Comparing Deep Recurrent Networks Based on the MAE Random Sampling, a First Approach

Andrés Camero, Jamal Toutouh, and Enrique Alba

Departamento de Lenguajes y Ciencias de la Computación,
Universidad de Málaga, Málaga, Spain
andrescamero@uma.es,{jamal,eat}@lcc.uma.es

Abstract. Recurrent neural networks have demonstrated to be good at tackling prediction problems, however due to their high sensitivity to hyper-parameter configuration, finding an appropriate network is a tough task. Automatic hyper-parameter optimization methods have emerged to find the most suitable configuration to a given problem, but these methods are not generally adopted because of their high computational cost. Therefore, in this study we extend the MAE random sampling, a low-cost method to compare single-hidden layer architectures, to multiple-hidden-layer ones. We validate empirically our proposal and show that it is possible to predict and compare the expected performance of an hyper-parameter configuration in a low-cost way.

Keywords: Deep learning · Recurrent neural network · MAE random sampling

1 Introduction

In recent years, Machine Learning (ML) approaches have gained significant interest as a way of building powerful applications by learning directly from examples, data, and experience. Increasing data availability and computer processing power have allowed ML systems to be efficiently trained on a large pool of examples. Deep learning (DL) is a specific branch of ML that focuses on learning features from data through multiple layers of abstraction by applying deep architectures, i.e., Deep Neural Networks (DNNs) [15]. DL has improved dramatically state-of-the-art in many pattern recognition and prediction applications [17, 18].

Deep feedforward networks have incorporated feedback connections between layers and neurons in order to capture long-term dependency in the input. These special case of DNNs are known as Recurrent Neural Networks (RNNs). Thus, RNNs have successfully been applied to address problems that involve sequential modeling and prediction, such as natural language, image, and speech recognition and modeling [15]. However, RNNs present a limitation on their learning process due to two main issues: the *vanishing* and the *exploding* gradient [3, 20].

A promising alternative to mitigate the problems related to the learning process in DNNs is to select/optimize the hyper-parameters of a network. By selecting an appropriate configuration of the parameters of the DNN (e.g. the activation functions, the number of hidden layers, the kernel size of a layer, etc.), the network is adapted to the problem and by this mean the performance is improved [4, 6, 12]. DNN hyper-parameter optimization methods can be grouped into two main groups: the manual exploration-based approaches, usually lead by expert knowledge, and the automatic search-based methods (e.g., grid, evolutionary or random search) [19].

The hyper-parameter optimization of DNN implies dealing with a high-dimensional search space. However, most methods (manual and automatic) are based on *trial-and-error*, i.e., each hyper-parameter configuration is trained and tested to evaluate its numerical accuracy. Thus, the high-dimensional search space and the high cost of the evaluation limit the results of this methodology.

Some authors have explored different approaches to speed up the evaluation of DNN architectures in order to improve the efficiency of the automatic hyper-parameter optimization algorithms [8, 9].

A promising early approach to evaluate one-hidden-layer stacked RNN architectures is the *MAE (mean absolute error) random sampling* [8]. The main idea behind this method, inspired by the linear time-invariant theory (LTI), is to infer the numerical accuracy of a given network without actually training it. Given an input, they generate sets of random weights and analyze the output in terms of the MAE. Then, they estimate the probability of finding a set of weights whose MAE is below a predefined threshold.

In this study, we propose to extend the *MAE random sampling* to multiple-hidden-layer networks and to study the suitability of this method to deal with deeper RNNs. Particularly, we implemented our proposal and empirically tested it using stacked RNNs with up to three hidden layers. The results show that as the RNN gets deeper (more complex), the proposal is capable of given even better results. The reminder of this paper is organized as follows: the next section outlines the related work. Section 3 introduces the multiple-hidden-layer extension of the MAE random sampling. Section 4 presents the results, and finally, Section 5 discusses the conclusions drawn from this study and presents the future work.

