applied sciences

MDPI

Article

TAPP: DNN Training for Task Allocation through Pipeline Parallelism Based on Distributed Deep Reinforcement Learning

Yingchi Mao [1,*], Zijian Tu [1], Fagang Xi [2], Qingyong Wang [1] and Shufang Xu [1]

[1] School of Computer and Information, Hohai University, Nanjing 211100, China; 201307040020@hhu.edu.cn (Z.T.); 181307040030@hhu.edu.cn (Q.W.); 20050012@hhu.edu.cn (S.X.)
[2] Huaneng Lancang River Hydropower Co., Ltd., Kunming 650214, China; xifagang@lcjgs.chng.com.cn
* Correspondence: yingchimao@hhu.edu.cn

Abstract: The rapid development of artificial intelligence technology has made deep neural networks (DNNs) widely used in various fields. DNNs have been continuously growing in order to improve the accuracy and quality of the models. Moreover, traditional data/model parallelism is hard to expand due to communication bottlenecks and hardware efficiency issues. However, pipeline parallelism trains multiple batches, reducing training overheads, so that it can achieve better acceleration effect. Considering the complexity of solving the pipeline parallel task allocation problem in heterogeneous computing resources, in this paper, a task allocation in pipeline parallelism (TAPP) based on deep reinforcement learning, is proposed. In TAPP, the predictive network is trained by a policy gradient until it obtains the optimal pipeline parallel task allocation scheme and speeds up the model training. Experimental results show that, on average, the single-step training time of TAPP is decreased by 1.37 times and the proportion of communication time is reduced by 48.92%, compared with the data parallelism, bulk synchronous parallel (BSP).

Keywords: computing node; pipeline parallelism; heterogeneous computing resources; task allocation

check for updates

Citation: Mao, Y.; Tu, Z.; Xi, F.; Wang, Q.; Xu, S. TAPP: DNN Training for Task Allocation through Pipeline Parallelism Based on Distributed Deep Reinforcement Learning. *Appl. Sci.* **2021**, *11*, 4785. https://doi.org/10.3390/app11114785

Academic Editor: Byung-Gyu Kim

Received: 29 April 2021
Accepted: 20 May 2021
Published: 23 May 2021

1. Introduction

DNNs have been popularly used to solve various problems such as image classification [1–3], speech recognition [4–6], and language translation [7,8] by virtue of their complex modeling capabilities. The essential reason why DNNs can make such huge progress is that they simulate the human brain neural-learning system, which enables the machine to learn high-level features from a huge amount data by increasing the layers of the network. The size of DNN models (i.e., the number of parameters) have been continuously increasing in order to improve the accuracy and quality of models and to deal with complex features of data [9–11]. The size of input data and batches used for training have also increased to achieve higher accuracy and throughput [12,13]. OpenAI researchers have already trained an autoregressive language model called GPT-3 with 175 billion parameters, 10× more than any previous non-sparse language model [14]. In addition, with the increase in the number of parameters and the complexity of the DNNs, more data is needed to train the model to avoid overfitting.

The model parameters and intermediate outputs that need to be in the memory during training cannot be stored in a single computing node due to the limited memory space. In recent years, heterogeneous computing platforms represented by general processing units (GPUs) and field programmable gate arrays (FPGAs) have the advantage of high performance and are widely used to accelerate the training of DNN models. More and more scholars have begun to study distributed deep learning in depth, looking at how to divide training data, allocate training tasks, allocate computing resources, and integrate distributed training results in order to achieve the balance of training speed and training accuracy. For training large DNN models, data parallelism [15,16], which employs multiple workers using parameter servers or all-reduce communication, and model parallelism [17,18], which divides the network layers of a DNN model into multiple partitions and assigns each partition to a different GPU, have commonly been leveraged. Furthermore, to mitigate the critical issue of low GPU utilization of naive model parallelism, pipelined model parallelism, where minibatches are continuously fed to the GPUs, one after the other, and processed in a pipelined manner, has recently been proposed [13]. In this work, we design a training model, a task allocation in pipeline parallelism (TAPP) based on deep reinforcement learning aimed at the low efficiency of the asynchronous pipeline parallel training method. TAPP can improve the efficiency of model training without loss of precision.

In summary, the continuous growth of training data sets and DNN model scales have caused a single computing node to be unable to efficiently complete training tasks, and the various drawbacks of traditional distributed parallel training methods will cause expensive training costs. Therefore, combining the characteristics of DNN models, it is significant to study the use of distributed computing clusters to optimize existing training methods, reduce model training costs, and speed up model training.

The contributions of this paper are as follows:

1. Task partition is based on the feedforward neural network. The DNN model is grouped by layers. For a DNN model, the task partition network can divide each layer into corresponding groups. For each layer, the partition network generates the probability value of each packet and divides it into the packet with the highest probability.
2. Attention mechanism is introduced to task allocation. According to the grouping result of the task partition network, the task allocation network allocates corresponding computing nodes for each grouping. Before the task allocation network is processed, the information of each group needs to be integrated. Similar to the task partitioning network, the allocation network also transforms the grouping results into word vectors. The allocation network considers the operation name, parameter metric, and the next packet information of all layers in the whole packet. In this paper, the average of the relevant information of the containing layer in the group is taken as the word vector of the group.
3. Strategy gradient joint training is carried out. The prediction network uses the policy gradient to train the task partition and allocation network. The purpose of the network is to get the task allocation scheme which can minimize the total training time of a single training batch. The DNN model is trained in the total time of a batch under the condition of the task allocation scheme, including a forward calculation, a backward propagation, and a parameter update. In order to avoid running time measurement variance, we allocate a scheme to each predicted task. Then the DNN model is run on the actual computing node according to the scheme, the running time of the DNN model is measured several times to calculate the average value, the reward function of the current scheme is calculated, the parameters of the prediction network model are updated backwards, and the task allocation scheme with the shortest actual running time is obtained through several iterations.

2. Related Works

Data parallelism and model parallelism are the most commonly used methods in distributed deep learning model training. For computationally intensive operations with a small number of training parameters (e.g., convolutional layers), data parallelism is effective, however, for operations with a large number of parameters (e.g., fully connected layers), data parallelism is noneffective. Although model parallelism eliminates parameter synchronization between one computing node and another, it requires intermediate computation results to be transmitted. Model parallelism usually achieves faster training speed than data parallelism due to the used batch data volume being smaller than in data parallelism. The communication mechanism of the parallel training method can select a synchronous parameter update such as data-parallel stochastic gradient descent (DSGD) [19], elastic averaging stochastic gradient descent (EASGD) [20], or an asynchronous parameter update such as averaging stochastic gradient descent (ASGD) [21]. The synchronous communication method can maintain the weight consistency, but the hardware efficiency is low. However, the asynchronous communication method can improve the hardware efficiency, but it will cause weight inconsistency, which is not conducive to system convergence.

