Chapter 10

Recursive Networks

This chapter focuses on recursive networks, also rebranded as graph neural networks (GNNs). This general concept is essential for desigining recurrent network as well as networks that can handle variable-size structured data and their many applications.

10.1 Variable-Size Structured Data

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 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 often come with a structure, often represented by a graph, which may also include labels on the nodes or the edges. Examples include:

- **Sets:** typically represented by lists, but where the order has no intrinsic meaning. For instance, the representation of a small molecule as a bag of atoms together with the x, y, z coordinates of the corresponding nuclei; the representation of documents as bags of words; the representation of particles showers in high energy physics as lists of objects (e.g. jets, tracks, vertices) and their properties.

- **Sequences, or ordered lists:** For instance, time series, sentences in natural language, pieces of software code, mathematical expressions, DNA/RNA sequences, protein sequences, SMILES and SMIRKS strings in chemoinformatics, and various string representations of chemical reactions.

- **Trees:** For instance parse trees for text sequences in natural language processing, parse trees for software code, parse trees for mathematical expressions, and evolutionary trees.

- **Small graphs:** For instance graphs representing Feynman diagrams in physics, small molecules or chemical reactions in chemistry, and RNA secondary structure in biology.

- **Large graphs:** representing networks, such as biological networks associated with biological pathways (e.g. protein-protein interactions, metabolic networks), friendships and other social networks, organization networks.

- **Grids, or matrices, or tensors:** For instance 2D contact maps and distance maps for nucleotide or amino acid sequences in computational biology, or 3D atmospheric grids, or grids associated with games and puzzles (e.g. Rubik's

cubes of different dimensions and sizes, such as $3 \times 3 \times 3$, $2 \times 2 \times 2 \times 2$, and $4 \times 4 \times 4$).

Note that these variable-size structures can appear at the input level, at the output level, or both. For instance, a translation problem, can be viewed as a sequence-to-sequence problem. The input and output sequences need not be of the same length. The prediction of the secondary structure of a protein can be viewed as a translation problem from an alphabet with 20 letters, one for each naturally occurring amino acid, to an alphabet with three letters, corresponding to the three main secondary structure classes (alpha-helix, beta-strand, and coil). In this particular case, the input sequence and the output sequence have the same length. In parsing problems, the input is a sequence and the output is a tree. In protein contact map prediction the input is a sequence, and the output is a matrix, and so forth. In all these problems, labels can exist on the nodes, on the edges, or both. A small molecule in organic chemistry can have labels on the nodes corresponding to the atom type (e.g. C, N, O, H), as well as on the edges, corresponding to the bond type (e.g. single, double, triple, and aromatic).

In some cases, it may be possible to use fixed size vectors, with zeros padding, to represent these objects and process them using feedforward neural networks. However, this is not very elegant and in many cases not efficient, especially if the sizes vary over a wide range. In general, it is better to use recursive neural network (RNNs), or graph neural networks (GNNs), to process these variable-size structured data. In the next sections, we describe what these are and how to design the corresponding architectures by describing two different approaches, the inner approach and the outer approach [69]. The distinction between inner and outer is not a fundamental one, as discussed further down. Rather it is a convenient way of organizing the heterogenous variety of RNN/GNN approaches described in the literature to help the reader get oriented.

10.2 Recursive Networks and Design

A recursive network is a network that contains connection weights, often entire subnetworks, that are shared, often in a systematic way (Figure 9.1). 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 a recursive network becomes most useful when there is a regular pattern of connections, associated for instance with a lattice(see examples below). The basic idea in recursive approaches is to reuse over and over the same neural network modules to process and pass information between nodes of graphs associated with the data, or derived from the data.

