Autoencoders for QoS Prediction at the Edge

Gary White, Andrei Palade, Christian Cabrera, Siobhán Clarke
School of Computer Science and Statistics,
Trinity College Dublin
Dublin, Ireland
{whiteg5, paladea, cabrerac, siobhan.clarke}@scss.tcd.ie

***Abstract*—In service-oriented architectures, collaborative filtering is a key technique for service recommendation based on QoS prediction. Matrix factorisation has emerged as one of the main approaches for collaborative filtering as it can handle sparse matrices and produces good prediction accuracy. However, this process is resource-intensive and training must take place in the cloud, which can lead to a number of issues for user privacy and being able to update the model with new QoS information. Due to the time-varying nature of QoS it is essential to update the QoS prediction model to ensure that it is using the most recent values to maintain prediction accuracy. The request time, which is the time for a middleware to submit a user's information and receive QoS metrics for a candidate services is also important due to the limited time during dynamic service adaptations to choose suitable replacement services. In this paper we propose a stacked autoencoder with dropout on a deep edge architecture and show how this can be used to reduce training and request time compared to traditional matrix factorisation algorithms, while maintaining predictive accuracy. To evaluate the accuracy of the algorithms we compare the actual and predicted QoS values using standard error metrics such as MAE and RMSE. In addition, we propose an alternative evaluation technique using the predictions as part of a service composition and measuring the impact that the predictions have on the response time and throughput of the final composition. This more clearly shows the direct impact that these algorithms will have in practice.**

***Index Terms*—QoS, IoT, Edge, Deep Learning, Reliability**

I. Introduction

The success of the Internet of Things (IoT) has created a need for edge computing, where processing occurs in part at the network edge, rather than completely in the cloud. The number of devices connected to the IoT is predicted to grow at an exponential rate, with latest forecasts predicting that there will be 26 billion connected devices by 2020 [1]. The management of the services provided by these devices is recognised as one of the most important areas of future technology and has gained widespread attention from research institutes and industry in a number of different domains, including transportation, industrial automation and emergency response [2]. The ability to manage services in these domains relies on having fast and accurate estimates for the QoS of candidate services available at the edge and an adaptive middleware capable of responding to failures in currently executing services. Edge computing can address a number of traditional IoT concerns such as latency, mobile device battery life, bandwidth costs security and privacy. The aggregation of QoS values from other users and services allows the collaborative filtering algorithm to make an accurate prediction of the possible QoS values this service will have when invoked, without directly invoking it [3], [4]. This reduces the load on the network as the service composition engine does not have to make additional invocations to retrieve QoS values.

Recommending services based on QoS has attracted a huge amount of interest since the initial work proposing matrix factorisation [5] and the collecting of a comprehensive dataset to evaluate extensions of the original algorithm [6]. Since then, a number of extensions have been proposed to the original approach [7], [8], [9]. Many of these approaches have focussed on extending matrix factorisation-based collaborative filtering by taking into account alternative sources of information such as time [10] or combining content-based features [11] and taking into account different service providers in the IoT [9]. There are also approaches that take into account additional factors such as the location of users in the environment [12].

The training and invocation time of matrix factorisation-based algorithms is something not mentioned in the state of the art with the overwhelming focus being on prediction accuracy (MAE/RMSE between the predicted and actual matrix). The resource intensiveness of generating latent features in the matrix factorisation process can make it difficult to deploy these algorithms at the edge of the network, where device resources may be constrained. Unlike traditional collaborative filtering applications such as film recommendations where ratings from users do not often change over time, QoS factors such as response time and throughput can be volatile and vary with time. These algorithms need to be retrained frequently to deal with new values being reported by users. There is also a need to reduce the invocation time of the algorithms to ensure that users get quick responses and a middleware has access to the most recent QoS values to make the best possible service composition. The reduction in invocation time also helps with dynamic service recomposition allowing a composition engine to choose a suitable replacement service before the application fails. In our previous work [9], we illustrated a means to increase prediction accuracy for QoS values. In this paper, we reduce the training time for QoS predictions, which allows us to analyse the QoS for users in a highly dynamic environment that would otherwise not be possible in sufficient time to influence a service composition process. We use a stacked autoencoder to achieve this, which we have designed to be executed on edge devices.

The same dataset is commonly used to evaluate new matrix factorisation approaches for QoS prediction and comes from

978-1-5386-9148-9/19/$31.00 ©2019 IEEE

WS-DREAM [6]. While this allows for comparison between the accuracy of the algorithms, it still remains unclear what the implication of this is, when the algorithm is used in an actual service composition [4]. If an algorithm has a RMSE of 0.25 less than another on the Throughput dataset, how will this affect the actual service composition? Will it be 25% better? 50% better? The evaluation of service recommender systems on one dataset is good for repeatability, however there is a trade-off to be made between offline, online and user studies in recommender systems [13]. In our experiments, we consider the direct prediction accuracy using standard error metrics to allow us to compare against the state of the art approaches. We also use the predictions as input to a service composition algorithm to show the direct impact on the final application composition. In this context, an application composition is the result of composing individual services. We publicly release our prediction and service composition algorithms to allow future QoS prediction approaches to evaluate themselves against state of the art approaches in a repeatable manner (https://goo.gl/rNz8ia).

