Recurrent Neural Networks

Zhiyao Duan
Associate Professor of ECE and CS
University of Rochester

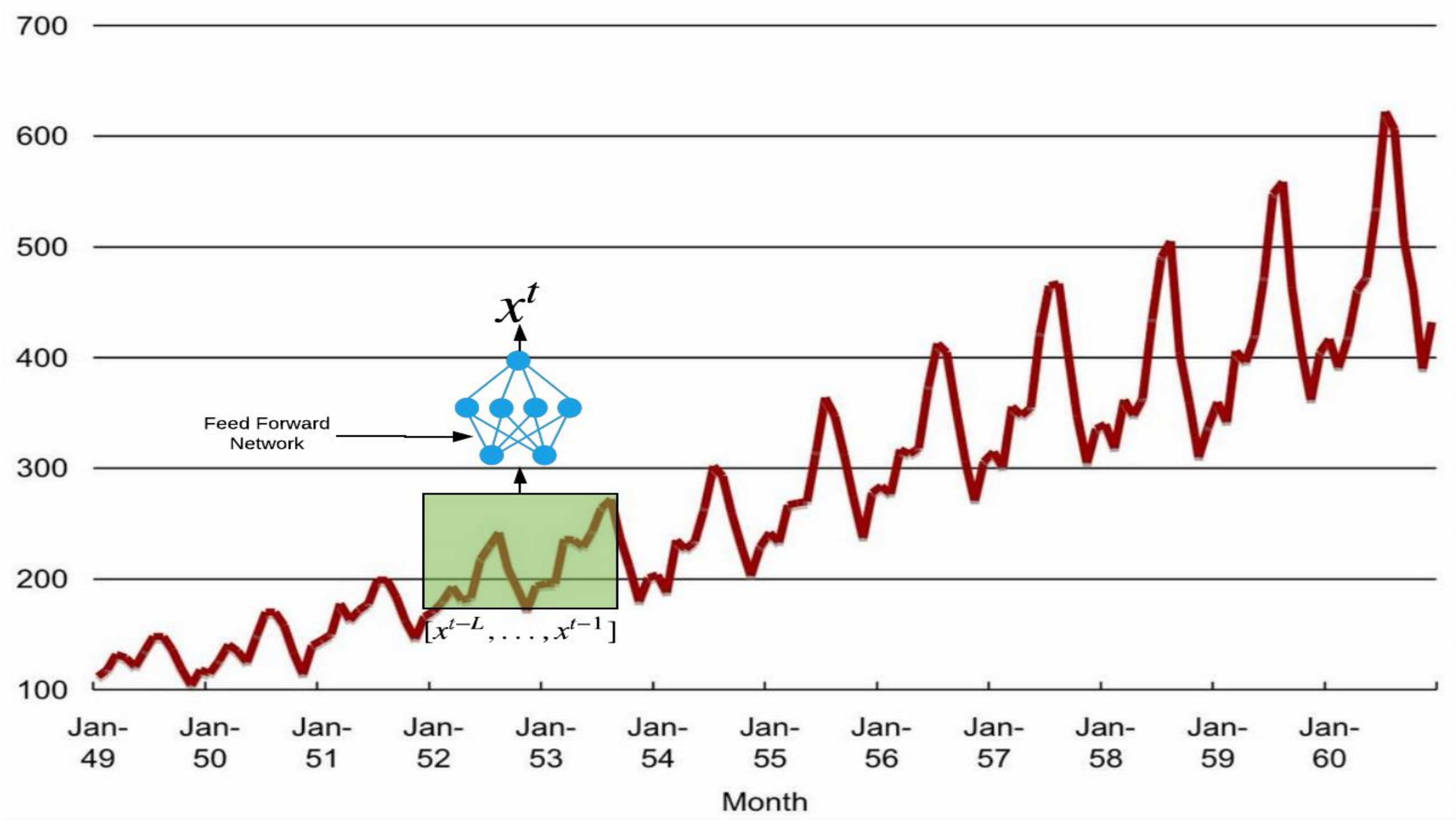
Some figures are copied from the following book

- **GBC** - Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press.
- **Mitchell** - Tom M. Mitchell, Machine Learning, McGraw-Hill Education, 1997.

Let's start from Multi-Layer Perceptron

- Model time series with MLP, e.g., predicting the next data point
  - Limited memory
  - Fixed window size L
  - Number of weights increases with L quickly
  - Predictions at different times are independent

- How to better model past information?

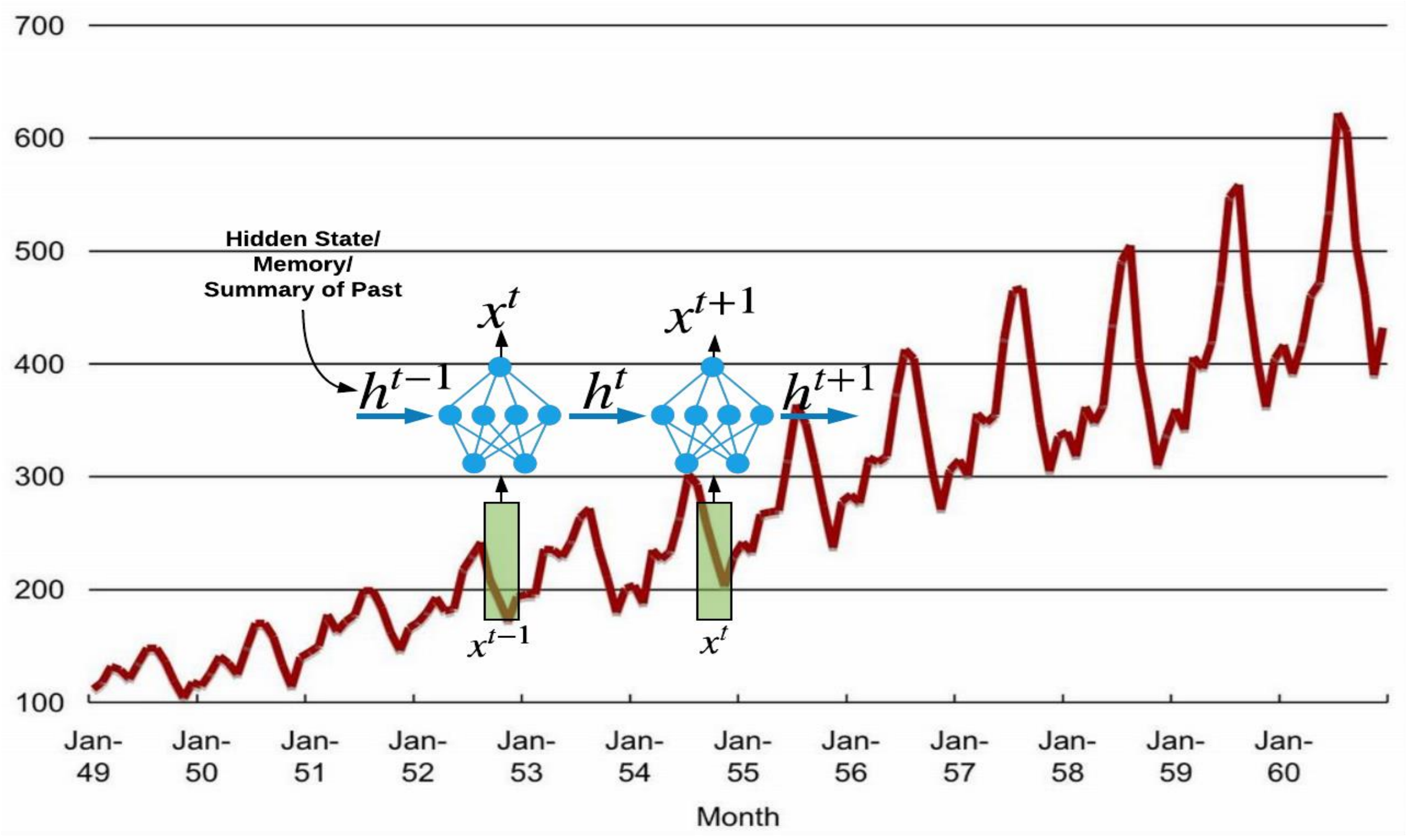


(Figure from Box and Jenkins, *Time Series Analysis: Forecasting and Control,* 1976)

