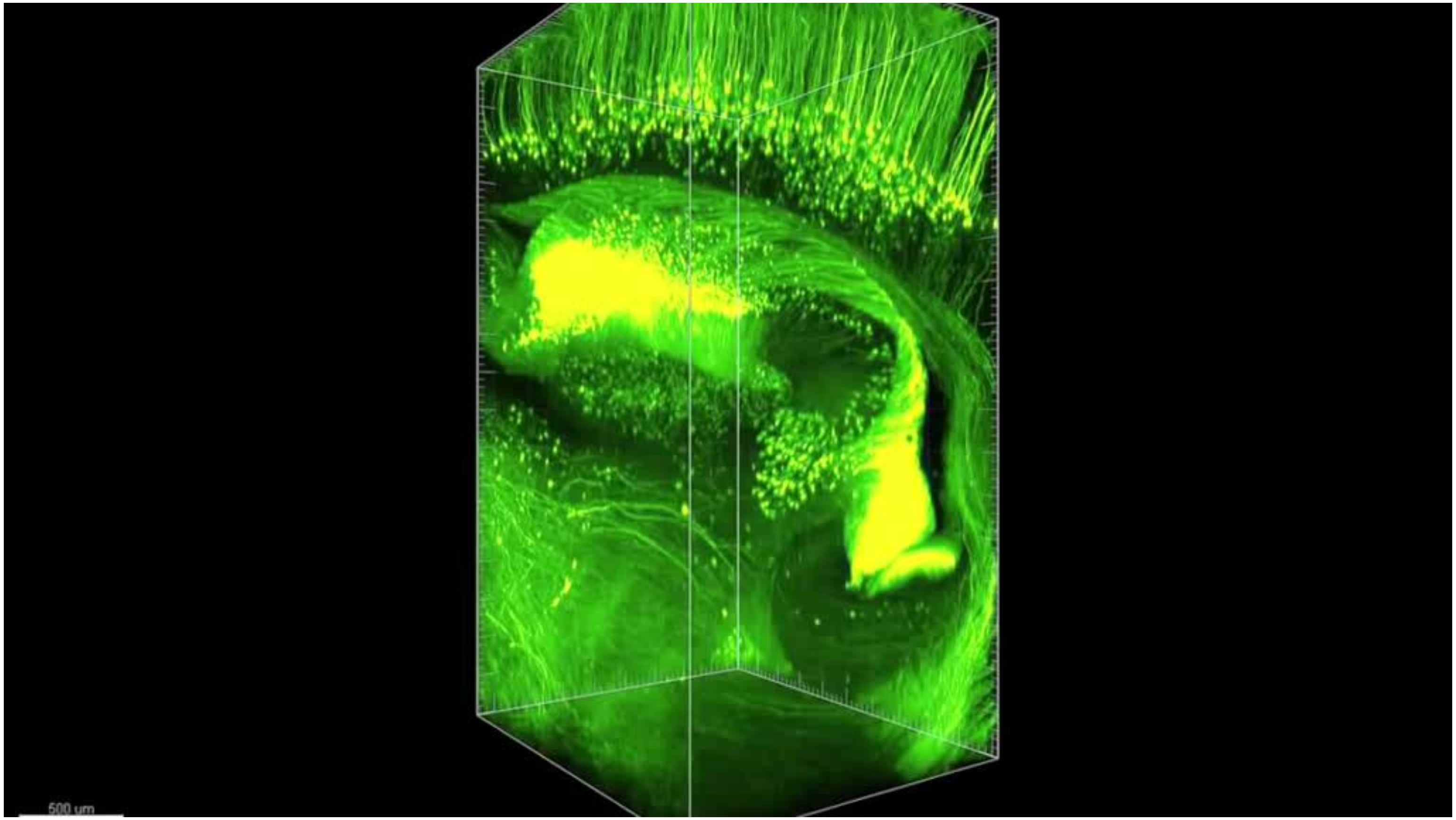
Neural Networks

Lecture 4: Introduction to Recurrent Neural Networks

http://www.informatik.uni-hamburg.de/WTM/

From Neuron Structures to Recurrent Localist and Distributed Representations

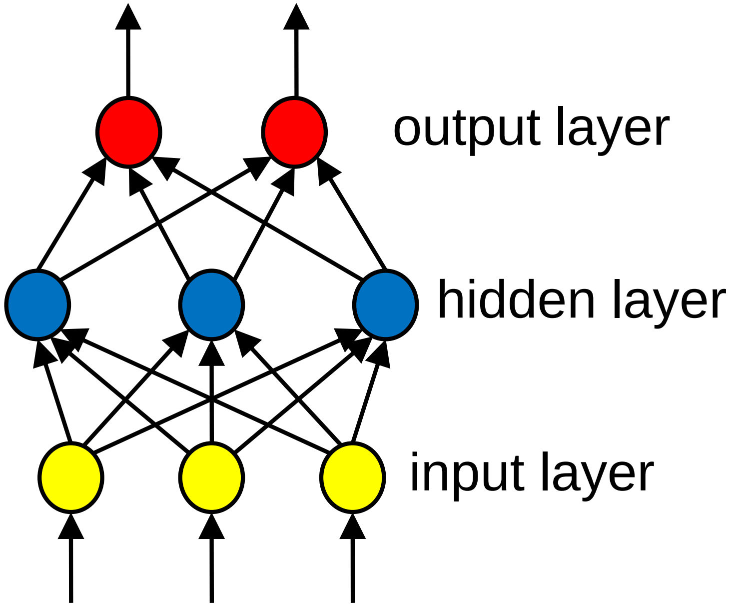
Neurons in an intact mouse hippocampus visualized using CLARITY and fluorescent labelling



Shen, H. See-through brains clarify connections. Nature, vol. 496, pp. 151, Macmillan Publishers Limited, 11 April 2013. Video online

Revision: The multilayer perceptron

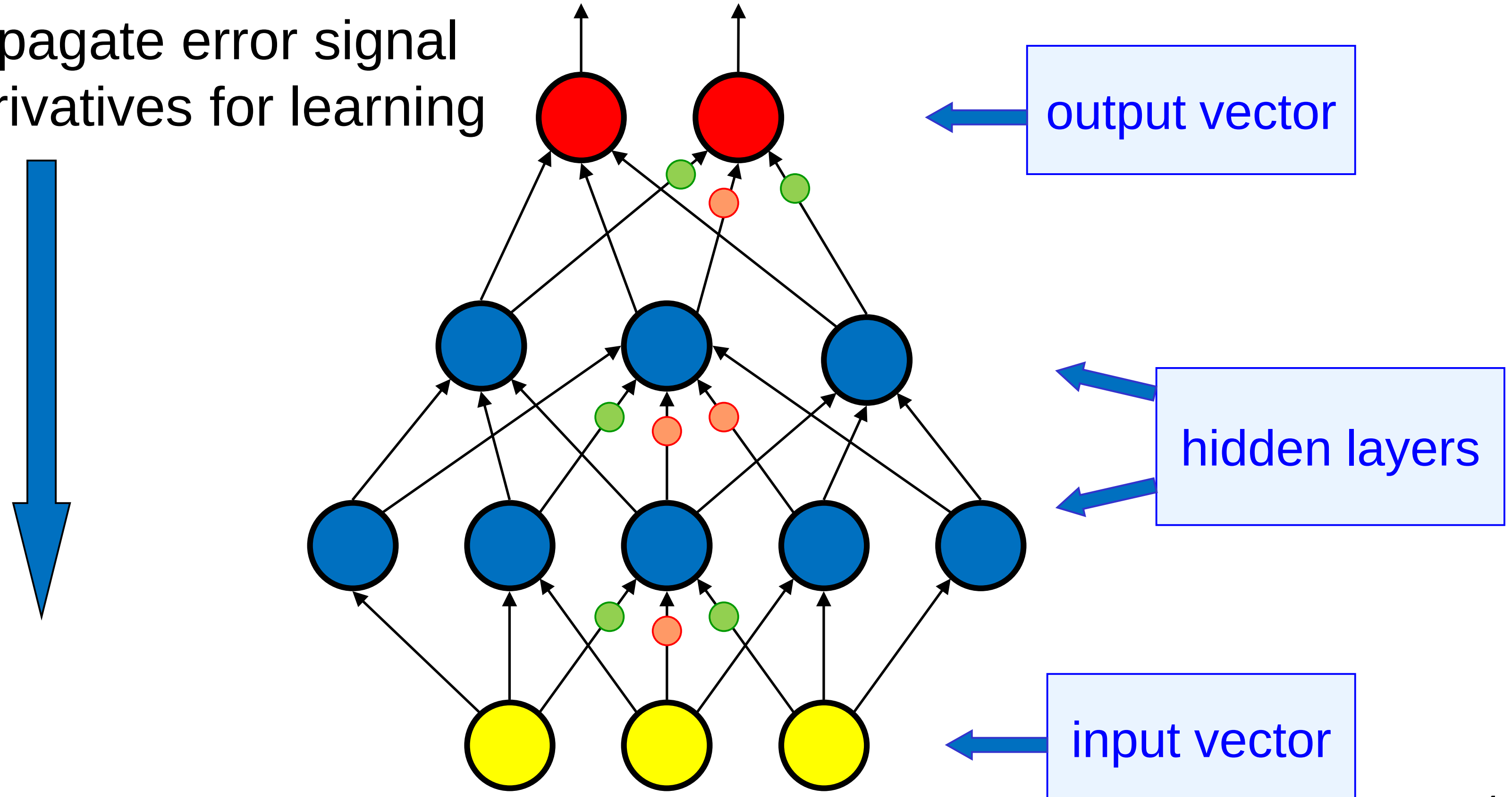
- Adding hidden layers to the network
- We use differentiable, nonlinear activation functions in the hidden layer to learn complex relationships
- The network is trained using backpropagation



Revision: Backpropagation

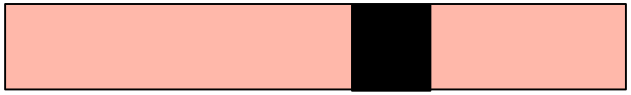
Compare outputs with correct answer to get error signal

Back-propagate error signal to get derivatives for learning



Localist representations

- The simplest way to represent things with neural networks is to dedicate *one neuron to each concept/feature*.
  - Easy to understand.
  - Easy to code by hand
    - Often used to represent inputs to a net
  - Easy to learn
  - Easy to associate with other representations or responses.



  - "One-hot encoding" in machine learning and natural language processing contexts

- But localist models are inefficient whenever the data has *componential* structure; not enough neurons to code all possibilities.

