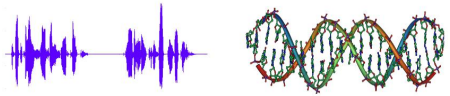
Idea

Use temporal covariations/context to refine prediction at some time t

Recurrent Neural networks

Sequence learning is the study of machine learning algorithms designed for sequential data



Sequential data: data points come in order and successive points may be **dependent**, e.g.,

- Letters in a word
- Words in a sentence/document
- Phonemes in a spoken word utterance
- Page clicks in a Web session
- Frames in a video, etc.

So far, we assume that data points in a dataset are i.i.d (independent and identically distributed)

Does not hold in many applications

Recurrent Neural networks

- Plain CNNs are not typically good at length-varying input and output.
- Difficult to define input and output
  - Remember that
    - Input image is a 3D tensor (width, length, color channels)
    - Output is a fixed number of classes.
  - Sequence could be:
    - "I know that you know that I know that you know that I know that you know that I know that you know that I know that you know that I know that you know that I don't know"
    - Or "I don't know"
- Input and output are strongly correlated within the sequence.
- (Still, people figured out ways to use CNN on sequence learning)

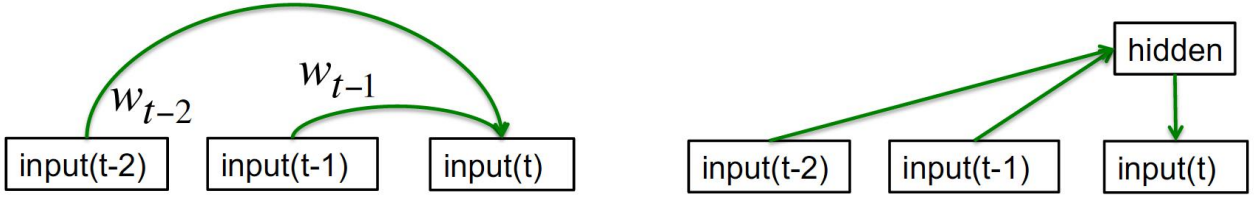
Sequence labeling

1. **Autoregressive models**

- Predict the next term in a sequence from a **fixed number of previous terms** using delay taps.

2. **Feed-forward neural nets**

- These **generalize autoregressive models** by using one or more layers of non-linear hidden units



Memoryless models: limited word-memory window; hidden state cannot be used efficiently.

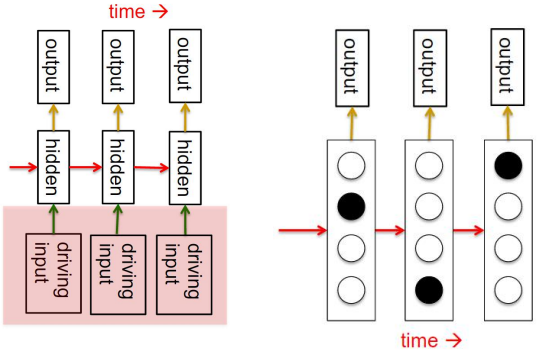
Sequence labeling

3**. Linear Dynamical Systems**

- Generative models. They have a hidden state that cannot be observed directly.

4. **Hidden Markov Models**

- Have a **discrete one-of-N hidden state**. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.

