JOURNAL OF CHEMICAL INFORMATION AND MODELING

Article

pubs.acs.org/jcim

Deep Neural Nets as a Method for Quantitative Structure–Activity Relationships

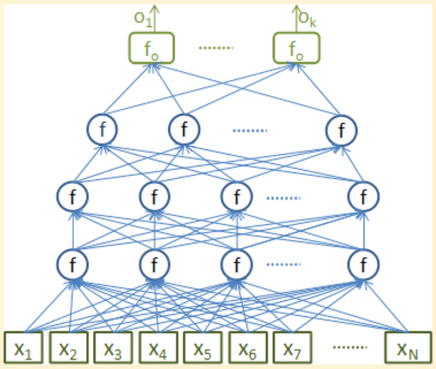
Junshui Ma,*[†] Robert P. Sheridan,[‡] Andy Liaw,[†] George E. Dahl,[§] and Vladimir Svetnik[†]

[†]Biometrics Research Department and [‡]Structural Chemistry Department, Merck Research Laboratories, Rahway, New Jersey 07065, United States

[§]Computer Science Department, University of Toronto, Toronto, Ontario ON M5S, Canada

Ⓢ *Supporting Information*

ABSTRACT: Neural networks were widely used for quantitative structure–activity relationships (QSAR) in the 1990s. Because of various practical issues (e.g., slow on large problems, difficult to train, prone to overfitting, etc.), they were superseded by more robust methods like support vector machine (SVM) and random forest (RF), which arose in the early 2000s. The last 10 years has witnessed a revival of neural networks in the machine learning community thanks to new methods for preventing overfitting, more efficient training algorithms, and advancements in computer hardware. In particular, deep neural nets (DNNs), i.e. neural nets with more than one hidden layer, have found great successes in many applications, such as computer vision and natural language processing. Here we show that DNNs can routinely make better prospective predictions than RF on a set of large diverse QSAR data sets that are taken from Merck's drug discovery effort. The number of adjustable parameters needed for DNNs is fairly large, but our results show that it is not necessary to optimize them for individual data sets, and a single set of recommended parameters can achieve better performance than RF for most of the data sets we studied. The usefulness of the parameters is demonstrated on additional data sets not used in the calibration. Although training DNNs is still computationally intensive, using graphical processing units (GPUs) can make this issue manageable.



INTRODUCTION

Quantitative structure–activity relationships (QSAR) is a very commonly used technique in the pharmaceutical industry for predicting on-target and off-target activities. Such predictions help prioritize the experiments during the drug discovery process and, it is hoped, will substantially reduce the experimental work that needs to be done. In a drug discovery environment, QSAR is often used to prioritized large numbers of compounds, and in that case the importance of having each individual prediction be accurate is lessened. Thus, models with predictive R^2 of as low as ~0.3 can still be quite useful. That said, higher prediction accuracy is always desirable. However, there are practical constraints on the QSAR methods that might be used. For example

1. QSAR data sets in an industrial environment may involve a large number of compounds (>100 000) and a large number of descriptors (several thousands).
2. Fingerprint descriptors are frequently used. In these cases, the descriptors are sparse and only 5% of them are nonzero. Also, strong correlations can exist between different descriptors.
3. There is a need to maintain many models (e.g., dozens) on many different targets.
4. These models need to be updated routinely (e.g., monthly).

Even in well-supported, high-performance in-house computing environments, computer time and memory may become limiting. In our environment, an ideal QSAR method should be able to build a predictive model from 300 000 molecules with 10 000 descriptors within 24 h elapsed time, without manual intervention. QSAR methods that are particularly computer-intensive or require the adjustment of many sensitive parameters to achieve good prediction for an individual QSAR data set are less attractive.

Because of these constraints, only a small number of the many machine learning algorithms that have been proposed are suitable for general QSAR applications in drug discovery. Currently, the most commonly used methods are variations on random forest (RF)[1] and support vector machine (SVM),[2] which are among the most predictive.[3,4] In particular, RF has been very popular since it was introduced as a QSAR method by Svetnik et al.[5] Due to its high prediction accuracy, ease of use, and robustness to adjustable parameters, RF has been something of a "gold standard" to which other QSAR methods are compared. This is also true for non-QSAR types of machine learning.[6]

In 2012, Merck sponsored a Kaggle competition (www.kaggle.com) to examine how well the state of art of machine

Received: December 17, 2014

DOI: 10.1021/ci500747n

Table 1. Data Sets for Prospective Prediction

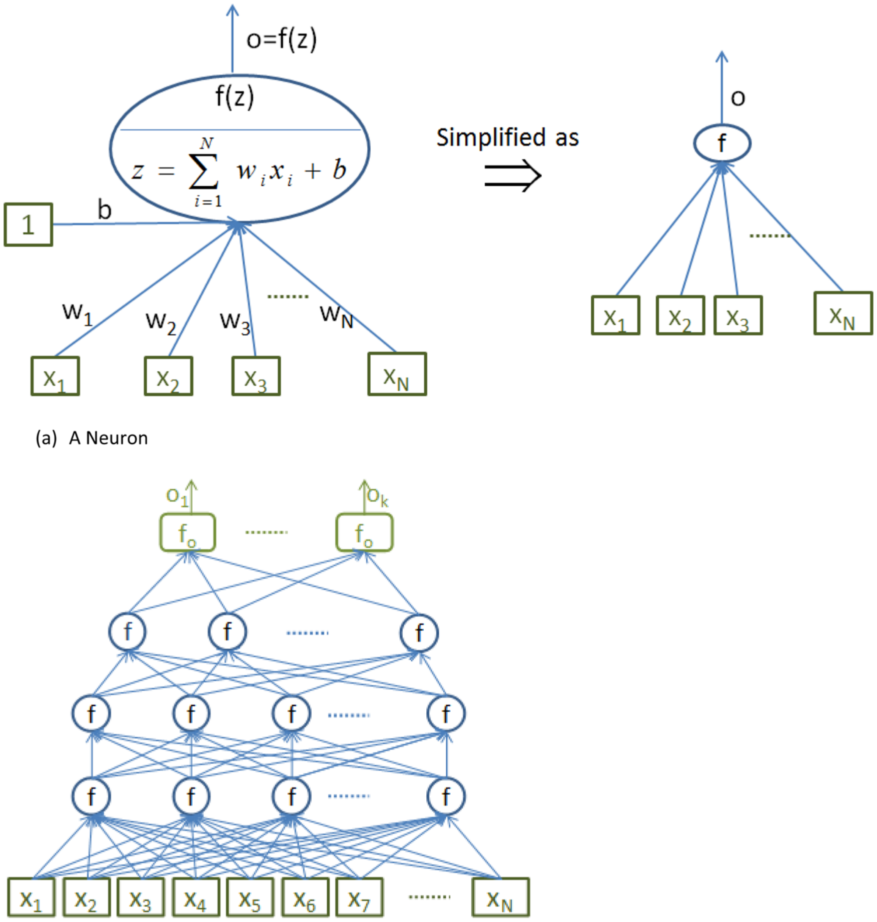
data set	type	description	number of molecules	number of unique AP, DP descriptors
		Kaggle Data Sets		
3A4	ADME	CYP P450 3A4 inhibition −log(IC50) M	50000	9491
CB1	target	binding to cannabinoid receptor 1 −log(IC50) M	11640	5877
DPP4	target	inhibition of dipeptidyl peptidase 4 −log(IC50) M	8327	5203
HIVINT	target	inhibition of HIV integrase in a cell based assay −log(IC50) M	2421	4306
HIVPROT	target	inhibition of HIV protease −log(IC50) M	4311	6274
LOGD	ADME	logD measured by HPLC method	50000	8921
METAB	ADME	percent remaining after 30 min microsomal incubation	2092	4595
NK1	target	inhibition of neurokinin1 (substance P) receptor binding −log(IC50) M	13482	5803
OX1	target	inhibition of orexin 1 receptor −log(K_i) M	7135	4730
OX2	target	inhibition of orexin 2 receptor −log(K_i) M	14875	5790
PGP	ADME	transport by *p*-glycoprotein log(BA/AB)	8603	5135
PPB	ADME	human plasma protein binding log(bound/unbound)	11622	5470
RAT_F	ADME	log(rat bioavailability) at 2 mg/kg	7821	5698
TDI	ADME	time dependent 3A4 inhibitions log(IC50 without NADPH/IC50 with NADPH)	5559	5945
THROMBIN	target	human thrombin inhibition −log(IC50) M	6924	5552
		Additional Data Sets		
2C8	ADME	CYP P450 2C8 inhibition −log(IC50) M	29958	8217
2C9	ADME	CYP P450 2C9 inhibition −log(IC50) M	189670	11730
2D6	ADME	CYP P450 2D6 inhibition −log(IC50) M	50000	9729
A-II	target	binding to Angiotensin-II receptor −log(IC50) M	2763	5242
BACE	target	inhibition of beta-secretase −log(IC50) M	17469	6200
CAV	ADME	inhibition of Cav1.2 ion channel	50000	8959
CLINT	ADME	clearance by human microsome log(clearance) μL/min·mg	23292	6782
ERK2	target	inhibition of ERK2 kinase −log(IC50) M	12843	6596
FACTORXIA	target	inhibition of factor Xla −log(IC50) M	9536	6136
FASSIF	ADME	solubility in simulated gut conditions log(solubility) mol/L	89531	9541
HERG	ADME	inhibition of hERG channel −log(IC50) M	50000	9388
HERG (full data set)	ADME	inhibition of hERG ion channel −log(IC50) M	318795	12508
NAV	ADME	inhibition of Nav1.5 ion channel −log(IC50) M	50000	8302
PAPP	ADME	apparent passive permeability in PK1 cells log(permeability) cm/s	30938	7713
PXR	ADME	induction of 3A4 by pregnane X receptor; percentage relative to rifampicin	50000	9282

