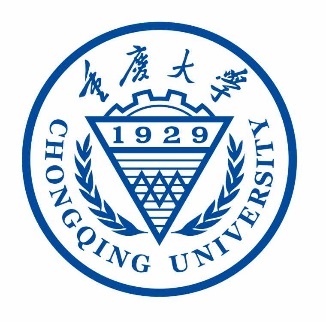
重庆大学本科学生毕业设计（论文）

基于强化学习的虚拟网络功能放置研究与实现



学 生：李林轩

学 号：20184320

指导教师：郭松涛

校外指导教师：郭得科

专 业：计算机科学与技术

重庆大学计算机学院

2022年6月

**Graduation Design(Thesis) of Chongqing University**

# **Research on the placement of virtual network functions based on reinforcement learning**

# 

# **Undergraduate: Li Linxuan**

# **Supervisor: Prof. Guo Songtao**

# **External Supervisor: Prof. Guo Deke**

# **Major: Computer Science and Technology**

## **School of Computer Science**

### Chongqing University

### June 2022

# 摘 要

随着网络功能的快速开发，底层硬件的高速更新迭代，目前基于特定用途硬件的网络放置开销以及运营成本很高，它们不能以简单的方式更新网络功能，而是需要在更新时让网络功能在特定的硬件上重新部署，这带来了极大的不方便，因此现在越来越多的网络功能是由网络功能虚拟化（Network Function Virtualization，NFV）的软件形式提供的。以软件形式实现的网络功能通常被组合为以服务功能链（Service Function Chain，SFC）的形式为用户提供服务，这带来了显而易见可扩展性和灵活性，网络功能也不再依赖于专用网络的设备的放置位置。而网络功能虚拟化带来的挑战之一是如何在放置虚拟网络功能（Virtualized Network Function，VNF）的同时优化它需要的资源配置。

本文提出了一种基于强化学习与序列到序列（Sequence-to-Sequence，Seq2Seq）网络模型的SFC放置策略，通过使用策略梯度方法，将策略表示为Seq2Seq模型中的网络，从而使Seq2Seq模型作为强化学习方法中的Agent，按照策略给出放置方案，在考虑物理网络基础设施限制、VNFs自身和服务器限制的基础上寻找最佳的VNF放置策略。现有的大多数方法都集中在启发式，以及元启发式算法的设计上，本文将这两类算法与本文提出的方法进行对比，在解决VNF-FGE（VNF First Graph Embedding，VNF前向图嵌入）问题上有更好的效果，在不同长度的SFC下测试，求出的目标函数值相较于遗传算法平均降低了59.1%；在高占用率的网络环境中，相对于启发式算法，本文提出的算法仍能够找到较优的可行解。

关键词：虚拟网络功能，强化学习，序列到序列模型

# **ABSTRACT**

With the rapid development of network functions and the high rate of iteration of the underlying hardware, the overhead and operational costs of placing networks based on purpose-built hardware are very high now. That’s the reason why they cannot be updated in a simple way, but rather require network functions to be redeployed on specific hardware at the time of the update, which presents a great inconvenience. Therefore, more and more network functions are now being provided by Network Function Virtualization(NFV) is increasingly being provided in software form. Network functions implemented in software form are often combined to provide services to users in the form of Service Function Chain(SFC), which brings obvious scalability and flexibility. And network functions are no longer dependent on the placement of devices on a dedicated network. One of the challenges posed by network function virtualization is how to optimize the allocation of resources required by a Virtualized Network Function(VNF) while placing it.

In this paper, we propose an SFC placement policy based on reinforcement learning with Sequence-to-Sequence network model. The algorithm use the policy gradient method to represent the policy as a network in a Sequence-to-Sequence model, so that the Sequence-to-Sequence model acts as an Agent in the reinforcement learning method. The Agent would give placement solutions according to the policy, in finding the best VNF placement policy while considering the physical network infrastructure constraints, the VNFs themselves and the server constraints. Most existing methods in the existing literature focus on the design of heuristic, meta-heuristic algorithms, and in this paper, one algorithm from each of these two categories is selected to compare with this method proposed in this paper, which has better results in solving the VNF-FGE problem, and the objective function values derived are reduced by 59.1% on average compared to genetic algorithms when tested under different lengths of SFCs; in a high occupancy network environment, the algorithm proposed in this paper is still able to find better feasible solutions compared to the heuristic algorithm.

**Key words：** VNF, Reinforce Learning, Sequence-to-Sequence Model

**目 录**

中文[摘要 I](#_Toc103078304)

[**ABSTRACT** III](#_Toc103078305)

[1 绪论 1](#_Toc103078306)

[1.1研究背景和意义 1](#_Toc103078307)

[1.1.1 研究意义 1](#_Toc103078308)

[1.1.2 研究背景 1](#_Toc103078309)

[1.2 国内外研究现状 3](#_Toc103078310)

[1.3 研究内容 5](#_Toc103078311)

[1.4 论文组织结构 5](#_Toc103078312)

[2 相关技术与理论 7](#_Toc103078313)

[2.1 长短期记忆神经网络 7](#_Toc103078314)

[2.2 序列-序列模型 10](#_Toc103078315)

[2.3 Bahdanau 注意力机制 11](#_Toc103078316)

[2.4 策略梯度方法 11](#_Toc103078317)

[3 虚拟网络功能放置算法的研究与设计 13](#_Toc103078318)

[3.1 虚拟网络功能放置问题阐述 13](#_Toc103078319)

[3.2 模型定义 14](#_Toc103078320)

[3.3 序列到序列模型的策略优化方法 16](#_Toc103078321)

[3.3.1 编码器 17](#_Toc103078322)

[3.3.2 解码器 17](#_Toc103078323)

[3.3.3 Bahdanau 注意力机制 18](#_Toc103078324)

[3.4 基于策略梯度的强化学习方法 18](#_Toc103078325)

[3.5基线估计器 21](#_Toc103078326)

[4 实验分析 23](#_Toc103078327)

[4.1实验环境 23](#_Toc103078328)

[4.2 参数设置 23](#_Toc103078329)

[4.3 结果分析 25](#_Toc103078330)

[4.3.1 模型评估 25](#_Toc103078331)

[4.3.2 算法对比 29](#_Toc103078332)

[5 总结与展望 33](#_Toc103078333)

[5.1 总结 33](#_Toc103078334)

[5.2 展望 34](#_Toc103078335)

[致谢 35](#_Toc103078336)

[参考文献 37](#_Toc103078337)

# 1 绪论

## 1.1研究背景和意义

1.1.1 研究意义

为了使通信设备拥有更高的性能和更好的可靠性，通信行业通常会将软硬件结合，并且采用专用的硬件设备来构建网络。这些专用通信设备虽然有着诸多优点，但是同时也面临着扩展性受限、技术创新难度大、业务开发周期长、管理复杂度高等一系列问题。这些使用专用的硬件设备实现的网络功能增加了网络服务提供商的资金和运营成本，同时也造成了网络耦合的问题。当一种新的服务出现时，相关的硬件设备必须按照一定的顺序进行放置和连接。这种操作极其耗时、耗力和容易出错，阻碍了新服务的添加和网络的升级。伴随着用户对移动通信需求的不断增加，这些问题日益严重。它们不仅影响了运营商的经济利益，同时也极大地限制了下一代通信网络的创新与发展。为此，迫切需要引入新的技术和架构来为通信网络技术持续发展注入新的动力。于是，NFV（Network Functions Virtualization，网络功能虚拟化）技术应运而生[1]。NFV将网络功能的实现从专用硬件转移到基于软件的组件，命名为虚拟网络功能（Virtualized Network Function，VNF）[2]，它是网络功能的虚拟化，需要放置到物理网络设备中运行。在物理设备上放置的有序的VNF集合被称为前向图，本文的主要工作是研究VNF前向图嵌入问题（VNF Forwarding Graph Embedding，VNF-FGE），也就是如何将VNF映射到物理网络设备上，它是NFV资源分配问题之一。

NFV的出现带来了显而易见的可扩展性和灵活性，为了让这项技术很好的应用在网络设备上，我们需要找到合适的VNF放置方案，使得VNF能够成功放置在底层物理网络上。对放置方案的优化主要体现在它的底层网络基础设施中资源分配的优化，这些优化能够节省大量的硬件开销，从而达到减少放置的成本的目的。

1.1.2 研究背景

VNF-FGE问题是网络功能虚拟化方向的重要问题。当设计一个算法来解决VNF-FGE问题时，我们必须考虑选择哪些特征将被建模为问题的一部分。通常考虑的特征包括主机的处理能力（通常被形式化为CPU的资源量）、链路容量（通常被建模为链路的带宽）、VNF的处理延迟以及链路延迟和物理或虚拟链路的跳数等因素，考虑的不同因素的组合会带来针对这些因素的更适用和现实的解决方案。如图1.1所示，以往的文章在解决VNF-FGE问题时通常考虑的因素包括处理器处理能力，处理延迟，容量，链路容量，链路延迟，以及解决方案的成本。我们设计解决VNF-FGE问题时，主要考虑了底层物理网络中处理器处理能力，处理延迟，链路带宽和链路延迟等因素。

