Data Min Knowl Disc (2018) 32:218–230
https://doi.org/10.1007/s10618-017-0531-0

TECHNICAL NOTE

The inner and outer approaches to the design of recursive neural architectures

Pierre Baldi[1]

Received: 31 May 2016 / Accepted: 10 July 2017 / Published online: 26 July 2017

Abstract Feedforward neural network architectures work well for numerical data of fixed size, such as images. For variable size, structured data, such as sequences, d dimensional grids, trees, and other graphs, recursive architectures must be used. We distinguish two general approaches for the design of recursive architectures in deep learning, the inner and the outer approach. The inner approach uses neural networks recursively inside the data graphs, essentially to "crawl" the edges of the graphs in order to compute the final output. It requires acyclic orientations of the underlying graphs. The outer approach uses neural networks recursively outside the data graphs and regardless of their orientation. These neural networks operate orthogonally to the data graph and progressively "fold" or aggregate the input structure to produce the final output. The distinction is illustrated using several examples from the fields of natural language processing, chemoinformatics, and bioinformatics, and applied to the problem of learning from variable-size sets.

Keywords Deep learning · Recurrent neural networks · Recursive neural networks · Convolutional neural networks · Structured input

1 Introduction

Many problems in machine learning involve data items represented by vectors or tensors of fixed size. This is the case, for instance, in computer vision with images

Responsible editor: Johannes Fürnkranz.

✉ Pierre Baldi
pfbaldi@uci.edu

[1] Department of Computer Science, University of California, Irvine, CA 92617, USA

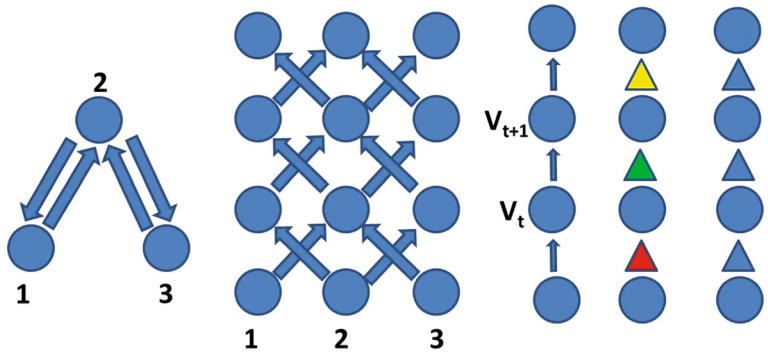

Fig. 1 From *left* to *right*: **a** recurrent networks with 3 neurons; **b** the same network unfolded in time; **c** a sequence of vectors V_t such that $V_{t+1} = F_{t+1}(V_t)$; **d** the same sequence reparameterized using neural networks, shown as *triangles* of different *colors* with $V_{t+1} = NN_{t+1}(V_t)$ (note in turn that each network can be different and, for instance, include many hidden layers); **e** corresponding recursive network obtained by using identical networks, or weight sharing, with $V_{t+1} = NN(V_t)$ (Color figure online)

of fixed size. In these cases, feedforward architectures with fixed-size input can be applied. However, there exist many applications where the data items are not of fixed size. Furthermore, the data items come with a structure often represented by a graph. This is the case of sentences or parse trees in natural language processing, molecules or reactions in chemoinformatics, or nucleotide or amino acide sequences in bioinformatics and their two-dimensional contact maps. In all these cases, recursive neural network architectures must be used to process the variable-size structured data, raising the issue of how to design such architectures. Some approaches have been developed to design recursive architectures, but they have not been organized systematically. We present a classification of the known approaches into two basic classes: the inner class and the outer class. While we give examples of both, the goal of this brief technical note is not to review the entire literature on recursive networks. The goal, rather is to introduce the inner/outer distinction through a few examples and show that is helpful both to better understand and organize previous approaches, and to develop new ones.

Before delving into the description of the inner and outer approaches, it is useful to clarify the use of the terms recurrent and recursive networks. A recurrent neural network is a neural network that is represented by a directed graphs containing at least one directed cycle. A recursive network is a network that contains connection weights, often entire subnetworks, that are shared, often in a systematic way (Fig. 1). There is some flexibility in the degree and multiplicity of sharing that is required in order to be called recursive. For instance, at the low end, a convolutional network could be considered a recursive network, although the non-convolutional part of the network could be dominant. Likewise, a siamese network can also be called recursive, although it may combine only two copies of the same network. At the high end, any recurrent network unfolded in time yields a highly recursive network. Of course, the notion of recursive network becomes most useful when there is a regular pattern of connections, associated for instance with a lattice (see examples below).