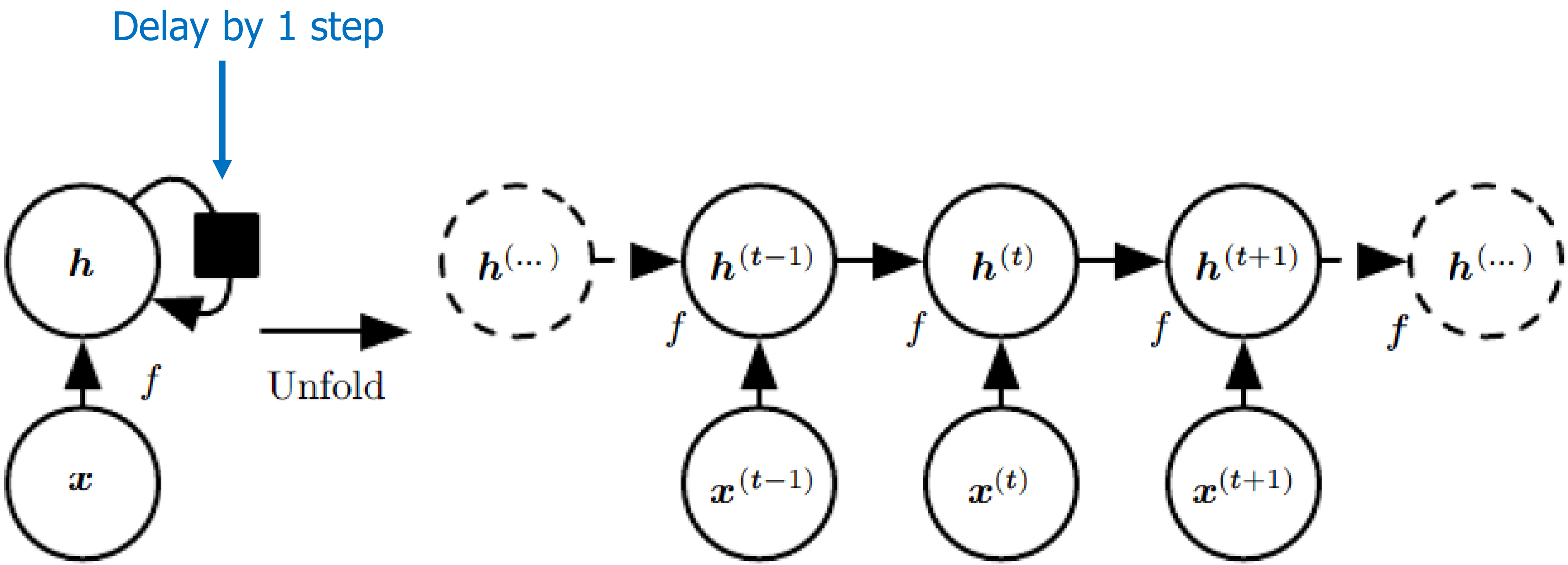
Make Network Recurrent

- Parameter sharing
  - Different positions use the same network
- Add recurrent links
  - Current computation affects future computation
  - Carry past information to the future

- Compared with 1D convolution
  - Both have weight sharing
  - Convolution has limited receptive field
  - Recurrency can carry information infinitely long (in theory)



Unfold Recurrency



(Fig. 10.2 in GBK)

$\boldsymbol{h}^{(t)}$ is affected all past input: $\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(t)}$

Different Types of Recurrency

- RNNs that produce an output at each time step and have recurrent connections between hidden units
- Take classification / labeling as example
- Forward propagation

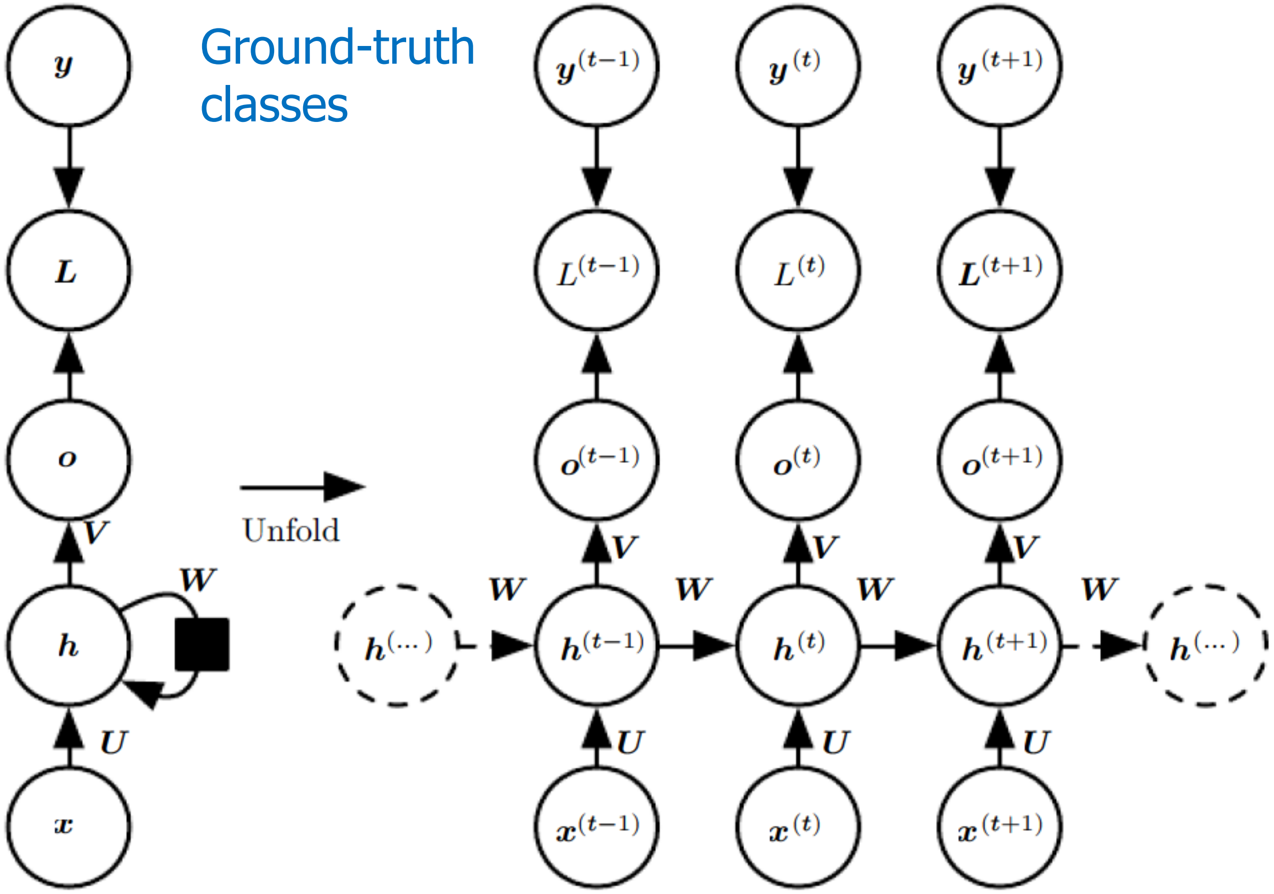
Net input to hidden $\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$

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

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

Softmax -> class prob. $\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)}),$

Cross entropy loss: $L = -\sum_t \log\left(\left[\hat{\boldsymbol{y}}^{(t)}\right]_{y^{(t)}}\right)$

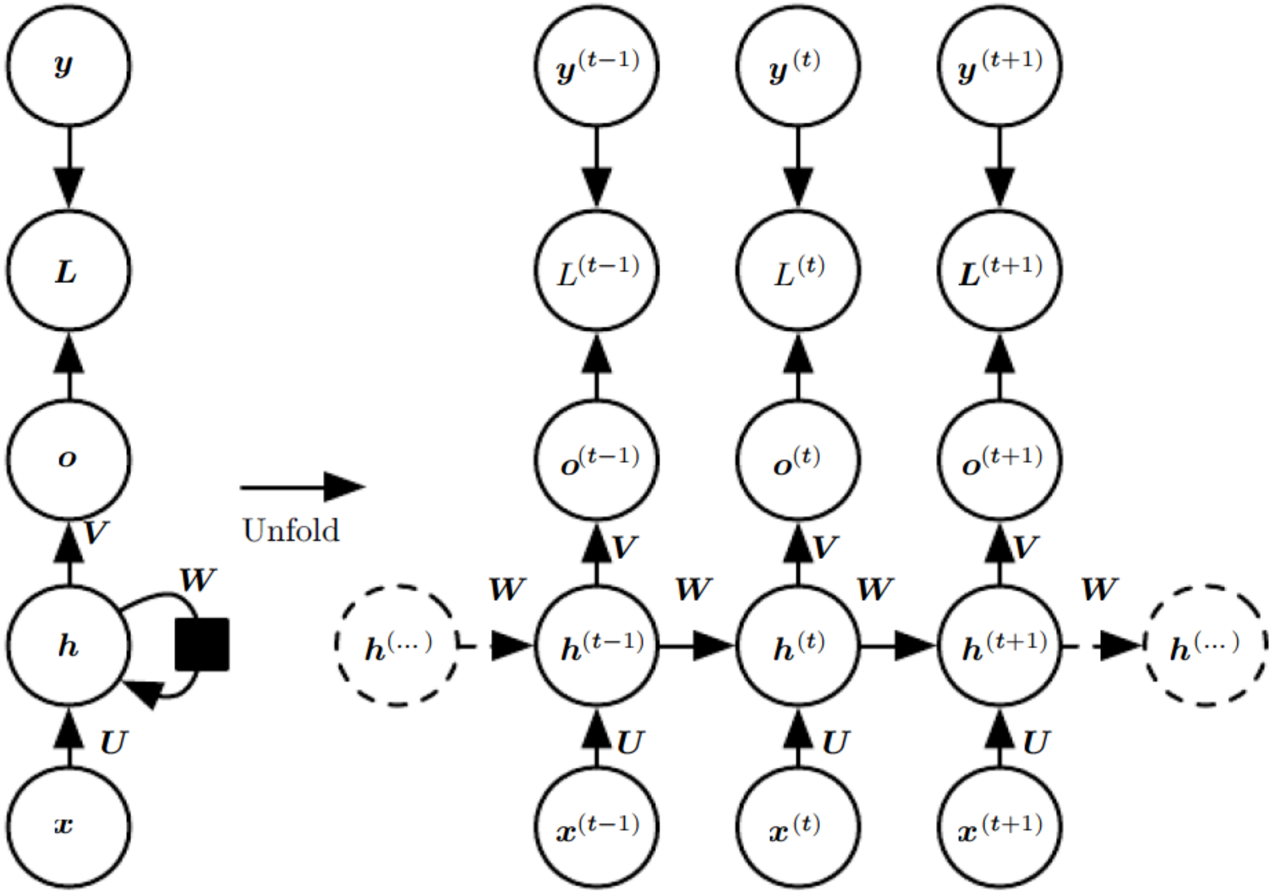


(Fig. 10.3 in GBK)

