On Finding the Structure in Time

Alexander G. Ororbia II

Presented by Cyril Weerasooriya

Introduction to Machine Learning

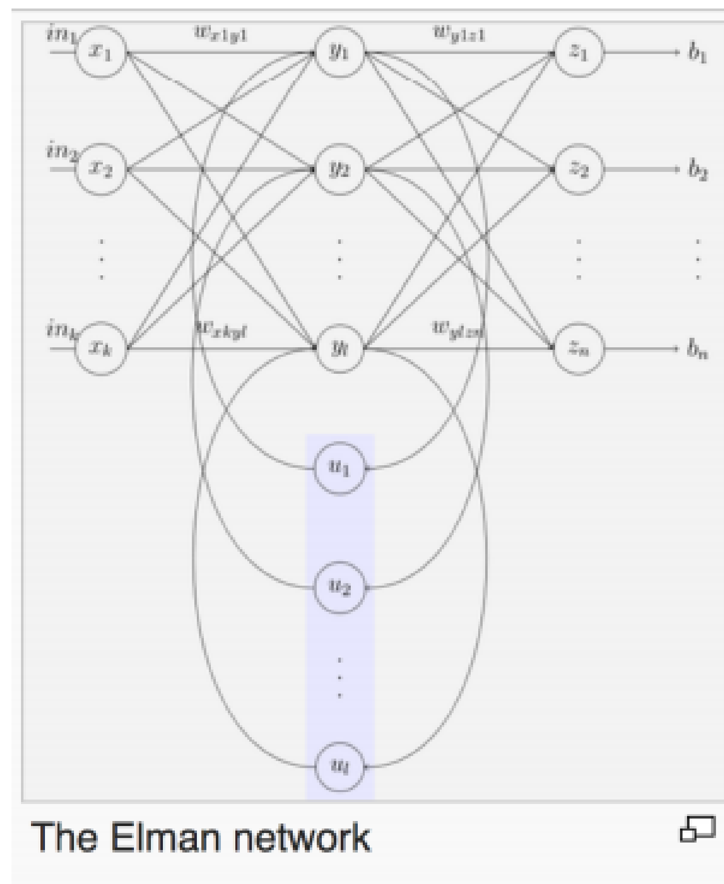
CSCI-635

2/9/2023

Without “time funnels” or “temporal queues”…

RNN as a network with cycles

- An RNN is a class of neural networks where connections between units form a directed cycle
- This creates an internal state of the network which allows it to exhibit dynamic temporal behavior
- The internal memory can be used to process arbitrary sequences of inputs



The Elman network

Three layer network with input x, hidden layer y and output z. Context units u maintain a copy of the previous value of the hidden units

Transformers

- Next generation of RNN
- Source https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

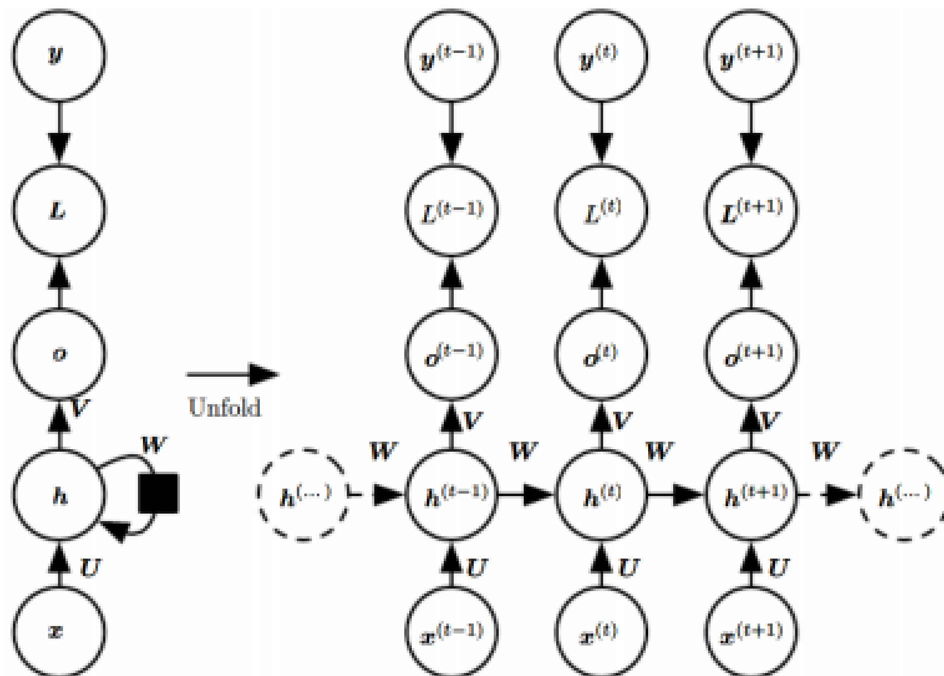
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

