DOI: 10.1002/minf.201600045

CGBVS-DNN: Prediction of Compound-protein Interactions Based on Deep Learning

Masatoshi Hamanaka,[a] Kei Taneishi,[b] Hiroaki Iwata,[c] Jun Ye,[d] Jianguo Pei,[d] Jinlong Hou,[d] and Yasushi Okuno*[a]

Abstract: Computational prediction of compound-protein interactions (CPIs) is of great importance for drug design as the first step in in-silico screening. We previously proposed chemical genomics-based virtual screening (CGBVS), which predicts CPIs by using a support vector machine (SVM). However, the CGBVS has problems when training using more than a million datasets of CPIs since SVMs require an exponential increase in the calculation time and computer memory. To solve this problem, we propose the CGBVS-DNN, in which we use deep neural networks, a kind of deep learning technique, instead of the SVM. Deep learning does not require learning all input data at once because the network can be trained with small mini-batches. Experimental results show that the CGBVS-DNN outperformed the original CGBVS with a quarter million CPIs. Results of cross-validation show that the accuracy of the CGBVS-DNN reaches up to 98.2 % ($\sigma < 0.01$) with 4 million CPIs.

Keywords: deep learning · in-silico screening · compound-protein interactions (cpis) · chemical genomics-based virtual screening (cgbvs) · support vector machine

1 Introduction

Although high-throughput screening[1] has been used for drug discovery, the number of potential compound-protein interactions (CPIs) that could be assayed is essentially infinite, so a brute-force experimental screening approach for CPIs is wasteful, because it needs a lot of time and money. Attention has thus been focused on CPI prediction models that can guide researchers to fast lanes for hit discovery.

Two categories of CPI prediction techniques have been proposed for drug development: structure based virtual screening (SBVS)[2,3] and ligand based virtual screening (LBVS).[4,5,6,7] SBVS methods estimate the likelihood that the ligand will bind to the protein by using three-dimensional structures of both ligands and proteins.[2,3] Thus, SBVS methods cannot be used for sets of ligands and proteins whose three-dimensional structures are unknown. Some LBVS methods build a model of a receptor by using a binding ligand that binds the receptor.[4] Other LBVS methods predict CPIs by using chemical or shape similarity.[5] LBVS often requires many known active molecules for a target of interest.[4,5] On the other hand, target predictions by SEA[6] or SPiDER[7] have demonstrated the high accuracy of LBVS, outperforming SBVS methods.

We previously proposed chemical genomics-based virtual screening (CGBVS), which is based on a support vector machine.[8] The main advantage of using CGBVS is that it enables CPIs of novel ligands without known three-dimensional structures to be predicted. CGBVS shows high performance in the cross-validation of over 250 thousand datasets containing 125 thousand positive interaction datasets we collected and 125 thousand negative datasets we artificially generated. However, the size of available datasets can be in the millions, and since SVMs require an exponential increase in resources, no model can be feasibly constructed on so many datasets.

In light of this, we have investigated the ability of deep-layered neural networks, also known as Deep learning, to handle such large volumes of CPIs that cannot be readily processed by SVMs. In this modelling process, many input and output layers are chained together before a final prediction layer is output.

In the case of machine learning with large-scale CPIs, the following points can be considered.

[a] M. Hamanaka, Y. Okuno
Graduate School of Medicine, Kyoto University
Shogoin-kawaharacho, city/>Sakyo-ku Kyoto 606-8507, Japan,
Tel: +81-75-751-4881
Fax: +81-75-751-4881
*e-mail: okuno.yasushi.4c@kyoto-u.ac.jp

[b] K. Taneishi
Advanced Institute for Computational Science, RIKEN
7-1-28, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan

[c] H. Iwata
Foundation for Biomedical Research and Innovation
1-6-5, Minatojima-Minamimachi Chuo-ku, Kobe 650-0047, Japan

[d] J. Ye, J. Pei, J. Hou
Software and Services Group, Intel Corporation
Shanghai, China

- Deep Learning Versus Support Vector Machine
  When learning large-scale CPIs, there are two choices of machine learning techniques: SVM and deep learning. Deep learning is better than SVM; it requires a smaller amount of memory. For example, deep learning does not require learning on all input data at once as in the standard SVM, but rather, the model is generatively tuned over the course of data input. The calculation time when using deep learning depends on the structure of the network but is clearly shorter than SVM.
  In Section 3.2, we compare the performances of deep learning and SVM with a data set of 250 thousand CPIs.
- CPU Versus GPGPU
  There are two types of calculation with deep learning: one is on a central processing unit (CPU), and the other is on a general-purpose computation on graphics processing unit (GPGPU). The GPGPU accelerates the calculation by using thousands of cores. However, because the size of the memory on it is limited, the calculation time is much longer depending on the overhead of data transfer if the size of data is bigger than the memory.
  We think calculation on a CPU is more suitable than that on a GPGPU when dealing with large scale data because the memory on a CPU machine is easy to enlarge. However, the mathematical libraries for deep learning are not optimized for CPUs.
  In Section 2.5, we optimize the Python[9] mathematical library called *Theano*[10] for CPUs and then benchmark the performances of the CPU and GPGPU.
- Deep Learning Algorithms and Software
  Several deep learning algorithms have been proposed, such as the convolution neural network (CNN)[11] recurrent neural network (RNN),[12] and deep neural network (DNN) such as deep belief network (DBN).[13] Image data is fit at the RNN and time-series data is fit at the DNN. We selected the DNN as the deep learning algorithm because CPI data has features that are different from image or time-series data and the DNN is compatible with various types of data. Several libraries and software for the DNN have been provided.[10,14,15,16] We use Theano because it is a widely used library and can use both the CPU and GPGPU.
- Pre-training
  Generally, unsupervised pre-training[17] helps deep learning, because a randomly initialized deep layered network often does not pass any information, so back propagation[18] is difficult to do. However, in practice, unsupervised pre-training is not necessary when the number of labelled training samples is large enough and has a long enough training time.
  In Section 3.2, we compare the performance of prediction with pre-training and without pre-training.
- Structures of Network
  Deep learning performs differently depending on the network structure. When the deep layered network is too simple, the prediction performance is low because the network has a low representation ability. When the network is too complex, it is prone to overfitting because when the network has a high representation ability, it needs a large amount of data.
  Therefore, we have to design a network structure that has a good enough representation ability and will not be overfitted by considering both the quantity and property of datasets.
  In Section 3.1, we test dozens of kinds of network structures by changing the numbers of units and layers, compare their prediction performances, and discuss the best network structures.
- Hyperparameters
  Deep learning has hyperparameters such as mini-batch size and learning rate. These parameters must be tuned because the convergence speed and stability of convergence depend on the values of parameters and the proper values of hyperparameters differ depending on the learning task.
  These parameters can be optimized by using Bayesian optimization[19] or random sampling.[20] However, in our case, a large-scale search is unnecessary because the initial values of the hyperparameters that we decided show better prediction performance.
  In Section 3.4, we change the hyperparameters of several values near the initial values and compare the prediction performances.

