Review

www.molinf.com

molecular informatics

DOI: 10.1002/minf.201501008

Deep Learning in Drug Discovery

Erik Gawehn,[a] Jan A. Hiss,[a] and Gisbert Schneider*[a]

Abstract: Artificial neural networks had their first heyday in molecular informatics and drug discovery approximately two decades ago. Currently, we are witnessing renewed interest in adapting advanced neural network architectures for pharmaceutical research by borrowing from the field of "deep learning". Compared with some of the other life sciences, their application in drug discovery is still limited. Here, we provide an overview of this emerging field of molecular informatics, present the basic concepts of prominent deep learning methods and offer motivation to explore these techniques for their usefulness in computer-assisted drug discovery and design. We specifically emphasize deep neural networks, restricted Boltzmann machine networks and convolutional networks.

Keywords: bioinformatics · cheminformatics · drug design · machine-learning · neural network · virtual screening

1 Introduction

Machine-learning provides a theoretical framework for the discovery and prioritization of bioactive compounds with desired pharmacological effects and their optimization as drug-like leads. Biological target identification and protein design are emerging areas of application. Among the many machine-learning approaches in molecular informatics, chemocentric methods have found widespread application. Their underlying logic typically follows three steps. First, there is the selection of a problem-specific set of descriptors that are believed to capture the essential properties of the molecules involved. At present, there are over 5000 diverse molecular representations ("descriptors") that address the various properties of chemical entities.[1] Second, a metric or scoring scheme is used to compare the encoded molecules to one another.[2] Finally, a machine-learning algorithm is employed to identify the features that may serve to qualitatively or quantitatively distinguish the active from the inactive compounds.[3] Artificial neural networks (ANNs) were among the first methods borrowed from the computer sciences for this purpose.[4]

In 2013, public attention was drawn to a multi-problem QSAR machine-learning challenge in drug discovery posted by Merck. This competition on drug property and activity prediction was won by a deep learning network with a relative accuracy improvement of approximately 14% over Merck's in-house systems and resulted in an article in *The New York Times*.[5] Here, we present state-of-the-art of advanced chemocentric machine-learning methods with a focus on emerging "deep learning" concepts. We highlight recent advances in the field and point to prospective applications and developments of this potentially game-changing technology for drug discovery. A general task for machine-learning is to uncover the relationship between the molecular descriptors used and the measured activity of the compounds to obtain qualitative classifiers or quantitative structure-activity relationship (QSAR) models. Feature extraction from the descriptor patterns is the decisive step in the model development process.[6,7] In current cheminformatics applications, the prevalent machine-learning architectures are "shallow" and contain a single layer of feature transformation. These architectures include linear and nonlinear principle component analysis, *k*-means clustering methods, partial least square projection to latent structures, decision trees, multivariate linear regression, linear discriminant analysis, support vector machines (SVMs), logistic and kernel regression, multi-layer Perceptrons and related neural network approaches.[8] Although all of these methods have proven to be useful for (Q)SAR modeling and molecular design,[9] the single feature transformation step into a suitable space for the subsequent application of a linear pattern separation model might limit their modeling and representational power when applied to more complex data and setups. A reason for their success in pharmacological applications may stem from the fact that a major part of the complexity inherent to molecular interactions has been engineered into the descriptors employed as patterns for model training, thereby allowing single layer machine-learning architectures to tackle the problem.[10] One challenging question is whether the underlying data complexity and hidden features can be more efficiently dealt with by shifting attention from descriptor engineering to the architecture of the machine-learning system and the training of the algorithms involved. This is the domain of "deep learning".[11,12]

[a] *E. Gawehn, J. A. Hiss, G. Schneider*
Swiss Federal Institute of Technology (ETH), Department of Chemistry and Applied Biosciences, Vladimir-Prelog-Weg 4, CH-8093 Zurich, Switzerland
Fax: +41 44 633 13 79, Tel: +41 44 633 74 38
**e-mail: gisbert.schneider@pharma.ethz.ch*

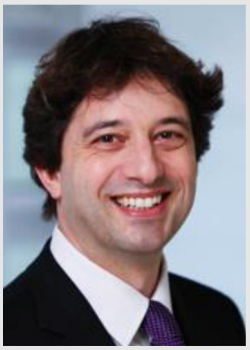
Deep neural networks possess multiple hidden layers and are capable of computing layers of adaptive non-linear features that capture increasingly complex data patterns with each additional layer.[12,13] Deep learning may be particularly well-suited for data mining in the life sciences because this approach deals with complex patterns in nature, systems biology and heterogeneous "big data".[14] Genomic studies currently lead the way in this field.[15] The first deep ANN learning methods were developed in approximately 1980 with the introduction of the Neocognitron[16] and the back-propagation of errors algorithm.[17] In 1988, Qian and Sejnowski published the first application of neural networks in computational (bio)chemistry[18] (protein secondary structure prediction). This application has since been improved through optimized ANN architectures,[19] including some of the recently promoted deep learning approaches.[20] In 1989, LeCun presented a neural network for the recognition of zip codes that learned by error back-propagation.[21] However, this and following deep neural networks suffered from the "vanishing gradient" problem,[22] which was one reason why other machine-learning methods became more prominent at the time. This problem was partially alleviated in 2006 when Hinton *et al.* proposed an unsupervised, greedy layer-by-layer pre-training scheme that targeted the vanishing gradient effect and introduced a new class of deep generative algorithms called "deep belief networks".[23] Although ANNs were soon identified as useful tools for computational drug design,[24] the computationally-oriented medicinal chemists and drug designers have never intensively used deep learning. Instead, other machine-learning methods such as SVMs and random forests[25,26] progressed to dominate the field today. This development occurred in part due to the development of more efficient learning algorithms,[27] but was also due to the desire to interpret and understand the features extracted in a chemically meaningful way, which became increasingly difficult for higher-order features.[28]

There are several examples of cascaded learning, which can be observed as an intermediate step to contemporary deep learning. Cascaded approaches differ from today's approach of deep neural networks in that depth is achieved by combining the strengths of different types of machine-learning algorithms. One representative example is a jury network (weighted voting network)[29,30]. The cascaded jury shown in Figure 1 was successfully applied to predict peptide binding to major histocompatibility complex 1.[31] Its first feature extraction layer consists of ANN and SVM models that are fed by different descriptor sets, and the output values of the first-layer models serve as the input to the jury network. This machine-learning architecture already contains important aspects of deep learning. Since the introduction of deep belief networks approximately one decade ago, deep learning has seen remarkable improvements, such as better regularization techniques,[32] enhanced activation functions,[33] and parallel computing on graphics processing units.[34] For an extensive historic account of deep learning, we refer to a recent article by Schmidhuber.[35]

A rapidly growing number of university groups, start-up companies, and IT global players such as Google[36] and Facebook[37] have joined research efforts to improve deep learning algorithms and develop specialized hardware. For image recognition tasks, deep learning already achieves accuracies comparable to or even surpassing humans on several specialized datasets, including traffic signs,[38] human faces,[39] and handwritten digits.[40] In February 2015, deep learning beat the human performance on the 1000-class ImageNet dataset.[41]

Erik Gawehn studied physics at the Ludwig-Maximilians-Universität Munich, Germany. In 2015, he joined the Computer-Assisted Drug Design at ETH Zurich, Switzerland, as a PhD student. His research focuses on the application of deep neural networks to the virtual screening and the *de novo* design of drug-like molecules with tailored target profiles.

Dr. Jan A. Hiss is a staff scientist in the Computer-Assisted Drug Design group at ETH Zurich. He studied bioinformatics and received his doctoral degree from the Goethe-University Frankfurt, Germany. During his post-doctoral studies at Goethe-University and ETH Zurich, he developed computational methods for analyzing peptide-membrane interaction. His current research focus is on nature-inspired algorithms and computer-assisted peptide design.



Gisbert Schneider is a Full Professor of Computer-Assisted Drug Design at ETH Zurich. He received his doctoral degree in biochemistry from the Freie Universität Berlin. Prior to joining ETH, he worked at F. Hoffmann-La Roche Pharmaceuticals, Basel, and held the Beilstein Endowed Chair for Chem- and Bioinformatics at Goethe-University, Frankfurt, where he now is an adjunct professor. In 2015, he became a Fellow of the University of Tokyo. His main research interests are adaptive autonomous systems for molecular design.

