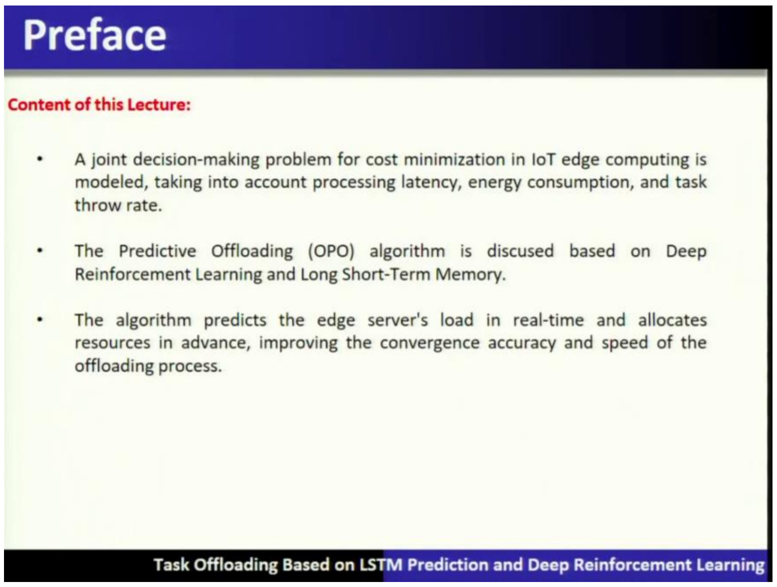
Foundation of Cloud IoT Edge ML
Professor Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology Patna
Lecture 13

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

I am Doctor Rajiv Mishra from IIT, Patna; topic of today's lecture is Task offloading Based on LSTM Predictions and Deep Reinforcement Learning.

(Refer Slide Time: 00:28)



Preface

Content of this Lecture:

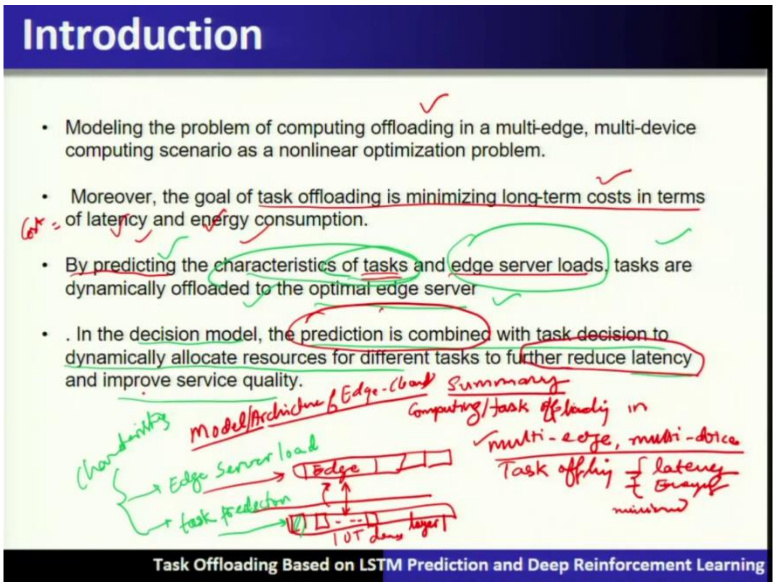
- A joint decision-making problem for cost minimization in IoT edge computing is modeled, taking into account processing latency, energy consumption, and task throw rate.
- The Predictive Offloading (OPO) algorithm is discused based on Deep Reinforcement Learning and Long Short-Term Memory.
- The algorithm predicts the edge server's load in real-time and allocates resources in advance, improving the convergence accuracy and speed of the offloading process.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Content of this lecture, this lecture, we are going to cover the joint decision-making problem for cost optimization that is cost minimization in IoT edge computing is modelled and is taking into account various processing the latency, energy consumption and task throw rate. Using this, we will discuss a predictive offloading algorithm, which is based on the deep reinforcement learning and long short-term memory.

This particular algorithm that is predictive offloading algorithm predicts the edge servers load in real time and allocate the resources in advance, thereby improving the accuracy and that is the convergence accuracy and the speed of offloading process.

(Refer Slide Time: 01:26)



Introduction

- Modeling the problem of computing offloading in a multi-edge, multi-device computing scenario as a nonlinear optimization problem.
- Moreover, the goal of task offloading is minimizing long-term costs in terms of latency and energy consumption.
- By predicting the characteristics of tasks and edge server loads, tasks are dynamically offloaded to the optimal edge server
- . In the decision model, the prediction is combined with task decision to dynamically allocate resources for different tasks to further reduce latency and improve service quality.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, let us get started with the introduction of this particular problem setting. Now, modelling the problem of computing offloading in a multi edge multi device computing scenario often is a nonlinear optimization problem. Moreover, the goal of task offloading is minimizing the long-term cost in terms of latency and energy consumption, by predicting the characteristics of the tasks and its server loads tasks are dynamically offloaded to an optimal edge server.

So, in the decision model, the prediction is combined with task decision to dynamically allocate resources for different tasks to further reduce the latency and improve the service quality. In summary, we need to say that we are considering the computing or the task offloading in multi edge and multi device scenario.

So, meaning to say if you recall the architecture or the model architecture of the edge cloud, what you will find is that there are multiple devices at the lowest level or let us say the layer number one these are all IoT devices. Now, the task of these IoT devices because of constraints of battery or the energy and also the latency, these tasks are offloaded to the edge. Now, the edge is a multi-edge maybe that it is not only single node, but multiple nodes out there.

So, multi devices and multi edge task offloading problem is around the goal of minimizing the long term costs in terms of latency and energy. So, the cost function is a is a function of minimizing the latency and energy consumption jointly. So, this is the aim of computing or a task offloading from IoT device to the edge. So, the edge we are considering here as multi edge.

Now, what is happening is that this particular task offloading is incurring latency of transferring the data from IoT to the edge for computation. So, it requires the network latency that is in the terms of network latency is incurred. And also, it requires some amount of energy. So, energy is also required here in the task offloading.

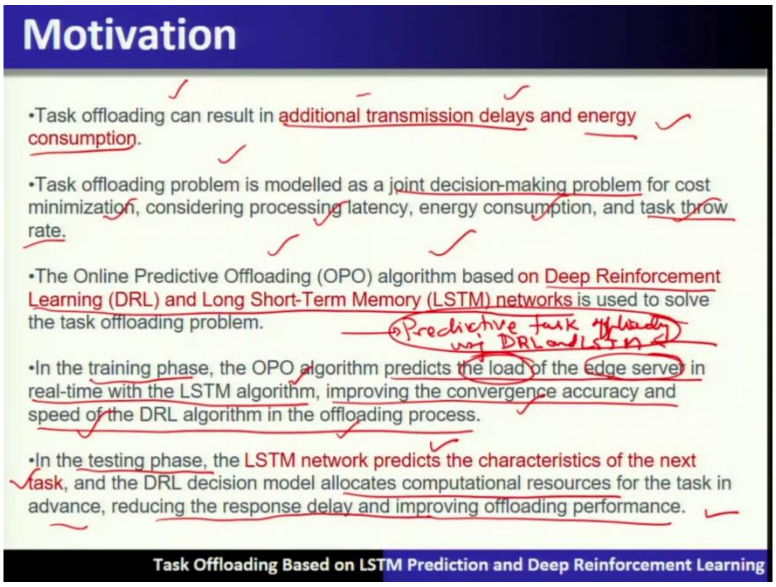
So, how to minimize these two things, that is the overhead of task offloading is to be minimized. So, this is done with the help of the predictions. So, by predicting the characteristics of the task and the edge server load, why because, so, this is an edge server. So, this is an edge server load which is occurring at the at the edge and also the IoT will now generate the task.

So, there are two characteristics that is the task the characteristics of the task which are being generated out of these IoT devices and also the server loads the edge server load there are two aspects and these characteristics needs to be used in the predictions. So, once these characteristics of the task, incoming task at the user devices and the edge server load, these tasks are dynamically offloaded to the optimal edge server.

If these two things are being understood, then using the way of prediction they can be offloaded to the optimal edge server. So, therefore, the in the decision model, the prediction is combined with a task decision to dynamically allocate the resources for different tasks to further reduce the latency and improve the service quality, meaning to say that this particular task allocation or task offloading if it is combined with the predictions.

And then in that case, it will further there is a scope of further reducing the latency and improve the service quality, how that is all done, we are going to discuss this aspect in this particular issue.

(Refer Slide Time: 07:06)



Motivation

- Task offloading can result in additional transmission delays and energy consumption.
- Task offloading problem is modelled as a joint decision-making problem for cost minimization, considering processing latency, energy consumption, and task throw rate.
- The Online Predictive Offloading (OPO) algorithm based on Deep Reinforcement Learning (DRL) and Long Short-Term Memory (LSTM) networks is used to solve the task offloading problem.

→ Predictive task offloading using DRL and LSTM

- In the training phase, the OPO algorithm predicts the load of the edge server in real-time with the LSTM algorithm, improving the convergence accuracy and speed of the DRL algorithm in the offloading process.
- In the testing phase, the LSTM network predicts the characteristics of the next task, and the DRL decision model allocates computational resources for the task in advance, reducing the response delay and improving offloading performance.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, therefore, let us understand through the motivation that the task offloading can result in the additional transmission delays and the energy consumption that I have already stated in the previous slide. Therefore, this aspect into the task offloading which is incurring the additional transmission delays and energy consumption, how this is we overcome reduced to a minimum value.

So, therefore, task offloading problem is modelled as joint decision making problem for cost minimization considering the processing latency and energy consumption and task throw rate. tasks throw rate means, we will understand here this particular term with the help of that prediction models or the constraints based on this particular task or are the computation constraints if it is not met then those particular kinds of offloading will be of wastage that is called task throw rate.

So, the online predictive offloading algorithm often is based on deep reinforcement learning and long short-term memory that is LSTM. So, both of them is used to solve this kind of predictive task offloading problem. So, we are now formulating this task offloading as a predictive task of flooding with using deep reinforcement learning DRL and LSTM.

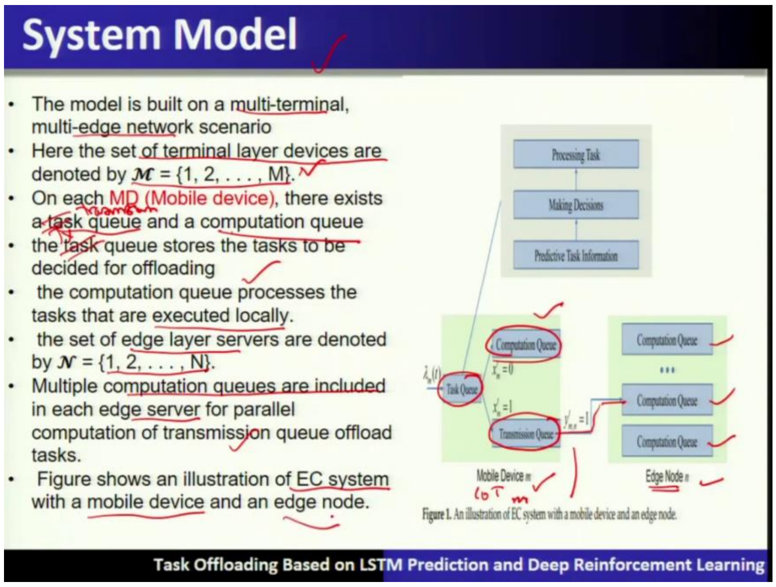
Now, such a model which is from the deep learning and the machine learning, it has a training phase of this particular predictive task offloading algorithm. In the training phase, the algorithm predicts the load of the edge server in the real time using LSTM algorithm and improve the convergence accuracy and the speed of deep reinforcement learning in the offloading process.

