Construction of Heuristic for Protein Structure Optimization Using Deep Reinforcement Learning

Rok Hribar[1,2](✉), Jurij Šilc[1], and Gregor Papa[1,2]

[1] Jožef Stefan Institute, Ljubljana, Slovenia
{rok.hribar,jurij.silc,gregor.papa}@ijs.si
[2] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

Abstract. Deep neural networks are constructed that are able to partially solve a protein structure optimization problem. The networks are trained using reinforcement learning approach so that free energy of predicted protein structure is minimized. Free energy of a protein structure is calculated using generalized three-dimensional AB off-lattice protein model. This methodology can be applied to other classes of optimization problems and represents a step toward automatic heuristic construction using deep neural networks. Trained networks can be used to construct better initial populations for optimization. It is shown that differential evolution applied to protein structure optimization problem converges to better solutions when initial population is constructed in this way.

Keywords: Protein folding · Heuristic · Deep learning
Differential evolution

1 Introduction

Prediction of protein structure from the sequence of its residues is a hard optimization problem. All proteins are endowed with a primary structure consisting of the chain of amino acids. Folding of this chain results into so-called 3D protein structure. The biological functional role of the protein is strictly dependent on the protein 3D structure. Knowledge of a proteins structure provides insight into how it can interact with other proteins, DNA/RNA, and small molecules. It are these interactions which define the proteins function and biological role in an organism. Thus, protein structure and structural feature prediction is a fundamental area of computational biology. Its importance is exacerbated by large amounts of sequence data coming from genomics projects and the fact that experimentally determining protein structures remains expensive and time consuming [1].

Over the last decades a lot of effort have been invested in reducing the computational cost of calculating the 3D structures of proteins. One way to decrease the computational cost is the introduction of approximate models for the calculation of protein's free energy which is minimal for appropriate 3D structure.

Because the computation of free energy is less costly, optimization that finds the right structure is also less costly to preform. Examples of such approximate models include models using a cubic lattice [2] and AB type models [3]. Another way to speed up the optimization process is to make optimization more efficient, so that less free energy evaluations are needed. This resulted in development of heuristics that are tailored specifically for this optimization problem. Since protein folding process is in nature guided only by the physical laws, optimization methods were devised that include principles from statistical physics. Heuristics from this category include annealing contour Monte Carlo [4] and conformational space annealing [5]. Another approach to developing a specialized optimization algorithm is to modify known metaheuristics such as artificial bee colony [6] or evolutionary algorithm [7].

A different approach to prediction of protein 3D structure is the use of machine learning. Here the prediction of 3D structure is based only on features directly calculated from the sequence of amino acids. There is no optimization performed during prediction. The structure is calculated simply by applying the model. Optimization is used only during the model training, when appropriate model is searched for. Currently, deep neural networks (DNNs) are the most widely used models for this problem. Properly trained DNNs are very successful at predicting protein's secondary structure ($\approx$80% accuracy) [8] and its disordered regions ($\approx$90% accuracy) [9]. However, full 3D structure prediction is much less accurate ($\approx$20% accuracy) [10]. DNN models are usually trained using supervised learning where experimentally acquired 3D structures of proteins are used as training examples. Advantage of this approach is that protein's free energy does not need to be calculated. But on the other hand, by using only experimental data one is limited to possibly insufficient amount of training examples to properly train DNN.

In this paper a different approach to DNN training is presented and used in which explicit training examples are not needed. Instead, the free energy of a protein is used to provide information about the quality of predicted solutions. This is possible because this problem can be interpreted as an optimization problem or a prediction problem. This allows the combination of both views to generate a new method for addressing the protein structure problem. In this regard such methodology can be applied to any optimization problem to generate DNNs able to predict a solution of an optimization problem. In other words, given a class of optimization problems one can construct a DNN that represents extremely fast heuristic specially designed for this class of optimization problems. This is a step toward automatic heuristic generation.

2 Deep Neural Network as an Optimization Algorithm

Optimization problems are often solved using approximate algorithms (heuristics) that are tailored for a specific class of problems. For example, there are specific heuristics that work well for vehicle routing [11], production scheduling [12], protein folding [13] and so on. Heuristics are especially useful if similar problems need to be solved over and over again. In such cases it is sensible to develop specialized optimization procedures which are optimized for that specific class of problems.