learning methods can perform in QSAR problems. We selected 15 QSAR data sets of various sizes (2000−50 000 molecules) using a common descriptor type. Each data set was divided into a training set and test set. Kaggle contestants were given descriptors and activities for the training set and descriptors only for the test set. Contestants were allowed to generate models using any machine learning method or combinations thereof, and predict the activities of test set molecules. Contestants could submit as many separate sets of predictions as they wished within a certain time period. The winning entry (submitted by one of the authors, George Dahl) improved the mean R^2 averaged over the 15 data sets from 0.42 (for RF) to 0.49. While the improvement might not seem large, we have seldom seen any method in the past 10 years that could consistently outperform RF by such a margin, so we felt this was an interesting result.

The winning entry used an ensemble of different methods, including deep neural net (DNN), gradient boosting machine (GBM),[3] and Gaussian process (GP) regression.[7] Here we focus on DNN, since it is the major contributor to the high prediction accuracy of the winning entry, and we would like to investigate the usefulness of DNN by itself as a QSAR method.

DNNs were one of the increasingly popular methods in the machine learning community in the past 8 years and produced disruptively high performance in speech recognition,[8] computer vision,[9] and other artificial intelligence applications. One of the major differences between DNNs today and the classical artificial neural networks widely used for chemical applications in the 1990s is that DNNs have more than one intermediate (i.e., hidden) layer and more neurons in each layer and are thus both "deeper" and "wider."

The classical neural networks suffered from a number of practical difficulties. For example, they could handle only a limited number of input descriptors. Therefore, descriptor selection or extraction methods had to be applied to reduce the effective number of descriptors from thousands to tens or at most hundreds. Valuable predictive information was thus lost. Also, to avoid overfitting the training data and to reduce computation burden, the number of hidden layers was limited to one, and the number of neurons in that hidden layer had to be limited. Thanks to the advancements in theoretical methods, optimization algorithms, and computing hardware, most of the issues with classical neural networks have been resolved. Nowadays, neural networks with multiple hidden layers and thousands of neurons in each layer can be routinely applied to data sets with hundreds of thousands of compounds and thousands of descriptors without the need of data reduction. Also, overfitting can be controlled even when the nets have millions of weights.



(a) A Neuron

(b) A Deep Neural Net (DNN)

Figure 1. Architecture of deep neural nets.

On the other hand, DNNs, as with any neural network method, require the user to set a number of adjustable parameters. In this paper, we examine 15 diverse QSAR data sets and confirm that DNNs in most cases can make better predictions than RF. We also demonstrate that it is possible to have a single set of adjustable parameters that perform well for most data sets, and it is not necessary to optimize the parameters for each data set separately. This makes DNNs a practical method for QSAR in an industrial drug discovery environment. Previously, Dahl et al.[10] used DNNs for QSAR problems, but with a less realistic classification formulation of the QSAR problem, and on public data without a prospective time-split of training and test sets. Additionally, Dahl et al. optimized adjustable parameters separately on each assay and did not focus on the practicalities of industrial drug discovery tasks.

■ METHODS

Data Sets. Table 1 shows the data sets used in this study. These are in-house Merck data sets including on-target and ADME (absorption, distribution, metabolism, and excretion) activities. The 15 labeled "Kaggle Data Sets" are the same data sets we used for the Kaggle competition, which are a subset of the data sets in the work of Chen et al.[11] A separate group of 15 different data sets labeled "Additional Data Sets" are used to validate the conclusions acquired from the Kaggle data sets.

For this study, it is useful to use proprietary data sets for two reasons:

1. We wanted data sets that are realistically large and whose compound activity measurements have a realistic amount of experimental uncertainty and include a non-negligible amount of qualified data.
2. Time-split validation (see below), which we consider more realistic than any random cross-validation, requires dates of testing, and these are almost impossible to find in public domain data sets.

The Kaggle data sets are provided as Supporting Information. Due to the proprietary nature of the compounds, as in the Kaggle competition, the descriptor names are disguised so the compound structures cannot be reverse engineered from the descriptors. However, comparisons can be made between different QSAR methods.

A number of these data sets contain significant amounts of "qualified data". For example, one might know IC50 > 30 μM because 30 μM was the highest concentration tested. It is quite common for affinity data in the pharmaceutical industry to have this characteristic. Most off-the-shelf QSAR methods can handle only fixed numbers, so for the purposes of regression models, those activities were treated as fixed numbers, for example, IC50 = 30 μM or −log(IC50) = 4.5. Our experience is that keeping such data in the QSAR models is necessary;

DOI: 10.1021/ci500747n
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

otherwise, less active compounds are predicted to be more active than they really are.

In order to evaluate QSAR methods, each of these data sets was split into two nonoverlapping subsets: a training set and a test set. Although a usual way of making the split is by random selection, i.e. "split at random," in actual practice in a pharmaceutical environment, QSAR models are applied "prospectively". That is, predictions are made for compounds not yet tested in the appropriate assay, and these compounds may or may not have analogs in the training set. The best way of simulating this is to generate training and test sets by "time-split". For each data set, the first 75% of the molecules assayed for the particular activity form the training set, while the remaining 25% of the compounds assayed later form the test set. We have found that, for regressions, R^2 from time-split validation better estimates the R^2 for true prospective prediction than R^2 from "split at random" scheme.[12] Since training and test sets are not randomly selected from the same pool of compounds, the data distributions in these two subsets are frequently not the same, or even similar to, each other. This violates the underlying assumption of many machine learning methods and poses a challenge to them. Both the training and test data sets of the Kaggle data sets are provided as Supporting Information.

Descriptors. Each molecule is represented by a list of features, i.e. "descriptors" in QSAR nomenclature. Our previous experience in QSAR favors substructure descriptors (e.g., atom pairs (AP), MACCS keys, circular fingerprints, etc.) for general utility over descriptors that apply to the whole molecule (e.g., number of donors, LOGP, molecular weight, etc.). In this paper, we use a set of descriptors that is the union of AP, the original "atom pair" descriptor from Carhart et al.[13] and DP descriptors ("donor−acceptor pair"), also called "BP" in the work of Kearsley et al.[14] Both descriptors are of the following form:

$$\text{atom type } i - (\text{distance in bonds}) - \text{atom type } j$$

For AP, atom type includes the element, number of nonhydrogen neighbors, and number of pi electrons; it is very specific. For DP, atom type is one of seven (cation, anion, neutral donor, neutral acceptor, polar, hydrophobe, and other).

