Meta-Learning-Based Deep Learning Model Deployment Scheme for Edge Caching

Kyi Thar[1], Thant Zin Oo[1], Zhu Han[1,2], Choong Seon Hong[1],
[1] Dept. of Computer Science and Engineering, Kyung Hee University, Korea.
[2] Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204-4005, USA.
E-mail:{kyithar, tzoo, cshong}@khu.ac.kr, zhan2@mail.uh.edu.

***Abstract*—Recently, with big data and high computing power, deep learning models have achieved high accuracy in prediction problems. However, the challenging issues of utilizing deep learning into the content's popularity prediction remains open. The first issue is how to pick the best-suited neural network architecture among the numerous types of deep learning architectures (e.g., Feed-forward Neural Networks, Recurrent Neural Networks, etc.). The second issue is how to optimize the hyperparameters (e.g., number of hidden layers, neurons, etc.) of the chosen neural network. Therefore, we propose the reinforcement (Q-Learning) meta-learning based deep learning model deployment scheme to construct the best-suited model for predicting content's popularity autonomously. Also, we added the feedback mechanism to update the Q-Table whenever the base station calibrates the model to find out more appropriate prediction model. The experiment results show that the proposed scheme outperforms existing algorithms in many key performance indicators, especially in content hit probability and access delay.**

***Index Terms*—Autonomous deep learning model generation, meta-learning, edge caching, content's popularity prediction.**

I. Introduction

According to Cisco, viewing videos on wireless devices has significantly increased Internet traffic and is expected to continue to increase exponentially [1]. Expanding the capacity of network infrastructure to handle the exponential increase of user demand is economically costly and unrealistic. Thus, several future Internet network architectures (such as Named Data Networking) have been proposed with in-network caching capability to handle the increasing Internet traffic. Accordingly, caching enabled edge nodes such as base-stations can temporarily store video contents in their cache storage to preserve user requests in the future, rather than re-downloading the same contents from the original content servers.

Basically, caching schemes can be classified into i) reactive, and ii) proactive caching. In *reactive caching*, the edge node performs the cache decision (to store the content at the cache or not) based on the content's popularity[1] only when the request for a particular content arrives [2]. In the *proactive caching*, the edge node predicts a content's popularity before any user's request and make the cache decision [3], [4]. In this paper, we apply the proactive caching scheme because we would like to reduce the backhaul traffic by using unutilized bandwidth at the off-peak hours and the performance of this caching scheme is directly related with the content's popularity prediction accuracy. Depending on the assumption such as popularity follows Zipf distribution, many researchers proposed various edge-network cache decision schemes[2] [5], [6]. In practice, the popularity of the content is dynamically changing depending on different factors (e.g., events, type of content, and the lifespan of the content). Hence, the content's popularity prediction becomes one of the most challenging matters to design effective proactive caching scheme.

Recently, deep learning has been utilized in numerous areas, such as object detection. Due to its high performance in prediction, deep learning based prediction models are utilized in content's popularity prediction to gain more precise results [4], [7]–[10]. Stack Auto Encoder, has been used in [8] to predict the content's popularity in a software-defined networking environment. The utilization of echo state network to support cache decision was presented in [4] and the neural network with several hidden layers was presented in [7]. In addition, the utilization of the Convolutional Neural Networks (CNNs) to track the context to support caching decision was shown in [9].

Constructing an appropriate deep learning model is complicated and time-consuming process even for the human experts. This is because human experts need to find out the relevant deep learning architectures, training procedures, and hyperparameters to solve the domain-specific problem with satisfactory performance. Thus, we need an autonomous solution to search the appropriate deep learning architecture for domain-specific tasks (i.e., find the appropriate model among Feedforward Neural Networks (FNNs), Recurrent Neural Networks (RNNs), etc.) as well as autonomously manage the hyperparameters optimization (i.e., find the number of appropriate hidden layers, number of neurons, etc.) [11], [12].