Back Propagation Through Time (BPTT)

- Output (hence loss) at time t is affected by past inputs and hidden nodes through the recurrent links
- To perform gradient descent, gradients need to pass backwards through the recurrent links

- Each update of weights requires
  - Forward computation of all hidden nodes and output nodes
  - Backpropagation of gradients
  - Both computations are sequential → cannot be parallelized → slow to train



(Fig. 10.3 in GBK)

BPTT Sketch

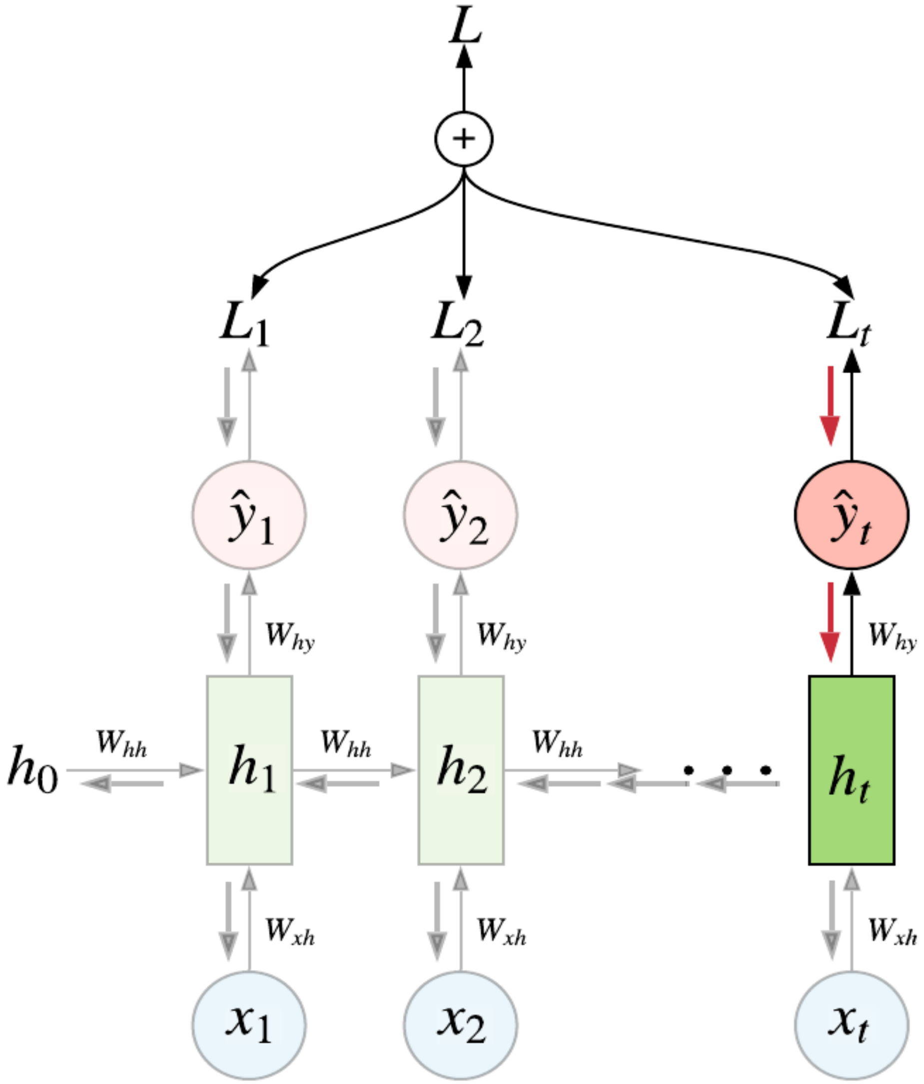
- Same as regular backpropagation → repeatedly apply chain rule
- For W_{hy}, we propagate along the vertical links

$$\frac{\partial L}{\partial W_{hy}} = \sum_{i=0}^{t} \frac{\partial L_i}{\partial W_{hy}}$$

$$\frac{\partial L_t}{\partial W_{hy}} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{W_{hy}}$$

$$\hat{y}_t = W_{hy} h_t$$

Easy to calculate



BPTT Sketch

- Same as regular backpropagation → repeatedly apply chain rule

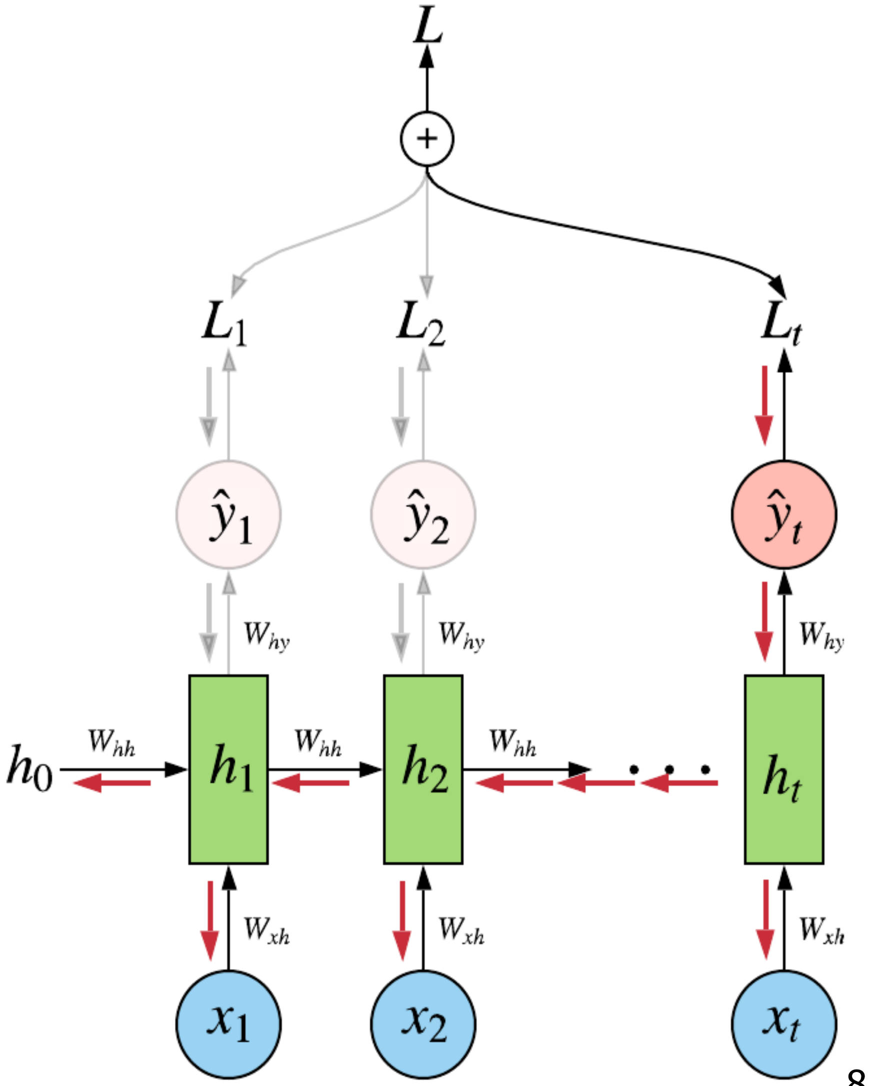
- For W_{hh} and W_{xh}, we also propagate along the horizontal (i.e., recurrent) links

$$\frac{\partial L}{\partial W_{hh}} = \sum_{i=0}^{t} \frac{\partial L_i}{\partial W_{hh}}$$