2 Related Work

An RNN incorporates recurrent (or feedback) edges that may form cycles and self connections. This approach introduces the notion of time to the model. Thus, at a time t, a node connected to a recurrent edge receives input from the current data point x^t and also from the hidden node h^{t-1} (the previous state of the network). The output y^t at each time t is computed according to the hidden node values at time t (h^t). Input at time $t-1$ (x^{t-1}) can determine the output at time t (y^t) and later by way of recurrent connections [16].

Most of DL approaches to train a network are based on gradient-based optimization procedures, e.g., using a local numerical optimization such as stochastic gradient descent or second order methods. However, these methods are not suitable for RNNs. This is mainly because they keep a vector of activations, which makes RNNs extremely deep and aggravates the exploding and the vanishing gradient problems [3, 14, 20].

More recently, Long Short-Term Memory (LSTM) have emerged as a specific type of RNN architectures, which contain special units called memory blocks in the recurrent hidden layer [11]. LSTM mitigates gradient problems, and therefore, they are easier to train than standard RNNs. Not just the network architecture affects the learning process. The weight initialization procedure determines the learning rate, the convergence rate, and the probability of classification error. Ramos et al. [21] analyzed different weight initialization strategies and provided a quantitative measure for each one of them.

A promising research line in DL proposes to define a specific hyper-parameter configuration for a neural network to improve its numerical accuracy, instead of using a generalized one [4, 6, 12]. The idea is to select the most suitable number of layers, number of hidden unit per layer, activation function, kernel size of a layer, etc. for a given dataset.

When dealing with hyper-parameter configuration, human experts are able to discard hyper-parameterizations without requiring their evaluation by using their expertise. However, intelligent automatic hyper-parameter configuration procedures searches more efficiently through the high-dimensional hyper-parameter space. Even though the intelligent methods are more competitive than the humans, they are not generally adopted because they require high computational resources. This is mainly because they require fitting a model and evaluating its performance on validation data (i.e., they are data driven), which can be an expensive process [2, 4, 22].

Therefore, few methods have been proposed to address this issue by speeding up the evaluation of the proposed hyper-parameterization. Domhan et al. [9] analyzed an approach that detects and finishes the neural networks evaluations that under-perform a previously computed one. This solution was able to reduce the hyper-parameterization search time up to 50%. More recently, Camero et al. [8] presented the MAE random sampling, a novel low-cost method to compare one-hidden layer RNN architectures without training them. MAE random sampling evaluates RNN architecture by generating a set of random weights and evaluating their performance.

In line with the latter approach, we propose to extend MAE random sampling to evaluate RNNs with multiple-hidden-layers. Therefore, it will be suitable to evaluate deeper RNNs.

3 Proposal

We start our discussion with an inspiring fact: changing the weights of a neural network affects its output [10]. In spite of the simplicity (and even triviality) of this fact, it might hide some important clues to characterize the behavior of a net. Camero et al. [8] introduced a novel approach to compare RNN architectures based on this fact: the *MAE random sampling.* They showed its usefulness for comparing the expected performance (in terms of the error) of RNNs with a single-hidden-layer.

In this study we propose to extend the MAE random sampling [8] to multiple-hidden-layers, aiming to validate its usefulness for comparing deeper RNNs. Given an input time series, the idea is to take an arbitrary number of samples of the output of a specific RNN architecture, whose weights are normally initialized independently every time a sample is taken. Then, we propose to fit a truncated normal distribution to the MAE values sampled and estimate the probability p_t of finding a set of weights whose error is below an arbitrary *threshold*. Finally, we propose to use p_t as a predictor of the performance (in terms of the error) of the analyzed architecture.

Fig. 1 depicts the MAE random sampling originally introduced by Camero et al. [8] extended to multiple-hidden-layers RNNs. The distribution of the sampled errors is used to estimate the probability of finding a *good solution*.

