end{aligned}\tag{4-7}$$

例如：在某一个时间步 t 时，在状态$\mathrm{s}=(\mathrm{tc},\mathrm{ac})$，第 i 个 UE 选择了决策计算卸载，并分配了 MEC 服务器的 f 的计算资源，环境给予的奖励为r，转移到状态状态$\mathrm{s}'=(\mathrm{tc}',\mathrm{ac}')$那么此刻的 Q 表更新方法如公式（4-8）所示：

$$\begin{aligned}&\mathrm{q}table1[\mathrm{tc},\mathrm{ac},i,1]+\\&\quad=\alpha*(\mathrm{r}+\gamma*\max(qtable1[tc', ac', i, a']-qtable1[tc, ac, i, 1]))\\&\mathrm{q}table2[\mathrm{tc},\mathrm{ac},i,f]+\\&\quad=\alpha*(\mathrm{r}+\gamma*\max(qtable2[tc', ac', i, f']-qtable2[tc, ac, i, f]))\end{aligned}\tag{4-8}$$

4.5 DQN 算法

4.5.1 DQN 算法概述

尽管 Q-Learning 可以通过获得最大的奖励来解决问题，但它并不是最好的方案。如果我们使用 Q-Learning，这意味着对于每个状态动作组，我们需要计算并将其对应的 Q 值存储在一个表中。但在实际问题上，可能的状态数可能超过 100 万个。如果我们将所有的 Q 值存储在 Q 表中，矩阵 Q（s，a）将非常大，尤其在本实验环境下，我们从 4.4 章节就已经了解到，我们的状态空间是二维的，UE 数目同样需要一个维度，还包括了两大类决策，Q 表的维度已经到达一定的高度了。因此很难获得足够的样本来遍历每个状态，这将导致算法效率不够，也就是 Q-learning 的维度灾难。

DQN 顾名思义就是将深度学习（Deep learning）和（Q-learning）相结合的方法。DeepMind 公司也就是用 DQN 从玩各种电子游戏开始，直到训练出阿尔法狗打败了人类围棋选手。

我们使用深度神经网络来估计 Q（s，a），而不是计算每个状态-动作对的 Q 值，这是深度 Q 网络（dqn）的基本思想，常见的深度学习网络在第 2.2 章有谈到，例如全连接的，CNN，以及 RNN 等等。

4.5.2 DQN 算法流程

基本思想：相比 Q-learning 算法而言不再通过存储完整的 Q 表来决策，而是利用神经网络，如图 4.2 所示：

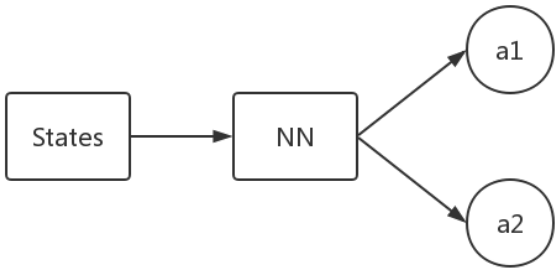

图 4.2. 使用神经网络估计 Q 值

DQN 算法流程如下：

1．存储一定的$(s_t, a_t, r_{t+1}, s_{t+1})$经验池。

其中s_t是当前状态，a_t是在 s_t状态下进行的动作，s_{t+1}是s_t在a_t动作下到达的下一个状态，r_{t+1}是s_t状态到s_{t+1}状态时，环境给予的奖励。

2．经验池中采样一个小批量 batch。

S_t，S_{t+1}放进网络，在 a 下的状态-动作值函数与S_{t+1}下的$maxQ(s_t+1)=next_qvalue$，计算期望决策的动作值函数，如公式（4-9）所示：

$$expecte\_qvalue = reward + \gamma * next\_qvalue \quad (4\text{-}9)$$

3. $Loss = Lossfunction(qvalue, expected\_qvalue)$。

这里损失选择 L2 范数损失，即平方损失。

如图 4.3 所示，就是训练网络时的损失的构造过程：

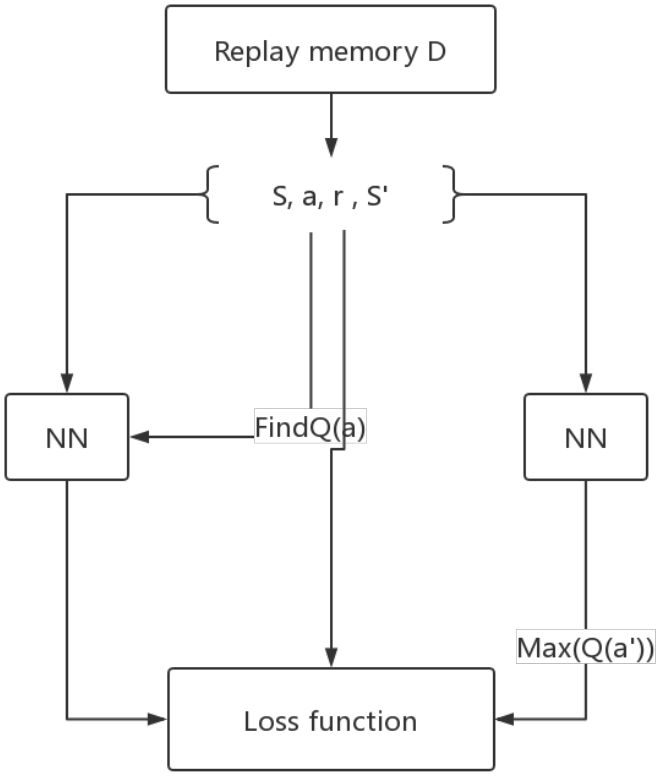

图 4.3 DQN 损失构造过程

以上，我们已经了解了 DQN 算法损失函数构造的流程，下面是 DQN 算法和环境交互，学习经验的流程图，和 Q-learning 的流程类似，初始化经验池，和神经网络，以及状态 s，利用ε-贪心策略，选择移动动作 a，环境在状态 s 下执行动作 a,返回下一状态，以及奖励，这样，我们有可以更新这个经验池，从算法长远意义上来说，这个经验池，存储的样本应该越来越好，因此这里的经验池应该采用先进先出的结构，然后在需要的时候，在经验池中采样小批量数据，更新神经网络，以便今后能够采取到更优，更能考虑到长远期望奖励的动作，直到算法收敛，或者状态下一状态以及不合法为止。DQN 的全部流程，如图 4.4 所示：

