Technische Universität Berlin

Open Distributed Sytems

ODS

Seminar Open Distributed Systems

Learning IoT in edge: Deep learning for the Internet of Things with edge computing

Name: Mariia Kucheiko
Matrikel: 400224

13th July 2020

Contents

1 Abstract

Internet of Things generates a tremendous volume of data. On the one hand, this provides an unprecedented opportunity to learn more about the environment, society and individuals. On the other hand, it posts an urgent need for methods of fast and secure data processing. Machine Learning has been exploited in Data Science for decades, but its traditional methods often cannot hold on with high requirements on efficiency, speed and complexity of the current tasks. This is where the advantages of Deep Learning come into play. Its advanced techniques can deal with big volumes of unstructured data from messy environments. They extract sophisticated features and perform predictions or classifications of a high accuracy. The advantages of Deep Learning have nevertheless their costs: the models are often very computation- and memory intensive. At the same time, the computational power of the Internet of Things is also rising, so that Deep Learning models can be carried out and provide a high-quality decision basis for the IoT devices. This paper provides insights into the question: how can Deep Learning methods be efficiently implemented in the context of the Internet of Things. For this purpose we state a scheduling optimization problem and suggest two solution algorithms, which proved to outperform other solution mechanisms.

2 Introduction

Internet of Things (IoT) is a global network of internet-enabled devices, sensing information from the environment, exchanging it and acting with or without direct human intervention. The number of installed IoT devices grows tremendously and so does the volume of data they generate and base their behaviour on. Therefore, efficient methods of processing are required in order to make use out of this data. They must also provide the basis for high-quality decision making for IoT devices to reach high levels of autonomy [1],[2].

The data under consideration is collected from complex noisy environments, and therefore often confuses classical machine learning methods. Moreover, beside structured data, such as temperature, pressure, etc., there is an increasing demand on processing of high-volume unstructured multimedia data, such as video, sound and images [3].

Hence, a more powerful analytical tool is needed to enable gaining information out of invaluable raw data. Deep Learning is a brunch of Machine Learning, which exploits techniques that often outperform its traditional methods on large scale data. They are also capable of mining more complex characteristics and dependencies. Its models are used for end-to-end learning, with automatic extraction of features of interest. This eliminates the necessity of theirs hard-crafted specification[1].

Nevertheless, these advantages do not come without costs: Deep Learning models are computation- and memory-intensive, their training even more so than their execution. Therefore the state-of-the-art approach is to transmit the raw data generated in the IoT to the cloud and run the Deep Learning model there. Several challenges occur during this process. First, many IoT applications have soft or hard real-time requirements. Transmission latency may cause delays in system response which are incompatible with them. Second, the scalability of this architecture might be an issue since the network access to the cloud might become a bottleneck. Thirdly, transmission of unprocessed data from IoT devices to the cloud causes significant internet traffic. And finally, this approach causes privacy concerns since sensitive raw data is susceptible to theft and fraudulent activities.

For the above mentioned reasons a shift of paradigms took place in the recent years. The opportunities of data processing close to its source have been intensively explored. A new concept of edge computing emerged as a result of these efforts. Its advantages lay in lower internet traffic, reduced workload of the bottleneck resources and better privacy preservation.

The layered structure of Deep Learning models enables its deconstruction and assignment of tasks to different hierarchical levels of the IoT network: edge devices, edge servers and the cloud servers. Such hybrid architectures provide flexibility for optimal end-to-end computing. The distribution of tasks is nevertheless a highly nontrivial problem for a number of reasons. First, edge devices are usually equipped with lower computational power than cloud servers and often have memory and energy consumption constraints. Therefore only a limited amount and complexity of tasks can be performed at place. Moreover, splitting of a Deep Learning model between different actors inevitably increases coordination overhead of the process [4].

