Samar Shaaban Abdelfattah Abdelghani Haytamy

Cloud Service composition framework

Definition

Project Overview

The cloud computing idea is based on reusability of IT capabilities. The enterprises as well as individuals can use these Cloud-based services as an partial solution to their operational and business problems. The leading cloud computing providers(Google, Microsoft, Amazon, E-Bay, IBM, etc) have built An online marketplaces to facilitate the publication and searching of different types of cloud services in a more suitable way. The marketplaces provide services on demand, paying per usage and managing automatic service elasticity to meet users' requirements (Cloud services).

The Cloud consumer usually needs to use Cloud services as a partial solution to his requirements. So, the appropriate Cloud services have been composed and provided as a single virtual service to the Cloud consumers.
In this project, I create a desktop application that capable of composing multiple services from different cloud providers (e.g. IBM, Google, Rackspace,). The application uses a predictor trained using QoS (quality of services) dataset[1] to predict future provision values to accurately select providers.

The project was inspired by this paper[2] (Z. Ye, 2016).

Problem Statement

The goal is to create a service composition application capable of recommending best providers to contract with; the tasks involved are the following:

1. Download and preprocess the QoS attributes data (provider side).
2. Generate the consumer preferences/services weights (consumer side).
3. Train predictor1 that can predict future provisioned QoS values(provider side)
4. Train predictor2 that can predict future QoS weights(consumer side)
5. Develop the composer (broker) application
6. Rank or recommend the best cloud providers.

[1] https://github.com/SamarShabanCS/Math_for_ML/tree/master/time%20series%20data%20QoS

[2] https://ieeexplore.ieee.org/document/6964807

The final application is expected to be useful for cloud end users (consumers) because it help him to contract with the best providers appropriate to his requirements.

Metrics

Root Mean Square Error (RMSE): RMSE is used to evaluate the performance of a prediction model.

Let assume that the time series of an individual attribute in the QoS history fits by many prediction models. If the predicted time series are ($\widehat{Q_{1t}}, \widehat{Q_{2t}}, \dots, \widehat{Q_{mt}}$), the prediction error is calculated using the following equation. A lesser value of RMSE imposes a better prediction model.

$$\text{RMSE (i)} = \sqrt{\frac{\sum_{i=1}^{m}(\widehat{Q_{\iota t}}-Q_{it})^2}{m}}$$

Root Mean Square Error (RMSE) is chosen for some reasons as following:

1. Intuitively, the compared-with paper (benchmark model) use this metric.
2. According to the cost function during the training stage, we try to use Mean Absolute Error (MAE), but it fails to get better results, but when RMSE is used, it gets better results.
3. Actually RMSE is better because the loss becomes a parabola so it converges more quickly compared to the graph of absolute value created by MAE. It learns more quickly when the loss is high. So, RMSE penalizing large deviations more.

Analysis

Data Exploration

- A synthetic dataset is used to represent the cloud consumer preferences or his requirements whic h are his preferred weights for response time, throughput, availability for the required service :

  - It is a historical time series data follow Gaussian distribution with ‘matern’ kernel. It is generated using the TimeSynth open source library (https://github.com/TimeSynth/TimeSynth).

  - The end user preferences data will be 3 series of floating point values of length 1440

    Data Shape= 1440 rows × 3 columns

    Note:

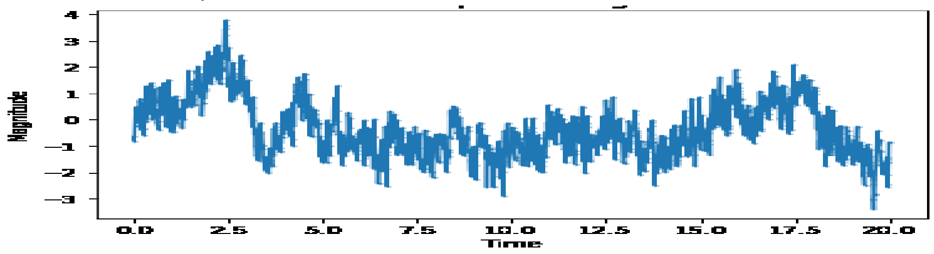
    - The history of four years is 1440=360*4.
    - The interested attributes (3 columns/series); throughput, response time, availability.