This work was partially supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea goverment(MSIT) (No.2019-0-01287, Evolvable Deep Learning Model Generation Platform for Edge Computing) and US MURI AFOSR MURI 18RT0073 , NSF CNS-1717454, CNS- 1731424, CNS-1702850, CNS-1646607. *Dr. CS Hong is the corresponding author.

[1]Content's popularity can be defined as the proportion of the number of requests for a particular content to the total number of requests from users, ordinarily obtained for a specific region through a given period.

[2]The process to decide whether to store the new content or remove the cached content

978-3-903176-24-9

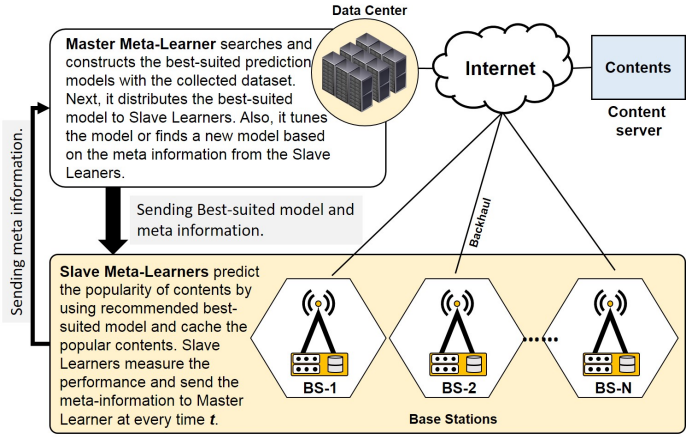


Fig. 1: System Model

A. Our Methodology

Thus, in this paper, we proposed "a meta-learning-based autonomous deep-learning model deployment scheme," to support intelligence edge caching. Previously, we applied random-search and reinforcement learning methods to construct the deep learning models, [11], [12]. But, random-search method can only explore in a random direction to find the best deep learning model within possible configurations [11]. Also, reinforcement learning used in [12] has lack of feedback mechanism from the BSs to search and construct the better model than the currently used model. So, in this paper, we applied meta-learning with reinforcement learning to solve the aforementioned problem of [12]. Also, we enhance the searching space design and model generation rules from [12] to be able to improve the searching performance. Meta-learning (learning to learn) is the approach to create learning models that can learn new abilities or acclimate to new circumstances rapidly from experience (meta-data).

Our contributions are summarized as follows:

- We propose the system design to jointly work with reinforcement (Q-Learning) meta-learning based deep neural network deployment scheme with mobile edge computing architecture. Our proposed scheme includes two learner i) master meta-learner (MML), and ii) slave meta-learner (SML). The MML construct the potential deep learning models and SMLs utilize the best-suited model among constructed model to predict content's popularity.
- We formulate the optimization problem to minimize the content retrieval cost from the original content server.

II. System Model

Fig. 1 shows the proposed system model, in which the MML is located at the cloud data center, and each SML is located at BS. Since finding the best-suited deep learning model requires high computing power, we deploy the MML at the cloud data center. After getting the best-suited model, the MML distributes the best-suited model to each SML (each BS). Then, the SML uses the best-suited model and predicts the content's popularity. Also, SML tunes the best-suited model with the locally collected data (e.g., every 6 hours, every night). If the best-suited model does not reach to the absolute accuracy, even after the model tuning process, the SML send the meta-information to the MML. Then, based on the meta-information, MML explores the new model.

A. Network Model

Let a set of base stations is denoted as $\mathcal{B} = \{1, 2, \ldots, B\}$ where each BS $b \in \mathcal{B}$ has a limited cache capacity and it can be denoted as S_b. Let the set of contents be denoted by $\mathcal{F} = \{1, 2, \ldots, F\}$ and the size of each content f is denoted as κ_f. Let us introduce the binary decision variable $x_{b,f}^{(t)}$ to do the cache decision. $x_{b,f}^{(t)} = 1$ when the content f is cached at the local cache of the BS $b \in \mathcal{B}$ at time (t), and $x_{b,f}^{(t)} = 0$ otherwise. Thus, at each time slot (t), the cache capacity of S_b is limited by

$$\sum_{f \in \mathcal{F}} x_{b,f}^{(t)} \kappa_f \leq S_b, \quad x_{b,f}^{(t)} \in \{0, 1\}, \ \forall b \in \mathcal{B}, \ \forall t. \tag{1}$$