So, therefore, based on the characteristics of the edge server and the characteristics of the load happening, these two things are being used with the LSTM for making the predictions of edge server load and the incoming load in the real time, and using this that deep reinforcement learning based offloading will be now seen as improvement in terms of convergence accuracy and the speed of offloading process decision making.

Whereas, in the testing phase, the LSTM network will predict the characteristics of the next task and the DRL decision making Decision model allocates the computational resources for the task in advance. So, if it is done in advance, then it will reduce the response delay and improving the offloading process.

Therefore, we are going to see that this additional transmission delay and energy consumption is quite reduced with the help of LSTM base predictions, that is the prediction for the edge server loads in during the training phase and in the testing phase, it will be LSTM network will predict the characteristics of the next task.

(Refer Slide Time: 10:35)



System Model

- The model is built on a multi-terminal, multi-edge network scenario
- Here the set of terminal layer devices are denoted by $\mathcal{M} = \{1, 2, . . . , M\}$.
- On each MD (Mobile device), there exists a task queue and a computation queue
- the task queue stores the tasks to be decided for offloading
- the computation queue processes the tasks that are executed locally.
- the set of edge layer servers are denoted by $\mathcal{N} = \{1, 2, . . . , N\}$.
- Multiple computation queues are included in each edge server for parallel computation of transmission queue offload tasks.
- Figure shows an illustration of EC system with a mobile device and an edge node.

Figure 1. An illustration of EC system with a mobile device and an edge node.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us understand the system model. So, the model is built on multi terminal that is multiple IoT devices and multi edge network scenario. Now, here let us understand that here we have to now consider that the terminal layers devices are denoted by different values 1, 2, 3 and so, on up to M. So, that is shown over here.

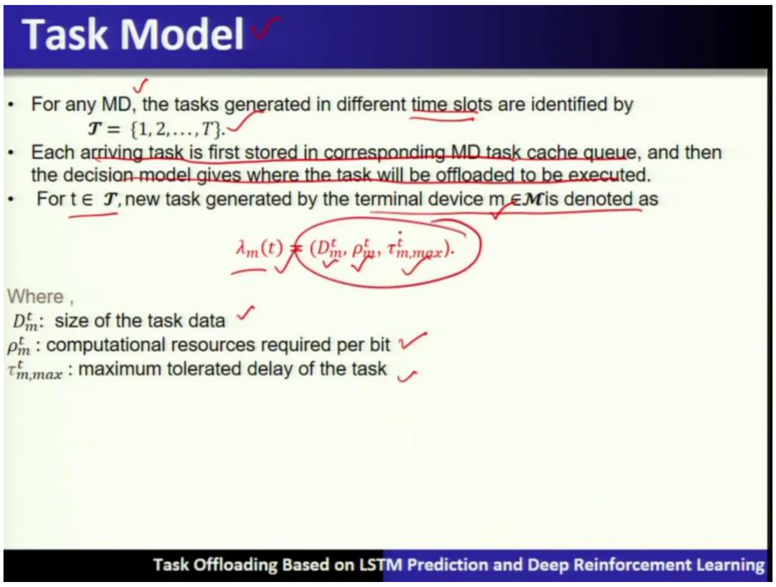
So, we see that these terminal devices are nothing but let us see the mobile devices or IoT devices which are m in number. So, m different devices are denoted by a set a set of terminal devices. Now, the element of this particular set is either an IoT device or a mobile device, let us assume that it is a mobile device.

So, on each mobile device, there exists a task queue and a computation queue which is shown in this particular figure. So, on a particular mobile device, there exists a transmission queue, there is a task queue and a computation queue. So, the task queue stores the transmission queue and the computation queue, transmission queue. So, the transmission queue stores transmission queue stores the task to be offloaded.

So, here those tasks which require to be offloaded on the edge will be stored over here, the computation queue processes that task locally that is on the mobile device now, we have to now consider the set of edge layer servers which are denoted by capital N. So, that is n different edge server nodes are there. So, there will be a mapping from the task transmission queue to the edge nodes.

Now, the multiple computation queues are included in each now, every edge node will have the multiple computation queue in each edge server for parallel computation of the transmission queue offloading task. So, here we can see this particular edge cloud system with a mobile device and an edge node is shown over here.

(Refer Slide Time: 12:53)



Task Model

- For any MD, the tasks generated in different time slots are identified by $\mathcal{T} = \{1, 2, \ldots, T\}$.
- Each arriving task is first stored in corresponding MD task cache queue, and then the decision model gives where the task will be offloaded to be executed.
- For t ∈ $\mathcal{T}$, new task generated by the terminal device $m \in \mathcal{M}$ is denoted as

$$\lambda_m(t) = (D_m^t, \rho_m^t, \tau_{m,max}^t).$$

Where ,

D_m^t: size of the task data

ρ_m^t : computational resources required per bit

$\tau_{m,max}^t$: maximum tolerated delay of the task

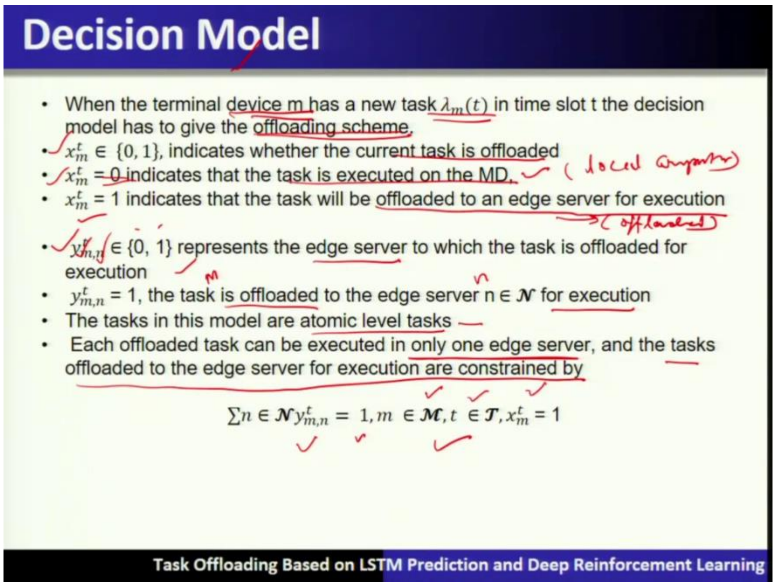
Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, let us see about the task model. So, that means for any mobile device, the tasks are generated in different time slots and are identified by a particular time set that is 1, 2, 3 and so, on up to T. Now, each arriving task is first stored in the corresponding mobile device task cache queue and then the decision model gives where the task will be offloaded to be executed.

Now, for a particular time slot T out of the total time the new task is generated by the terminal device and let us say that for that is m is denoted as this. So, the task which is generated by the mobile device is defined as the task model which has the following three components.

The first component is the size of the task data, and the second component is called the computational resource required per bit and the third one is the maximum tolerated delay of the task. So, that particular task model has three different characteristics and all these characteristics is defined under the task model.

(Refer Slide Time: 14:05)



Decision Model

- When the terminal device m has a new task $\lambda_m(t)$ in time slot t the decision model has to give the offloading scheme.
- $x_m^t \in \{0,1\}$, indicates whether the current task is offloaded
- $x_m^t = 0$ indicates that the task is executed on the MD. (local computation)
- $x_m^t = 1$ indicates that the task will be offloaded to an edge server for execution $\rightarrow$ (offloaded)
- $y_{m,n}^t \in \{0, 1\}$ represents the edge server to which the task is offloaded for execution
- $y_{m,n}^t = 1$, the task is offloaded to the edge server $n \in \mathcal{N}$ for execution
- The tasks in this model are atomic level tasks
- Each offloaded task can be executed in only one edge server, and the tasks offloaded to the edge server for execution are constrained by

$$\sum n \in \mathcal{N} y_{m,n}^t = 1, m \in \mathcal{M}, t \in \mathcal{T}, x_m^t = 1$$

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

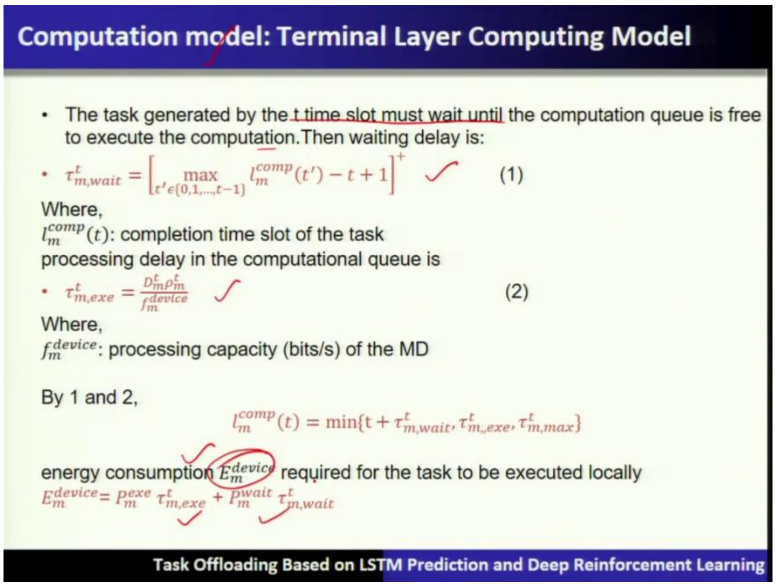
Now, let us see about the decision model. Now, when the terminal device m has a new task that we have defined in the previous slide in a time slot, that decision model has to give the offloading scheme that is xt of m is either from 0 or 1. So, that is indicates whether the current task is offloaded or not.

So, if xt of m is equal to 0 this indicates that the task is executed on the mobile device that is a local computation. Whereas, if xt of m is 1 that indicate that task will be offloaded to an edge server. Whereas, yt of mn which is also from 0 and 1 represents the edge server to which the task is offloaded for the execution.

So, once it is decided the set of tasks which are to be offloaded to the edge server, the now the mapping of these tasks to the server is to be done in this case. So, therefore, this yt mn represents the edge server to which that task is being offloaded. So, m means mn task and n means that nth edge server where this particular thing will be offloaded. So, if yt mn is equal to 1 that is the task m is offloaded to the edge server n for the execution.

So, the tasks in this model or atomic level tasks that means they cannot be they are indivisible. Now, each offloaded task can be executed in only one edge server and the tasks offloaded to the edge server for the executions are often constrained by this particular equation, you will see that the summation of all n is from this particular set, if it is 1 and m is from that is mobile from the mobile devices set nt is from time slot and xt of m is equal to 1 that is it is decided that the task is to be offloaded.

