Biologically Inspired Computation

Dr Marta Vallejo

m.vallejo@hw.ac.uk

Lecture 4

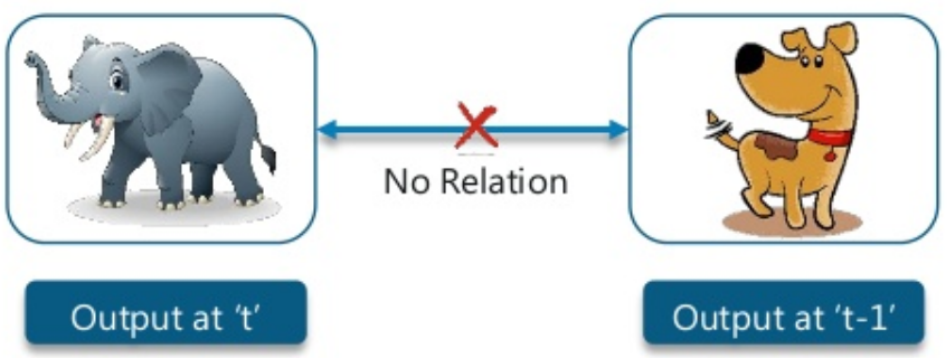
Recurrent Neural Networks

- Limitations of Feed-forward Networks
- Examples of Non-independent Output
- Network Transformation
- Vanilla Recurrent Neural Networks
- Bidirectional Recurrent Neural Network
- Limitations in Recurrent Neural Networks
- Long sort-term Memory
- Gated Recurrent Unit
- Hopfield Neural Networks
  - Neuronal Weights
  - Energy Function
  - Training in Hopfield Networks
  - Applications in Hopfield Networks

Limitations of Feedforward Networks

Feedforward networks are output independent.

If a trained neural network is able to classify an output as an elephant, does not help me to classify the next input as a dog



Some problems require previous history/context in order to be able to give proper output (speech recognition, stock forecasting, target tracking, etc...)

Using a feedforward network I cannot predict the next word of a sentence: it does not include a "context"

Recurrency is highly represented in the nervous system (but feed-forward structures exist too)

Examples of Non-independent Output

Time Series

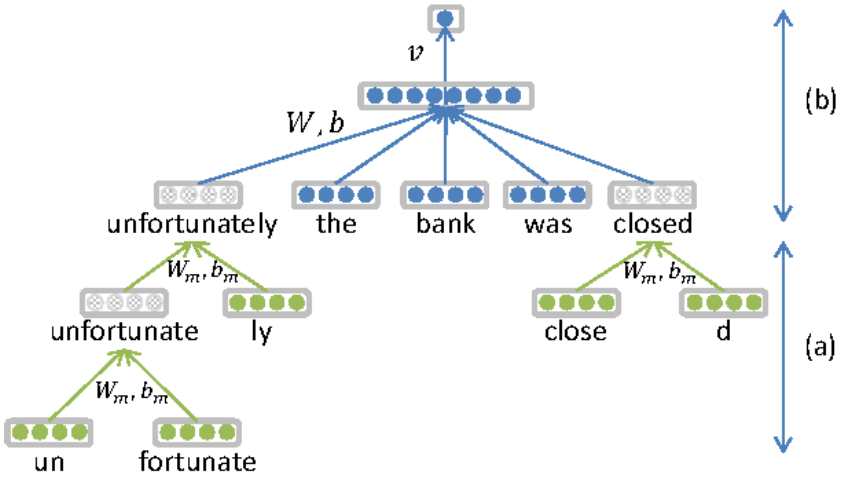
Current target dependent on some combination of current and past inputs

A sequence of data in progressive order, sometimes occurring in fix intervals over a period of time. We want:

- Extract meaningful statistics
- Find hidden patterns
- Predict following values

time	x	*Target*
1	x_1	y_1
2	x_2	y_2
3	x_3	y_3
4	x_4	y_4
5	x_5	y_5
6	x_6	y_6
7	x_7	y_7

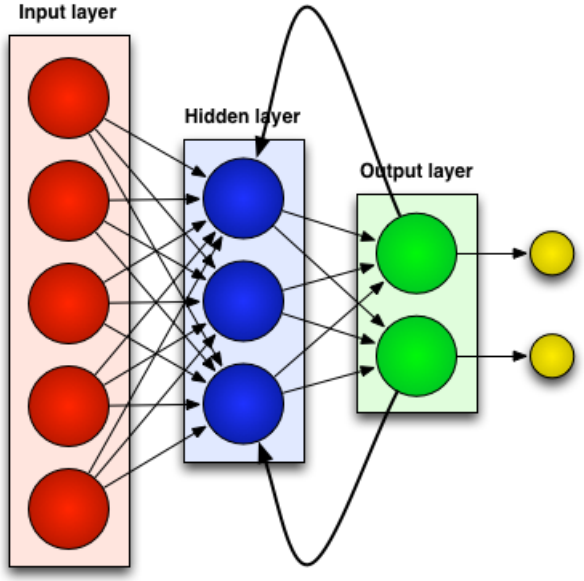
Natural Language Processing/Translation



NLP problems like connected handwriting recognition or speech recognition.

Network transformation

We need to find the mechanisms to gather useful data from previous samples in order to compute properly our new output



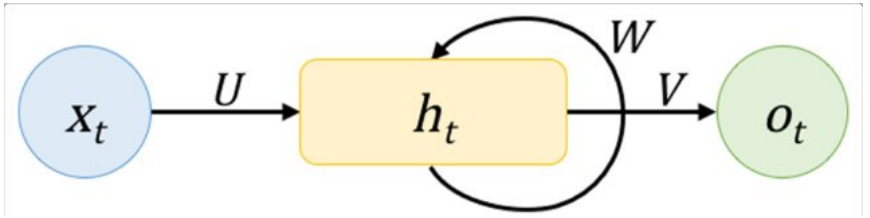
We achieve this behaviour by allowing cycles between units

Recurrent Neural Networks process sequences of data, but to do that, we require to memorise and recall the **memory**.