RNN operating on a sequence

- RNNs operate on a sequence that contain vector $\boldsymbol{x}^{(t)}$ with time step index t, ranging from 1 to τ
    - Sequence: $\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(\tau)}$
    - RNNs operate on minibatches of sequences of length τ
- Some remarks about sequences
    - The steps need not refer to passage of time in the real world
    - RNNs can be applied in two-dimensions across spatial data such as image
    - Even when applied to time sequences, network may have connections going backwards in time, provided entire sequence is observed before it is provided to network

Extension of Computational Graphs

- To study RNNs extend computational graphs to include cycles
  - These cycles represent the influence of the present value of a variable on its own value at a future time step
  - Such computational graphs allow us to define RNNs
  - We then define ways to construct, train and use RNNs



Computational graph of RNN that maps an input sequence of $\boldsymbol{x}$ values to corresponding output sequence of $\boldsymbol{o}$ values. Loss $\boldsymbol{L}$ measures how far each output $\boldsymbol{o}$ is from the training target $\boldsymbol{y}$

Forward propagation is given as follows. For each time step t, t=1 to t=τ

Apply the following equations

$\boldsymbol{o}^{(t)}=\boldsymbol{c}+V\boldsymbol{h}^{(t)}$

$\boldsymbol{h}^{(t)}=\tanh(\boldsymbol{a}^{(t)})$

$\boldsymbol{a}^{(t)}=\boldsymbol{b}+W\boldsymbol{h}^{(t-1)}+U\boldsymbol{x}^{(t)}$

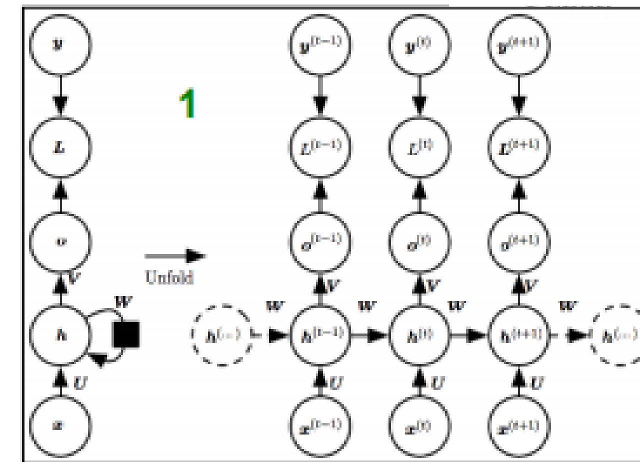
Unrolling Recurrent Nets

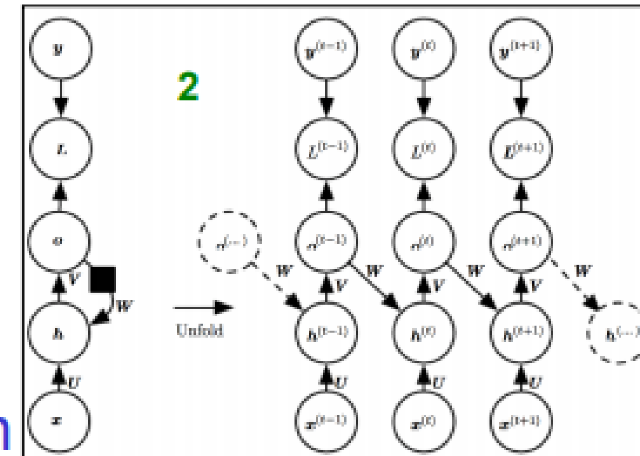
- In RNNs, at each time step the network takes as input its previous state $s^{(t-1)}$ and its current input $x^{(t)}$ and produces an output $y^{(t)}$ and a new hidden state $s^{(t)}$.

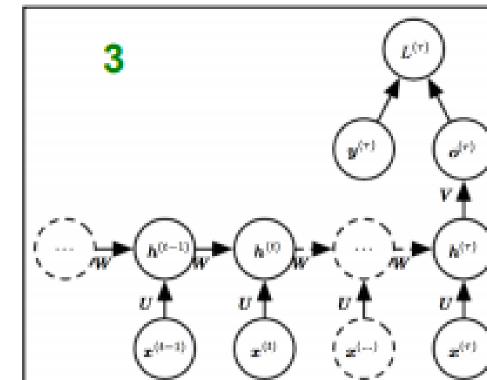
- With RNNs, you can 'unroll' the net and think of it as a large feedforward net with inputs $x^{(0)}$, $x^{(1)}$, ..., $x^{(T)}$, initial state $s^{(0)}$, and outputs $y^{(0)},y^{(1)},\ldots,y^{(T)}$, with T varying depending on the input data stream, and the weights in each of the cells tied with each other.