Examples of componential structure

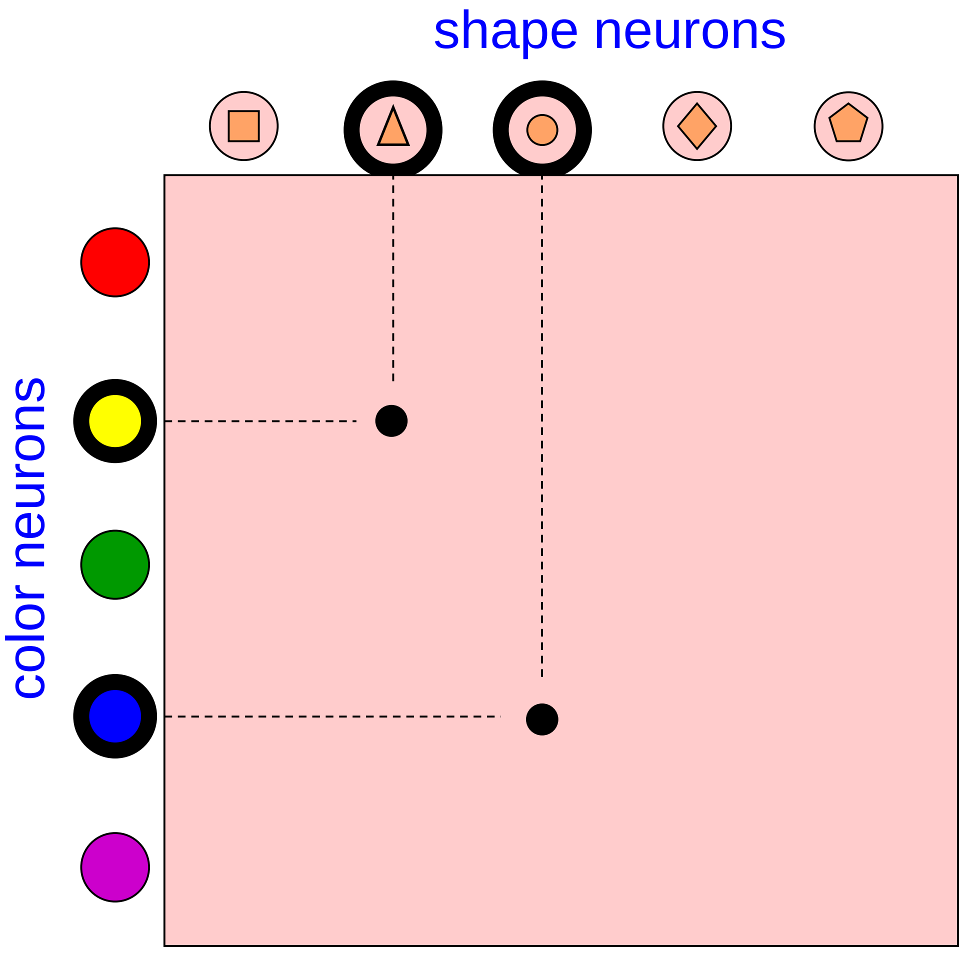
- Consider a *visual scene*
  - It contains many different objects
  - Each object has many properties like shape, color, size, motion
  - Objects have spatial relationships to each other

- Big, yellow Volkswagen
  - Do we have a neuron for this combination?
    - Is the BYV neuron set aside in advance?
    - Is it created on the fly?
    - How is it related to the neurons for big and yellow and Volkswagen?

Using simultaneity to bind things together

- Represent conjunctions by activating all the constituents at the same time.
  - One instance would not require connections between the constituents.
  - But what if we want to represent yellow triangle and blue circle at the same time?

- *Binding problem*: How to build up the information for an object based on features ?



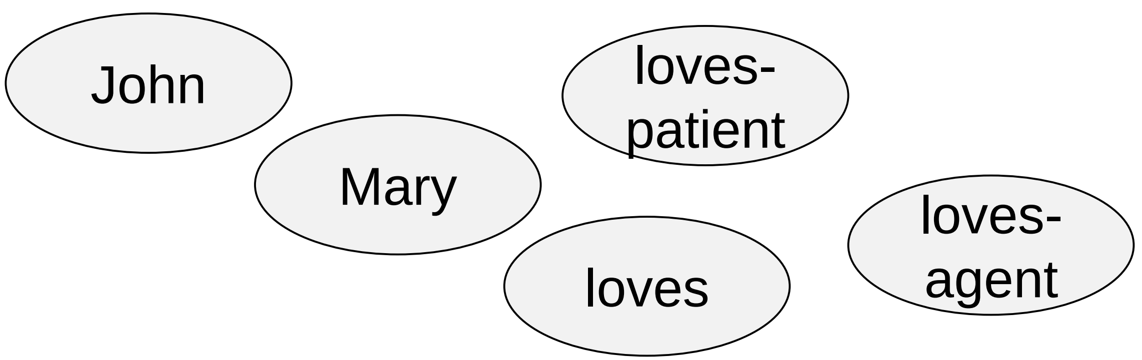
How to distinguish from representing yellow circle and blue triangle?

Using space to bind things together

- Conventional computers can bind things together by putting them into neighboring memory locations
  - This works nicely in vision. Surfaces are generally opaque, so we only get to see one thing at any location in the visual field
    - If we map different properties topographically, we can assume that properties at the same location belong to the same thing
- Leads to self-organizing maps and topographical ordering in feature maps in the general case (more later)
- Simple form: distributed versus localist representations

Using time to bind things together

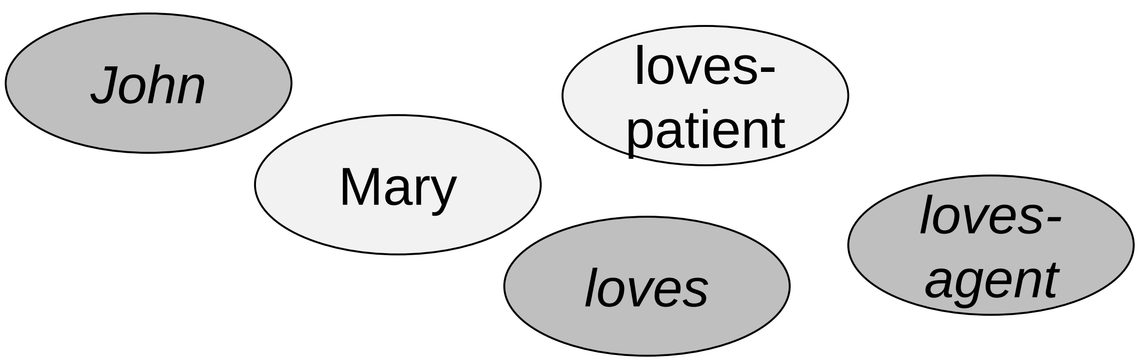
- Another way to represent structure: Temporal binding of units
- Localist representation scheme: Each unit has meaning
- *loves(John, Mary):*
  - Units for relational roles: *loves*, *loves-agent*, *loves-patient*
  - Units for relation *loves* and objects, *John*, *Mary*
- Binding between relations and arguments occurs over time
- Temporal Synchrony: Units active at same time are bound together



Using time to bind things together

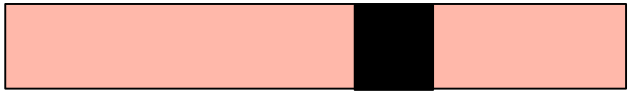
- Another way to represent structure: Temporal binding of units
- Localist representation scheme: Each unit has meaning
- *loves(John, Mary):*
  - Units for relational roles: *loves*, *loves-agent*, *loves-patient*
  - Units for relation *loves* and objects, *John*, *Mary*
- Binding between relations and arguments occurs over time
- Temporal Synchrony: Units active at same time are bound together

Units active at same time representing that *John* is the agent of a *loves* relation

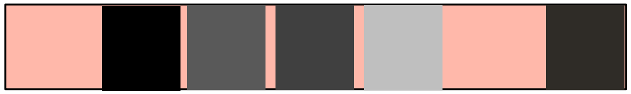


The definition of "distributed representation"

- If each neuron represents one concept, this must be a localist representation.



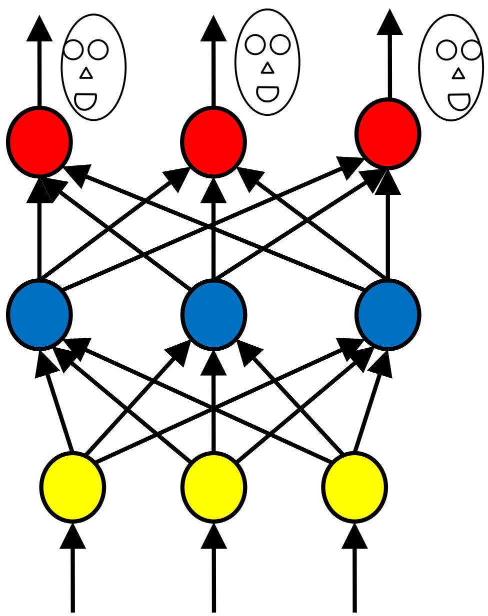
- "Distributed representation" means a many-to-many relationship between two types of representation (such as concepts and neurons).
  - Each concept is represented by many neurons.
  - Each neuron participates in the representation of many concepts.