This paper deals with optimization of distribution of DL tasks over the IoT network under consideration of the above mentioned issues and trade-offs. It is structured as follows. First, in order to provide the reader with necessary basic knowledge of Deep Learning and Internet of Things, these 2 topics are introduced in the section Background. Then the application of Deep Learning methods in the context of IoT is presented. Following topics are covered: established approach for implementation of DL in the IoT and the recent trends; performance evaluation of DL models; enabling techniques for DL implementation in different IoT architectures. In the final section a DL task scheduling problem and two solution methods are discussed in details. Section Conclusion provides a summary of the accomplished work.

3 Background

3.1 Deep Learning

Articles [4] and [5] provide an excellent introduction to Deep Learning. According to them, Deep Learning is a brunch of Machine Learning which explores the deep structure of data in order to perform classifications and/or predictions. Its core tool is Artificial Neural Network, a model inspired by the biological neural network of the human brain. Artificial neurons are organized in layers, including an input layer, an output layer, and at least one more hidden layers. Neural Networks with multiple hidden layers are called Deep Neural Networks.

The data to be analyzed passes through the layers subsequently. Each layer receives its input either form outside of the model in case of input layer, or from the upstream layer. Then it performs computations and forwards its results to the next layer. The result produced by the output layer is then a prediction or a feature.

In its training phase a model is fed with labelled data and each neuron receives an initial random set of weights stored in form of a matrix. Each layer than performs matrix multiplications on its input data. After the data has passed through the network, the performance of the model is evaluated by comparison of actual labels with the predicted ones. With the help of an optimization method, such as Stochastic Gradient Descend, the weights of the layers are updated. This process starts from the final layer and is repeated in each run of the algorithm until the weights of all layers are updated. At the end, when the error rate reaches a predefined acceptable level, the model is ready to be applied on the unlabelled test data in the inference stage. Here each data set passes only once through the Neural Network from the input to the output layer and the computations are performed with the parameters set during the training stage.

The crucial takeaways for the further content of this paper are following facts:

- Training and applying Deep Learning models is memory-, computation- and energy-intensive due to their complex structure, large number of runs needed to fine-tune parameters and extensive computations associated with it. The training stage requires even more resources than the inference stage.

- The degrees of freedom of the model structure, such as number of layers and neurons per layer, their inter-connectivity and parameters setting etc., make the model design highly nontrivial. On the other hand this property provides flexibility for solving the trade-offs between accuracy and resource consumption of the model.

3.2 IoT network and its application domains

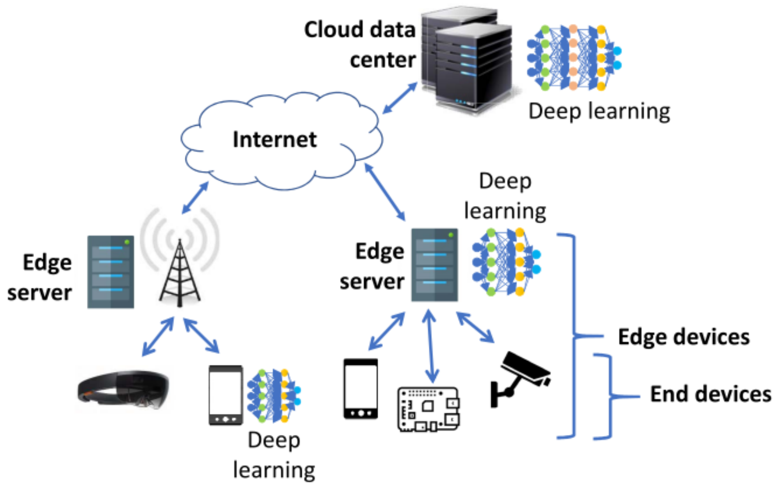

Figure 1: Typical structure of an IoT network

Figure 1 provides an overview of a typical IoT network structure [4]. Its main actors are:

1. Cloud server, providing flexible infrastructure to run computationally intensive tasks;
2. Edge servers, devices with significant computational power, which often lie on the border between public network and intranet;
3. IoT end devices, such as sensors, cameras, smart mobile devices, etc.