- The cloud providers' data set will be represented using a real cloud service data (W. Jiang, 2012) which is updated in (Haytamy S.S.). It contains 5 historical time series for 100 cloud service providers collected through 6 months as 28 time slots as follows: 1. Availability 2. Max Response time 3. Min Response time 4. Avg. Response time 5. Throughput.
  - `The values are floating point numbers`
  - It is available here: (https://github.com/SamarShabanCS/Math_for_ML/tree/master/time%20series%20data%20QoS)

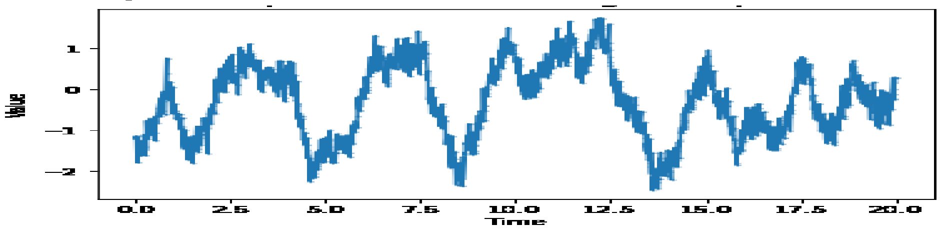
Exploratory Visualization

1. The cloud consumer preferences are synthetic data generated using TimeSynth library. The following graphs are availability, response time and throughput time series.

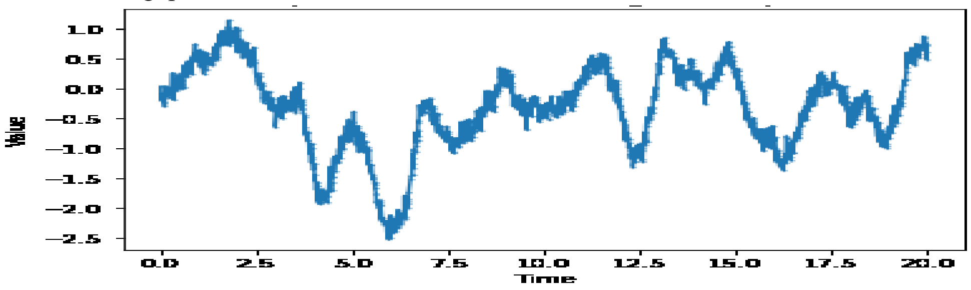
✓ Availability attribute



✓ Response time attribute



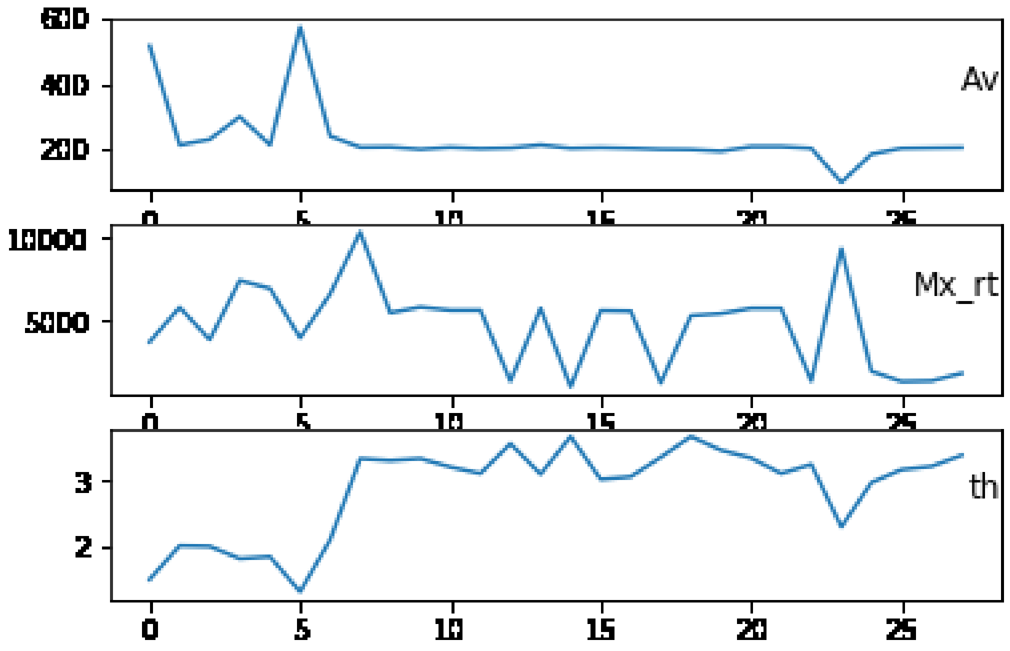
✓ throughput attribute



2. For provider QoS data, the following plot show the distribution or history of the QoS attributes (Av: availability, Mx_rt: maximum response time, Th: throughput) provisioned from one provider selected randomly from the cloud providers' data set. It shows that there is strong negative correlation ship between response time and throughput, when the response time is slower; the throughput is higher and vice versa. When calculating the correlation operator between these two parameters, it is found