The remainder of the paper is organised as follows: Section II outlines the related work, Section III describes the middleware architecture and deployment devices to allow QoS predictions for currently executing services and candidate services to take place at the edge of the network and increase the reliability of IoT applications. Section IV describes existing collaborative filtering approaches and our proposed stacked autoencoder approach with dropout. Section V describes the experimental approach using traditional standard error metrics as well as using the predictions as input to a service composition approach to evaluate the impact on the final composition. Section VI presents the results and Section VII concludes and outlines additional future work.

II. Related Work

To encourage additional user engagement with IoT services one of the key research challenges to overcome is the unreliability that can be caused by services provided by devices that can be mobile, low power and wireless. This problem can be tackled at a number of different stages in service composition with approaches proposed for dynamic service composition [14], service selection [15], run-time QoS evaluation [16] and service ranking prediction [17]. These approaches assume that the QoS values are already known, but, in reality, user side QoS may vary significantly, given unpredictable communication links, mobile service providers and resource constrained devices. We can use collaborative filtering based methods to make predictions for these QoS values based on other similar users in the environment.

There are two main collaborative filtering methods, which can typically be classified as either memory or model-based. Memory-based approaches store the training data in memory and in the prediction phase, similar users are sorted based on the current user. There are a number of approaches that use neighbourhood-based collaborative filtering, including user-based approaches [18], item-based approaches [19] and their combination [20].

Model-based approaches, which employ a machine learning technique to train a predefined model from the training datasets, have become increasingly popular. Several approaches have been studied, including clustering models [21], latent factor models [22] and aspect models [23]. Latent factor models create a low-dimensional factor model, on the premise that there are only a small number of factors influencing the QoS [24]. Since then, matrix factorisation has emerged as one of the most-used approaches for collaborative filtering [7], [25].There has been work on extending matrix factorisation-based collaborative filtering by taking into account alternative sources of information such as time [10] or combining content-based features [11]. There are also approaches that take into account additional factors such as the location of the users in the environment [12], [25]. Other approaches have focused on handling users who are contributing false values using a reputation-based matrix factorisation [26]. There has been little use of deep learning models for QoS predictions except for initial explorations using Restricted Boltzmann Machines [27].

IoT adds additional constraints such as the resources available on devices. This requires alternative approaches that can produce accurate predictions with less training time. In our previous paper we focused on making more accurate predictions based on standard error metrics for IoT services using a matrix factorisation based approach [9]. One limitation of this approach is that it is resource intensive to generate the latent features, which increases the training time needed to incorporate new values and makes it difficult to deploy on edge devices. This can lead to a number of issues with user privacy especially with the introduction of the General Data Protection Regulation (GDPR) as users have to report their QoS values to the cloud. There is also the additional problem that deploying the algorithms in a cloud environment rather than at the edge can increase the time taken to receive QoS predictions. This can lead to more failures especially during dynamic service re-composition, where suitable replacement services must be chosen quickly before the application fails. This paper demonstrates an alternative stacked autoencoder algorithm that can be deployed on an edge device due to the dramatic reduction in training time compared to previous approaches, while maintaining state of the art service compositions. We hope our results will encourage future approaches to consider both the training and validation time to allow future algorithms to be deployed at the edge of the network in a dynamic IoT environment.

III. Reliable IoT Applications

Figure 1 shows a small scale IoT scenario with deep edges. The services are provided from a number of different services types including including web services (WS) residing on resource rich devices, wireless sensor networks (WSN), which may be resource-constrained and controlled by a software defined network, and autonomous service providers (ASP),

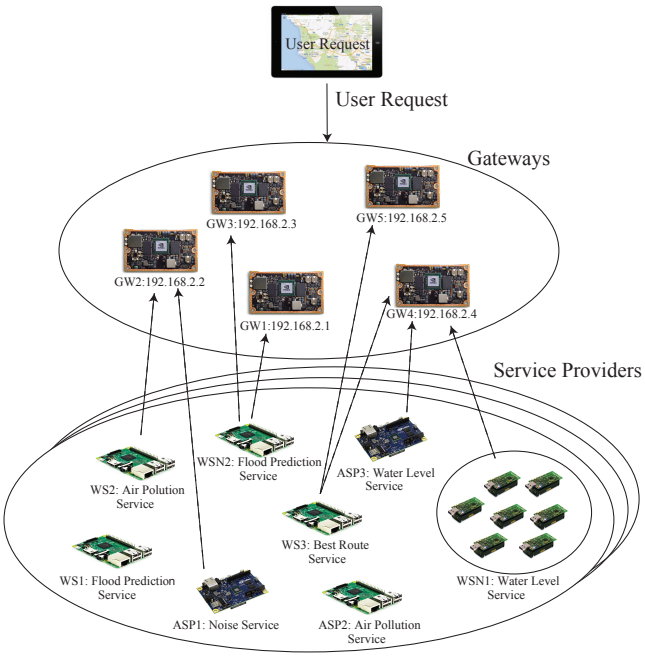


Fig. 1: Deep Edge [28]

who are independent mobile users in the environment with intermittent availability. The services provided by these devices are registered in the gateways with embedded GPUs (Nvidia Jetson Tx2) capable of running deep neural networks at the edge of the network. The access to GPUs at the edge of the network allows for accelerated analytics and more decision-making at the edge using the increased accuracy that can be achieved with deep learning models [29] combined with the quick response of edge networks. This will reduce traffic flow to the cloud as the data can be aggregated and processed at the edge with only a subset sent to the cloud. For example, the edge device can run computer vision analytics in near real time and send the results (content tags, recognised faces, etc.) along with the metadata (owner, capture location, timestamp, etc.) to the cloud. This can dramatically reduce the bandwidth to the cloud by three to six orders of magnitude [30].