B. Cost Model

For each arriving request $r_k^b \in \mathcal{R}_b$ from users at BS $b \in \mathcal{B}$, BS first checks whether the requested content is located in its local cache or not, and the missing contents are retrieved from the content server. Here, we use a proactive caching scheme where cache decision to store content f for time $(t+1)$ is made at the end of time (t) based on the predicted content's popularity. Then, the actual request is arriving at time $(t+1)$, and the content retrieval cost can be calculated as follows,

$$c_{b,\text{cache}}^{(t+1)} = \sum_{f \in \mathcal{F}} (1 - x_{b,f}^{(t+1)}) \, \phi_{b,f}^{(t+1)} \kappa_f \, p_b^{\text{retrieve}}, \quad \forall b \in \mathcal{B}, \tag{2}$$

where $x_{b,f}^{(t+1)}$ is the cache decision variable which is decided at the end of time (t) (e.g., every midnight), $\phi_{b,f}^{(t+1)}$ is the request counts of content f at BS b for $(t+1)$ and p_b^{retrieve} is the unit retrieval cost or backhaul usage for each content f at BS b.

III. Problem Formulation

Then, we formulate the simple content retrieval cost minimization problem as follows:

$$\begin{aligned} \underset{x_{b,f}^{(t+1)}}{\text{minimize:}} \quad & \frac{1}{T} \sum_{t \in \mathcal{T}} \sum_{b \in \mathcal{B}} c_{b,\text{cache}}^{(t+1)} \\ \text{subject to:} \quad & (1). \end{aligned} \tag{3}$$

where the objective function is minimize the content retrieval cost by controlling the cache decision variable $x_{b,f}^{(t+1)}$. The cache capacity is limited by (1). Moreover, the future request counts of each content $\phi_{b,f}^{(t+1)}$ are not known in advance and the optimization problem (3) is a combinatorial problem. Hence, exhaustive search is not feasible because of a large number of configuration combinations. These reasons claim the optimization problem in (3) challenging to solve in the presence of limited information. Hence, to solve (3), we need

TABLE I: Table of notations.

Notation	Description
System Notations	
$\mathcal{B}$, B, b	Set, number, element of BSs
$\mathcal{F}$, F, f	Set, number, element of contents
S_b	Physical cache size at BS b
κ_f	Size of content f
p_b^{retrieve}	Unit backhaul usage price of BS b
$\phi_f^{(t)}$, $\hat{\phi}_f^{(t)}$	Real and predicted request counts of movie f at time (t)
$x_{b,f}^t$	Cache decision variable for content f at BS b and time (t)
Machine Learning Notations	
m	Prediction model
λ_m, γ_m	Learning rate and regularization rate of model m
l_m^{train}, l_m^{valid}	Training and validation loss of model m
l_m^{pred}	Prediction loss of model m
$\sigma, \omega, \theta, \epsilon$	Reward, ratio of immediate reward, state (Q-Learning's Parameters) and loss threshold

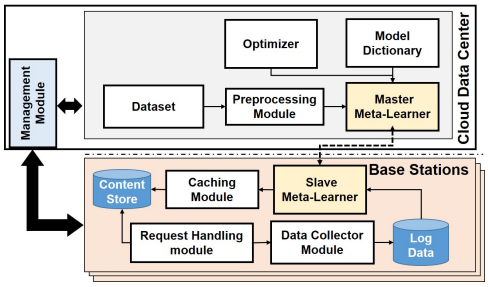


Fig. 2: Components for meta-learning based deep-learning model deployment platform for content' popularity prediction

a three-step solution: i) searching the best-suited model to predict the content's popularity, ii) predicting the popularity of each content f, $\hat{\phi}_{b,f}^{(t+1)}$, and iii) caching the popular content based on predicted results $\hat{\phi}_{b,f}^{(t+1)}$.