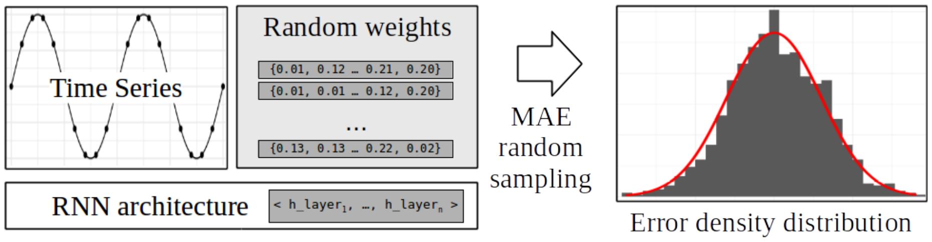

Fig. 1. MAE error sampling based on a random weight initialization

Algorithm 1 presents the adaptation of the MAE random sampling [8] to multiple hidden layers. Given an architecture ($ARCH$), encoded as a vector whose terms represents the number of LSTM cells in the correspondent hidden layer, a number of time steps or look back (LB), and a user defined time series (data), the algorithm takes $MAX\_SAMPLES$ samples of the MAE by initializing the weights with a normal distribution. After the sampling is done, a truncated normal distribution is fitted to the MAE values sampled, and finally p_t is estimated for the inputed $THRESHOLD$.

4 Results

We implemented our proposal[1] in Python 3, using **dlopt** [7], **keras** (version 2.1) and **tensorflow** (version 1.3) [1]. Then, we tested our proposal using a standard problem: the sine wave. We selected this problem because of two main reasons: Camero et al. [8] also studied it, thus we have a baseline to compare to, and any periodic waveform can be approximated by adding sine waves [5].

[1] Code available at https://github.com/acamero/dlopt

Algorithm 1 MAE random sampling of a given architecture.

```
1: $data \leftarrow$ LoadData()
2: $rnn \leftarrow$ InitializeRNN($ARCH$, $LB$)
3: $mae \leftarrow \emptyset$
4: while sample $\leq$ $MAX\_SAMPLES$ do
5:     $weights \leftarrow$ GenerateNormalWeights(0,1)
6:     UpdateWeights($rnn$, $weights$)
7:     $mae$[sample] $\leftarrow$ MAE($rnn$, $data$)
8:     sample++
9: end while
10: $mean, sd \leftarrow$ FitTruncatedNormal($mae$)
11: $p_t \leftarrow$ PTruncatedNormal($mean$, $sd$, $THRESHOLD$)
```

Particularly, a sine wave can be noted as a function of time (t), where A is the peak amplitude, f is the frequency, and ϕ is the phase (Equation 1). We defined the input of our test to be the sine wave described by $A = 1$, $f = 1$, and $\phi = 0$, in the range $t \in [0, 100]$ seconds (s), sampled at 10 samples per second.

$$y(t) = A \cdot sin(2\pi \cdot f \cdot t + \phi) \tag{1}$$

4.1 Single-Hidden-Layer Architectures

To begin with our experimentation we studied the MAE random sampling performance prediction in the set of RNNs with 1 to 100 LSTM cells in the hidden layer and with a look back ranging from 1 to 30. For each architecture we took 100 samples (*MAX_SAMPLES*) and estimated $p_{0.01}$ (i.e., *THRESHOLD*=0.01). Fig. 2 shows the relation between the number of hidden cells and $p_{0.01}$, each color represent a different look back. The probability rapidly increases from 1 to 25 cells, from that point $p_{0.01}$ tends to *converge*.

We selected 100 architectures (i.e. the number of LSTM cells and the look back) and trained them using Adam optimizer [13]. Then, we analyzed the relation between the estimated probability $p_{0.01}$ and the observed MAE. Table 1 presents the correlation between the MAE random sampling results (Mean, Sd, and log $p_{0.01}$) and the observed MAE after training the RNNs for a predefined number of epochs. The table also presents the mean (Mean MAE) and standard deviation (Sd MAE) values of the observed MAE.

