Recurrent NN

ENEE 4583/5583 Neural Nets

Dr. Alsamman

Slide Credits:

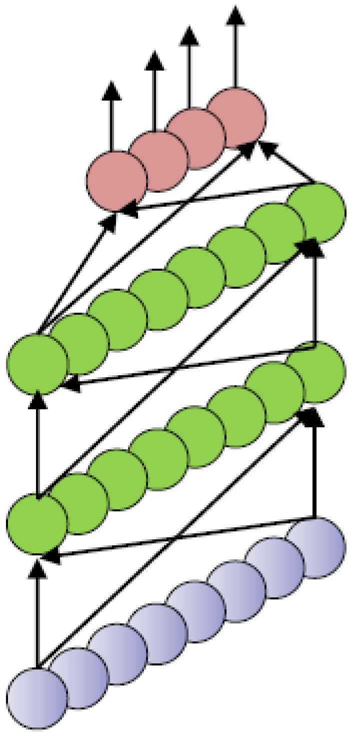
Sequential Data Input

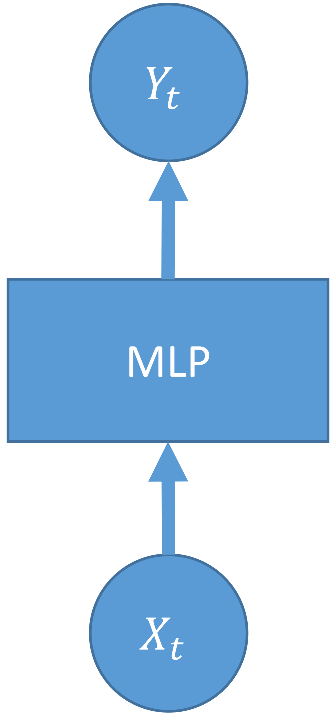
- Ordered data
- Spatial dependent order: text
- Physical order: chemical, DNA sequence
- Time order: audio, finance, medical
- Spatial and time order: video,

Sequential Driven Output

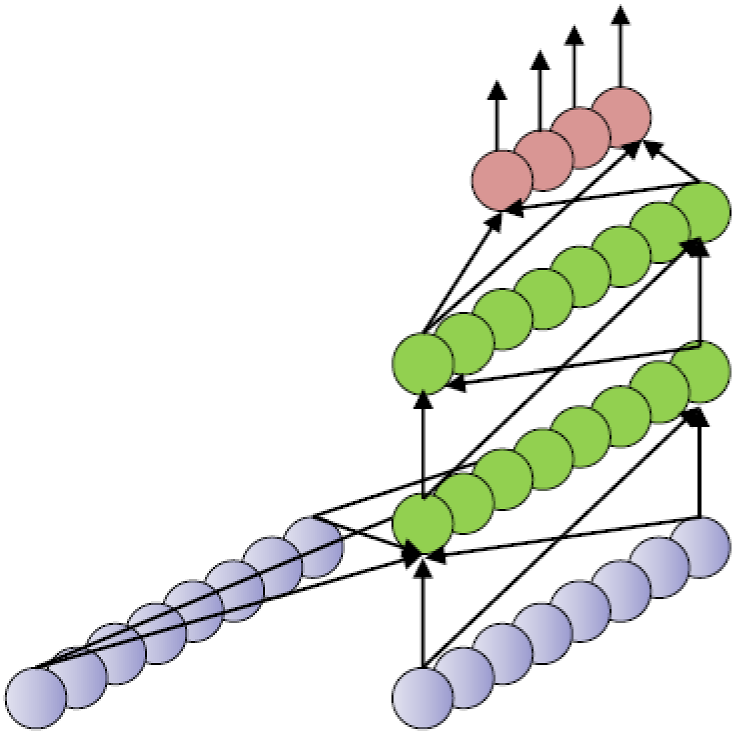
- Single output dependent on past sequence of input
  - Classification of a sequence
  - E.g. sentiment classification
- Sequence output dependent on past single input
  - Sequence generation
  - E.g. image caption
- sequence output dependent on current sequence of input
  - Updated prediction
  - E.g. stock prediction
- Current sequence output dependent on past sequence of input
  - Delayed prediction
  - E.g. language translation

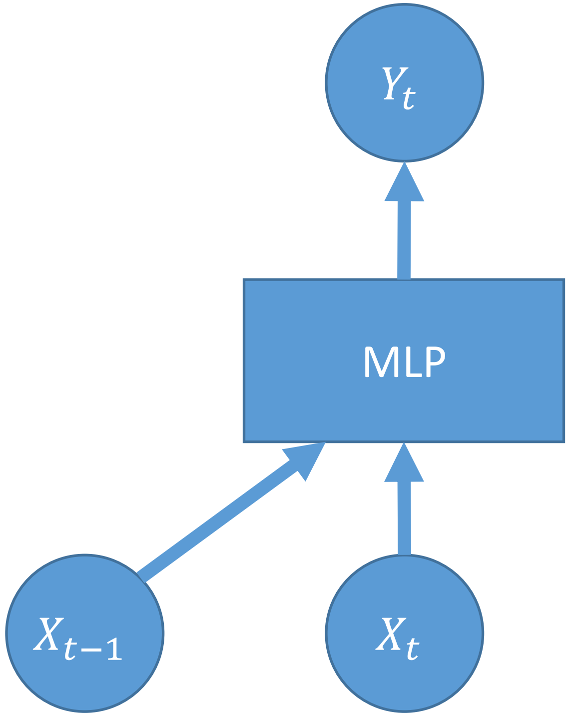
Diagram Representation: Example 1



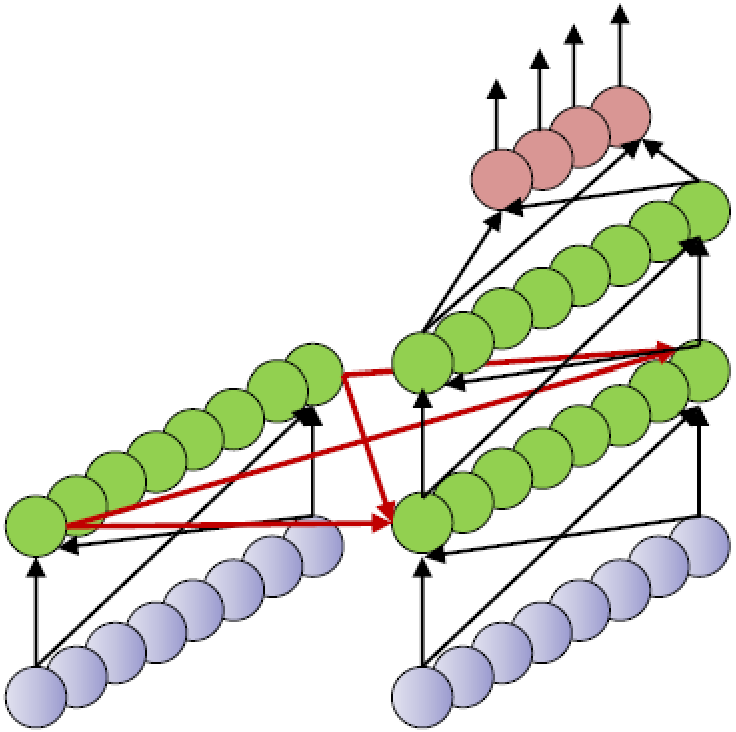


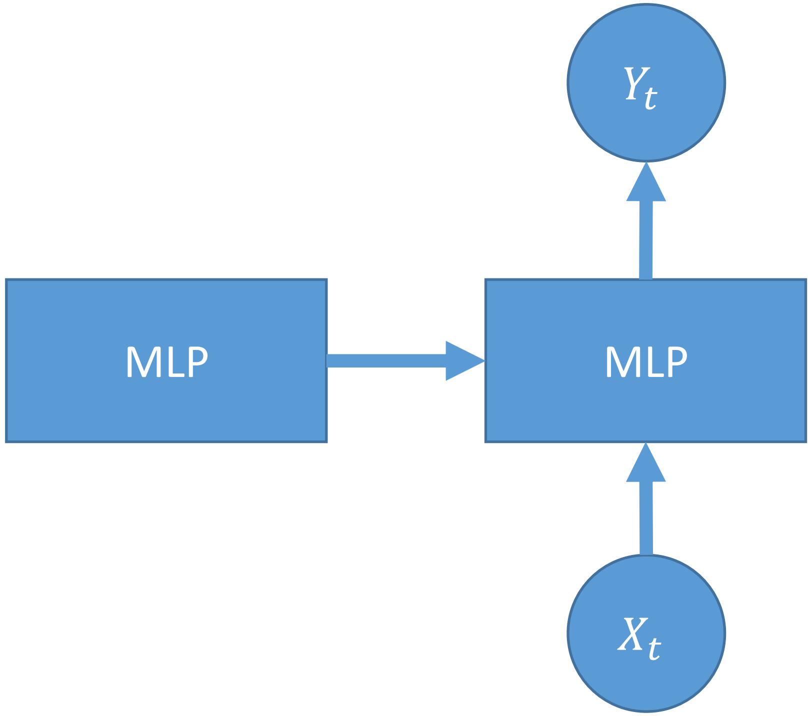
Example 2





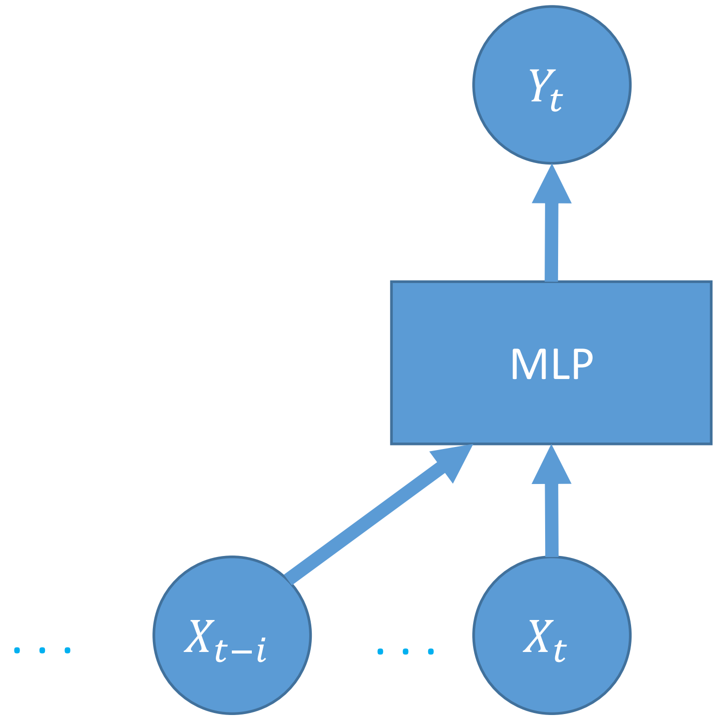
Example 3





Problems with MLP

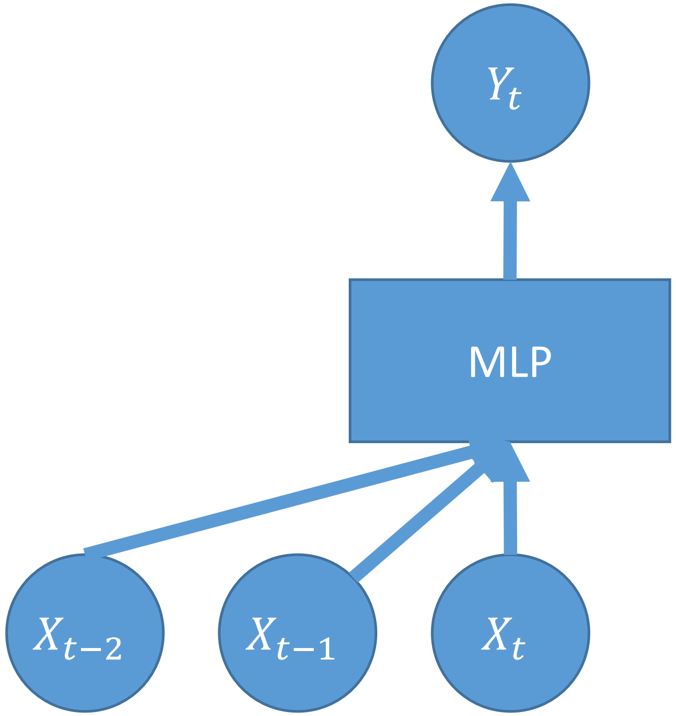
- Problem 1: Sequence inputs can be arbitrary length
- Solution: fixed window size as input
- Problem 2: choosing problem size
- Problem 3: number of parameters can explode
  - 100 hidden neurons x 100 inputs x 100 words > 1M



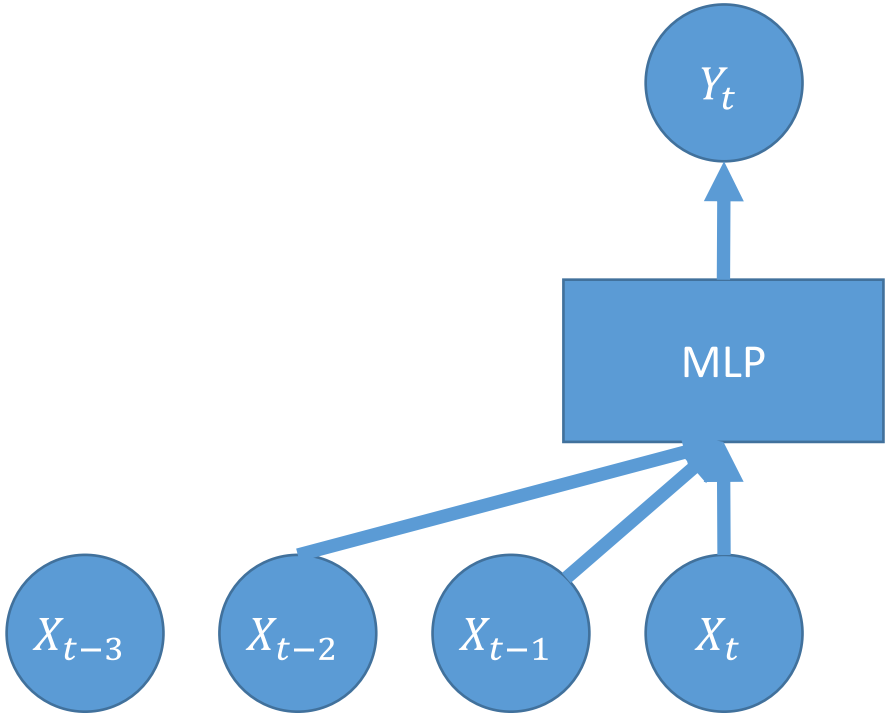
TDNN (1989)

- Time delayed NN
  - E.g stock predictor
- Paper: *Phoneme Recognition Using Time-Delay Neural Networks*
  - Weibel, Hanzawa, Hinton, Shikano
  - Inspired by Fukushima’s Neocognitron

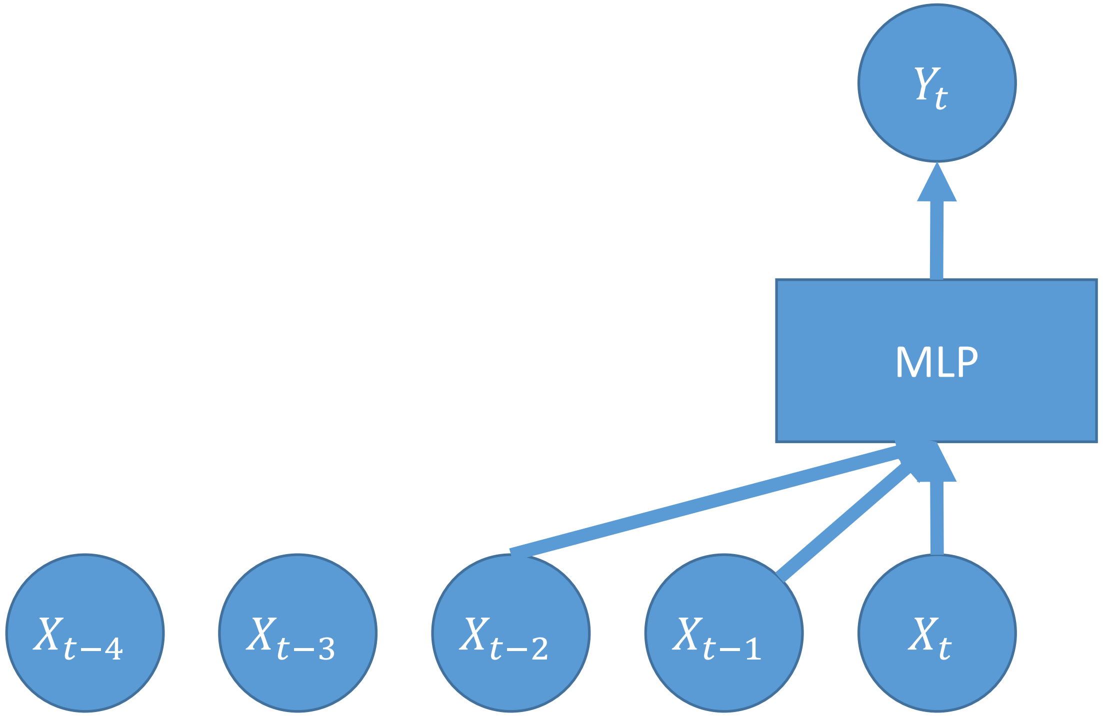
T=3



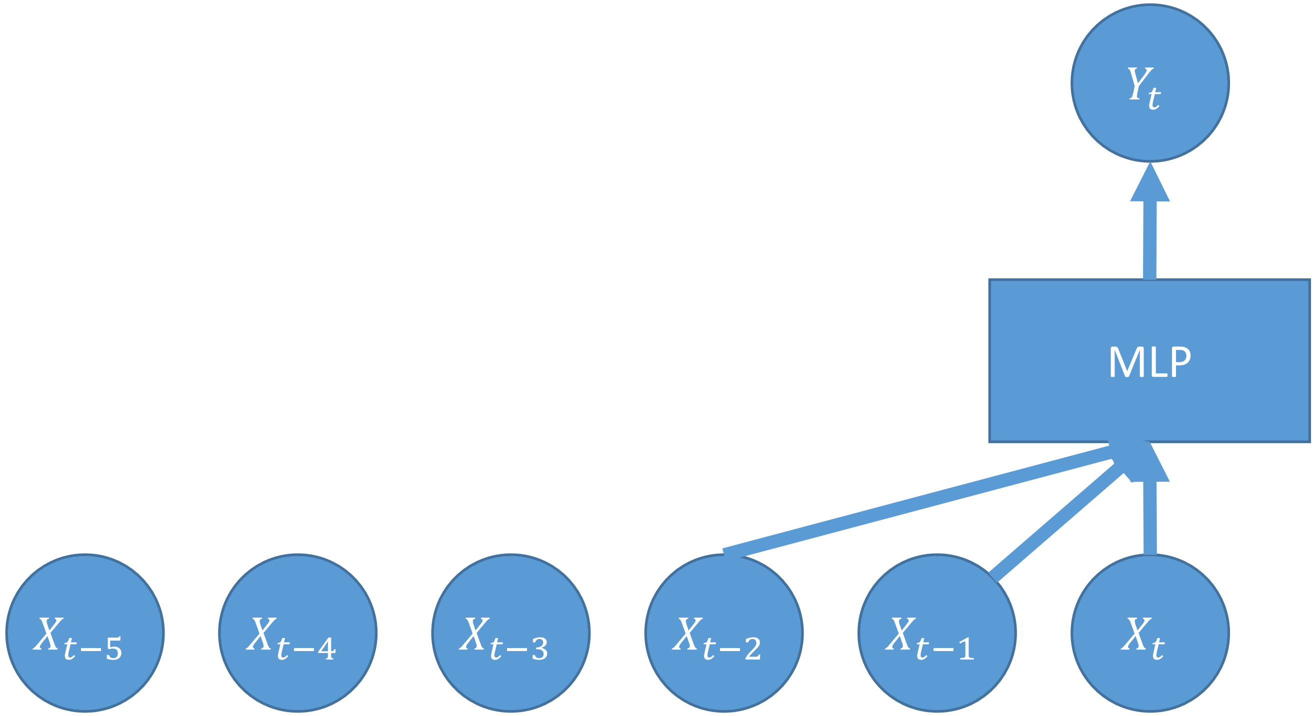
T=4



T=5



T=6

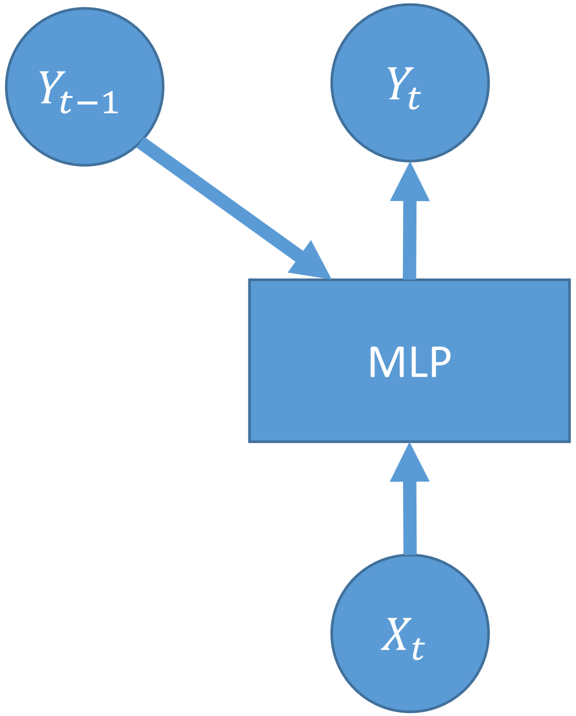


Limitations of TDNN

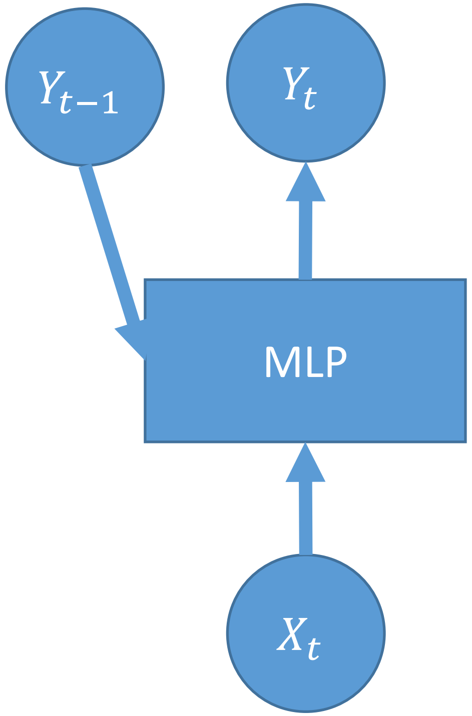
- Finite response system:
  - Output driven by past N-T=s only
- Sliding predictor
  - Much like a convolutional nnet
- Problem: some trends are “seasonal”
  - Bias the output
- Prefer: Infinite response system
  - Would like to learn “trends”
  - With weaker and weaker influence