Three design patterns of RNNs

1. Produce output at each time step and have recurrent connections between hidden units
2. Produce output at each time step and have recurrent connections only from output at one time step to hidden units at next time step
   - Less powerful than (1) but easier to train
     - Each step can be trained in isolation
     - Greater parallelization during training
3. Recurrent connections between hidden units that read an entire input sequence and produce a single output
   - Can summarize a sequence and produce a fixed size representation for further processing



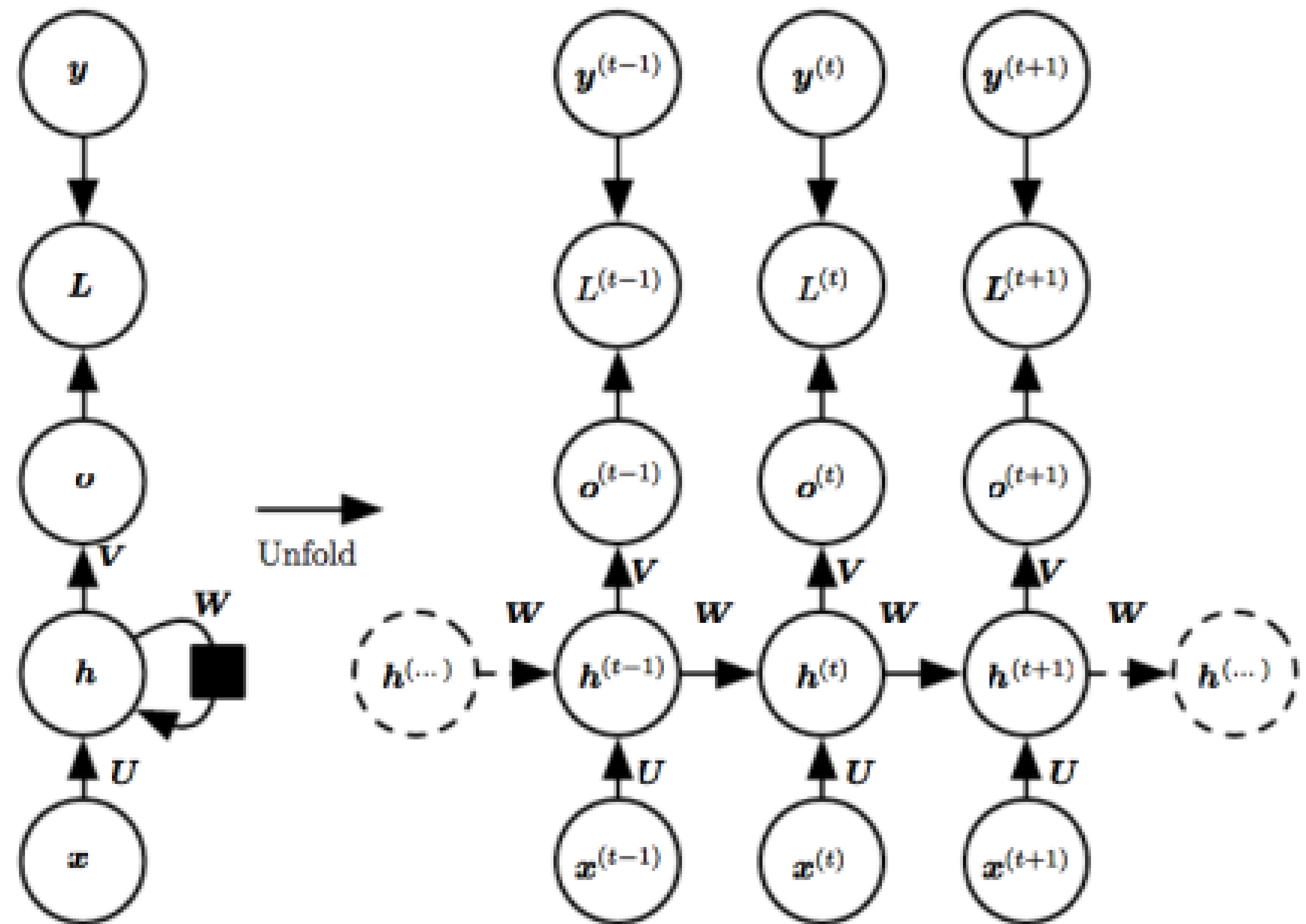




RNN with recurrence between hidden units

- Maps input sequence $\boldsymbol{x}$ to output $\boldsymbol{o}$ values
  - Loss L measures how far each $\boldsymbol{o}$ is from the corresponding target $\boldsymbol{y}$
    - With softmax outputs we assume $\boldsymbol{o}$ is the unnormalized log probabilities
    - Loss L internally computes $\boldsymbol{y}=\text{softmax}(\boldsymbol{o})$ and compares to target $\boldsymbol{y}$
- Update equation

$$\boldsymbol{a}^{(t)}=\boldsymbol{b}+W\boldsymbol{h}^{(t-1)}+U\boldsymbol{x}^{(t)}$$