Deep edges also help with some of the recent challenges that have been introduced with the establishment of new laws such as GDPR, which came into enforcement on the 25^{th} May 2018 as part of the EU Data Protection Directive. Users now have more control over their data and have become much more interested in what data is being collected about them, how that data is stored and who will have access to their data. To allow for accurate QoS predictions we need to collect data from users about services that they have invoked. Current cloud-based approaches assume that users will allow their metrics to be monitored and uploaded to the cloud. By using deep edges instead of a traditional centralised cloud based approach, we can have boundaries on data that align with the physical model of users in the environment. The edge device can run trusted software modules called privacy mediators that execute on the device and perform denaturing and privacy-policy enforcement on the sensor streams [31]. Edge computing can provide a foundation for scalable and secure privacy that aligns with natural boundaries of trust and conforms to GDPR, while still allowing for QoS predictions to provide reliable IoT applications

There are a number of levels of reliability that can be achieved in IoT applications. One way to categorise them is by their QoS requirements, which are based on the sensitivity and criticality of the application. IoT applications can typically be categorised as best effort (no QoS), differentiated services (soft QoS) and guaranteed services (hard QoS) [32]. In the hard QoS case, there are strict hard real-time QoS guarantees. This is appropriate for safety critical applications such as collision avoidance in a self-driving car system. Soft QoS does not require hard real-time guarantees but needs to be able to reconfigure and replace services that fail. This could be a routing application, which uses air quality, flooding and pedestrian traffic predictions, to provide the best route through the city. If one of the services is about to fail, the application should be recomposed using suitable replacement services. The final case is best effort, where there are no guarantees if a service fails. In this paper we target Soft QoS as we feel this level of reliability is attainable and will be suitable for the majority of IoT applications.

A. QoS Monitoring & Prediction

To maintain Soft QoS guarantees for IoT applications we use a middleware to support additional functionality such as service discovery, registration and composition. The middleware also has a monitoring engine that monitors currently executing services. This data is input to the prediction engine, which makes predictions for the QoS of both currently executing and candidate services. The components are independent and communicate using a publish/subscribe MQTT broker allowing alternative components to easily be added or removed. The services have a range of different QoS properties including response-time, location of sensors and energy consumption. The communication links for invoking these services are diverse, which may affect the personal QoS experience of users, so they are monitored using the monitoring engine. In this paper, we focus on the prediction of QoS values for candidate services at the edge of the network.

1) Currently Executing Services: For currently executing services that require hard or soft QoS guarantees we want to monitor and make predictions to identify whether any service in the composition is about to fail or degrade in quality. We monitor the services at the gateways and build up a time series dataset, which is used to train our LSTM model [29]. This model is then used to identify when a service may be about to fail or degrade in quality and commit an SLA violation. When a possible SLA violation is detected, the prediction engine sends an alert to the service composition engine to compose a new path using services already registered. If none of the registered services has the functionality to replace the service, a message is sent to the service discovery engine for proactive

service discovery, to identify whether any new services have entered the environment and can replace the service about to commit an SLA violation. This can allow for the replacement of services in a composition before the user identifies that there is a problem with the application.

2) Candidate Services: When an alert is triggered from the prediction engine that a service is about to degrade in quality, a suitable replacement service needs to be chosen from a list of possible candidates, selected by the service discovery engine based on their functional capabilities. The final candidate service is chosen from this list based on non-functional requirements such as response time and throughput. The candidate service is chosen using a collaborative filtering approach, which uses the reported QoS values from other users in the environment to make predictions for services that the user has not invoked. A composition engine can decide to always choose the service with the best predicted QoS or to choose one of the top-k services that satisfy the users request to avoid potentially overloading the top service. The problem is to maintain the accuracy of the QoS predictions of services, while using minimum resources. The goal is to enable reliable service compositions in mobile, dynamic environments, where devices may be resource constrained. We propose a new approach to evaluate these types of algorithms focused on assessing the impact on the service composition. We include the standard error metrics to show the difference in evaluation approaches. We also evaluate other important metrics for QoS prediction at the edge such as training time and invocation time to ensure that these algorithms are suitable to be used in an IoT environment and conform to the time requirements required for a dynamic service composition.

IV. Collaborative Filtering

A. Matrix Factorisation

Matrix factorisation is the most commonly-used approach for collaborative filtering as it is known to give good results even in sparse matrices [9]. The idea is to factorise the sparse user-service matrix into $V^T H$ to approximate the original matrix, where the low-dimensional matrix V denotes the user latent feature space and the low-dimensional matrix H represents the service latent feature space, using the latent factor model [22]. The features in the latent feature space represents the underlying structure in the data, computed from the original dataset using matrix factorisation. For example, in a movie rating dataset a latent feature may be the genre of the movie and another latent feature may be a combination of the year released and the director.