Past information is implicitly stored in the hidden units called **state vectors.** The output for the current input is computed considering all previous inputs using these state vectors

Vanilla Recurrent Neural Networks

General Characteristics:



- The state is a single hidden vector h
- Multidireccional information
- Keep some sense of time and memory of previous state

We can process a sequence of inputs by applying a recurrence formula at every time step (one time step recurrence):

$$h_t = f_W(h_{t-1}, x_t)$$

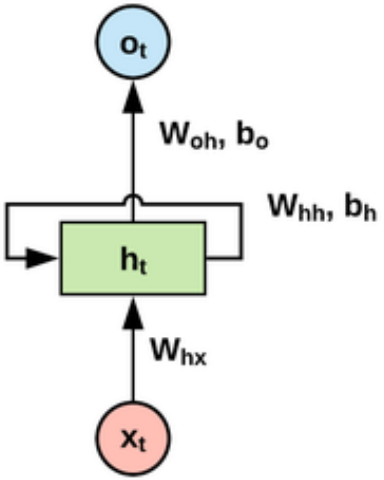
new state

old state

input vector at some time step

some function with parameters W

Basic Structure of the Vanilla RNN



Structure:

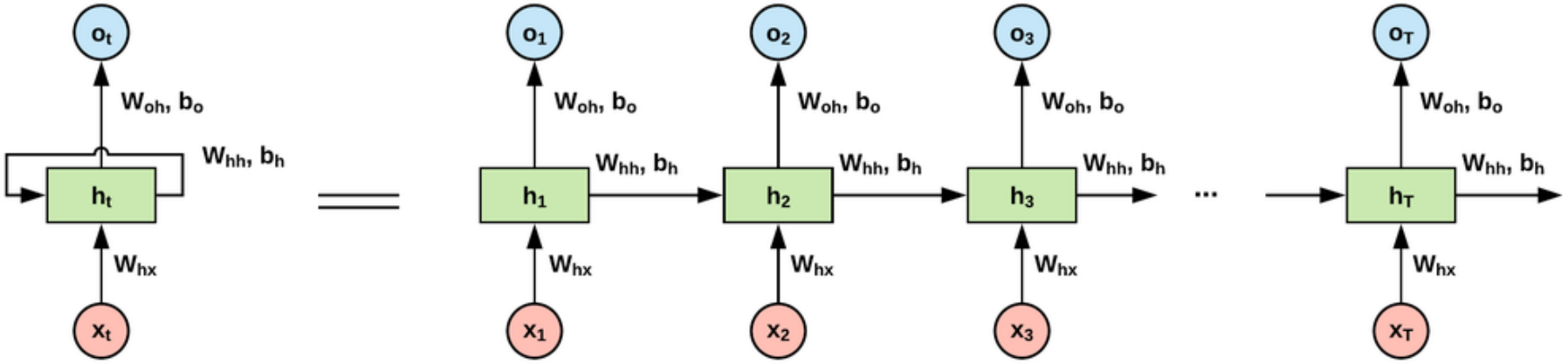
- an input unit x
- a hidden unit h
- an output unit o

The index of each symbol represents the time step

Figure: http://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/vanilla-rnn-for-classification

Basic Structure of the Vanilla RNN

If we unroll/unfold the network:



Unrolling means that we write out the network for the complete sequence

Cyclic connection: the hidden unit receives inputs from the hidden unit at the previous time step and from the input unit at the current time step.

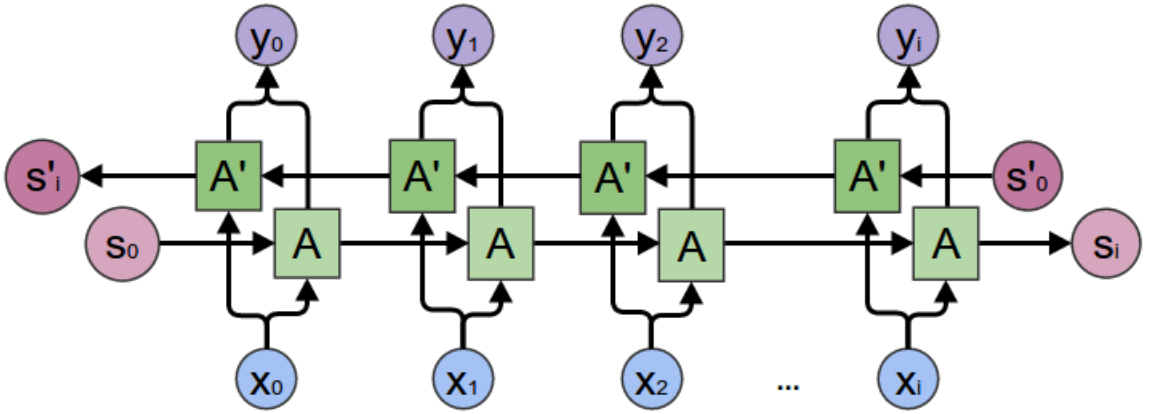
If a RNN is unrolled in time t:

h_t receives input from x_t and h_{t-1} and then propagates to o_t and h_{t+1}

Although RNNs do not seem too deep in terms of the number of layers, they can be regarded as one of the deepest structures if unrolled in time

Bidirectional Recurrent Neural Networks

Often both, past and future inputs, affect the output for the current input. Bidirectional recurrent neural networks have also been designed and used widely for example in speech recognition.