The organization of the paper is as follows. In Section 2.1, we mention related works, and in Section 2.2, we briefly describe the data of CPIs. In Sections 2.3 and 2.4, we explain our previous method called "CGBVS" and a novel method called "CGBVS-DNN." We briefly explain the optimization of *Theano* in Section 2.5 and evaluate the prediction performance in Chapter 3. Finally, in Chapter 4, we conclude with a summary and an overview of future work.

2 Methods

2.1 Related Works

We briefly look at related works that used deep learning to predict CPIs.

Several groups have attempted to use deep learning for drug discovery.[21,22,23,24] A team using deep learning won the competition for predicting CPIs called the "Merck Molecular Activity Challenge.[21]" Although the team showed the potential of Multi-task Neural Networks for predicting CPIs, they only used 15 molecular targets and could not show the full ability of deep learning because the capacity of the deep learning increases as data increases. Large pharma data are also used for learning in a multitask neural network.[22,23] The input of the network is molecular descriptors of compounds, and each output corresponds to each molecular target of a classification task[21,22,23]. For example, one group used 2 million CPIs from the ChEMBL database.[22] They showed the performance gains due to the multitask neural networks. Another group used nearly 40 million CPIs.[23] They investigated the relationship between the numbers of tasks and the performance.

Our method called "CGBVS-DNN" can predict the interaction between a novel ligand and/or a novel protein because the input of the deep layered network is both ligand and protein descriptors.

2.2 Data of CPIs

We collected more than 2 million pieces of positive interaction data between compounds and proteins in the family of G-protein coupled receptors (GPCR) from the GVK-BIO[25] database and the ChEMBL database[26]. Compound-protein interactions (CPIs) represent a credible data set because only interactions with relatively high affinities (Ki, EC50, and IC50 < 30 μM). We artificially generated negative data because most data that we can acquire are positive. After generating the negative data, we made CPI data by concatenating the positive and negative datasets and randomly shuffled the order. Then, we split the data: four fifths for learning data and one fifth for test data.

As a result of removing duplication, we collected more than 2 million interactions. After deciding the number of datasets to measure, we randomly extracted half of that number from the 2 million interactions. For example, when we constructed 250 thousand datasets, we extracted 125 thousand positive CPIs and generated 125 thousand negative CPIs.

In total, we obtained a 1974-dimensional feature vector consisting of 894-dimensional chemical descriptors and 1,080 protein descriptors after we quantified the chemical structure and protein sequence by using Dragon[27] and PROFEAT.[28] We called the positive datasets the "sets of the 1974-dimensional feature vector" and labelled them as binding.

Figure 1 shows the flow of how to generate the negative data. First, we randomly shuffle the positive dataset and pick out two positive interactions from top to bottom (Fig. 1a, b). Then, we change the combinations between chemicals and proteins (Fig. 1c). If the changed combinations do not overlap with either positive or negative data, the combinations are added to the negative data, and then the next two interactions are picked out. Otherwise, the overlapped one and the next interaction are picked out, and the combinations change (Fig. 1d, e). We call the negative datasets the "sets of the 1974-dimensional feature vector from a generated set of compounds and protein" and label them as non-binding. Our method described in Section 2.4 can use other descriptors. We use Dragon and PROFEAT as the benchmark of the prediction performance because our previous work described in Section 2.3[8] used them. Those descriptor vectors were separately scaled to the range −1 to 1.

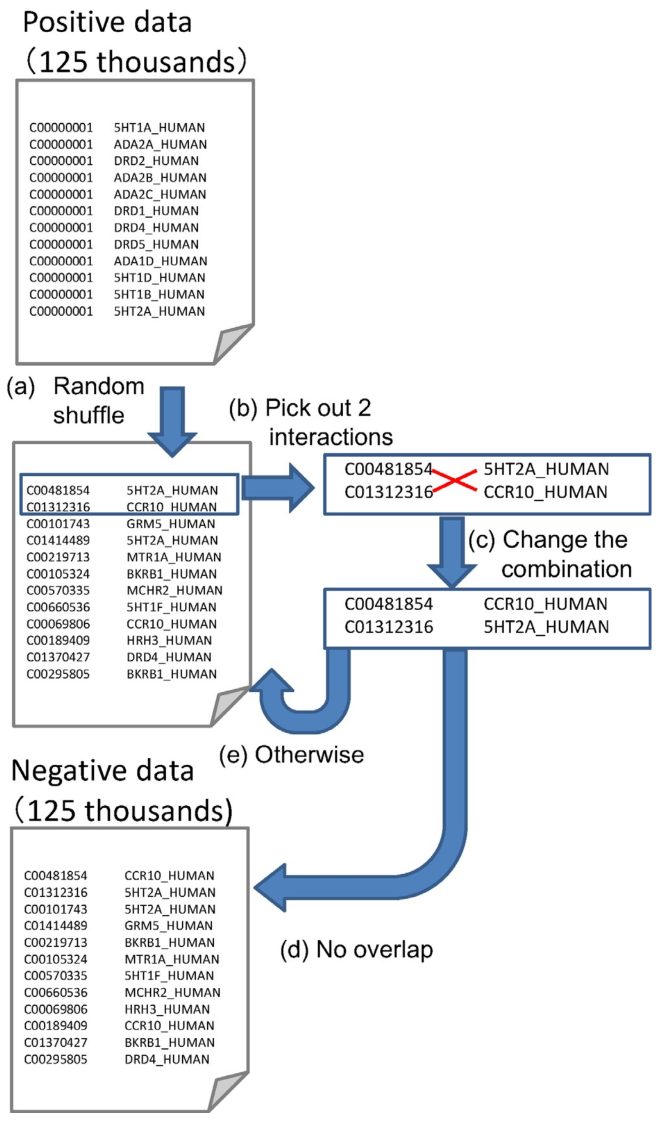

Figure 1. Generation of negative data.

For all of the experiment data in Chapter 3, we split the data into four fifths for learning data and one fifth for test data. The ratio of positive and negative data was one to one for both the training and test data.