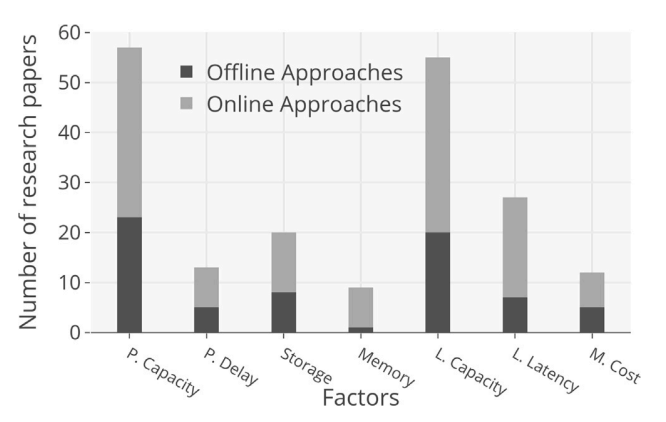


图1.1以往文献中VNF-FGE问题离线与在线算法选择不同因素进行考虑的比例[3]

针对VNF-FGE问题，以往的研究提出的方法大致可分为两种：在线放置方法和离线放置方法。在线放置方法解决的问题是处理动态的SFC放置请求，它在设计时对需要传入的放置需求是未知的。它们的解决方案通常是考虑将VNF放在使用边缘云（小云）的运营商的云基础设施上，并考虑它们之间的最佳平衡[4]。我们主要针对离线放置方法进行着重讨论。

对于离线的VNF-FGE问题，大量的解决方案都是基于线性编程的[5,6]，这些解决方案通常使用网络配置和约束条件以及选定的目标函数来指定要解决的优化问题。根据选定的问题变量，这项工作可能会依赖于整数线性规划（Integer Linear Programming，ILP）或混合整数线性规划（Mixed Integer Linear Programming，MILP），这两种方法可用的工具包括CPLEX2和YALMIP[7]，然后用它们来解决指定的优化问题，给出精确或最优的解决方案。由于ILP是NP-Hard问题，精确解通常被用作后面提出更好的算法的基准，这些算法可能使用启发式或元启发式算法来求解。这些启发式或元启发式的算法通常在物理网络资源较丰富时能有很好的效果，但是在物理网络很难满足VNF的资源要求，即无法或很难找到满足底层网络物理资源的限制和VNF自身要求的放置方案时，这些算法往往无法产生解。

1.2 国内外研究现状

以前的一些研究工作已经着手研究解决VNF-FGE问题。VNF-FGE问题可以分为离线和在线，离线方法不考虑先前放置在网络中的SFC，它专注于解决NP-hard[8]的问题；而在线方法主要是为了处理在动态到达的SFC请求。由于我们实现的算法是一种离线的VNF-FGE方法，我们重点讨论离线方法的研究现状。

有许多文章提出了VNF-FGE问题的离线解决方案。有一些研究工作涉及VNF的放置问题，提出了使用启发式算法给出解决方案的离线算法[5,6,9-11]。虽然他们关注的是同一个问题，但他们的目的是找到一个实现不同目标的解决方案。Cohen等人[5]的目标是找到一个使VNF的连接和放置成本最小的解决方案。为了实现这一目标，他们实现了一个线性程序，对每个VNF的存储要求和放置成本以及NFVI的可用资源（Network Function Virtualization Infrastructure Point of Presence，NFVI-PoP）进行建模。这是第一个考虑在多个SFC之间共享实例化VNF的可能性的工作。他们实现的线性程序的解决方法是通过两种算法实现的；一种是不考虑NFVI-PoP资源限制的算法，即无约束的放置算法，另一种是考虑这种限制的算法，即有约束的放置算法。这两种算法都依赖于一种启发式算法，即根据存储量大小（降序）将VNF分配给NFVI-PoP，类似与First-Fit算法。

在不同的目标下，有部分工作专注于最小化VNF实例的数量[6,9]。Chi等人[6]提出了一个替代的线性程序，以解决VNF放置问题。他们的程序将根据VNF的处理能力进行建模。Chi等人[6]还依靠启发式方法，将放置问题简化为bin-packing问题[12]。Sang等人[9]则提出了一种贪婪的启发式方法，它在具有未处理请求的NFVI-PoP上进行迭代，并将VNF的组合放置于NFVI-PoP中来处理具有最大数量的未处理流量。

Tomassilli等人[10]为VNF放置问题创建了两个启发式方法和一个精确解决方案。他们认为所有的需求都是提前规定好的，他们的目的是最大限度地减少与主机有关的资源，比如许可费用或能耗。他们引入了三种方法：

① 动态编程的贪婪启发式方法，在预先计算的路径中选择成本最低的NFVI-PoP来承载VNF；

②近似算法，从ILP公式中随机选择约束条件，以指导近似最佳解决方案；

③针对树状拓扑结构的最佳动态编程技术，在所有流量在树中上游或下游的特殊情况下以多项式时间运行。

最后，Sallam等人[11]还开发了两种贪婪的启发式方法，旨在通过选择最佳的NFVI-PoP来承载VNF，使完全处理的流量总量最大化。此外，他们还考虑到了使用NFVI-PoP的成本以及VNF的处理能力。第一个启发式算法是为具有统一成本的NFVI-PoP量身定做的，它将每个VNF分配给与给定目标函数的最大值对应的NFVI-PoP。第二，针对异质成本的情况，算法引入了基于枚举的贪婪启发算法[13]。它将满足预算约束的NFVI-PoP按已放置的VNF数量分组，然后将每个VNF放置在该组的NFVI-PoP中，该组呈现出关于目标函数的最大收益。

有三项工作开发了元启发式方法[14-16]来解决能够减少网络资源分配任务的放置问题。两个元启发式方法[14,15]旨在减少端到端的服务延迟，同时保证一定的可靠性[14]或可用性[15]水平。Chantre等人[14]制定了一个非线性混合整数程序，其中VNFs由其处理延迟和可靠性来描述。作者开发了一种元启发式算法，使用粒子群优化算法（Particle Swarm Optimization，PSO）[17]来寻找冗余VNF的最佳数量和它们各自的NFVI-PoPs。PSO是一种基于共享信息的粒子互动的搜索算法。基于局部和全局信息，这些算法向全局最优迁移，并在算法中添加了非线性混合整数编程公式被用来指导如何搜索到良好的解决方案。

同样的，Yala等人[15]开发了一种遗传算法（Genetic Algorithm，GA），在边缘云或中心云分配VNFs。前者的延迟低但资源稀缺，而后者的延迟高但计算资源充足。GA算法考虑到了链路延迟和实例化成本，尽管有目标函数，但它仍能找到接近最优的解决方案。Manias等人[16]没有使用ILP来模拟网络，而是选择了使用机器学习算法决策树[18]来实现VNF的放置，目的是减少VNF之间的延迟。作者认为，这种方法大大减少了协调VNFs所需的计算量，因为它的学习阶段是在其他地方提出的近乎最优的启发式算法的基础上离线进行的[19]。他们的评估显示，他们的元启发式算法在分析过的一半场景中优于用于训练模型的启发式算法。

除了前面的参考文献，使用强化学习来解决VNF-FGE问题也成为了一种解决方案。Mijumbi等人 [20]使用强化学习中的Q-learning算法来控制NFV管理系统如何进行资源分配。在文献[21]中，Mijumbi还采用了一个人工神经网络，在其之前的工作基础上进行改进，对资源分配按照新的方式进行决策。Yao等人 [22]首次实现了历史网络请求数据和基于策略的强化学习方法来优化节点映射。他们将节点和链接使用嵌入embedding表示，在网络的每个时间步中，都会根据网络属性的变化进行更新。这个网络中的嵌入通过卷积神经网络来选择使长期收益最大化的底层节点。

综上所述，放置NFV的主要挑战之一是在NFV基础设施中优化NFV基础设施的最佳资源配置，这是一个NP-hard问题，而现有的解决离线VNF-FGE的方法包括启发式算法，元启发式算法，以及强化学习方法。想要推断出一个有竞争力的启发式算法是一项艰巨的任务，因此现在越来越多的解决方案都是通过强化学习来模拟出最优的放置策略。

1.3 研究内容

本课题重点研究离线VNF-FGE问题。我们将VNF-FGE形式化为一个包含约束的组合优化问题，其中网络功能需要被放置在满足服务水平协议的网络基础设施之上，即在物理网络基础设施上有效地映射一组网络服务请求。特别是需要在考虑到虚拟环境的状态下获得网络功能链中VNF放置的最佳位置，从而实现特定的资源目标，例如，剩余资源的最大化，整体功耗的最小化，特定QoS指标的优化等等。此外，VNF的一些属性，如转发延迟、放置在物理网络上时所需的资源等等，也必须加以考虑。

1.4 论文组织结构

本文正文内容分为五个章节进行介绍：

第一章绪论，主要介绍了虚拟网络功能前向图嵌入问题的研究背景，研究意义，国内外研究现状和研究内容。

第二章相关技术和理论，主要介绍了本文设计算法时用到的有关技术，并对这些技术进行了一定的分析。

第三章虚拟网络功能放置算法的研究与设计在将问题阐述定义后，从强化学习方法中的Agent，即Sequence-to-Sequence网络开始介绍，介绍了学习过程中需要优化的具体目标以及计算方式，最后针对模型的弱点提出了用于改进的一些方法。

第四章实验分析主要介绍了实验环境，实验的各种参数设置，以及实验的结果分析，包括对模型的测试结果进行分析，以及和启发式算法First-Fit，元启发式算法遗传算法进行比较。

