CS60010: Deep Learning

Recurrent Neural Network

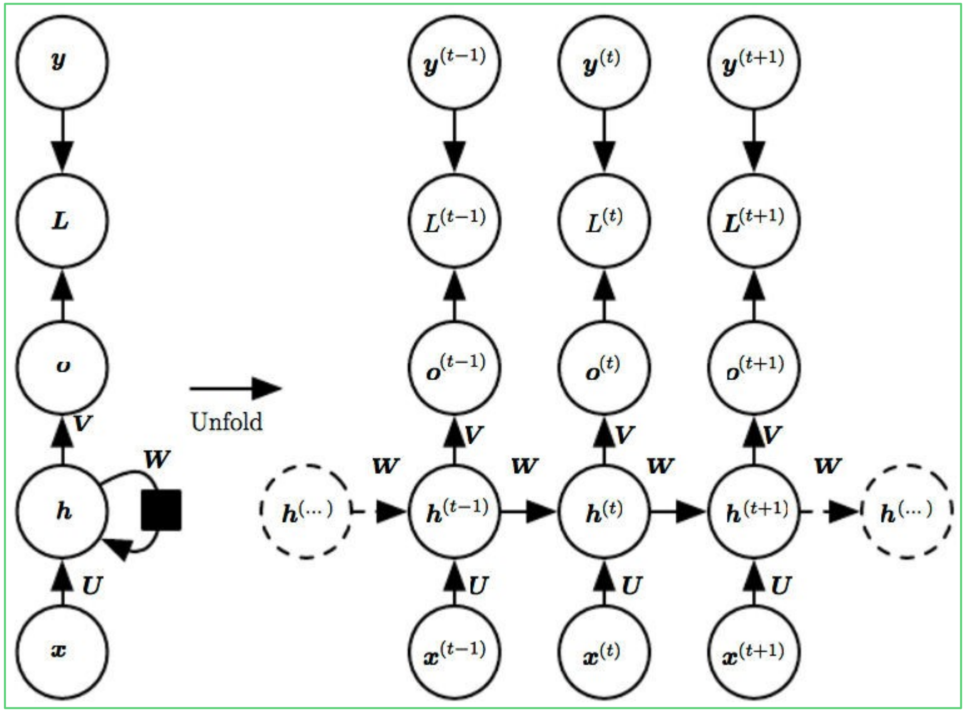
Sudeshna Sarkar

Spring 2018

6 Feb 2018

RNN1: with recurrence between hidden units

Maps input sequence $\boldsymbol{x}$ to output $\boldsymbol{o}$



With softmax outputs Loss L internally computes $\hat{y}$= softmax(o) and compares to target y

- Update equation applied for each time step from $t = 1$ to $t = \tau$

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Parameters:

- 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

- The total loss 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

$$L\left(\left\{x^{(1)}, x^{(2)}, \dots, x^{(t)}\right\}, \left\{y^{(1)}, y^{(2)}, \dots, y^{(t)}\right\}\right) = \sum_{t} L^{(t)}$$

$$= -\sum_{t} \log p_{model}\left(y^{(t)} \middle| \left\{x^{(1)}, x^{(2)}, \dots, x^{(t)}\right\}\right)$$

Backpropagation through time

- We can think of the recurrent net as a layered, feed-forward net with shared weights and then train the feed-forward net with weight constraints.

- We can also think of this training algorithm in the time domain:
  - The forward pass builds up a stack of the activities of all the units at each time step.
  - The backward pass peels activities off the stack to compute the error derivatives at each time step.
  - After the backward pass we add together the derivatives at all the different times for each weight.

Gradients on V, **c, W** and **U**

$$\frac{\partial L}{\partial L_t} = 1, \frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial L_t}\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial o_t}$$

$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial o_t}\frac{\partial o_t}{\partial V}$$

$$\frac{\partial L}{\partial c} = \sum_t \frac{\partial L_t}{\partial o_t}\frac{\partial o_t}{\partial c}$$

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial W}$$

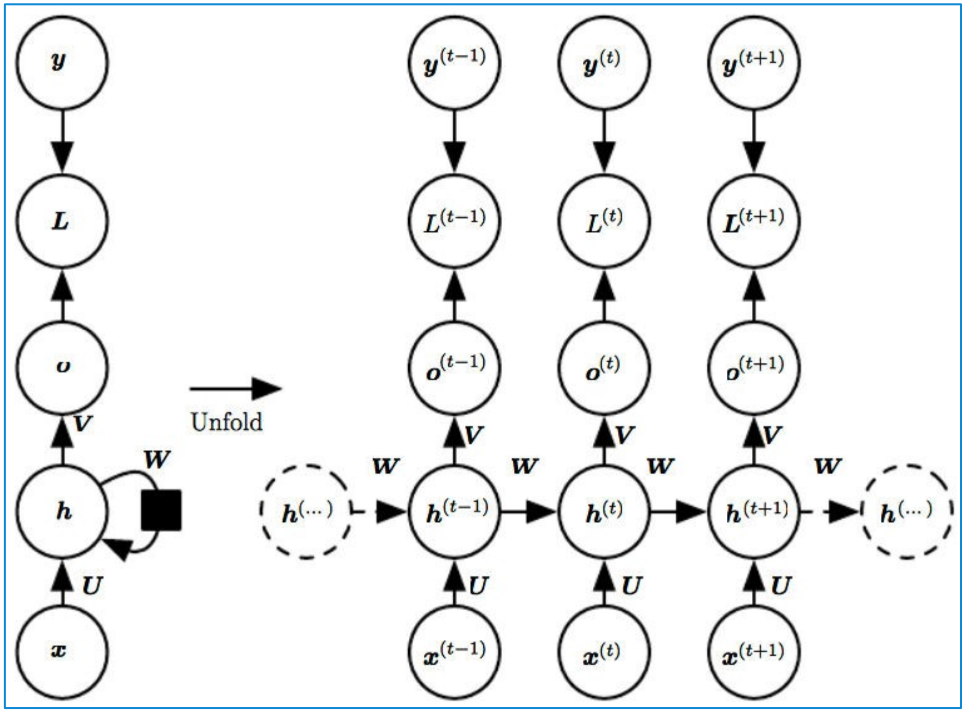
$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial o_t}\frac{\partial o_t}{\partial h_t}$$