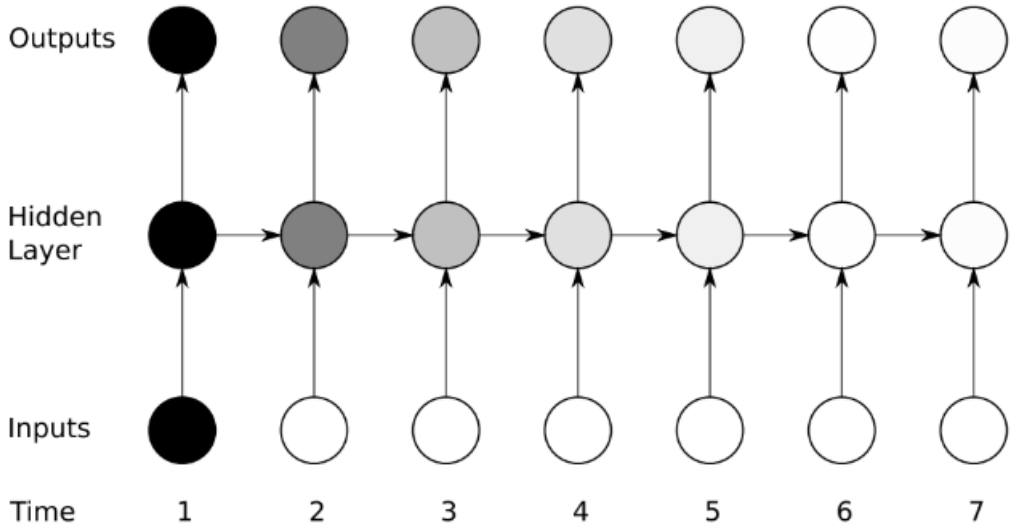


The structure is formed by adding **two independent RNNs together**. The input is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are usually concatenated at each time step (also other options like summation).

This structure allows the networks to have both **backward** and **forward** information about the sequence at every time step.

Limitations in RNNs

- **Problem 1**: Vanishing gradient problems while training: the sensitivity to the input (the darkest, the greatest) decay exponentially over the time.



From: Supervised Sequence Labelling with Recurrent Neural Networks - Alex Graves

It arises If while training with long sequential data, the gradient at early time steps goes to zero (by multiplying too many numbers that are < 1)

Can affect any deep net

- **Problem 2**: Learning long-term dependency among data due to repeated fractional weight multiplication

Limitations in RNNs

Solution:

Substituting simple perceptron hidden units with more complex units:

- Long Short-Term Memory (LSTM).
  Long short-term memory – Hochreiter Schmidhuber (1997)

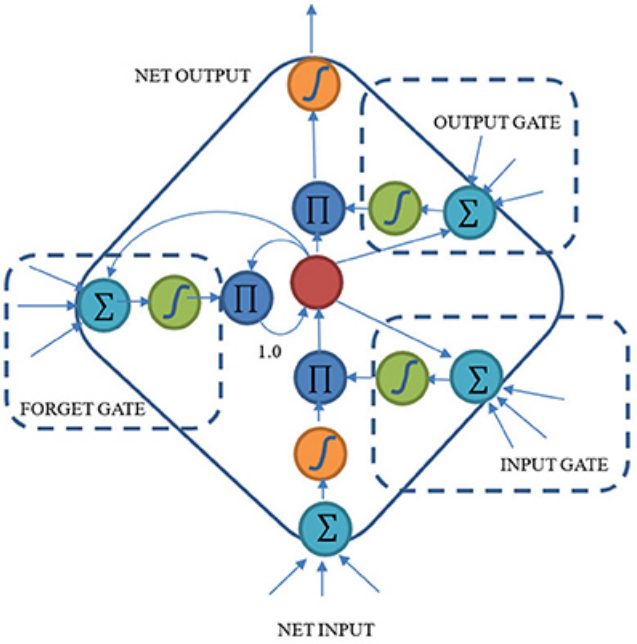
- Gated Recurrent Unit (GRU):
  Learning phrase representations using RNN encoder-decoder for statistical machine translation - Cho, Kyunghyun et al. (2014)

They function as memory cells, which significantly helps to prevent these problems.

RNNs ofter **overperform** their feed-forward counterparts.
They learn much faster and have better generalisation

Long short-term memory

LSTM was developed as an engineering model to solve the vanishing gradient problem



Structure of LSTM block with a single cell.
Peephole connections: from centre (in red) to summation

Structure of LSTM:

Instead of the single activation function, 3 gates governed by sigmoid units that control in/out information

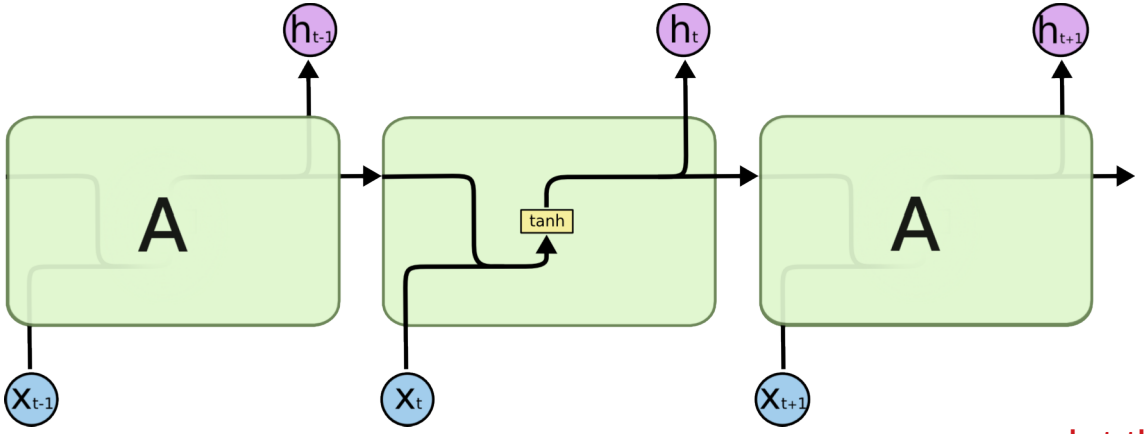
- Input: used to control whether the cell should accept the input information or not.
- Output gates: decides whether the cell should output the contents stored in the cell.
- Forget gate: resets the contents of cells.