2 The inner approach

The inner approach requires that the input structure be represented by a directed acyclic graph (DAG). In the case of sequences problems, for instance, these graphs are typically based on left-to-right chains associated with Bayesian network representations such as Markov Models, Hidden Markov Models (HMMs), Factorial HMMs, and Input–Output HMMs (Koller and Friedman 2009). In the inner approach, the variables associated with each node of the DAG are a function of the variables associated with all the parent nodes, and this function is parameterized by a neural network [see, for instance, Baldi and Chauvin (1996), Goller and Kuchler (1996) and Frasconi et al. (1998)]. Furthermore, the weights of neural networks sharing a similar functionality can be shared, yielding a recursive neural network approach (Fig. 1c–e). Thus in this recursive approach the individual networks can be viewed as "crawling" the DAG from the inside, along its edges, in order to produce the final output. The acyclic nature of the overall graph ensures that the forward propagation always converges to a fixed output and that the backpropagation equations can be applied in order to compute gradients and adjust each parameter during learning. The application of the inner approach to a hidden Markov model (HMM) is illustrated in Fig. 2. In this case, the variable H_t representing the hidden state at time t and the variable O_t representing the output symbol (or the distribution over output symbols) can be parameterized recursively using two neural networks NN_H and NN_O in the form

$$H_t = NN_H(H_{t-1}) \quad O_t = NN_O(H_t)$$

The inner approach has been successfully applied to, for instance, protein secondary structure or relative solvent accessibility prediction using bidirectional Input–Output HMMs (Baldi et al. 1999; Mooney and Pollastri 2009; Magnan and Baldi 2014). In this case, three neural networks are used to reparameterize the corresponding Bayesian network (Fig. 3) and compute at each position t the three key variables associated with the output (O_t), the forward hidden state (H_t^F), and the backward hidden state (H_t^B) in the form:

$$O_t = NN_O(I_t, H_t^F, H_t^B) \quad H_t^F = NN_F(I_t, H_{t-1}^F) \quad H_t^B = NN_B(I_t, H_{t+1}^B)$$

The same network NN_O is shared by all the outputs O_t, and similarly for the forward network NN_F, and the backward network NN_B.

The same approach has been generalized to the case of grids with 2 or more dimensions (Baldi and Pollastri 2003) and applied, for instance, to the problem of protein contact map prediction (Tegge et al. 2009), or the game of GO (Wu and Baldi 2008). [A contact map is a 2D matrix representation of a 3-D chain, where the (i, j) entry of the matrix is 1 if and only if the corresponding elements i and j in the chain are close to each other in 3D, and 0 otherwise]. In this case, the corresponding Input–Output HMM Bayesian network (see Fig. 8 in "Appendix") comprises four hidden 2D-grids or lattices, each with edges oriented towards one of the four cardinal corners, and one output grid corresponding to the predicted contact map (Baldi and Pollastri 2003). The complete system can be described in terms of five recursive neural net-

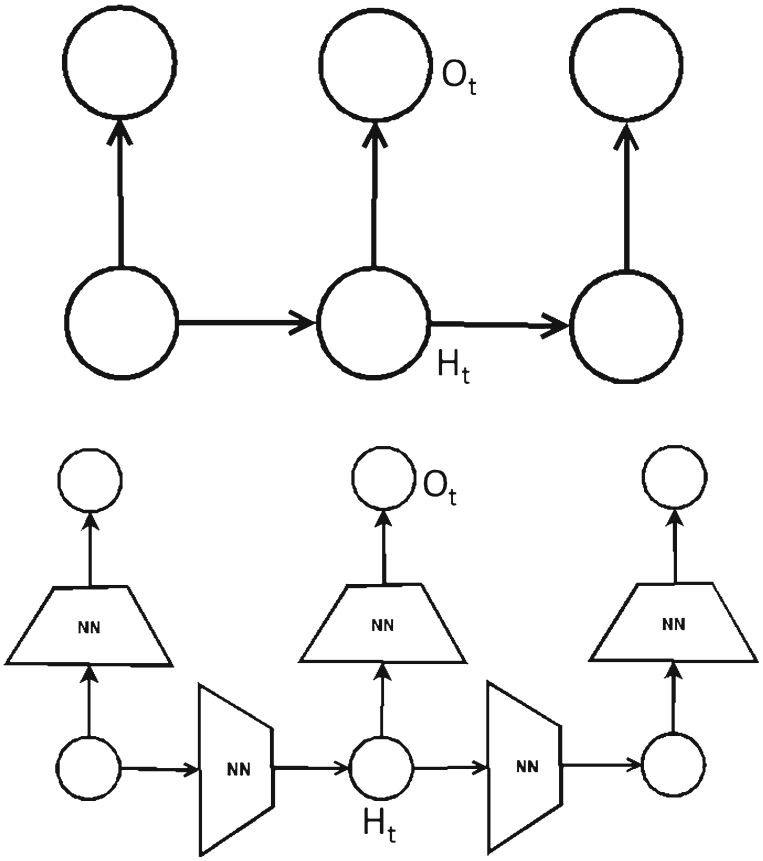

Fig. 2 The inner approach applied to a hidden Markov model (HMM) for sequences. *Top* Bayesian network representation of an HMM, where H_t denotes the hidden state at time t, and O_t the symbol produced at time t. A recursive neural network model derived from the HMM by parameterizing the transitions between hidden states and the production of symbols using neural networks. The transition neural network can be shared for all values of t, and similarly for the production neural network. Note that the output of the production network can either be a deterministic function of H_t or, using a softmax unit, a probabilistic distribution over the symbols of the alphabet dependant on H_t