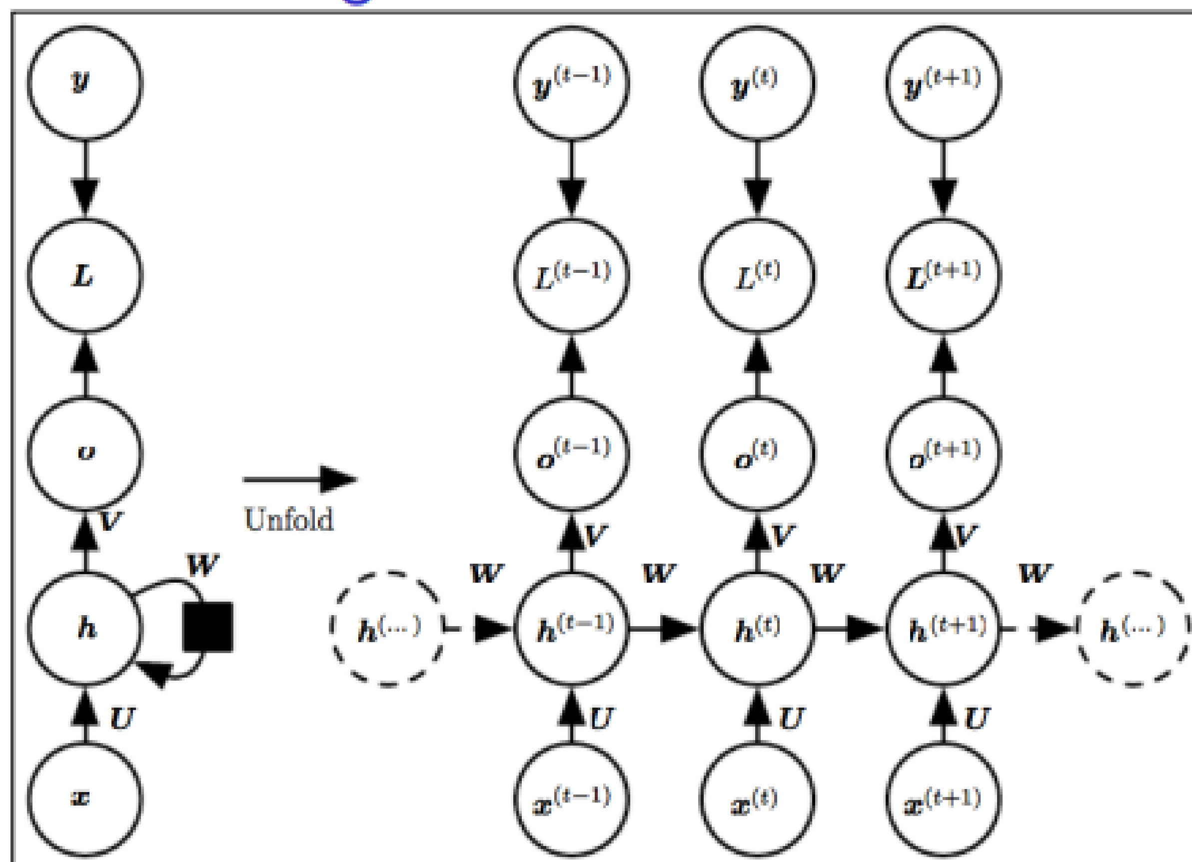
RNN with hidden unit recurrence is a Universal TM

- RNN with hidden unit connections together with

  $a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$

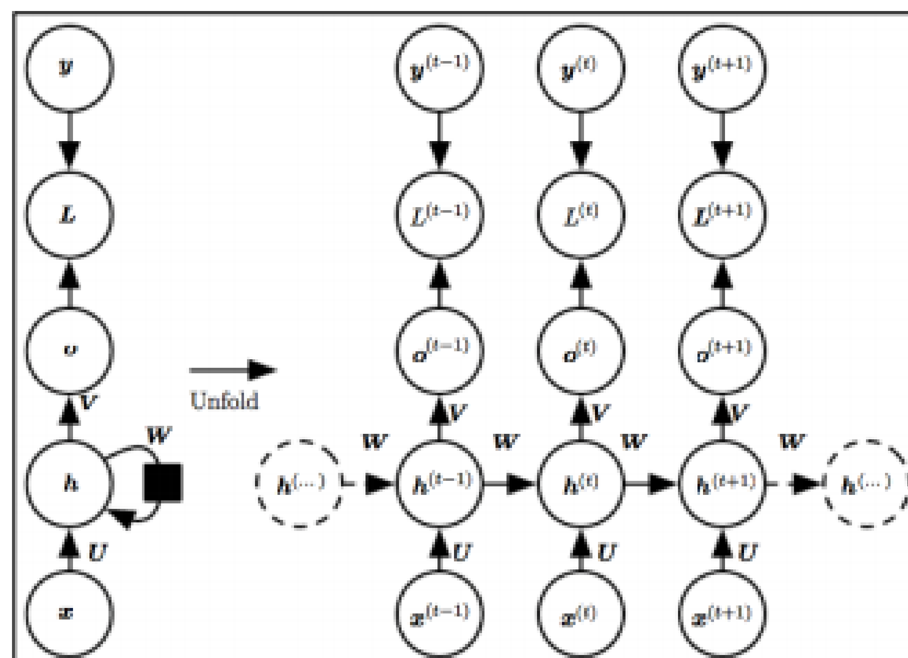
  is Universal, i.e.,

  - Any function computable by a TM is computable by such an RNN of finite size



Forward prop for RNN with hidden unit recurrence

- This design does not specify
  1. activation functions for hidden units,
  2. form of output and
  3. loss function



