Connectionist Computing
COMP 30230/41390

Gianluca Pollastri

office: E0.95, Science East.

email: gianluca.pollastri@ucd.ie

Credits

- **Geoffrey Hinton, University of Toronto.**
  - **borrowed some of his slides for "Neural Networks" and "Computation in Neural Networks" courses.**

- **Ronan Reilly, NUI Maynooth.**
  - **slides from his CS4018.**

- **Paolo Frasconi, University of Florence.**
  - **slides from tutorial on Machine Learning for structured domains.**

Lecture notes on Brightspace

- **Strictly confidential...**
- **Slim PDF version will be uploaded later, typically the same day as the lecture.**
- **If there is demand, I can upload onto Brightspace last year's narrated slides.. (should be very similar to this year's material)**

Books

- No book covers large fractions of this course.

- Parts of chapters 4, 6, (7), 13 of Tom Mitchell's "Machine Learning"
- Parts of chapter V of Mackay's "Information Theory, Inference, and Learning Algorithms", available online at:

http://www.inference.phy.cam.ac.uk/mackay/itprnn/book.html

- Chapter 20 of Russell and Norvig's "Artificial Intelligence: A Modern Approach", also available at:

http://aima.cs.berkeley.edu/newchap20.pdf

- More materials later..

Marking

- **3 landmark papers to read, and submit a 10-line summary on Brightspace about: each worth 6-7%**
- **a connectionist model to build and play with on some sets, write a report: 30%**
- **Final Exam in the RDS (50%)**

Programming assignment

- Implement a Multi-Layer Perceptron, test it.
- The description on Brightspace.
- Submit through Brightspace code and test results by Dec the 5th at 23:59, any time zone of your choice (Baker Island?).
- 30% of the overall mark
- One third of a grade down every day late, that is: if you deserve an A and you're 1 day late you get an A-, 2 days late a B+, etc.

Deep learning

- **Deep nets are expressive**
- **But gradients vanish**

- **A long history of ad hoc solutions**
- **Over the last ten years, new solutions**

"New" deep learning 1.0

- **Layer by layer pre-training based on:**
  - **auto-association**
  - **RBM (Deep Belief Networks)**
- **Hard targets for inner layers**
- **More CPU and patience**

About pre-training

- If by auto-association, can be done on unlabelled data:
  - more data
  - generic compression
- Or, it can be done with a target (no auto-association):
  - less data (needs to be labelled)
  - target-driven compression

It depends on the problem!

- **Deep learning has produced some pretty stunning results in some fields (e.g. computer vision).**
- **In other fields, going from shallow to less shallow usually helps, but there is no need (or scope) for true deep learning.**

Algorithms plus data plus CPU (or GPU) plus ease of use

- Many algorithms used in deep learning have been around for a while. At most they have been combined and shuffled cleverly.
- The big changes are:
  - immense amounts of data
  - faster computers and, especially, the ability to run training algorithms on graphics cards 1+ orders of magnitude faster, $ for $
  - A number of environments/libraries that have made formerly highly complicated implementations accessible
  - A LOT of buzz..

Next..

- **A number of deep architectures I have worked with.**
- **Interestingly, most of them need only relatively lightweight (or no) deep learning techniques to be trained.**

Structure and neural nets

- **A great historical limitation of neural networks is that you have to decide beforehand how many inputs, outputs one has.**
- **This means that only maps/functions where inputs and outputs are vectors of fixed, known length can be dealt with directly.**

Structure and neural nets (2)

- Data tend to not come in fixed lengths.
- We have to brutalise them into them.

- Think of the features used for handwritten digit recognition. They may be good or not. But there is a good chance that they don't contain the full information available in the raw data.

We should think ahead of the data

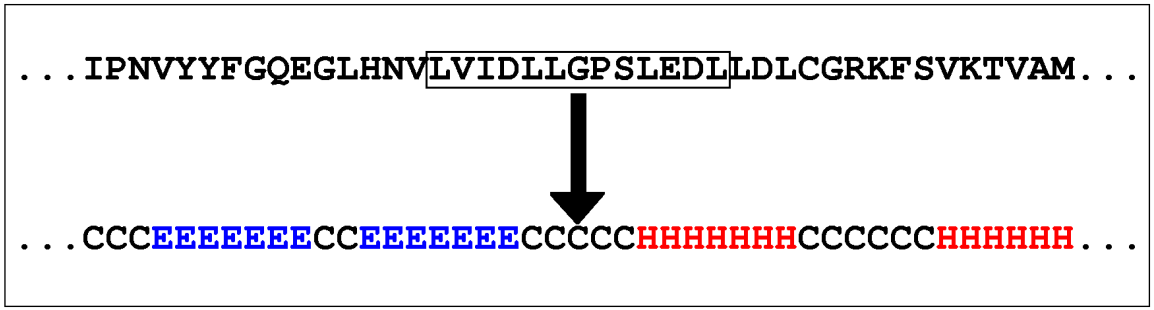
- **As more data become available and more complex problems are tackled, clever machine learning methods that stand on a few more parameters and a few more layers of complexity may also become useful…**

Sequences as structure: N to N

- **Think of language (e.g. mapping words into sounds). It's an N-to-N map, where N is unknown a priori, and variable between different streams of text.**
- **Think of biological sequences (DNA, proteins). Their lengths are wildly variable.**
-
- **How do you design a network that can deal with ALL the different lengths?**

N-to-N: traditional solution

- **N is variable, and this is a problem.**
- **Neural networks (and SVM, etc. etc.) like fixed sizes..**
- **Split N-to-N into N W-to-1 maps (W fixed).**



Problem

- **If we cut up the input string into windows of size W, we only see what is inside the window..**
- **We consider only a few letters, we ignore most of them.**
- **But we know that letters that matter may be anywhere, in some cases.**
- **Infinite trial and error W? Overfitting?**

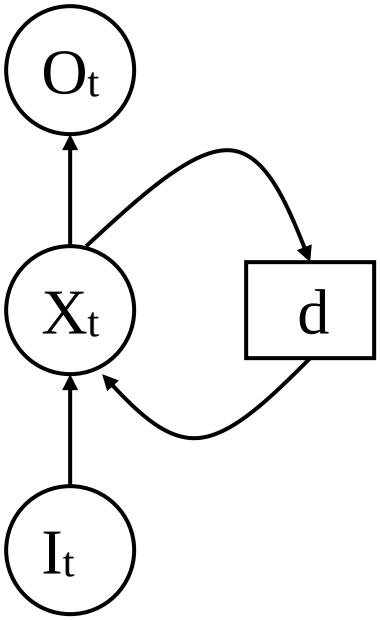
Recurrent Neural Networks (RNN)

- **One of the earliest versions: Jeffrey Elman, 1990, Cognitive Science.**

- **Problem: it isn't easy to represent time with Feedforward Neural Nets: usually time is represented with space.**
- **Attempt to design networks with memory.**

RNNs

- **The idea is having discrete time steps, and considering the hidden layer at time t-1 as an input at time t.**
- **This effectively removes cycles: we can model the network using an FFNN, and model memory explicitly.**



d = delay element

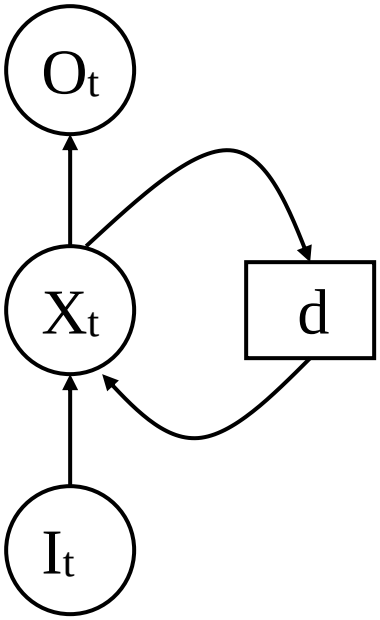
BPTT

- **BackPropagation Through Time.**
- **If O_t is the output at time t, I_t the input at time t, and X_t the memory (hidden) at time t, we can model the dependencies as follows:**

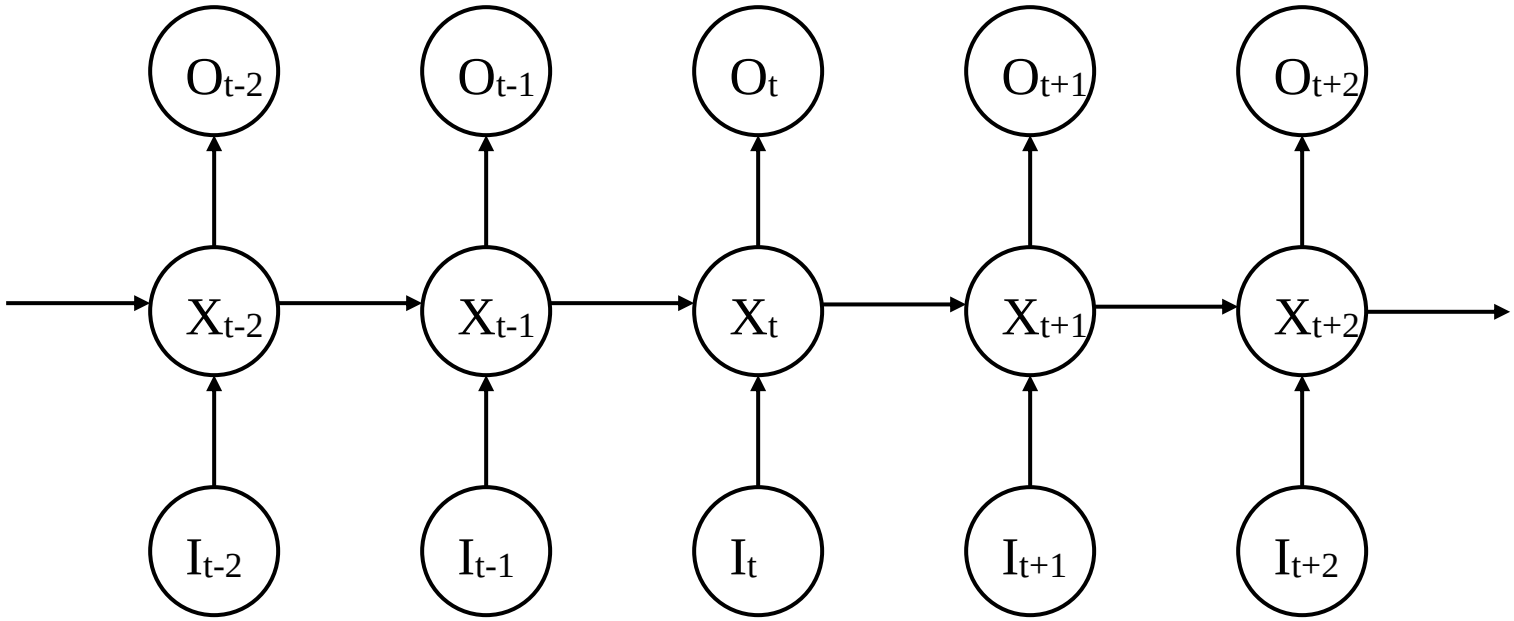
$$X_t = f(X_{t-1}, I_t)$$
$$O_t = g(X_t, I_t)$$

BPTT

- **We can model both f() and g() with (possibly multilayered) networks.**
- **We can transform the recurrent network by unrolling it in time.**
- **Backpropagation works on any Directed Acyclic Graph (DAG). An RNN becomes one once it’s unrolled.**