2.3 CGBVS: Chemical Genomics-based Virtual Screening

We propose CGBVS-DNN, which is a machine learning technique converted from SVM to DNN. Thus, the concepts of CGBVS and CGBVS-DNN are the same. Figure 2 shows a processing flow of CGBVS. As described in Section 2.2, we quantified the chemical structure and protein sequence by using Dragon and PROFEAT (Fig. 2b). The support vector machine models learned using interaction vectors with bind or non-bind labels (Fig. 2d). Finally, when we input a new pair of chemical and protein, the model predicts if it will be binding or non-binding (Fig. 2e).

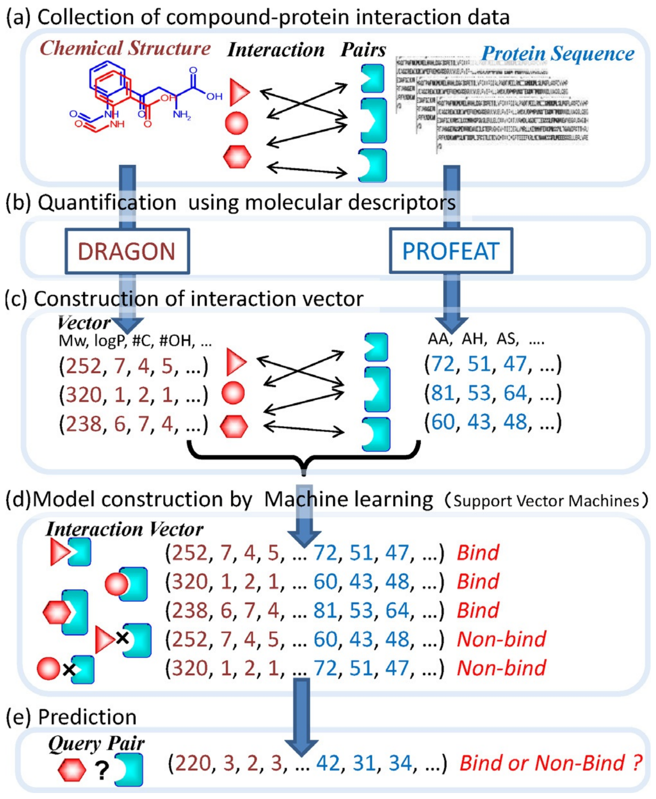
(a) Collection of compound-protein interaction data
Chemical Structure
Interaction Pairs
Protein Sequence
(b) Quantification using molecular descriptors
DRAGON
PROFEAT
(c) Construction of interaction vector
Vector
Mw, logP, #C, #OH, ...
(252, 7, 4, 5, ...)
(320, 1, 2, 1, ...)
(238, 6, 7, 4, ...)
AA, AH, AS, ...
(72, 51, 47, ...)
(81, 53, 64, ...)
(60, 43, 48, ...)
(d)Model construction by Machine learning (Support Vector Machines)
Interaction Vector
(252, 7, 4, 5, ... 72, 51, 47, ...) Bind
(320, 1, 2, 1, ... 60, 43, 48, ...) Bind
(238, 6, 7, 4, ... 81, 53, 64, ...) Bind
(252, 7, 4, 5, ... 60, 43, 48, ...) Non-bind
(320, 1, 2, 1, ... 72, 51, 47, ...) Non-bind
(e) Prediction
Query Pair
(220, 3, 2, 3, ... 42, 31, 34, ...) Bind or Non-Bind ?

Figure 2. CGBVS: chemical genomics-based virtual screening.

2.4 CGBVS-DNN: Chemical Genomics-based Virtual Screening Using Deep Neural Network

The CGBVS-DNN uses a deep neural network (DNN)[13] for machine learning. The unsupervised training called "pre-training" helps to learn a DNN. Figure 3 shows the structure of the DBN that we use. The input of the DBN is 1974 units corresponding to the 1974 dimensional vectors from chemical and protein descriptors, in which each dimension is normalized by zero mean and variance of one. The output of the DBN is two units such as non-bind (=0) and bind (=1).

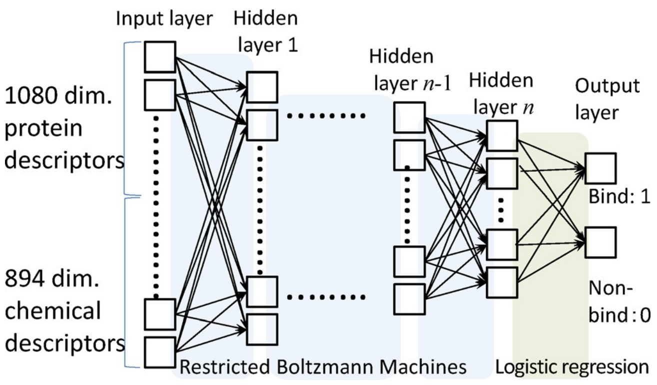

Figure 3. Deep belief networks.

Unsupervised pre-training is done by stacking restricted Boltzmann machines (RBMs)[26] from the input layer to the final hidden layer beside the output layer. We have several choices for connecting the final hidden layer to the output layer, such as SVM or logistic regression.[29] In the experiment in chapter 3, we use logistic regression. Finally, the network has supervised fine-tuning for all of the parameters of the network such as the bias of all of the units and the weight of all of the connections between units applied by back propagation.

2.5 Optimization of *Theano* for CPU

When using a CPU, a large dataset can be easily held in memory without the performance and energy overhead of loading it from a disk repeatedly. However, the mathematical libraries for deep learning are not optimized for CPUs. In this chapter, we briefly describe the optimization of a Python mathematical library called "*Theano*" for the CPU.

We optimized *Theano* with the following points.

- Thread-level Parallelism
  To effectively use CPU cores, we make a fine-grained parallel thread from one process, because the original *Theano* cannot effectively use CPU resources. For example, stochastic gradient decent is used for both pre-training and fine-tuning a DNN, and the process can be a fine-grained parallel thread.
- Instruction-level Parallelism
  A superscalar processor enables more than one instruction to be executed using multiple parallel pipelines. To utilize the superscalar pipeline, the compiler extracts an instruction-level parallelism from a single thread task. This optimization can be done by optimizing the compiler.
- Manual Assembly
  Generally, manually tuned manual assembly code outperforms compiler optimized code. After the code is compiled, there are some bottleneck operation codes. We find these codes and replace them with faster manual assembly codes.
- Vectorization of Loops
  When improving the performance, we vectorize a loop in a code to utilize 256-bit advanced vector extensions (AVX). The AVX is a set of CPU built-in instructions for doing single instruction multiple data (SIMD).
- Software Prefetch
  If the data to be used is not loaded yet, the idle time of the CPU will increase because there is latency in loading the data. To hide the latency for memory access, software prefetch loads the data to cache the memory of the CPU before executing the instruction.

