Deep Neural Networks for Conditional Density Estimation of Service Metrics

HONGYI ZHANG <hongyiz@kth.se>

January 14, 2019

Introduction

Nowadays, a deep neural network is widely used in various fields, such as voice recognition, figure recognition, target predicting. But normally, when facing a complex data set, people need to construct a large size of network in order to achieve smaller predicting bias. Finding a small architecture or using some strategies to reduce the time on searching best hyper-parameters becomes urgent. This project is based on CNSM paper [1] and the main purpose is to find a smallest general architecture which is suitable for all four scenarios.

In this report, the first section will briefly describe the whole experiment setting and data I used. The second section gives an introduction on deep neural network (DNN) and Mixture Density Network (MDN) as defined by Bishop [2]. The following third part will discuss challenges and strategies in finding small DNN architecture. In the Fourth section, I will present and analyze results I got and compare them with CNSM paper.

1 Experiment Setting and Data Choice

The data used in this project is generated by the server cluster which requesting Video-on-Demand or Key-value traffic. The details of server structure are defined in [1]. For the VoD trace, it is available at [3] and KV is available at [4] . While generating the data, there are two kinds of load patterns:

- Periodic-load Pattern: Generator produces the load following the Poisson process. Its arrival rate can be modulated as a sinusoidal function.
- Flashcrowd-load Pattern: Generator produce the load following the Poisson process whose arrival rate can be modulated as a flashcrowd model described in [5]

Before training a model, all the data are cleaned and drop the columns which contain String, Nan values, constant or low variance. Besides, a selectKBest method has been used to reduce the dimension of the X feature sets. In VoD trace, 30 features have been chosen while KV has chosen 200 features. Also, in order to converge faster, all the input

X data have been normalized to $\mu = 0$ and $\sigma = 1$. Finally, for the whole training and testing process, all the traces use the full samples. 70% are used for training while 30% is for the test set.

2 Deep Neural Network and Mixture Density Network

2.1 Deep Neural Network (DNN)

A neural network is a mathematical or computational model that imitates the structure and function of biological neural network (animal central nervous system, especially the brain), and is used to estimate or approximate functions. Neural networks are computed by a large number of artificial neurons. In most cases, the artificial neural network can change the internal structure on the basis of external information. A modern neural network is a non-linear statistical data modeling tool.

Deep neural network(DNN) can be understood as a neural network with many hidden layers. Of course, DNN is sometimes also called Multi-Layer Perceptron (MLP) [6]. The following figure 1 shows a typical Neural Network with two hidden layers.

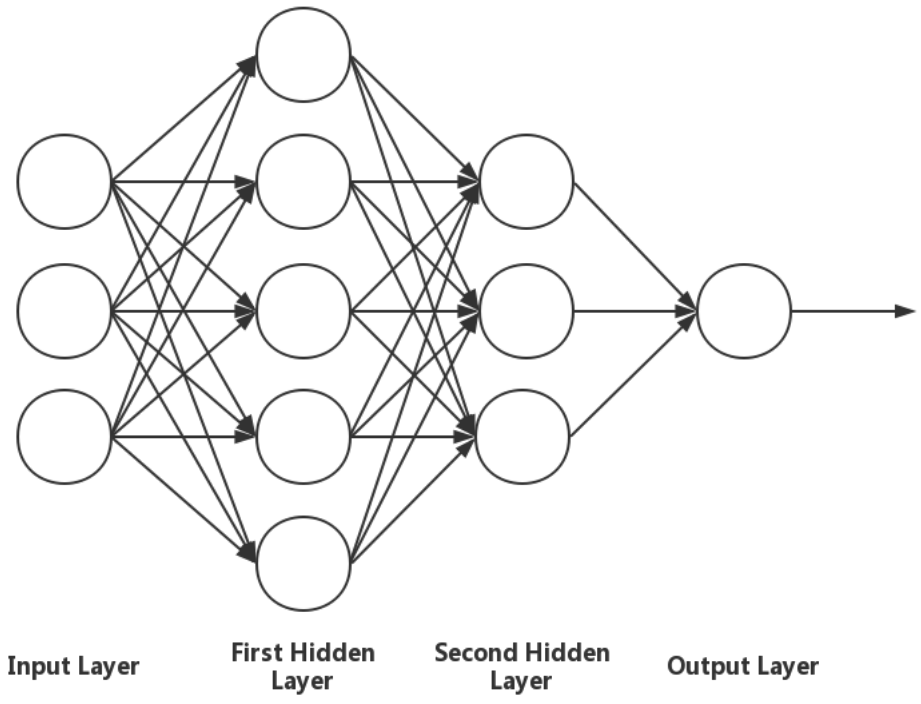


Figure 1: Typical Neural Network Structure

If the activation function we choose is $\sigma(z)$, and the output value of the hidden layer is a. We can use the output of the upper layer to calculate the output of the next layer, which

is the so-called DNN forward propagation algorithm.

Assuming that there are m neurons in the $(l-1)$th layer, and the output a_j^l of the jth neuron in the lth layer is obtained.

$$a_j^l = \sigma(z_j^l) = \sigma(\sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l)$$

The predictions $\hat{y}$ of the network can be evaluated by an L2 loss function.

$$E = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

2.2 Mixture Density Function (MDN)

Traditional neural networks choose Gaussian distribution as conditional probability distribution when generating a model. But in actual machine learning, it often encounters a probability distribution which is really different from Gaussian distribution. For example, a probability distribution can be multi-peaked. In this case, the assumption of Gaussian distribution will produce rather poor prediction results, but MDN has overcome these problems.