$$\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}}$$

$$h_t = tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

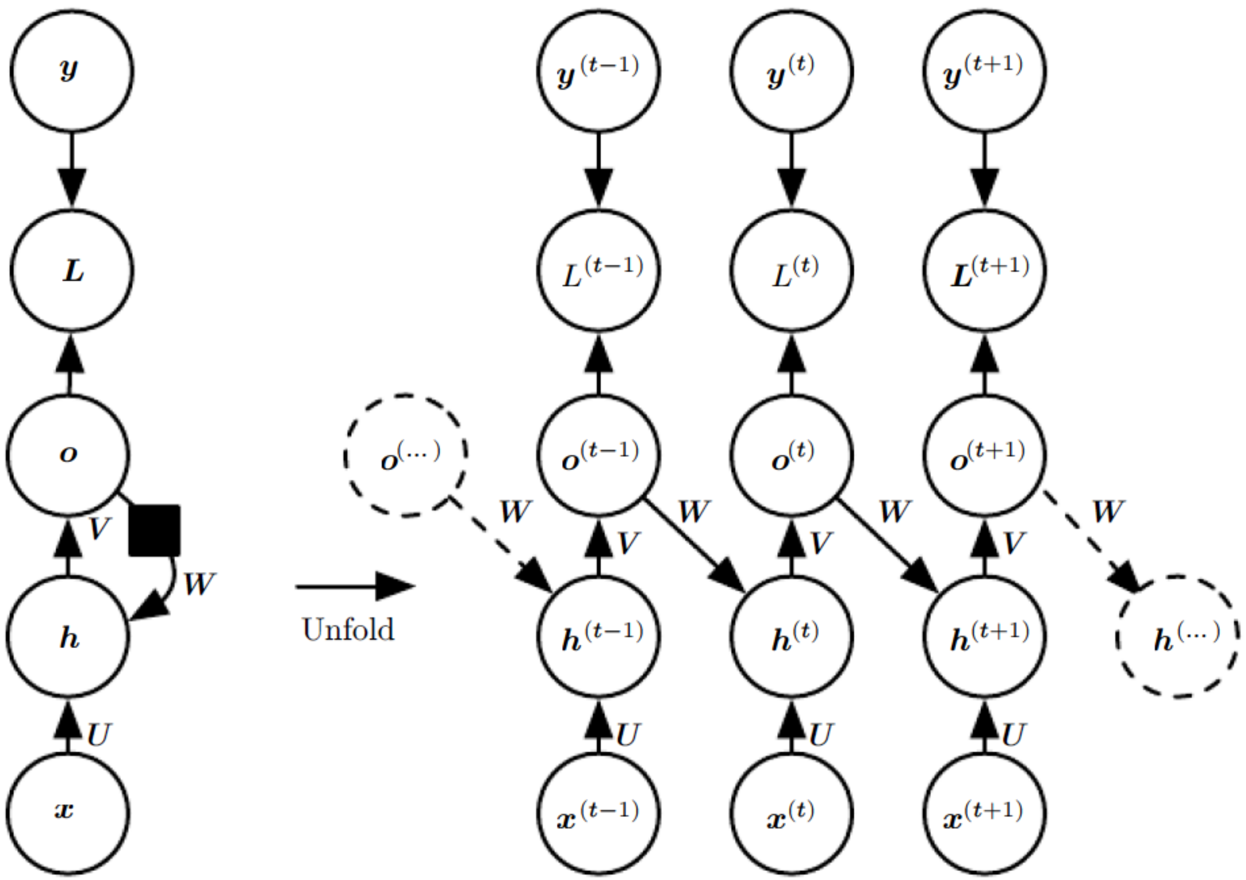
It also depends on W_{hh}



Different Types of Recurrency

- RNNs that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step

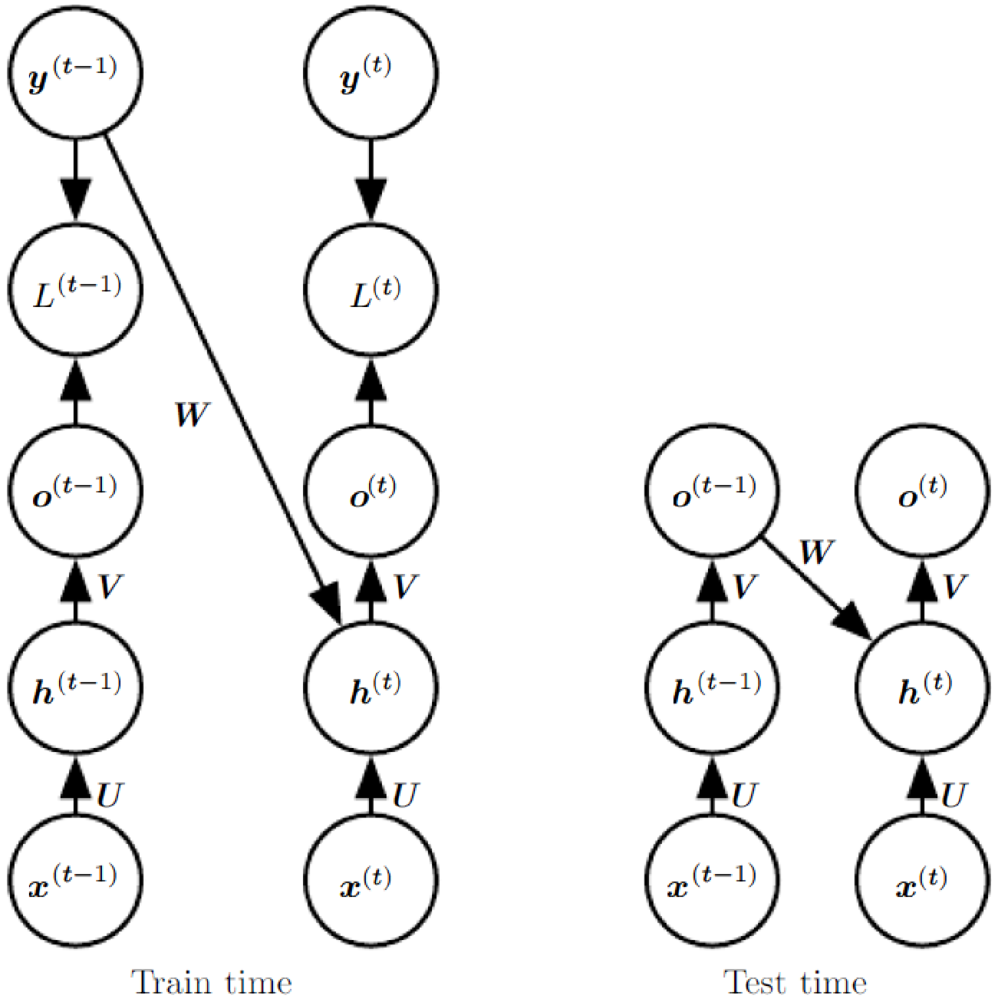
- Carry less information from past, because
  - Output nodes typically have a lower dimensionality than hidden nodes
  - Output nodes are strongly influenced by ground-truth y during training



(Fig. 10.4 in GBK)

Teacher Forcing

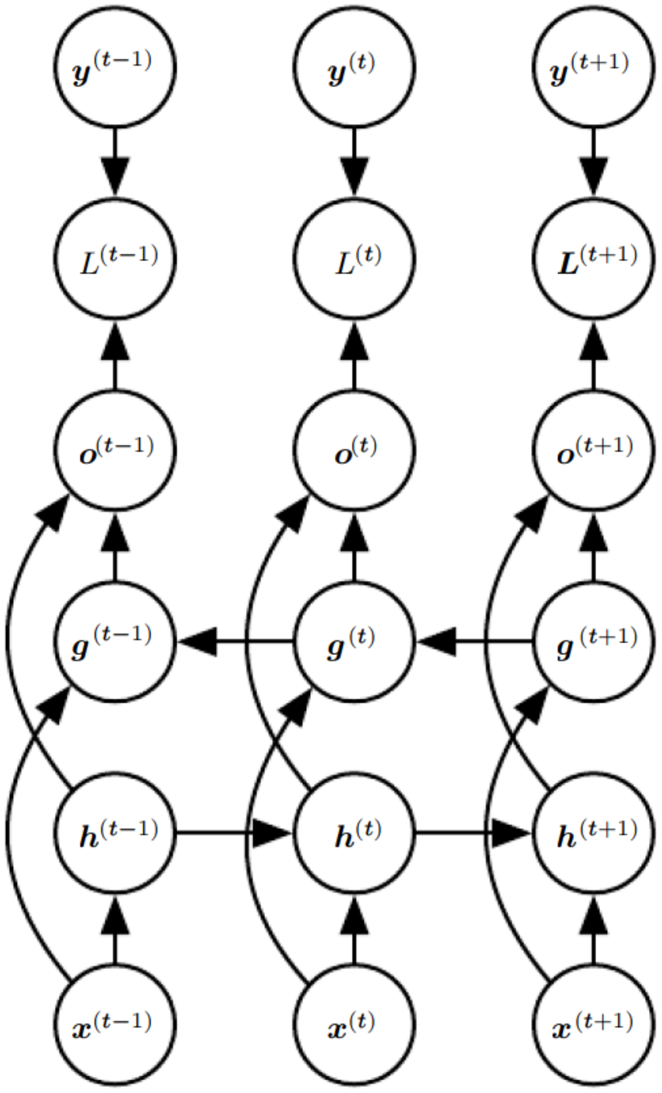
- When recurrent links are only from output to hidden, then teacher forcing (i.e., feeding ground-truth output to hidden) can be used to parallelize training
- It essentially only trains network to make 1-step predictions
- During inference, $y^{(t-1)}$ is not available for predicting $y^{(t)}$, causing mismatch from training
  - Scheduled sampling: mix ground-truth outputs and free-run outputs during training with a ratio that gradually decreases



(Fig. 10.6 in GBK)