3 Results and Discussion

In the experiments, we did four types of evaluation. The first one was to evaluate dozens of structures of DNNs and compare the prediction performance with that of a small-scale dataset. The second one was to evaluate the effectiveness of pre-training with a medium-scale dataset. Then, we evaluated the performance when the data size was changed from medium to large. Finally, we evaluated the performance when changing the value of the hyperparameters of DNNs. To make the deep learning model, we used *Theano*. For all of the experiment data in this section, we split the data into four fifths for learning data and one fifth for test data. All of the evaluations in this section are open; that is, all of the accuracy in the figures and table are the accuracy of the test data. The ratio of positive and negative data was one to one for both training and test data.

3.1 Structure of DNNs

We evaluated the performance of DNNs by changing the numbers of units and layers. There were 3 numbers of units (1000, 2000, and 3000) and 1 to 9 layers. For example, Fig. 4 shows a DNN that has 3 hidden layers, each of which has 3000 units.

Figure 5 shows the results of changing the numbers of layers and units with datasets of 50 thousand CPIs. In these figures, the horizontal axes indicate the epochs of the fine-tuning after 100 epochs of pre-training, and the vertical axes indicate accuracy as follows.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \qquad (1)$$

where: TP is True Positive, TN is True Negative, FP is False Positive, FN is False Negative.

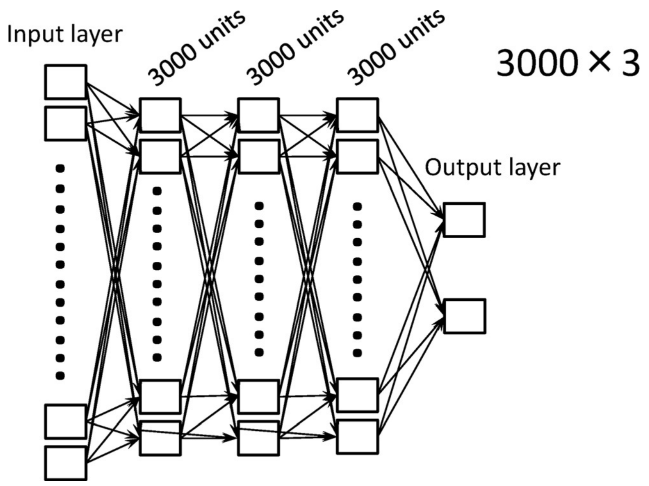

Figure 4. DNN with three layers of 3000 units (3000×3).

Figures 5a, b, and c show the results of the networks where each layer had 1000, 2000, and 3000 units.

The figures are plotted for the best performances. Table 1 lists the mean and standard deviation of the best performances of each network structure as the results of

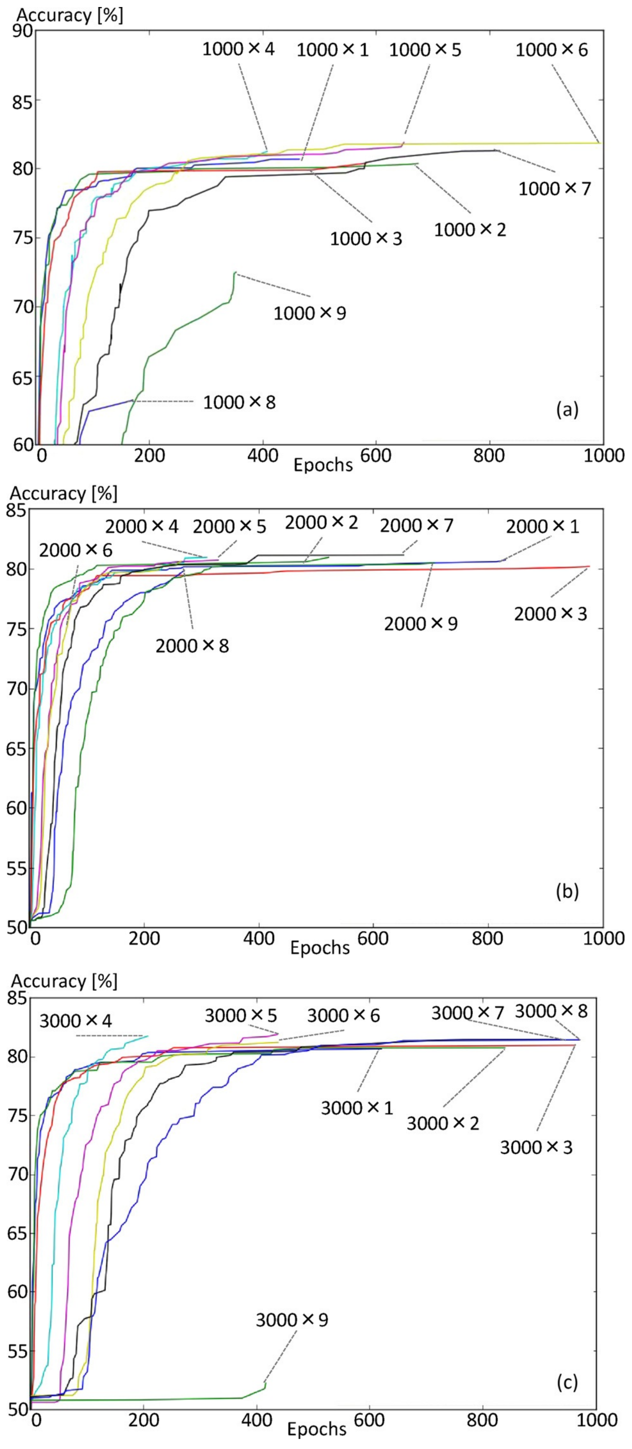

Figure 5. Learning curves of each network structure. (a) 1000xn: each layer has 1000 units; (b) 2000xn: each layer has 2000 units; 3000xn: each layer has 3000 units.