MDN will predict a class of mixture Gaussian distribution, where the output value is modeled as the sum of many Gaussian random variables, each of which has a different mean and standard deviation. Therefore, for the input x, we will predict a probability distribution function of $P(Y = y \mid X = x)$ which is the probability-weighted sum of several Gaussian probability distributions [2].

$$P(Y = y|X = x) = \sum_{k=0}^{K-1} \pi_k(x)\phi(y, \mu_k(x), \sigma_k(x)) \text{ where } \sum_{k=0}^{K-1} \pi_k(x) = 1$$

According to its definition, MDN can't simply use the minimum square error loss function, since the output is an entire description of the probability distribution. A more appropriate loss function is to minimize the likelihood function of the distribution versus the training data:

$$E(y|x) = -log(\sum_{k}^{K} \pi_k(x)\phi(y, \mu(x), \sigma(x)))$$

Therefore, for each (x, y) point in the training data set, we can calculate the cost function based on the predicted distribution and the actual point, and then try to minimize the sum of all costs.

3 Challenges and Strategies for Searching a Small DNN Architecture

There is no good rule to determine which structure or architecture is perfect. At now people shall try different hyper-parameters, such as learning rate, hidden units and layers, to achieve an acceptable result. This is also called hyper-parameter optimization. I have done some experiments, about hundreds of times. This is not so sufficient for drawing conclusions, but some of the experiences and strategies can be learnt and applied into our future work.

3.1 Base Structure Choice

Also, there is no golden rule for architecture choice. A really common structure is a rectangular structure, almost all the neural networks will choose this structure. But when searching for a smaller architecture, I am applying a triangular shape one. The following figure 2 shows the difference between these two base structure.

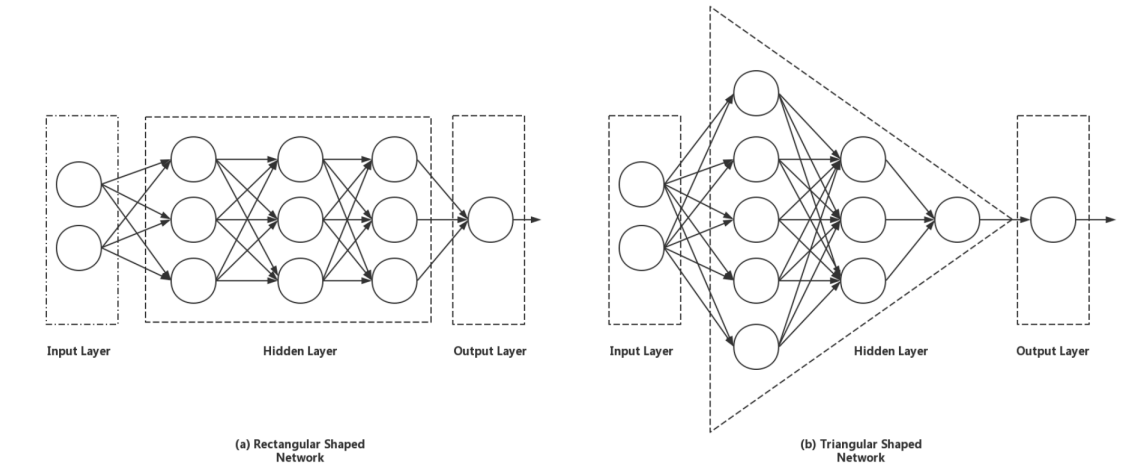


Figure 2: Two different shape of network structure

A triangular shape network will usually contain more parameters (If the number of output is smaller than the number of the input). For example, consider a rectangular network has n hidden layers and m hidden units for each layer, while the input size is a and output size is b ($a > b$ *or* $a >> b$). The number of parameters N_{rec} of this network should be:

$$N_{rec} = m \times (a+b) + (n-1) \times m^2$$

If we simply increase the first layer size and decrease the last to guarantee the total number of hidden units, we can achieve a larger size than the original network. For example, we can make the first hidden layer to be $\frac{3m}{2}$ and decrease the last to $\frac{m}{2}$. Then the size of this simple triangular network will be:

$$N_{tri} = \frac{m}{2} \times (3a+b) + (n-1) \times m^2$$

To compare these two formula, if a is bigger than b, then N_{tri} will always be bigger than N_{rec}. And actually, in the reality, we always have much larger input size than the output, since if we want to achieve a better result, we need to feed more input features into model training and predicting. Based on a simple experiment, it is also proved that a model with more parameters will lead to a better result (Each model should have the same total number of hidden layers). We can see the figure 3 below, which shows the difference between the structures with the same total hidden units. The total number is all 900.