In contrast, pipeline parallelism can improve the training speed. PipeDream [22] divides the DNN layer into several stages, and it uses asynchronous parameter synchronization to inject multiple batches of training data into the system, so that each stage executes different batches of training tasks. At the same time, in order to ensure that the pipeline system achieves good load balancing, PipeDream allocates one or more GPUs for each stage to achieve data parallelism, reducing the difference in calculation time of each stage. The task allocation scheme based on a graph search has been extensively studied. For example, Isard et al. [23] proposed Quincy, which maps task allocation to the flow network and uses the minimum-cost maximum-flow (MCMF) algorithm to search for the optimal task allocation plan. Gog et al. [24] further improved Quincy, using a variety of MCMF optimization algorithms to reduce the delay of the task allocation algorithm and improve the efficiency of task allocation. Jia et al. [25] proposed the use of hierarchical parallelism, applying different parallel strategies for each layer of the model, turning the DNN parallel task allocation problem into a graph search problem, and using a graph search algorithm based on dynamic programming to find the global optimal parallel strategy. But this method is not suitable for DNN models such as language modeling and machine translation. In order to explore more parallel strategies, Jia et al. studied the dimensions of samples, operations, attributes, and parameters [26], extended the parallel dimension to four dimensions, and used Markov chain Monte Carlo (MCMC) method. This method searches for the optimal parallel policy in a larger search space to achieve training acceleration. However, the method has high time complexity and is not conducive to expansion. Pipe-torch [27] sorts the computing nodes by network bandwidth and assigns them to each stage of the DNN model in order to achieve pipeline parallel training acceleration, but it does not consider the heterogeneity of node computing capabilities. Park et al. [13] considered the computing power and memory differences between different nodes, so they proposed the concept of a virtual worker. The virtual worker is encapsulated by several heterogeneous GPUs, so that the computing power and storage capacity of each virtual worker are roughly the same, so as to achieve a semantically homogeneous cluster, but this method does not consider the network differences between GPUs. Dapple [28] also combines pipeline parallel and data parallel under the condition of isomorphism. It defines three kinds of device allocation mechanism of sensing topology to achieve good acceleration effect and save memory consumption. However, it does not consider the limitation of heterogeneous hardware.

Heuristic algorithms often fall into the local optimal solution. In order to explore the optimal solution of the global solution space, reinforcement learning algorithms have been widely studied. Mirhoseini [29] and others used neural network and reinforcement learning to divide the computational graph of TensorFlow to accelerate the training process of the target network.

Aimed at the low efficiency of the asynchronous pipeline parallel training method, this paper proposes a pipeline parallel task allocation method based on deep reinforcement learning to improve the efficiency of model training without losing accuracy.

3. Overall Framework

Considering the complexity of solving the pipeline parallel task assignment problem in heterogeneous computing resources, TAPP focuses on better acceleration effect. TAPP analyzes the operation of the DNN model to obtain information such as the number of model layers, names, parameters, and calculations. Two key novelties exist in this architecture. First, according to the acquired text information of the model parameters, the task allocation prediction network is constructed. Then the prediction network is trained using the policy gradient until the optimal pipeline parallel task allocation scheme is generated. The second novelty is that, according to the generated optimal task allocation plan, the model is deployed in heterogeneous computing nodes to complete the training task. The overall processing framework of TAPP is shown in Figure 1.

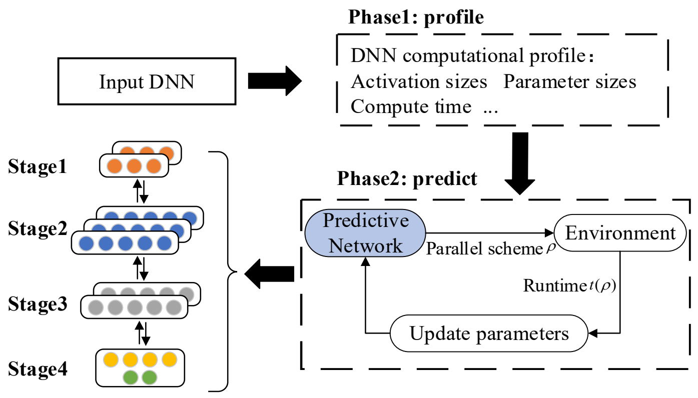


Figure 1. TAPP's automated mechanism to partition DNN layers into stages.

The task allocation includes two phases:

(1) Analyze the operating status of the target model and build a model parameter text library. Then analyze the short-term operating status of the model to be trained and record detailed model information, including the total number of layers of the model, the name of each layer, calculation time, calculation amount, parameter amount, and activation value, and build a text library of the model parameters to be trained.

(2) Construct a task allocation prediction network, according to the parameter information of the model to be trained. Then use the predictive network to generate a pipelined parallel training task allocation plan, ρ, for the DNN model to be trained. According to the allocation plan, ρ, several batches of training are performed in the experimental environment, sampling to obtain the average training time of a single batch $t(\rho)$, calculating the reward function, and updating the parameters in the prediction network backwards. After that, the prediction network is trained several times until it generates a task allocation plan with the smallest running time $t(\rho)$. If the model is deployed to heterogeneous computing nodes by the plan, a pipelined parallel training allocation plan for the target network to be trained will be obtained.

The DNN model is deployed and trained according to the optimal task allocation plan by the task allocation prediction network so that the pipeline parallel training acceleration can be realized. Since model parameter information is not affected by hardware conditions such as computing nodes, this article performs 1000 batches of training in a single computing node, records detailed information of each layer of the model, and builds a model parameter text library. Finally, a task allocation prediction network is constructed based on the obtained parameter text library.

4. Task Allocation Prediction Network

The task allocation prediction network generates a pipeline parallel training task allocation plan for one DNN model. The prediction network includes two phases: DNN task partition and task allocation. The task partition divides all layers of the model into groups. Each group corresponds to a stage in the pipeline system. Task allocation assigns several computing nodes to each group to achieve parallel training. Therefore, the prediction network is divided into two parts. The first part is the task partition network, which is responsible for dividing all layers in the DNN into several groups. The second part is the task allocation network, which is responsible for assigning each grouping task to the corresponding computing node. The predictive network uses deep reinforcement learning methods to solve the task allocation problem [29]. The difference is that the previous work uses manual grouping to divide the calculation operations in the model into several groups to reduce the complexity of allocating computing nodes. This method is not suitable for processing larger DNN models. In this paper, by a feedforward neural network instead of manual grouping, the DNN model is divided into layers to achieve rapid allocation.