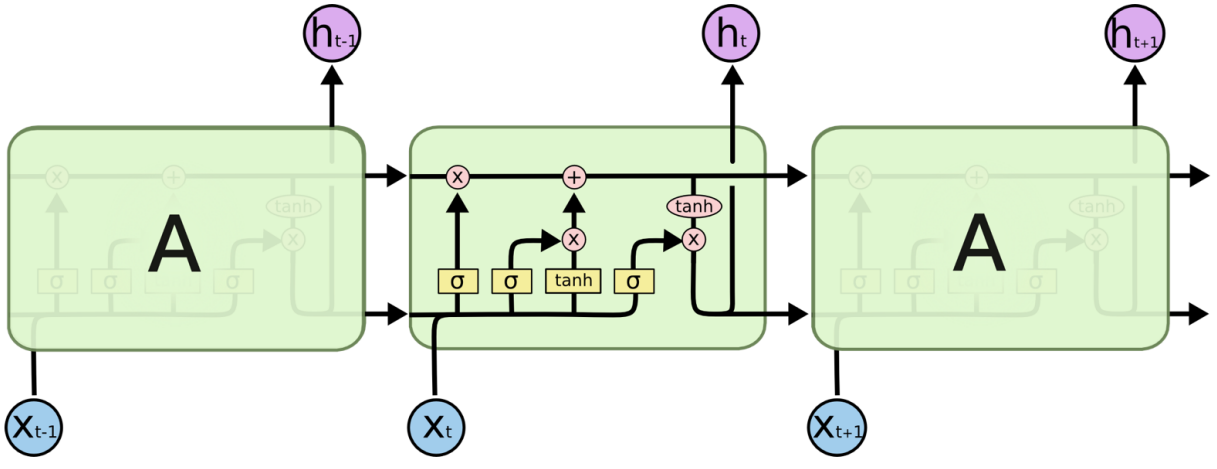
LSTM has the ability to add or remove information via the three gates and each gate learns to do so through backpropagation.

Figure: Yu, Z., Moirangthem, D.S. and Lee, M., 2017. Continuous timescale long-short term memory neural network for human intent understanding.

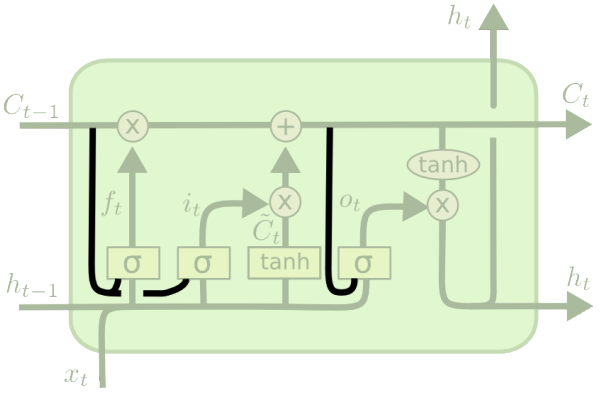
Long short-term memory

Standard RNN





we let the gate layers look at the cell state



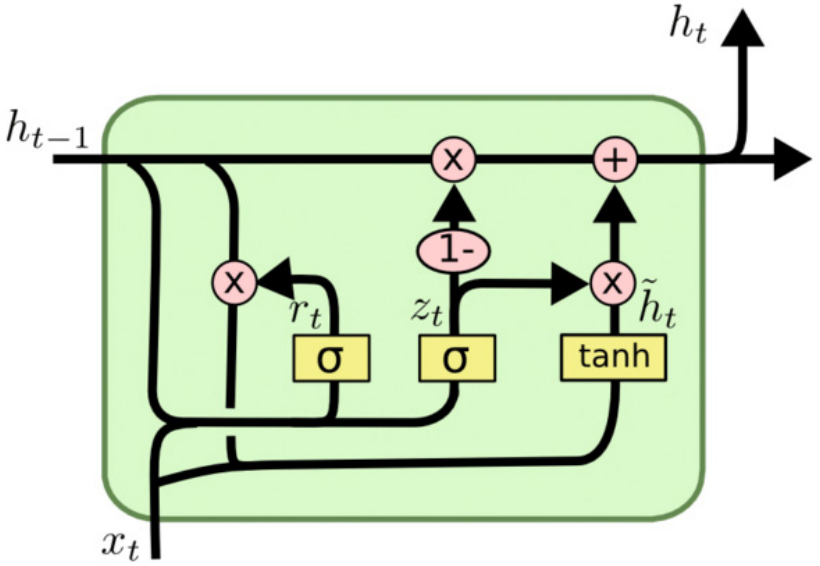
LSTM was further improved adding **peephole connections** to make it possible for the cells to control the time for gate opening inside the block.

Gers, Felix A., and Jürgen Schmidhuber. "Recurrent nets that time and count."

Figures source and article: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit

Similar performance than LSTM but with less computation, due to have fewer number of internal gates. It is therefore faster to train



Structure of GRU:

Similarly gates governed control in/out information. It has only **two** gates:

- Input → called update
- Output → functionality merged in the update gate
- Forget → called reset

For each time step, the GRU first calculates the **reset gate**, which determines the amount of selection for the previous state (if it is going to be ignored), and **update gate** that says how much the unit will be updated.