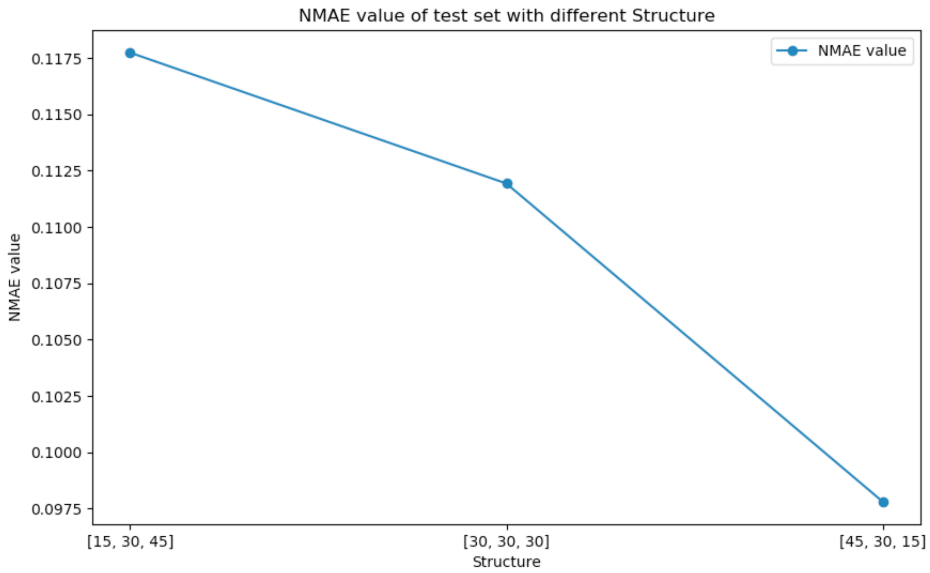


Figure 3: Simple Experiment on comparing NMAE value on test set with different structure of the same total hidden units

The testbed I choose is VOD Flash-crowd since it is easier to train. And for the base structure, I choose [30, 30, 30]. And for the input and output size, they are 30 and 12. For each, I test them twice and plot the average value of test NMAE. We can see that the last structure wins and it improved a lot compared with the first one. The key reason for this is the number of parameters. Since the last structure has much more parameters than the other two structures. The following table shows the total number of parameters for each structure mentioned in the figure.

Structure	Total Parameters
[15, 30, 45]	2790
[30, 30, 30]	3060
[45, 30, 15]	3330

Table 1: Total number of parameters for each structure

Also, in order to prove that the number of parameters is the key factor in determining whether a model is good or not. I have chosen three models with the same total parameters and compare their NMAE value. Their number of parameters are all around 3060 with a little difference. The following figure 4 shows the results.

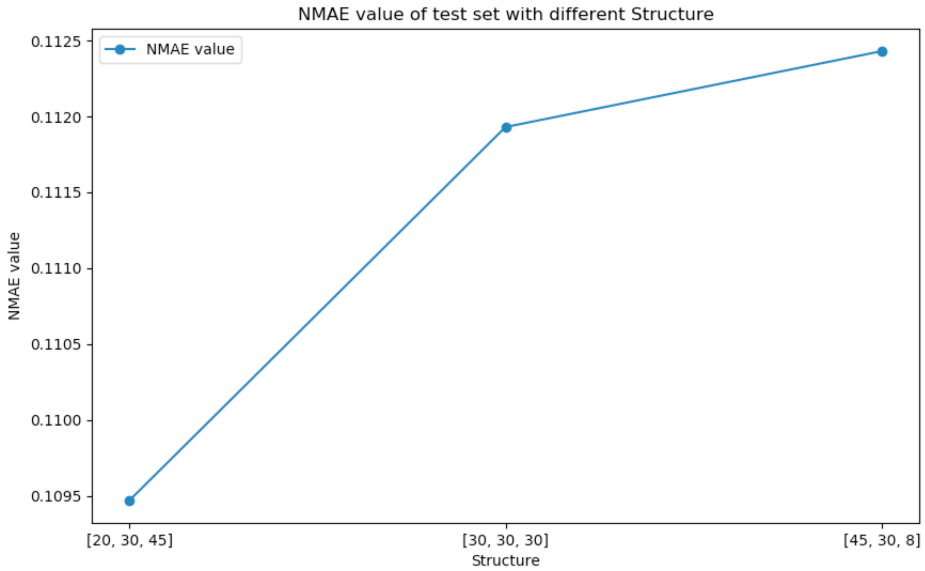


Figure 4: Experiment on comparing NMAE value on test set with different structure of the same total parameters

The best model occurs at [20, 30, 45] but the difference between the best and the worst is only 0.002. I think the reason for the change is still due to the slightly more parameters. The following table shows the parameters for these three structures.

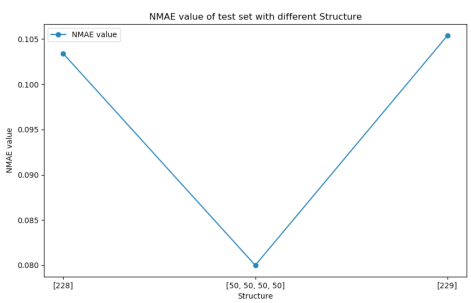
Structure	Total Parameters
[20, 30, 45]	3090
[30, 30, 30]	3060
[45, 30, 8]	3036

Table 2: Total number of parameters for each structure

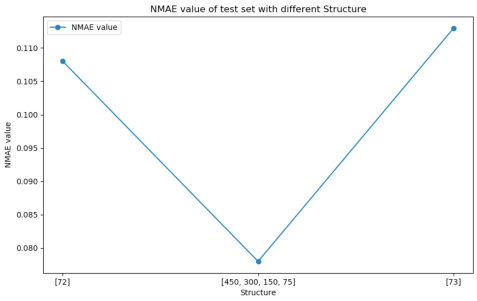
3.2 Choice for the Number of hidden layers

This is a really interesting question during my experiment. In practical, there is also no rule to determine the number of hidden layers. But in theory and experiments, a multi-layer network should usually behave better than a single layer network [7].

Assume that the number of parameters is roughly the same, but with the different number of hidden layers, the kind of representations aren't. Upper-level representations can be really hard (or even impossible) to obtain directly from lower level representations, and inputs can be always treated as zero-level representations (raw representation). A single hidden layer might be enough, but only if the input is continuous and representable within a specific space. I have also conducted an experiment on this, which can give us a basic knowledge.