(Refer Slide Time: 16:14)



Computation model: Terminal Layer Computing Model

- The task generated by the t time slot must wait until the computation queue is free to execute the computation.Then waiting delay is:
- $\tau_{m,wait}^{t} = \left[\max_{t' \epsilon \{0,1,\dots,t-1\}} l_m^{comp}(t') - t + 1\right]^{+}$ ✓ (1)

Where,
$l_m^{comp}(t)$: completion time slot of the task
processing delay in the computational queue is

- $\tau_{m,exe}^{t} = \frac{D_m^t \rho_m^t}{f_m^{device}}$ ✓ (2)

Where,
f_m^{device}: processing capacity (bits/s) of the MD

By 1 and 2,

$$l_m^{comp}(t) = \min\{t + \tau_{m,wait}^{t}, \tau_{m,,exe}^{t}, \tau_{m,max}^{t}\}$$

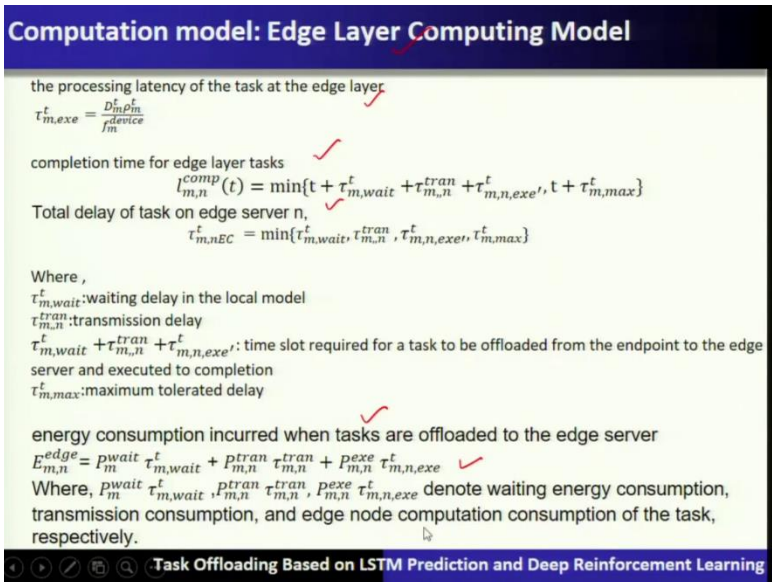
energy consumption E_m^{device} required for the task to be executed locally
$E_m^{device} = P_m^{exe}\,\tau_{m,exe}^{t} + P_m^{wait}\,\tau_{m,wait}^{t}$

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now computation model that is called the terminal layer computing model. So, the task generated by the t time slot must wait until the computation queue is free to execute the computation therefore, the waiting delays are calculated in this particular manner. So, the processing delay in the computation delay is also being calculated.

And if you calculate the total energy consumption of a particular device, which is required for a task to be executed locally is mentioned here in this particular equation.

(Refer Slide Time: 16:48)



Computation model: Edge Layer Computing Model

the processing latency of the task at the edge layer

$\tau^{t}_{m,exe} = \frac{D^{t}_{m}\rho^{t}_{m}}{f^{device}_{m}}$

completion time for edge layer tasks

$$l^{comp}_{m,n}(t) = \min\{t + \tau^{t}_{m,wait} + \tau^{tran}_{m,,n} + \tau^{t}_{m,n,exe'}, t + \tau^{t}_{m,max}\}$$

Total delay of task on edge server n,

$$\tau^{t}_{m,nEC} = \min\{\tau^{t}_{m,wait}, \tau^{tran}_{m,,n}, \tau^{t}_{m,n,exe'}, \tau^{t}_{m,max}\}$$

Where ,

$\tau^{t}_{m,wait}$:waiting delay in the local model

$\tau^{tran}_{m,,n}$:transmission delay

$\tau^{t}_{m,wait} + \tau^{tran}_{m,,n} + \tau^{t}_{m,n,exe'}$: time slot required for a task to be offloaded from the endpoint to the edge server and executed to completion

$\tau^{t}_{m,max}$:maximum tolerated delay

energy consumption incurred when tasks are offloaded to the edge server

$E^{edge}_{m,n} = P^{wait}_{m}\,\tau^{t}_{m,wait} + P^{tran}_{m,n}\,\tau^{tran}_{m,n} + P^{exe}_{m,n}\,\tau^{t}_{m,n,exe}$

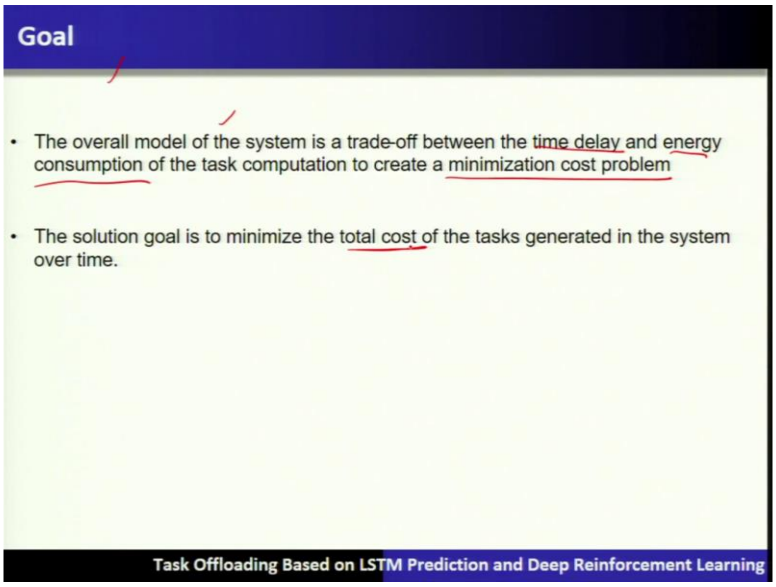
Where, $P^{wait}_{m}\,\tau^{t}_{m,wait}$,$P^{tran}_{m,n}\,\tau^{tran}_{m,n}$, $P^{exe}_{m,n}\,\tau^{t}_{m,n,exe}$ denote waiting energy consumption, transmission consumption, and edge node computation consumption of the task, respectively.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, the computational model where an edge layer computing model which we are now considering, the processing latency of the task at the edge layer is mentioned and the completion time for the edge layer task is also mentioned over here total delay of the task on an edge server and is now mentioned.

Now, energy consumption incurred when the tasks are offloaded to the to the edge server is given by this particular equation. And this denotes the waiting energy consumption transmission consumption and edge node computation consumption of the task respectively.

(Refer Slide Time: 17:25)



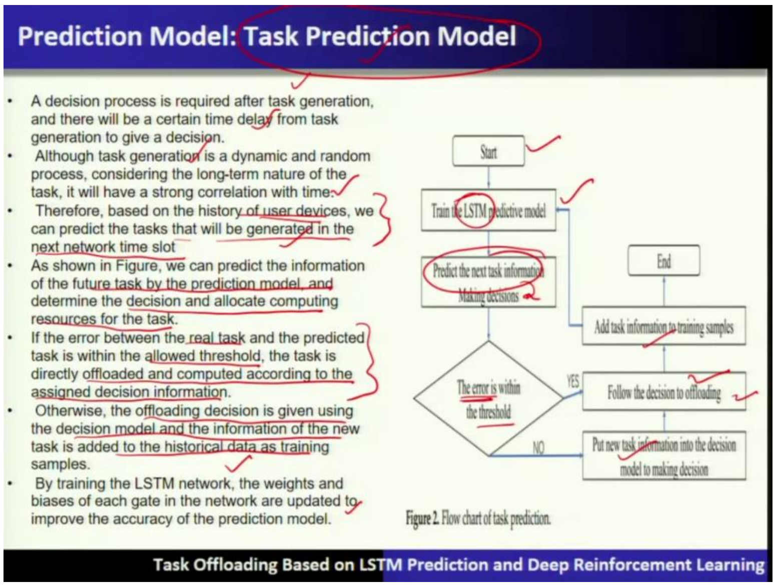
Goal

- The overall model of the system is a trade-off between the time delay and energy consumption of the task computation to create a minimization cost problem
- The solution goal is to minimize the total cost of the tasks generated in the system over time.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us understand the goal. So, overall model of the system is a trade-off between the time delay and the energy consumption of the task computation to create the minimization of a cost problem. So, the solution goal is to minimize the total cost of the task generated in the system or this period of time.

(Refer Slide Time: 17:44)



Prediction Model: Task Prediction Model

- A decision process is required after task generation, and there will be a certain time delay from task generation to give a decision.
- Although task generation is a dynamic and random process, considering the long-term nature of the task, it will have a strong correlation with time.
- Therefore, based on the history of user devices, we can predict the tasks that will be generated in the next network time slot
- As shown in Figure, we can predict the information of the future task by the prediction model, and determine the decision and allocate computing resources for the task.
- If the error between the real task and the predicted task is within the allowed threshold, the task is directly offloaded and computed according to the assigned decision information.
- Otherwise, the offloading decision is given using the decision model and the information of the new task is added to the historical data as training samples.
- By training the LSTM network, the weights and biases of each gate in the network are updated to improve the accuracy of the prediction model.

Start
Train the LSTM predictive model
Predict the next task information
Making decisions
The error is within the threshold
YES
NO
End
Add task information to training samples
Follow the decision to offloading
Put new task information into the decision model to making decision

Figure 2. Flow chart of task prediction.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, let us see the prediction model that is called a task prediction model. So, that decision process is required after the task generation and there will be a certain time delay from the task generation to give this particular decision. Although the task generation is dynamic and random process considering the long-term nature of the task, it will have a strong correlation with the time therefore, based on the history of the user devices, we can predict the task that will be generated in the next time slot.

So, that is the premise that if we know the characteristics of that particular devices, so, using that particular history and having a machine learning model, it can predict the tasks that will be generated in the next time slot. So, this is shown in this particular figure, now, we can predict the information of the future tasks by the prediction model. So, therefore, we are discussing about task prediction model and then determine the decision and allocate the computing resources upfront for that particular task. So, that is what shown were here.

Now, if we have a predictive model task predictive model, which is based on LSTM then it can predict the next task information and this will be used in making the decision and even when a new task when the real task comes this particular prediction and the actual process, actual task requirements are now compared as an error.

