Deep Learning
lecture 8
Sequence Modeling (1)

Yi Wu, IIIS

Spring 2024

Apr-19

Logistics

- Special Arrangement: **Two Lectures next week**
  - **Apr. 22 (Monday) 19:20, 舜德楼301, Lec.9**
  - **Apr. 26 (Friday) 13:30, 舜德楼301, Lec.10**

- Coding Project 3 is due on Apr. 28 (Sunday)
  - No format/submission/evaluation error allowed!

- Final Project Proposal to be announced on next Monday
  - 2~3 students per team
  - Only specific topics allowed
  - A final poster session ☺

Today’s Topic

- Basic models for sequence data
  - Recurrent neural networks
  - LSTM

- Basic techniques for modeling natural language

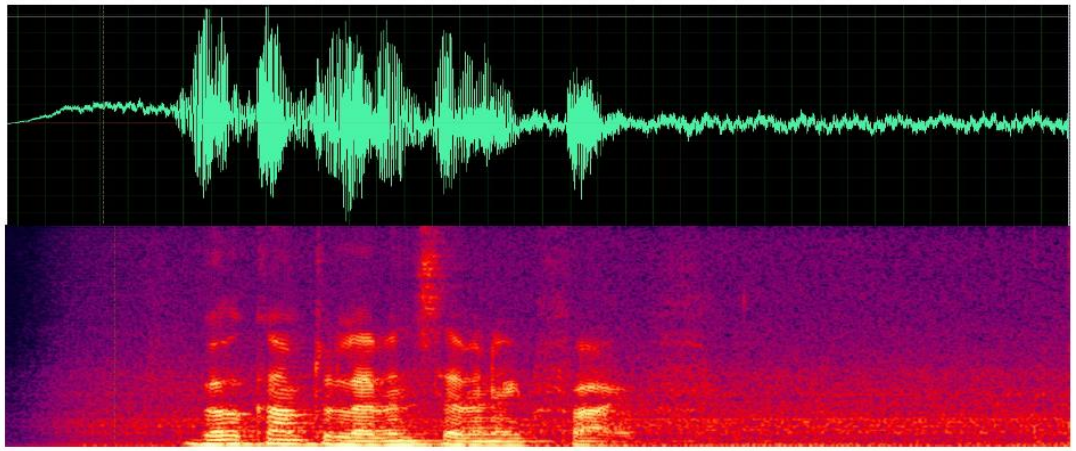
Story So Far

- Supervised Learning (Lec. 2~3)
  - Discriminative Models
  - Network architectures and learning algorithms

- Generative Models (Lec. 4~7)
  - Energy-based models (contrastive divergence + MCMC)
  - Flow model (bijections)
  - VAE (variational inference)
  - GAN (neural loss function)
  - Trade-offs between expressiveness, inference and training

Sequence Data

- Most existing discussions assume fixed dimensions
  - E.g.: Image classification and generation
    - Input image has fixed width and height
    - Fixed output dimension
  - Fixed amount of network layers and parameters
- What if the dimension of input varies a lot?
  - Finding the “welcome” (lecture 2)



Sequence Data

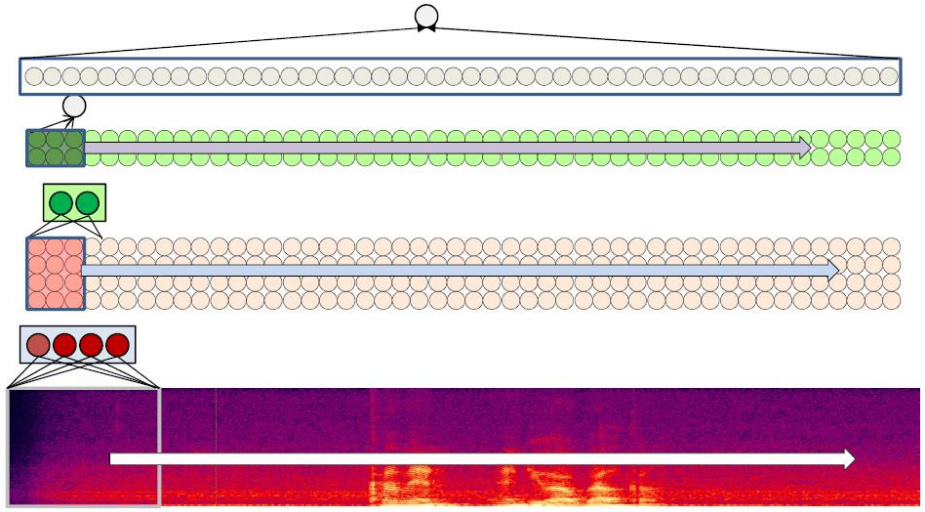
- Most existing discussions assume f
  - E.g.: Image classification and generat
    - Input image has fixed width and height
    - Fixed output dimension
  - Fixed amount of network layers and
- What if the dimension of input var
  - Finding the “welcome” (lecture 2)
  - Generating poet

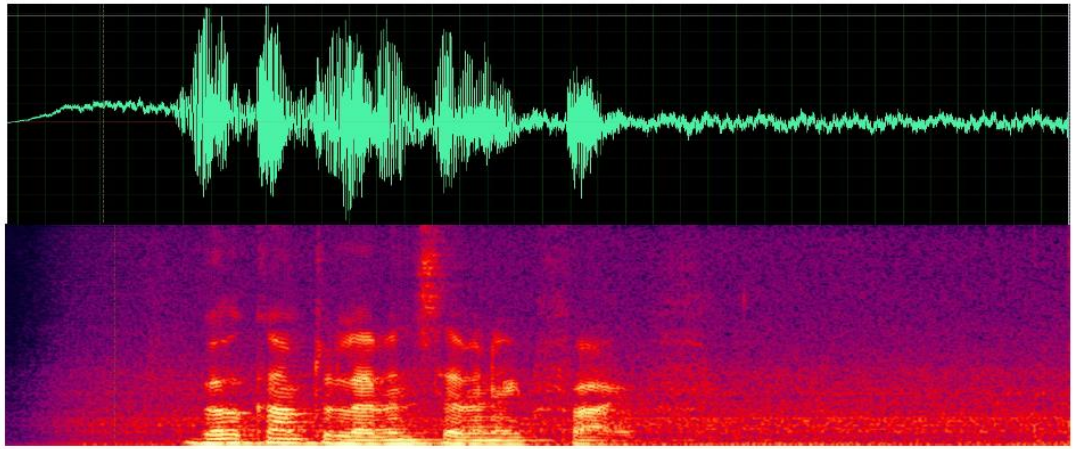
Sequence Data

- Most existing discussions assume fixed dimensions
  - E.g.: Image classification and generation
    - Input image has fixed width and height
    - Fixed output dimension
  - Fixed amount of network layers and parameters
- What if the dimension of input varies a lot?
  - Finding the “welcome” (lecture 2)
  - Generating poet
  - Machine translation

Sequence Data

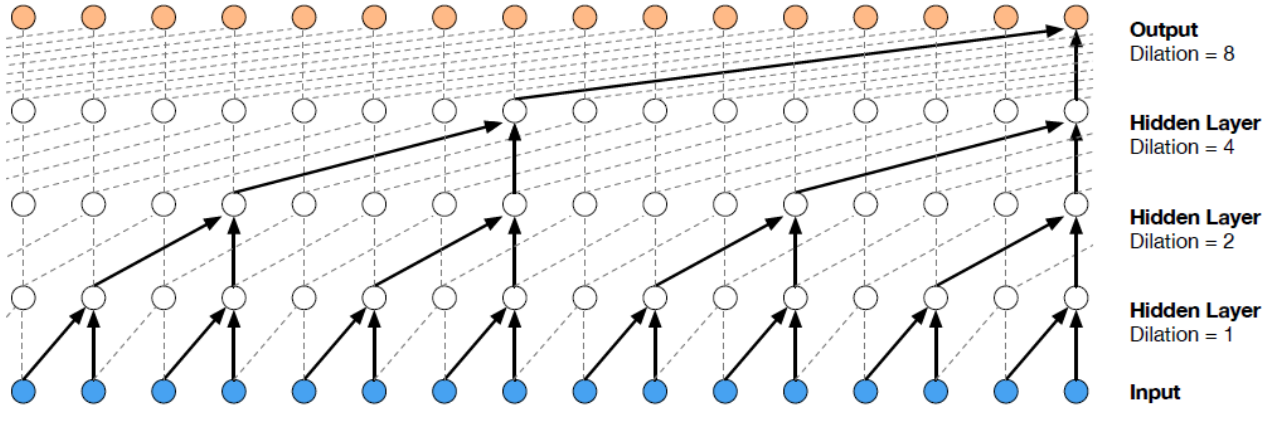
- Finding the “Welcome”
  - Input data $X_1 \dots X_L$, L may vary
  - Whether the voice contains “Welcome”
- Possible Solution: Time-Delayed Network (Lec. 2, Temporal Conv-Net)
  - Suppose kernel size is d, how may layers we need?
    - K layers can at most classify a “Welcome” of length $O(k \cdot d)$

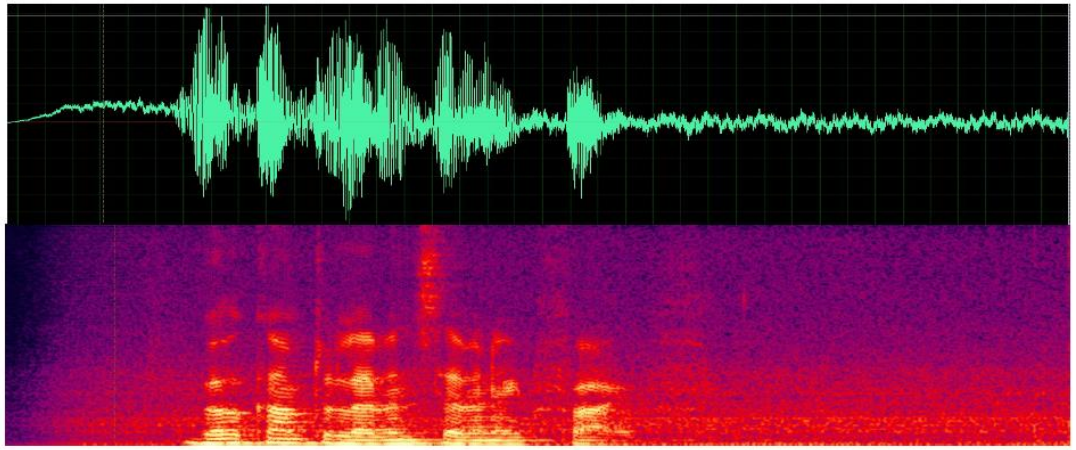




Sequence Data

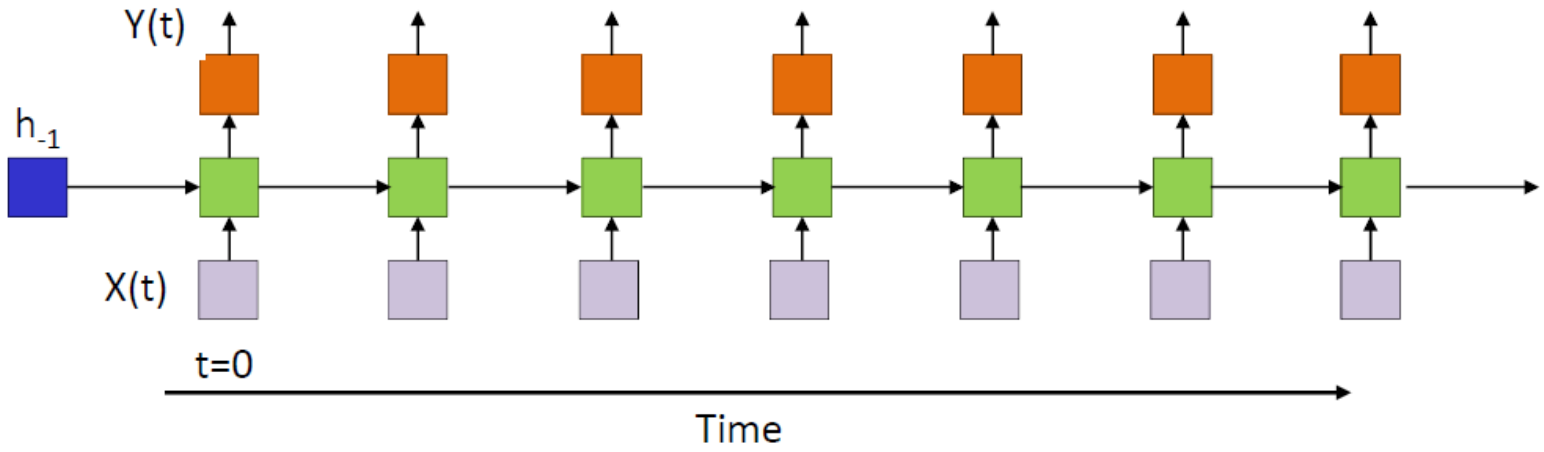
- Finding the “Welcome”
  - Input data $X_1 \ldots X_L$, L may vary
  - Whether the voice contains “Welcome”
- Improved Solution: Dilated ConvNet (Lec 6, WaveNet)
  - How may layers do we need?
    - $O(\log N)$
    - we still need a network of varying depth for arbitrarily long sequence





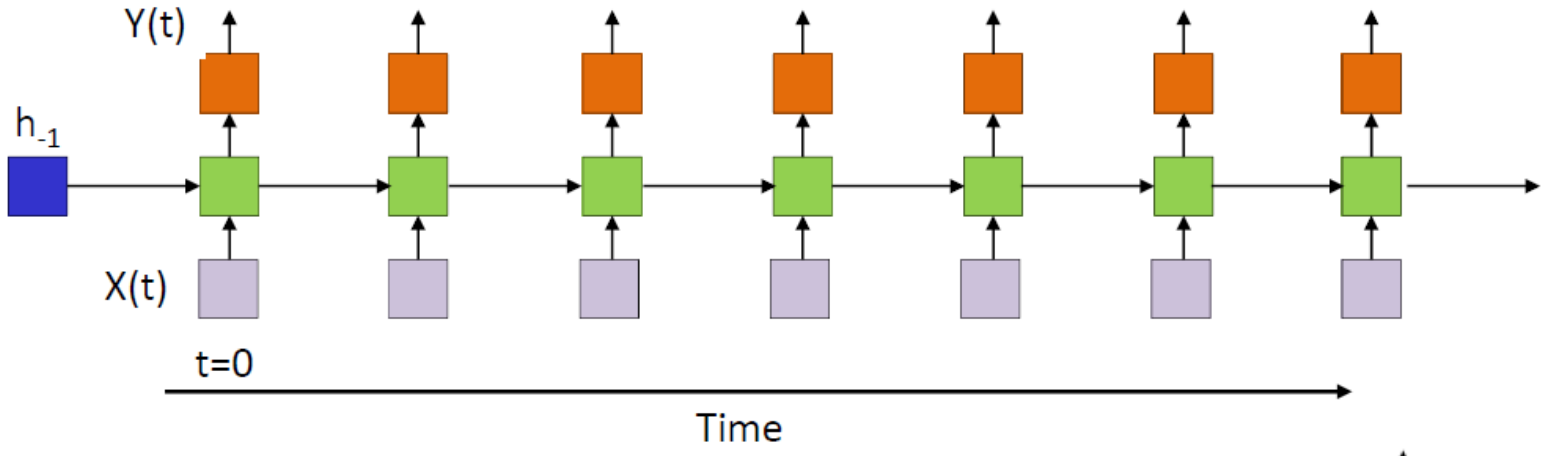
Sequence Data

- Finding the “Welcome”
  - Input data $X_1 \dots X_L$, L may vary
  - Whether the voice contains “Welcome”
- Goal: a fixed-size model for arbitrarily long sequences
  - Idea: reuse the same “layer” repeatedly
- State-Space Model
  - h_t: (hidden) state
  - X_t: input
  - Y_t: output
  - $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
  - h_{-1}: initial state

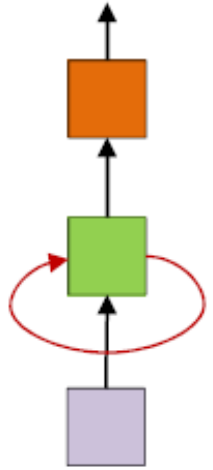


Recurrent Neural Network

- State-Space Model
  - h_t: (hidden) state
  - X_t: input; Y_t: output
  - $h_t = f_1(h_{t-1}, X_t; \theta)$
  - $Y_t = f_2(h_t; \theta)$
  - h_{-1}: initial state

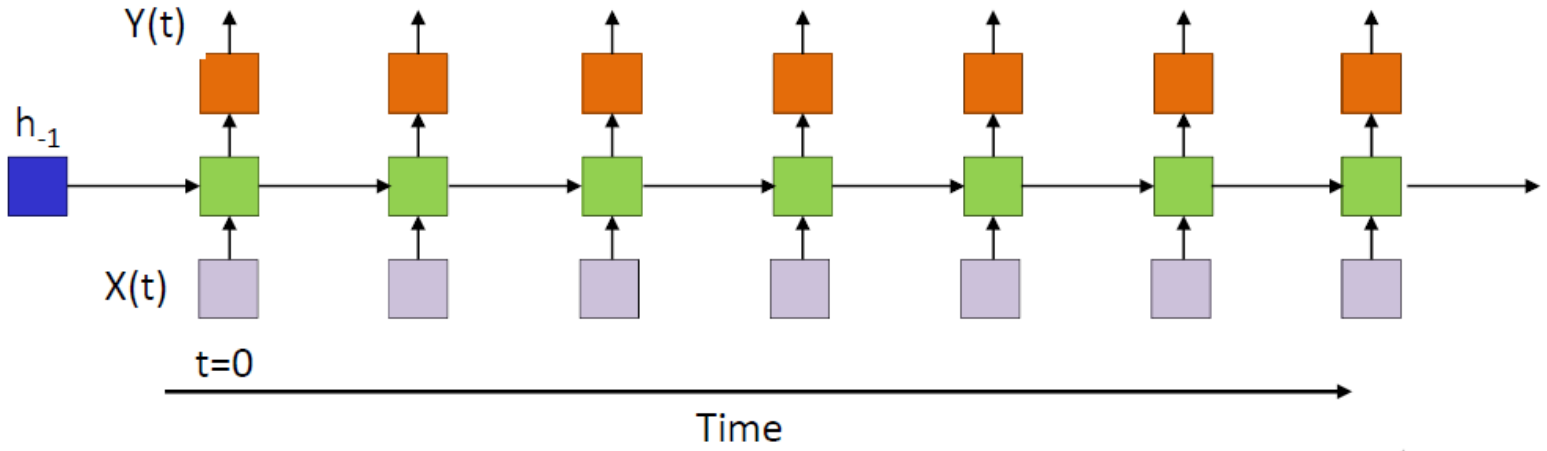


- Same neural network across all the columns!
  - Simplified drawing (loops implies recurrence)
  - h_t: a vector that summarizes all past inputs (also called “memory”)
  - h_{-1} affects the whole network (typically set to zero)
  - Y_t is computed over $X_0, \dots, X_t$
  - X_t affect all the outputs and states after t

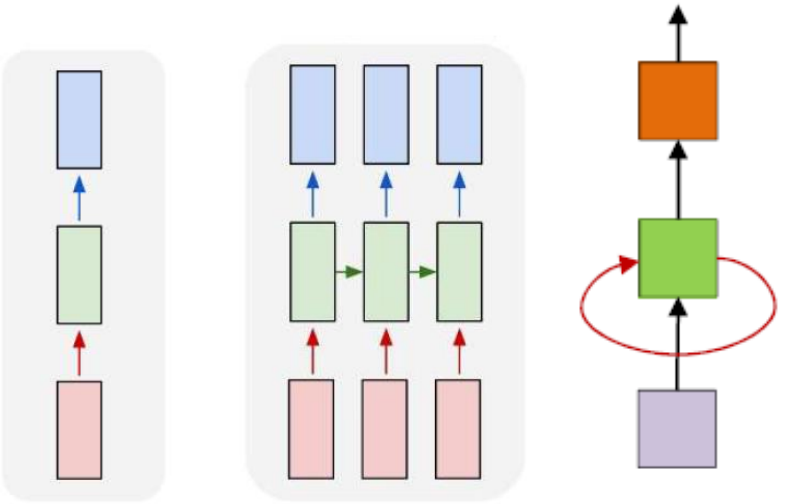


Recurrent Neural Network

- State-Space Model
  - h_t: (hidden) state
  - X_t: input; Y_t: output
  - $h_t = f_1(h_{t-1}, X_t; \theta)$
  - $Y_t = f_2(h_t; \theta)$
  - h_{-1}: initial state

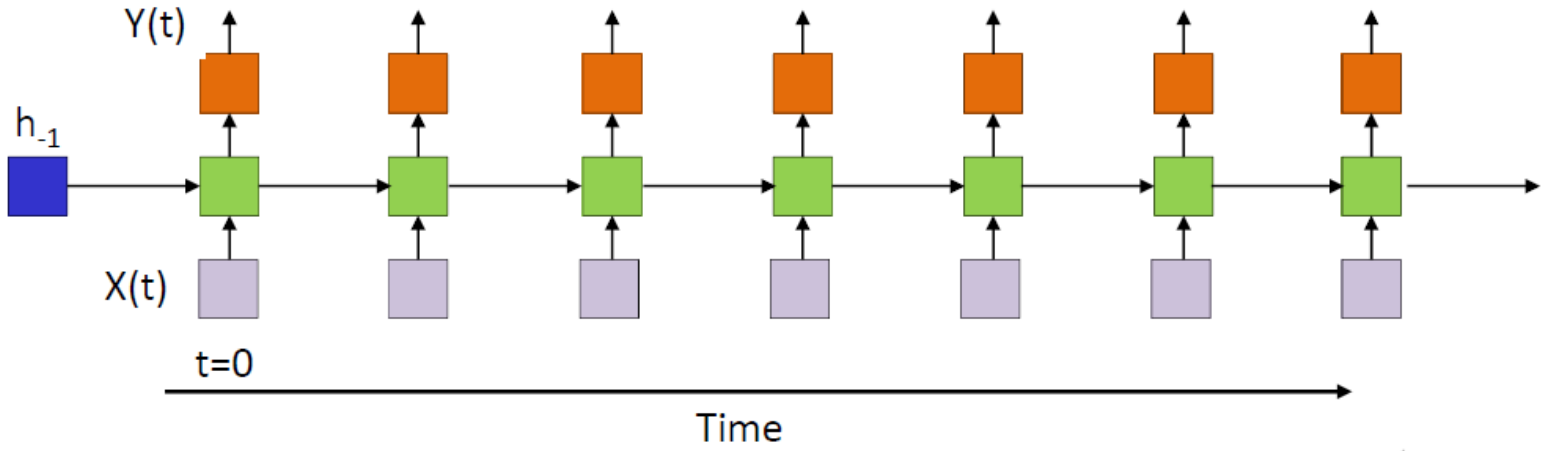


- Same neural network across all the columns!
  - MLP v.s. RNN
    - RNN can be viewed as repeatedly applying MLPs
    - $h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$
    - $Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$
    - f_1, f_2 are activations (e.g., Sigmoid, tanh, ReLU, Softmax)



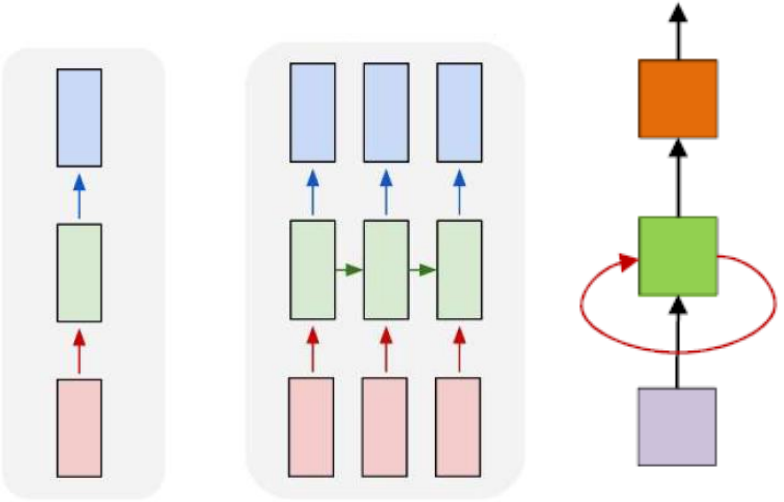
Recurrent Neural Network

- State-Space Model
  - h_t: (hidden) state
  - X_t: input; Y_t: output
  - $h_t = f_1(h_{t-1}, X_t; \theta)$
  - $Y_t = f_2(h_t; \theta)$
  - h_{-1}: initial state



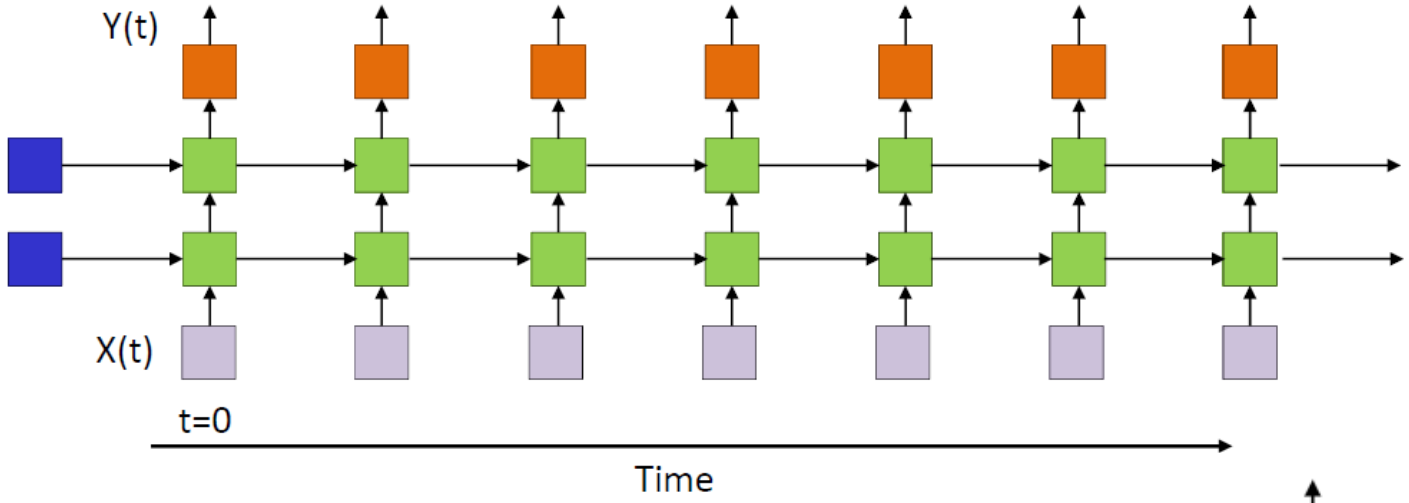
- Same neural network across all the columns!
  - MLP v.s. RNN
    - RNN can be viewed as repeatedly applying MLPs
    - $h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$
    - $Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$
    - f_1, f_2 are activations (e.g., Sigmoid, tanh, ReLU, Softmax)

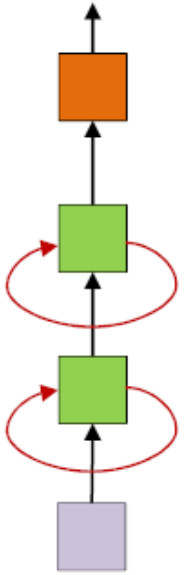
Recurrent weights!