works computing, at each (i, j) position the output $O_{i,j}$ and the four hidden variables $(H_{i,j}^{NE}, H_{i,j}^{NW}, H_{i,j}^{SE}, H_{i,j}^{SW})$ in the form:

$$O_{i,j} = NN_O\left(I_{i,j}, H_{i,j}^{NE}, H_{i,j}^{NW}, H_{i,j}^{SE}, H_{i,j}^{SW}\right)$$

with

$$H_{i,j}^{NE} = NN_{NE}\left(I_{i,j}, H_{i-1,j}^{NE}, H_{i,j-1}^{NE}\right)$$

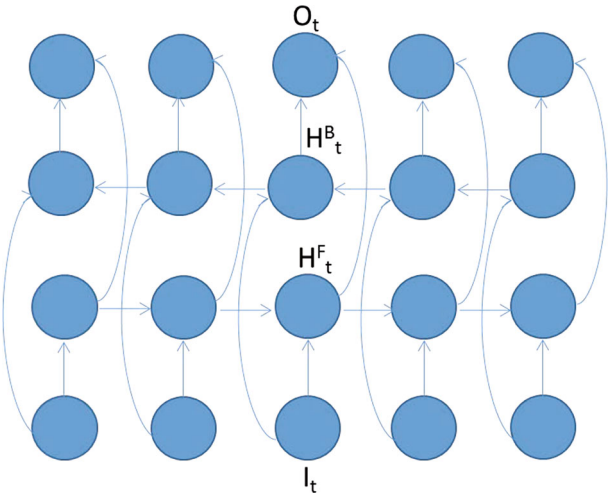

Fig. 3 Directed acyclic graph depicting a bi-directional input–output HMM. The recursive inner approach applied to this Bayesian network uses three neural networks: one to compute the output O_t, one to compute the forward hidden variable H_t^F, and one to compute the backward hidden variable H_t^B, as a function of the variables associated with the corresponding parent nodes

and similarly for the other three hidden variables (NE = North East, NW = North West, and so forth). In three dimensions, in the complete case, one would use nine recursive networks associated with nine cubes to compute one output variable $O_{i,j,k}$ and eight hidden variables at each position (i, j, k). In each hidden cubic lattice, all the edges are oriented towards one of the height corners. In K dimensions, the complete system would use 2^K hidden K-dimensional lattices, each with edges oriented towards one of the possible corners, giving rise to $2^K + 1$ neural networks with weight sharing, one for computing outputs, and 2^K networks for propagating context in all possible directions in each of the 2^K hidden lattices. To reduce the complexity, it is of course possible to use only a subset of the hidden lattices.

The same inner approach has been successfully used also in natural language processing, for instance in sentiment prediction, essentially by orienting the edges of parse trees from the leaves to the root, and crawling the parse trees with neural networks accordingly (Socher et al. 2013).

When the input structures are undirected graphs this approach can still be used but requires either finding a canonical way of acyclically orienting the edges, or sampling the possible acyclic orientations in some systematic way, or using all possible acyclic orientations. This problem arises, for instance, in chemoinformatics where molecules are represented by undirected graphs, where the vertices correspond to atoms and the edges correspond to bonds. The typical goal is to predict a particular physical, chemical, or biological property of the molecules. Note that in this case there is no natural way of orienting the corresponding edges in an acyclic fashion. This problem is addressed in Lusci et al. (2013), using solubility prediction as an example, by an inner approach that relies on orienting the molecular graphs in all possible ways. Each acyclic orientation is obtained by selecting one vertex of the molecule and orienting