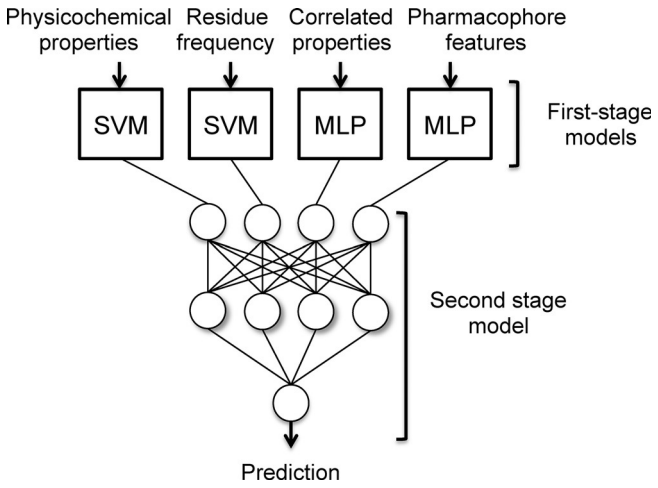


Figure 1. Schematic of a jury network (adapted from ref. 31). In this particular architecture, feature extraction is performed in two stages. The second-stage model is a feed-forward network that weighs the relevance of the first-stage model and improves the overall prediction performance. Note that each first-stage model receives a different set of molecular descriptors as input. SVM, support vector machine; MLP, multi-layer Perceptron.

2 A Simple Deep Network

The most basic form of a deep neural network is a fully connected feed-forward network with more than one hidden layer, as depicted in Figure 2. The term "fully connected" refers to an architecture in which each neuron in a network layer is connected to all non-bias neurons in the next layer, and "feed-forward" indicates that there are no loops or backward connections within the network architecture. A supervised learning problem for an ANN consists of approximating a multidimensional nonlinear function $f(\mathbf{x})$. This approximation is attempted by multiple nonlinear, weighted transformations of the input data vector within the hidden layers. Nonlinearity is implemented into the overall network by defining the appropriate architecture and the use of nonlinear activation functions. Common choices for activation functions are sigmoidal functions, such as the logistic function $a(z)=1/(1+\exp(-z))$ and the rectifier function $a(z)=\max(0,z)$.

During the learning process, the approximation of the network function $f(\mathbf{x})$ is improved by gradually tweaking the network's weights in such a way that the mapping of input data to the computed output values better resembles the unknown function $f(\mathbf{x})$. The network's mapping performance is estimated by comparing its output values o_l calculated for an input data vector $\mathbf{x}$ to the desired true output values t_l using a cost function or error-function $C(o_l)$. An example of a frequently used cost function is the *sum of squared errors* function (SSE) $C\left(\mathrm{o}_l^s\right)=\frac{1}{2}\sum_s\sum_l\left(\mathrm{t}_l^s-\mathrm{o}_l^s\right)^2$, where s is an index denoting one training sample (molecular descriptor vector) fed to the network. For the sake of clarity, Figure 2 shows only a forward and backward pass for one sample, thereby omitting the sum over different input data vectors. Once the cost function is calculated, attempts can be made to minimize it using the weights to increase the network performance. Calculating the gradient of the cost function and subsequently adjusting the weights accordingly achieves weight optimization:

$$\Delta \mathrm{w}_{qp}^{(n)}=-\eta\frac{\partial C}{\partial \mathrm{w}_{qp}^{(n)}}=\begin{cases}-\eta\delta_q^{(n)}\mathrm{x}_p, n=1\\ -\eta\delta_q^{(n)}\mathrm{h}_p^{(n-1)}, n\neq 1\end{cases}, \text{ with}$$

$$\delta_q^{(n)}=\frac{\partial C}{\partial \mathrm{z}_q^{(n)}}\begin{cases}E'\left(\mathrm{o}_q\right)a'\left(\mathrm{z}_q^{(n)}\right), n=N\\ a'\left(\mathrm{z}_q^{(n+1)}\right)\sum_r\delta_r^{(n+1)}\mathrm{w}_{rq}^{(n+1)}, n\neq N\end{cases}$$

where $\Delta \mathrm{w}_{qp}^{(n)}$ denotes the change of one of the network's weights (p, q and r are arbitrary indices for neurons within the involved layers), η is the learning rate, $n\in\{1,\ldots,N\}$ represents the index over the number of network layers (excluding the input layer, which contains fan-out units only), $\mathrm{h}_p^{(n-1)}$ is the output value of a hidden neuron p in layer (n-1), $\mathrm{z}_q^{(n)}$ is the preactivation of a neuron q in layer n, and $a(\ldots)$ is a nonlinear activation function with $a'(\ldots)$ denoting its derivative. The error function depends indirectly on all of the network's weights through the output values $\mathrm{o}_l\left(\mathrm{w}_{lk}^{(N)},\mathrm{h}_k^{(N-1)}\right)$ (Figure 2).

The definition of the quantity $\delta_q^{(n)}$ (called "sensitivity") merely facilitates notation.[42] The sensitivity $\delta_q^{(n)}$ and thus the weight update depend multiplicatively on the sensitivity $\delta_r^{(n+1)}$, the activation function derivative $a'\left(\mathrm{z}_q^{(n+1)}\right)$ and the weights $\mathrm{w}_{rq}^{(n+1)}$ of the next layer. This dependence leads to the problem of vanishing gradients: if these quantities are smaller than one, then the repeated multiplication of small values leads to cost function gradients in the lower layers that rapidly approach zero. Thus, no substantial updates are achieved for the network's lower level weights. For a choice of a logistic function as the activation function, $a'(\ldots)$ will be in a range [0,1] that often results in vanishing gradients. However, the problem has been observed for a wide range of setups. Since Hochreiter *et al.* formally investigated this problem for recurrent neural networks,[43] many additional studies have focused on the issue of vanishing gradients and the closely related problems of exploding gradients or oscillating weights for recurrent neural networks and other deep architectures.[44] Several approaches have been proposed to mitigate these issues, including unsupervised pre-training of deep neural networks layer-by-layer[45] or special architectures such as the *Long Short Term Memory* networks.[46] Successful training of a back-propagation neural network is problem-dependent, and even for a fully connected feed-forward network there are hyper-parameters and choices (*e.g.*, error and activation functions) that may influence the final result. However, some heuristics have been identified that may assist the user in decision-making.[47] For QSAR research, a preliminary

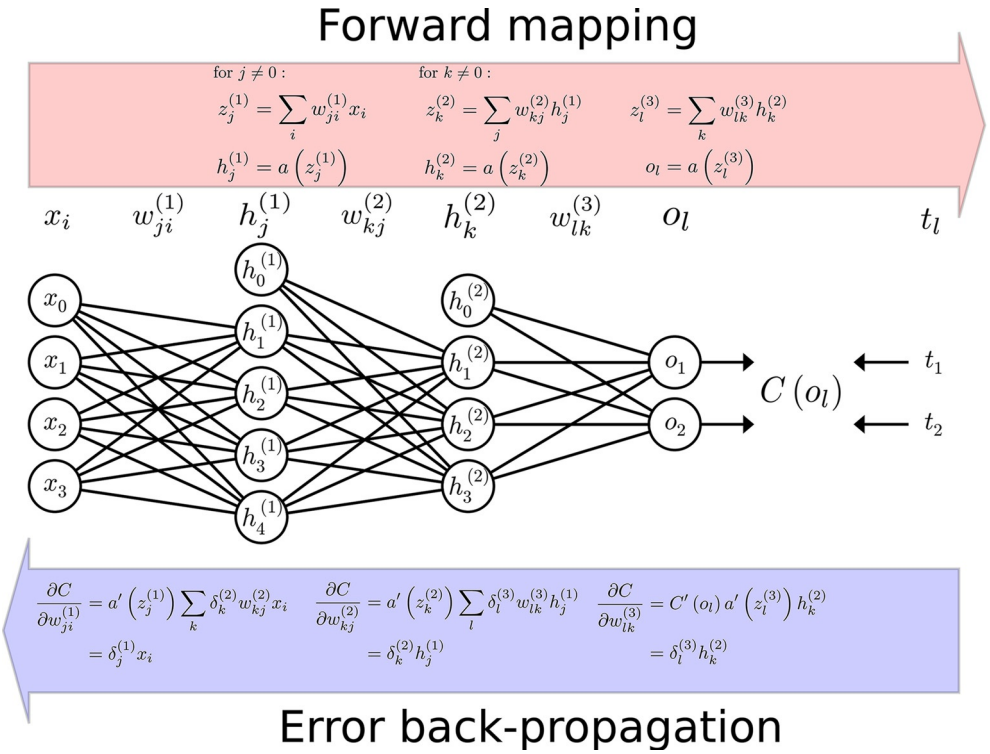


Figure 2. Schematic of a deep feed-forward neural network with one input layer **x**, two hidden layers $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$, and one output layer **o**. The input values for the biases are $x_0 = h_0^{(1)} = h_0^{(2)} = 1$. During a forward pass (red arrow), the pre-activation $z_j^{(1)}$, which is a linear combination of the preceding layer's output values, is calculated for each non-bias neuron in the first hidden layer. Next, a nonlinear activation function $a\left(z_j^{(1)}\right)$ is applied to compute the output values $h_j^{(1)}$. Once the neurons' activations for one layer are calculated, they serve as input values for the following layer until one finally arrives at the network's output values o_l. Then, the output values are evaluated against the true output values t_l using an error-function $C(o_l)$. To resolve the indirect dependency of the error function on the network's weights, the error is back-propagated through the network (blue arrow) using the chain rule for derivatives. Other training techniques may be applied.