(a) Compare to structure (50, 50, 50, 50)



(b) Compare to structure (450, 300, 150, 75)

Figure 5: Experiment on same total parameters but different number of hidden layers

For each figure, the structure has almost the same total parameters. But when we increase the number of hidden layers, the accuracy can be greatly improved for both situations.

Based on the universal approximation theorem [8], it shows that if a feed-forward neural network has a linear output layer and at least one activation function with any kind of non-linear property (for example, Logistic sigmoid or ReLU activation function), then this network with sufficient parameters can approximate any continuous function from one finite dimensional space to another finite dimensional space with arbitrary precision. The universal approximation theorem means that no matter what function we are trying to learn, we know that a large MLP must be able to represent it. However, we cannot guarantee that the training algorithm will be able to learn. Even if MLP can represent it, the learning process can be failed for the following two different reasons:

- An optimization algorithm for training may not find correct parameters for the expectation function.
- The training algorithm may have chosen the wrong function due to overfitting.

The feedforward network provides a versatile system to represent functions. In this sense, given a function, there must be a feed-forward network that approximates the function. But there is no process that can verify any particular samples on the training set and select a function to extend the points that are not on the training set.

In summary, a feedforward network with a single layer is sufficient to represent any function, but the network layer may be too large to implement and may not be properly learned and generalized (Model cannot converge or cost too much training time). In many cases, using a deeper model can reduce the number of cells needed to represent the expectation function and can reduce generalization errors.

3.3 Searching Strategy

There are many strategies for hyper-parameter optimization in machine learning. The most commonly used ones are Grid Search and Random Search. In Grid Search, we try every combinations of a preset list with different values of the hyper-parameter and choose

the best combination based on training result. Random search tries random combinations of a range of values. It is good in testing a wide range of values and normally it reaches a very good combination very fast, but the problem is that it doesn't guarantee to give the best parameters combination, which means there's still the possibility that you take a larger amount of time but get noting. On the other hand, Grid search will give the best combination but it will take more time. Figure 6 below shows a diagram about grid searching and random searching.

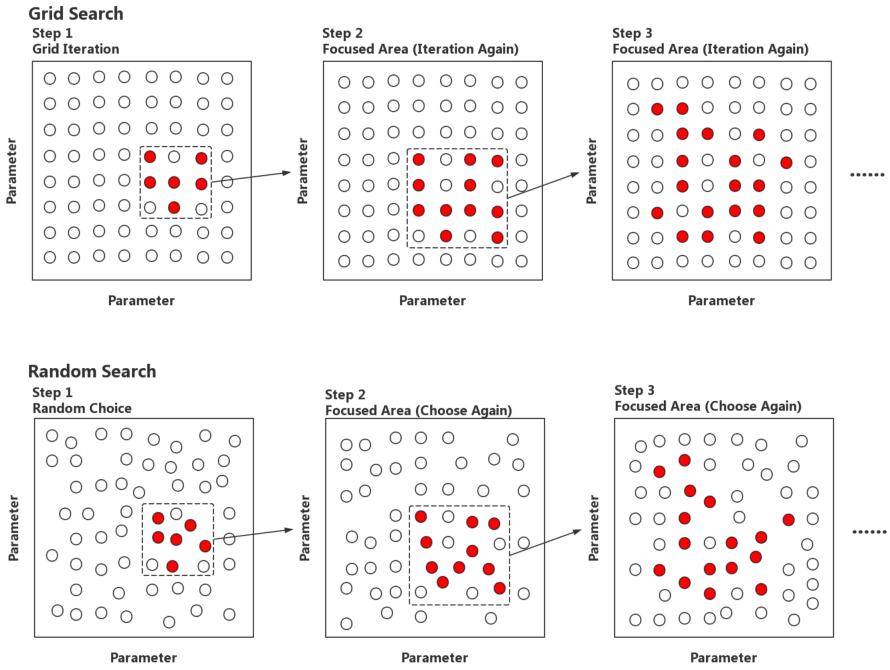


Figure 6: Schematic diagram of Grid Searching & Random Searching (Red points represents good results in each step)

In most cases, since random search has been shown to be extremely effective in practice[9], people will use this strategy to conduct the experiment and try to find a good model. But in this project, I have chosen Grid Searching as my strategy because I have already had start points for four scenarios and I don't want to miss any hyper-parameter which may give me a good model. My start points are from CNSM paper, and the models presented from the paper are obtained by using random searching strategy.

Since my experiment is to improve the result of the CNSM paper, I have a startup model which can give us a good performance. Also, according to the conclusion I mentioned in the previous section, I have chosen my fundamental structure $[m, \frac{m}{2}, \frac{m}{4}, \frac{m}{8}]$. Based on that model and the structure I choose, I create a list and traverse all the value with a constant step:

1. Create the list from m to n with step of h. Traverse all the values and implement in the network.

2. Decrease the range based on the training result, make the range become $\frac{n-m}{2}$ and step $\frac{h}{2}$. Ensure that the middle value is the best of the previous step.

3. Repeat step 2 until find out the best model.

With steps above, I can avoid a lot of time on verifying useless hyper-parameters. For example, in VoD periodic scenario, the following figure 7 shows how these works.