$$\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}$$

$$L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right)$$

$$= \sum_t L^{(t)}$$

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

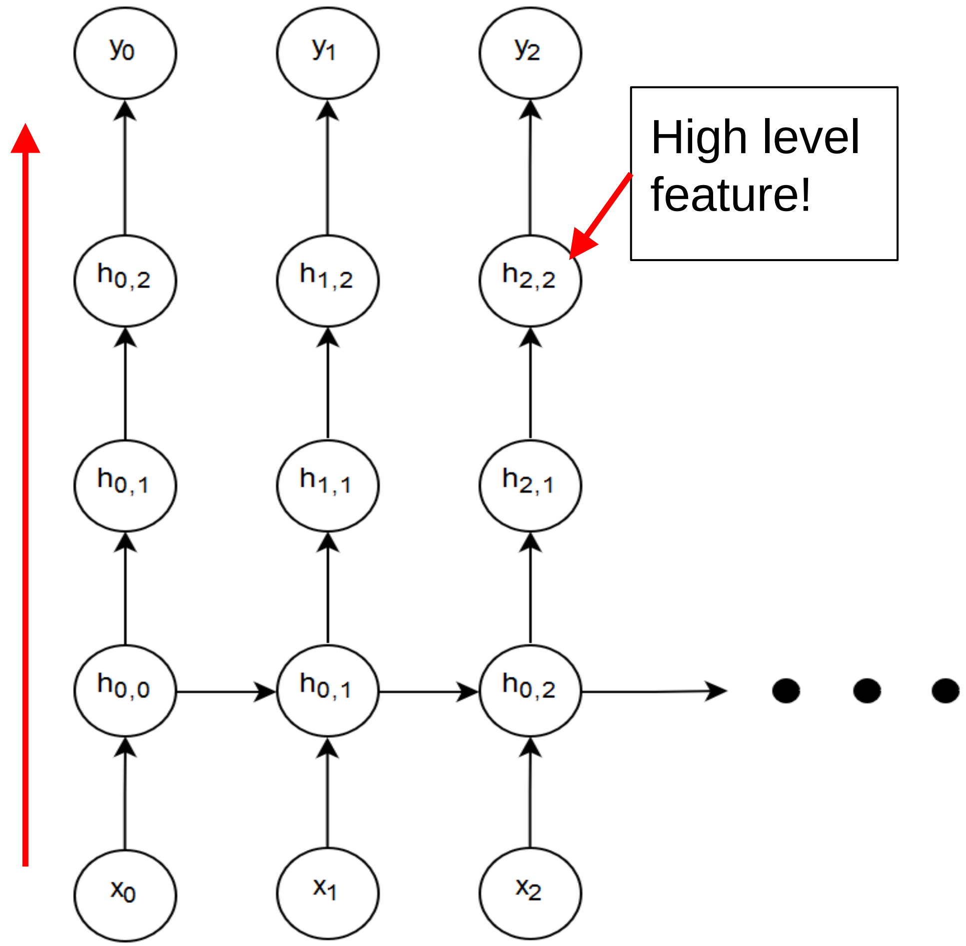


Feedforward Depth (d_f)

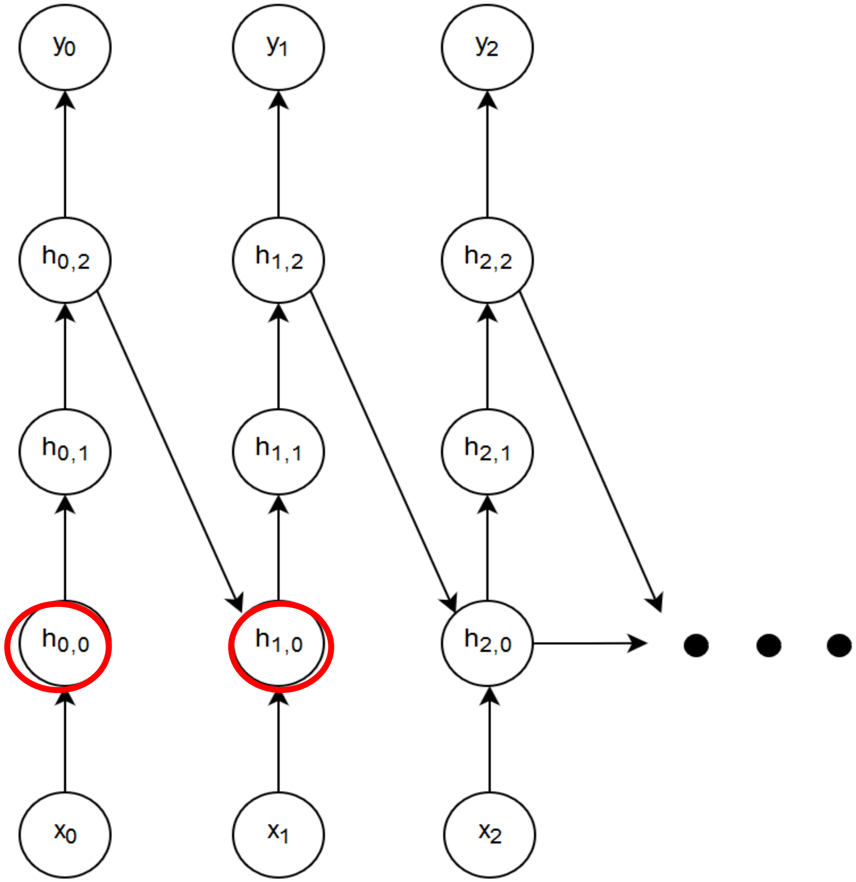
Notation: $h_{0,1} \Rightarrow$ time step 0, neuron #1

<u>Feedforward depth</u>: longest path between an input and output at the **same timestep**

Feedforward depth = 4



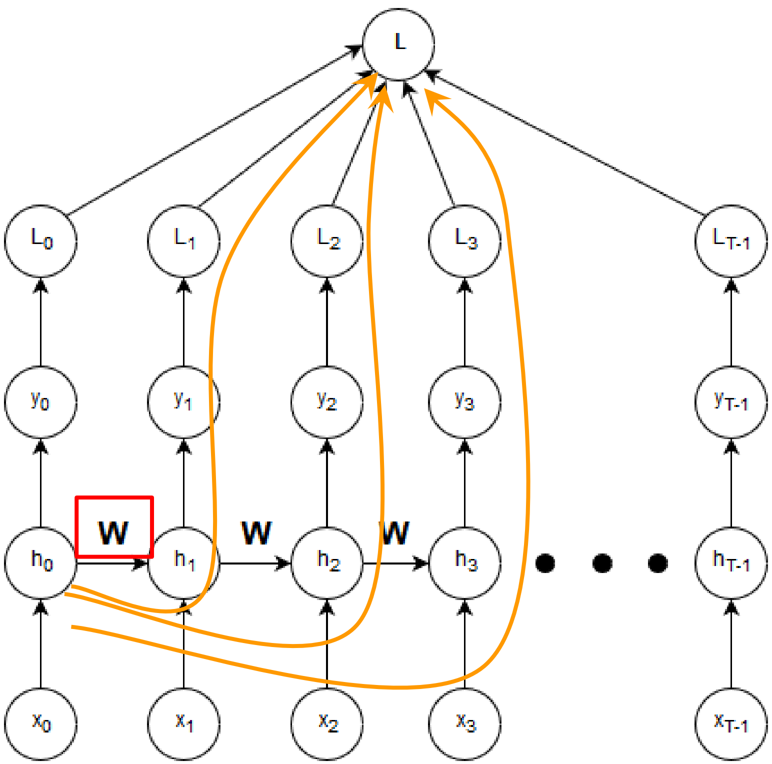
Recurrent Depth (d_r)