Random Forest. The main purpose of this paper is to compare DNN to RF. RF is an ensemble recursive partitioning method where each recursive partitioning "tree" is generated from a bootstrapped sample of compounds and a random subset of descriptors is used at each branching of each node. The trees are not pruned. RF can handle regression problems or classification problems. RF naturally handles correlation between descriptors, and does not need a separate descriptor selection procedure to obtain good performance. Importantly, while there are a handful of adjustable parameters (e.g., number of trees, fraction of descriptors used at each branching, node size, etc.), the quality of predictions is generally insensitive to changes in these parameters. Therefore, the same set of parameters can be effectively used in various problems.

The version of RF we are using is a modification of the original FORTRAN code from the work of Breiman.[1] It has been parallelized to run one tree per processor on a cluster. Such parallelization is necessary to run some of our larger data sets in a reasonable time. For all RF models, we generate 100 trees with $m/3$ descriptors used at each branch-point, where m is the number of unique descriptors in the training set. The tree nodes with 5 or fewer molecules are not split further. We apply these parameters to every data set.

Deep Neural Nets. A neural network is network composed of simulated "neurons". Figure 1a shows a neuron in its detailed form and simplified form. Each neuron has multiple inputs (the input arrows) and one output (the output arrow). Each input arrow is associated with a weight w_i. The neuron is also associated with a function, $f(z)$, called the activation function, and a default bias term b. Thus, when a vector of input descriptors $\mathbf{X} = [x_1 \cdots x_N]^T$ of a molecule goes through a neuron, the output of the neuron can be represented mathematically in eq 1:

$$O = f\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{1}$$

A row of neurons forms a layer of the neural network, and a DNN is built from several layers of neurons, which is illustrated in Figure 1b.

Normally, there are three types of layers in a DNN:

(1) the input layer (i.e., the bottom layer), where the descriptors of a molecule are entered

(2) the output layer (i.e., the top layer) where predictions are generated

(3) the hidden (middle) layers; the word "deep" in deep neural nets implies more than one hidden layer.

There are two popular choices of activation functions in the hidden layers: (1) the sigmoid function and (2) the rectified linear unit (ReLU) function. Both functions and their derivatives are shown in Figure 2.

The output layer can have one or more neurons, and each output neuron generates prediction for a separate end point (e.g., assay result). That is, a DNN can naturally model multiple end points at the same time. The activation function of the neurons in the output layer is usually a linear function, which is shown in Figure 3.

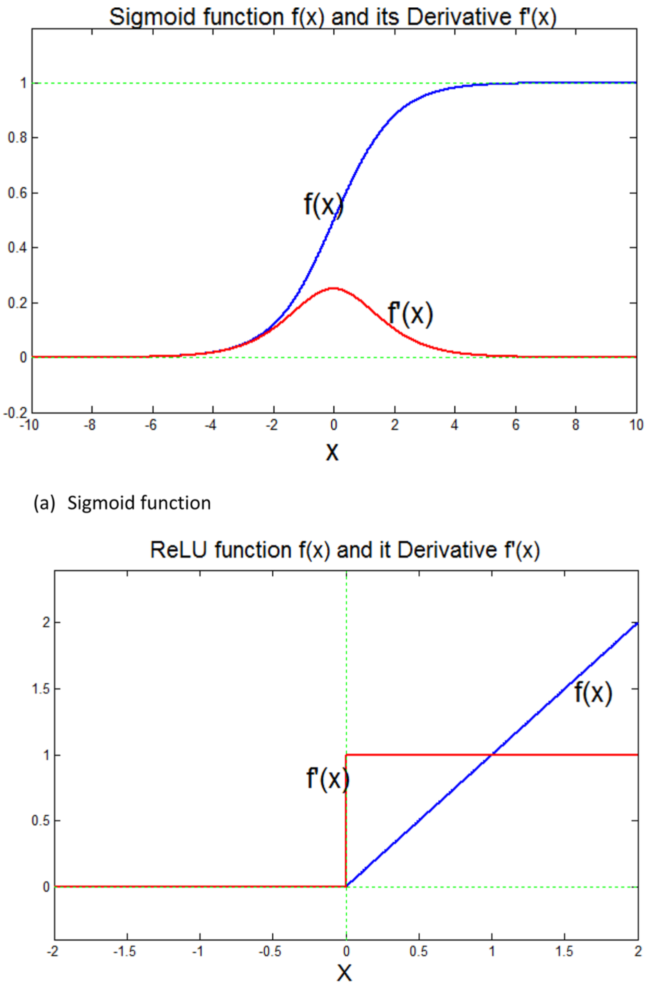
The layout of a DNN, including the number of layers and the number of neurons in each layer, needs to be prespecified, along with the choice of the activation function in each neuron. Therefore, to train a DNN is to maximize an objective function by optimizing the weights and bias of each neuron

$$\varnothing = (\{w_{i,j}\}, \{b_j\}, i = 1, ..., N_j, j = 1, ..., L + 1)$$

where N_j is the number of neurons in the jth layer and L is the number of hidden layers. The extra one layer of j is for the output layer.

The training procedure is the well-known backward propagation (BP) algorithm implemented using mini-batched stochastic gradient descent (SGD) with momentum.[15] The individual values for $\varnothing$ are first assigned random values. The molecules in the training set are randomly shuffled and then evenly divided into small groups of molecules called "mini-batches". Each mini-batch is used to update the values of $\varnothing$ once using the BP algorithm. When all the mini-batches from the training set are used, it is said that the training procedure finishes one "epoch". The training procedure of a DNN usually requires many epochs. That is, the training set is reused many times during the training. The number of epochs is an adjustable parameter.

The number of elements in $\varnothing$ for a QSAR task can be very large. For example, the training data set can have 8000 descriptors, and the DNN can have three hidden layers, each



(a) Sigmoid function

(b) Rectified Linear Unit (ReLU)

Figure 2. Activation functions used in the hidden layers.

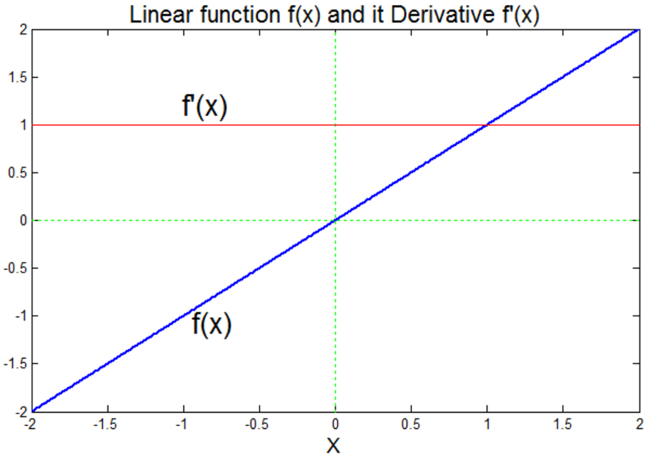


Figure 3. Activation function in the output layer.

layer having 2000 neurons. Under this condition, the DNN will have over 24 million tunable values in ∅. Therefore, the DNN trained using the BP algorithm is prone to overfitting. Advancements in avoiding overfitting made over the past eight years played a critical role in the revival of neural networks. Among the several methods to avoid overfitting, the two most popular ones are (1) the generative unsupervised pretraining proposed by Hinton et al.[16] and (2) the procedure of "drop-out" proposed by Srivastava et al.[17] The first approach can mitigate overfitting because it acts as a data-dependent regularizer of ∅, i.e. constraining the values of ∅. Instead of using random values to initialize ∅ in a DNN, it generates values of ∅ by using an unsupervised learning procedure conducted on the input descriptors only, without considering the activities of the compounds they represent. The subsequent supervised BP training algorithm just fine-tunes ∅ starting from the values produced from the unsupervised learning. The second approach introduces instability to the architecture of the DNN by randomly "dropping" some neurons in each mini-batch of training. It has been shown that the drop-out procedure is equivalent to adding a regularization process to the conventional neural network to minimize overfitting.[18] These two approaches can be used separately or jointly in a DNN training process.