Table 1. Mean and standard deviation of best accuracy of each network structure

Numbers of Layers	Numbers of Units		
	1000	2000	3000
1	80.7 % (0.53)	80.8 % (0.48)	80.7 % (0.60)
2	80.4 % (0.68)	81.0 % (0.73)	80.8 % (0.41)
3	80.4 % (0.82)	80.2 % (0.64)	81.0 % (0.46)
4	81.3 % (0.53)	81.0 % (0.50)	81.8 % (0.68)
5	82.0 % (0.66)	80.8 % (0.38)	82.0 % (0.21)
6	81.9 % (0.59)	80.7 % (0.36)	81.3 % (0.56)
7	81.4 % (0.30)	81.2 % (0.46)	81.5 % (0.44)
8	63.3 % (2.65)	79.9 % (0.47)	81.5 % (0.66)
9	72.5 % (0.71)	80.5 % (0.44)	52.3 % (1.23)

a 5-fold cross validation. From the results of the evaluation, most of the networks showed a similar performance. However, some networks performed very low, such as eight 1000-unit layers, nine 1000-unit layers, and nine 3000-unit layers. Overall 2000- and 3000-unit layers outperformed 1000-unit layers.

Therefore, if there are too many layers, the performance did not improve because it was difficult for the information to propagate from input to output through each layer. In comparison, if the network was complex enough and also had enough data, the performance improved. We use a deep neural network of three 2000-unit layers in 3.2, 3.3, and 3.4 because it performed well and had a short calculation time.

3.2 Effectiveness of Pre-training and Benchmark with SVM

Generally, unsupervised pre-training is effective at preventing over-fitting when labeled data is scarce. However, it is not needed when labeled data is abundant.[17] Figure 6 shows the results of our evaluation with datasets of a quarter of a million CPIs. The blue and green lines indicate the performances with a hundred epochs and without (with zero epoch) pre-training, respectively. The figures are plotted for the best performances. The dotted line indicates the baseline performance of CGBVS with the SVM. Therefore, in our dataset, the CGBVS-DNN outperformed CGBVS, and fine-tuning without pre-training performed better than that with pre-training.

3.3 Evaluation with Large-scale Data

We used *Theano*[10] to make a deep learning model. The GPGPU accelerates the computing time of code written

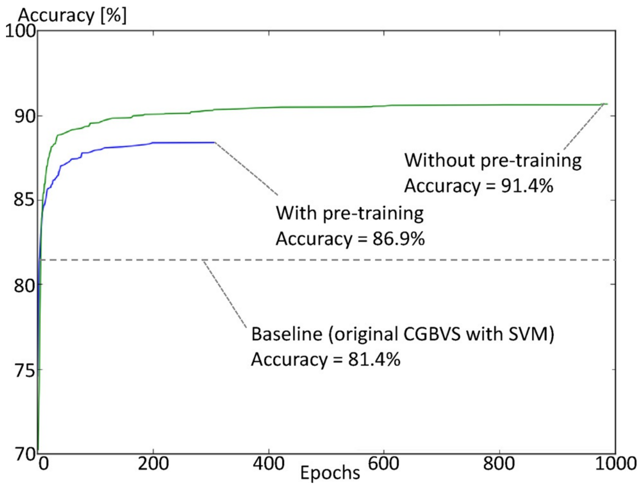

Figure 6. Fine-tuning with and without pre-training.

with *Theano*. However, when the size of data is increased, the learning time for one epoch becomes longer in proportion to the size. For example, if the calculation time of 1000 epochs with 250 thousend datasets is 1 week, 16 weeks are needed for 1000 epochs with 4 million datasets. Moreover, the calculation time is much longer depending on the overhead of data transfer if the size of data is bigger than the memory because the size of memory on a GPGPU is limited.

To deal with large-scale data, we attempted to optimize *Theano* for an Intel CPU. Table 2 shows the results of the optimization comparing Intel Haswell-EP (Xeon E5-2699v3 × 2 with 128 GB of memory) and Nvidia Tesla (K40 hosted by Ivy Bridge). In the pre-training, the Haswell-EP ran 2.4 times faster than Tesla. In the fine-tuning, Haswell-EP ran 1.8 times faster than Tesla. Also, the Haswell-EP was able to run almost 13 times as many datasets as Tesla.

Figure 7 shows the results of using the optimized Theano after a hundred epochs pre-training. The vertical axis indicates accuracy, which is the difference between the error rate of test data and one.

When we used half a million datasets, the accuracy was 92.9 %. This is higher than the result of CGBVS (91.4 %) based on a support vector machine. Learning the DNN took 1.1 days. The highest performances when we used 1 and 2 million datasets were 95.2 % and 96.9 %, respectively. When learning with 4 million datasets, the accuracy took 8.8 days to maximize at 98.1 %.

Table 2. Comparison of Intel Haswell-EP using optimized Theano and Nvidia Tesla

	Haswell	Tesla	Outperform
Pre-training(per epoch)	59 sec.	143 sec.	×2.4 faster
Fine tuning (per epoch)	47 sec.	88 sec.	×1.8 faster
Max supported dataset size	16 million CPIs	1.25 million CPIs	×12.8 larger