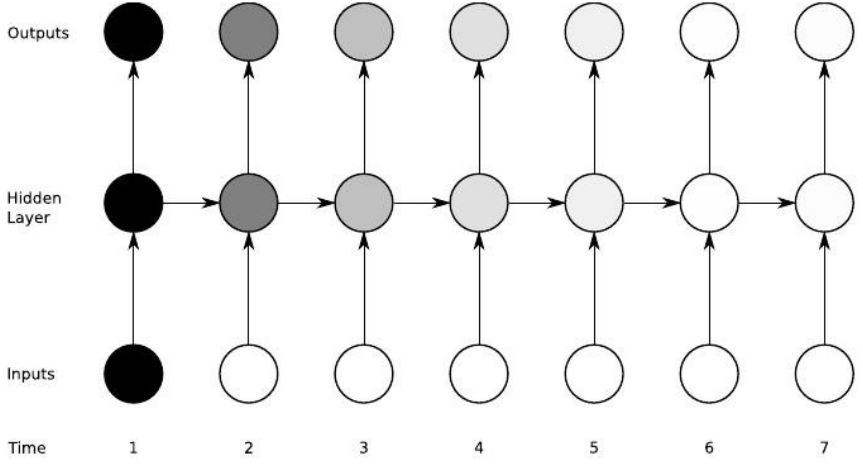


Memoryful models,
time-cost to infer the hidden state distribution.

Sequence labeling

Recurrent Neural Networks

- Update the hidden state in a deterministic nonlinear way.
- In a simple speaking case, we send the chosen word back to the network as input.



Vanilla forward path RNN

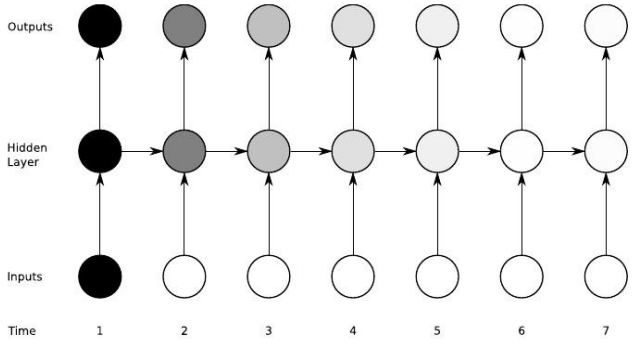
Recurrent Neural Networks

1. The forward pass of a vanilla RNN
    1. The same as that of a perceptron with a single hidden layer
    2. Except that activations arrive at the hidden layer from both the current external input and the hidden layer activations one step back in time.
2. For the input to hidden units we have

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1} \qquad b_h^t = \theta_h(a_h^t)$$

3. For the output unit we have

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$



Vanilla forward path RNN

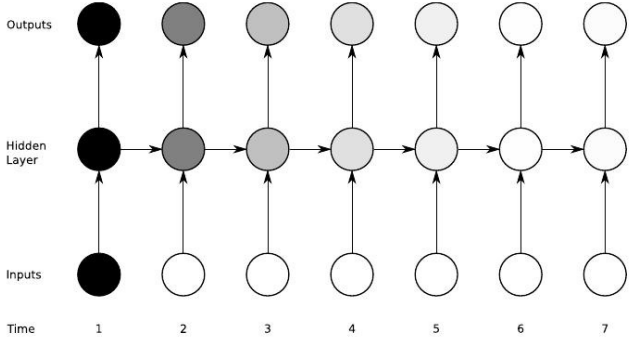
Recurrent Neural Networks

•The complete sequence of hidden activations can be calculated by starting at t = 1 and recursively applying the three equations, incrementing t at each step.

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$



Vanilla forward path RNN

Recurrent Neural Networks

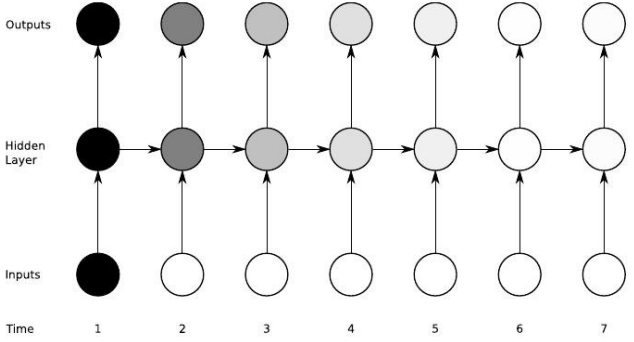
•Given the partial derivatives of the objective function with respect to the network outputs, we now need the derivatives with respect to the weights.

•We focus on BPTT since it is both conceptually simpler and more efficient in computation time (though not in memory). Like standard back-propagation, BPTT consists of a repeated application of the chain rule.