In this section a methodology is presented where DNNs are trained in a way that they are able to approximately solve an optimization problem that belongs to a given class of problems. Such class of optimization problems can be represented with a fitness function f with two inputs.

$$\begin{aligned} f &: \mathcal{S}, \mathcal{X} \to \mathbb{R} \\ f_s(x) &= \min. \end{aligned} \tag{1}$$

Set $\mathcal{S}$ holds all possible optimization problems in the class, while set $\mathcal{X}$ holds all possible candidate solutions for that class of problems. For example, in case f represents a class of production scheduling problems, s encodes the orders that need to be fulfilled and x encodes the production schedule.

Given function f, it is possible to define a function g that takes a problem specification s as an input and returns the position x_s^{optimal} where function f_s has a global minimum.

$$\begin{gathered} g : \mathcal{S} \to \mathcal{X} \\ g(s) = \arg\min_{x \in \mathcal{X}} f_s(x) = x_s^{\text{optimal}} \end{gathered} \tag{2}$$

In other words, $g(s)$ is a solution of optimization problem $f_s(x) = \min$. Calculation of function g is in general intractable. But it might be possible to find a model that approximates g to some degree. One aim of this paper is to find out whether a trained DNN is able to approximate g. It is important to note that the input and output of DNN are traditionally floating point numbers. Therefore, s and x should be encoded as vectors of floating point numbers. Even for discrete s and x it is usually possible to find such an encoding.

It is known that a neural network can approximate arbitrary function to an arbitrary precision [14]. So g can be approximated well using DNN, however it is unknown how large such a network should be and whether it is possible to find it using known training techniques. If DNN could be trained to approximate g, such DNN can preform partial optimization extremely quickly. While DNN training is known to be resource intensive, prediction is usually not.

Training DNN to approximate g is also an optimization problem, however optimization landscape of DNN training is not similar to f_s landscape. DNN parameters encode a strategy for predicting x_s^{optimal} from s, so optimization is not performed on a single problem encoded by s, but for all possible s at once. Also DNN optimization landscape has particular properties, like the fact that saddle points are exponentially more common compared to local minima [15]. Therefore, a suitable optimization method that takes those specific properties into account should be used for training them. Currently, stochastic gradient descent (SGD) is the prevalent and very successful approach to DNN training [16].

2.1 Supervised Learning Approach

DNN can be trained to approximate g using supervised learning. Let $\hat{g}(s)$ be the output of DNN when s is its input. In supervised learning pairs $(s_i, x_{s_i}^{\text{optimal}})$ are provided and DNN error is minimized using SGD so that

$$J = \sum_i \left\| \hat{g}(s_i) - x_{s_i}^{\text{optimal}} \right\| = \min. \tag{3}$$

Since x_s^{optimal} is in general unknown, its best approximation has to be used which results to nonideal DNN model. So $x_{s_i}^{\text{optimal}}$ need to acquired using external optimization algorithms. In order to prevent DNN overfitting it is necessary to provide a large amount of training examples, i.e. much more than the number of DNN parameters. Therefore, such approach is extremely resource intensive.

2.2 Reinforcement Learning Approach

Another approach to DNN training is reinforcement learning. In this case functions f_s are used to calculate the error of DNN. DNN is trained so that

$$E = \sum_i f_{s_i}(\hat{g}(s_i)) = \min. \tag{4}$$

In reinforcement learning terminology, E can be understood as a penalty that needs to be minimized. In this case $x_{s_i}^{\text{optimal}}$ are not needed and so no external optimization is required. The downside is that SGD can not be applied as simply as with supervised learning. Error function J from Eq. (3) can be easily differentiated with respect to DNN parameters using backpropagation. But penalty E from Eq. (4) also includes application of f_s. This makes the calculation of the gradient difficult and different methods have been introduced by deep reinforcement learning community to mend this problem.

In this paper an adapted version of deterministic policy gradient method [17] is used. This method uses a differentiable model called a critic that approximates function f from Eq. (4). The derivative of E can then be approximately calculated using the derivative of the critic by applying the chain rule. Our adaptation of this method is to not model f with a critic but instead use f directly. In order to calculate derivative of E in this scope, the derivative $\nabla_x f_s(x)$ is required.