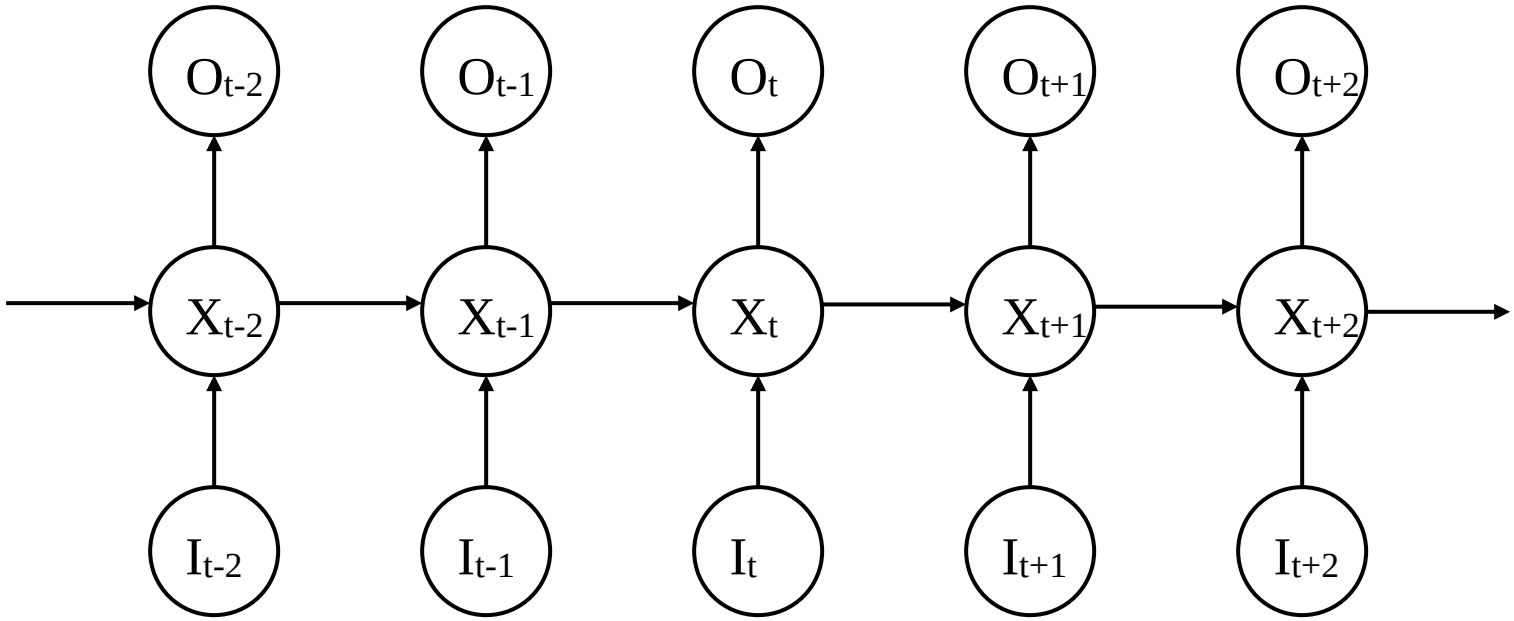


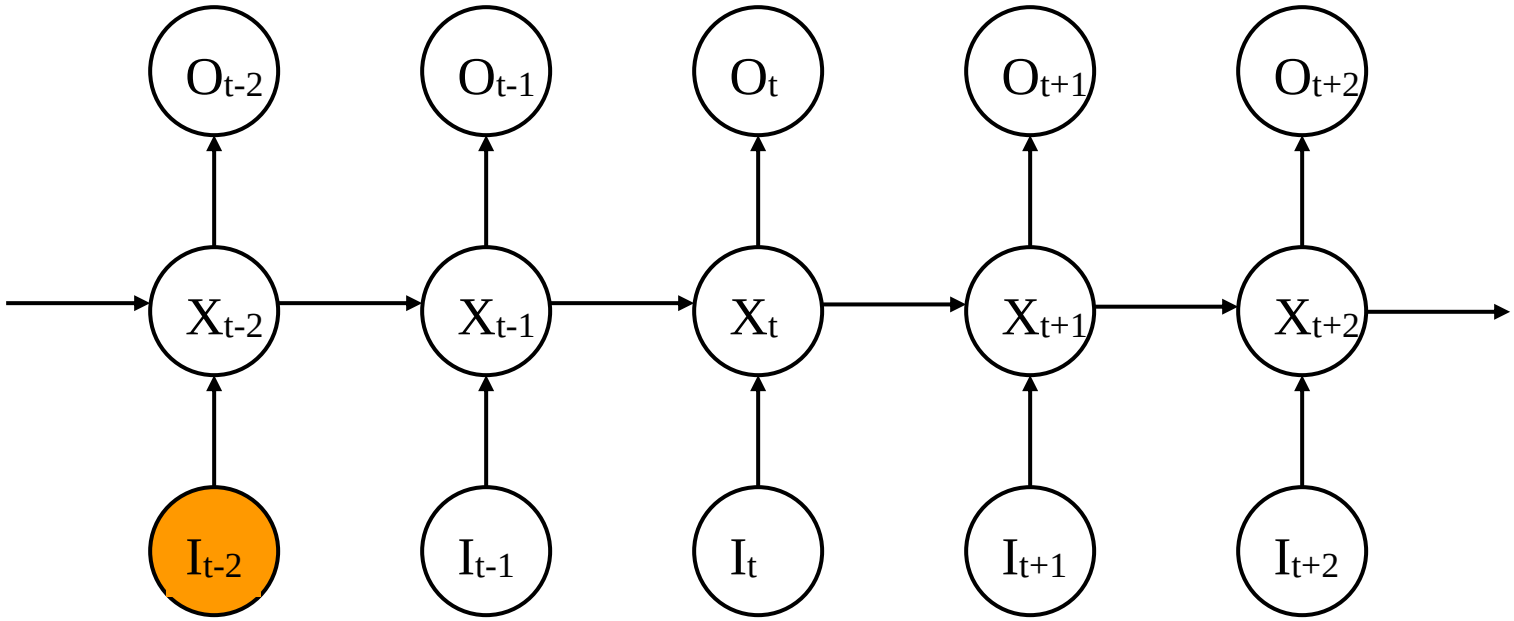
d = delay element

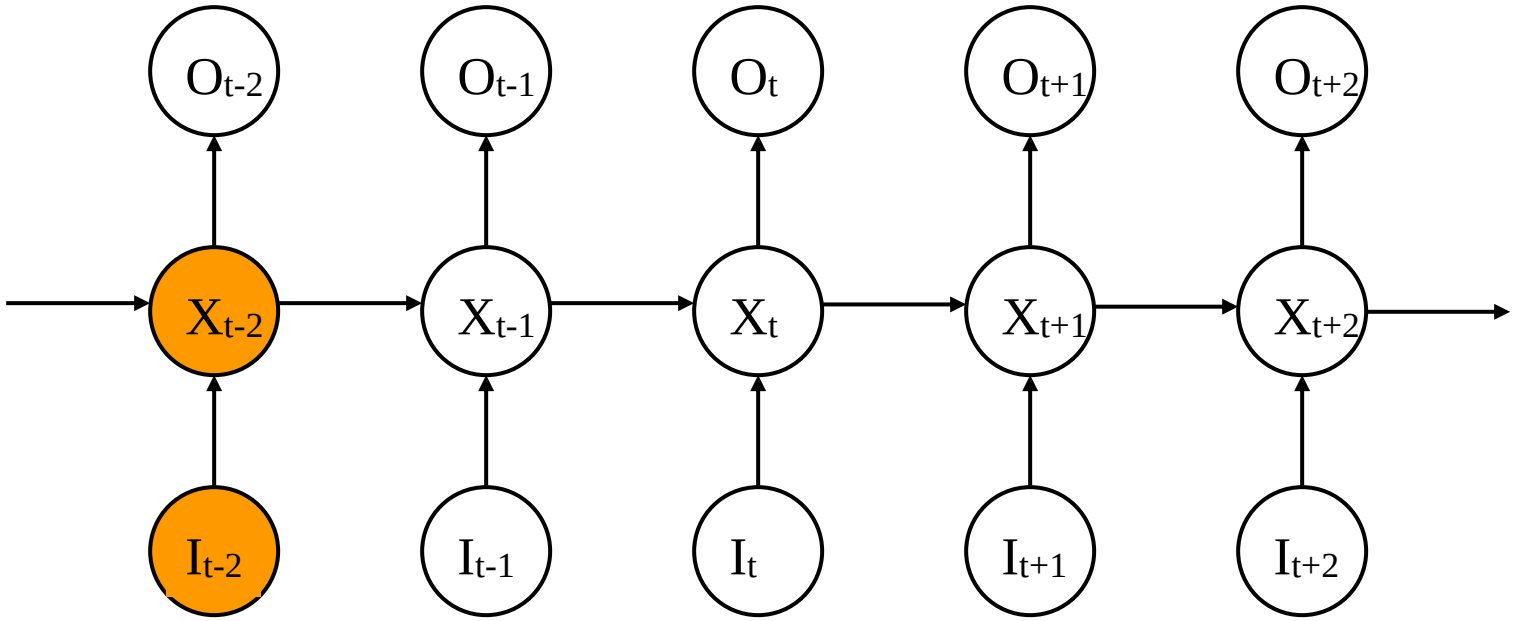

Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2

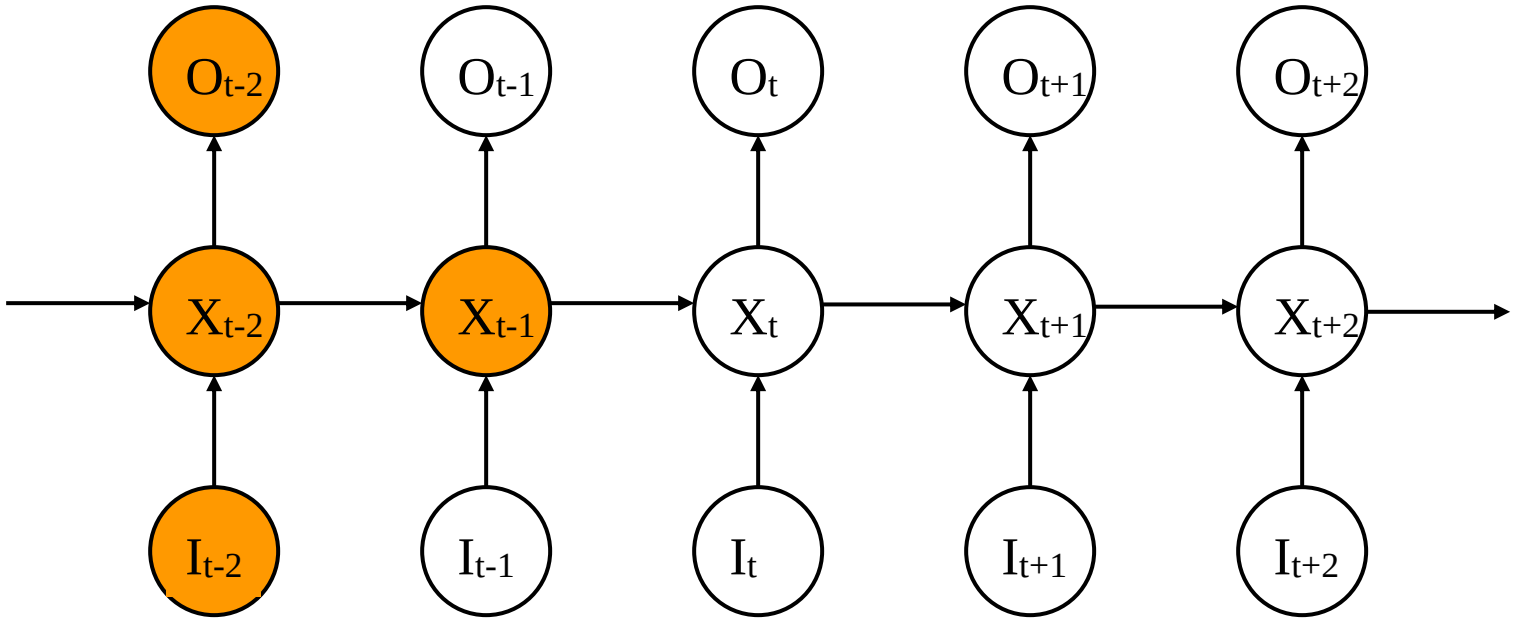
gradient in BPTT

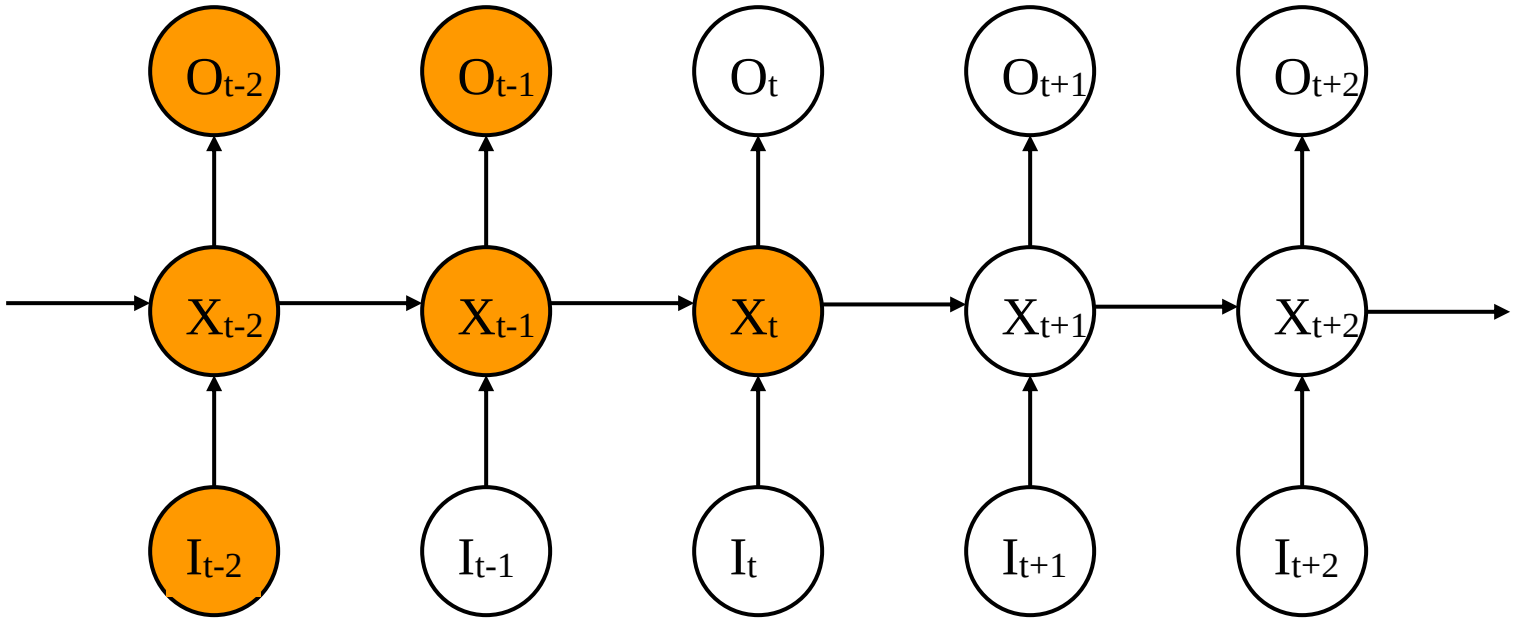
```
GRADIENT(I,O,T) {
# I=inputs, O=outputs, T=targets
T := size(O);
X_0 := 0;
for t := 1..T
    X_t := f( X_t-1 , I_t );
for t := 1..T {
    O_t := g( X_t , I_t );
    g.gradient( O_t - T_t );
    δ_t = g.deltas( O_t - T_t );
}
for t := T..1
    f.gradient( δ_t );
    δ_t-1 += f.deltas( δ_t );
}
```