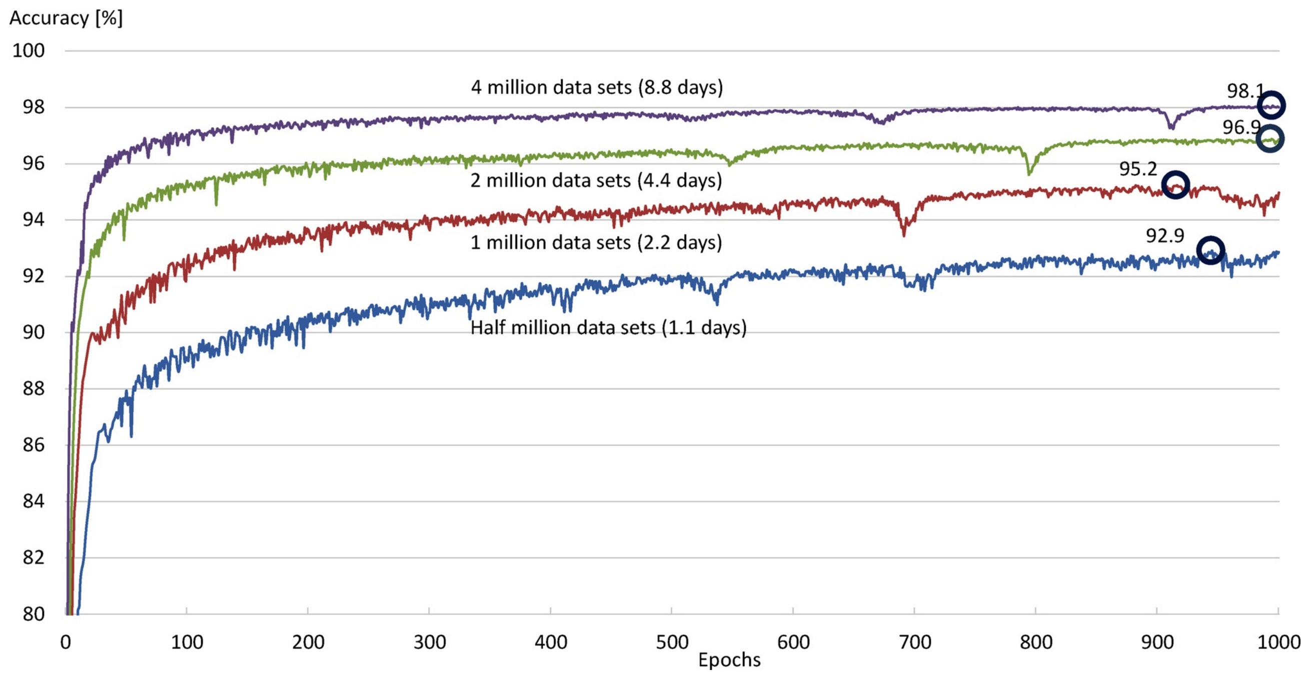

Figure 7. Evaluation with large-scale datasets.

3.4 Evaluating Shuffled Mini-batches and Tuning of Hyperparameters

In the evaluations in Sections 3.1, 3.2, and 3.3, we used mini-batches[30] in order. That is, we first shuffled the learning data and then separated them as mini-batches. We used the size 10 mini-batches. In this chapter, we compare the performances of in-order and shuffled mini-batches. A shuffled mini-batch means that, in each epoch, the order of the mini-batch was randomly shuffled. Therefore, the network learns a different order for every epoch. We also tune the hyperparameters of networks by changing several values and compare the prediction performances.

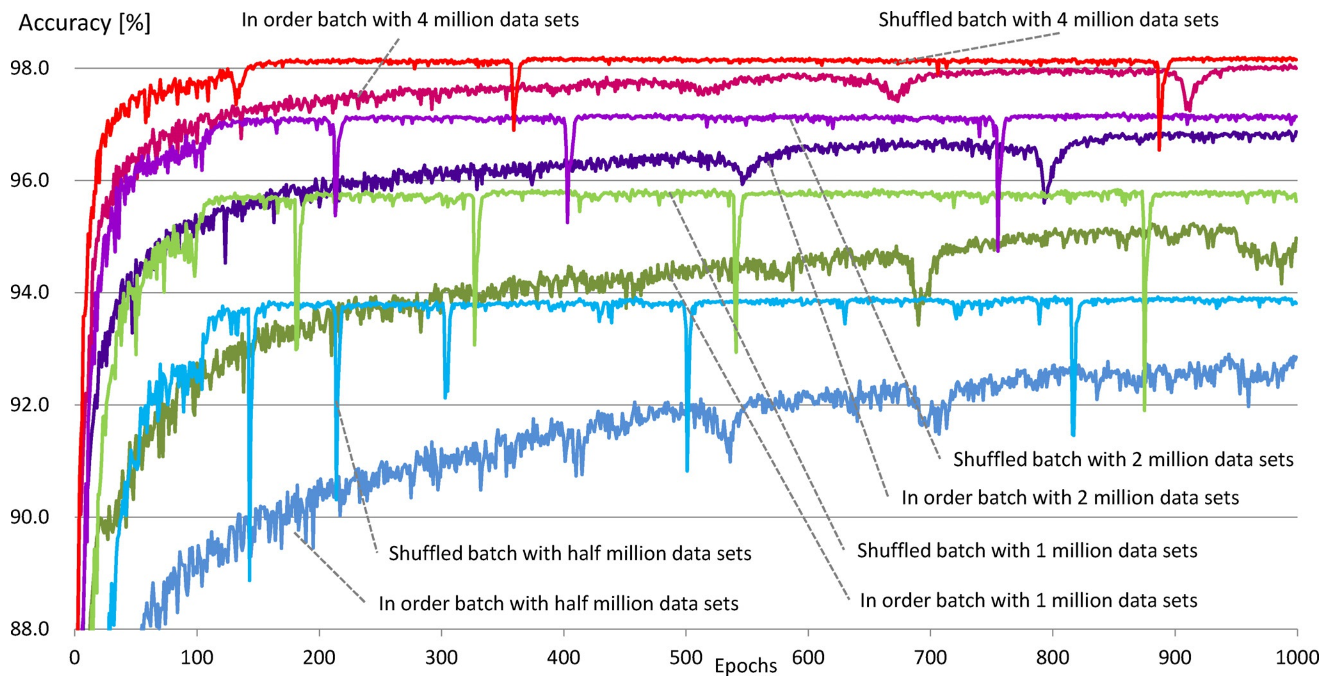

Figure 8. Shuffled batch results.

Shuffled Batch Study

The training data accuracy was completely the same in both configurations of in-order and shuffled batches. In comparison, in the test data, the shuffled batch had both a fast convergence speed and much improved accuracy (Fig. 8).

Batch Size Study (with shuffled batches)

Theoretically, a smaller batch size will increase training accuracy and a larger batch size will increase processor efficiency. Moreover, the batch cannot be smaller than the category size; otherwise, the system will not converge. There are two categories in our case: binding and non-binding.

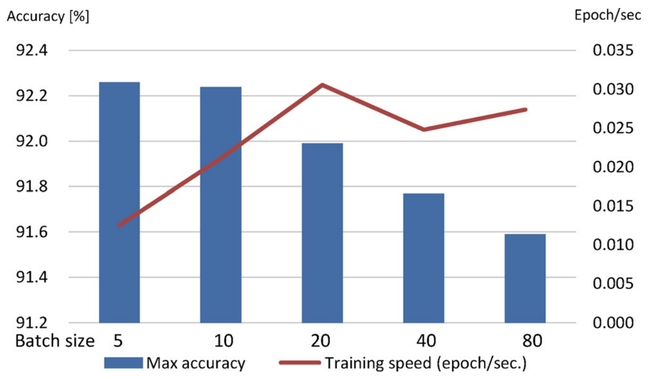

Figure 9. Accuracy and epoch/sec for each batch size.

We evaluated five kinds of batch sizes: 5, 10, 20, 40, and 80. Figures 9 and 10 show the results of the evaluation with up to four million datasets. The size 5 batch achieved the best test data accuracy; however, it took a very long time for training. In comparison, the calculation time of the size 20 batch was the fastest; however, it had worse accuracy than the size 5 and 10 batches. Therefore, size 10 was the best batch size because it achieved both good accuracy and a short network training time.