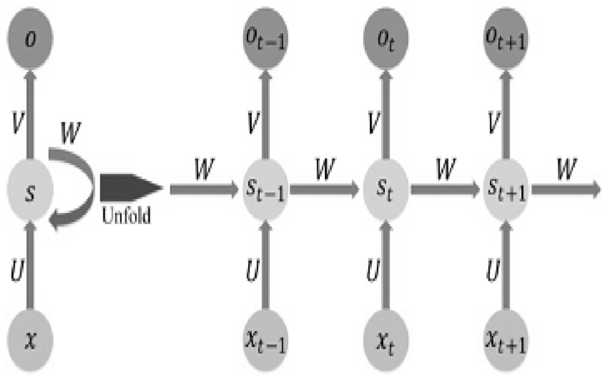
it is equal to -0.67 which ensure the mentioned description. This information helps us to use machine learning model that is capable of capturing this relation between Th and Rt.

Also, the QoS values are varied, so they need to preprocessed (scaling).



Algorithms and Techniques

Long short term memory (LSTM) is one type of recurrent neural network architectures (RNN) that can learn and predict long sequences. The RNN is type of deep neural network (DNN) which used in time series modeling. As a result of the independence of input and output vectors of traditional neural network, it can't make analysis or use sequential information (time series). In contrast, the RNN is designed for treating with sequences where the order of the data is matter. The input of RNN at certain time step is the current input besides a hidden state vector obtained from the previous observation. The RNN architecture is illustrated from the following Figure (colah's blog, 2019).



Unfolded RNN architecture

The Figure symbols are:

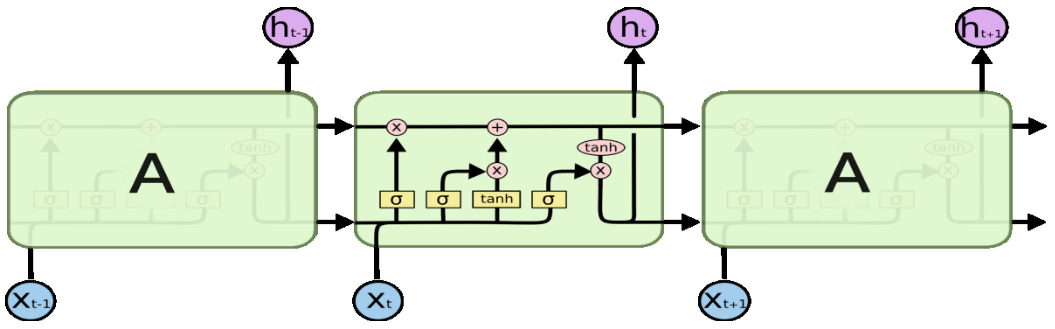
1- *U*, *V* and *W* are the weights for hidden layer, output layer, and hidden state respectively.
2- x_t , o_t and s_t are the input, output and hidden state vectors at time ***t*** respectively.

3- s_t can be calculated based on the input vector and the previous hidden state by using the following formula:
$s_t = f(Ux_t + Ws_{t-1})$

Where; f is the activation function which has many alternatives non- linear functions such as Relu and tanh, and s_0 is initialized by zeros to calculate s_1

4- $\boldsymbol{o_t}$ is calculated using the following formula:
$o_t = f(Vs_t)$

In the standard RNN models, it is hard to learn long-term dependencies because of the vanishing gradient problem where the input values sensitivity decays over time. Therefore, LSTM is an effective solution to overcome this problem by using memory cells, where memory cell would remember the inputs as long as it needed. LSTM memory cell composes four units; forget gate, input gate, update gate, and output gate. The interactions between adjacent memory cells and the memory cell itself are controlled by these gates. The cell state is seen as a conveyor belt. The following Figure illustrates unrolled LSTM network and describes how each gate value is updated (colah's blog, 2019).



The Figure symbols are as follows:

1. x_t and h_t are the input vector and output result to/from the memory cell at time t, respectively.
2. i_t, f_t and o_t are values of the input gate, the forget gate and the output gate at time t, respectively.
3. $\tilde{C}_t$ is the candidate state of the memory cell at time t.

The LSTM network has the ability to add or remove information to/from the cell state through structuring of the cell gates. The forget gate decides what information is going to throw away from the cell state by using the sigmoid layer.

The value of forget gate can be estimated using the following formula:

$f_t = \sigma\ (W_f\ .[h_{t-1}\ , x_t\ ]+ b_f)$

On the other hand, the input gate decides what new information is going to store in the cell state. At first, it decides which information values will be updated, then, it creates new candidates values using tanh layer.