- Assume for this graph:
  1. Hyperbolic tangent activation function
  2. Output is discrete
     - E.g., RNN predicts words/characters
  - Natural way to represent discrete vars:
    - Output $\boldsymbol{o}$ gives unnormalized log probabilities $\hat{\boldsymbol{y}}$ of each possible value
    - Can apply $\mathrm{softmax}$ as a postprocessing step to obtain vector of normalized probabilities over the output
- Forward prop begins with specification of initial state $\boldsymbol{h}^{(0)}$

Forward Prop Equations for RNN with hidden recurrence

- Begins with initial specification of $\boldsymbol{h}^{(0)}$
- Then for each time step from $t=1$ to $t=\tau$ we apply the following update equations

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + V\boldsymbol{h}^{(t)}$$
$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$$
$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$$

- Where the parameters are:
  - bias vectors $\boldsymbol{b}$ and $\boldsymbol{c}$
  - weight matrices U (input-to-hidden), V (hidden-to-output) and W (hidden-to-hidden) connections

Loss function for a given sequence

- This is an example of an RNN that maps an input sequence to an output sequence of the same length
- Total loss for a given sequence for a given sequence of $\boldsymbol{x}$ values with a sequence of $\boldsymbol{y}$ values is the sum of the losses over the time steps
- If $L^{(t)}$ is the negative log-likelihood of $\boldsymbol{y}^{(t)}$ given $\boldsymbol{x}^{(1)},..\boldsymbol{x}^{(t)}$ then

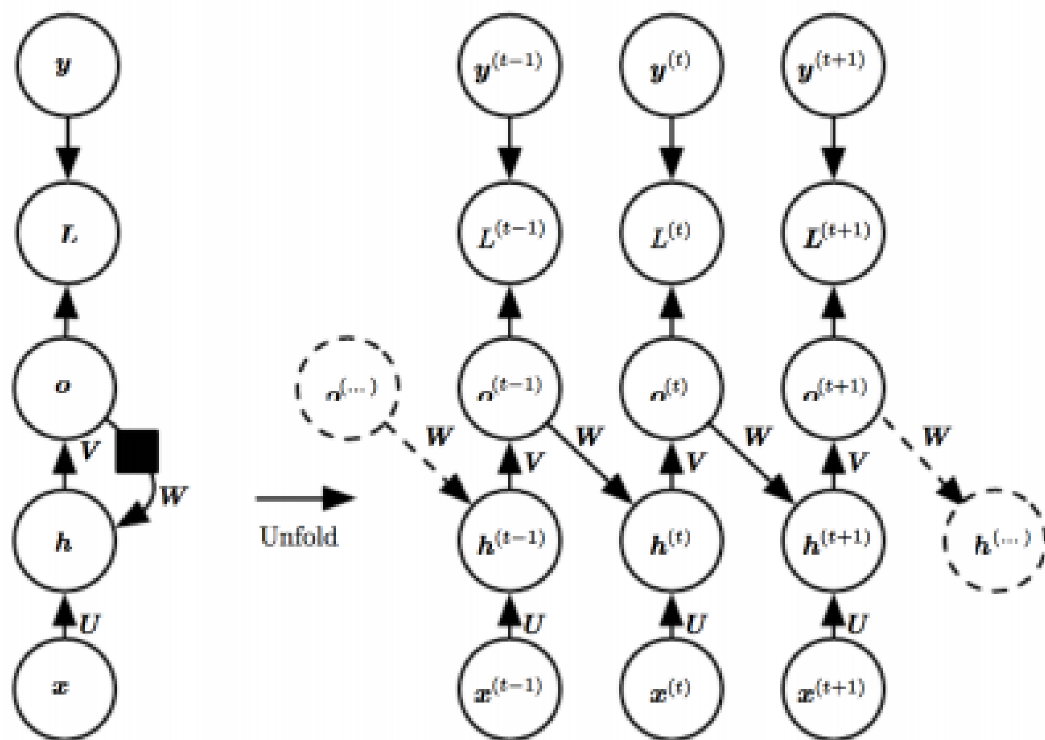
$$L\left(\left\{\boldsymbol{x}^{(1)},..\boldsymbol{x}^{(t)}\right\},\left\{\boldsymbol{y}^{(1)},..\boldsymbol{y}^{(t)}\right\}\right)=\sum_t L^{(t)}$$

$$=-\sum_t \log p_{\text{model}}\left(\boldsymbol{y}^{(t)} \mid \left\{\boldsymbol{x}^{(1)},..\boldsymbol{x}^{(t)}\right\}\right)$$

  - where p_{model} is given by reading the entry for $\boldsymbol{y}^{(t)}$ from the model's output vector $\hat{\boldsymbol{y}}^{(t)}$