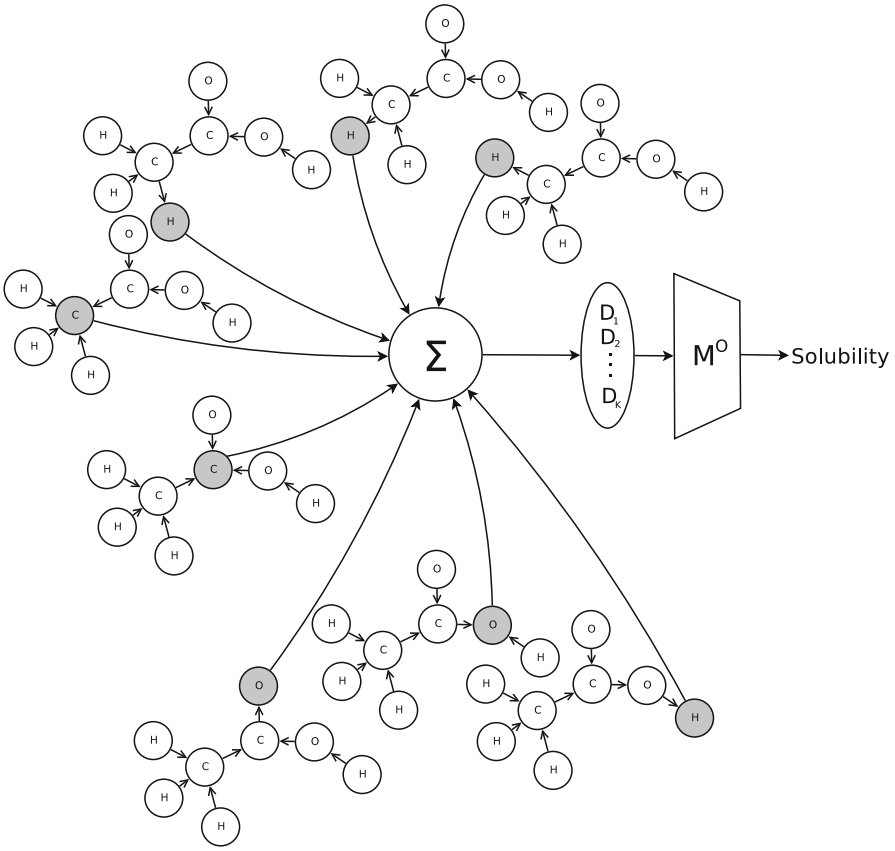

Fig. 4 Example of directed acyclic graph associated with the inner approach applied to a small molecule in organic chemistry where the overall goal is to predict solubility. In each copy of the molecules, all the edges are oriented towards the *grey node* (sink). The same neural network is used to crawl the molecular graphs in all possible orientations and the corresponding outputs are added and fed to the final prediction neural network. The networks used for crawling and for computing the final outputs are shared across all molecules in the training set

all the edges towards that vertex (Fig. 4). Considering all possible orientations is computationally feasible for small molecules in organic chemistry because the number of vertices is relatively small, and so is the number of edges due to valence constraints.

3 The outer approach

In the outer approach, the graphs representing the data can be directed or undirected and acyclic orientation is not required. This is because acyclic orientation is used in a different graph that is built "orthogonally" to the initial graph. In the most simple form of the approach, illustrated in Fig. 5 in the case of sequences, consider stacking K copies of the original graph on top of each other into K levels, and connecting the corresponding vertices of consecutive levels with directed edges running from the

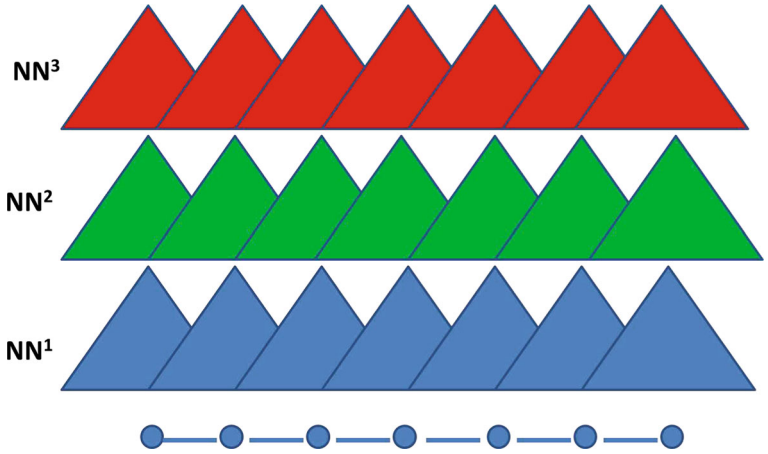

Fig. 5 Illustration of the outer approach in the case of sequence processing with three outer layers. In the figure, all the networks in a given layer share the same weights, have the same shape, and neural networks in different layers are distinct. All three conditions can be relaxed. Networks in the higher layers of the hierarchy integrate information over longer scales in the sequences