The value of the input gate and the candidate state of the memory cell at time *t* can be formulated using the following Equations respectively.

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c . [h_{t-1}, x_t] + b_c)$$

The update gate amalgamates the output of forget and input gates to create an updated cell state. The state of the memory cell at time *t* is calculated by Equation:

$$Ct = it * \tilde{C}_t + ft * Ct-1$$

Finally, the output will be based on the cell state. The value of output gate and memory cell can be estimated by the following formulas:

$$o_t = \sigma(W_o . [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

The following parameters can be tuned to optimize the predictor:

1. Input Parameters:
    - Preprocessing and Normalization
2. Neural Network Architecture:
    - Model Type (MLP or LSTM; mostly focused on LSTM)
    - Number of Layers (how many layers of nodes in the model)
    - Number of Nodes (how many nodes per layer)
3. Training Parameters:
    - Training / Test Split (how much of dataset to train versus test model on

    - Look back (how many prior days are included in the input sequence
    - Batch Size (how many time steps to include during a single training step
    - Optimizer Function (which function to optimize by minimizing error
    - Epochs (how many times to run through the training process

Benchmark

According to my research, the nearest and closest paper to the mentioned problem is "Long-Term QoS-Aware Cloud Service Composition Using Multivariate Time Series Analysis" (Z. Ye, 2016) (Z. Ye, 2016). So, it will be used as the benchmark model. It propose a multiple QoS prediction model (MQPM) by using the

Arima model to predict the QoS values then compose the services that match cloud consumer requirements by using the Euclidian distance.

The performance of MQPM of its predicted QoS attributes:

Prediction model	RMSE	
	Avg. throughput	Avg. response time
MQPM	0.32	59

Methodology

Data Preprocessing

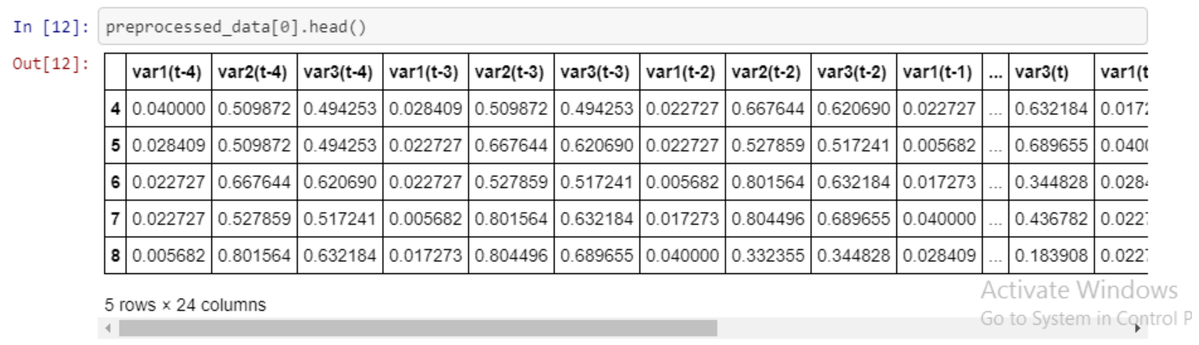
The following steps are done in the preprocessing phase before building and training the prediction model:

1. Generate the user QoS preferences values using TimeSynth generator. This was done in a Jupyter notebooks (titled "END_user_preferences.ipynb"

2. Read the QoS history.
3. put the data points into a Pandas DataFrame for ease of organization and visualization
4. Define train test split ratio
5. Create the training and test datasets
6. Define Look back (how many prior days to include at each time step)
7. convert from series form to supervised form(X,Y)
8. Normalize data (normalize 0.0 to 1.0 for better performance)

⇨ Steps from 2 to 8 are done for both consumer data and provider data in the following notebooks:
(end_user_prediction model .ipynb, M_prediction model.ipynb)

The following graph show sample of preprocessed data (from M_prediction model.ipynb)



```
In [12]: preprocessed_data[0].head()
Out[12]:
```

	var1(t-4)	var2(t-4)	var3(t-4)	var1(t-3)	var2(t-3)	var3(t-3)	var1(t-2)	var2(t-2)	var3(t-2)	var1(t-1)	...	var3(t)	var1(t
4	0.040000	0.509872	0.494253	0.028409	0.509872	0.494253	0.022727	0.667644	0.620690	0.022727	...	0.632184	0.017
5	0.028409	0.509872	0.494253	0.022727	0.667644	0.620690	0.022727	0.527859	0.517241	0.005682	...	0.689655	0.040
6	0.022727	0.667644	0.620690	0.022727	0.527859	0.517241	0.005682	0.801564	0.632184	0.017273	...	0.344828	0.028
7	0.022727	0.527859	0.517241	0.005682	0.801564	0.632184	0.017273	0.804496	0.689655	0.040000	...	0.436782	0.022
8	0.005682	0.801564	0.632184	0.017273	0.804496	0.689655	0.040000	0.332355	0.344828	0.028409	...	0.183908	0.022