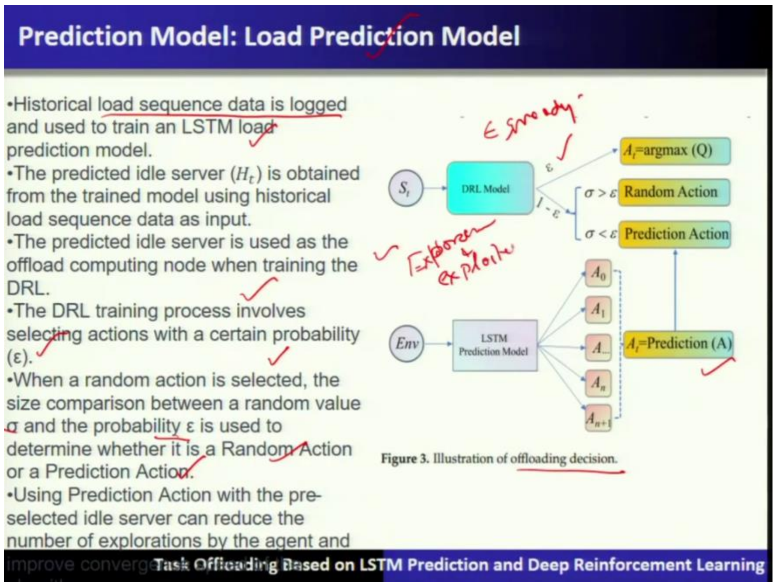
If the error is within a particular amount that is called a threshold, then we have to follow these decisions which we have taken beforehand using the prediction model about predicting the task and therefore, follow that particular task offloading based on the a priori decision making, I add this task information to the training samples and so, on.

If the error is not within the threshold that means, our prediction is going wrong then in that case, we have to put this new task information into the decision model to making the decision and then follow the rest of the other steps that is we have already taken. So, if the error is within the between the real task and the predicted task is within allowed threshold, the task is directly offloaded and computed according to the assign decision information.

Now, if this is done, then it is going to save time and therefore latency. It In terms of task allocation, so, otherwise the offloading decision is given using the decision model and the information of a new task is added in the in the historical data as the training samples. So, by training the LSTM network, the weights and the biases of each gate in the network are updated to improve the accuracy of the prediction model.

So, that is quite important to understand that here in this particular problem, we are using LSTM base task prediction model and this particular task prediction model that means, is able to decide about the resources which are required for a new task to be arrived beforehand and that task allocation is almost finalized provided it is the real task is the air is within the threshold.

(Refer Slide Time: 20:54)



Prediction Model: Load Prediction Model

- Historical load sequence data is logged and used to train an LSTM load prediction model.
- The predicted idle server (H_t) is obtained from the trained model using historical load sequence data as input.
- The predicted idle server is used as the offload computing node when training the DRL.
- The DRL training process involves selecting actions with a certain probability (ε).
- When a random action is selected, the size comparison between a random value σ and the probability ε is used to determine whether it is a Random Action or a Prediction Action.
- Using Prediction Action with the pre-selected idle server can reduce the number of explorations by the agent and improve converg

Figure 3. Illustration of offloading decision.

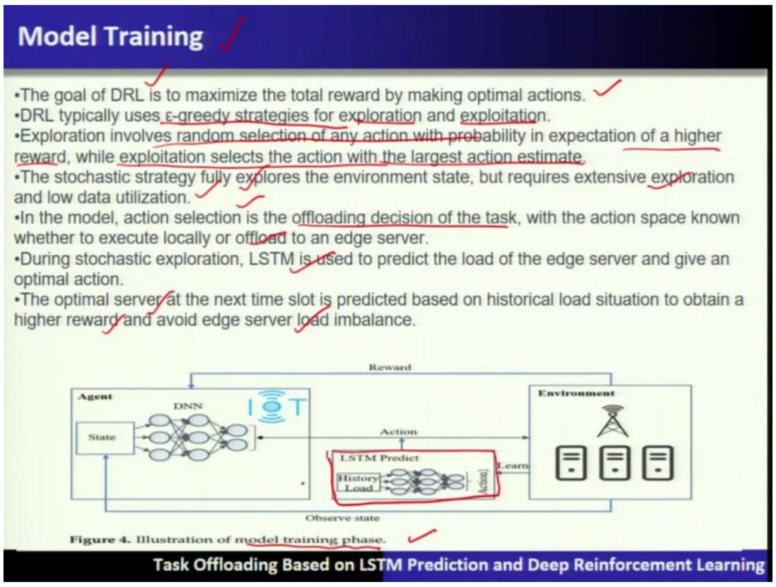
Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, therefore, the node prediction model also let us see about that. So, historical load sequence data is long and used to train LSTM load prediction model. So, the predicted idle server is obtained from the train model using historical sequence data as the input. So, the predicted idle server is used as the offload the computing node when the training DRL. So, DRL training process involves selecting the actions with a certain probability epsilon.

Now, when the random action is selected, the size comparison between the random value and the probability that is epsilon is used to determine whether the random action or a predicted action is going to take. So, epsilon greedy approach is used here for exploration purposes and otherwise it will be exploitation on purpose. So, using the prediction action and pre-selected idle server can reduce the number of explorations by the agent and improve the convergence in this particular offloading decision making. So, offloading decision is illustrated over here.

Now, you can see that as far as the environment is concerned LSTM based prediction model is used and it is able to predict upfront beforehand. Now, this particular prediction Action. Now, as far as is used in the deep reinforcement learning, so, the states of a deep reinforcement learning will now use the action. So, there are two ways. So, it uses an epsilon greedy approach in DRL. So, it says that the values of epsilon that means, it is able to support exploration and exploitation.

(Refer Slide Time: 22:56)



Model Training

•The goal of DRL is to maximize the total reward by making optimal actions.
•DRL typically uses ε-greedy strategies for exploration and exploitation.
•Exploration involves random selection of any action with probability in expectation of a higher reward, while exploitation selects the action with the largest action estimate.
•The stochastic strategy fully explores the environment state, but requires extensive exploration and low data utilization.
•In the model, action selection is the offloading decision of the task, with the action space known whether to execute locally or offload to an edge server.
•During stochastic exploration, LSTM is used to predict the load of the edge server and give an optimal action.
•The optimal server at the next time slot is predicted based on historical load situation to obtain a higher reward and avoid edge server load imbalance.

Figure 4. Illustration of model training phase.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

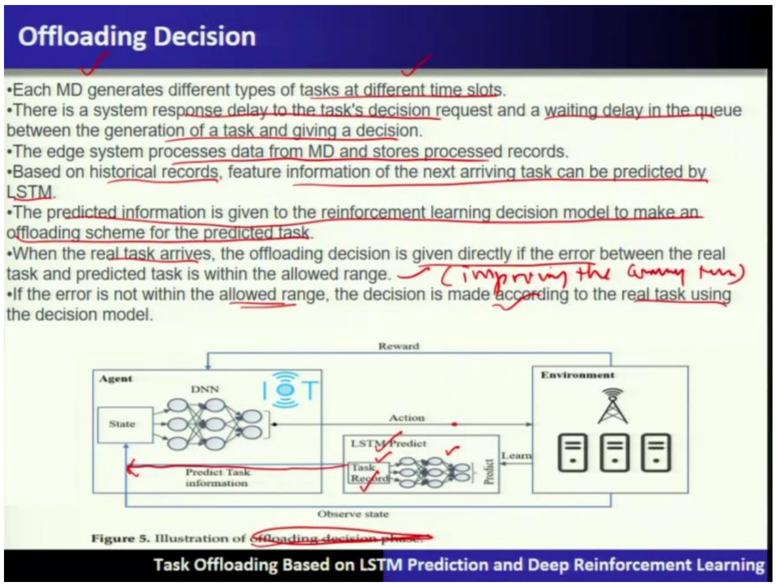
So, let us see about the first part which is called a model training. So, the goal of DRL is to maximize the total reward by making the optimal actions. So, DRL uses epsilon greedy strategies for exploration and exploitation. So, exploration involves the random selection of any action with the probability in the expectation of a higher reward, while exploitation selects the action with the largest action estimate.

So, this is targeting strategy fully explores the environment state but requires extensive exploration and a low data utilization. So, therefore, the model the action selection is the offloading decision of the task with the action in space known whether to execute locally or to offload to an edge server. So, during stochastic exploration LSTM is used to predict the load of the edge server and give an optimal action.

So, optimal server at the next time slot is created based on historical load situation to obtain a higher reward and avoid the edge server load imbalance. So, let us see this particular model training phase. So, there is a LSTM and based on this historical load, it will make a prediction and this prediction is combined into the action and this will be fed to the agent whereas, the entire enrolment is being captured as a part of the state and is given to the deep you know network for selecting that particular.

So, as I told you that this particular LSTM during the training process will take the information historical load of the edge servers and based on that it trains this particular model of deep reinforcement learning here in this case, this training processes happening let us see the offloading process.

(Refer Slide Time: 24:51)



Offloading Decision

- Each MD generates different types of tasks at different time slots.
- There is a system response delay to the task's decision request and a waiting delay in the queue between the generation of a task and giving a decision.
- The edge system processes data from MD and stores processed records.
- Based on historical records, feature information of the next arriving task can be predicted by LSTM.
- The predicted information is given to the reinforcement learning decision model to make an offloading scheme for the predicted task.
- When the real task arrives, the offloading decision is given directly if the error between the real task and predicted task is within the allowed range. (improving the accuracy here)
- If the error is not within the allowed range, the decision is made according to the real task using the decision model.

Figure 5. Illustration of offloading decision phase.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

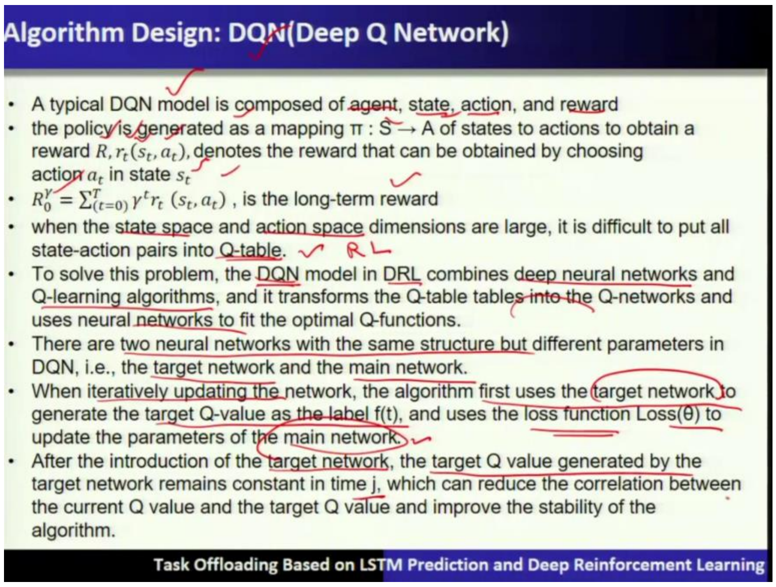
So, when offloading process then, LSTM prediction model uses the task record or a task history that will be from the mobile devices or from the devices. So, each mobile device will generate a different type of task at different time slots and there is a system response delay to the task decision request and the waiting delay in the queue between the generation of the task and giving a decision.

So, the edge system processes the data from the mobile device and stores the process records. Based on these historical records, the feature information of the next arriving can be predicted by LSTM. So, this particular LSTM base prediction of the next task is fed to the state here in deep reinforcement learning. So, the predicted information is given to the deep reinforcement learning to make the offloading scheme for the predicted task.

Now, when the real task arrives, the offloading decision is given directly, if the error between the real task and the predicted task is within the allowed range, thereby improving the convergence time and thereby latency. Now, if the error is not within that allowed range, the decision is made according to the real task using the decision model and this particular sample will be stored here for further improvement of that prediction model.