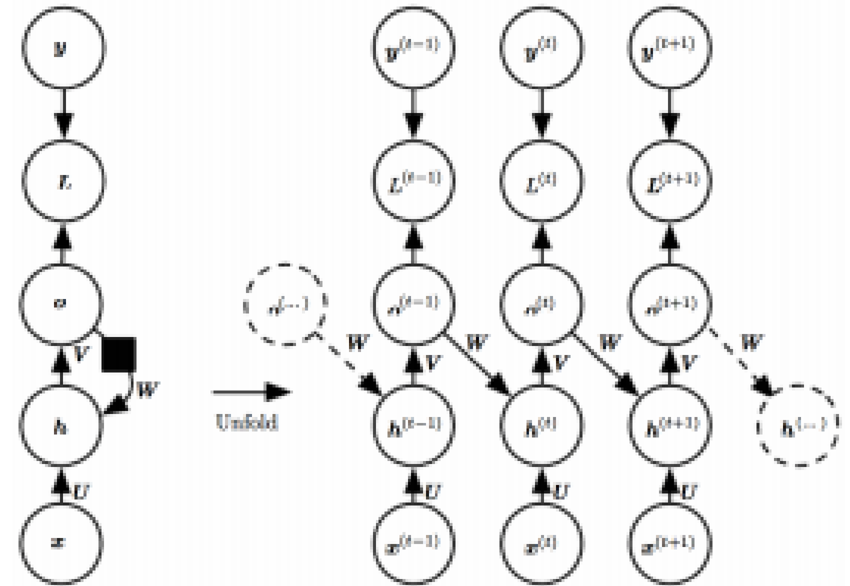
RNN with recurrence from output to hidden

- A network with recurrent connections from output (at one time step) to hidden units (at next time step)
- It is strictly less powerful than hidden-to-hidden recurrence
  1. Cannot simulate a universal TM
  2. Because output units are trained to match training set targets, they are unlikely to capture past history of input unless user provides it as part of training set targets



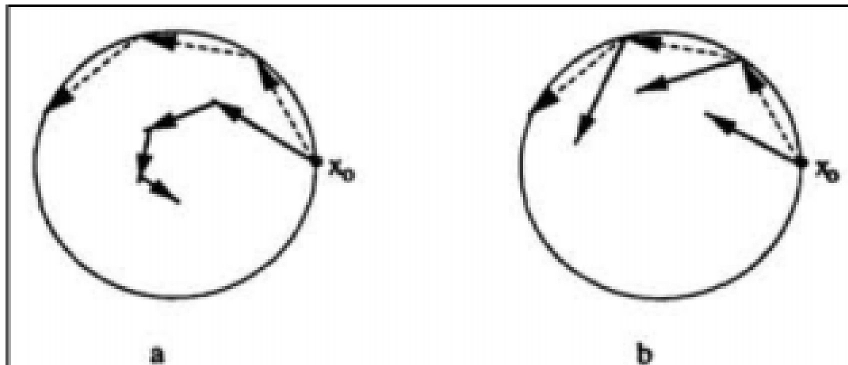
Advantage of eliminating hidden recurrence

- For any loss function based on comparing prediction at time t to training target at time t all time steps are decoupled
- Training can be parallelized with gradient for each time step t computed in isolation
- There is no need to compute the output for the previous time step first
  - Because the training set provides the ideal value of that output
- Models that have recurrent connections from their outputs leading back to the model may be trained with *teacher forcing*



Teacher Forcing

- Models that have recurrent connections from their outputs leading back to the model may be trained with *teacher forcing*
- Teacher forcing during training means
  - Instead of summing activations from incoming units (possibly erroneous)
  - Each unit sums correct teacher activations as input for the next iteration
- Visualizing effect of teaching forcing
  - imagine that the network is learning to follow a trajectory; it goes astray (because the weights are wrong) but teacher forcing puts the net back on its trajectory by setting the state of all the units to that of teacher's.



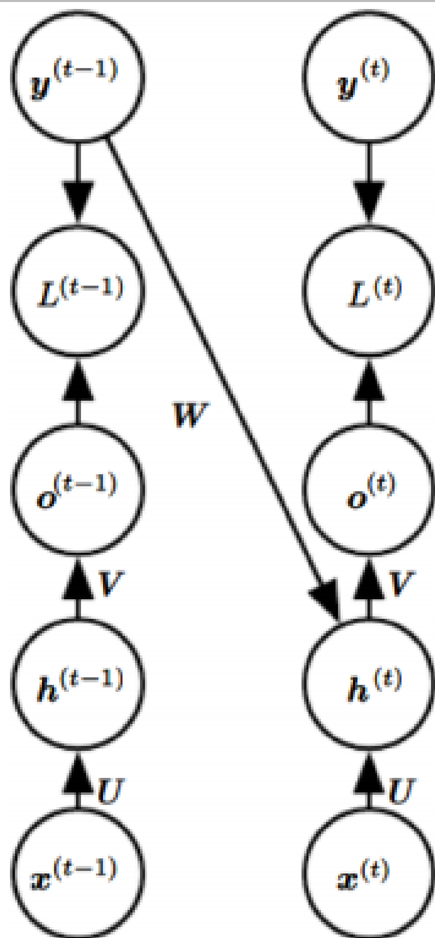
(a) Without teacher forcing, trajectory runs astray (solid lines) while the correct trajectory are the dotted lines