Recurrent Neural Network

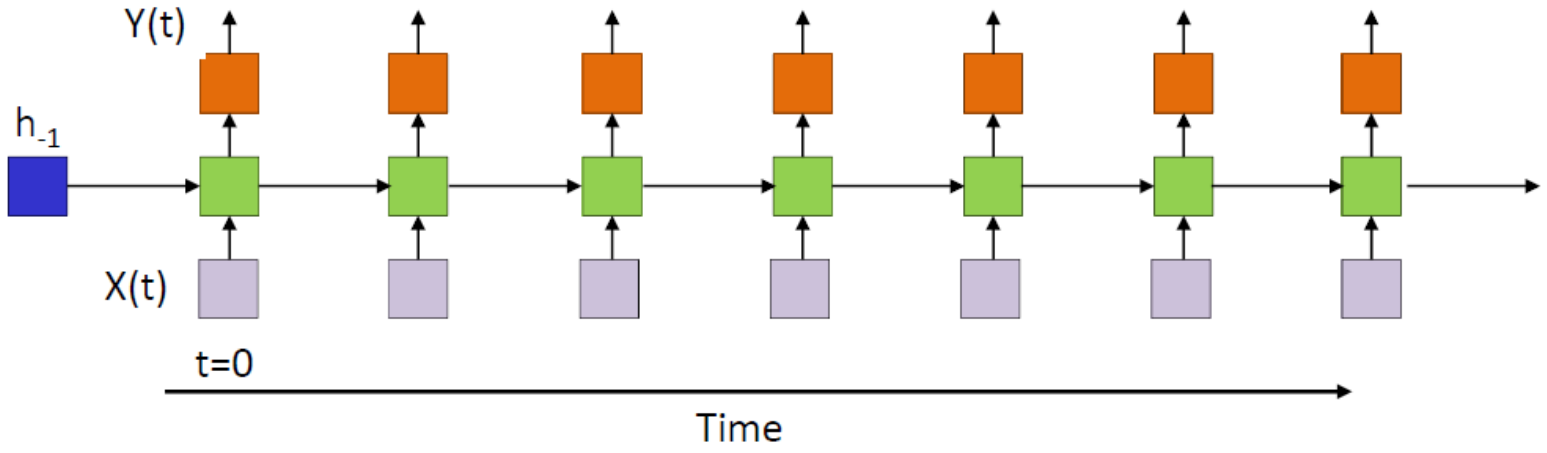
- Stack K layers of RNNs!
  - Multi-layer RNN
- State-Space Model
  - $h_t^{(k)}$: (hidden) states
  - X_t: input; Y_t: output
  - $h_t^{(1)} = f_1^{(1)}\left(h_{t-1}^{(1)}, X_t; \theta\right)$
  - $h_t^{(k)} = f_1^{(k)}\left(h_{t-1}^{(k)}, h_t^{(k-1)}; \theta\right)$
  - $Y_t = f_2\left(h_t^{(K)}; \theta\right)$
  - $h_{-1}^{(k)}$: initial states





Recurrent Neural Network

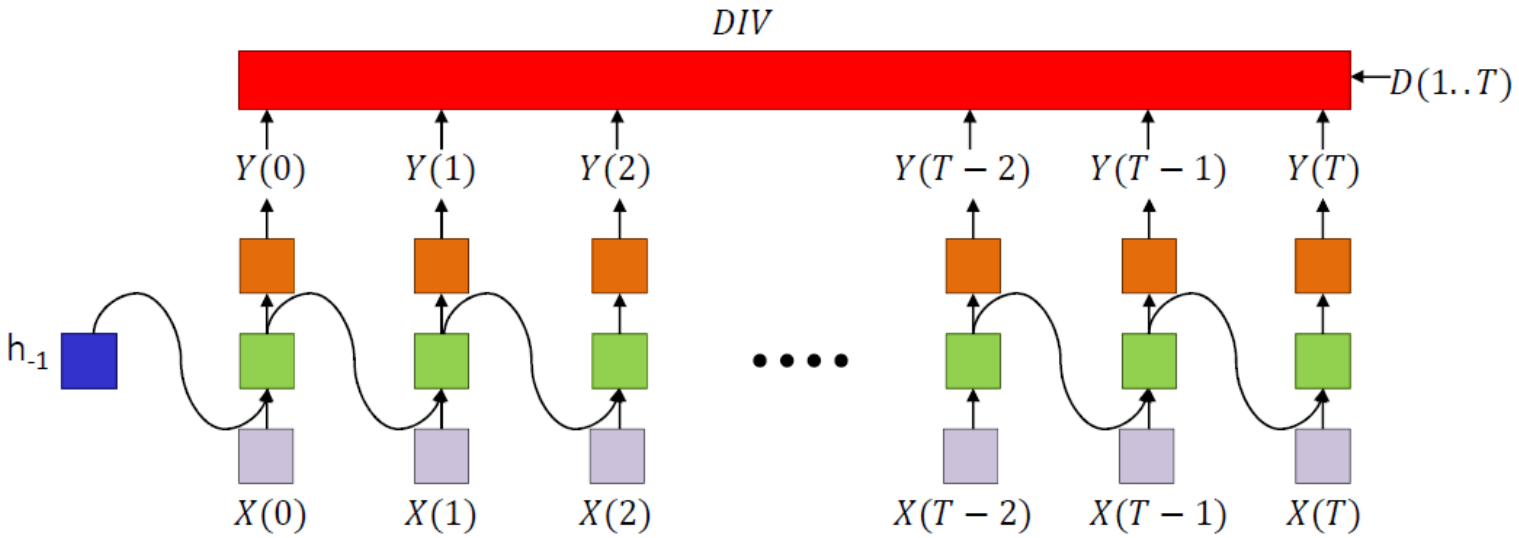
- State-Space Model
  - h_t: (hidden) state
  - X_t: input; Y_t: output
  - $h_t = f_1(h_{t-1}, X_t; \theta)$
  - $Y_t = f_2(h_t; \theta)$
  - h_{-1}: initial state



- How to train an RNN?
  - Assume we have paired input and target sequence $\{(X_t, D_t)\}_t$
    - Remark
      - RNN can handle much more flexible data format than fully paired data
      - But let's simply keep this assumption for now

Recurrent Neural Network

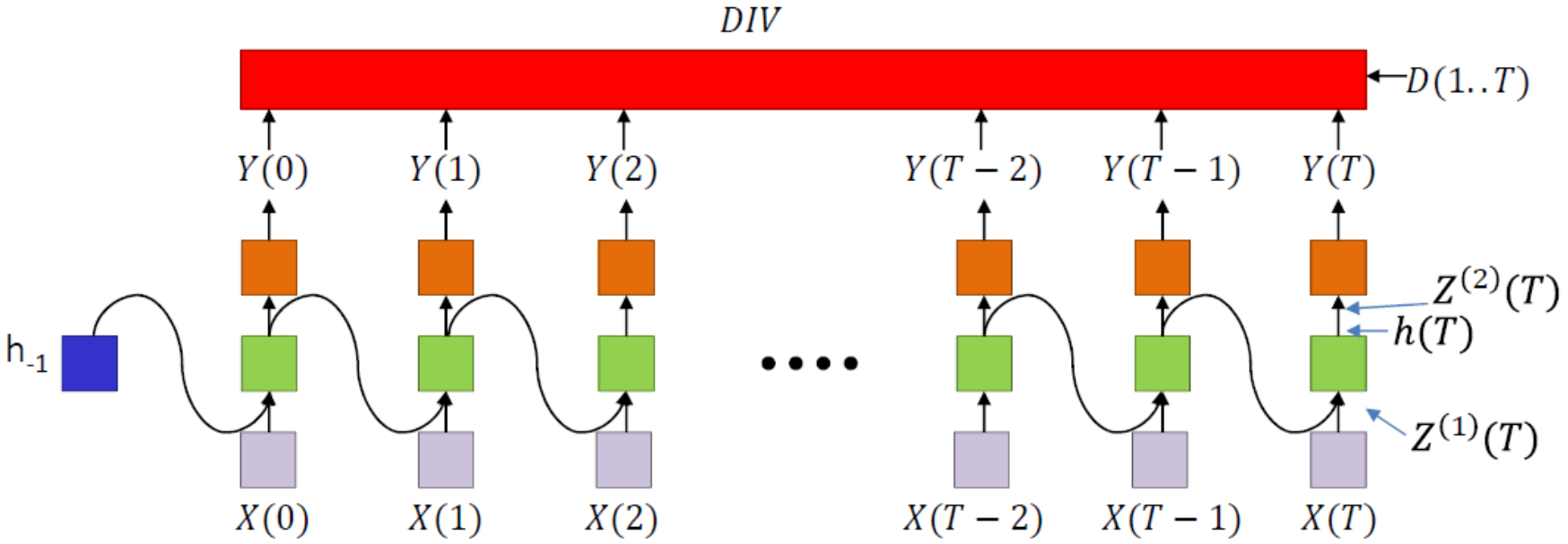
- State-Space Model
  - h_t: (hidden) state
  - X_t: input; Y_t: output
  - $h_t = f_1(h_{t-1}, X_t; \theta)$
  - $Y_t = f_2(h_t; \theta)$
  - h_{-1}: initial state



- How to train an RNN?
  - Assume we have paired input and target sequence $\{(X_t, D_t)\}_t$
  - We can define the loss function $L(\theta) = \sum_t Div(Y_t, D_t)$
    - Goal: learn the best parameter $\theta^\star$ via gradient descent
    - Let's do backpropagation for RNN!

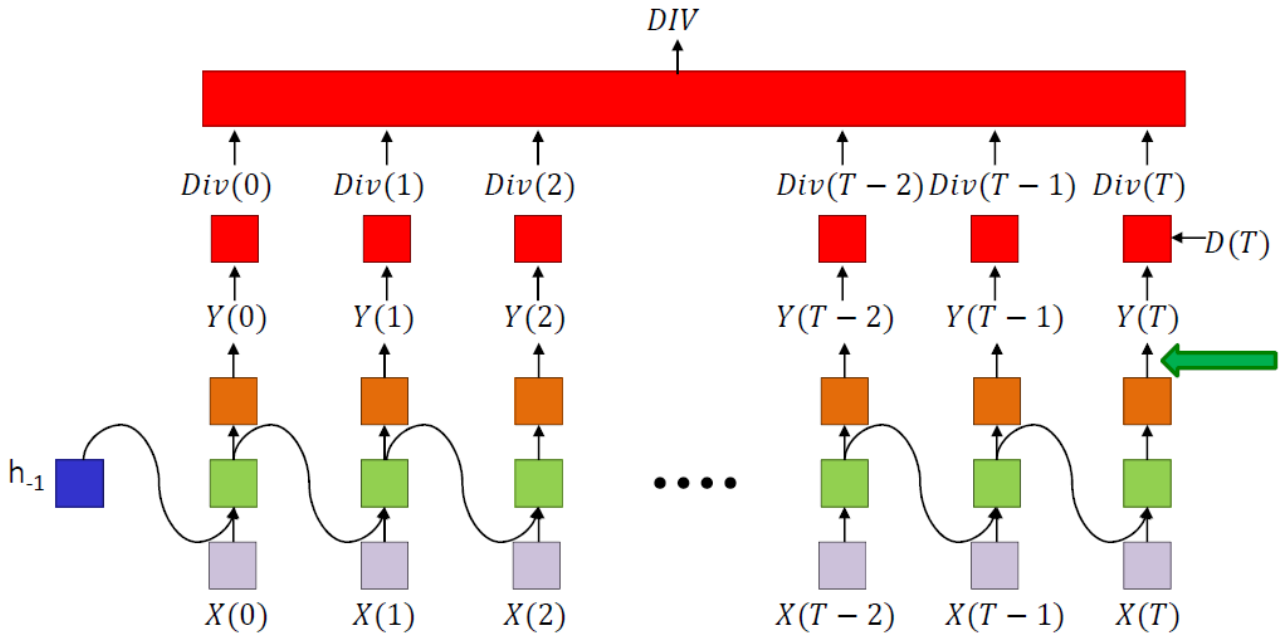
Back Propagation Through Time

- Backpropagation for RNN
  - Let's focus on 1 training instance
  - $Z_t^{(2)}$: pre-activation of output ($Y_t = f_2\left(Z_t^{(2)}; \theta\right)$)
  - $Z_t^{(1)}$: pre-activation of hidden ($h_t = f_1\left(Z_t^{(1)}; \theta\right)$)



Back Propagation Through Time

- Backpropagation for RNN
  - We start from the final timestep T
  - We first compute $\frac{\partial Div}{\partial Y_T}$



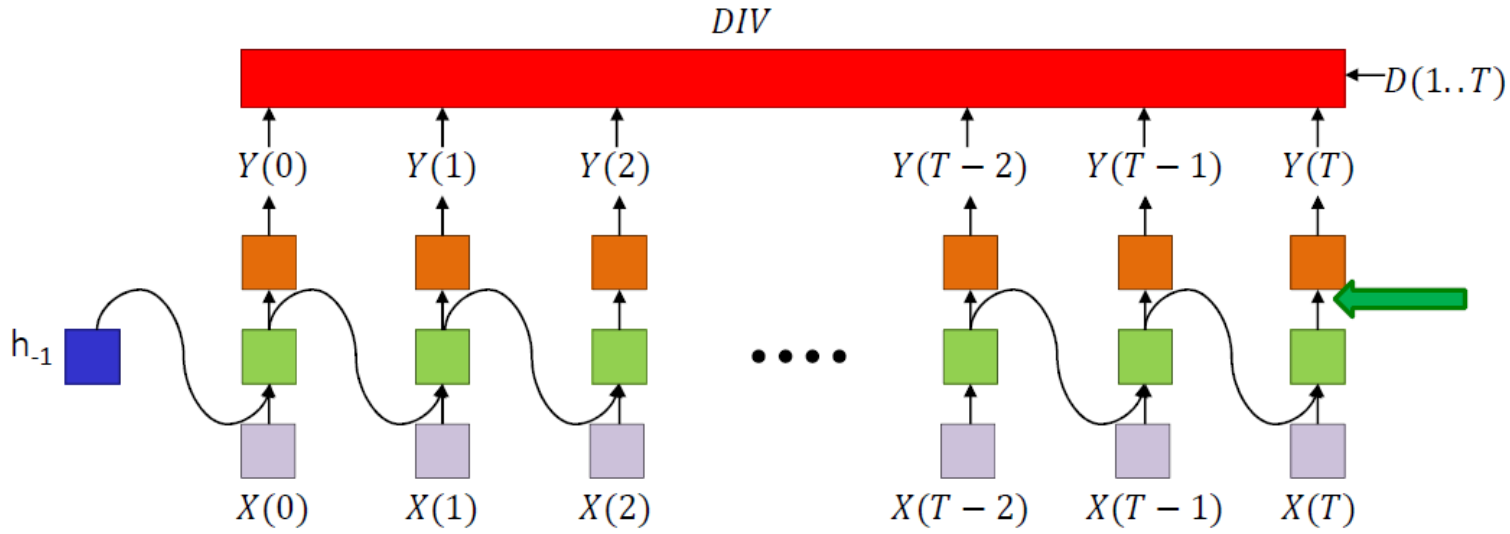
Back Propagation Through Time

- Backpropagation for RNN
  - We start from the final timestep T
  - We first compute $\frac{\partial Div}{\partial Y_T}$
  - Then $\frac{\partial Div}{\partial Z_T^{(2)}}$

$$Y_t = f_2\left(Z_t^{(2)}; \theta\right)$$

Vector output activation

$$\frac{dDIV}{dZ_i^{(2)}(T)} = \frac{dDIV}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(2)}(T)}$$

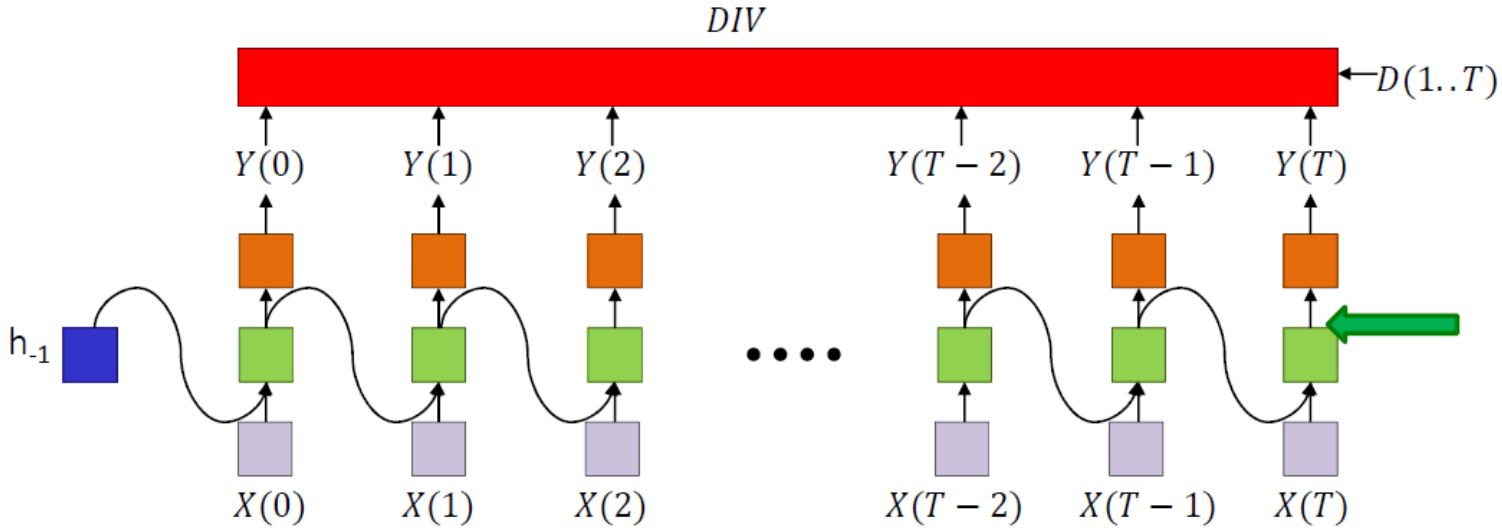


Back Propagation Through Time

- Backpropagation for RNN
  - We start from the final timestep T
  - We first compute $\frac{\partial Div}{\partial Y_T}$
  - Then $\frac{\partial Div}{\partial Z_T^{(2)}}$ and $\frac{\partial Div}{\partial h_T}$

$$Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$$

$$\frac{dDIV}{dh_i(T)} = \sum_j \frac{dDIV}{dZ_j^{(2)}(T)} \frac{dZ_j^{(2)}(T)}{dh_i(T)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T)}$$