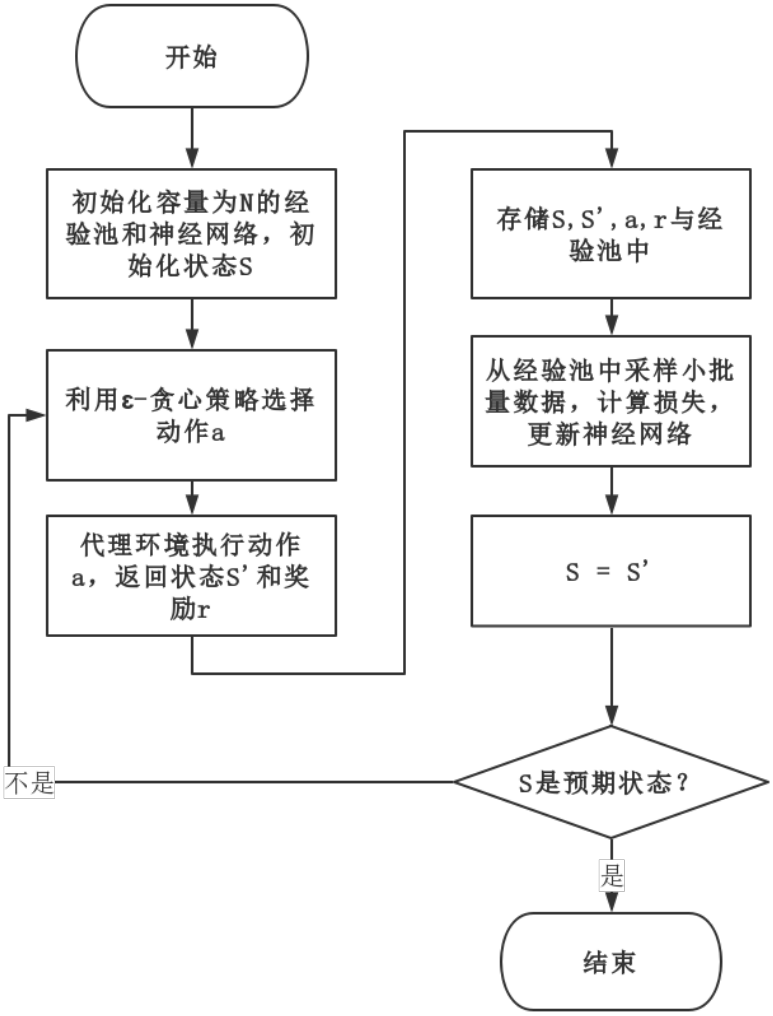

图 4.4 DQN 算法流程图

DQN 算法伪代码：

Algorithm 2 Deep Q-learning with Experience Delay

Initialize replay memory D to capacity N

Initialize Q table

for epoch = 1, M do

Initialize sequence $s_1 = x_1$ and preprecessed sequenced $\varphi_1 = \varphi(s_1)$

for i = 1, T do

With probability ε select random action a_t

otherwise select $a_t = \max_a Q\ (\varphi(s_t), a; \theta)$

Execute action a_t in environment and return the reward r_t and the state x_{t+1}

Set $s_{t+1} = s_t$, a_t, x_{t+1} and process $\varphi_{t+1} = \varphi(s_{t+1})$

Store transition $(\varphi_t\ a_t\ r_t\ \varphi_{t+1})$ in D

Sample random minibatch of transitions $(\varphi_j, a_j, r_j, \varphi_{j+1})$ from D

Set $y = \begin{cases} r_j & for\ terminal\ \varphi_{j+1} \\ r_j + \gamma max_{a'} Q(\varphi_{j+1}, a'; \theta) & otherwise \end{cases}$

Perform a gradient descent step on $(y_j - Q(\varphi_j, a_j; \theta))^2$ with respect to the network parameters θ

end for

end for

4.5.3 DQN 与模型的结合

在建立 Q-learning 算法模型时，我们已经了解到，我们的模型考虑的一个多用户，两大类的决策：

1. $A = [\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n]$
2. $f = [f_1, f_2, f_3, \dots, f_n]$

且 $\alpha_i \in \{0,1\}$，$f_i \in \{0,1,2,\dots,F\}$，于是在网络的输出层，节点数目应该节点数目 $*2+$ 节点数目 $*(F+1)$，每个节点代表某一个 UE 用户的计算卸载的决策，以及 MEC 服务器给他分配的计算资源的最大长期奖励。如图 4.5 所示，这里采用的是全连接层构造网络，其输出部分包含了所有的决策部分：