Fortunately, derivation of f_s can also be preformed using backpropagation principles, i.e. applying chain rule coupled with dynamic programming. There are good libraries that can preform such automatic differentiation, for example `theano`, `TensorFlow` and `CNTK`. In this paper `theano` was used to write expressions for the calculation of E. These expressions are then transformed to a computational graph for calculation of E which can be used to build computational graph for gradient calculation ∇E using the chain rule. In this respect procedure is returned for analytical gradient calculation ∇E without any assistance from the user. Computational graphs for E and ∇E calculation can be compiled to `C++`, `CUDA` or `OpenCL` which brings multi processor support and can easily be accelerated on GPGPU or even FPGA [18]. Therefore, use of `theano` is beneficial even if just calculation of E is needed.

Therefore, by using `theano` DNN can be trained using SGD so that E from Eq. (4) is minimized. Great advantage of this approach is an unlimited amount of training examples s_i which can be drawn from desired distribution over s_i. This generation of training examples is extremely cheap compared to supervised learning approach where training examples need to be generated by optimization algorithm or acquired by experimental measurement.

3 Generalized Three-Dimensional AB Off-Lattice Protein Model

AB off-lattice model has been widely used to describe the protein secondary structure folding process for decades [3]. The off-lattice protein model was initially developed to consider 2D folding problems and was extended to deal with 3D scenarios where additional torsional energy contributions of each bond are taken into account [19]. According to the AB off-lattice model, the main driving forces that contribute to protein structure formulation are the hydrophilic and hydrophobic interactions.

The protein chain is modeled as a vector s where each component s_i specifies the hydrophilicity of amino acid at the site i. The distance of two neighboring amino acids is set to one ($d_{i,i+1} = 1$). Under this model free energy G is calculated as

$$G(u,d) = \frac{1}{4}\sum_{i=1}^{n-2}(1 - u_i \cdot u_{i+1}) + 4\sum_{i=1}^{n-2}\sum_{j=i+2}^{n}\left(d_{ij}^{-12} - C(s_i,s_j)d_{ij}^{-6}\right), \qquad (5)$$

where u_i is a vector from amino acid on site i to amino acid on site $i+1$ and d_{ij} is a distance between amino acids on site i and j (see Fig. 1). The interaction between two amino acids is specified by a function

$$C(s_i,s_j) = \frac{1}{8}(1 + s_i + s_j + 5s_is_j). \qquad (6)$$

Structure of a protein of length n can be encoded using angles θ_i and φ_i that tell how vectors u_i are oriented in space (see Fig. 2). Therefore, a protein structure of length n is fully determined by

$$x = (\theta_2, \ldots, \theta_{n-1}, \varphi_3, \ldots, \varphi_{n-1}). \qquad (7)$$

Use of this encoding reduces the dimensionality of search space and allows us to automatically fulfill the constraint $\|u_i\| = 1$. The values u_i and d_{ij} that are needed for free energy calculation can be calculated from x in the following way

$$u_i = (\cos\theta_i \sin\varphi_i, \sin\theta_i \sin\varphi_i, \cos\varphi_i) \qquad (8)$$

$$r_{i+1} = r_i + u_i \qquad (9)$$

$$d_{ij} = \|r_i - r_j\|. \qquad (10)$$

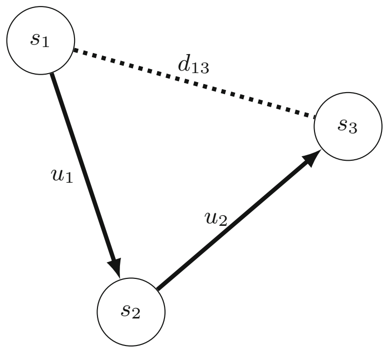

Fig. 1. Visualization of direction vectors u_i and distances d_{ij} on a protein with three amino acids.

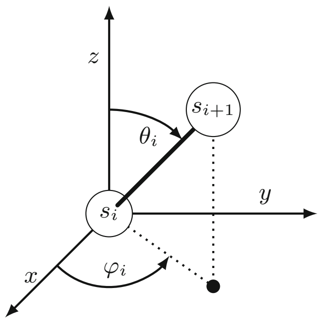

Fig. 2. Visualization of angles θ_i and φ_i from Eq. (7) that determine the direction of vectors u_i.