IV. Implementation Process

The details discussion for the implementation process of meta-learning based autonomous deep learning model deployment platform are as follows.

A. Components needed to implement meta-learning

Master Node: consists of five main components, i) management module, ii) preprocessing module, iii) optimizer, iv) model dictionary and iv) MML. The management module manages model deployment and tracking the performance of the master and slave nodes. The preprocessing module is responsible for cleaning, extracting the log files to construct the dataset to train the models. The optimizer module provides various type of optimization scheme such as Stochastic Gradient Descent (SGD) for training process. The Model Dictionary stores the various deep learning architectures such as CNN and RNN, which are discussed in details in Sec. IV-B. The MML explores, constructs and selects the best-suited popularity prediction model based on collected data $\mathcal{D} \triangleq \{d^t | t = 1, 2, ..., t\}$, where (t) is time when the data is collected.

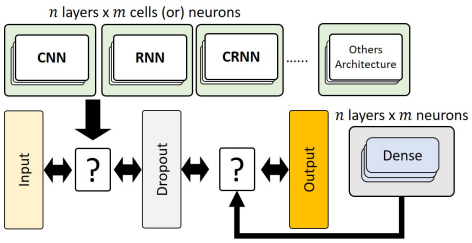


Fig. 3: Generic Deep Learning Model Framework for different architectures

Slave Node: consists of i) SML, ii) Caching Module, iii) Preprocessing module, iv) Data Collector module, and v) Request Handling Module. The SML manages the model calibration, and model replacement based on the performance of the current utilizing model and the new model send from the MML. Also, SML provides the predicted popularity scores to the caching module to make proactive cache decision. The Data Collector gathers information such as the number of cache hits and the total number of requests. The Caching module makes the cache decision and downloads contents from a server based on the predicted popularity scores. When the request arrives at the BS, the Request Handler checks whether the requested content is located in its Content Store (cache storage). If the requested content is in the cache, then the Request Handler Module provides the content to the user. Otherwise, the Request Handler downloads the content directly from the servers.

B. Generic Deep Learning Model Framework

In this section, we focus on the generic deep learning model framework. The set of content's popularity prediction models can be denoted as $\mathcal{M}$. Each model $m \in \mathcal{M}$ represents various types of prediction models such as RNNs. Among them, we choose the best-suited model $\boldsymbol{m}^* = \{(m_{\text{req}}) | m_{\text{req}}^* \in \mathcal{M}_{\text{req}}\}$, which has the lowest validation loss.

Fig. 3 shows the deep learning model framework to construct the potential models to find the best-suited model. The basic framework includes input layers, unknown architecture layer, unknown dense layers, dropout layer, and the final output layer. The *input layer* is the initial point to feed the features information for training and prediction processes. The *unknown architecture layer* arrangement determines the architecture of the deep learning model, which can be FFNs, CNNs, RNNs or Convolutional Recurrent Neural Networks (CRNNs). The Model Dictionary stores the generic deep learning framework and the rules to construct the potential deep learning models among various deep learning architectures.

If the unknown layer is searching for CNNs [13], we tune number of the *Convolutional layers* and *pooling layer*. The *Convolutional layers* utilize a convolution operation to the input and pass the results to the next layer. The *pooling layer*

Algorithm 1 Finding best-suited popularity prediction model

Input: $\mathcal{D}_s$, $\mathcal{M}_{\text{arc}}$, M, $\mu_{\min}$, $\mu_{\max}$, $\nu_{\min}$, $\nu_{\max}$,
Output: Predicted popularity scores, $\hat{\phi}_f^b$