So, this illustrates the offloading decision phase. So, here you can see that now, this LSTM based task prediction model is fed that is, it will predict the task information and that is given to the state. So, therefore, DRL using the predicted task is fed to the state and then it will be evaluated based on the actual task arrival. And then the difference between the predicted and the actual is being used to measure with the help of a threshold if it is within the threshold and the predicted is more accurate and whatever DRL has taken the decision.

(Refer Slide Time: 27:10)



Algorithm Design: DQN(Deep Q Network)

- A typical DQN model is composed of agent, state, action, and reward
- the policy is generated as a mapping π : S → A of states to actions to obtain a reward $R, r_t(s_t, a_t)$, denotes the reward that can be obtained by choosing action a_t in state s_t
- $R_0^{\gamma} = \sum_{(t=0)}^{T} \gamma^t r_t\ (s_t, a_t)$, is the long-term reward
- when the state space and action space dimensions are large, it is difficult to put all state-action pairs into Q-table. RL
- To solve this problem, the DQN model in DRL combines deep neural networks and Q-learning algorithms, and it transforms the Q-table tables into the Q-networks and uses neural networks to fit the optimal Q-functions.
- There are two neural networks with the same structure but different parameters in DQN, i.e., the target network and the main network.
- When iteratively updating the network, the algorithm first uses the target network to generate the target Q-value as the label f(t), and uses the loss function Loss(θ) to update the parameters of the main network.
- After the introduction of the target network, the target Q value generated by the target network remains constant in time j, which can reduce the correlation between the current Q value and the target Q value and improve the stability of the algorithm.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

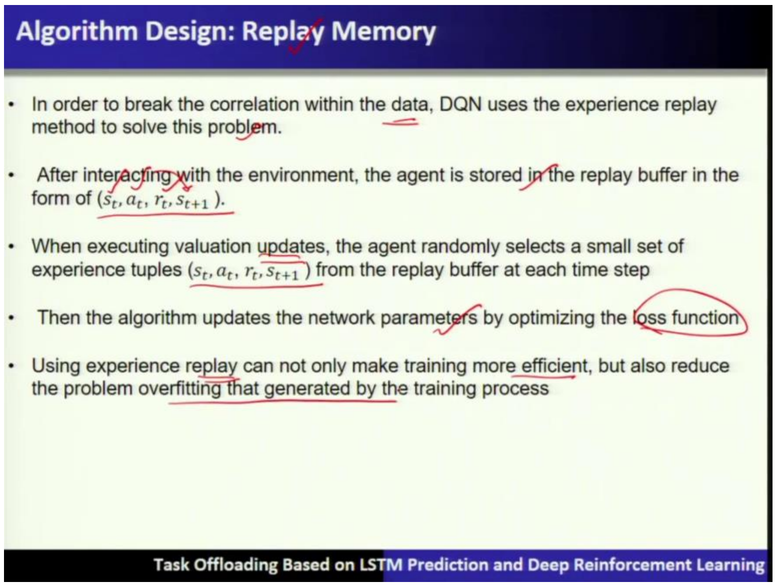
So, now, let us understand more about more detail about the DRL. So, that is nothing but a deep Q network. So, typical deep Q network is often composed of agent the state, action and reward. So, the policy is formulated as a mapping from a state to the action of the states to the action to obtain the reward or. So, the world is given over the state and a particular action denotes the reward that can be obtained by choosing the action in a given state st.

So, this particular reward function is used and summation of all the rewards is the long-term reward which is shown over here. Now, when the state space and action space dimensions are large, then it is difficult to put these action space into the Q table in a normal reinforcement learning situation. Now, to solve this, the deep Q networks model in a DRL combines a neural network also deep neural network and with the Q learning algorithm and therefore, this Q table are put into the queue networks uses the neural network to fit the optimal Q functions.

So, therefore, there are two neural networks with the same structure, but a different parameter in DQN one is called target network the other is called a main network both are being maintained simultaneously. Now, when iteratively updating the network, the algorithm first uses the target network to generate the target key values with the label f of t and uses the loss function to update the parameters of the main network.

So, just see that the target network and the main network, both are used at the same point of time here in maintaining the Q values and these parameters are updated in the main network. So, after the introduction of the target network, the target Q values are generated by the target network remains constant in time, which can reduce the correlation between the current Q values and the target Q values and improve the stability of this particular algorithm.

(Refer Slide Time: 29:18)



Algorithm Design: Replay Memory

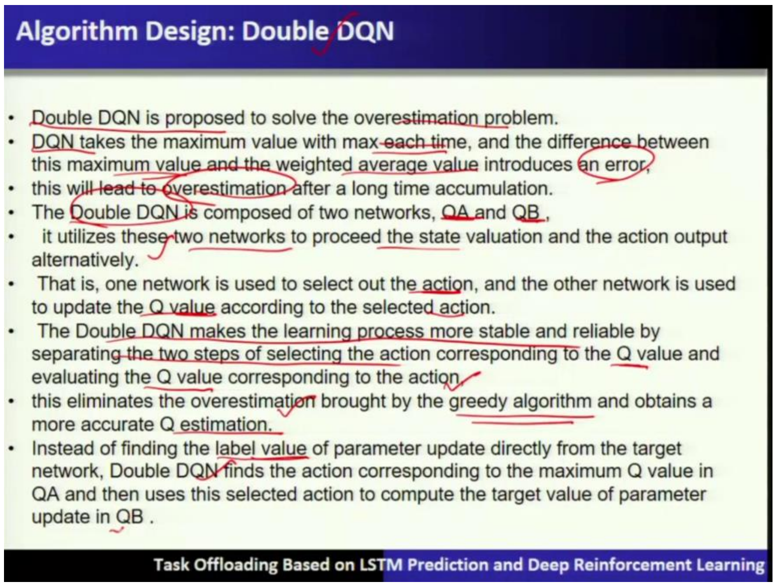
- In order to break the correlation within the data, DQN uses the experience replay method to solve this problem.
- After interacting with the environment, the agent is stored in the replay buffer in the form of (s_t, a_t, r_t, s_{t+1}).
- When executing valuation updates, the agent randomly selects a small set of experience tuples (s_t, a_t, r_t, s_{t+1}) from the replay buffer at each time step
- Then the algorithm updates the network parameters by optimizing the loss function
- Using experience replay can not only make training more efficient, but also reduce the problem overfitting that generated by the training process

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us see about the use of replay memory in this algorithm. So, in order to break the correlation within the data deep Q network uses the experience replay method to solve this problem. So, after interacting with the environment, the agent is installed in the replay buffer in this particular form. So, you can see that state at a time t Action t will make the next state and the reward at a time t.

So, when executing these valuation updates, the agent randomly selects a small set of experience tuples from the replay buffer, and then the algorithm updates the network parameter by optimizing this particular loss function. So, using experience replay can not only make this training more efficient, but also reduce the problem of overfitting that generates during by the training process.

(Refer Slide Time: 30:09)



Algorithm Design: Double DQN

- Double DQN is proposed to solve the overestimation problem.
- DQN takes the maximum value with max each time, and the difference between this maximum value and the weighted average value introduces an error,
- this will lead to overestimation after a long time accumulation.
- The Double DQN is composed of two networks, QA and QB ,
- it utilizes these two networks to proceed the state valuation and the action output alternatively.
- That is, one network is used to select out the action, and the other network is used to update the Q value according to the selected action.
- The Double DQN makes the learning process more stable and reliable by separating the two steps of selecting the action corresponding to the Q value and evaluating the Q value corresponding to the action.
- this eliminates the overestimation brought by the greedy algorithm and obtains a more accurate Q estimation.
- Instead of finding the label value of parameter update directly from the target network, Double DQN finds the action corresponding to the maximum Q value in QA and then uses this selected action to compute the target value of parameter update in QB .

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

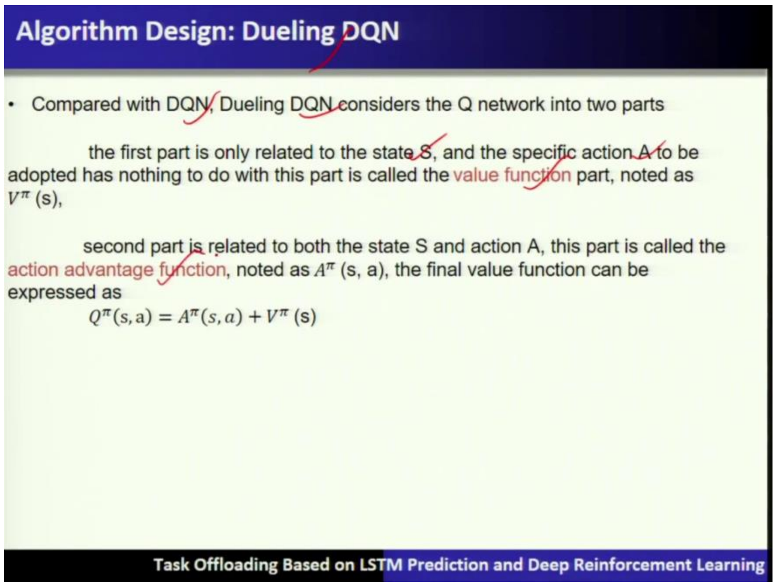
Therefore, it is proposed to solve this overfitting or over estimation problem with the help of double DQN. So, DQN takes the maximum value with a with a maximum each time and the difference between this maximum value and the weighted average value will introduce an error and this particular error will lead to an over estimation after a long-time accumulation.

Therefore, double DQN is composed of two networks QA and QB that we call it utilizes these two network to proceed the state valuation and action output alternatively, that is one network is used to select out the action and the other network is used to update the key values according to the selected action.

So, double DQN makes this learning process more stable and reliable by separating two steps of selecting the action corresponding to the Q value and evaluating the Q values corresponding to that particular action. This eliminates the overestimation brought by the greedy epsilon, greedy algorithm and obtains more accurate Q estimation.

So, instead of finding the label values of the parameter update directly from the target network double DQ one finds the action corresponding to the maximum Q value in QA and then uses the selected action to compute the target value and update in QB.