As with a conventional neural network, a DNN can have multiple neurons in the output layer with each output neuron correspond to a different QSAR model. We will call this type of DNN *joint DNNs*, which was called *multitask DNNs* in the work of Dahl et al.[10] Joint DNNs can simultaneously model multiple QSAR tasks, and all QSAR models embedded in a joint DNN share the same weights and bias in the hidden layers but have their own unique weights and bias in the output layer. Generally speaking, the hidden layers function as a complex feature/descriptor optimization process, while the output layer acts as a classifier. That is, all involved QSAR activities share the same feature-extraction process but have their own prediction based on the weights and bias associated with the corresponding output neuron. As we will see, joint DNNs are especially useful for those QSAR tasks with a smaller training set. The training set of a joint DNN is formed by merging training sets of all involved QSAR tasks. DNNs generally benefit from a large training set and can potentially borrow learned molecule structure knowledge across QSAR tasks by extracting better QSAR features via the shared hidden layers. Therefore, a DNN user can choose either to train a DNN from a single training set or to train a joint DNN from multiple training sets simultaneously. Many models presented in this paper were trained as joint DNNs with all 15 data sets. Since jointly training multiple QSAR data sets in a single model is not a standard approach for most non-neural-net QSAR methods, we need to show the difference in performance between joint DNNs and individual DNNs trained with a single data set.

In order to improve the numeric stability, the input data in a QSAR data set is sometimes preprocessed. For example, the activities in the training set are usually normalized to zero mean and unit variance. Also, the descriptors, x, can also undergo some transformations, such as *logarithmic transformation (i.e.,* $y = \log(x + 1)$*)* or *binary transformation (i.e.,* $y = 1$ *if* $x > 0$*, otherwise* $y = 0$*)*. Both transformations were specially designed for substructure descriptors, which are used in this study, where the possible values are integers 0, 1, 2, 3, ... For other descriptor types, one would have to adjust the mathematic form of both transformations to achieve the same goal.

For a typical QSAR task, training a DNN is quite computationally intensive due to the large number of molecules in the training set, and the large number of neurons needed for the task. Fortunately, the computation involved in training a DNN is primarily large matrix operations. An increasingly popular computing technique, called GPU (graphical processing unit) computing, can be very efficient for such large matrix

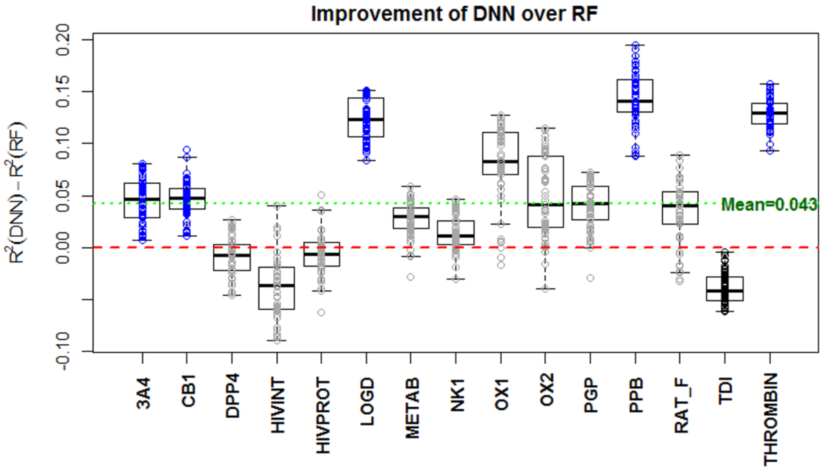


Figure 4. Overall DNN vs RF using arbitrarily selected parameter values. Each column represents a QSAR data set, and each circle represents the improvement, measured in R^2, of a DNN over RF. The horizontal dashed red line indicates 0, where DNNs have the same performance of RF. A positive value means that the corresponding DNN outperforms RF. The horizontal dotted green line indicates the overall improvement of DNNs over RF measured in mean R^2. The data sets, in which DNNs dominates RF for all arbitrarily parameter settings, are colored blue; the data set, in which RF dominates DNNs for all parameter settings, is colored black; the other data sets are colored gray.

operations and can dramatically reduce the time needed to train a DNN.

To summarize, the adjustable algorithmic parameters (also called metaparameters or hyperparameters in the machine learning literature) of a DNN are as follows:

- Related to the data
  – Options for descriptor transformation: (1) no transformation, (2) logarithmic transformation, i.e. $y = \log(x + 1)$, or (3) binary transformation, i.e. $y = 1$ if $x > 0$, otherwise $y = 0$.
- Related to the network architecture
  – Number of hidden layers
  – Number of neurons in each hidden layer
  – Choices of activation functions of the hidden layers: (1) sigmoid function and (2) rectified linear unit (ReLU)
- Related to the DNN training strategy
  – Training a DNN from a single training set or a joint DNN from multiple training sets
  – Percentage of neurons to drop-out in each layer
  – Using the unsupervised pretraining to initialize the parameter or not
- Related to the mini-batched stochastic gradient descent procedure in the BP algorithm
  – Number of molecules in each mini-batch, i.e. the mini-batch size
  – Number of epochs, i.e. how many times the training set is used
  – Parameters to control the gradient descent optimization procedure, including (1) *learning rate*, (2) *momentum strength*, and (3) *weight cost strength*.[10]

One of the goals of this paper is to acquire insights into how adjusting these parameters can alter the predictive capability of DNNs for QSAR tasks. Also, we would like to find out whether it is possible for DNNs to produce consistently good results for a diverse set of QSAR tasks using one set of values for the adjustable parameters, which is subsequently called an algorithmic parameter setting.

Metrics. In this paper, the metric to evaluate prediction performance is R^2, which is the squared Pearson correlation coefficient between predicted and observed activities in the test set. The same metric was used in the Kaggle competition. R^2 measures the degree of concordance between the predictions and corresponding observations. This value is especially relevant when the whole range of activities is included in the test set. R^2 is an attractive measurement for model comparison across many data sets, because it is unitless, and range from 0 to 1 for all data sets. We found in our examples that other popular metrics, such as normalized root mean squared error (RMSE), i.e. RMSE divided by the standard deviation of observed activity, is inversely related to R^2, so the conclusions would not change if we used the other metrics.

Workflow. One key question that this paper tries to answer is whether we can find a set of values for the algorithmic parameters of DNNs so that DNNs can consistently make more accurate predictions than RF does for a diverse set of QSAR data sets.

Due to the large number of adjustable parameters, it is prohibitively time-consuming to evaluate all combinations of possible values. The approach we took was to carefully select a reasonable number of parameter settings by adjusting the values of one or two parameters at a time, and then calculate the R^2s of DNNs trained with the selected parameter settings. For each data set, we ultimately trained and evaluated at least 71 DNNs with different parameter settings. These results provided us with insights into sensitivities of many adjustable parameters, allowed us to focus on a smaller number of parameters, and to finally generate a set of recommended values for all algorithmic parameters, which can lead to consistently good DNNs across the 15 diverse QSAR data sets.

The DNN algorithms were implemented in Python and were derived from the code that George Dahl's team developed to win the Merck Kaggle competition. The python modules, gnumpy[19] and cudamat,[20] are used to implement GPU computing. The hardware platform used in this study is a Windows 7 workstation, equipped with dual 6-core Xeon CPUs, 16 GB RAM, and two NVIDIA Tesla C2070 GPU cards.

RESULTS

DNNs Trained with Arbitrarily Selected Parameters. First, we want to find out how well DNNs can perform relative to RF. Therefore, over 50 DNNs were trained using different parameter settings. These parameter settings were arbitrarily selected, but they attempted to cover a sufficient range of values of each adjustable parameter. (A full list of the parameter settings is available as Supporting Information.) More specifically, our choices for each parameter are listed as follows:

- Each of three options of data preprocess (i.e., (1) no transformation, (2) logarithmic transformation, and (3) binary transformation) was selected.
- The number of hidden layers ranged from 1 to 4.
- The number of neurons in each hidden layer ranged from 100 to 4500.
- Each of the two activation functions (i.e., (1) sigmoid and (2) ReLU) was selected.
- DNNs were trained both (1) separately from an individual QSAR data set and (2) jointly from a data set combining all 15 data sets.
- The input layer had either no dropouts or 10% dropouts. The hidden layers had 25% dropouts.
- The network parameters were initialized as random values, and no unsupervised pretraining was used.
- The size of mini-batch was chosen as either 128 or 300.
- The number of epochs ranged from 25 to 350.
- The parameters for the optimization procedure were fixed as their default values. That is, learning rate is 0.05, momentum strength is 0.9, and weight cost strength is 0.0001.