Learning Rate Study (with shuffled batch, batch size = 10)

So far, we have used the fixed learning rate of 0.1. Figure 11 shows the results of using three kinds of learning rates, 0.05, 0.1, and 0.2, with a quarter of a million datasets. From the results, the learning rate of 0.2 showed the best accuracy, with 0.1 the next best and 0.05 the worst. The higher learning rate leads to a faster convergence speed but may have caused a large swing.

4 Conclusions

In this paper, we proposed a method called "CGBVS-DNN" that predicts CPIs from descriptors of ligands and proteins by using a deep learning technique.

Results of an experiment revealed the following six findings.

- Deep learning achieved an accuracy of 98.2% ($\sigma < 0.01$) with shuffled mini-batches with 4 million datasets.

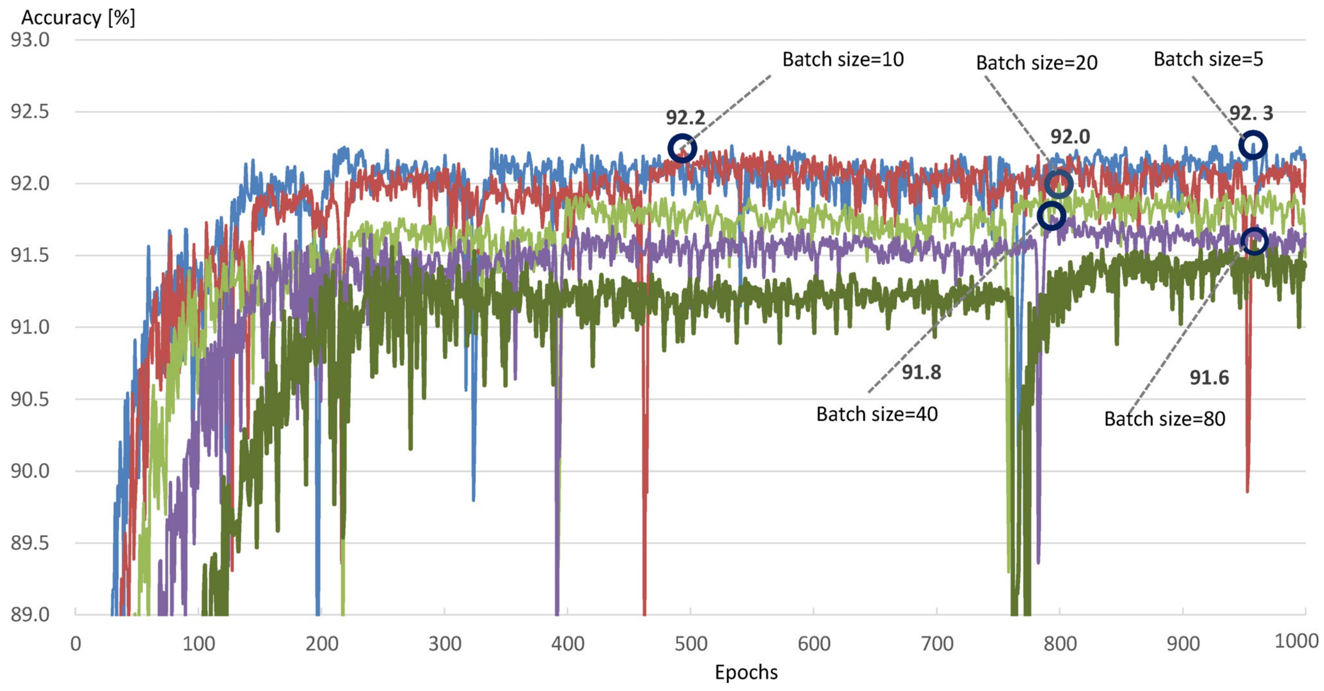

Figure 10. Batch size results.

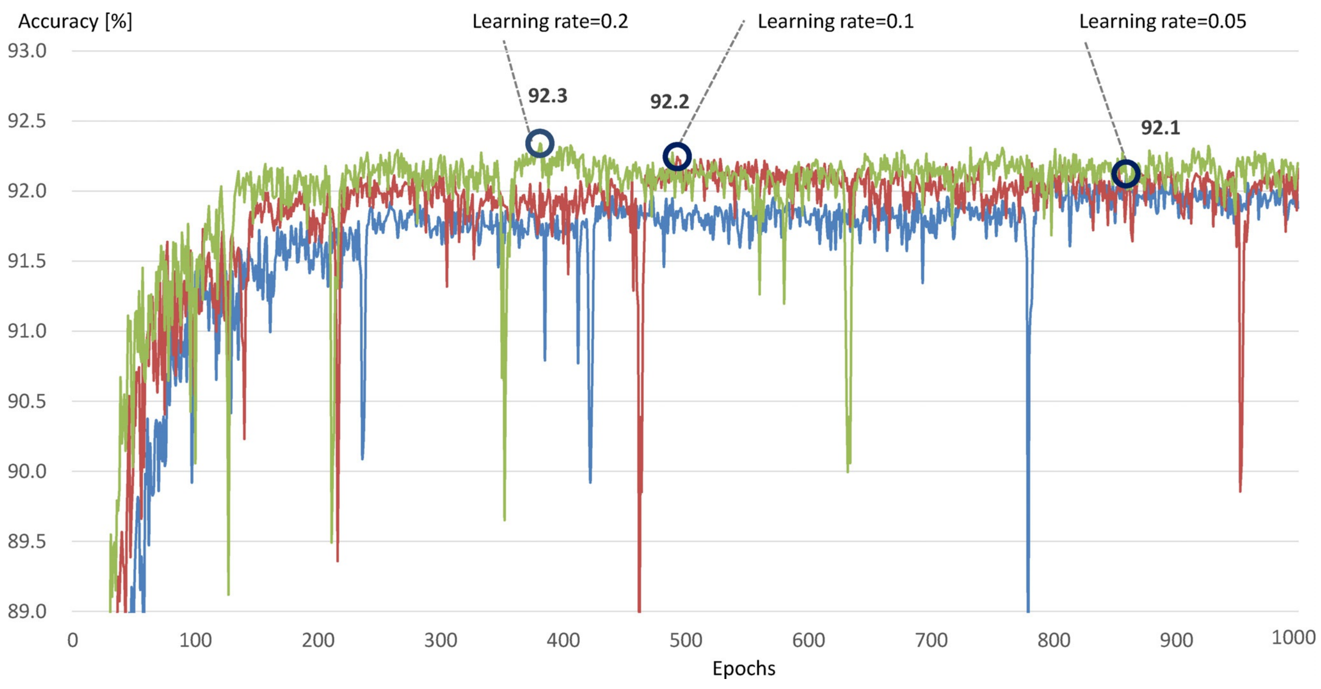

Figure 11. Learning rate results.