The first two amino acids in a sequence are fixed to specific coordinates and the third one is constrained to xy-plane.

$$r_1 = (0, 0, 0) \tag{11}$$

$$r_2 = (1, 0, 0) \tag{12}$$

$$\varphi_2 = \frac{\pi}{2} \tag{13}$$

By this choice, rotational symmetry of the model is eliminated.

In protein structure optimization problem we want to find a structure of a protein that minimizes the free energy G. Therefore, given a protein defined with s, we want to find x that determines the structure of that protein. So the problem class is defined by a function

$$f_s(x) = G(u(x), d(x)) = \min. \tag{14}$$

Traditionally $s_i = \pm 1$, where hydrophobic amino acid has $s_i = -1$ and hydrophilic $s_i = 1$. In this regard quantity s_i tells how hydrophilic an amino acid is. However, this paper uses a generalization of this model where $s_i \in \mathbb{R}$. One reason for this choice is the fact that amino acids in nature are not hydrophilic to the same degree [20]. Some may attract or repel water more than others. Also hydrophilicity changes with temperature [21] which allows one to use this generalized model to study how protein structure changes with temperature. Use of generalized model is also beneficial with regard to DNN training because this brings a richer set of training examples and makes the training landscape smoother.

4 Experiments

In this section DNN training procedure using reinforcement learning is presented and how solutions predicted by DNNs were used as initial population of differential evolution (DE) algorithm. A variant of SGD called Adam [22] was used and gradient of E was used to guide the training. The calculation of ∇E was calculated using `theano` library.

$$E = \sum_{i=1}^{B} f_{s_i}(\hat{g}(s_i)) \tag{15}$$

In each step of Adam algorithm a batch of proteins s_i of random length was randomly selected. Distribution over length was uniform and over hydrophilicity a mixture of two Gaussians with mean at ± 1 and standard deviation 0.15. Based on the selected s_i calculation of E and ∇E was done by `theano`. The number of sampled proteins for E and ∇E calculation is called a batch size B. The training is more stochastic if B is low and becomes more deterministic if B is high. By experimenting with different batch sizes a good balance between speed and accuracy was found at $B = 512$.

Stated more informally, in each step, DNN tries to solve 512 random protein optimization problems and gets updated in direction that would improve its solving capabilities for those 512 proteins. Because DNN gets a different batch of random proteins in each step, it converges to a state that is able to solve all protein optimization problems equally well. Picking training batches randomly also ensures that DNN can not overfit since duplicates in the training data are extremely unlikely. Therefore, the training error of DNN is not a biased estimate of its accuracy and validation set is not needed.

A DNN structure was chosen that can take proteins with up to $n = 100$ amino acids. To allow prediction on smaller proteins zero padding was used. Example of small scale DNN is shown in Fig. 3. DNNs with different number of hidden layers was trained in order to quantify how DNN depth influences the accuracy. In all cases the width of hidden layers was chosen to be $2n = 200$. Rectified linear units were used as activation functions on hidden layers and tanh on the output layer to ensure that $\theta_i, \varphi_i \in [-\pi, \pi]$.

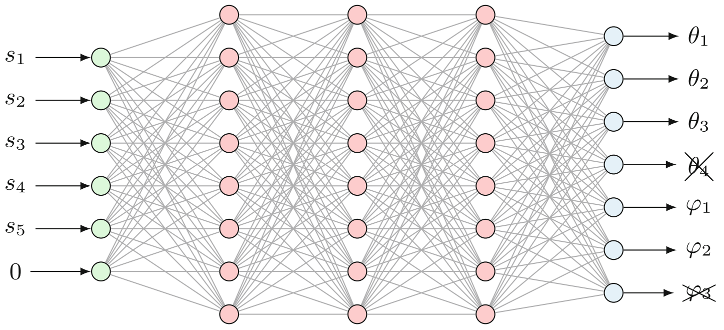

Fig. 3. Small scale example of how DNN receives the protein specification and how its output is interpreted. If a protein is shorter than DNN input layer, zero is placed on sites where there are no amino acids. In this case some angles from the output are discarded (crossed out outputs).

Before training, initial DNN weights need to be set. Initial weights were generated randomly using Glorot initialization [23]. However, if the magnitude of initial weights is to large, it can happen that initial DNN predicts very densely packed structures which causes a very large gradient due to d_{ij}^{-12} repulsive term in free energy. This causes an unstable gradient descent. Therefore, a prefactor w_i was added to initialization and DNN models with $w_1 = 0.1$ and $w_2 = 1.0$ was trained.

