CSE 5526: Introduction to Neural Networks

Recurrent Networks

Motivation

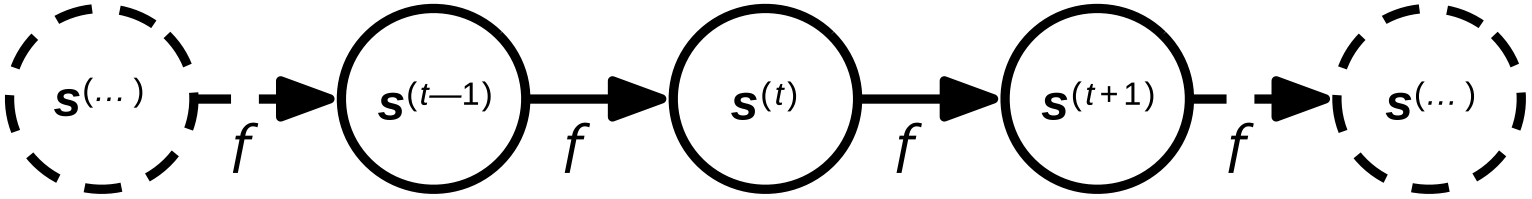
- Conventional neural networks have a fixed number of input dimensions and outputs
- Sequences often have variable lengths in which the different items (dimensions) of the sequence are related to one another
  - Words in a sentence
  - Values in a time series
  - Biological sequences
- Recurrent neural networks (RNNs) address such tasks
  - RNNs take the previous outputs or hidden states as inputs
  - The composite input at time *t* provides historical information about the activities prior to *t*

Classical dynamical systems

- Consider the classical form of a dynamical system:

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

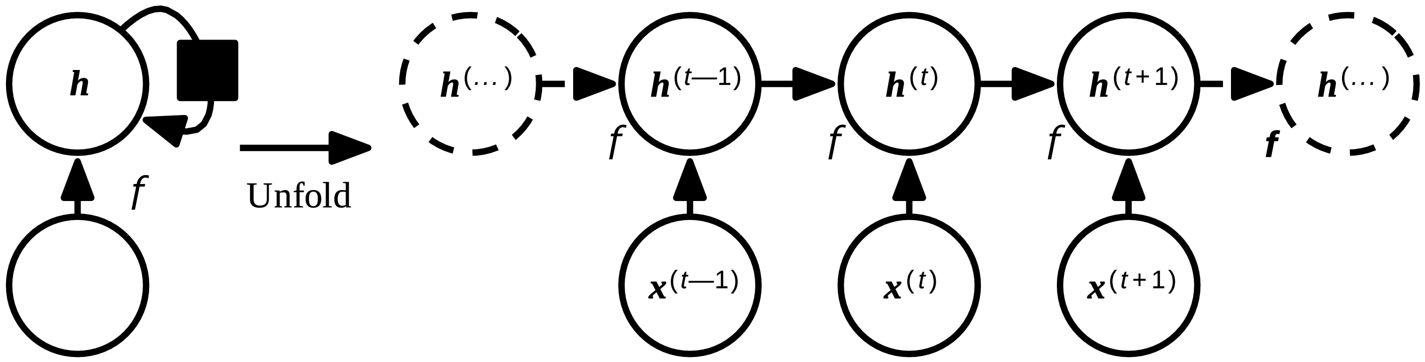
  - $\boldsymbol{s}^{(t)}$: the state of the system at t
  - $\boldsymbol{\theta}$: the parameters of the system



Unfolding computations

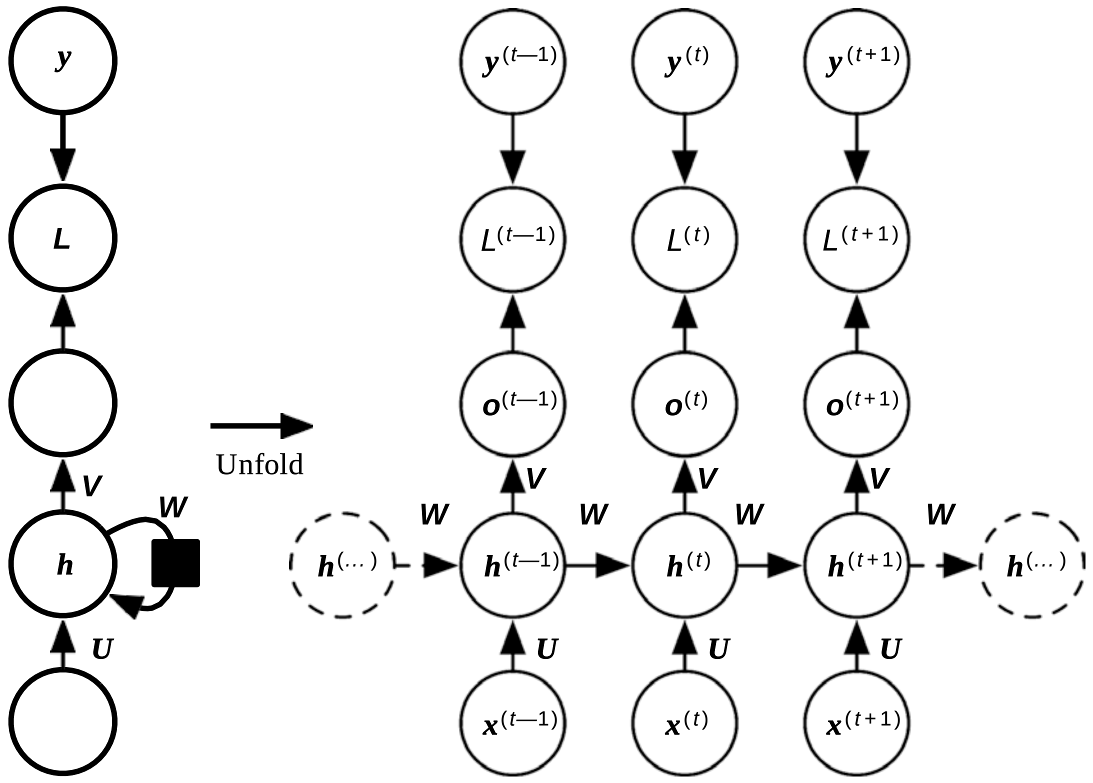
- There are many ways to create recurrent connections
- A recurrent network with no output