(Refer Slide Time: 31:28)



Algorithm Design: Dueling DQN

- Compared with DQN, Dueling DQN considers the Q network into two parts

  the first part is only related to the state S, and the specific action A to be adopted has nothing to do with this part is called the value function part, noted as V^{π} (s),

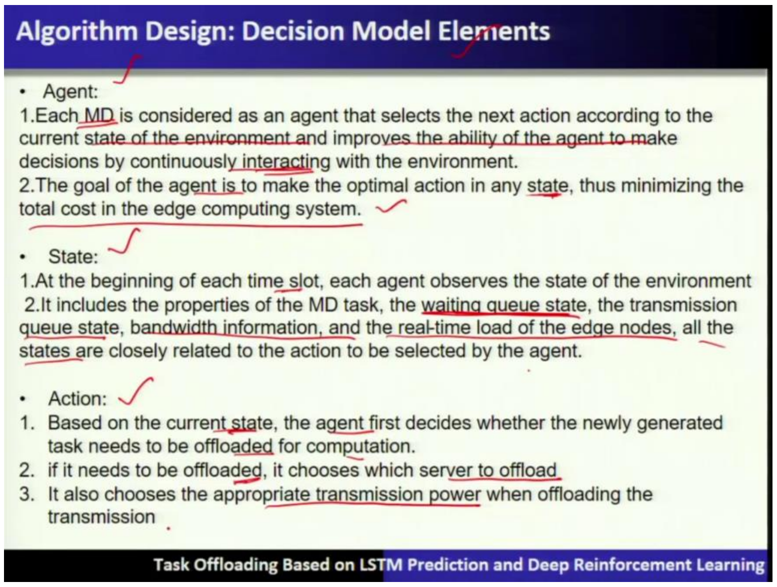
  second part is related to both the state S and action A, this part is called the action advantage function, noted as A^{π} (s, a), the final value function can be expressed as

  $Q^{\pi}(s,a) = A^{\pi}(s,a) + V^{\pi}$ (s)

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, duelling DQN compared to the DQN duelling DQ and considers the Q network into two parts, the first part only relates to the state s and a specific action to be adopted has nothing to do with this part and call value function apart noted by the second part is related to both state and action and this part is called action advantage function. So, all these things is expressed using an expression of a Q values.

(Refer Slide Time: 31:57)



Algorithm Design: Decision Model Elements

- Agent:

1.Each MD is considered as an agent that selects the next action according to the current state of the environment and improves the ability of the agent to make decisions by continuously interacting with the environment.
2.The goal of the agent is to make the optimal action in any state, thus minimizing the total cost in the edge computing system.

- State:

1.At the beginning of each time slot, each agent observes the state of the environment
2.It includes the properties of the MD task, the waiting queue state, the transmission queue state, bandwidth information, and the real-time load of the edge nodes, all the states are closely related to the action to be selected by the agent.

- Action:

1. Based on the current state, the agent first decides whether the newly generated task needs to be offloaded for computation.
2. if it needs to be offloaded, it chooses which server to offload
3. It also chooses the appropriate transmission power when offloading the transmission

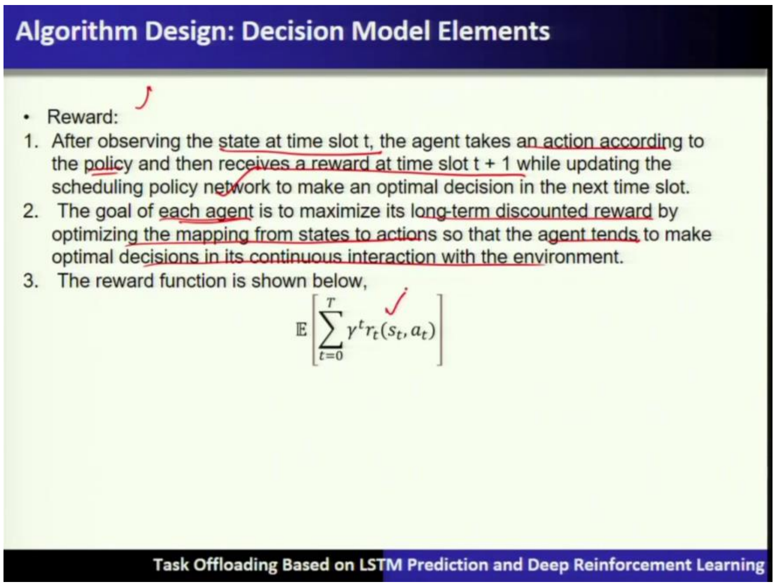
Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us see about the decision model elements here in this case, so agent means that each mobile device is considered or as an agent that selects the next action according to the current state of the environment and improve the ability of the agent to make the decision by continuously interacting with the environment. So, the goal of the agent is to make the optimal action at any state and thus minimizing the total cost in this edge computing system.

Whereas as far as the states are concerned the beginning of each timeslot each agent observes the state and it includes the properties of mobile device that is waiting que state, the transmission queue state and the bandwidth information and the real time load of the edge nodes all are the estates and are closely related to the action to be selected by the agent let us see about the action.

So based on the current state, the agent first decides whether the newly generated task needs to be offloaded for the configuration or not, if it needs to be offloaded is it chooses which server to offload, it also chooses the appropriate transmission power when offloading the transmission. So, all these are part of particular action.

(Refer Slide Time: 33:08)



Algorithm Design: Decision Model Elements

- Reward:

1. After observing the state at time slot t, the agent takes an action according to the policy and then receives a reward at time slot t + 1 while updating the scheduling policy network to make an optimal decision in the next time slot.
2. The goal of each agent is to maximize its long-term discounted reward by optimizing the mapping from states to actions so that the agent tends to make optimal decisions in its continuous interaction with the environment.
3. The reward function is shown below,

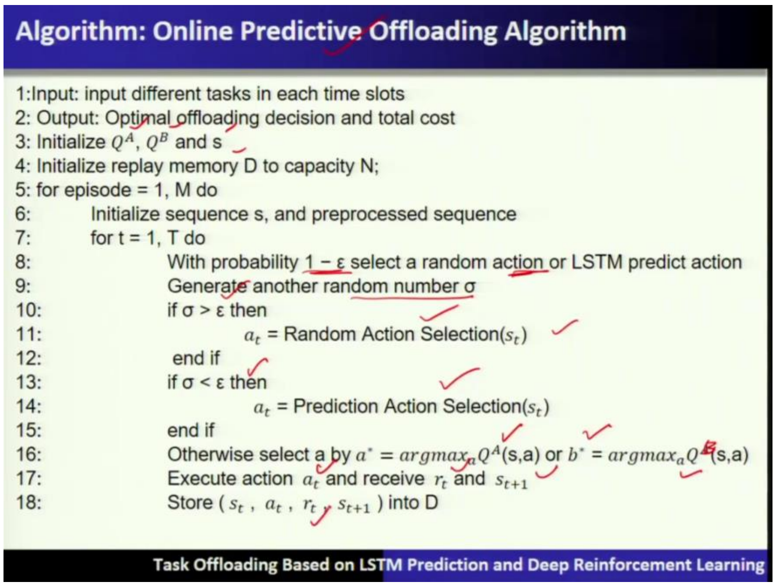
$$\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t(s_t, a_t)\right]$$

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us consider about the reward function. So, reward after observing the state at a time slot t the agent takes an action according to the policy and that receives a reward at a time slot t plus 1. So, while updating the scheduling policy network to make the optimal decision at the next time slot, so, the goal of each agent here is to maximize its long-term discounted reward by optimizing the mapping from stage to action.

So, the agent tends to make this optimal decision in its continuous interaction with the environment. So, the reward function is therefore shown here in this particular manner.

(Refer Slide Time: 33:47)



Algorithm: Online Predictive Offloading Algorithm

```
1:Input: input different tasks in each time slots
2: Output: Optimal offloading decision and total cost
3: Initialize $Q^A$, $Q^B$ and s
4: Initialize replay memory D to capacity N;
5: for episode = 1, M do
6:          Initialize sequence s, and preprocessed sequence
7:          for t = 1, T do
8:                    With probability 1 − ε select a random action or LSTM predict action
9:                    Generate another random number σ
10:                   if σ > ε then
11:                             $a_t$ = Random Action Selection($s_t$)
12:                   end if
13:                   if σ < ε then
14:                             $a_t$ = Prediction Action Selection($s_t$)
15:                   end if
16:                   Otherwise select a by $a^* = argmax_a Q^A$(s,a) or $b^* = argmax_a Q^B$(s,a)
17:                   Execute action $a_t$ and receive $r_t$ and $s_{t+1}$
18:                   Store ( $s_t$ , $a_t$ , $r_t$ , $s_{t+1}$ ) into D
```

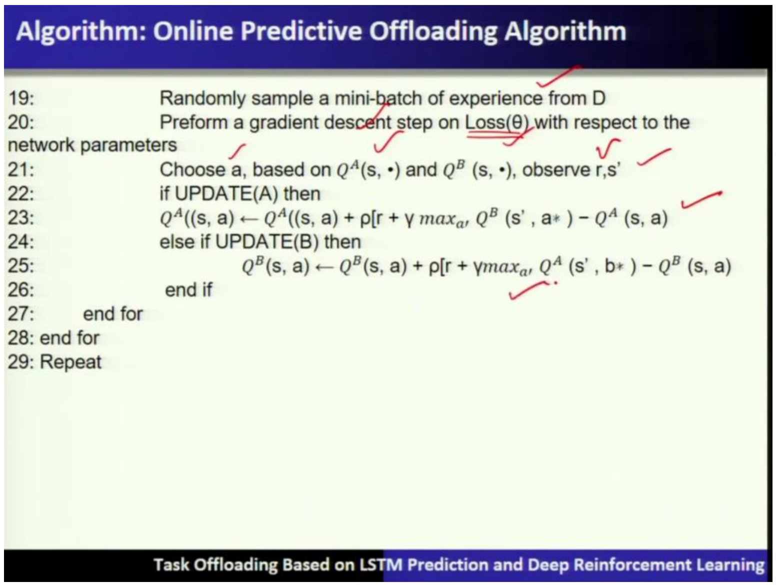
Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, the entire online predictive task offloading algorithm is now put together here and let us discuss about that. So, input is that input different tasks at each time slots output is optimal offloading decision and a total cost. So, you have to initialize QA, QB and s and initialize the replay memory D to the capacity N. Now, for each episode from 1 to M.

Now, we need to do initialize the sequences and pre-processors the sequence for each time slot t, we have to know use the probability 1 minus epsilon, to select a random action or LSTM predict action and generate another random number. So that random number is if it is greater than epsilon, then random action is selected that is st.

And if it is less than epsilon then predicted action selection is done. Otherwise select an action such that it will be the maximum of that Q value to a maximum of QA or are it is B. Then it is if it is in B then it is maximum of or B is equal to maxim of QA here in QB. Now, we have to execute the action at and receive the reward and the next state and this will be stored in the replay memory.