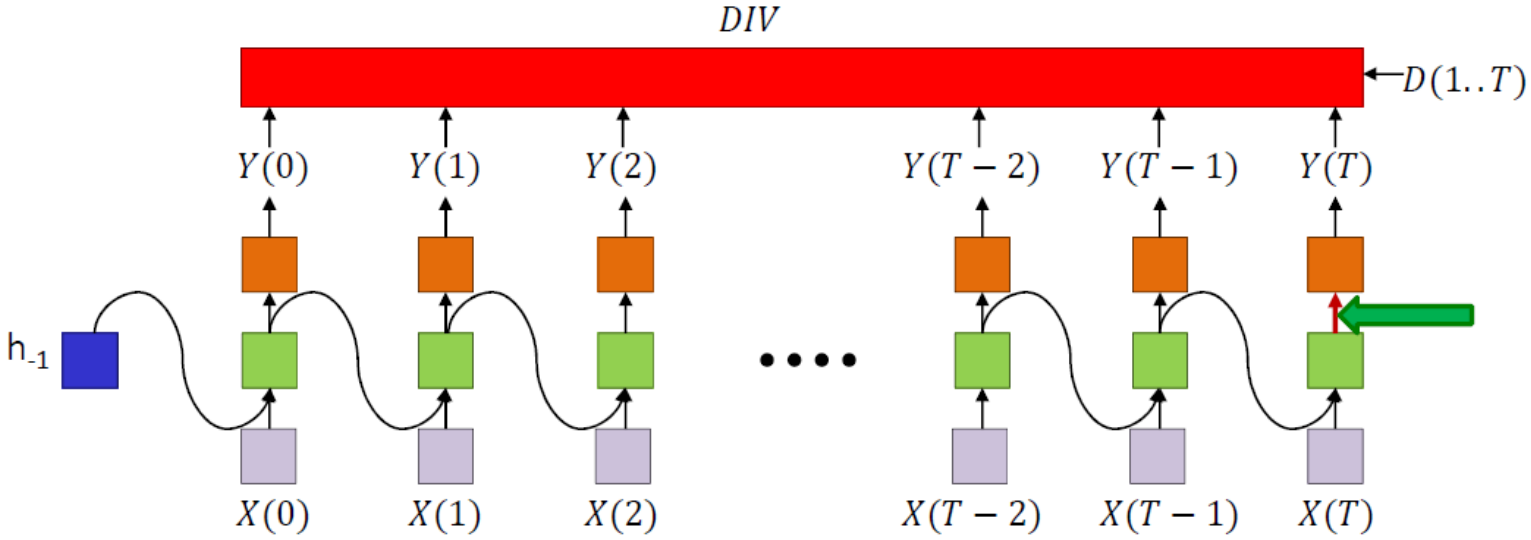


Back Propagation Through Time

- Backpropagation for RNN
  - We start from the final timestep T
  - We first compute $\frac{\partial Div}{\partial Y_T}$
  - Then $\frac{\partial Div}{\partial Z_T^{(2)}}$ and $\frac{\partial Div}{\partial h_T}$
  - Gradient of $W^{(2)}$

$$Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$$

$$\frac{dDIV}{dw_{ij}^{(2)}} = \frac{dDIV}{dZ_j^{(2)}(T)} h_i(T)$$

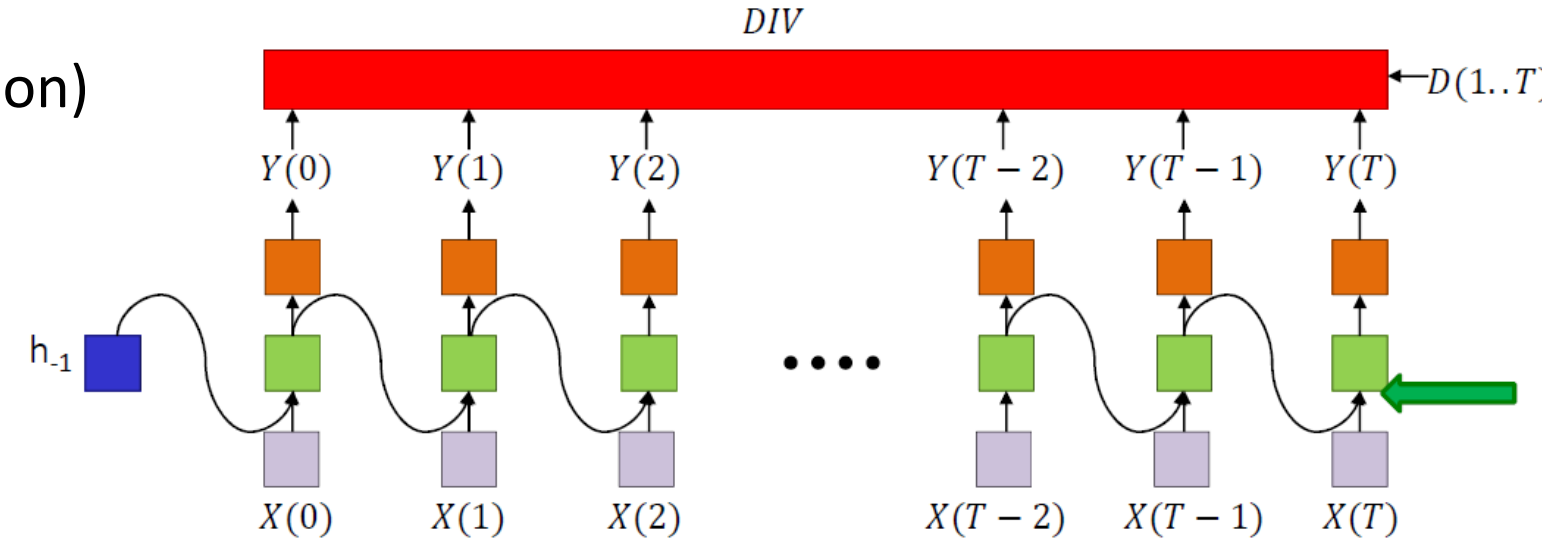


Back Propagation Through Time

- Backpropagation for RNN
  - We start from the final timestep T
  - We first compute $\frac{\partial Div}{\partial Y_T}$
  - Then $\frac{\partial Div}{\partial Z_T^{(2)}}$ and $\frac{\partial Div}{\partial h_T}$
  - Gradient of $W^{(2)}$
  - Then $\frac{\partial Div}{\partial Z_T^{(1)}}$ (vector activation)

$$h_t = f_1\left(Z_t^{(1)}; \theta\right)$$

$$\frac{dDIV}{dZ_i^{(1)}(T)} = \frac{dDIV}{dh_i(T)} \frac{dh_i(T)}{dZ_i^{(1)}(T)}$$

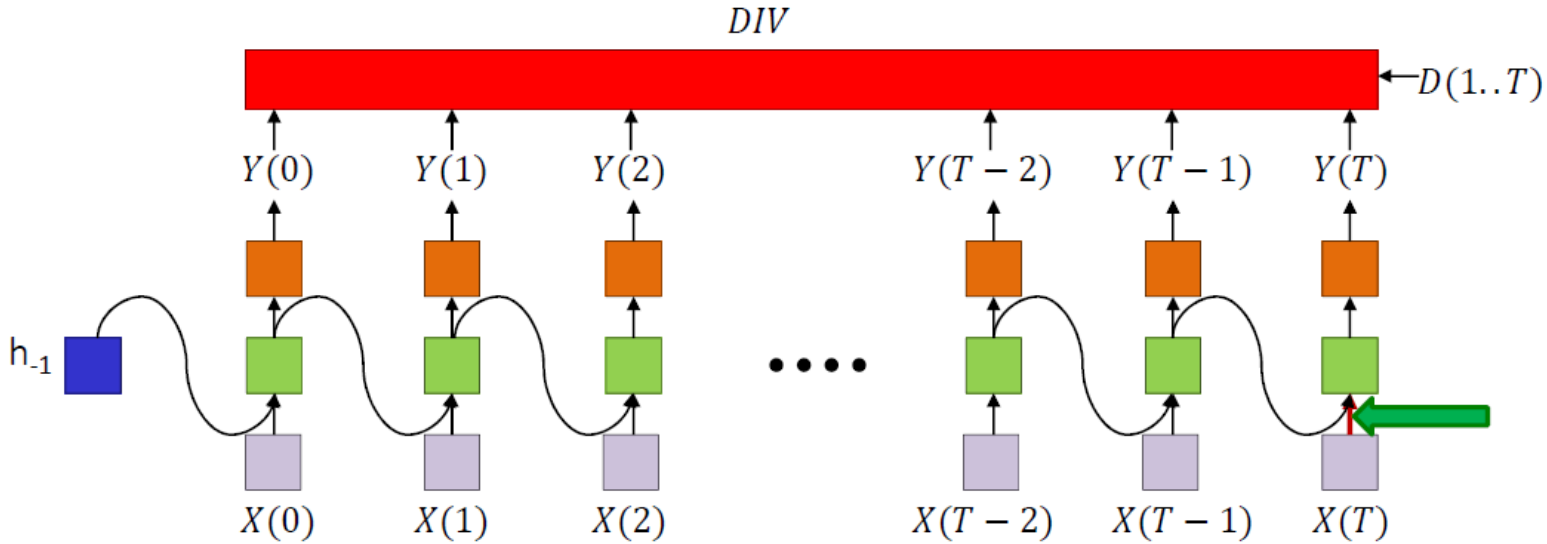


Back Propagation Through Time

- Backpropagation for RNN
  - We start from the final timestep T
  - We first compute $\frac{\partial Div}{\partial Y_T}$
  - Then $\frac{\partial Div}{\partial Z_T^{(2)}}$ and $\frac{\partial Div}{\partial h_T}$
  - Gradient of $W^{(2)}$
  - Then $\frac{\partial Div}{\partial Z_T^{(1)}}$
  - $\frac{\partial Div}{\partial W^{(1)}}$

$$h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$$

$$\frac{dDIV}{dw_{ij}^{(1)}} = \frac{dDIV}{dZ_j^{(1)}(T)} X_i(T)$$

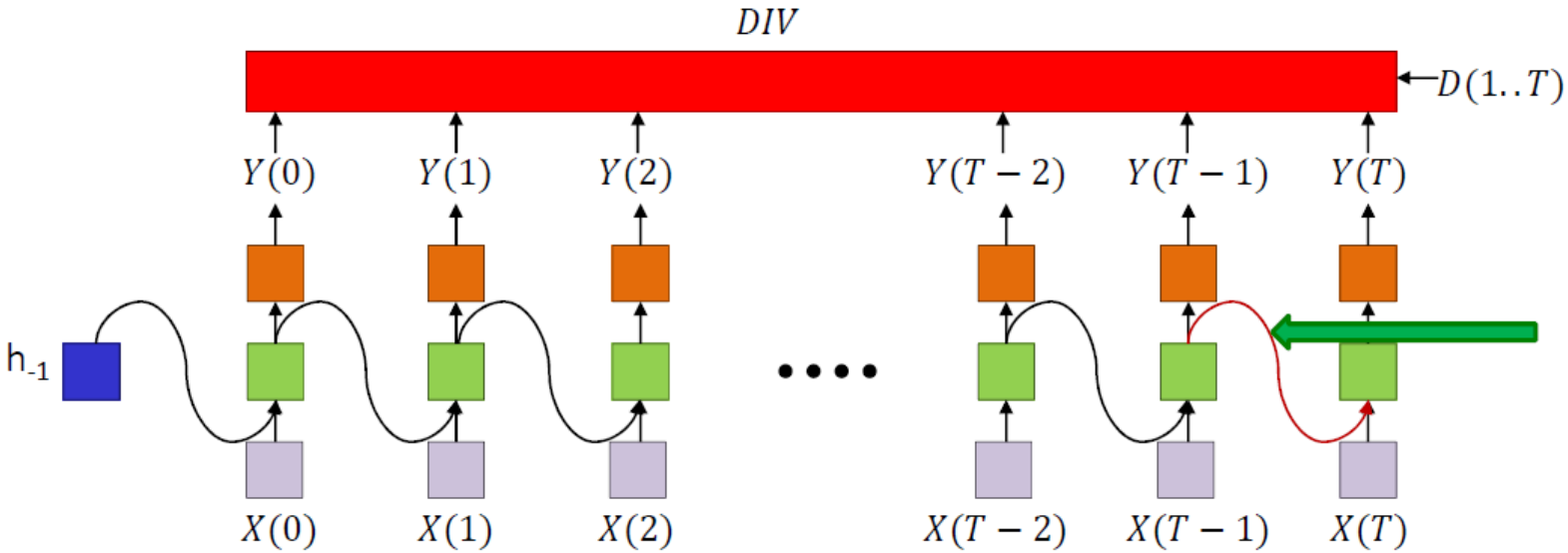


Back Propagation Through Time

- Backpropagation for RNN
  - We start from the final timestep T
  - We first compute $\frac{\partial Div}{\partial Y_T}$
  - Then $\frac{\partial Div}{\partial Z_T^{(2)}}$ and $\frac{\partial Div}{\partial h_T}$
  - Gradient of $W^{(2)}$
  - Then $\frac{\partial Div}{\partial Z_T^{(1)}}$
  - $\frac{\partial Div}{\partial W^{(1)}}$ and $\frac{\partial Div}{\partial W^{(11)}}$

$$h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$$

$$\frac{dDIV}{dw_{ij}^{(11)}} = \frac{dDIV}{dZ_j^{(1)}(T)} h_i(T-1)$$

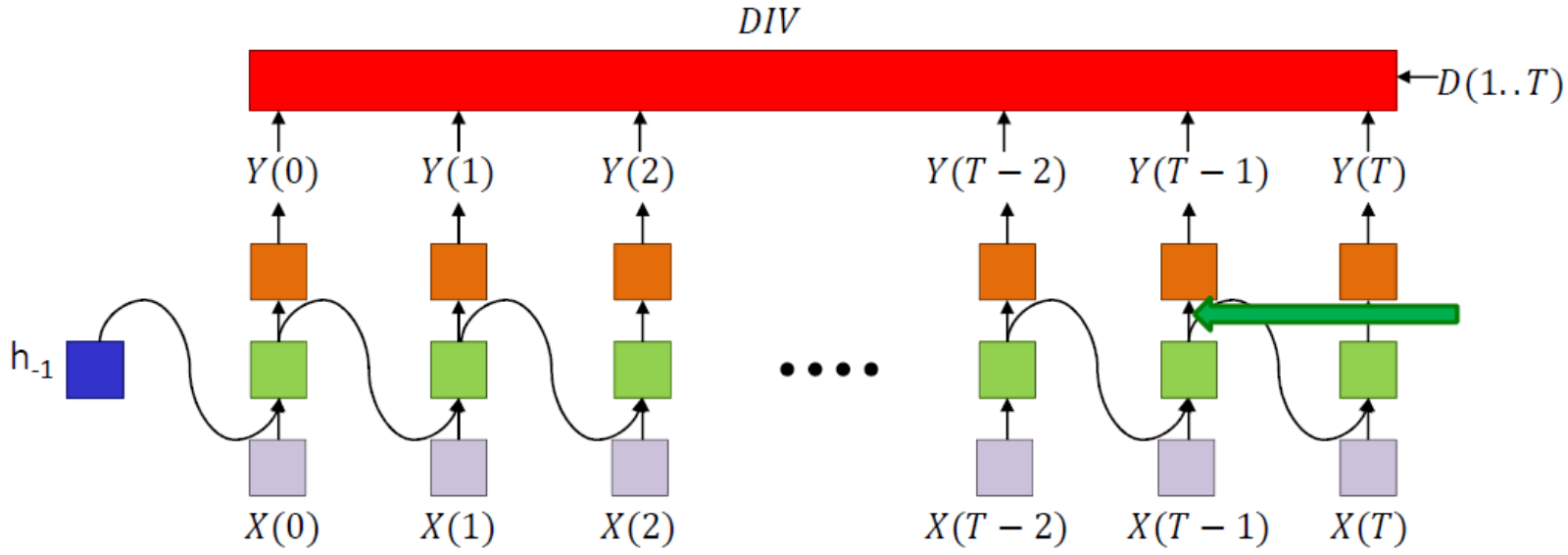


Back Propagation Through Time

- Backpropagation for RNN
  - We continue to timestep $T-1$
  - Compute $\frac{\partial Div}{\partial Y_{T-1}}$ and $\frac{\partial Div}{\partial Z^{(2)}_{T-1}}$

$$Y_t = f_2\left(Z_t^{(2)}\right)$$

$$\frac{dDIV}{dZ_i^{(2)}(T-1)} = \frac{dDIV}{dY_i(T-1)} \frac{dY_i(T-1)}{dZ_i^{(2)}(T-1)}$$



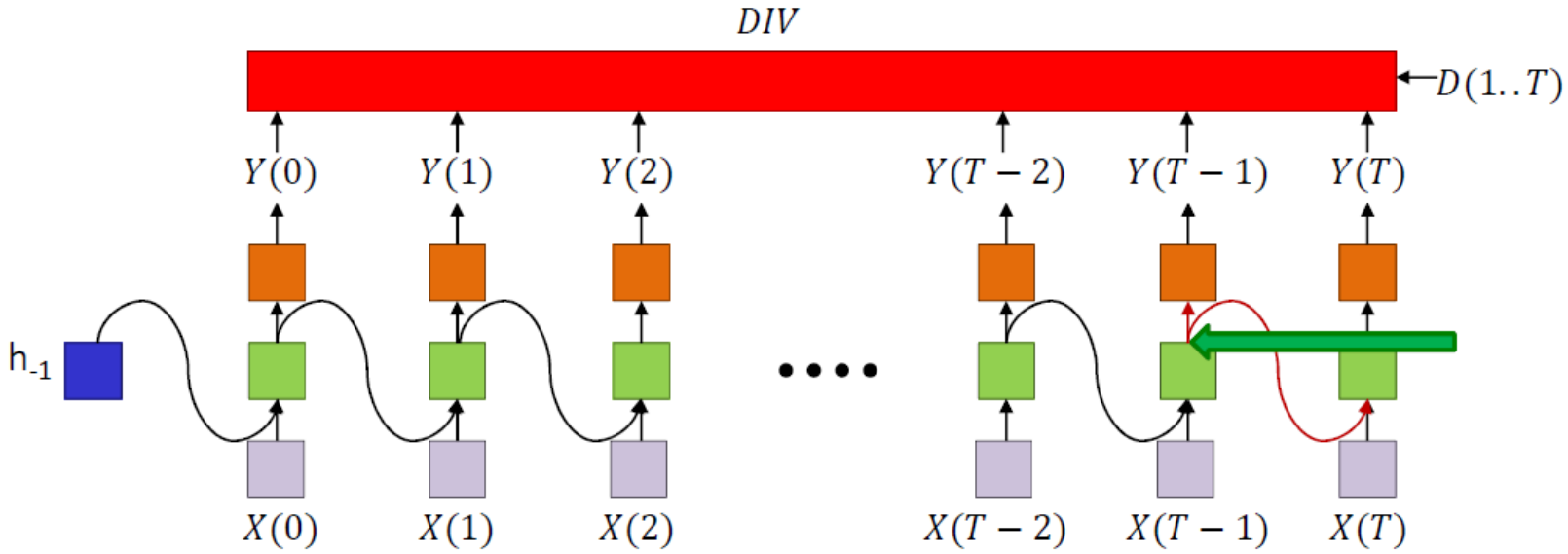
Back Propagation Through Time

- Backpropagation for RNN
  - We continue to timestep $T-1$
  - Compute $\frac{\partial Div}{\partial Y_{T-1}}$ and $\frac{\partial Div}{\partial Z^{(2)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial h_{T-1}}$
    - Note the addition!

$$h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$$

$$Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$$

$$\frac{dDIV}{dh_i(T-1)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T-1)} + \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(T)}$$



Back Propagation Through Time

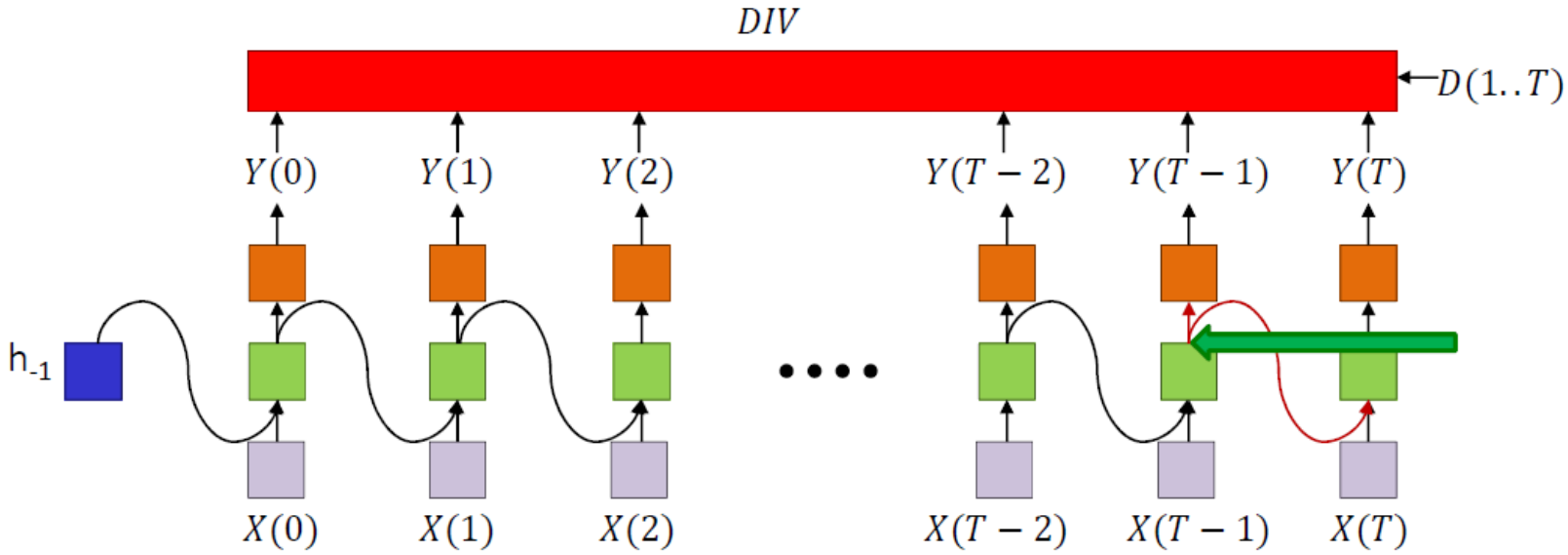
- Backpropagation for RNN
  - We continue to timestep $T-1$
  - Compute $\frac{\partial Div}{\partial Y_{T-1}}$ and $\frac{\partial Div}{\partial Z^{(2)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial h_{T-1}}$ and $\frac{\partial Div}{\partial W^{(2)}}$
    - Note the addition!

$$h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$$

$$Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$$

$$\frac{dDIV}{dh_i(T-1)} = \sum_j w^{(2)}_{ij} \frac{dDIV}{dZ^{(2)}_j(T-1)} + \sum_j w^{(11)}_{ij} \frac{dDIV}{dZ^{(1)}_j(T)}$$

$$\frac{dDIV}{dw^{(2)}_{ij}} \mathrel{+}= \frac{dDIV}{dZ^{(2)}_j(T-1)} h_i(T-1)$$

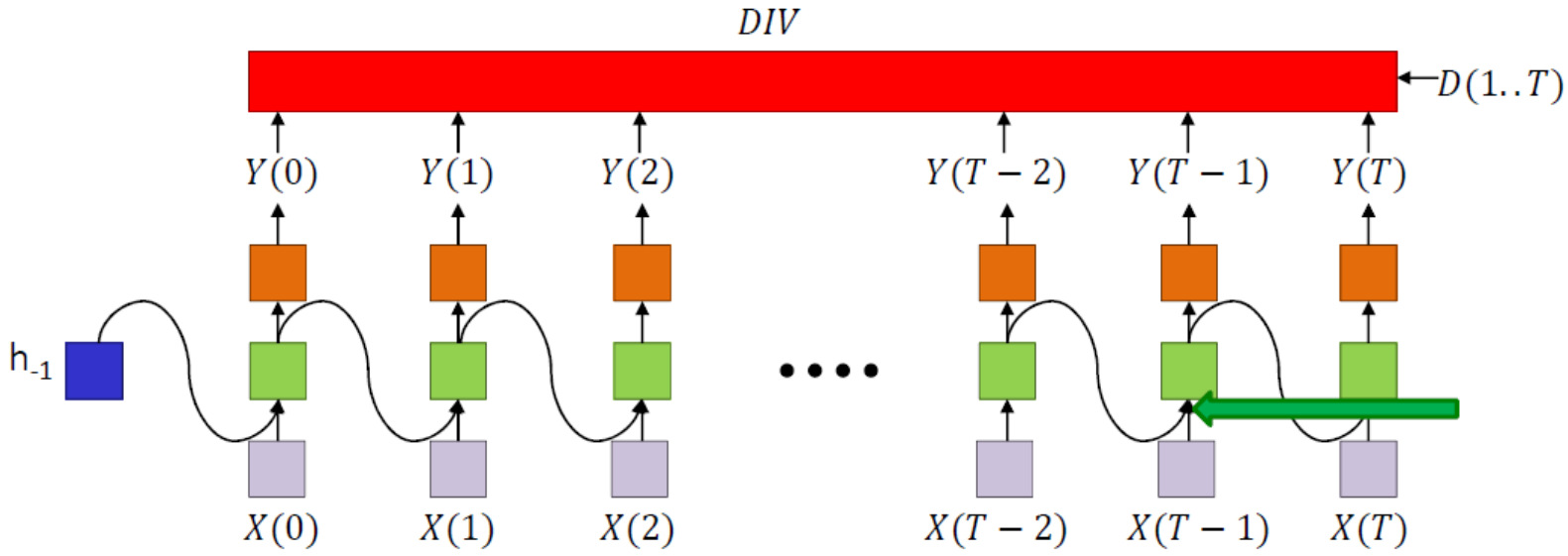


Back Propagation Through Time

- Backpropagation for RNN
  - We continue to timestep $T-1$
  - Compute $\frac{\partial Div}{\partial Y_{T-1}}$ and $\frac{\partial Div}{\partial Z^{(2)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial h_{T-1}}$ and $\frac{\partial Div}{\partial W^{(2)}}$
  - Compute $\frac{\partial Div}{\partial Z^{(1)}_{T-1}}$

$$h_t = f_1\left(Z_t^{(1)}\right)$$

$$\frac{dDIV}{dZ_i^{(1)}(T-1)} = \frac{dDIV}{dh_i(T-1)}\frac{dh_i(T-1)}{dZ_i^{(1)}(T-1)}$$

