

Topology Preserving Neural Networks for Peptide Design in Drug Discovery

J. Wichard and R. Kühne

FMP Berlin, Molecular Modelling Group
Robert-Roessle-Str. 10, 13125 Berlin, Germany
email: {wichard, kuehne}@fmp-berlin.de

Keywords: Peptide Design, Drug Discovery, QSAR, Cellular Neural Network, TPNN

Abstract We describe a construction method and a training procedure for a topology preserving neural network (TPNN) in order to model the sequence activity relation of peptides. The building blocks of a TPNN are single cells (neurons) which correspond one-to-one to the amino acids of the peptide. The cells have adaptive internal weights and the local interactions between cells govern the dynamics of the system. The TPNN can be trained by gradient descent techniques, which rely on the efficient calculation of the gradient by back-propagation.

1 Introduction

Building artificial neural networks with problem specific topology has a long tradition. In particular for pattern recognition the topology of the retina served as a template for the famous Perceptron (Rosenblatt [1], Minsky and Papert [2]) and later Fukushima's Cognitron [3] and Neocognitron [4]. Further examples are the self-organizing maps of Kohonen [5] and the Cellular Neural Network (CNN) from Chua [6].

The idea of translating the topology of a chemical compound into a so called Molecular Graph Network (MGN) was developed recently [7–9]. In a MGN a compound can be described as graph where each atom is a node and each chemical bond is an edge. In this way a chemical structure is translated into cellular network topology: Each atom becomes a cell, each bond a local interaction between the cells and the weights depend on element and bond types. The MGN works in this case as a discrete time cellular neural network [10]. In the current work we propose a simpler strategy that is dealing with the particular structure of peptides. Instead of building an MGN based on the atoms of the peptide, we use the amino acids as building blocks. This has two reasons: First of all, it reduces the complexity of the problem. MGNs work fine for small molecules but in the case of peptides there are on average 16 Atoms per amino acid which leads to large MGNs even for small peptides. As a consequence learning rates are quite poor and the adaption of weights is slow and time consuming. The second reason is the intrinsic structure of the problem. Our method should optimize the properties of peptides with respect to several pharmacological aspects. We suggest several new peptides to be synthesized and tested during the modelling process described in this paper. In this terms it is straightforward to build models from amino acids rather than from atoms of molecules.

The TPNN should work as a tool for peptide design in drug discovery. Therefore it is necessary, that it catches some fundamental items of the underlining structure activity relation. It is very difficult to run any statistical method that could learn a correlation from a few datapoints in such a high dimensional space (there are for example 10^{20} different possible decapeptides from the 20 natural amino acids). Many attempts have been made to utilize Artificial Neural Networks in order to generate predictive models of quantitative sequence activity relationships between a set of molecular descriptors and activity. An overview could be found in Weekes and Fogel [11] or in Terfloth and Gasteiger [12]

Earlier works from Schneider and Wrede [13] proposed the use of neural networks and computer-based evolutionary search for peptide design. Their work was criticized by Darius and Rojas [14]