NARX (1985)

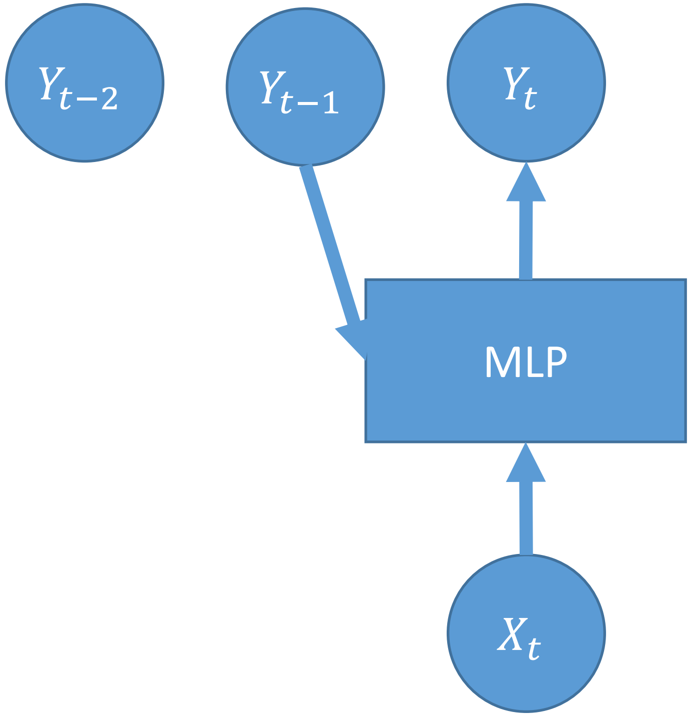
- Nonlinear autoregressive exogenous model
  - Leontaritis & Billings
- Recursion from output
- Popular for time-series
  - Weather
  - Stock-market
  - Tracking



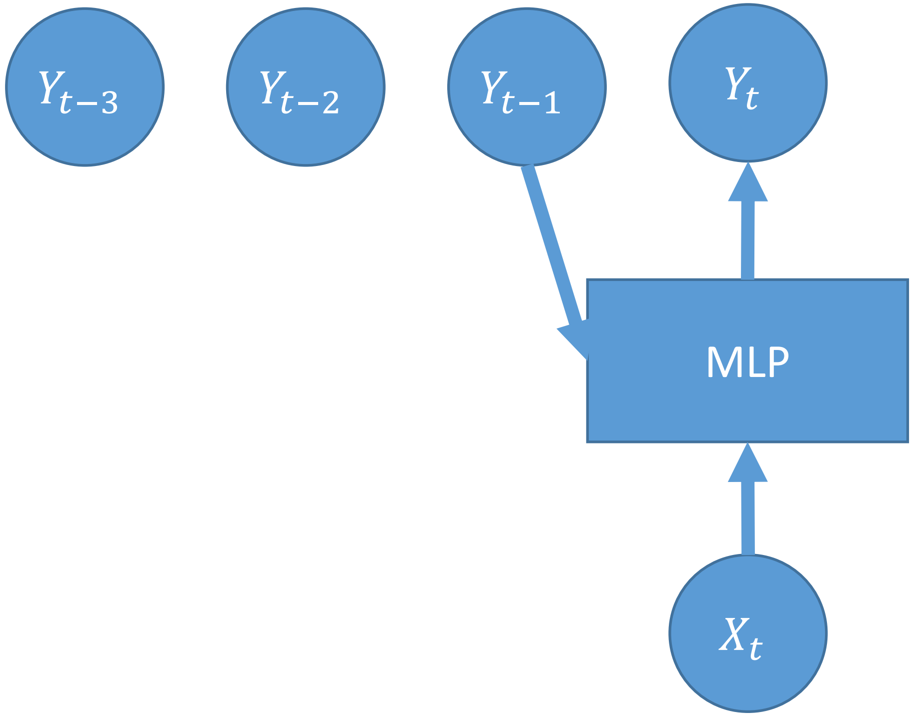
T=2



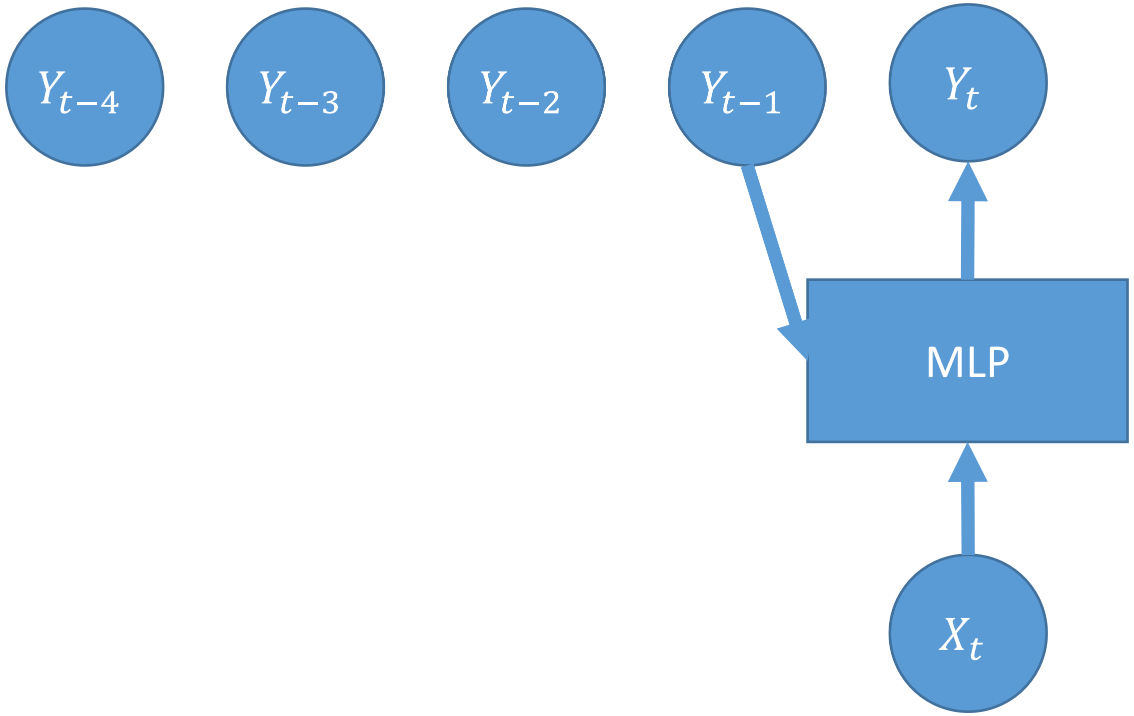
T=3



T=4

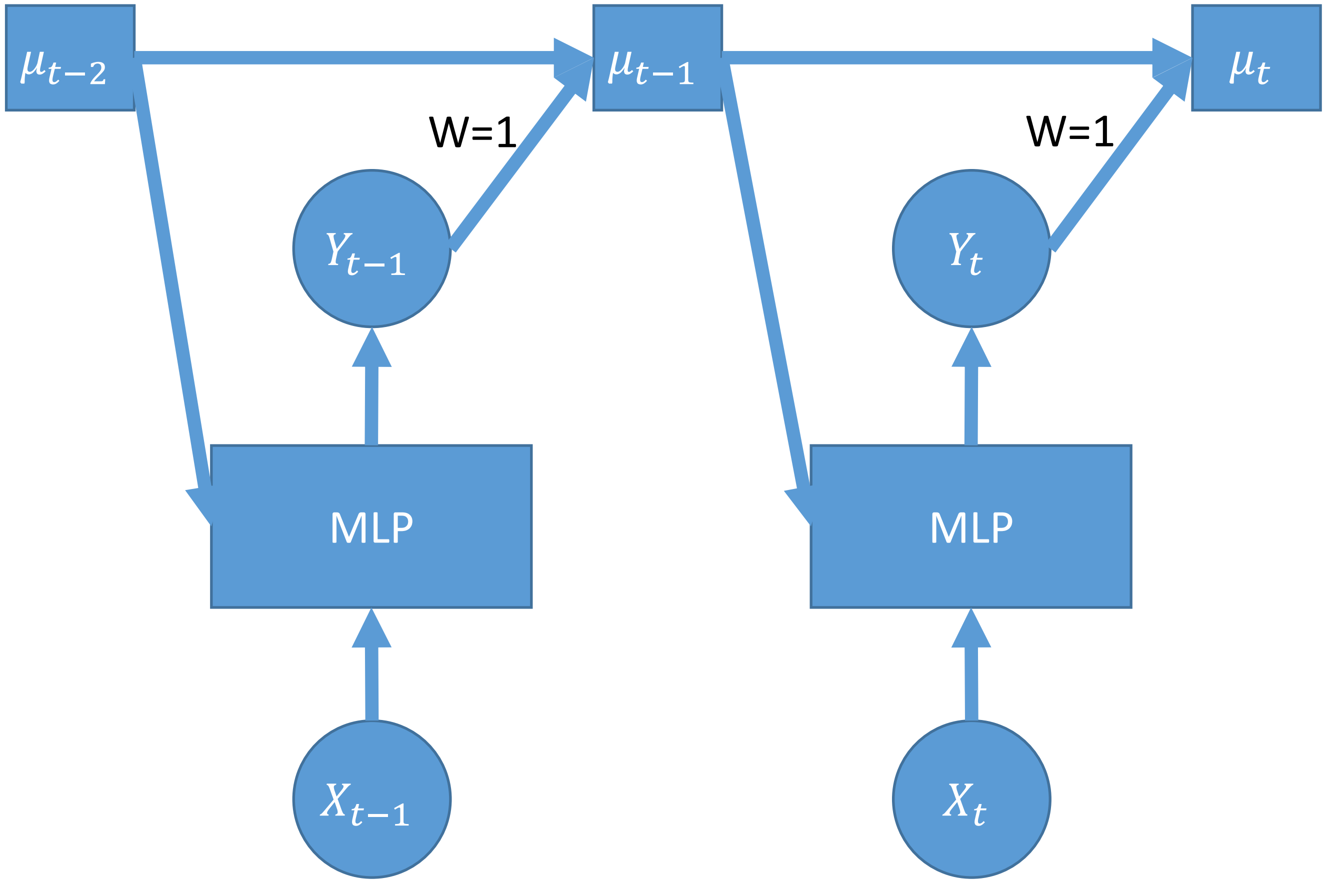


T=5



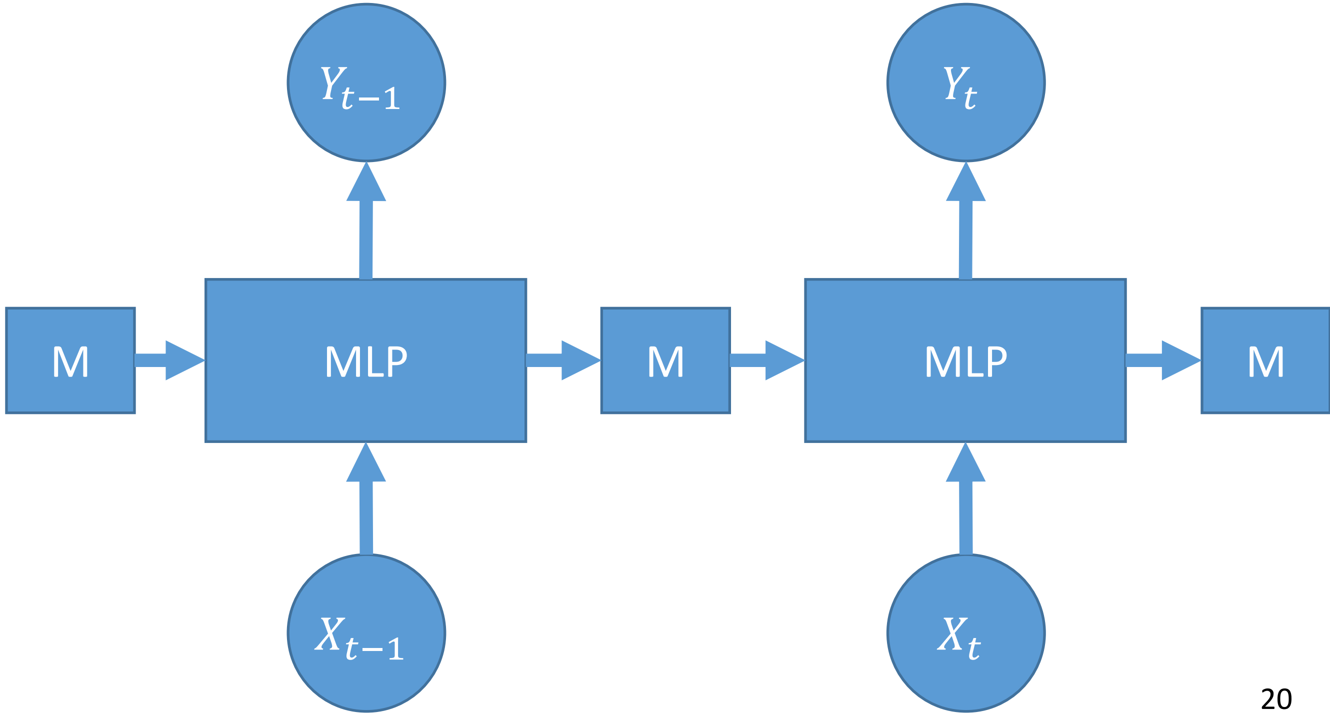
Jordan Net (1986)

- Memory is a running average of outputs
  - Stored statistic
  - Doesn’t learn to remember
- Fixed weights
  - $W_{Y\mu} = 1$



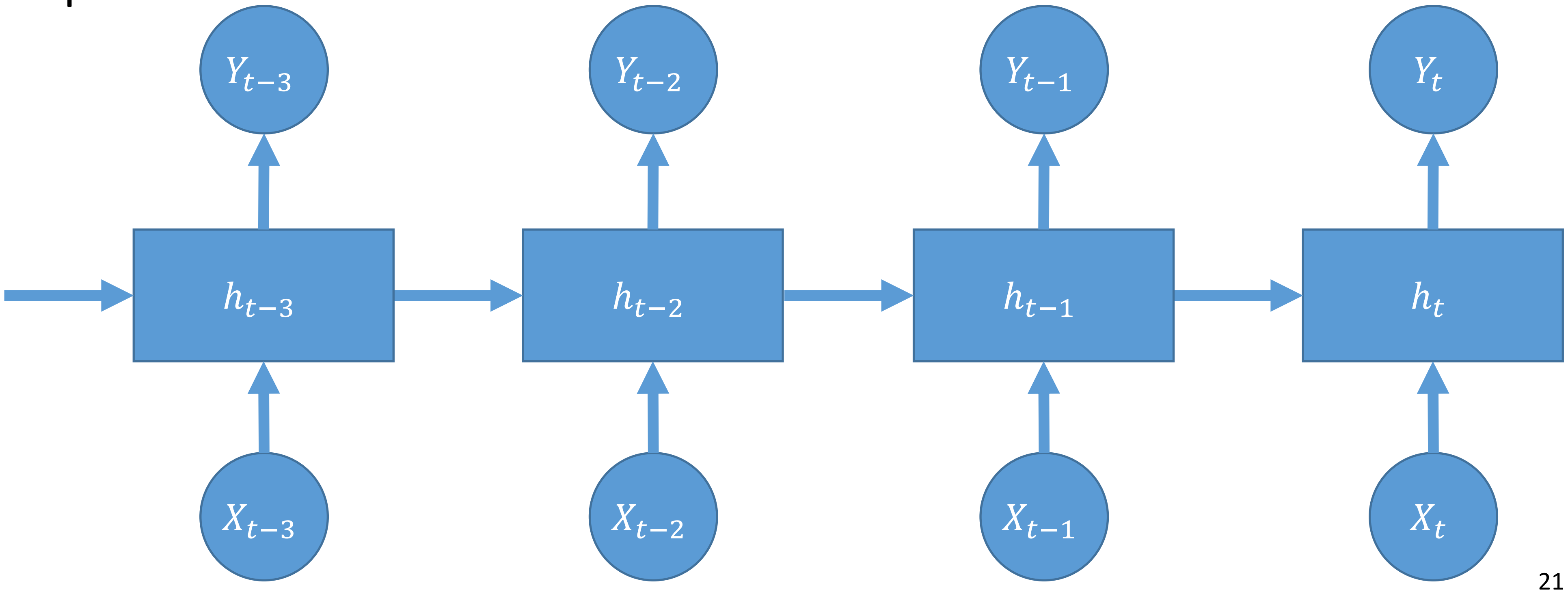
Elman Net (1990)

- M is memory: store the previous state
- Only the weight from M to MLP is learned
- M is approximated as independent 1-step history nets
  - Backrop only back 1 step
  - Can't backprop to the beginning



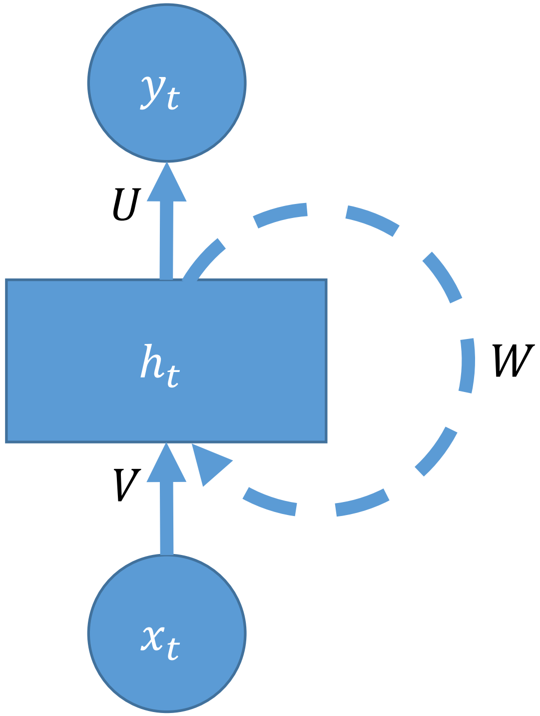
State-Space Model

- Fix number of input
- Share parameters of MLP at each step
  - Memory is embedded into state, h
- MLP arbitrarily complex