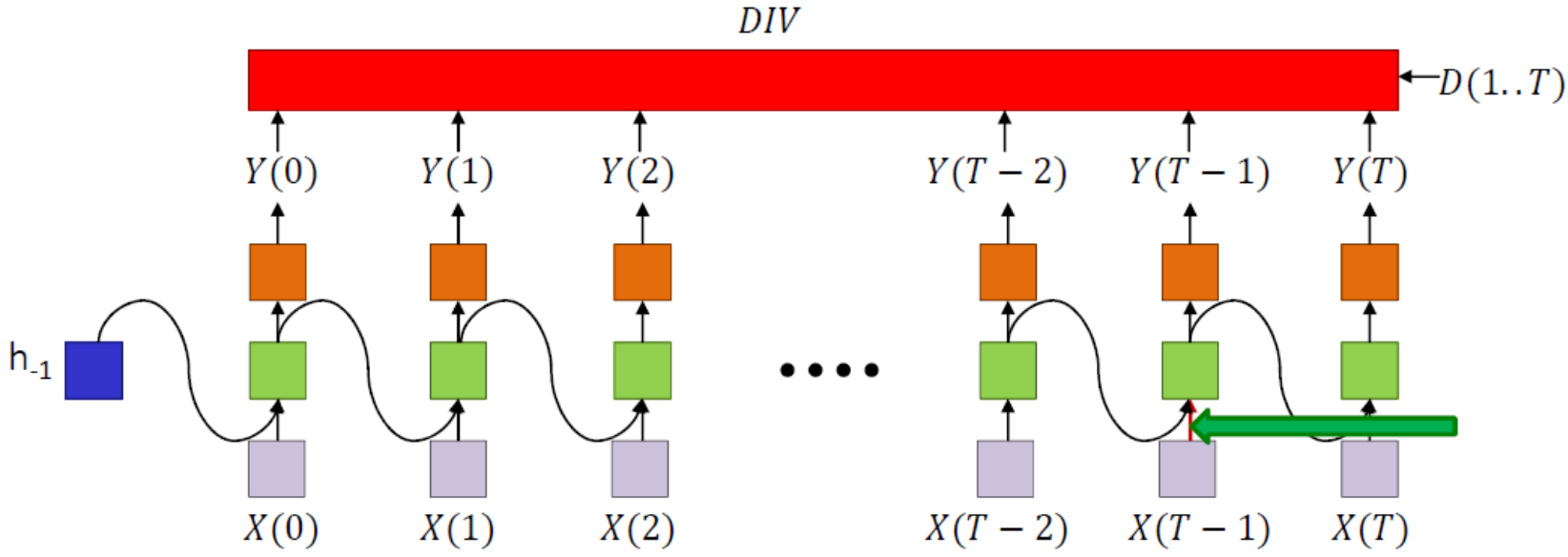


Back Propagation Through Time

- Backpropagation for RNN
  - We continue to timestep $T-1$
  - Compute $\frac{\partial Div}{\partial Y_{T-1}}$ and $\frac{\partial Div}{\partial Z^{(2)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial h_{T-1}}$ and $\frac{\partial Div}{\partial W^{(2)}}$
  - Compute $\frac{\partial Div}{\partial Z^{(1)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial W^{(1)}}$
    - Note the addition!

$$h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$$

$$\frac{dDIV}{dw_{ij}^{(1)}} \mathrel{+}= \frac{dDIV}{dZ_j^{(1)}(T-1)} X_i(T-1)$$

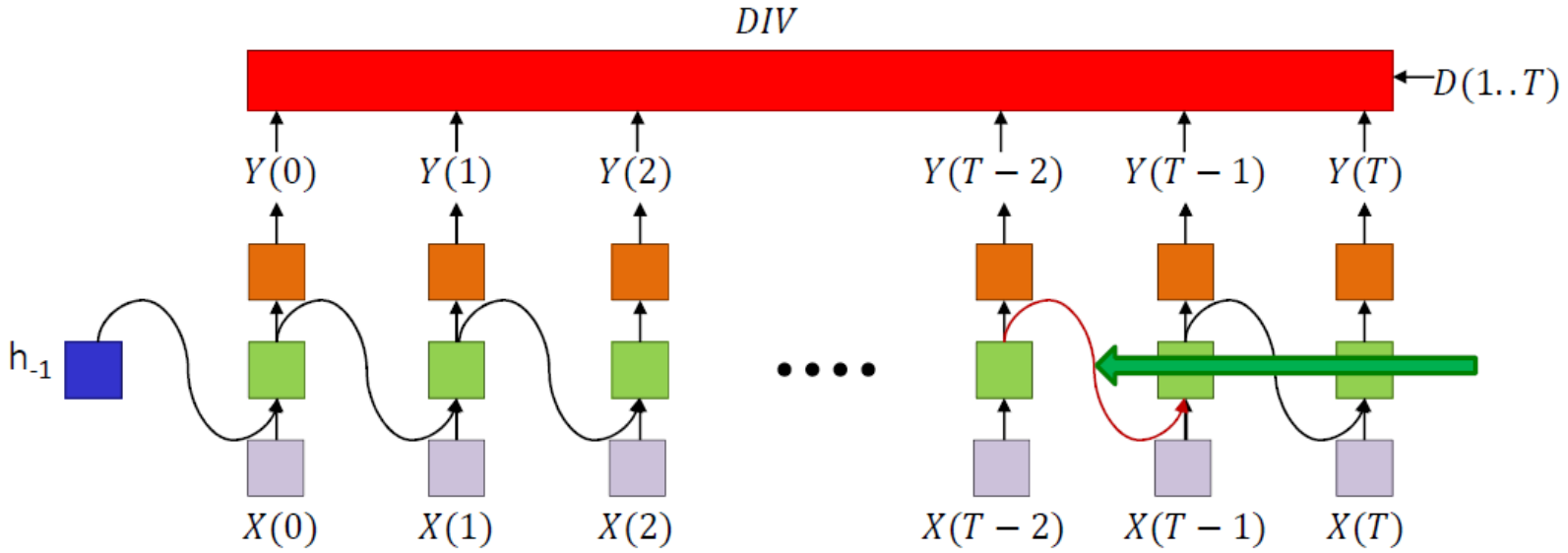


Back Propagation Through Time

- Backpropagation for RNN
  - We continue to timestep $T-1$
  - Compute $\frac{\partial Div}{\partial Y_{T-1}}$ and $\frac{\partial Div}{\partial Z^{(2)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial h_{T-1}}$ and $\frac{\partial Div}{\partial W^{(2)}}$
  - Compute $\frac{\partial Div}{\partial Z^{(1)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial W^{(1)}}$ and $\frac{\partial Div}{\partial W^{(11)}}$
    - Note the addition!

$$h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$$

$$\frac{dDIV}{dw^{(11)}_{ij}} \mathrel{+}= \frac{dDIV}{dZ^{(1)}_j(T-1)} h_i(T-2)$$

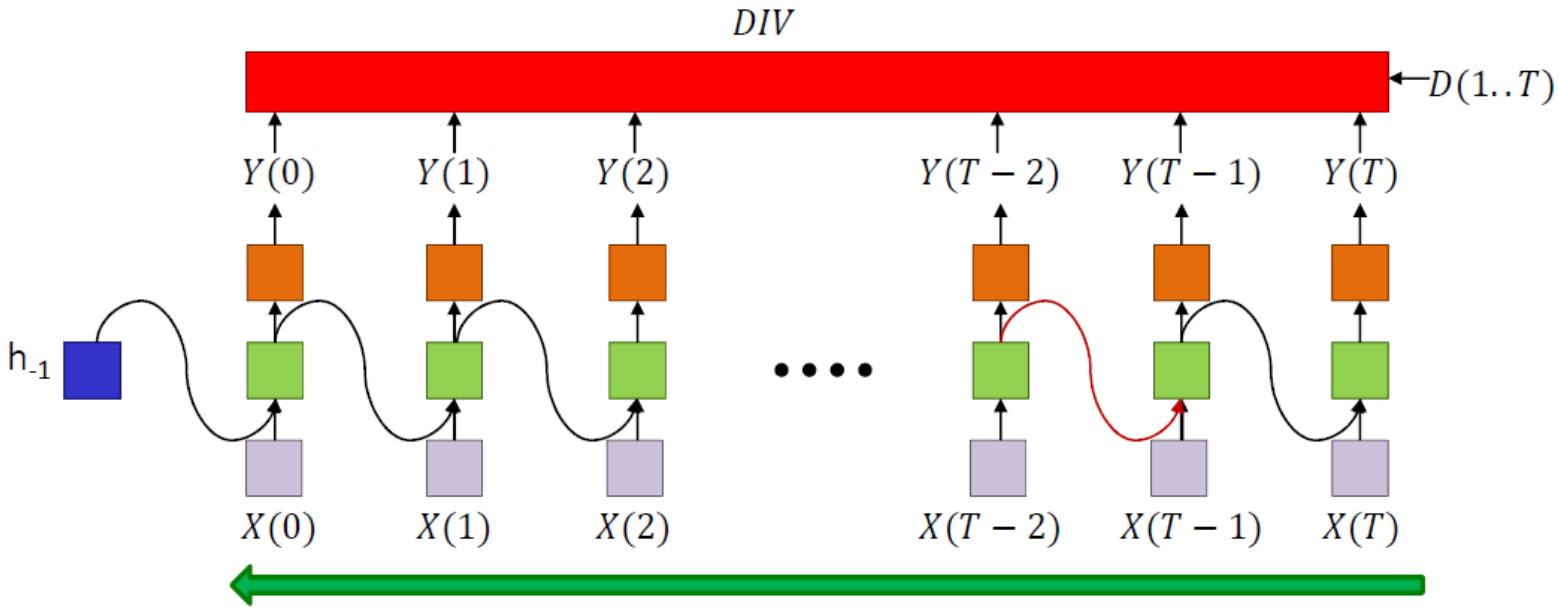


Back Propagation Through Time

- Backpropagation for RNN
  - We continue to timestep $T-1$
  - Compute $\frac{\partial Div}{\partial Y_{T-1}}$ and $\frac{\partial Div}{\partial Z^{(2)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial h_{T-1}}$ and $\frac{\partial Div}{\partial W^{(2)}}$
  - Compute $\frac{\partial Div}{\partial Z^{(1)}_{T-1}}$
  - Then $\frac{\partial Div}{\partial W^{(1)}}$ and $\frac{\partial Div}{\partial W^{(11)}}$
  - Until $\frac{\partial Div}{\partial h_{-1}}$
    - Short for BPTT

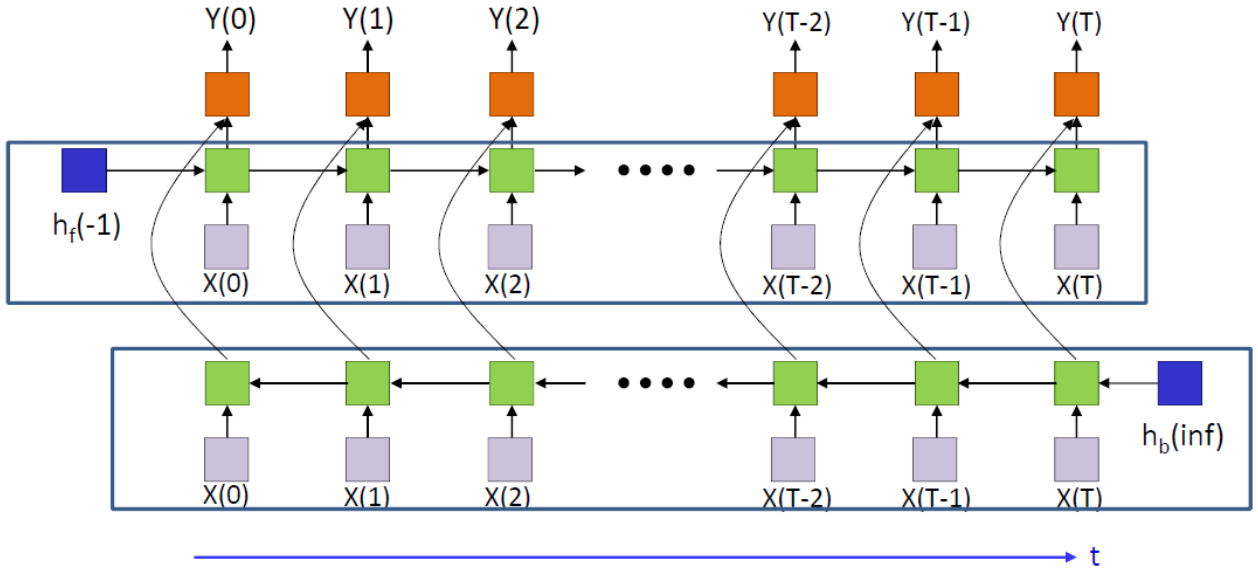
$$h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$$

$$Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$$



Extension

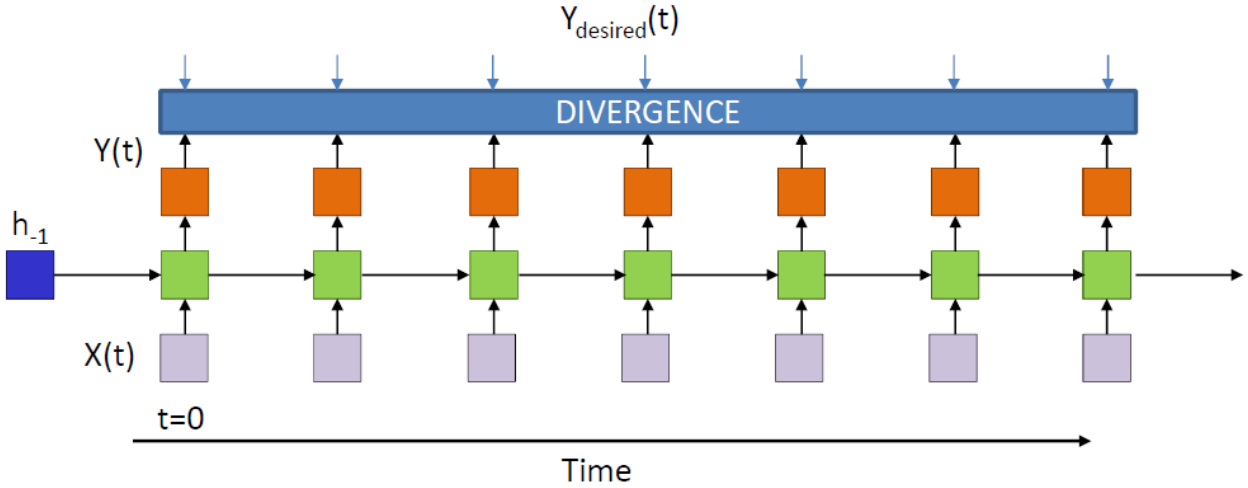
- In a standard RNN, Y_t only captures previous inputs
  - What if we want Y_t to handle the entire inputs?
- Bidirectional RNN
  - An RNN for forward dependencies
    - $t = 0 \ldots T$
  - An RNN for backward dependencies
    - $t = T \ldots 0$
  - $Y_t = f_2\left(h_t^f, h_t^b; \theta\right)$
  - BPTT for bidirectional RNN?
    - $\partial Div/\partial Y_t$ for all t
    - $t = T \ldots 0$ for h_f
    - $t = 0 \ldots T$ for h_b

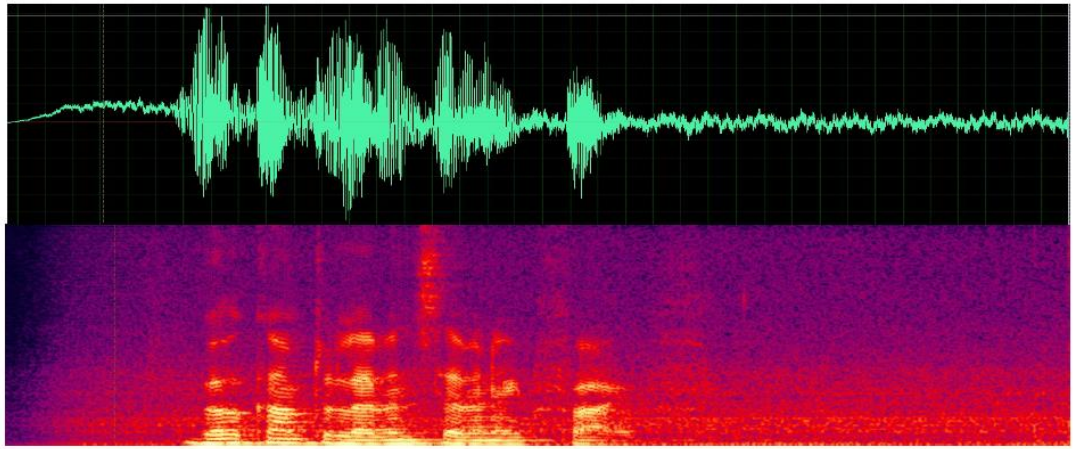


Extension

- Finding the “Welcome”
  - Input data $X_1 \ldots X_L$, L may vary
  - Whether the voice contains “Welcome”
- RNN for sequence classification
  - $Y = \max_t Y_t$
  - $L(\theta) = cross\_entropy(Y, Y_{\text{desired}})$

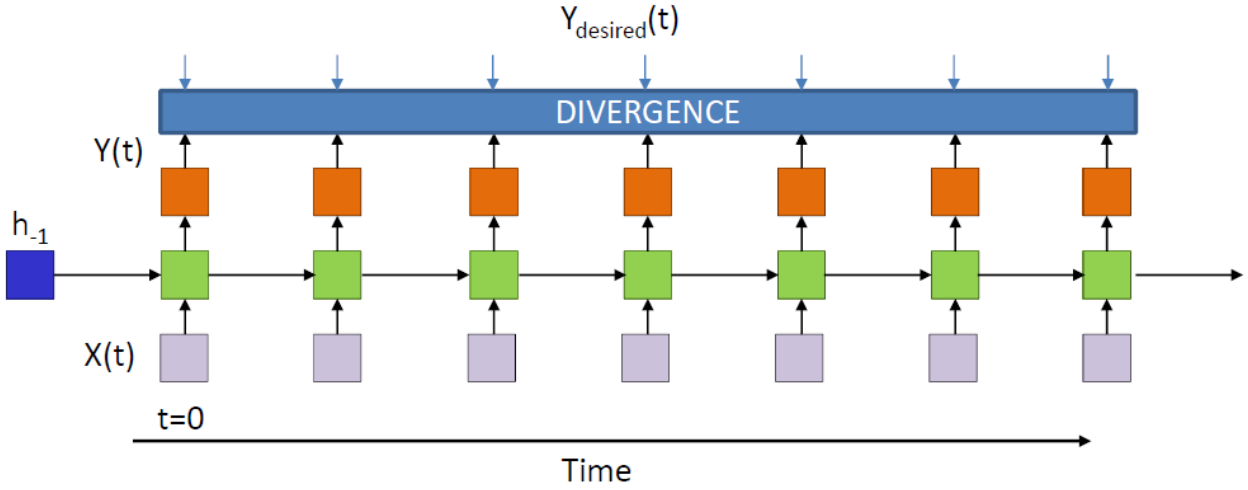
A 1-layer RNN can handle arbitrarily long sequence data





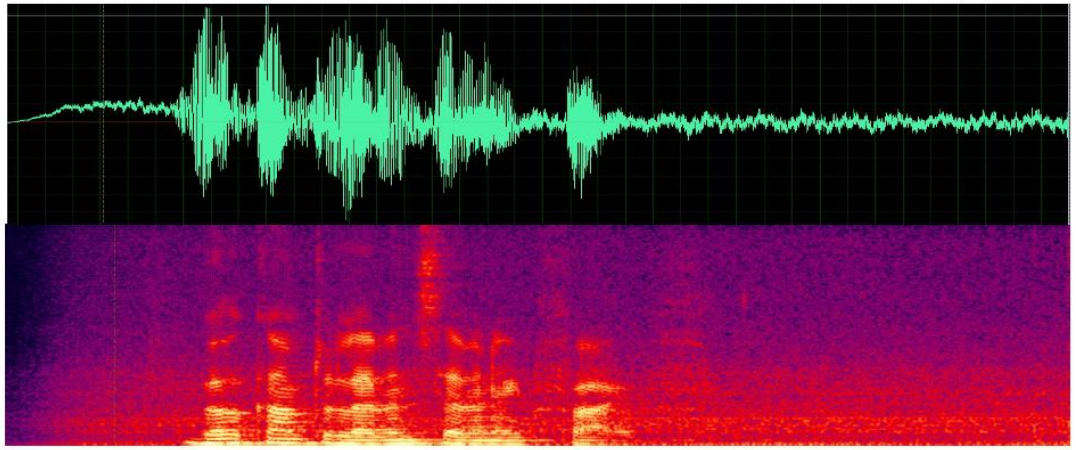
Extension

- Finding the “Welcome”
  - Input data $X_1 \ldots X_L$, L may vary
  - Whether the voice contains “Welcome”
- RNN for sequence classification
  - $Y = \max_t Y_t$
  - $L(\theta) = cross\_entropy(Y, Y_{\text{desired}})$



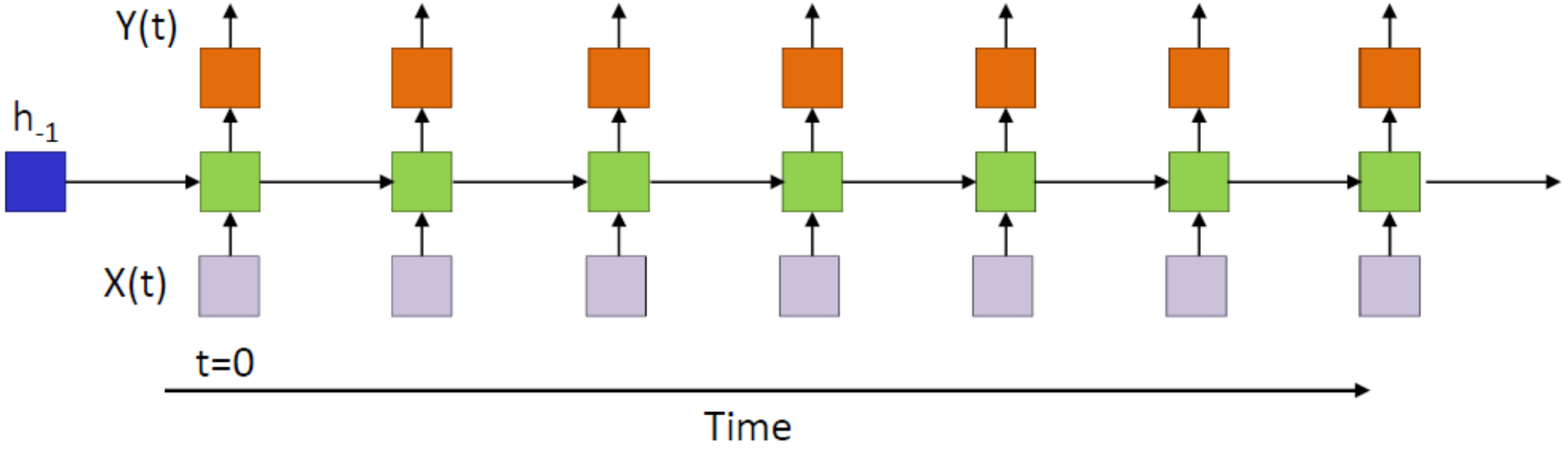
A 1-layer RNN can handle arbitrarily long sequence data

…… in theory!