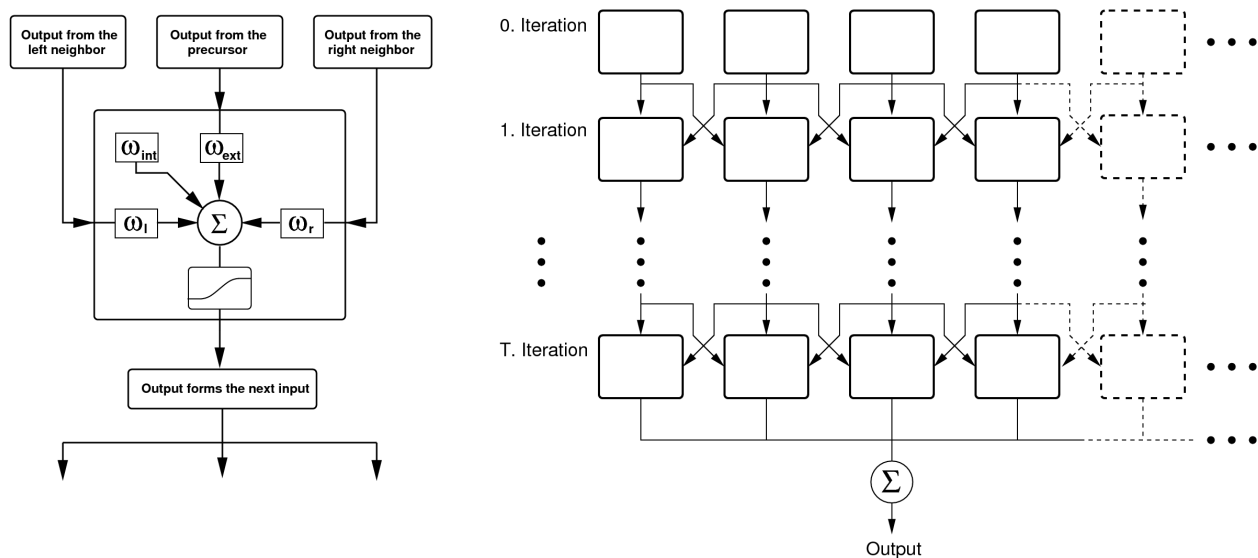


Figure 1: The left part of this figure shows a sketch plot of a single cell which forms the building block of the topology preserving neural network. The internal state of the cell is defined by ω_{int} . The inputs from the left and the right elements of the chain are given by ω_l and ω_r . The input from the precursor cell goes with ω_{ext} . An individual cell has the weight vector $\vec{\omega} = (\omega_{int}, \omega_l, \omega_r, \omega_{ext})$. The sum of all inputs passes the activation function and forms the output for the next iteration. The right part of the figure shows the linkage between the individual cells that mimic the chain topology of the peptide and the propagation of the states in a TPNN over time. After the last iteration, the mean over all cell outputs gives the final result (i.g. the activity of the peptide under investigation).

who questioned the statistical significance of their results in general. But Schneider et al. demonstrated later that their method can generate novel peptides with substantial biological activity in real world experiments [15]. Another group reported the effective application of a tailored genetic algorithm for the de novo construction of peptidic thrombin inhibitors [16].

The novelty of our method lies in the construction of the TPNN architecture that mimics the peptide chain and that the learning takes place in the weights of the cells that correspond to the amino acids of the peptide. With the adapted cells, we are able to assemble new *virtual* peptides and the resulting TPNNs define the fitness function in a genetic algorithm that is used to search for new peptides.

The rest of this paper is structured as follows. In the next Section we define a single cell as the building block of the network and we show how to translate an amino acid chain into a topology preserving neural network. Section 3 describes the training of the TPNN. In section 4 we give an example how a TPNN could be used for peptide design in drug discovery.

2 Peptides and Topology Preserving Neural Networks

Peptides are short polymers of amino acids that are linked with an amide bond. Short in this sense means that the peptide chains are short enough to be made synthetically from the constituent amino acids. Among the variety of amino acids are the 20 proteinogenic amino acids the most important ones. They are found in natural proteins and are coded in the standard genetic code in almost all known life forms. Nevertheless is our approach not restricted to the proteinogenic amino acids and can be extended to any set of "non-standard" amino acids.

A peptide has a representation as a string \vec{S} wherein the symbols denote the amino acids (by convention a three letter code is used to name the most common ones). In general the sequence is reported from the N-terminal end containing free amino group to the C-terminal end containing free carboxyl group. We further assume that our peptides are composed of amino acids from a pool of M different individuals and we call this pool the *alphabet*. In our topology preserving neural network (TPNN) each amino acid from the alphabet is represented as a single cell with four individual weights that are adjusted during the network training. In figure 1 we show a schematic