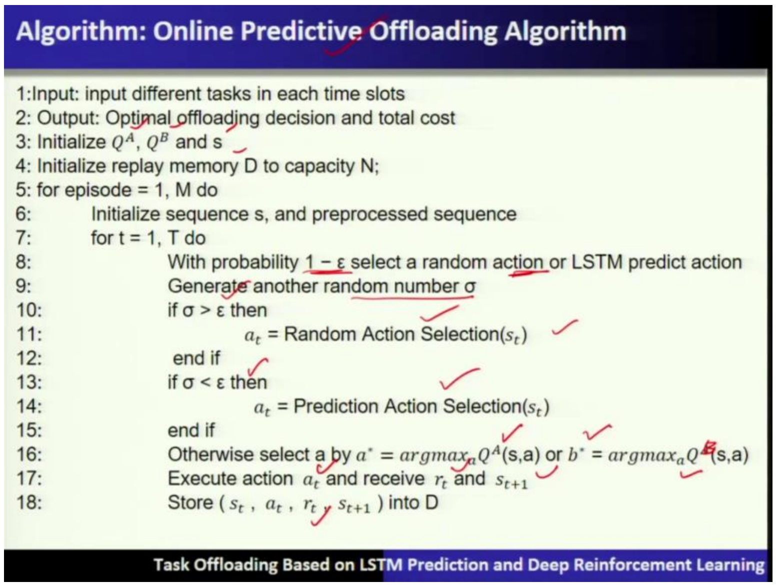
(Refer Slide Time: 35:16)



Algorithm: Online Predictive Offloading Algorithm

```
19:            Randomly sample a mini-batch of experience from D
20:            Preform a gradient descent step on Loss(θ) with respect to the
network parameters
21:            Choose a, based on $Q^A$(s, •) and $Q^B$ (s, •), observe r,s'
22:            if UPDATE(A) then
23:            $Q^A((s, a) \leftarrow Q^A((s, a) + \rho[r + \gamma\, max_{a'}\, Q^B (s', a*) - Q^A (s, a)$
24:            else if UPDATE(B) then
25:                $Q^B(s, a) \leftarrow Q^B(s, a) + \rho[r + \gamma max_{a'}\, Q^A (s', b*) - Q^B (s, a)$
26:            end if
27:      end for
28: end for
29: Repeat
```

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning



Algorithm: Online Predictive Offloading Algorithm

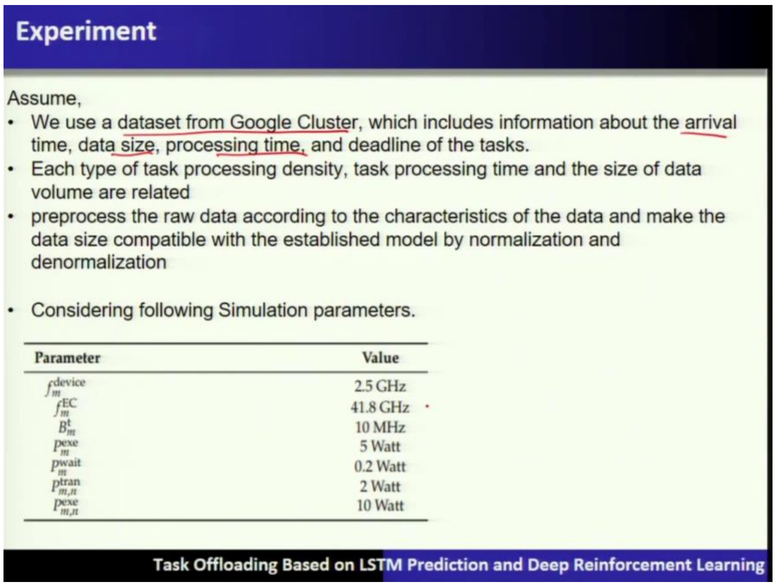
```
1:Input: input different tasks in each time slots
2: Output: Optimal offloading decision and total cost
3: Initialize $Q^A$, $Q^B$ and s
4: Initialize replay memory D to capacity N;
5: for episode = 1, M do
6:      Initialize sequence s, and preprocessed sequence
7:      for t = 1, T do
8:            With probability 1 − ε select a random action or LSTM predict action
9:            Generate another random number σ
10:           if σ > ε then
11:                 $a_t$ = Random Action Selection($s_t$)
12:           end if
13:           if σ < ε then
14:                 $a_t$ = Prediction Action Selection($s_t$)
15:           end if
16:           Otherwise select a by $a^* = argmax_a Q^A$(s,a) or $b^* = argmax_a Q^B$(s,a)
17:           Execute action $a_t$ and receive $r_t$ and $s_{t+1}$
18:           Store ( $s_t$ , $a_t$ , $r_t$ , $s_{t+1}$ ) into D
```

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, randomly sampled this mini batch of the experience from D and then perform the gradient descent on the loss function with respect to the network parameter. Now, then we have to choose A which is based on the QA network or and QB network and observe the value of reward function and as price or the next state.

Now, if an update A is done in this Bellman equation, and if you update B, then you will follow this Bellman equation and this particular algorithm, now, you can understand that it uses the LSTM based task predictions here.

(Refer Slide Time: 35:57)



Experiment

Assume,

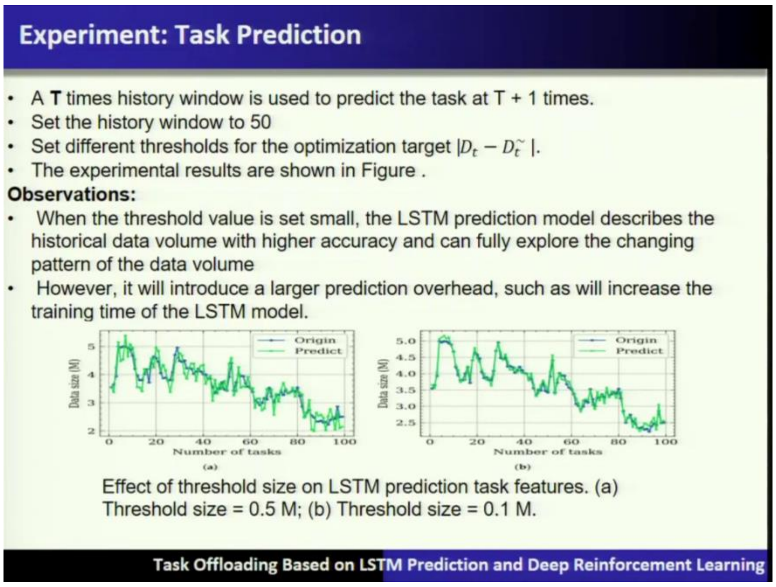
- We use a dataset from Google Cluster, which includes information about the arrival time, data size, processing time, and deadline of the tasks.
- Each type of task processing density, task processing time and the size of data volume are related
- preprocess the raw data according to the characteristics of the data and make the data size compatible with the established model by normalization and denormalization
- Considering following Simulation parameters.

Parameter	Value
f_m^{device}	2.5 GHz
f_m^{EC}	41.8 GHz
B_m^{t}	10 MHz
p_m^{exe}	5 Watt
p_m^{wait}	0.2 Watt
$p_{m,n}^{\text{tran}}$	2 Watt
$p_{m,n}^{\text{exe}}$	10 Watt

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us see this particular kind of problem how experiments will be done. So, a lot of data sets are available in public. So, let us see that we use data set using the Google cluster information, which includes the information about arrival time, data size, processing, time and so on. And therefore, considering the following simulation parameters which are available in the literature.

(Refer Slide Time: 36:25)



Experiment: Task Prediction

- A **T** times history window is used to predict the task at T + 1 times.
- Set the history window to 50
- Set different thresholds for the optimization target $|D_t - \tilde{D_t}|$.
- The experimental results are shown in Figure .

Observations:

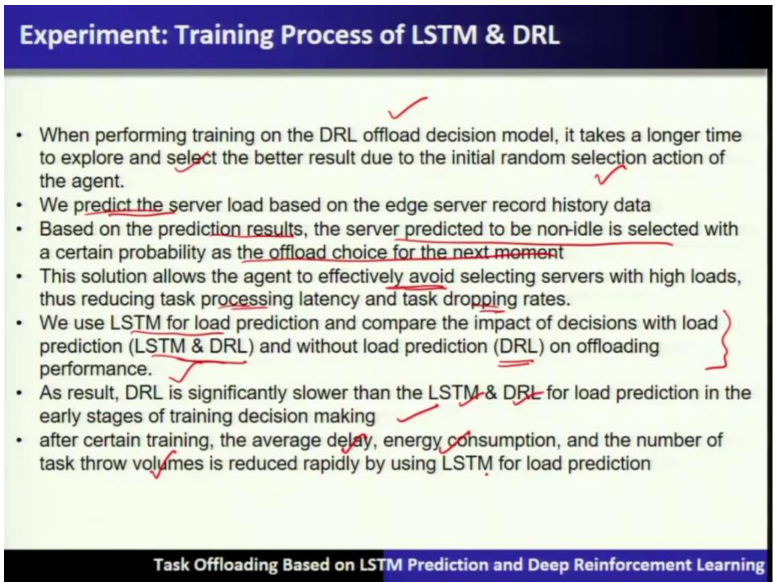
- When the threshold value is set small, the LSTM prediction model describes the historical data volume with higher accuracy and can fully explore the changing pattern of the data volume
- However, it will introduce a larger prediction overhead, such as will increase the training time of the LSTM model.

Effect of threshold size on LSTM prediction task features. (a) Threshold size = 0.5 M; (b) Threshold size = 0.1 M.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, this particular task prediction for this kind of problem, which are available in the literature is shown here about different values and the training process of LSTM and DRL we have explained.

(Refer Slide Time: 36:34)



Experiment: Training Process of LSTM & DRL

- When performing training on the DRL offload decision model, it takes a longer time to explore and select the better result due to the initial random selection action of the agent.
- We predict the server load based on the edge server record history data
- Based on the prediction results, the server predicted to be non-idle is selected with a certain probability as the offload choice for the next moment
- This solution allows the agent to effectively avoid selecting servers with high loads, thus reducing task processing latency and task dropping rates.
- We use LSTM for load prediction and compare the impact of decisions with load prediction (LSTM & DRL) and without load prediction (DRL) on offloading performance.
- As result, DRL is significantly slower than the LSTM & DRL for load prediction in the early stages of training decision making
- after certain training, the average delay, energy consumption, and the number of task throw volumes is reduced rapidly by using LSTM for load prediction

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

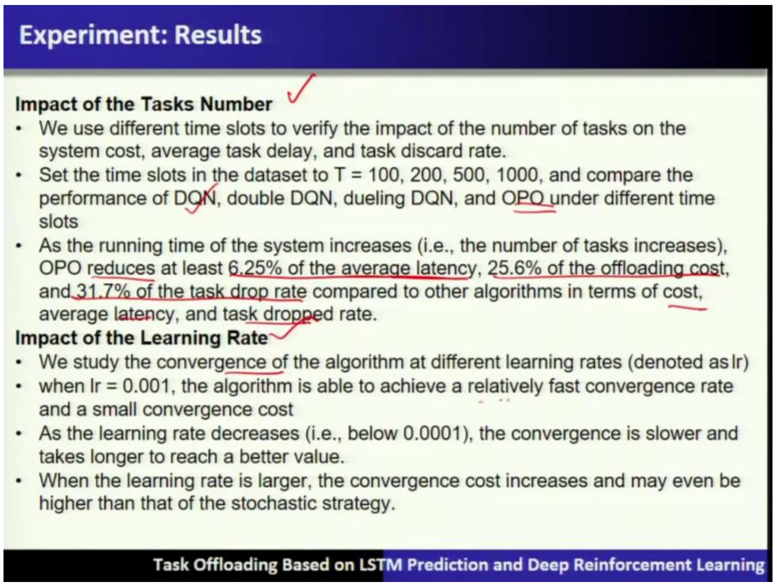
So, when performing the training on DRL. So it will offload the decision model and it takes a longer time to explore and select the better result due to the initial random selection actions of the agent. Now, we predict the server load based on the edge server record history and based on these prediction results, the server predicts to be non-ideal is selected with a certain probability as the as the offload choice for the next moment.

So, therefore, you can see that there are two different historical data sets are maintained, whether it is at the training phase or at the testing phase. So, this solution allows the agent to effectively avoid selecting the servers with a high load and thus reducing the task processing latency and task dropping rates.