5 rows × 24 columns

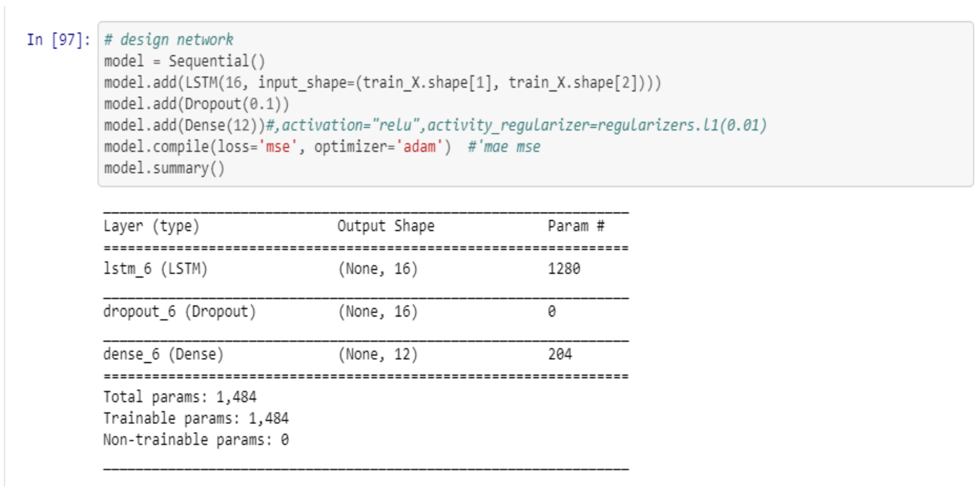
Implementation

The implementation process can be split into two main stages:

- The predictor training stage
- The application development stage

- During the first stage, the predictor was trained on the preprocessed training data. This was done in a Jupyter notebooks (titled "M_prediction model.ipynb": for forecasting the cloud providers' QoS attributes values and "end_user_prediction model .ipynb": for forecasting the user preferences), and can be further divided into the following steps:
  1. Load both the training and validation sequences into memory, preprocessing them as described in the previous section(convert series into (X,Y) or the supervised form, normalize data,..)
  2. Define the network architecture and training parameters
  3. Define the loss function, MSE
  4. Train the network, logging the validation/training loss
  5. Plot the logged values
  6. If the MSE is not low enough, return to step 2
  7. Save and predicted QoS Values

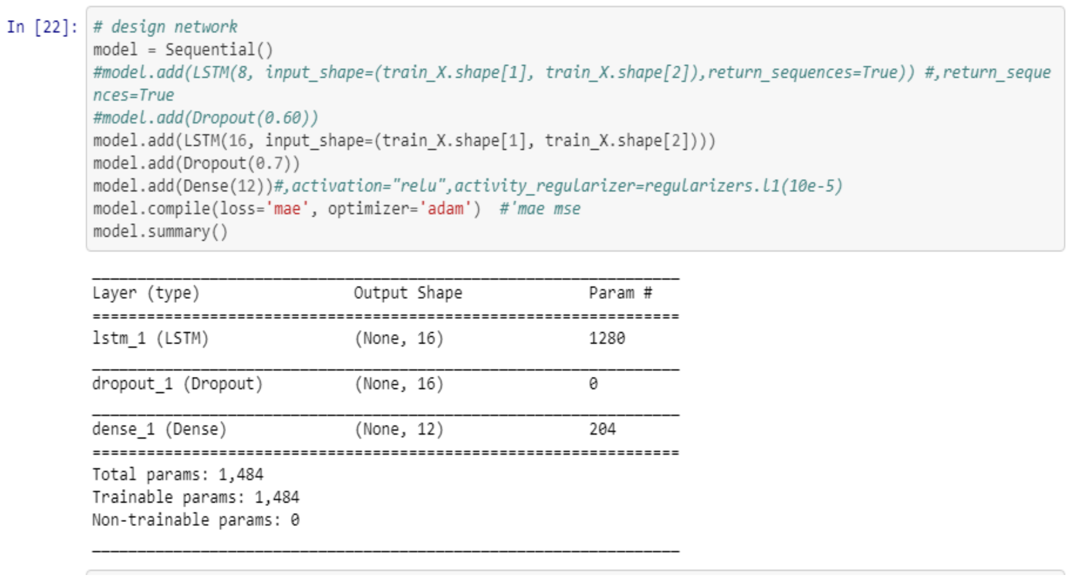
- The following graph is the summary of the network architecture for cloud consumer predictor:



```
In [97]: # design network
         model = Sequential()
         model.add(LSTM(16, input_shape=(train_X.shape[1], train_X.shape[2])))
         model.add(Dropout(0.1))
         model.add(Dense(12))#,activation="relu",activity_regularizer=regularizers.l1(0.01)
         model.compile(loss='mse', optimizer='adam')  #'mae mse
         model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_6 (LSTM)                (None, 16)                1280      
_________________________________________________________________
dropout_6 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense_6 (Dense)              (None, 12)                204       
=================================================================
Total params: 1,484
Trainable params: 1,484
Non-trainable params: 0
_________________________________________________________________
```