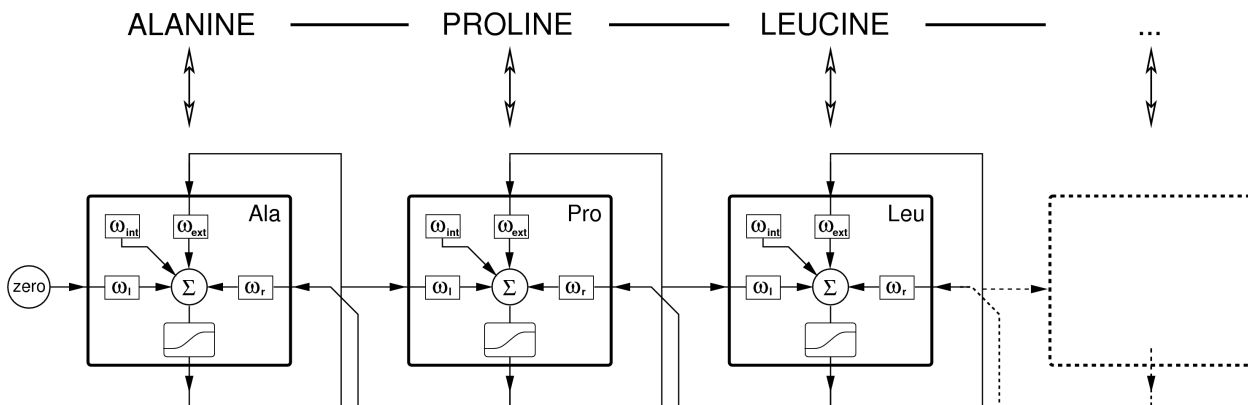


Figure 2: This example shows how to translate the amino acid sequence $\vec{S} = (Ala, Pro, Leu, \dots)$ into a TPNN: The peptide starts with the amino acids *ALANINE-PROLINE-LEUCINE*... and the cells that correspond to these amino acids are connected in the same order, thus preserving the topology of the amino acid chain of the peptide. The open ends of the TPNN get zeros as input.

plot of a single cell and the linkage between the individual cells that mimics the chain topology of the peptide. The internal weight of the cell is defined by ω_{int} . The inputs from the left and the right neighboring cells are connected with the weights ω_l and ω_r . The input from the former iteration of the cell goes with ω_{ext} . The four weights are combined in the weight vector $\vec{\omega} = (\omega_{int}, \omega_l, \omega_r, \omega_{ext})$. The cells are connected to form a chain as shown in figure 1 and the whole TPNN is iterated through time. The state of the i -th TPNN cell y_i^t evolves for iterations $t = 0, \dots, T - 1$ according to:

$$x_i^{t+1} = \omega_{int} + y_{i-1}^t \omega_l + y_{i+1}^t \omega_r + y_i^t \omega_{ext} \quad (1)$$

$$y_i^{t+1} = \sigma(x_i^{t+1}), \quad (2)$$

wherein the activation function $\sigma(x)$ is a hyperbolic tangent with an additional linear term, that does not fully saturate for input values x far from the origin:

$$\sigma(x) = \tanh(x) + \lambda x \quad \text{with } \lambda \ll 1. \quad (3)$$

This feature ensures a non-vanishing derivative of the training error which is important for training algorithms that use gradient information to adjust the weights [17]. The number of iterations T is set to be the average length of the sequences under investigation. Translating a peptide into a TPNN is straightforward: The cells that correspond to the amino acids are connected in the same order as in the peptide. This is shown in figure 2.

The modelling task of the TPNN is to find a mapping between the peptide and an associated biochemical property, in general the activity. The biological activity of chemical compounds is usually measured in assays to establish the level of inhibition of particular signal transduction or metabolic pathways. Chemicals can also be biologically active in terms of toxicity. The most important candidates in drug discovery are those chemical structures that have good inhibitory effects on specific targets and that have low toxicity. Our approach helps to identify these candidates and opens a way to implement an active learning procedure in order to optimize peptides with respect to the pharmacological relevant properties.

The novelty of our approach lies in the topology preservation network structure, wherein we use single cells as building blocks in order to mimic the chain of amino acids forming the peptide. Training the TPNN means adaption of the weights of the cells that represent the amino acids with respect to the desired output.

3 Training TPNN Models

The training of the TPNN is based on a combination of back propagation and stochastic gradient descend. In order to cope with the particular structure of the TPNN, we have to introduce several

improvements that make the training feasible and that were reported in earlier works dealing with the training of CNNs for pattern recognition [18].

In stochastic gradient descent, the true gradient is approximated by the gradient of the loss function which is evaluated on a single training sample. The network weights are then adjusted by an amount proportional to this approximate gradient. A training sample consists of two parts: The first part is the peptide sequence \vec{S} that is composed of the alphabet including the M possible amino acids. The second part is the measured activity a that could be a continuous value or a class label, for example the classes *active*, *weak-active* or *non-active*. Let's assume that we have a collection of N training samples $\{\vec{S}_n, a_n\}_{n=1, \dots, N}$ and the weights $\vec{\omega}^i = (\omega_{int}^i, \omega_l^i, \omega_r^i, \omega_{ext}^i)_{i=1, \dots, M}$ of the individual cells that correspond to the M different amino acids in the alphabet are organized in the weight vector $\Omega = (\vec{\omega}^1, \dots, \vec{\omega}^M)$.