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$



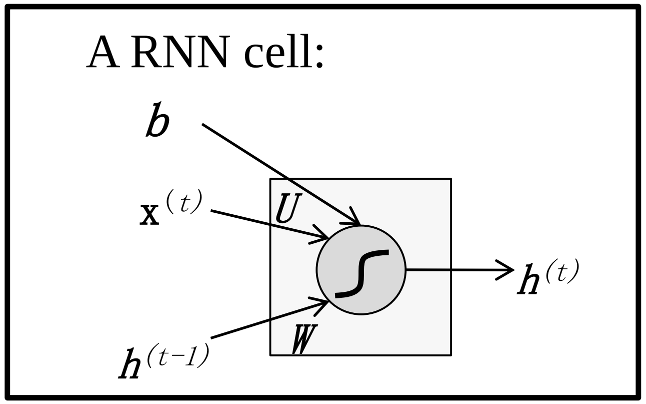
Recurrent hidden units

- ***y***: desired output
- *L*: loss (cost)

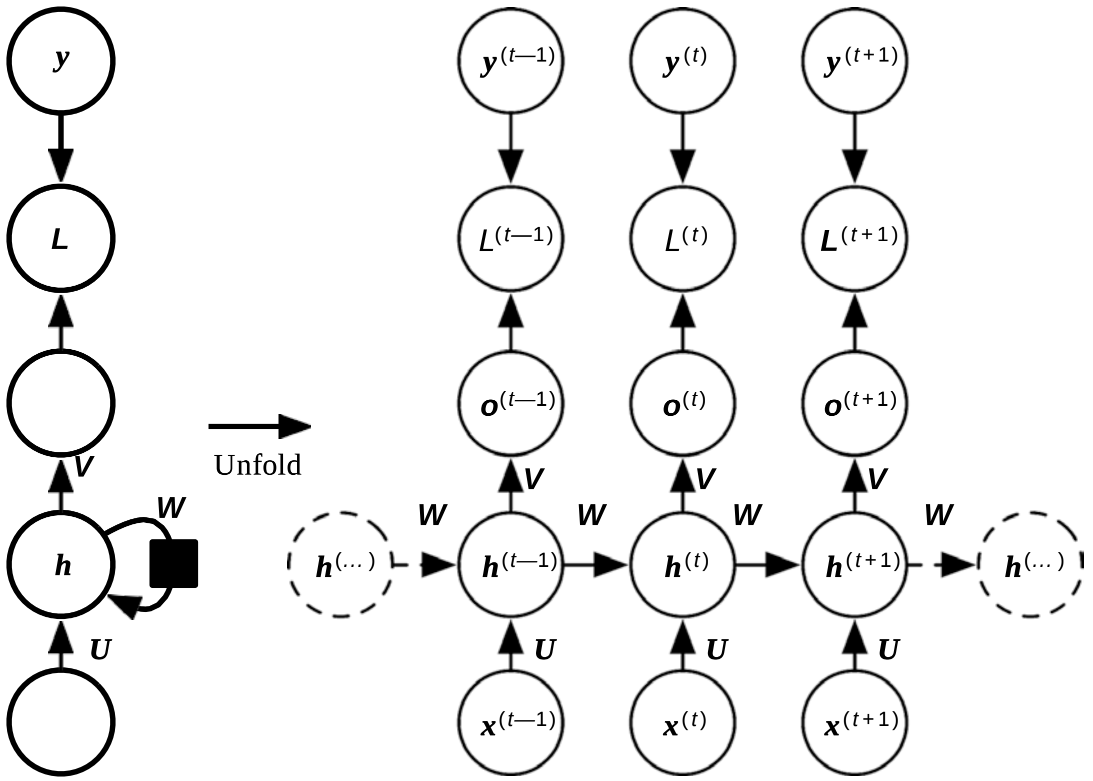


A typical RNN cell

- $\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$
  $= \tanh(\boldsymbol{v}^{(t)})$
- $\boldsymbol{o}^{(t)} = \boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}$
- $\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$
  - $\boldsymbol{b}, \boldsymbol{c}$: bias vectors
  - $\mathrm{softmax}(\boldsymbol{v}_i) = \frac{\exp(v_i)}{\sum_j \exp(v_j)}$