There is a moderate to strong negative correlation between $p_{0.01}$ and the MAE observed, thus the results suggest that the estimated probability is useful for comparing RNN architectures. In a rough sense, given two RNNs we might select the one that has a higher probability. Note that we are not predicting the performance, instead we are predicting how *likely* would be to find a set of weights that have a *good* error performance.

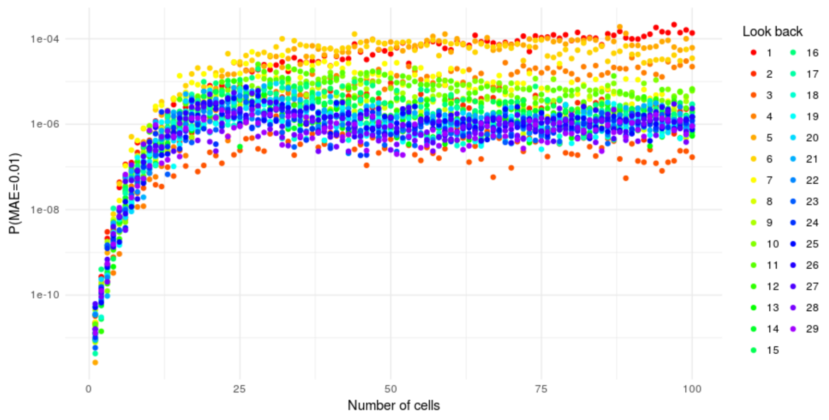

Fig. 2. Probability of finding a *good* set of weights for single-hidden-layer RNNs

Table 1. Correlation between the MAE observed after training the model and the MAE random sampling results (single hidden layer)

Epochs	*Correlation*			*Results*	
	Mean	Sd	log $p_{0.01}$	Mean MAE	Sd MAE
1	-0.447	-0.317	-0.211	0.618	0.017
10	-0.726	-0.431	-0.321	0.517	0.111
100	-0.790	-0.641	-0.650	0.133	0.146
1000	-0.668	-0.458	-0.515	0.036	0.079

4.2 Two-Hidden-Layer Architectures

The previous results suggest that the proposal is useful to compare single-hidden-layer architectures (as it is also stated in [8]), but what about multiple hidden layers? To begin with the validation in a deeper context, we studied the problem using the same sine wave and the search space defined by the stacked RNNs with 7 to 31 LSTM cells per layer, with up to two-hidden-layer, and a look back in {1, 10, 20, 30}. The selection of the number of cells and the look back was made upon the results observed in the single-hidden-layer study, the greatest variation of p_t occurs in the referred region (see Fig. 2). Note that there are 625 possible architectures, without taking into account the look back.

We repeated the MAE random sampling for each architecture, i.e. we took 100 samples, and estimated $p_{0.01}$. Fig. 3 presents the estimated $p_{0.01}$ smoothed using a logarithmic model (to ease the visualization of trends). The x-axis shows the number of cells in the first hidden layer, the y-axis represents the estimated probability, the different colors represent the number of cells in the second hidden layer, and LB stands for the look back. A value equal to 0 in the second hidden layer implies that the RNN is a single-hidden-layer one.