In general the training of a TPNN means minimizing the average loss function with respect to the weight vector Ω . Let $f(\vec{S}_i, \Omega)$ denote the output of the TPNN for a given sequence \vec{S}_i . This output value has to be compared to the training label a_i by means of a *loss function*. The loss function measures the deviation of the TPNN output from the desired value a_i . In optimization usually a quadratic loss function is used, basically due to the simplicity of the resulting derivatives. We propose the use of an ϵ -insensitive loss function λ_ϵ . The advantages of this strategy as well as the functional form of λ_ϵ are described in section 3.1. The training error $E(\Omega, \epsilon)$ is simply the loss averaged over the entire training set

$$E(\Omega, \epsilon) := \sum_{i=1}^N \lambda_\epsilon(a_i - f(\vec{S}_i, \Omega)). \quad (4)$$

As already mentioned, the training error is a function of the networks weights Ω and the level of ϵ . The training essentially is concerned with the minimization of the training error $E(\Omega, \epsilon)$ with respect to \vec{S}_i .

3.1 The ϵ -insensitive Loss Function

To calculate the training error, we use the *ϵ -insensitive squared loss function* defined as

$$\lambda_\epsilon(\xi) := \begin{cases} 0 & : \xi \leq \epsilon \\ (\xi - \epsilon)^2 & : \xi > \epsilon. \end{cases} \quad (5)$$

It is used to calculate the loss contribution of the training samples. The output of the TPNN has zero loss and gradient if it lies inside the ϵ -margin of the desired output. This forces the training algorithm to focus on the training samples that are not properly explained by the current model rather than adjusting the network weights by gradient steps of already correctly learned samples. The use of ϵ -insensitive loss functions in the context of learning problems is described in greater detail by Vapnik [19].

3.2 Weight Decay

Weight decay is a regularization method that penalizes large weights in the network. The *weight decay penalty term* causes the insignificant weights to converge to zero. This results in a model with a minimum number of free parameters, according to Occam's razor also known as the *law of parsimony*. It tells us to prefer the simplest of equally good models. The penalty term is given by

$$P(\Omega) = \gamma \sum_{i=1}^N \frac{\omega_i^2}{1 + \omega_i^2}, \quad (6)$$

where Ω denotes the weight vector of the TPNN and the regularization parameter $\gamma = 0.001$ is small. The penalty term is added to the training error and contributes to the gradient. In figure 3 we show an one dimensional example.

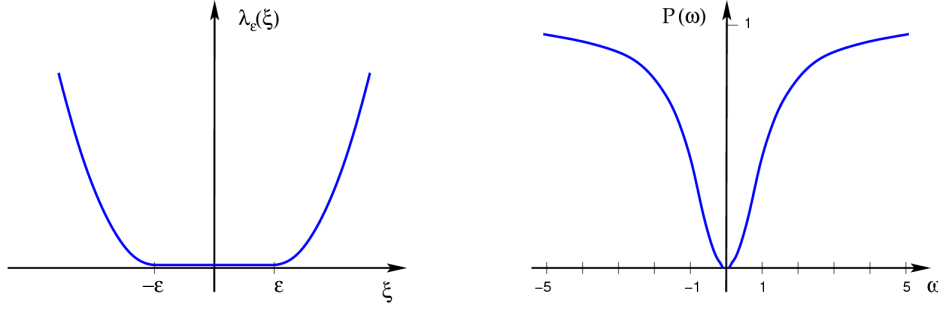


Figure 3: On the left side: The ϵ -insensitive quadratic loss function used for the training of the TPNN. The right side shows the the penalty function for a single network weight $P(\omega) = \frac{\omega^2}{1+\omega^2}$.

3.3 Stochastic Gradient Descent and Backpropagation

Training a learning machine is put to effect by minimizing the training error as defined in Equ. 4 with respect to the network weights Ω . The method of training a TPNN presented in this paper is based on a *stochastic gradient descend* or *on-line learning*, which is a common method for training neural networks [17]. The gradient of the entire training error (Equ. 4) and the penalty term (Equ. 6) is a sum of terms of the form

$$\frac{\partial}{\partial \Omega} \left[\lambda_{\epsilon}(a_i - f(\vec{S}_i, \Omega)) + P(\Omega) \right]. \quad (7)$$