Recurrent depth = 3

Recurrent depth: Longest path between **same hidden state** in **successive timesteps**

Backpropagation Through Time (BPTT)

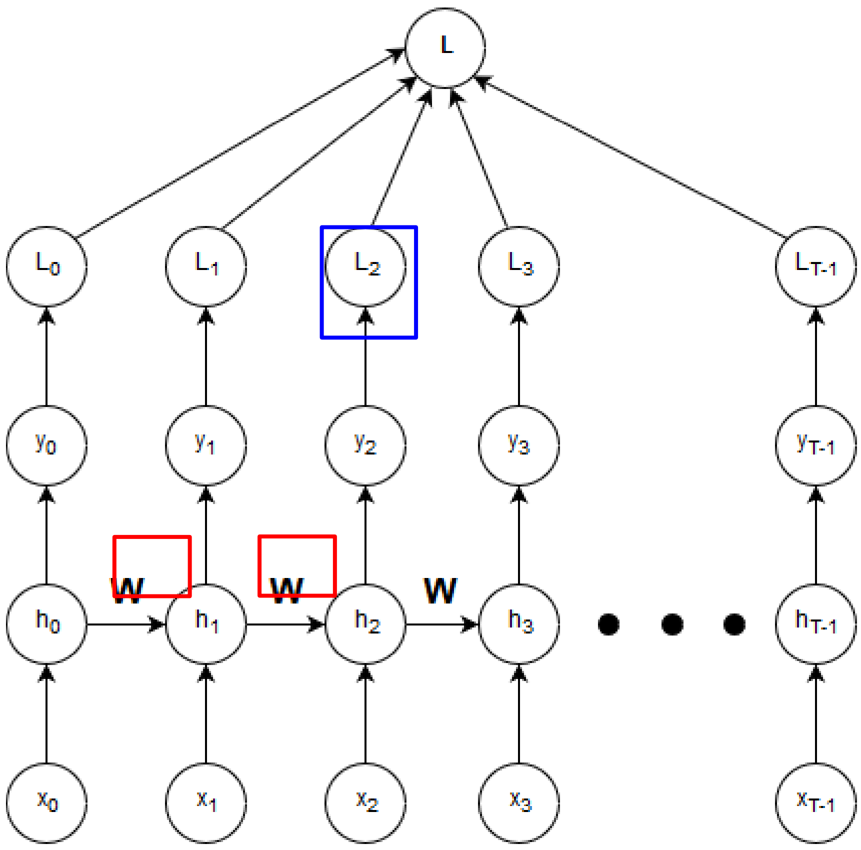


- Update the weight matrix:

$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

- Issue: **W** occurs each timestep
- **Every** path from **W** to L is one dependency
- Find all paths from **W** to L

Systematically Finding All Paths



- How many paths exist from W to L through L1?
  - 1
- How many paths from W to L through L2?
  - 2 (originating at h0 and h1)

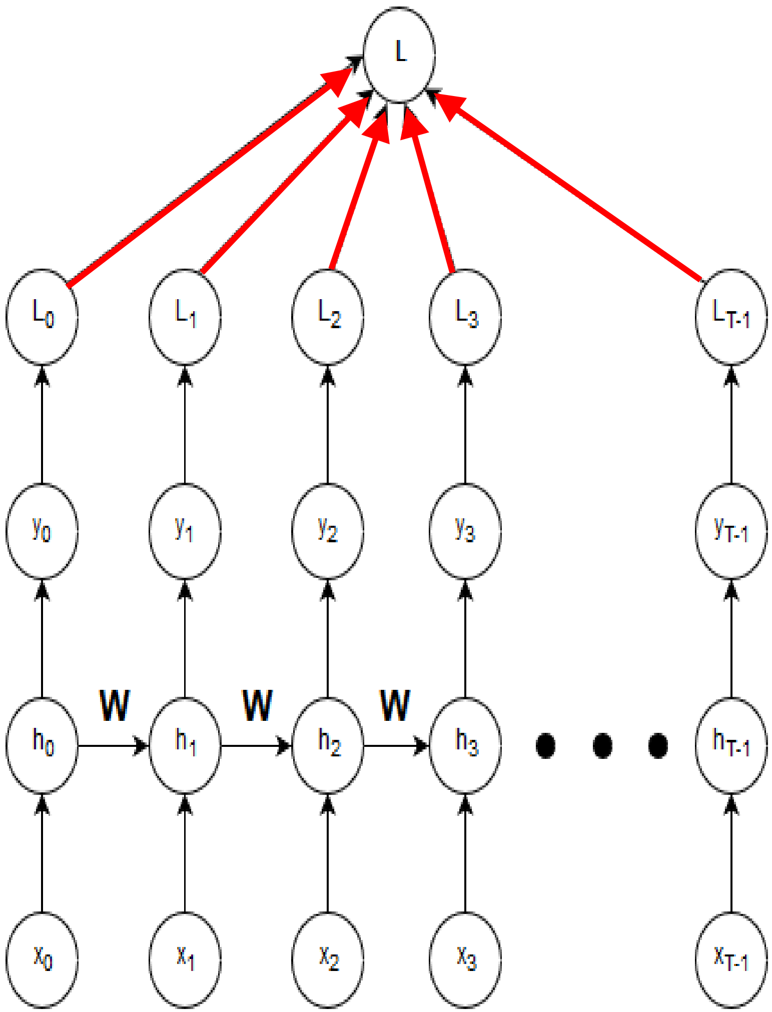
$$\frac{\partial L}{\partial \mathbf{W}}$$

The gradient has two summations:

1: Over L_j

2: Over h_k

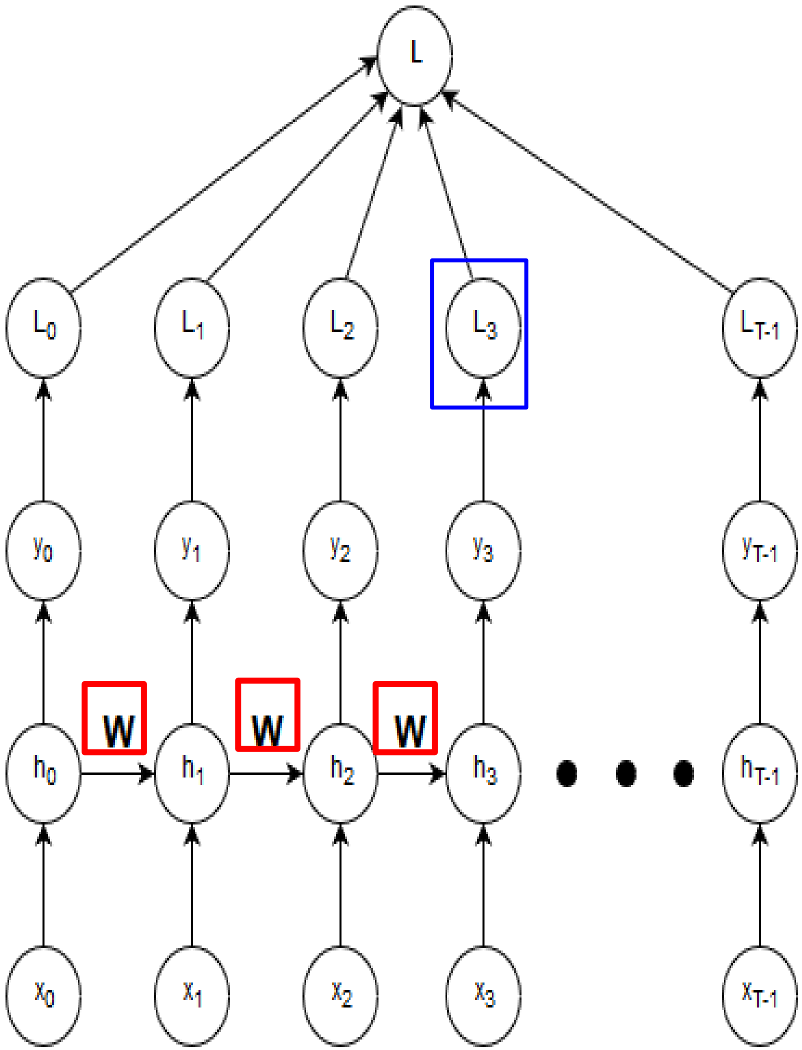
Backpropagation as two summations



First summation over L

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}}$$

Backpropagation as two summations

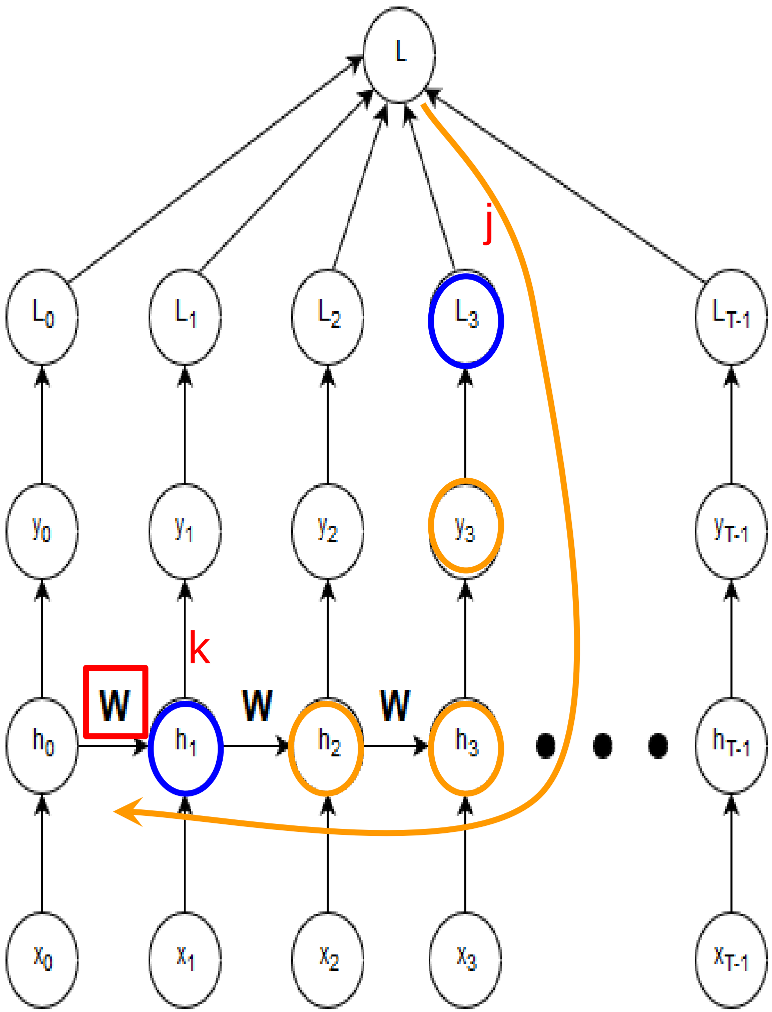


Second summation over h: Each $\mathbf{L_j}$ depends on the weight matrices *before it*

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial h_k}} \frac{\partial h_k}{\partial \mathbf{W}}$$

L_j depends on all h_k before it.