IoT networks enable revolutionary new ways of managing domains, many of which have potential to improve users' quality of life or productiveness of their activities. Intelligent transportation systems, for example, suggest methods to improve traffic efficiency, drivers' and passengers' safety, and freight transport. Applications range from helping drivers find a free parking slot, over detection and minimization of traffic congestions, to autonomous self-driving vehicles. Smart Healthcare covers methods for predicting and preventing disease outbreaks, symptoms detection and reporting, support while treatment and recovery and many more. Disaster management and crowd surveillance are important applications of IoT supported public safety. They help to identify suspicious activities, detect injuries, find missing people, etc. IoT was also the engine of the 4. Industrial revolution. It enables faster access to information, higher level of autonomy of production process and therefore less need for human intervention. Smart home and smart building are concepts assisting their users in managing their resource consumption, as well as surveillance and access control [6]. This list is by no means exhaustive, but continuing it would go beyond the scope of this paper.

4 Deep Learning in the context of IoT

Considering the characteristics of Deep Learning models mentioned in the Background, it is hardly surprising that conventional approach to their training and application is cloud-centric. Nevertheless the centralized approach has a number of disadvantages. For instance, as discussed in the introduction, transmission of data from the edge devices to the cloud causes high traffic and, even more important, high latency.

Besides that, there is a number of factors enabling the change of the cloud-centric paradigm. First, the layered structure of Deep Learning models allows their distribution among multiple devices. Those can perform computations, communicate the results and consolidate them to solve the problem at hand. And second, the IoT end devices are becoming smarter, e.g. they evolve from a rather passive sources of data to actual actors, capable of performing computations and making decisions.

Thus, for many applications it might be beneficial to consider shift of responsibilities from the cloud to the rest of the IoT network. The options here range from binary decisions, i.e. whether to offload the model or not, over hierarchical distribution of tasks along the network, to peer-to-peer implementations. This has a potential to reduce the total response time of the application by avoiding unnecessary data traffic and exploiting computational power of the whole IoT network [4],[5].

In this section we will first discuss in more details, to which extend offloading of the tasks from the cloud to the edge is possible. Then we review the metrics which help to decide for the optimal degree of decentralization. And at the end, we present techniques, which help to enable Deep Learning on resource constrained IoT devices.

4.1 Degrees of decentralization

Figure 2 shows that training and application of Deep Learning models can be decentralized to different extends.

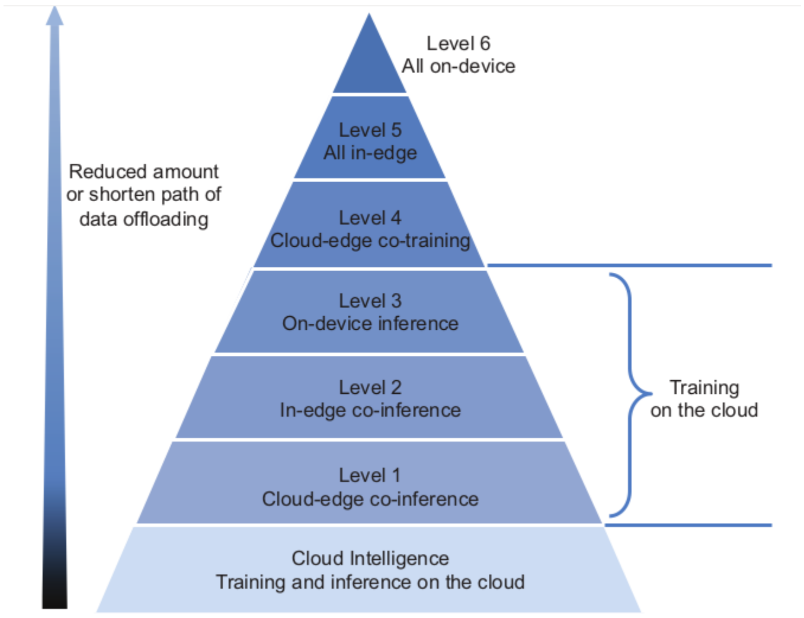

Figure 2: Levels of decentralization

The completely centralized approach with both training and inference in the cloud rests on the level 0. Middle layers represent different hybrid architectures with the amount of tasks taken over by the edge rising with the the level's number. It is worth noting that since training of a DL model is more computationally- and memory-intense than the inference, the letter can be easier offloaded from the cloud to the edge. Therefore in the first 3 levels of decentralization, training is performed in the cloud. Only with the rising degree of decentralization also the training part is partially or completely done in the edge. Level 6 stands for a fully decentralized approach with both training and inference performed on the end devices. Here, each node trains its own DL model with local data and afterwards communicates the updates with other nodes, which is possible completely without involving the cloud [5].