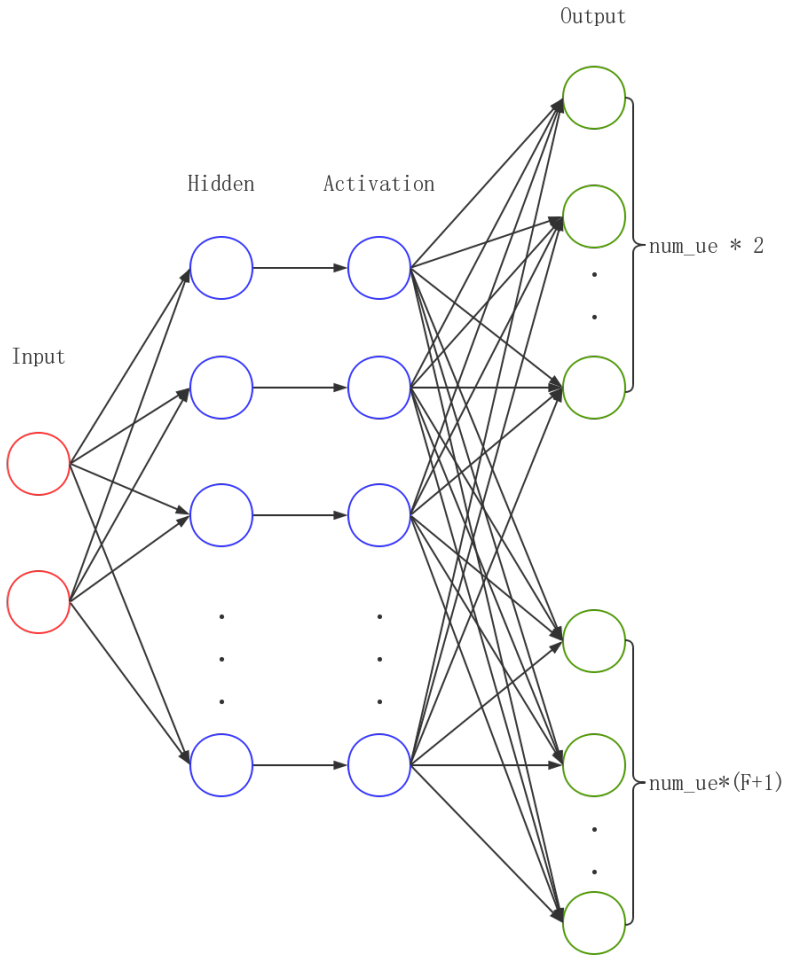

图 4.5 神经网络结构

网络构造好之后，接下来是利用这个网络构造损失函数，在 4.5.2 小节中，我们已经了解如何构造损失，我们将经验池中随机采样一个小的批次 batch，$(s_t, a_t, r_{t+1}, s_{t+1})$，拿出$s_t$，$s_{t+1}$，放进网络中，计算出$qvalues$，以及$next_qvalues$，由于计算出来的是所有决策的最大奖励期望值，例如：UE_1 通过比较$next_qvalues$发现应该采取本地计算，$expected_qvalues$的采取计算卸载的决策所得期望应该和$qvalues$保持一样，只需要将奖励增加在采取本地计算的单元上。如图 4.6 所示，是将奖励增加在采取本地计算单元的过程：

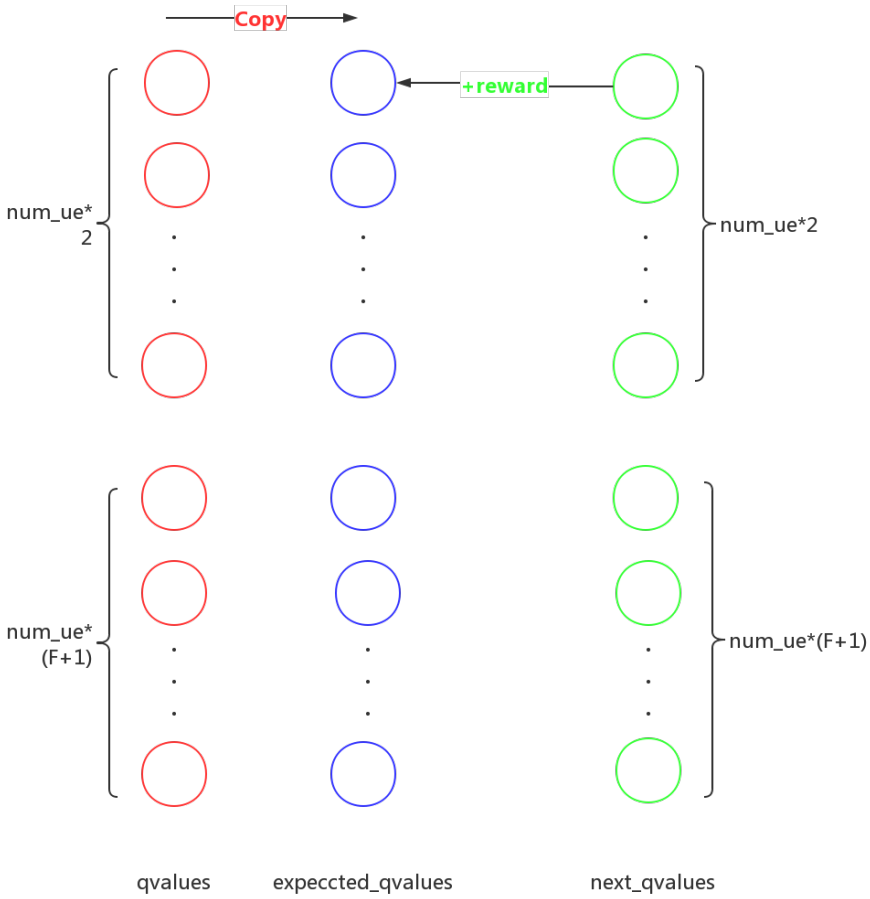

图 4.6. DQN 将奖励增加在本地计算单元过程

例如，图 4.6，发现第一个 UE 通过对比，发现应该选择本地计算，于是，找到输出单元的地方，即第一个神经元，增加其奖励值。

同样，如果UE_1通过比较$next_qvalues$发现应该采取计算卸载，则需要继续更具$next_qvalues$的计算资源分配单元，选择最大的奖励期望单元，更新$expected_qvalues$的决策单元与资源分配单元。如图 4.7 所示是奖励增加在采取计算卸载单元的过程：