- The following graph is the summary of the network architecture for cloud provider predictor:



```
In [22]: # design network
         model = Sequential()
         #model.add(LSTM(8, input_shape=(train_X.shape[1], train_X.shape[2]),return_sequences=True)) #,return_seque
         nces=True
         #model.add(Dropout(0.60))
         model.add(LSTM(16, input_shape=(train_X.shape[1], train_X.shape[2])))
         model.add(Dropout(0.7))
         model.add(Dense(12))#,activation="relu",activity_regularizer=regularizers.l1(10e-5)
         model.compile(loss='mae', optimizer='adam')  #'mae mse
         model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 16)                1280      
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 12)                204       
=================================================================
Total params: 1,484
Trainable params: 1,484
Non-trainable params: 0
_________________________________________________________________
```

- The application development stage which was done in a Jupyter notebooks titled "serviceComposition.ipynb" can be split into the following steps:

1. Load future provisioned QoS attributes values of the 100 providers.
2. Load future preferences QoS attributes values of the user.
3. Split 100 providers services into three classes
4. Define composition function, particle class, particle swarm class (PSO)
5. Run PSO class to obtain the recommended services.
6. Run the brute force algorithm to obtain all possible services composition combination
7. Sort results of brute force algorithm. Assume the top 10 results as the relevant solution
8. Calculate the intersection between PSO algorithm and brute force algorithm

9. Calculate recall and precision according to the following equations

Notations	Relevant	Non Relevant
Retrieved	true positives (tp)	false positives (fp)
Not Retrieved	false negatives (fn)	true negatives (tn)