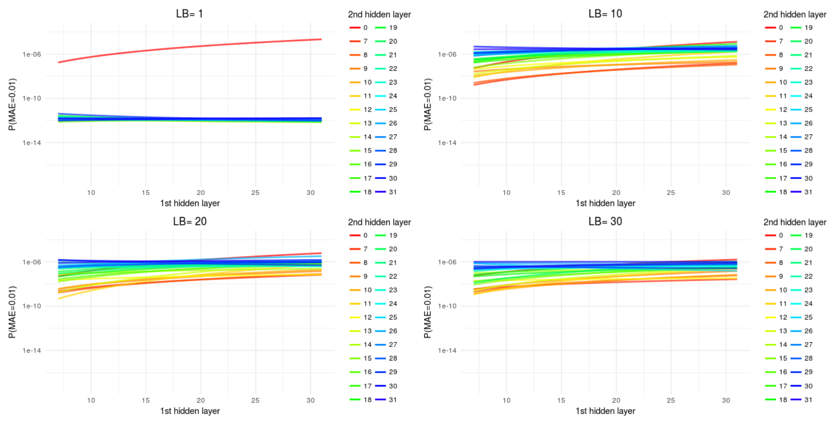

Fig. 3. Probability of finding a *good* set of weights for two-hidden-layer RNNs

We uniformly selected 100 architectures and trained them using Adam optimizer [13]. Table 2 presents the correlation between the MAE random sampling results (Mean, Sd, log $p_{0.01}$) and the MAE observed after training the model. The table also presents the mean (Mean MAE) and standard deviation (Sd MAE) values of the observed MAE.

Table 2. Correlation between the MAE observed after training the model and the MAE random sampling results (two-hidden-layer)

Epochs	*Correlation*			*Results*	
	Mean	Sd	log $p_{0.01}$	Mean MAE	Sd MAE
1	-0.086	-0.135	-0.171	0.617	0.009
10	-0.450	-0.632	-0.635	0.591	0.018
100	-0.709	-0.827	-0.905	0.147	0.145
1000	-0.695	-0.843	-0.922	0.106	0.147

Again, the results show a string negative correlation between the estimated probability and the MAE observer after training. In this case, two things gain our attention: first, the single-hidden-layer architectures outperforms the two-hidden-layer ones on average in terms of the observed MAE, and second the correlation is higher for two-hidden-layer architectures (refer to Table 1 and 2). Intuitively we suspect that both insights are related and both can be explained by the fact that two-hidden-layer architectures are more complex than single-layer ones. Therefore, the optimizing algorithm has to struggle harder to find a *good* set of weights as the number of hidden layers increases.

4.3 Three-Hidden-Layer Architectures

We added a third hidden layer to the search space (with 7 to 31 LSTM cells on it). We performed the MAE random sampling for all the architectures (15625 architectures, without considering the look back variations) and estimated $p_{0.01}$.

Afterwards, we uniformly selected 100 three-hidden-layer architectures and trained them using Adam optimizer [13]. Table 3 presents the correlation between the MAE random sampling results (Mean, Sd, log $p_{0.01}$) and the MAE observed after training the model. The table also presents the mean (Mean MAE) and standard deviation (Sd MAE) values of the observed MAE.

Table 3. Correlation between the MAE observed after training the model and the MAE random sampling results (three-hidden-layer)

Epochs	*Correlation*			*Results*	
	Mean	Sd	log $p_{0.01}$	Mean MAE	Sd MAE
1	-0.334	-0.447	-0.475	0.616	0.003
10	-0.546	-0.724	-0.745	0.605	0.010
100	-0.720	-0.869	-0.906	0.180	0.159
1000	-0.130	-0.873	-0.911	0.143	0.164

The results follow the same trends highlighted in the two-hidden-layer study. The correlation between the observed MAE and $p_{0.01}$ suggests that for this specific scenario the probability acts as a proxy of the expected performance. Moreover, the results indicate that as the problem gets more complex, the probability is even more correlated to the actual performance.

4.4 Memory and Time Comparison

Finally, after showing the usefulness of the MAE random sampling to compare stacked RNN architectures in terms of the expected performance, we studied the time and memory needed to perform the referred sampling.

We analyzed the execution logs and extracted the memory and time consumed by each process. It is important to notice that every process was executed using similar hardware and software configurations. Table 4 summarizes the time and memory usage. Despite the simplicity of the input, there is notable difference between the time needed to train an RNN and to perform a MAE random sampling. On the other hand, there is only a small difference in the memory required. Due to the *low-cost* of sampling an RNN and the usefulness of the approach for comparing architectures, we believe that using the MAE random sampling is worthwhile.