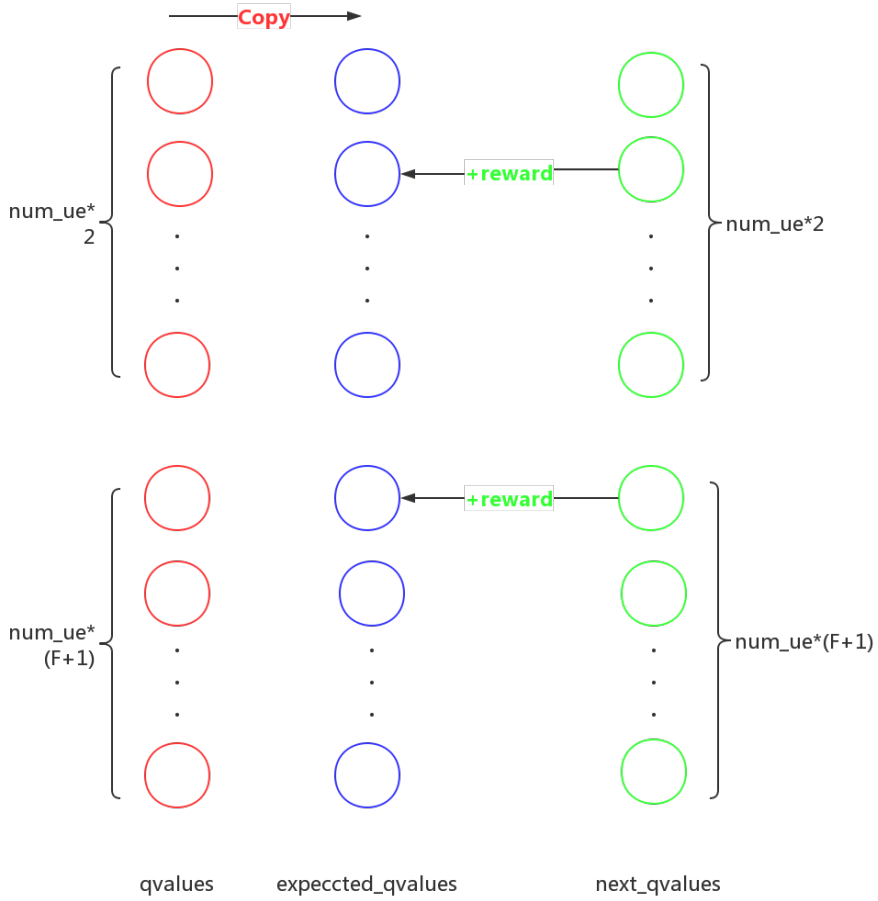

图 4.7. DQN 将奖励增加在计算卸载单元过程

例如，图 4.7，发现第一个 UE 通过对比，发现应该选择计算卸载，并且，应该选择申请 MEC 资源 2 个 GHz/sec 于是，找到输出单元的地方，即第 num_ue*2 + 1+2 个神经元，增加其奖励值。

4.5.4 优化函数

本文在更新神经网络时，采用的小批量随机梯度（sgd）下降。

假设连续可导的函数 $f:\mathbb{R}\to\mathbb{R}$ 的输入和输出都是标量。给定绝对值足够小的数 ϵ ，根据泰勒展开公式，我们得到如公式（4-10）所示的近似：

$$f(\mathrm{X}+\epsilon)\approx f(\mathrm{X})+\epsilon f'(\mathrm{X}). \qquad (4\text{-}10)$$

接下来，找到一个常数 $\eta>0$ ，使得 $\|\eta f'(\mathrm{x})\|$ 足够小，那么可以将 ϵ 替换为 $-\eta f'(\mathrm{x})$得到公式（4-11）：

$$f(\mathrm{X}-\mathrm{H}f'\ (\mathrm{X}))\approx f(\mathrm{X})-\mathrm{H}f'(\mathrm{X})^2 \tag{4-11}$$

如果导数 $\mathrm{f}'\ (\mathrm{x})\neq 0$ ，那么 $\eta f'(\mathrm{x})^2>0$ ，所以得到公式（4-12）所示的不等式：

$$f(\mathrm{X}-\mathrm{H}f'\ (\mathrm{X}))\lesssim f(\mathrm{X}) \tag{4-12}$$

公式（4-11）说明了，可以通过$\mathrm{x}\leftarrow \mathrm{x}-\eta f'(\mathrm{x})$来迭代 x ，函数 $f(\mathrm{x})$ 的值可能会降低。因此在梯度下降中，我们先选取一个初始值 x 和常数 $\eta>0$ ，然后不断通过上式来迭代 x ，直到达到停止条件。

目标函数的输入为向量，输出为标量。假设目标函数 $f:\mathbb{R}^d\rightarrow\mathbb{R}$ 的输入是一个 d 维向量 $x=[x_1,x_2,\ldots,x_d]^\top$ 。目标函数 $f(x)$的导数则改为偏导数，如公式(4-13)所示：

$$\nabla_x f(x)=[\frac{\partial f(x)}{\partial x_1},\frac{\partial f(x)}{\partial x_2},\ldots,\frac{\partial f(x)}{\partial x_d}]^\top \tag{4-13}$$

其中每个偏导数元素 $\frac{\partial \mathrm{f}(\mathrm{x})}{\partial \mathrm{x}_i}$ 代表着 f 在 x_i 的变化率。同理：可以通过$x\leftarrow x-\eta\nabla_x f(x)$迭代来降低目标函数$f$ 的值。

而本文使用的 sgd， 则在迭代时，不再使用全部训练数据，而是随机采样一个小批量的数据。

例如回归类型的小批量随机梯度下降中。有模型如公式（4-14）所示：

$$y\ =x_1w_1+x_2w_2+b \tag{4-14}$$

假设采集的样本数为 n ，第 i 个的样本的特征为 $\mathrm{x}_1^{(\mathrm{i})}$ 和 $\mathrm{x}_2^{(i)}$ ，标签为 y(i) 。对于第 i 个样本，线性回归模型的预测表达式为如公式（4-15）所示：

$$\hat{y}\ (i)=x_1^{(i)}w_1+x_2^{(i)}w_2+b. \tag{4-15}$$

仍然使用 L2 损失，则在小批量随机梯度下降中，参数的更新如公式（4-16），（4-17）和（4-18）所示：

$$w_1\leftarrow w_1-\frac{\eta}{|\mathrm{B}|}\sum_{i\in\mathrm{B}}x_1^{(i)}(x_1^{(i)}w_1+\ x_2^{(i)}w_2+b-y^{(i)}) \tag{4-16}$$

$$w_2\leftarrow w_2-\frac{\eta}{|\mathrm{B}|}\sum_{i\in\mathrm{B}}x_2^{(i)}(x_1^{(i)}w_1+\ x_2^{(i)}w_2+b-y^{(i)}) \tag{4-17}$$

$$b\leftarrow b-\frac{\eta}{|\mathrm{B}|}\sum_{i\in\mathrm{B}}(x_1^{(i)}w_1+\ x_2^{(i)}w_2+b-y^{(i)}) \tag{4-18}$$