Example: Head pose detection

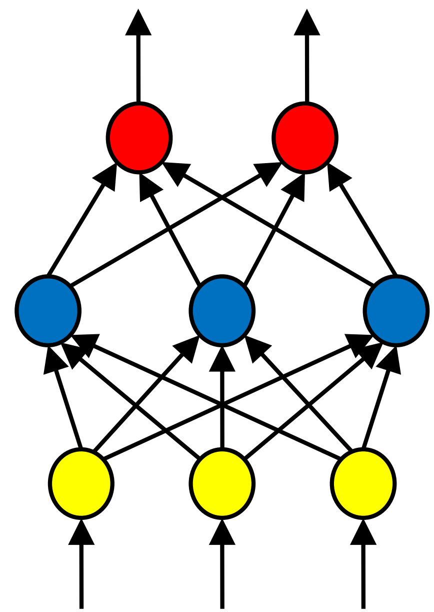
- Localist:
  3 neurons, 3 classes

none:	0	0	0
left:	1	0	0
right:	0	0	1
center:	0	1	0



- Distributed:
  2 neurons, 3 classes, real values also common…

none:	0	0
left:	1	0
right:	0	1
center:	1	1



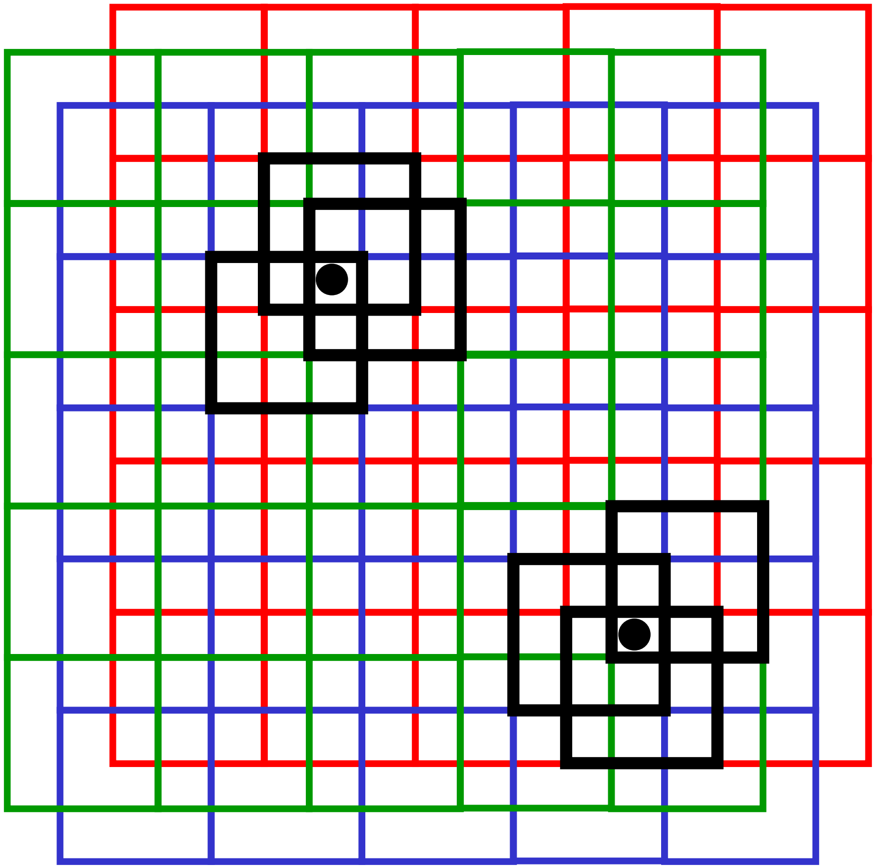
Advantages/disadvantages?

Coarse coding

- Localist representation: using one neuron per entity is inefficient.
  - An efficient code would rather have each neuron active half the time.
- Distributed representation: can we get accurate representations by using lots of inaccurate neurons?
  - If we can it would be very ***robust*** against hardware failure.

Coarse coding

- Use three *overlapping arrays* of large cells to get an array of fine cells
  - If a point falls in a fine cell, code it by activating 3 coarse cells.
- This is *more efficient* than using one neuron for each fine cell.
  - It loses by needing 3 arrays
  - It wins by a factor of 3x3 per array
  - Overall it wins by a factor of 3



How efficient is coarse coding?

- The efficiency depends on the dimensionality
  - one dimension: coarse coding does not help.
  - 2-D: the saving in neurons is proportional to the ratio of the fine radius to the coarse radius.
  - k dimensions: by increasing the radius by a factor of r we can keep the same accuracy as with fine fields and get a saving of:

$$saving = \frac{\#\ fine\ \ neurons}{\#\ coarse\ \ neurons} = r^{k-1}$$

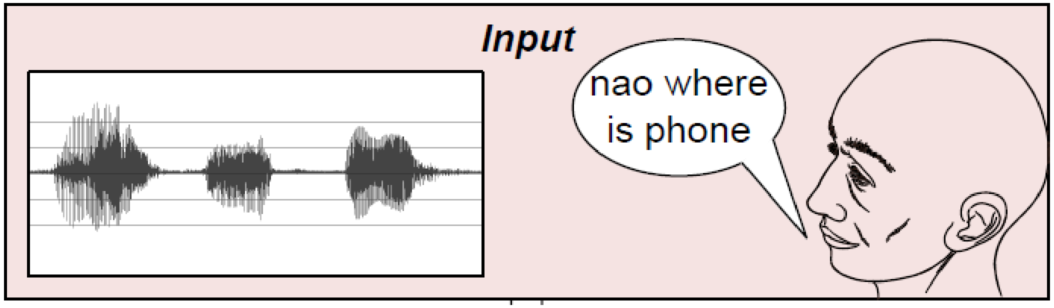
Sequences in neural networks

- Sequences everywhere, in vision, in speech, in text, in condition monitoring, in movement…

- Neural networks need to represent sequential knowledge

- Spatial or temporal approaches

- How to represent sequences?



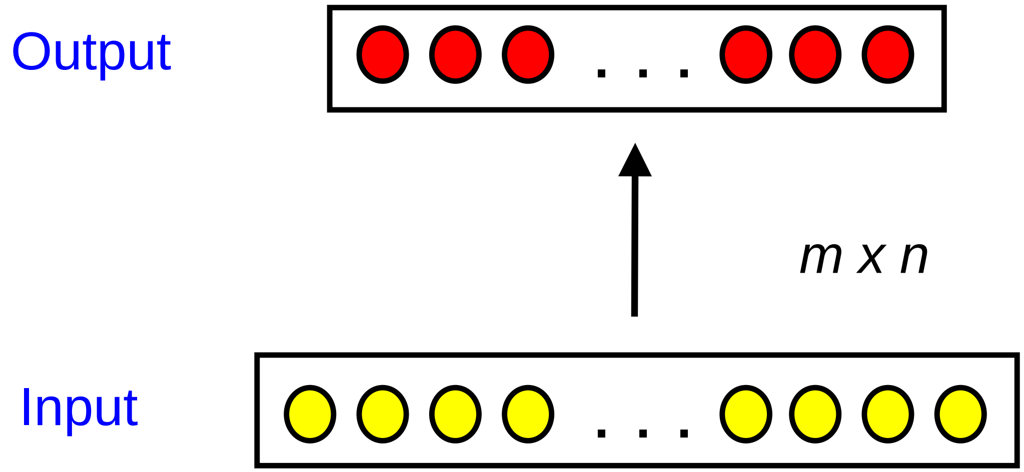
Processing sequences with fixed input and output

- Case role assignment: fixed input features and fixed number of semantic case role classes as output
- Example: semantic case role assignment e.g. "break:
  1. The boy broke the window
  2. The rock broke the window
  3. The window broke
  4. The boy broke the window with the rock
  5. The boy broke the window with the curtain
- First noun phrase can be: Agent (1,4,5), Instrument (2), Patient (3)
- Prepositional phrase can be: Instrument (4), Modifier (5)

- Length of sentences and positions of case roles vary

Fixed input and output (cont)

- Feedforward network for restricted input and output (fixed sequences)

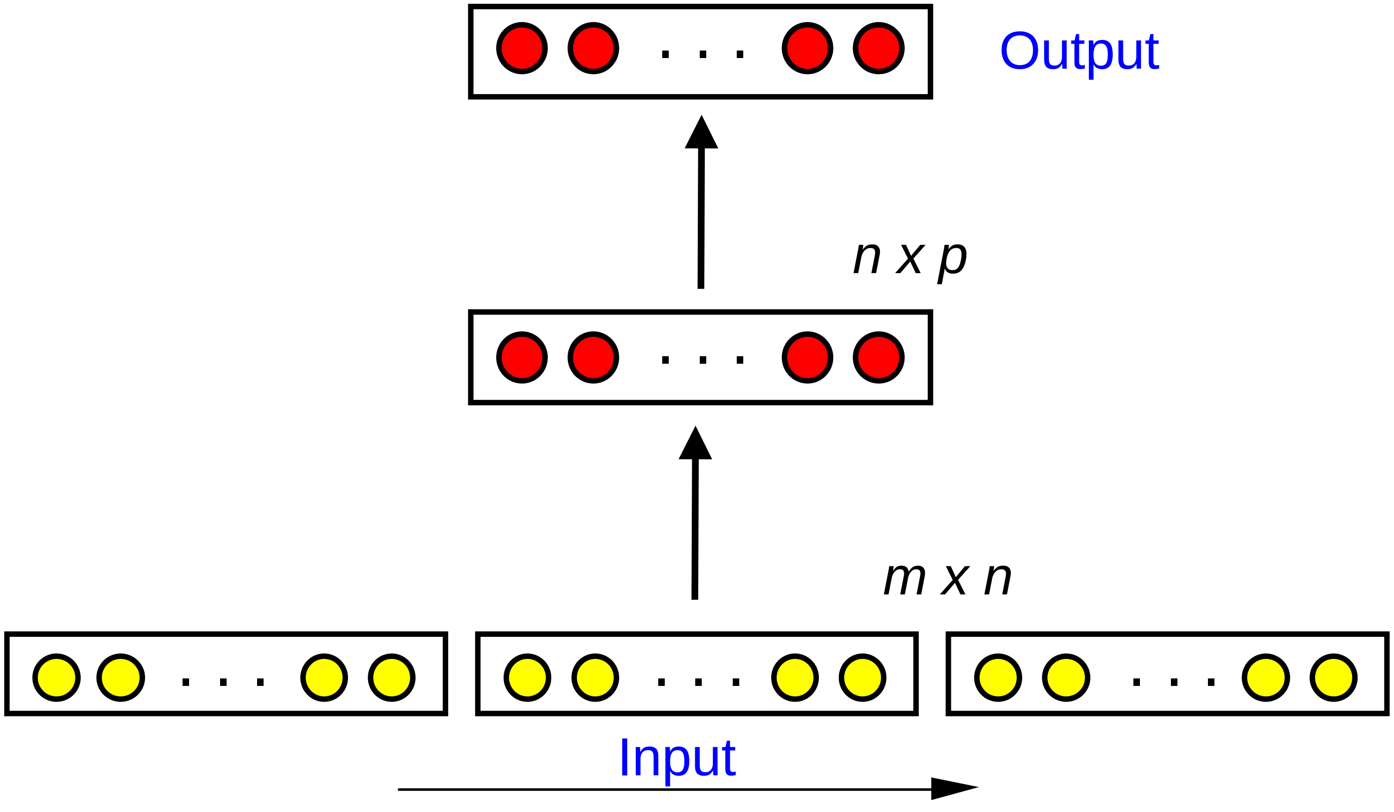


- But how to represent unrestricted sequences?