1: **Initialization:** l_m^{train}, $l_m^{\text{valid}} = 0$, $\lambda_m, \gamma_m = 10^{-4}$
2: **`Master Meta-Learner (MML)`**
3: **for** k in range(M) **do**
4: **if** rand(0,1) $> p_k$ **then** ▷ **Exploration**
5: model ← rand_choice($\mathcal{M}_{\text{arc}}$)
6: layer ← rand_choice(layer_list($\mu_{\min}$, $\mu_{\max}$))
7: layer_config ← rand_choice(config(layer))
8: m ← rand_choice(model_config((model, layer_config, $\nu_{\min}$, $\nu_{\max}$)))
9: **else** ▷ **Exploitation**
10: model, layer, layer_config ← min_Q(QT-model, QT-layer, QT-config)
11: m ← rand_choice(model_config((model, layer_config, $\nu_{\min}$, $\nu_{\max}$)))
12: **end if** ▷ **Updating Q-values**
13: Train and validate m with $\mathcal{D}_s$ to get the reward $\sigma \leftarrow l_m^{\text{valid}}$
14: Update QT-config, QT-layer, and QT-model with (4)
15: **end for** ▷ **Model Deployment**
16: Choose and distribute the model $m^* = \arg\min(l_m^{\text{valid}})$ to Slave Meta-Learner
17: **`Slave Meta-Learner (SML)`**
18: Predict the popularity with model m^* and send the results $\hat{\phi}_f^b$ to the Caching Module ▷ **Model Calibration**
19: **if** $l_{m^*}^{\text{pred}} > \varepsilon$ **then**
20: Minimize prediction loss with locally collected data
21: **if** current $l_{m^*}^{\text{pred}}$ is lower than the previous **then**
22: Update and store the parameters of m^*
23: **end if**
24: **end if**
25: Send reward σ to MML at whenever update the local model to update the Q-Values (line 14 to 18).

performs a down-sampling operation along the spatial dimensions. The most common types of RNNs [14] are Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM). We choose LSTM because neural networks using LSTM cells have offered better performance than standard recurrent units. CRNN [15] architecture results from combining CNN and RNN. As such, we can configure the Convolutional layer, Polling layer, LSTM to obtain the best-suited model. Next, the *dense layer* is added to extract the features information for the final prediction. The *dropout layer* is added to prevent the over-fitting problem. Finally, the *final output layer* is designed based on the prediction problem (e.g., if the problem is the classification problem, the final output layer will be constructed with the softmax activation function).

C. Reinforcement Meta-Learning

1) Applying Q-learning for the Master Meta-Learner: Among the various reinforcement learnings, we apply Q-learning [16] to search the best-suited model. Firstly, we define the set of states $\theta = \{1, 2, \ldots, \theta\}$, which represents architecture types, layers, cells, neurons, filters information in the hierarchical structure to construct search space for finding the potential models. Secondly, we denote the validation loss σ_m of each model m as the reward of the model m, where our main objective is to choose the model m with minimum validation loss. Thirdly, we define the set of actions $\mathcal{A} = \{1, 2, \ldots, a\}$, which represents the way to construct the model m in each round. Fourthly, we modify the hierarchical Q-table from [12] to stores Q-values in hierarchical manner instead of storing as in normal Q-table. Then, we update the Q-values of hierarchical Q-table as follows,

$$\begin{aligned} Q(\theta^{(t+1)}, a^{(t+1)}) \leftarrow & Q(\theta^{(t)}, a^{(t)}) \\ & + \alpha[\sigma^{(t+1)} + \omega \min_a Q(\theta^{(t+1)}, a) \\ & - Q(\theta^{(t)}, a^{(t)})], \end{aligned} \quad (4)$$

where t is the current time step, $\sigma^{(t+1)}$ is the immediate reward (validation loss) received at time $(t+1)$, $\theta^{(t)}$ is the state at time t, a^t is the action performed at time (t). α is the learning parameter within the range $0 \leq \alpha \leq 1$. ω controls the amount of immediate reward to return form future states.