- CGBVS-DNN outperformed the original CGBVS with a quarter of a million CPIs.
- From the results of our dataset, the Intel Haswell-EP (Xeon E5-2699v3×2 with 128 GB of memory) with optimized Theano outperformed Nvidia Tesla (K40 hosted by Ivy Bridge) in terms of both speed and max supported data size.
- In our dataset, fine-tuning without pre-training outperformed that with pre-training.

We plan to evaluate the performance of deep learning for more than 4 million datasets with kinases or other protein families and also analyze the network after fine-tuning.

Abbreviations

AVX	advanced vector extensions
CGBVS	chemical genomics-based virtual screening
CNN	convolution neural network
CPIs	compound-protein interactions
CPU	central processing unit
DBN	deep belief network
DNN	deep neural network
GPCR	G-protein coupled receptors
GPGPU	general-purpose computation on graphics processing
LBVS	ligand based virtual screening
RBM	restricted Boltzmann machine
RNN	recurrent neural network
SBVS	structure based virtual screening
SIMD	single instruction multiple data
SVM	support vector machine

Conflict of Interest

None declared.

Acknowledgements

This work was supported by CREST, Japan Science and Technology Agency, Innovative Drug Discovery Infrastructure through Functional Control of Biomolecular Systems, Priority Issue 1 in Post-K Supercomputer Development (project ID: hp150272 and hp160213), FOCUS Establishing Supercomputing Center of Excellence, and Subsidy for Kobe Biomedical Innovation Cluster promoting projects.

References

[1] X. D. Zhang, Optimal High-throughput Screening: Practical Experimental Design and Data Analysis for Genome-scale RNAi Research, Cambridge University Press, **2011**, 1–224.
[2] H. Sun, *Curr. Med. Chem.* **2008**, *15*, 1018–1024.
[3] P. Willet, J. M. Barnard, G. M. Downs, *J. Chem. Inf. Comp. Sci.* **1998**, *38*, 983–996.
[4] T. Ewing, I. Kuntz, *J. Comput. Chem.* **1998**, *18*, 1175–1189.

[5] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, A. J. Olson, *J. Comput. Chem.* **1998**, *19*, 1639–1662.

[6] M. J. Keiser, B. L. Roth, B. N. Armbruster, P. Ernsberger, J. J. Irwin, B. K. Shoichet, *Nat. Biotech.* **2007**, *25*, 197–206.

[7] D. Rekera, T. Rodriguesa, P. Schneidera, G. Schneider, *MARCH.* **2014**, *18*, 4062–4072.

[8] H. Yabuuchi, S. Niijima, H. Takematsu, T. Ida, T. Hirokawa, T. Hara, T. Ogawa, Y. Minowa, G. Tsujimoto, Y. Okuno, *Mol. Syst. Biol.* **2011**, *7*, 1–12.

[9] G. V. Rossum, *The Python Language Reference Manual*, Network Theory Ltd., **2003**, 1–121.

[10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, *Proc. 9th Python in Sci. Conf.*, **2010**, 1–7.

[11] Y. LeCun, K. Kavukcuoglu, C. Farabet, *Proc. of IEEE ISCAS*, **2010**, 253–256.

[12] H. Sak, A. W. Senior, F. Beaufays, *Proc. of Interspeech2014*, **2014**, 338–342.

[13] G. E. Hinton, S. Osindero, Y. W. Teh, *Neural Computation* **2006**, *18*, 1527–1554.

[14] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, Y. Bengio, *arXiv preprint arXiv:1308.4214*, 1–9, Available at http://arxiv.org/abs/1308.4214, 2013.

[15] J. Dean, R. Monga et al., Available at http://download.tensorflow.org/paper/whitepaper2015.pdf, 2015, 1–19.

[16] S. Tokui, K. Oono, S. Hido, J. Clayton, *NIPS 2015 deep learning workshop*, Available at http://learningsys.org/papers/LearningSys_2015_paper_33.pdf, 2015, 1–6.

[17] D. Erhan, A. Courville, Y. Bengio, P. Vincent, *Proc. of AISTATS 2010*, **2010**, 201–208.

[18] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *Nature* **1986**, *323*, 533–536.

[19] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, *Proc. of NIPS2011*, **2011**, 2546–2554.

[20] J. Bergstra, Y. Bengio, *JMLR.* **2012**, *13*, 281–305.

[21] Announcement of the winners of the Merck Molecular Activity Challenge URL: https://www.kaggle.com/c/MerckActivity/details/winners.

[22] T. Unterthiner, A. Mayr, G. Klambauer, M. Steijaert, H. Ceulemans, J. K. Wegner, S. Hochreiter, *NIPS 2014 deep learning workshop*, Available at http://www.bioinf.jku.at/publications/2014/NIPS2014c.pdf, 2014, 1–4.

[23] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, V. Pande, *Under review by the ICML2015*, *arXiv:1502.02072*, Available at http://arxiv.org/pdf/1502.02072.pdf, 2015, 1–27.

[24] E. Gawehn, J. A. Hiss, G. Schneider, *Mol. Inf.* **2016**, *35*, 3–14.

[25] GVKBIO database. URL: http://www.gvkbio.com.

[26] A. P. Bento, A. Gaulton, A. Hersey, L. J. Bellis, J. Chambers, M. Davies, F. A. Krüger, Y. Light, L. Mak, S. McGlinchey, M. Nowotka, G. Papadatos, R. Santos, J. P. Overington, *Nucleic Acids Res.* **2014**, *42*, 1083–1090.

[27] M. Andrea, C. Viviana, P. Manuela, T. Roberto, *MATCH.* **2006**, *56*, 237–248.

[28] H. B. Rao, F. Zhu, G. B. Yang, Z. R. Li, Y. R. Chen, *Nucleic Acids Res.* **2006**, *34*, 385–390.

[29] A. Agresti, *Categorical Data Analysis*. New York: Wiley-Interscience, **2002**, 1–774.

[30] M. Li, T. Zhang, Y. Chen, A. J. Smola, *Proc. of Int. conf. on Knowledge discovery and data mining*, **2014**, 661–670.

Received: March 31, 2016
Accepted: July 9, 2016
Published online: August 12, 2016