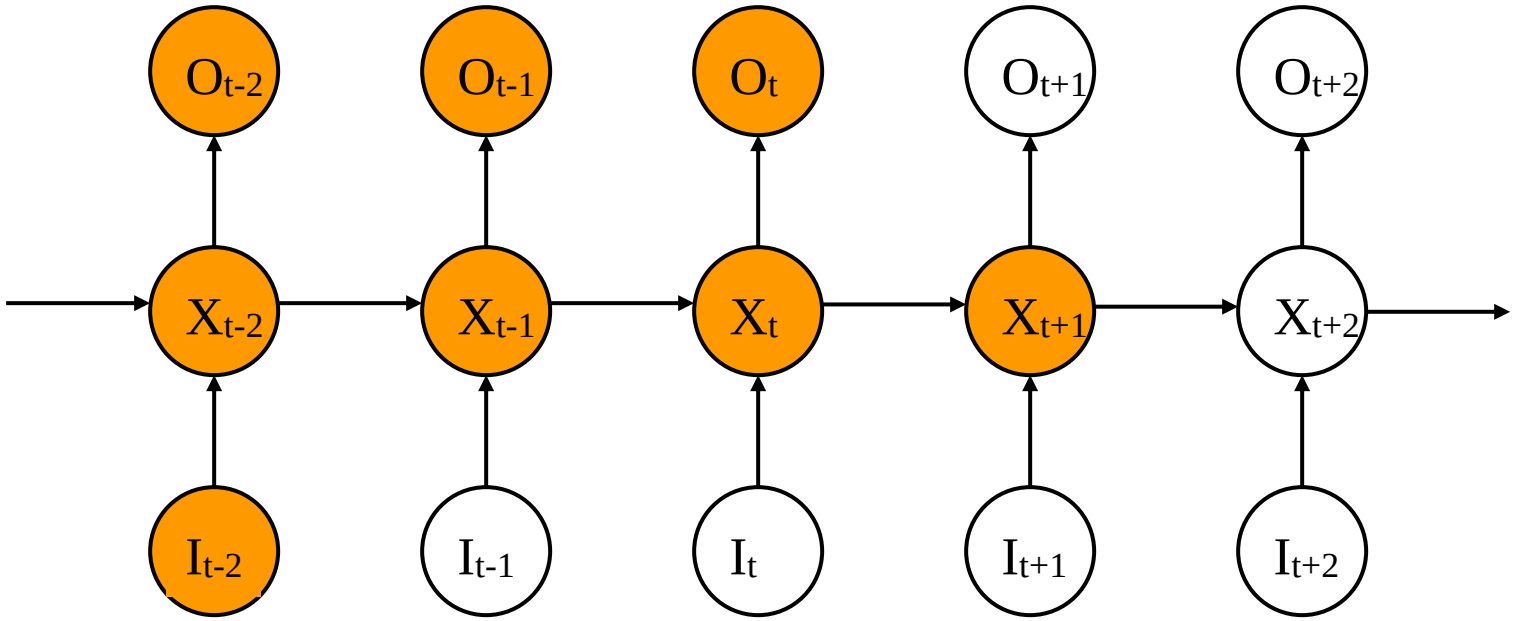

Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2

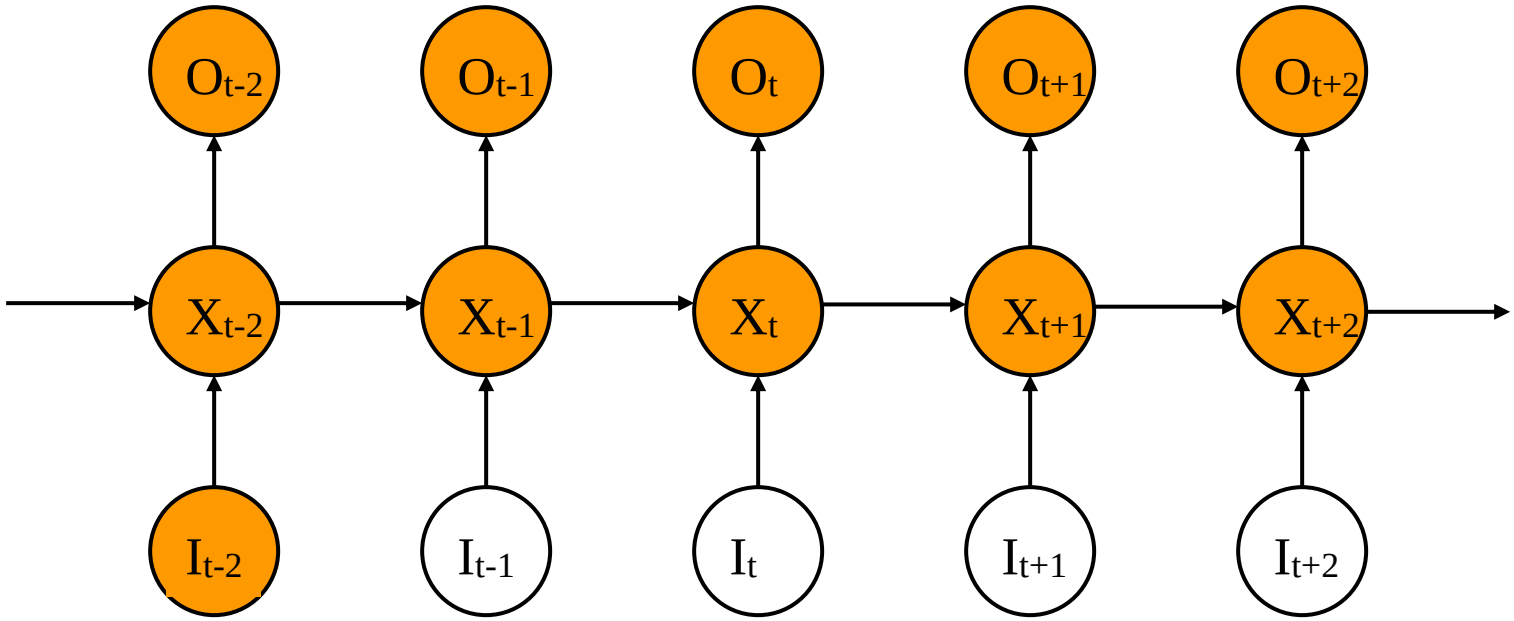

Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2

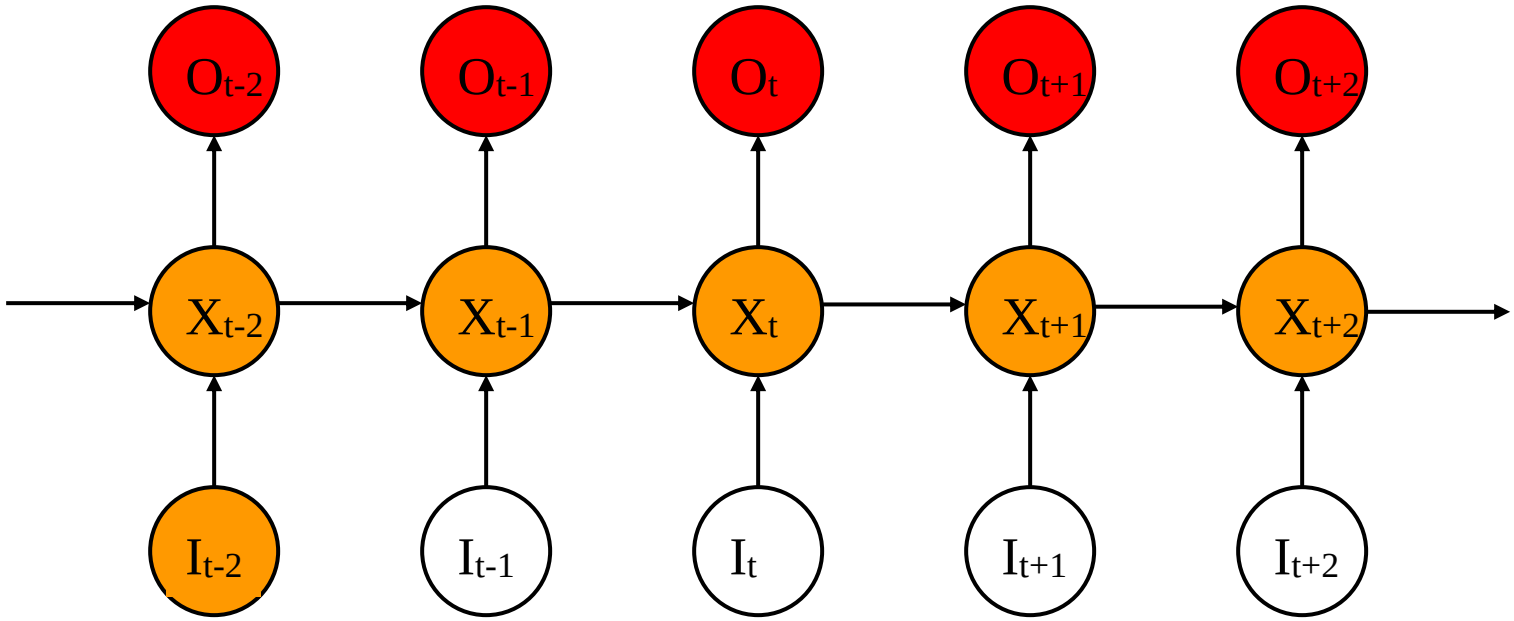

Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2

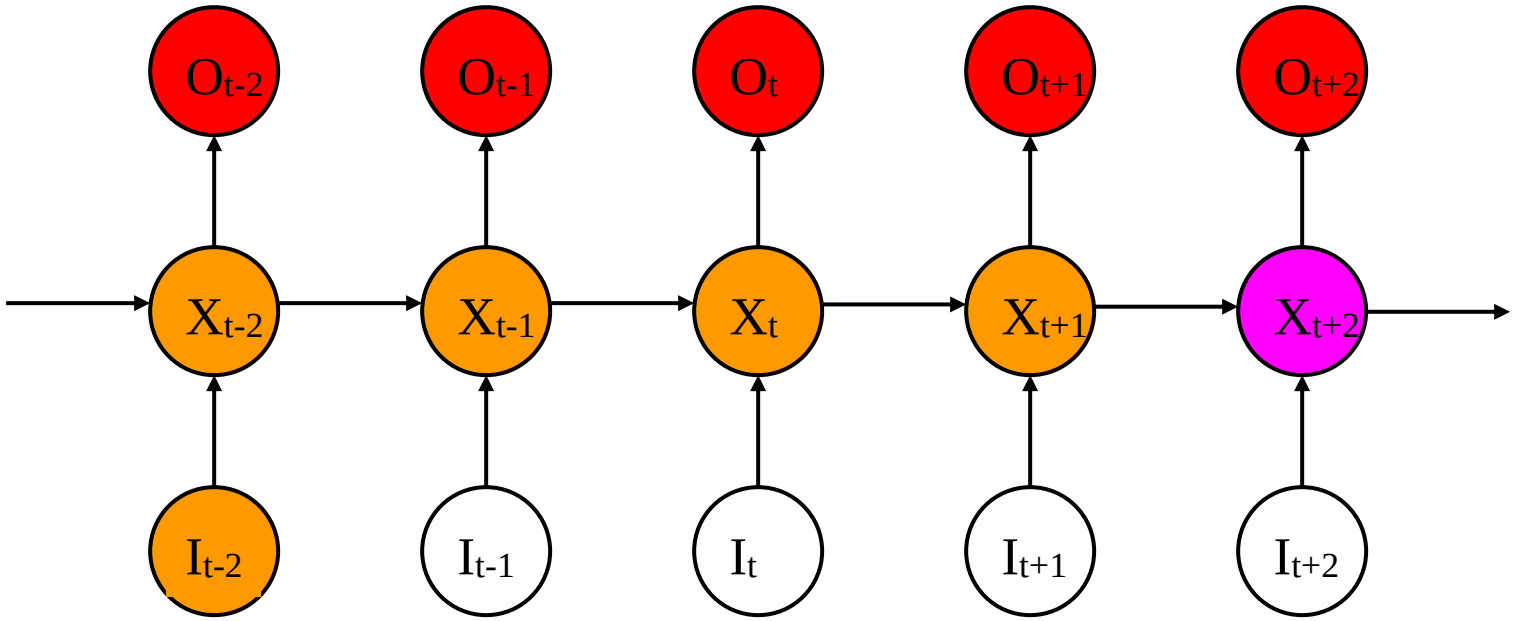

O_{t-2}
O_{t-1}
O_t
O_{t+1}
O_{t+2}