Backpropagation as two summations

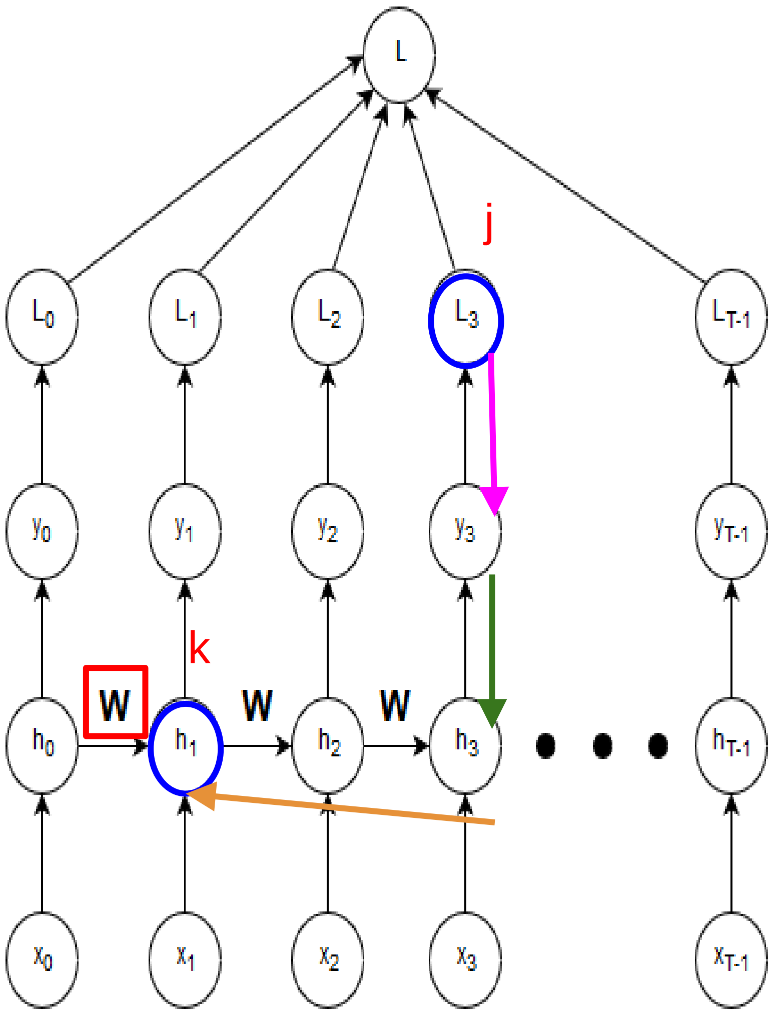


$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial h_k}} \frac{\partial h_k}{\partial \mathbf{W}}$$

- No explicit dep of L_j on h_k
- Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k}} \frac{\partial h_k}{\partial \mathbf{W}}$$

Backpropagation as two summations

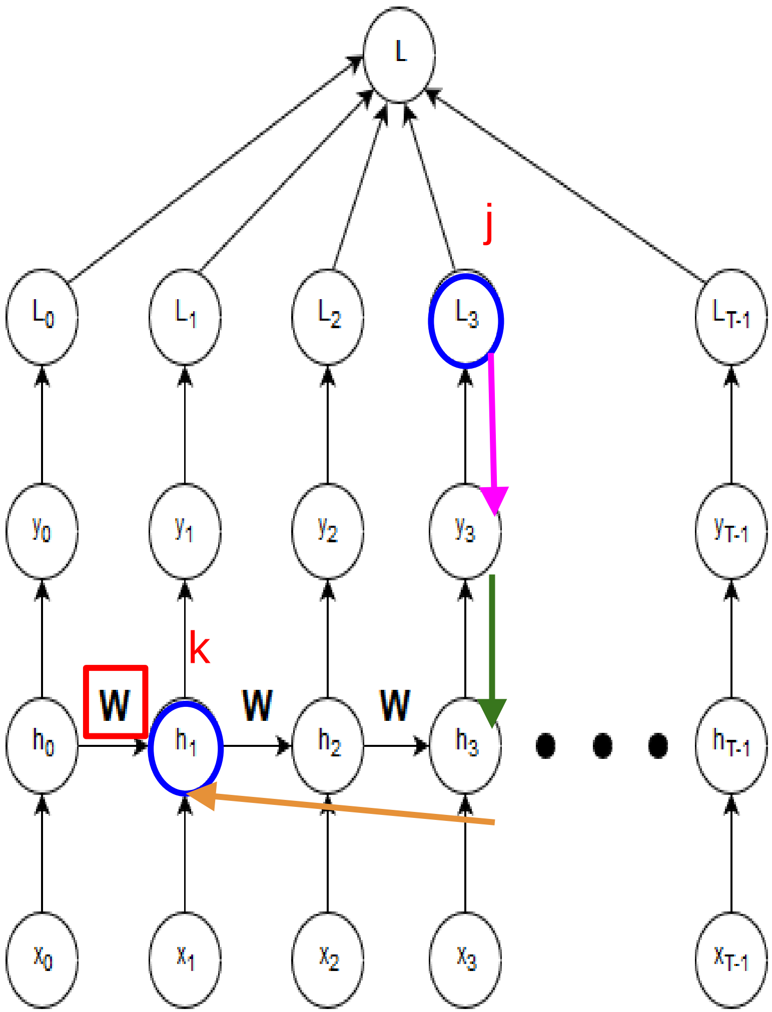


$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial h_k}} \frac{\partial h_k}{\partial \mathbf{W}}$$

- No explicit of L_j on h_k
- Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial y_j}} \boxed{\frac{\partial y_j}{\partial h_j}} \boxed{\frac{\partial h_j}{\partial h_k}} \frac{\partial h_k}{\partial \mathbf{W}}$$

The Jacobian



$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial y_j}} \boxed{\frac{\partial y_j}{\partial h_j}} \boxed{\frac{\partial h_j}{\partial h_k}} \frac{\partial h_k}{\partial \mathbf{W}}$$

Indirect dependency. One final use of the chain rule gives:

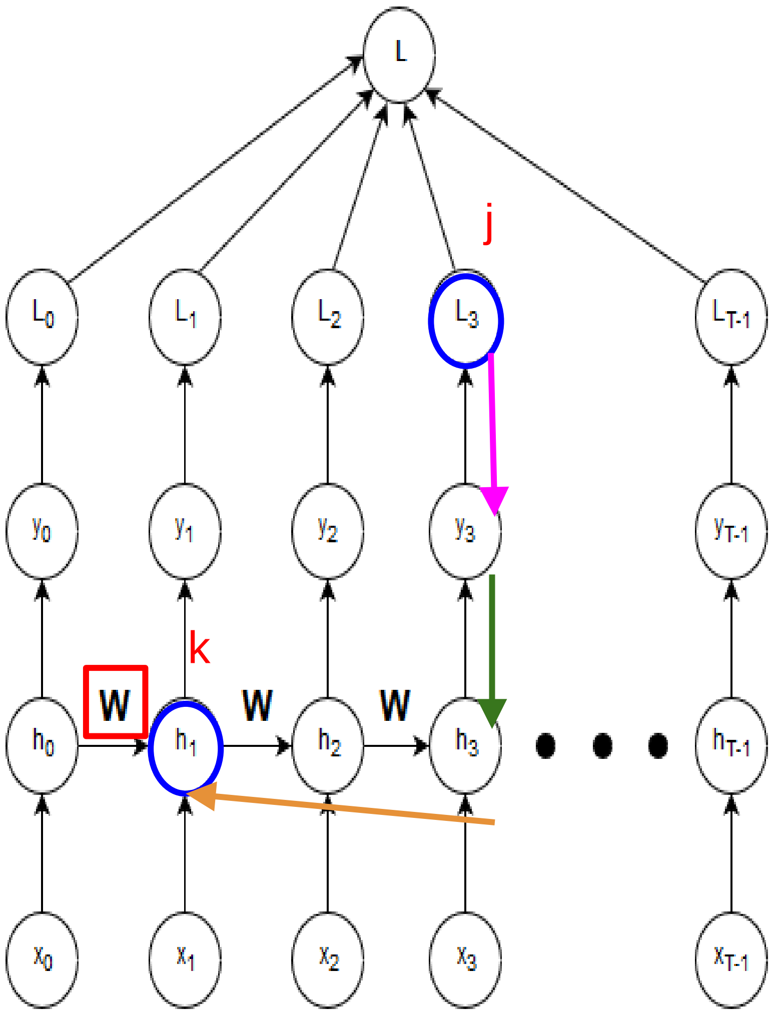
$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$