5 仿真实验结果

在本节中，将给在模拟 MEC 代理环境上实验的结果，以评估所提出的模型的性能。在仿真环境中，我们假设一个场景如下。我们考虑一个使用带宽$W = 10MHz$的单个小单元，并且一个部署了 MEC 服务器的 ENB 位于中心。UE在距基站eNB 200 米范围内随机分布。MEC 服务器的计算能力为$F = 5GHz/sec$，每个 UE 的 CPU 频率为$f_n^l = 1GHz/sec$。UE 的传输功率和空闲功率设置为$Pn = 500mw$和$Pni = 100mw$[17]。我们假设计算卸载bn的数据大小（以 kbit 为单位）服从(300,500)之间的均匀分布，CPU 周期数dn（以兆周期为单位）服从（900，1100）之间的均匀分布。此外，为了简单起见，每个 UE 的决策权重设置为$\mathrm{I}_n^t = \mathrm{I}_n^e = 0.5$。

我们将所提出的算法与其他两种方法在以下几个参数上进行了比较。“Full Local”是指所有的用户通过本地计算来执行他们的任务。“Full Offload”是指所有 UE 将任务都卸载到 MEC 服务器，并且 MEC 服务器对所有任务分配的资源均等。

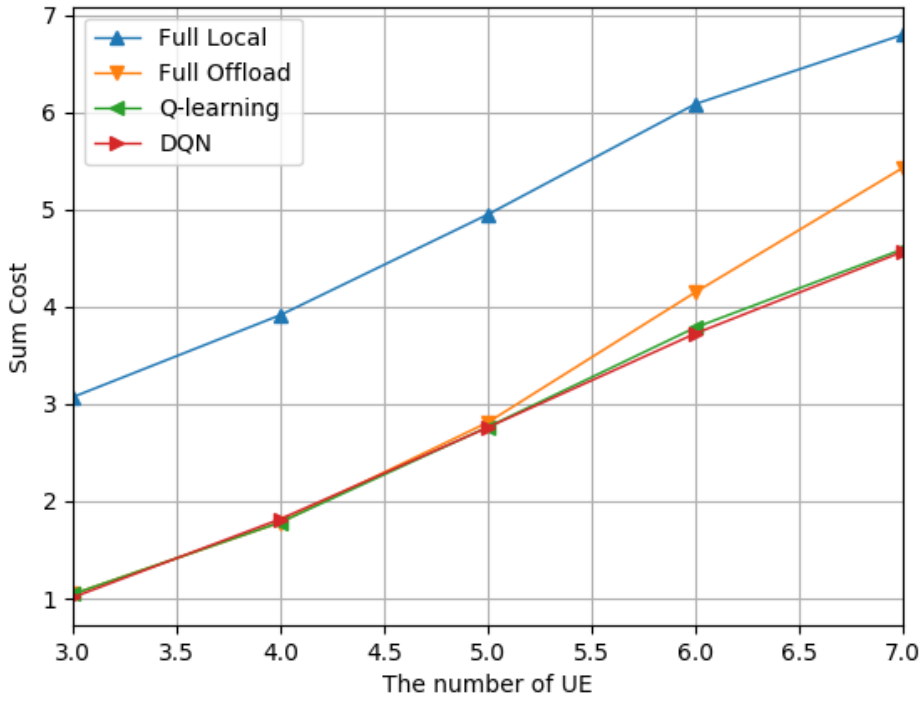

图 5.1 capacity 为 5 时，总代价与 UE 数目的关系

如图 5.1 所示，我们首先给出了以 UE 数量为横坐标，MEC 系统的总成本为纵坐标，其中 MEC 服务器的计算能力为 F=5GHz/sec。从整体上看，当使用单位数量不断增加时，四种方法的总成本都在增加。在图 5.1 中，所提出的 DQN 方法可以达到最佳的学习效果，然后 Q 学习有一个小的差距，这两种方法的性能相对稳定。在 3 个 UES 点上，全部计算卸载的曲线略高于 DQN 和 Q 学习，但当有更多的 UE 时，其增长速度更快。这是因为当用户数变大时，MEC 服务器的容量不足以通过卸载计算来提供所有的 UE。一个容量有限的 MEC 服务器不应该服务于太多的 UE，因此在这种情况下，如何选择已卸载的 UE 变得非常重要。