Bidirectional RNN

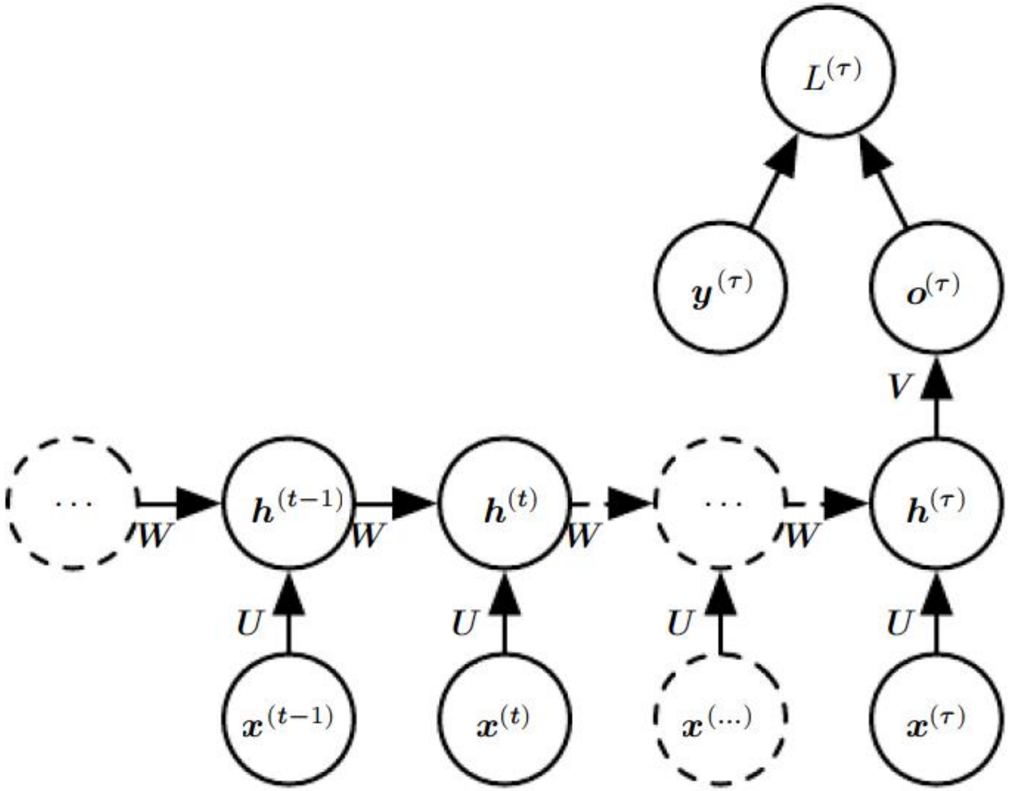
- RNNs introduced so far are causal, i.e., the output at the current time step is only affected by the current input and past inputs
- In some applications (e.g., filling a missing word in a sentence, speech recognition), output has dependencies on inputs from both sides
- Let's use two RNNs, one for each direction
- Their hidden values work together to give output



(Fig. 10.11 in GBK)

RNN with a Single Output

- Some tasks only require a single output from the input sequence
  - E.g., phoneme classification, sound event recognition

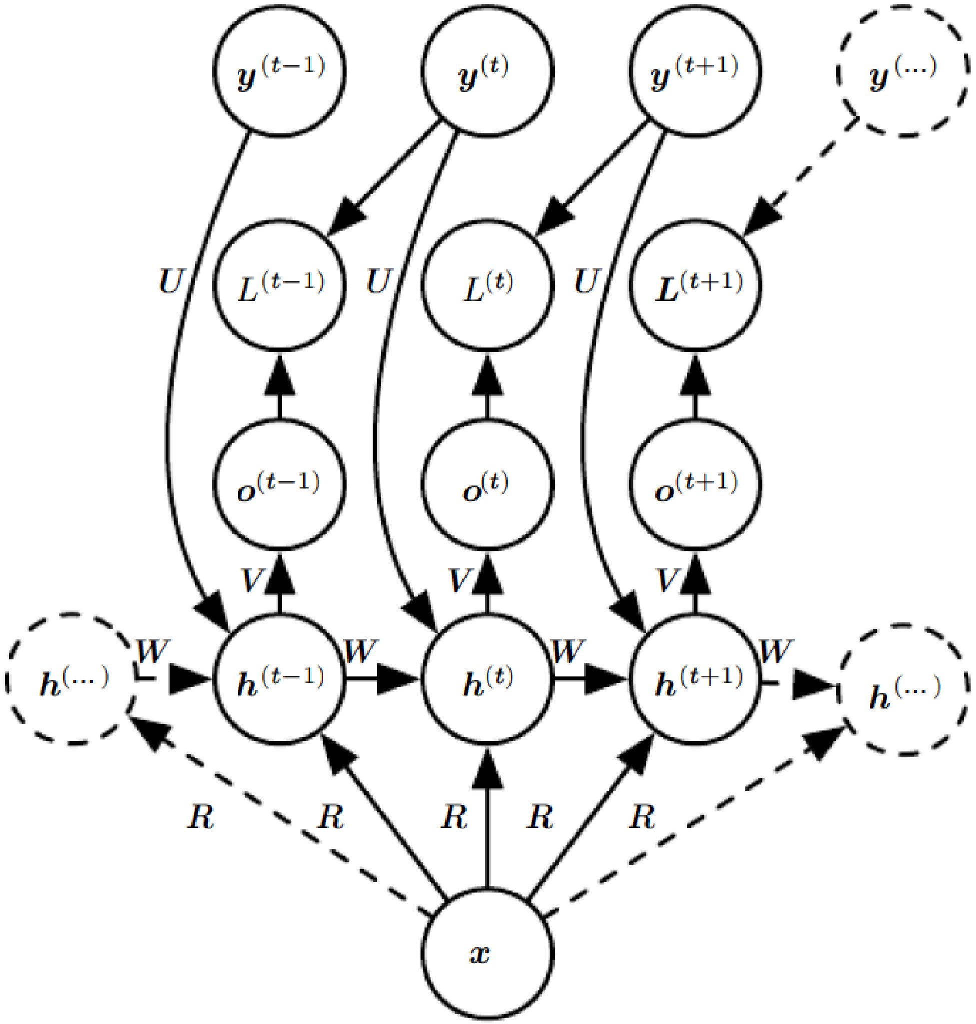


(Fig. 10.5 in GBK)

RNN with Context Conditioning

- Output a sequence from a conditioning vector
  - E.g., laughter sound generation, conditioned on the type of laughter
  - E.g., image captioning, conditioned on image
  - E.g., emotional talking face generation, conditioned on emotion label

- This conditioning vector can be input to the network
  - As extra input at each time step (right figure)
  - As the initial state $\boldsymbol{h}^{(0)}$
  - Both



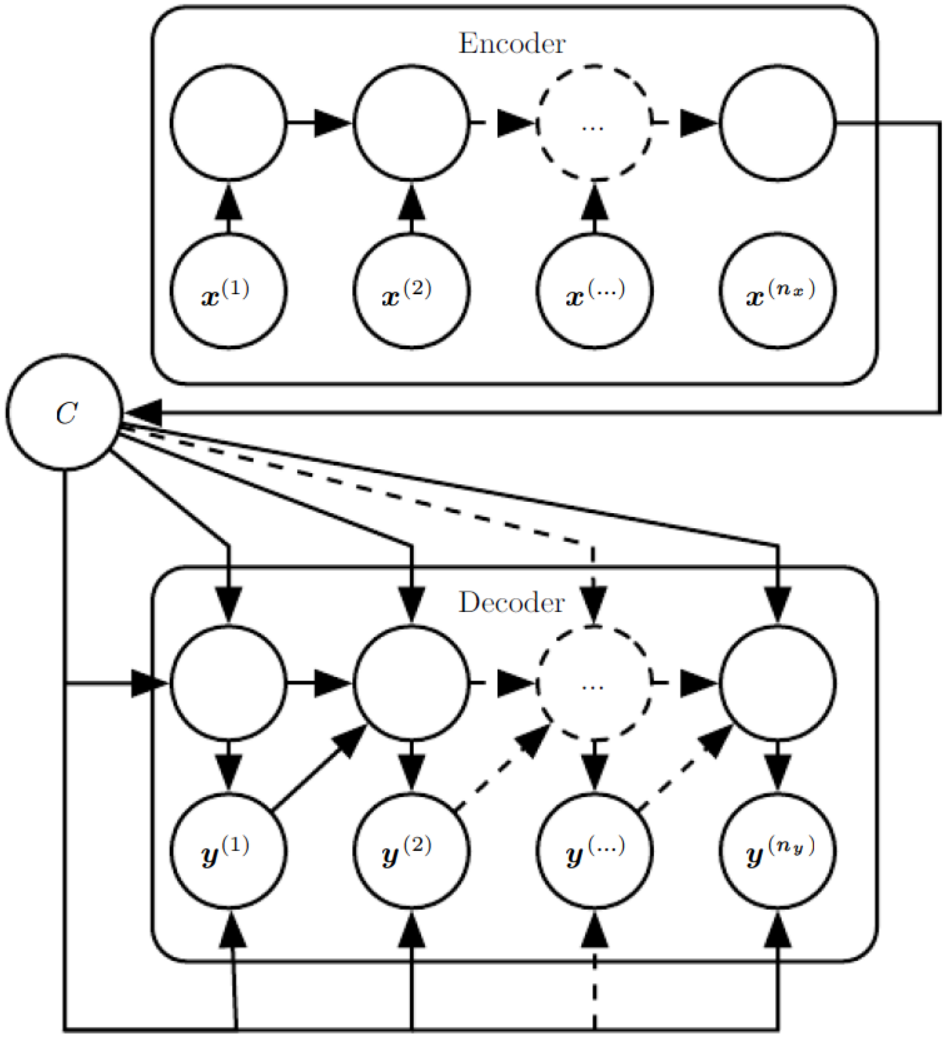
(Fig. 10.9 in WBK)