第五章总结与展望总结部分对设计出的算法进行了总结，展望部分指出了算法设计上的一些可供优化的想法。

# 2 相关技术与理论

### 2.1 长短期记忆神经网络

长短期记忆神经网络（Long Short Term Memory networks，LSTM）是循环神经网络（Recurrent Neural Network，RNN）的一个变种，LSTM 由Schmidhuber 在1997年提出[23]，并在近期被Alex Graves进行了改良和推广。由于独特的设计结构，LSTM是适合处理和预测序列数据非常好的网络。为了理解LSTM，我们先对RNN进行介绍。

一般的RNN网络结构如图2.1所示，它包含输入层，隐藏层和输出层，并将上一个时间步更新的隐藏层的权重输入到下一时间步的隐藏层中。将其展开后如图2.2所示。

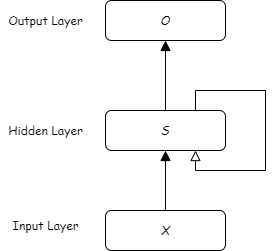


图2.1 RNN网络结构

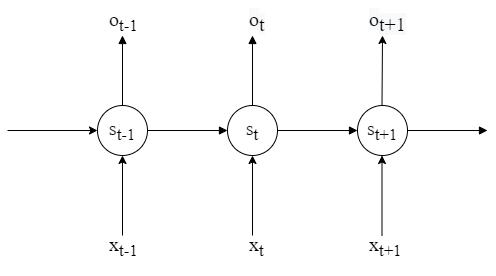


图2.2 RNN网络结构按照时间步展开图

RNN输出层输出的公式与隐藏层状态更新的公式如下所示。其中代表RNN中的第个时间步，代表时间步的输出, 代表时间步的隐藏层的值。是每个时间步之间的权重矩阵，分别代表输入层，隐藏层，输出层的权重。 代表激活函数，一般为softmax，因为RNN通常用于输出不同类别的概率。RNN之所以可以解决序列问题，是因为它可以记住每一时间步的信息，每一时间步的隐藏层不是只由该时刻的输入层决定，还由上一个时间步的隐藏层决定，也就是公式（1.2）中的的计算方式不仅仅依赖于这个时间步输入的，同样也要将上一个状态纳入计算中。

（2.1）

（2.2）

然而RNN的缺点也很明显，它会带来梯度消失与梯度爆炸。产生这种原因的本质是因为神经网络的更新方法，梯度消失是因为反向传播过程中对梯度的求解会产生激活函数的导数和参数的连乘，导致靠近输入层的梯度几乎为0，得不到更新。梯度爆炸是也是同样的原因，如果初始权重设置的值大于1，或者更大一些，多个大于1的值连乘，在最后得到的值将会很大或溢出，导致梯度更新过大，模型无法收敛。

LSTM相较于一般RNN最大的优化就是它有效解决了梯度消失的问题。相较于RNN将所有的信息都存下来，LSTM可以选择性的存储信息。LSTM的结构图如图2.3所示，相较于RNN只传递一个状态，LSTM会传递两个状态，与。

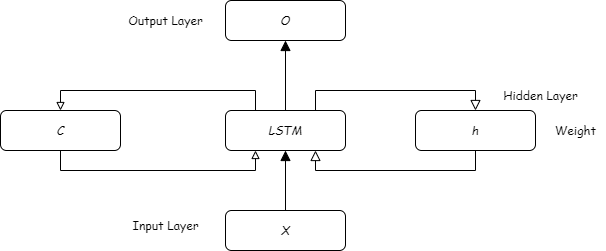


图2.3 LSTM结构图

LSTM通过当前输入的和上一个状态传递下来的拼接后乘以权重，通过训练得到四种状态：

上面公式中的是由拼接向量乘以权重矩阵之后，再通过一个sigmoid激活函数转换成0到1之间的数值，来将其作为门控的状态。而则是将结果通过一个 tanh激活函数将转换成-1到1之间的值。

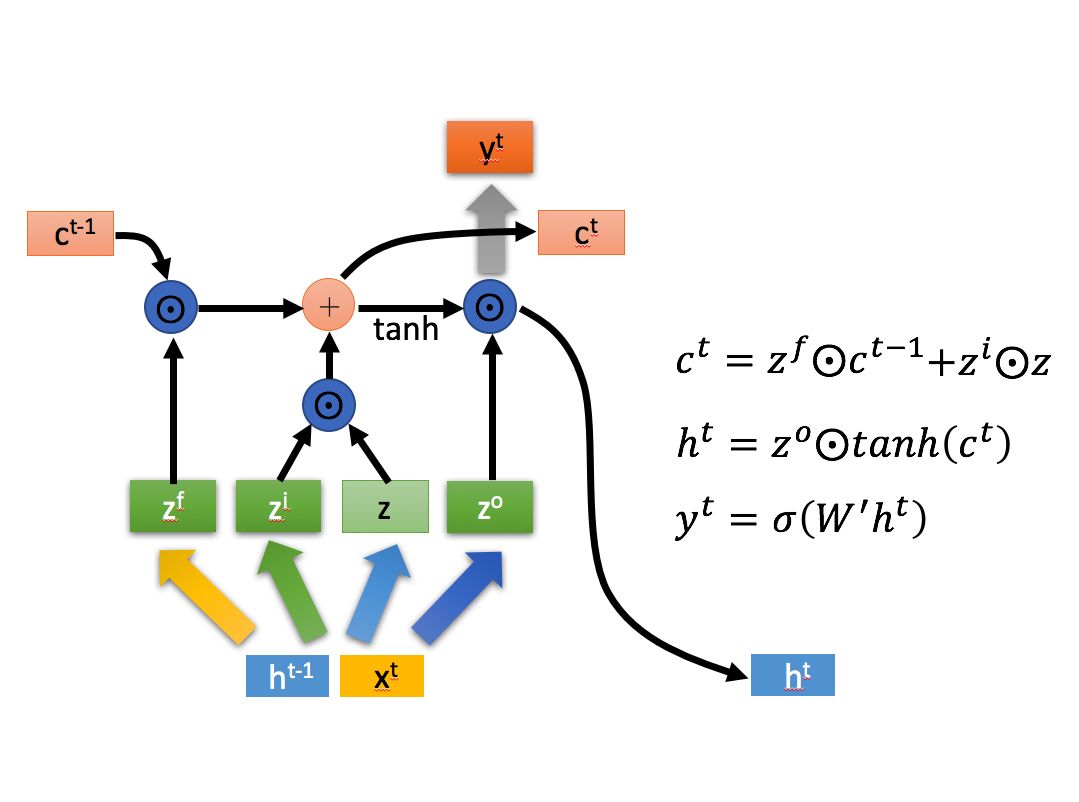


图2.4：,四个状态在LSTM内部的使用[24]

LSTM内部主要包括三个阶段：

①忘记阶段：这个阶段主要是对上一个节点传进来的输入进行选择性忘记。通过计算得到的来作为忘记门的控制信号，来控制上一个状态的哪些部分需要留下哪些部分需要忘记，这里的代表forget。

②选择记忆阶段：这个阶段将这个阶段的输入有选择性地进行记忆。主要是会对输入的进行选择记忆。哪些重要则着重记录下来，哪些不重要，则少记一些。当前的输入内容由前面计算得到的z表示。而选择的门控信号则是由来进行控制，这里的代表information。将上面两步得到的结果相加，即可得到传输给下一个状态的，也就是图2.4中的第一个公式。

③ 输出阶段：这个阶段将决定哪些将会被当成当前状态的输出。主要是通过来进行控制的。并且还通过一个tanh激活函数进行变化，对上一阶段得到的进行了放缩。

与普通的RNN类似，LSTM单元的输出往往最终也是通过变化得到。

LSTM通过门控状态来控制传输状态，记住需要长时间记忆的内容，忘记不重要的内容，而不像普通的RNN那样仅有一种记忆叠加方式。LSTM对很多长序列的任务来说相较于RNN有更好的效果。

2.2 序列-序列模型

Sequence-to-Sequence模型（简称Seq2Seq）在2014年由Sutskever等人[25]提出，它用于处理序列到序列的任务，它由Encoder编码器和Decoder解码器两个部分组成，两个部分均为一个RNN网络，论文中使用的是LSTM，它的结构如图2.5所示，其中输入序列ABC的部分为编码器Encoder，输出序列WXYZ<EOS>的为解码器Decoder。

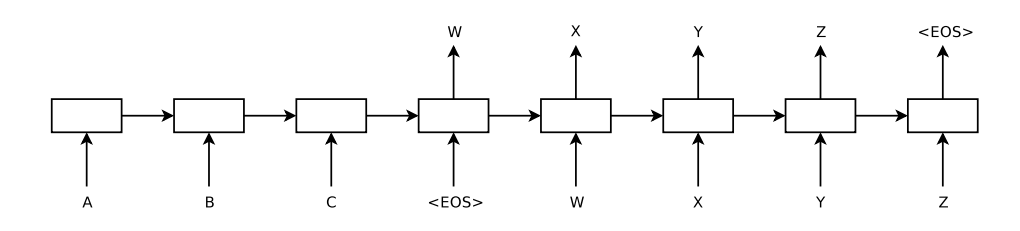


图 2.5 Seqence-to-Sequence模型结构图

Seq2Seq模型首先通过Encoder读入输入的序列，并通过Encoder将其编码成一个固定维度的向量：

其中是编码器RNN网络的隐藏状态，是它的输出，就是编码器输出的向量，是Encoder中最后的隐藏状态，它记录了整个序列的信息。

得到向量后再将它输入另外一个作为Decoder解码器的RNN网络中。在收到向量时，与另一篇文章中Cho等人提出的Seq2Seq模型[26]不同，该模型直接将向量作为Decoder的初始隐藏状态，接着通过下列公式在RNN网络中不断传递。