To avoid unstable gradient descent the predictions of initial DNN should be unfolded protein structures. This, however, produces another problem. The first summand in Eq. (5) forces proteins to be unfolded which means that unfolded structure is a local minimum that is common to all proteins. To avoid getting stuck in this common local minimum the first summand in Eq. (5) was simply not included in the calculation of free energy at the beginning of training. When DNN predicted structures began to fold, the previously ignored summand was gradually added during training. In the last stage of training the full version of free energy was used.

During DNN training it can happen that for some s_i in the batch DNN predicts a structure where two amino acids are very close to each other. A repulsive interaction causes the gradient ∇E to be very large for the entire batch. In the next step of gradient descent the DNN is thrown away from a possibly good region. Such events might be rare, but can severely disturb the progress of training. To mitigate the effect of such events, gradient norm clipping can be used. In other words, if the gradient length exceeds a given threshold, the gradient is clipped so that its length is equal to the threshold.

Solutions predicted by DNNs might be a good initial population for optimization. To test whether this is true protein structure optimization using DE was implemented. DE was shown to be the best known optimization method for this problem [7]. Specifications of implemented DE algorithm was taken from [7], but without parameter control. DE type was `best/1/bin`, population size was 100, mutation with dithering was employed with mutation constant taken between 0.1 and 1 and recombination constant was 0.9. DE was run 30 times for three proteins found in nature (1CB3, 1CRN and 2EWH) with random initial population and with initial population where 50% of candidates were predicted by DNNs.

5 Results

To measure how good a candidate solution is, we use free energy G of the protein. In case of comparing solutions gotten by DE, this is a sound measure from the point of view of statistical physics. That is, the protein is most likely to be in states with low free energy. In ideal case one could check if the solution is equivalent to the native structure, but because global minima of this protein model are unknown this is not possible. To evaluate the performance of DNN, training error is used. It is equivalent to validation error and defined as a mean of free energy values predicted by DNN for a batch of random proteins. The variance of this error measure is very low because the batch size $B = 512$ is large.

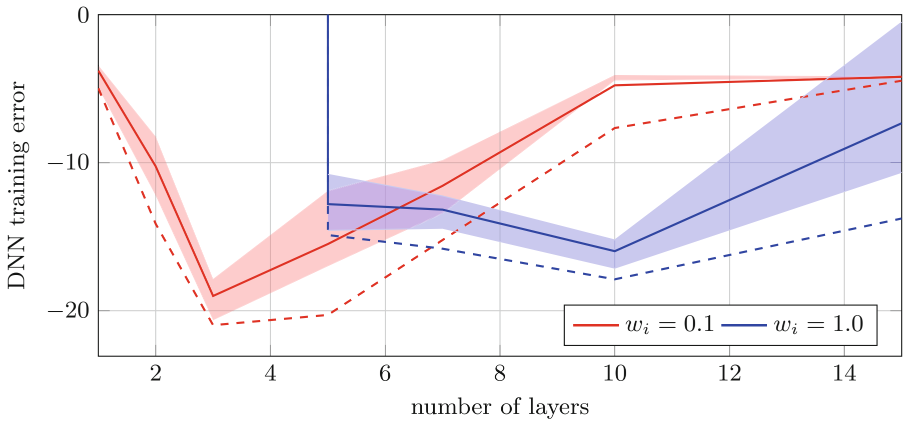

Fig. 4. Training error of DNNs with respect to the number of hidden layers for two different magnitudes of initial DNN weights. Full line is the mean error of models, shaded area shows the range of error for central 66% of the models and the dashed line is the error of the best model.

Table 1. Free energy calculated for three proteins found in nature by using structures predicted by DNN and by DE.

Protein	Length	Best DNN	Mean DE	Best DE
1CB3	13	−4.6235	−2.1513	−6.7700
1CRN	46	−42.765	−64.948	−79.906
2EWH	98	−65.163	−148.32	−170.47

The training error of DNNs depends of the number of layers. This dependence is usually monotonically decreasing [16], but for this problem this is not the case (see Fig. 4). This can be attributed to the sensitivity of training to selection of initial DNN weights. DNNs were initialized using random matrices. Therefore, the magnitude of DNN output is exponentially dependent of the number of layers. Since initial weights have small components ($\ll$1), this means that initial DNNs with small number of layers predict very folded structures, while initial DNNs with high number of layers predict practically straight structures. Figure 4 shows that initial DNN predictions should not be very folded nor very straight. Best models are somewhere in between.