Encoder-Decoder Sequence-to-Sequence RNNs

- Sometimes the input and output sequences are of different length
  - E.g., machine translation from English to Chinese
  - E.g., audio captioning

- Encoder is an RNN with a single output

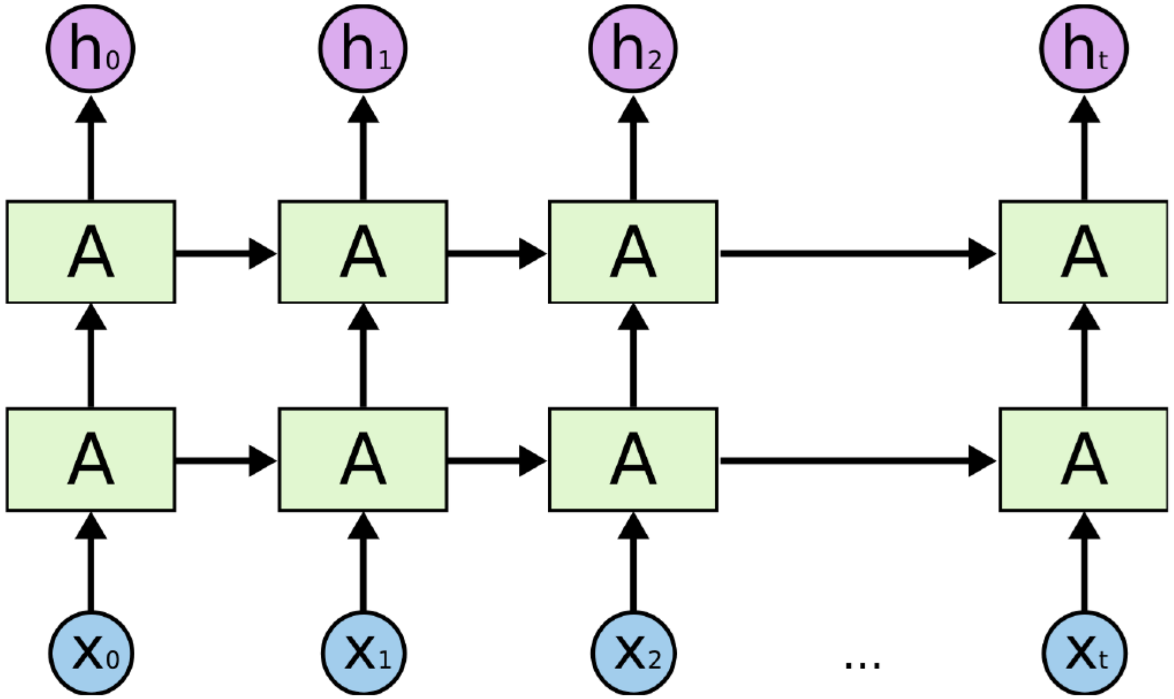
- Decoder is an RNN with context conditioning



(Fig. 10.12 in GBK)

Deep RNNs

- RNNs we introduced so far have only one hidden layer
- There are many ways to make them deeper, but a common way is to stack RNNs



Vanishing & Exploding Gradients

- Recurrency applies the same function repeatedly, and will exponentially diminish or boost certain effects
- Look at linear recurrency as an example

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}\boldsymbol{h}^{(t-1)} = \boldsymbol{W}^t\boldsymbol{h}^{(0)}$$

- Let $\boldsymbol{W}$ have eigenvalue decomposition

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$$

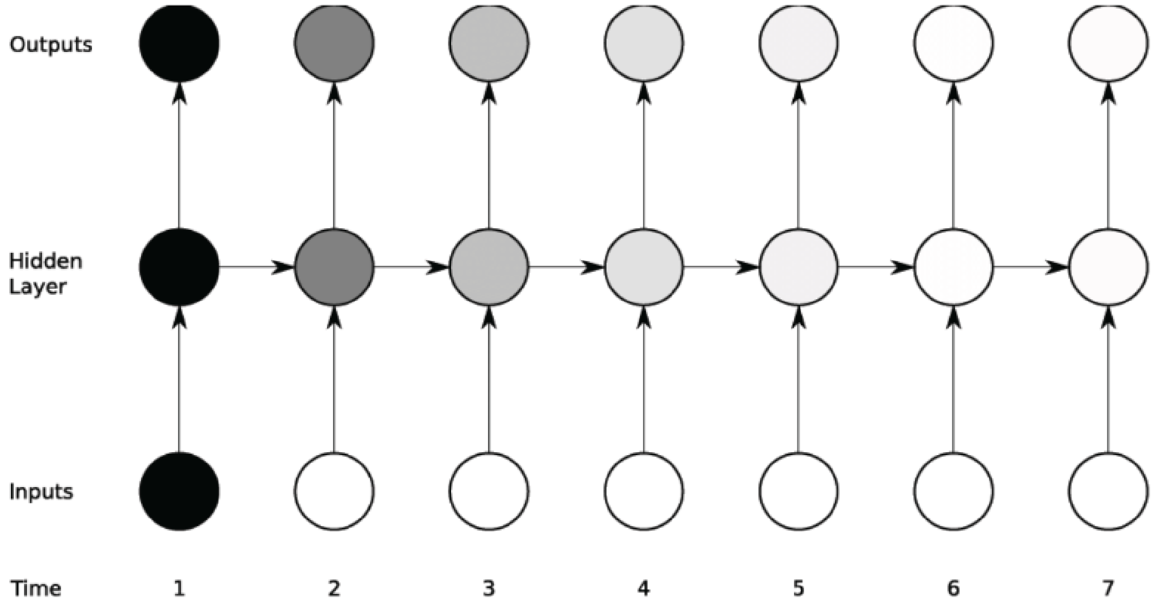
- Then we have

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}\boldsymbol{\Lambda}^{\mathrm{t}}\boldsymbol{Q}^{-1}\boldsymbol{h}^{(0)}$$

- Eigenvalues are raised to the power of t!
  - If $\boldsymbol{h}^{(0)}$ is aligned with an eigenvector with eigenvalue greater than 1, then explode
  - If $\boldsymbol{h}^{(0)}$ is aligned with an eigenvector with eigenvalue smaller than 1, then vanish

Vanishing & Exploding Gradients

- Vanishing gradients are very common for RNNs



Darkness indicates the influence of input at time 1
Figure from [Graves, 2008]

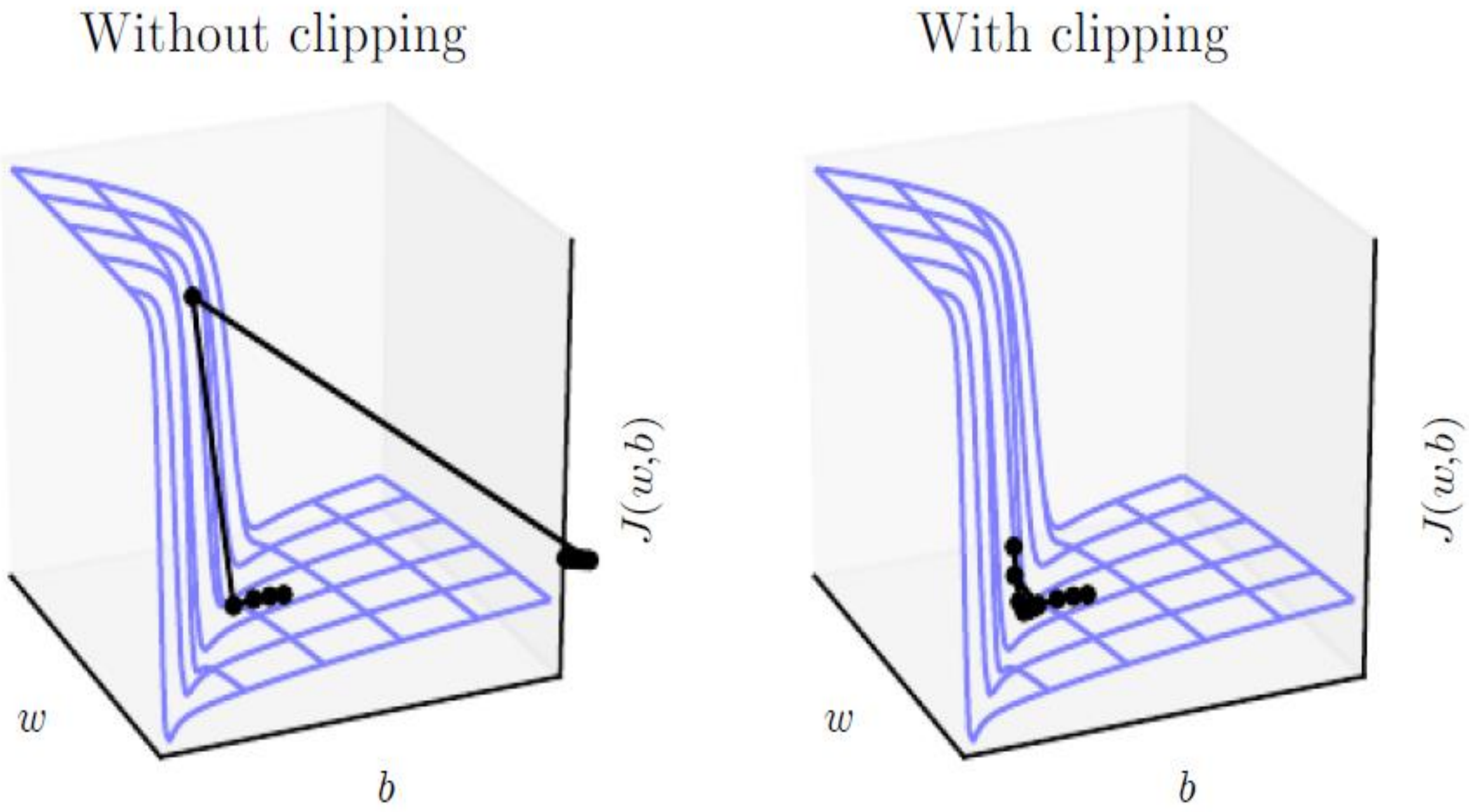
- Exploding gradients also happen, and it damages the optimization very much