Thus, a classical gradient descent or batch learning algorithm first computes all these terms and then performs a gradient step into the direction given by their sum. The stochastic gradient descent however performs a series of very small consecutive steps, determining each step direction from the gradient of an individual summand only. After each step, the new parameter set Ω is re-inserted into the loss before the next gradient is computed. In other words, we devise an update rule for the parameters of the form

$$\Omega_i = \Omega_{i-1} - \delta \Omega_i, \quad (8)$$

with $i = 1 \dots N$, wherein N is the number of training samples. The update $\delta \Omega_i$ depends on the i -th training sample only and is given by

$$\delta \Omega_i = \mu \frac{\partial}{\partial \Omega} \left[\lambda_{\epsilon}(a_i - f(\vec{S}_i, \Omega_{i-1})) + P(\Omega_{i-1}) \right]. \quad (9)$$

We calculate the update $\delta \Omega_i$ with the standard error back-propagation technique as it is used for the common feed-forward multilayer Perceptron [17]. The parameter μ is controlling the stepsize of the gradient descend. The initial step size is $\mu = 0.01$ and it is decreased after each training epoch with a constant factor until it reaches an lower bound of $\mu = 0.001$. This is necessary to achieve a slow convergence of the weights. Note that in each training step only a few selected values of the entire weight vector Ω are adjusted, namely the ones that correspond to amino acids that appear in the sequence of the training sample.

A sweep through the whole data set is called one *epoch*. Up to 1000 epochs are used to train the TPNN in a typical experiment (20 amino acids in the alphabet, peptides of length 10, 30 training sample from measurements as a starting set).

3.4 Building Classifier Ensembles

A common way to improve the performance of regression or classification models is building ensembles. This is done by training several (10-20) TPNNs on randomly chosen subsets of the training data where the training starts with random weight initializations. To compute the output of the ensemble for one input sequence, the output variables of all TPNNs belonging to the ensemble are averaged.

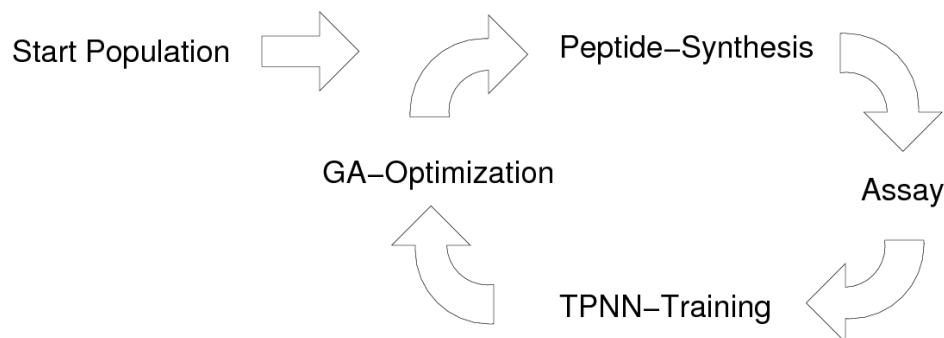


Figure 4: The peptide optimization is done in several cycles. The start population is generated based on experts knowledge concerning the target, i.g. natural ligands, known ligands or compounds that bind to similar targets. The trained TPNN is used as fitness function in a GA.

4 TPNN Models and Peptide Design

We developed the TPNN in order to model the sequence activity relation of peptides. If we assume that the TPNN has learned some important properties of peptides with respect to the biochemical assay under investigation then we could use this TPNN in order to optimize the peptides in terms of biological activity. The general way is to build a proper model that is further used as a fitness measure in an optimization algorithm. In the case of peptide design, any standard evolutionary algorithm [20–23] could be used in order search for new peptides that are selected with the previously learned TPNN.

Similar attempts have been reported by Schneider and Wrede [13, 24] and Schneider et al. [15] termed *simulated molecular evolution*. In their approach an artificial neural network is used to model the locally encoded amino acid sequence features of the peptides. In contrast to our approach, the neural networks are used in a conventional way as nonlinear function approximation, while we explicitly use the chain topology of the peptide to construct our TPNN model.

A typical cycle for peptide design works as follows (see figure 4). A start population of peptides is selected, based on prior knowledge of the target under investigation. If there are no natural ligands that bind to the target, one could look for ligands that bind to similar targets and build a small peptide library based on that. The sequence strings of the peptides together with the measurements from the biological assay deliver the data for TPNN training. The adopted cells work as building blocks for the assembling of new peptides and the resulting TPNN is the fitness function for the genetic algorithm that is generating new suggestions for peptide synthesis. This process is repeated several times.