从公式（2.10），（2.11）中可以看到，解码器不仅仅依赖于RNN上一个时间步的状态，也同时接收了上一个时间步的输出。将每个时间步的输出拼接起来即为预测的输出结果。

Seq2Seq模型允许输入序列长度和输出序列长度为任意比例，输出序列长度可以不等于输入序列长度，因此它通常用于处理机器翻译，并在机器翻译方面取得了很好的效果。在本文中，输入序列是由一组VNF组成的服务功能链序列，而最终输出的是它们的放置向量，同样也是要解决序列到序列的问题，因此我们在预测序列部分采用了该模型。

2.3 Bahdanau 注意力机制

Bahdanau Attention 是一种经典的注意力机制[27]，在2015年由Bahdanau提出，它通常应用在Seq2Seq模型中。Seq2Seq模型由Encoder将其编码为一个固定大小的向量，再输入Decoder进行解码并输出结果。在这个过程中，输入Encoder的所有特征都被整合到了它输出的向量中，当输入较长时，Encoder输出的向量很可能并不能完全保留输入Encoder序列的特征，并且随着输入信息长度的增加，由于Encoder输出的向量长度是固定的，先前编码好的信息有可能会被后来的信息覆盖，从而丢失很多信息。

Bahdanau Attention的提出者认为长向量的使用是Encoder-Decoder框架的性能瓶颈。而Attention机制就是让Decoder可以选择性地使用编码部分的信息，与人类观察事物相似，注意力会放在一些重要的地方，而不是对全部内容给予相同的注意力。通过在Seq2Seq模型中加入Attention机制，可以提升它处理较长的输入序列的效果。

2.4 策略梯度方法

强化学习使用值和值来评估价值。值用来评估动作的价值，它代表智能体选择该动作之后一直到最终状态时奖励综合的期望；值用来评估状态的价值，它用来表示智能体到达该状态之后一直到最终状态的奖励综合的期望。

一般的强化学习使用的是值函数的方法，即对强化学习方法中产生的策略先进行值，值的评估再进行改善，值函数最优时对应的策略即为最优的策略。然而对于较复杂的情况，无法描述针对每种状态采取怎样的应对方式。这种情况下就可以采用策略梯度方法，通过神经网络直接代替我们想要表示的规则。

策略梯度方法将寻找的策略进行参数化，并用线性或非线性（神经网络）的方法对策略进行表示，并寻找最优的网络参数来使得强化学习的奖励最大。在值函数的方法中，我们迭代计算的是值函数，不断地优化值函数，使其得到最大（最小）值，然后再根据值函数对策略进行优化；而在策略梯度方法中，我们直接对策略进行迭代计算，也就是迭代更新用于表示策略的网络中的参数值，使得获得更大的奖励的动作被选择的概率更高，直到累积回报的期望最大，此时网络中的参数所对应的策略为最优策略。

总而言之，策略梯度方法的基本思想是，按照奖励最大化的方式来更新模型参数。与值函数方法中对值函数进行参数化表示相比，策略梯度方法将策略参数化更简单，它也可以处理具有无限个行为和状态的连续行为空间。对于值函数来说，状态值的微小变化，就可能会产生完全不同的策略，所以值函数方法对于扰动是很敏感的。但是对于策略梯度方法，由于策略函数是可微的，因此变化与值函数方法相比也是相对平滑的，稳定的。

策略梯度方法也有一些缺点，通过策略梯度搜索到的方法容易收敛到局部最小值，并且策略梯度通常搭配蒙特卡洛方法进行采样，这样在评估单个策略时容易评估不充分，方差过大，需要合理使用。

# 3 虚拟网络功能放置算法的研究与设计

3.1 虚拟网络功能放置问题阐述

我们先将研究的问题进行说明。我们使用表示服务功能链的集合，中包含许多条服务功能链，每一条服务功能链都包含一组网络服务，这些网络服务需要放置在一组主机服务器上，其中是服务器的集合。我们的工作就是让其以最优的方式放置在这组主机服务器上。为了简化问题，我们假设这组主机服务器通过各自的链路连接到一台路由器，使用L表示链路的集合，整个服务器组为星型拓扑结构，如图3.1所示。每个VNF会有自身需要的资源，放置策略需要满足VNF的资源要求以及底层网络的资源限制。底层网络的资源限制使用表示，它包括主机服务器自身的资源限制，每条链路使用的带宽不能超过链路的总带宽，服务功能链上允许的最大传播时延限制，其中是服务器和VNF包含限制的集合。我们最终的目标是找出一个满足各项限制并且使得放置的服务功能链能耗最小的放置策略。

我们将主机服务器的集合命名为*，*将VNF的集合命名为。一个网络服务功能由个VNF组成，它们构成了一个服务功能链，其中*f*代表VNF，属于。

我们需要解决的问题是寻找能耗最低的放置策略P，该放置策略使用二进制变量表示，代表虚拟网络功能f是否放置在了主机上，虚拟网络功能放置在主机上则为1，未放置则为0。同样，我们用二进制变量表示主机是否激活，如果主机上至少放置了一个虚拟网络功能，那么为1，反之为0。为了简化问题，我们认为一个虚拟网络功能只能且必须放在一个主机上。

为了直观地定义我们需要优化的问题，我们定义了以下变量。每个工作中的服务器会有一个基础的能耗，它的能耗会随着放置在该服务器上的虚拟网络功能的数量而增加。服务器的功耗特点是线性曲线，随着它的计算利用率的增加按比例增长。因此我们设定放置在服务器中的每个虚拟网络功能都会额外增加的功率。此外，链路上的能耗也需要考虑在内，我们设定它的计算方法为链接服务器的链路二进制激活变量乘链路的带宽利用率能耗（表示）乘上每条链路使用的带宽。为了方便计算链路是否使用，我们定义二进制变量来表示服务器上的VNF v是否需要通过链接该服务器的链路链接到下一个VNF。

每个服务器都有自己的可用资源，我们将服务器看作是按照时间片工作，每个服务器有自己的时间片分配个数，使用表示。VNF 所需的资源量表示为，即该VNF工作需要占用服务器中几个时间片。VNF 的数据传输所要求的最低带宽表示为。对于时延方面的变量，我们使用表示由于服务功能链导致的链路上的延迟，使用表示由于VNF 的计算时间导致的延迟。最后，我们用表示每个服务链允许的最大延时。

我们通过图3.1的例子来理解整个问题。假设存在一条服务功能链SFC 1，它包含五个VNF，需要放置在的四台服务器上，且网络包传输的路径必须与SFC中VNF的顺序相同。每台服务器都有自身的可用资源，放置在上面的VNF也有所需的最小资源要求。假设现在寻找到的放置方案如图所示，那么该放置方案的能耗为每台服务器的能耗加上链路上的能耗。注意，当多个VNF放置在同一台服务器上时，我们认为它们之间不需要链路，且默认满足VNF的带宽与延迟限制。对于放置在不同服务器上的VNF，我们需要将它们之间链路的相关成本计算在总成本内，且需要考虑VNF的带宽与延迟限制。我们引入放置策略向量来描述放置策略，向量的长度与SFC长度相同，代表放置在上。图中的放置策略即为。

图形用户界面, 应用程序

描述已自动生成

图3.1 一个SFC放置策略的例子

3.2 模型定义

根据在3.1节中定义的变量，我们将需要优化的成本函数表示为公式：

成本函数的计算方法是遍历每台服务器，将激活的服务器的基础能耗求和，再将每个VNF所占用的资源数（CPU内时隙数量）与代表VNF是否在该主机上放置的二进制变量相乘，计算出每台服务器由于放置VNF造成的额外的能耗。最后加上链路的能耗，即所有放置在该链路连接的主机上的VNF所需的带宽之和乘链路的带宽利用率成本。放置方案需要受到以下限制:

公式（3.2）表示在一台激活了的服务器上放置的所有VNF所需的资源量不能大于该服务器的可用资源量；公式（3.3）表示在服务器上放置的VNF 如果需要通过链路将数据包转发到放置在另一台服务器的VNF上时，这台服务器上这种情况的VNF所要求的总带宽不能大于链接该服务器的链路带宽；公式（3.4）表示每台服务器上VNF计算时间产生的延迟与由于服务功能链产生的所有链路上的延迟之和不能大于服务功能链s允许的最大时延。

然而很多情况下，不存在能满足以上约束条件的VNF放置方案。针对这种情况，我们认为能够找到一个成功放置所有VNF的方案更重要，因此我们将放松约束的限制，为违反约束的放置策略添加惩罚项C，目标是优先找到放置方案。对限制的处理会在后文详细介绍。

找到成功放置VNF的方案只需要保证SFC中每个VNF都放置在了服务器上，即保证：

这样一来，该问题的放置方案的数量是有限的，设问题的搜索空间为，我们可以将该问题看作是公式（3.6）描述的组合优化问题：

*Cost(P)+C(P)*

3.3 序列到序列模型的策略优化方法

我们使用神经网络组合优化模型对上述组合优化问题进行求解，模型结构如图3.2所示，模型中包含一个基于编码器-解码器的序列-序列LSTM网络模型，它扮演的角色是强化学习中的Agent，用于预测VNF放置的策略；接着使用强化学习对策略与强化学习的环境进行交互，并进行学习。强化学习的Agent接收一个大小可变的个VNF组成的服务功能链作为输入，通过神经网络输出该服务功能链的放置向量，表示对于服务功能链中每个VNF的放置位置。神经网络的权重使用表示，通过神经网络产生一个放置策略，它表示基于产生的条件概率，再通过强化学习对策略进行调整。