Figure 4 shows the difference in R^2 between DNNs and RF for each data set. Each column represents a QSAR data set, and each circle represents the improvement, measured in R^2, of a DNN over RF. A positive value means that the corresponding DNN outperforms RF. A boxplot with whiskers is also shown for each data set. Figure 4 demonstrates that, with rather arbitrarily selected parameter settings, DNNs on average outperform RF in 11 out of the 15 Kaggle data sets. Moreover, in five data sets, DNNs do better than RF for all parameter settings. Only in one data set (TDI), the RF is better than all the tested DNNs. The mean R^2 averaged over all DNNs and all 15 data sets is 0.043 higher than that of RF, or a 10% improvement.

Additional information can be found in Table 2. It shows that, even when the worst DNN parameter setting was used for each QSAR task, the average R^2 would be degraded only from 0.423 to 0.412, merely a 2.6% reduction. These results suggest that DNNs can generally outperform RF.

Table 2. Comparing Test R^2s of Different Models

			DNNs trained with arbitrarily selected, but reasonable, parameters		
data set	random forest	DNN with recommended parameter values	median R^2	worst case	best case
3A4	0.469	**0.521**	*0.515*	0.476	0.550
CB1	0.283	**0.345**	*0.331*	0.294	0.377
DPP4	**0.232**	0.194	0.224	0.186	0.259
HIVINT	0.353	**0.403**	0.316	0.263	0.393
HIVPROT	0.530	**0.543**	0.524	0.467	0.581
LOGD	0.685	**0.822**	*0.808*	0.769	0.836
METAB	0.631	**0.687**	*0.661*	0.603	0.690
NK1	0.403	**0.448**	*0.414*	0.373	0.449
OX1	0.501	**0.657**	*0.584*	0.484	0.628
OX2	0.580	**0.707**	*0.621*	0.540	0.695
PGP	0.550	**0.616**	*0.592*	0.521	0.622
PPB	0.420	**0.598**	*0.561*	0.508	0.615
RAT_F	0.098	**0.168**	*0.138*	0.065	0.187
TDI	**0.381**	0.350	0.339	0.319	0.377
THROMBIN	0.222	**0.375**	*0.351*	0.315	0.380
mean	***0.423***	***0.496***	***0.465***	***0.412***	***0.509***

Understanding the Impacts of Some DNN Parameters. In order to gain insights into the impacts of some adjustable parameters, we studied how R^2s of DNNs changed when a DNN parameter of interest was adjusted while the other parameters were kept unchanged. We realize that that the prediction performance of DNNs likely depends on the interaction between different algorithmic parameters. Therefore, our conclusions should be interpreted with caution. However, investigating effects of parameters one at a time is a frequently used efficient way of narrowing down the list of important parameters.

First, we would like to understand the impact of the network architecture (i.e., number of layers, and number of neurons in each layer). In order to limit the number of different parameter combinations, we assume that each hidden layer of the DNNs has the same number of neurons, although DNN methods allow each layer to have different number of neurons. Thus, the network architecture has two parameters: (1) number of hidden layers and (2) number of neurons in each layer. Thirty two (32) DNNs were trained for each data set by varying both parameters. Meanwhile, the other key adjustable parameters were kept unchanged. That is, no data preprocessing was done, the activation function used was ReLU, and all DNNs were trained jointly from a data set combining all 15 data sets. The difference in R^2 between DNN and RF averaged over 15 data sets is presented in Figure 5.

The most obvious observation from Figure 5 is that, when the number of hidden layer is two, having a small number of neurons in the layers degrades the predictive capability of DNNs. This makes sense. When the data are fed into the input layer, the next layer of neurons can see only the information passed up from the previous layer. A layer with a smaller number of neurons has less capacity to represent the data. If the number of neurons in a layer is too small to accurately capture important factors of variation in the data, then the next layer will also not be able to capture this information and adding more layers will just make the optimization harder.

Also, it can be seen that, given any number of hidden layers, once the number of neurons per layer is sufficiently large, increasing the number of neurons further has only a marginal benefit. For example, once the number of neurons per layer reaches 128 for a two-hidden-layer DNN, its performance begins to approach a plateau. Also, Figure 5 suggests that a DNN tends to have a higher plateau in prediction performance when further increasing the number of neurons per layer. Three- and four-hidden layer networks behave much like a two-hidden layer networks, i.e. these deeper DNNs requires a much larger number of neurons per layer to research a plateau, which is higher than that of a two-hidden-layer DNN. Results of a four-hidden-layer DNN are presented subsequently.

A surprising observation from Figure 5 is that the neural network achieved the same average predictive capability as RF when the network has only one hidden layer with 12 neurons. This size of neural network is indeed comparable with that of the classical neural network used in QSAR. This clearly reveals some key reasons behind the performance improvement gained

DOI: 10.1021/ci500747n

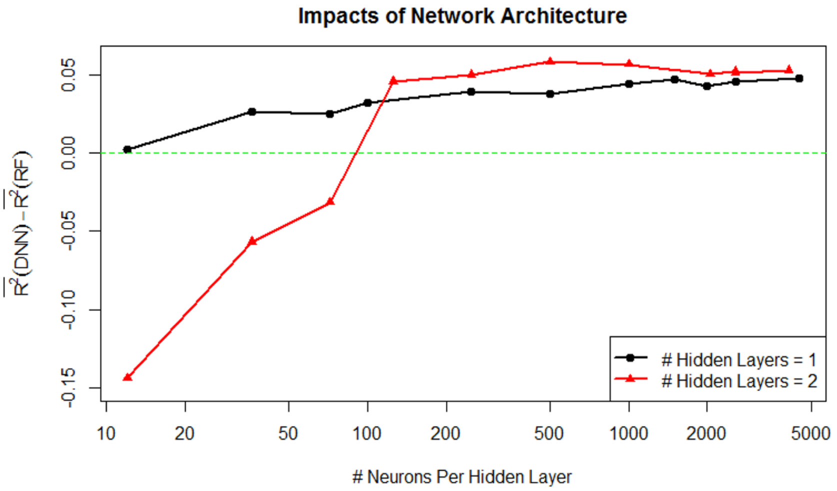


Figure 5. Impacts of Network Architecture. Each marker in the plot represents a choice of DNN network architecture. The markers share the same number of hidden layers are connected with a line. The measurement (i.e., *y*-axis) is the difference of the mean R^2 between DNNs and RF. The mean R^2 of DNNs is obtained by averaging over all DNNs sharing the same network architecture and over 15 data sets. The horizontal dotted green line indicates 0, where the mean R^2 of DNNs is the same as that of RF.

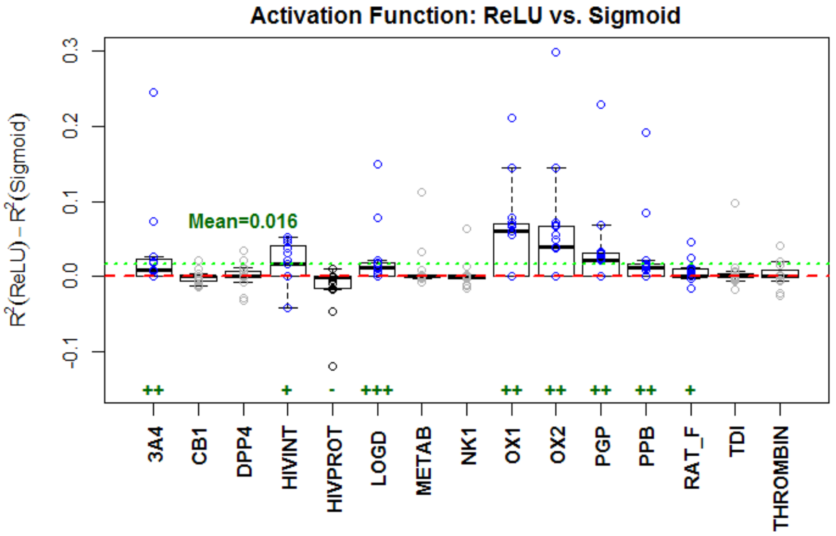


Figure 6. Choice of activation functions. Each column represents a QSAR data set, and each circle represents the difference, measured in R^2, of a pair of DNNs trained with ReLU and Sigmoid, respectively. The horizontal dashed red line indicates 0. A positive value indicates the case where ReLU outperforms Sigmoid. The horizontal dotted green line indicates the overall difference between ReLU and Sigmoid, measured in mean R^2. The data sets where ReLU is significantly better than Sigmoid are marked with "+"s at the bottom of the plot and colored blue. "+" indicates p-value < 0.05, and "++" indicates p-value < 0.01, while "+++" indicates p-value < 0.001. In contract, the data set where Sigmoid is significantly better than ReLU is marked with "−"s at the bottom of the plot and is colored black. "−" indicates p-value < 0.05. The remaining data sets are not marked and are colored gray.

by our way of using DNN: (1) a large input layer that accepts the thousands of descriptors without the need for feature reduction and (2) the dropout method that successfully avoids overfitting during training.