The most accurate DNN models have three layers. Given that DNNs are able to partially solve an optimization problem this is a surprisingly shallow DNN architecture. Free energy of structures predicted by DNNs and by DE is shown in Table 1. It was found that gradient norm clipping is very beneficial for DNN training. Figure 5 shows the progress of DNN training with and without the use of gradient norm clipping. Occurrence of very high gradients is rare, however they substantially alter the progress of DNN training.

Using structures predicted by DNN in initial population of DE was found to be beneficial. When using predicted initial population, DE converges to lower

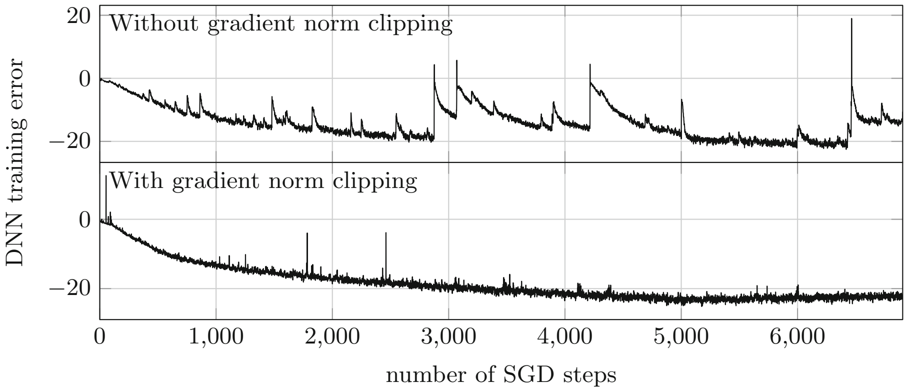

Fig. 5. DNN error during training via SGD. The upper plot shows the progress of usual SGD procedure, while the lower plot shows SGD progress when gradient norm has a predefined upper bound by using gradient norm clipping.

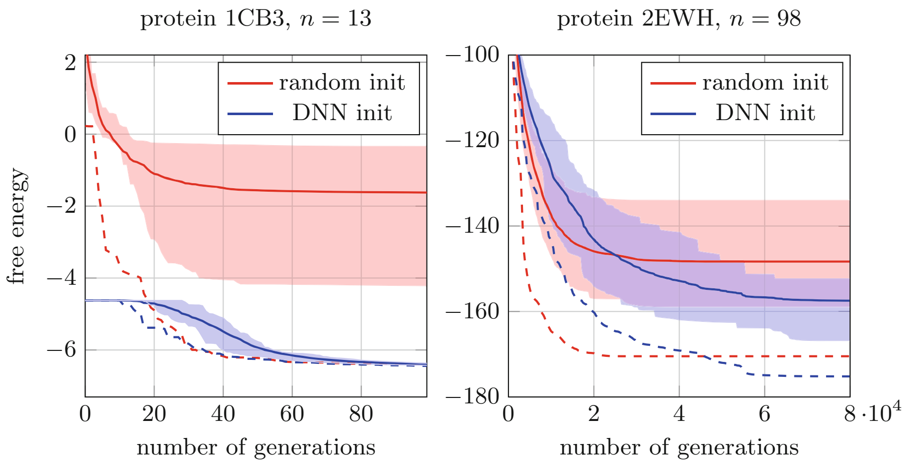

Fig. 6. Progress of 30 runs of DE by using a random initial population or initial population partially populated with solutions predicted by DNNs. Full line is the mean over all runs, shaded area shows the range of central 66% of runs and the dashed line shows the best run.

values of free energy (see Fig. 6). But the convergence is slightly less rapid for predicted population which could indicate that the population is actually more diverse. On the other hand, DE progresses are less dispersed among runs which means that less runs are needed to find a satisfactory solution. In case of 1CB3 protein the predicted structures are in fact so good that all DE runs converge to the best known solution in just 100 generations. For larger proteins the predicted solution are not as good, however the DE performs considerably better when using the predicted population.