网络的输入是一串包含VNF序号的服务功能链，这些服务功能链长度是可变的，而需要输出的是该服务功能链中VNF的放置策略。在3.1节中提到了，我们使用向量对放置策略进行表示。基于编码器-解码器的Seq2Seq模型在序列处理，预测方面已经取得了突出的成果，并且由于我们前面提到的，输入的VNF序列长度是可变的，而Seq2Seq模型可以处理长度可变的序列，因此我们使用该模型来预测VNF的放置序列。

图3.2展示了虚拟网络功能放置算法的模型结构。模型以强化学习方法为主体，强化学习中的环境为物理网络环境，而做出决策的Agent则是Seq2Seq网络，Seq2Seq网络内部的参数即为放置策略。在内部的编码器-解码器结构内添加了Bahdanau 注意力机制，用于改善Seq2Seq模型应对长序列的效果。Agent将根据环境的状态State以及当前网络的策略做出动作，也就是放置方案，放置方案是由Seq2Seq网络解码器迭代生成的，它每一个时间步输出编码器中输入的服务功能链SFC序列对应时间步的VNF放置的位置，将所有时间步的输出Placement\_n拼接起来输出Placement Vector就变成了放置方案，也就是本次Agent产生的动作。环境在接收到动作后会更新自身的状态，输出新的状态和Agent这次动作的奖励，同时将环境的状态返回给基线估计器Baseline Estimator，基线估计器将根据环境的状态估计出用于评估策略好坏的基线Baseline，奖励减去基线的值将输入Agent，Agent会根据它来更新内部的网络参数，更新VNF的放置策略。

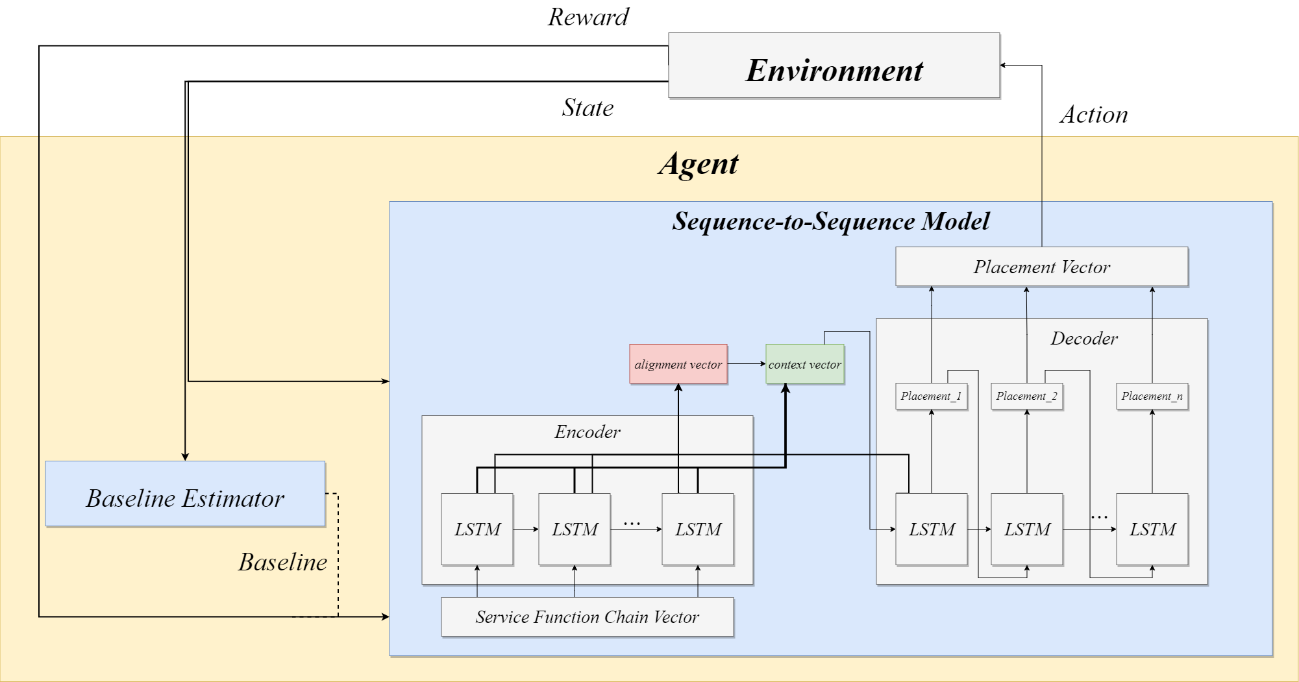


图3.2 虚拟网络功能放置算法模型结构

3.3.1 编码器

编码器（Encoder）使用的是LSTM模型，隐藏层的维数根据需要训练的模型输入的VNF序列长度有所调整，从32到128。输入的embedding向量长度为10，LSTM的层数也同样根据需要训练的模型输入的VNF序列长度有所调整，最低1层，最高4层。

当输入序列过长的时候，传统的RNN很容易出现梯度消失或者梯度爆炸的问题，梯度爆炸可以通过梯度截断来解决，而梯度消失对于RNN来说并不容易解决，所以我们使用了LSTM来代替RNN。在2.1节中提到过，LSTM通过控制遗忘门、输入门和输出门来决定忘记多少历史信息以及记住哪些历史信息，从而可以更好的保留所需要的信息。

Encoder最终输出带注意力机制的向量context vector，并将其输入到解码器Decoder中。

3.3.2 解码器

解码器（Decoder）是一个添加了注意力的LSTM模型，它的解码步骤与输入序列的数量相同。在每一次迭代时，解码器都会输出放置的主机序号，表示在编码器中输入的服务功能链相同位置的VNF放置的位置。解码器网络隐藏状态是其自身先前状态与对编码器隐藏状态的注意力机制相结合的函数，代表在解码器中时间步时的隐藏状态，代表在编码器中时间步时的隐藏状态。context vector由输入序列的隐藏状态的总和组成，由对齐向量（alignment vector）进行加权。解码器按照公式（3.7）输出步骤时LSTM单元的。

公式（3.7）中是表示在解码器解码过程中对应的每个编码器隐藏状态权重的概率向量，也称为注意力分布向量。

3.3.3 Bahdanau 注意力机制

当解码器在解码第时刻时，我们希望网络仅仅只是将注意力集中到编码器第t时刻上，而尽可能忽略其它时刻的影响。因此，只要网络在解码第t时刻时，将注意力主要集中于向量中和解码器第时间步对应的部分即可。由于此时解码器产生的向量与编码器编码部分的隐含状态处于同一个嵌入空间（embedding space），所以我们可以通过将两者的相似度进行对比，并告诉解码器哪个编码时刻的隐含状态与当前解码时刻的隐含状态最为相似。这样，在解码器解码当前时刻时，网络就能将注意力尽可能多的集中于对应编码时刻的隐含状态。

编码器在输出context vector前，还将输出由注意力权重组成的对齐向量（alignment vector）。大小可变的对齐向量与输入编码器的序列有着相同的长度，它是通过计算解码器的当前目标的隐藏状态与编码器的每个隐藏状态的相似程度来进行对比的。

其中公式（3.8）中用于计算相似程度的函数定义为以下形式：

*)*

公式（3.9）采用加性模型进行计算，公式中的变量和、是模型中要学习的权重矩阵。

3.4 基于策略梯度的强化学习方法

我们要解决的VNF-FGE问题是寻找能耗最低的VNF放置方案，但是对于一组VNF，我们并不知道它的最优放置方案是什么。因此我们没有办法获得正确的标签来对Seq2Seq模型进行训练。这时候我们在Seq2Seq模型的基础上采用强化学习方法。

强化学习方法中的智能体的动作空间对应于前文转化的组合优化问题中的搜索空间，当VNF序列增长时，搜索空间将会变得非常大。因此使用策略梯度和蒙特卡罗方法来进行强化学习。我们将Seq2Seq模型看作强化学习的策略，写作，将网络内的参数写作。网络的输入是SFC中的VNF序列，网络的输出是VNF放置向量，也是这次智能体将要采取的动作。环境将对这次智能体采取的动作进行反馈，输出新的状态和这次动作所得到的奖励。环境的状态变化就是将VNF放置在了环境中，每次动作所产生的状态变化都是唯一的。对于奖励函数，我们不能简单地将其定义为能耗的函数，需要将前文提到的约束也考虑在内。接下来我们推导奖励函数。

我们将智能体输出的动作和环境输出的状态放在一起，组成一条轨迹：

这条轨迹发生的概率即为：

环境对于Agent产生的动作，即环境对当前状态产生的VNF放置策略而输出的是一定的，因此公式（3.11）变为：

对公式（3.12）其两边取对数得到：

我们的目标是最小化放置成本，通过调整，使得能耗Cost的期望最小，同时尽量满足约束条件，我们先将能耗的计算方式表示为公式（3.14）：

同时我们需要考虑环境中限制带来的约束，我们用表示环境执行放置策略后由于违反约束而产生的惩罚项。

我们为不同约束赋予不同的系数，这样如果需要着重于保证其中某一项约束得到遵守，则可以将对应的系数设置的更大。

需要优化的目标函数是放置策略带来的能耗与惩罚项之和，它的计算方式如下：

限制的惩罚项均为前文提到的约束的差值，如果约束为负，则证明Agent给出的策略很好的遵守了网络的限制，能耗加上限制就会变得更小，网络会有更大概率产生满足限制的策略。