4.2 Key Performance indicators

Latency can be subdivided into latency of the training phase and of the inference phase. The former captures the time spent until the model is set up and ready to be applied. The latter refers to pre-processing of the raw data, run of the model and communication of the results. Latency depends on a number of factors, such as: placement of the Deep Learning tasks along the network, computational power of the executing devices, the size of data transmitted during the process and the available network bandwidth.

Accuracy refers to the capability of the model to correctly classify objects of interest. Often higher accuracy is achieved by more sophisticated models with larger number of parameters or/and layers. Hence the trade-off between accuracy on the one hand and memory and energy consumption on the other. Besides the actual inference capability of the model, accuracy depends on the quality of the communication network. For example, if data needed for inference is lost due to missing transmission capacity, the quality of the results might well be lower than it is under ideal circumstances.

Communication overhead can be high in both centralized and decentralized applications. The former are based on extensive exchange of raw and intermediate data between the edge devices and the cloud server. In the latter case, separately acting devices need to coordinate and reach consensus. The required effort and therefore communication overhead is the higher the more the model is decomposed and spread over different devices.

Both *energy-* and *memory consumption* of a Deep Learning model become the more important, the more decentralized the model is applied. Devices located further downstream the network usually have less memory to store and run their assigned tasks, as well as are often battery-constrained.

Privacy is naturally better preserved if data is pre-processed in the protected environment on device or in the edge server. After pre-processing the semantics of data is often changed, e.g. data is aggregated, which makes it harder to deduce the single person-, location- or object-related data. Therefore it can be safer sent via public internet if needed. Besides protection against leakage of sensitive data, another aspect of privacy has to be assured, namely the so called differential privacy. This means that non of the participants of a Deep Learning algorithm may remember details about any specific device's input data.

Convergence is a KPI specific for decentralized training. It characterizes whether and how fast the nodes with their separately trained sub-models can reach a consensus on the parameters for the aimed coherent model. [4],[5]

4.3 Enabling techniques

4.3.1 Inference in the edge

The term *model compression* unites a set of techniques which reduce the memory and energy consumption of a Deep Learning model thus making it applicable on resource-constrained devices. Among them are parameter quantization, pruning, knowledge distillation, edge caching, early-exit, input filtering, etc.

Weight pruning includes checking an existing model for redundant weights, i.e. neuron connections, and their removal. Neurons are first sorted in order of their contribution to the output of the model. Those which do not pass a threshold are then removed. If this process negatively influences prediction accuracy, a further fine-tuning of the remaining neurons and their weights might be necessary. Weight pruning naturally reduces the memory needed for storage and computation of the model.

Parameter quantization is a technique which leverages light-weight formats for storage of input information and/or neurons' weights instead of heavy ones, such as 32-bit floats. This reduces memory consumption and avoids costly floating-point multiplications.

Knowledge distillation refers to a method of creation of a light-weight DL model out of a more complex one. This is done by training the former based on the result of the latter. Eventually the smaller model approximates the prediction function and thus mimics the behaviour of the more powerful model.

Edge caching makes use of the insight that requests received in a short time span from closely located devices can be similar. Thus it is worth trying to reuse the results of previous runs. As the name suggests, the requests and the corresponding results can be stored on the end devices or at the edge server. In case of the cache miss, the data can be either forwarded to computationally stronger device or the Deep Learning model is run in place. And in case of cache hit, the response time of the system is significantly reduced.

Early-exit is a strategy that uses output of intermediate levels of a DL model for final classification if a certain level of inaccuracy is acceptable. The threshold for different classes can be set differently and thus easy-to-recognise objects don't need to go through the whole model.