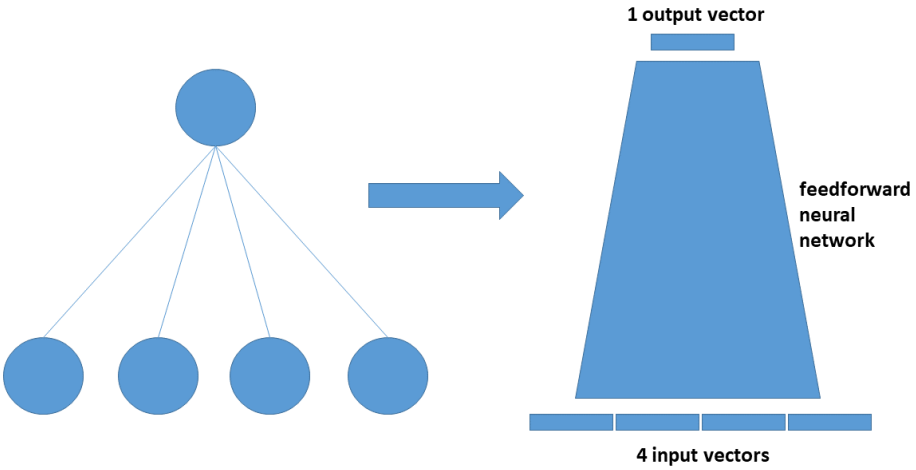

Figure 10.1: The basic building block of recursive approaches. Left: a piece of a graph consisting of four nodes connected to a fifth node which has been selected as the center. Each node comes with an associated vector. Right: a neural network that computes, or updates, the vector at the center node, as a function of the four vectors associated with the four other nodes.

To be more precise, by recursive networks we refer to a general set of approaches for designing and applying neural networks to variable-size data. In these recursive approaches the starting point is a class of graphs associated with the data, or derived from the data, such as a data model (e.g. HMMs). These graphs may have labels associated with the nodes or the edges, and may be directed or undirected. Extensions to hypergraphs are also possible. The recursive approach can be centered on the nodes, or the edges, or both. For simplicity here we will stay centered on the nodes, while still allowing for the possibility of processing labels located on edges.

The basic building block of recursive approaches is illustrated in Figure 10.1). In this approach a vector is attached to each node and the basic idea is that the vector of one node is a function of all the information (e.g. other vectors and node/edge labels) contained in a neighborhood of the node. While larger neighborhoods can

be considered, for simplicity we will consider only neighborhoods of radius one, i.e. immediate neighbors of a node. In the recursive neural network approach, the function associated with each node is parameterized by a neural network. The approach is called recursive because if the graph has some regular structure the same neural network can be reused (weight sharing) by all the nodes in the graph that are similar. If the underlying graphs are directed and acyclic, then the node vectors can be updated in an orderly fashion by propagation from the source nodes to the sink nodes. Below this approach is also called the inner-approach. Note that when a recurrent neural network is unfolded in time (Figure 9.1), this can be viewed as the application of an inner approach to a very simple chain graph, where the vector associated with each node is the vector of states of all the units at time t, and the same neural network is applied at all discrete time steps to go from the state at time t to the state at time $t + 1$. In this sense, the recursive approach is more general than the recurrent approach.

If the underlying graphs are not acyclic, then one typically uses synchronous update. This general approach is called the convolutional, or outer approach, in the literature. The synchronous update can be repeated but then one must specify how many times as, in general, there is no guarantee of convergence to stable values. It is also possible to use a first set of neural networks during the first pass of synchronous updates, then a second set of neural networks (e.g. with different architectures) during the second pass of asynchronous updates, and so forth. And finally, it is also possible to change the structure of the underlying graphs (e.g. by merging neighboring nodes) from one synchronous pass to the next. The fundamental point of these outer processing approaches is that when properly unfolded they operate on a directed acyclic (operational) graph, although the original data-associated graphs may not be directed, or acyclic. As a result, for each set of input variables they produce a unique set of output variables and, if input-output target pairs are available for training, then backpropagation can be applied through the operational acyclic graph. These approaches for propagating information through graphs using neural networks have broad connections to other areas of machine learning, statistics, and operation research (e.g. belief propagation). In the next sections, we give specific examples of inner and outer approaches.

10.2.1 Inner Approaches

The inner approach requires that the structures of interest be represented by directed acyclic graphs (DAGs) (Figures 10.2). There is no established nomenclature for the the DAGs but we will use some of corresponding names from the theory of probabilistic graphical models, in particular Bayesian networks which are associated with DAGs, while discarding their probabilistic interpretation. Even then, the

nomenclature is not completely established. Important parameters to keep in mind are:

- The presence of inputs, outputs, or both.
- The dimensionality (1D, 2D, 3D, etc) of the inputs, outputs, or both. When the inputs and output have the same dimensionality only one dimension may be mentioned.
- The presence of non visible nodes (hidden factors, hidden chains, or hidden lattices), their number, and connectivity within each group of nodes, and across groups.
- Whether the hidden chains or lattices are oriented in all directions (bidirectional in 1D, omni-directional in higher dimensions) or whether only one subset of the orientations (e.g. left-right in 1D) is represented.

In the case of 1D sequence problems, for instance, these graphs are typically based on left-to-right chains associated with Bayesian network representations such as Markov Chains, Hidden Markov Models (HMMs) (Figure 10.3), Factorial HMMs, and Input-Output HMMs (IOHMMs) (Figure 10.4) [376, 464]. HMMs and factorial HMMs have outputs only, whereas IOHMMs have both inputs and outputs. Factorial HMMs have one, or multiple, chains of hidden factors, all running from left to right. Higher order Markov versions of all these structures are possible (see exercises). A bidirectional HMM or IOHMM has an additional chain of hidden states running from right to left. A bidirectional multifactorial IOHMMs could have multiple chains, running in both directions (and one would have to specify exactly how many and their exact pattern of connectivity).