Seq2Seq模型的网络通过随机梯度下降法来更新，其中为学习率。

接着计算，即计算关于的梯度，也就是策略梯度。由公式（3.21），将target的梯度表示为公式（3.22）

最终将target的关于策略的梯度转化为求公式的值。在代码实现的过程中，由于搜索空间太大，无法将所有的放置方案都遍历一遍，因此我们采用蒙特卡洛方法进行采样，对target的期望进行估算。于是公式（3.22）变为下列形式：

其中N为蒙特卡洛方法采样的数量。是对第n个采样的放置策略计算出的值。的计算方式如公式（3.24）：

由于环境的初始状态与环境接收到上一次环境和动作后更新的环境是固定且唯一的，与策略无关，因此计算最终只需要考虑根据当前环境会产生什么样的动作即可。将公式（3.24）代入公式（3.23）得到公式（3.25）：

我们的目标是求得target的最小值。损失函数的值沿着梯度的方向呈下降趋势，然而，如果梯度出现过大的情况，函数值就会在最小值附近反复震荡，出现收敛不到最值的情况，或是在不断地连乘后导致梯度爆炸。为了预防这两种情况的发生，在模型中添加了梯度裁剪的方法，对梯度进行裁剪，这里选择的是对梯度的L2范数进行裁剪，即将所有参数偏导数的平方和再开方。如下式：

自定义裁剪阈值为，当大于c时，的值按公式（3.28）进行计算：

当小于等于时，不变。

除了对梯度做了裁剪之外，在2.4节提到，策略梯度方法本身同样有缺点，即模型容易收敛到局部最小值，为了尽可能解决这个问题，我们在训练时针对一种长度的服务功能链将同时训练多个模型，并在模型推理时同时使用这些模型，找到这些模型中效果最好的结果作为最终的结果返回。

3.5基线估计器

事实上，不管Agent选择哪种动作，只要将VNF放置到服务器中，总会增加，这样的话不管Agent执行什么动作，采取每一种动作的概率都会减小。虽然在归一化之后会使得产生更小的策略产生的概率更大，产生更大的策略的概率更小，但是这样收敛的速度会慢许多。并且由于我们是通过采样求出的每个状态和动作的期望值，因此求出的期望值是近似值，这样有些策略是没有被采样到的。如果这些策略恰巧是比较优秀的策略，那么经过一次迭代并将概率归一化后，这些策略的概率反而降低了。采样的随机性是非常大的，尤其是使用策略梯度方法，在每次更新参数前都需要进行采样，采样的次数过多导致我们一次采样不能采样过多数据，这样采样到的数据方差是非常大的。我们为了让产生更小的动作的概率更高，产生更大的动作的概率更低，减少梯度的方差，使得网络收敛更快，我们在网络中添加一个基线估计器，使用基线估计器估计出的基线来评估策略的好坏。

算法这里的思路是采用Actor-Critic方法，其中Actor即为Agent中的神经网络，它代表这策略函数，而Critic是指值函数，它将对当前策略的值函数进行估计，而Critic网络估计出来的值的期望将作为该动作的值。在我们的模型中，动作就代表了放置方案，我们求一个batch内的值的期望，就构成了当前放置策略的基线。

基线将预测当前策略的值，我们用表示，基线估计器基于当前环境的状态，也就是底层网络的放置状态进行预测。将基线代入公式（3.25），用当前策略产生的轨迹的值减基线，得到新的计算方式，其中代表第次采样时的环境。

这样使用得到的结果有正值也有负值，在训练的过程中就能使得我们希望得到的动作概率会更高，而不是训练时所有动作的概率都在下降。

基线估计器是一个简单的序列网络，它输出的基线取值通过神经网络进行估算。它接受的输入是当前网络的服务功能链请求序列，输出的是估计的基线，因此仍然需要一个编码器对输入的序列进行编码。编码器的LSTM单元隐藏层数量与前文预测放置策略的Seq2Seq模型隐藏层数量相同。基线估计器输入的embedding向量长度为10，它使用随机梯度下降，并通过Adam优化器对权重进行优化，误差计算方式采用均方误差，对它预测出的值与值之间的均方误差进行训练。

# 4 实验分析

## 4.1实验环境

实验是在本地环境运行，没有租赁云服务器。机器的内存为16G，显卡为GeForce GTX 1070Ti，处理器为Interl(R) Core(TM) i7-8750H。

4.2 参数设置

实验所使用的网络环境实例有两种大小，大规模网络环境实例相较于小规模网络环境实例增加了一倍的主机数量以供服务功能链进行放置。

表4.1 小规模网络环境实例参数设置表

|  |  |
| --- | --- |
| 参数名 | 参数值 |
| 链路带宽（Mbps） | 1000,1000,500,400,300,300,300,300,300,300 |
| 链路延迟（ms） | 30,50,10,50,50,50,50,50,50,50 |
| CPU个数 | 10,9,8,7,6,6,6,6,6,6 |

表4.2 大规模网络环境实例参数设置表

|  |  |
| --- | --- |
| 参数名 | 参数值 |
| 链路带宽（Mbps） | 1000,1000,500,400,300,300,300,300,300,300,  300, 300, 300, 300, 300, 300, 300, 300, 300, 300, |
| 链路延迟（ms） | 30,50,10,50,50,50,50,50,50,50,  50, 50, 50, 50, 50, 50, 50, 50, 50, 50, |
| CPU个数 | 10,9,8,7,6,6,6,6,6,6,  6,6,6,6,6,6,6,6,6,6 |

VNF的相关参数在VNF字典中设置，它规定了不同种类VNF在放置到服务器上所要求的资源。

表4.3 VNF字典

|  |  |
| --- | --- |
| 参数名 | 参数值 |
| VNF要求带宽（Mbps） | 100,80,60,20,20,20,20,20 |
| VNF计算产生的时延（ms） | 100,80,60,20,20,20,20,20 |
| VNF要求CPU个数 | 4,3,3,2,2,2,1,1 |

用于计算能耗的参数设置为如下值：

用于计算由于违反了限制而产生的惩罚的参数设置为如下值：

表4.4 神经网络部分的超参数设置表

|  |  |
| --- | --- |
| 超参数名称 | 值 |
| Batch大小 | 128  0.0001  1  10  0.1  6 |
| Agent学习率  Agent梯度L2范数裁剪阈值  Seq2Seq模型中embedding大小  基线估计器学习率  每种服务功能链长度训练模型个数 |

表4.5 训练中epoch数量， LSTM隐藏层数量， LSTM层数设置表

|  |  |  |  |
| --- | --- | --- | --- |
| SFC长度 | epoch数量 | LSTM隐藏层数量 | LSTM层数 |

|  |  |  |  |
| --- | --- | --- | --- |
| 12  14  16  18  20  24  28  30 | 20000  20000  20000  20000  50000  50000  50000  50000 | 32  32  32  32  64  64  64  128 | 1  1  1  1  3  3  3  4 |

表4.6 遗传算法部分的参数设置表

|  |  |
| --- | --- |
| 参数名称 | 值 |
| 种群迭代次数 | 1000  50  同服务功能链序列长度  5  轮盘赌选择  单点交叉（single point crossover）  0.1  0.05  random |
| 种群中染色体的数量  解决方案的基因数量  种群中保持的父母数量  父母选择类型  交叉操作类型  交叉操作的概率  变异操作的概率  变异操作类型 |

4.3 结果分析

由于VNF-FGE问题是NP-hard问题，它不存在数据集以供训练或测试。这也是选择使用强化学习方法的原因之一。为了产生数据，我们在设定好底层网络环境，以及设定好VNF字典，来产生随机的SFC请求以供测试。产生的SFC请求内的VNF类型是从VNF字典中随机选取的，并且会根据当前需要测试的模型所训练的SFC长度，VNFD大小，和batch的大小来随机生成。模型将通过这些产生的数据来进行训练或测试。

4.3.1 模型评估

我们对不同长度的服务功能链序列进行了测试，长度为12，14，16，18的服务功能链将放置在小规模的网络环境实例上，长度为20，24，28，30的服务功能链将放置在大规模的服务器集群上。