Single Recurrent Network Model

- Folded Model (in time)
- Parameters
  - x_t : input sequence at t
  - y_t : output (prediction)
  - h: state of network
  - V: weights of input
  - U: weights of outputs
  - W: shared weights
  - b_h: biases for hidden state
  - b_y: bias for output
  - f: typically tanh function
  - g: for classification softmax is typically used



$$h_t = f(Vx_t + Wh_{t-1} + b_h)$$
$$y_t = g\left(Uh_t + b_y\right)$$

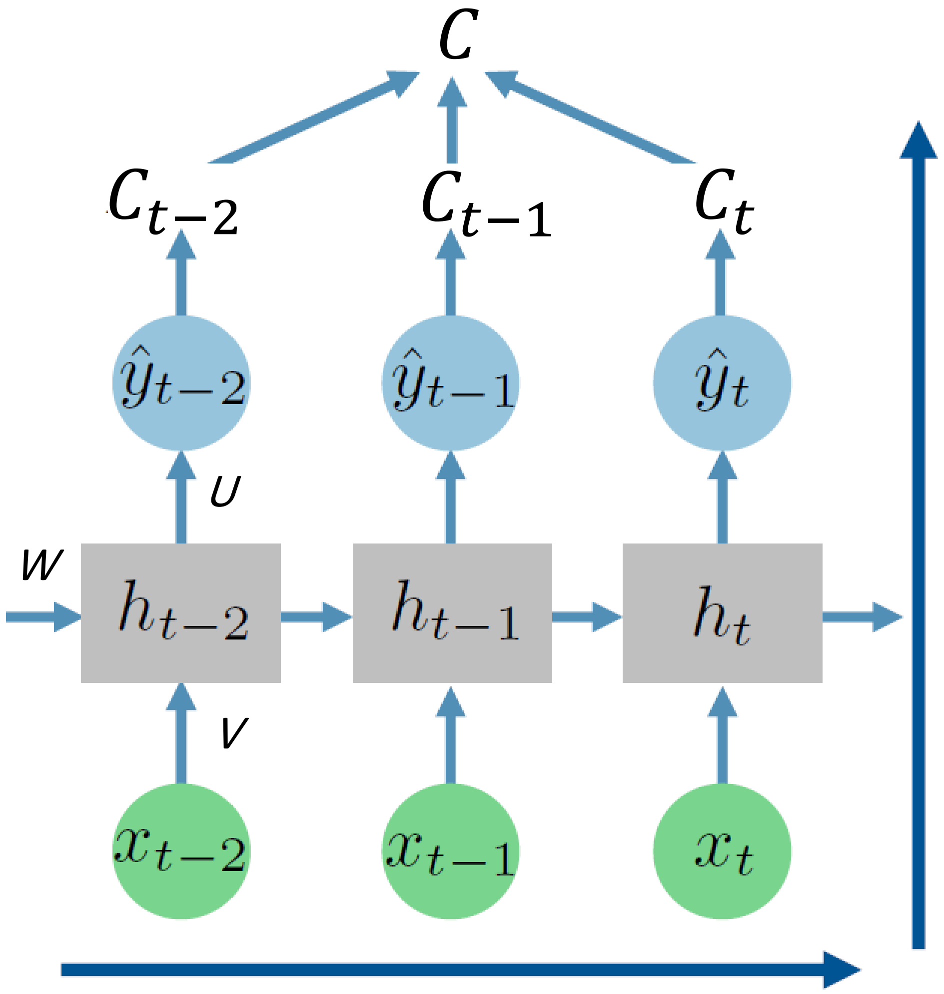
Training RNN

- Random weight initialization
- Feedforward (through time/sequence) to generate prediction, $\hat{y}_t$
- Compute cost function, C_t, based on actual y_t
- Total cost:

$$C = \sum_{i}^{t} C_i(y_i, \hat{y}_i)$$

- Use backprop to compute:

$$\frac{\partial C}{\partial U}, \frac{\partial C}{\partial V}, \frac{\partial C}{\partial W}, \frac{\partial C}{\partial b_y}, \frac{\partial C}{\partial b_h}$$



Backprop Through Time (BPTT)

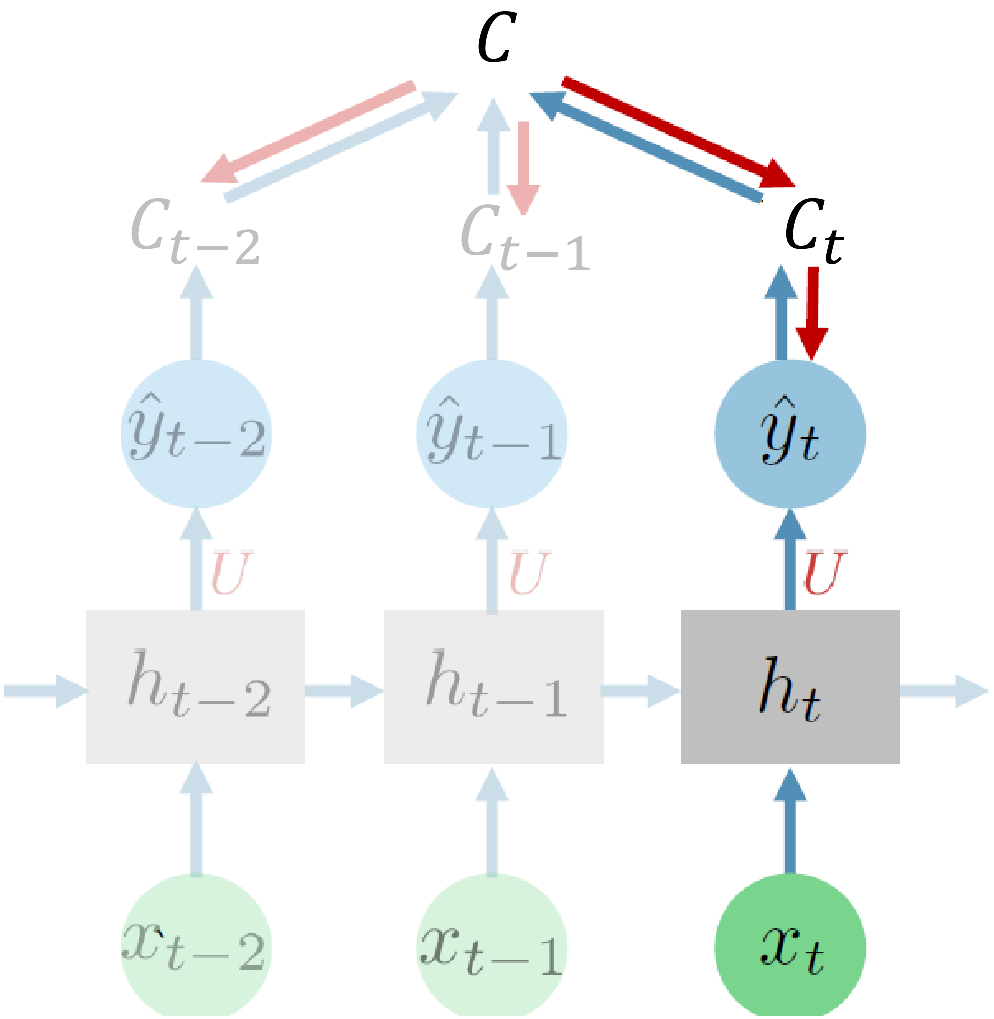
- Output function:

$$\hat{y}_t = f\left(Uh_t + b_y\right)$$

- Shared weights:

$$\frac{\partial C}{\partial U} = \sum_{i}^{t} \frac{\partial C_i}{\partial U} = \sum_{i}^{t} \frac{\partial C_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial U}$$

- For classification $\frac{\partial C_i}{\partial \hat{y}_i}$ based on the softmax



$\frac{\partial C}{\partial W}$

$$h_t = f(Vx_t + Wh_{t-1} + b_h)$$

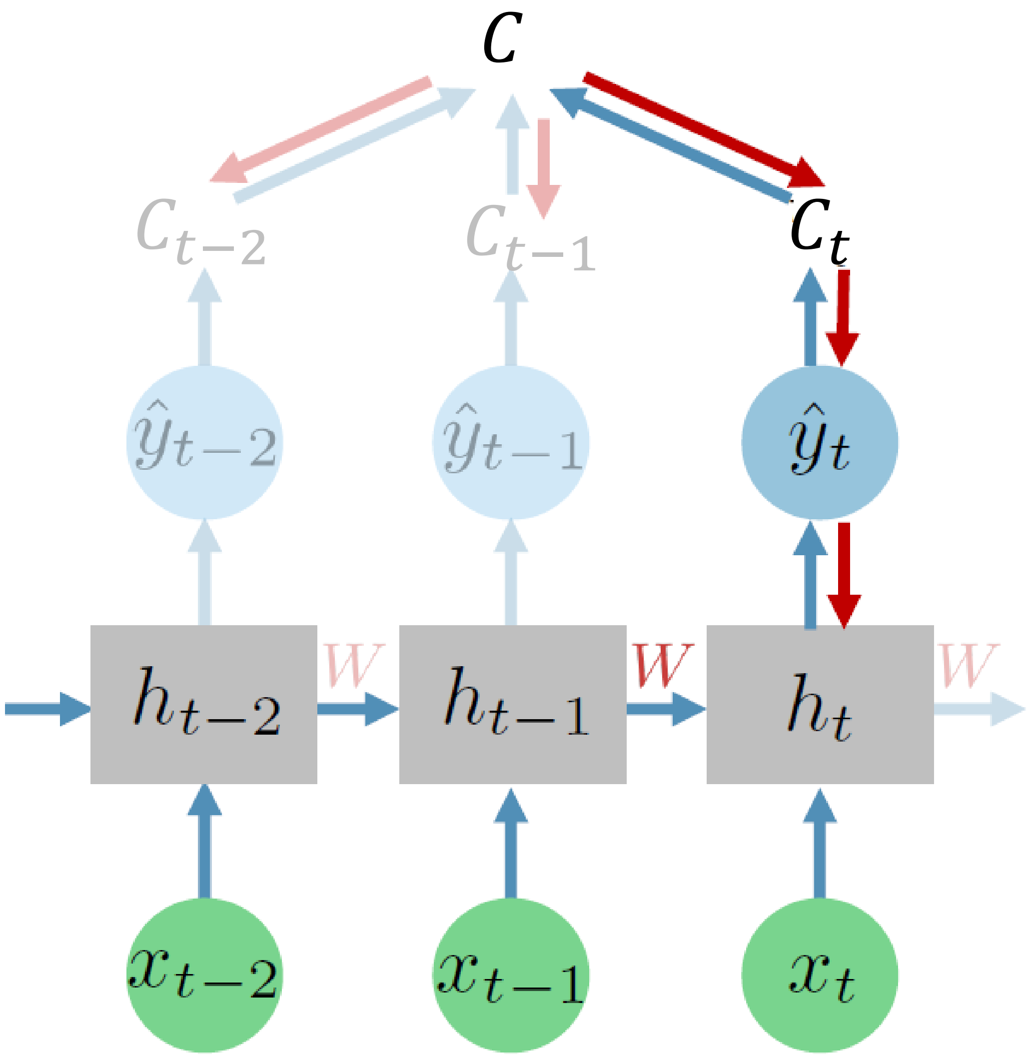
- h_{t-1}dependes on W

$$\frac{\partial C_i}{\partial W} = \frac{\partial C_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial h_i}\left(\frac{\partial h_i}{\partial W} + \frac{\partial h_i}{\partial h_{i-1}}\frac{\partial h_{i-1}}{\partial W} + \cdots + \frac{\partial h_1}{\partial h_0}\frac{\partial h_0}{\partial W}\right)$$

$$= \frac{\partial C_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial h_i}\left(\frac{\partial h_i}{\partial W} + \sum_{j=0}^{i-1}\frac{\partial h_{j+1}}{\partial h_j}\frac{\partial h_j}{\partial W}\right)$$

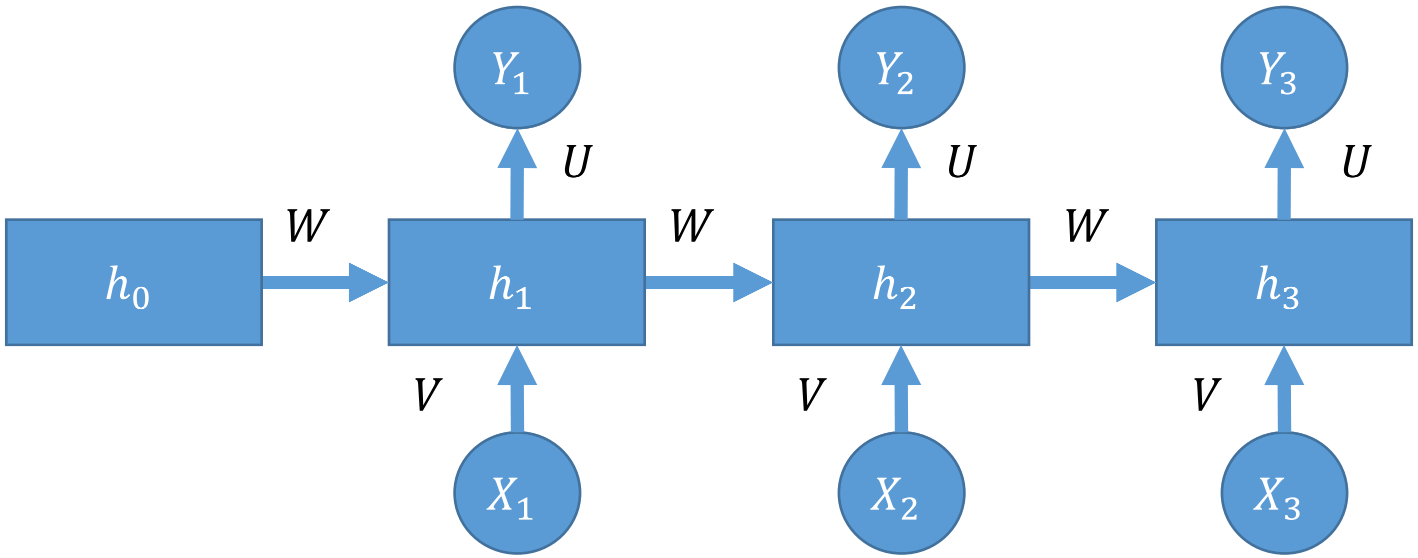
$$= \frac{\partial C_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial h_i}\left(\frac{\partial h_i}{\partial W} + \sum_{j=0}^{i-1}\left(\prod_{k=j+1}^{i-1}\frac{\partial h_{k+1}}{\partial h_k}\right)\frac{\partial h_j}{\partial W}\right)$$

$$\frac{\partial C}{\partial W} = \sum_{0}^{T}\frac{\partial C_i}{\partial W}$$



BPTT Example

$$h_1 = f(Wh_0 + Vx_1 + b_h) = f(Z_1)$$
$$h_2 = f(Wh_1 + Vx_2 + b_h) = f(Wf(Z_1) + Vx_2 + b_h) = f(Z_2)$$
$$= (W^2h_0 + WVx_1 + Wb_h) + Vx_2 + b_h$$
$$h_3 = f(Wh_2 + Vx_3 + b_h) = f(Wf(Z_2) + Vx_3 + b_h) = f(Z_3)$$



h_n

$$
\begin{aligned}
h_1 &= f(Wh_0 + Vx_1 + b_h) \\
h_2 &= f(Wh_1 + Vx_2 + b_h) = (W^2h_0 + WVx_1 + Wb_h) + Vx_2 + b_h \\
h_3 &= Wh_2 + Vx_3 + b_h = W\big((W^2h_0 + WVx_1 + Wb_h) + Vx_2 + b_h\big) + Vx_3 + b_h \\
&= W^3h_0 + W^2Vx_1 + W^2b_h + WVx_2 + Wb_h + Vx_3 + b_h \\
&= W^3h_0 + W^2Vx_1 + WVx_2 + Vx_3 + W^2b_h + Wb_h + b_h
\end{aligned}
$$

Let $x_0 = h_0$:

$$h_3 = W^3x_0 + W^2Vx_1 + WVx_2 + Vx_3 + W^2b_h + Wb_h + b_h$$

$$h_n = W^n\, x_0 + \sum_{t=1}^{n} W^{n-t}\, Vx_t + W^{n-t}\, b_h$$

$\frac{\partial h_3}{\partial W}$

$$\frac{\partial h_3}{\partial W} = \frac{\partial f(Z_3)}{\partial Z_3}\frac{\partial Z_3}{\partial g(Z_2)}\frac{\partial f(Z_2)}{\partial Z_2}\frac{\partial Z_2}{\partial g(Z_1)}\frac{\partial f(Z_1)}{\partial Z_1}\frac{\partial Z_1}{\partial h_0}$$

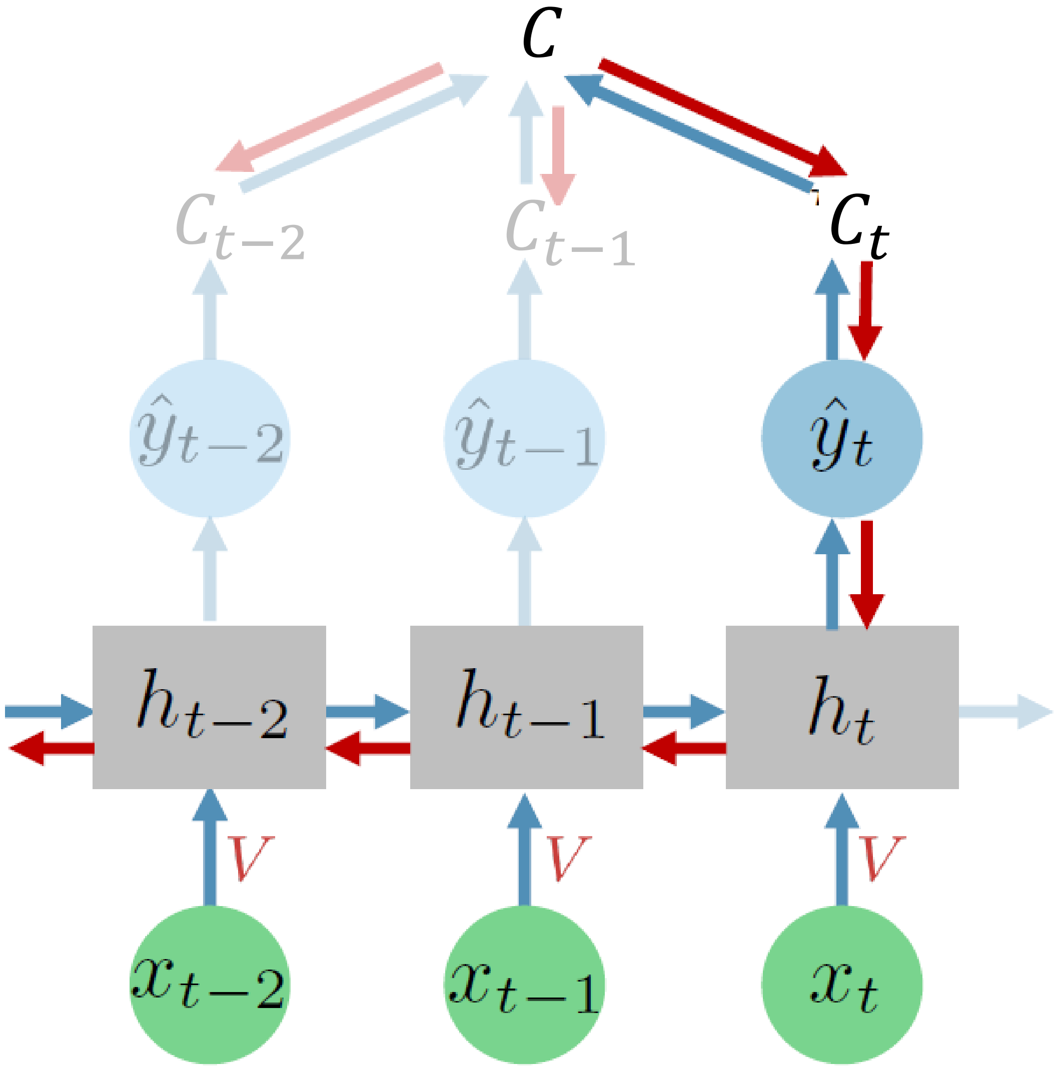
$$\frac{\partial h_3}{\partial W} = \frac{\partial h_3}{\partial Z_3}\frac{\partial Z_3}{\partial h_2}\frac{\partial h_2}{\partial Z_2}\frac{\partial Z_2}{\partial h_1}\frac{\partial h_1}{\partial Z_1}\frac{\partial Z_1}{\partial h_0} = f'(Z_3)Wf'(Z_2)Wf'(Z_1)W$$