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, [100, 284, 259]). Furthermore, the weights of the neural networks sharing a similar functionality can be shared, yielding a recursive neural network approach (Figures 9.1c, 9.1d, 9.1e). For instance, in the case of an HMM DAG, two neural networks can be used, one shared across all outputs, and one shared across all hidden transitions (Figure 10.3). In the case of a factorial HMM DAG with two hiddent factors, three neural networks can be used, and so forth. Thus in this recursive approach the individual networks can be viewed as "crawling" the DAG along its edge the inside–hence the name of the approach–starting from the source nodes and ending in the sink nodes of the DAG. At the end of this crawling process, a well defined vector is assigned to each node in the DAG. The acyclic nature of the overall graph ensures that the forward propagation always converges to a final state and that the backpropagation equations can be applied

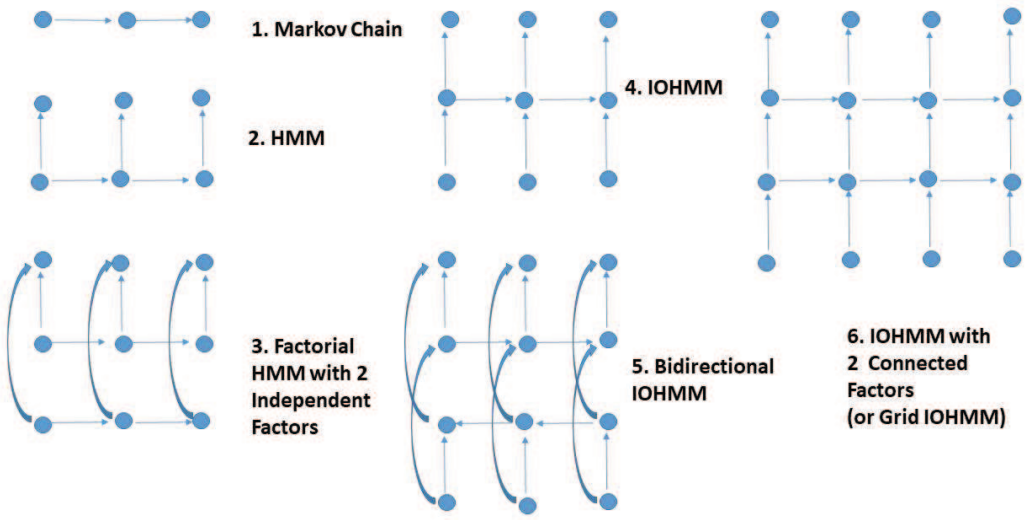

Figure 10.2: Examples of DAGs and their associated probabilistic graphical model names for 1D objects, such as time series or sequences. (1) First-order Markov chain DAG where the state at time t directly depends on the state at time $t-1$. This is the DAG structure ones uses when unfolding a recurrent neural network in time. (2) First-order Hidden Markov Model (HMM) DAG which produces outputs as a function of hidden states. (3) First-order Factorial HMM DAG where the outputs are function of two different kinds of hidden states, corresponding to two separate left-right Markov chains. Connections from one hidden chain to the other can be added without introducing directed cycles. (4) First-order input-output HMM (IOHMM) DAG which produces outputs as a function of both inputs and hidden states. Connections from inputs to outputs can be added without introducing directed cycles. (5) First-order bidirectional IOHMM DAG which produces outputs as a function of both inputs and hidden states. The hidden states are organized in two chains, one running left-to-right and one running right-to-left. Connections from one hidden chain to the other could be added without introducing directed cycles. (6) First-order IOHMM with 2 Connected Hidden Factors (or Grid IOHMM) DAG which produces outputs as a function of both inputs and hidden states. As the number of hidden factors is increased, this DAG becomes a 2D grid. All edges are oriented towards the North-East corner.

in order to compute gradients and adjust each parameter during learning. If there is no specific information at the sources nodes, which is typical for hidden source nodes, the corresponding vectors can be initialized to 0. It is also possible to use backpropagation to optimize the initial value of such vectors.