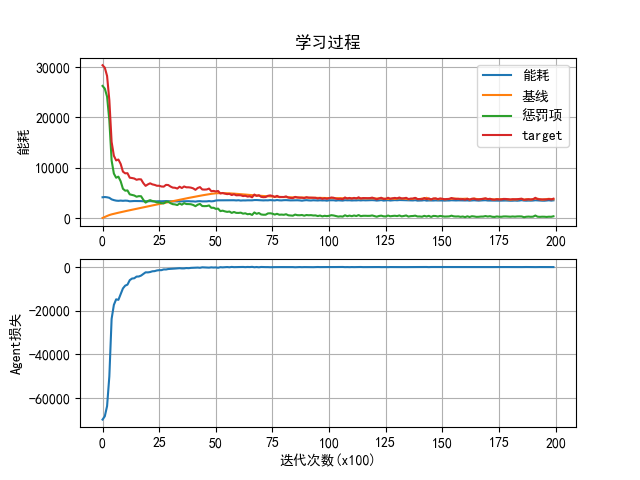


图4.1 小规模低占用率网络环境实例下训练过程

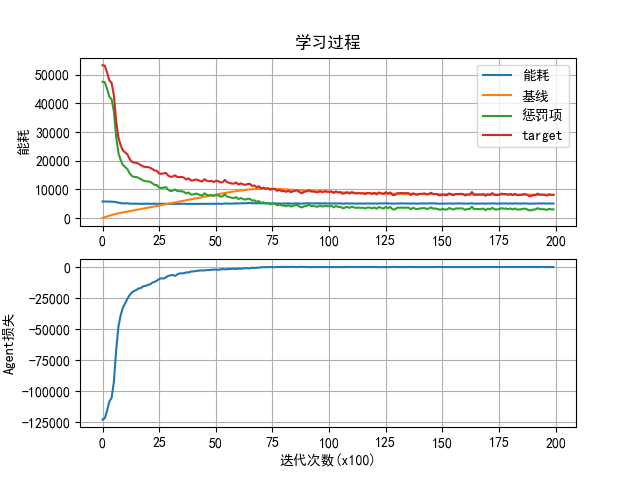


图4.2 小规模高占用率网络环境实例下训练过程

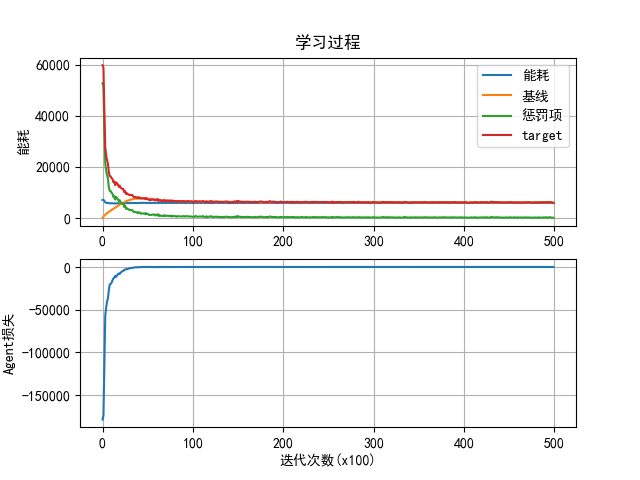


图4.3 大规模低占用率网络环境训练过程

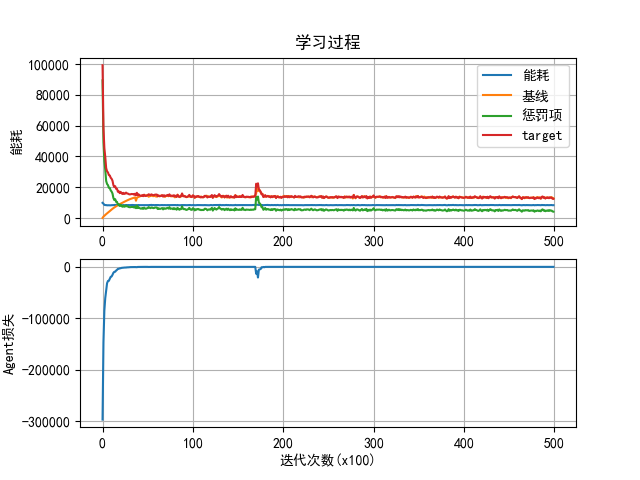


图4.4 大规模高占用率网络环境训练过程

我们将模型的学习过程分为四种情况进行对比，图4.1，4.2分别代表小规模网络环境下低占用率与高占用率的学习过程，图4.3，4.4分别代表大规模网络环境下低占用率与高占用率的学习过程。占用率的变化是通过改变服务功能链的长度，即改变需要放置在服务器上的虚拟网络功能数量实现的。在小规模网络环境中，选取长度为12的服务功能链序列的模型学习过程作为低占用率的图像，选取长度为18的服务功能链序列的模型学习过程作为高占用率的图像；在大规模网络环境中，选取长度为20的服务功能链序列的模型学习过程作为低占用率的图像，选取长度为30的服务功能链序列的模型学习过程作为高占用率的图像。

上半部分子图为能耗energy，基线baseline，约束惩罚penalty以及目标函数target随训练过程的曲线；下半部分为强化学习Agent在训练过程中的loss曲线。可以看到，两种环境下能耗energy在逐步下降，最终趋于稳定。基线baseline在训练过程中逐渐上升并稳定，在稳定后与target值基本重合，可以看到Critic网络训练的结果是非常理想的。约束惩罚penalty也在训练过程中逐渐下降并稳定。target的值等于能耗与惩罚之和，它也是基线的训练目标值，可以看到在模型收敛后它与基线基本相等；而下半部分Agent的损失为负值，因为我们求得是最小值，而强化学习中往往是想要最大化奖励。它随着训练逐渐上升并稳定。模型在训练过程中收敛。

可以看到，两种网络环境下低占用率的学习过程中，在模型的开始阶段，Agent产生的随机序列违反了许多约束条件，导致这些放置策略会产生很高的惩罚项。随着学习的进展，Agent通过随机梯度下降优化网络参数，改进它产生的放置策略，来最小化训练目标target。在训练结束时，由于违反限制而产生的惩罚项几乎为0。

在高占用率的情况下学习过程与低占用率的学习过程中也类似，但是由于序列过长，很有可能在网络环境中无法寻找到能够满足限制的策略，因此在学习过程结束时低占用率和高占用率的惩罚项分别收敛到8000，4000左右。

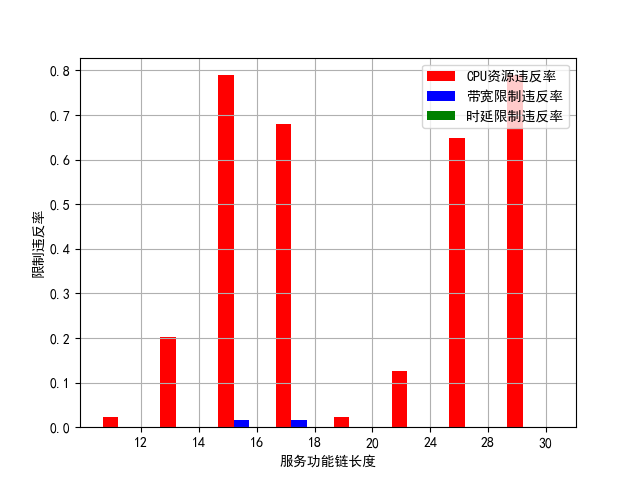


图4.5 三种资源限制违反率随服务功能链请求序列长度增长的变化

图4.5显示随着服务功能链序列长度的增长，三种底层网络资源限制的违反率的变化。最为明显的是CPU资源限制的违反，它随着服务功能链序列长度的增长大幅度增长。服务功能链长度为12~18的为小规模网络环境，长度为20~30的为大规模网络环境，在这两部分网络环境中，服务功能链序列在逐渐增加时，网络的占用率是逐渐增加的，因此违反CPU资源限制的比率也在逐渐增加。其中服务功能链长度为16时的测试结果中违反CPU资源限制的比率大于长度为18时违反CPU资源限制的比率，我猜想是长度为16时训练的模型找到的是局部最优解。

另外两项资源限制分别是带宽限制和时延的限制。带宽限制的违反只出现在服务功能链长度为16和18的测试结果中，而时延限制的违反则从未出现过。这有可能是在定义违反的阈值时将其定义的过大，后续可以对其进行改进。

4.3.2 算法对比

我们在用于解决离线VNF-FGE问题的启发式算法和元启发式算法中分别挑选了一种并实现，来与我们的算法进行对比。其中启发式算法挑选的是First-Fit算法，First-Fit算法大体思想是寻找到第一个能够放置的位置，且将VNF放置在这里不会违反任何约束，与操作系统中物理内存分配算法的First-Fit类似，因此它得出的放置策略是不存在违反约束的情况的。如果某个服务功能链请求无论如何都无法满足，则First-Fit将无法找到可行解。它的伪代码实现见附录6.2节；元启发式算法挑选的是遗传算法，它的伪代码实现见附录6.3节。三种算法效果对比情况如图4.5所示：

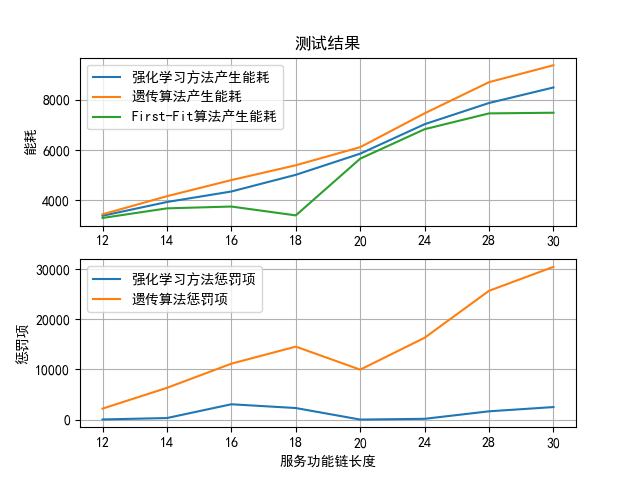


图4.5 强化学习，遗传算法，First-Fit算法效果对比

我们将服务功能链长度作为横坐标，将三种算法产生的能耗与限制产生的惩罚在图4.5中进行对比。测试包含128个batch，每个batch都会根据VNF字典生成随机的服务功能链序列。可以看到，使用本文提出的算法产生的结果介于First-Fit算法和遗传算法之间，且此处统计的First-Fit算法产生的能耗均不包括无法找到可行解的情况；由于First-Fit算法不会有违反限制而产生的惩罚，我们将本文提出的算法与遗传算法进行对比，可以看到随着服务功能链长度的增加，遗传算法产生的惩罚也在逐渐增加，而强化学习方法产生的惩罚则变化不大，相较于遗传算法的惩罚要小很多，将能耗与惩罚项相加，求出的目标函数值相较于遗传算法平均降低了59.1%；在高占用率的网络环境中，本文提出的算法仍能够找到较优的可行解，而First-Fit算法则很难找到可行解。在高占用率的网络环境下，First-Fit算法能够找到的可行解不超过所有测试个数的40%。同时，在网络占用率变小时，也就是服务功能链长度从18变为20，从小规模网络环境变为大规模网络环境时，本文提出的算法的惩罚项有明显的减小，这与我们在模型评估时的想法是对应的。