“The Jacobian”

The Final Backpropagation Equation

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

Backpropagation as two summations



$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \boxed{\frac{\partial L_j}{\partial y_j}} \boxed{\frac{\partial y_j}{\partial h_j}} \boxed{\left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right)} \frac{\partial h_k}{\partial \mathbf{W_h}}$$

- Often, to reduce memory requirement, we truncate the network
- Inner summation runs from $j - p$ to j for some p ==> truncated BPTT

Expanding the Jacobian

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \boxed{\left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right)} \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \mathrm{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

The Issue with the Jacobian

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \boxed{\mathbf{W}_h^T} \boxed{\mathrm{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)}$$

Weight Matrix

Derivative of activation function

Repeated matrix multiplications leads to **vanishing and exploding gradients**.

Eigenvalues and Stability

Consider identity activation function

If Recurrent Matrix $\mathbf{W_h}$ is a diagonalizable:

$$W_h = Q^{-1} * \Lambda * Q$$

Q matrix composed of eigenvectors of W_h

Λ is a diagonal matrix with eigenvalues placed on the diagonals

Computing powers of $\mathbf{W_h}$ is simple:

$$W_h^n = Q^{-1} * \Lambda^n * Q$$

Bengio et al, "On the difficulty of training recurrent neural networks." (2012)

Eigenvalues and stability

Vanishing gradients

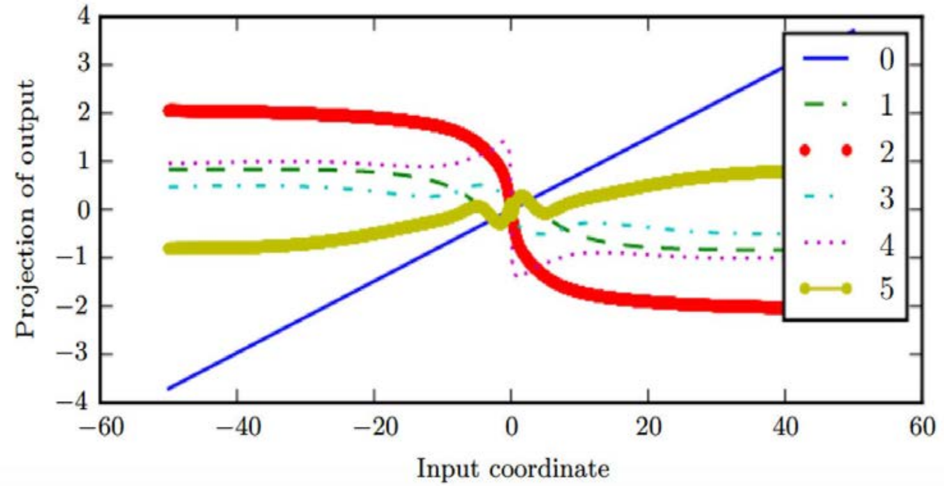
$$\Lambda = \begin{bmatrix} -0.6180 & 0 \\ 0 & 1.6180 \end{bmatrix} \longrightarrow \Lambda^{10} = \begin{bmatrix} \boxed{0.0081} & 0 \\ 0 & \boxed{122.9919} \end{bmatrix}$$

Exploding gradients

$$W_h^n = Q^{-1} * \Lambda^n * Q$$

Function Composition in RNNs

- RNNs involve composition of the same function multiple times, one per step
- These compositions can result in extremely nonlinear behaviour



Composing many nonlinear functions: eg tanh

h has 100 dimensions mapped to a single dimension

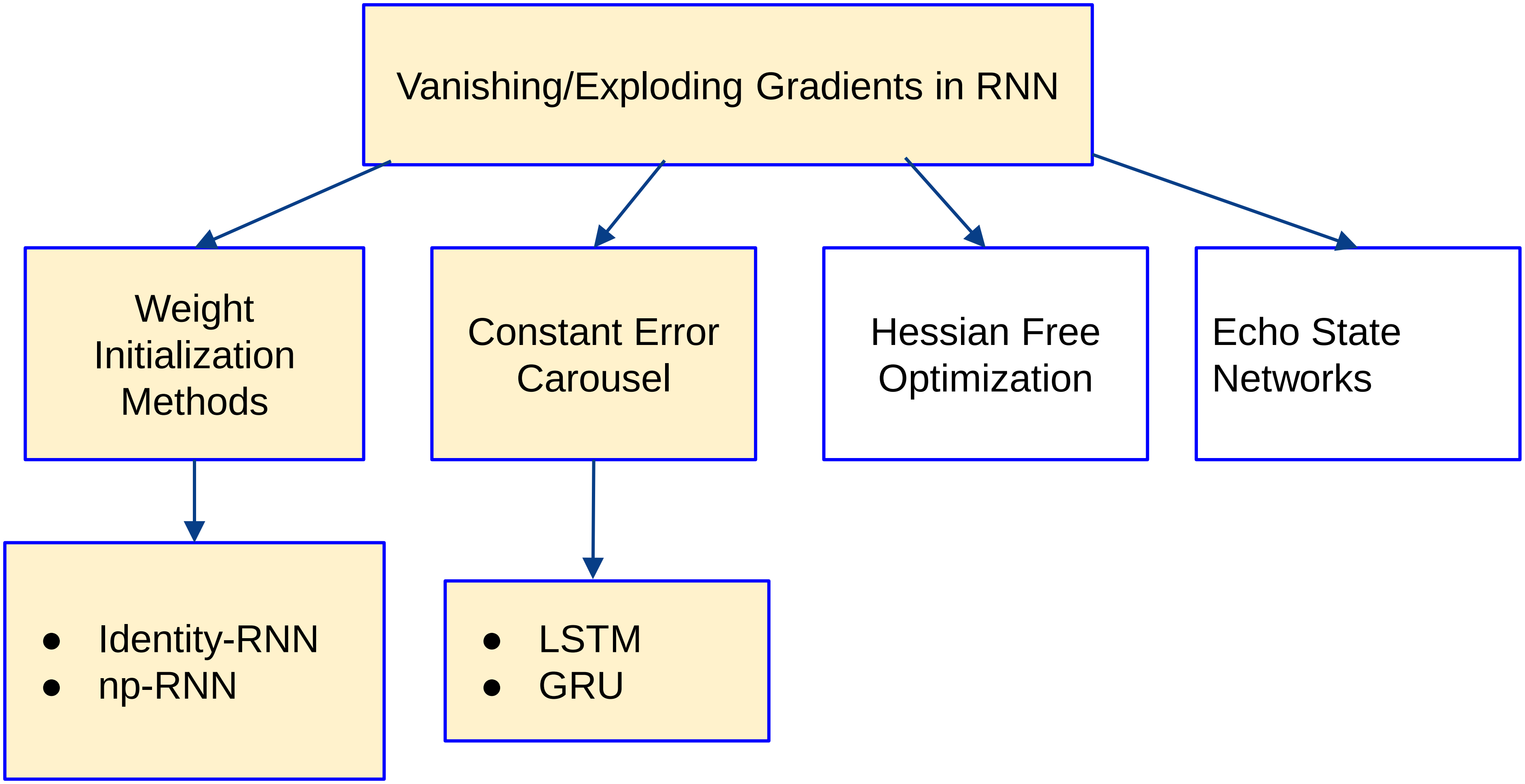
Most of the space it has a small derivative and highly nonlinear elsewhere