The overall structure of the prediction network is shown in Figure 2. A feedforward neural network in which the last layer is the Softmax layer is used in the DNN partition network and the output size is equal to the number of groups. The task allocation network adopts the Seq2Seq model based on the attention mechanism [30]. The details in our network are as follows: We used a feedforward network with a hidden size of 64 and a Seq2seq model with a DNN hidden size of 256. For the encoder of the Seq2seq model, we used two layers of DNN to form a new DNN. We used a uni-directional DNN for the decoder. The Softmax output size was equal to the number of groups, which we set to 256 in our experiments. We train policies using Adam optimizer with a fixed learning rate of 0.1, gradient clipping of norm 1.0, tanh constant C = 5.0. The DNN partition network divides each layer in the model into corresponding groups. Once all layers are grouped, the average information of all layers in the group is used to generate the word vectors of the group. Then, these grouped word vectors are passed as input to the task allocation network. The task allocation network assigns corresponding computing nodes to each grouped task which can deploy the DNN model, according to the results of task partition and allocation network, to get the final pipeline task allocation plan.

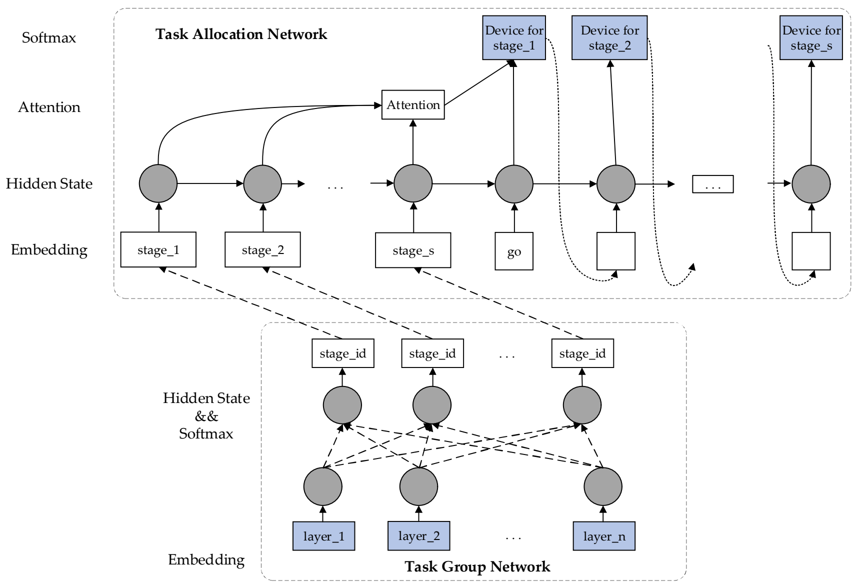


Figure 2. The overall framework of the prediction network. For the "go" in Figure 2, there is usually a "go" in the Seq2seq diagram, which represents an order of execution.

4.1. Task Partition Based on Feedforward Neural Network

The task partition network groups a DNN model by layer. For a DNN model, the task partition network can divide each layer into corresponding groups. The partition network adopts a feedforward neural network in which the last layer is the Softmax layer and the output type is equal to the number of groups. For each layer, the partition network generates its corresponding probability value for each group and divides it into the group with the highest probability.

Each layer of the DNN model includes several attributes, such as the name, operation, input, output, and dependent layers. In order to better evaluate the association relationship of each layer, this article focuses on the following attributes: the operation name of each layer, the parameter metrics, and the next associated layer.

(1) Operation name: In the DNN model, each layer corresponds to a different operation, such as convolution operation, pooling operation, and fully connected operation. Since the operation names are all character types, they are converted to word vectors. The experiment encodes the operation name of each layer and builds a dictionary library, which will be constantly updated during the training process. In the experiment, the width of the code is set to 16 bits and five commonly used operation names are selected: Convolution, Relu, Pooling, Fully-Connected, and Padding. For the remaining operation names, use Unknown instead.
(2) Parameter metric: The parameter metric of each layer of the model includes the number of parameters, the amount of calculation, and the size of the generated activation value. The required parameter quantity of each layer is calculated according to the model structure definition for which the unit is expressed in MB, the required calculation quantity according to the parameter quantity of each layer and the corresponding calculation operation is estimated for which the unit is expressed in GFLOPS, and the size of the generated activation value of each layer according to the parameter quantity of each layer and the corresponding calculation operation is estimated for which the unit is expressed in MB. Each metric is allocated 4 bits of coding width, that is, the total coding width of the parameter metric is 12 bits.
(3) Associated next layer: The DNN model is composed of several layers. The output of each layer is used as the input of other layers. In this paper, the DNN model is numbered layer by layer. The numbering starts from 1. The number is used to indicate the next layer associated with the current layer. Because the number of the next layer associated with each layer in the model is different, the number of experiments is set to 2, and each number is assigned a 2-bit code width. If there is no information in the next layer, we will fill it with -0.1.

For a DNN model, each layer in the model is traversed. After the above-mentioned processing steps, each layer obtains 32-bit vector information. This information is input into the task partition network to predict the partition result of the DNN model layers.

4.2. Task Allocation Based on Attention Mechanism

The task allocation network assigns a corresponding computing node to each group according to the grouping results of the task partition network. Before the task allocation network processes, it needs to integrate the information of each group. Similar to the task partition network, the allocation network also converts the grouping results into word vectors. The allocation network considers the operation names of all layers in the entire group, the parameter metric, and the next group information which is associated with them. In this paper, the relevant information of the contained layers in the group is averaged as the word vector of the group.

(1) Operation name: The word vectors of the operation names in the group are averaged to get the operation name information of the current group.
(2) Parameter metric: The parameter metric information of all layers in the group is averaged to get this part of information.
(3) The next set of information: One-hot encoding is used to indicate the dependency between groups. If a DNN layer in the current group is connected to the layer in the i^{th} group, this article will set the i^{th} bit of the current group to 1, otherwise it will be set to 0.

A connection relationship between each group exists. The information contained in each group in DNN model is treated as sequence data in this paper. In addition, an unequal-length relationship exists between the original input information and the output target calculation node sequence, so the Seq2Seq model based on the attention mechanism is used in this paper.