guideline for choosing hyper-parameters has already been developed.[4,5] Meta-optimization of neural network parameters by particle swarms and related adaptive stochastic algorithms were also suggested.[48]

Although adding neurons can increase the capacity of deep learning architectures until they can cope with very large data sets, deep neural networks with excessive free parameters can easily overfit even the largest datasets. Therefore, training techniques attempt to stimulate the network to learn the most general of the possible weight combinations. Prominent examples are l_1 and l_2 regularization, Bayesian regularization with penalties built into the cost function, and the automatic relevance determination of the input variables.[49] Another popular technique is referred to as "early termination":[50] The idea behind this technique is to not only separate the dataset into a training and a test set, but also to further separate the training set into a training and at least one external validation set. Then, the network is optimized using the training data and simultaneously tested on the validation set. By stopping the training procedure as soon as the accuracy starts to decrease on the validation set, the risk of overfitting on the training data can be reduced. Another widely used technique is called "dropout".[32] For the dropout method, a certain percentage of randomly selected neurons in a layer "vanish" (*i.e.*, the activation function is fixed to *output* = 0) on each presentation of each training case. Dropout is believed to lead to a competition of neurons, thereby preventing collaboration because one neuron cannot rely on the presence of other neurons. Thus, each neuron is forced to learn some general feature. Another way of looking at dropout regularization is as an averaging model over an exponential number of the many different neural networks produced by deleting random subsets of hidden units and inputs. This idea of an adaptive network architecture has been proposed before and is currently receiving renewed attention.[51] One of the early applications was the sequence-based prediction of the transmembrane helices of integral membrane proteins.[52]

3 Properties of Deep Learning Architectures

There are several advantages of deep learning neural networks that suggest that they should replace shallow ANNs:

(i) A deep, hierarchical architecture enables ANNs to perform many nonlinear transformations, which leads to

the learning of more abstract features compared to shallow networks.[53] This development allows for a more sophisticated combination of low-level features (*i.e.*, nonlinear descriptor combinations) to achieve a better molecular representation and thus classification of compounds. This approach reduces the need for the *a priori* engineering of more sophisticated descriptors. More abstract nonlinear features tend to be invariant to local changes in the input, leading to inherent noise reduction and increased network robustness. Therefore, deep learning architectures capture certain families of functions exponentially more efficiently than shallow architectures.

(ii) Deep architectures promote the reuse of features. This ensemble view of the same data corresponds to an improvement on the principle of cascading networks that share the intelligence of different algorithms. How well-regularized deep learning networks can actually exploit commonalities between different tasks to transfer knowledge as inductive bias remains a matter of debate. This aspect is relevant for missing classes within the training set, which is typical for large QSAR data sets. Aside from enabling "educated guesses" on missing information, multitask learning also has the benefits of presenting a shared, learned feature extraction pipeline for multiple tasks. Dahl suggested that this concept might have a regularization effect because weights tend to develop in a way that is useful for many targets instead of overfitting one target in particular.[54] All of these findings promise the simultaneous testing of bioactivities against many related targets at a realistic computational cost. Such applications may be driven even further by the field of polypharmacology.[55] In this case, deep learning might accompany current kernel methods to find additional applications for drugs ("re-purposing") and to assist in the identification of undesired off-target activities.[56]

(iii) As highlighted by the Merck QSAR contest,[5] deep multitask neural networks can be successfully applied to QSAR modeling.[57] Generative deep architectures complement evolutionary algorithms in computational drug discovery that are often used to find new compounds with specific features identified by some previously trained machine-learning architecture.[58] A trained deep generative neural network has learned to separate compound activity classes within its architecture while being able to generate output according to the learned representation; thus, this network contains both functions in one algorithm. Therefore, deep learning might provide a fresh approach to solving the "inverse QSAR problem".[59]

However, simply trading network width for depth alone does not automatically lead to better models. In the following sections, we highlight two deep learning architectures that we think deserve special attention and consideration for drug design and discovery: the *Restricted Boltzmann Machine* (RBM) and *Convolutional Neural Networks* (CNN). Many more deep learning approaches have been conceived, and the interested reader is referred to the respective literature.[35,66]

4 Restricted Boltzmann Machine

Boltzmann machines are undirected, generative, stochastic neural networks that rose to prominence when Hinton and coworkers proposed *contrastive divergence*[60] as a fast unsupervised learning algorithm.[61] Restricted Boltzmann Machines (RBM) are instances of Boltzmann machines without intra-layer interactions (Figure 3). Boltzmann machines are

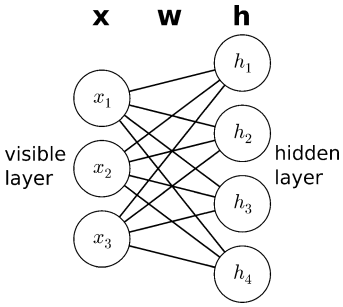


Figure 3. Schematic of a restricted Boltzmann machine. The visible layer is the interface between the network and the outside world. Input can be clamped to this layer, and the resulting visible neuron configuration represents a sample from the network's equilibrium distribution after an equilibration process. Hidden neurons (latent variables) transform the input data and detect prominent features. The hidden layer models the visible layer in conjunction with the weight matrix.

not feed-forward models, which is a crucial difference from the deep feed-forward neural networks presented in Figure 2. The given input states serve as input to the hidden neurons but in turn are dependent on the resulting hidden neuron states that modify the "visible" neurons. This back and forth of mutual layer influencing eventually leads to an equilibrium of neuron activations regardless of the originally given input vector (Figure 4). The overall distribution governing the activation of visible neurons at one of these steps is the network's "belief" of how the visible neurons behave. The goal is to nudge the network's belief of the visible neuron state distribution into the direction of the actual input data distribution by intermittently adding new input data vectors and adjusting the weights in such a way that their probability is increased. The idea is that over time the believed visible state distribution eventually approximates the real state distribution, i.e. the network "learns" the input distribution in an unsupervised fashion. A popular criterion to evaluate the learning process is the reconstruction error, which is a measure of the ability of the

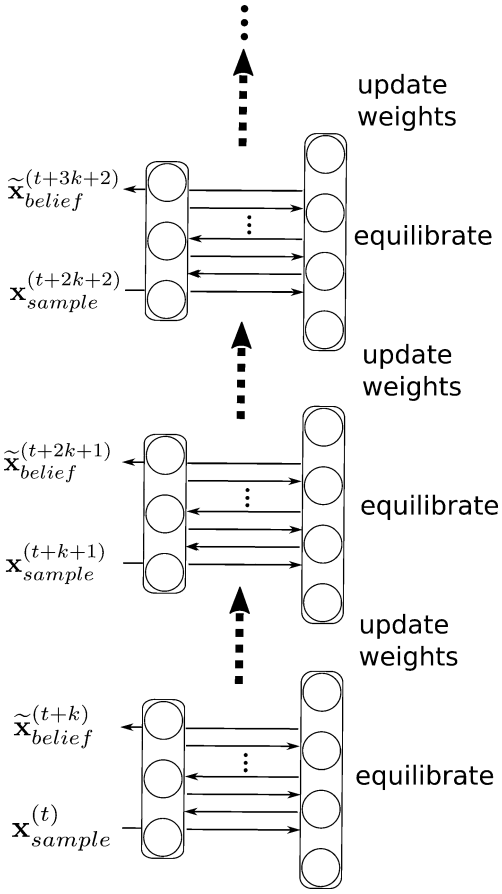


Figure 4. Restricted Boltzmann machine-learning. First, an input vector from the dataset enters the network. Subsequently, the hidden layer state is calculated with regard to the new input vector, from which a new input layer state can be calculated by the network. This iterative process continues for *k* steps, and the resulting visible layer state serves as a representative for the network's belief in the visible layer states. A weight update is calculated and the training cycle is re-iterated from the data input vector and the belief state vector.