$\frac{\partial C}{\partial V}$

$$h_t = f(Vx_t + Wh_{t-1} + b_h)$$

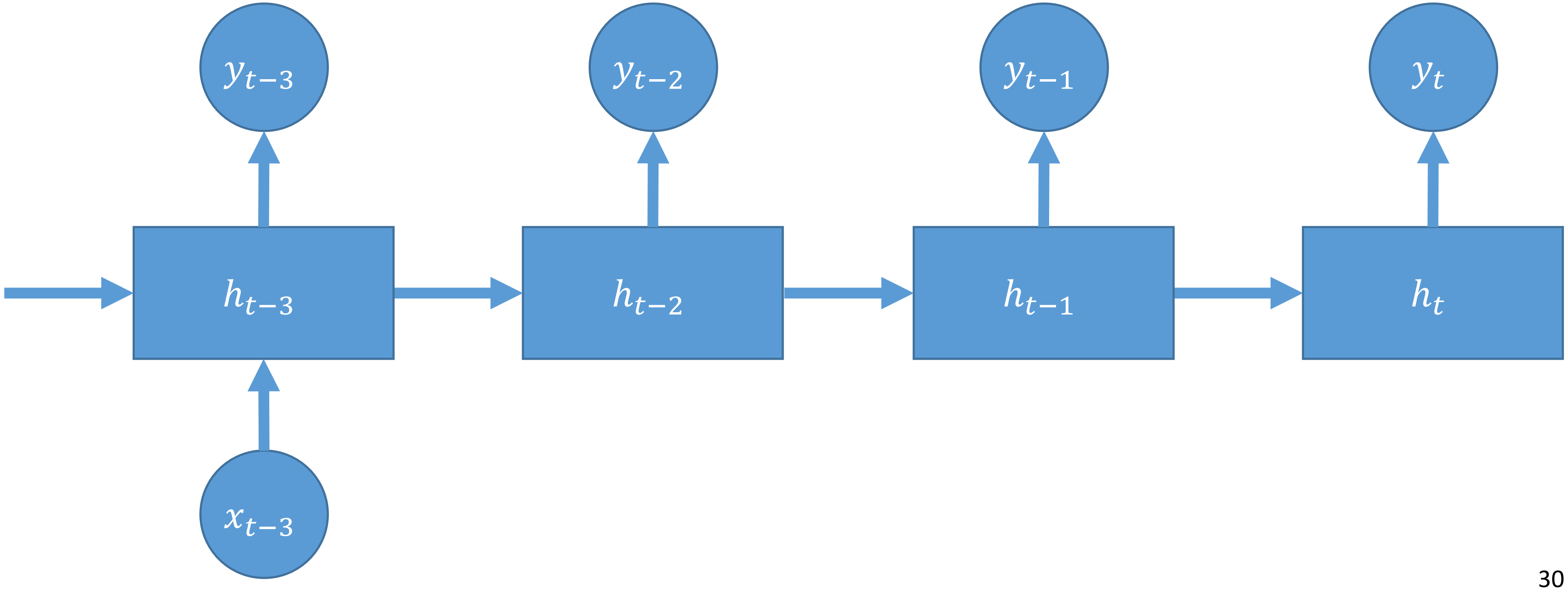
- Similarly, h is dependent on V

$$\frac{\partial C_i}{\partial V} = \frac{\partial C_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial h_i}\left(\frac{\partial h_i}{\partial V} + \sum_{j=0}^{i-1}\left(\prod_{k=j+1}^{i-1}\frac{\partial h_{k+1}}{\partial h_k}\right)\frac{\partial h_j}{\partial V}\right)$$



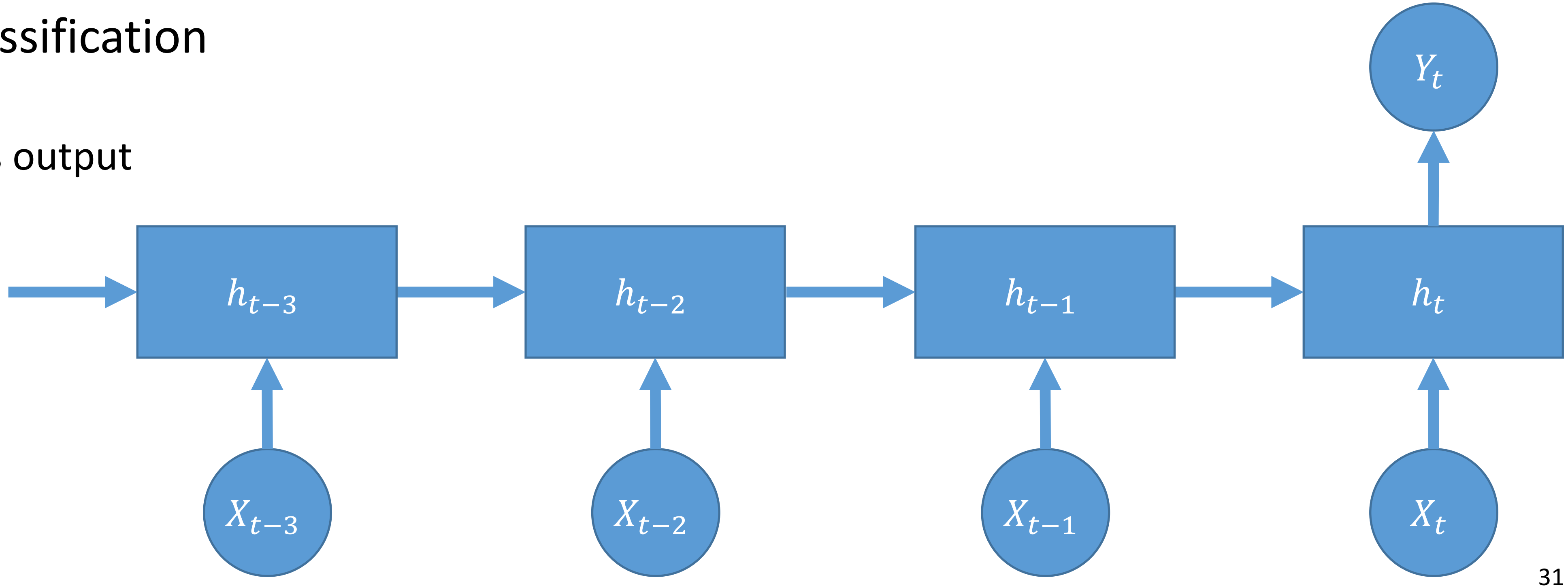
RNN One input – Many Outputs

- Sequence generation
- Example: image caption
  - Image as input
  - Text as output



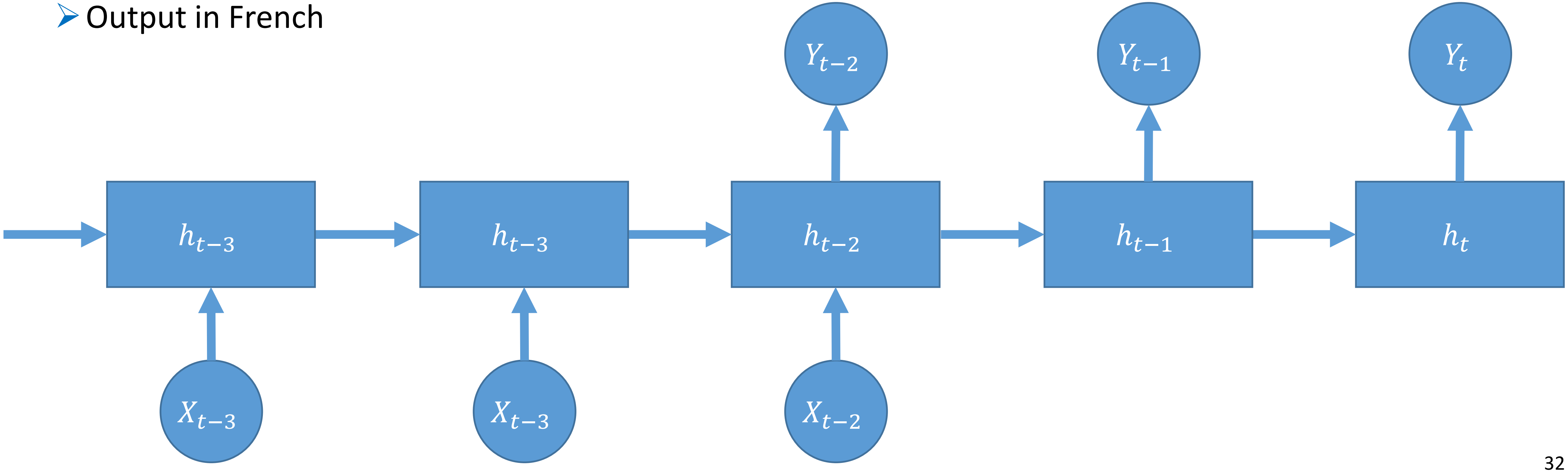
RNN Many inputs – One output

- Sequence based classification/prediction
- Example: speech recognition
  - Audio clips as input
  - Word as output
- Example: text classification
  - Words as input
  - Subject/topic as output



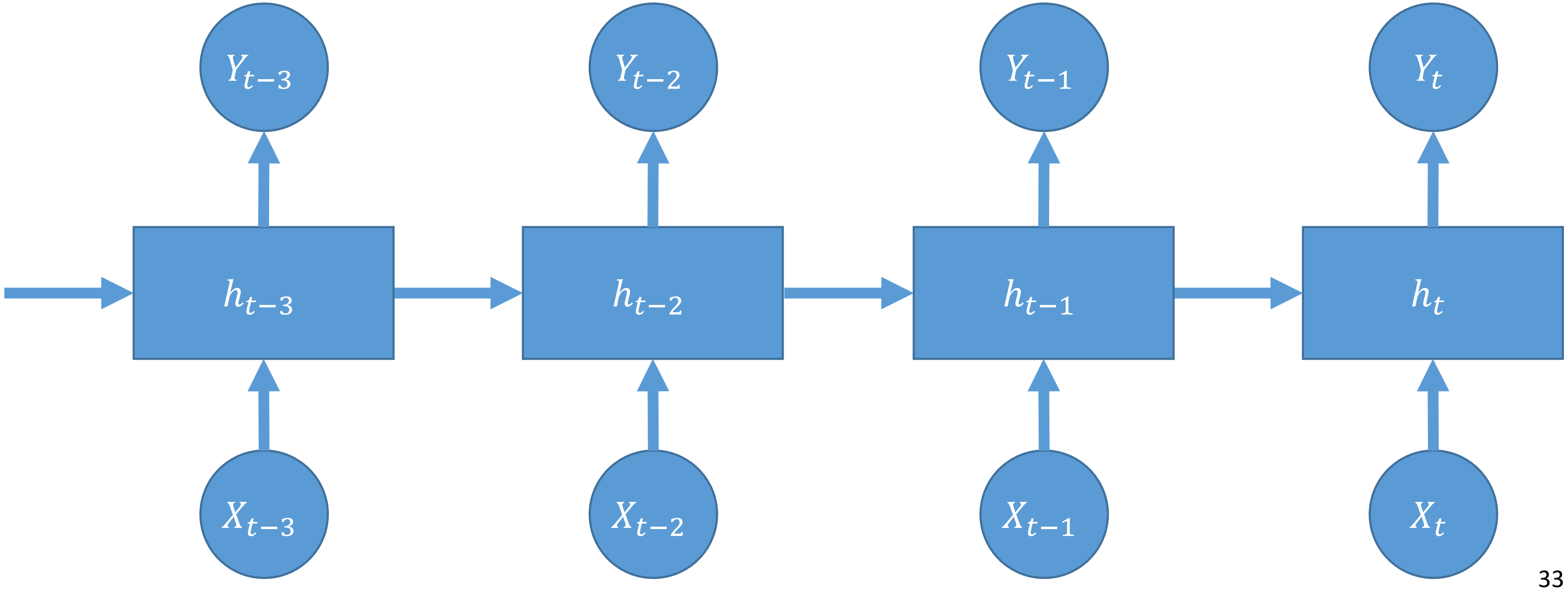
RNN Many inputs – Many outputs

- Delayed output
  - Encoder-Decoder design
- Example: machine translation
  - Input in English
  - Output in French

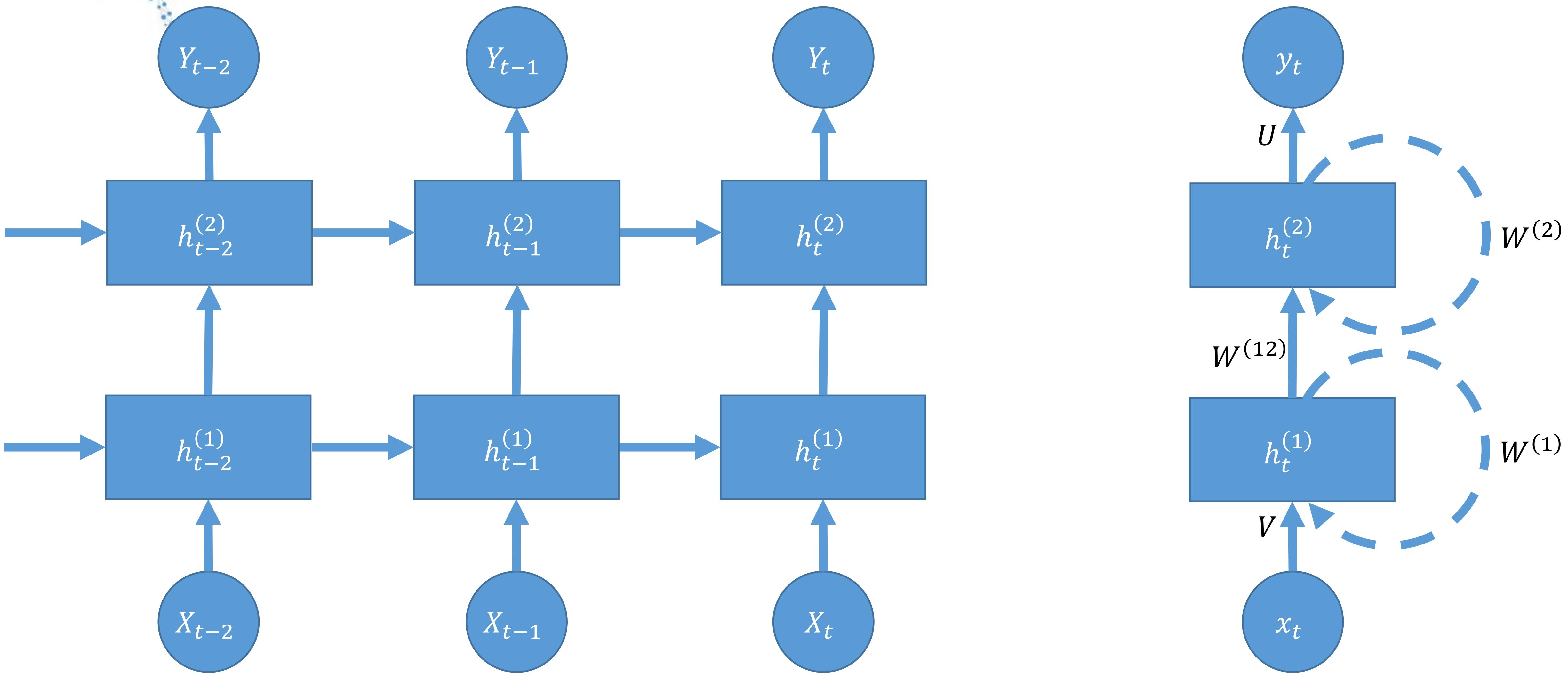


RNN Many inputs – Many outputs

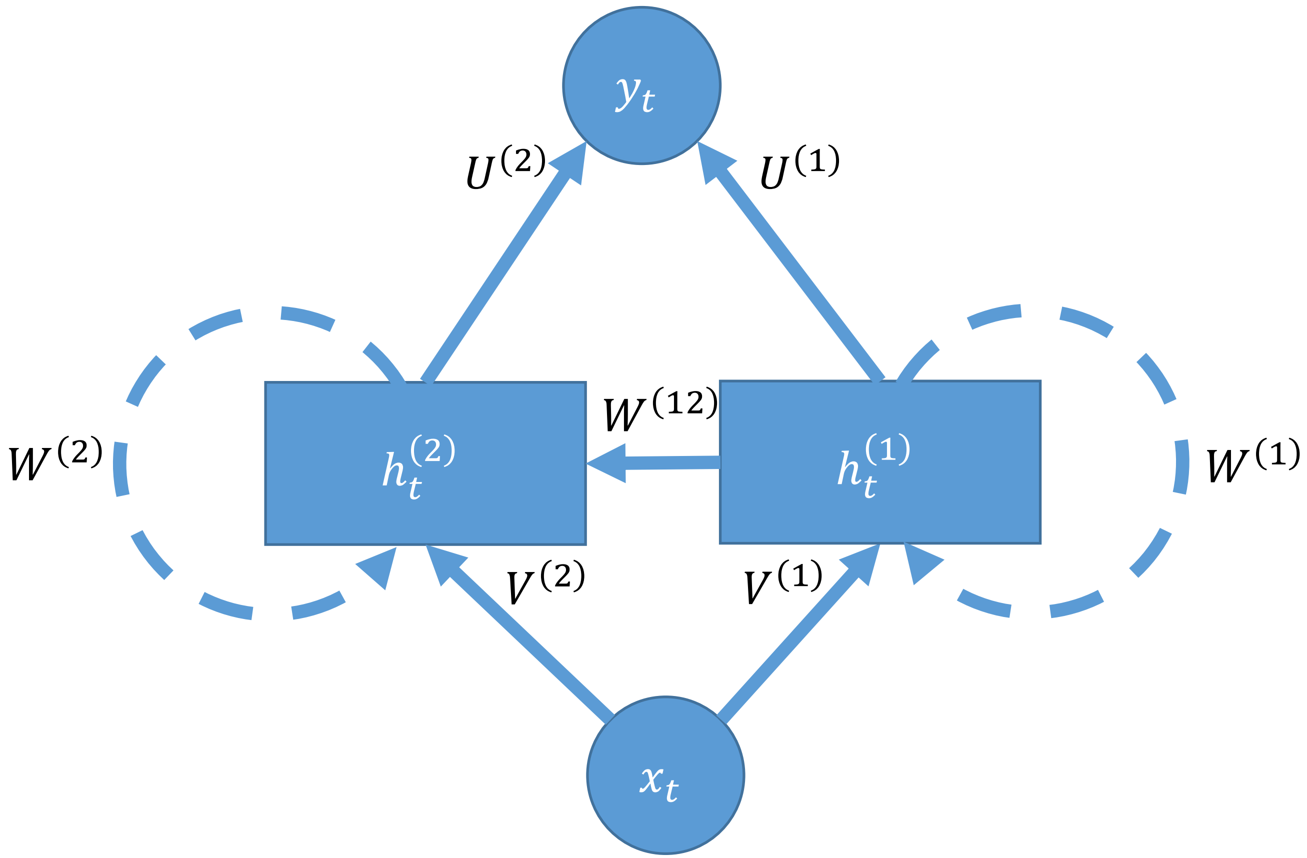
- Synched output
- Example: stock prediction
  - Value of stock is fed in at each iteration
  - Predicted value of stock is outputted at each sequence