In the task allocation process, the grouping results of the DNN model predicted by the task partition network are processed by the encoder to generate semantic vectors, which are sequentially transmitted to the decoder. Then the decoder allocates computing nodes according to the grouping order. In each step of the decoder's prediction, the attention mechanism is used to pass the number of the calculation node assigned in the previous step to the decoder and generate the calculation node assignment in the current step. Finally, we deploy the DNN model partition result and task allocation result in the actual computing node to get the overall task allocation plan.

4.3. Joint Training of Policy Gradient

The policy gradient is jointly trained with the task partition and allocation network in the prediction network which can obtain a task allocation plan that minimizes the total training time of a single training batch. The total time of training a batch of DNN models under task allocation plan, ρ, is defined as $t(\rho)$, including one forward calculation, one backward propagation, and one parameter update for which the unit of time is seconds. In order to avoid the variance of the running time measurement, for each predicted task allocation plan, we run the DNN model on the computing node according to the plan. Then we measure it several times to calculate the average value to get the reward function of the current plan. Finally, after several iterations, the task allocation plan with the shortest actual running time is obtained. In the experiment, each task allocation plan was run 20 times. And the average value of the last 10 running results was taken as the running time measurement value.

In the actual running process, sampling the running time as the reward function directly will cause poor training effect. On one hand, the task allocation scheme which was proposed in the early stage of training may lead to a longer running time with producing inappropriate learning signals for the convergence of the prediction network. On the other hand, as the prediction network converges, the predicted task allocation schemes are more similar, which will result in smaller differences in the corresponding execution time. Therefore, the reward function for a task allocation plan, ρ, is defined as:

$$R_\rho = -\sqrt{t(\rho)} \tag{1}$$

The prediction network should minimize the expected value of R_ρ to achieve a better training effect. In summary, the loss function [31], $J\left(\theta_g, \theta_d\right)$, is defined as:

$$\begin{aligned} J\Big(\theta_g, \theta_d\big) &= \mathrm{E}_{\mathrm{P}(\rho;\theta_g,\theta_d)}[R_\rho] \\ &= \sum_{g\sim\pi_g}\sum_{d\sim\pi_d} p(g;\theta_g)p(d|g;\theta_d)\,R_\rho \end{aligned} \tag{2}$$

where θ_g and θ_d, respectively, represent the parameters of the task partition network and the task allocation network; $p(g;\theta_g)$ represents the probability of the task partition network generating the grouping result, g; and $p(d|g;\theta_d)$ represents the probability of the task allocation network generating the task allocation result, d, according to the grouping result, g.

The task partition network independently predicts the partition of the DNN model. Then the task allocation network allocates computing nodes to each group according to the task partition. The changes in the grouping results make the predicted task allocation plan correspond to a large difference in running time. This paper introduces exponential moving average (EMA) to better reflect the actual execution time of the predicted allocation plan. Combining EMA to obtain the partial derivative of Equation (2), the gradients of parameters θ_g and θ_d [31] of the prediction network are as follows:

$$\begin{aligned} \nabla_{\theta_g} J(\theta_g, \theta_d) &= \sum_{g\sim\pi_g} \nabla_{\theta_g} \log p(g,\theta_g) \sum_{d\sim\pi_d} (R_\rho - Base) \\ &= \sum_{g\sim\pi_g} \nabla_{\theta_g} p(g;\theta_g) \sum_{d\sim\pi_d} p(d|g;\theta_d)\,(R_\rho - Base) \end{aligned} \tag{3}$$

$$\begin{aligned}\nabla_{\theta_d} J(\theta_g,\theta_d) &= \sum_{d\sim\pi_d}\sum_{g\sim\pi_g} p(g;\theta_g)\nabla_{\theta_d} p(d|g;\theta_d)(R_\rho - Base)\\ &= \sum_{d\sim\pi_d}\Big(\sum_{g\sim\pi_g}\nabla_{\theta_d}\log p(d|g;\theta_d)\Big)(R_\rho - Base)\end{aligned} \tag{4}$$

Among them, *Base* is the EMA that rewards R_ρ. The gradients of parameters, θ_g and θ_d, related to the prediction network can be calculated by Equations (3) and (4). Combined with the stochastic gradient ascent algorithm, detailed training steps are shown in Algorithm 1.

Algorithm 1 Task allocation in pipeline parallel training based on deep reinforcement learning

```
Input: DNN model, $G$; heterogeneous computing node set, D; number of iterations, n.
Output: Task allocation plan, $\rho$.
1: Initialize $\theta_g$, $\theta_d$
2: for step = 0 to n do
3: for layer in $G$
4: Assign layer to group
5: for group in $G$
6: Assign group to device
7: Assign DNN to devices, $D$, according to parallelism scheme, $\rho$
8: Sample runtime, $t(\rho)$
9: Update parameters $\theta_g$, $\theta_d$
10: end for
```

According to heterogeneous computing node set and the DNN model, Algorithm 1 trains the prediction model to obtain the optimal task allocation plan. The task partition network obtains the initial grouping; at the same time, it inputs the grouping information into the task allocation network. The model is deployed on the computing node to run several batches of data according to the distribution plan. The execution time is sampled and averaged to obtain the single-step execution time, and, the predicted network parameters are updated inversely using this as a reward until the model converges.

4.4. Pipeline Parallel Training of Task Allocation

Pipeline parallel training adopts an asynchronous parameter update architecture. Multiple computing nodes concurrently execute forward and backward transfer tasks of different training batches to ensure that the system achieves better throughput and hardware utilization efficiency.

Figure 3 shows a typical pipeline parallel training example. Assume that a DNN model with five layers is divided into four groups, the first three layers are in different groups and the last two layers are divided into the same group. Each packet corresponds to a stage in the pipeline system. Each stage executes the training tasks in a pipelined manner according to the grouping sequence. In stages 1, 2, and 3, multiple computing nodes are configured, improving the training throughput. The combination of data parallelism and pipeline parallelism can reduce the difference of calculation time to ensure that the pipeline has a higher throughput. The data communication in the pipeline system mainly exists at the boundary of each stage.

Pipeline parallel training is different from traditional stand-alone training, which involves two-way pipelines. Each training batch in the pipeline system starts the forward pass from the initial stage of the pipeline. After the forward pass is completed in the final stage, the backward pass is performed. Then the results are sequentially pushed to the upstream stage for calculation. Each batch may be located at a different computing node and perform forward or backward pass calculations. In a stable pipeline system, each computing node is performing any one of the following two tasks to ensure an orderly training: perform a forward pass calculation on a batch and push the calculation result to the next stage or perform a backward pass calculation on different batch and push the calculation result to the upstream stage.