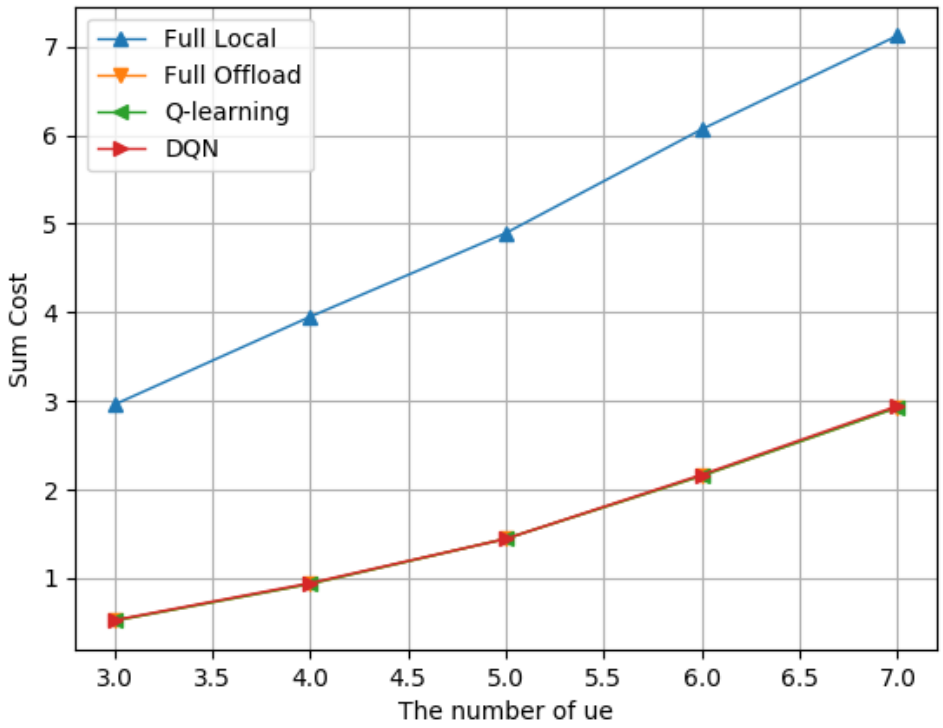

图 5.2 capacity 为 10 时，总代价与 UE 数目的关系

如图 5.2 所示，我们以 UE 为横坐标，MEC 系统总代价为纵坐标，更改 MEC 服务器的计算能力为 F=10GHz/sec。从整体上看，当使用单位数量不断增加时，四种方法的总成本都在增加。但是，完全卸载， Q-learning，DQN 三种方案，曲线大致重合，说明了如果 MEC 服务器的计算资源足够量大，那么只要 MEC 服务器计算资源允许，则计算卸载往往能够得到更好的效果。

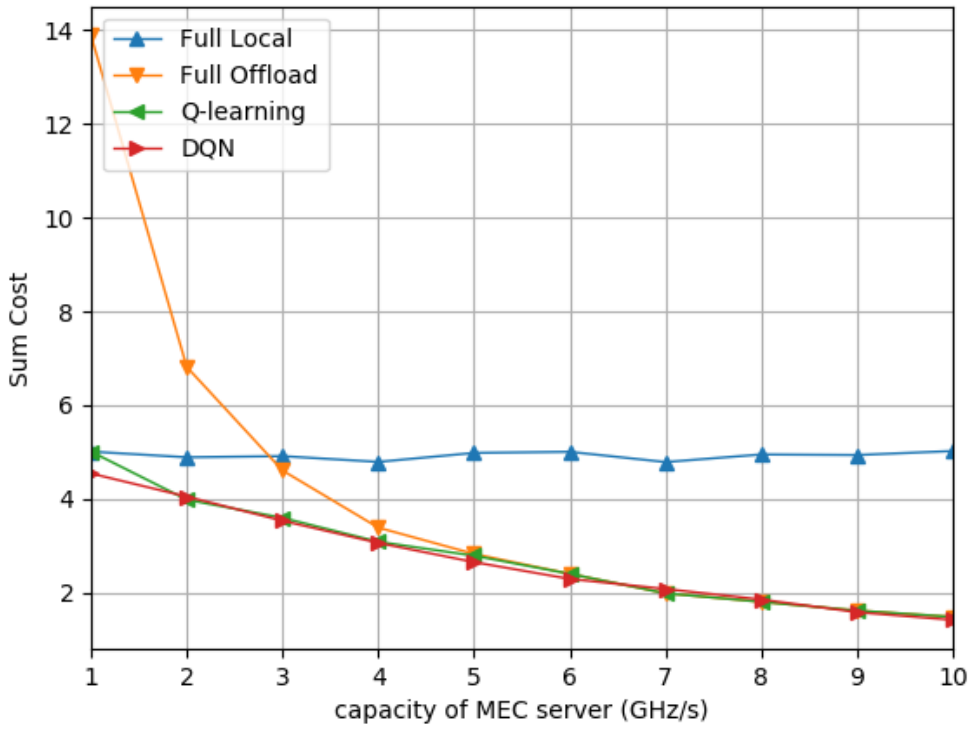

图 5.3 UE 数量为 5 时，系统总代价与 MEC 计算能力的关系

如图 5.3 所示，显示了以 MEC 服务器计算能力为横坐标，MEC 系统总成本为纵坐标，其中，UE 数为 5。从图中可以看出，所提出的 DQN 方法可以达到最佳的学习效果，而 Q 学习方法与 DQN 的差距很小。在图中，全部本地计算不会随着 MEC 服务器容量的增加而改变，这是因为本地计算不使用 MEC 服务器的计算资源。其他曲线随着 MEC 服务器计算能力的增加而减小，因为如果每个 UE 被分配更多的计算资源，执行时间会变短。此外，当 F>8GHz/s 时，完全卸载、Q 学习和 DQN 的总成本缓慢下降，这些卸载方法的性能几乎相同。结果表明，MEC 系统的总成本主要受其他因素的制约，例如，当 MEC 服务器上的计算资源比本地计算多时，传输速率就是限制的因素。

