Research on Application Strategy of Deep Learning of Internet of Things Based on Edge Computing Optimization Method

Huanyu Wu1*, Yating Guo1 and Jingcheng Zhao1

1School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, Anhui, 230601, China

*Corresponding author’s e-mail: wuhuanyu0426@gmail.com

Abstract. Due to the limitations of network performance IoT deep learning model, in order to optimize the performance of IoT network, this paper proposes an application strategy of IoT deep learning based on edge computing optimization method. Based on the multi-layered structure of learning, the edge calculation method is optimized, and the reduced intermediate data is uploaded at the edge node, thereby reducing the network traffic from the IoT device to the cloud server. On the basis of considering the limitation of edge node service capability in the edge computing process, the optimal strategy of task offload scheduling is formulated to improve the performance of the IoT deep learning model based on edge computing. The experimental results show that the IoT deep learning application strategy based on the edge computing optimization method can efficiently execute multiple deep learning tasks in the edge computing environment, which is superior to other algorithms.

1. Introduction

Extending the current Internet through interconnected objects and devices and their virtual representations has become a trend, and the contribution of the Internet of Things (IoT) is through the number of interconnections between things and for the benefit of society. The value of information created by the transformation of information into knowledge increases. Due to the large amount of data and complex data content of IoT, traditional machine learning technology is more cumbersome when processing IoT data. For example, data processing requires a large number of feature extraction operations, while deep learning only needs to pass data directly to the network effectively, which overcomes the limitations of traditional machine learning techniques. In addition, deep learning can automatically extract new features for different problems, can accurately learn the advanced features of multimedia information, and improve the efficiency of processing multimedia information[1-4].

In order to improve the performance of the IoT deep learning model, this paper introduces the deep learning of the Internet of Things into the edge computing environment to improve learning performance and reduce network traffic[5]. Elastic models compatible with different deep learning models are developed. Due to the different amount of intermediate data and preprocessing overhead of different deep learning models, This paper proposes a scheduling problem that maximizes the number of deep learning tasks at the edge network's limited network bandwidth and service capabilities. In order to guarantee the quality of service of each IoT deep learning service in scheduling, offline and online scheduling algorithms are designed to solve this problem. Experimental results show that the proposed solution is superior to other optimization methods.
2. Deep learning based on edge calculation

IoT devices generate large amounts of data and transmit it to the cloud for further processing. These data include multimedia information. Traditional multimedia processing technologies require complex calculations and are not suitable for IoT services. Due to deep learning technology, multimedia information processing is improved. The efficiency of more and more works is beginning to introduce deep learning into multimedia IoT services. Deep learning improves the multimedia processing efficiency of IoT services, but its communication performance will be the bottleneck for improving processing efficiency. Edge computing is a solution for transferring collected data from IoT devices to cloud services. In the edge computing environment, since only intermediate data or results need to be transferred from the device to the cloud service, the transmission data is reduced and the network pressure is reduced. Therefore, edge calculation is effective for deep learning tasks because in the deep learning network layer, the extracted feature size is reduced by the filter. Figure 1 shows the edge computing structure of the IoT deep learning task proposed in this paper. The structure consists of cloud and edge layers and a typical edge computing structure.

In the edge layer, the edge server is deployed in the IoT gateway to process the collected data. At first, the deep learning network is trained on the cloud server. After the training, the learning network is divided into two parts, one part is close to the lower layer of the input data, and the other part is close to the higher layer of the output data. Deploy a portion with a lower tier to the edge server and a portion with a higher tier to the cloud for offload processing. Therefore, the collected data is input to the first layer in the edge server, which loads the intermediate data from the lower layer and then transfers the data as higher layer input data to the cloud server. The intermediate data generated by the higher layers is smaller than the intermediate data generated by the lower layers, and deploying more layers to the edge server can reduce more network traffic. However, compared to cloud servers, edge servers have limited server capacity, and it is impossible to handle unlimited tasks in edge servers. Each layer in the deep learning network brings additional computational overhead to the server, so only part of the deep learning network can be deployed to the edge server. At the same time, because different deep learning networks and tasks have different intermediate data and computational overhead, effective scheduling is needed in the edge computing structure to optimize the deep learning of the Internet of Things. In response to this problem, this paper designs an effective scheduling strategy, which is explained in the next section. In the edge computing structure of IoT deep learning, the deep learning network uses a deep self-encoder, and the network structure is shown in Figure 2.