# 5 总结与展望

5.1 总结

网络功能虚拟化将网络功能以软件的形式进行封装，它需要找到放置策略将其映射到底层的物理网络上，而VNF-FGE问题就是寻找VNF放置策略，同时使得这组放置策略在满足VNF要求的资源配置情况下，尽量满足底层网络的资源限制。现有大量解决VNF-FGE问题的方法是基于启发式方法或是元启发式方法，然而想要推导出这两种方法需要大量时间以及推导过程相对复杂，现在的一些较新的文献开始采用强化学习方法来寻找VNF的最优放置策略。

本文使用强化学习方法中的策略梯度方法与Seq2Seq模型，对输入的服务功能链请求输出序列中每个VNF的放置位置。在策略梯度方法中，由于放置策略的搜索空间过大，我们无法通过遍历所有放置策略来求得期望值，因此采用了蒙特卡罗方法求得状态的奖励期望和动作的奖励期望值。为了针对长序列，我们在Seq2Seq模型中加入了Bahdanau 注意力机制，使Seq2Seq模型中的解码器解码当前时间步时网络能将注意力尽可能多的集中于对应编码时刻的隐含状态，让Seq2Seq模型对长序列也能保持较好的效果。同时，为了防止长序列中梯度过大时连乘造成梯度爆炸的情况，在Agent中添加了梯度L2范数裁剪阈值，将梯度最大值设定为1。

在Agent产生放置策略的过程中，不管模型产生什么策略，这些策略都会使得环境的能耗增加，为了减少梯度的方差，加快模型收敛速度，我们在Agent中添加了一个基线估计器，它的内部包含一个编码器，它将根据网络的状态估计出当前Agent产生的放置策略将产生的能耗与惩罚的期望作为基线值，以此基线值为标准对放置策略进行评估。为了防止采用策略梯度方法学习的模型陷入局部最小值或是鞍点，我们在训练相同长度的服务功能链模型时同时训练了多个模型，并在推理时同时使用这些模型，将最优的结果返回。

我们将本文的算法与启发式算法中的First-Fit算法，元启发式算法中的遗传算法进行对比，我们的算法相较于另外两种算法有着更好的效果。

5.2 展望

在我们的模型中，我们考虑的底层网络结构是相对简单的星型拓扑结构。而现有的网络拓扑结构相对多变并且相对复杂，后续的工作可以考虑将强化学习中的环境更换为这些更加复杂的网络环境。

同时，我们的方法将限制放松，将惩罚项与网络能耗的和作为优化目标，使得在底层网络资源紧张，占用率高的情况下能够找到可行的放置策略。然而这样做的代价是我们无法确保找到满足限制条件的解，即如果存在惩罚项与网络能耗的和的放置策略小于满足限制的网络能耗的放置策略，则模型会优先选取违反了限制的放置策略。

此外，尽管我们的方法中针对策略梯度带来的局部最优问题采用了同时训练多个模型，并将这些模型中表现最好的结果返回的解决方法，然而这没有从根本上解决这个潜在的问题。后面的研究可以尝试是否存在更好的方法能够找到目标函数的全局最优解。

# 致 谢

终于迎来了大学最后的一块拼图，在这篇论文写完时，我的大学生活基本结束，人生又要迎来新的阶段了。这四年时光回想起来过的真的非常快，但是总体是非常充实而幸福的。感谢我的母校重庆大学对我的培养，我非常幸运在这里能够遇到一群优秀的同学共同学习进步，遇到耐心的老师无私地答疑解惑，你们在我的学习过程中都起到了非常重要的作用，给了我许多帮助。

我衷心的感谢郭松涛老师，郭得科老师在百忙之中给予的指导，在教学与科研任务十分繁重时也会抽出时间来组会，为我讲解我毕设选题的相关知识。

同时非常感谢刘贵燕学姐，当我在学习的过程中产生了问题，总是能够悉心的指导，帮助我理清思路，解答我的疑惑，学姐给予了我非常大的帮助。

最后感谢涂佳雪同学，在我对毕设焦虑，烦恼的时候，让我平静下来认真思考，我们一起度过了一段幸福的大四时光，我相信这份幸福还会持续下去。

谢谢大家的帮助与陪伴，我会继续努力，不坠青云之志。

# 参 考 文 献

[1] 龚峰, 程闻博. 网络功能虚拟化技术的发展现状与面临的挑战[J]. 新型工业化, 2018, 8(6):6.

[2] Zheng D, Peng C, Liao X, et al. Toward optimal hybrid service function chain embedding in multiaccess edge computing[J]. IEEE Internet of Things Journal, 2019, 7(7): 6035-6045.

[3] Schardong F , Nunes I , Schaeffer-Filho A . NFV Resource Allocation: a Systematic Review and Taxonomy of VNF Forwarding Graph Embedding[J]. Computer Networks, 2021, 185(3):107726.

[4] Jemaa F B , Pujolle G , Pariente M . QoS-Aware VNF Placement Optimization in Edge-Central Carrier Cloud Architecture[C]// Global Communications Conference. IEEE, 2017.

[5] Cohen R, Lewin-Eytan L, Naor J S, et al. Near optimal placement of virtual network functions[C]//2015 IEEE Conference on Computer Communications (INFOCOM). IEEE, 2015: 1346-1354.

[6] Chi P W , Huang Y C , Lei C L . Efficient NFV deployment in data center networks[C]// IEEE. IEEE, 2015:1-6.

[7] Lofberg J . YALMIP : a toolbox for modeling and optimization in MATLAB[C]// IEEE International Symposium on Computer Aided Control Systems Design. IEEE, 2005:1-6.

[8] Rost M , Schmid S . On the Hardness and Inapproximability of Virtual Network Embeddings[J]. IEEE/ACM Transactions on Networking, 2020, PP(99):1-13.

[9] Yu S , Bo J , Gupta G R , et al. Provably Efficient Algorithms for Joint Placement and Allocation of Virtual Network Functions[J]. IEEE, 2017.

[10] Tomassilli A , Giroire F , Huin N , et al. Provably Efficient Algorithms for Placement of Service Function Chains with Ordering Constraints[C]// IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. IEEE, 2018.

[11] Sallam G , Ji B . Joint Placement and Allocation of VNF Nodes With Budget and Capacity Constraints[J]. IEEE/ACM Transactions on Networking, 2021, PP(99):1-14.

[12] Karmarkar N , Karp R M . An efficient approximation scheme for the one-dimensional bin-packing problem[C]// 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982). IEEE, 2008.

[13] Khuller S, Moss A, Naor J S. The budgeted maximum coverage problem[J]. Information processing letters, 1999, 70(1): 39-45.

[14] Chantre H D , Fonseca N . Redundant placement of virtualized network functions for LTE evolved Multimedia Broadcast Multicast Services[C]// Icc IEEE International Conference on Communications. IEEE, 2017.

[15] Yala L , Frangoudis P A , Ksentini A . Latency and Availability Driven VNF Placement in a MEC-NFV Environment[C]// 2018 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019.

[16] Manias D M , Jammal M , Hawilo H, et al. Machine Learning for Performance-Aware Virtual Network Function Placement[J]. IEEE, 2020.

[17] Kennedy J , Eberhart R . Particle swarm optimization[C]// Proceedings of ICNN'95 - International Conference on Neural Networks. IEEE, 1995.

[18] Hawilo H , Jammal M , Shami A . Network Function Virtualization-Aware Orchestrator for Service Function Chaining Placement in the Cloud[J]. IEEE Journal on Selected Areas in Communications, 2019.

[19] Pedregosa F , Varoquaux G , Gramfort A , et al. Scikit-learn: Machine Learning in Python[J]. 2012.

[20] Mijumbi R , Gorricho J L , Serrat J , et al. Design and evaluation of learning algorithms for dynamic resource management in virtual networks[C]// Network Operations & Management Symposium. IEEE, 2014:1-9.

[21] Mijumbi R , Gorricho J L , Serrat J , et al. Neural network-based autonomous allocation of resources in virtual networks[C]// European Conference on Networks & Communications. IEEE, 2014.

[22] Yao H, Zhang B, Zhang P, et al. RDAM: A reinforcement learning based dynamic attribute matrix representation for virtual network embedding[J]. IEEE Transactions on Emerging Topics in Computing, 2018, 9(2): 901-914..

[23] Gers F A, Schmidhuber J, Cummins F. Learning to forget: Continual prediction with LSTM[J]. Neural computation, 2000, 12(10): 2451-2471.

[24] [李宏毅机器学习笔记 – Github](https://datawhalechina.github.io/leeml-notes/)（https://datawhalechina.github.io/leeml-notes/）

[25] Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks[J]. Advances in neural information processing systems, 2014, 27.

[26] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.

[27] Bahdanau D , Cho K , Bengio Y . Neural Machine Translation by Jointly Learning to Align and Translate[J]. Computer Science, 2014.