X_{t-2}
X_{t-1}
X_t
X_{t+1}
X_{t+2}
I_{t-2}
I_{t-1}
I_t
I_{t+1}
I_{t+2}

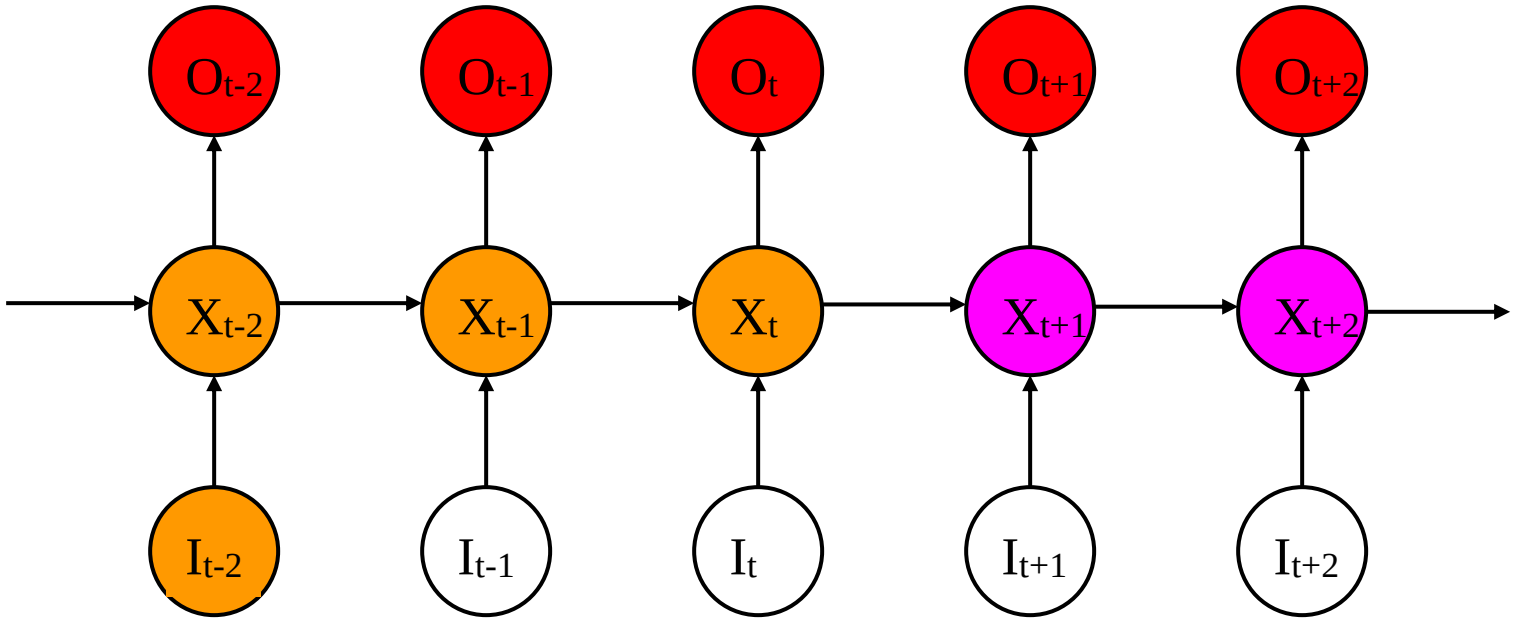

O_{t-2}
O_{t-1}
O_t
O_{t+1}
O_{t+2}
X_{t-2}
X_{t-1}
X_t
X_{t+1}
X_{t+2}
I_{t-2}
I_{t-1}
I_t
I_{t+1}
I_{t+2}

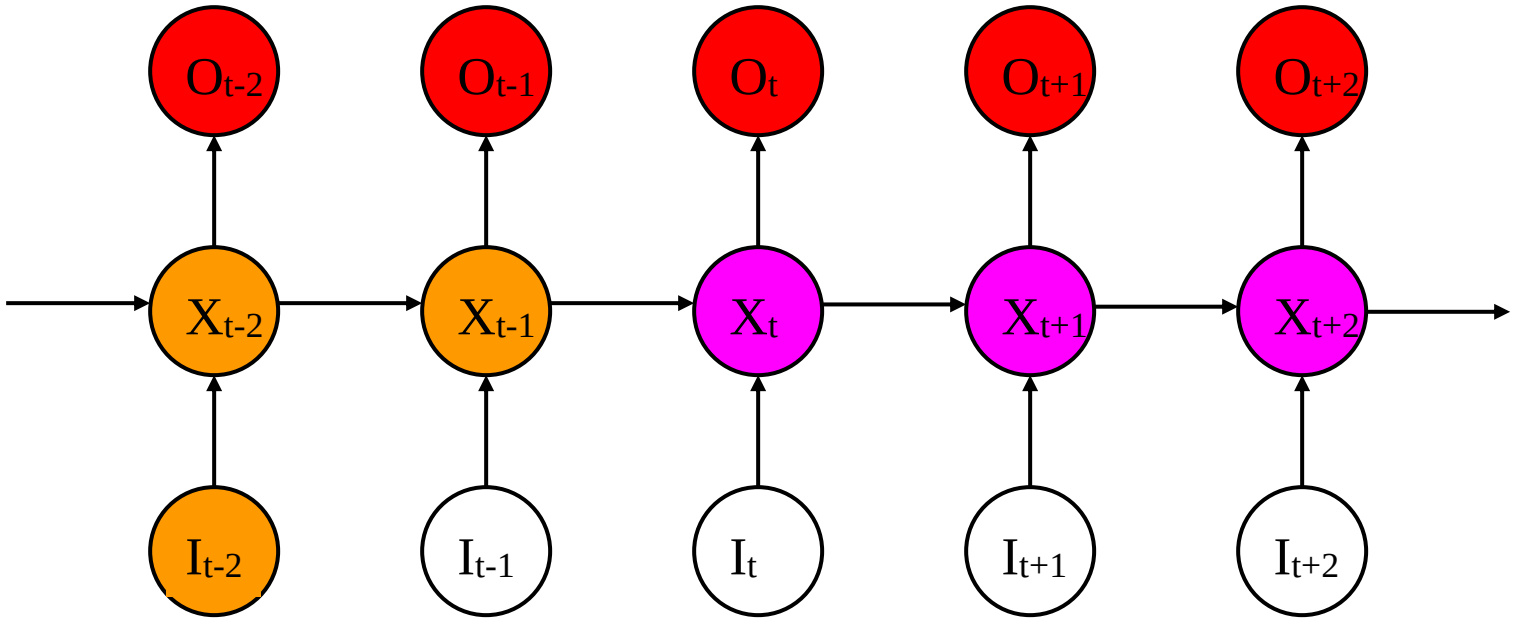

O_{t-2}
O_{t-1}
O_t
O_{t+1}
O_{t+2}
X_{t-2}
X_{t-1}
X_t
X_{t+1}
X_{t+2}
I_{t-2}
I_{t-1}
I_t
I_{t+1}
I_{t+2}