(b) With teacher forcing trajectory corrected at each step

Illustration of Teacher Forcing

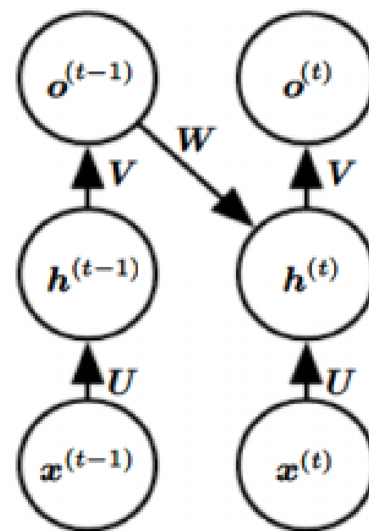
- Teacher Forcing is a training technique applicable to RNNs that have connections from output to hidden states at next time step

Train time:
We feed the correct output $\boldsymbol{y}^{(t)}$ (from teacher) drawn from the training set as input to $\boldsymbol{h}^{(t+1)}$



Train time

Test time:
True output is not known.
We approximate the correct output $\boldsymbol{y}^{(t)}$ with the model's output $\boldsymbol{o}^{(t)}$ and feed the output back to the model



Test time

Computing Gradient of Loss Function

- Computing gradient of this loss function wrt parameters is expensive
  - It involves performing a forward propagation pass moving left to right through the unrolled graph followed by a backward propagation moving right to left through the graph
- Run time is $O(\tau)$ and cannot be reduce by parallelization
  - Because forward propagation graph is inherently sequential
    - Each time step computable only after previous step
  - States computed during forward pass must be stored until reused in the backward pass
    - So memory cost is also $O(\tau)$
- Backpropagation applied to the unrolled graph with $O(\tau)$ cost is called *Backward Propagation through time* (BPTT)

Backward Propagation through Time (BPTT)

- RNN with hidden unit recurrence is very powerful but also expensive to train
- Is there an alternative?

Training with both Teacher Forcing and BPTT

- Teacher forcing can be used together with Backward Propagation through time (BPTT)
- When there are both hidden-to-hidden recurrences as well as output-to-hidden recurrences