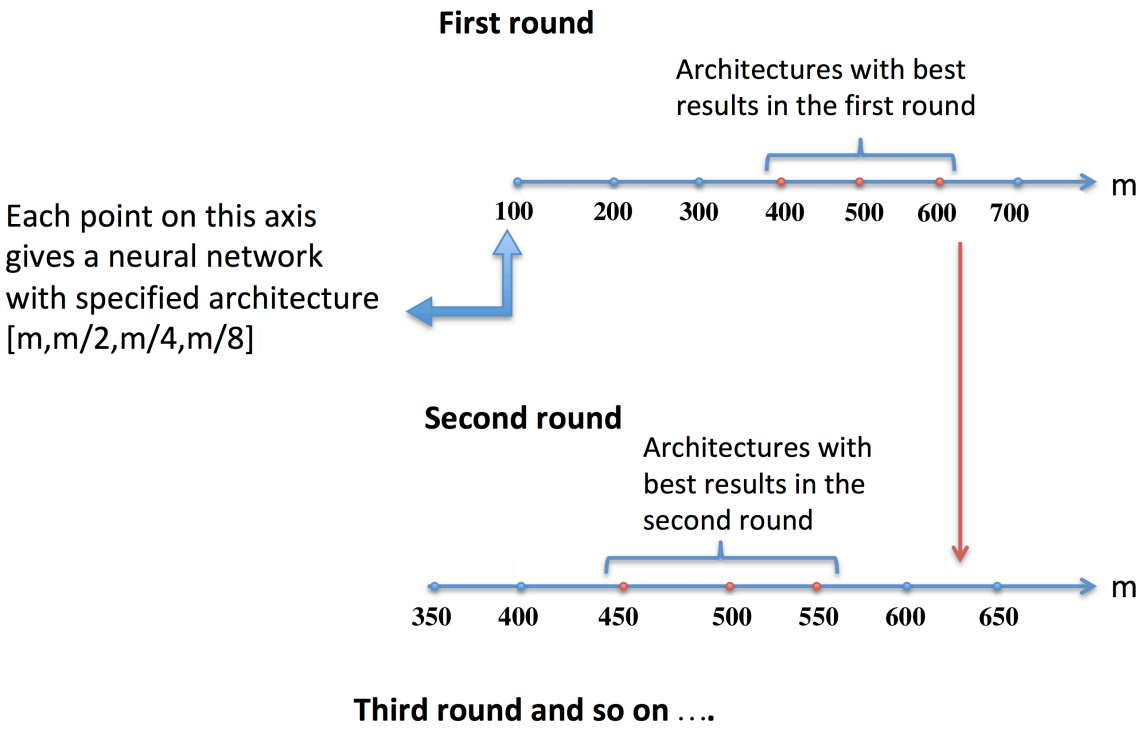


Figure 7: Diagram of the searching strategy applied in this project

In the first round, I traversed the first hidden layer size from 100 to 700 with step 100. In other words, since I had already selected the base structure $[m, \frac{m}{2}, \frac{m}{4}, \frac{m}{8}]$, I tried architecture from [100, 50, 25, 12] to [700, 350, 175, 87] in this round. After obtaining the best results which were between 400 and 600, I narrowed the searching range in second round. Then I repeated all the steps until I got the best value. After determining the best m value for the first hidden layer, I used the same strategy to increase the second, third and fourth layer in order.

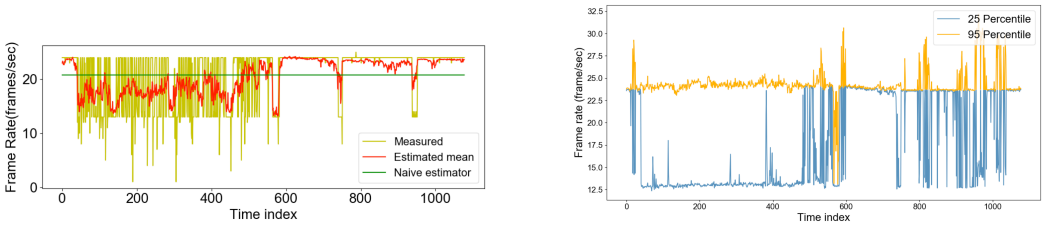
Overall, the process of finding the best architecture is hard and time-consuming. For example, in VoD Periodic scenario, it took 3 rounds to get the best value for the first hidden layer size and 2 rounds for determining the rest. For each round, I tried 7 architectures. Each architecture cost around 1.5 hour to train. Therefore, in order to get the best architecture, 35 architectures had been tried and taken 52.5 hours in total. The above experience also proves that it is necessary to use a good searching strategy.

4 Results and Analysis

4.1 Finding Small Architecture (VoD Periodic Scenario)

In CNSM paper the architecture of the VOD period scenario is really huge. The architecture is [30, 512, 512, 512, 512, 512, 512, 18] which has 6 hidden layers and 512 hidden units for each layer. The total number of parameters of this architecture has reached to 1335296. With this huge network, we can have a perfect NMAE result, but on the other hand, it needs more resources to predict and calculate. Since we only have 15 thousand training samples to generate this huge model, it may cause overfitting to some extent.

Besides, this scenario is really essential for the general architecture because other scenarios cannot handle too many parameters otherwise they won't converge at last. With the method I mentioned above, I have found a good model with smaller architecture and acceptable NMAE value. Figure 8 shows different evaluation results for this model while (a) gives us the time series of the predicted target values for the test set and (b) shows the 25% and 95% percentile.



(a) Time Serie Plot (b) 25%, 95% Percentile

Figure 8: Results of VoD periodic scenario

The figures are really similar to the results presented in CNSM paper. The total number of parameter is 206100, which is about 6 times smaller than the original architecture. Also, the table attached below shows a comparison between CNSM paper result and this smaller architecture.