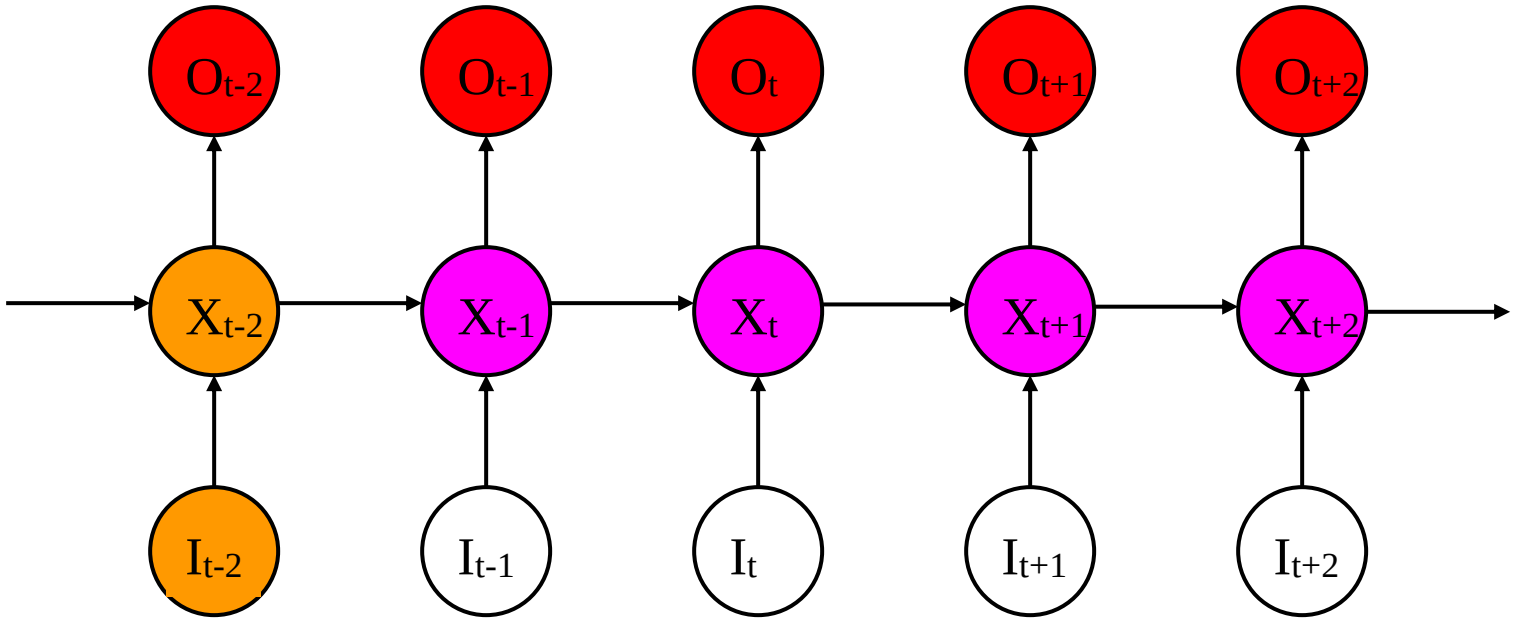

O_{t-2}
O_{t-1}
O_t
O_{t+1}
O_{t+2}
X_{t-2}
X_{t-1}
X_t
X_{t+1}
X_{t+2}
I_{t-2}
I_{t-1}
I_t
I_{t+1}
I_{t+2}

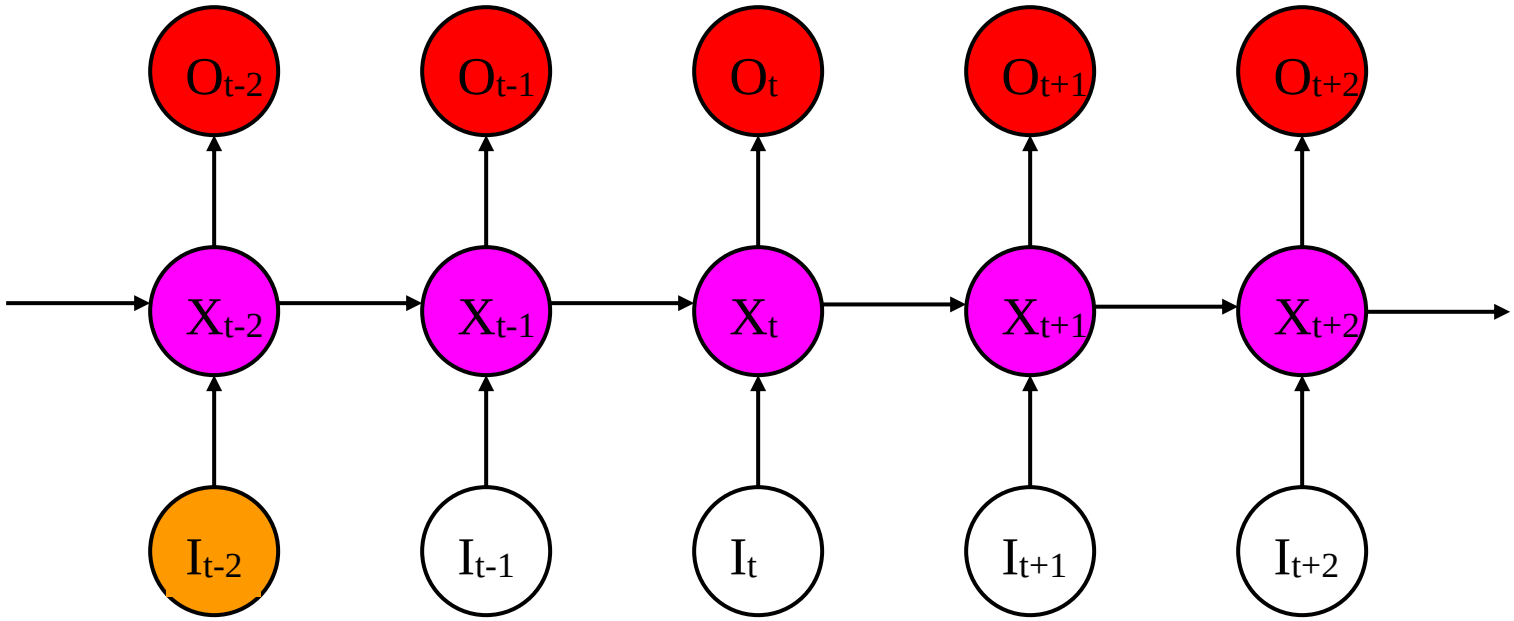

O_{t-2}
O_{t-1}
O_t
O_{t+1}
O_{t+2}
X_{t-2}
X_{t-1}
X_t
X_{t+1}
X_{t+2}
I_{t-2}
I_{t-1}
I_t
I_{t+1}
I_{t+2}


O_{t-2}
O_{t-1}
O_t
O_{t+1}
O_{t+2}
X_{t-2}
X_{t-1}
X_t
X_{t+1}
X_{t+2}
I_{t-2}
I_{t-1}
I_t
I_{t+1}
I_{t+2}


Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2


Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2


Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2


Ot-2
Ot-1
Ot
Ot+1
Ot+2
Xt-2
Xt-1
Xt
Xt+1
Xt+2
It-2
It-1
It
It+1
It+2

Past and future

- **RNN are OK (ish) for time dependency.**
- **E.g. one would expect the past text to be more important than the future text to interpret language (though a little look-ahead is necessary).**
- **But what about sequences in space?**
- **With a time metaphor, you'll need to know the past and the future.**

BIOINFORMATICS

Vol. 15 no. 11 1999
Pages 937–946

Exploiting the past and the future in protein secondary structure prediction

Pierre Baldi[1,*], Søren Brunak[2], Paolo Frasconi[3], Giovanni Soda[3] and Gianluca Pollastri[4]

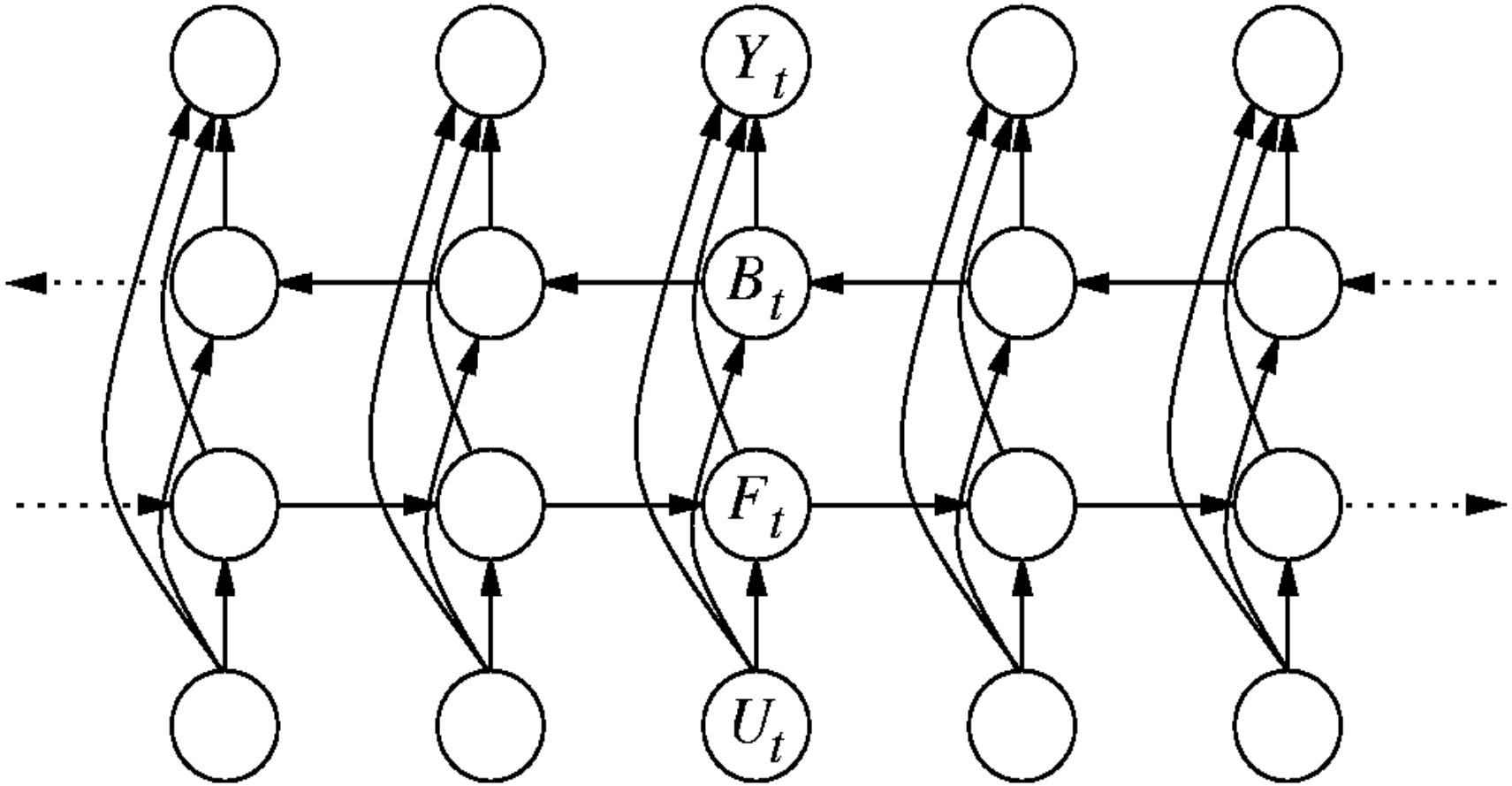
[1]Department of Information and Computer Science, and Department of Biological Chemistry, College of Medicine, University of California, Irvine, Irvine, CA 92697-3425, USA, [2]Center for Biological Sequence Analysis, The Technical University of Denmark, DK-2800 Lyngby, Denmark, [3]Department of Informatics and Systems, University of Florence, 50139 Florence, Italy and [4]Department of Information and Computer Science, University of California, Irvine, Irvine, CA 92697-3425, USA