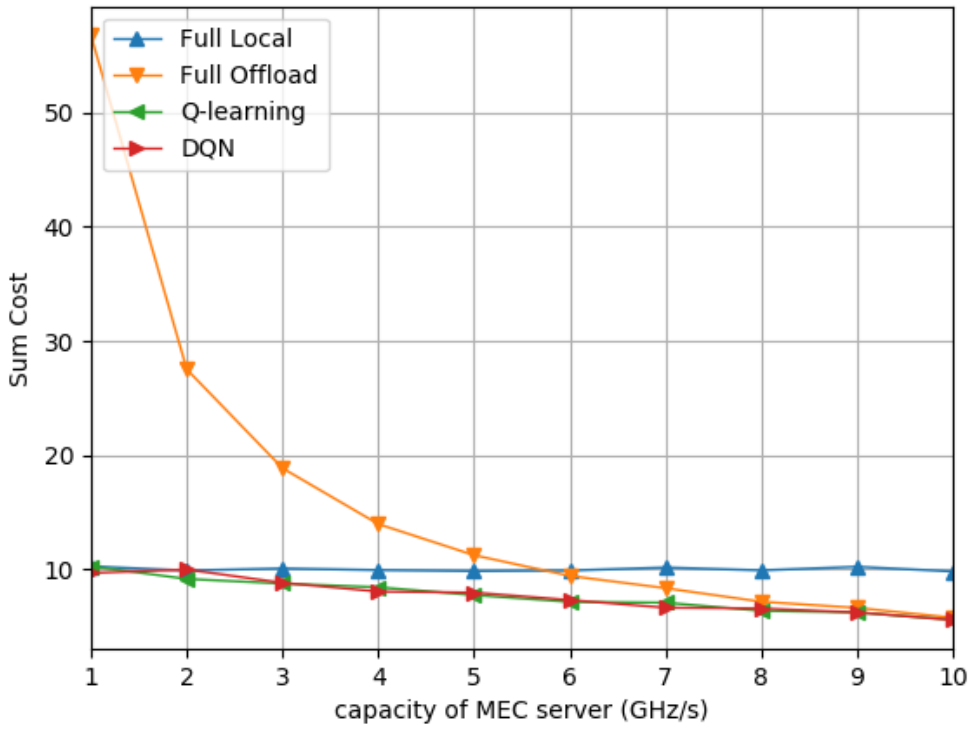

图 5.4 UE 数量为 10 时，系统总代价与 MEC 计算能力的关系

如图 5.4 所示，以 MEC 服务器计算能力为横坐标，MEC 系统总成本为纵坐标，其中，UE 数为 10。其趋势同图 5.3 类似，DQN 仍然表现出良好的性能，当 F>8GHz/s 时，完全卸载、Q 学习和 DQN 的总成本缓慢下降，这些卸载方法的性能几乎相同。同图 5.3，展现了 DQN 扩展到多用户计算卸载的场景良好的稳定性，因此，可以得出，DQN 即避免了像 Q-learning 遇到的维度灾难的情况，又可以较为容易的适应多用户计算卸载的场景。

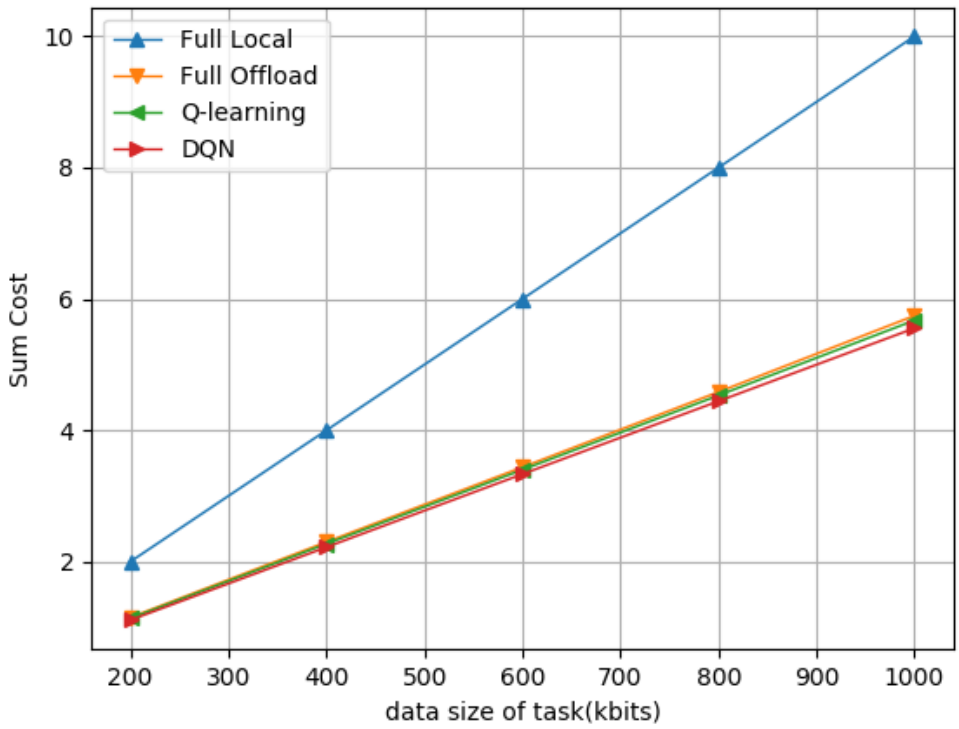

图 5.5 UE 数量为 5 时，系统总代价与任务量的关系

如 5.5 所示，以任务传输大小 Bn 为横坐标，MEC 系统的总成本为纵坐标，其中，UEs 数量为 5，所有方法的总成本随着卸载任务的数据大小的增加而增加，因为较大的数据大小会导致更多的传输时间和能源消耗。增加 MEC 系统的总成本。所提出的 DQN 方法由于其增长趋势慢于其它方法，可以达到最佳效果。随着数据量的增加，全部本地计算的增长速度比其它三种方法快得多，这表明任务的数据量越大，计算的延迟和能耗效益就越大。此外，我们可以看到，数据量的增加会导致 MEC 系统总成本的显著增加，这是因为 Dn 和 Bn 之间目标函数的双倍增长正相关。