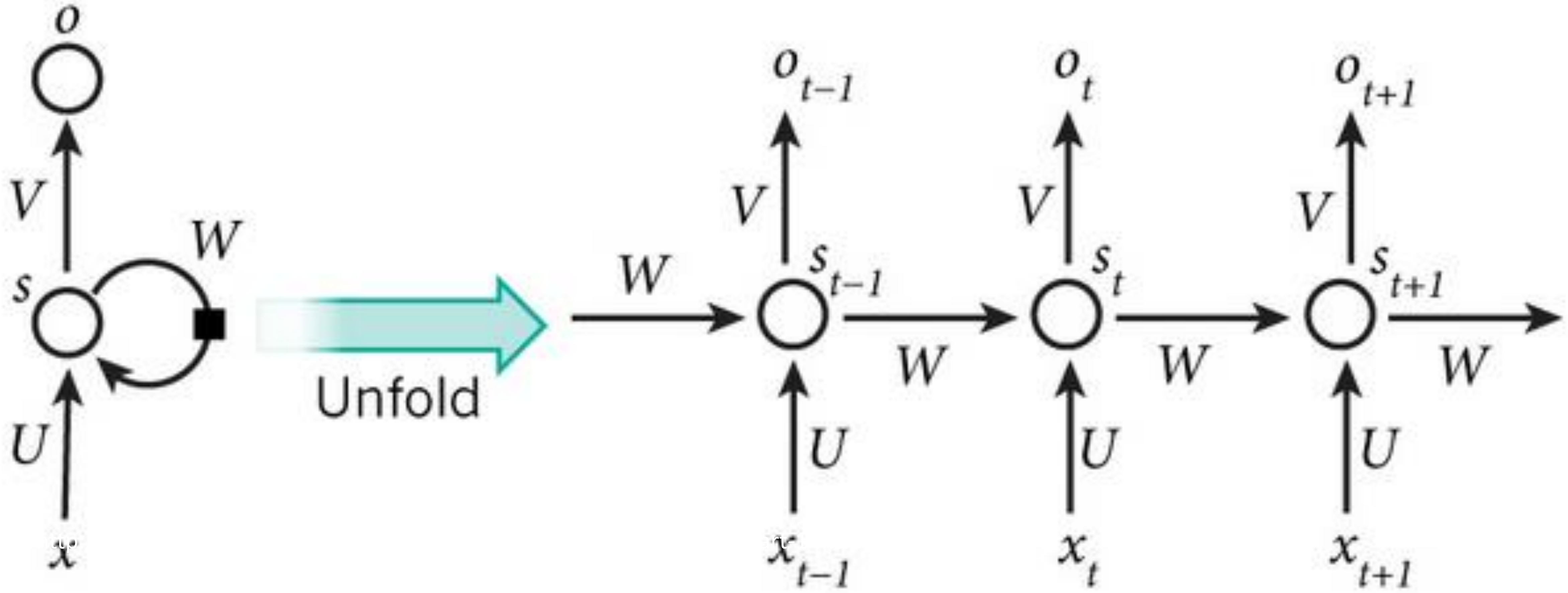
Computing gradient in RNN using BPTT

- Computing the gradient through an RNN is straightforward
- One simply applies the generalized back-propagation algorithm to the unrolled computational graph.
- No specialized algorithms are necessary.
- Gradients obtained by back-propagation may then be used with any general-purpose gradient-based techniques to train an RNN.

Unfolding over Time

Recurrence similar to

- Hidden Markov Model (HMM)
- Kalman Filter (KF, EKF, UKF)



White Board Time!

- “the cat sat on the mat”

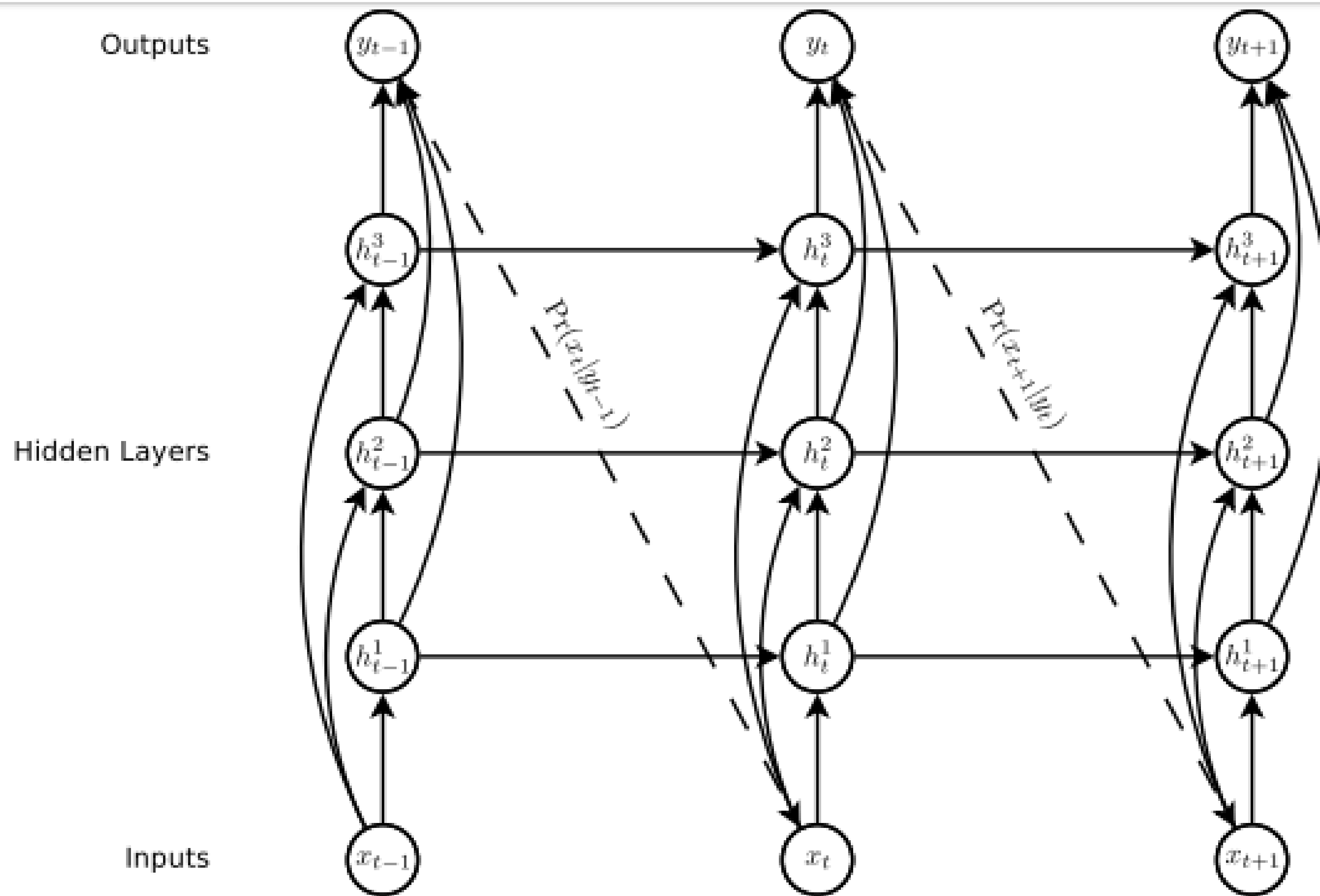


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

Questions?

Deep robots!

Deep questions?!