In order to ensure that the pipeline system has a higher throughput, an asynchronous parameter update method is used to perform parameter updates asynchronously for different training batches. This paper is based on the asynchronous pipeline parallel architecture. At the beginning of the pipeline system, a specified number of batch training samples are injected into the system. Once the final stage of the pipeline completes the forward pass calculation task of the first batch, it will perform the backward pass in the same batch. Then it will start to perform the forward and backward pass calculation tasks of the subsequent batches in turn. When the backward pass task is propagated to the initial stage of the pipeline, the system starts to continuously inject new training batches, so that each stage starts to execute different batches of forward and backward pass calculation tasks. After the pipeline system reaches a stable state, each computing node alternately executes different batches of forward and backward delivery tasks, so that all computing nodes in the pipeline system participate in training to ensure system utilization and throughput.

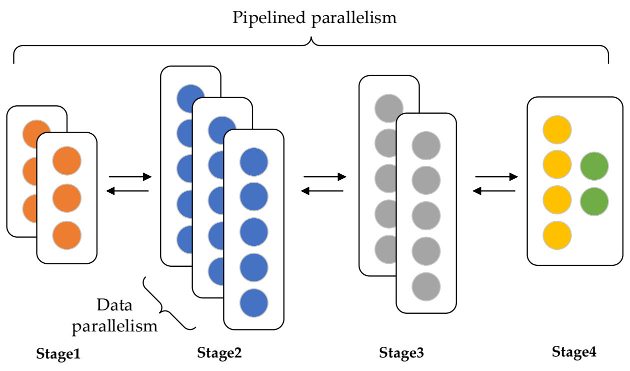


Figure 3. Example of pipeline parallel training.

5. Performance Evaluation

5.1. Experiment Setup

5.1.1. Data Set

The experiment used the CIFAR-10 [32] data set to train the DNN model. CIFAR-10 is a commonly used data set for universal object recognition in the field of deep learning. It contains 50,000 training pictures and 10,000 test pictures in total. There are 10 categories in this total. Each picture is a 32×32 color picture. The experiment trains three DNN models on the CIFAR-10 data set: AlexNet [33], VGG-16 [34], and Inception-V3 [35].

AlexNet uses an eight-layer neural network, including five convolutional layers and three fully connected layers (the maximum pooling layer is added after the three convolutional layers). The model size is approximately 240 MB. The batch size used in the experiment is set to 128 with the RMSprop optimizer [36] with the learning rate set to 0.01.

VGG-16 uses several consecutive 3×3 convolution kernels to replace the larger convolution kernel in AlexNet. The model size is about 550 MB. The batch size used in the experiment was set to 64 with the momentum SGD optimizer which the initial learning rate was set to 0.01 and the momentum was set to 0.9.

Inception-V3 consists of several blocks. Each block consists of several convolutional layers and pooling layers, where the layers can be executed in parallel, however the output is connected to form the input of the next block. Blocks are executed in sequence from block to block. The model size is approximately 157 MB. The batch size used in the experiment was set to 128 with the Adam optimizer being used, for which the initial learning rate was set to 0.01.

5.1.2. Benchmark

The pipelined parallel training task allocation method proposed in this paper was implemented using deep reinforcement learning, which aims to solve the optimal task allocation plan and accelerate the neural network model training process. A single CPU node and the data parallel training method were selected as the experimental benchmark with using the synchronous parameter update method BSP. The pipeline parallel training method Pipe-torch based on heterogeneous network was selected as a control to evaluate the acceleration effect of the task allocation scheme.

5.1.3. Metric

Based on the above data sets and methods to carry out the experiment, the evaluation criteria used in the experiment are considered from the following perspectives:

(1) Single-step training time

The single-step training time is used to evaluate the acceleration effect of pipeline parallel training. The single-step training time is the time that it takes to complete a batch of forward pass, backward pass, and parameter update. In order to avoid the error of the actual environment operation, the experiment selects 100 training batch running time results. After that, the average value is taken as the single-step running time of the current allocation plan. The shorter the training time used, the better the acceleration effect of the task distribution plan.

(2) Proportion of communication time

The ratio of communication time to total training time is used to evaluate the optimization effect of pipeline parallel training communication. The smaller the proportion of communication time used, the better the effect the pipeline parallel training overlapped computing and communication tasks achieved—this will speed up the. Suppose that the DNN model layer is divided into s groups, corresponding to s stages in the pipeline. The optimization objective of task allocation method can be expressed as follows:

$$T = \min \sum_{s=1}^{S} (A_s + C_s) \tag{5}$$

where A_s is the total calculation time in stage s, and C_s is the communication time between stage s and stage s + 1. The goal of task allocation method is to find the optimal pipelined parallel task allocation to minimize T.

(3) Training accuracy

The model training accuracy is used to evaluate the effectiveness of the pipeline parallel training method. The experiment performed 100 epoch training for the model and recorded model training accuracy and time changes. The higher the accuracy, the better the effect achieved by the pipeline parallel training.

5.2. Experimental Results and Analysis

5.2.1. Pipeline Parallel Training

(1) Single-step training time

Four different training methods were designed in the experiment. In this way, the single-step training time in each model training was calculated, for which the time unit was the second. Taking the data-parallel BSP as the baseline, we studied the acceleration effect of TAPP single-step training. The experimental results are shown in Table 1.

Table 1. Experimental results of pipeline single-step training time (seconds).

	SingleCPU	**Pipe-Torch**	**BSP**	**TAPP**	**Speedup (BSP/TAPP)**
AlexNet	0.811	0.675	0.803	0.602	1.33×
VGG-16	1.972	1.634	1.865	1.432	1.30×
Inception-V3	0.985	0.632	0.914	0.614	1.49×

It can be seen from Table 1 that the training speed of TAPP in the three models was better than that of BSP, which was increased by 1.33 times, 1.30 times, and 1.49 times, respectively, with an average decrease of 1.37 times. In Table 1, single CPU is the time required to complete the training task with one CPU. Speedup is a comparison between TAPP method and data-parallel BSP method. BSP is affected by slow nodes and network bandwidth, so the improvement effect was the worst. In the VGG-16 model training, due to the large amount of parameter communication, the BSP acceleration effect is not obvious. According to the network bandwidth, Pipe-torch sorts the computing nodes and allocates them in order, which can effectively reduce the communication delay. The effect was most obvious in the training of the Inception-V3 model, which was 1.44 times faster. Pipe-torch did not consider the heterogeneity of computing power. However, due to the relatively high network bandwidth and strong computing power between the two GPU nodes in the experiment, Pipe-torch put the two GPUs in the early stage of the pipeline training system, which can be effective. The communication time consumption of parallel training of the model was reduced. Because of it, a higher acceleration effect was achieved. TAPP comprehensively considers the heterogeneity of node computing capabilities and network transmission capabilities, so it uses predictive networks to find the optimal task allocation method, improves training speed, and achieves the best effect.