Let Ω be the set of all pairs $\{i, j\}$ and Λ be the set of all known pairs (i, j) in Ω. Consider the matrix $W \in \mathbb{R}^{m \times n}$ consisting of m users and n services. Let $V \in \mathbb{R}^{l \times m}$ and $H \in \mathbb{R}^{l \times n}$ be the latent user and service feature matrices. Each column in V represents the l-dimensional user-specified latent feature vector of a user and each column in H represents the l-dimensional service-specific latent feature of a service. We can employ an approximating matrix $\tilde{W} = V^T H$ to learn the user-service relationship W [33]:

$$w_{ij} \approx \tilde{w}_{ij} = \sum_{k=1}^{l} v_{ki} h_{ki} \tag{1}$$

To learn matrices V and H from the obtained QoS values in the original matrix W, we need to construct a cost function to evaluate the accuracy of the approximation. We use the standard Euclidean distance between the two matrices as the cost function.

$$F(W, \tilde{W}) = \parallel W - \tilde{W} \parallel_F^2 = \sum_{ij} (w_{ij} - \tilde{w}_{ij})^2 \tag{2}$$

where $\| \cdot \|_F^2$ denotes the Frobenius norm, which can then be minimised using incremental gradient descent to find a local minimum [34]. Once we have access to the low dimensional dense matrices we can then compute the similarity between different users and services using their latent features. The final prediction can then be made using a weighted average of the most similar users and services.

B. Autoencoder

An Autoencoder learns to compress data from the input layer to the hidden layer and decode the hidden layer into something that closely matches the original data, as can be seen in Figure 2. As the hidden layers contain fewer neurons than the input, the autoencoder engages in dimensionality reduction through Non Linear Principle Component Analysis (NPCA). In our experiments, we use a stacked autoencoder with additional layers for greater expressive power. The use of stacked autoencoders captures useful hierarchical grouping of the input. An example from the use of autoencoders in vision problems is that the first layer of a stacked autoencoder tends to learn first-order features in the raw input such as the edges of an image. The second layer can then use these features to learn second-order features such as what edges tend to occur together to form contours or corner detectors, with additional layers using these second-order features [35].

The stacked autoencoder extends this model with multiple hidden layers stacked together to generate a richer representation that leads to better performance [36]. One of the best ways to obtain good parameters for a stacked autoencoder is to use greedy layer-wise training [37]. The raw inputs are trained on the first layer to obtain parameters and transform the raw input into a vector consisting of the activation of the hidden units. The second layer is then trained on this vector and this process is repeated for subsequent layers, using the output of each layer as input for the subsequent layer. The weights learned during the training process are then used to make predictions for users with missing QoS values.

A classical auto-encoder is typically implemented as a one hidden-layer neural network that takes a vector $x \in \mathbb{R}^D$ as input and maps it to a hidden representation $z \in \mathbb{R}^K$ through a mapping function:

$$z = h(x) = \sigma(W^T x + b), \tag{3}$$

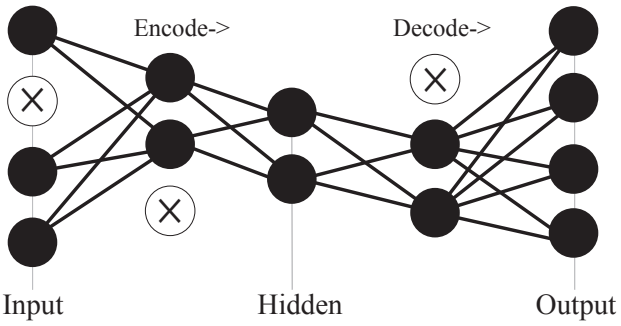


Fig. 2: Autoencoder Architecture with Dropout

where W is a $D \times K$ weight matrix and $b \in \mathbb{K}$ is an offset vector. The resulting latent representation is then mapped back to a reconstructed vector $\hat{x} \in \mathbb{R}^D$ through

$$\hat{x} = \sigma(W^{'} z + b^{'}) \tag{4}$$

The reverse mapping may optionally be constrained by tied weights, where $W^{'} = W$. The parameters of this model are trained to minimize the average reconstruction error

$$\underset{W,W',b,b'}{argmin} \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, \hat{x_i}) \tag{5}$$

where ℓ is a loss function such as the square loss or the cross entropy loss.

1) Denoising Autoencoders: Traditional autoencoders can often just become identity networks and fail to learn the relationship between data. This issue has been tackled by corrupting inputs, pushing the network to denoise the final inputs [38], [39]. The input can be corrupted using a number of different approaches such as Guassian noise, Masking noise or Salt and Pepper noise. This modifies a traditional autoencoder loss function to emphasize the denoising aspect of the network. It is based on two main hyperparameters α, β, which focus on whether the network would focus on denoising the input α or reconstructing the input β.

$$L_{2,\alpha,\beta}(x, \tilde{x}) = \alpha(\sum_{j \in J(\tilde{x})} [nn(\tilde{x}_j) - x_j]^2) + \beta(\sum_{j \notin J(\tilde{x})} [nn(\tilde{x}_j) - x_j]^2) \tag{6}$$

Where $nn(x)_k$ is the k^{th} output of the network, $\tilde{x}$ is the corrupted input x, J are the indices of the corrupted element of x. Another regularisation technique that is often used in deep neural networks is dropout [40]. It randomly drops out units from both the hidden and visible layers. Figure 2 shows an example of a stacked autoencoder after applying dropout. The neurons marked with x have been dropped and contain no input or output connections. Each unit in the network is retrained with a fixed probability p, which is equal to 0.5 in our experiments. Hidden layer activation before dropout is according to the following equation:

$$h^k(x) = g(a^k(x)) \tag{7}$$

where layer k ranges from 1 to L, which is the label of the hidden layer. g is the activation function sigmoid that is used in our experiments. The equation after dropping out units in hidden layers with probability p is:

$$h^k(x) = g(a^k(x)) \odot m^k \tag{8}$$

With $k = L + 1$, output layer is:

$$h^{L+1}(x) = o(a^{L+1}(x)) \tag{9}$$

where o is also an activation function.