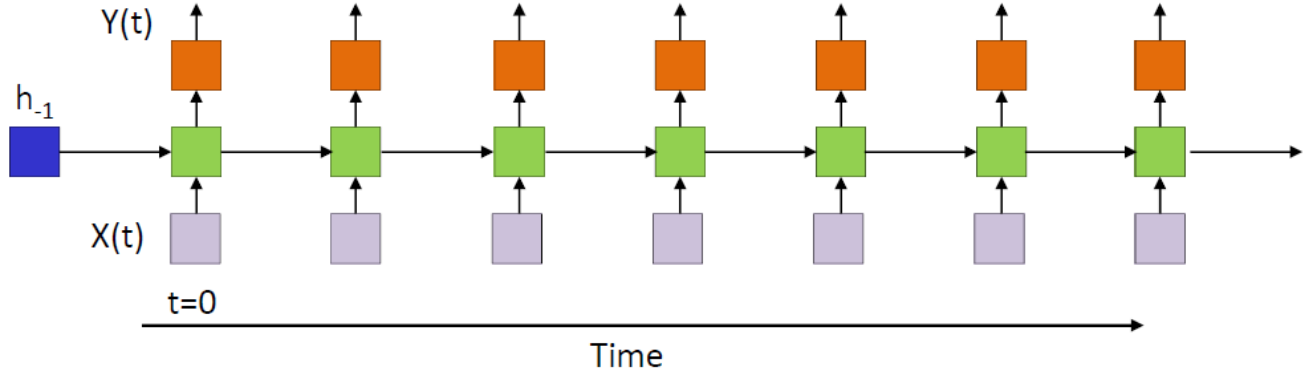
Practice Issues of RNN

- We start with a linear RNN
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = z_t$
  - All activations are identity functions
    - We will add activations back later



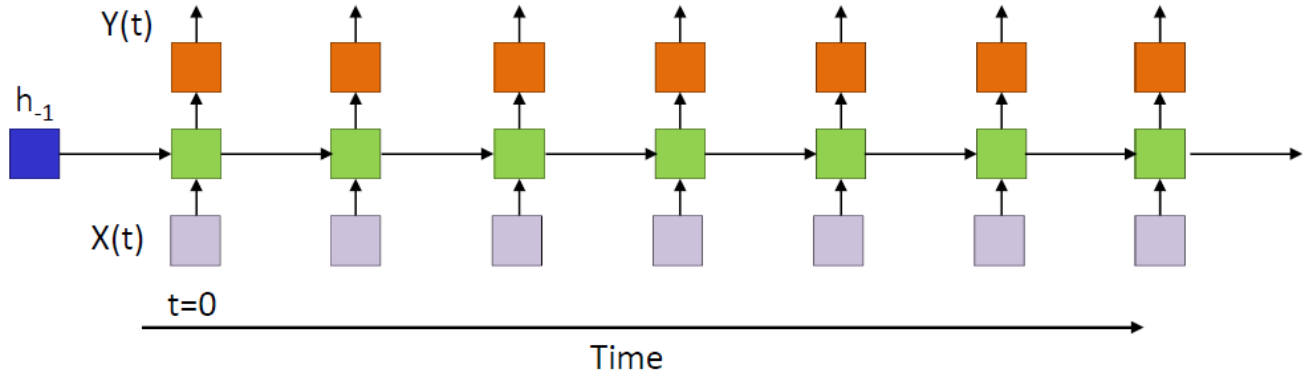
Practice Issues of RNN

- We start with a linear RNN
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = z_t$
- Let's expand the recursions
  - $h_k = W_h \cdot h_{k-1} + W_x \cdot X_k$
  - $= W_h^2 h_{k-2} + W_h W_x \cdot X_{k-1} + W_x \cdot X_k$
  - $= W_h^3 h_{k-3} + W_h^2 W_x \cdot X_{k-2} + W_h W_x \cdot X_{k-1} + W_x \cdot X_k$
  - $= W_h^{k+1} h_{-1} + \sum_{i=0}^{k} W_h^{k-i} W_x \cdot X_i$



Practice Issues of RNN

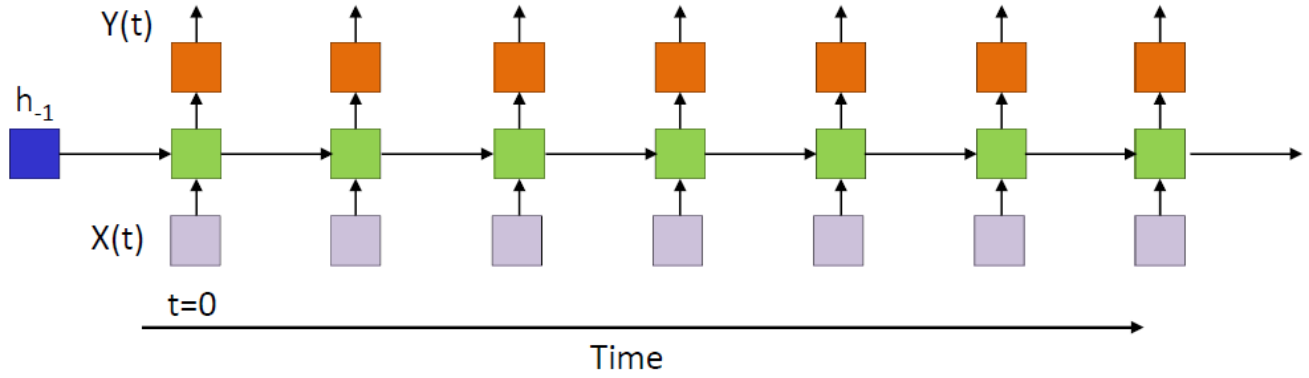
- We start with a linear RNN
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = z_t$



- Let's expand the recursions
  - $h_k = W_h \cdot h_{k-1} + W_x \cdot X_k$
  - $= W_h^2 h_{k-2} + W_h W_x \cdot X_{k-1} + W_x \cdot X_k$
  - $= W_h^3 h_{k-3} + W_h^2 W_x \cdot X_{k-2} + W_h W_x \cdot X_{k-1} + W_x \cdot X_k$
  - $= W_h^{k+1} h_{-1} + \sum_{i=0}^{k} W_h^{k-i} W_x \cdot X_i$
  - The coefficient of signal at position i is exponential over W_h
    - The dynamics of the system is highly depending on the maximum eigenvalue of W_h

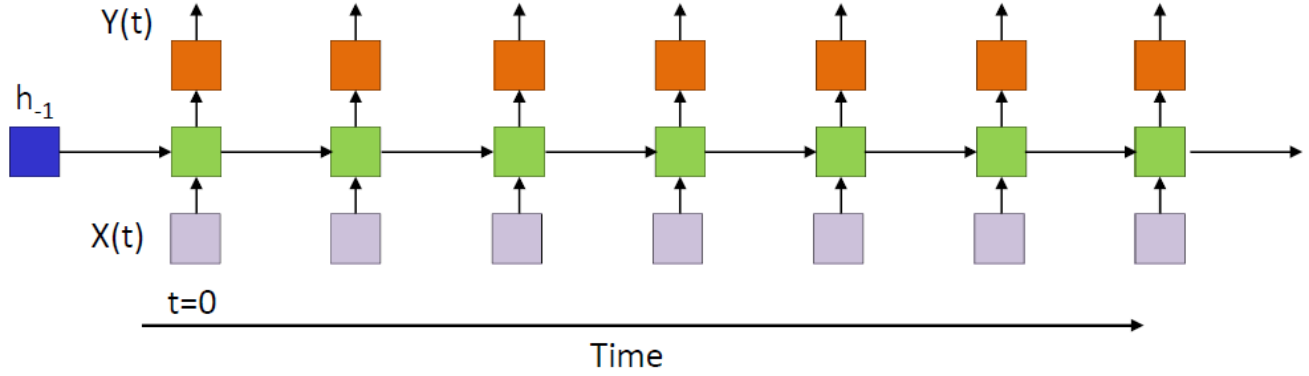
Practice Issues of RNN

- We start with a linear RNN
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = z_t$
- Let's expand the recursions
  - $h_k = W_h^{k+1} h_{-1} + \sum_{i=0}^{k} W_h^{k-i} W_x \cdot X_i$
  - The coefficient of signal at position i is exponential over W_h
    - If $|\lambda_{\max}| > 1$, the system explodes
    - If $|\lambda_{\max}| < 1$, the system cannot capture long-term dependencies
    - If $|\lambda_{\max}| = 1$, the second largest eigenvalue matters

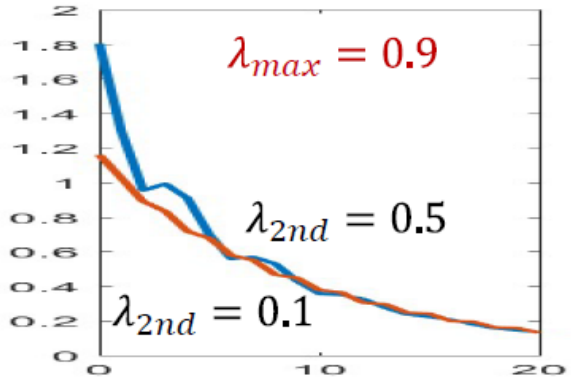


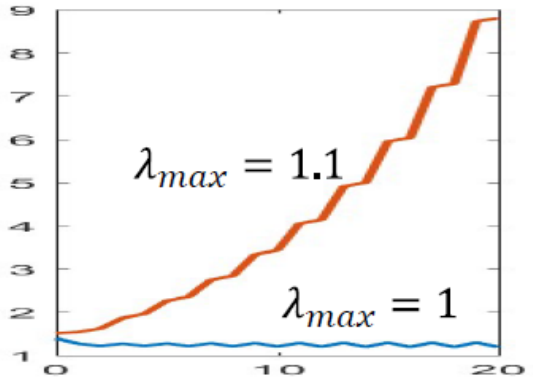
Practice Issues of RNN

- We start with a linear RNN
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = z_t$



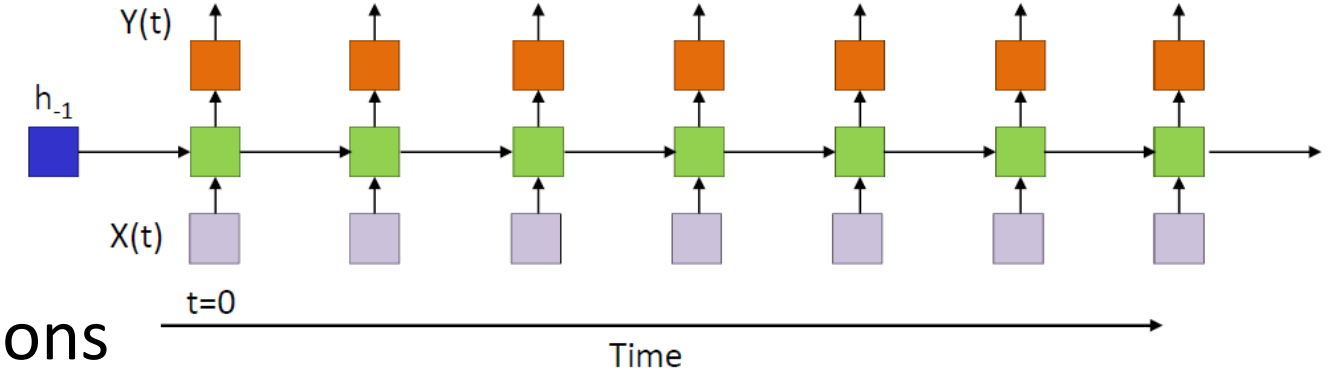
- Let's expand the recursions
  - $h_k = W_h^{k+1} h_{-1} + \sum_{i=0}^{k} W_h^{k-i} W_x \cdot X_i$
  - Let's consider the response to $h_{-1} = [1,1,1,1]$ at time t
    - We simulate $|W_h^{k+1} h_{-1}|$ with various eigenvalues

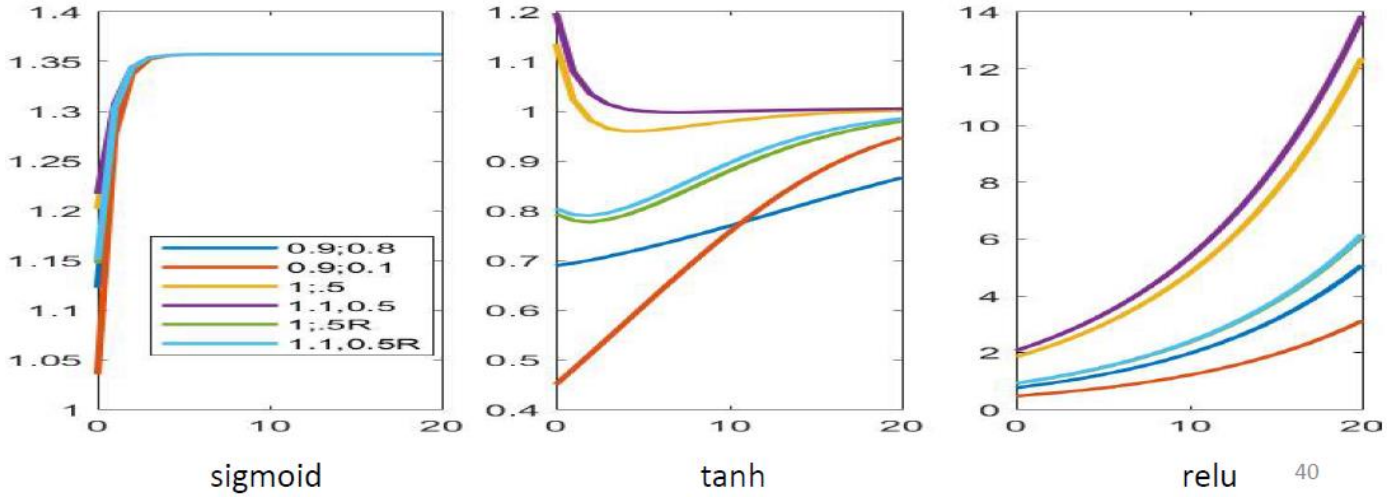




Practice Issues of RNN

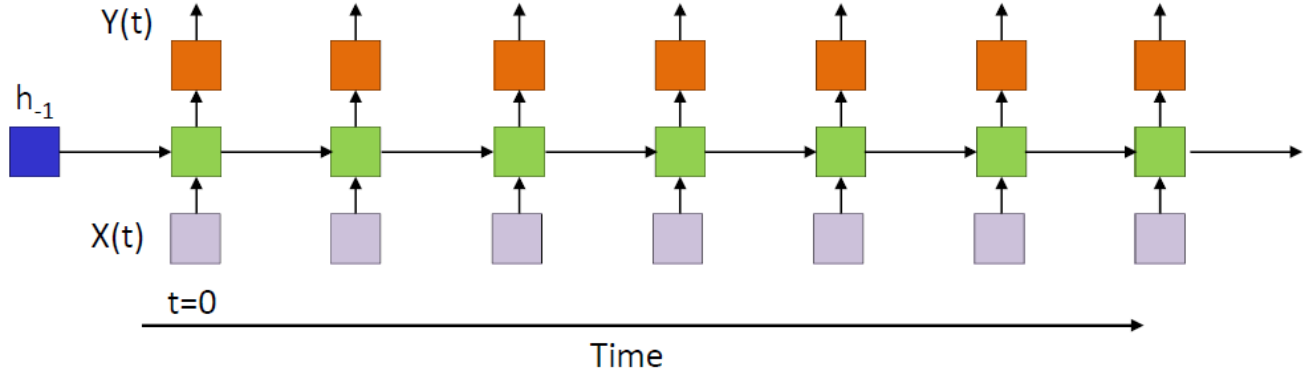
- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Simulation results with activations
  - A uniform start $h_{-1} = [1,1,1,1, \dots]/\sqrt{N}$
    - We simulate $|W_h^{k+1} h_{-1}|$ with various eigenvalues in W_h
    - Remark:
      - Tanh is preferred
      - … but still saturates
    - What about backward pass?



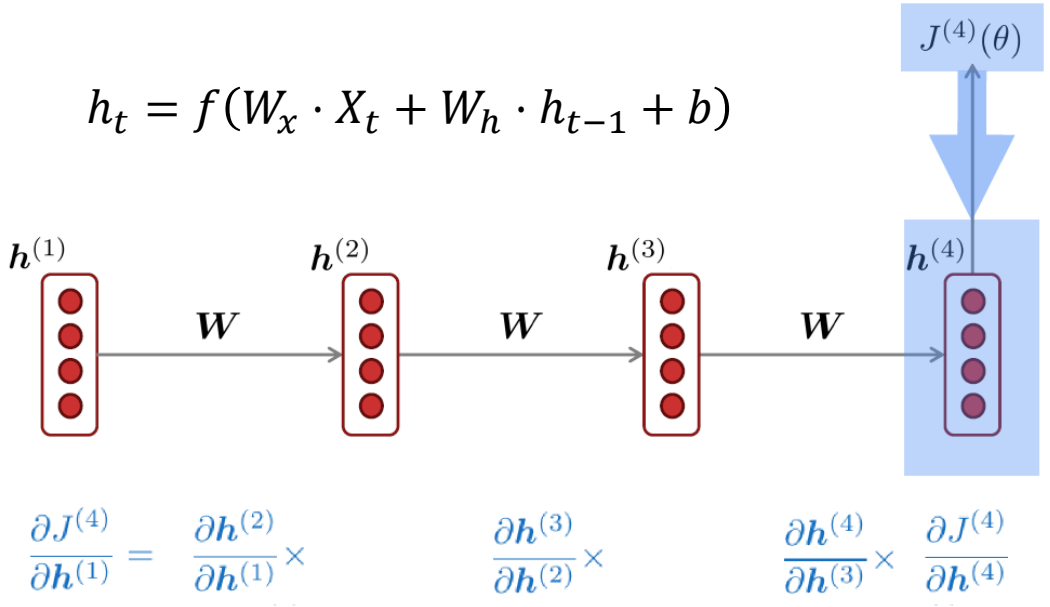


Practice Issues of RNN

- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- BPTT for RNN
  - Consider $J_k(\theta) = Div(Y_k, D_k)$
  - $\frac{\partial J_k}{\partial h_0} = \frac{\partial J_k}{\partial h_k} \prod_t \frac{\partial h_t}{\partial h_{t-1}}$
  - $\propto \prod_t W_h f'(Z_t) = W_h^k \prod_t f'(Z_t)$

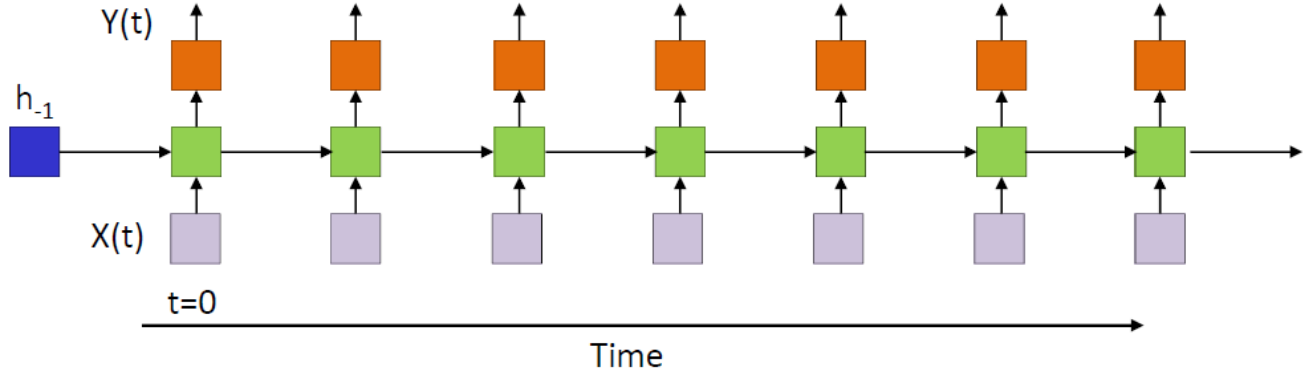


$$h_t = f(W_x \cdot X_t + W_h \cdot h_{t-1} + b)$$

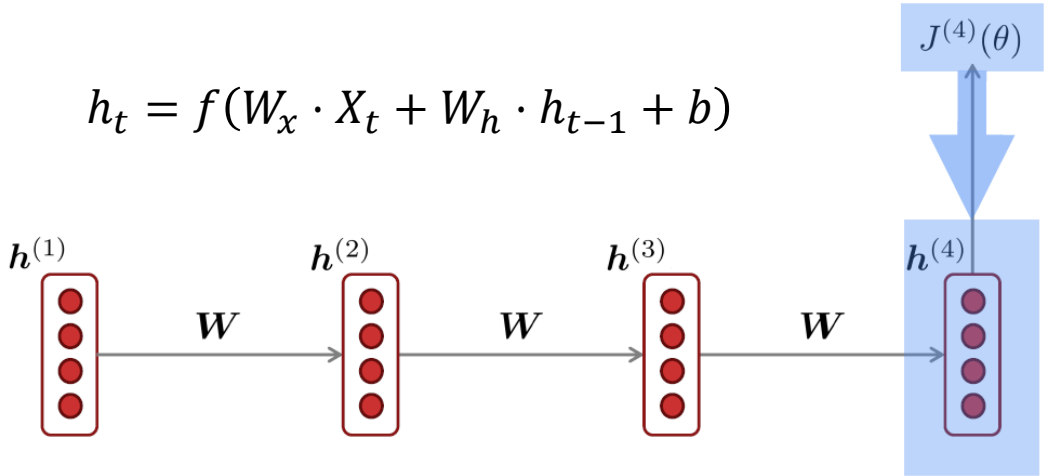


Practice Issues of RNN

- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- BPTT for RNN
  - Consider $J_k(\theta) = Div(Y_k, D_k)$
  - $\frac{\partial J_k}{\partial h_0} = \frac{\partial J_k}{\partial h_k} \prod_t \frac{\partial h_t}{\partial h_{t-1}}$
  - $\propto \prod_t W_h f'(Z_t) = W_h^k \prod_t f'(Z_t)$
  - Possible gradient explosion!

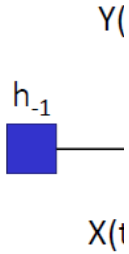


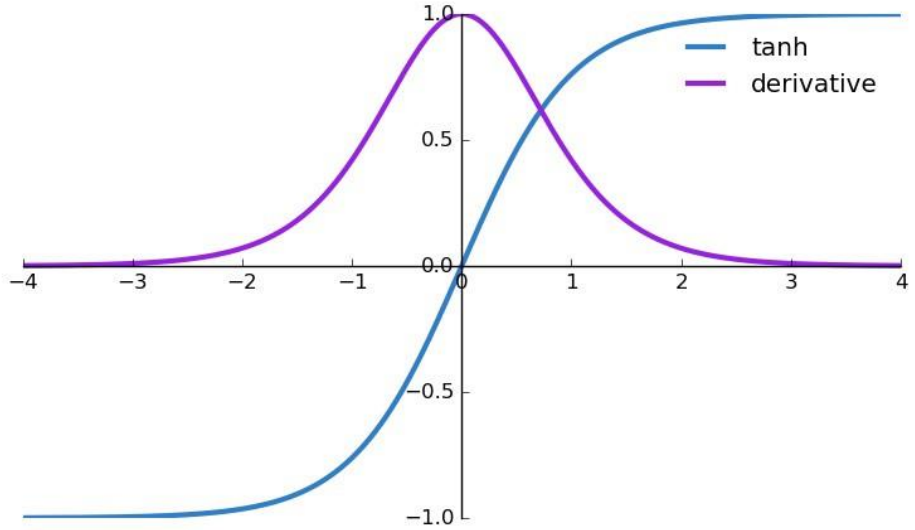
$$h_t = f(W_x \cdot X_t + W_h \cdot h_{t-1} + b)$$



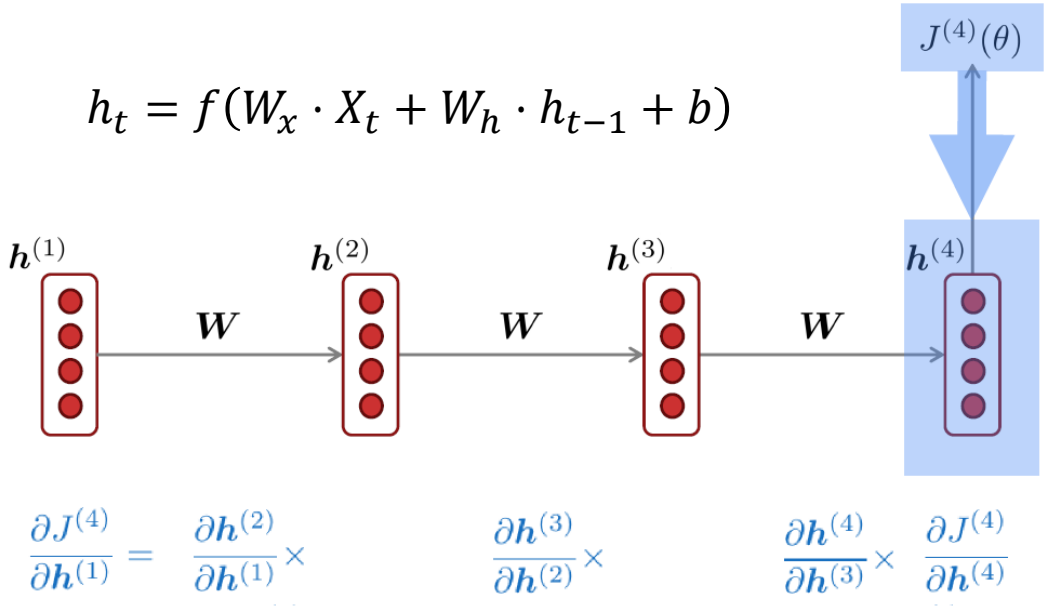
Practice Issues of RNN

- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- BPTT for RNN
  - Consider $J_k(\theta) = Div(Y_k, D_k)$
  - $\frac{\partial J_k}{\partial h_0} = \frac{\partial J_k}{\partial h_k} \prod_t \frac{\partial h_t}{\partial h_{t-1}}$
  - $\propto \prod_t W_h f'(Z_t) = W_h^k \prod_t f'(Z_t)$
  - f is activation (e.g., tanh)
    - $|f|_L \leq 1$
  - Gradient vanishing!
    - RNN "forgets" long-term past!