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

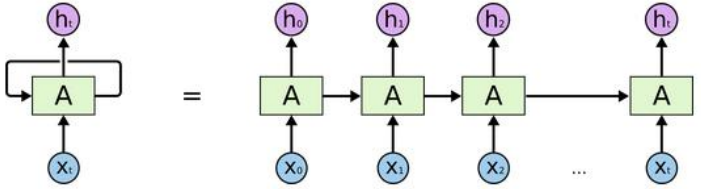
$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$



Vanilla backward path RNN

Recurrent Neural Networks

- Back-propagation through time
- Don't be fooled by the fancy name. It's just the standard back-propagation.



An unrolled recurrent neural network.

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'}\right),$$

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial a_j^t}$$

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

Vanilla backward path RNN

- Back-propagation through time
- The complete sequence of delta terms can be calculated by starting at t = T and recursively applying the below functions, decrementing t at each step.
- Note that $\delta_j^{T+1} = 0 \; \forall j,$ since no error is received from beyond the end of the sequence.

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right), \qquad a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial a_j^t} \qquad b_h^t = \theta_h(a_h^t)$$

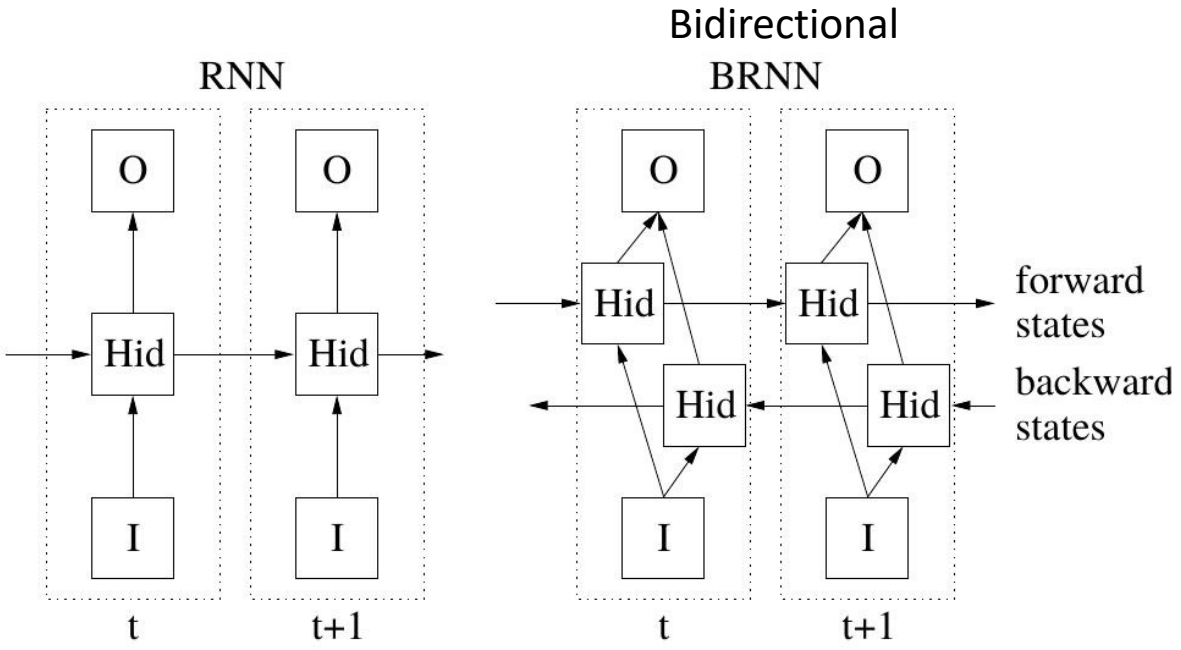
$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

- Finally, bearing in mind that the weights to and from each unit in the hidden layer are the same at every time-step, we sum over the whole sequence to get the derivatives with respect to each of the network weights

$$\frac{\partial O}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial O}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