Input filtering is a data pre-processing technique. It erases part of the raw data which most probably does not carry any important insights and is therefore of no interest for the following steps of Deep Learning. It is widely used in computer vision because it can significantly reduce the size of transmitted data, which is crucial in this application.[4],[5]

4.3.2 Edge-based Training

Edge-based training is an approach, in which edge devices learn in place as a distributed system without exchanging their data sets or sending them to the cloud. The main question here is: how the final coherent model can be build out of the components provided by the nodes. Obviously, they need to aggregate their results and reach a consensus, which might happen with or without a coordinating actor.

In centralized systems there are two methods for the devices to provide their updates, i.e. results of a computation run, to the coordinating entity: synchronous and asynchronous stochastic gradient descend (SGD). In the former case, it happens in a lockstep after all participants are done with calculations of their gradients. Naturally, this approach slows the whole process down to the speed of the weakest entity. If asynchronous SGD is chosen, peers do not wait for each other. This speeds up the learning, but might converge to a model of a poor quality, since learning might be based on outdated results of previous runs. To overcome these issues, a number of hybrid approaches has been suggested, which usually either focus on speeding up synchronous SGD or on improving accuracy of asynchronous SGD.

Elastic averaging, for example, allows devices to compute several runs without providing an update to the coordinating entity. The extent of this independence is constrained by the risk of overfitting of each device's model to the local data. In order to avoid this problem, the knowledge distillation principle described in the previous section can also be used in this case. According to it, a node trains its model not only based on its own data sets. It also makes use of the learning outcomes of its peers.

Another strategy for speeding up synchronous SGD is by using redundant or backup devices. They jump in if any regular device has difficulties communicating his updates. As soon as the coordinating entity receives sufficient number of updates, it aggregates them and moves on to the next iteration without waiting for feedback from every single node.

A fully decentralized and asynchronous approach in edge-based training is *gossip learning*. Here, each peer exchanges updates of its model with one or multiple other peers. In order to avoid overfitting, exchange partners can be selected randomly. The main issue here is convergence, since the models learned by peers which had less interaction will deviate significantly.

Due to a risk of high communication overhead, a special attention has to be payed to the frequency of communication and the size of exchanged information. Two methods described above, elastic averaging and gossip learning, contribute to decrease of update communication system-wide.

Regarding the size of updates, research estimates that 99,9% of exchanged gradients do not contribute to accuracy of the final model. Here is where optimization potential of *gradients sparcification* lies. With the help of thresholds, only non-redundant gradient updates can be filtered out. Besides that, outdated residual gradients should be eliminated. Similar to the *parameter quantization* used for speeding up inference, gradient quantization can be leveraged to reduce the size of updates by using light-weight formats for model parameters. [4],[5]

4.3.3 Both training and inference

In this subsection we will discuss several enabling methods, which are not specific for training or inference phase of Deep Learning. They help to overcome issues arising in the edge or contribute to improvement of KPI of the system. Among them are *hardware-centric* methods as well as *privacy* preserving techniques.

In order to deal with the resource constrains of the edge, hardware manufacturers suggest either specific software solutions for existing CPUs and GPUs, or produce application-specific integrated circuits. The latter sacrifice the variety of tasks which can be performed on the hardware in favour of efficient memory access for specific operations.

Privacy can be preserved by *adding noise* to the data before transmitting it over public network. This can be done to the training data, which is sent from the end device to the processing unit. Alternatively, a small Deep Learning model running on edge devices can extract features from raw data, add noise to its results and forward it to the cloud for further processing. Crucial point is that the Deep Learning model in the cloud is aware of the fact that the data is noisy and knows how to deal with it, in contrast to an external actor.