Table 4. Time and memory usage comparison

	Mean time [s]	Sd time	Mean mem [MB]	Sd mem
Training 1000 epochs	996	0.006	127	6.338
MAE random sampling	6	0.001	150	98.264

5 Conclusions and Future Work

In this study we extend the MAE random sampling technique [8] to multiple-hidden-layer architectures. Given an RNN architecture and an input time series, we generate a set of normally distributed weights and compute the MAE (the *samples*). Then, we fit a truncated normal distribution to the MAE samples and estimate the probability of finding a set of weights whose MAE is below an arbitrary threshold.

We test our proposal on stacked RNN architectures with up to three-hidden-layers, using a sine wave. The results show that there is a strong negative correlation between the estimated probability and the MAE measured after training the model using Adam optimizer. Moreover, as we add hidden-layers to the RNN, the correlation between the probability and the MAE measured increases. We think that this might be explained in part by the increasing complexity of the training process, however further analysis is required to explain this observation.

Overall, the results suggest that the MAE random sampling is a "*low-cost*, *training-free*, rule of thumb" method to compare deep RNN architectures.

As future work we propose to validate the results of this study using other time series and to explore a broader hyper-parameter search space.

Acknowledgements

This research was partially funded by Ministerio de Economía, Industria y Competitividad, Gobierno de España, and European Regional Development Fund grant numbers TIN2016-81766-REDT (http://cirti.es) and TIN2017-88213-R (http://6city.lcc.uma.es). Universidad de Málaga, Andalucía TECH.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)
2. Albelwi, S., Mahmood, A.: A framework for designing the architectures of deep convolutional neural networks. Entropy **19**(6), 242 (2017)
3. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5**(2), 157–166 (1994)
4. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 24, pp. 2546–2554. Curran Associates, Inc. (2011)

5. Bracewell, R.N., Bracewell, R.N.: The Fourier transform and its applications, vol. 31999. McGraw-Hill New York (1986)
6. Camero, A., Toutouh, J., Stolfi, D.H., Alba, E.: Evolutionary deep learning for car park occupancy prediction in smart cities. In: Learning and Intelligent OptimizatioN (LION) 12. pp. 1–15. Springer (2018)
7. Camero, A., Toutouh, J., Alba, E.: DLOPT: Deep learning optimization library. arXiv preprint arXiv:1807.03523 (july 2018)
8. Camero, A., Toutouh, J., Alba, E.: Low-cost recurrent neural network expected performance evaluation. arXiv preprint arXiv:1805.07159 (may 2018)
9. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Proceedings of the 24th International Conference on Artificial Intelligence. pp. 3460–3468. IJCAI'15, AAAI Press (2015)
10. Haykin, S.: Neural networks and learning machines, vol. 3. Pearson Upper Saddle River, NJ, USA: (2009)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
12. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37. pp. 2342–2350. ICML'15, JMLR.org (2015)
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
14. Kolen, J.F., Kremer, S.C.: Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, pp. 464–479. Wiley-IEEE Press (2001)
15. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436 (2015)
16. Lipton, Z.C., Berkowitz, J., Elkan, C.: A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019 (2015)
17. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., van der Laak, J.A., van Ginneken, B., Sánchez, C.I.: A survey on deep learning in medical image analysis. Medical image analysis **42**, 60–88 (2017)
18. Min, S., Lee, B., Yoon, S.: Deep learning in bioinformatics. Briefings in bioinformatics **18**(5), 851–869 (2017)
19. Ojha, V.K., Abraham, A., Snášel, V.: Metaheuristic design of feedforward neural networks: A review of two decades of research. Engineering Applications of Artificial Intelligence **60**, 97–116 (2017)
20. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28. pp. III–1310–III–1318. ICML'13, JMLR.org (2013)
21. Ramos, E.Z., Nakakuni, M., Yfantis, E.: Quantitative measures to evaluate neural network weight initialization strategies. In: 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC). pp. 1–7 (2017)
22. Smithson, S.C., Yang, G., Gross, W.J., Meyer, B.H.: Neural networks designing neural networks: multi-objective hyper-parameter optimization. In: Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on. pp. 1–8. IEEE (2016)