network to produce an output that is consistent with the input data.[62]

Neurons in a basic RBM are binary and stochastic. The RBM's overall state under the current weight configuration is described by the energy function

$$E(\mathbf{x},\mathbf{h}) = -\sum_i \sum_j w_{ji} x_i h_j - \sum_i b_i x_i - \sum_j c_j h_j \ ,$$

where x_i denotes the value for visible neuron *i*, h_j is the output value of hidden neuron *j*, and w_{ij} is the weight connecting the two. The parameters b_i and c_j denote the bias for visible layer neurons and hidden layer neurons, respectively. These parameters differ from the deep feed-forward network where input neuron bias is not considered and the hidden layer bias is usually incorporated into the weight matrix. The dynamics within the RBM network depend on whether w_{ij} is a positive or negative connection between neurons x_i and h_j, which indicates whether they tend to collaboratively fire or hinder each other's firing as well as each neuron's intrinsic bias to fire. The Boltzmann probability of a joined configuration of visible and hidden units is

$$p(\mathbf{x},\mathbf{h}) = \frac{e^{-E(\mathbf{x},\mathbf{h})}}{Z} \ , \text{ with}$$

$$Z = \sum_{\mathbf{x},\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})} \ ,$$

where *Z* denotes the RBM's combinatorial partition function. $\sum_{\mathbf{x}}$ and $\sum_{\mathbf{h}}$ denote the sum over all possible states of the input layer neurons and hidden layer neurons, respectively. Due to the generally extremely large number of possible states, the partition function is considered to be intractable. Marginalizing over all corresponding hidden configurations derives the probability of a particular configuration x of visible units:

$$p(\mathbf{x}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}}{Z} .$$

To maximize the probability of the data-vectors, the parameters Θ (weights and biases) are adjusted so that the negative log-likelihood of the corresponding states is minimized:

$$\boldsymbol{\Theta} = \boldsymbol{\Theta} - \eta\left(\nabla_{\boldsymbol{\Theta}}\left(-\log p\left(\mathbf{x} = \mathbf{x}^{(t)}\right)\right)\right) \ ,$$

where $\mathbf{x}^{(t)}$ signifies the visible layer configuration of a particular sample inserted into the network at time *t*, and Θ is a parameter at that time *t*. Deriving the log-likelihood within the brackets with regard to the parameter Θ yields:

$$\frac{\partial}{\partial_\Theta}\left(-\log p\left(\mathbf{x}^{(t)}\right)\right) =$$

$$\frac{-\sum_{\mathbf{x},\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})} \partial_\Theta\left(E(\mathbf{x},\mathbf{h})\right)}{Z} + \frac{\sum_{\mathbf{h}} e^{-E\left(\mathbf{x}^{(t)},\mathbf{h}\right)} \partial_\Theta\left(E\left(\mathbf{x}^{(t)},\mathbf{h}\right)\right)}{\sum_{\mathbf{h}} e^{-E\left(\mathbf{x}^{(t)},\mathbf{h}\right)}}$$

$$= -\left\langle \partial_\Theta\left(E(\mathbf{x},\mathbf{h})\right)\right\rangle_{\mathbf{x},\mathbf{h}} + \left\langle \partial_\Theta\left(E\left(\mathbf{x}^{(t)},\mathbf{h}\right)\right)\right\rangle_{\mathbf{h}|\mathbf{x}^{(t)}} .$$

The expressions $\langle \ldots \rangle_{\mathbf{x},\mathbf{h}}$ and $\langle \ldots \rangle_{\mathbf{h}|\mathbf{x}^{(t)}}$ denote the expected values based on the probability distributions over all possible configurations or the conditional values depending on the current sample configuration. The intuition for the two expected values (called the "negative phase" and the "positive phase") is as follows: the positive phase increases the probability of the input vector $\mathbf{x}^{(t)}$, whereas the negative phase decreases the probability of the model's belief (*i.e.*, the probability of all other model configurations). For a weight parameter $\Theta = w_{ji}$, the positive phase becomes

$$\left\langle -h_j x_i \right\rangle_{\mathbf{h}|\mathbf{x}^{(t)}} = \sum_{h_j=\{0.1\}} -h_j x_i p\left(h_j|\mathbf{x}^{(t)}\right) = -x_i p\left(h_j = 1|\mathbf{x}^{(t)}\right).$$

The lack of intra-layer interactions in RBMs suggests that the hidden layer neurons are independent of one another, which is also true for the visible layer neurons: $p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{h}) = \prod_i p(x_i|\mathbf{h})$. Using this independence leads to expressions for the inference of states that resemble the calculation of neuron activations in deep neural networks:

$$p(h_j = 1|\mathbf{x}) = \sigma\left(\sum_i w_{ji} x_i + c_j\right),$$

$$p(x_i = 1|\mathbf{h}) = \sigma\left(\sum_j h_j w_{ji} + b_i\right).$$

A random number *rand* uniformly drawn from the interval [0,1] can be used to model the neurons' stochastic behavior:

$$h_j = \begin{cases} 1, p(h_j = 1|\mathbf{x}) > rand \\ 0, p(h_j = 1|\mathbf{x}) \leq rand \end{cases}$$

The input vector $\mathbf{x}^{(t)}$ can now be used to infer a sample vector $\tilde{\mathbf{h}}^{(t)}$ from the hidden layer to calculate the positive phase. Moreover, this inferred layer configuration can be reused to sample from the original layer distribution: $\mathbf{x}^{(t)} \rightarrow \tilde{\mathbf{h}}^{(t)} \rightarrow \tilde{\mathbf{x}}^{(t+1)}$. This method, in which each variable draws a sample from its posterior while keeping the other variables fixed, is also known as Gibbs sampling.[63] Essentially, the same trick of inferring states by using conditional probabilities may be used to derive an expression for the negative phase term. The expected value of the negative phase is replaced by a point estimate $\tilde{\mathbf{x}}$ that corresponds to a one-step Monte-Carlo approximation of the expected value. This $\tilde{\mathbf{x}}$ is obtained by performing k steps of Gibbs sampling starting from a data vector $\mathbf{x}^{(t)}$ (Figure 4). For an infinite number of Gibbs sampling steps ($k = \infty$), this procedure results in an unbiased sample from the network's believed distribution over the visible states. Hence, the new expression for the log-likelihood derivation reads

$$\frac{\partial}{\partial_\Theta}\left(-\log p\left(\mathbf{x}^{(t)}\right)\right) \approx -\partial_\Theta\left(E\left(\tilde{\mathbf{x}}^{(t+k)}, \tilde{\mathbf{h}}^{(t+k-1)}\right)\right) + \partial_\Theta\left(E\left(\mathbf{x}^{(t)}, \tilde{\mathbf{h}}^{(t)}\right)\right).$$

Hinton and coworkers showed that a single iteration ($k = 1$) of Gibbs sampling might be sufficient for an RBM to efficiently find features in the input data, even if the log-likelihood estimate might not be perfect.[64] The corresponding method was termed "contrastive divergence learning", which is not derived by minimizing a negative log-likelihood but by minimization of another objective function. In 2006, Hinton *et al.* proposed a new method to stack RBM modules and initialize the weights between all of the layers in a way that enabled fast training of the resulting "deep belief network".[23] An advantage of using several layers is that the lower bound on $\log p(\mathbf{x}^{(t)})$ increases with each additional layer added. Although this does not necessarily mean that $\log p(\mathbf{x}^{(t)})$ improves with additional layers, this approach ensures that the model does not deteriorate beyond a certain point.[23,65] To make a deep belief network discriminative, an additional classifier can be added on top. If the training data labels are known (*e.g.*, compound activity classes), back-propagation can subsequently be used to fine-tune the network weights. The resulting deep architecture has its weights initialized so that they already correspond to a representation of the input data distribution. This type of unsupervised RBM pre-training can lead to better weight initialization with the additional advantage of using unlabeled data. Because the method of unsupervised pre-training addresses the problem of network overfitting, it can be considered a strong regularization procedure.[66] These properties render the approach particularly appealing for big data analysis in drug discovery. A recent pioneering RBM study achieved near-perfect ROC AUC values for the prediction of drug-target interactions.[67]