Another method of privacy preservation- *secure communication*- is borrowed from cryptography. It enables performing Deep Learning tasks by actors knowing only their part of information. For example, if an edge server holds parameters of Deep Learning model, the edge device does not know them. And vice versa, the server does not have access to raw data held by the edge device. This feature makes it harder for an external actor to gain access to sensitive data.[4],[5]

5 Scheduling problem

As has already been mentioned, partitioning of a Deep Learning model and offloading its parts to the edge can be beneficial for many applications. Nevertheless it is also a highly nontrivial optimization problem since it influences all the KPIs mentioned in the previous section. Moreover, many IoT devices run more than one Deep Learning task concurrently. This kind of multi-tenancy results in task competing for the limited resources of the IoT device [5]. We consider such a scenario of model partitioning under condition of multi-tenancy in this section. In the following subsections first the framework of the problem is described. Afterwards the optimization problem is stated and an offline as well as an online solution algorithm are discussed.

5.1 Optimization problem

The article [3] considers an optimization problem with following framework: there is one cloud server, multiple edge servers and multiple Deep Learning tasks to be executed. Input data for each task is received by all edge servers, but in different volume, i.e. edge server i can receive more data for task j than edge server i'. The Deep Learning models for the tasks are to be divided into 2 parts: the first is to be executed on the edge and the second on the cloud server. If not enough capacity on the edge is available to run even smallest part of a task's Deep Learning model, this task is not deployed at the edge.

Therefore, the optimization problem is stated as follows.

Objection (1): assign as many tasks as possible to the edge servers.

Constraint(2): each edge server may offload to the cloud not more than it is allowed to send via its assigned effective bandwidth.

Constraint(3): if a task j is deployed at server i until its level k, then the remaining part of the processing takes place in the cloud. The authors assume that the major part of the total system response time is the data transfer latency. Therefore if it is within the predefined maximum latency, the Quality of Service is guaranteed.

Constraint(4): total computational overhead offloaded from an edge server to the cloud may not exceed the capacity assigned to this edge server.

Maximize:

$$\sum_{i=1}^{|E|}\sum_{j=1}^{|T|} X_{ij} \tag{1}$$

So that:

$$\sum_{j=1}^{|T|} b_{ij} \leq b_i * V \tag{2}$$

$$X_{ij} * d_{ij} * r_{kj} / b_{ij} \leq Q_j \tag{3}$$

$$\sum_{j=1}^{|T|} X_{ij} * l_{kj} * d_{ij} \leq c_i \tag{4}$$

For the list of symbols and their description see Table 1.

Symbol	Description
b_{ij}	bandwidth assigned for task t_j and edge server e_i
B_{max}, B_{min}	historical maximum and minimum required bandwidth of a task, respectively
b_i	network bandwidth provided by cloud server to edge server e_i
c_i	service capacity provided by the cloud server to edge server e_i
d_{ij}	input data size of task t_j in edge server e_i per time unit
$e_i \in E$	edge server i
E	set of all edge servers
e	mathematical constant
i_j^m	edge server with the highest input data size for task j
l_{kj}	computational overhead of a unit of input data after the kth layer of task t_j
N_j	number of task j$'$s deep learning network layers
Q_j	maximum allowed latency for task t_j
r_{kj}	average ratio of the intermediate data size generated by the kth layer ($k \in [1, N_j]$) to the total input data size
$t_j \in T$	deep learning task j
T	set of all deep learning tasks
$V < 1$	ratio of bandwidth spent on effective interaction between the cloud server and an edge server*
X_{ij}	1 if task j is deployed to edge server e_i, 0 otherwise

Table 1: Symbols

*in contrast to total available bandwidth, which is partially spent on background communication between servers

5.2 Solution algorithms

We will now consider two solution algorithms described the article [3]: an offline and an online ones. The former assumes that all information needed for decision making, such as precise set of tasks to be scheduled, are known in advance. It is therefore run before the actual start of the tasks, so that when they arrive, a ready schedule is available.