(2) Proportion of communication time

The experiment counted the ratio of communication time to total training time in the three training methods. The experimental results are shown in Figure 4.

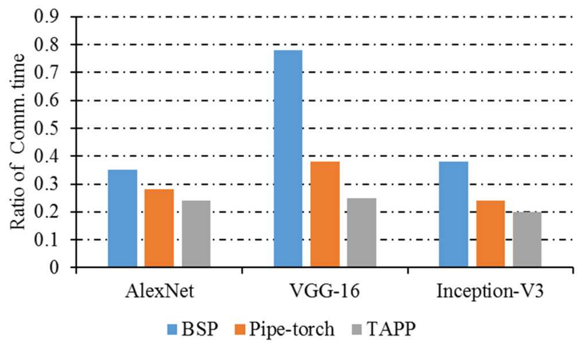
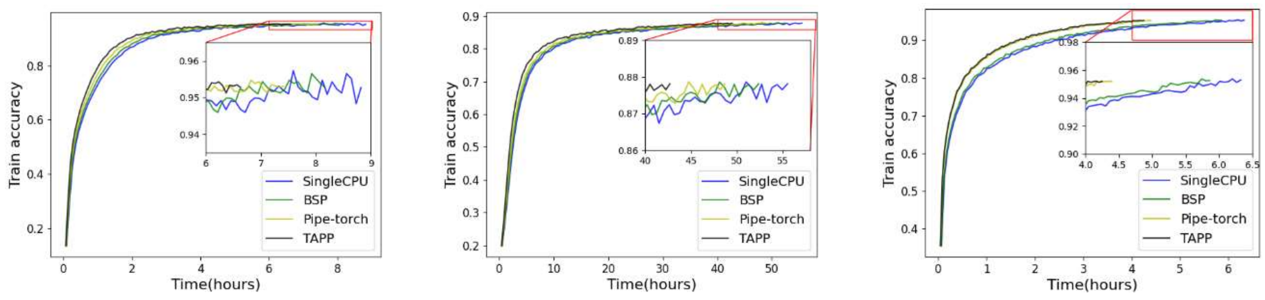


Figure 4. Experimental results of the proportion of communication time.

It can be seen that in different parallel training methods of the three models, the communication time and the total training time had the same general trend with BSP taking the highest proportion, Pipe-torch second, and TAPP the lowest. In terms of the effect of reducing communication time, TAPP was significantly better than other methods. Compared with BSP, the proportion of communication time in TAPP training of the three models decreased by 31.43%, 67.95%, and 47.37%, respectively, with an average decrease of 48.92%. The VGG-16 model has huge parameters, so the parallel data requires the most communication data. Pipeline parallel training can effectively overlap computing and communication tasks, so the communication time can be significantly reduced. Inception-V3 has a secondary effect of reduced communication time because the scale of intermediate communication in the parallel model is larger than the size of the parameter activation value. Compared with the other two models, the parameters of the AlexNet model are small and uncomplicated, but the proportion of communication time is also 31.43% lower than that of the BSP. Pipe-torch can effectively reduce the amount of data communication by considering the different bandwidths between computing nodes when assigning tasks. TAPP comprehensively considers the heterogeneity of node computing capabilities and network bandwidth. It uses pipelines to parallel training, overlapping communication and computing tasks, which can reduce the communication time obviously.

(3) Training accuracy

The experiment counted the accuracy and training time change about the three models under different training methods for 100 epoch training for which the time unit was the hour. The experimental results are shown in Figure 5.



(**a**) AlexNet (**b**) VGG-16 (**c**) Inception-V3

Figure 5. Experimental results of training accuracy and time.

It can be seen from Figure 5 that the model acceleration effect ranked TAPP as the fastest, Pipe-torch second, and BSP the slowest. TAPP had the shortest training time among the three model trainings. The total training time was shortened by about 1.5 h, 10 h, and 1.6 h, respectively, and training accuracy consistent with BSP was achieved. BSP is affected by slow nodes and network bandwidth, so the acceleration effect was not obvious. TAPP and Pipe-torch use pipeline parallel training, which overlaps computing and communication tasks well, to achieve significant acceleration. The acceleration effect was the best in the VGG-16 model training. VGG-16 has many parameters, strong model fitting ability, and long training time. It took about 52 h for BSP to train the VGG-16 model for 100 epochs.

TAPP can predict the optimal task allocation plan, which can alleviate expensive data communication and shorten the training time of the VGG-16 model by 1.24 times, achieving the best training acceleration effect. The experimental results prove that TAPP can effectively accelerate the model training speed and achieve model training accuracy consistent with data-parallel BSP.

5.2.2. TAPP Algorithm Evaluation

(1) Experiment with different computing power

The experiment used two different computing nodes to perform the prediction network training process to calculate the time consumption of prediction network training. SingleCPU used the No. 1 node CPU and SingleGPU used the No. 2 node GPU, for which the time unit is the hour. The experimental results are shown in Table 2.

Table 2. Predicted network training convergence time (hours).

	AlexNet	**VGG-16**	**Inception-V3**
SingleCPU	0.5	1.6	0.8
SingleGPU	0.3	0.6	0.5

It can be seen from Table 2 that, due to the powerful computing performance of GPU, the convergence speed of the SingleGPU method model was generally faster than that of SingleCPU. For the VGG-16 model, the GPU accelerated model had the most obvious convergence effect, with the training time shortened by 2.17 times. The AlexNet and Inception-V3 models are smaller than VGG-16, so the training was faster. However, the training time was not significantly improved, being 1.67 times and 1.6 times, respectively. Figure 6 shows the training-loss curve of the pipeline parallel task allocation scheme generated by the prediction network for the VGG-16 model. The SingleGPU method can complete the convergence in about 40 min. Therefore, it can be concluded that the use of GPU nodes can effectively improve the training speed of the prediction network and reduce the time overhead.