Abstract

Motivation: *Predicting the secondary structure of a protein (alpha-helix, beta-sheet, coil) is an important step towards elucidating its three-dimensional structure, as well as its function. Presently, the best predictors are based on machine learning approaches, in particular neural network architectures with a fixed, and relatively short, input window of amino acids, centered at the prediction site. Although a fixed small window avoids overfitting problems, it does not permit capturing variable*

as a result of genome and other sequencing projects. One significant step towards elucidating the structure and function of a protein is the prediction of its secondary structure (SS). The SS consists of local folding regularities maintained by hydrogen bonds and traditionally subdivided into three classes: alpha-helices, beta-sheets and coils representing all the rest. In alpha-helices, backbone hydrogen bonds link residues i and $i+4$, whereas in beta-sheets, hydrogen bonds link two sequence segments, in either parallel or antiparallel fashion. The SS can be

Bidirectional Recurrent Neural Networks (BRNN)

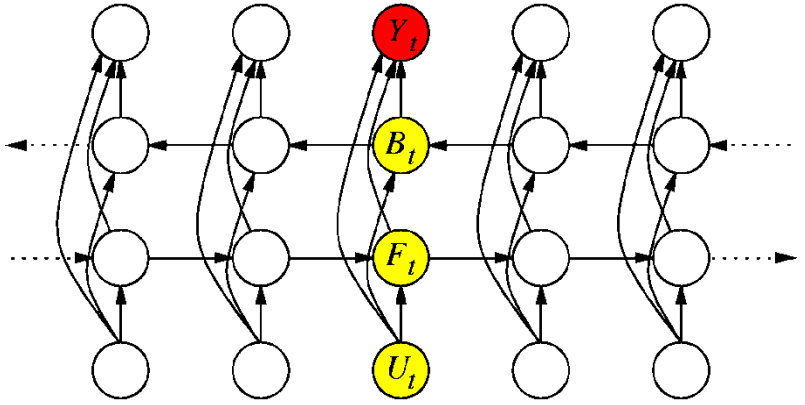


BRNN

$F_t = \phi(F_{t-1} , U_t)$

$B_t = \beta(B_{t+1} , U_t)$

$Y_t = \eta(F_t , B_t , U_t)$



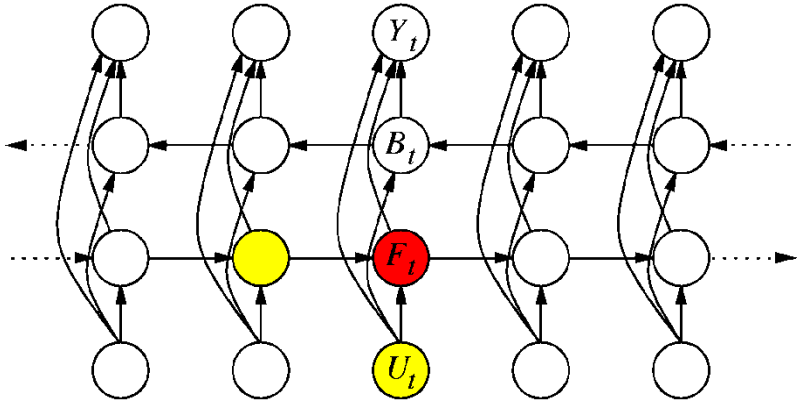
- $\phi()$ $\beta()$ ed $\eta()$ are realised with NN
- $\phi()$, $\beta()$ and $\eta()$ are independent from t: stationary

BRNN

$F_t = \phi(F_{t-1} , U_t)$

$B_t = \beta(B_{t+1} , U_t)$

$Y_t = \eta(F_t , B_t , U_t)$



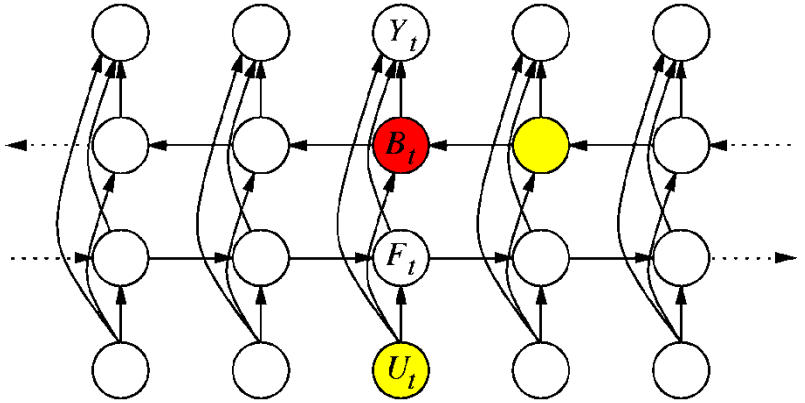
- $\phi()$ $\beta()$ ed $\eta()$ are realised with NN
- $\phi()$, $\beta()$ and $\eta()$ are independent from t: stationary

BRNN

$F_t = \phi(F_{t-1} , U_t)$

$B_t = \beta(B_{t+1} , U_t)$

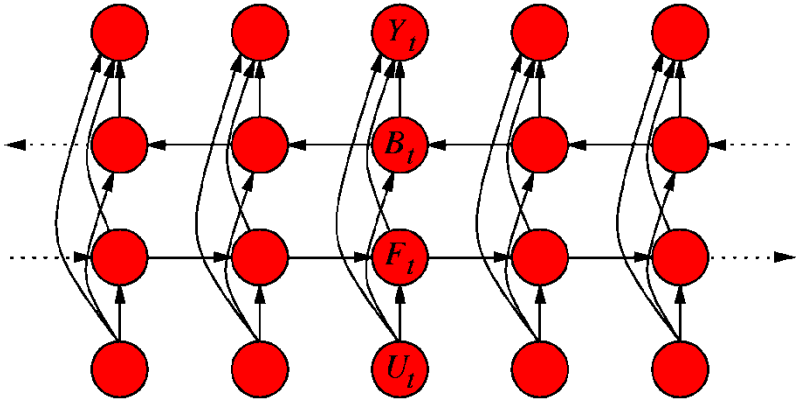
$Y_t = \eta(F_t , B_t , U_t)$



- $\phi()$ $\beta()$ ed $\eta()$ are realised with NN
- $\phi()$, $\beta()$ and $\eta()$ are independent from t: stationary

Inference in BRNNs

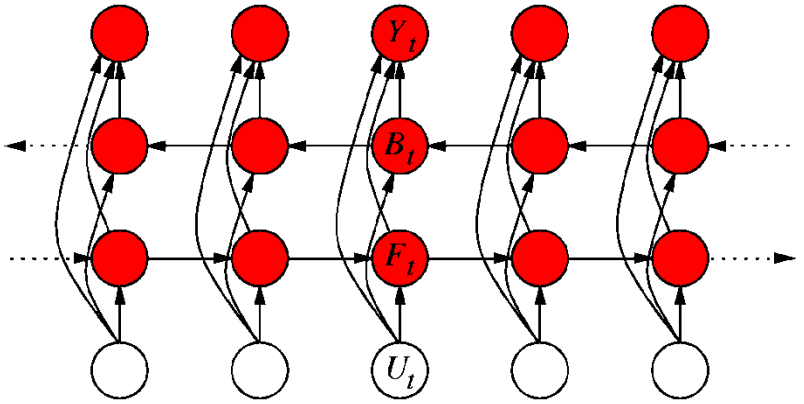
- FORWARD(U) {
- $T \leftarrow$ size(U);
- $F_0 \leftarrow B_{T+1} \leftarrow 0$;
- for $t \leftarrow 1..T$
- $\quad F_t = \phi(F_{t-1} , U_t)$;
- for $t \leftarrow T..1$
- $\quad B_t = \beta(B_{t+1} , U_t)$;
- for $t \leftarrow 1..T$
- $\quad Y_t = \eta(F_t , B_t , U_t)$;
- return Y;
- }



Learning in BRNNs

- GRADIENT(U,Y) {
- $T \leftarrow$ size(U);
- $F_0 \leftarrow B_{T+1} \leftarrow 0$;
- for $t \leftarrow 1..T$
- $F_t = \phi(F_{t-1} , U_t)$;
- for $t \leftarrow T..1$
- $B_t = \beta(B_{t+1} , U_t)$;
- for $t \leftarrow 1..T$ {
- $Y_t = \eta(F_t , B_t , U_t)$;
- $[\delta_{F_t}, \delta_{B_t}] = \eta$.backprop&gradient($Y_t - Y_t$);
- }

- for $t \leftarrow T..1$
- $\delta_{F_{t-1}} += \phi$.backprop&gradient(δ_{F_t});
- for $t \leftarrow 1..T$
- $\delta_{B_{t+1}} += \beta$.backprop&gradient($\delta_{B_t}$);
- }



What's good with BRNN

- They find the ideal "window size" by themselves. In theory they see the whole input.
- They are DEEP (or, most paths are). Which means they are clever.

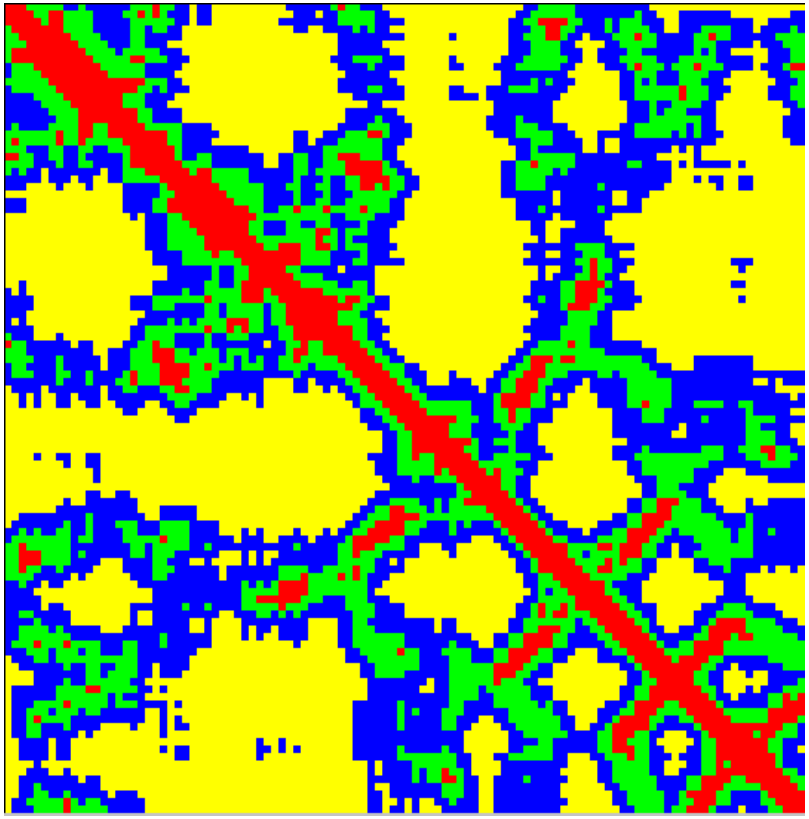
- They have proven to be one of the best models for processing biological sequences.