Designing an autoencoder architecture for the edge is an interesting problem as you have to balance the limitations of the devices that you are training on, while also being able to deliver accurate QoS predictions. We designed our autoencoder approach to have a small number of neurons to reduce training and invocation time [41]. We also tested multiple different numbers of layers to identify the impact that this had on the composition accuracy and response time of the algorithm. We found that the use of dropout on each layer while training allowed us to avoid overfitting while maintaining better prediction accuracy compared to adding noise to the inputs.

The autoencoder is implemented with 4 fully connected layers. The encoding part of the network has two layers of 20 and 10 neurons and the decoding part of the network has two layers of 10 and 20 neurons. We use the sigmoid activation between the layers of the network. We also include dropout on the middle layer with p=0.2. We use the mean squared error as the loss function RMSprop as the optimiser with learning rate=0.01 and weight decay=0.5. We experimented using Adam optimisation [42], but found better results using RMSprop. The network is trained for 20 epochs (passes through the entire training dataset).

V. Experimental Setup

A. Dataset

To test our algorithm, we used an established dataset to control for any differences in QoS that can be caused by invoking services at different times. We used the dataset released by Zheng et al. [6], which consists of a matrix of the response time and throughput of 339 users from 30 countries for 5,825 real-world web services from 73 counties. The users are a number of distributed computers from PlanetLab and are not co-located with the services. Response time is the time duration between a user sending a request and receiving a response, while throughput denotes the data transmission rate (e.g. kbps) of a user invoking a service. The reason for the use of a dataset instead of a testbed is to allow more users and services to be evaluated as we are not aware of any testbeds that have access to 339 users and 5825 services. The time varying nature of QoS can also lead to the algorithms being evaluated using different values in testbeds making it harder to evaluate the impact of the algorithm.

As this dataset is for web services, which are usually deployed in the cloud, they have better response time than

might be expected from low power devices. To make the data applicable for the experimentation we use the HetHetNets traffic model to add heterogeneous traffic data to the existing dataset [43]. This provides a realistic and manageable traffic model that can be applied in many contexts such as Wi-Fi, ad-hoc and sensor networks. We set the parameters to $\lambda = 2$, for modelling network traffic in sensor networks and the IoT [43]. There are different scales in the datasets with the response time ranging from 0.002s to 27.285s and the throughput ranging from 0.004kbps to 1000kbps. There is also a high level of kurtosis in both dataset with response time=11.03 and throughput=28.58.

B. Metrics

We use a combination of metrics to evaluate the prediction accuracy of the proposed algorithms, in particular, standard error metrics such as the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). MAE is defined as:

$$MAE = \frac{1}{N}\sum_{i,j}|w_{i,j} - w^*_{i,j}| \tag{10}$$

RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i,j}(w_{i,j} - w^*_{i,j})^2} \tag{11}$$

where $w_{i,j}$ is the QoS value of service c_j observed by user, u_i, $w^*_{i,j}$ denotes the predicted QoS value of the service c_j by user u_i. N is the number of predicted QoS values, which normalises the prediction accuracy across the matrix densities. $RMSE$ gives large weight to extreme errors due to the squaring term.

One of the problems with standard error metrics is that it can be difficult to know how the difference in accuracy of the predictions will perform as part of a realistic IoT application. To evaluate this, we use the predictions as part of a service composition process to compose an application. We generated a number of composition paths with services available in the environment identified to satisfy a user request by the request handler. The service composition engine creates a list of service flows based on the concrete service providers received from the service discovery engine. The flows are then merged based on the service description. If two or more services in the flow have the same input, the composition engine creates a guidepost to enable the invocation of the services based on QoS.

In this work, we consider the QoS metrics of response time and throughput for each branch. The response time is calculated by the mean response time values of each service in the branch and the throughput is the minimum throughput of the set of selected candidate services. The predictive composition process uses the predicted values generated through collaborative filtering to choose the optimal flow. We make 10 composition branches with 10 services in each. Once the branch has been chosen using the predicted values we report the actual values based on the original data. This allows for comparison between the two prediction approaches and the optimal composition that could have been chosen.

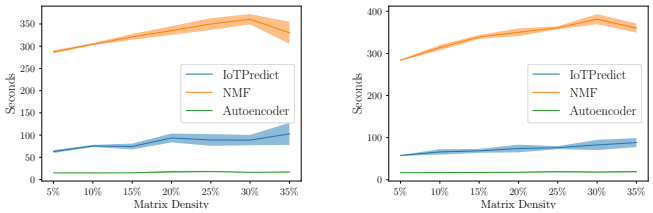
To evaluate the suitability of deploying the algorithm at the edge of the network we evaluate the training time on an edge device, in this case a Jetson Tx2. This time constrains how often the algorithm can be updated with new values, which has an impact on the accuracy when used on time varying values such as QoS. We measured the time taken to train the algorithms at different matrix densities. IoTPredict and NMF are chosen as the matrix factorisation algorithms [9], [33] and the stacked autoencoder algorithm proposed in Section IV-B. We repeat the experiment 20 times showing the average and standard deviation of the predictions in the results section.

We also evaluate the request time of the algorithms showing how quickly a distributed middleware could receive QoS predictions for users in the environment. In this measurement we deploy each algorithm in the environment needed to train it and include the network delay in the response. For the request time we make 5000 requests to the services running on each of the devices and show the distribution of data using a box plot. This gives a realistic request time and shows the impact of being able to deploy an algorithm on an edge node vs. deploying the algorithm in the cloud. We conduct the network test in our university, Trinity College Dublin and connect to each of the devices using a wifi based network.