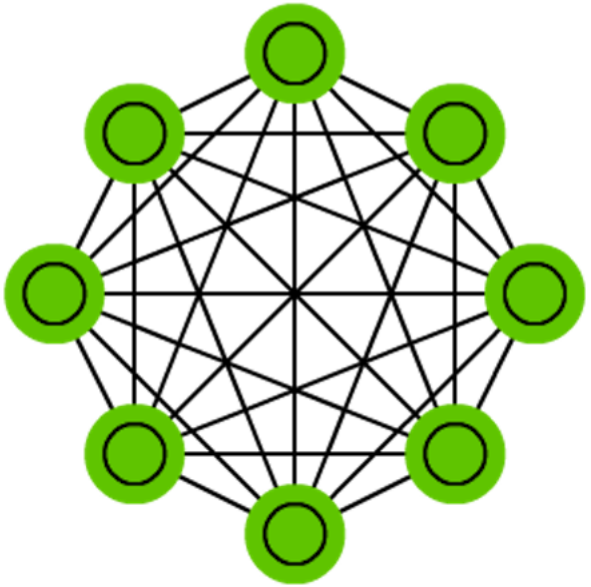
Figure: colah.github.io/posts/2015-08-Understanding-LSTMs/.

GRU vs LSTM

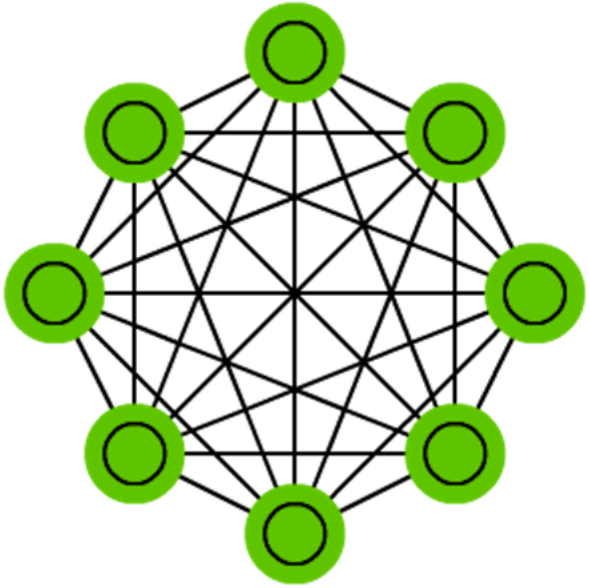
- GRUs train faster and perform better than LSTMs on less training data.
- GRUs are simpler and thus easier to modify, for example adding new gates in case of additional input to the network.
- LSTMs should in theory remember longer sequences than GRUs and outperform them in tasks requiring modelling long-distance relations.
- For huge datasets and if time is not a constrain, **LSTM may be better** in the long run due to its greater complexity - especially if you add peephole connections.

Hopfield Neural Network

- Proposed by John Hopfield (1982)
- The key contribution was to provide an analysis of its network in terms of energy of the system
- Hopfield Neural Networks are associative memory devices
- Can be used for different pattern recognition problems



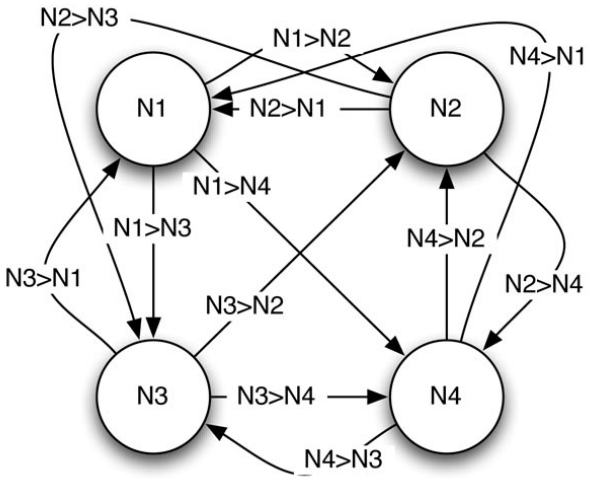
Structure of Hopfield NN



- A single layer network with no hidden layers
- The network is fully connected $w^{i,j} = w^{j,i}$
- But there is not self-connections $w^{i,i} = 0$
- Symmetry weights in connections
- Binary thresholds neurons: node activations are either -1 or +1
- All the neurons act as input and output
- Weights are fixed and activations are adaptive

Neuronal weights

Each connection has a weight assigned to it. This value is the same in both directions between each two neurons.



$$\begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

$$w_{ij} = w_{ji}$$

Because no self-connections are allowed, the elements of the weights matrix diagonal are equal to 0.

Energy Function

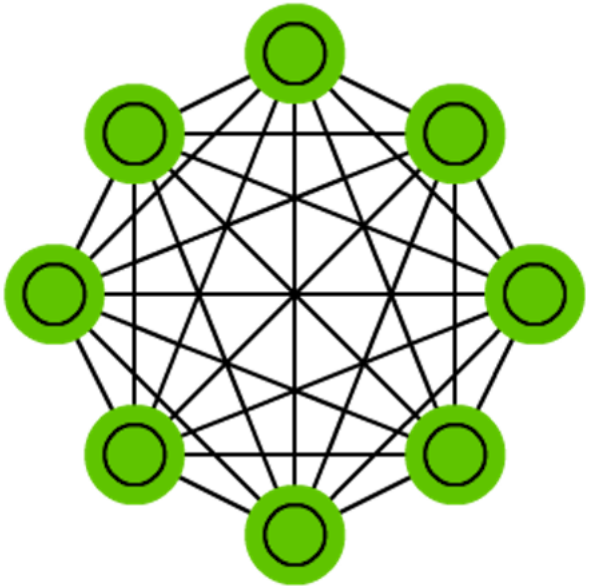
The network can have different behaviours: stable, oscillate and chaotic. However if the network:

- Only updates the activation of one unit at a time: asynchronous updating
- Symmetry weights in connections:

This allows to find a global energy function → then, it is proved that the network **converge** and not oscillate

Each configuration of the network has an energy value: sum of many contributions (from every unit)

Execution involves iteratively recalculating the activation of each node until a stable state is achieved – energy minimisation concept



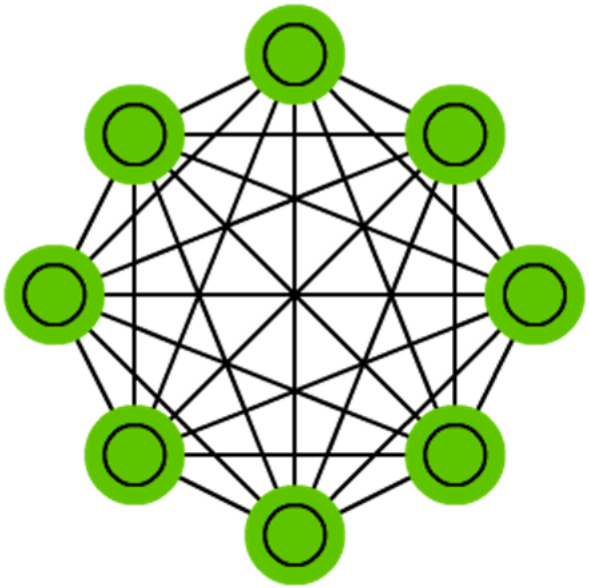
Training in Hopfield Networks

Like any other ANN, Hopfield network needs to be **trained** first before it can be used. Hopfield network is trained to store data (patterns) represented by **binary values**

Training is performed in one pass. Once the network is trained, we can try to recall any of the stored patterns

Number of stable storable and recalled patterns is ≈ 0.15n, where n is the number of neurons in the net.

Hopfield Neural Network



The network stores a set of binary patterns

$$P = p_1 \dots p_n$$

The weight matrix:

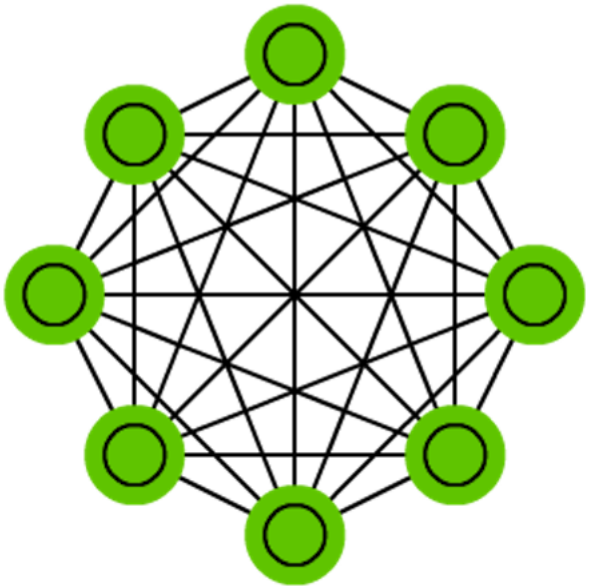
$$w_{i,j} = \frac{1}{N} \sum_{k=1}^{n} p_i^k p_j^k \quad where\ i \neq j$$

$$w_{i,i} = 0$$

where:

- $W_{i,j}$ is the weight between nodes i and j
- N is the number of nodes in the network
- n is the number of patterns to be learnt
- p_i^k is the value required to the i-th node in the pattern k

Hopfield Neural Network



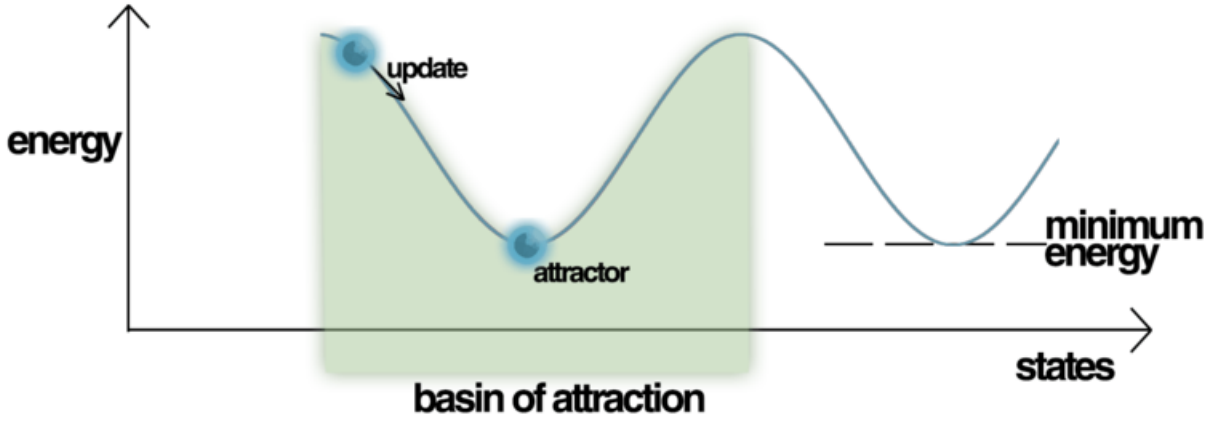
Training:

Execution is performed iteratively:

$$s_i = sign\left(\sum_{j=1}^{N} w_{i,j} s_j\right)$$

where:

- s_i is the activation of the i-th node

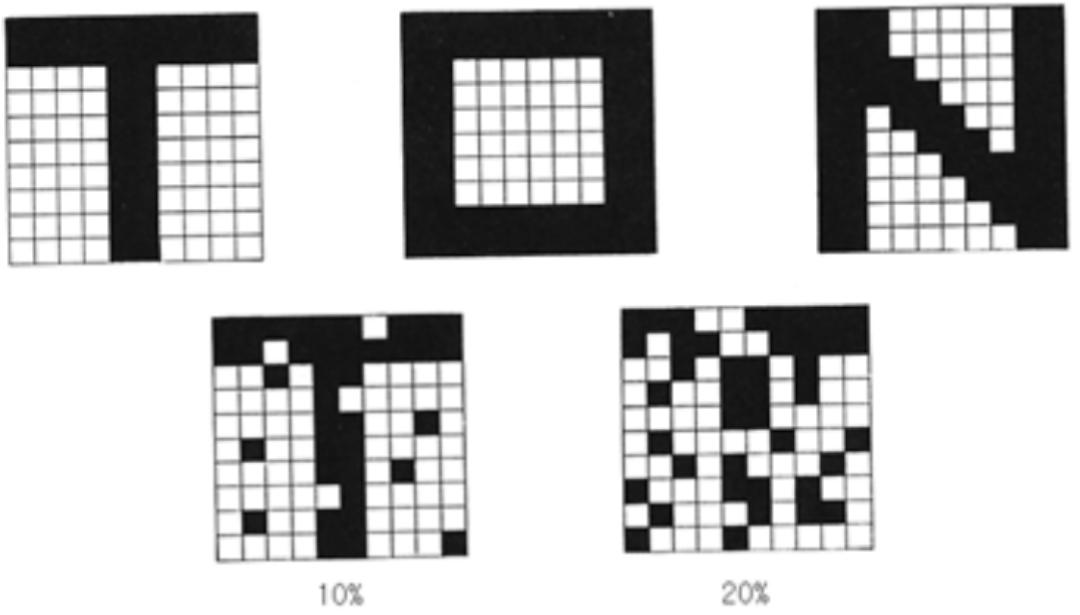


http://facstaff.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html

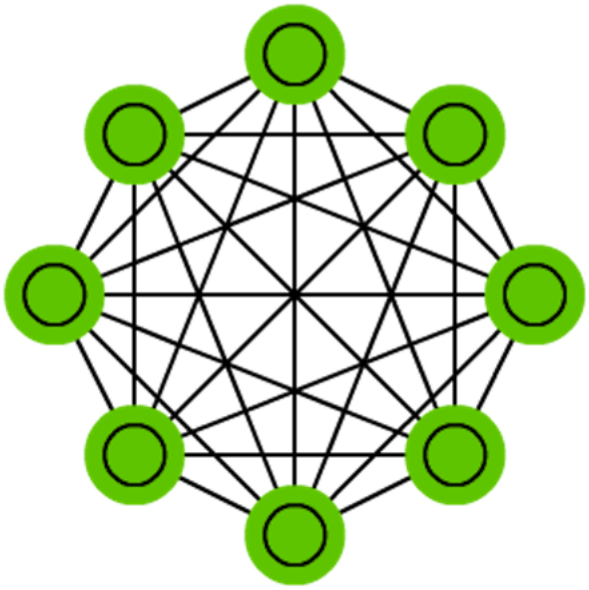
Applications in Hopfield Networks

- Image detection and recognition
- Enhancing X-Ray images
- In medical image restoration
- Combinatorial problems like travelling salesman problem

We can reconstruct corrupted patterns



Hopfield Neural Network



Example recalling the pattern “0101”

TRAINING

- **Step 1**: Initialise the weights to store the pattern. Create the weight matrix, nxn where n is the length of the pattern

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

length(“0101”) = 4

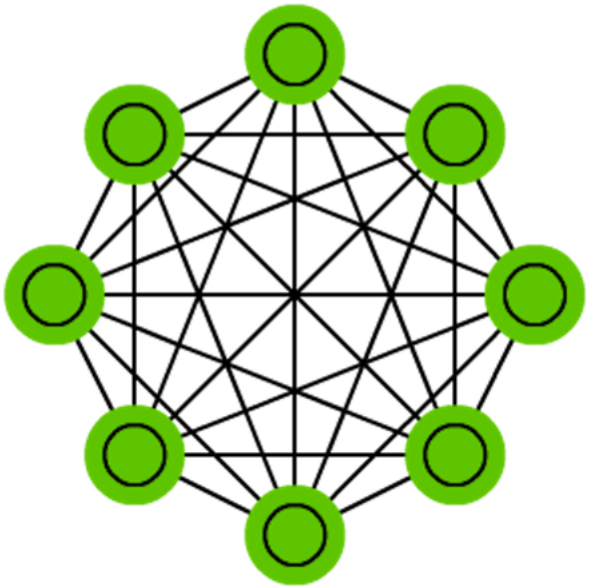
- **Step 2**: Create the contribution matrix for the pattern “0101”

By converting the pattern in bipolar (-1,1) and multiply by its transpose

$$\begin{bmatrix} -1 & 1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

Hopfield Neural Network

Example recalling the pattern “0101”