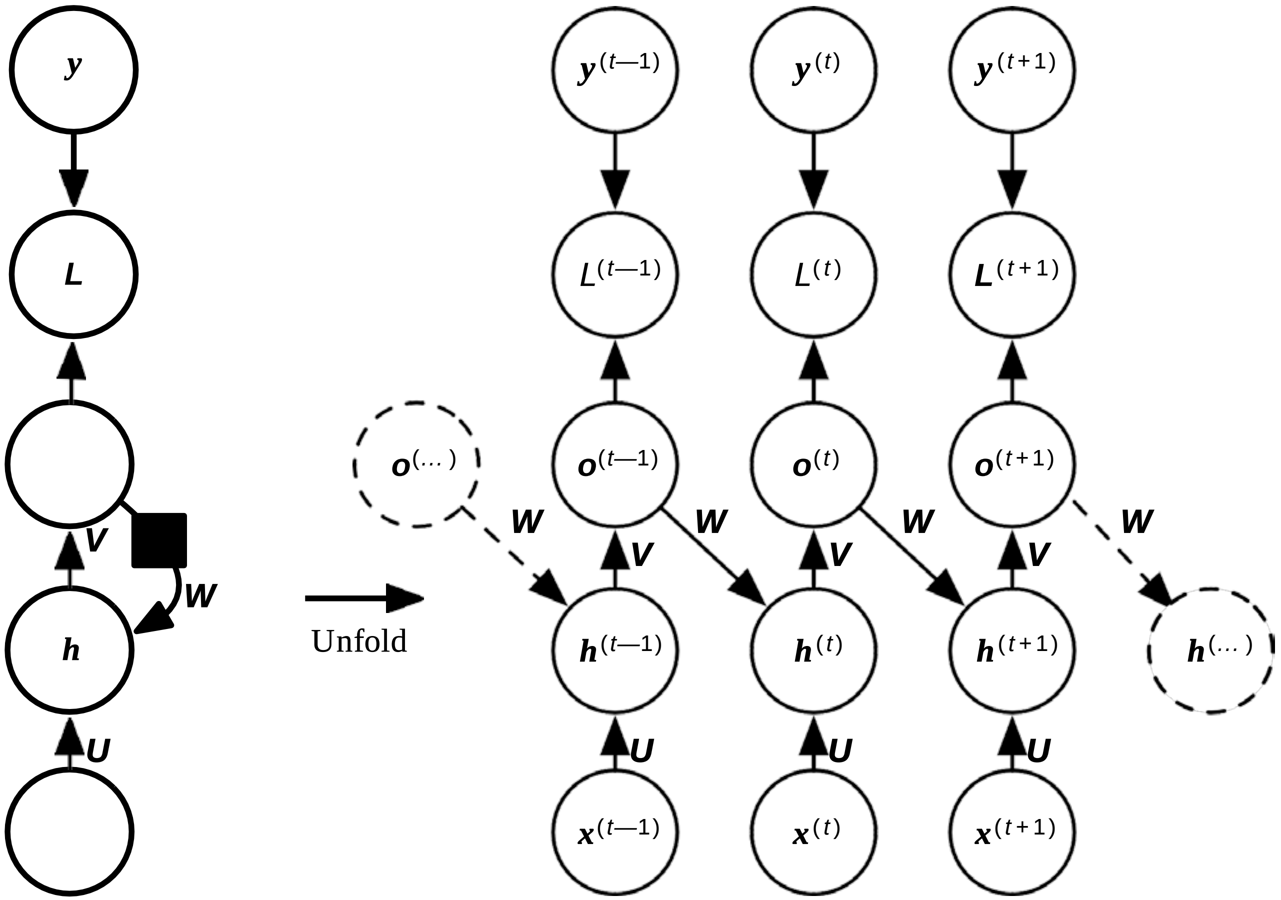


Recurrent hidden units (cont.)

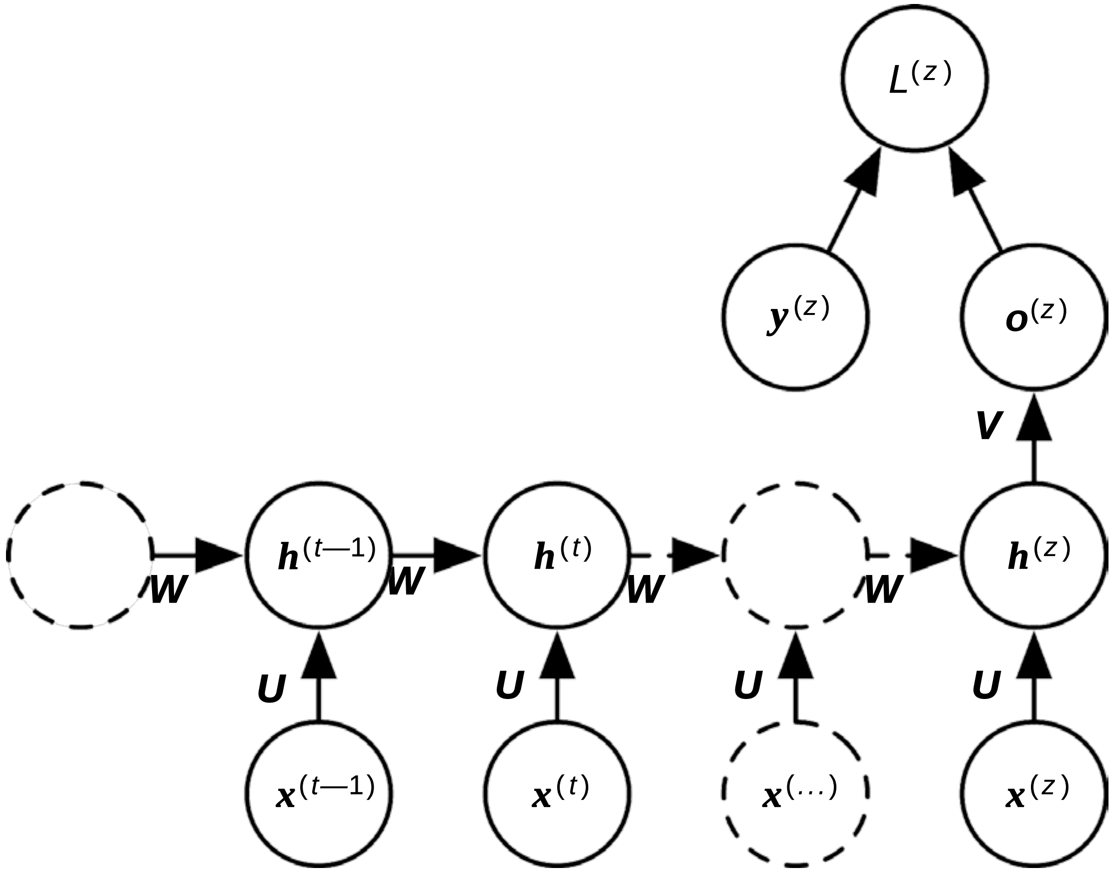


- Treat the unfolded network as one big feedforward network
- This big network takes in an entire sequence as input
- Compute gradients through usual backpropagation
- Update shared weights