VI. Results

A. Training Time

Looking at Figure 3a we can see that there is a large difference between the training time of the three algorithms with the two matrix factorisation approaches taking much longer and increasing in time as the matrix density increases. The average training time for the NMF algorithm is over 300s, IoTPredict is over 60s and the autoencoder approach takes 15s. This is a relative speed-up of 20 and 4 times using the autoencoder approach relative to matrix factorisation approaches.



(a) Response Time Training Time (b) Throughput Training Time

Fig. 3: Training Time of algorithms

The results are similar for the throughput training time as seen in Figure 3b, with a large difference between the three approaches. At 5% matrix density, the autoencoder takes 15s, IoTPredict takes 57s and NMF takes 283s. As the matrix density increases the matrix factorisation approaches increase the time to train the model, while the autoencoder approach stays almost the same. At 30% matrix density the autoencoder

takes 18s, IoTPredict takes 88s and NMF taking 381s. In this case the speed-up using the autoencoder approach is 4.8 times and 21 times. The dramatic reduction in training time for the autoencoder approach allows it to be deployed and trained on devices at the edge of the network one hop away from users. This allows the model to constantly update with new values reported by users in the environment. This keeps the predictions accurate as QoS attributes can vary with time so it is important to update the model with recent QoS values.

B. Request Time

The request time is the total end-to-end time from requesting QoS predictions from the prediction algorithm to returning a response including network delay. This is an important factor when using a collaborative filtering approach for time varying values especially in dynamic service re-compositions as the composition engine needs quick access to QoS values for candidate services. The validation time of IoTPredict and NMF are the same as the matrix has been completed during training so we only need a constant time lookup to find the QoS value of the service. We deploy the IoTPredict algorithm in the cloud as the larger training time is not suitable for edge devices. We use a number of cloud instances in different countries to show how the cloud location can impact request time. The autoencoder approach is deployed on the Jetson Tx2 where it is trained.

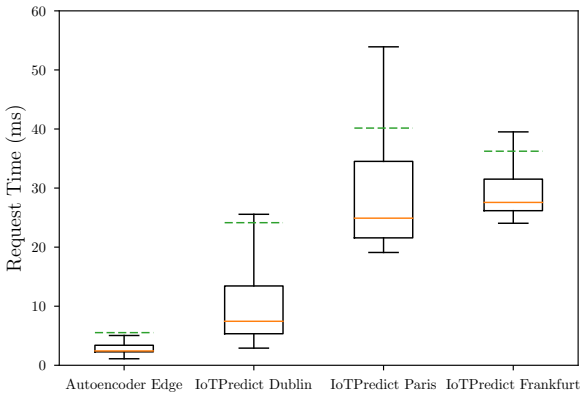


Fig. 4: Request Times

Figure 4 shows the boxplot with the median request time in orange and the average request time in green. We implement the IoTPredict algorithm in three data centres and invoke the request from a node in our university Trinity College Dublin. This explains some of the variation between the data centre locations with the IoTPredict approach having reduced request time in Dublin compared to the other locations, with an average response time of 24.1ms. This is a good response time for a cloud-based approach, however it may be seen as a special case as the Amazon Dublin data centre is located very close to Trinity College, which would not typically be the case for most cloud based services. To evaluate this, we test two other geographically close data centre locations in Paris and Frankfurt. The average response time was 40.2ms in Paris and 36.2ms in Frankfurt.

However, we can clearly see the advantage of being able to deploy the algorithm at edge, with the average response time reduced to 5.40ms. This improvement in response time is especially important for dynamic service recomposition, where one of the services is forecast to fail and a new candidate service needs to be chosen. The reduction in this time will allow for a greater number of applications to recompose before the user notices a failure. Given the current distribution of data centres worldwide the results would typically be worse for cities in South America and Asia where there may be greater distance to the nearest data centre and worse network links.

C. Prediction Accuracy

Figure 5 shows the MAE and RMSE between the actual and predicted values for the evaluated algorithms. There is a clear difference in the prediction accuracy of the matrix factorisation approaches and the autoencoder based approach, with the matrix factorisation approaches producing much more accurate results for both MAE and RMSE in the response time dataset in Figure 5a and Figure 5b. The same results are found for the MAE and RMSE in the throughput dataset in Figure 5d and Figure 5c. However, it is difficult to evaluate how this will have an impact in an actual service composition. When we are selecting a suitable service composition we care more about the ranking and choosing the best ranked composition rather than the individual predictions being accurate. This is why we also evaluate the composition accuracy when using these individual prediction approaches.

D. Composition Accuracy

To evaluate the composition accuracy we used predictions from the algorithms for missing values in the service compositions. We then used a greedy-based service composition algorithm to choose one of the 10 paths based on the predictions and show the average result for all the users in the dataset. The experiment is repeated 20 times and the average values are shown. As the QoS values are taken from a dataset, they do not vary with time and allow the algorithms to be evaluated fairly without showing the limitations of the matrix factorisation approaches to incorporate new QoS data due to a long training time. Figure 6a shows the impact of the predictions for the response time of the final composition. For low matrix densities, which we would expect in an IoT environment the results are similar with less than one second separating the final compositions at 5% matrix density. As the matrix density improves we can see a reduction in response time by using the NMF algorithm. However, for most IoT scenarios we would typically expect a density between 5-10%.