Gradient Clipping

- Too big gradients will make too big updates of network parameters
- Clip the norm of gradients $\boldsymbol{g}$ to v:

$$\text{if } ||\boldsymbol{g}|| > v$$

$$\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}v}{||\boldsymbol{g}||}$$



(Fig. 10.17 in GBC)

Improving Long-Term Dependency Modeling

- Temporal dependencies in data can be very long
  - E.g., music rhythmic structure is at the scale of seconds, where each second often contains 44100 samples (time domain) or ~100 frames (time-frequency domain)

- Influence of input vanishes exponentially over time steps
  - In practice, after ten steps, influence is already negligible

- Several ways to improve long-term dependency
  - Add skip connections through time: allows information to flow with fewer time steps
  - Add linear self-connections to hidden units, called leaky units, similar to running average: $\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)}$. When α is close to 1, it allows hidden units to remember information for a long time.
  - Add gates to control information flow

Gated Architectures - LSTM

- Cell state (leaky unit) is the internal memory
- Three information gates perform delete/write/read operations on memory

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$

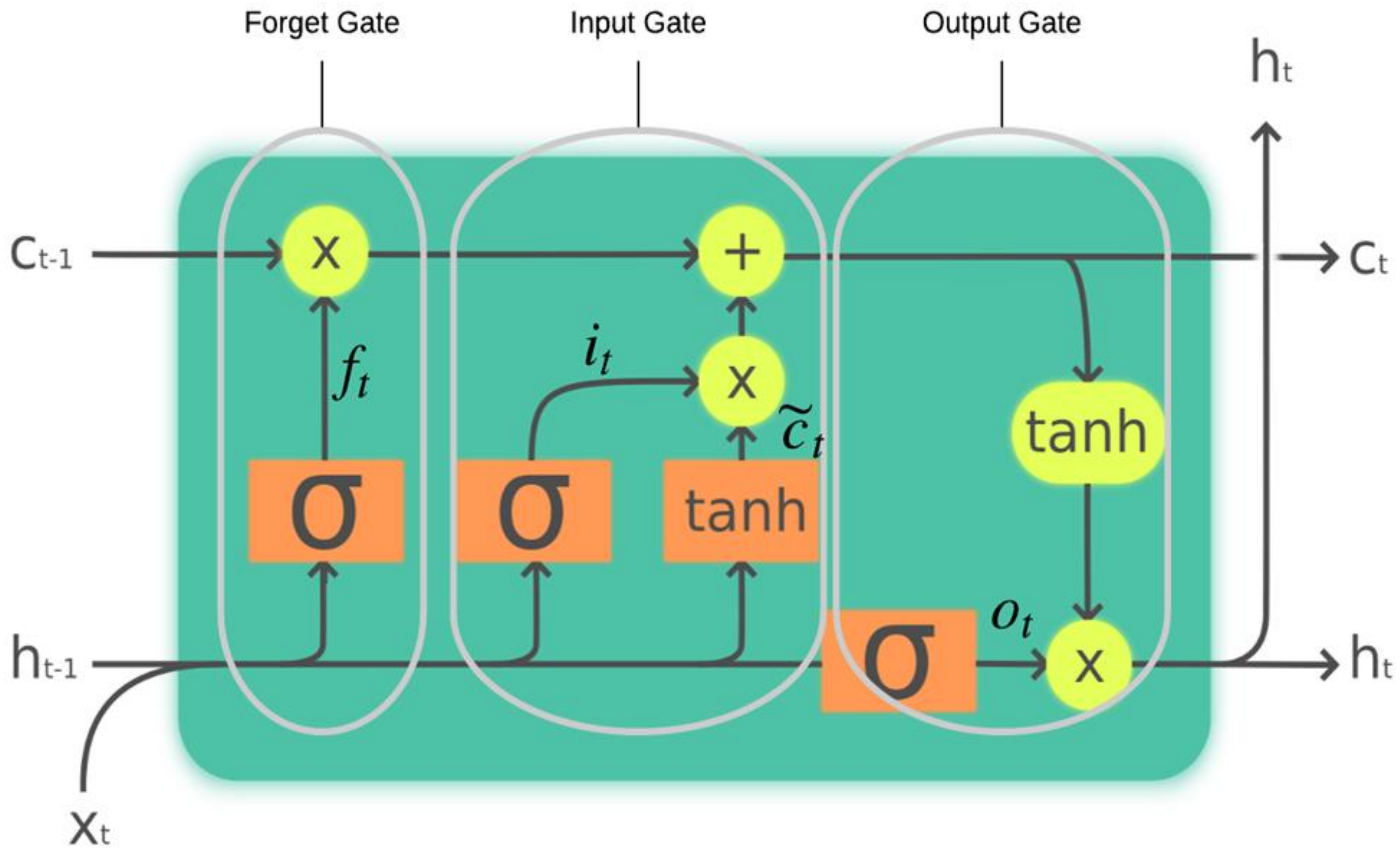
$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

$$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

$$h_t = o_t * tanh(c^t)$$



Gated Architecture - GRU

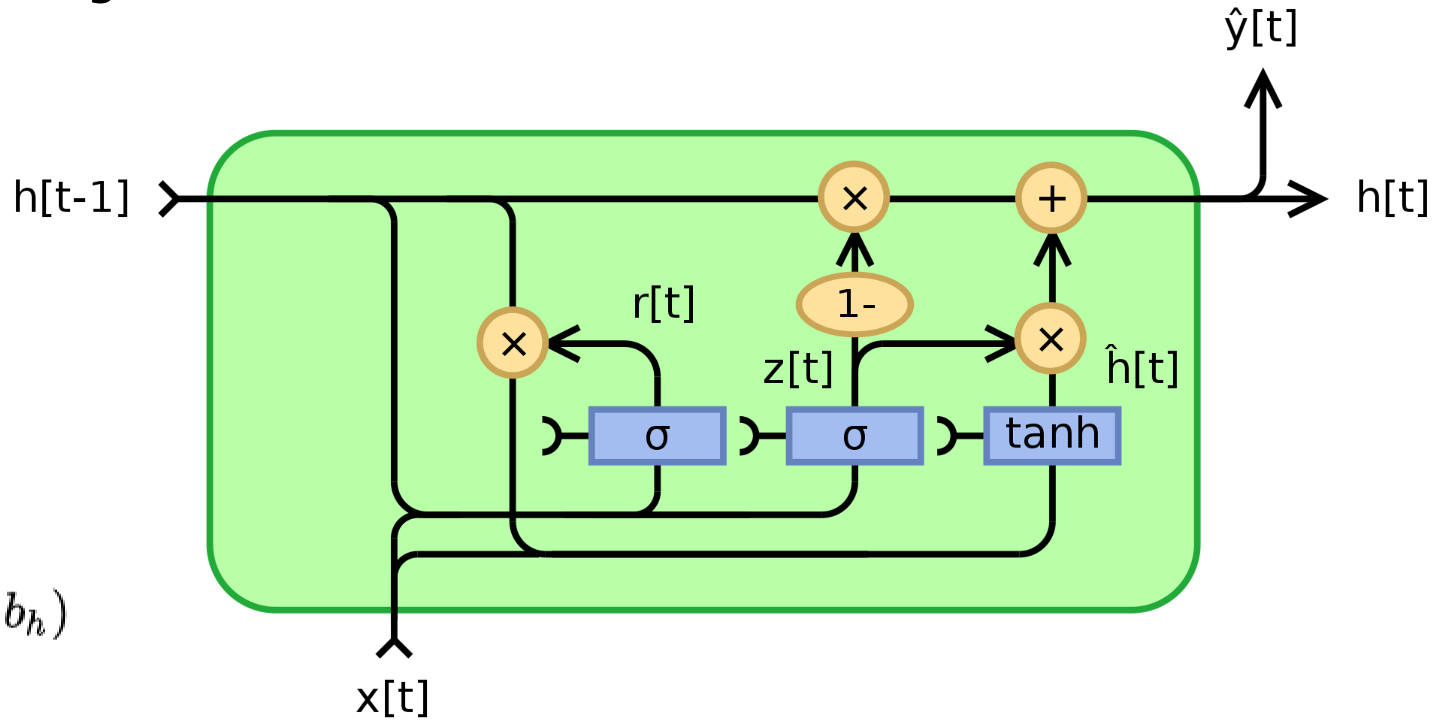
- Gated Recurrent Unit (GRU)
  - A single gate to simultaneously control the forgetting factor and the updating operation of the state unit
  - Fewer parameters than LSTM
  - Similar performance