2) Deep Learning Model Deployment Scheme: To find out the potential deep learning models and to tune the initial hyper-parameter settings of deep learning models, we randomly extract a small portion of the data from the data set to create a small data set. Let $\mathcal{D}_s = d^{[t=1:t=j]}$ be the small data set and $\mathcal{D}_s \subseteq \mathcal{D}$. Alg. 1 shows the way to construct the potential deep learning models at the MML and the way to deploy the best-suited model to predict the content's popularity for SML. The input parameters for the algorithm Alg. 1 are: i) Features and target labels of the small dataset $\mathcal{D}_s$, ii) Parameters to choose the minimum and maximum number of layers ($\mu_{\min}$ and $\mu_{\max}$), iii) Parameters to choose the minimum and maximum number of cells or neurons in each layer ($\nu_{\min}$ and $\nu_{\max}$), iv) Number of models to be searched M, and v) Model architectures $\mathcal{M}_{\text{arc}}$ to be explored (i.e., CNN). The output of Alg. 1 is the predicted popularity scores. We initialized the model training loss $l_m^{\text{train}} = 0$, model validation loss $l_m^{\text{valid}} = 0$, learning rate $\lambda_m = 10^{-4}$ and regularization rate $\gamma_m = 10^{-4}$.

Alg. 1 runs on two separated parts: MML and SML. MML construct the potential deep learning models by using two phase: i) *exploration* phase (from line (4) to (8)) and *exploitation* phase (from line 9 to line 11). The MML choose the *exploration* or *exploitation* phase based on the threshold p_k, where p_k parameter maintains the *exploration–exploitation* balance. The MML performs more *exploration* in the initial stage and more *exploitation* in the later stage.

The *exploration* phase includes the following steps as depicted in Alg. 1: Randomly select the model among the various types of deep learning model such as CNNs (line 5). Randomly select the number of layers for each model (line 6). Randomly select the layer's configuration (eg., [1,5] means 1 CNN layer and 5 Dense layer) (line 3). Randomly choose the neuron/cell configuration (eg., [5],[1,2] means 5 CNN stacks at the CNN

Algorithm 2 Cache decision process at slave node

Input: Predicted content's popularity scores $\hat{\phi}_{b,f}^{(t+1)}$
Output: Cache the popular contents for $(t+1)$

1: Sort contents in descending ordered based on predicted content's popularity scores $\hat{\phi}_{b,f}^{(t+1)}$
2: Based on the content list, store the contents from the beginning until the cache space $\hat{s}_b^{(t+1)}$ is full

layer, 1 neuron at first dense layer, and 2 neuron at second dense layer) (line 4).

The *exploitation* phase includes the following steps: Choose the model which has the minimum Q-value (line 10). Choose the total number of layers, which has a minimum of Q-values (line 11). Choose the layer's configuration, which has the minimum Q-values (line 11). Randomly choose the neuron/cell configuration (eg., [5],[2,1] means 5 CNN stacks at the CNN layer, 2 neuron at first dense layer, and 1 neuron at second dense layer) (line 11).

The Q-values *updating* process which includes the following steps: Train the chosen model with the small dataset(e.g., one-month worth data) and get the reward (e.g., validation loss) (line 8 and 11). Update the QT-values of QT-config table, QT-layer table and QT-model table with their individual rewards, respectively (line 16 to 18). Then, model deployment module of the MML selects the best-suited model which has the minimum validation loss value (line 19), i.e., $m^* = \arg\min(l_m^{\text{valid}})$ and distributes to the SML.

The SML utilize the model m^* to predict the content popularity scores $\hat{\phi}$ at every time t (e.g., every 6 hours or 12 hours) and send the results to the caching module to make cache decision (line 22). Also, the Performance Checking module from SML checks the performance of the prediction model (line 23). SML calibrates the model with locally collected data when the performance of the model is lower than the threshold. At the same time, SML feedbacks the reward σ to the MML to update the Q-Table to improve the model construction scheme (line 27).

D. Caching Decision

Alg. 2 explains the cache decision process at the BS to improve the cache hit. The inputs for this algorithm is predicted content's popularity scores $\hat{\phi}_{b,f}^{(t+1)}$. The output of this algorithm is the decision on whether to store the expected popular contents. After receiving the predicted content's popularity scores $\hat{\phi}_{b,f}^{(t+1)}$, the Slave Node (BS) sorts contents in descending order based on the predicted content's popularity scores (line 1). Then, the *Slave Node* at BS b retrieves and stores contents in descending order until its cache space is full (line 2).