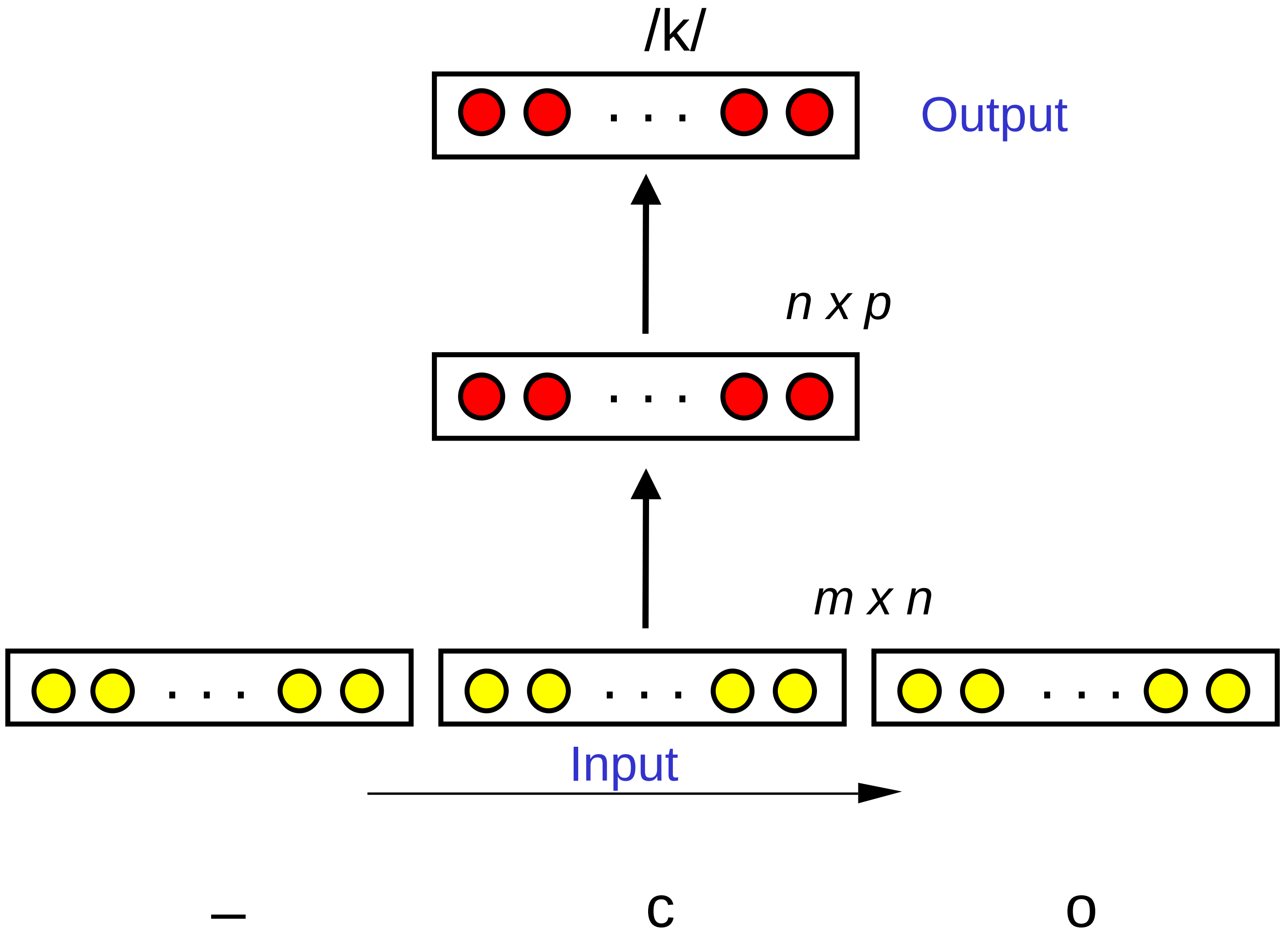
Sliding windows: space for time

- Trading space for time
- Instead of presenting whole sequence only limited part is presented
- E.g. in NETtalk, a window of seven letters moved over text
- Task was to produce the central phoneme
  - for instance, words ending in “ave” like “brave”, “gave”, …
  - but exceptions: “have”
- Disadvantage of fixed context?

Sliding windows for seqentiality
(e.g. NETtalk; Sejnowski & Rosenberg 1986)



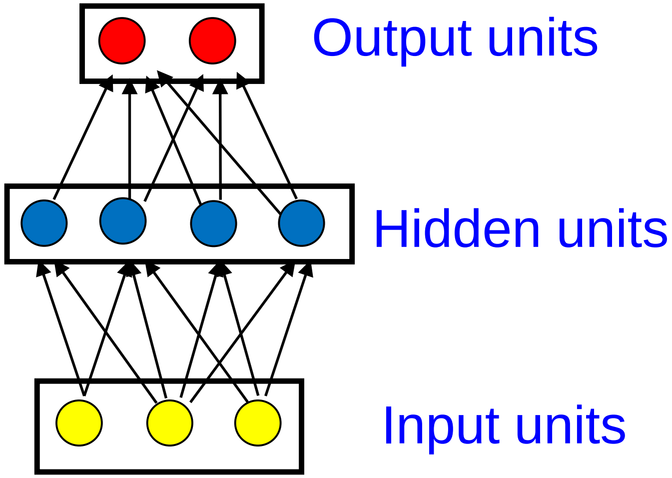
Demonstration of NETtalk



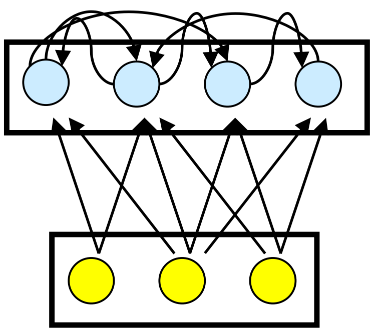
[http://cnl.salk.edu/Media/nettalk.mp3]

Introduction to recurrent neural networks: Types of connectivity

- Feedforward networks
  - compute a series of transformations
  - Typically, first layer is input and last layer is output;
  - Efficient mappings



- Recurrent networks
  - have directed cycles in their connection graph. They can have complicated temporal dynamics
  - More biologically realistic

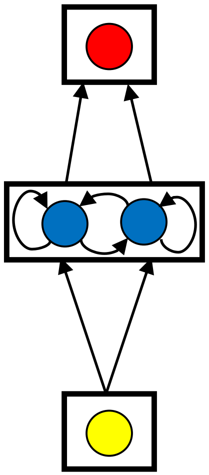


Recurrent neural networks (RNN)

- Feed-forward network accepts fixed-size vector as input (e.g. image) and produces a fixed-size vector as output
- Recurrent network with cycles:
  - Network has internal state: It can remember past states
  - Natural modeling of sequential data: the network can operate over *sequences* of vectors, not just fixed-length windows
  - It can behave chaotically or oscillate. This is computationally interesting and may be useful in adversarial situations (-> weather)
- The memory of a recurrent network is potentially unbounded but is practically limited by the vanishing gradient problem

Example task: Sequential XOR

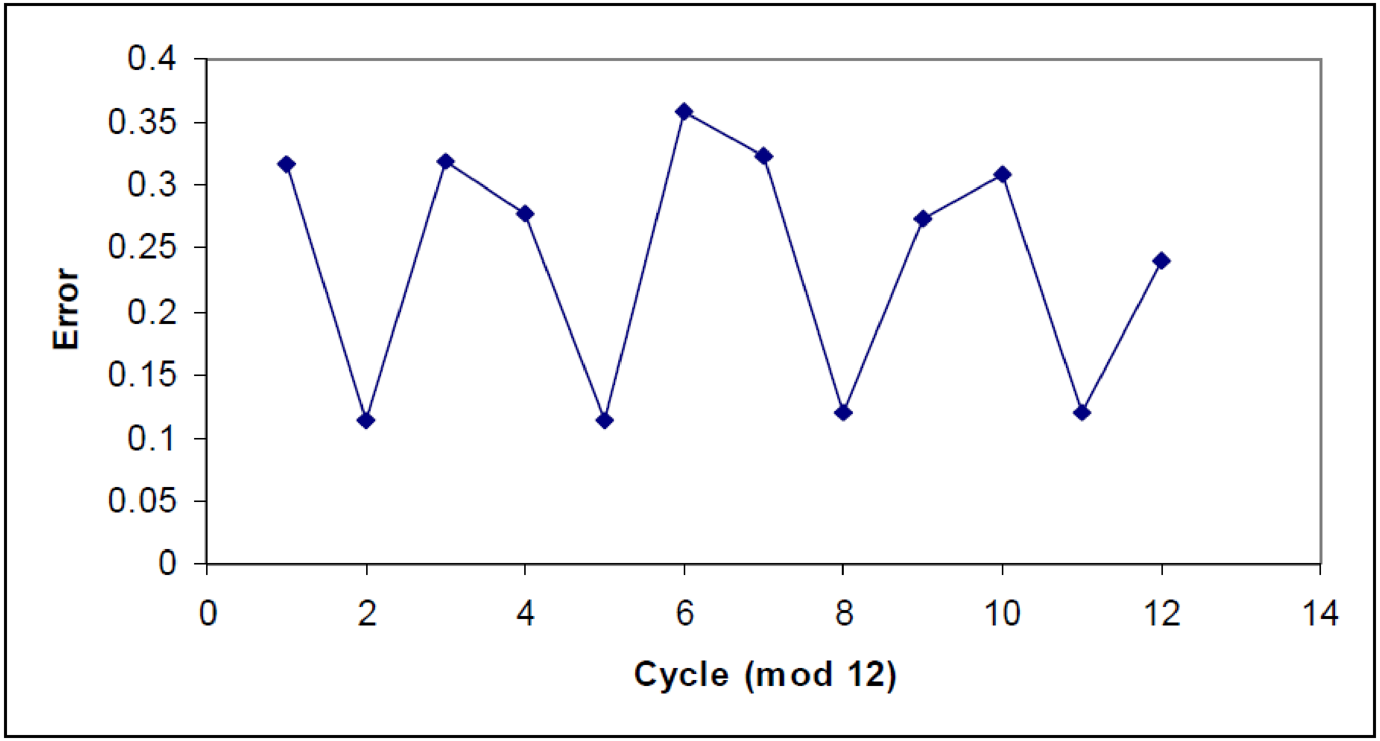
- RNN with 1 input, 2 hidden, 1 output units can learn XOR
- Instead of using fixed-length vectors we model XOR as a sequence prediction problem:



1 0 **1** 0 0 **0** 0 1 **1** 1 1 **0** 1 0 **1** . . .