Update gate $z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$

Reset gate $r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$

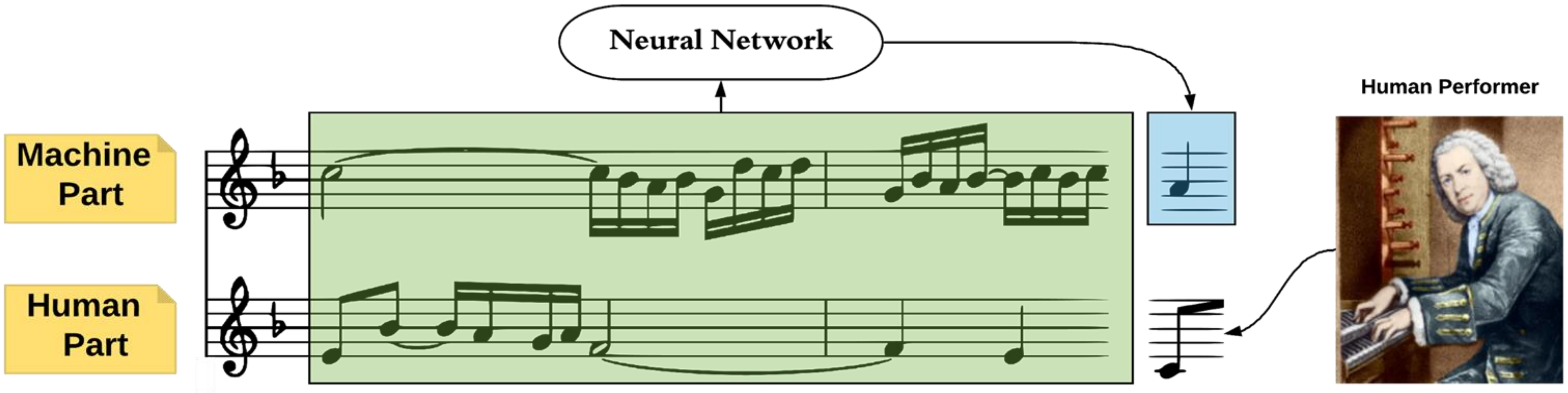
$$\hat{h}_t = \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

Output $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t$

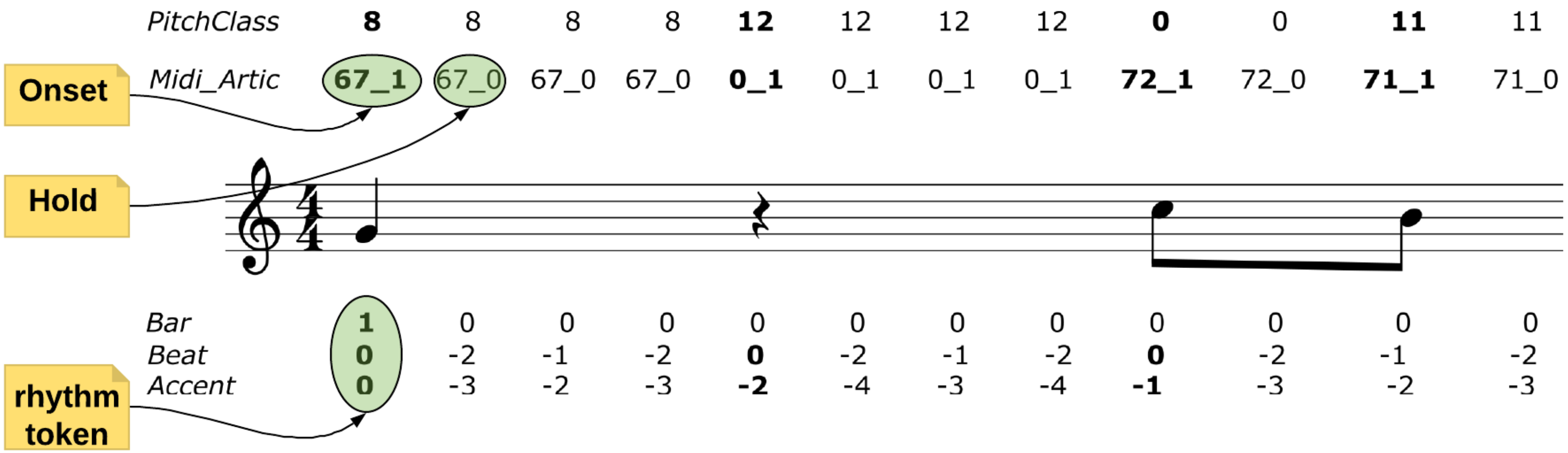


(Figure from https://en.wikipedia.org/wiki/Gated_recurrent_unit)

Application: Music Generation

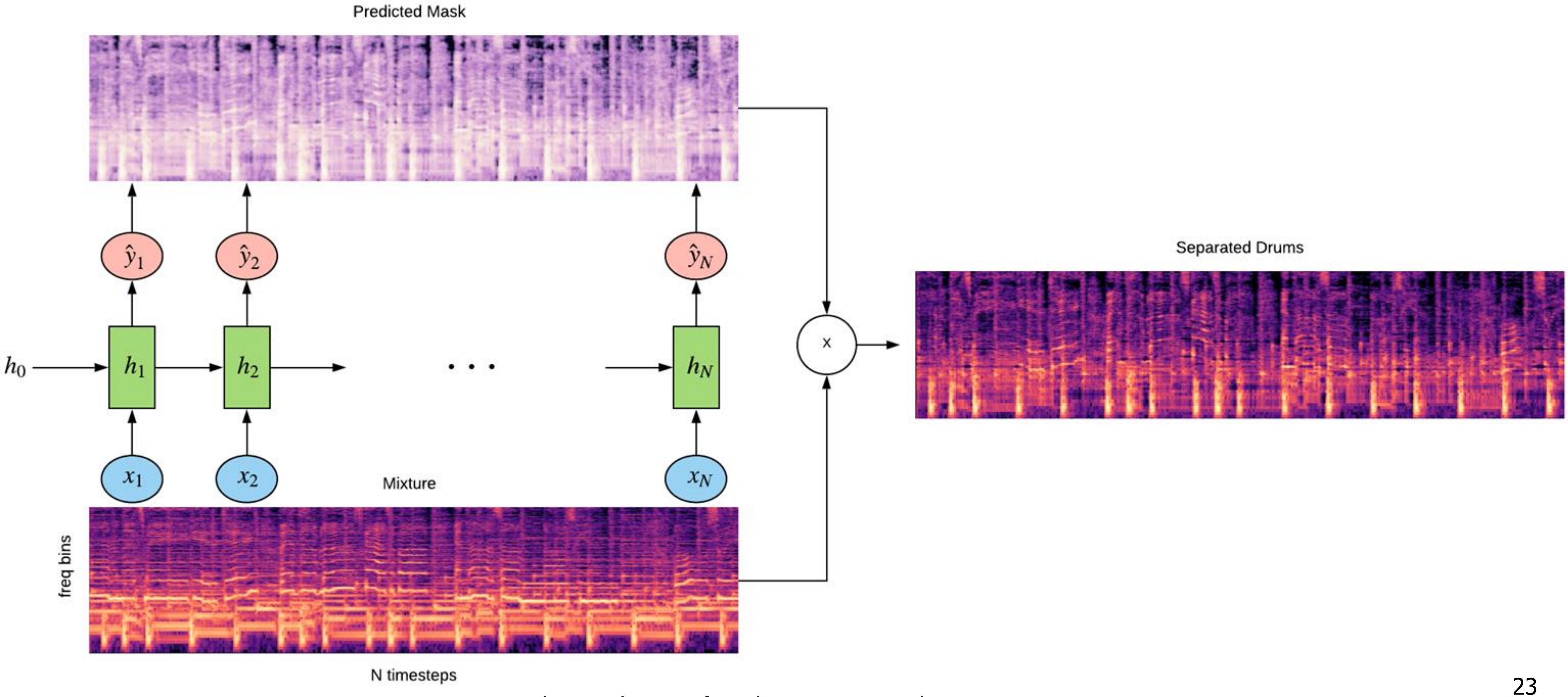


Benetatos, VanderStel, & Duan, **BachDuet: A deep learning system for human-machine counterpoint improvisation**, NIME, 2020.



Yan, Lustig, Vaderstel, & Duan, **Part-invariant model for music generation and harmonization**, ISMIR, 2018.

Application: Audio Source Separation



Summary

- Recurrent Neural Networks (RNNs)
  - Weight sharing over time
  - Recurrent links to carry information infinitely long (in theory)
- Different kinds of recurrencies
  - Hidden to hidden
  - Output to hidden
- Different RNN architectures
  - N to N, N to 1, 1 to N, N to M
- Back Propagation Through Time (BPTT)
  - Vanishing and exploding gradients due to repeatedly compositing the same function
  - Gradient clipping
- Long Short-Term Memory
  - Linear self connections to remember information longer
  - (Learnable) gated architecture to control information flow