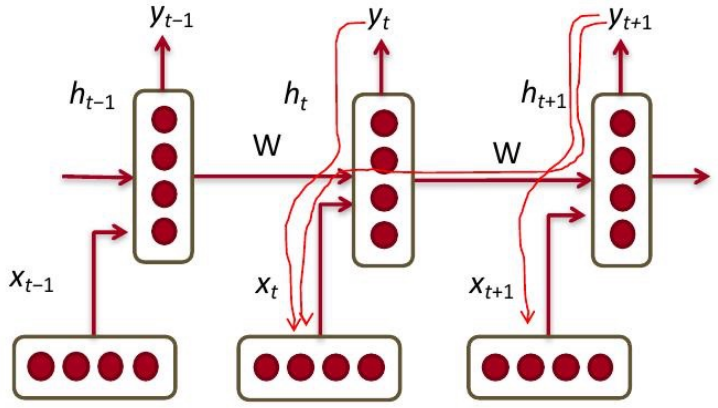
Vanilla bidirectional path RNN

•For many sequence labeling tasks, we would like to have access to future.



Problem: Vanishing and exploding gradients

- Multiply the same matrix at each time step during back- prop



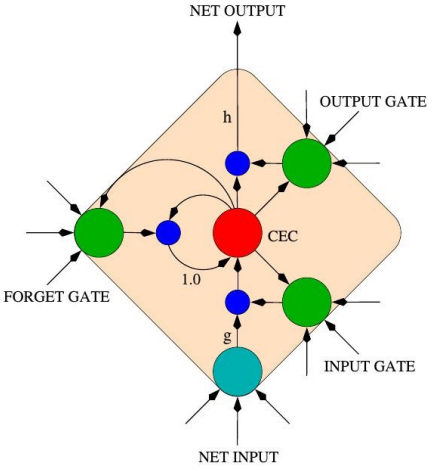
LSTM: long short term memory

- As discussed earlier, for standard RNN architectures, the range of context that can be accessed is limited.
- The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections.
- The **most effective solution so far is the Long Short Term Memory** (LSTM) architecture (Hochreiter and Schmidhuber, 1997).
- The LSTM architecture consists of a set of recurrently connected subnets, known as **memory blocks**. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each block contains one or more self-connected memory cells and three multiplicative units that provide continuous analogues of **write, read and reset operations for the cells**
- The **input, output and forget gates**.

LSTM: long short term memory

•The multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby avoiding the vanishing gradient problem

•For example, as long as the input gate remains closed (i.e. has an activation close to 0), the activation of the cell will not be overwritten by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate.



LSTM: long short term memory

- Comparison

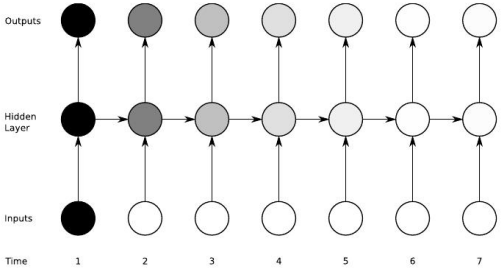


Figure 4.1: **Vanishing gradient problem for RNNs.** The shading of the nodes indicates the sensitivity over time of the network nodes to the input at time one (the darker the shade, the greater the sensitivity). The sensitivity decays exponentially over time as new inputs overwrite the activation of hidden unit and the network 'forgets' the first input.

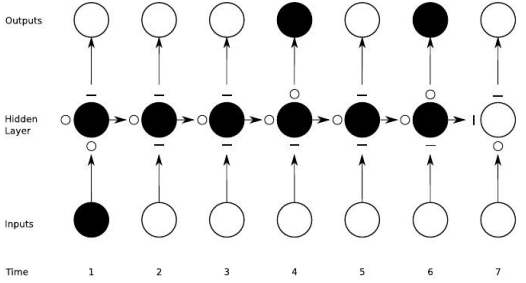


Figure 4.3: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the input unit at time one. The state of the input, forget, and output gate states are displayed below, to the left and above the hidden layer node, which corresponds to a single memory cell. For simplicity, the gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed, and the sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.