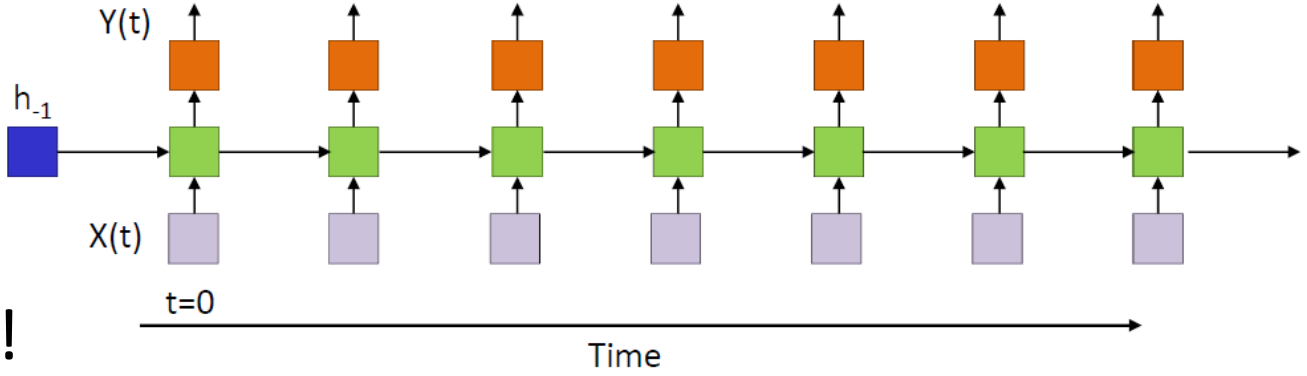


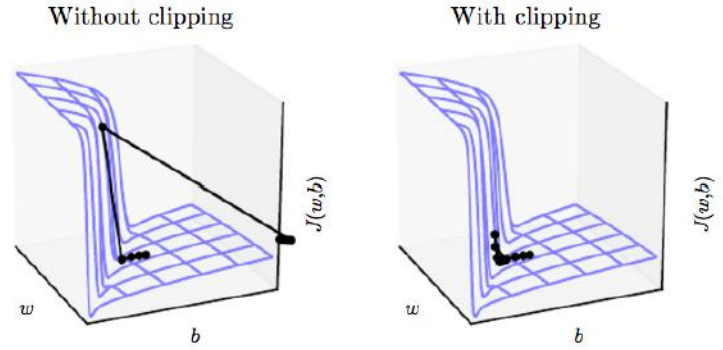
$$h_t = f(W_x \cdot X_t + W_h \cdot h_{t-1} + b)$$



Practice Issues of RNN

- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
    - Take a smaller step when gradient is too large
    - Gradient clipping is an important trick in practice



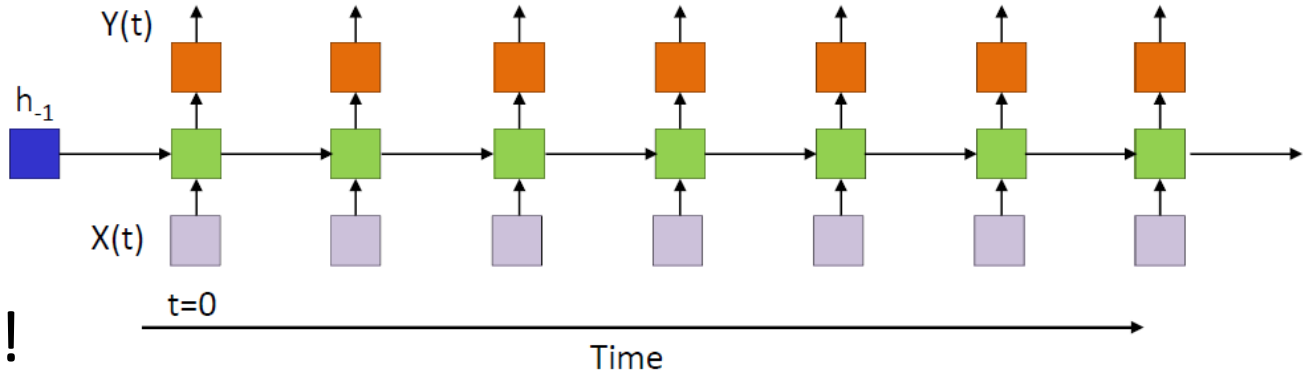


Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

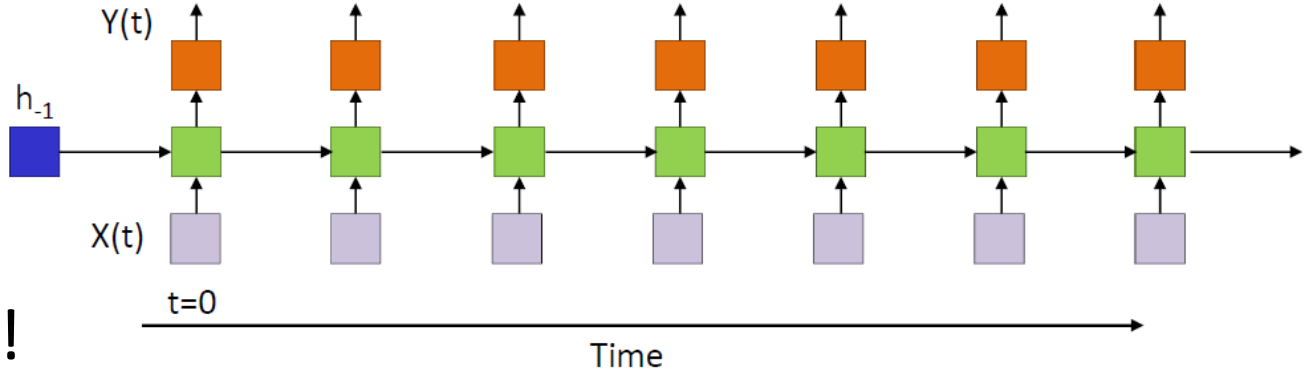
Practice Issues of RNN

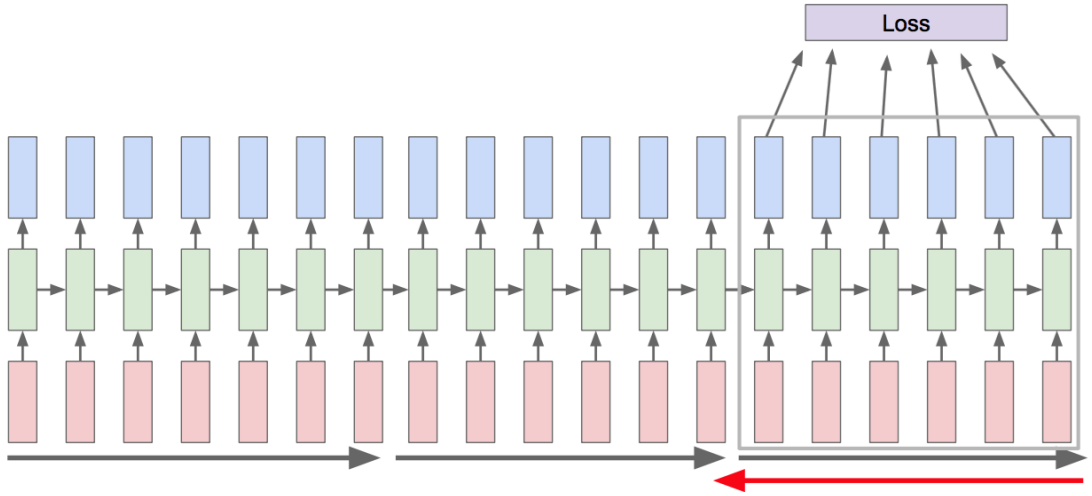
- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
  - Identity initialization
    - Make sure the weight matrix is initialized to have $\lambda_{\max} = 1$



Practice Issues of RNN

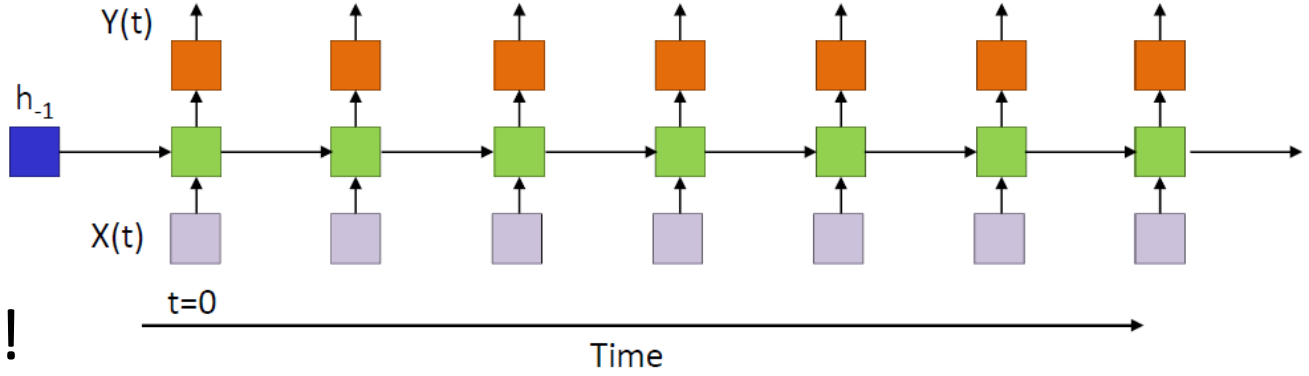
- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
  - Identity initialization
  - Truncated Backprop Through Time
    - Only backpropagate for a few timesteps

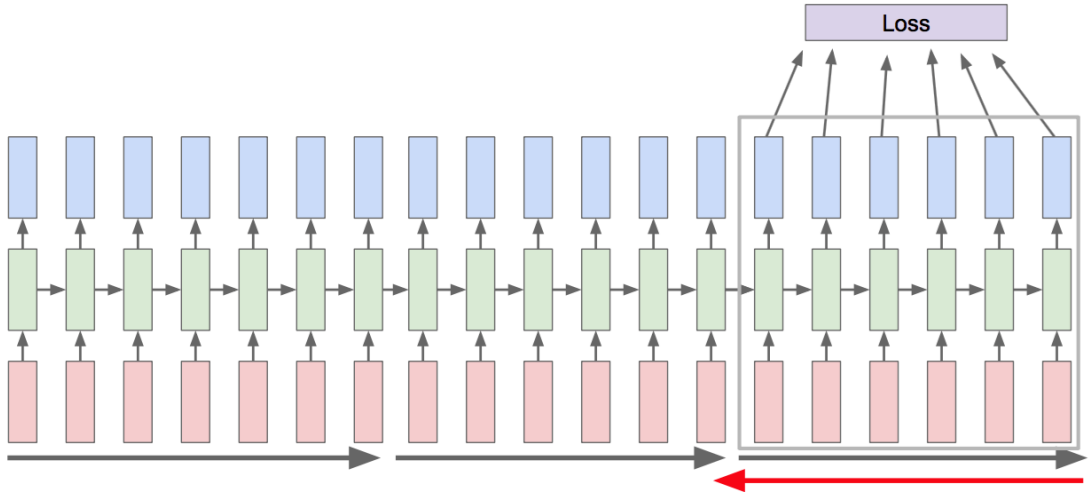




Practice Issues of RNN

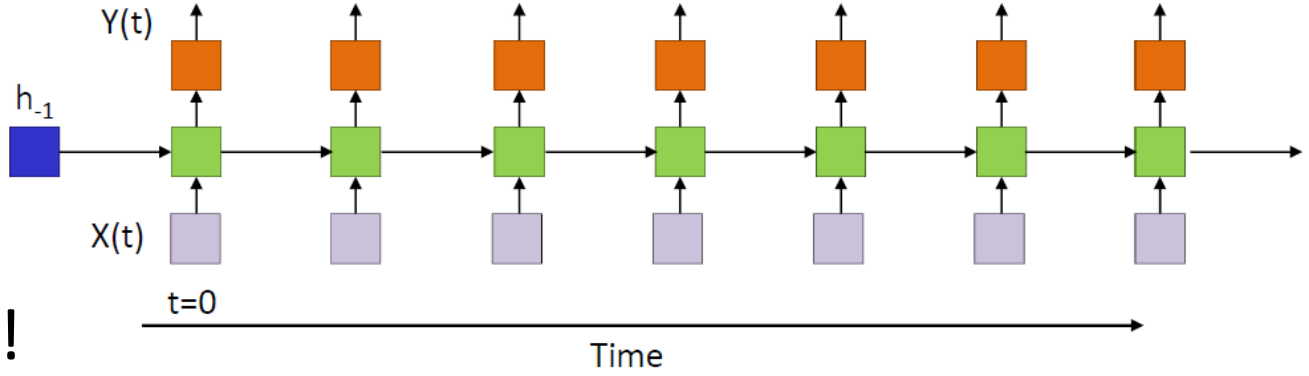
- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
  - Identity initialization
  - Truncated Backprop Through Time
    - Only backpropagate for a few timesteps
  - Gradient explosion is easy to solve

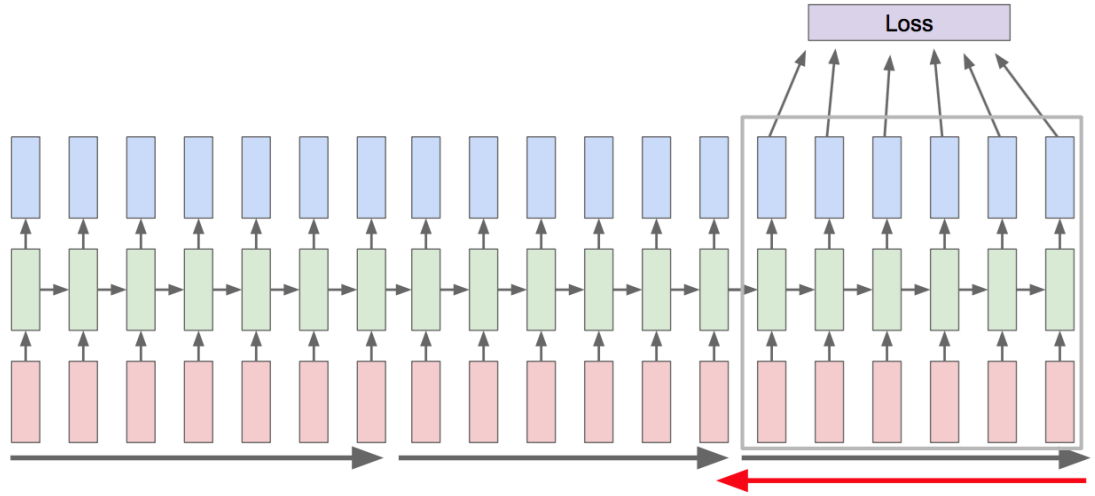




Practice Issues of RNN

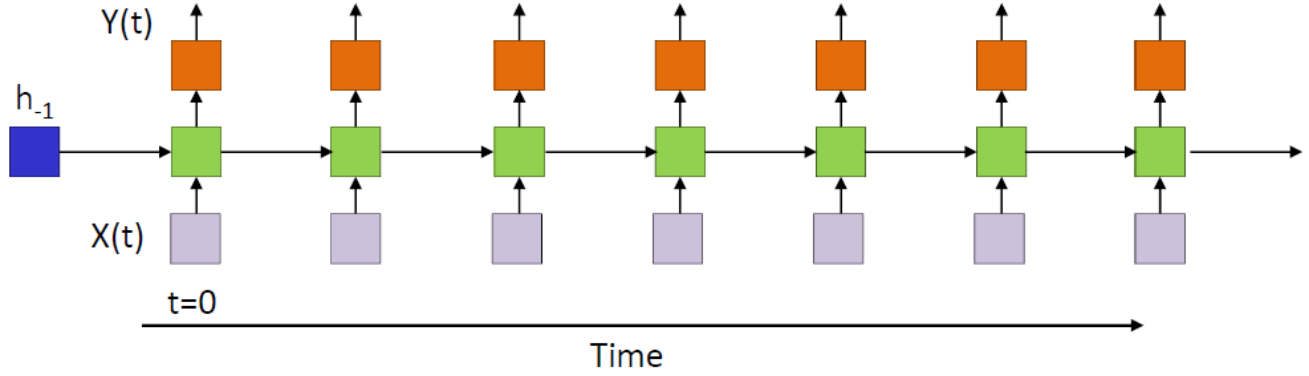
- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
  - Identity initialization
  - Truncated Backprop Through Time
- What about memory?
  - *RNN forgets past due to activation*



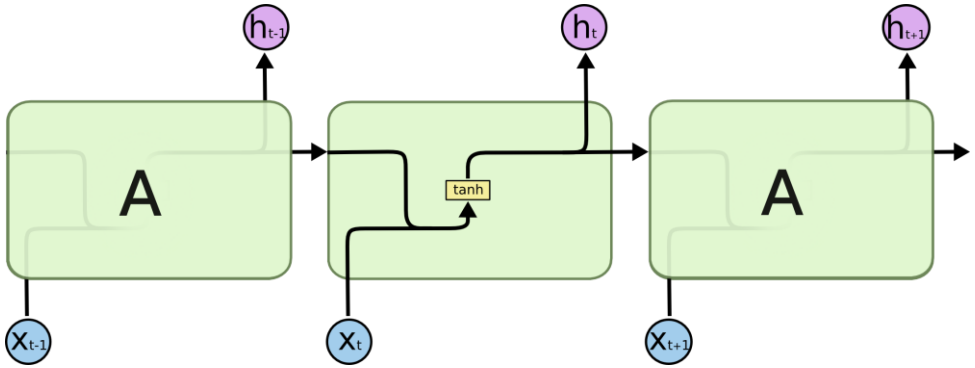


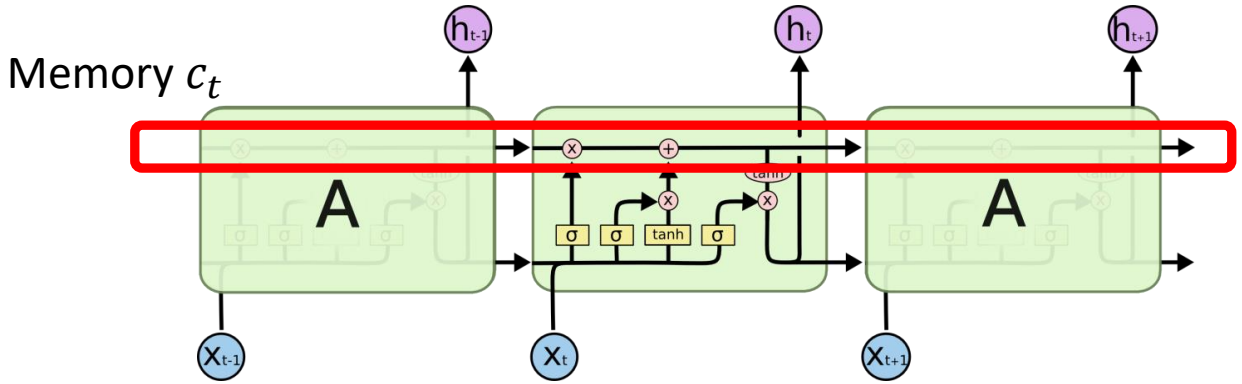
Preserve Long-Term Memory

- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$



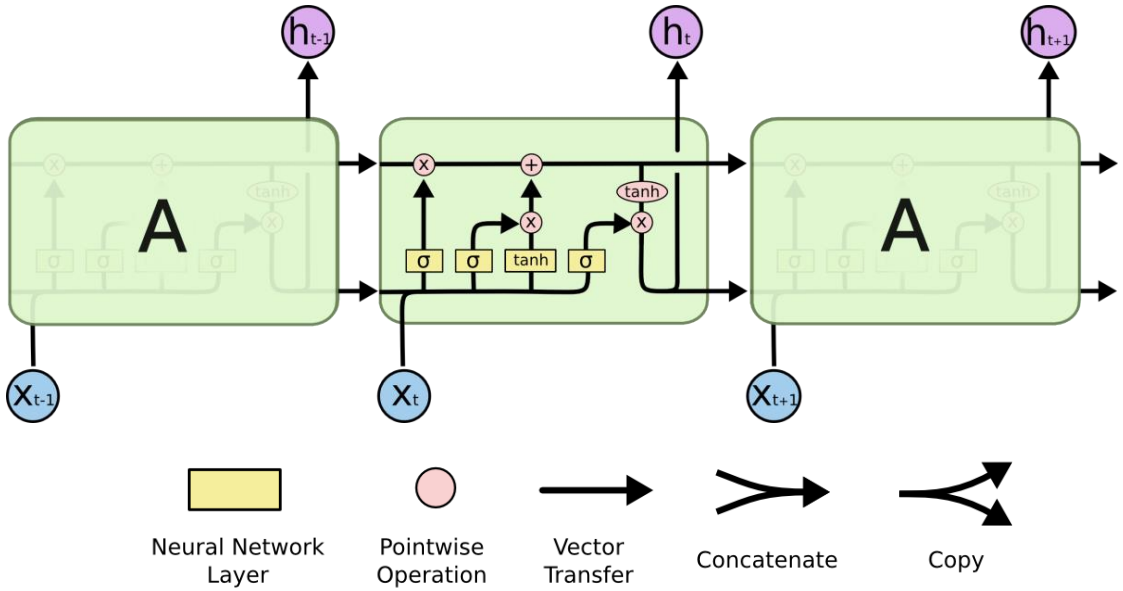
- It is difficult for RNN to preserve long-term memory
  - The hidden state h_t is constantly being written (short-term memory)
  - Let's keep a separate cell for maintaining long-term memory