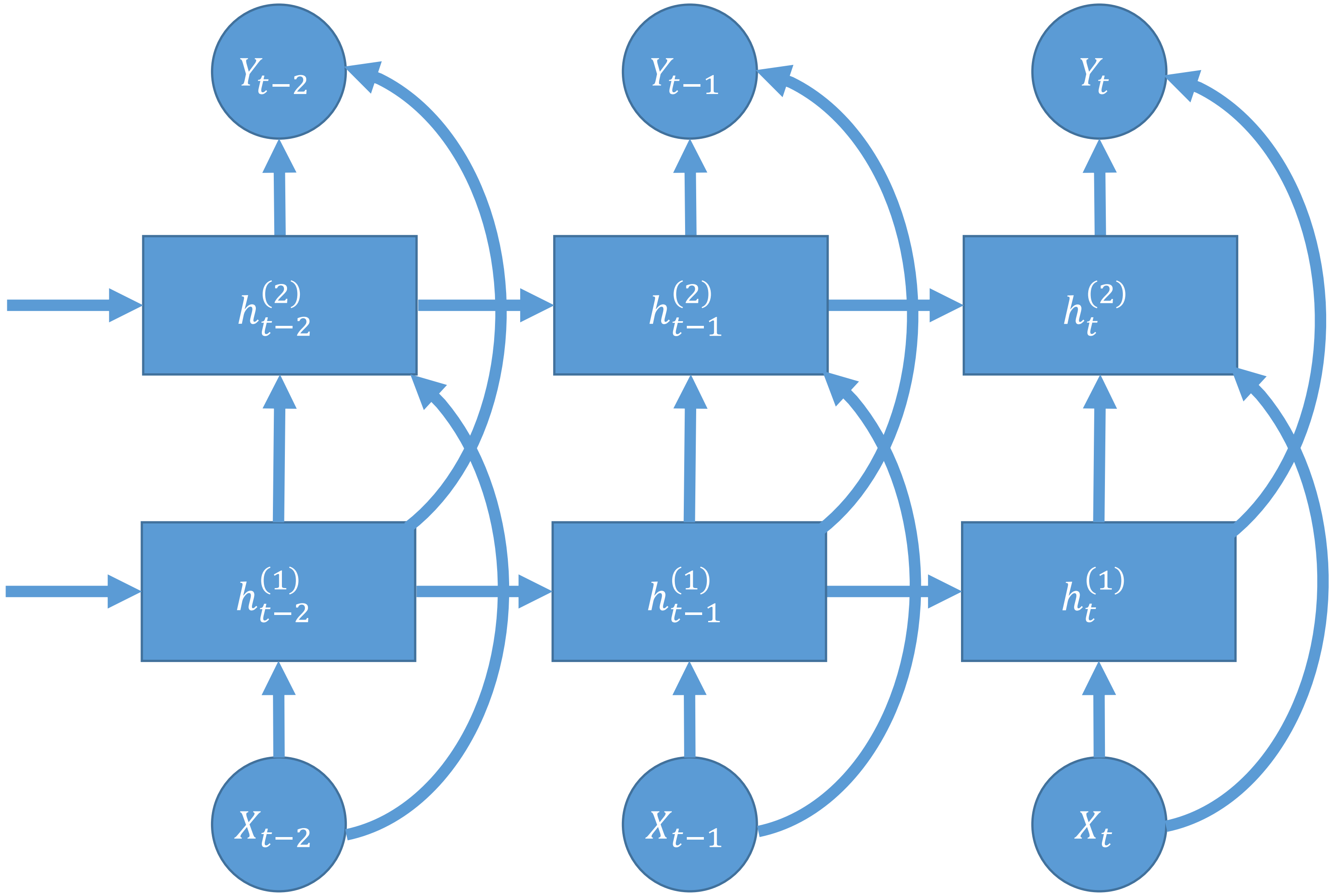
Sequential Recursion



Parallel Recursion

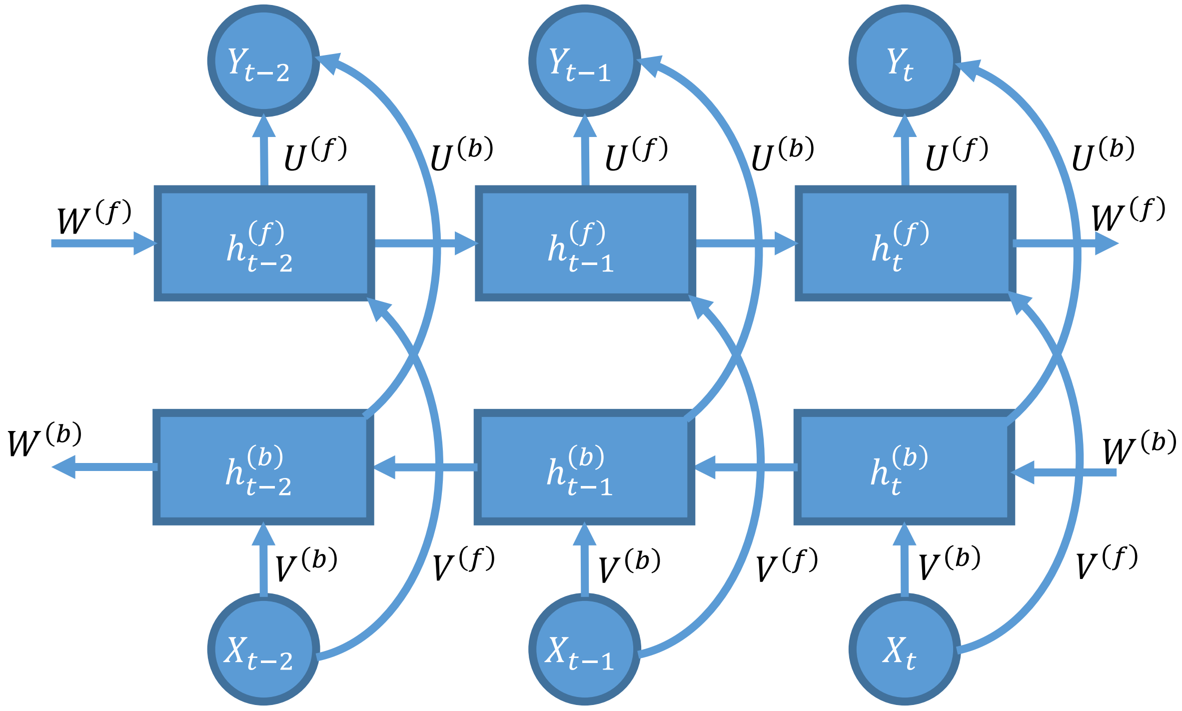


Unfolded Parallel Recursion



BRNN (1997)

- Bi-Directional RNN: Parallel backward and forward recursions
  - Schuster and Paliwal
  - especially useful when the context of the input
- Forward layer
  - Predict future from past
  - Processes data from t=0 to T
- Backward layer
  - Deduces past from future
  - Processes data from t=T to 0
- Applications:
  - Speech Recognition
  - Translation
  - Handwritten Recognition
  - Protein Structure Prediction

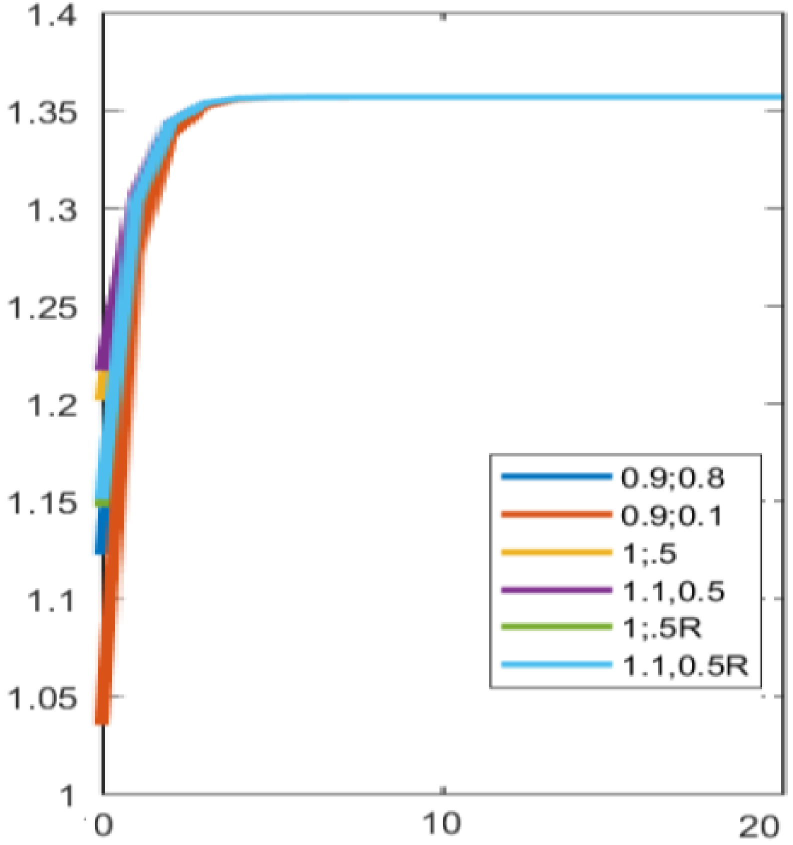


Stability Analysis

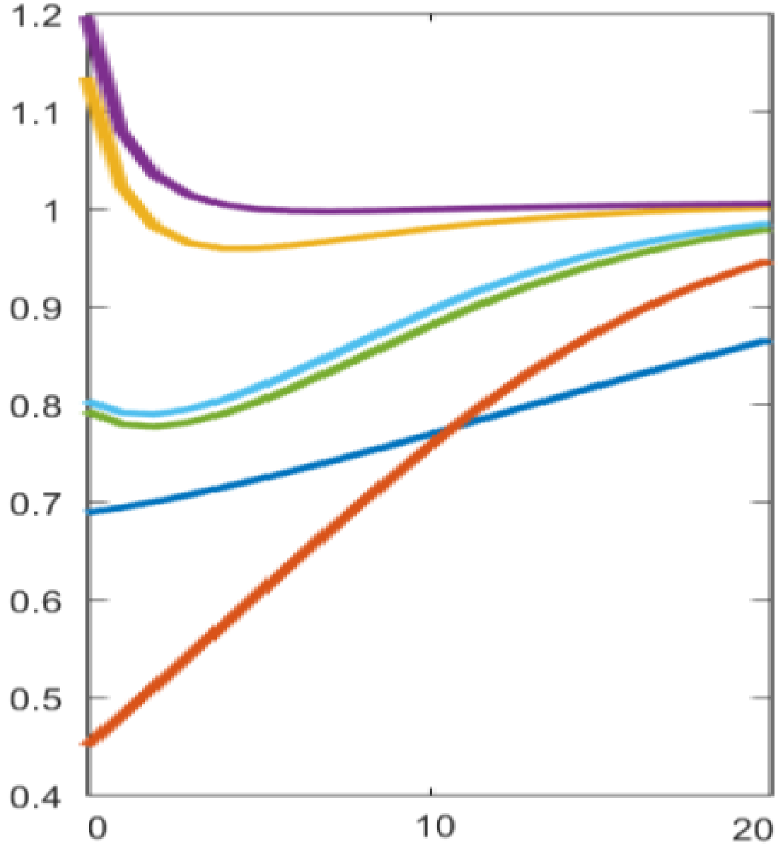
- Problem: recursion and output saturation
  - For sigmoid/Tanh: saturation
  - For ReLU: explosion
- Output stability, single tap: $y_t = g(Uh_t)$
  - if h_t is bounded (stable) then output is stable
  - Ignoring bias
- Hidden layer stability, single tap: $h_t = f(Vx_t + Wh_{t-1})$
  - x_t bounded (naturally)
  - Process depends on recursion

Stability of AF for 1 initialization

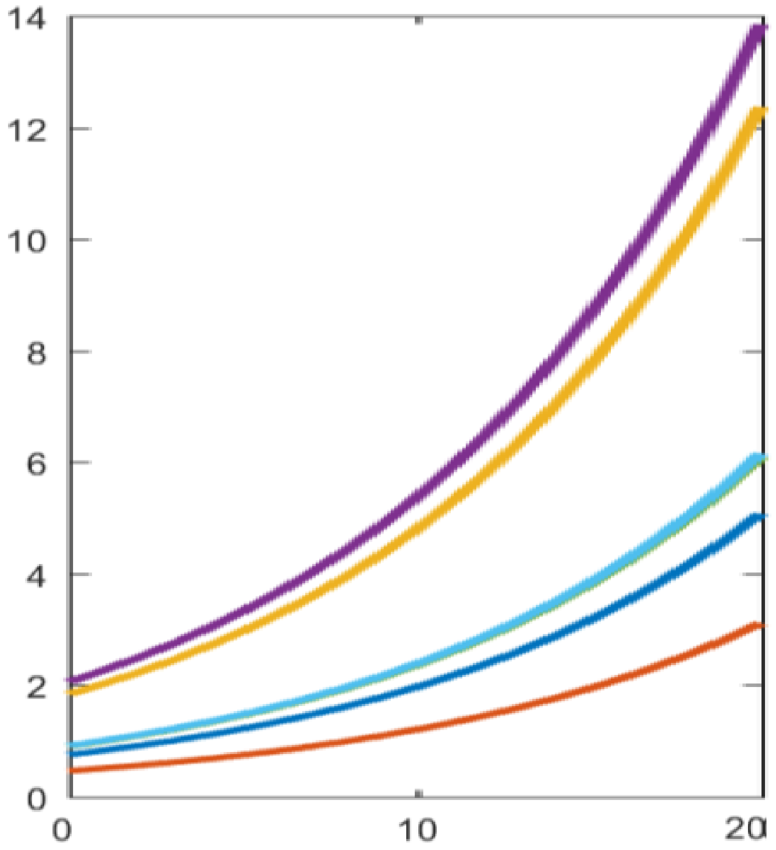
Sigmoid



Tanh



ReLU

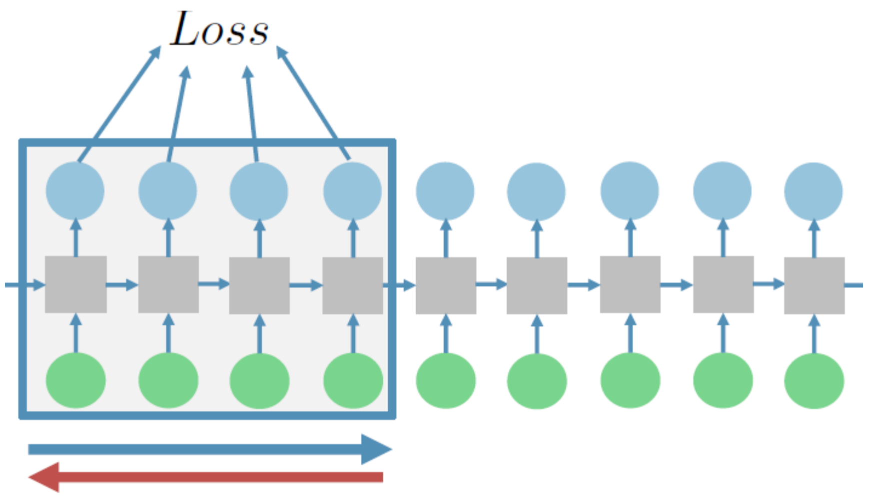


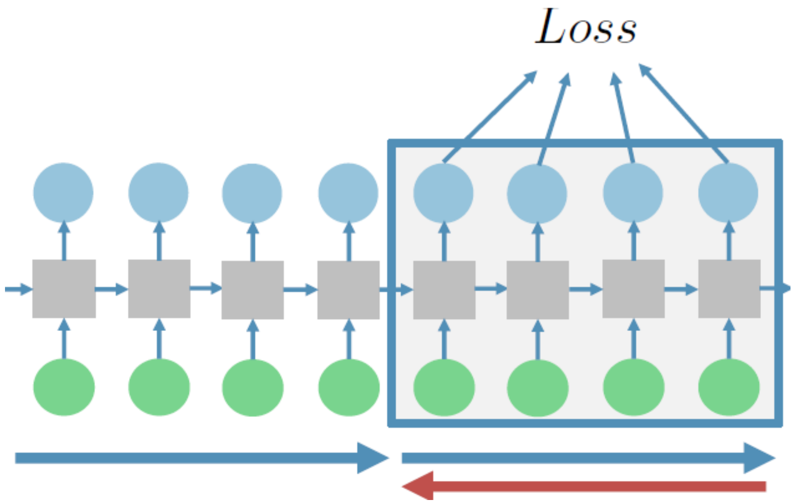
Stability and Memory

- Weight of recursion can cause instability
- Bipolar functions hold memory
- Low stability => memory is low
- Exponential instability => memory is forgotten exponentially

Gradient Stability

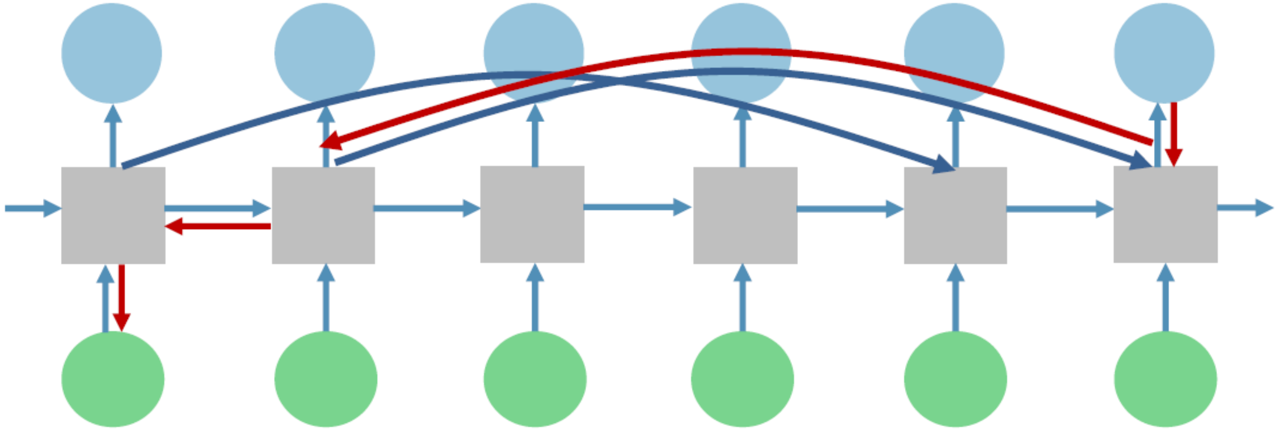
- Deep recursions synonymous with deep networks
  - Gradients will explode or vanish
- Exploding Solutions:
  - Clipping grads: if $||g|| > Thresh \Rightarrow g = \frac{Thresh}{||g||} g$
  - Batch Process Loss: Forward pass and backward pass chunks



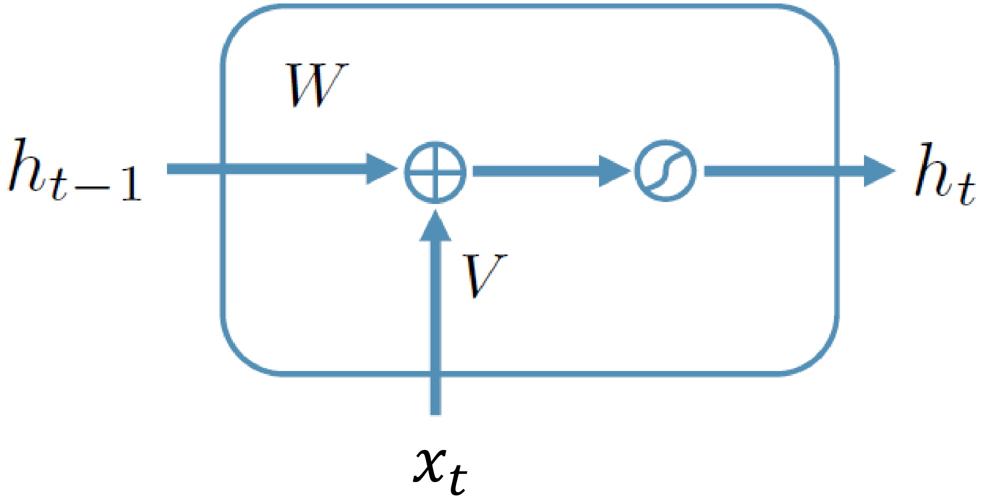


Vanishing Grad Solutions

- Vanishing grads $\Rightarrow$ loss of memory
- ReLU activation
  - Sigmoid/tanh: saturate and cause gradient to vanish
- Initialization solutions
  - Choose orthogonal W matrix: $W^T = W^{-1}$
    - $w_{ij}^{-1} = w_{ji}$
  - Orthogonal W doesn't vanish (or explode)
- Skip connections
  - BP across skip connections
  - Vanishes slower than other connections



Hidden State as a Cell

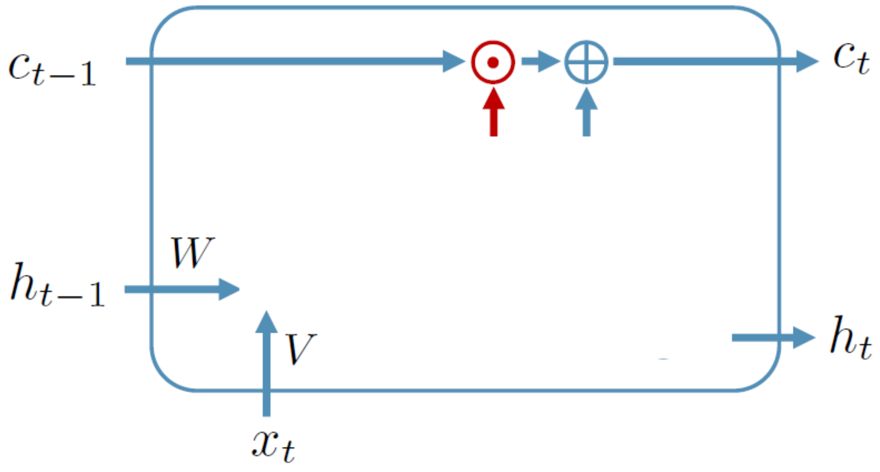


$$h_t = f(Wh_{t-1} + Vx_t + b_h)$$

- Nonlinearity causes vanishing gradient in the backward-pass
- Memory and gradient are tied together
  - Vanishing memory ⇒ lost memory