- Constructing a sequence of 1-bit inputs by presenting the 2-bit inputs one bit at a time (i.e., in two time steps), followed by the 1-bit output

Network learns something about temporal structure:

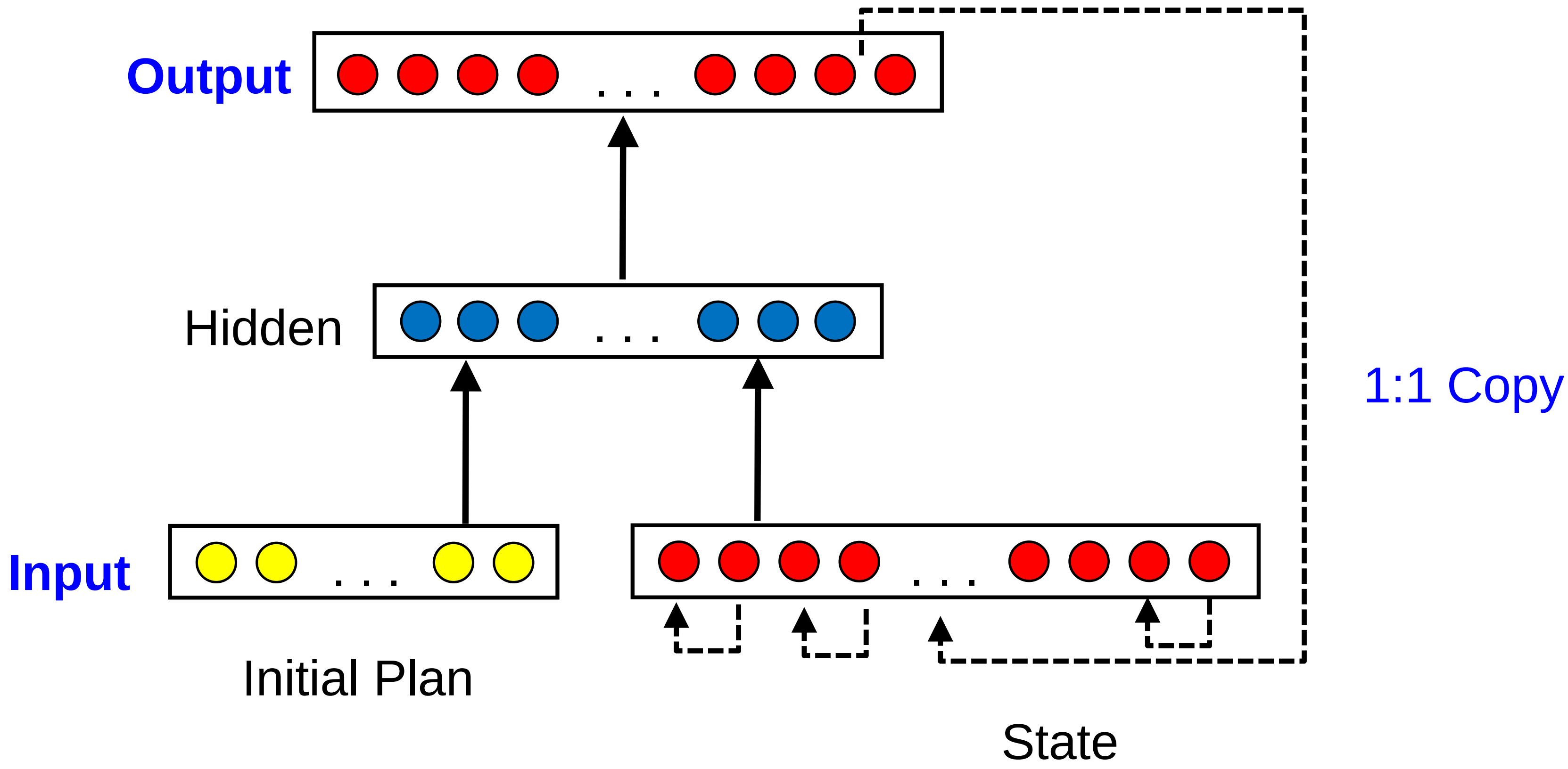


- Root mean squared error over 12 consecutive inputs in sequential XOR task
- Data points averaged over 1200 sequences.
- Success every 3 bits plus sometimes fortuitous successes.

Jordan network

- Plan units receive initial input as a plan for action planning
- Output units receive desired action
- Sequential feedback with 1:1 copied state units
- Advantage: sequential knowledge not limited
- Disadvantages:
  - amount of feedback limited by the dimensionality of the output
  - direct feedback of action values injects noise and possible error into the network

Jordan network



- Activations are copied from output layer to state layer on a one-for-one basis, with fixed weight of 1.0
- Straight lines represent trainable connections

Simple recurrent network (SRN)

- Problem with Time

[0 1 1 1 0 0 0 0 0]

[0 0 0 1 1 1 0 0 0]

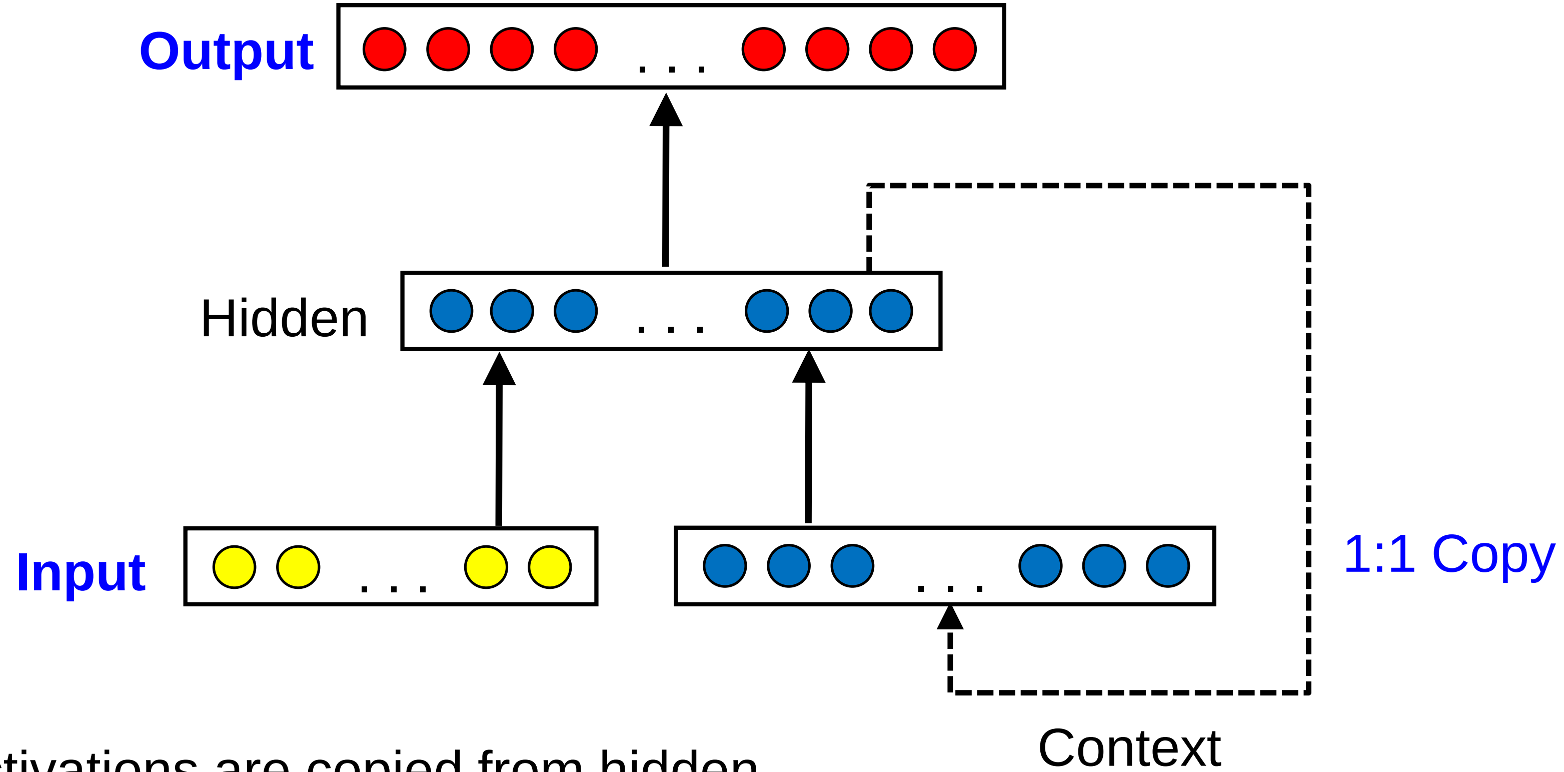
- Two vectors appear to be instances of the *same* basic *pattern*, but displaced in space
- Relative temporal structure should be *preserved* in the face of absolute temporal displacements
- Related to variances in vision? Why ?

Simple recurrent network (SRN)

- Problem of using output-input recurrent connections in Jordan Network
- Motivation for using internal state information
- Copy the internal hidden layer for next input
- The SRN is usually considered the base RNN model. Countless variations extend the SRN.

[Landmark Paper: *Elman* J., Finding Structure in Time, Cognitive Science 14, 1990]

Simple recurrent network (SRN)



- Activations are copied from hidden layer to context layer on a one-for-one basis, with fixed weight of 1.0
- Straight lines represent trainable connections.