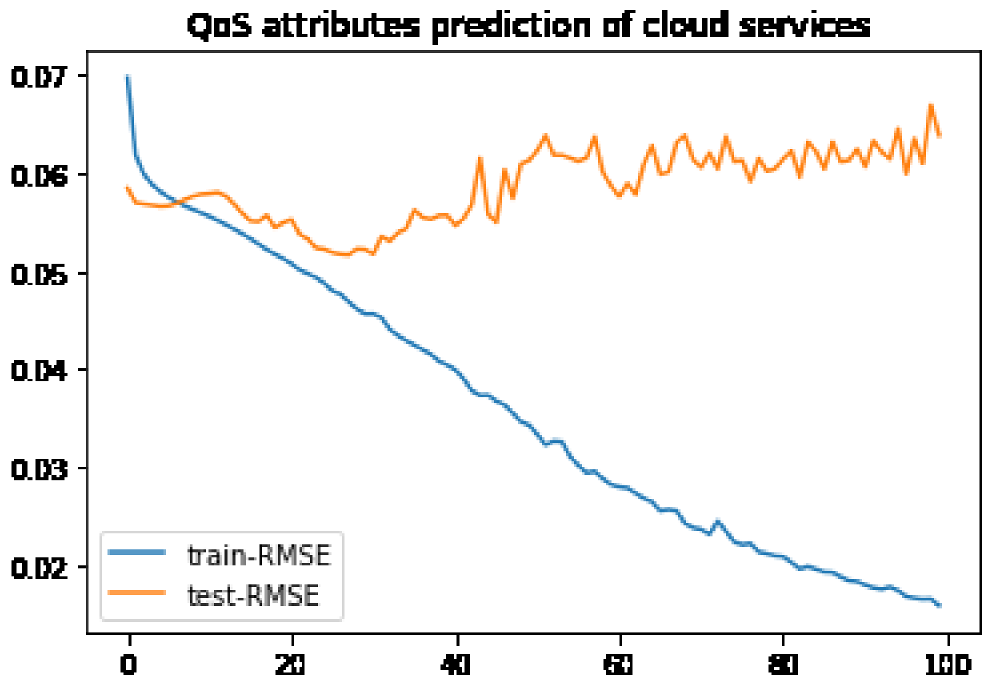
Precision (P) = $\frac{tp}{tp+fp}$

Recall (R) = $\frac{tp}{tp+fn}$

Refinement

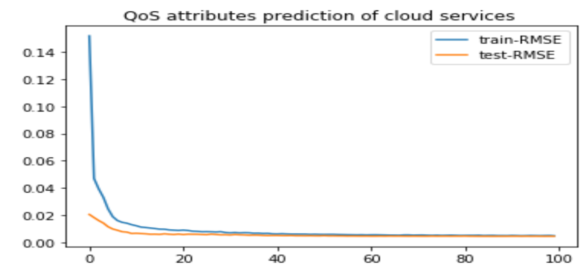
The initial solution has very large RMSE. After adding a drop out layer, the RMSE is significantly reduced.

The following plot of the training/validation losses has a divergence indicates overfitting, which can be addressed by adding dropout layer, or reducing the model complexity (e.g. reducing the number of layers), among other techniques.

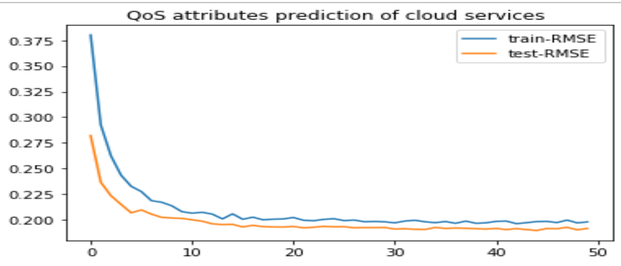


The following plots of the training/validation losses after adding drop out layer and minimizing model complexity:

- ✓ The user model



- ✓ The provider model



Results

Model Evaluation and Validation

During development, a test set was used to evaluate the model. The final architecture and hyper parameters were chosen because they performed the best among the tried combinations.

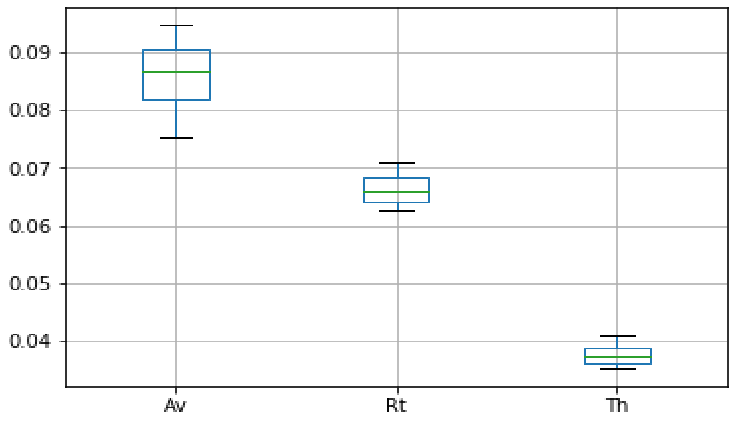
For a complete description of the final model and the training process, the following graph shows the network architecture:

It trained for 50 iterations and each has batch size of 2.

Justification

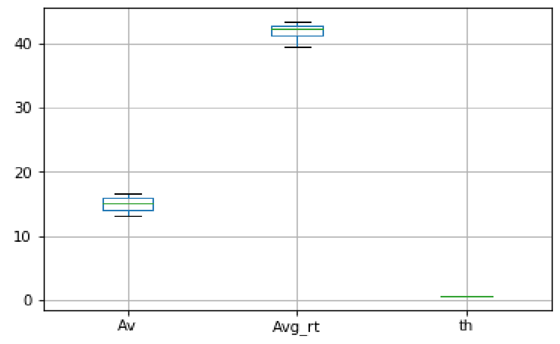
1. The average test RMSE of the user preferences for all attributes is shown through the following box graph:

```
Out[107]:  <matplotlib.axes._subplots.AxesSubplot at 0x7f8e3f68de80>
```



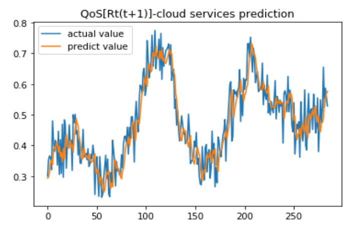
2. The average test RMSE of the provider behavior for all attributes is shown through the following box graph:

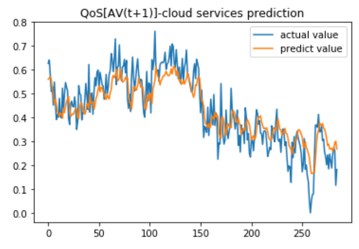
```
Out[33]:  <matplotlib.axes._subplots.AxesSubplot at 0x7f3398110320>
```

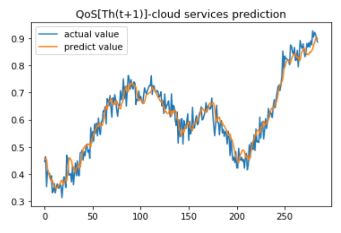


This is significantly less than the RMSE of the benchmark. The following graphs show the predicted throughput and response time.