The results for the throughput of the composition path follow a similar pattern in Figure 6b with the composition engine able to generate very similar compositions based on each of the prediction algorithms for low matrix densities. For larger matrix densities greater than 15% we can see a

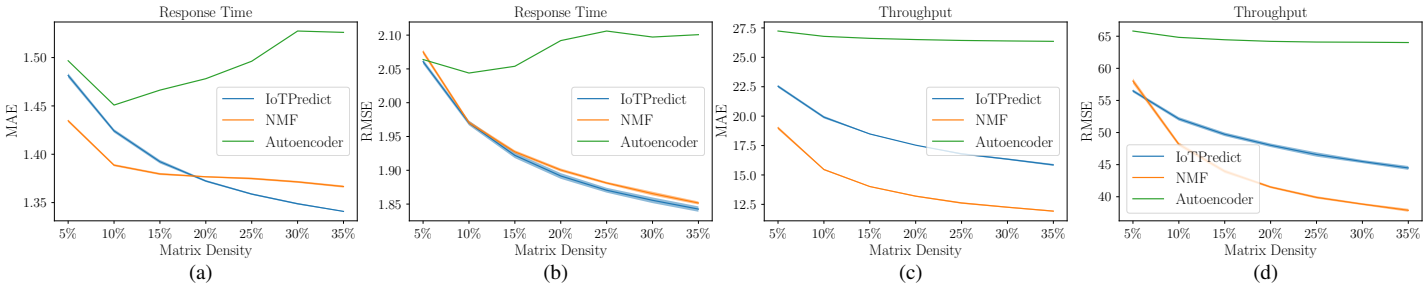
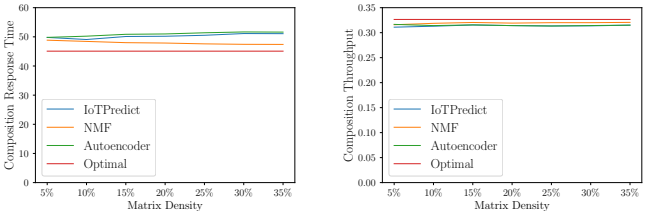


Fig. 5: Impact of Matrix Density on Response Time and Throughput

slight improvement in using the NMF approach. However, as the dataset we use is static, this does not show the benefits of the improved response time and request time achieved by the autoencoder. The NMF approach has a much larger training time, which may over-fit on the current QoS data, which is one of the threats to validity of this test of composition accuracy.



(a) Response time of composition (b) Throughput of composition

Fig. 6: QoS attributes of composition

VII. Conclusion

The results of this paper have provided some interesting insights for QoS prediction especially for future work at the edge of the network. We have shown the dramatic improvement in training times that can be achieved using a stacked autoencoder approach with dropout on an embedded GPU at the edge of the network. This reduced training time will have an impact when using the predictions in a dynamic environment with the QoS varying over time. With the autoencoder approach able to train 4 and 20 times faster than the two matrix factorisation approaches it allows it to consume much more of the recent QoS data and keep up to data with changing QoS values, which will increase the accuracy of predictions.

The second result of this paper focused on the improvement in request time that can be achieved using an autoencoder approach at the edge compared to other approaches that are deployed in the cloud. This allow the algorithm to retrain faster taking into account the latest values by users in the environment. This reduction in time is especially critical in dynamic service recompositions caused by increased workload or network delay. An LSTM based neural network in the prediction engine is used to forecast if one of the services may be about to fail or degrade in quality [29]. This forecast is one timestep ahead so relies on being able to receiving predictions about the QoS of candidate services in the environment quickly to recompose the application with suitable services. We have shown how the use of an autoencoder on an embedded GPU at the edge of the network can reduce the request speed by 5 times compared to traditional cloud based approaches.

We also proposed an evaluation method using the predictions from the algorithms as part of a service composition in addition to comparing the actual and predicted values using standard error metrics. This gives a more realistic evaluation of the impact that the predictions will have when used in a composition. We can see that for low matrix densities, which we would expect in an IoT environment the autoencoder approach is able to make accurate service compositions even with reduced training time. At the moment we are using a greedy-based service composition, however we will explore alternative composition models to evaluate how they impact the final composition when using predicted values. We will publicly release these composition models to allow the community to evaluate future approaches in a more realistic manner.

As part of our future work we are investigating alternative methods of training autoencoders without the user giving access to their data. By training the algorithm at the edge we can use federated learning to download a master model from the cloud and train the model on edge devices. Users then do not have to upload their data to the cloud but only the changes in gradient to the model when trained on their data, which can be encrypted. This new method of training would allow for conformation to GDPR, while also providing accurate QoS predictions. The deep edge architecture that we discuss in this paper allows for a range of alternative training approaches such as federated learning and distributed optimisation. The improved QoS and deep edges can allow for future methods of interacting with services using augmented reality [44].

Acknowledgment

This work was funded by Science Foundation Ireland (SFI) under grant 13/IA/1885. The Jetson Tx2 used for this research was donated by the NVIDIA Corporation.

References

[1] H. Bauer, M. Patel, and J. Veira, "The internet of things: Sizing up the opportunity," *McKinsey*, 2014.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Comm. Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] Y. Yin, W. Xu, Y. Xu, H. Li, and L. Yu, "Collaborative qos prediction for mobile service with data filtering and slopeone model," *Mobile Information Systems*, vol. 2017, 2017.