- Problem particular to RNNs

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the gradients shrink exponentially.
  - If the weights are big the gradients grow exponentially.
- In an RNN trained on long sequences the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with long-range dependencies.

Addressing Vanishing / exloding gradients



Complexity of BPTT

- Computing gradient of the loss function wrt parameters is expensive
  - It involves performing a forward propagation pass followed by a backward propagation through the graph
- Run time is $O(\tau)$ and cannot be reduced by parallelization
- States computed during forward pass must be stored until reused in the backward pass
  - So memory cost is also $O(\tau)$
- RNN with hidden unit recurrence is very powerful but also expensive to train

RNN Variation 2: output2hidden, sequence output

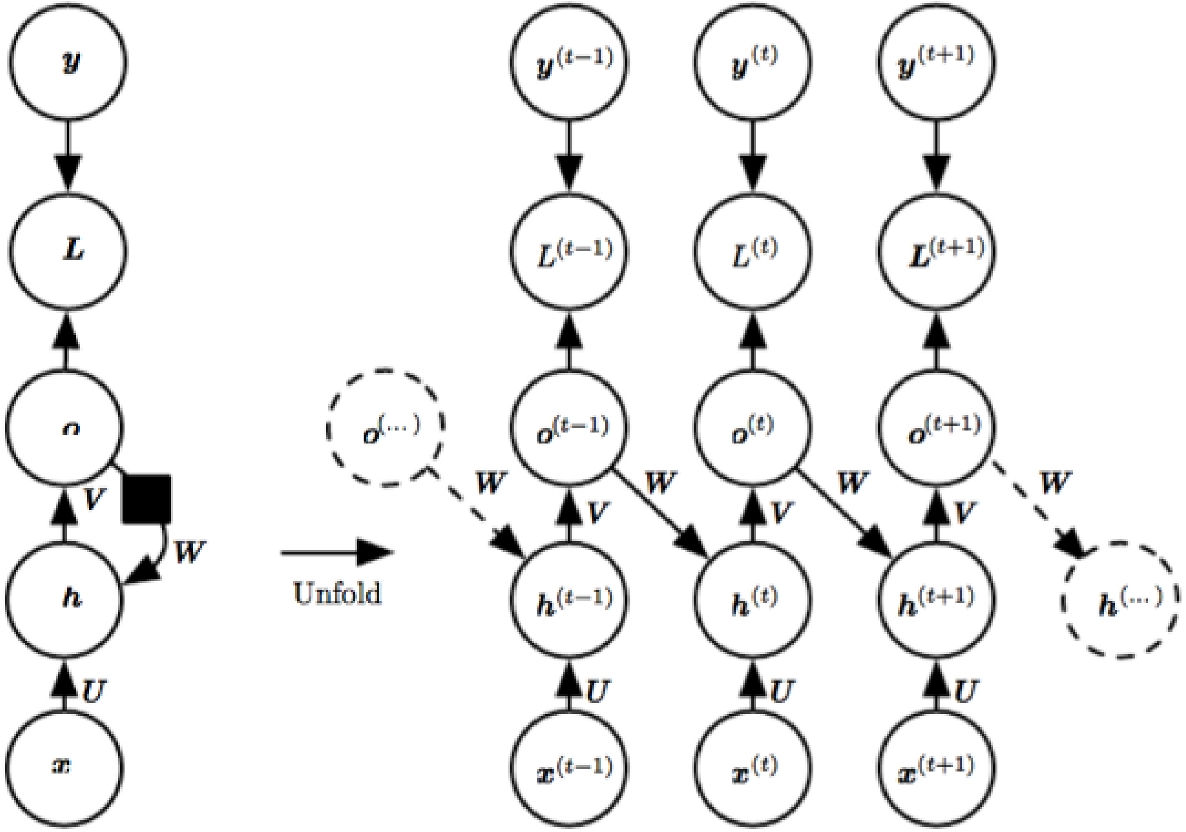


Figure 10.4: An RNN whose only recurrence is the feedback connection from the output to the hidden layer. At each time step t, the input is $\boldsymbol{x}_t$, the hidden layer activations are $\boldsymbol{h}^{(t)}$, the outputs are $\boldsymbol{o}^{(t)}$, the targets are $\boldsymbol{y}^{(t)}$ and the loss is $L^{(t)}$. *(Left)* Circuit diagram.