	CNSM Architecture	My Architecture
Training NMAE	0.10	0.129
Training R^2	0.53	0.43
Test NMAE	0.12	0.1379
Test R^2	0.43	0.393
25 Percentile	0.31	0.24
50 Percentile	0.52	0.44
95 Percentile	0.94	0.92

Table 3: Evaluation metrics between CNSM paper and new architecture (VoD Periodic)

As we can see from the table above, all the results have only a little decline. In smaller architecture, it has a smaller difference between the training NMAE and test NMAE. With the decreasing number of parameters, the variance will be improved but the cost is that the model will have a higher bias. The following table gives the hyper-parameters which can generate this model.

	CNSM Architecture	My Architecture
Layer Strcture	[30,512,512,512,512,512,512,18]	[30, 450, 300, 150, 75, 18]
Mixture Kernel	6	6
Epochs	50000	100000
Learning Rate	0.01	0.01
Number of Tranning Samples	15120	15120
Number of Model Parameters (Weights)	1335296	206100

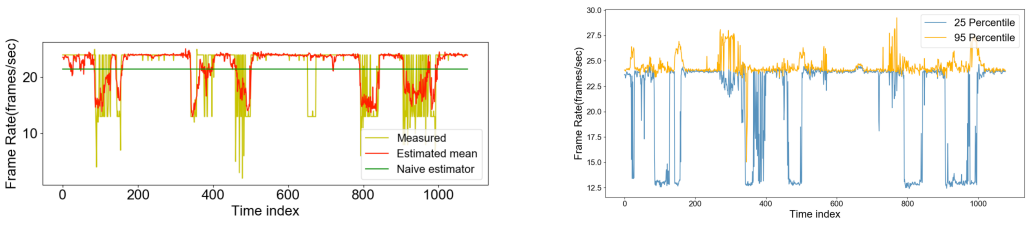
Table 4: Hyper-parameters for VoD Periodic scenario

4.2 Finding Same Architecture for different scenarios

It isn't hard to find a same or similar architecture for each scenario (Periodic and Flashcrowd) under same trace (VoD or KV) since they have similar characteristics and probability distributions. Usually, if the architecture is under a reasonable range, a larger architecture will behave better than a smaller one.

4.2.1 Video-on-Demand (VoD) Trace

The general architecture for VoD trace is [30, 450, 300, 150, 75, 18] for both periodic and flashcrowd scenarios. For periodic scenario, the result has already been shown in the previous section 4.1. The following I will give out the result of VoD flashcrowd.



(a) Time Serie Plot (b) 25%, 95% Percentile

Figure 9: Results of VoD flashcrowd scenario

Also, Figure 8 above shows the evaluation of this architecture while (a) gives us the

time series plot of the test set and (b) shows the 25% and 95% percentile. The table below shows all the evaluation metrics compared to the paper.

	CNSM Architecture	My Architecture
Training NMAE	0.08	0.0721
Training R^2	0.44	0.548
Test NMAE	0.08	0.0788
Test R^2	0.43	0.4603
25 Percentile	0.26	0.30
50 Percentile	0.54	0.50
95 Percentile	0.90	0.93

Table 5: Evaluation metrics between CNSM paper and new architecture (VoD Flashcrowd)

The results here have been slightly increased since this architecture is bigger than the one presented in the paper. While NMAE values have been improved, variance becomes worse due to the larger architecture. The following table shows the hyper-parameters for flashcrowd scenario which is also the general architecture for the whole VoD trace.

	CNSM Architecture	My Architecture
Layer Strcture	[30,50,50,50,50,12]	[30, 450, 300, 150, 75, 18]
Mixture Kernel	4	6
Epochs	50000	100000
Learning Rate	0.01	0.01
Number of Tranning Samples	15120	15120
Number of Model Parameters (Weights)	9600	206100

Table 6: Hyper-parameters for VoD Flashcrowd scenario

4.2.2 Key-value (KV) Trace

It is much easier to find the general architecture for KV trace. The following tables show the hyper-parameter comparison for both scenarios. In periodic scenario, the number of parameters of the new architecture is almost the same as CNSM architecture. But in Flashcrowd scenario, new architecture reduced the number of parameters by nearly half.

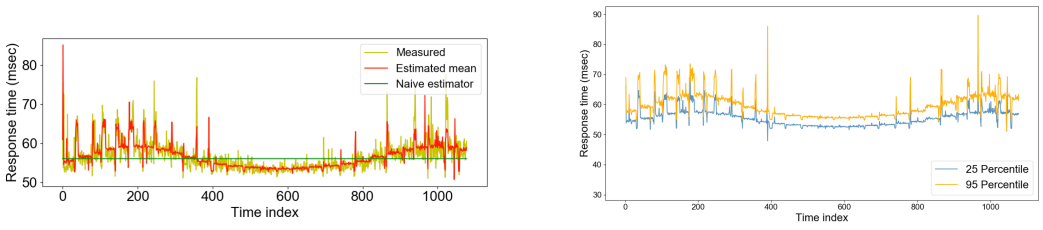
	CNSM Architecture	My Architecture
Layer Strcture	[200, 10, 10, 12]	[200, 10, 10, 10, 12]
Mixture Kernel	4	4
Epochs	50000	50000
kernel regularizer	0.02	0.02
Learning Rate	0.01	0.01
Number of Tranning Samples	20272	20272
Number of Model Parameters (Weights)	2220	2320

Table 7: Hyper-parameters for KV Periodic scenario

	CNSM Architecture	My Architecture
Layer Strcture	[200, 20, 20, 20, 12]	[200, 10, 10, 10, 12]
Mixture Kernel	4	4
Epochs	50000	50000
kernel regularizer	0.02	0.02
Learning Rate	0.01	0.01
Number of Tranning Samples	13609	13609
Number of Model Parameters (Weights)	5040	2320