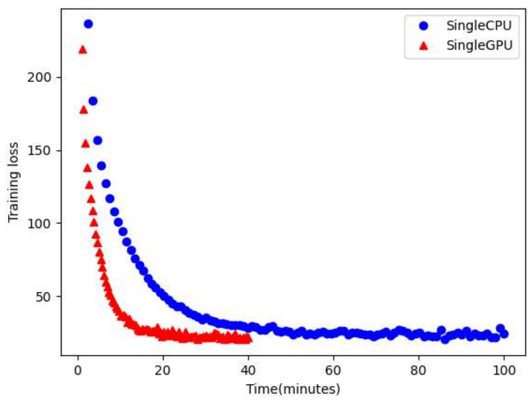


Figure 6. The impact of different computing power: the training-loss curve of the pipeline parallel task allocation scheme generated by the prediction network for the VGG-16 model.

(2) Experiment with different batch sizes

The experiment put the No. 1 node CPU into train the AlexNet model, as an example, to explore the convergence of the pipeline parallel task allocation scheme generated by the network under different batch size conditions. The experimental results are shown in Figure 7. It can be seen from the Figure 7 that the batch size affects the convergence speed of the prediction network with the smaller batch size having the faster convergence speed. The prediction network was trained based on the word vector constructed by the runtime parameters of the DNN model as input. The batch size affects the parameter amount, calculation time, activation value, and other information of each layer of the model to be trained, thereby affecting the training complexity of the prediction network.

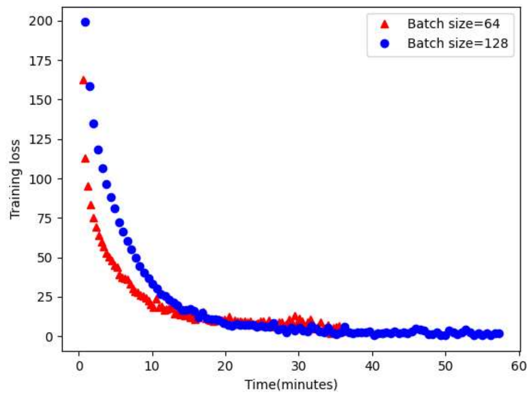


Figure 7. The impact of different batch sizes: the batch size affects the convergence speed of the prediction network with the smaller batch size having the faster convergence speed.

It can be seen from the experimental results that the prediction network had different convergence times due to different batch sizes, but the overall convergence time was within 1 h. Combined with the analysis of the pipeline parallel training experiment, it can be shown that the sum of the predicted network training time and the pipeline parallel training time is always less than the BSP running time, which proves the effectiveness of the task allocation scheme based on deep reinforcement learning proposed in this paper.

5.2.3. Experimental Results

TAPP uses deep reinforcement learning to complete the prediction of the pipeline parallel training task allocation method, which effectively avoids the imbalance load problems caused by the difference of node computing power and network bandwidth in improving the acceleration of model training. The pipeline parallel training experiment results showed that the single-step training time of TAPP in the three models is better than that of data parallel, which was increased by 1.33 times, 1.30 times, and 1.49 times, respectively. The training time of TAPP was the shortest among the three models, which was shortened by 1.5 h, 10 h, and 1.6 h, respectively. The training accuracy of TAPP was consistent with the data-parallel BSP. The evaluation experiment of the TAPP algorithm explored the influence of different computing powers and different batch sizes on the time required to train the prediction network.

Experimental results showed that high-performance computing nodes can effectively improve the prediction network convergence time, and a smaller batch size requires a shorter convergence time. In the experiment, the sum of the predicted network convergence time and the pipeline system parallel training time was less than the time required for data parallelism, obviously in the VGG-16 model. This experiment proves that the TAPP method achieves a good training acceleration effect in DNN model training, and the greater the number of the model parameter, the more obvious the lifting effect achieved.

6. Conclusions

In this paper, we propose a task allocation method, TAPP. According to a DNN model and the heterogeneous computing resources, the predictive network has been used to continuously train to generate the optimal pipeline parallel task allocation plan. TAPP divides the DNN model into pipeline tasks by constructing task allocation prediction network. It uses the Seq2seq network based on attention mechanism to allocate computing nodes for each task. TAPP uses the policy gradient to train the prediction network and obtains the pipeline parallel task allocation scheme within the shortest training time. Compared with different parallel training methods on DNN models, the experimental results show that TAPP can comprehensively consider the heterogeneity of hardware node computing capacity and network transmission capacity. At the same time, it uses the predictive network to solve the optimal task allocation method, which had the best improvement effect. Furthermore, compared with the BSP method, without affecting accuracy, the TAPP model single-step training time was reduced by an average of 1.37 times, and the proportion of communication time was reduced by an average of 48.92%.

Author Contributions: Conceptualization, Y.M., Z.T., F.X. and Q.W.; methodology, Y.M., Z.T. and Q.W.; validation, Z.T. and Q.W.; formal analysis, Y.M. and S.X.; investigation, Z.T., F.X. and Q.W.; writing—original draft preparation, Z.T. and Q.W.; writing—review and editing, Z.T.; supervision, Y.M.; project administration, Y.M. and S.X.; funding acquisition, Y.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Technology Research and Development Program of the Ministry of Science and Technology of China, grant number 2018YFC0407105; the National Natural Science Foundation of China, grant number 61832005; the Key Technology Project of China Huaneng Group, grant number HNKJ19-H12.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
2. Bakkali, S.; Ming, Z.; Coustaty, M.; Rusinol, M. Cross-Modal Deep Networks for Document Image Classification. In Proceedings of the IEEE International Conference on Image Processing, Abu Dhabi, United Arab Emirates, 25–28 October 2020; pp. 2556–2560.
3. Xie, X.; Zhou, Y.; Kung, S.Y. Exploring Highly Efficient Compact Neural Networks for Image Classification. In Proceedings of the IEEE International Conference on Image Processing, Abu Dhabi, United Arab Emirates, 25–28 October 2020; pp. 2930–2934.
4. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to Sequence Learning with Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 3104–3112.