Example Prediction

Input: w_1 w_2 w_3 .... w_n

Output: w_2 w_3 w_4 .... w_{n+1}

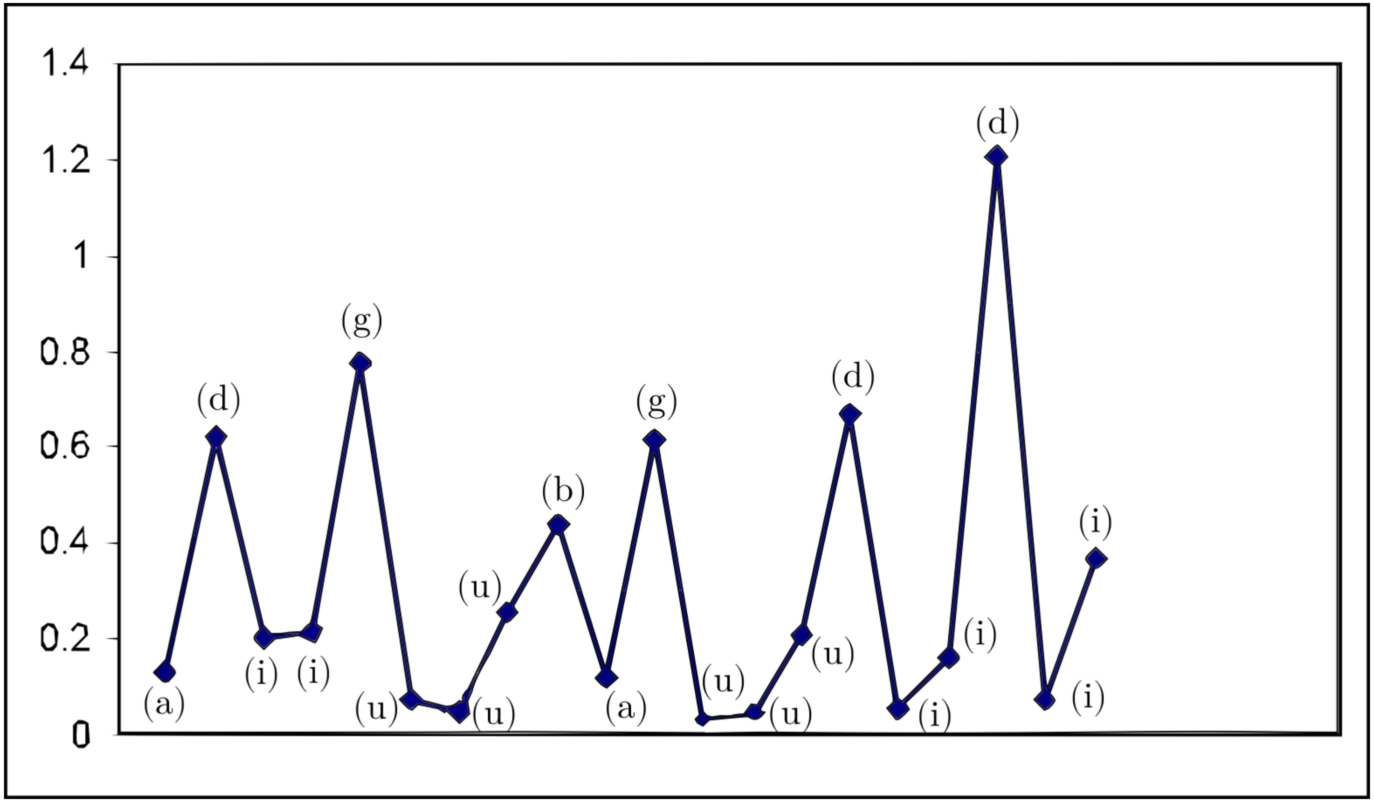
SRN as a predictor of letter sequences

- Multi-bit inputs of sequences
- 3 consonants (**b**, **d**, **g**) combined in random order to obtain 1000-letter sequence. Then each consonant replaced using rules
  - b->ba
  - d->dii
  - g->guuu
- **dbgbddg**... into **diibaguuubadiidiiguuu**
- SRN with: 6 input units, 20 hidden units, 6 output units

Vector definitions of alphabet

	Consonant	Vowel	Interrupted	High	Back	Voiced
b	[1	0	1	0	0	1]
d	[1	0	1	1	0	1]
g	[1	0	1	0	1	1]
a	[0	1	0	0	1	1]
i	[0	1	0	1	0	1]
u	[0	1	0	1	1	1]

Root mean squared error in letter prediction task

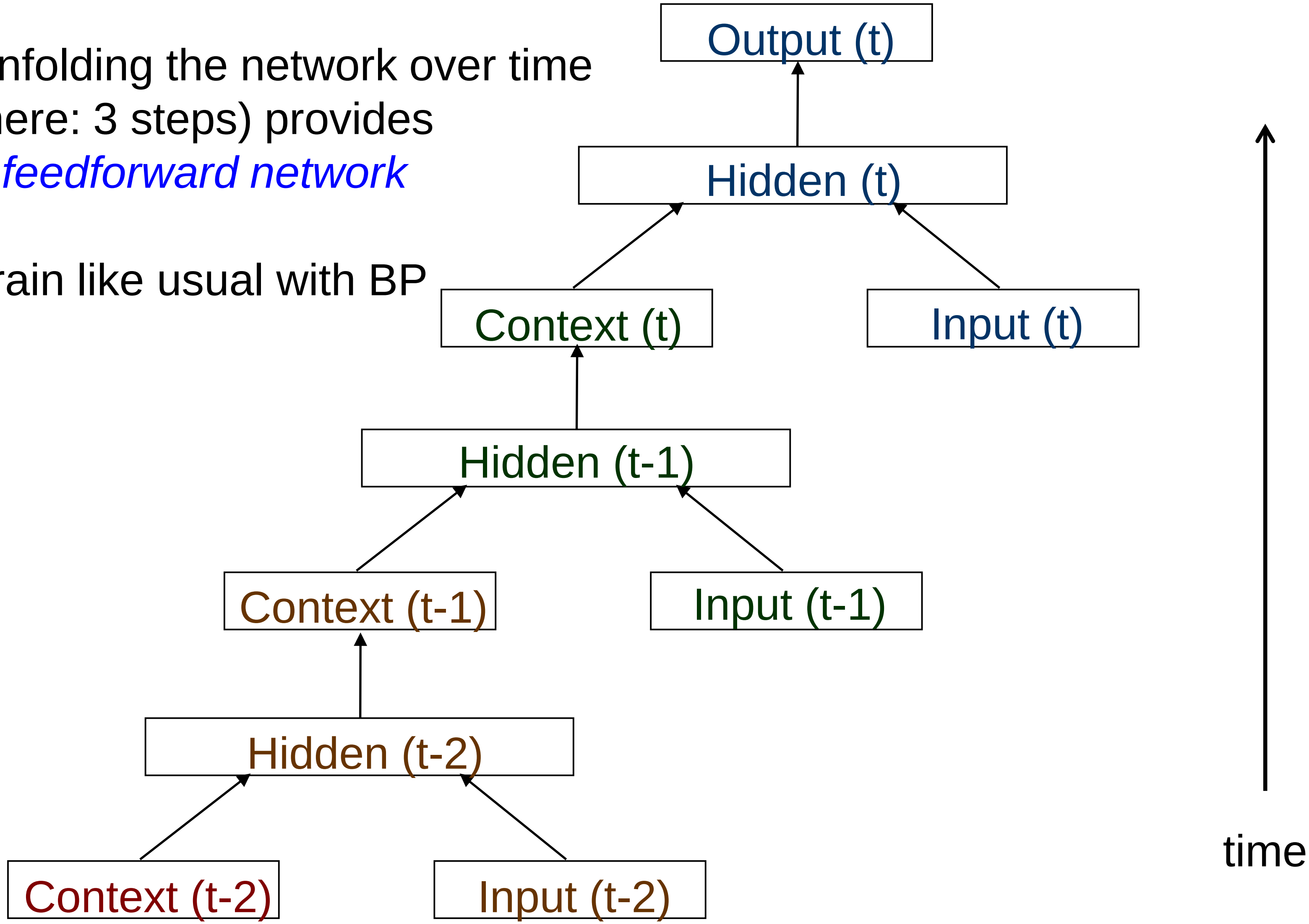


- Labels indicate the correct output prediction at each point in time.
- Given a consonant as input, the network can predict following vowel.

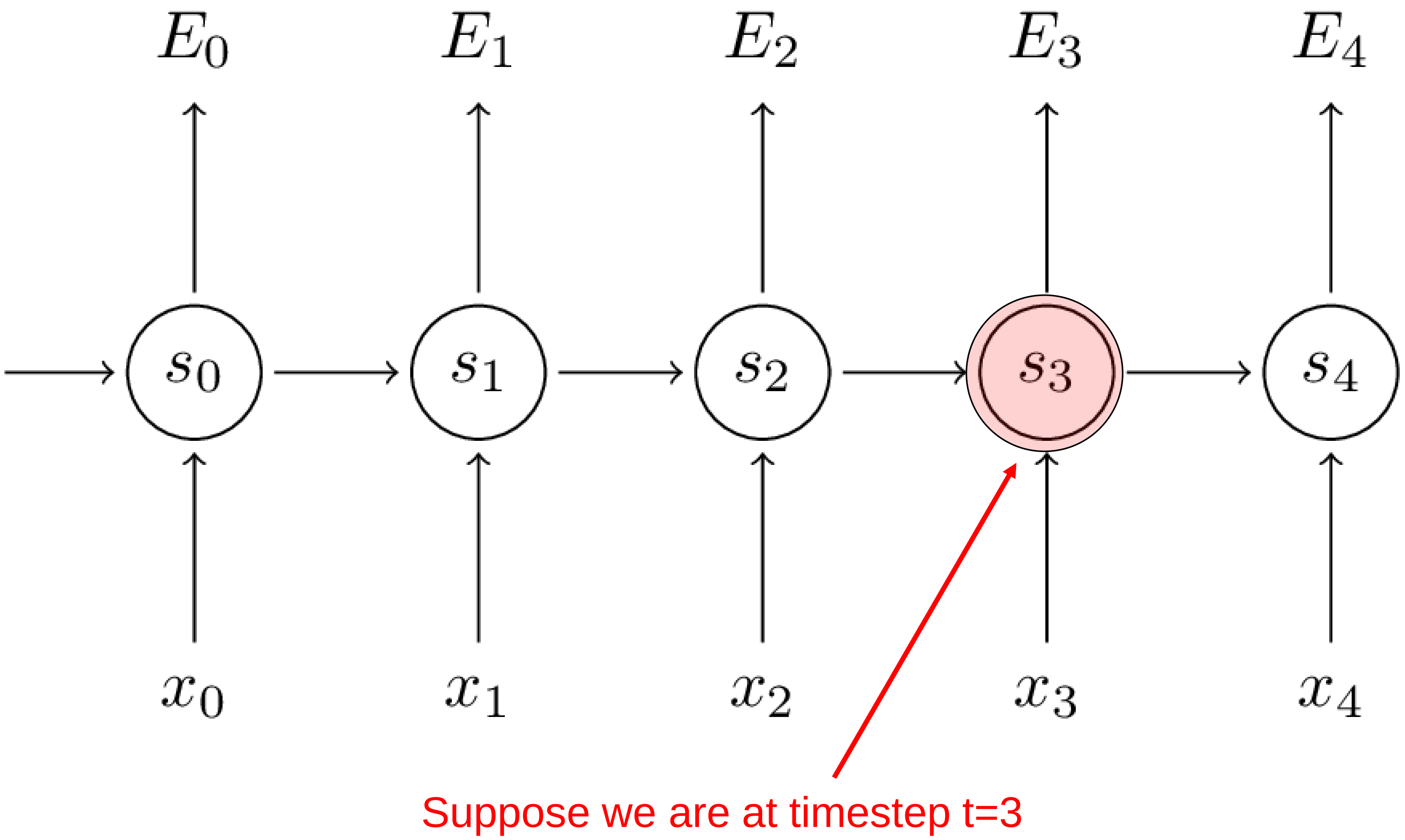
Backpropagation Through Time (BPTT)

- Unfolding the network over time (here: 3 steps) provides a *feedforward network*