Next, we would like to decide which activation function, Sigmoid or ReLU, is a better choice for QSAR. For each data set, at least 15 pairs of DNNs were trained. Each pair of DNNs shared the same adjustable parameter settings, except that one DNN used ReLU as the activation function, while the other used Sigmoid function. The difference in R^2 between the pair of DNNs is presented in Figure 6 as a circle. Any circle above the horizontal line of 0 indicates that the DNN using ReLU has better predictive capability. For each data set, an one-sample Wilcoxon test is conducted to find out if the difference in R^2 is significantly larger or smaller than 0. The data sets where ReLU is significantly better than Sigmoid are colored in blue, and marked at the bottom with "+"s. In contrast, the data set where Sigmoid is significantly better than ReLU is colored in black, and marked at the bottom with "−"s. Figure 6 shows that, in 8 out of 15 data sets, ReLU is statistically significantly better than Sigmoid, while only in 1 data set, Sigmoid is significantly better. Overall ReLU improves the average R^2 over Sigmoid by 0.016. Thus, our results suggest that ReLU is more preferable for QSAR tasks than Sigmoid.

One issue with using ReLU is that it sometimes may cause numerical instability because it has no upper bound (see Figure 2b). A solution to this issue for QSAR tasks is to preprocess the input data with either a logarithmic transformation or a binary transformation, as previously presented in the Methods section.

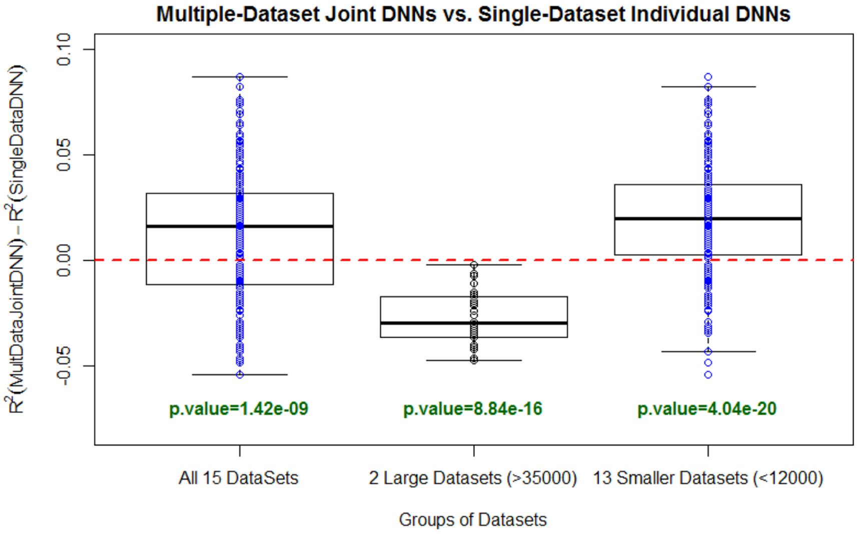


Figure 7. Difference between joint DNNs trained with multiple data sets and the individual DNNs trained with single data sets. Each column represents a scenario for comparing joint DNNs with single-task DNNs. Each circle represents the difference, measured in R^2, of a pair of DNNs trained from multiple data sets and a single data set, respectively. The horizontal dashed red line indicates 0. A positive value indicates the case where a joint DNN outperforms an individual DNN. The *p*-value of a two-side paired-sample *t* test conducted for each scenario is also provided at the bottom of each column.

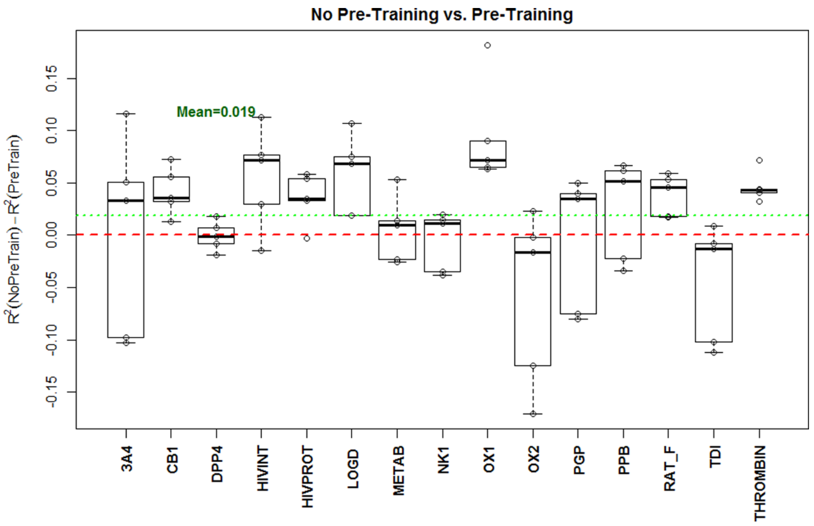


Figure 8. Impacts of unsupervised pretraining. Each column represents a QSAR data set, and each circle represents the difference, measured in R^2, of a pair of DNNs trained without and with pretraining, respectively. The horizontal dashed red line indicates 0. A positive value indicates that a DNN without a pretraining outperforms the corresponding DNN with a pretraining. The horizontal dotted green line indicates the overall difference between DNNs without and with pretraining, measured in mean R^2.

Joint DNNs Trained from Multiple Data Sets vs Individual DNNs Trained from a Single Data Set. Next, we would like to decide whether training a joint DNN using multiple training sets is preferable over training an individual DNN using a single data set. Given a DNN parameter setting, each of the 15 QSAR data sets was first used to train 15 individual DNNs. Then, using the same DNN parameter settings, a joint DNN was trained from the data combined from the 15 data sets. This joint DNN was capable of producing predictions for each of the 15 QSAR tasks. Thus, the prediction obtained from the corresponding individual DNN and that from the joint DNN form a comparison pair. For each data set, 17 comparison pairs were produced using 17 different DNN parameter settings. One analysis of the result is shown in Figure 7. The difference in R^2 between the comparison pair is presented in Figure 7 as a circle. Any circle above the horizontal line of 0 indicates that modeling the QSAR task as a joint DNN is preferred. When averaged over all data sets, there seems to be a difference favoring the joint DNN. However, the size of the training sets plays a critical role on whether a joint DNN is beneficial. For the two very largest data sets (i.e., 3A4 and LOGD), the individual DNNs seem better, as shown in Figure 7. An in-depth explanation of this detectable boost for the joint DNN warrants a future investigation, because out of the 129 295 unique molecules in all the data sets, 85% occur only in a single data set, 97% occur in two or fewer data sets, and >99% occur in three or fewer data sets. Also most of the overlap of molecules is accounted for by 3A4 and LOGD, and for these data sets, the joint DNN is worse. These facts do not support