V. Performance Evaluation

We run our experiments on a PC with the followings configurations; CPU: Intel core i7-7700k, RAM: 32GB, graphics card: GeForce GTX 1080 Ti, and operating system: Ubuntu 16.04 LTS. We used pandas [17] to preprocess the dataset, which is an open source data analysis tool. We implement our proposed scheme using the GPU version of Tensorflow 1.4 [18] as a backend and Keras [19] as a frontend.

TABLE II: Parameters used in the experiments

Parameters	Value
Learning and regularization rate	10^{-3}
num_models	100
Window size	12
Neighbors information	6
Features	2
Output size	1
Output activation function	Relu
Batch Size	500
Training data	$2012 - 2015$
Validation data	2016
Loss function	Mean Absolute Percentage Error
Optimizer	ADAM
Cache size	2% to 10%

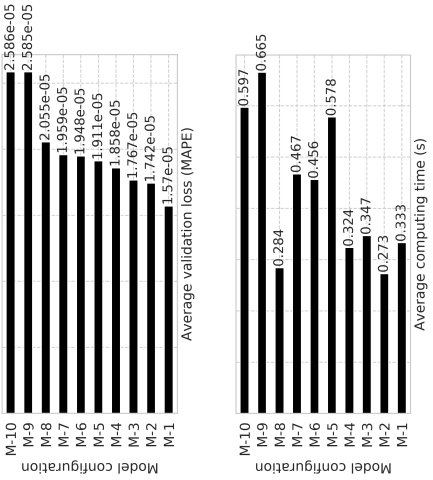
For the training and validation, we choose Movielens 1M [20] dataset, which includes 6040 users and 1,000,209 ratings over 3706 movies. We assume that the rating counts in the dataset are request counts of the movies and that the times when users rate movies are the same as the request arrival times for those movies. The details configurations to train and test the models are presented in Table. II.

We find the 100 constructed appropriate deep learning model configurations, but because of page limitation, we only show the validation accuracy and computation time of the best 10 out of 100 randomly constructed deep learning models in Fig. 4. The best model is the M1 which has the best validation accuracy and the configuration as follows: LSTM $\langle 73, 73 \rangle$, Dense $\langle 176, 106, 64, 143 \rangle$, (i.e., 73 cells in layer-1&2 of LSTM and 176 cells, 106 cells, 64 cells, and 143 cells, respectively, in layers 1,2,3 and 4 of dense layers). We choose the *probability of hit* as in [12] and, *content retrieving cost* (objective function), as Key Performance Indicator (KPI) metrics.

A. Results and discussion

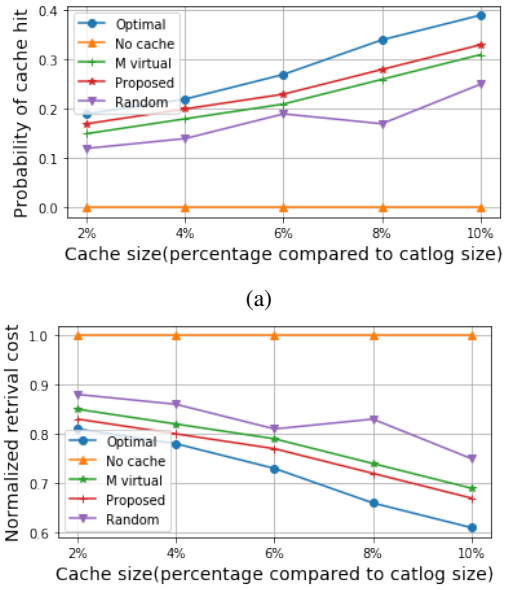
1) Benchmark: As we hold the full data set of all requested contents, we can get the upper and lower bounds to act as benchmarks to measure the proposed scheme performance. The *optimal scheme* scheme is set as the upper bound in which BSs cache the contents which possibly get the most requests in next day. As for the lower bound, we use the *no cache scheme*. In the *no cache scheme*, we deploy no cache in the network, and the contents are directly downloaded from the content store. Furthermore, we define the baseline scheme as the *randomized caching scheme*, where we randomly cache the contents. Also, we compared our proposed scheme with the existing work (M virtual {the best model}) from the [12].