first to the last level. Additional diagonal edges running from level k to level $k+1$ can be added to represent neighborhood information. The new constructed graph is obviously acyclic and the inner approach can now be applied to it. So the activity O_i^k of the unit associated with vertex i in layer k is given by $O_i^k = F_i^k(O^{k-1}_{\mathcal{N}^{k-1}(i)}) = NN_i^k(O^{k-1}_{\mathcal{N}^{k-1}(i)})$, where $\mathcal{N}^{k-1}(i)$ denotes the neighborhood of vertex i in layer $k-1$. The last equality indicates that the function F_i^k is parameterized by a neural network. Furthermore the neural network can be shared, for instance within a layer, so that $O_i^k = NN^k(O^{k-1}_{\mathcal{N}(i)})$. It is also possible to have direct connections running from the input layer to each layer k (as in Fig. 6), or from any layer k to any layer l with $l > k$. In general, as the layer number increases, the corresponding networks are capable of integrating information over large scales in the original graph, as if the graph was being progressively "folded". At the top, the output of the different networks can be summed or averaged. Alternatively, it is also possible to have an outer architecture that tapers off to produce a small output. While this weight sharing is reminiscent of a convolutional neural network, and the outer approach is sometimes called convolutional, this is misleading because the outer approach is different and more general than the standard convolutional approach. In particular the weight sharing is not necessary, can be partial, can occur across layers, and so forth. Furthermore, convolutions can also be used within the neural networks of an inner approach, thus the terms convolutional and inner are not exclusive. Note also that the outer approach can be centered on the edges of the original graph, rather than its vertices. Finally, to further stress the duality between these two approaches, note that the outer approach can be viewed as an inner approach applied to an acyclic graph that is orthogonal to the original data graph, and the inner approach can be viewed as an outer approach applied to the boundary vertices of the original graph.

The early work on protein secondary structure prediction (Qian and Sejnowski 1988; Rost and Sander 1997) can be viewed as an outer approach, albeit a shallow

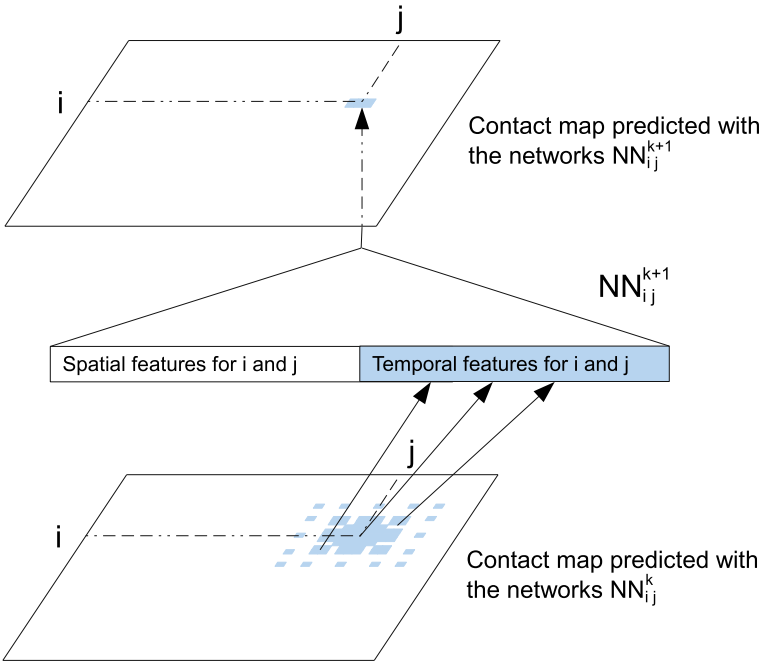

Fig. 6 Outer approach applied to the problem of protein contact map prediction. Within a layer, the networks have the same weights and receive input both from the networks in the previous layer (temporal features), and fixed input from the input layer (spatial features)

one, where essentially a single network with a relatively small input window (e.g. 11 amino acids) is used to predict the secondary structure classification (alpha-helix, beta-strand, or coil) of the amino acid in the center of the window. The same network is shared across all positions in a protein sequence, and across all proteins sequences.

A deep outer approach is applied systematically to protein contact map prediction in Di Lena et al. (2012) by stacking neural networks on top of the contact map, as shown in Fig. 6. In this case, the different layers are literally trying to progressively fold the entire protein. The shapes of the networks are identical across layers. Training can proceed either by providing contact targets at the top and backpropagating through the entire stack, or by providing the same contact targets at each layer. In the latter case, the weights obtained for layer k can be used to initialize the weight of layer $k+1$ before training.

Another example of outer approach is the recent application to small molecules (Duvenaud et al. 2015), obtained by stacking neural networks on top of each atom, using an approach inspired by circular fingerprints (Glen et al. 2006).

4 The problem of learning from sets

An example of a new application is the problem of learning from *sets* where the input consists, for instance, of a set of vectors and the set itself has variable size (the vectors

themselves could have variable size but for simplicity one can assume first that they have the same size). A set does not have a natural graph or DAG associated with it, other than perhaps the fully connected graph, and by definition its elements are unordered. This kind of problem occurs, for instance, in high-energy physics applications where collisions of particles can create complex jets of derived particles and associated events (Guest et al. 2016). Physicists have created tools that can parse these jets into primary and secondary tracks, as well as primary and secondary vertices (e.g. Piacquadio and Weiser 2008). Thus, for example, the input to a classifier could consist of a variable-size set of tracks, each track being described by a vector of fixed length. The overall goal of the classifier is to detect whether a particular event, such as the production of a particular particle, underlies the corresponding input or not.