---

**Figure 1.** Edge computing structure of IoT deep learning

**Figure 2.** Deep self-encoder network structure
For the input vector $X$ to be encoded as a low-dimensional eigenvector $A$ and then reconstructed to a vector $Y$ that approximates $X$, then in the depth self-encoder, the lower dimension $A$ can be expressed as:

$$A = f(\sum WX + b)$$

(1)

Where: $f(\cdot)$ is the input mapping function, this article uses the sigmoid function, $W$ is the weight vector, and $b$ is the offset vector.

3. Scheduling strategy for IoT deep learning in edge computing

Firstly, the scheduling problem in the edge computing structure of IoT deep learning is expounded, and then the solution is proposed. In a given edge computing environment, the set $E$ is used to represent all edge servers, and $E_i$ is the edge server in set $E$. From the edge server $E_i$ to the cloud server, the value $C_i$ is used to represent the service capacity, and $B_i$ is the network bandwidth. Because there is some interaction traffic between the edge server and the cloud server, a threshold $V$ is also added to avoid network congestion. Therefore, the maximum available bandwidth between the edge server $E_i$ and the cloud server is represented by $B_i \cdot V$.

The mobile node $i$ generates a scheduling task of $K_i(t)$ bits in each time period $T$. For the IoT, the amount of scheduling tasks of $i$ at $T$ is defined as $T_i(t)$, and the update rule in the next time period can be expressed as:

$$T_i(t+1) = T_i(t) + K_i(t) - U_i(t) - V_i(t)$$

(2)

Where: $U_i(t)$ represents the amount of local task execution, $V_i(t)$ represents the amount of unloaded tasks.

$T_i$ represents a deep learning task, and the number of deep learning network layers of task $T_i$ is $N_j$. Assume that the reduced data size is close to the average of each task with different input data. The average ratio of the intermediate data size ($k \in [1, N_j]$) generated by the $k$th layer to the total input data size is represented by $R_{ij}$. For task $T_i$ and edge server $E_i$, the allocated bandwidth is represented by $B_{ij}$. Let $D_{ij}$ denote the input data size per time unit of task $T_i$ in Edge Server $E_i$. If the $k$ layers of the task $T_i$ are placed in the edge server, the transfer latency of the task $T_i$ in the edge server $E_i$ can be expressed as $D_{ij} \cdot R_{ij} / B_{ij}$. To ensure quality of service, the transmission delay should be less than the maximum value indicated by $Q_j$. For task $T_i$, the computational overhead of the input data unit after the $k$th layer is denoted by $I_{ij}$. Therefore, for task $T_i$, the overhead of the computer in the edge server $E_i$ is $I_{ij} \cdot D_{ij}$.

Scheduling the problem of IoT deep learning network layer in computing: Given the edge computing structure, the scheduling problem attempts to allocate the maximum task in the edge computing structure by deploying the depth learning layer in the IoT edge server, so that the required transmission delays each task can be guaranteed, as shown below:

$$\max \sum_{i=1}^{\left|E\right|} \sum_{j=1}^{\left|E\right|} X_{ij}
\text{s.t. } \sum_{i=1}^{\left|E\right|} B_{ij} \leq B_{ij} \cdot V
B_{ij} \cdot D_{ij} \cdot R_{ij} / B_{ij} \leq Q_j
\sum_{j=1}^{\left|E\right|} I_{ij} \cdot D_{ij} \cdot X_{ij} \leq C_i$$

(3)

If task $T_i$ is deployed in edge server $E_i$, then $X_{ij}=1$; otherwise, $X_{ij}=0$.

In order to solve the above problems, this paper proposes an offline algorithm and an online algorithm for solving scheduling problems. The offline scheduling algorithm first finds the $K_i$ that maximizes the value of $R_{ij} / I_{ij}$ and the edge server $E_i$ that maximizes the input data of the task $T_i$, and then the algorithm sorts all tasks in ascending order of the maximum input data size. The schedule first deploys the task $T_i$ with the smallest input data to the edge server. The algorithm traverses all edge servers and checks if
the edge server has sufficient service capabilities and network bandwidth to deploy task $T_j$. If all edge servers have sufficient service capacity and bandwidth, the algorithm deploys task $T_j$ to all edge servers. If the edge server does not have enough upload bandwidth or service capacity, the algorithm will change the value of $k$ and find that the value of $k$ is appropriate for the deployment task $T_j$ in all edge servers. If the edge server does not have sufficient service capacity or network bandwidth even after changing $k$, the scheduling algorithm will not deploy task $T_j$ in the edge server.

In the worst case, the complexity of the offline algorithm is $O(|T|\cdot |E|^2 \cdot K)$, where $K$ is the maximum number of deep learning network layers per task. Since the number of tasks is much larger than the number of edge servers and deep learning network layers, the complexity of the proposed algorithm is $O(|T|)$, which is good enough for actual scheduling, and the efficiency of the algorithm is approximately $2/V$.