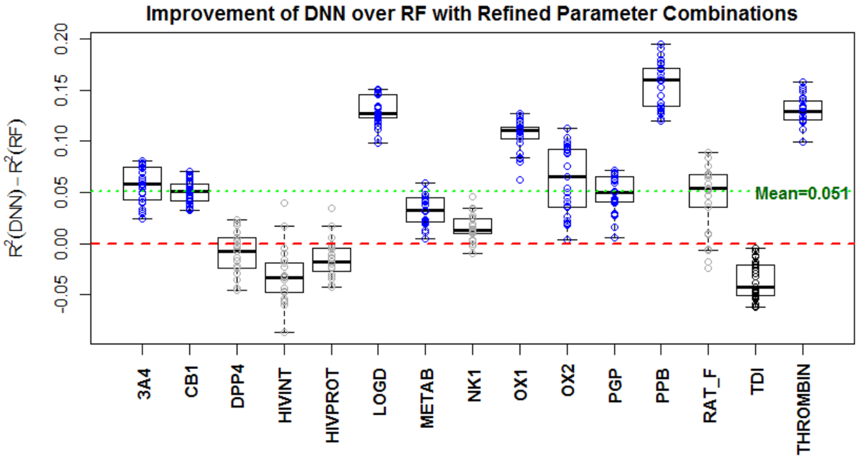


Figure 9. DNN vs RF with refined parameter settings. Each column represents a QSAR data set, and each circle represents the improvement, measured in R^2, of a DNN over RF. The horizontal dashed red line indicates 0. A positive value means that the corresponding DNN outperforms RF. The horizontal dotted green line indicates the overall improvement of DNNs over RF measured in mean R^2. The data sets, in which DNNs dominates RF for all arbitrarily parameter settings, are colored in blue; the data set, in which RF dominates DNNs for all parameter settings, is colored black; the other data sets are colored gray.

the simple explanation that benefit of the joint DNN is because it makes use of molecules shared across different data sets.

The Impact of DNN Pretraining. Finally, we would like to find out whether unsupervised pretraining a DNN can help improve its predictive capability. The invention of unsupervised pretraining[16] played an important role in the revival of neural network and the wide acceptance of DNNs. On problems where the distribution of inputs, $P(x)$, is informative about the conditional distribution, $P(y|x)$, of outputs given inputs, unsupervised pretraining could dramatically improve the predictive capability of DNNs.

The first issue that we ran into when we tried to pretrain a DNN is that our software for unsupervised pretraining only accepts the sigmoid function as the activation function. Since we have shown that ReLU is more preferable than sigmoid for QSAR tasks, this is a disadvantage of using pretraining in our setup. Please note that Nair and Hinton[21] describe a method for performing unsupervised pretraining with ReLU even though ReLUs are not in the exponential family that makes the algorithm used in pretraining (i.e., contrastive divergence algorithm) valid. Unfortunately, that method was not implemented in our DNN software when we conducted this study.

For each data set, we trained five different DNN configurations with and without pretraining. Each pair of DNNs with and without pretraining is compared, and their difference in R^2 is presented as a data point (i.e., a circle) in Figure 8. Although the results are fairly variable among data sets, it is still evident that DNNs without pretraining on average are better than DNNs with pretraining. Figure 8 shows that, averaging over all DNN pairs and over 15 data sets, pretraining operation degrades R^2 by 0.019. Since pretraining on average degrades the predictive capability of a DNN on our data, and existing software does not support pretraining with our preferred activation function, ReLU, the unsupervised pretraining seems an undesirable procedure for QSAR tasks.

DNNs Trained with Refined Parameter Settings. After studying the results of DNNs with different parameter settings, more insights were gained regarding the impacts of some adjustable parameters on DNNs' predictive capability.

- Binary transformation generally hurts the performance, while the logarithmic transformation is helpful for achieving a better numeric stability.
- The number of hidden layers should be at least 2.
- The number of neurons in each hidden layer should be at least 250.
- The activation function of ReLU is generally better than Sigmoid.

If we refine our selections of adjustable parameters using these gained insights, an improved group of results can be obtained, which is shown in Figure 9. Comparing these results with those in Figure 4, indicates that there are now 9 out of 15 data sets, where DNNs outperforms RF even with the "worst" parameter setting, improving from 5 data sets in Figure 4. Also, the R^2 averaged over all DNNs and all 15 data sets is 0.051 higher than that of RF.

Recommended DNN Parameter Setting. Until now, the number of neurons and the percentage of dropouts in each hidden layer were always set the same for all layers. After studying the DNNs trained using different sizes and dropouts at different hidden layers, more insights regarding these parameters were gained. The sizes of initial stages of hidden layers should be larger than those of at the latter stage of hidden layers to ensure that the initial stages of the network have enough capacity to explore the data, and do not become an information bottleneck. Also, no dropout at the input layer is needed, and fewer dropouts at the latter stages of hidden layers are helpful. Two-, three-, and four-hidden-layer DNNs were studied, the best performance was achieved by four-hidden-layer DNNs. DNNs with more than four hidden layers were not investigated. That was because our results suggested that the negligible incremental performance gain could not justify the dramatic increase in computation required by DNNs with much wider and deeper hidden layers. It should be noted that the increase in performance of four-layer over two-layer DNNs is larger in multi-task problems than in single-task problems.

Combining the insights we gained regarding the adjustable parameters of DNNs, a parameter setting of DNNs is recommended, which is listed as follows:

- Data preprocessing: logarithmic transformation (i.e., $y = \log(x + 1)$).
- The DNN should have four hidden layers. The numbers of neurons in these four hidden layers are recommended to be 4000, 2000, 1000, and 1000, respectively.
- The dropout rates of the DNN are recommended to be 0 in the input layer, 25% in the first 3 hidden layer, and 10% in the last hidden layer.
- The activation function of ReLU should be used.
- No unsupervised pretraining is needed. The network parameters should be initialized as random values.
- The number of epochs should be set to as large as the computation capability is allowed, since dropout can prevent overfitting.
- The parameters for the optimization procedure were fixed as their default values. That is, learning rate is 0.05, momentum strength is 0.9, and weight cost strength is 0.0001.

As for choosing between joint DNNs trained from multiple data sets and individual DNNs trained from single data set separately, we do not have enough evidence to have a clear strategy. On one hand, joint DNNs on average outperform individual DNNs across all 15 Kaggle data sets. On the other hand, individual DNNs outperform joint DNNs on two larger data sets. We do not have enough data to recommend how many and which QSAR data sets should be merged to train a better joint DNN. Therefore, either joint DNNs or individual DNNs are acceptable at this moment.

Table 2 lists the Test R^2s of the DNNs jointly trained from 15 Kaggle data sets under this recommended set of parameters, along with those of RF, and those of DNNs in Figure 4, where the DNNs were trained using arbitrarily selected parameter settings. Please note that the recommended parameter setting is not one of the parameter settings that were used to train DNNs in Figure 4. Table 2 shows that the DNN with recommended parameters outperforms RF in 13 out of the 15 data sets.

Testing the Recommended Parameter Settings on New Data Sets. So far, all the results, along with our recommended DNN parameter setting, were obtained using the 15 Kaggle data sets. One concern is that we have somehow tuned the adjustable parameters to favor those specific data sets. Thus, is it important to show that the recommended parameters will also apply to other QSAR data sets that have not been part of the calibration. Thus, 15 additional QSAR data sets were arbitrarily selected from in-house data. Each additional data set was time-split into training and test sets in the same way as the Kaggle data sets. Individual DNNs were trained from the training set using the recommended parameters, and the test R^2s of the DNN and RF were calculated from the test sets. The results for the additional data sets are listed in Table 3. Table 3 shows that the DNN with recommended parameters outperforms RF in 13 out of the 15 additional data sets. The mean R^2 of DNNs is 0.411, while that of RFs is 0.361, which is an improvement of 13.9%. This is, the improvement of DNN over RF on the 15 additional data sets is similar to what we observed for the Kaggle data sets.