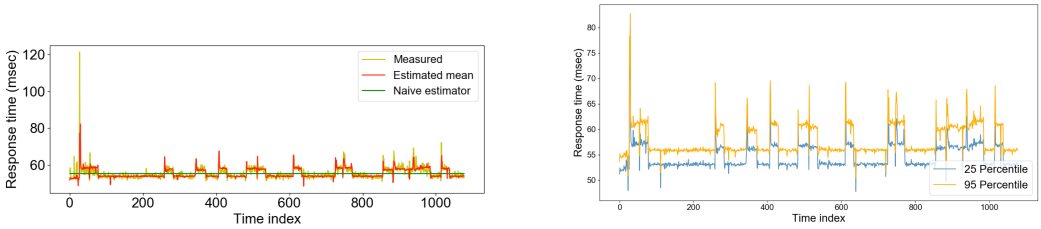
Table 8: Hyper-parameters for KV Flashcrowd scenario

This architecture is much smaller than the one of VoD scenario. The following figure 10, 11 show the results of KV trace for both periodic and flashcrowd scenarios with time plot and 25, 95 percentile.



(a) Time Serie Plot (b) 25%, 95% Percentile

Figure 10: Results of KV Periodic scenario



(a) Time Serie Plot (b) 25%, 95% Percentile

Figure 11: Results of KV Flashcrowd scenario

The following table gives out the evaluation metrics for both KV scenarios. Since the original architecture is really similar to this, the metrics haven't changed a lot on both scenarios.

	CNSM Architecture	My Architecture
Training NMAE	0.024	0.025
Training R^2	0.52	0.649
Test NMAE	0.027	0.0277
Test R^2	0.26	0.607
25 Percentile	0.25	0.25
50 Percentile	0.49	0.50
95 Percentile	0.93	0.93

Table 9: Evaluation metrics comparison for KV Periodic

	CNSM Architecture	My Architecture
Training NMAE	0.0188	0.0190
Training R^2	0.71	0.685
Test NMAE	0.02	0.020
Test R^2	0.59	0.57
25 Percentile	0.24	0.28
50 Percentile	0.49	0.52
95 Percentile	0.90	0.94

Table 10: Evaluation metrics comparison for KV Flashcrowd

4.2.3 Summary

In general, for VoD traces, this report gives out a common architecture [30, 450, 300, 150, 75, 18] for both scenarios. The number of parameters of the final common architecture is 206100. For KV traces, the common architecture is [200, 10, 10, 10, 12] while the number of the parameters of this architecture is 2320.

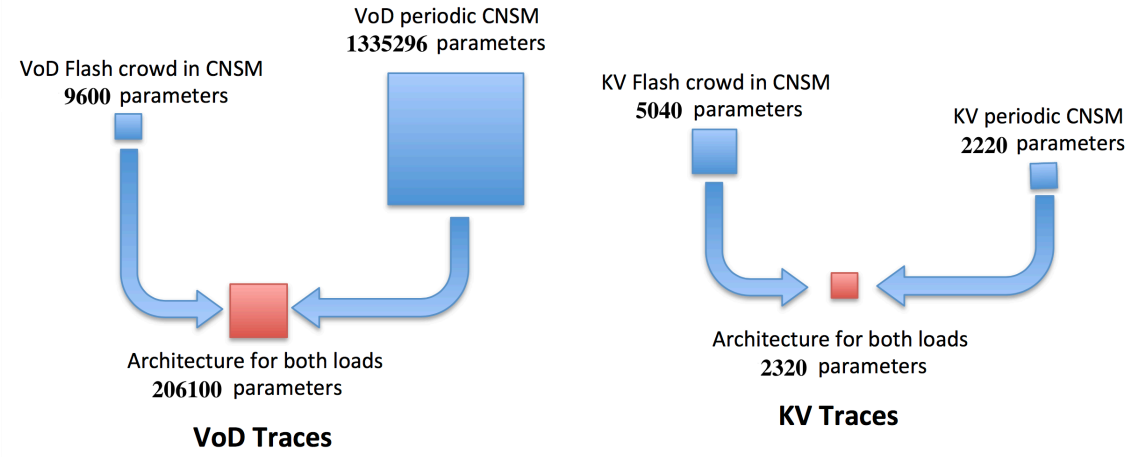
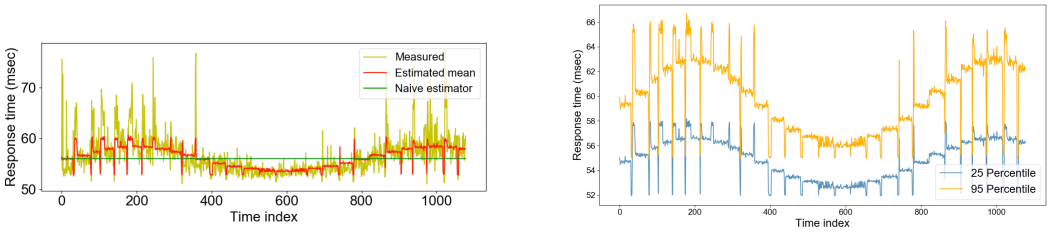