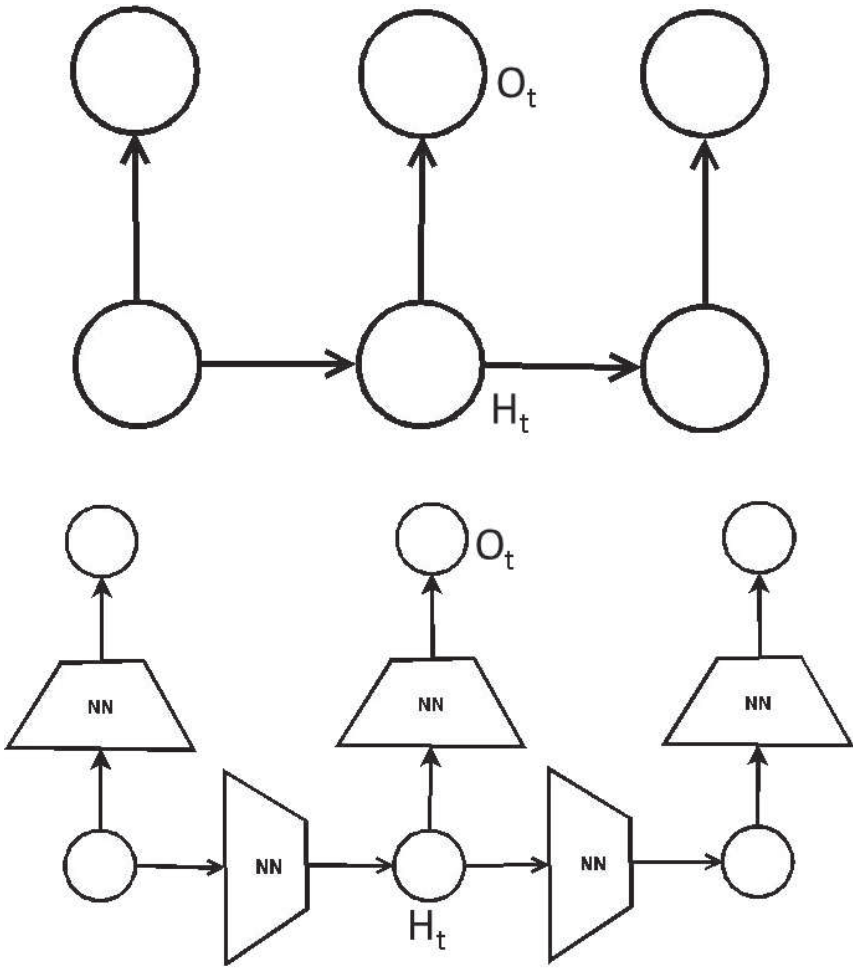

Figure 10.3: The inner approach applied to a hidden Markov model(HMM) DAG for sequences. Top: Bayesian network representation of an HMM, where H_t denotes the hidden state at time t, and O_t the symbol produced at time t. A recursive neural network model derived from the HMM by parameterizing the transitions 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 dependent on H_t.

The application of the inner approach to a hidden Markov model (HMM) DAG is illustrated in Figure 10.3. 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

$$H_t = NN_H(H_{t-1}) \quad O_t = NN_O(H_t) \tag{10.1}$$

In a second order HMM DAG, the same relations could be expanded to:

$$H_t = NN_H(H_{t-1}, H_{t-2}) \quad O_t = NN_O(H_t, H_{t-1}) \tag{10.2}$$

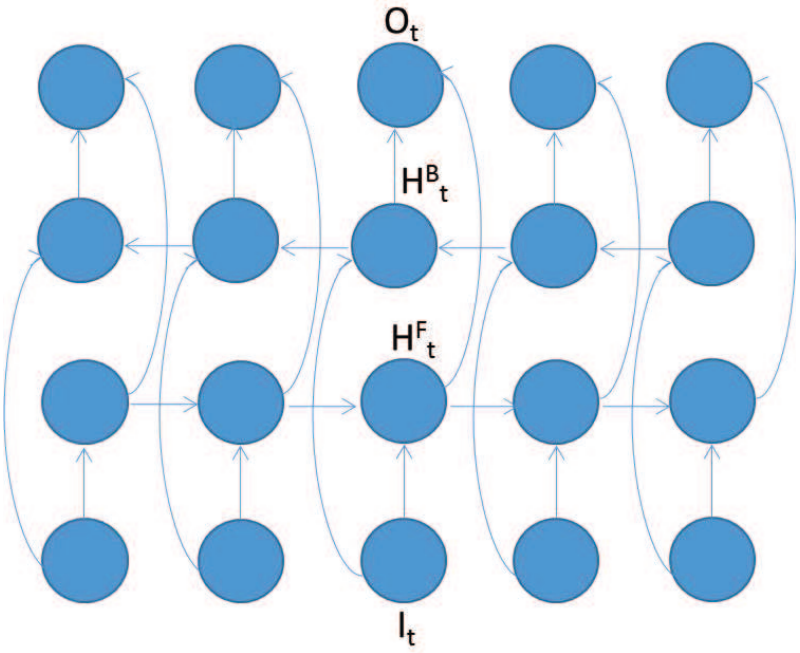

Figure 10.4: Directed acyclic graph depicting a bi-directional Input-Output HMM DAG. The recursive inner approach applied to this DAG relies on three neural networks: one to compute the output O_t, one to compute the forward hidden variable H_t^F, and one to compute the backward hidden variable H_t^B, as a function of the variables associated with the corresponding parent nodes.