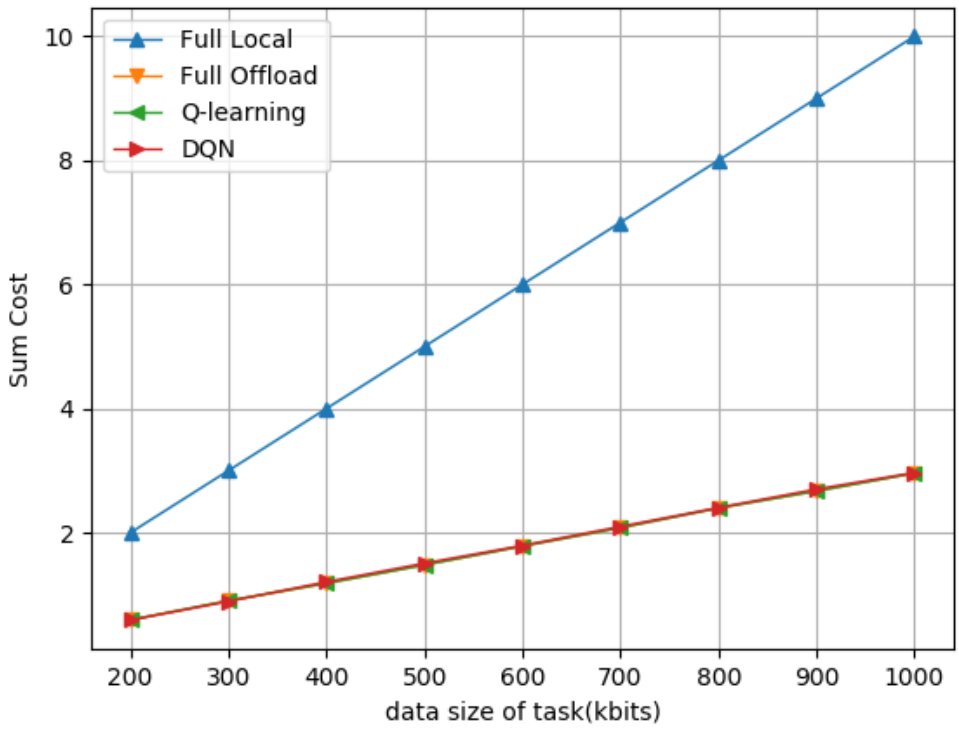

图 5.6 capacity 为 10 时，系统总代价与任务量的关系

如图 5.6 所示，以任务量大小为横坐标，系统总代价为纵坐标，UE 数目为 5，MEC 服务器计算能力为 10GHz/sec；和图 5.5 类似，所有方法的总成本随着卸载任务的数据大小的增加而增加，由于 MEC 服务器计算能力的提高，完全卸载，Q-learning 以及 DQN 方案的总代价相比图 5.5 更加平缓，这说明了在必要时，适当提高 MEC 计算能力是有必要的。

总 结

在本文中，主要介绍了有关机器学习，深度学习，强化学习以及移动边缘计算的相关理论知识。并且提出了一种基于移动边缘计算的多用户计算卸载和资源分配的模型框架。在此框架下，模拟了一个多用户计算卸载和资源分配的场景，在此场景下提出了边缘任务迁移能效优化问题。

我们设计了移动边缘计算的在基于强化学习的代理环境，并且在此环境下将Q-learning 算法和 DQN 算法在模型上结合，得到了这两个问题的基于 RL 的解。通过与其他方案的比较，给出了 Q-learning 和 DQN 方案的性能评价。仿真结果表明，在不同的系统参数下，该方案比其他计算卸载决策和资源分配的方案会具有更好的性能。

今后的研究将考虑更复杂的场景，以及网络模型的优化，以及无线电干扰的复杂环境。

致 谢

迎来了又一年的毕业季。而是我们自己即将离开这充满激情、欢笑和青春美好回忆的大学校园了。在最后的时光里，交上一份刻画着自己的大学知识的答卷，来对其做一个总结。此份毕业设计是在导师邝祝芳教授的关切指导下、攻克一个一个难题，不断实验总结完成的。从最初的设计和思想方向，老师给了我很大的帮助和指导，在此对您说一声谢谢。

在研究过程中，同学也给了我很多灵感，思考方向等等。和他们的交流，也让我对研究的问题越来越清晰，没有他们的帮助，很难完成本次研究工作，很开心能有一群志同道合的小伙伴们一起努力完成最后的一份答卷。

至此论文付梓之际，我的心情十分复杂，有言道：金无足赤，人无完人。我知道还存在着很多的不足和改进的空间，在今后的研究中继续改良和完善。

参考文献

[1] M. Peng, Y. Sun, X. Li, Z. Mao, and C. Wang, "Recent advances in cloud radio access networks: System architectures, key techniques, and open issues," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2282–2308, thirdquarter 2016.

[2] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, Oct 2016.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[4] S. Li, N. Zhang, S. Lin, L. Kong, A. Katangur, M. K. Khan, M. Ni, and G. Zhu, "Joint admission control and resource allocation in edge computing for internet of things," *IEEE Network*, vol. 32, no. 1, pp. 72–79, Jan 2018.

[5] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[6] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks," *IEEE Signal Processing Magazine*, vol. 31, no. 6, pp. 45–55, Nov 2014.

[7] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[8] J.Liu,Y.Mao,J.Zhang,andK.B.Letaief,"Delay-optimalcomputation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 1451-1455.

[9] S. T. Hong and H. Kim, "Qoe-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, June 2016, pp. 1-9.

[10] S. Xiao and X. lin Wang, "The method based on q-learning path planning in migrating workflow," in *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, Dec 2013, pp. 2204-2208.

[11] P. Zhao, H. Tian, C. Qin, and G. Nie, "Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing," *IEEE Access*, vol. 5, pp. 11 255-11 268, 2017.

[12] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 8, pp. 7432-7445, Aug 2017.

[13] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 2716-2720.

[14] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795-2808, October 2016.

[15] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896-5907, 2016.

[16] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment," IEEE

Transactions on Evolutionary Computation, vol. 6, no. 4, pp. 402-412, Aug 2002.

[17] Y. Cao, T. Jiang, and C. Wang, "Optimal radio resource allocation for mobile task offloading in cellular networks," *IEEE Network*, vol. 28, no. 5, pp. 68-73, September 2014.

[18] N. Takahashi, H. Tanaka, and R. Kawamura, "Analysis of process assignment in multi-tier mobile cloud computing and application to Edge Accelerated Web Browsing", IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 233-234, 2015.

[19] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile Code Offloading: From Concept to Practice and Beyond", IEEE Communication Magazine, 53(3), 80-88, 2015.

[20] E. Cuervo et al., "MAUI: Making smartphones last longer with code offload," Int. Conf. Mobile Syst., Appl. Serv. (Mobysis), 4962, 2010.

[21] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," Eur. Conf. Comput. Syst. (Eurosys), 301314, 2011.

[22] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading", IEEE INFOCOM, 945-953, 2012.

[23] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient Scheduling Policy for Collaborative Execution in Mobile Cloud Computing", IEEE INFO- COM, 190-194, 2013.

[24] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel, IEEE Trans. Wireless Commun., 14(1), 8193, Jan. 2015.

[25] Y. Wen, W. Zhang, and H. Luo, "Energy-Optimal Mobile Application Execution: Taming Resource-Poor Mobile Devices with Cloud Clones", IEEE INFOCOM, 2716-20, 2012.

[26] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile Code Offloading: From Concept to Practice and Beyond", IEEE Communication Magazine, 53(3), 80-88, 2015.