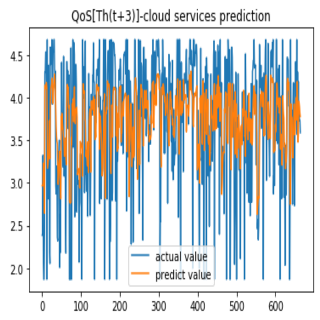
✓ The predicted QoS attributes for cloud consumer:

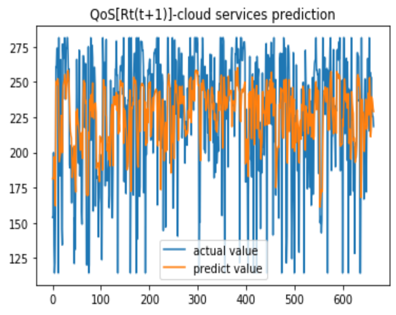


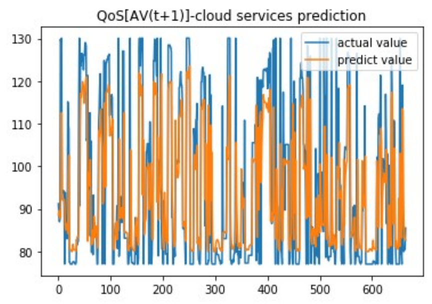




✓ The predicted QoS attributes for cloud provider:







Conclusion

Free-Form Visualization

4. The following graph shows the output of the composition application:

```
In [52]: PSO(func1,x0,bounds,num_particles=10,maxiter=500)

         FINAL:
         best global position:
          [[31 25 22]]


          best global target value:   2.835753094343131
Out[52]: <__main__.PSO at 0x7fd5734f67f0>
```

It shows that according to the user preferences: the user should contract

In the first time period with

Provider number 31 at class 1

Provider number 25 at class 2

Provider number 22 at class 3.

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant, public datasets were found
2. The data was downloaded and preprocessed
3. A benchmark was determined for the predictor
4. The predictor was trained using the data (multiple times, until a good set of parameters were found)
5. The composition application was adapted to use the results of the predictor
6. Use the composition application to remark the cloud providers and contract with the best ones.

I found steps 4 the most difficult, as I had to familiarize myself with LSTM model which was a model that I was not familiar with before the project.

Also, the data set for both the user and the cloud providers, according to the cloud end user preferences; I use the synthetic generator Timesynth, which is the first time to use it.

According the cloud providers QoS values, the used data set has a small length which is only 28 time slot. This hampers to get high performance (lesser RMSE).

Honestly, I tried to extend the length of the dataset, but I failed.(if you have a practical method that extend the time series dataset to keep the correlation ship between the parameters without using randomization, please tell me)

As for the most interesting aspects of the project, I'm also happy about getting to develop LSTM model using multi variables with multiple lags to predict multiple time step. Also, I'm happy about using Keras which using tensorflow backend, as I believe it will be the deep learning library in the future.

Improvement

To achieve high performance we can:

1. Use more capable hardware and use the tensorflow GPU version instead of CPU one.
2. Get more dataset, this will significantly help in getting lesser RMSE.

References:

Cloud services. (n.d.). Retrieved August 2018, from , http://www.webopedia.com/TERM/C/cloud_services.html

Haytamy S.S., K. H. (n.d.). *(2018) ICSD: Integrated Cloud Services Dataset. In: Yang A. et al. (eds) Services – SERVICES 2018. SERVICES 2018. Lecture Notes in Computer Science, vol 10975. Springer, Cham.*

W. Jiang, D. L. (2012). Large-scale longitudinal analysis of soap-based and restful web services. *Proc. Web Services (ICWS), 2012 IEEE 19th International Conference on* (pp. 218–225). Honolulu, HI, USA: the 2012 IEEE 19th International Conference on Web Services.

Z. Ye, S. M. (2016, june). Long-Term QoS-Aware Cloud Service Composition Using Multivariate Time Series Analysis. *IEEE Transactions on Services Computing, 9*(3), pp. 382-393.

(2019, 2). Retrieved from colah's blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.