Once started..

- **Sequences are dealt with.**
- **But structured problems do not stop there.**
- **In general one would like to be able to deal with graphs of any type (even undirected ones containing cycles).**
- **Sequences are 1D entities. But what about 2D? E.g. images?**

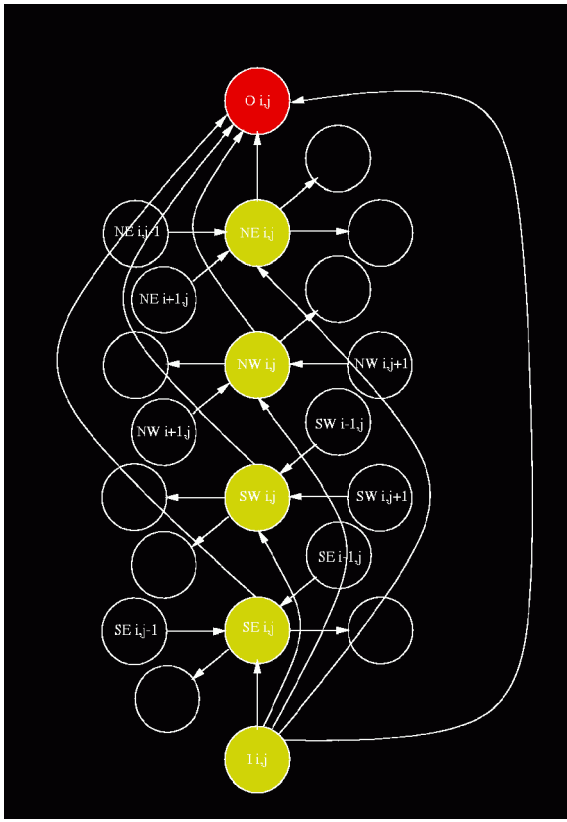
Example: Distance maps

3D



2D RNNs

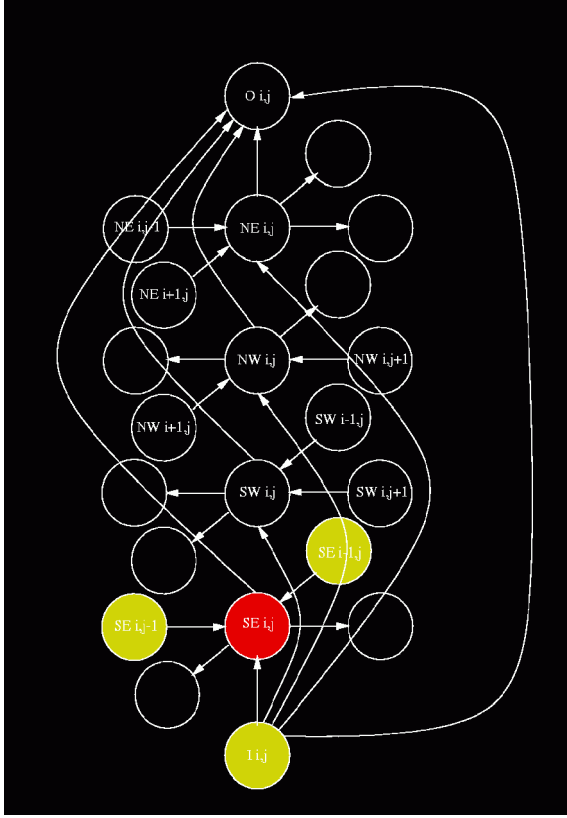
$$\begin{cases} O_{ij} = \mathcal{N}_O(I_{ij}, H^{NW}_{i,j}, H^{NE}_{i,j}, H^{SW}_{i,j}, H^{SE}_{i,j,}) \\ H^{NE}_{i,j} = \mathcal{N}_{NE}(I_{i,j}, H^{NE}_{i-1,j}, H^{NE}_{i,j-1}) \\ H^{NW}_{i,j} = \mathcal{N}_{NW}(I_{i,j}, H^{NW}_{i+1,j}, H^{NW}_{i,j-1}) \\ H^{SW}_{i,j} = \mathcal{N}_{SW}(I_{i,j}, H^{SW}_{i+1,j}, H^{SW}_{i,j+1}) \\ H^{SE}_{i,j} = \mathcal{N}_{SE}(I_{i,j}, H^{SE}_{i-1,j}, H^{SE}_{i,j+1}) \end{cases}$$



Pollastri & Baldi 2002, *Bioinformatics*
Baldi & Pollastri 2003, *JMLR*

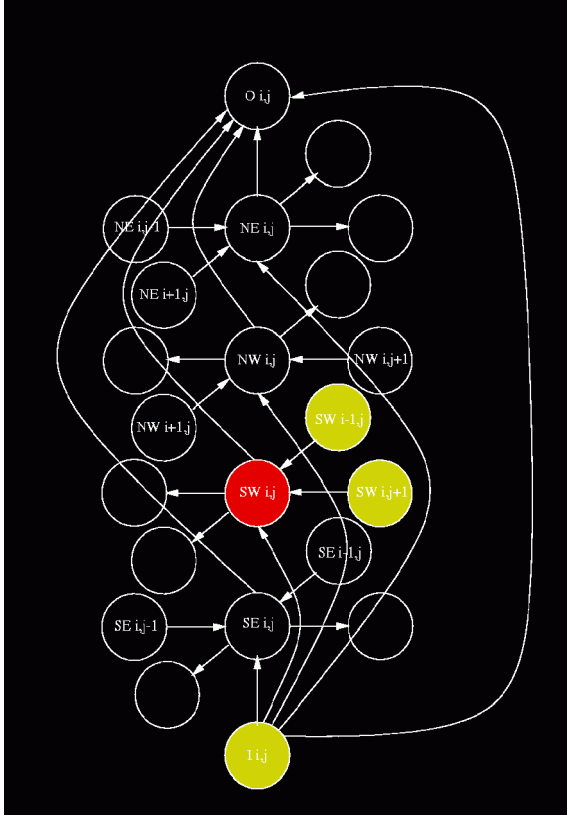
2D RNNs

$$
\begin{cases}
O_{ij} = \mathcal{N}_O(I_{ij}, H^{NW}_{i,j}, H^{NE}_{i,j}, H^{SW}_{i,j}, H^{SE}_{i,j,}) \\
H^{NE}_{i,j} = \mathcal{N}_{NE}(I_{i,j}, H^{NE}_{i-1,j}, H^{NE}_{i,j-1}) \\
H^{NW}_{i,j} = \mathcal{N}_{NW}(I_{i,j}, H^{NW}_{i+1,j}, H^{NW}_{i,j-1}) \\
H^{SW}_{i,j} = \mathcal{N}_{SW}(I_{i,j}, H^{SW}_{i+1,j}, H^{SW}_{i,j+1}) \\
H^{SE}_{i,j} = \mathcal{N}_{SE}(I_{i,j}, H^{SE}_{i-1,j}, H^{SE}_{i,j+1})
\end{cases}
$$



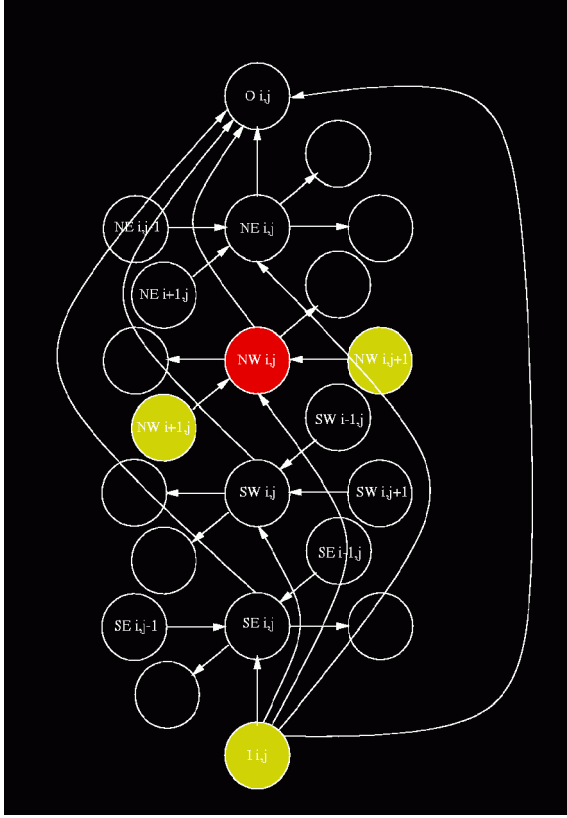
2D RNNs

$$
\begin{cases}
O_{ij} = \mathcal{N}_O(I_{ij}, H^{NW}_{i,j}, H^{NE}_{i,j}, H^{SW}_{i,j}, H^{SE}_{i,j,}) \\
H^{NE}_{i,j} = \mathcal{N}_{NE}(I_{i,j}, H^{NE}_{i-1,j}, H^{NE}_{i,j-1}) \\
H^{NW}_{i,j} = \mathcal{N}_{NW}(I_{i,j}, H^{NW}_{i+1,j}, H^{NW}_{i,j-1}) \\
H^{SW}_{i,j} = \mathcal{N}_{SW}(I_{i,j}, H^{SW}_{i+1,j}, H^{SW}_{i,j+1}) \\
H^{SE}_{i,j} = \mathcal{N}_{SE}(I_{i,j}, H^{SE}_{i-1,j}, H^{SE}_{i,j+1})
\end{cases}
$$



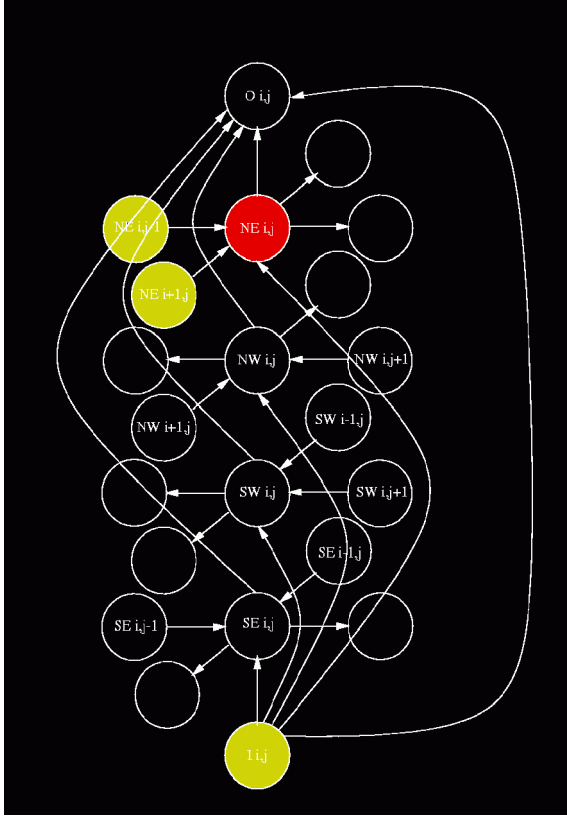
2D RNNs

$$
\begin{cases}
O_{ij} = \mathcal{N}_O(I_{ij}, H^{NW}_{i,j}, H^{NE}_{i,j}, H^{SW}_{i,j}, H^{SE}_{i,j,}) \\
H^{NE}_{i,j} = \mathcal{N}_{NE}(I_{i,j}, H^{NE}_{i-1,j}, H^{NE}_{i,j-1}) \\
H^{NW}_{i,j} = \mathcal{N}_{NW}(I_{i,j}, H^{NW}_{i+1,j}, H^{NW}_{i,j-1}) \\
H^{SW}_{i,j} = \mathcal{N}_{SW}(I_{i,j}, H^{SW}_{i+1,j}, H^{SW}_{i,j+1}) \\
H^{SE}_{i,j} = \mathcal{N}_{SE}(I_{i,j}, H^{SE}_{i-1,j}, H^{SE}_{i,j+1})
\end{cases}
$$