At the same time, an online scheduling algorithm is designed to determine the deployment of the task $T_j$ when it arrives. Since task scheduling has little information about feature tasks, deployment decisions are based on historical tasks. Use $B_{\text{max}}$ and $B_{\text{min}}$ to represent the maximum and minimum expected bandwidth of the task, respectively. Thus, for the task $T_j$, $k^m_j$ and $P_j$ are first calculated, then a value defined:

$$\Phi(C_j, m) \leftarrow (B_{\text{min}} \cdot E / B_{\text{max}})^{C_j, m} \cdot (B_{\text{max}} \cdot E)$$

Where: the remaining service capacity of the edge server $E_{ij}$ is $C_j, m$. In case:

$$(B_j \cdot m - D_j \cdot m_j \cdot R_j \cdot m_j / Q_j) \cdot (C_j \cdot m - D_j \cdot m_j \cdot I_j \cdot m_j) \leq \Phi(C_j, m)$$

And other edge servers have sufficient bandwidth and service capacity, and the scheduling algorithm deploys the task $T_j$ to the edge server. The approximate ratio of the online algorithm is:

$$\text{Ratio} = 1 / \left[ \ln \left( \frac{B_{\text{max}}}{B_{\text{min}}} \right) + 1 \right] \cdot V$$

4. Experimental results and analysis

The experiment defined 10 different deep self-encoder networks, performing 10 deep self-encoder tasks using different networks and recording the number of operations and intermediate data generated in each layer. Figure 3 shows the computational overhead and reduced data size ratio indicators for deep networks. The deep learning network has five levels and different neuron settings.

As can be seen from Figure 3, the input data can be reduced by the deep learning network, more intermediate data can be reduced by the lower layer, and the computational overhead increases rapidly as the number of layers increases.
of edge servers in the network is increased from 20 to 90. The input data size for each task is set from 100KB to 1MB, and the number of layers for all deep networks is set to 5 to 10. The bandwidth of each edge server is evenly distributed between 10Mbit/s and 1Mbit/s, and the required delay time is 0.2s. We compare the performance of the online scheduling algorithm with two popular online scheduling algorithms, namely first in first out (FIFO) and low bandwidth priority deployment (LBF), and input a random sequence of 1000 tasks to the edge network. Both algorithms deploy the task to the edge server, and the number of edge servers is set to 50. The result is shown in Figure 4.

As can be seen from Figure 4, the FIFO algorithm deploys each task until there is not enough capacity and bandwidth. Therefore, after deploying 360 tasks, the FIFO algorithm will pop up the first deployed task for attaching future tasks and deploying. The number of tasks no longer increases or decreases as the number of input tasks increases. When capacity and bandwidth are insufficient, LBF is similar to FIFO. The number of deployed tasks no longer increases with the number of input tasks. When there is no space to deploy future tasks, LBF deletes the task with the largest bandwidth requirement. The online algorithm in this paper decides whether to deploy future tasks to the edge server. When the number of input tasks is close to 600, the online algorithm can deploy more tasks than FIFO and LBF. Therefore, when the capacity and bandwidth are insufficient, the number of tasks deployed by the online offload scheduling algorithm will increase with the input task and is better than the FIFO and LBF algorithms.

5. Conclusion

This article introduces deep learning of the Internet of Things into the edge computing environment to optimize network performance. The edge computing architecture allows edge nodes to reduce intermediate data for uploading data, reducing network traffic from IoT devices to cloud servers. For the limited service capabilities of edge nodes, an offload scheduling algorithm is proposed to maximize the number of tasks in the edge computing environment. The experimental results show that the IoT deep learning application and the offload scheduling algorithm for edge computing can increase the number of tasks deployed in the edge server while ensuring the quality of service requirements.

References

[1] Ahmed, E., Jones, M., Marks, TK. (2015) An Improved Deep Learning Architecture for Person Re-Identification. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition. Boston, pp:3908-3916.
[2] Li, H., Ota, K., Dong, MX. (2018) Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. IEEE NETWORK., 32: 96–101.
[3] Gulshan, V., Peng, L., Coram, M., et al. (2016) Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA-J AM MED ASSOC., 316: 2402–2410.
[4] Schmidhuber, J. (2015) Deep learning in neural networks: An overview. NEURAL NETWORKS., 61: 85–117.
[5] Shi, WS., Cao, J., Zhang, Q., et al. (2015) Edge Computing: Vision and Challenges. IEEE INTERNET THINGS., 3: 637-646.