Now, here LSTM is used for a load prediction and compare the impact of decision with a with a load prediction that is LSTM plus DRL. And without LSTM, only DRL is used for offloading. So, if you if you compare you will find that there is a good amount of possible reduction in or overcoming from the two problems which we have stated that is latency and energy.

Now, as a result, DRL is significantly slower than LSTM and DRL for load prediction in the earliest stages of the training, after certain training, the average delay, energy consumption and the number of throw is reduced rapidly using LSTM for the load predictions.

(Refer Slide Time: 38:07)



Experiment: Results

Impact of the Tasks Number ✓

- We use different time slots to verify the impact of the number of tasks on the system cost, average task delay, and task discard rate.
- Set the time slots in the dataset to T = 100, 200, 500, 1000, and compare the performance of DQN, double DQN, dueling DQN, and OPO under different time slots
- As the running time of the system increases (i.e., the number of tasks increases), OPO reduces at least 6.25% of the average latency, 25.6% of the offloading cost, and 31.7% of the task drop rate compared to other algorithms in terms of cost, average latency, and task dropped rate.

Impact of the Learning Rate

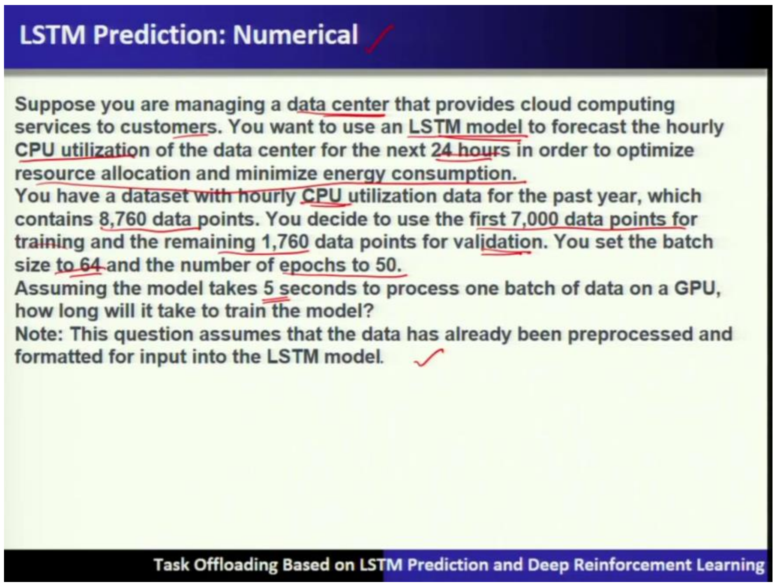
- We study the convergence of the algorithm at different learning rates (denoted as lr)
- when lr = 0.001, the algorithm is able to achieve a relatively fast convergence rate and a small convergence cost
- As the learning rate decreases (i.e., below 0.0001), the convergence is slower and takes longer to reach a better value.
- When the learning rate is larger, the convergence cost increases and may even be higher than that of the stochastic strategy.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Impact of the task numbers are also studied here. So that means if you consider the data set t is called 100 200-500-1000 and compare the performance of DQN with the double DQN duelling DQN your optimal prediction algorithm under different time slots. So, what you will find that the system will increase the number of time slots increases. So, this predictive task offloading will reduce its latency on an average by 6 percent.

And offloading cost is also reduced by 25 percent. And task drop rate is also reduced by 31 percent, which is quite substantial. And therefore, compared to the other algorithm, the cost average latency and the task drop rates are quite improved with the help of LSTM based prediction model. So, now, let us see about the impact of learning rate and now study the convergence of the algorithm at different learning rate. So, the algorithm is able to achieve a relatively fast convergence rate and a small convergence cost.

(Refer Slide Time: 39:12)



LSTM Prediction: Numerical

Suppose you are managing a data center that provides cloud computing services to customers. You want to use an LSTM model to forecast the hourly CPU utilization of the data center for the next 24 hours in order to optimize resource allocation and minimize energy consumption.
You have a dataset with hourly CPU utilization data for the past year, which contains 8,760 data points. You decide to use the first 7,000 data points for training and the remaining 1,760 data points for validation. You set the batch size to 64 and the number of epochs to 50.
Assuming the model takes 5 seconds to process one batch of data on a GPU, how long will it take to train the model?
Note: This question assumes that the data has already been preprocessed and formatted for input into the LSTM model.

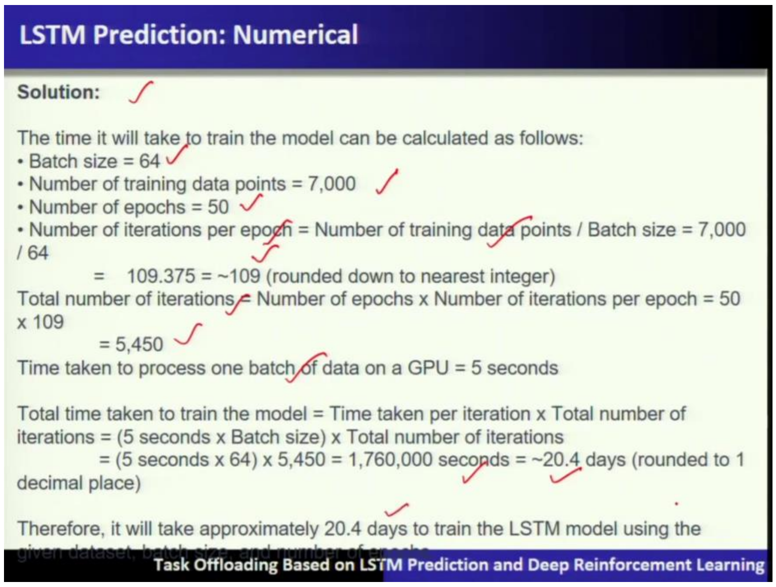
Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us see some numerical analysis of this LSTM based predictions. Suppose you are managing a data centre that provides a cloud computing services to the customers. And now you use LSTM model to forecast an hourly CPU utilization of the data centre within the next 24 hours in order to optimize this resource allocation and minimize the energy consumption.

Now then, if you have the data set with the hourly CPU utilization data for the last one year, which contains let us say 8,760 data points, so you must choose the first 7000 data points for training and the remaining 1760 data points for validation. Then you set the batch size to 64 and the number of epochs to 50.

Now assuming that the model takes 5 seconds and to process one batch of the data on a GPU. So, how long will it take to train the model? Now, this question assumes that the data has already been pre-processed and formatted into the LSTM model.

(Refer Slide Time: 40:15)



LSTM Prediction: Numerical

Solution:

The time it will take to train the model can be calculated as follows:
- Batch size = 64
- Number of training data points = 7,000
- Number of epochs = 50
- Number of iterations per epoch = Number of training data points / Batch size = 7,000 / 64

  = 109.375 = ~109 (rounded down to nearest integer)

Total number of iterations = Number of epochs x Number of iterations per epoch = 50 x 109

  = 5,450

Time taken to process one batch of data on a GPU = 5 seconds

Total time taken to train the model = Time taken per iteration x Total number of iterations = (5 seconds x Batch size) x Total number of iterations

  = (5 seconds x 64) x 5,450 = 1,760,000 seconds = ~20.4 days (rounded to 1 decimal place)

Therefore, it will take approximately 20.4 days to train the LSTM model using the given dataset, batch size, and number of epochs

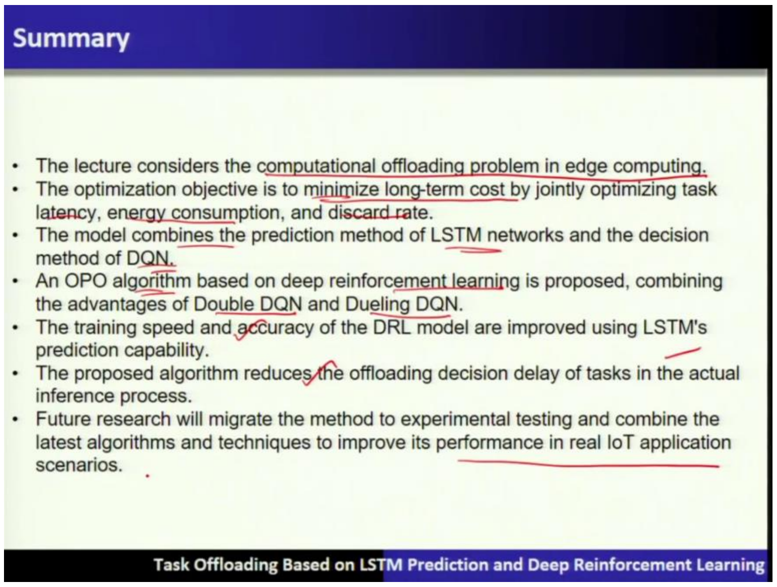
Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

So, let us see the problem solution, which is going to give you the insight into these numerical calculations. So, the time it takes to train the model can be calculated in this in this manner. So, batch size if you keep 64 and the number of training points, let us say you keep 7000 number of epochs 50 And the number of iterations per epochs.

So, that is nothing, but the number of training data points divided by the batch size, it comes out to be around 109, that is number of iterations per epoch. So, the number of iterations is equal to number of epochs into number of iterations per epoch that is 5450 and time taken to process one batch of the data on the GPU is 5 seconds.

So, the total time taken to train the model will be 5 multiplied by the batch size multiplied by total number of iterations, that comes out to be 20 point 4 days for full pledged convergence based training therefore, it will take approximately this much of days to train the LSTM base model using the given data set and batch size and the number of iterations are here.

(Refer Slide Time: 41:23)



Summary

- The lecture considers the computational offloading problem in edge computing.
- The optimization objective is to minimize long-term cost by jointly optimizing task latency, energy consumption, and discard rate.
- The model combines the prediction method of LSTM networks and the decision method of DQN.
- An OPO algorithm based on deep reinforcement learning is proposed, combining the advantages of Double DQN and Dueling DQN.
- The training speed and accuracy of the DRL model are improved using LSTM's prediction capability.
- The proposed algorithm reduces the offloading decision delay of tasks in the actual inference process.
- Future research will migrate the method to experimental testing and combine the latest algorithms and techniques to improve its performance in real IoT application scenarios.

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning

Now, let us summarize this entire lecture that in this lecture, we have considered the computational offloading problem, which is very important in bringing up this edge computing. So, this optimization objective is to minimize the long-term cost by directly optimizing task latency energy consumption and discard task discard rate.

So, so, the model combines this that is a prediction model of LSTM networks and the addition model of deep Q network. So, this prediction-based algorithm that is task of learning algorithm is based on deep reinforcement learning has the advantage if let us say that you combine it with double DQN and duelling DQN.

So, that means, advanced level of advanced models are more complicated complex models are at advantages. So, that is the training speed and accuracy of deep reinforcement learning models are quite improved with the help of LSTM prediction capabilities. So, the proposed. So, this particular algorithm produces the offloading decision delay of the task in the actual inference model.

So, therefore, this can be taken forward in the development of edge computing and therefore, will improve the performance in the real IoT applications with the help of edge computing. Thank you.