The inner bidirectional IOHMM DAG approach, also known as bidirectional recursive neural networks (BRNNs) was introduced in [73] for the problems of protein secondary structure or relative solvent accessibility prediction. It has been refined over the years and has led to the best predictors in the field (e.g. [177, 459, 427]). In this case, three neural networks are used to refactor the corresponding Bayesian

network (Figure 10.4) 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:

$$O_t = NN_O(I_t, H_t^F, H_t^B) \quad H_t^F = NN_F(I_t, H_{t-1}^F) \quad H_t^B = NN_B(I_t, H_{t+1}^B) \tag{10.3}$$

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.

Without tediously examining other intermediate possibilities, we can directly ask the question of how this approach could be generalized to 2D? In other words, how should a 2D omni-directional IOHMM DAG look like? Note that East-West or North-South chains could be added to the Grid IOHMM of Figure 10.2, but not both. The natural generalization to 2D of 1D Bidirectional IOHMM was introduced in [87, 507] and subsequently applied to the problem of predicting protein contact or distance maps (see chapter on applications to biomedical data) and learning how to play the game of GO [88, 512, 684, 685]. (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 (Figure 10.5), the corresponding Input-Output HMM Bayesian network DAG comprises four hidden 2D-grids or lattices, each with edges oriented towards one of the four cardinal corners (NE, NW, SE, SW), and one output grid corresponding to the predicted contact map. The complete system can be described in terms of five recursive neural networks 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:

$$O_{i,j} = NN_O(I_{i,j}, H_{i,j}^{NE}, H_{i,j}^{NW}, H_{i,j}^{SE}, H_{i,j}^{SW}) \tag{10.4}$$

with:

$$H_{i,j}^{NE} = NN_{NE}(I_{i,j}, H_{i-1,j}^{NE}, H_{i,j-1}^{NE}) \tag{10.5}$$

and similarly for the other three hidden variables.

In 3D, in the complete omni-directional case, one would use nine recursive networks associated with nine cubic grids to compute one output variable $O_{i,j,k}$ and eight hidden variables at each position (i, j, k) in each hidden grid. In each hidden cubic lattice, all the edges are oriented towards one of the eight corners. The input at position (i, j, k) is connected to the corresponding positions in the eight hidden

cubic grids, as well as in the final output grid. The output at position (i, j, k) receives connections from the corresponding positions in the eight hidden cubic grids, as well as in the input plane.

In d dimensions, the complete system would use 2^d hidden d-dimensional lattices, each with edges oriented towards one of the possible corners, giving rise to $2^d + 1$ recursive neural networks with weight sharing, one for computing outputs, and 2^d networks for propagating context in all possible directions in each of the 2^d hidden lattices. Each input location is connected to the corresponding location in all the 2^d hidden lattices and the output lattice. Each output location received connections from the corresponding locations in all the 2^d hidden lattices and the input hyperplane. To reduce the complexity of these models, it is of course possible to use only a subset of the hidden lattices.

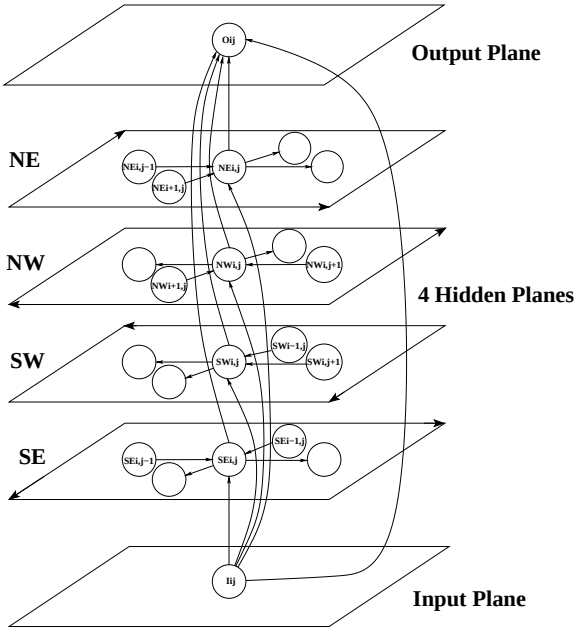

Figure 10.5: Two-dimensional Input-Output DAG for contact or distance map prediction. When the inner approach is applied to this DAG, the output (i, j) is the probability that residues i and j are in contact (or their distance). This probability is computed by a neural network, shared at all (i, j) positions, as a function of the corresponding input vector, and four hidden vectors, one in each of the four hidden planes (North East, North West, South-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.