6 Conclusion and Future Work

In this paper it is shown that deep neural networks can be trained to partially solve optimization problems belonging to a given class. The networks can be successfully trained using reinforcement learning method by knowing only the fitness function of the class of optimization problems. This is shown for the class of protein structure optimization problems. The predicted solutions were found to be good initial points for further optimization. Such trained networks can be used to acquire moderately good solutions of optimization problem when solution is needed very quickly. Therefore, the method is suitable for optimization problems that need to be solved repeatedly and is a step towards automatic heuristic construction.

For future work it should be possible to extend the method to combinatorial optimization problem where unified methodology should be further developed. The procedure of finding the best architecture of deep neural network could be more automated so that depth and width of the network is automatically found. The same goes for the training specification. Another opportunity for future work is to combine supervised learning approach with reinforcement learning approach so that the training is guided by both approaches simultaneously.

Acknowledgments. The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0098 and PR-07606) and from the European Union's Horizon 2020 research and innovation program under grant agreement No. 692286.

References

1. Cheng, J., Randall, A.Z., Sweredoski, M.J., Baldi, P.: SCRATCH: a protein structure and structural feature prediction server. Nucleic Acids Res. **33**(Suppl. 2), W72–W76 (2005)
2. Bošković, B., Brest, J.: Genetic algorithm with advanced mechanisms applied to the protein structure prediction in a hydrophobic-polar model and cubic lattice. Appl. Soft Comput. **45**, 61–70 (2016)
3. Stillinger, F.H., Head-Gordon, T., Hirshfeld, C.L.: Toy model for protein folding. Phys. Rev. E **48**(2), 1469 (1993)
4. Liang, F.: Annealing contour Monte Carlo algorithm for structure optimization in an off-lattice protein model. J. Chem. Phys. **120**(14), 6756–6763 (2004)
5. Kim, S.Y., Lee, S.B., Lee, J.: Structure optimization by conformational space annealing in an off-lattice protein model. Phys. Rev. E **72**(1), 011916 (2005)
6. Li, B., Lin, M., Liu, Q., Li, Y., Zhou, C.: Protein folding optimization based on 3D off-lattice model via an improved artificial bee colony algorithm. J. Mol. Model. **21**(10), 261 (2015)
7. Bošković, B., Brest, J.: Differential evolution for protein folding optimization based on a three-dimensional AB off-lattice model. J. Mol. Model. **22**(10), 252 (2016)
8. Pollastri, G., Przybylski, D., Rost, B., Baldi, P.: Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. Proteins: Struct. Funct. Bioinf. **47**(2), 228–235 (2002)

9. Cheng, J., Sweredoski, M.J., Baldi, P.: Accurate prediction of protein disordered regions by mining protein structure data. Data Min. Knowl. Disc. **11**(3), 213–222 (2005)
10. Di Lena, P., Nagata, K., Baldi, P.: Deep architectures for protein contact map prediction. Bioinformatics **28**(19), 2449–2457 (2012)
11. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur. J. Oper. Res. **231**(1), 1–21 (2013)
12. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: a review. IEEE Trans. Evol. Comput. **20**(1), 110–124 (2016)
13. Perez, A., MacCallum, J., Dill, K.A.: Using physics and heuristics in protein structure prediction. Biophys. J. **108**(2), 210a (2015)
14. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**(2), 251–257 (1991)
15. Dauphin, Y.N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., Bengio, Y.: Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: Advances in Neural Information Processing Systems, pp. 2933–2941 (2014)
16. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
17. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: ICML (2014)
18. Ling, A.C., Aydonat, U., O'Connell, S., Capalija, D., Chiu, G.R.: Creating high performance applications with Intel's FPGA OpenCL SDK. In: Proceedings of the 5th International Workshop on OpenCL. ACM (2017). Article No. 11
19. Bachmann, M., Arkın, H., Janke, W.: Multicanonical study of coarse-grained off-lattice models for folding heteropolymers. Phys. Rev. E **71**(3), 031906 (2005)
20. Parker, J., Guo, D., Hodges, R.: New hydrophilicity scale derived from high-performance liquid chromatography peptide retention data: correlation of predicted surface residues with antigenicity and X-ray-derived accessible sites. Biochemistry **25**(19), 5425–5432 (1986)
21. Wolfenden, R., Lewis, C.A., Yuan, Y., Carter, C.W.: Temperature dependence of amino acid hydrophobicities. Proc. Nat. Acad. Sci. **112**(24), 7484–7488 (2015)
22. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256 (2010)