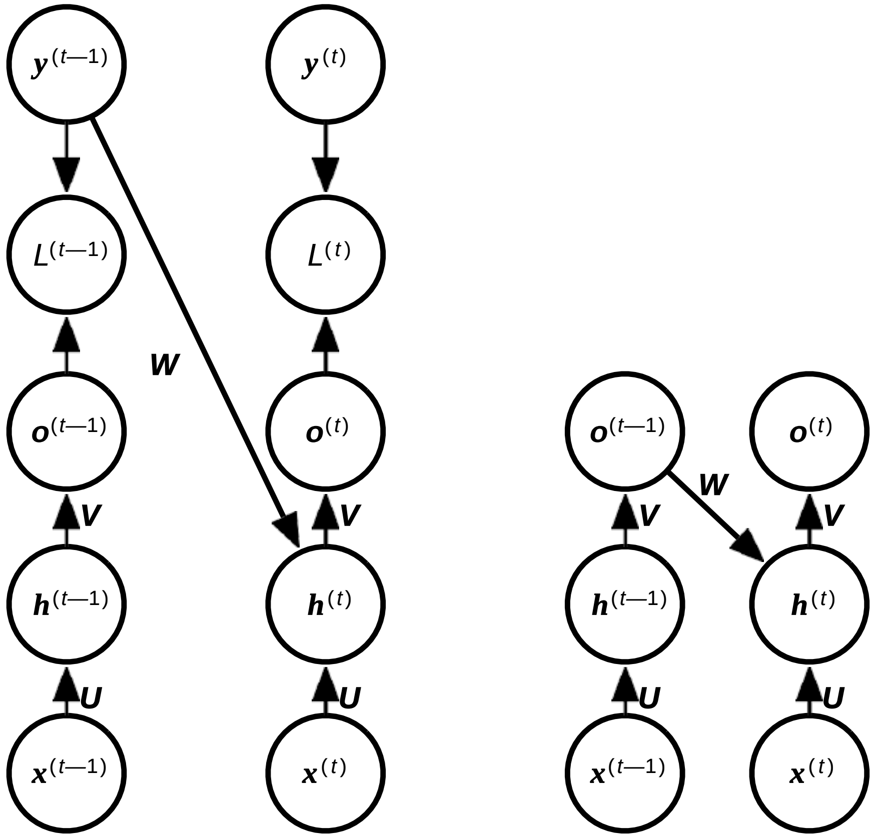
Recurrence through output only



Sequence input, single output



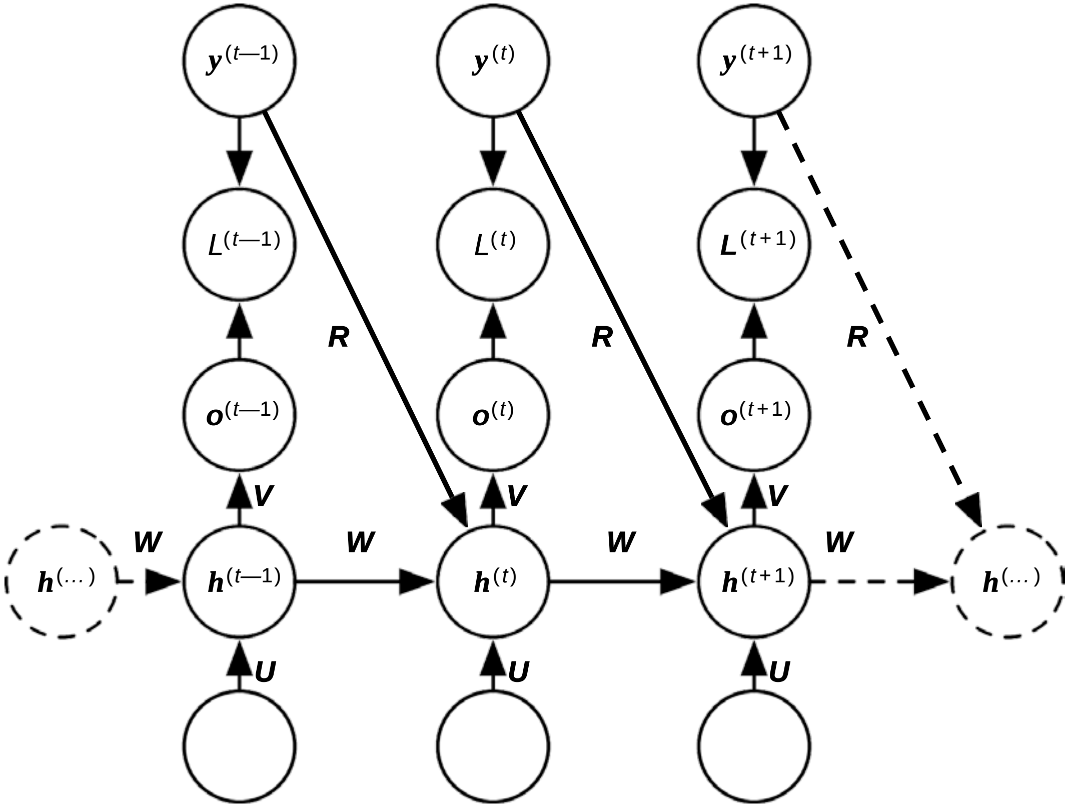
Teacher forcing



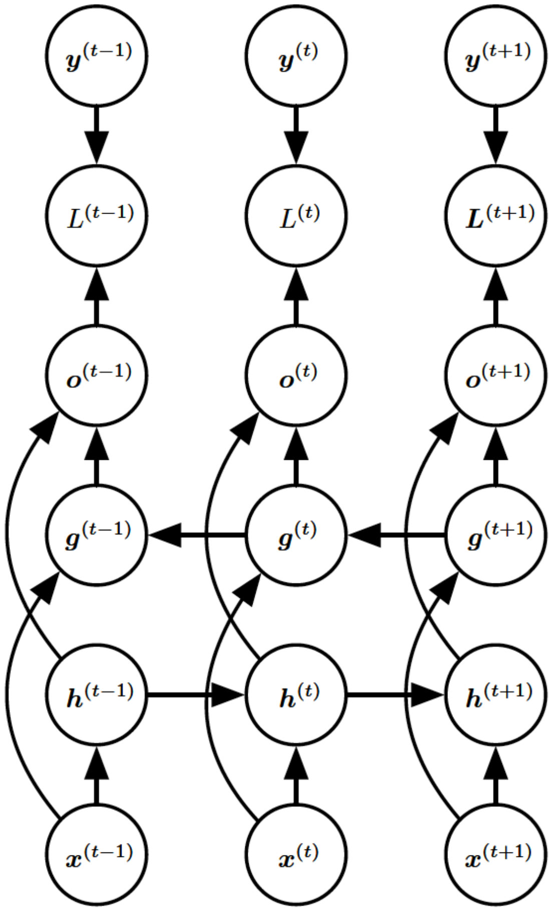
Training time　　　　Test time

Hidden and output recurrence

- The output values are not forced to be conditionally independent in this model



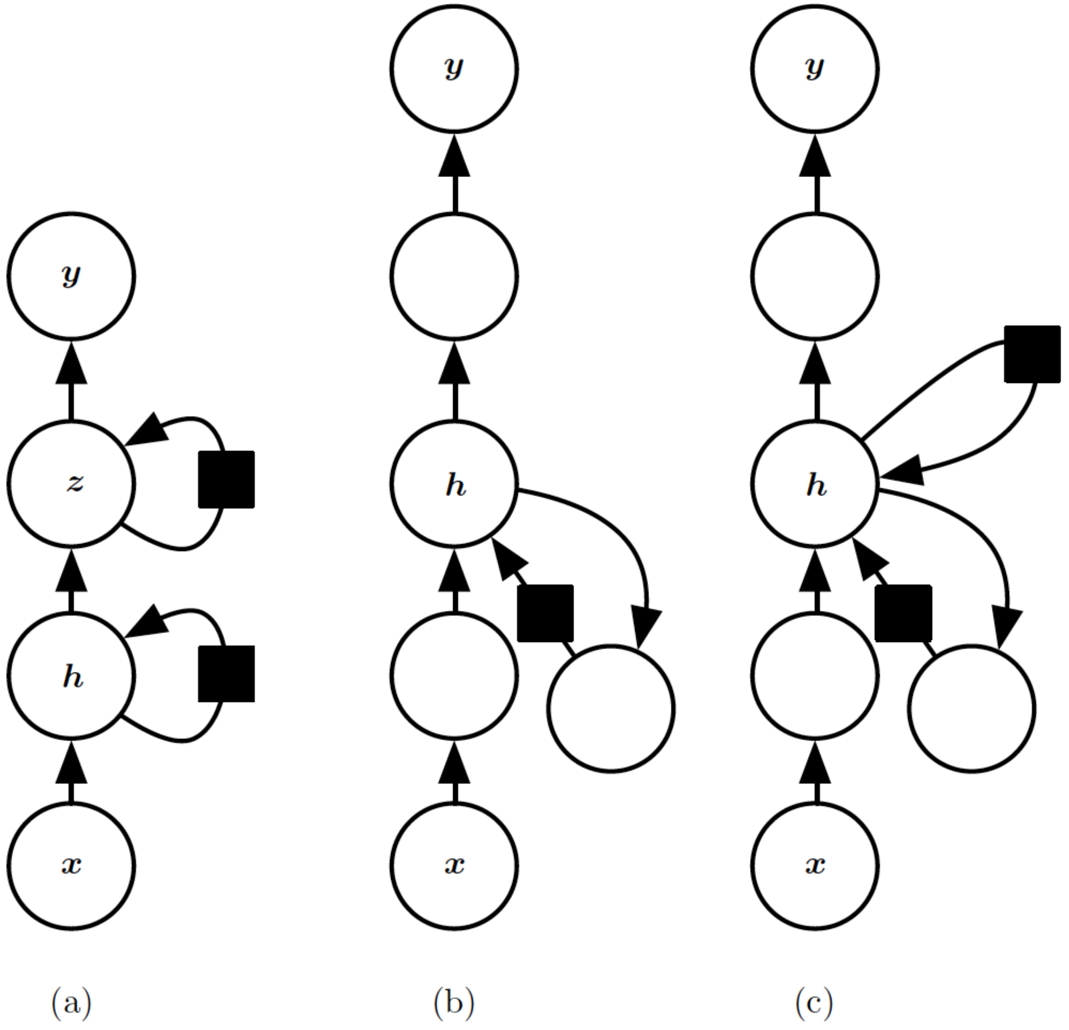
Bidirectional RNN



Backward states

Forward states

Deep RNNs

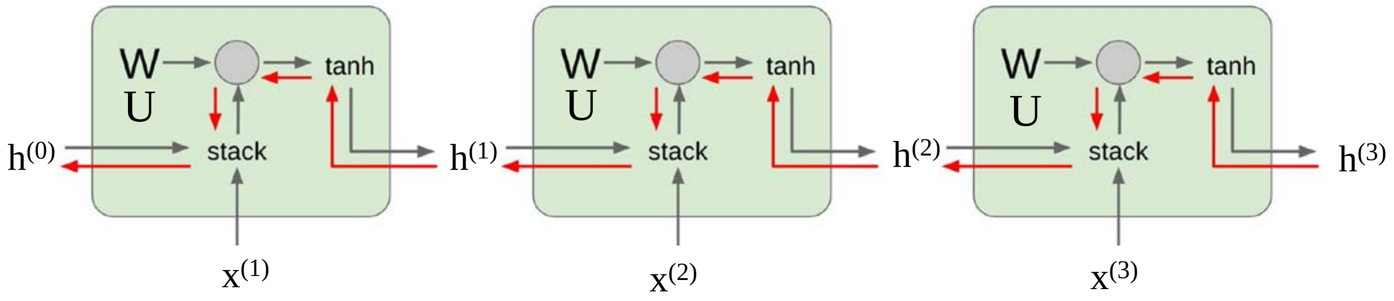


RNN training

- Essentially, an RNN cell is copied over time (unrolling or unfolding), with different inputs at different time steps
  - The weights are shared over time
- Key difference from traditional backpropagation is the use of shared parameters
  - Pretend that the weights are not shared and apply normal backpropagation to compute the gradients with respect to each copy of the weights
  - Summate (or average) the gradients over the various copies of each weight copy
  - Perform gradient descent update
- This algorithm is referred to as backpropagation through time (BPTT)