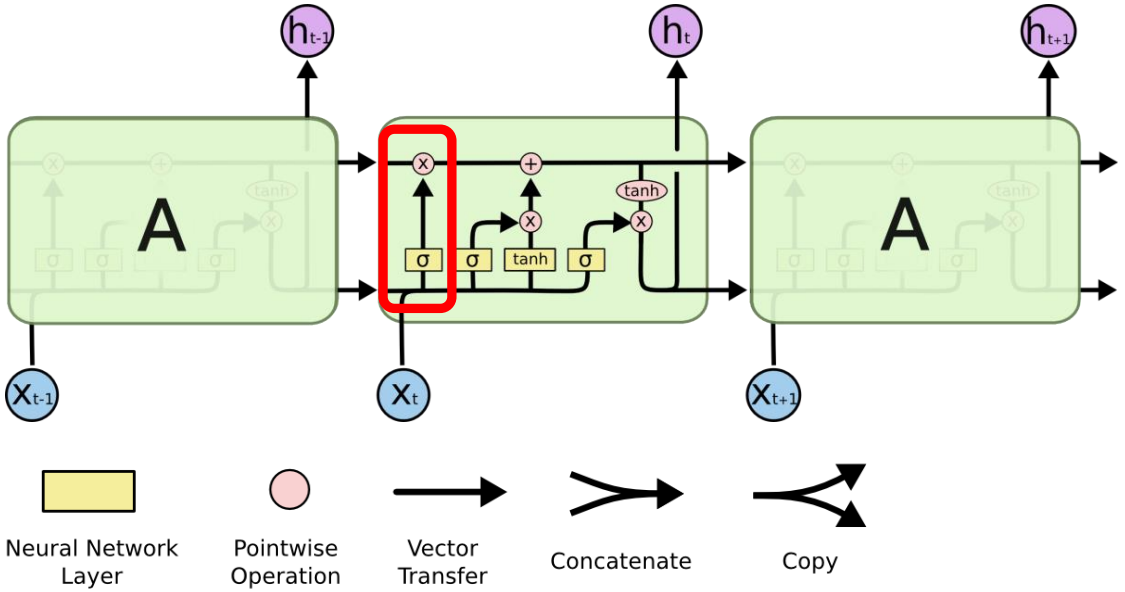
Long Short-Term Memory Network

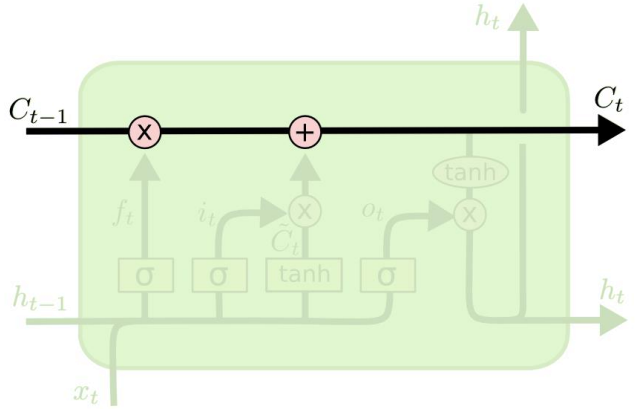
- LSTM (Hochreiter & Schmidhuber, 1997)
  - A special RNN architecture for learning long-term dependencies
  - σ: layer with sigmoid activation
  - Let's walk through the architecture
    - Diagrams from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

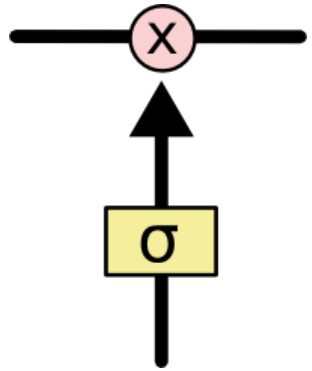


Long Short-Term Memory Network

- LSTM (Hochreiter & Schmidhuber, 1997)
  - Core idea: maintain separate state h_t and cell c_t (memory)
  - h_t: full update every iteration
  - c_t: only partially updated through gates
    - A σ layer outputs "importance" (0~1) for each entry and only modify those entries in c_t

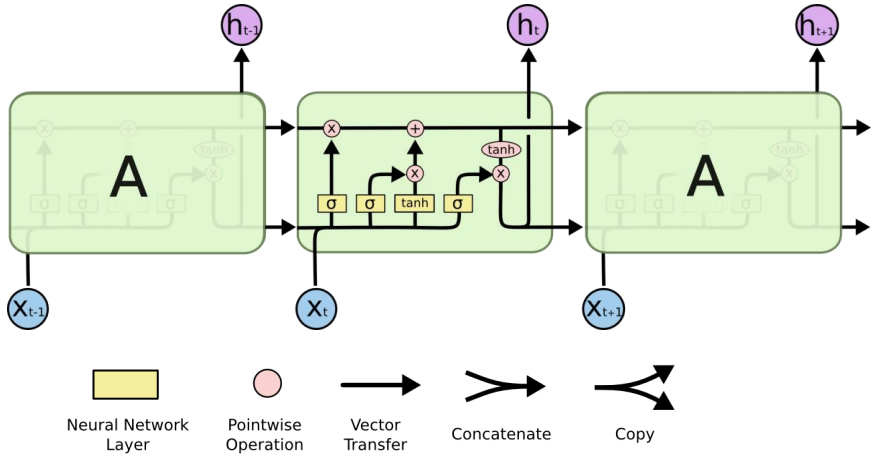


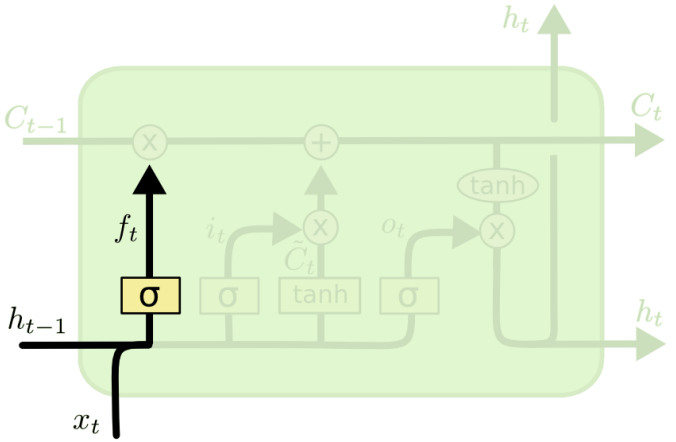




Long Short-Term Memory Network

- LSTM (Hochreiter & Schmidhuber, 1997)
  - Forget gate f_t
  - f_t outputs whether we want to "forget" things from c_t or carry it
    - Compute $c_{t-1} \odot f_t$ (element-wise)
    - $f_t(i) \to 0$: we want to forget $c_t(i)$
    - $f_t(i) \to 1$: we want to keep the information in $c_t(i)$

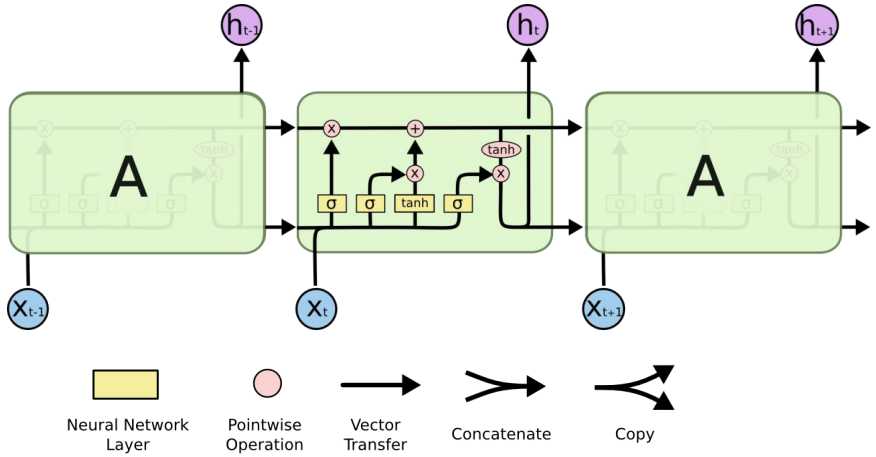


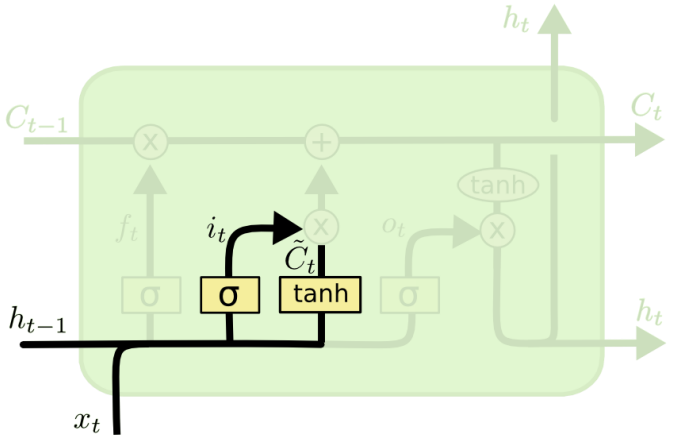


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Long Short-Term Memory Network

- LSTM (Hochreiter & Schmidhuber, 1997)
  - Input gate i_t
  - i_t extracts useful information from X_t to update memory
    - $\tilde{c}_t$: information from X_t to update memory (dimension projection)
    - i_t: which dimensions in the memory should be updated by X_t
      - $i_t(j) \rightarrow 1$: we want to keep the information in $\tilde{c}_t(j)$ to update memory
      - $i_t(j) \rightarrow 0$: $\tilde{c}_t(j)$ should not contribute to memory



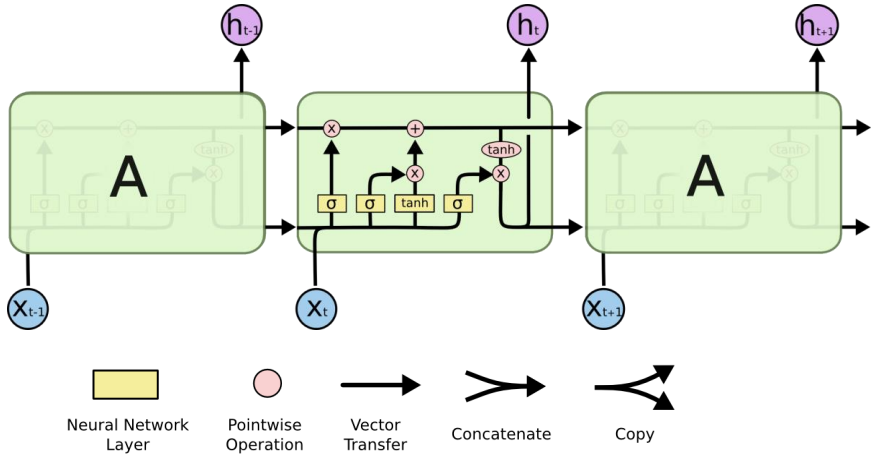


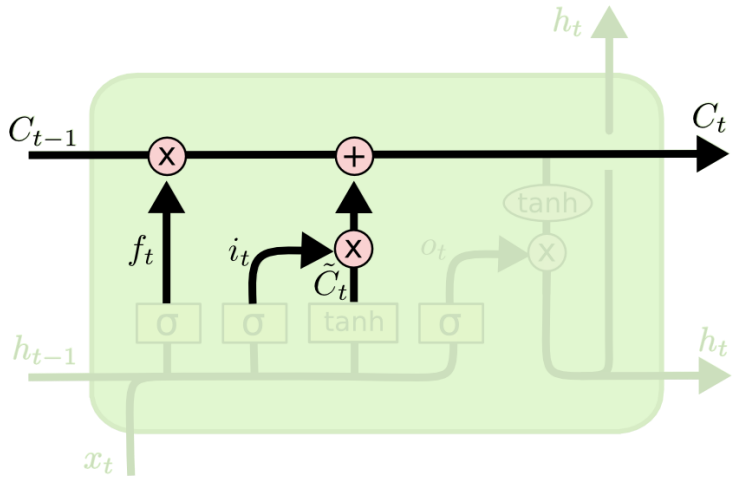
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Long Short-Term Memory Network

- LSTM (Hochreiter & Schmidhuber, 1997)
  - Memory update
    - $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
    - f_t forget gate; i_t input gate
    - $f_t \odot c_{t-1}$: drop useless information in old memory
    - $i_t \odot \tilde{c}_t$: add selected new information from current input

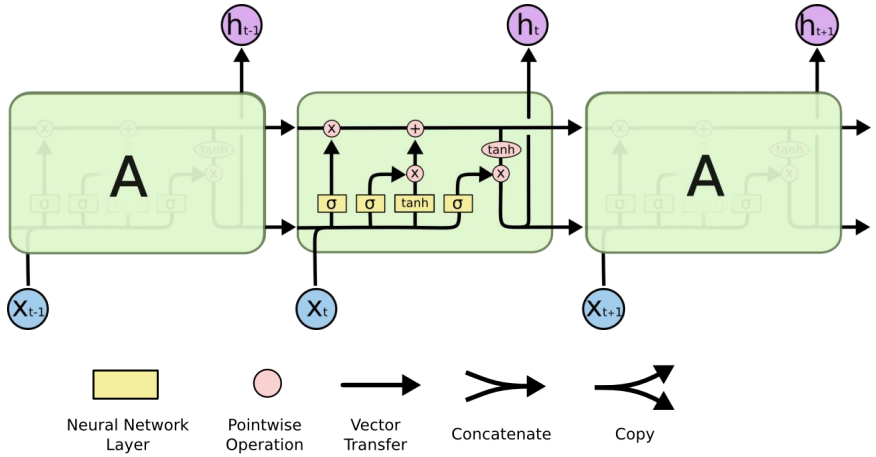


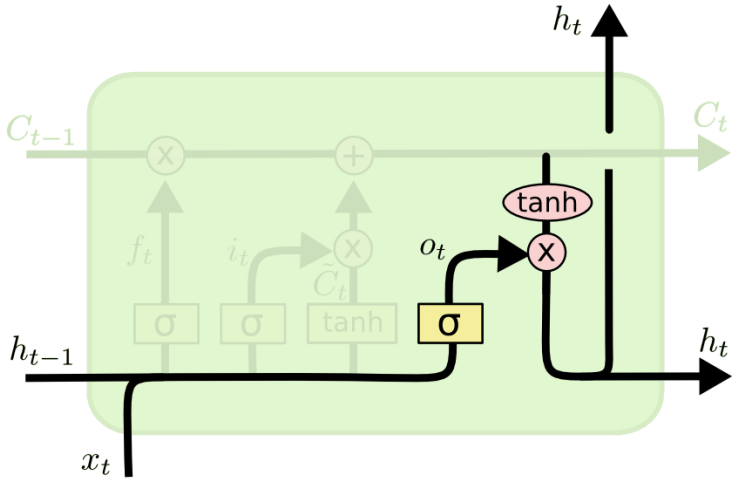


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Long Short-Term Memory Network

- LSTM (Hochreiter & Schmidhuber, 1997)
  - Output gate o_t
  - Compute next hidden state $h_t = o_t \odot \tanh(c_t)$
    - $\tanh(c_t)$: non-linear transformation over all past information
    - o_t: choose important dimensions for next state
      - $o_t(j) \rightarrow 1$: $\tanh(c_t(j))$ is critical for next state
      - $o_t(j) \rightarrow 0$: $\tanh(c_t(j))$ does not worth reporting



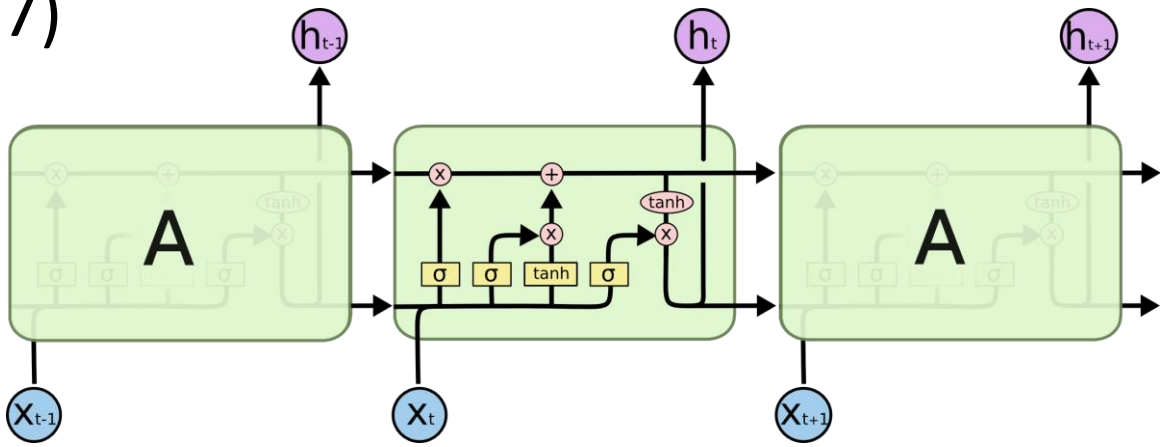


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Long Short-Term Memory Network

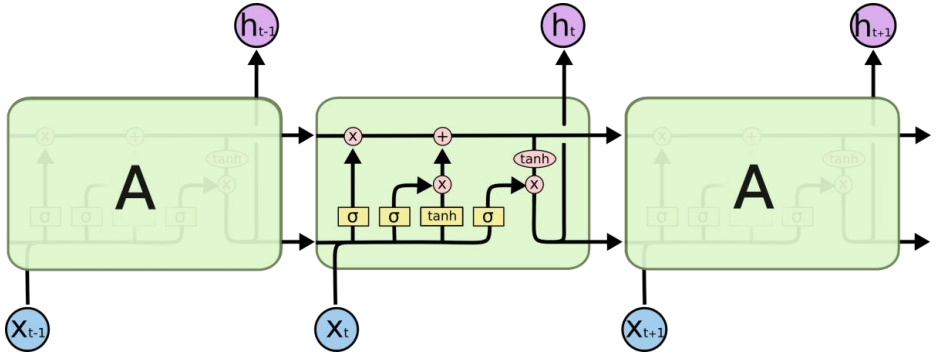
- LSTM (Hochreiter & Schmidhuber, 1997)
  - $h_t = o_t \odot \tanh(c_t)$
  - $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
  - $Y_t = g(h_t)$
  - Uninterrupted gradient flow!
    - No more matrix multiplication for c_t
    - In practice: ~100 timesteps of memory instead of ~7

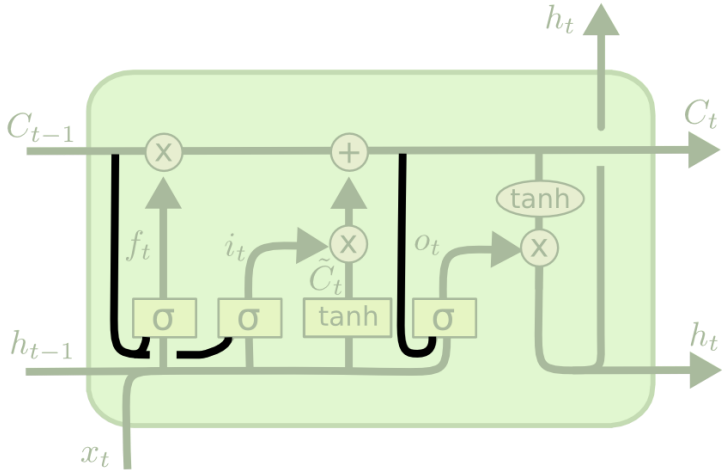


- Remark
  - LSTM does not have guarantees for gradient explosion/vanishing
  - An architecture that makes learning long-term dependency easier
  - LSTMs is the dominant approach for sequence modeling from 2013~2016

LSTM Variants

- Peephole Connections (Gers & Schmidhuber 2000)
  - Also allow gates to take in c_t information

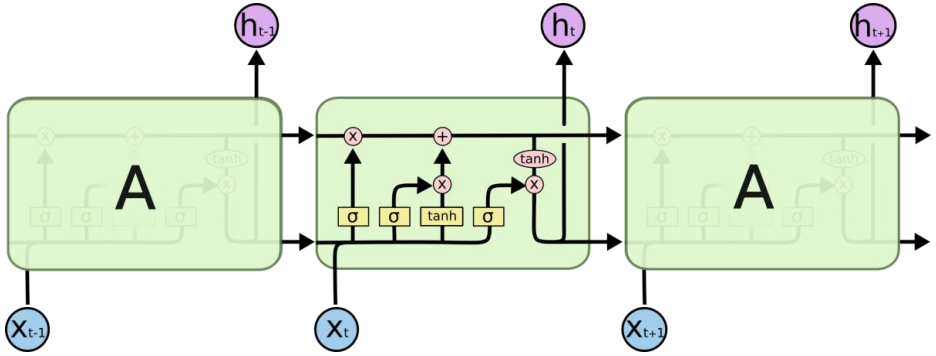


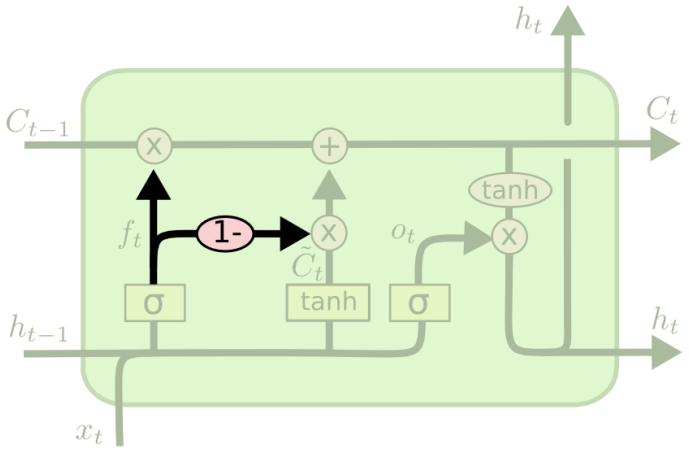


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

LSTM Variants

- Peephole Connections (Gers & Schmidhuber 2000)
- Simplified LSTM
  - Assume $i_t = 1 - f_t$
  - So only two gates are needed
    - Fewer parameters

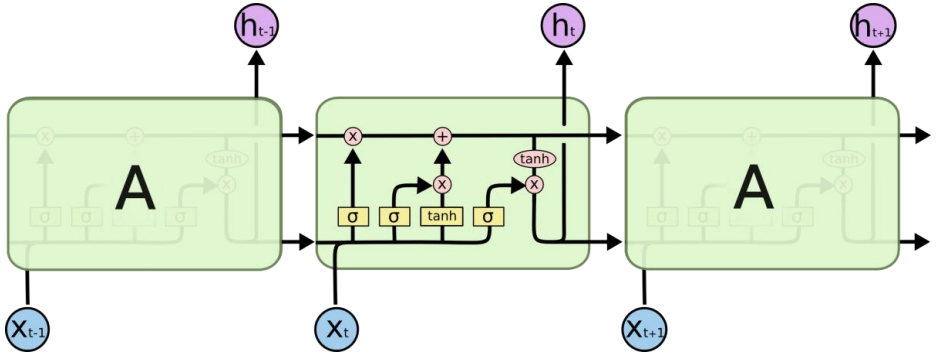


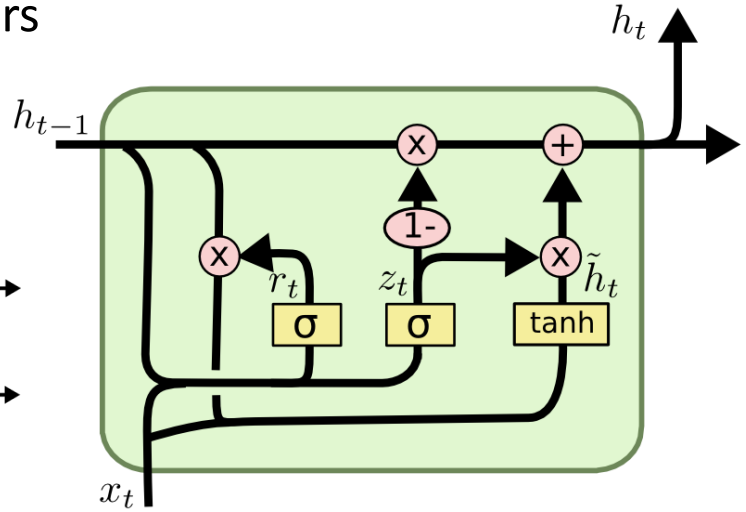


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

LSTM Variants

- Peephole Connections (Gers & Schmidhuber 2000)
- Simplified LSTM
- Gated Recurrent Unit (GRU, Cho et al, 2014)
  - Typically we only use h_t to produce outputs in LSTM
  - GRU: Merge h_t and c_t
    - Much fewer parameters