- Train like usual with BP

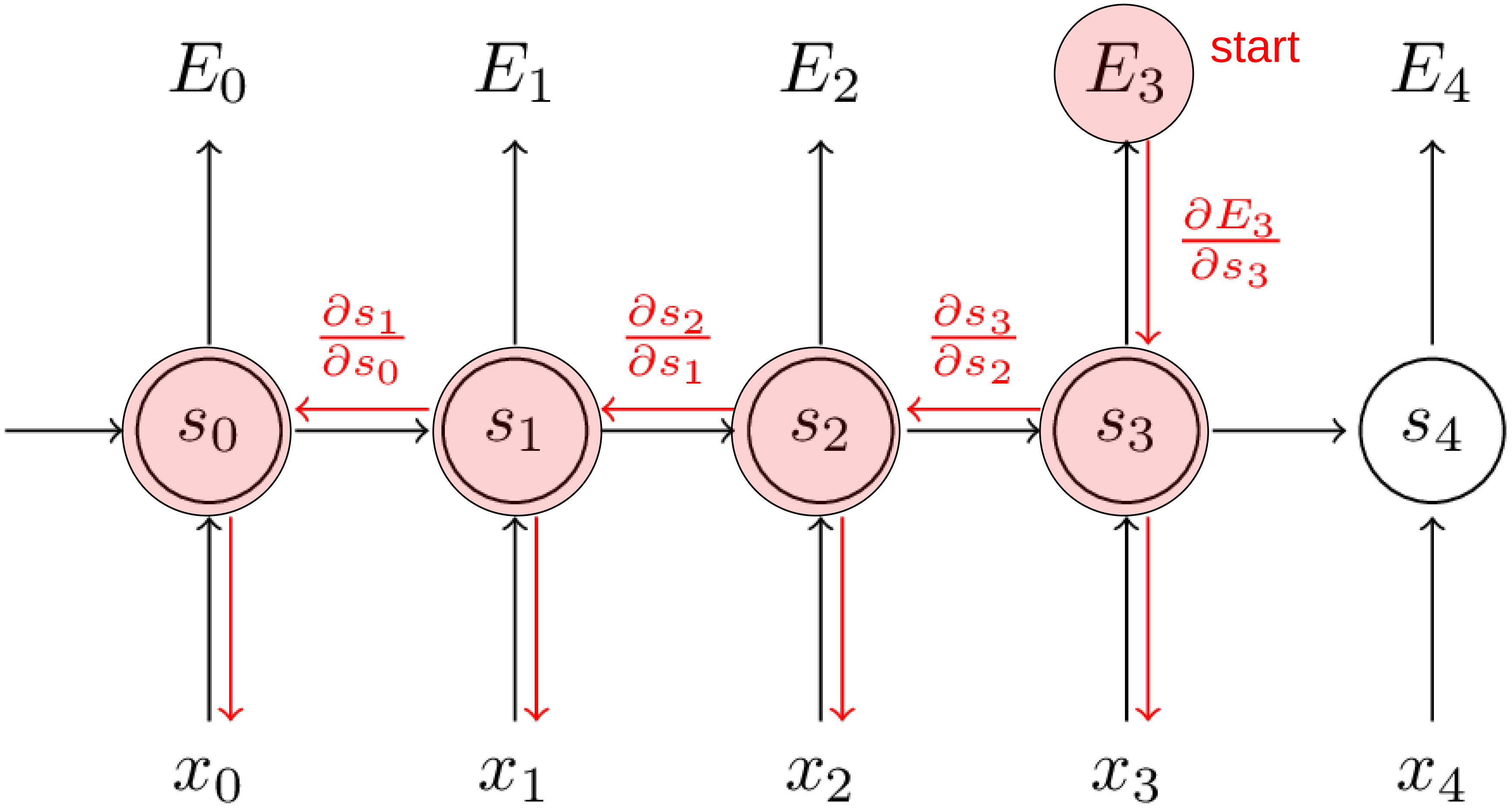


Backpropagation Through Time (BPTT)



E_t: Error, s_t: hidden state, x_t: input (at timestep t)

Backpropagation Through Time (BPTT)



Train the context-weights with normal backpropagation

E_t: Error, s_t: hidden state, x_t: input (at timestep t)

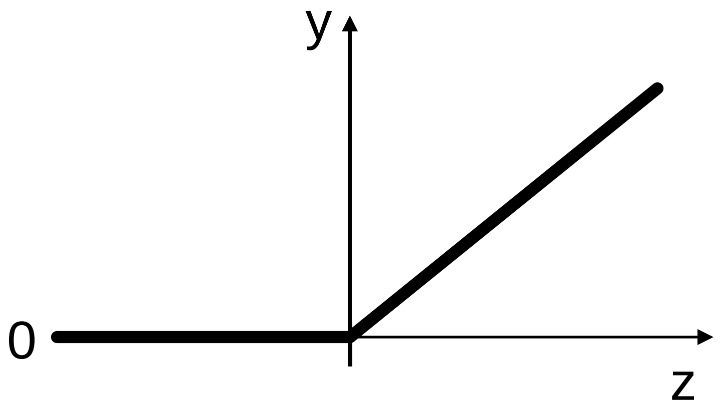
The vanishing or exploding gradient problem

- What happens to the magnitude of the gradients as we backpropagate through many layers?
- Multiplication and nonlinearities at each step amplify the signal:
  - If the weights are small, the gradients shrink exponentially
  - If the weights are big, the gradients grow exponentially
- This can easily happen in an RNN trained on long sequences, diminishing the influence of past inputs
  - RNNs have difficulty dealing with long-term dependencies

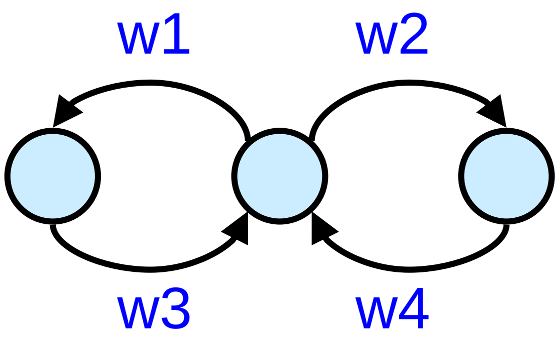
Background: [Hochreiter91, Bengio94]

Potential solutions

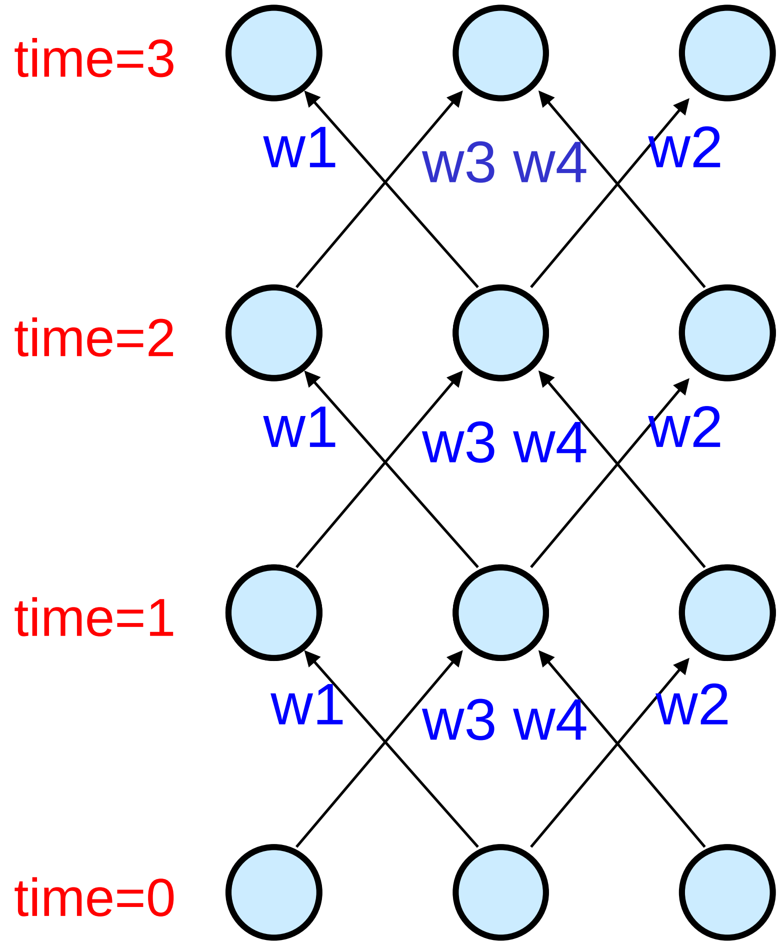
- Echo State Networks (to come)
- Long Short-Term Memory (to come)
- More sequential memory: Plausibility networks (to come)
- Better Optimization (to come)
  - No Backpropagation
  - Unsupervised pre-training
  - Simpler activation functions: Rectified linear units (ReLU) rather than sigmoid or hyperbolic tangent activation functions



The relationship between layered, feedforward nets and recurrent nets revisited



- Assume that there is a time delay of 1 in using each connection
- The recurrent net as a layered net that keeps *reusing* the same *weights*



Backpropagation with weight constraints

- It is easy to modify the backprop algorithm to *incorporate linear constraints between the weights*.

$$To \quad constrain \quad : \quad w_1 = w_2$$

$$we \quad need \quad : \quad \Delta w_1 = \Delta w_2$$

- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.

$$compute \quad : \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for \;\; w_1 \; and \;\; w_2$$

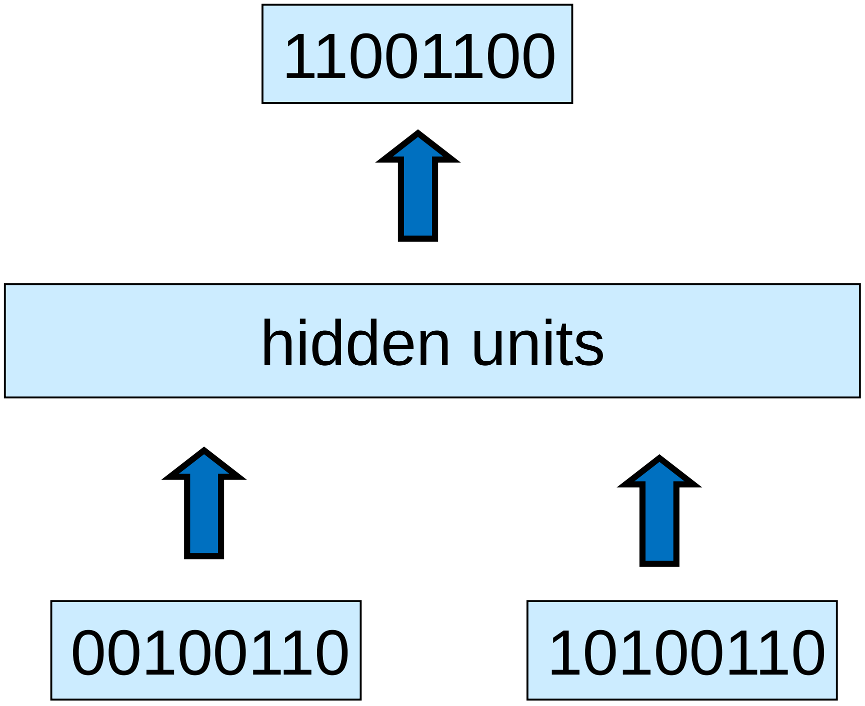
Initialization

- We need to specify initial activity state of all hidden and output units
  - We could just fix these initial states to have some default value like 0.5
  - better to treat the initial states as learning parameters