5 Convolutional Neural Network

CNNs are inspired by neuroscience and imitate the image representation within the visual cortex that is created by successive transformations of the retinal signals in cortical areas called the "ventral stream" that end in the inferior temporal (IT) cortex.[68] Contemporary CNNs rival the IT-cortex's representational performance of a primate in "core visual object recognition".[69] The prototype of CNNs is the Neocognitron,[16] which was conceived in the early 1980s. Related architectures for phoneme recognition followed a few years later; thereafter, CNNs for object recognition in images and document reading were developed just prior to the turn of the millennium.[70] CNN improvement together with the general advances in deep learning became a focal point of interest when a deep CNN won the ILSVRC image recognition challenge in 2012 by a wide margin.[71] CNNs have dominated this challenge ever since, as well as the field of object recognition and detection in images in general. They now constitute the underlying architecture for a wide range of applications ranging from image understanding, video analysis, speech recognition and language understanding to artificial intelligence.[72]

The two guiding principles of weight-sharing and local connectivity in a convolutional net allow for the detection of features within topologically structured input data. Additional use of pooling and many layers enable the network to learn increasingly abstract features with each additional layer. The concept of weight sharing and local connectivity is depicted in Figure 5. Here, the input data corresponds to the input image, the filter corresponds to the weight-connections, and the feature map (activation map) corresponds to the first hidden layer. In contrast to a fully connected network (Figure 2), hidden-layer neurons are only connect-

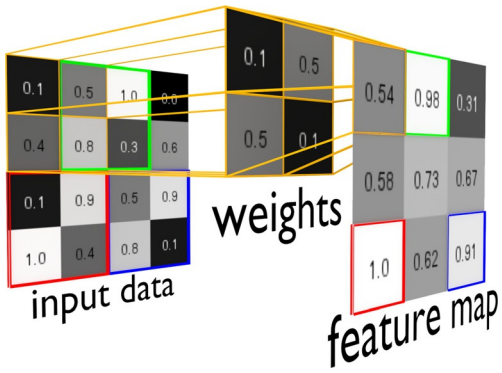


Figure 5. The figure depicts a toy example of a greyscale input image with pixel values between 0 and 1, a weight matrix (filter) and the resulting feature map. Pre-activations of hidden neurons within the feature map are obtained by weighting pixel-values within a 2×2 receptive field with the corresponding weights of the filter and summing up the resulting four terms as shown in orange. To obtain the next feature map value, the filter is moved across the image with a stride of 1 (*e.g.*, from the orange to the green quadrant). The resulting feature map is a map of where the feature encoded in the filter has occurred in the input image. Here, the filter corresponds to a diagonal that results in a strong pre-activation signal for the green, red and blue filter positions over the input image, which is reflected in the feature map accordingly.

ed to a local patch within the input image. In Figure 5, only four input values are connected to four weights and contribute to the pre-activation of a hidden neuron in the feature map instead of all 16 input values weighted accordingly in a fully connected layer. The spatial extent of this local patch is a hyper-parameter called the *receptive field*. Not considering biases, the pre-activation z of a hidden layer neuron in the feature map is computed by summing up the weighted input values within the receptive field that is multiplied by their corresponding weights (in Figure 5 this step is highlighted in orange: $0.1\cdot0.1+0.5\cdot0.5+0.4\cdot0.5+0.8\cdot0.1=0.54$):

$$(\boldsymbol{x}\star\boldsymbol{w})_{ij}=\sum_{ab}x_{i+a,j+b}w_{i+a,j+b}\ ,$$

where $\boldsymbol{x}$ corresponds to the input matrix, $\boldsymbol{w}$ signifies the weight matrix and a and b are two indices running from 0 to $r-1$ to capture the entire quadrant of size $r\times r$. Another way to achieve the same result is to transform the weight matrix into a convolution kernel **k** by flipping its rows and columns and subsequently performing a convolution of the input image:

$$(\boldsymbol{x}*\boldsymbol{k})_{ij}=\sum_{ab}x_{i+a,j+b}k_{r-a,r-b}\ .$$

This latter approach is usually employed, and it is this perspective that coined the notion of a "convolutional network".

Further pre-activation values within a feature map are calculated by moving the filter across the image by a predefined step size (*stride*) and calculating the pre-activations for each new position. The weights are kept constant while the weight-filter moves across the image, leading to the aforementioned sharing of weights among hidden neurons organized in the same feature map. This reduces the overall number of free parameters and the computational cost of calculating the linear pre-activation of a hidden neuron. Ultimately, this training process results in a feature map (a map of where the feature encoded in the filter has been found in the input data).[73, 74]

Different filters result in different feature maps. Similar to fully connected networks, a CNN gradually adjusts the weights to better approximate the target function. This technique corresponds to an adjustment of the filters and hence the features that are detected in the input data. The network "learns" the features it requires to minimize the error function.

Often the notion of an "input volume" and "output volume" is employed, which becomes clearer when the input data is composed of several input channels with a similar topology, such as the case of an RGB image (Figure 6). The multi-component weight filters (*filter banks*) between the input volume and the first hidden volume usually have as many components as the input volume. For the connections between higher hidden volumes, filter banks do not necessarily act on the entire depth of the input hidden volume.

A closer look at the convolutional method described and depicted in Figure 5 shows that the pixel values at the image boundaries are less often considered when sliding the convolution kernel across the image than values from the center of the image. To counteract these boundary effects, the input data for this convolution operation can be lined with additional rows and columns of zeroes, which leads to the effect that the original data is now located further towards the center within the new, larger data set.

It is also common to subsample feature maps by intermittently inserting pooling layers between convolutional layers.[75] In addition to reducing the size of the feature

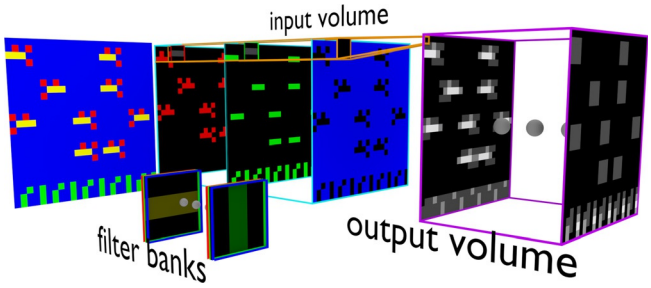


Figure 6. The image shows a toy example of different filter banks being applied to the three components of an RGB input volume. This results in different convolutional neural network feature maps forming an output volume (the first hidden volume).

Table 1. Applications of deep learning in drug discovery. There is ample opportunity to fill the uncharted fields.

Deep learning architecture	Virtual screening, QSAR, proteome mining	ADMET properties	Protein structure and function	Other applications
Deep feed-forward network	[5, 54, 57, 84–87]	[88,98]	[85, 89–92]	[93]
Deep restricted Boltzmann machine			[94, 95]	
Deep auto-encoder network			[90–92,96]	
Recursive neural network		[97,98]	[99–101]	[102]
Deep convolutional network	[103,104]	[105]		[102,106]
Cascaded neural network	[31]	[107]		
Other architectures	[85]		[85,108–109]	[110]

maps and thus the number of parameters, this method also improves the generalization of the features the network learns (because these features are derived from more coarse-grained feature maps in the preceding hidden volume). The most common method is MAX pooling, in which the feature map is subdivided and the MAX operation is applied to every resulting tile. This step is repeated for every feature map within a hidden volume. Recently, the notion of an *all-convolutional net* has been introduced, which abandons pooling layers altogether and seeks to achieve better generalization using a larger stride for the convolution operation.[76] Following the concept of jury networks, advanced CNN versions include "multi column" deep neural networks that effectively represent a combination of separately and differently initialized CNNs that are combined into one big network. The final prediction results from averaging all of the CNN predictions.[40]

A challenge for successful CNN applications in drug discovery will be to find appropriate molecular representation schemes to serve as the input. For example, a first positron emission tomography image analysis has been performed with CNNs.[77] Further applications are listed in Table 1.