5. Huang, Z.; Ng, T.; Liu, L.; Mason, H.; Zhuang, X.; Liu, D. SNDCNN: Self-Normalizing Deep CNNs with Scaled Exponential Linear Units for Speech Recognition. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Barcelona, Spain, 4–8 May 2020; pp. 6854–6858.
6. Ali, A.; Chowdhury, S.A.; Afify, M.; EI-Hajj, W.; Hajj, H.M.; Abbas, M.; Hussein, A.; Ghneim, N.; Abushariah, M.; Alqudah, A. Connecting Arabs: Bridging the gap in dialectal speech recognition. *Commun. ACM* **2020**, *64*, 124–129. [CrossRef]
7. Kim, J.; El-Khamy, M.; Lee, J. Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition. In Proceedings of the Annual Conference of the International Speech Communication Association, Stockholm, Sweden, 20–24 August 2017; pp. 1591–1595.
8. Arivazhagan, N.; Cherry, C.; Te, I.; Macherey, W.; Baljekar, P.; Foster, G.F. Re-Translation Strategies for Long Form, Simultaneous, Spoken Language Translation. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Barcelona, Spain, 4–8 May 2020; pp. 7919–7923.
9. Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, D.; Chen, M.X.; Lee, H.; Ngiam, J.; Le, Q.V.; Wu, Y.; et al. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 103–112.
10. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Regularized Evolution for Image Classifier Architecture Search. In Proceedings of the Association for the Advancement of Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 4780–4789.
11. Wang, M.; Huang, C.; Li, J. Supporting Very Large Models using Automatic Dataflow Graph Partitioning. In Proceedings of the European Conference on Computer Systems, Dresden, Germany, 25–28 March 2019; pp. 1–26.
12. Jin, T.; Hong, S. Split-CNN: Splitting Window-based Operations in Convolutional Neural Networks for Memory System Optimization. In Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems, Providence, RI, USA, 13–17 April 2019; pp. 835–847.
13. Park, J.H.; Yun, G.; Chang, M.Y.; Nguyen, N.T.; Lee, S.; Choi, J.; Noh, S.H.; Choi, Y. HetPipe: Enabling Large DNN Training on (whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism. In Proceedings of the USENIX Annual Technical Conference, Virtual, 15–17 July 2020; pp. 307–321.
14. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. In Proceedings of the Advances in Neural Information Processing Systems, Virtual, 6–12 December 2020.
15. Li, Y.; Yu, M.; Li, S.; Avestimehr, S.; Kim, N.S.; Schwing, A.G. Pipe-SGD: A Decentralized Pipelined SGD Framework for Distributed Deep Net Training. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 8056–8067.
16. Li, Y.; Zeng, Z.; Li, J.; Yan, B.; Zhao, Y.; Zhang, J. Distributed Model Training Based on Data Parallelism in Edge Computing-Enabled Elastic Optical Networks. *IEEE Commun. Lett.* **2021**, *25*, 1241–1244. [CrossRef]
17. Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Le, Q.V.; Mao, M.Z.; Ranzato, M.; Senior, A.W.; Tucker, P.A.; et al. Large Scale Distributed Deep Networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1223–1231.
18. Du, J.; Zhu, X.; Shen, M.; Du, Y.; Lu, Y.; Xiao, N.; Liao, X. Model Parallelism Optimization for Distributed Inference Via Decoupled CNN Structure. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 1665–1676.
19. Zinkevich, M.; Weimer, M.; Smola, A.J.; Li, L. Parallelized Stochastic Gradient Descent. In Proceedings of the Advances in Neural Information Processing Aystems, Vancouver, BC, Canada, 6–9 December 2010; pp. 2595–2603.
20. Zhang, S.; Choromanska, A.E.; LeCun, Y. Deep Learning with Elastic Averaging SGD. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 685–693.
21. Agarwal, A.; Duchi, J.C. Distributed Delayed Stochastic Optimization. In Proceedings of the 51st IEEE Conference on Decision and Control, Maui, HI, USA, December 10–13 2012; pp. 5451–5452.
22. Harlap, A.; Narayanan, D.; Phanishayee, A.; Seshadri, V.; Devanur, N.R.; Ganger, G.R.; Gibbons, P.B.; Zaharia, M. PipeDream: Generalized Pipeline Parallelism for DNN Training. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, Huntsville, ON, Canada, 27–30 October 2019; pp. 1–15.
23. Isard, M.; Prabhakaran, V.; Currey, J.; Wieder, U.; Talwar, K.; Goldberg, A.V. Quincy: Fair Scheduling for Distributed Computing Clusters. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky, MT, USA, 11–14 October 2009; pp. 261–276.
24. Gog, I.; Schwarzkopf, M.; Gleave, A.; Watson, R.N.M.; Hand, S. Firmament: Fast, Centralized Cluster Scheduling at Scale. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, 2–4 November 2016; pp. 99–115.
25. Jia, Z.; Lin, S.; Qi, C.R.; Aiken, A. Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks. In Proceedings of the 35th International Conference on Machine Learning, Stockholmsmässan, Stockholm, Sweden, 10–15 July 2018; pp. 2279–2288.
26. Jia, Z.; Zaharia, M.; Aiken, A. Beyond Data and Model Parallelism for Deep Neural Networks. In Proceedings of the Machine Learning and Systems, Stanford, CA, USA, 31 March–2 April 2019.

27. Zhan, J.; Zhang, J. Pipe-torch: Pipeline-Based Distributed Deep Learning in a GPU Cluster with Heterogeneous Networking. In Proceedings of the Seventh International Conference on Advanced Cloud and Big Data, Suzhou, China, 21–22 September 2019; pp. 55–60.
28. Fan, S.; Rong, Y.; Meng, C.; Cao, Z.; Wang, S.; Zheng, Z.; Wu, C.; Long, G.; Yang, J.; Xia, L.; et al. DAPPLE: A Pipelined Data Parallel Approach for Training Large Models. In Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Virtual, 27 February 27–3 March 3 2021; pp. 431–445.
29. Mirhoseini, A.; Pham, H.; Le, Q.V.; Steiner, B.; Larsen, R.; Zhou, Y.; Kumar, N.; Norouzi, M.; Bengio, S.; Dean, J. Device Placement Optimization with Reinforcement Learning. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; pp. 2430–2439.
30. Bahdanau, D.; Cho, K.; Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
31. Liu, W.; Peng, L.; Cao, J.; Fu, X.; Liu, Y.; Pan, Z.; Yang, J. Ensemble Bootstrapped Deep Deterministic Policy Gradient for Vision-Based Robotic Grasping. *IEEE Access* **2021**, *9*, 19916–19925. [CrossRef]
32. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. In *Handbook of Systemic Autoimmune Diseases*; Elsevier: Amsterdam, The Netherlands, 2009; Volume 1.
33. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
34. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015; pp. 1–14.
35. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 448–456.
36. Tieleman, T.; Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.* **2012**, *4*, 26–31.