LSTM: Memory as a Separate Path

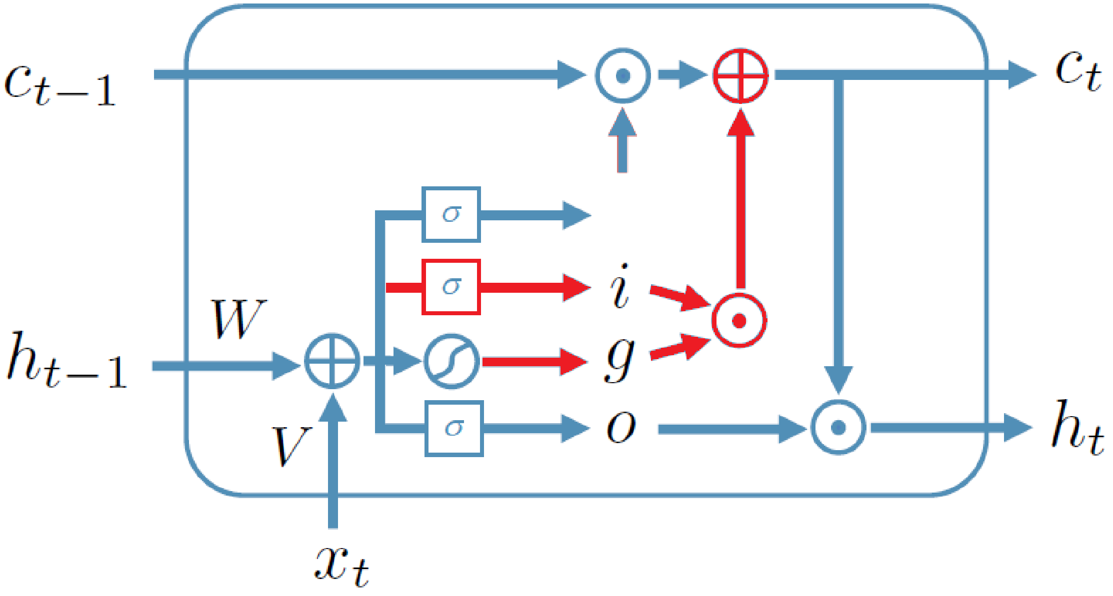
- Long Short Term Memory
- Idea1 : Separate path for cell memory
  - Allows easier derivative ($\partial c_t / \partial c_{t-1} = 1$)
  - No nonlinearities in the path
- c_t is cell memory
  - Same dimension as h_t
- Idea 2: Update information from the cell into memory
  - Input & previous states contains information
  - Updates can be added $\oplus$
  - Updates can be scaled $\odot$



LSTM: Memory Updates

- Idea 2: Update memory with information from the cell into memory
  - Based on input and previous state
- Updates can be added $\oplus$
  - Percentage of x_t & h_{t-1}
- Updates can be multiplicative $\odot$
  - Cause cell to forget
  - Use sigmoid function, $\sigma(\quad)$
  - Need a percentage for scaling purposes

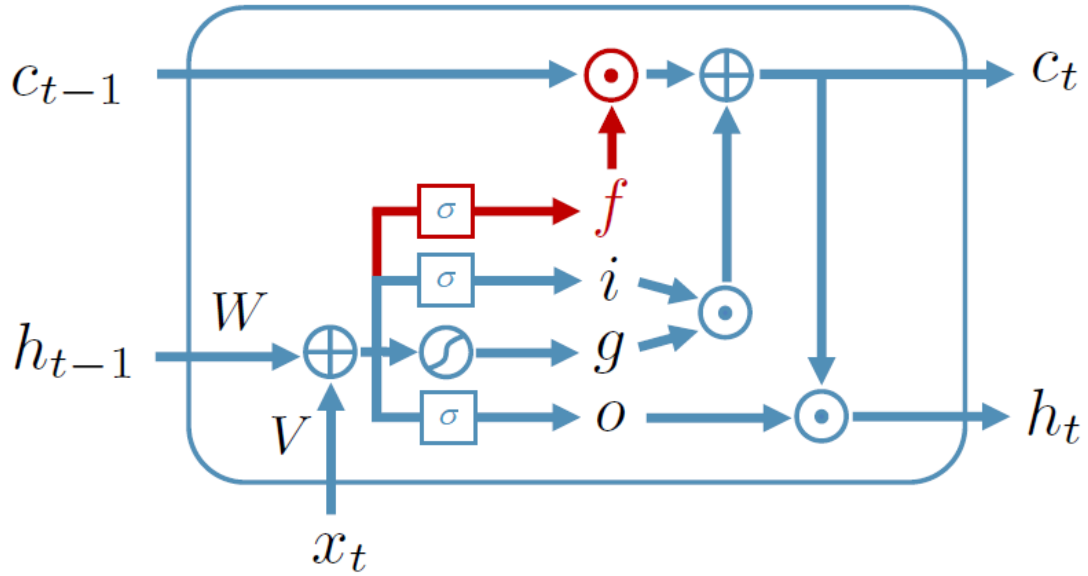
LSTM: Update Memory with new Input



$$i = \sigma(Wh_{t-1} + Vx_t + b_i)$$
$$g = \tanh\left(Wh_{t-1} + Vx_t + b_g\right)$$
$$c_t = {\color{red} i \odot g} + f \odot c_{t-1}$$

- Input: $i \in [0{:}1]$
  - Controls how much of past memory moves forward

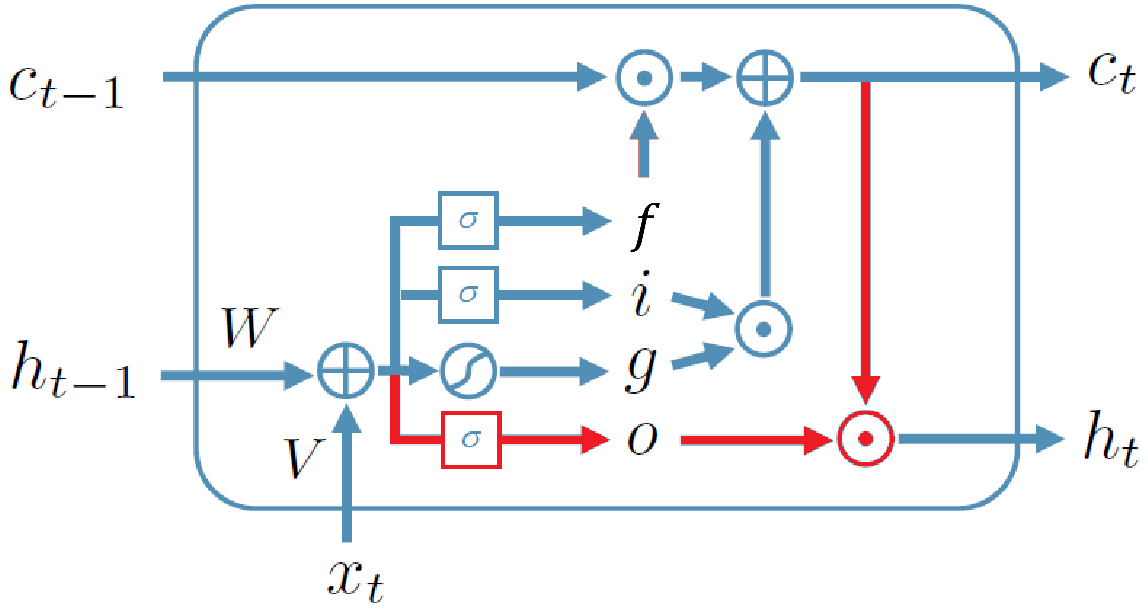
LSTM: Forget Past Memory



$$f = \sigma\left(W h_{t-1} + V x_t + b_f\right)$$
$$c_t = f \odot c_{t-1} + i \odot g$$

- Forget: $f \in [0{:}\,1]$
  - Controls how much of past memory moves forward

LSTM: Output



$$o = \sigma(W h_{t-1} + V x_t + b_o)$$
$$h_t = o \odot c_t$$

- Output: $o \in [0{:}\,1]$
  - Controls how much of new memory is encoded in new state

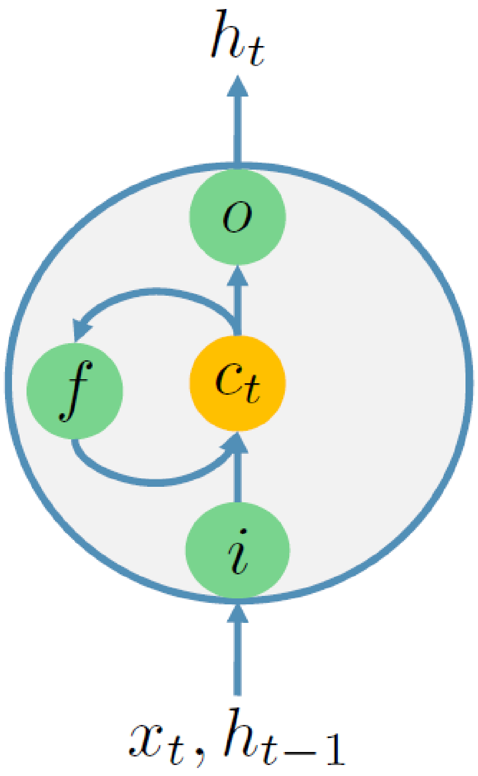
LSTM Cell Model

$$i = \sigma(Wh_{t-1} + Vx_t + b_i)$$

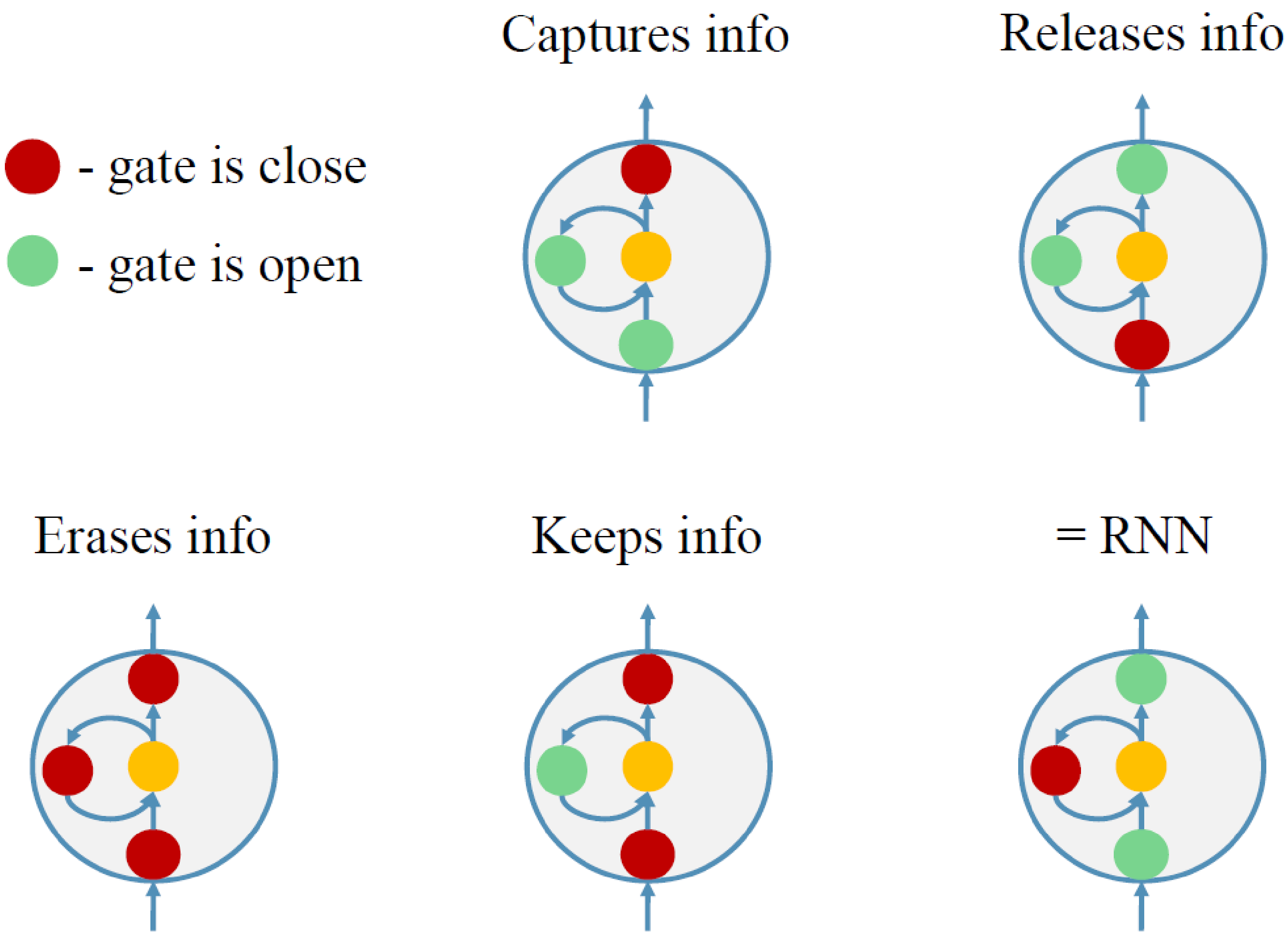
$$c_t = i \odot g + f \odot c_{t-1}$$

$$f = \sigma\left(Wh_{t-1} + Vx_t + b_f\right)$$

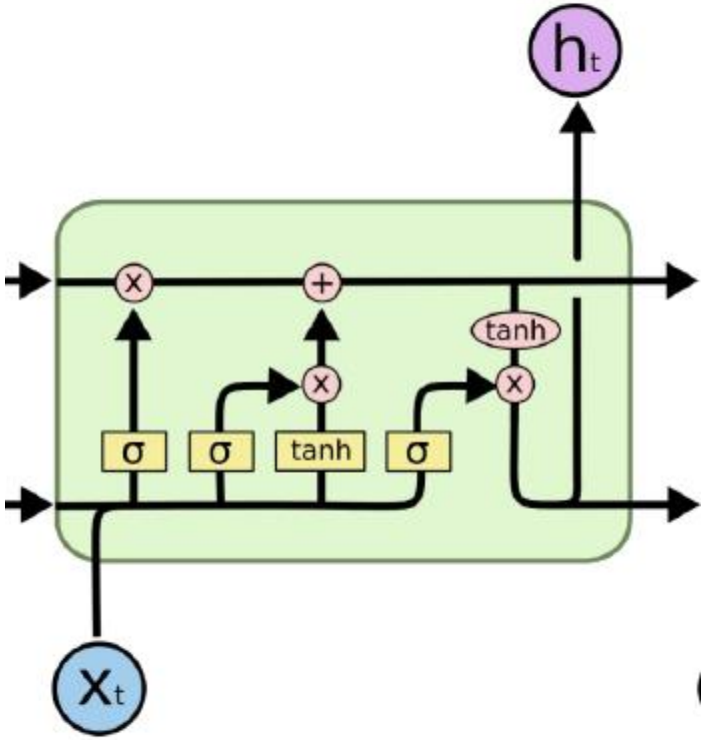
$$o = \sigma(Wh_{t-1} + Vx_t + b_o) \odot c_t$$



Extreme Conditions

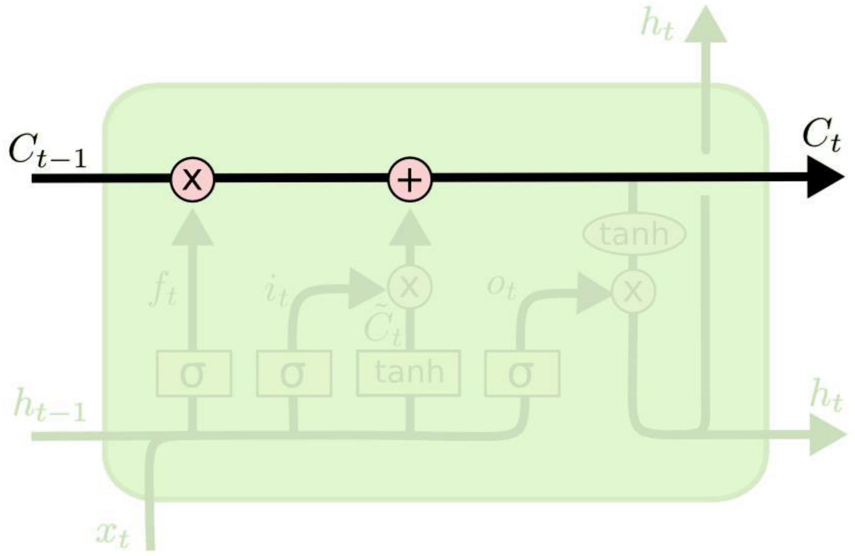


Alternative LSTM2

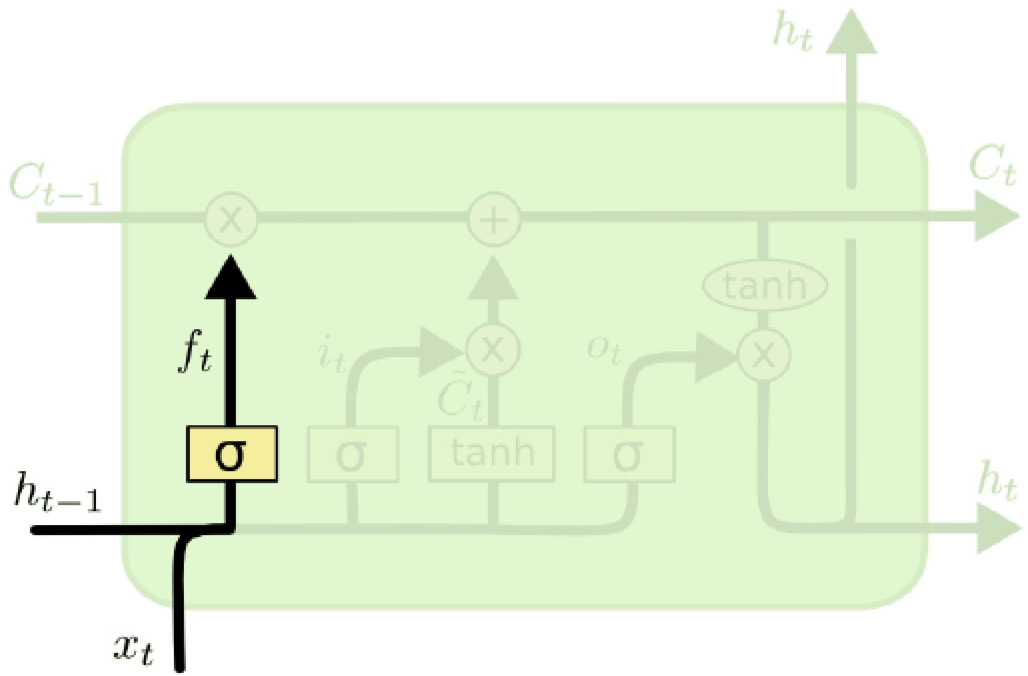


http://colah.github.io/posts/2015

LSTM2: Memory Path

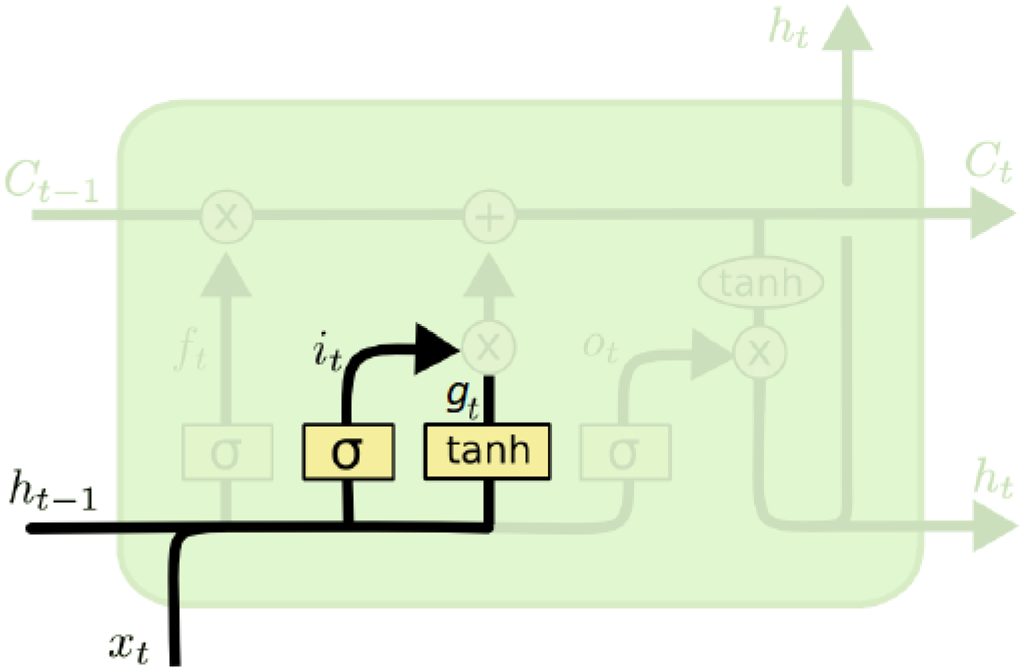


LSTM2: Forget Gate



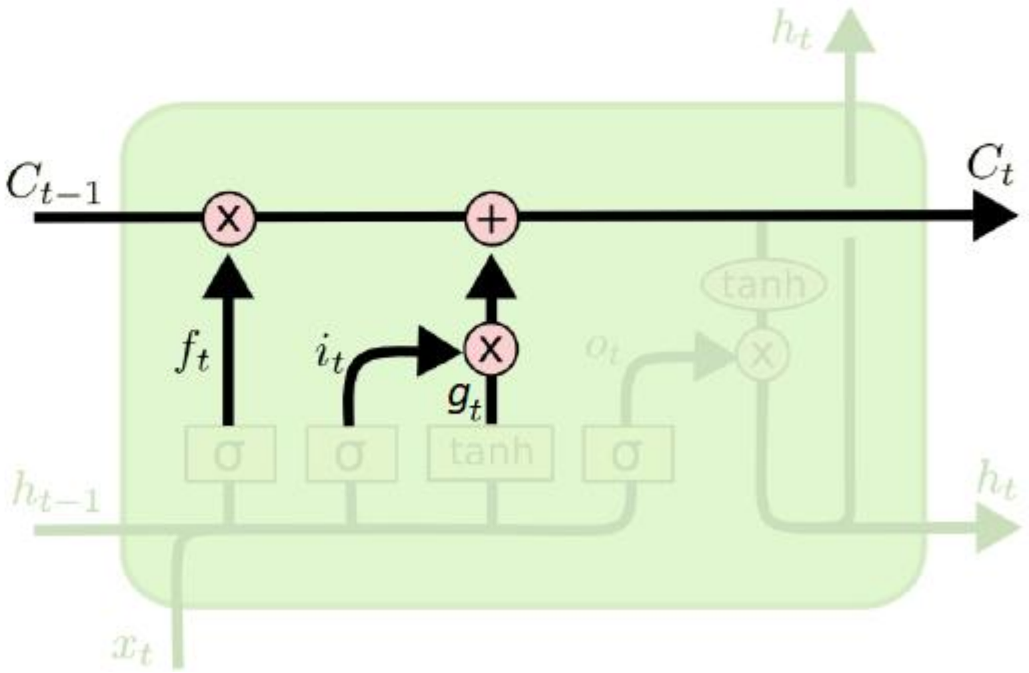
$$f_t = \sigma\left(W h_{t-1} + V x_t + b_f\right)$$