2) Probability of cache hit: Fig. 5(b) shows the cache hit probability where a higher probability of hit means better the caching performance as well as lower the backhaul usage. As the increment in cache size, the cache hit probability also increases due to the bigger storage capacity to store more



(a) (b)

Fig. 4: Popularity prediction performance comparison for the best 10 out of 100 deep learning model configurations (a) average validation loss, and (b) average computing time.



(a)

(b)

Fig. 5: Caching related performance comparisons: (a) Cache hit, and (b) Content retrieval cost comparison.

contents. The results from Fig. 5(b) shows that learning model M1, created by our proposed scheme improves 16% of the cache hit on average compared to other schemes.

3) Content retrieval cost: Fig. 5(c) shows the content retrieval cost comparison where the lower the value, the better the performance. The results in Fig. 5(c) shows that the content retrieval cost of model M1, constructed by our proposed scheme reduces the network traffic by 16% on average compared to other schemes. Hence, Fig. 5(c) validates that our algorithm can predict the most popular contents with high accuracy.

VI. Conclusion

In this paper, we proposed the reinforcement meta-learning based deep learning model deployment scheme for edge caching to provide the appropriate content's popularity prediction deep learning models autonomously. We developed the proposed scheme by using Tensorflow and train the model with MoveLens dataset. As for the future work, we are going to work on generalized metal-learning based deep learning model deployment scheme for various types of the domain-specific prediction problem.

References

[1] Accessed: Feb. 7, 2019. [Online]. Available: http://www.cisco.com

[2] S. Li, J. Xu, M. van der Schaar, and W. Li, "Trend-aware video caching through online learning," *IEEE Transactions on Multimedia*, vol. 18, no. 12, pp. 2503–2516, 2016.

[3] E. Zeydan, E. Bastug, M. Bennis, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data caching for networking: Moving from cloud to edge," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 36–42, 2016.

[4] M. Chen, W. Saad, C. Yin, and M. Debbah, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3520–3535, 2017.

[5] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[6] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: Design aspects, challenges, and future directions," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, 2016.

[7] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 28–35, June 2018.

[8] W. Liu, J. Zhang, Z. Liang, L. Peng, and J. Cai, "Content popularity prediction and caching for ICN: A deep learning approach with SDN," *IEEE Access*, vol. 6, pp. 5075–5089, 2018.

[9] K. C. Tsai, L. Wang, and Z. Han, "Mobile social media networks caching with convolutional neural network," in *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, April, pp. 83–88, April 2018.

[10] R. Devooght and H. Bersini, "Collaborative filtering with recurrent neural networks," *arXiv preprint arXiv:1608.07400*, 2016.

[11] K. Thar, N. H. Tran, T. Z. Oo, and C. S. Hong, "Deepmec: Mobile edge caching using deep learning," *IEEE Access*, vol. 6, pp. 78 260–78 275, Dec 2018.

[12] K. Thar, T. Z. Oo, Y. K. Tun, D. H. Kim, K. T. Kim, and C. S. Hong, "A deep learning model generation framework for virtualized multi-access edge cache management," *IEEE Access*, vol. 7, pp. 62 734–62 749, 2019.

[13] Y. Goldberg, "Neural network methods for natural language processing," *Synthesis Lectures on Human Language Technologies*, vol. 10, no. 1, pp. 1–309, April 2017.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov 1997.

[15] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Convolutional recurrent neural networks for music classification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2392–2396, March 2017.

[16] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[17] Accessed: Feb. 7, 2019. [Online]. Available: https://pandas.pydata.org/

[18] Accessed: Feb. 7, 2019. [Online]. Available: https://www.tensorflow.org/.

[19] Accessed: Feb. 7, 2019. [Online]. Available: https://keras.io/

[20] Accessed: Feb. 7, 2019. [Online]. Available: https://grouplens.org/datasets/movielens/latest/