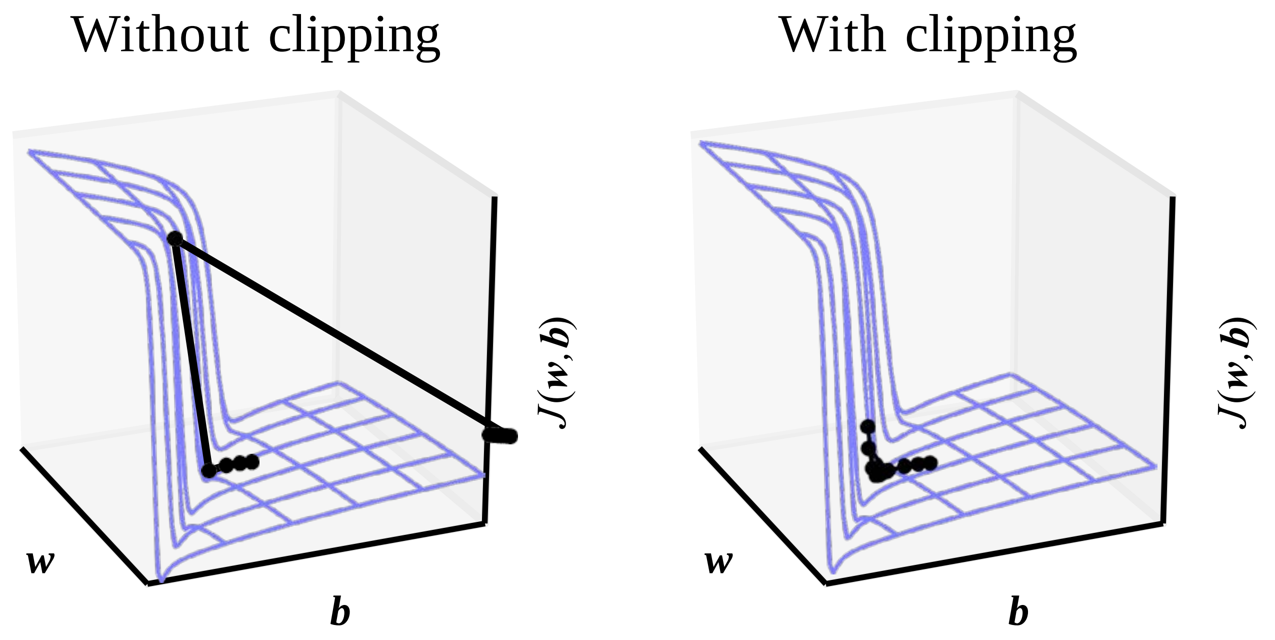
Exploding and vanishing gradients

- Backpropagated partial derivatives get multiplied by weights and derivatives of activation functions



- Unless the values are exactly one, these partial derivatives either continuously increase (explode) or decrease (vanish)
- Exploding gradients: gradient clipping, i.e. scaling a gradient if its magnitude is too large
- Vanishing gradients: changing RNN architecture

Gradient clipping illustration



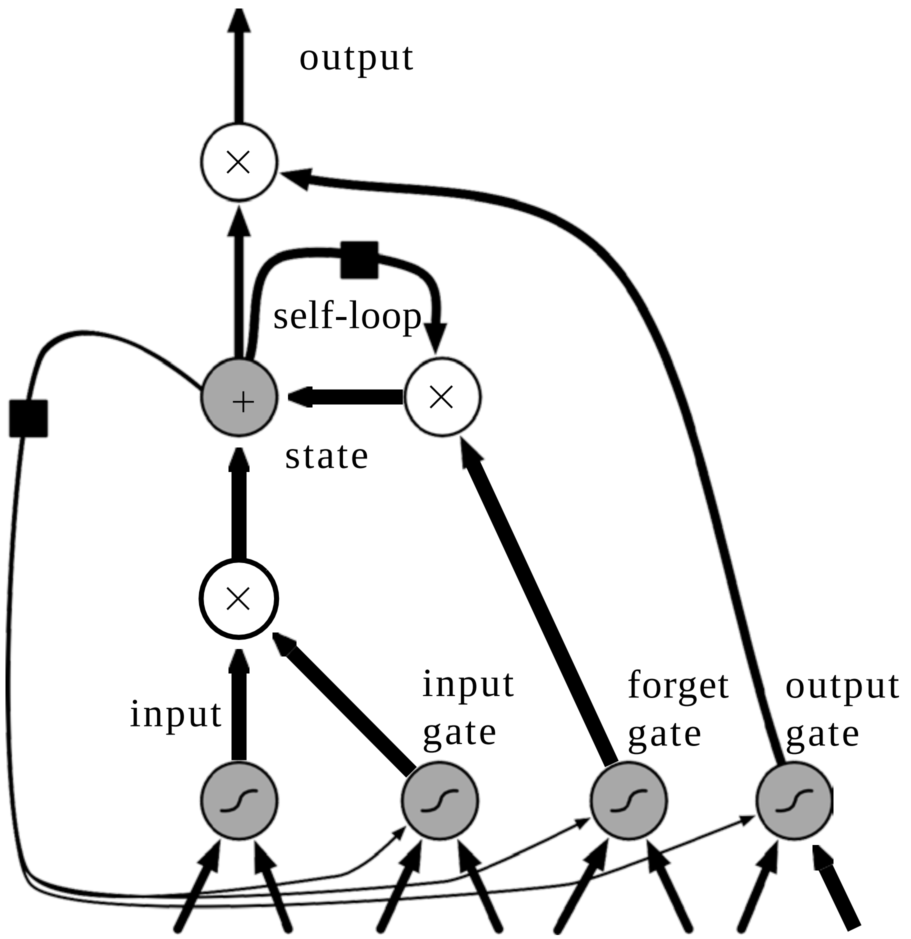
- **Without clipping**: Gradient descent overshoots the bottom of a small ravine, and then receives a very large gradient from the cliff surface. The large gradient drives the parameters outside the plot, far away from the solution
- **With clipping**: Gradient descent has a moderate reaction to the cliff. While it does ascend the cliff surface, the step size is restricted so that it cannot be driven away from the steep region near the solution