Teacher Forcing and Networks with Output Recurrence

RNN Variation 2: output2hidden, sequence output

Less powerful than with hidden-to- hidden recurrent connections

- It cannot simulate a universal TM
- It requires that the output capture all information of past to predict future

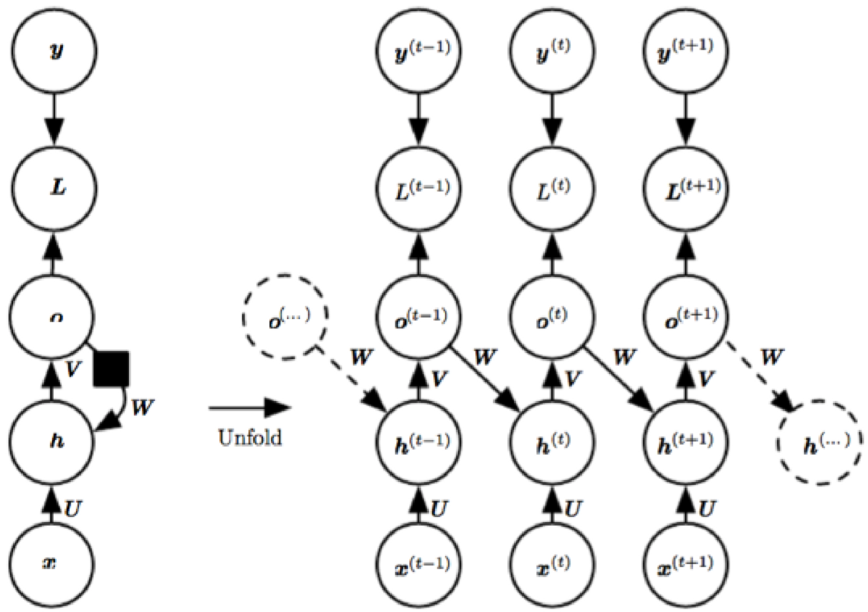
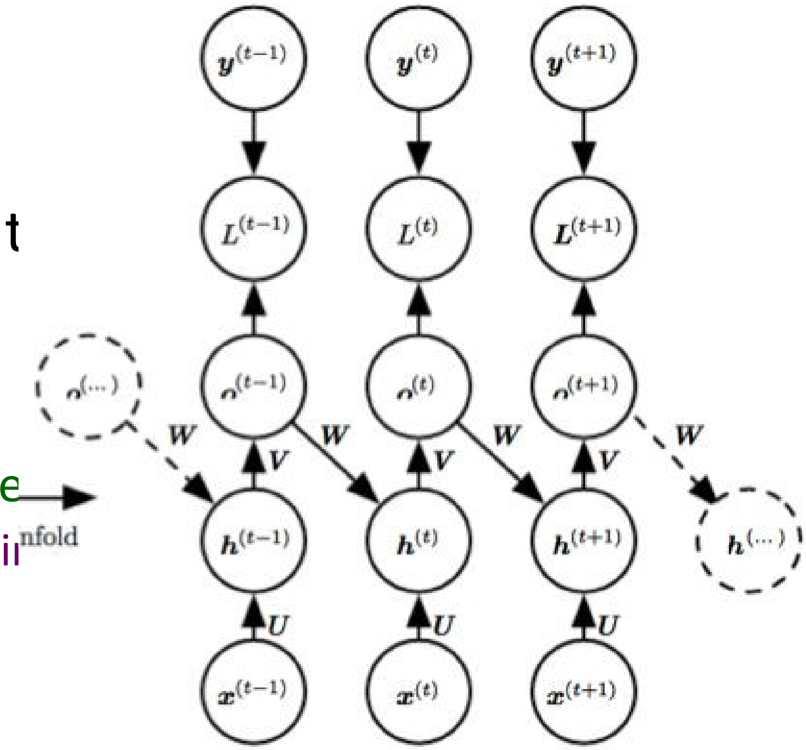


Figure 10.4: An RNN whose only recurrence is the feedback connection from the output to the hidden layer. At each time step t, the input is $\boldsymbol{x}_t$, the hidden layer activations are $\boldsymbol{h}^{(t)}$, the outputs are $\boldsymbol{o}^{(t)}$, the targets are $\boldsymbol{y}^{(t)}$ and the loss is $L^{(t)}$. *(Left)* Circuit diagram.

- The model is trained to maximize the conditional probability of current output y(t), given both the x sequence so far and the previous output y(t-1)

Training with Teacher forcing

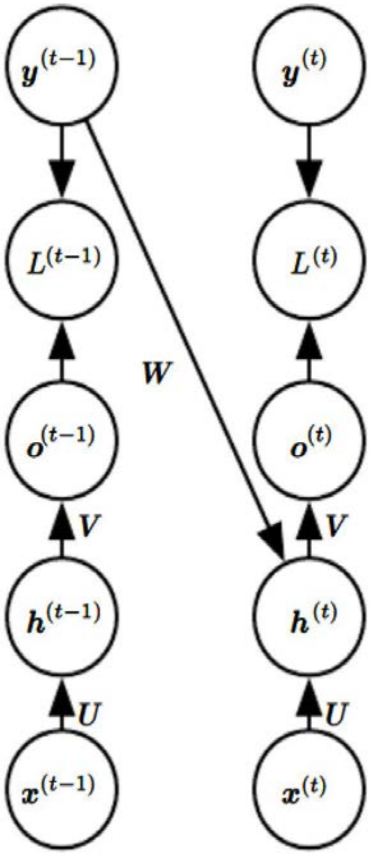
- Teacher forcing is a procedure that emerges from the maximum likelihood criterion, in which during training the model receives the ground truth output $y^{(t)}$ as input at time $t+1$.
- Advantage
  - In comparing loss function to output all time steps are decoupled each step can be trained in isolation
  - Training can be parallelized
    - Gradient for each step *t* computed in isolation
    - No need to compute output for the previous step first, because training set provides ideal value of output



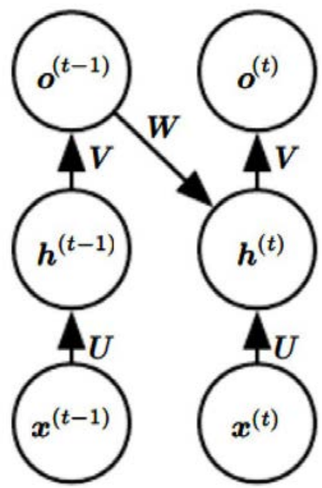
Teacher forcing

- Train time: We feed the correct output y(t) (from teacher) drawn from the training set as input to h(t+1)

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



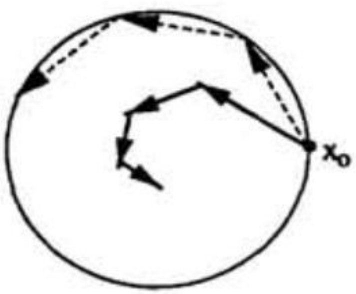
Train time



Test time

Visualizing Teacher 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

b

(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

Training with both Teacher Forcing and BPTT

- Some models may be trained with both Teacher forcing and Backward Propagation through time (BPTT)
  - When there are both hidden-to-hidden recurrences as well as output-to- hidden recurrences

Disadvantage of Teacher Forcing

- Limited expressive power