Many of the approaches developed for natural language processing problems

are special cases of the inner approach and the models described so far. For instance, in translation problems, an inner approach is used using a DAG corresponding to a multifactorial IOHMM, which ends up looking like a 2D grid. It is also possible to use the inner approach to process trees. For instance, to predict the sentiment of a sentence, one can orient the edges of a parse tree from the leaves to the root, and crawl the directed parse tree with neural networks accordingly [598]. In this case, the basic recursive module is a neural networks that takes two vectors in the input and outputs a single vector.

We can now mention a subtle point hidden in Figure 10.1: in order to apply the recursive approach systematically, one must know in which order to concatenate the vectors associated with the neigbors as input to the corresponding neural network. In all the examples reviewed so far the ordering problem does not come up either because the nodes have a single incoming edge (HMMs) or because they have a small number of edges and each one has a different type associated with it. For instance a node in a 2D grid where all the nodes are oriented towards the NE corner, has two incoming edges, each of a different kind (latitude and longitude). Therefore it is easy to impose an order (e.g. latitude first, longitude second) and share it across all the similar nodes. However this issue can come up in situations characterized by more heterogeneous graphs.

When the graphs associated with, or derived from, the data are undirected, the inner approach can still be used. However its application requires either finding a canonical way of acyclically orienting the edges, as we did above in the case of parse trees, 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. In in this case there is no obvious natural way of orienting the corresponding edges in an acyclic fashion. This problem is addressed in [420] for the problem of solubility prediction, using 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 all the edges towards that vertex (see chapter on chemistry applications, Figure 12.3). 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.

10.2.2 Outer Approaches

In the outer approach, the graphs associated with the data can be directed or undirected, and an acyclic orientation is not required. This is because an acyclic orientation is used in a different graph that is built "orthogonally" to the initial graph, hence the outer qualifier. In the simplest form of the approach, illustrated in Figure 10.6 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 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:

$$O_i^k = F_i^k(O^{k-1}_{\mathcal{N}^{k-1}(i)}) = NN_i^k(O^{k-1}_{\mathcal{N}^{k-1}(i)}) \tag{10.6}$$

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:

$$O_i^k = NN^k(O^{k-1}_{\mathcal{N}(i)}) \tag{10.7}$$

It is also possible to have direct connections running from the input layer to each layer k (as in Figure 10.7), 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 mandatory, or it can be partial, or it can occur across layers, and so forth. The convolutional approach used in computer vision is a special case of outer approach. The outer approach can also be deployed on the edges of the original graph, rather than its vertices.

For instance, the early work on protein secondary structure prediction [517, 547] can be viewed as a 1D outer approach, albeit a shallow one, where essentially a single network with a relatively small input window (e.g. 11 amino acids) is used

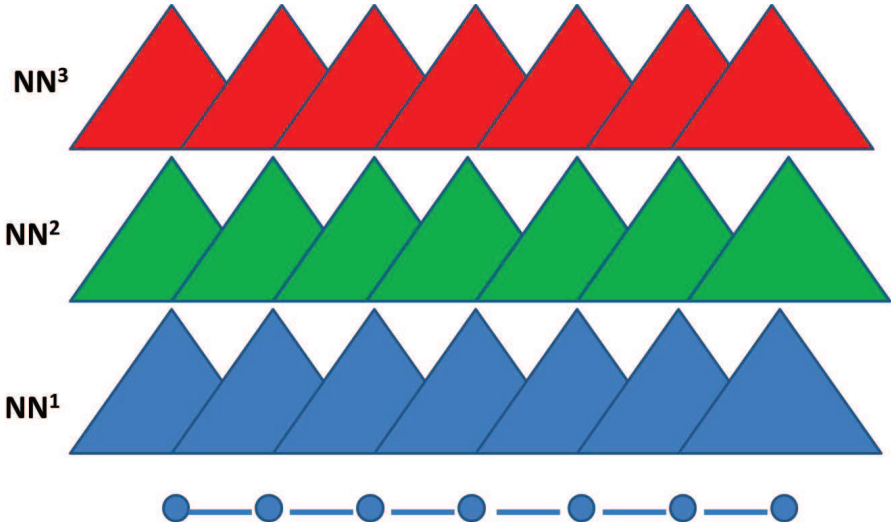

Figure 10.6: Illustration of the outer approach in the case of sequence processing where neural networks are stacked up in a direction perpendicular to the sequence in three layers. In the figure, all the networks in a given layer share the same weights, have the same shape, and neural networks in different 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.