To tackle such a problem, one can consider a shallow or deep neural network with an input window size equal to the largest input set size multiplied by the length of the vectors, with 0-padding for inputs associated with smaller sets. However this approach is cumbersome and possibly not scalable if the variability in the set sizes is large, and furthermore it requires an an arbitrary ordering of the elements in the sets.

In conjunction, or as an alternative to this padding approach, one could create an arbitrary ordering of the vectors (e.g. by the size of the first component) and then treat the input as a variable-length sequence of vectors. Both the inner and outer approaches to sequences could then be applied to this representation, although this has the shortcoming of requiring an artificial ordering of the elements of the sets.

The outer approach can be used here to build more symmetric deep architectures, for instance by applying the outer approach to the edges of the fully connected graph of all the pairs. Assuming up to 10 tracks in each example, there are at most $\binom{10}{2} = 45$ unordered pairs, or 90 ordered pairs, which is manageable especially considering that there is a single network shared by all pairs. Using ordered pairs would yield the most symmetric overall network. Suppose for instance that the input consists of 4 vectors representing 4 tracks, a, b, c, and d. Then there are 6 unordered pairs, or 12 ordered pairs, and in the outer approach we can have, at the first level, a shared network for each pair (Fig. 7). In the case of ordered pairs, the output of the networks applied to (a, b) and (b, a) can be summed together to provide a single symmetric output for the pair. The second level of the outer approach can be built at the level of the nodes, or at the level of the edges. At the level of the nodes, one would use a shared network for each node, where the network associated with node a receives inputs from all the networks from the previous layer associated with ordered or unordered pairs that contain a [e.g. (a, b), (a, c)]. At the level of the edges, one would use a shared network for each edge, where the network associated with edge (a, b) receives inputs from all the networks in the previous layers associated with (a, b) as well all the other edges adjacent to it [e.g. (a, c), (b, d)]. This approach can be iterated vertically up to a final network combining the outputs of the previous layers into a final classification.

5 Discussion

We have organized the main existing approaches for designing recursive architectures into two classes, the inner and the outer class. It is natural to wonder whether in general

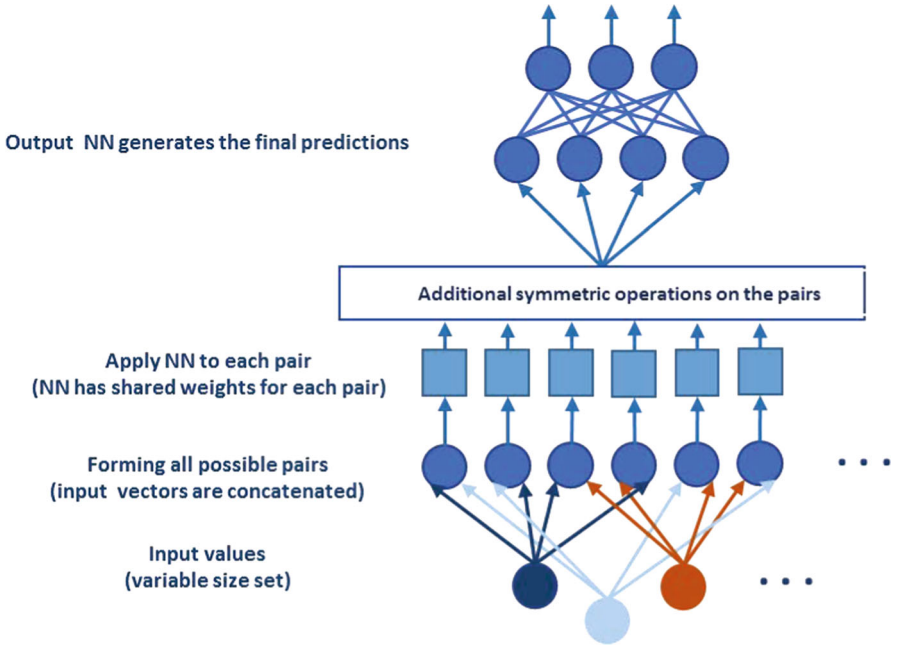

Fig. 7 Outer approach for processing sets of variable size in a symmetric way that does not require ordering the elements in the set

one class may be intrinsically better than the other one. A more careful examination, however, reveals that such a question is not really fruitful for several reasons.

First, the choice of one approach versus the other in a specific situation may depend on several other considerations, including ease of programming in a particular setting.