The offline algorithm is presented as pseudo code 1. As can be seen, it aims to deploy the heaviest parts (hence the starting value of layer k) of the biggest possible quantity of tasks (hence the sorting in ascending order) of the tasks to all edge servers. If for a task j

Algorithm 1 Offline scheduling algorithm

```
$k_j^{max} \leftarrow$ layer $k$ which maximizes the value of $r_{kj} * l_{kj}$
$ListOfTasks \leftarrow$ all tasks sorted in ascending order of the largest input data size
$k \leftarrow k_j^{max}$
for all tasks from $ListOfTasks$ do
  for all edge servers do
    if constraints (2)-(4) are not violated then
      continue
    else
      if there is another $k$ which has not been tried out then
        try another $k$
      else
        break
      end if
    end if
  end for
  deploy this task up to layer $k$ into all edge servers
end for
```

Algorithm 2 Online scheduling algorithm

```
for arrived task $j$ do
  $i_j^m \leftarrow$ the edge server with the highest input data size of task $j$
  calculate threshold: $\Phi_j \leftarrow (B_{min} * e / B_{max})^{c_{i_j^m}} * (B_{max} * e)$
  $k_j^{max} \leftarrow$ layer $k$ which maximizes the value of $r_{kj} * l_{kj}$
  $k \leftarrow k_j^{max}$
  if $(b_{i_j^m} - d_{i_j^m j} * r_{i_j^m j} / Q_j)) * (c_{i_j^m} - d_{i_j^m j} * l_{i_j^m j}) \leq \Phi_j$ then
    for all edge servers, except $i_j^m$ do
      if constraints (2) and (4) are not violated then
        continue
      else
        if there is another $k$ which has not been tried out then
          try another $k$
        else
          break
        end if
      end if
    end for
    deploy this task up to layer $k$ into all edge servers
  end if
end for
```

there is no layer k with which this task can be deployed at the edge, the next task j' from the sorted list is examined.

Due to quite strong assumptions of deterministic input parameters, this kind of algorithms is only suitable for stable systems, whose properties can be estimated in advance with high probability. If that is the case, an offline algorithm is preferable since it makes use of available knowledge and often delivers better results than online algorithms.

In contrast to offline solutions, online scheduling algorithms are based on the assumption, that the exact set of tasks to be scheduled is not known in advance. They therefore make a decision about each task as it arrives. The offline algorithm described as pseudo code 2 leverages historical data to estimate crucial parameters. Please note that in this case c_i stands for remaining (and not for total as in Algorithm1) free capacity provided by cloud to the edge server i. The algorithm makes its scheduling decisions based on a threshold value, which has been chosen experimentally.

5.3 Performance evaluation

Performance of both algorithms described in the previous section has been evaluated by means of simulation. In both cases the number of deep learning tasks has been set to 1000. The input data size of each task ranges from 100 kB to 1 MB. The number of layers of all CNN networks is set from 5 to 10. The bandwidth of each edge server is uniformly distributed from 10 Mb/s to 1 Gb/s. The required latency is set to 0.2 s.