Table 3. Comparing RF with DNN Trained Using Recommended Parameter Settings on 15 Additional Datasets

data set	random forest (R^2)	individual DNN (R^2)
2C8	0.158	**0.255**
2C9BIG	0.279	**0.363**
2D6	0.130	**0.195**
A-II	0.805	**0.812**
BACE	0.629	**0.644**
CAV	0.399	**0.463**
CLINT	0.393	**0.554**
ERK2	**0.257**	0.198
FACTORXIA	0.241	**0.244**
FASSIF	**0.294**	0.271
HERG	0.305	**0.352**
HERGfull	0.294	**0.367**
NAV	0.277	**0.347**
PAPP	0.621	**0.678**
PXR	0.333	**0.416**
mean	0.361	***0.411***

DISCUSSION

In this paper, we demonstrate that DNN can be used as a practical QSAR method, and can easily outperform RF in most cases. While the magnitude of the change in R^2 relative to RF may appear to be small in some datasets, in our experience it is quite unusual to see a method that is on average better than RF. In order to address the difficulty in using DNN due to its large number of adjustable algorithmic parameters, we provide insights into the effects of some key parameters on DNN's predictive capability. Finally, we recommend a set of values for all DNN algorithmic parameters, which are appropriate for large QSAR data sets in an industrial drug discovery environment. It should be noted, however, that although the recommended parameters result in good predictions for most datasets in both joint and single-task modes, this does not necessarily imply that the recommended parameters are the most computationally efficient for any specific dataset. For example, we found that most single-task problems could be run with two hidden layers with fewer neurons (1000 and 500) and fewer epochs (75). This resulted in a very small decrease in predictive R^2 relative to that from the recommended parameters, but with a several-fold saving in computational time.

Both RF and DNN can be efficiently speeded up using high-performance computing technologies, but in a different way due to the inherent difference in their algorithms. RF can be accelerated using coarse parallelization on a cluster by giving one tree per node. In contrast, DNN can efficiently make use of the parallel computation capability of a modern GPU. With the dramatic advance in GPU hardware and increasing availability of GPU computing resources, DNN can become comparable, if not more advantageous, to RF in various aspects, including easy implementation, computation time, and hardware cost. For example, the hardware (i.e., a computer with GPU capability) used in this study costs about USD $4,000, and the supporting software (i.e., the python modules of gnumpy[19] and cudamat[20]) is free. This setting is more economical than a conventional computer cluster. With the help of the supporting python modules, programming GPU computing in python is almost exactly the same as conventional python programing.

Figure 7 shows that training a joint DNN from all 15 data sets simultaneously on average can slightly improve the predictive capability of some QSAR tasks with smaller training sets. These 15 data sets are quite different from each other in several aspects, such as the type of molecules involved and the purpose of the QSAR tasks. Generally with other QSAR methods, one processes only one data set at a time, and it is not intuitively obvious for chemists to understand how a QSAR model embedded in a joint DNN can borrow information from other unrelated QSAR data sets. Therefore, further investigation is needed to gain better understanding regarding how a joint DNN works. Fortunately, it is clear that even when data sets are treated individually, DNN can still achieve better results than RF.

Our results show that the unsupervised pretraining, which played a critical role in the success of DNNs in many machine learning applications, generally degrades a DNN's predictive capability in QSAR tasks. At this moment, we do not have a complete account of why pretraining hurts performance, although it likely has to do with the properties of substructure descriptors. Also, the limitation of our software, which prevents us from using the preferred activation function of ReLU, kept us from doing a better test of pretraining. We plan to revisit this issue by applying DNNs to data sets using different types of molecule descriptors.

Another future effort is to develop an effective and efficient strategy for refining the adjustable parameters of DNNs for each particular QSAR task. Although the performance of DNNs is generally better than RF using the standard DNN parameter settings, their predictive capability is variable under different parameter settings. Table 2 shows that, if the "best" parameter setting among those arbitrarily selected settings is selected for each data set, nearly 10% of improvement can be achieved over the median result. Therefore, in order to maximize the performance of a DNN, it may still be desirable to automatically fine-tune the DNN parameters for each data set. For many machine learning tasks, this can be done by using different variants of the cross-validation method. However, as previously mentioned, the QSAR tasks in an industrial drug R&D environment are better mimicked by time-split training and test sets. Our results (not shown) suggested that cross-validation failed to be effective for fine-tuning the algorithmic parameters for some data sets in this scenario. Automatic methods for tuning DNN parameters used in the machine learning community depend on validation performance being an accurate predictor of test performance. Therefore, new approaches that can better indicate a DNN's predictive capability in a time-split test set need to be developed before we can maximize the benefit of DNNs.

■ ASSOCIATED CONTENT

Supporting Information

Table of adjustable parameter settings used to evaluate DNNs in the Results section. Training and test data for Kaggle data sets (with disguised molecule names and descriptors). This data set is suitable for comparing the relative goodness of different QSAR methods given the AP,DP descriptors. This material is available free of charge via the Internet at http://pubs.acs.org

■ AUTHOR INFORMATION

Corresponding Author

*junshui_ma@merck.com.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

The authors thank Joseph F. Heyse and Keith A. Soper for their support on the initiative of this study. The authors thank Johannes H. Voigt and Amit Kulkarni for their supervision over the Kaggle competition at Merck. Joseph Shpungin parallelized the random forest code from Breiman.

■ REFERENCES

(1) Breiman, L. Random forests. *Machine Learning* **2001**, *45*, 5−32.

(2) Cortes, C.; Vapnik, V. N. Support-vector networks. *Machine Learning* **1995**, *20*, 273−297.

(3) Svetnik, V.; Wang, T.; Tong, C.; Liaw, A.; Sheridan, R. P.; Song, Q. Boosting: an ensemble learning tool for compound classification and QSAR modeling. *J. Chem. Inf. Comput. Sci.* **2005**, *45*, 786−799.

(4) Bruce, C. L.; Melville, J. L.; Picket, S. D.; Hirst, J. D. Contemporary QSAR classifiers compared. *J. Chem. Inf. Model.* **2007**, *47*, 219−227.

(5) Svetnik, V.; Liaw, A.; Tong, C.; Culberson, J. C.; Sheridan, R. P.; Feuston, B. P. Random forest: a classification and regression tool for compound classification and QSAR modeling. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 1947−1958.

(6) Fernandez-Delgado, M.; Cernades, E.; Barro, S.; Amorim, D. A. Do we need hundreds of classifiers to solve real world problems? *J. Machine. Learning. Res.* **2014**, *15*, 3133−3181.

(7) Burden, F. R. Quantitative structure-activity relationship studies using Gaussian Processes. *J. Chem. Inf. Comput. Sci.* **2001**, *41*, 830−835.

(8) Hinton, G. E.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; Kingsbury, B. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Processing Magazine* **2012**, *29*, 82−97.

(9) Krizhevsky, A.; Sutskever, I.; Hinton, G. E. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* **2012**, *25*, 1097−1105.

(10) Dahl, G. E.; Jaitly, N.; Salakhutdinov, R. *Multi-task neural networks for QSAR predictions*; 2014; http://arxiv.org/abs/1406.1231; arXiv:1406.1231 [stat.ML].

(11) Chen, B.; Sheridan, R. P.; Hornak, V.; Voigt, J. H. Comparison of random forest and Pipeline Pilot naïve Bayes in prospective QSAR predictions. *J. Chem. Inf. Model.* **2012**, *52*, 792−803.

(12) Sheridan, R. P. Time-split cross-validation as a method for estimating the goodness of prospective prediction. *J. Chem. Inf. Model.* **2013**, *53*, 783−790.

(13) Carhart, R. E.; Smith, D. H.; Ventkataraghavan, R. Atom pairs as molecular features in structure-activity studies: definition and application. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 64−73.

(14) Kearsley, S. K.; Sallamack, S.; Fluder, E. M.; Andose, J. D.; Mosley, R. T.; Sheridan, R. P. Chemical similarity using physiochemical property descriptors. *J. Chem. Inform. Comp. Sci.* **1996**, *36*, 118−27.

(15) Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533−536.

(16) Hinton, G. E.; Osindero, S.; Teh, Y. W. A fast learning algorithm for deep belief nets. *Neural Computation* **2006**, *18*, 1527−1554.

(17) Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929−1958.

(18) Wager, S.; Wang, S.; Liang, P. Dropout training as adaptive regularization. *Advances in Neural Information Processing Systems 26 (NIPS 2013)*; MIT Press: Cambridge, MA, 2013; pp 351−359.

(19) Tieleman, T. *Gnumpy: an easy way to use GPU boards in Python*; Department of Computer Science, University of Toronto, 2010; Technical report UTML TR2010-002.

(20) Mnih, V. *Cudamat: a CUDA-based matrix class for Python*; Department of Computer Science, University of Toronto, 2009; Technical report UTML TR2009-004.

(21) Nair, V.; Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, June 21−24, 2010; pp 807−814.