Figure 12: Common Architecture in VoD and KV Traces

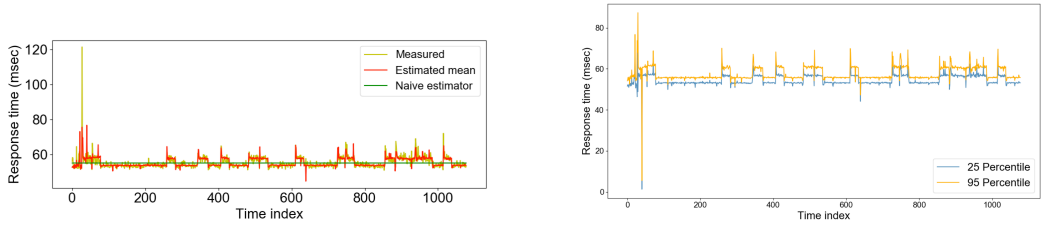
4.3 Finding Same Architecture for whole scenarios

Because of the architecture of VoD trace is much larger than the KV traces, architecture [450, 300, 150, 75] is tried on KV sets. I achieved a great result on flashcrowd scenario, but on period, this architecture is slightly worse. It seems that this architecture is too large for the periodic. The following I give out the result on both periodic and flashcrowd scenario.



(a) Time Serie Plot (b) 25%, 95% Percentile

Figure 13: Results of KV Periodic scenario



(a) Time Serie Plot (b) 25%, 95% Percentile

Figure 14: Results of KV Flashcrowd scenario

Also, the following tables show the metrics for both scenarios. All the values are almost the same compared with the results from CNSM paper.

	CNSM Architecture	My Architecture
Training NMAE	0.024	0.029
Training R^2	0.52	0.5189
Test NMAE	0.027	0.0308
Test R^2	0.26	0.4819
25 Percentile	0.25	0.23
50 Percentile	0.49	0.49
95 Percentile	0.93	0.94

Table 11: Evaluation metrics comparison for KV Periodic

	CNSM Architecture	My Architecture
Training NMAE	0.0188	0.019
Training R^2	0.71	0.50
Test NMAE	0.02	0.021
Test R^2	0.59	0.52
25 Percentile	0.24	0.27
50 Percentile	0.49	0.48
95 Percentile	0.90	0.93

Table 12: Evaluation metrics comparison for KV Flashcrowd

The most interesting thing is that for periodic scenario, a large regularization parameter has been added. This architecture is much larger than the original start point, but for the size of the training size, it is still the same. We need to increase the regularization hyper-parameter to avoid overfitting otherwise the model cannot converge at last.

The following tables show all the hyper-parameters for both scenarios.

	CNSM Architecture	My Architecture
Layer Strcture	[200, 10, 10, 12]	[200, 450, 300, 150, 75, 12]
Mixture Kernel	4	4
Epochs	50000	50000
kernel regularizer	0.02	0.25
Learning Rate	0.01	0.01
Number of Tranning Samples	20272	20272
Number of Model Parameters (Weights)	2220	282150

Table 13: Hyper-parameters for KV Periodic scenario

	CNSM Architecture	My Architecture
Layer Strcture	[200, 20, 20, 20, 12]	[200, 450, 300, 150, 75, 12]
Mixture Kernel	4	4
Epochs	50000	50000
kernel regularizer	0.02	0.02
Learning Rate	0.01	0.01
Number of Tranning Samples	13609	13609
Number of Model Parameters (Weights)	5040	282150

Table 14: Hyper-parameters for KV Flashcrowd scenario

5 Conclusion and Future Work

Although there's no perfect theorem or rule which can easily find the best architecture and best hyper-parameters, from this report, we can have some basic knowledge which can lead us to avoid some traps. This report shows that the essential factor which will have an impact on the model is the total number of the parameters (under same number of hidden layers). Also, a basic method and strategy on how to searching hyper-parameter or base structure have been given.

This report also shows the results of the general architecture for the whole scenarios. The best general architecture is with hidden layer [450, 300, 150, 75] and slightly changes on other hyper-parameters.

A further question is to find a same single model which can predict each traces, VoD and KV. Since for each scenario under the same trace, they have really similar distribution

and can be applied to the same model. Also, it is really great to find out a single model that can predict everything. This will greatly reduce the cost of hardware production and model establishment. Besides, since finding suitable hyper-parameters is really hard and time-consuming, it is really meaningful to do a research on different hyper-parameter optimization methods[10].

References

[1] S. R. Samani F, "Predicting distributions of service metrics using neural networks,." IEEE International Conference of Network and Systems Management (CNSM), Rome, Italy, November 5-9, 2018.

[2] C. M. Bishop, "Mixture density networks," 1994.

[3] "Mldata," 2018. [Online]. Available: http://mldata.org/repository/data/viewslug/realm-cnsm2015-vod-traces/

[4] "traces-jnsm-2017," 2018. [Online]. Available: https://github.com/rafaelpasquini/traces-jnsm-2017

[5] E. M. S. B. I. Ari, B. Hong and D. D. E. Long, "Managing flash crowds on the internet," *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium*, pp. 246–249, Oct 2003.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[7] A. J. Thomas, M. Petridis, S. D. Walters, S. M. Gheytassi, and R. E. Morgan, "Two hidden layers are usually better than one," in *Engineering Applications of Neural Networks*, G. Boracchi, L. Iliadis, C. Jayne, and A. Likas, Eds. Cham: Springer International Publishing, 2017. ISBN 978-3-319-65172-9 pp. 279–290.

[8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[9] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[10] B. Bernstein and C. Potts, "Optimizing the hyperparameter of which hyperparameter optimizer to use," 2016. [Online]. Available: https://roamanalytics.com/2016/09/15/optimizing-the-hyperparameter-of-which-hyperparameter-optimizer-to-use/