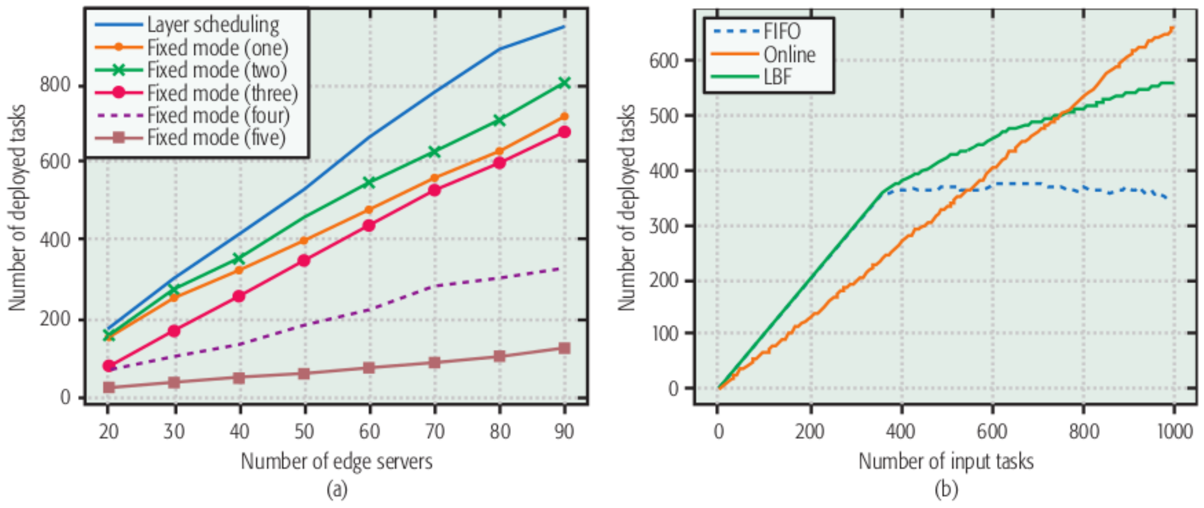

Figure 3: Performance evaluation

For the evaluation of the offline algorithm (see Figure 3 (a)) scenarios with varying number of edge servers have been considered (from 20 to 90, see x-axis). For those, the performance of the offline algorithm, here called "Layer scheduling", has been compared to 5 other algorithms. Their strategy is simple: they deploy fixed number of layers (here from 1 till 5) of as many tasks as possible to the edge. The results of the simulation show that the "Layer scheduling" algorithm outperforms the fixed approaches. For this particular case the second best solution is reached by deployment of 2 layers to the edge (green line), so it can be concluded, that "Layer scheduling" has also deployed most of the tasks up to the second layer, but for others a better strategy was found.

The results of the simulation of the online algorithm can be found on Figure 3 (b). Its performance is compared to the ones of FIFO (first in fist out) and LBF (low bandwidth first) algorithms. FIFO simply deploys tasks as long as enough capacity and bandwidth is available and then pops out the first deployed tasks for appending the following ones. LBF behaves similarly as long as enough capacity is available, but then removes the task with maximum bandwidth requirement in favour of newcomers. Thanks to the more sophisticated criterion for choice of tasks, the online algorithm outperforms both FIFO and LBF for higher number of tasks (approx. 600 and 800 respectively) [3].

6 Conclusions

In this paper we have provided background of Deep Learning in the context of the Internet of Things. We have considered the state-of-the-art approaches for applying Deep Learning models and discussed the current trends. We introduced the trade-offs and the issues arising in this context. Then we investigated in depth one exemplary optimization problem: layer-wise partitioning of Deep Learning models of various tasks between Edge servers and Cloud server. The objective was to maximize the number of tasks, which are partially performed in the edge, while meeting the latency, bandwidth and capacity constraints. For this optimization problem we discussed 2 solution methods, an offline and an online algorithms. We also mentioned their application fields: offline algorithms are suitable for systems, where the set of tasks is known in advance. Otherwise the online algorithm should be applied. At the end we provided performance evaluation of both algorithms. In the simulation they proved to outperform several widespread alternative scheduling methods.

The limitations of the suggested solutions lay in the assumptions of the optimization model. For example, if for an application it is not the case, that the transfer latency is the major part of the system response time, then the constraint (3) is too soft and will not guarantee the Quality of Service. It will then not consider e.g. the processing time in the edge and in the cloud. Besides that, it is not mentioned by the authors, what happens if even a smallest part of a task cannot be deployed to the edge. If the task is then completely performed in the cloud, then the data for it must be sent there. This would consume bandwidth, which is not accounted for in the optimization model. Therefore, several adjustments of the model might be needed for particular application cases.

The future steps will be to gain more insights into the Deep Learning in the context of the Internet of Things and continue working in this direction.

List of Tables

List of Figures

References

[1] X. Ma, T. Yao, M. Hu, Y. Dong, W. Liu, F. Wang, and J. Liu, "A survey on deep learning empowered iot applications," *IEEE Access*, vol. 7, pp. 181721–181732, 2019.

[2] A. R. et al., "Internet of things strategic research roadmap," *European Research Cluster on the Internet of Things*, 2009.

[3] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[4] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *CoRR*, vol. abs/1905.10083, 2019.

[6] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog computing for the internet of things: A survey," *ACM Transactions on Internet Technology*, vol. 19, 2019.