6 Conclusion: Deep Learning to the Rescue?

Although the concepts of deep learning were introduced with the conception of multi-layer neural networks,[78] the life science community has adapted some of these techniques only recently (Table 1). In fact, neural networks have a long and fruitful history in drug discovery and design. Because they bear the risk of being easily over-trained and are perceived as a "black box", they have often been substituted by other approaches such as SVM models.[27,79] As noted by Winkler in a review article from 2004, continuous methodological advances in the field of neural networks alleviate some of the pitfalls and may have much to offer for hit and lead discovery.[80] Evidently, deep network architectures will require a particularly careful analysis and thorough definition of their respective domains of applicability,[81,82] and ideally provide hands-on guidelines for chemists. Appropriate confidence measures for neural network classifiers have been developed and can readily be applied.[83] Full appreciation of deep learning methods will likely only be achieved when the user gains an understanding of the underlying molecular features that enable pattern association and classification. With the development of new deep learning concepts such as RBMs and CNNs, the molecular modeler's tool box has been equipped with potentially game-changing methods. Judging from the success of recent pioneering studies, we are convinced that modern deep learning architecture will be useful for the coming age of big data analysis in pharmaceutical research, toxicity prediction, genome mining and chemogenomic applications, to name only some of the immediate areas of application. Personalized health care, in particular, could benefit from these capabilities. However, it may be wise not to put all of our eggs in one basket. We still need to fully understand the advantages and limitations of deep learning techniques. One should not blindly use them without simultaneously applying straightforward linear methods and pursuing chemical similarity approaches. By maintaining some healthy skepticism, we feel that the time is ripe for (re)discovering and exploring the usefulness of deep learning in drug discovery.

Conflict of Interest

G. S. is a co-founder of inSili.com LLC, Zurich, and a consultant to the pharmaceutical industry.

Acknowledgements

This research was supported by the Swiss National Science Foundation (grant no. 200021_157190 and CR32I2_159737).

References

[1] a) R. Todeschini, V. Consonni, *Molecular Descriptors for Chemoinformatics*, Wiley-VCH, Weinheim, **2009**; b) R. Sawada, M. Kotera, Y. Yamanishi, *Mol. Inf.* **2014**, *33*, 719 – 731.
[2] P. Willett, *Mol. Inf.* **2014**, *33*, 403 – 413.

[3] a) A. Lavecchia, *Drug Discovery Today* **2015**, *20*, 318 – 331; b) E. J. Gardiner, V. J. Gillet, *J. Chem. Inf. Model.* **2015**, *55*, 1781 – 1803.

[4] a) J. Devillers (ed.), *Neural Networks in QSAR and Drug Design*, Elsevier, Amsterdam; b) G. Schneider, P. Wrede, *Prog. Biophys, Mol. Biol.* **1998**, *70*, 175 – 222; c) J. Zupan, J. Gasteiger, *Neural Networks in Chemistry and Drug Design*, Wiley-VCH, Weinheim, **1999**; d) S. Agatonovic-Kustrin, R. Beresford, *J. Pharm. Biomed. Anal.* **2000**, *22*, 717 – 727.

[5] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, V. Svetnik, *J. Chem. Inf. Model.* **2015**, *55*, 263 – 274; b) J. Markoff, *Scientists See Promise in Deep-Learning Programs. The New York Times*, November 23, **2012**.

[6] a) G. Schneider, S.-S. So, *Adaptive Systems in Drug Design*, Landes Bioscience, Georgetown, **2001**; b) M. Reutlinger, G. Schneider, *J. Mol. Graph. Model.* **2012**, *34*, 108 – 117.

[7] G. E. Hinton, *Cogn. Sci.* **2014**, *38*, 1078 – 1101.

[8] a) J. Gasteiger, T. Engel (eds.), *Chemoinformatics: A Textbook*, Wiley-VCH, **2003**; b) G. Schneider, G. Downs (eds.), *Machine Learning Methods in QSAR Modelling, QSAR Comb. Sci.* **2003**, *22(5)*; c) J. Bajorath (ed.) *Chemoinformatics: Concepts, Methods, and Tools for Drug Discovery*, Humana Press, Totowa, **2004**; d) M. Dehmer, K. Varmuza, D. Bonchev (eds.) *Statistical Modelling of Molecular Descriptors in QSAR/QSPR*, Wiley-Blackwell, New York, **2011**.

[9] a) G. Schneider, K.-H. Baringhaus, *Molecular Design - Concepts and Applications*. Wiley-VCH, Weinheim, **2008**; b) G. Schneider (ed.), *De Novo Molecular Design*. Wiley-VCH, Weinheim, **2013**.

[10] J. Gasteiger, *Mini Rev. Med. Chem.* **2003**, *3*, 789 – 796.

[11] Y. Bengio, *Foundations and Trends in Machine Learning* **2009**, *2*, 1 – 127.

[12] a) G. E. Hinton, R. R. Salakhutdinov, *Science* **2006**, *313*, 504 – 507; b) Y. LeCun, Y. Bengio, G. Hinton, *Nature* **2015**, *521*, 436 – 444.

[13] Y. Roudi, G. Taylor, *Curr. Opin. Neurobiol.* **2015**, *35*, 110 – 118.

[14] a) J. Fan, H. Liu, *Adv. Drug Deliv. Rev.* **2013**, *65*, 987 – 1000; b) D. C. Cireşan, A. Giusti, L. M. Gambardella, J. Schmidhuber, *Med. Image Comput. Comput. Assist. Interv.* **2013**, *16*, 411 – 418; c) N. T. Issa, S. W. Byers, S. Dakshanamurthy, *Expert Rev. Clin. Pharmacol.* **2014**, *7*, 293 – 298; d) F. F. Costa FF, *Drug Discovery Today* **2014**, *19*, 433 – 440; e) S. J. Lusher, R. McGuire, R. C. van Schaik, C. D. Nicholson, J. de Vlieg, *Drug Discovery Today* **2014**, *19*, 859 – 868; f) J. B. Brown, M. Nakatsui, Y. Okuno, *Mol. Inf.* **2014**, *33*, 732 – 741; g) L. Richter, G. F. Ecker, *Drug Discovery Today Technol.* **2015**, *14*, 37 – 41.

[15] Y. Park, M. Kellis, *Nat. Biotechnol.* **2015**, *33*, 825 – 826.

[16] a) K. Fukushima, S. Miyake, *Pattern Recognit.* **1982**, *15*, 455 – 469; b) K. Fukushima, *Neural Netw.* **2013**, *37*, 103 – 119.

[17] a) P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, **1974**; b) D. E. Rumelhart, G. E. Hinton, R. J. Williams, *Nature* **1986**, *323*, 533 – 536.

[18] N. Qian, T. J. Sejnowski, *J. Mol. Biol.* **1988**, *202*, 865 – 884.

[19] M. Punta, B. Rost, *Methods Mol. Biol.* **2008**, *458*, 203 – 230.

[20] M. Spencer, J. Eickholt, J. Cheng, *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2015**, *12*, 103 – 112.

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, LD. Jackel, *Adv. Neural Inf. Proc. Sys. (NIPS)* **1989**, *2*, 396–404.

[22] a) Y. Bengio, P. Simard, P. Frasconi, *IEEE Trans. Neural Networks* **1994**, *5*, 157 – 166; b) S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, in: *A Field Guide to Dynamical Recurrent Neural Networks* (S. C. Kremer, J. F. Kolen, eds.), IEEE Press, New York, **2001**.

[23] a) G. E. Hinton, *Trends Cogn. Sci.* **2007**, *11*, 428 – 434; b) G. E. Hinton, S. Osindero, Y. W. The, *Neural Comput.* **2006**, *7*, 1527 – 1554.

[24] a) G. Schneider, J. Schuchhardt, P. Wrede, *Comput. Appl. Biosci.* **1994**, *10*, 635 – 645; b) G. Schneider, *Neural Netw.* **2000**, *13*, 15 – 16; c) S. Agatonovic-Kustrin, R. Beresford, *J. Pharm. Biomed. Anal.* **2000**, *22*, 717 – 727.

[25] a) L. Breiman, *Machine Learning* **2001**, *45*, 5 – 32; b) N. Meinshausen, *J. Mach. Learn. Res.* **2006**, *7*, 983 – 999.

[26] a) Y. Sakiyama, *Expert Opin. Drug Metab. Toxicol.* **2009**, *5*, 149 – 169; b) D. Plewczynski, S. A. Spieser, U. Koch, *Comb. Chem. High Throughput Screen.* **2009**, *12*, 358 – 368; c) B. Sprague, Q. Shi, M. T. Kim, L. Zhang, A. Sedykh, E. Ichiishi, H. Tokuda, K. H. Lee, H. Zh, *J. Comput. Aided Mol. Des.* **2014**, *28*, 631 – 646; d) M. A. Khamis, W. Gomaa, W. F. Ahmed, *Artif. Intell. Med.* **2015**, *63*, 135 – 152; e) T. Rodrigues, D. Reker, M. Welin, M. Caldera, C. Brunner, G. Gabernet, P. Schneider, B. Walse, G. Schneider, *Angew. Chem. Int. Ed.* **2015**, in press.

[27] a) E. Byvatov, U. Fechner, J. Sadowski, G. Schneider, *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 1882 – 1889; b) E. Byvatov, G. Schneider, *Appl. Bioinformatics* **2003**, *2*, 67 – 77.