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.

The first deep outer approach for 2D contact map prediction was described in [224] by stacking neural networks on top of the contact map, as shown in Figure 10.7 (see chapter on biomedical applications). 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 application to small molecules [231], obtained by stacking neural networks on top of each atom, using an approach inspired by circular fingerprints [280].

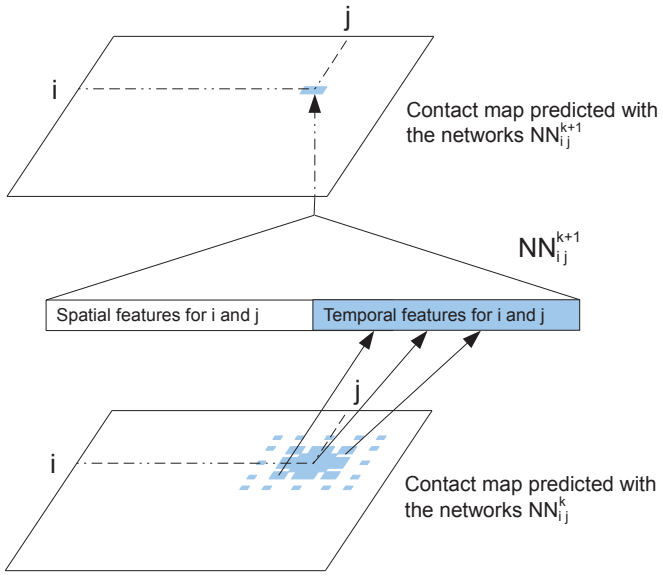

Figure 10.7: Outer approach applied to the problem of protein contact map prediction. Within a layer, the networks have the same weights and receive input both from the networks in the previous layer (temporal features), and fixed input from the input layer (spatial features).

10.3 Relationships between Inner and Outer Approaches

For simplicity, 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 the two approaches are fundamentally different and whether one approach is better than the other in general, or on specific classes of problems. As already hinted, the distinction is not fundamental and one approach is not better than the other.To see this in better details, consider the following points.

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 in parallel or sequentially in a data processing pipeline. For instance, when applied in parallel at the same level, each approach can be applied separately and the predictions produced by the two approaches can be combined by simple averaging. When applied sequentially, one approach can be used at one level and feed its results 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) ([316, 277, 371], a kind of recurrent neural network building block capable of learning or storing contextual information over different temporal or spatial length scales, 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 (see chapter on applications to chemistry). In terms of inner approaches, one can: (1) use the SMILES string representations and apply bidirectional recursive-neural networks (Figure 12.3), or Grid IOHMMs; (2) use the molecular graph representations and apply the method in [420]; (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 [88] (Figure 10.5); (4) represent the molecules as unordered lists of atomic nuclei and their 3D coordinates and develop an inner approach for sets; or (5) represent the molecules by their vector fingerprints (e.g. [280]) and apply an inner approach to these fixed size vectors. Likewise, a corresponding outer approach can be applied to each one of these representations.

In the case of small molecules, 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 [420] and the outer approach in [231] on the benchmark solubility data set in [223], we have obtained almost identical RMSE (root mean square error) of 0.61 and 0.60 respectively, in line with the best results reported in

the literature.

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. Likewise the inner approach Likewise the 2D grid recurrent neural network approach has not been applied systematically to small molecules represented by adjacency matrices. The inner/outer distinction also raises 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 available both from github: `www.github.com/Chemoinformatics/InnerOuterRNN`, and from the ChemDB web portal: `cdb.ics.uci.edu`.

10.4 Exercises

10.1 For the following list of 1D inner approach models: (a) describe (or draw) the corresponding directed acyclic graphs; (b) provide a proof that the corresponding graphs are acyclic; (c) identify edges that, if they were added, would cause the graph to have directed cycles; and (d) identify the number of recursive neural networks required to implement the corresponding model using the inner approach, identifying each network by its input and output variables. The list of models is as follows:
(1) Markov chain of order 2 where the state at time t depends on the states at times $t-1$ and $t-2$.
(2) Markov chain of order k where the state at time t depends on the states at times $t-1, \ldots, t-k$.
(3) HMM with hidden states of order 2, where the hidden states form a Markov chain of order 2.
(4) HMM with output states of order 2, where the output at time t depends on the hidden states at time t and $t-1$.
(5) HMM of order 2 where the hidden states from a Markov chain of order 2 and the output at time t depends on the hidden states at time t and $t-1$;
(6) Similarly for an HMM of order k.
(5) Factorial HMM with one hidden factor.
(6) Factorial HMM with k hidden factors.
(7) Factorial HMM of order 2 with k factors.
(8) Input-Output HMM of order 2. In each case, explain how to initialize the source nodes.