Second, given the universal approximation properties of neural networks, it is generally the case that anything that can be achieved using an inner approach can also be achieved using an outer approach, and vice versa. In fact the two approaches are somehow dual of each other. The outer approach can be viewed as an inner approach applied to a directed acyclic graph orthogonal to the original graph. Likewise the inner approach can be viewed as an outer approach applied to the source nodes of the original graph.

Third, the two approaches are not mutually exclusive, in the sense that they can be combined both at the same level and hierarchically. At the same level, for instance each approach can be applied separately and the predictions produced by the two approaches can be combined by simple averaging. Hierarchically, one approach can be used at one level to feed into the other approach at the next level. For instance in the case of variable-size sets consisting of variable-length vectors, the inner approach can be used at the level of the vectors viewing them as sequences of numbers, and the outer approach can be used at the level of the sets. Likewise, Long Short Term Memory (LSTM) recurrent units (Gers et al. 2000; Greff et al. 2015), which have proven to be useful for handling long-ranged dependencies, can also be combined with either approach.

Fourth, even when considering the same problem, each approach can be applied in many different ways to different representations of the data. For example, consider the problem of predicting the physical, chemical, or biological properties of small molecules in chemoinformatics. In terms of inner approaches, one can: (1) use the SMILES string representations and apply bidirectional recursive-neural networks (Fig. 3); (2) use the molecular graph representations and apply the method in Lusci et al. (2013) (Fig. 4); (3) represent the molecular graphs as contact maps (with entries equal to 0, 1, 2, or 3 depending on the number of bonds) and apply the 2D grid recurrent neural network approach (Baldi and Pollastri 2003) (Fig. 8); (4) represent the molecules as list of atomic nuclei 3D coordinates and apply an inner approach for lists or sets; or (5) represent the molecules by their fingerprints (e.g. Glen et al. 2006) and apply an inner approach to these fixed size vectors. Likewise, a corresponding outer approach can be applied to each one of these representations.

These various representations are "equivalent" in the sense that in general each representation can be recovered from any other one–with some technical details and exceptions that are not relevant for this discussion–and thus comparable accuracy results should be attainable using different representations with both inner and outer approaches. To empirically exemplify this point, using the inner approach in Lusci et al. (2013) and the outer approach in Duvenaud et al. (2015) on the benchmark solubility data set in Delaney (2004), we obtain almost identical RMSE (root mean square error) of 0.61 and 0.60 respectively, in line with the best results reported in the literature.

However, this is not to say that subtle trade offs between the different approaches and representations may not exist, but these tend to be problem specific. For instance, one would not recommend using a convolutional neural network (outer) approach for very long, sparse fingerprint vectors, without combining it first with a dimensionality reduction approach.

Finally, the inner/outer distinction is useful for revealing new approaches that have not yet been tried. For instance, to the best of our knowledge, a deep outer approach has not been applied systematically to parse trees and natural language processing, or to protein sequences and 1-D feature prediction, such as secondary structure or relative solvent accessibility prediction. The inner/outer distinction raises also software challenges for providing general tools that can more or less automatically deploy each approach on any suitable problem. As a first step in this direction, general software package implementations of the inner and outer approaches for chemoinformatics problems, based on the molecular graph representation, are being made publicly available both from github, www.github.com/Chemoinformatics/InnerOuterRNN, and from the ChemDB web portal, cdb.ics.uci.edu.

Acknowledgements This work was supported in part by Grants NSF IIS-1321053, NSF IIS-1550705, and DARPA D17AP00002. We also wish to acknowledge OpenEye and ChemAxon for free academic software licenses.