[28] a) L. Yang, P. Wang, Y. Jiang, J. Chen, *J. Chem. Inf. Model.* **2005**, *45*, 1804 – 1811; b) I. I. Baskin, V. A. Palyulin, N. S. Zefirov, *Methods Mol. Biol.* **2008**, *458*, 137 – 158.

[29] A. Givehchi, G. Schneider, *Mol. Divers.* **2005**, *9*, 371 – 383.

[30] a) A. C. Tsoi, A. D. Back, *IEEE Trans. Neural Netw.* **1994**, *5*, 229 – 239; b) B. Hammer, A. Micheli, A. Sperduti, M. Strickert, *Neural Netw.* **2004**, *17*, 1061 – 1085; c) D. V. Buonomano, *Neuron* **2009**, *63*, 423 – 425; d) D. Plewczynski, *J. Mol. Model.* **2011**, *17*, 2133 – 2141.

[31] a) J. A. Hiss, A. Bredenbeck, F. O. Losch, P. Wrede, P. Walden, G. Schneider, *Protein Eng. Des. Sel.* **2007**, *20*, 99 – 108; b) C. P. Koch, A. M. Perna, M. Pillong, N. K. Todoroff, P. Wrede, G. Folkers, J. A. Hiss, G. Schneider *PLoS Comput. Biol.* **2013**, *9*, e1003088; c) C. P. Koch, A. M. Perna, S. Weissmüller, S. Bauer, M. Pillong, R. B. Baleeiro, M. Reutlinger, G. Folkers, P. Walden, P. Wrede, J. A. Hiss, Z. Waibler, G. Schneider, *ACS Chem. Biol.* **2013**, *8*, 1876 – 1881.

[32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov, *Cornell University Library* **2012**, arXiv:1207.0580.

[33] F. Agostinelli, M. Hoffman, P. Sadowski, P. Baldi, *Cornell University Library* **2014**, arXiv:1412.6830.

[34] R. Raina, A. Madhavan, A. Y. Ng, *Proc. 26th* Annual Int. Conf. Mach. Learn. ICML **2009**, 40, 1 – 8.

[35] J. Schmidhuber, *Neural Networks* **2015**, *61*, 85 – 117.

[36] Q. V. Le, *IEEE International Conference on Acoustics, Speech and Signal Processing* **2013**, 8595 – 8598.

[37] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, *IEEE Conference on Computer Vision and Pattern Recognition* **2014**, 1701 – 1708.

[38] D. Cireşan, U. Meier, J. Masci, J. Schmidhuber, *Neural Networks* **2012**, *32*, 333 – 338.

[39] Y. Sun, Y. Chen, X. Wang, X. Tang, *Advances in Neural Information Processing Systems* **2014**, 1988 – 1996.

[40] a) D. Ciresan, U. Meier, J. Schmidhuber, *Comput. IEEE Conference on Visual Pattern Recognition (CVPR),* **2012**, 3642 – 3649, doi:10.1109/CVPR.2012.6248110; b) L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, R. Fergus, *Proceedings of the 30th* International Conference on Machine Learning (ICML-13) **2013**, 1058 – 1066.

[41] K. He, X. Zhang, S. Ren, J. Sun, *Cornell University Library* **2015**, arxiv.org/abs/1502.01852.

[42] R. O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification*, 2nd ed., Wiley, New York, **2000**.
[43] S. Hochreiter, *Diploma Thesis, Tech. Univ. München*, Title: "Untersuchungen zu dynamischen neuronalen Netzen," **1991**.
[44] R. Pascanu, T. Mikolov, Y. Bengio, *Proceedings of the 30th International Conference on Machine Learning* **2012**, 1310–1318.
[45] J. Schmidhuber, *Neural Comput.* **1992**, *4*, 131–139.
[46] S. Hochreiter, J. Schmidhuber, *Neural Comput.* **1997**, *9*, 1735–1780.
[47] G. Montavon, G. B. Orr, K.-R. Müller (eds.), *Neural Networks: Tricks of the Trade*. Springer, Berlin, **2012**.
[48] a) M. Meissner, M. Schmuker, G. Schneider, *BMC Bioinformatics* **2006**, *7*, 125; b) G. Hanrahan, *Analyst* **2011**, *136*, 3587–3594; c) W. Van Geit, E. De Schutter, P. Achard, *Biol. Cybern.* **2008**, *99*, 241–251.
[49] a) I. I. Baskin, V. A. Palyulin, N. S. Zefirov, Neural networks in building QSAR models. *Methods Mol. Biol.* **2008**, *458*, 137–158; b) F. Burden, D. Winkler, Bayesian regularization of neural networks. *Methods Mol. Biol.* **2008**, *458*, 25–44; c) J. Tao, S. Wen, W. Hu, L1-norm locally linear representation regularization multi-source adaptation learning. *Neural Networks* **2015**, *69*, 80–98; d) K. Li, J. Deng, H. B. He, Y. Li, D. J. Du, *Int. J. Comput. Biol. Drug Des.* **2010**, *3*, 112–132.
[50] A. Tropsha, *Mol. Inf.* **2010**, *29*, 476–488.
[51] a) M. M. Islam, M. A. Sattar, M. F. Amin, X. Yao, K. Murase, *IEEE Trans. Syst. Man Cybern. B Cybern.* **2009**, *39*, 705–722; b) H. G. Han, L. D. Wang, J. F. Qiao, *Neural Networks* **2013**, *43*, 22–32.
[52] R. Lohmann, G. Schneider, P. Wrede, *Biopolymers* **1996**, *38*, 13–29.
[53] Y. Bengio, A. Courville, P. Vincent, *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828.
[54] G. Dahl. *Deep Learning Approaches to Problems in Speech Recognition, Computational Chemistry, and Natural Language Text Processing*. PhD thesis, University of Toronto, **2015**.
[55] a) M. A. Yildirim, K.-I. Goh, M. E. Cusick, A.-L. Barabási, M. Vidal, *Nat. Biotechnol.* **2007**, *25*, 1119–1126; b) G. Schneider, D. Reker, T. Rodrigues, P. Schneider, *Chimia* **2014**, *68*, 648–653.
[56] a) J. Scheiber, B. Chen, M. Milik, S. C. Sukuru, A. Bender, D. Mikhailov, S. Whitebread, J. Hamon, K. Azzaoui, L. Urban, M. Glick, J. W. Davies, J. L. Jenkins, *J. Chem. Inf. Model.* **2009**, *49*, 308–317; b) T. Van Laarhoven, S. B. Nabuurs, E. Marchiori, *Bioinformatics* **2011**, *27*, 3036–3043; c) M. Reutlinger, T. Rodrigues, P. Schneider, G. Schneider, *Angew. Chem. Int. Ed.* **2014**, *53*, 4244–4248.
[57] D. Erhan, P.-J. L'heureux, S. Y. Yue, Y. Bengio, *J. Chem. Inf. Model.* **2006**, *46*, 626–635.
[58] G. Schneider, O. Clément-Chomienne, L. Hilfiger, P. Schneider, S. Kirsch, H.-J. Böhm, W. Neidhart, *Angew. Chem. Int. Ed.* **2000**, *39*, 4130–4133.
[59] a) J. V. de Julian-Ortiz, *Comb. Chem. High Throughput Screen.* **2001**, *4*, 295–310; b) D. P. Visco Jr, R. S. Pophale, M. D. Rintoul, J. L. Faulon, *J. Mol. Graph. Model.* **2002**, *20*, 429–438; c) N. Brown, B. McKay, J. Gasteiger, *J. Comput. Aided Mol. Des.* **2006**, *20*, 333–341; d) W. W. Wong, F. J. Burkowski, *J. Cheminform.* **2009**, *1*, 4.
[60] M. A. Carreira-Perpignan, G. E. Hinton, in: *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics* (AISTATS) **2005**, 59–66.
[61] D. H. Ackley, G. E. Hinton, T. J. Sejnowski, *Cogn. Sci.* **1985**, *9*, 147–169.
[62] D. Buchaca, E. Romero, F. Mazzanti, J. Delgado, *Cornell University Library* **2013,** arXiv:1312.6062.
[63] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, Heidelberg, **2006**.
[64] G. E. Hinton, *Neural Comput.* **2002**, *8*, 1771–1800.
[65] R. Salakhutdinov, G. Hinton, *Neural Comput.* **2012**, *8*, 1967–2006.
[66] L. Deng, *APSIPA Trans. Signal Inf. Process.* **2014**, *3*, e2.
[67] Y. Wang, J. Zeng, *Bioinformatics* **2013**, *29*, i126-i134.
[68] a) B. Fasel, *Acta Neurol. Belg.* **2003**, *103*, 6–12; b) P. Wang, G. Cottrell, *J. Vis.* **2015**, *15*, 1091.
[69] a) J. J. DiCarlo, D. Zoccolan, N. C. Rust, *Neuron* **2012**, *73*, 415–434; b) C. F. Cadieu, H. Hong, D. L. Yamins, N. Pinto, D. Ardila, E. A. Solomon, N. J. Majaj, J. J. DiCarlo, *PLoS Comput. Biol.* **2014**, *10*, e1003963.
[70] a) R. P. Lippmann, *Neural Comput.* **1989**, *1*, 1–38; b) S. Lawrence, C. L. Giles, A. C. Tsoi, A. D. Back, *IEEE Trans. Neural Netw.* **1997**, *8*, 98–113.
[71] A. Krizhevsky, I. Sutskever, G. E. Hinton, in: *Advances in Neural Information Processing Systems* **2012**, 1097–1105.
[72] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, *Nature* **2015**, *518*, 529–533.
[73] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Proc. IEEE* **1998**, 11, 2278–2323.
[74] M. Zeiler, R. Fergus, *Comput. Vision–ECCV 2014* **2014**, 818–833.
[75] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, Y. LeCun, in *2009 IEEE 12th International Conference on Computer Vision*, **2009**, 2146–2153.
[76] J. T. Springenberg, A. Dosovitskiy, T. Brox, M. A. Riedmiller, *Cornell University Library* **2014**, arXiv:1412.6806.
[77] P. P. Ypsilantis, M. Siddique, H. M. Sohn, A. Davies, G. Cook, V. Goh, G. Montana, *PLoS One* **2015**, *10*, e0137036.
[78] D. E. Rumelhart, J. L. McClelland, PDB Research Group, *Parallel Distributed Processing*, MIT Press, Cambridge, **1986**.
[79] A. Mendyk, S. Güres, R. Jachowicz, J. Szlęk, S. Polak, B. Wiśniowska, P. Kleinebudde, *Comput. Math. Methods Med.* **2015**, *2015*, 863874.
[80] D. A. Winkler, *Mol. Biotechnol.* **2004**, *27*, 139–168.
[81] T. I. Netzeva, A. Worth, T. Aldenberg, R. Benigni, M. T. D. Cronin, P. Gramatica, J. S. Jaworska, S. Kahn, G. Klopman, C. A. Marchant, G. Myatt, N. Nikolova-Jeliazkova, G. Y. Patlewicz, R. Perkins, D. Roberts, T. Schultz, D. W. Stanton, J. J. van de Sandt, W. Tong, G. Veith, C. Yang, *Altern. Lab. Anim.* **2005**, *33*, 155–173.
[82] a) K. Roy, *Expert Opin. Drug Discov.* **2007**, *2*, 1567–1577; b) I. Tetko, *Methods Mol. Biol.* **2008**, 458, 185–202; c) A. Tropsha, *Mol. Inf.* **2010**, *29*, 476–488; d) N. Fjodorova, M. Novič, A. Roncaglioni, E. Benfenati, *J. Comput. Aided Mol Des.* **2011**, *25*, 1147–1158.
[83] a) D. T. Manallack, B. G. Tehan, E. Gancia, B. D. Hudson, M. G. Ford, D. J. Livingstone, D. C. Whitley, W. R. Pitt, *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 674–679; b) I. V. Tetko, D. J. Livingstone, A. I. Luik, *J. Chem. Inf. Comput. Sci.* **1995**, *35*, 826–833; c) I. Sushko, S. Novotarskyi, R. Körner, A. K. Pandey, V. V. Kovalishyn, V. V. Prokopenko, I. V. Tetko, *J. Chemometr.* **2010**, *24*, 202–208; d) I. Sushko, S. Novotarskyi, R. Körner, A. K. Pandey, A. Cherkasov, J. Li, P. Gramatica, K. Hansen, T. Schroeter, K.-R. Müller, L. Xi, H. Liu, X. Yao, T. Öberg, F. Hormozdiari, P. Dao, C. Sahinalp, R. Todeschini, P. Polishchuk, A. Artemenko, V. Kuz'min, T. M. Martin, D. M. Young, D. Fourches, E. Muratov, A. Tropsha, I. Baskin, D. Horvath, G. Marcou, C. Müller, A.