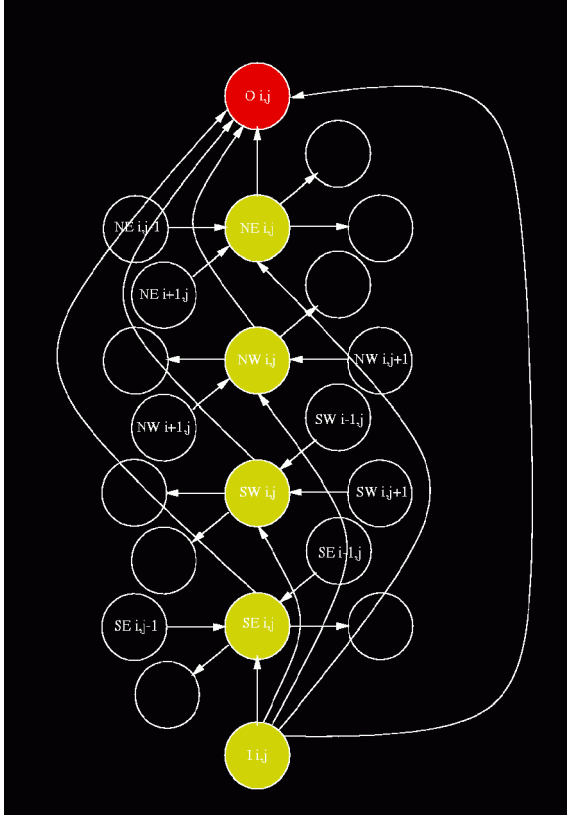
2D RNNs

$$
\begin{cases}
O_{ij} = \mathcal{N}_O(I_{ij}, H^{NW}_{i,j}, H^{NE}_{i,j}, H^{SW}_{i,j}, H^{SE}_{i,j,}) \\
H^{NE}_{i,j} = \mathcal{N}_{NE}(I_{i,j}, H^{NE}_{i-1,j}, H^{NE}_{i,j-1}) \\
H^{NW}_{i,j} = \mathcal{N}_{NW}(I_{i,j}, H^{NW}_{i+1,j}, H^{NW}_{i,j-1}) \\
H^{SW}_{i,j} = \mathcal{N}_{SW}(I_{i,j}, H^{SW}_{i+1,j}, H^{SW}_{i,j+1}) \\
H^{SE}_{i,j} = \mathcal{N}_{SE}(I_{i,j}, H^{SE}_{i-1,j}, H^{SE}_{i,j+1})
\end{cases}
$$



2D RNNs

$$
\begin{cases}
O_{ij} = \mathcal{N}_O(I_{ij}, H^{NW}_{i,j}, H^{NE}_{i,j}, H^{SW}_{i,j}, H^{SE}_{i,j,}) \\
H^{NE}_{i,j} = \mathcal{N}_{NE}(I_{i,j}, H^{NE}_{i-1,j}, H^{NE}_{i,j-1}) \\
H^{NW}_{i,j} = \mathcal{N}_{NW}(I_{i,j}, H^{NW}_{i+1,j}, H^{NW}_{i,j-1}) \\
H^{SW}_{i,j} = \mathcal{N}_{SW}(I_{i,j}, H^{SW}_{i+1,j}, H^{SW}_{i,j+1}) \\
H^{SE}_{i,j} = \mathcal{N}_{SE}(I_{i,j}, H^{SE}_{i-1,j}, H^{SE}_{i,j+1})
\end{cases}
$$



2D-RNNs

- **Excellent model, very DEEP!**
- **Top results where applied.**
- **Finds incredibly long-ranged dependencies.**

Complexity?

- A 2D-RNN contains 5xNxN individual neural networks.
- For a 1024x1024 pixel image that is 5+ million nets.
- Most synapses are shared, so there isn't an overfitting problem.
- But it's tens of billions of weights to train.
- Months of 1-core training, in some cases.

N-to-1

RPYACPVESCDRRFSQSGSLTRHIRIHTGQKPFQCRICMRNFSRSDHLTTHIRTHTGEKPFACDICGRK

Red/green/blue

Notice: N is variable, which is a problem..

Composition vs. sequence

- **A simple solution would be to look at the frequencies of letters (composition).**
- **All positional information is lost!**

- **Same composition:**
- **AAAAAAAAAACCCCVVVVVVVVVV**
- **AVAVCVAAVCVAAVCVAAVCVAVA**

- **And we know that *motifs* are important in most interesting problems.**

Motifs are hard to deal with

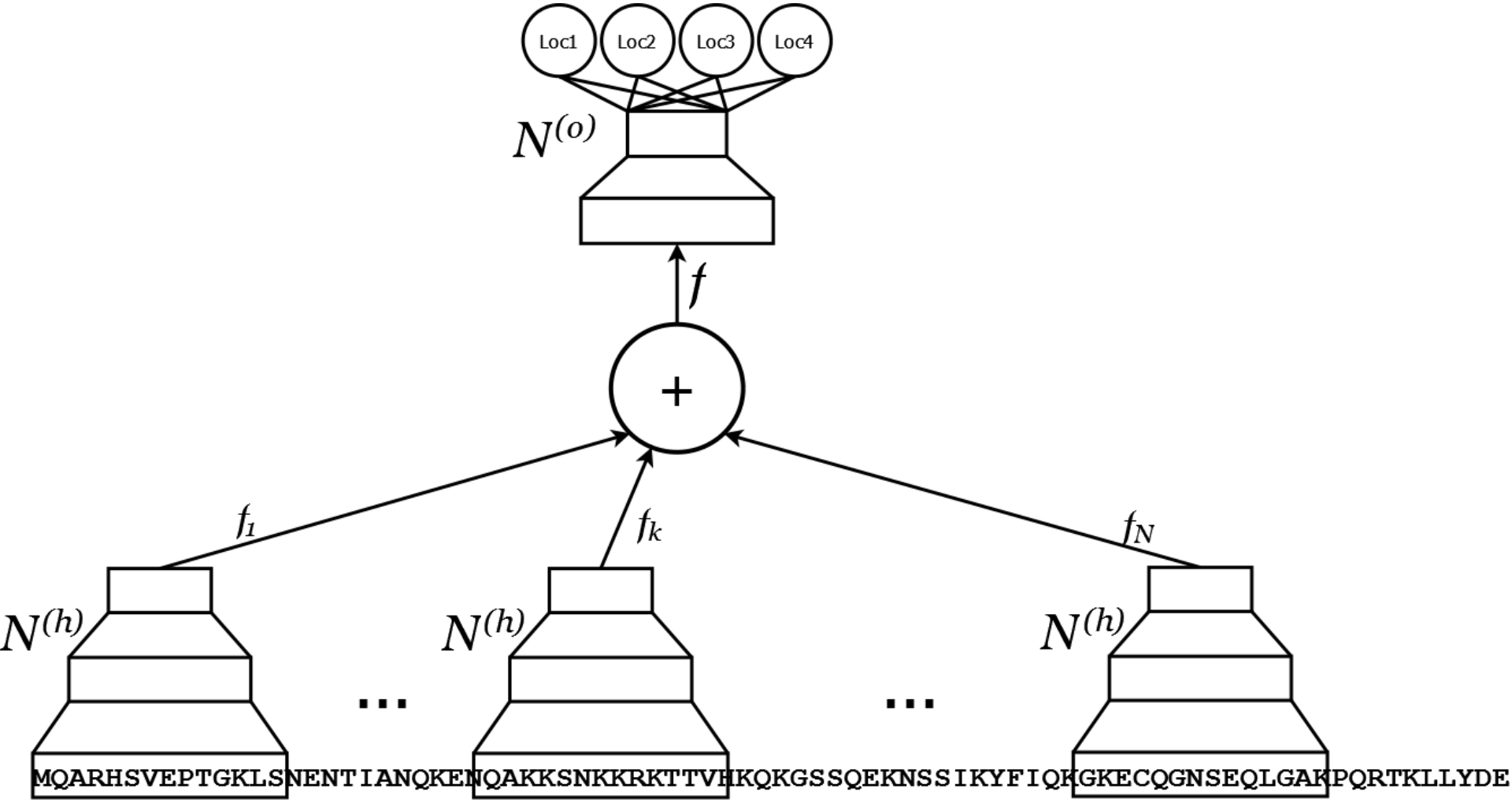
- **Can’t compute stats for all motifs of n letters: there are 20^n of them, and only hundreds of examples.**

- **Tried before. It didn't work even on motifs of 2 or 3 letters, unless one has *millions* of examples.**

Compress!

- **We need to *compress* the representation:**
- **Create a bottleneck: represent zillions of different motifs with only hundreds of parameters.**
- **And we need to do this with an N to 1 wiring.**

N to 1 Neural Networks



Mooney, Wang & Pollastri, *Bioinformatics*, 2011

N-to-1 by neural networks

- **Map W=2c+1 letters into a hidden vector *f*.**
- **Use the same function (network) for each of the N windows in a sequence.**
- **Now we have N hidden vectors.**
- **Just add them up!**

$$f = k \sum_{i=1}^{N} \mathcal{N}^{(h)}(r_{i-c}, \ldots, r_{i+c})$$

N-to-1 neural networks (2)

$$f = k \sum_{i=1}^{N} \mathcal{N}^{(h)}(r_{i-c}, \ldots, r_{i+c})$$

- **f is a vector which contains information about all 2c+1-substrings in a sequence. Say c=7 (15 letters).**
- **Say |f|=3 : ~1000 parameters in total to represent a monster space.**

- **(The trick is that we repeat the same net)**

N-to-1 neural networks (3)

$$f = k \sum_{i=1}^{N} \mathcal{N}^{(h)}(r_{i-c}, \ldots, r_{i+c})$$

$$o = \mathcal{N}^{(o)}(f)$$

- **Map f into output: another net. Now we have a full input-output (N to 1) map**

N-to-1 neural networks (4)

$$f = k \sum_{i=1}^{N} \mathcal{N}^{(h)}(r_{i-c}, \ldots, r_{i+c})$$

$$o = \mathcal{N}^{(o)}(f)$$

- **Training: this is a trivial feed-forward Neural Network (with many shared weights) –backpropagation.**

f

- **Vector *f* is a property-driven, adaptive compression of the whole sequence: fixed size!**
- **What does it tell us about whatever sequences we are dealing with?**

- **Can we explore/map the space of sequences by looking at their *f*?**

N-to-1 NN results

- **Best systems in the world for protein subcellular localisation prediction.**

- **Spare capacity. As sets grow bigger, it may do better and better.**