LSTM2: Input Gate



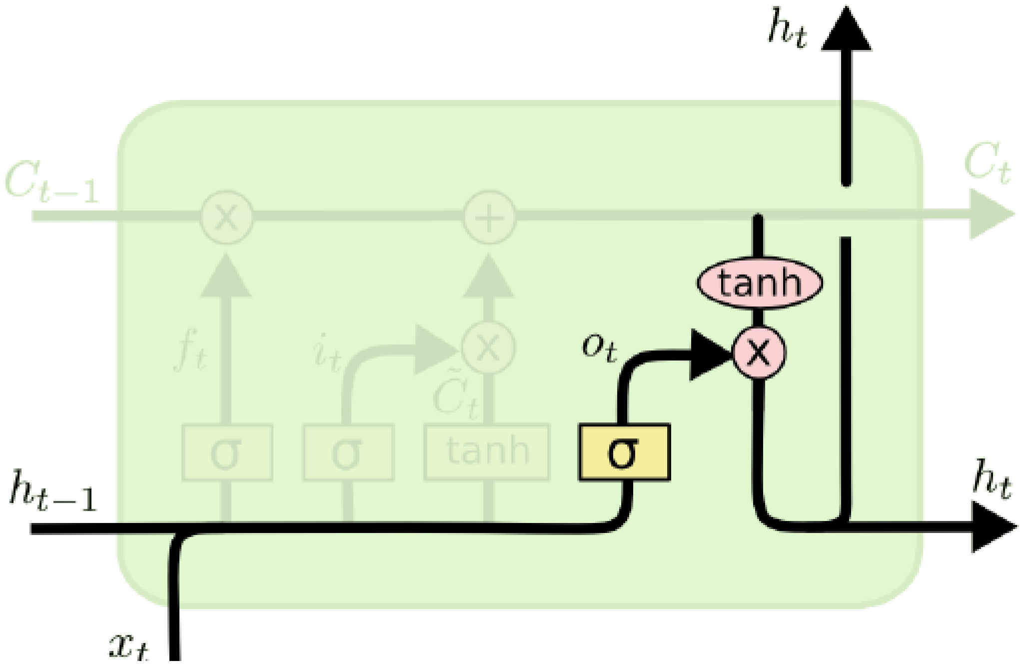
$$i_t = \sigma(W_i h_{t-1} + V_i x_t + b_i)$$
$$g_t = \tanh\left(W_g h_{t-1} + V_g x_t + b_g\right)$$

LSTM2: Update



$$c_t = f_t c_{t-1} + i_t g_t$$

LSTM2: Output Gate



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

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM2: Cell Model

$$i_t = \sigma(W_i h_{t-1} + V_i x_t + b_i)$$

$$f_t = \sigma\left(W_f h_{t-1} + V_f x_t + b_f\right)$$

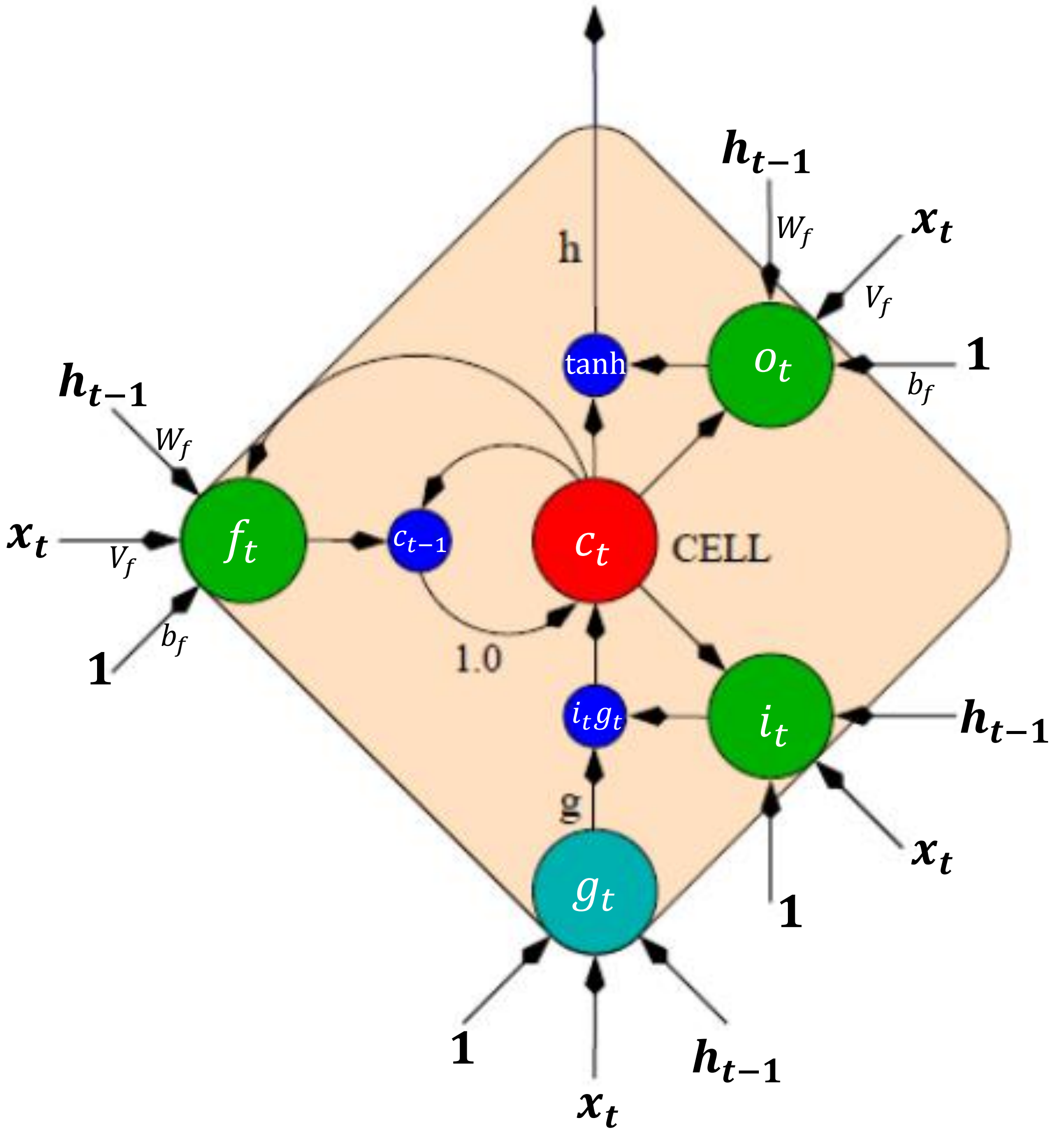
$$o_t = \sigma(W_h h_{t-1} + V_h x_t + b_h)$$

$$g_t = \tanh\left(W_g h_{t-1} + V_g x_t + b_g\right)$$

$$c_t = f_t c_{t-1} + i_t g_t$$

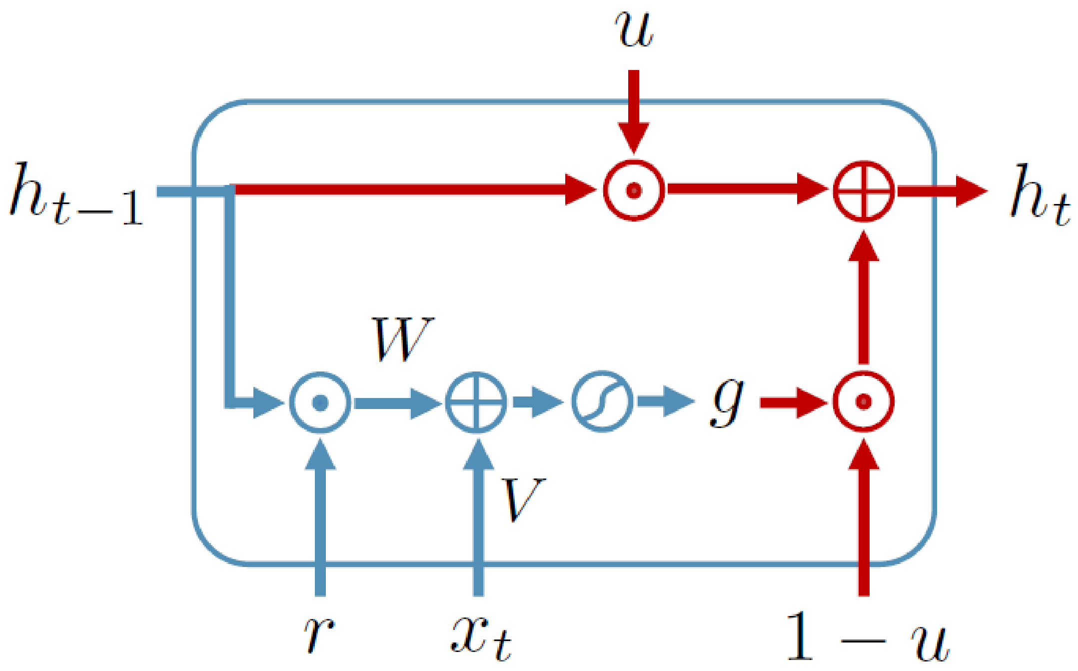
$$h_t = o_t \tanh(c_t)$$

$$y_t = \text{softmax}\,(U h_{t-1} + by)$$



GRU

- LSTM
  - Outputs: c, h
  - States: input, output, forget
  - Additional parameters: g, i, o, f
- Gated Recurrent Units
  - Output: h
  - States: output, forget
  - Additional Parameters: r, u



$$u = \sigma(Wh_{t-1} + Vx_t + b_u)$$

$$r = \sigma(Wh_{t-1} + Vx_t + b_r)$$

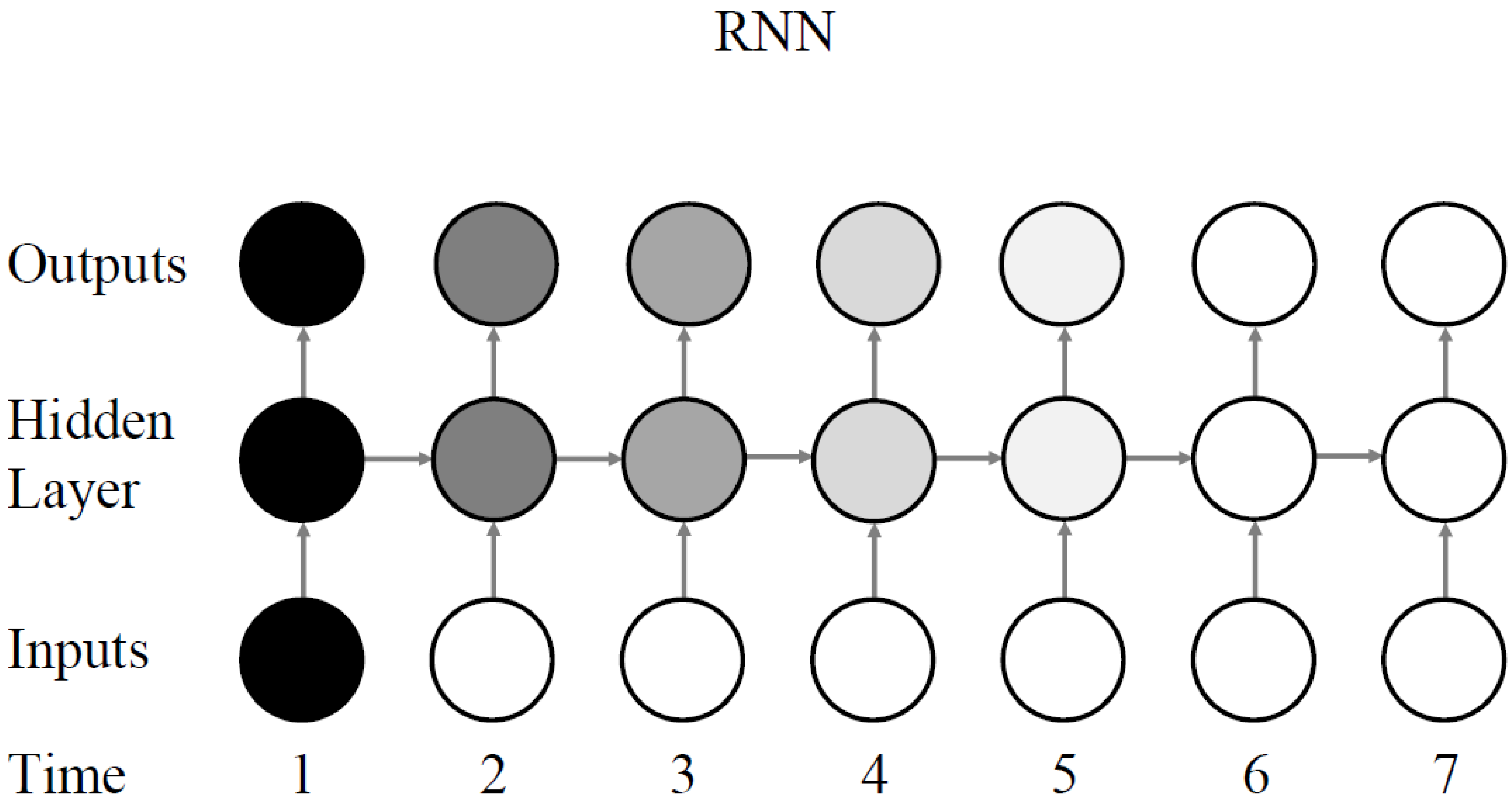
$$g = \mathrm{Tanh}\big(W(h_{t-1} \odot r) + Vx_t + b_g\big)$$

$$h_t = (1-u) \odot g + u \odot h_{t-1}$$

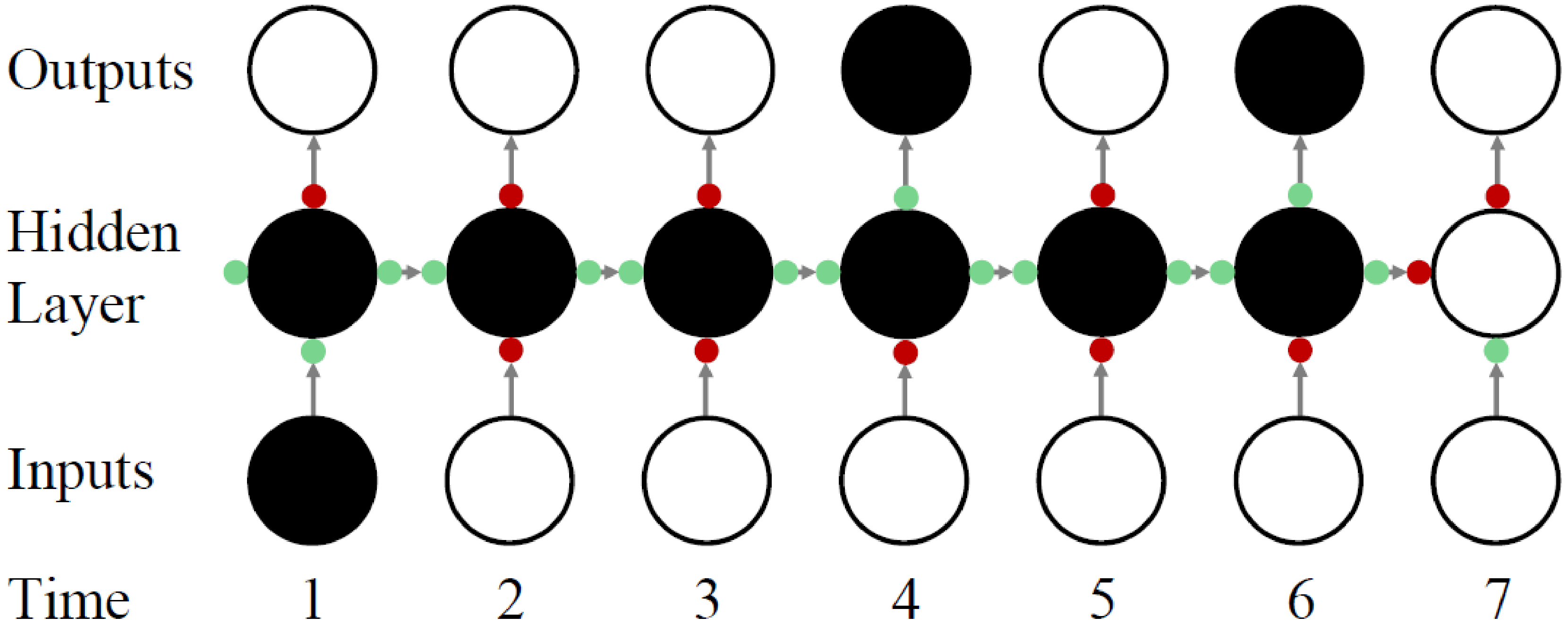
LSTM or GRU

- LSTM
  - more parameters => longer training
  - More flexible
- GRU
  - Less parameters => faster training
- Train using LSTM first
- Train using GRU next
- Choose GRU if performance is similar