TRAINING

- **Step 3**: Sum the two matrices

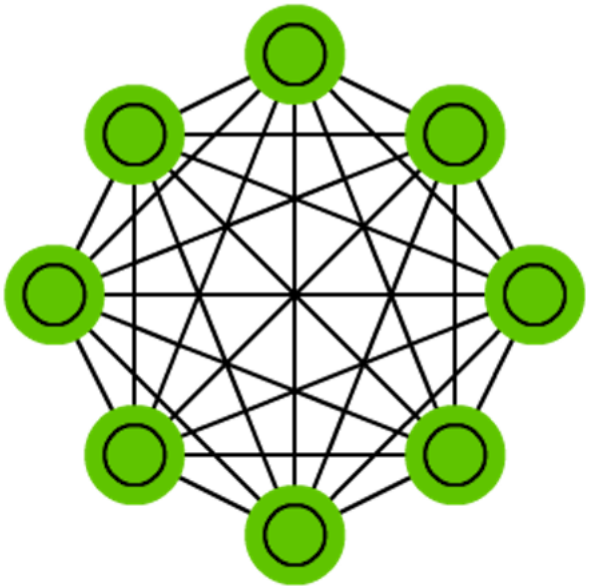
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

- **Step 4**: Set the diagonal to zero

$$\begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Remember that any node is connected with itself

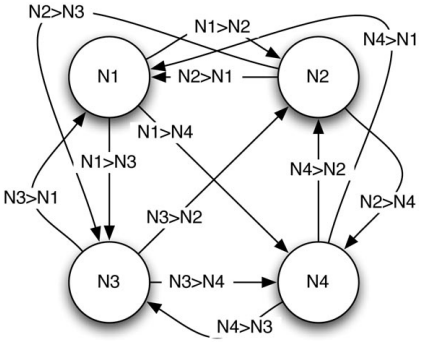
Hopfield Neural Network



Example recalling the pattern “0101”

EXECUTION

- **Step 1**: Present the pattern “0101” to each node in the network

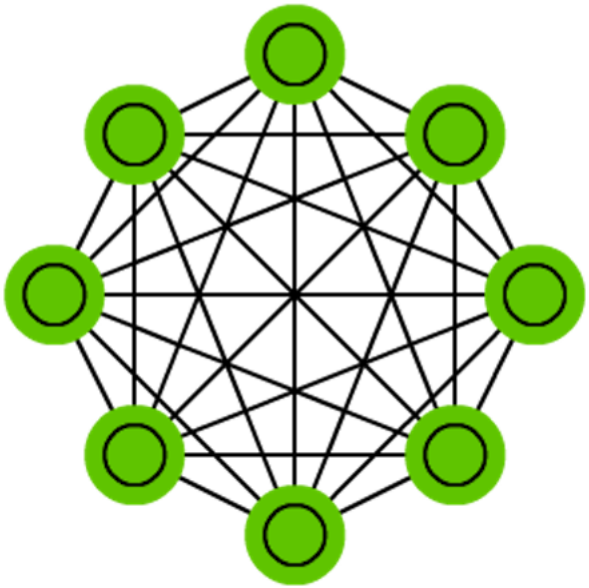


$$\begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

- **Step 2**: Calculate the activation of each neuron

$$s_i = sign\left(\sum_{j=1}^{N} w_{i,j} s_j\right)$$

Hopfield Neural Network



Example recalling the pattern “0101”

EXECUTION

- **Step 2**: Calculate the activation of each neuron

  Focus only on the two active columns of the pattern “0101”: columns 2 and 4.

$$\begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

For each row in these two columns:

N1 = -1 + -1 = -2
N2 = 0 + 1 = 1
N3 = -1 + -1 = -2
N4 = 1 + 0 = 1

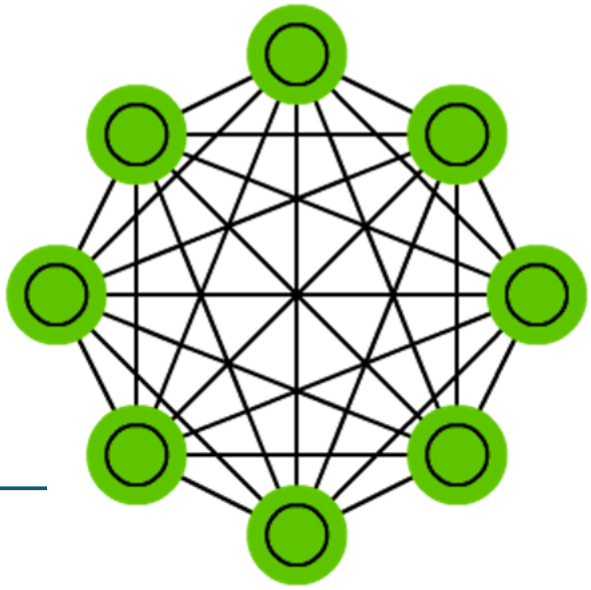
$$s_i = sign\left(\sum_{j=1}^{N} w_{i,j} s_j\right)$$

Hopfield Neural Network

Example recalling the pattern “0101”

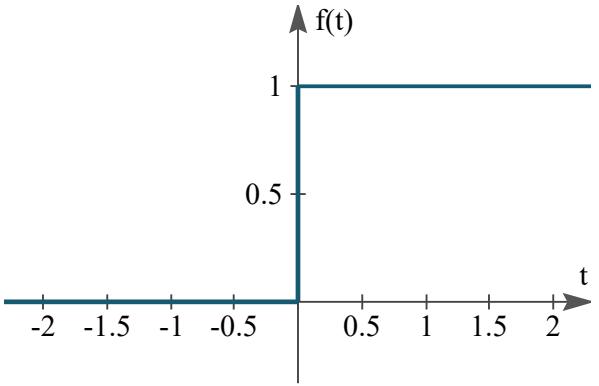
EXECUTION

- **Step 2**: Calculate the activation of each neuron



Threshold in zero: >0 fires

N1 = -2 < 0 wont fire **(0)**
N2 = 1 > 0 will fire **(1)**
N3 = -2 < 0 wont fire **(0)**
N4 = 1 > 0 will fire **(1)**



SUCCESS!! Our network recognised the pattern “0101”

What about the pattern “1010”?