RNN with long short-term memory

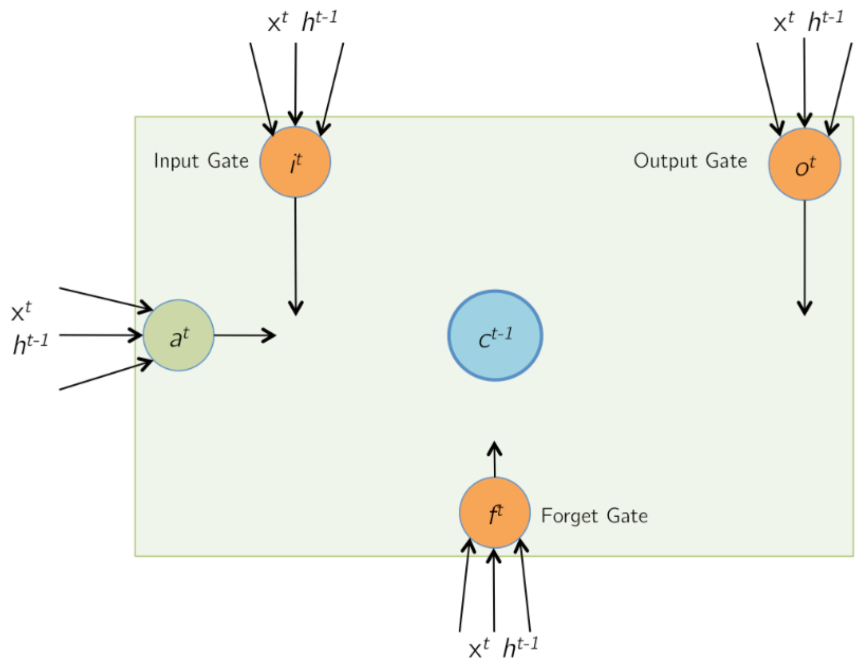
- With vanishing gradients, it is difficult to train RNNs to solve problems that require learning long-term temporal dependencies
- The most effective solution currently is gated RNNs, as exemplified by the long short-term memory (LSTM) architecture

LSTM unit

- An LSTM unit (cell) includes "memory" to maintain information for long periods of time
- A set of gates is used to control when information enters the memory, when it is output, and when it is forgotten
  - ***f***: forget gate, how much to erase
  - ***i***: input gate, how much to write to the unit
  - ***a***: input activation
  - ***o***: output gate, how much to output
  - ***c***: cell state vector
- This architecture lets the unit learn longer-term dependencies



LSTM: Forward pass



- Input and gate computation

$$\boldsymbol{a}^{(t)} = \tanh\left(\boldsymbol{W}_a \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_a \boldsymbol{x}^{(t)} + \boldsymbol{b}_a\right) = \tanh(\boldsymbol{v}_a^{(t)})$$

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i) = \sigma(\boldsymbol{v}_i^{(t)})$$

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right) = \sigma(\boldsymbol{v}_f^{(t)})$$

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o) = \sigma(\boldsymbol{v}_o^{(t)})$$

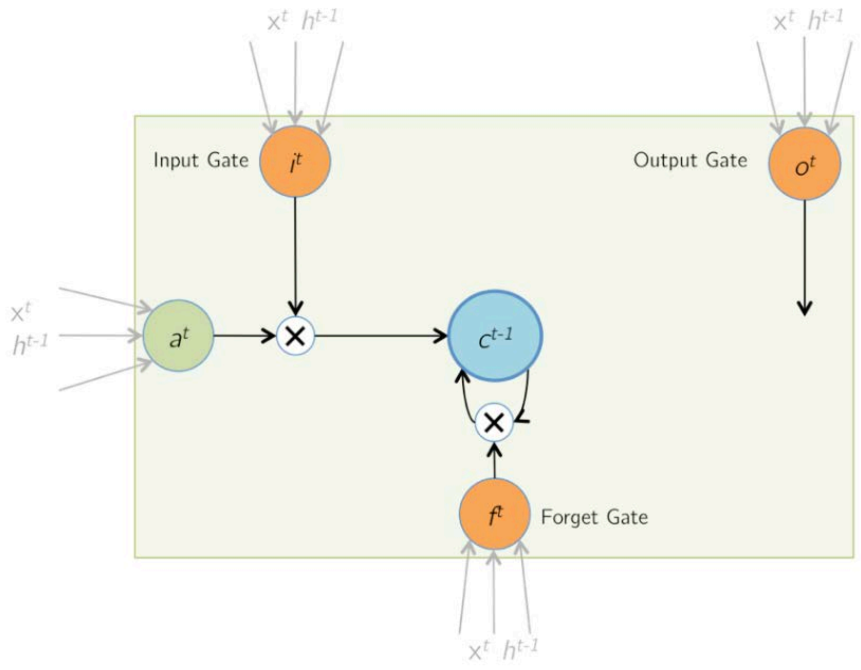
  - σ indicates a sigmoid

In other words

$$\boldsymbol{v}^{(t)} = \begin{bmatrix} \boldsymbol{v}_a^{(t)} \\ \boldsymbol{v}_i^{(t)} \\ \boldsymbol{v}_f^{(t)} \\ \boldsymbol{v}_o^{(t)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{W}_a & \boldsymbol{U}_a \\ \boldsymbol{W}_i & \boldsymbol{U}_i \\ \boldsymbol{W}_f & \boldsymbol{U}_f \\ \boldsymbol{W}_o & \boldsymbol{U}_o \end{bmatrix} \begin{bmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_a \\ \boldsymbol{b}_i \\ \boldsymbol{b}_f \\ \boldsymbol{b}_o \end{bmatrix}$$

$$= \boldsymbol{W}\boldsymbol{I}^{(t)} + \boldsymbol{b}$$

LSTM: Forward pass (cont.)