RNN Information Flow



LSTM Information Flow

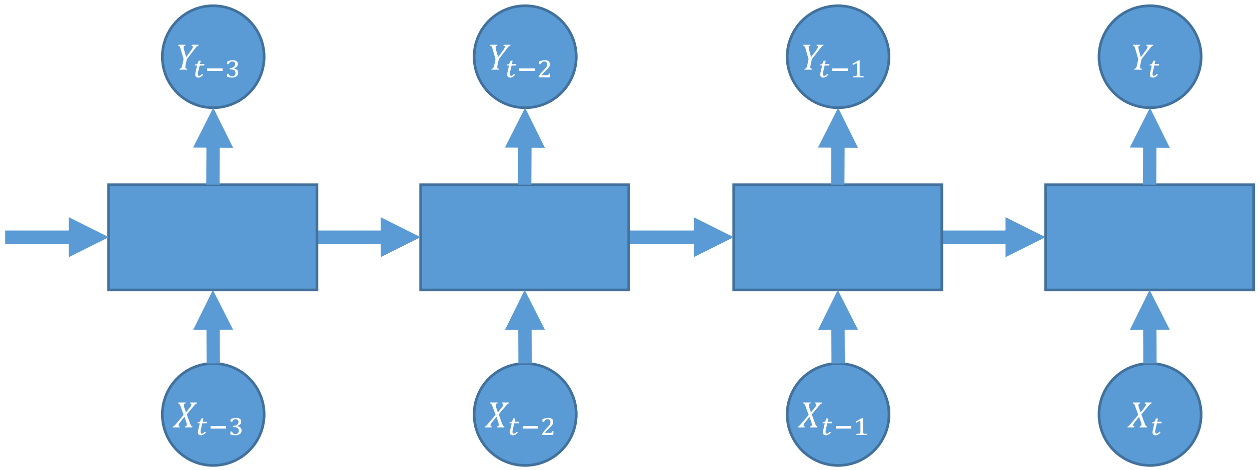


Regularization

- L2 regularization is very effective
- Dropout can be applied to V, U but not W (memory)

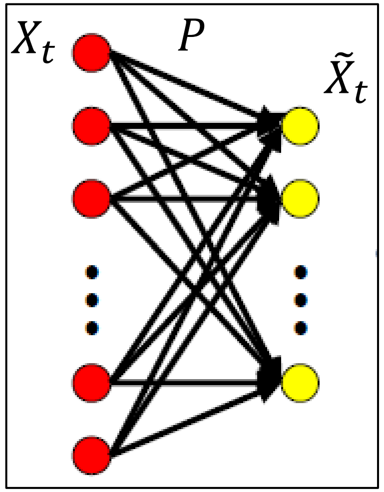
Text and Language Modeling

- Input as one-hot vector
- For text:
  - each letter is a vector
  - number of characters = dimension of vector
  - Includes upper case, lower case, hyphenated, commas, apostrophes as characters
  - Output: predict next character/word
- For language:
  - each word is a vector
  - Dictionary of all inputs
  - size of dictionary = dimension of vector
  - Includes upper case, lower case, hyphenated, commas, apostrophes as words
  - Output: predict next word/sentence



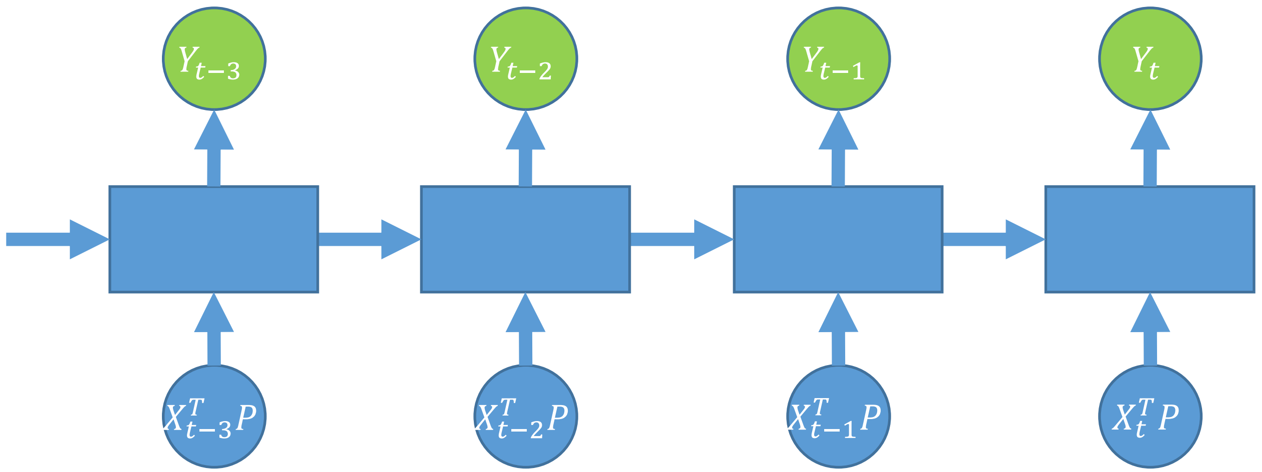
Curse of Dimensionality

- Number of training samples = multiple of each dimension
- Observation: sparse space
  - Vertices of the space used not volume
  - Density $= N/2^N$
  - Highly inefficient
- Observation: vectors are unordered
  - Same length
- Idea: project to lower dimension space
  - Input vector is X_t: $1 \times N$
  - Projection is $1 \times M$, $M < N$
  - Projection function P: $M \times N$
  - Projection: $\tilde{X}_t = PX_t^T$
  - Learn the projection function, P
  - Unsupervised

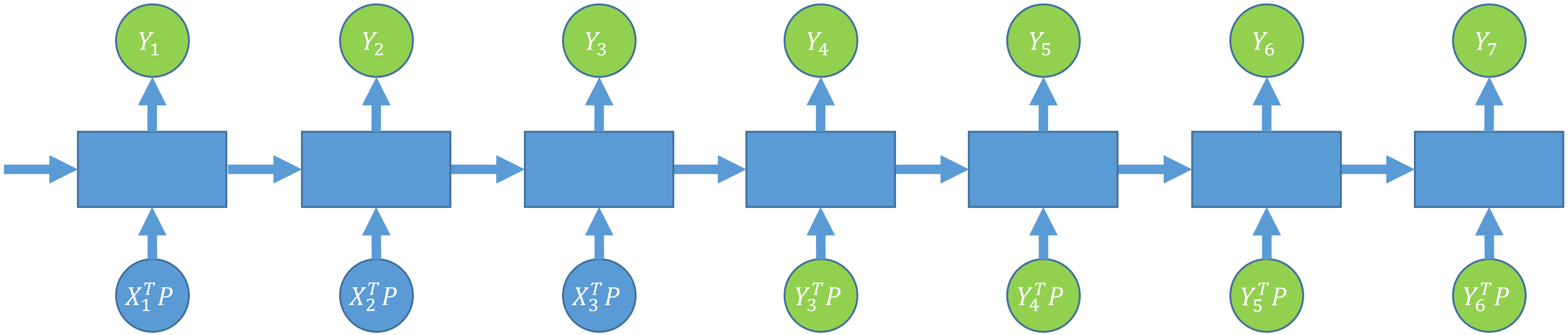


Language Synthesis: Training

- Use BPTT to train model



Language Synthesis: Generation



- Provide first few inputs
- Let the network feedback output back in

Beam Search

- Output: softmax gives most likely next character (or word)
  - Probability distribution over all dictionary vectors
  - Greedy: susceptible to propagating errors

$$\text{The quick br}\begin{bmatrix} a = 0.3 \\ b = 0.01 \\ c = 0.01 \\ \vdots \\ \color{red}{o = 0.4} \\ \vdots \end{bmatrix}$$

- Beam search:
  - pick a number of non-max outcomes (aka hypothesis)
  - Evaluate each hypothesis by its overall probability: $\prod_i \mathrm{p}(y_i)$
  - Prune weak hypothesis at each iteration
  - Repeat

$$\text{The quick br}\begin{bmatrix} \color{red}{a = 0.3} \\ b = 0.01 \\ c = 0.01 \\ \vdots \\ \color{red}{o = 0.4} \\ \vdots \end{bmatrix} \qquad \begin{bmatrix} a = 0.01 \\ b = 0.1 \\ c = 0.3 \\ \vdots \end{bmatrix} \qquad \begin{bmatrix} a = 0.2 \\ b = 0.01 \\ \vdots \\ w = 0.6 \\ \vdots \end{bmatrix}$$

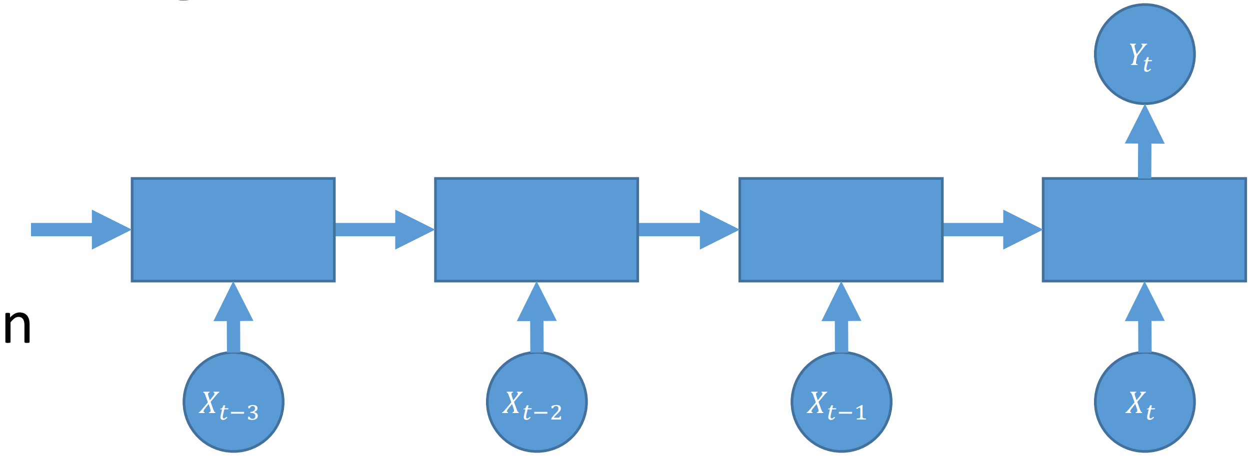
$$\begin{bmatrix} \cancel{\text{br}a = 0.3 \times 0.01} \\ \cancel{\text{br}b = 0.3 \times 0.1} \\ \text{br}c = 0.3 \times 0.3 \\ \vdots \\ \text{br}o a = 0.4 \times 0.2 \\ \cancel{\text{br}o b = 0.4 \times 0.01} \\ \text{br}o w = 0.4 \times 0.6 \\ \vdots \end{bmatrix}$$

Sampled Softmax

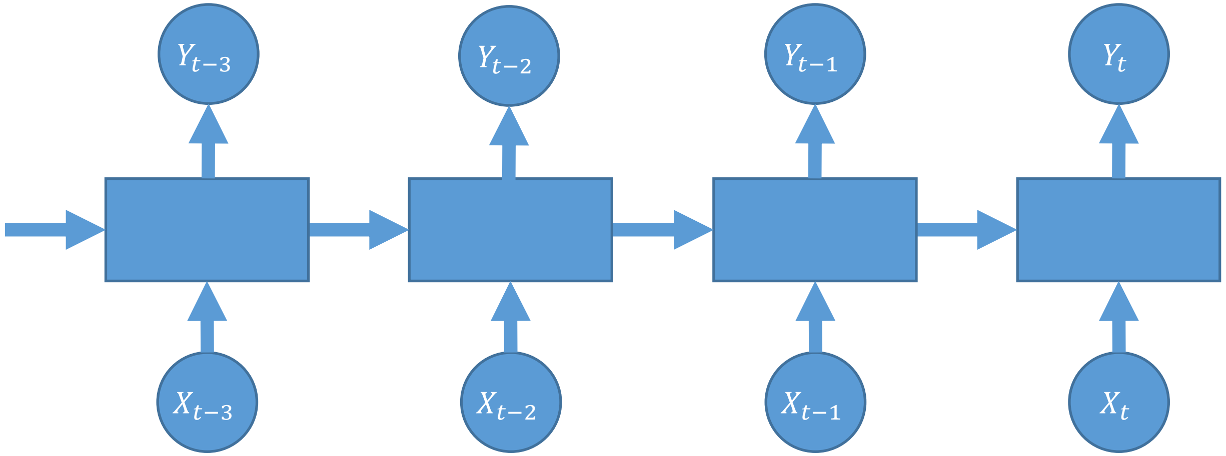
- Regular softmax: $e^{y_j} / \sum_i e^{y_i}$
  - j is the target output
  - i is the negative (non-targets)
- Problem: possible outputs i is large
  - large dictionary
- Randomly select negatives and use in softmax
  - Importance sampling (speedup x19)
  - Adaptive importance sampling (x100)
  - Target sampling (AIS but partitioning training data to limit words)
- Alternatives:
  - Self normalization (x15 higher accuracy)
  - Noise contrastive estimator (x45 higher accuracy)

Phenome Recognition

- Input: sequence of spectral data
- Output: phenome
- In reality output is produced in each iteration
    - Ignored until end

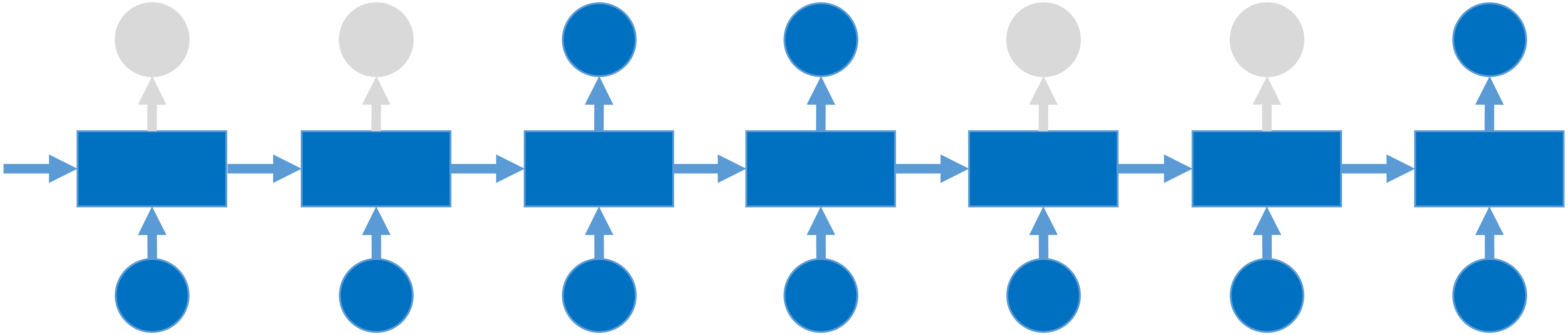


- Training: consider error at each iteration



Speech Recognition

- Input: sequence of inputs data (spectra)
- Output: <u>**asynchronously**</u> sequence of symbols (phenomes)



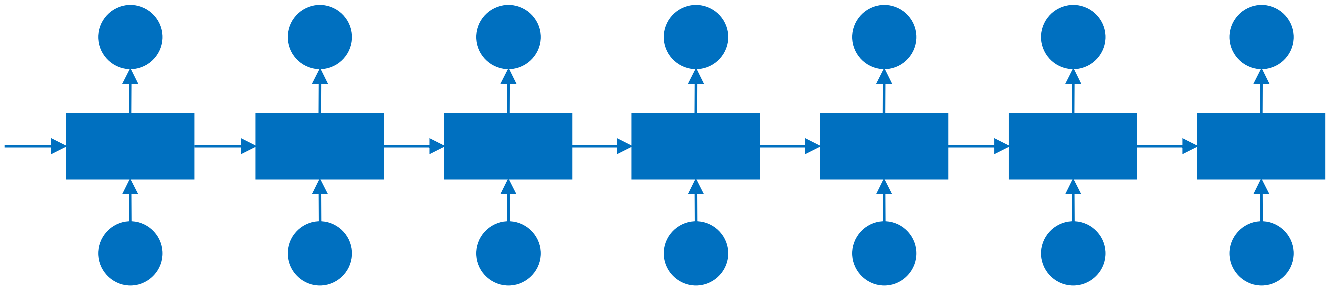
Speech Recognition Training

- No timing information
- Problem 1: output is a probability distribution over all symbols (phenomes)
- Problem 2: Can’t differentiate between symbol repetition and symbol extension
- Problem 3: Even if the sequence is know, the timed output is not

Speech Recognition Training

- No timing information
- Problem 1: output is a probability distribution over all symbols (phenomes)
- Solution: merge the symbols

a	0.1	0.15	0.4	0.55	0.45	0.2	0.15
b	0.5	0.6	0.3	0.05	0.1	0.05	0.15
d	0.1	0.05	0.2	0.05	0.3	0.3	0.25
e	0.1	0.10	0.05	0.1	0.05	0.2	0.15
i	0.1	0.05	0.05	0.2	0.05	0.1	0.2
f	0.1	0.05	0.05	.05	0.05	0.05	0.2



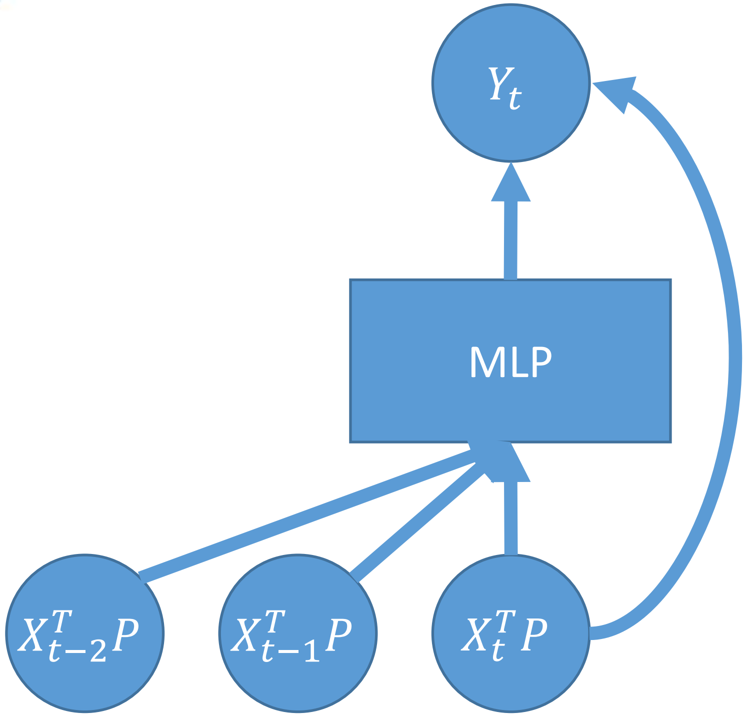
Speech Recognition Training

- Problem 2: Can’t differentiate between symbol repetition and symbol extension

Speech Recognition Training

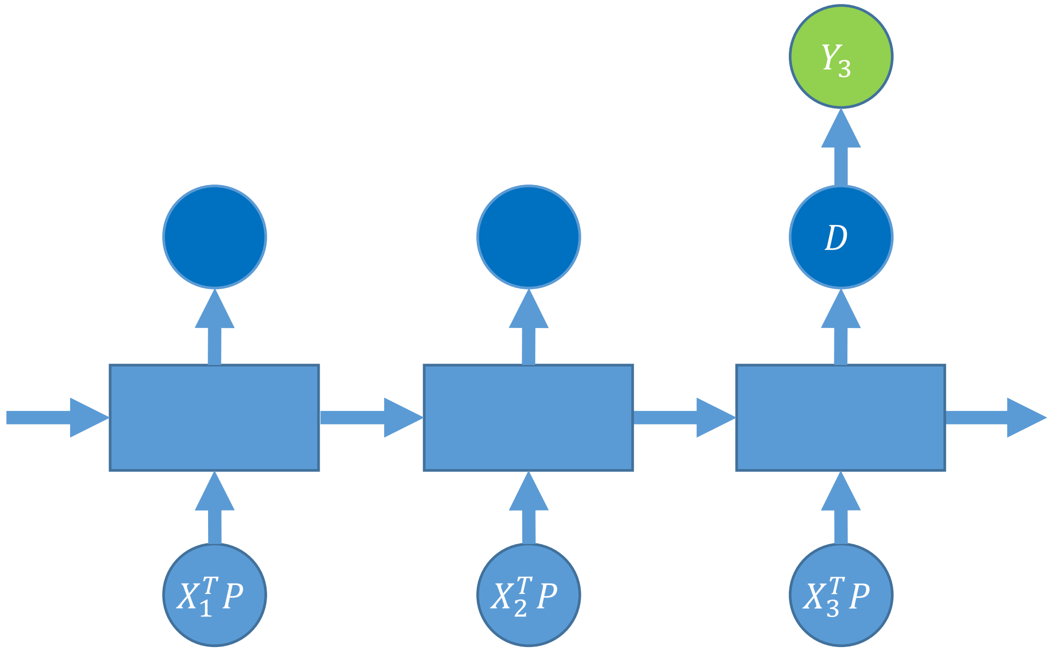
- No timing information
- Problem 1: output is a probability distribution over all symbols (phenomes)
- Problem 2: Can't differentiate between symbol repetition and symbol extension
- Problem 3: Even if the sequence is know, the timed output is not

TDNN Model



- Predict characters/words based on last N

Language Synthesis: Generation



- Provide first few inputs
- After last input, generate a probability distribution over all dictionary entries
- Draw an entry from the dictionary with the highest probability