Varnek, V. V. Prokopenko, I. V. Tetko, *J. Chem. Inf. Model.* **2010**, *50*, 2094 – 2111.
[84] G. E. Dahl, N. Jaitly, R. Salakhutdinov, *Cornell University Library* **2014**, arXiv:1406.1231.
[85] Y. Qi, M. Oja, J. Weston, W. S. Noble, *PLoS One* **2012**, 3, e32235.
[86] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, V. Pande, *Cornell University Library* **2015**, arXiv:1502.02072.
[87] T. Unterthiner, A. Mayr, G. Klambauer, M. Steijaert, J. K. Wegner, H. Ceulemans, S. Hochreiter, *Deep Learning and Representation Learning Workshop, NIPS* **2014**.
[88] T. Unterthiner, A. Mayr, G. Klambauer, S. Hochreiter, *Cornell University Library* **2015**, arXiv:1503.01445.
[89] P. Di Lena, K. Nagata, P. Baldi, *Bioinformatics* **2012**, *19*, 2449 – 2457.
[90] J. Lyons, A. Dehzangi, R. Heffernan, A. Sharma, K. Paliwal, A. Sattar, Y. Zhou, Y. Yang, *J. Comput. Chem.* **2014**, *28*, 2040 – 2046.
[91] R. Heffernan, K. Paliwal, J. Lyons, A. Dehzangi, A. Sharma, J. Wang, A. Sattar, Y. Yang, Y. Zhou, *Sci. Rep.* **2015**, 11476.
[92] S. P. Nguyen, Y. Shang, D. Xu, *Proc. Int. Jt. Conf. Neural Netw.* **2014**, 2071 – 2078.
[93] M. K. K. Leung, H. Y. Xiong, L. J. Lee, B. J. Frey, *Bioinformatics* **2014**, *12*, i121 – i129.
[94] J. Eickholt, J. Cheng, *Bioinformatics* **2012**, *23*, 3066 – 3072.
[95] J. Eickholt, J. Cheng, *BMC Bioinformatics* **2013**, *14*, S12.
[96] J. Zhou, O. G. Troyanskaya, *Cornell University Library* **2014**, arXiv:1403.1347.
[97] A. Lusci, G. Pollastri, P. Baldi, *J. Chem. Inf. Model.* **2013**, *7*, 1563 – 1575.
[98] Y. Xu, Z. Dai, F. Chen, S. Gao, J. Pei, L. Lai, *J. Chem. Inf. Model.* **2015**, *55*, 2085 – 2093.
[99] P. Baldi, S. Brunak, P. Frasconi, G. Soda, G. Pollastri, *Bioinformatics* **1999**, *11*, 937 – 946.
[100] P. Baldi, G. Pollastri, *J. Mach. Learn. Res.* **2004**, *4,* 575 – 602.
[101] S. K. Sønderby, O. Winther, *Cornell University Library* **2014**, arXiv:1412.7828.
[102] S. K. Sønderby, C. K. Sønderby, H. Nielsen, O. Winther, *Cornell University Library* **2015**, arXiv:1503.01919.
[103] Y. Park, M. Kellis, *Nat. Biotechnol.* **2015**, *8*, 825 – 826.
[104] B. Alipanahi, A. Delong, M. T. Weirauch, B. J. Frey, *Nat. Biotechnol.* **2015**, *8*, 831 – 838.
[105] T. B. Hughes, G. P. Miller, S. J. Swamidass, *ACS Cent. Sci.* **2015**, *4*, 168 – 180.
[106] O. Denas, J. Taylor. *Representation Learning Workshop, ICML* **2013**.
[107] W. Kew, J. B. O. Mitchell, *Mol. Inf.* **2015**, *9*, 634 – 647.
[108] P. Di Lena, K. Nagata, P. F. Baldi, *Advances in Neural Information Processing Systems*, **2012**, 512 – 520.
[109] M. J. Skwark, D. Raimondi, M. Michel, A. Elofsson, *PLoS Comput. Biol.* **2014**, 11, e1003889.
[110] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, R. P. Adams, *Cornell University Library* **2015**, arXiv:1509.09292.

Received: October 25, 2015
Accepted: December 1, 2015
Published online: December 30, 2015