[4] G. White, A. Palade, and S. Clarke, "Qos prediction for reliable service composition in iot," in *Service-Oriented Computing – ICSOC 2017 Workshops*. Springer International Publishing, 2018, pp. 149–160.

[5] Z. Zheng and M. R. Lyu, "Ws-dream: A distributed reliability assessment mechanism for web services," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. IEEE, 2008, pp. 392–397.

[6] Z. Zheng, Y. Zhang, and M. R. Lyu, "Investigating qos of real-world web services," *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 32–39, 2014.

[7] Z. Zheng, Y. Zhang, and M. R. lyu, "Cloudrank: A qos-driven component ranking framework for cloud computing," in *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems*, ser. SRDS '10, 2010, pp. 184–193.

[8] Z. Zheng *et al.*, "Collaborative web service qos prediction via neighborhood integrated matrix factorization," *IEEE Transactions on Services Computing*, vol. 6, no. 3, pp. 289–299, July 2013.

[9] G. White, A. Palade, C. Cabrera, and S. Clarke, "IoTPredict: collaborative QoS prediction in IoT," in *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom) (PerCom 2018)*, Athens, Greece, Mar. 2018.

[10] Y. Zhang, Z. Zheng, and M. R. Lyu, "Wspred: A time-aware personalized qos prediction framework for web services," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, Nov 2011, pp. 210–219.

[11] L. Yao, Q. Z. Sheng, A. Segev, and J. Yu, "Recommending web services via combining collaborative filtering with content-based features," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 42–49.

[12] X. Chen, Z. Zheng, X. Liu, Z. Huang, and H. Sun, "Personalized qos-aware web service recommendation and visualization," *IEEE Transactions on Services Computing*, vol. 6, no. 1, pp. 35–47, First 2013.

[13] J. Beel and S. Langer, "A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems," in *Research and Advanced Technology for Digital Libraries*, Cham, 2015, pp. 153–168.

[14] N. Chen, N. Cardozo, and S. Clarke, "Goal-driven service composition in mobile and pervasive computing," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[15] A. Yachir *et al.*, "Event-aware framework for dynamic services discovery and selection in the context of ambient intelligence and internet of things," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 1, pp. 85–102, 2016.

[16] G. Su, D. S. Rosenblum, and G. Tamburrelli, "Reliability of run-time quality-of-service evaluation using parametric model checking," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 73–84.

[17] Y. Huang *et al.*, "Time-aware service ranking prediction in the internet of things environment," *Sensors*, vol. 17, no. 5, p. 974, 2017.

[18] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'98, 1998, pp. 43–52.

[19] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, Jan. 2004.

[20] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2007, pp. 39–46.

[21] G.-R. Xue *et al.*, "Scalable collaborative filtering using cluster-based smoothing," in *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2005, pp. 114–121.

[22] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization." in *Advances in neural information processing systems*, 2007, pp. 1257–1264.

[23] P. Singla and M. Richardson, "Yes, there is a correlation: - from social networks to personal behavior on the web," in *Proceedings of the 17th International Conference on World Wide Web*. ACM, 2008, pp. 655–664.

[24] P. Resnick *et al.*, "Grouplens: An open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. ACM, 1994, pp. 175–186.

[25] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu, "Web service recommendation via exploiting location and qos information," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, July 2014.

[26] J. Xu, Z. Zheng, and M. R. Lyu, "Web service personalized quality of service prediction via reputation-based matrix factorization," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 28–37, March 2016.

[27] L. V. Thinh, "Qos prediction for web services based on restricted boltzmann machines," *Journal of Service Science Research*, vol. 9, no. 2, pp. 197–217, Dec 2017. [Online]. Available: https://doi.org/10.1007/s12927-017-0010-6

[28] G. White and S. Clarke, "Smart cities with deep edges," in *ECML International Workshop on Urban Reasoning*, 2018.

[29] G. White, A. Palade, and S. Clarke, "Forecasting qos attributes using lstm networks," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018.

[30] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 139–152. [Online]. Available: http://doi.acm.org/10.1145/2462456.2464440

[31] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy mediators: Helping iot cross the chasm," ser. HotMobile '16. New York, NY, USA: ACM, 2016, pp. 39–44.

[32] G. White, V. Nallur, and S. Clarke, "Quality of service approaches in iot: A systematic mapping," *Journal of Systems and Software*, vol. 132, pp. 186 – 203, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412121730105X

[33] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[34] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[35] J. Xu, L. Xiang, Q. Liu, H. Gilmore, J. Wu, J. Tang, and A. Madabhushi, "Stacked sparse autoencoder (ssae) for nuclei detection on breast cancer histopathology images," *IEEE Transactions on Medical Imaging*, vol. 35, no. 1, pp. 119–130, Jan 2016.

[36] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 513–520.

[37] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153–160.

[38] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08, 2008, pp. 1096–1103. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390294

[39] P. Vincent *et al.*, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1756006.1953039

[40] N. Srivastava *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[41] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, vol. abs/1605.07678, 2016. [Online]. Available: http://arxiv.org/abs/1605.07678

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[43] M. Mirahsan, R. Schoenen, and H. Yanikomeroglu, "Hethetnets: Heterogeneous traffic distribution in heterogeneous wireless cellular networks," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 10, pp. 2252–2265, 2015.

[44] G. White, C. Cabrera, A. Palade, and S. Clarke, "Augmented reality in iot," in *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Workshop (CIoTs), Hangzhou, Zhejiang, China, November 12-15, 2018, Proceedings*. Springer International Publishing, 2018.