Appendix

See Fig. 8

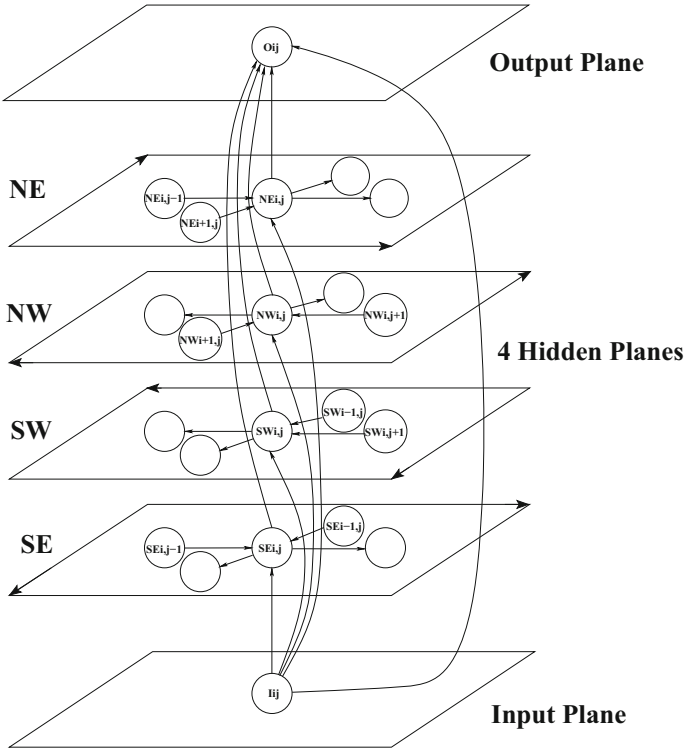

Fig. 8 Two-dimensional input–output HMM Bayesian network for contact map prediction. Whene the inner approach is applied to this Bayesian network, the output (i, j) is the probability that residues i and j are in contact. This probability is computed by a neural network, shared at all (i, j) positions, as a function of the corresponding input vector, and four hidden vectors, one in each of the four hidden planes (North East, North West, Sout-East, South-West). Each hidden vector in each hidden plane is computed by a neural network, shared by all the hidden nodes in the same hidden plane, as a function of the corresponding input vector, and the two neighboring hidden vectors in the same hidden plane

References

Baldi P, Brunak S, Frasconi P, Pollastri G, Soda G (1999) Exploiting the past and the future in protein secondary structure prediction. Bioinformatics 15:937–946

Baldi P, Chauvin Y (1996) Hybrid modeling, HMM/NN architectures, and protein applications. Neural Comput 8(7):1541–1565

Baldi P, Pollastri G (2003) The principled design of large-scale recursive neural network architectures–DAG-RNNs and the protein structure prediction problem. J Mach Learn Res 4:575–602

Delaney JS (2004) ESOL: estimating aqueous solubility directly from molecular structure. J Chem Inf Comput Sci 44(3):1000–1005

Di Lena P, Nagata K, Baldi P (2012) Deep architectures for protein contact map prediction. Bioinformatics 28:2449–2457. doi:10.1093/bioinformatics/bts475

Duvenaud DK, Maclaurin D, Iparraguirre J, Bombarell R, Hirzel T, Aspuru-Guzik A, Adams RP (2015) Convolutional networks on graphs for learning molecular fingerprints. In: Advances in neural information processing systems, pp 2215–2223

Frasconi P, Gori M, Sperduti A (1998) A general framework for adaptive processing of data structures. IEEE Trans Neural Netw 9(5):768–786

Gers FA, Schmidhuber J, Cummins F (2000) Learning to forget: continual prediction with LSTM. Neural Comput 12(10):2451–2471

Glen RC, Bender A, Arnby CH, Carlsson L, Boyer S, Smith J (2006) Circular fingerprints: flexible molecular descriptors with applications from physical chemistry to ADME. IDrugs 9(3):199

Goller C, Kuchler A (1996) Learning task-dependent distributed representations by backpropagation through structure. IEEE Int Conf Neural Netw 1:347–352

Greff K, Srivastava RK, Koutnik J, Steunebrink BR, SchmidhuberJ (2015) LSTM: a search space odyssey. arXiv:1503.04069

Guest D, Collado J, Baldi P, Hsu S-C, Urban G, Whiteson D (2016) Jet flavor classification in high-energy physics with deep neural networks. Phys Rev D 94:112002. doi:10.1103/PhysRevD.94.112002

Koller D, Friedman N (2009) Probabilistic Graphical Models: Principles and Techniques. The MIT Press, Cambridge

Lusci A, Pollastri G, Baldi P (2013) Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. J Chem Inf Model 53(7):1563–1575

Magnan CN, Baldi P (2014) SSpro/ACCpro 5: almost perfect prediction of protein secondary structure and relative solvent accessibility using profiles, machine learning, and structural similarity. Bioinformatics 30(18):2592–2597

Mooney C, Pollastri G (2009) Beyond the twilight zone: automated prediction of structural properties of proteins by recursive neural networks and remote homology information. Proteins Struct Funct Bioinform 77(1):181–190

Piacquadio G, Weiser C (2008) A new inclusive secondary vertex algorithm for b-jet tagging in atlas. J Phys Conf Ser 119:032032

Qian N, Sejnowski TJ (1988) Predicting the secondary structure of glubular proteins using neural network models. J Mol Biol 202:865–884

Rost B, Sander C (1997) Prediction of protein secondary structure at better than 70% accuracy. J Mol Biol 232:584–599

Socher R, Perelygin A, Wu JY, Chuang J, Manning CD, Ng AY, Potts C (2013) Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the conference on empirical methods in natural language processing (EMNLP), vol 1631, p 1642. Citeseer

Tegge AN, Wang Z, Eickholt J, Cheng J (2009) NNcon: improved protein contact map prediction using 2D-recursive neural networks. Nucleic Acids Res 37(suppl 2):W515–W518

Wu L, Baldi P (2008) Learning to play Go using recursive neural networks. Neural Netw 21(9):1392–1400