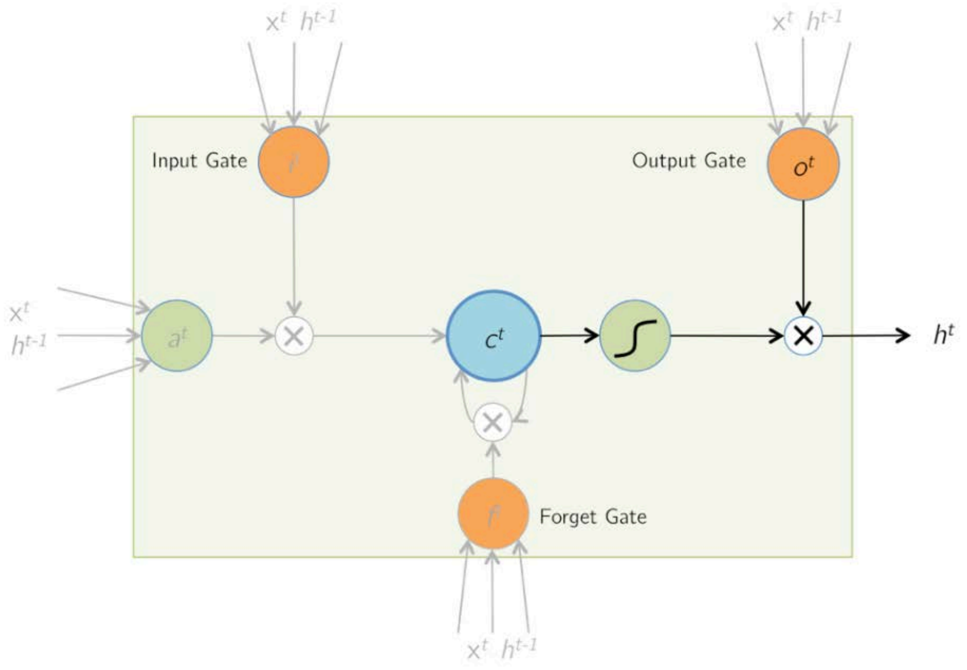


- Memory update: the state of the unit is updated to the latest values

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \boldsymbol{a}^{(t)}$$

  - Symbol $\odot$ denotes elementwise product

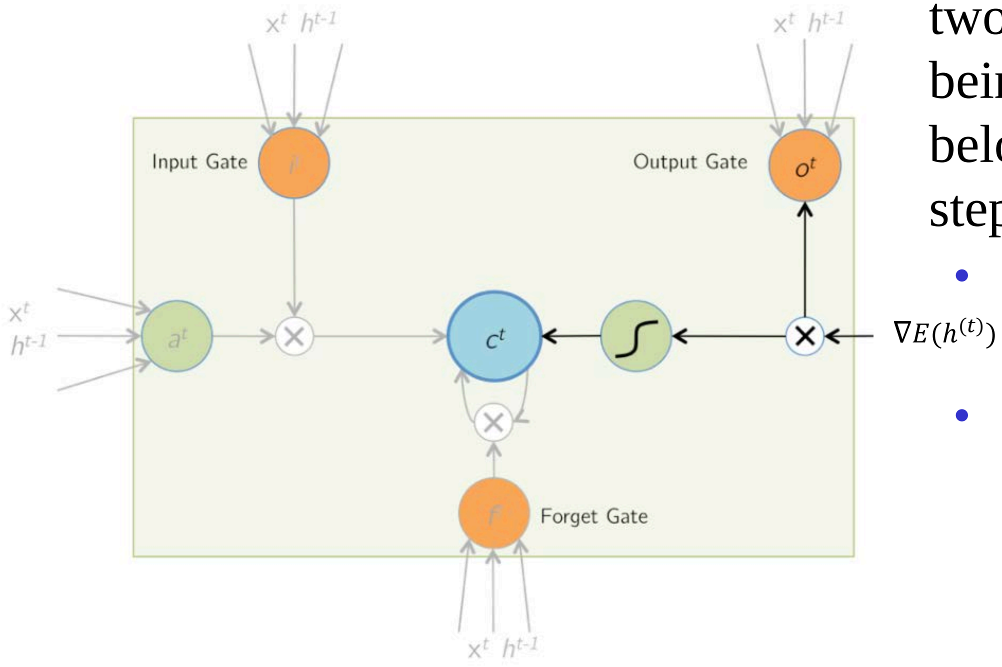
LSTM: Forward pass (cont.)



- Output: finally, the LSTM unit computes an output value by applying an activation function (nonlinearity) to the updated state value

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)})$$

LSTM: Backward pass



- The cell state at time *t*, $\boldsymbol{c}^{(t)}$, receives gradients from $\boldsymbol{h}^{(t)}$ as well as the next cell state $\boldsymbol{c}^{(t+1)}$. At time step *t*, these two gradients are accumulated before being backpropagated to the layers below the cell and the previous time steps
  - Forward pass: $\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)})$
  - Given $\nabla E(\boldsymbol{h}^{(t)}) = \frac{\partial E}{\partial \boldsymbol{h}^{(t)}}$, find $\nabla E(\boldsymbol{c}^{(t)})$

$$\frac{\partial E}{\partial c_i^{(t)}} = \frac{\partial E}{\partial h_i^{(t)}} \frac{\partial h_i^{(t)}}{\partial c_i^{(t)}} = \nabla E(h_i^{(t)}) o_i^{(t)} \left(1 - \tanh^2\left(c_i^{(t)}\right)\right)$$

Thus $\nabla E(\boldsymbol{c}^{(t)}) \mathrel{+}= \nabla E(\boldsymbol{h}^{(t)}) \odot \boldsymbol{o}^{(t)} \odot \Big(1 -$

Symbol += indicates this gradient is added to the gradient from (*t*+1) (i.e. gradient accumulation)

Summary

- RNNs are uniquely suited for sequence modeling and temporal (dynamical) tasks
- RNN training is more difficult than training a feedforward net
- Gated RNNs, particularly RNNs with LSTM, are very successful in real world applications