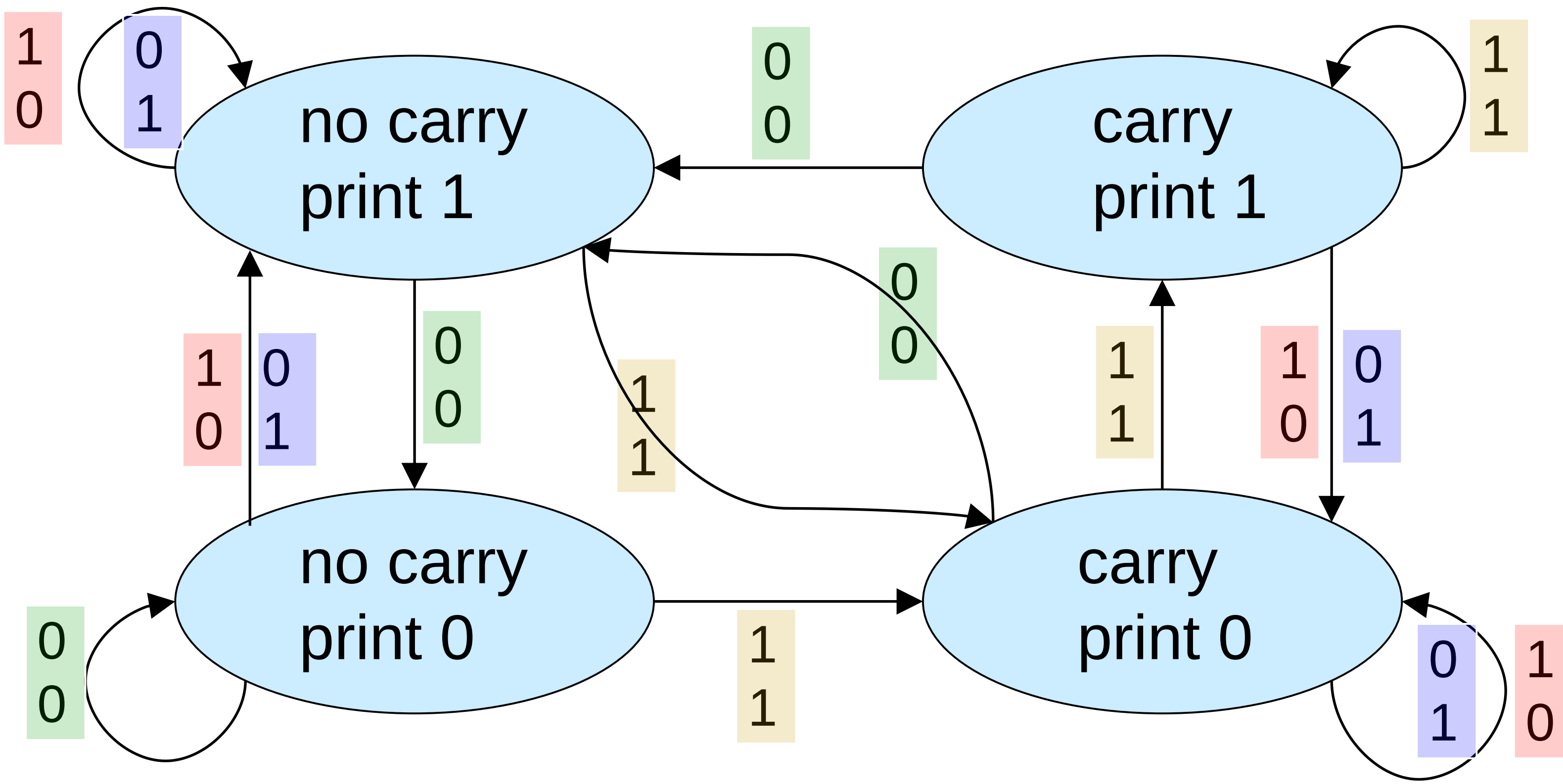
- We learn them in the same way as we learn the weights
  - Start off with initial random guess for the initial states
  - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state
  - Adjust initial states by following the negative gradient

A good problem for a recurrent network

- We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture:
  - We must decide in advance on maximum number of digits in each number
  - The processing applied to beginning of a long number does not generalize to the end of the long number because it uses different weights
- As a result, feedforward nets do not generalize well on the binary addition task



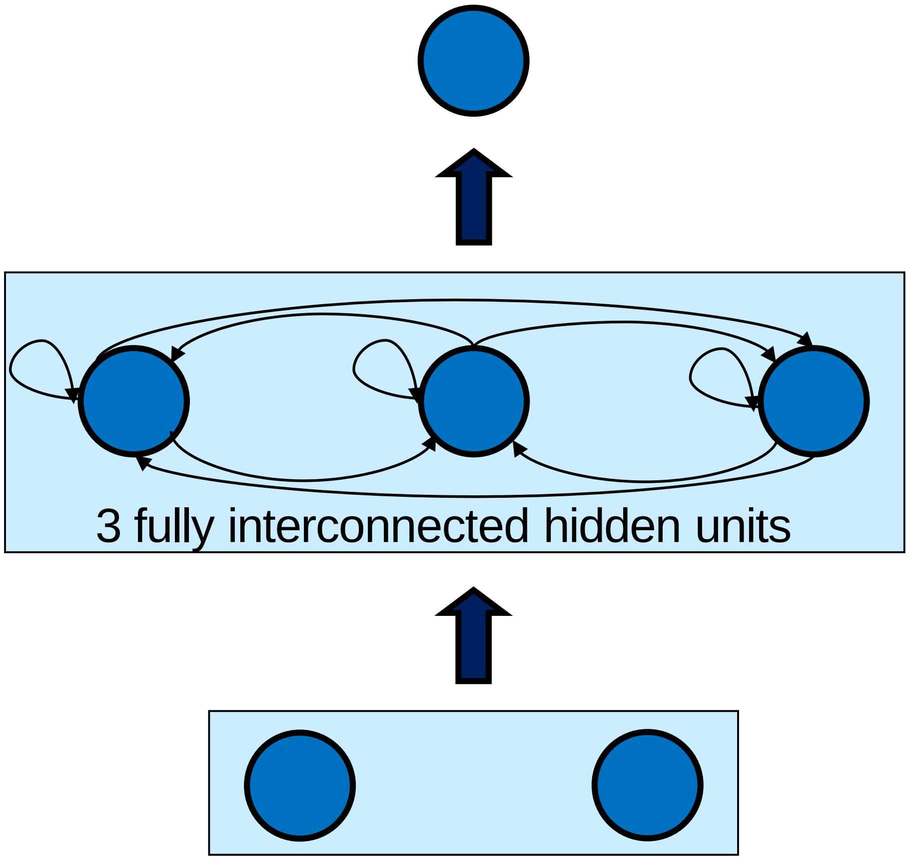
The algorithm for binary addition



Finite state automaton
- decides by looking at next column and prints after the transition
- moves from right to left over the two input numbers

The connectivity of the network

- The 3 hidden units have all possible interconnections in all directions
  - This allows a hidden activity pattern at one time step to *vote* for the hidden *activity pattern* at the *next time step*
- The input units have feedforward connections that allow them to vote for the next hidden activity pattern



A recurrent net for binary addition

- Network has two input units and one output unit at each time step
- Desired output at each time step is the output for the column that was provided as input two time steps ago.
  - It takes one time step to update the hidden units based on the two input digits.
  - It takes another time step for the hidden units to produce the output.

0 0 1 1 0 1 0 0

0 1 0 0 1 1 0 1

1 0 0 0 0 0 0 1

← **time**

What the network learns

- It learns *four distinct patterns of activity* for the 3 hidden units. These patterns correspond to the nodes in the finite state automaton
  - The automaton is restricted to be in exactly one *state* at each time. The hidden units are restricted to have exactly one *vector* of activity at each time

- A recurrent network can emulate a finite state automaton, but it is exponentially more powerful

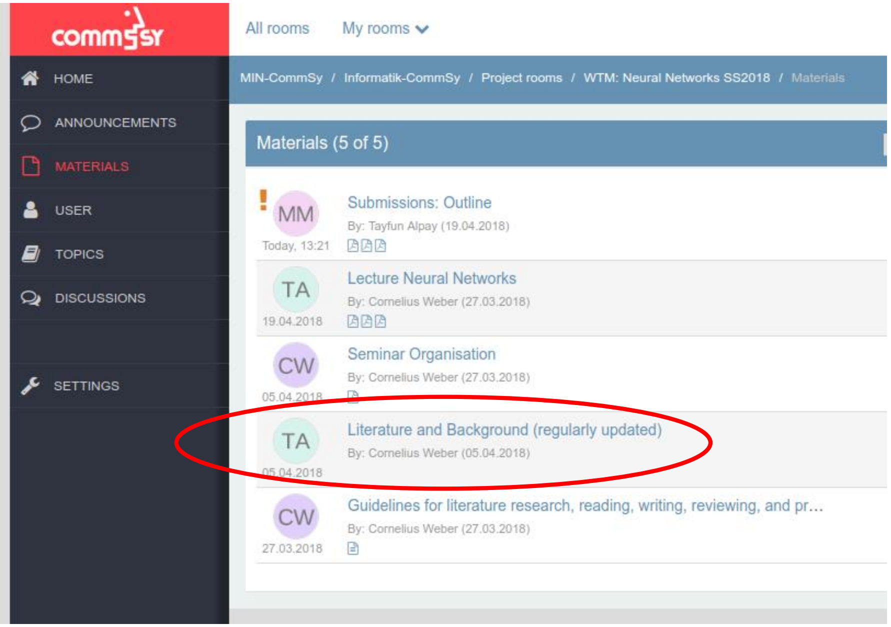
- With N hidden neurons it has 2^N possible binary activity vectors in the hidden units

Recurrent neural networks intermediate summary

- RNNs are very powerful, they are in fact equivalent to Turing machines
- One issue is the vanishing gradient problem
  - Recent advances in deep learning, learning algorithms, and highly specialized RNN architectures have however led to solutions and increasing performance
- RNNs are currently state of the art in sequence processing such as speech recognition, machine translation, handwriting recognition

Conclusion

- Recurrent Neural Networks (SRN and Jordan Network)
- Distributed and localist representations
- Next: Sequence Learning
- Reminder:
  Check the CommSy weekly for links on videos, tools, papers and interesting articles!



Recommended Reading

- Goodfellow (Chapter 10)
- Rojas (Chapter 7.4.1)
- Haykin (Chapter 15.1,15.2,15.6,15.7,15.12)