5 Conclusion and Outlook

We developed a new topology preserving network architecture - the TPNN - that mimics the chain structure of peptides. Training a TPNN means to adopt the weights of the cells that correspond to the amino acids of the peptide. The trained cells work as building blocks for the assembling of new (virtual) peptides and the resulting TPNN works as fitness function in GA based search of the sequence space.

Thus the concept of TPNN is the core of a novel peptide optimization process for drug discovery. This is work in progress. We developed and tested our method with some toy problems and data from earlier peptide screening campaigns. The first real experiments for a specific drug target started recently.

Acknowledgments

The authors would like to thank the members of the Molecular Modelling Group at FMP Berlin.

References

- [1] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.
- [2] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969/1988, expanded edition.
- [3] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, no. 3-4, pp. 121–36, Nov 1975.
- [4] —, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, April 1980.
- [5] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, January 1982.
- [6] L. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. on Circuits and Systems*, vol. 35, pp. 1257–1272, 1988.
- [7] C. Merkwirth and T. Lengauer, "Automatic generation of complementary descriptors with molecular graph networks," *Journal of Chemical Information and Modeling*, vol. 45, no. 5, pp. 1159–1168, 2005.
- [8] M. Ogorzałek, C. Merkwirth, and J. Wichard, "Pattern recognition using finite-iteration cellular systems," in *Proceedings of the 9th International Workshop on Cellular Neural Networks and Their Applications*, 2005, pp. 57 – 60.
- [9] C. Merkwirth and M. Ogorzałek, "Applying CNN to cheminformatics," in *Proceedings of the ISCAS*, 2007, pp. 2918 – 2921.
- [10] H. Harrer and J. Nossek, "Discrete-time cellular neural networks," *Int. J. Circuit Theory and Applications*, vol. 20, pp. 453–467, 1992.
- [11] D. Weekes and G. Fogel, "Evolutionary optimization, backpropagation, and data preparation issues in QSAR modeling of HIV inhibition by HEPT derivatives," *Biosystems*, vol. 72, no. 1-2, pp. 149–158, November 2003.
- [12] L. Terfloth and J. Gasteiger, "Neural networks and genetic algorithms in drug design," *Drug Discovery Today*, vol. 6, no. Supplement 2, pp. 102–108, August 2001.
- [13] G. Schneider and P. Wrede, "The rational design of amino acid sequences by artificial neural networks and simulated molecular evolution: De novo design of an idealized leader peptidase cleavage site," *Biophysical Journal*, vol. 66, pp. 335–344, February 1994.
- [14] F. Darius and R. Rojas, "Simulated molecular evolution or computer generated artifacts?" *Biophysical Journal*, vol. 67, pp. 2120–2122, November 1994.
- [15] G. Schneider, W. Schrodler, G. Wallukat, J. Müller, E. Nissen, W. Ronspeck, P. Wrede, and R. Kunze, "Peptide design by artificial neural networks and computer-based evolutionary search," *Proceedings of the National Academy of Sciences*, vol. 95, no. 21, pp. 12 179–12 184, 1998.

- [16] S. Kamphausen, N. Hölftge, F. Wirsching, C. Morys-Wortmann, D. Riestler, R. Goetz, M. Thürk, and A. Schwienhorst, “Genetic algorithm for the design of molecules with desired properties,” *Journal of Computer-Aided Molecular Design*, vol. 16, no. 8-9, pp. 551–567, 2002.
- [17] Y. LeCun, L. Bottou, G. Orr, and K. Müller, “Efficient BackProp,” in *Neural Networks: Tricks of the trade*, ser. Lecture Notes in Computer Science, G. Orr and K. Müller, Eds. Springer Verlag, 1998, vol. 1524, pp. 9–50.
- [18] J. Wichard, M. Ogorzałek, C. Merkwirth, and J. Bröcker, “Finite iteration DT-CNN with stationary templates,” in *Proceedings of the 8th IEEE International Workshop on Cellular Neural Networks and their Applications*, Budapest, Hungary, 2004, pp. 459–464.
- [19] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer Verlag, 1999.
- [20] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. New York, USA: John Wiley, 1966.
- [21] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann Verlag, 1973, in German.
- [22] J. H. Holland, *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1975.
- [23] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.
- [24] G. Schneider and P. Wrede, “Artificial neural networks for computer-based molecular design,” *Progress in Biophysics and Molecular Biology*, vol. 70, no. 3, pp. 175–222, November 1998.