10.2 For the following list of 1D bidirectional inner approach models: (a) describe (or draw) the corresponding directed acyclic graphs; (b) provide a proof that the corresponding graphs are acyclic; (c) identify edges that, if they were added, would cause the graph to have directed cycles; and (d) identify the number of recursive neural networks required to implement the corresponding model using the inner approach, identifying each network by its input and output variables. The list of models is as follows: (1) Bidirectional Factorial HMM.
(2) Bidirectional Factorial HMM with k bidirectional factors.
(3) Bidirectional Factorial HMM of order 2.
(4) Bidirectional Input-Output HMM of order 2.
(5) Bidirectional Input-Output HMM of order k.
In each case, allow or disallow connections between the hidden factors and explain how to initialize the source nodes.

10.3 For the following list of 2D inner approach models: (a) describe (or draw) the corresponding directed acyclic graphs; (b) provide a proof that the corresponding graphs are acyclic; (c) identify edges that, if they were added, would cause the graph to have directed cycles; and (d) identify the number of recursive neural networks required to implement the corresponding model using the inner approach, identifying each network by its input and output variables. The list of models is as follows:
(1) 2D omni-directional HMM (4 hidden factors).
(2) 2D omni-directional HMM of order 2 (4 hidden factors).
(3) 2D IOHMM with only 2 hidden factors (e.g. NE and NW).
(4) 2D IOHMM of order 2 with only 2 hidden factors (e.g. NE and NW) Input-Output HMM.
(5) 2D omni-directional IOHMM.
(6) 2D omni-directional IOHMM of order 2.
In each case, allow or disallow connections between the hidden factors and explain how to initialize the source nodes.

10.4 (1) Implement the code to deploy and train a 3D omni-directional IOHMM inner approach (with 8 hidden cubes).
(2) Implement the code to deploy and train a 3D omni-directional IOHMM inner approach of order 2 (with 8 hidden cubes).

10.5 (1) Implement the code to deploy and train an omni-directional IOHMM inner approach in d dimensions (with 2^d hidden cubes).
(2) Implement the code to deploy and train an omni-directional IOHMM inner approach in d dimensions of order 2 (with 2^d hidden cubes).
(3) Implement the code to deploy and train an omni-directional IOHMM inner approach in d dimensions of order k (with 2^d hidden cubes).

10.6 In all the IOHMM DAG models above, the input and output spaces are of the same dimensionality. Explore possible DAGs for IOHMMs between spaces that do not have the same dimensionality, for instance from 1D to 2D, 1D to 3D, 2D to 3D, 2D to 1D, 3D to 1D, 3D to 2D, 1D to trees, and describe possible corresponding recursive neural networks.

10.7 Can the 2D omni-directional IOHMM inner approach be applied if each input consists of 2 distinct sequences of different length? Provide examples of problems where this approach could be applied.

10.8 In all the IOHMM models above, the input and o:utput spaces are of the same dimensionality. Explore possible DAGs for IOHMMs between spaces that do not have the same dimensionality, for instance from 1D to 2D, 1D to 3D, 2D to 3D, 2D to 1 D, 3D to 1D, 3D to 2D and then describe the corresponding recursive neural networks. For each case, provide an example of a problem where this approach could be applied.

10.9 Consider the general parsing problem (for natural language or for mathematical expressions) as a translation problem going from a sequence to a binary tree. Show that the problem can be converted into a sequence- to-sequence translation problem. Develop at least two different inner approaches and two different outer approaches for dealing with this problem. Identify a publicly available data set of annotated mathematical expressions and test your approaches on this data set.

10.10 Describe two distinct inner and two distinct outer approaches for the prediction of contact or distance maps in proteins. Estimate the total number of parameters of each corresponding recursive neural network architecture.

10.11 Design as many recursive neural network architecture as possible for input-to-output problems, where the underlying structure of all the inputs or outputs belongs to one of these four categories: sets, sequences, trees, small graphs (16 possibilities in total). For example, develop a recursive architecture for dealing with a set-to-graph problem. For each case, provide an example of a problem where this approach could be applied.