$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

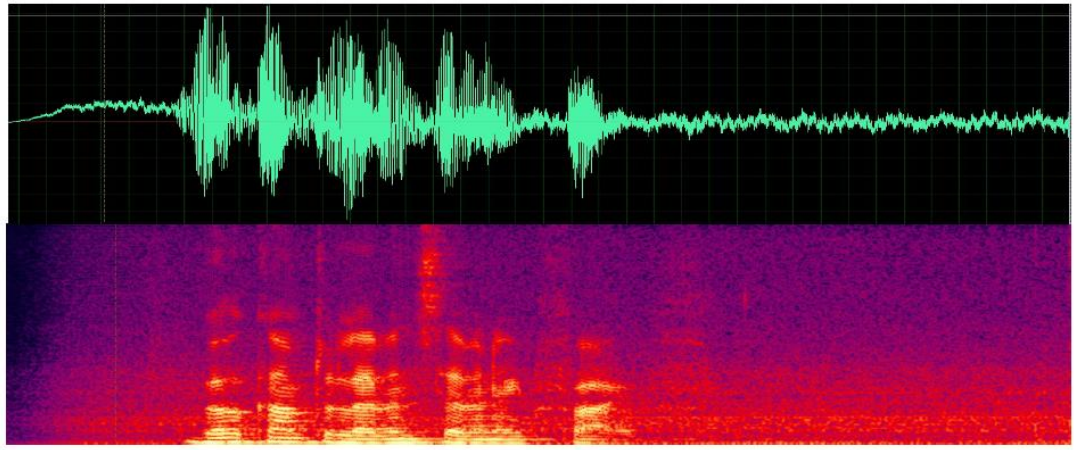
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

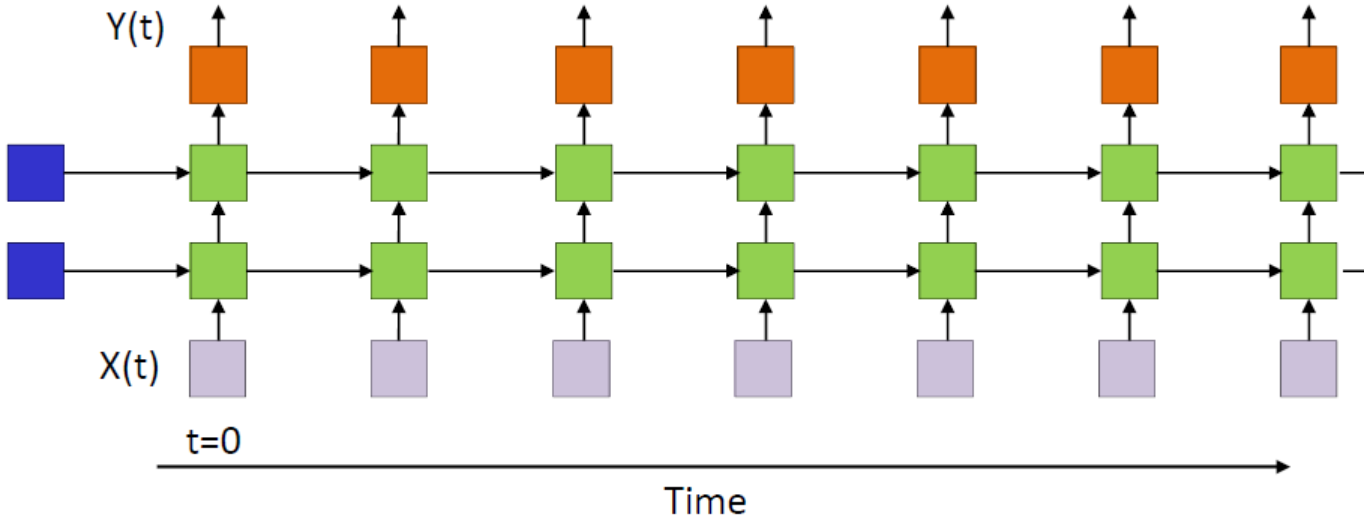
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM Applications

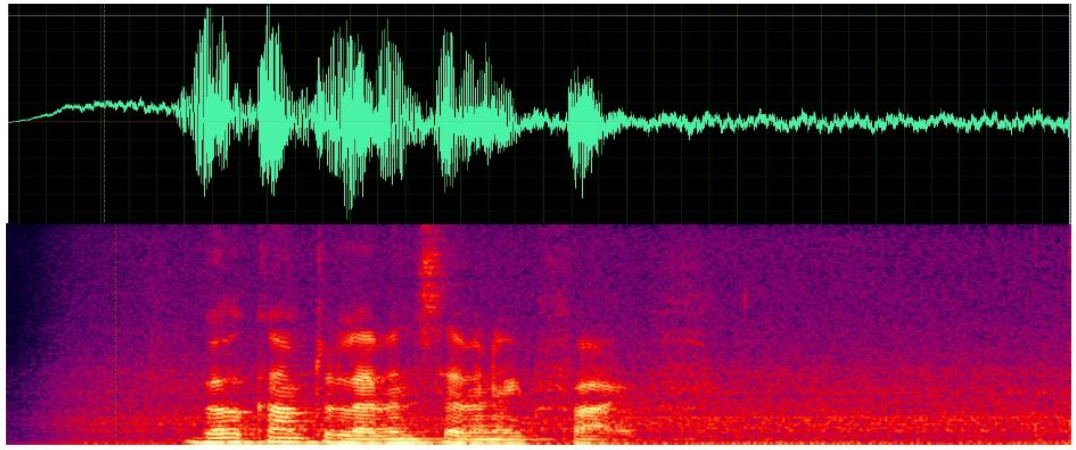
- Finding the “Welcome”
  - Input data $X_1 \ldots X_L$, L may vary
  - Whether the voice contains “Welcome”
- Solution
  - Multi-layer LSTM and max-pooling over Y_t
    - Sometimes also just use h_T to compute output for simplicity

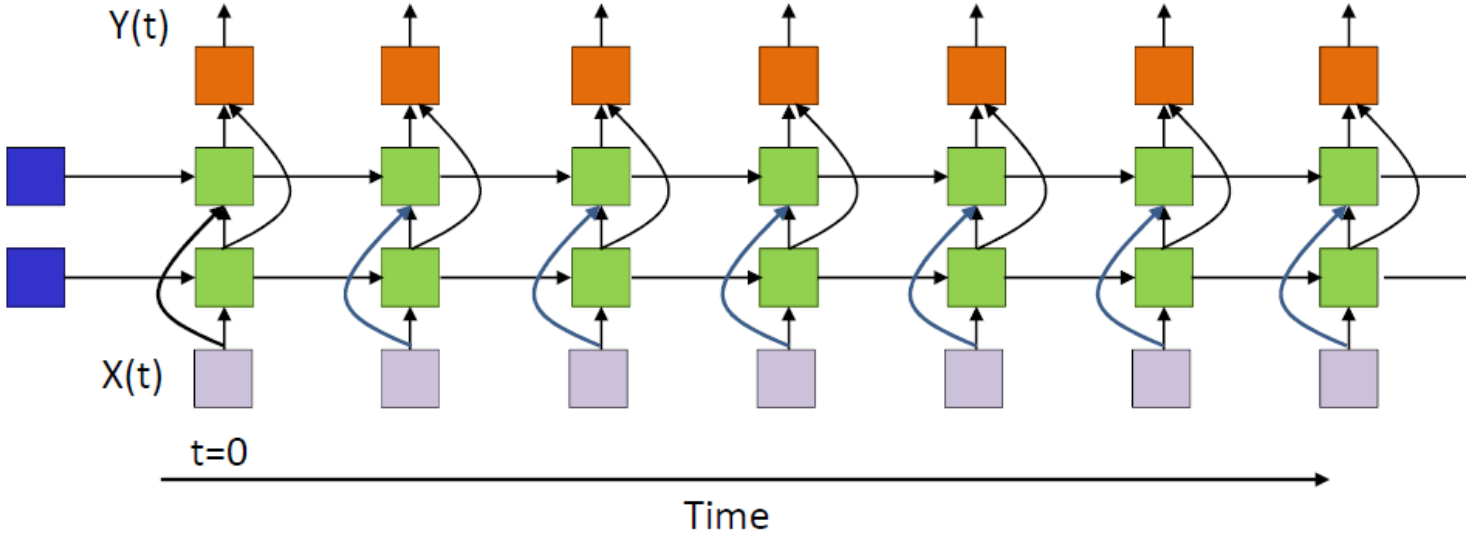




LSTM Applications

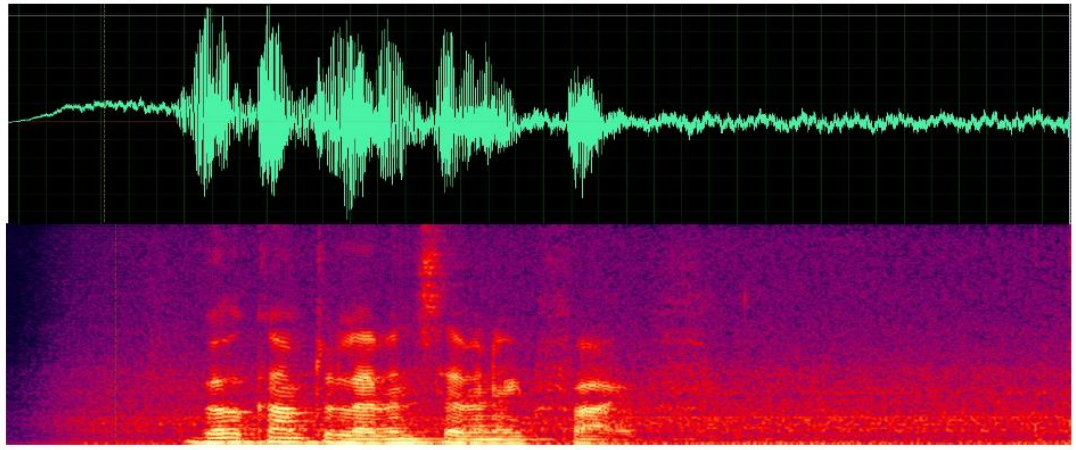
- Finding the “Welcome”
  - Input data $X_1 \dots X_L$, L may vary
  - Whether the voice contains “Welcome”
- Solution
  - Multi-layer LSTM and max-pooling over Y_t
    - Skip-connections for deeper LSTMS!

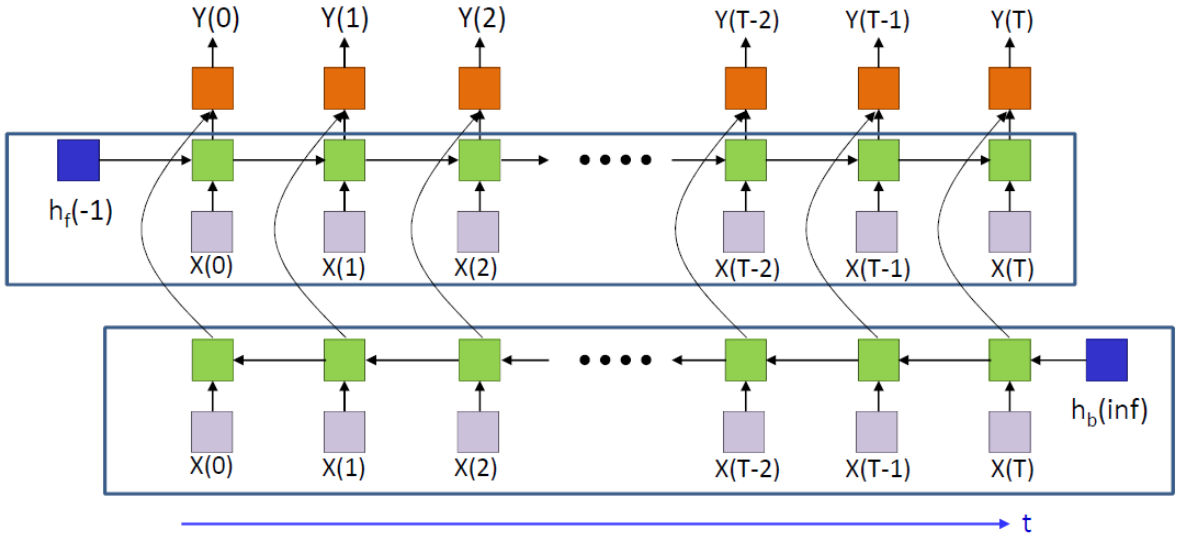




LSTM Applications

- Finding the “Welcome”
  - Input data $X_1 \ldots X_L$, L may vary
  - Whether the voice contains “Welcome”
- Solution
  - Multi-layer LSTM and max-pooling over Y_t
    - Skip-connections for deeper LSTMS!
  - Bidirectional LSTMs!
    - Sometimes use $h_f(T)$ & $h_b(T)$ for output
  - Remember gradient clipping!





LSTM Applications

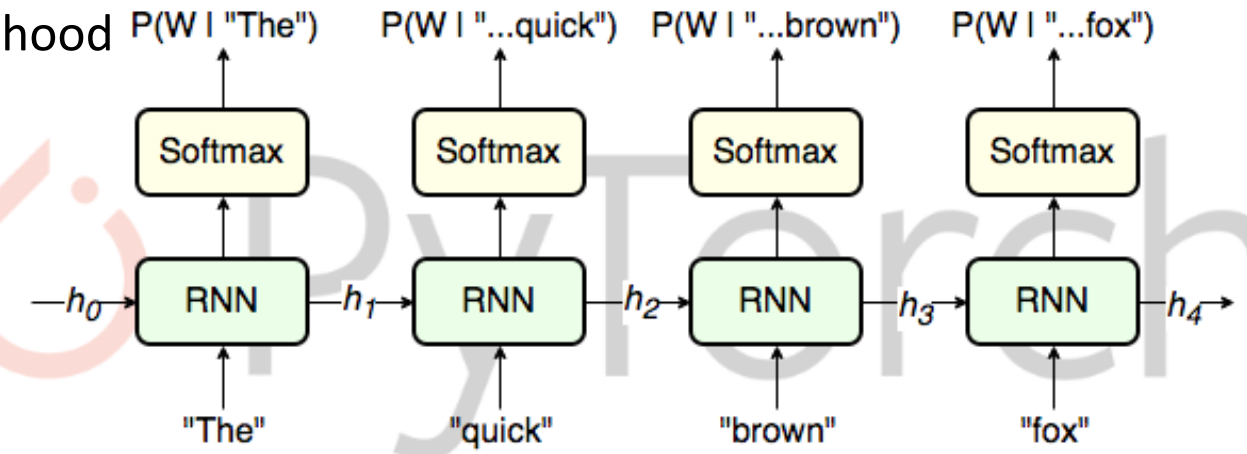
- What about text generation?
  - A generative model over texts
  - $p(X;\theta)$: the probability for X
  - Training data:
    - A collection of texts
    - E.g.: 唐诗宋词合集
  - Even conditional generation!
    - Next lecture

Language Model

- Language Model $p(X)$
  - A generative model over natural language X
- Autoregressive Language Model

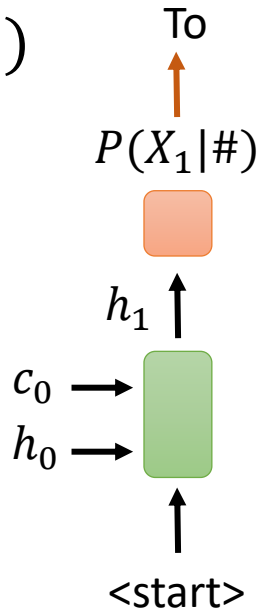
$$P(X;\theta) = \prod_{t=1}^{L} P(X_t | X_{i<t}; \theta)$$

  - The most popular model assumption
    - Sequential generation & tractable likelihood
  - LSTM language model
    - X_t: word at position t
    - Y_t: $P(X_t | X_{i<t})$, softmax over all words
  - MLE Training!
    - MLE over a training corpus D



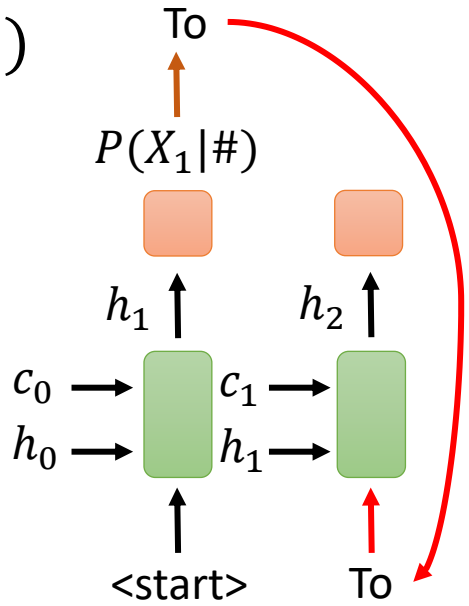
Language Model

- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Language generation
  - Draw $X \sim P(X;\theta)$
  - Sample $X_1 \sim Y_1(h_0, c_0, \#;\theta)$



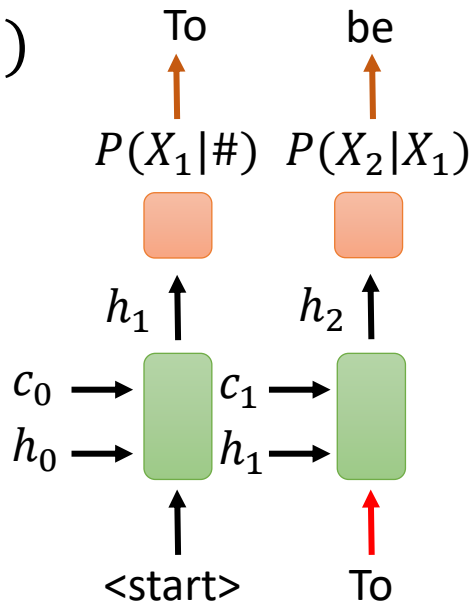
Language Model

- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Language generation
  - Draw $X \sim P(X;\theta)$
  - Sample $X_1 \sim Y_1(h_0, c_0, \#;\theta)$
  - Generate X_2
    - Feed X_1 into LSTM
    - Compute Y_2



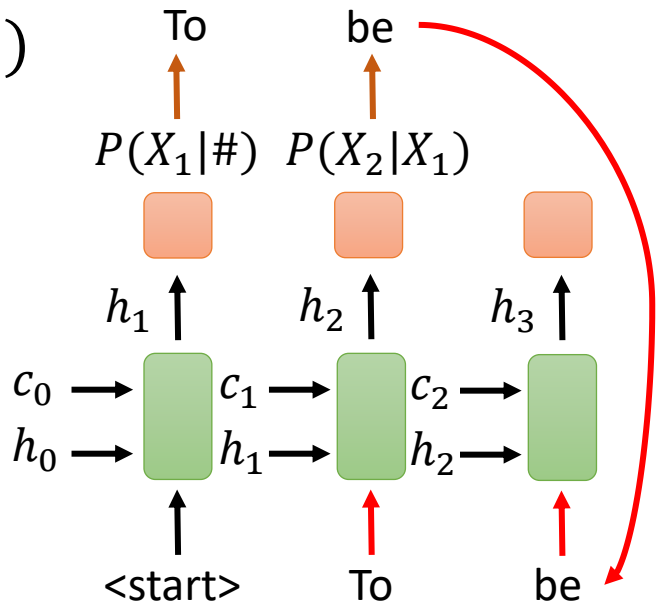
Language Model

- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Language generation
  - Draw $X \sim P(X;\theta)$
  - Sample $X_1 \sim Y_1(h_0, c_0, \#;\theta)$
  - Generate X_2
    - Feed X_1 into LSTM
    - Compute Y_2
    - Sample $X_2 \sim Y_2(h_1, c_1, X_1;\theta)$



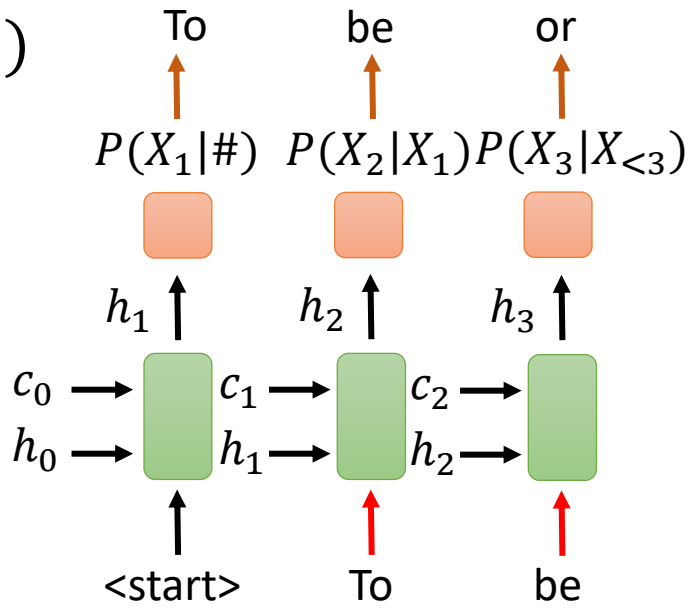
Language Model

- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Language generation
  - Draw $X \sim P(X;\theta)$
  - Sample $X_1 \sim Y_1(h_0, c_0, \#;\theta)$
  - Generate X_2
  - Generate X_3
    - LSTM forward step



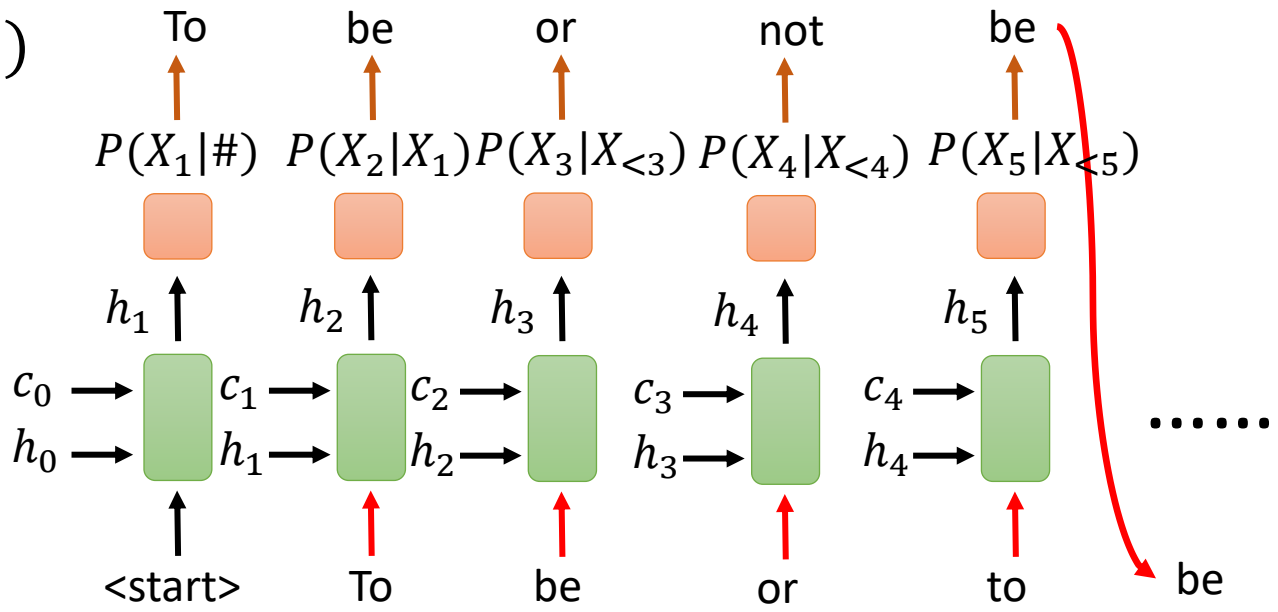
Language Model

- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Language generation
  - Draw $X \sim P(X;\theta)$
  - Sample $X_1 \sim Y_1(h_0, c_0, \#;\theta)$
  - Generate X_2
  - Generate X_3
    - LSTM forward step
    - Sample $X_3 \sim Y_3(c_2, h_2, X_2;\theta)$



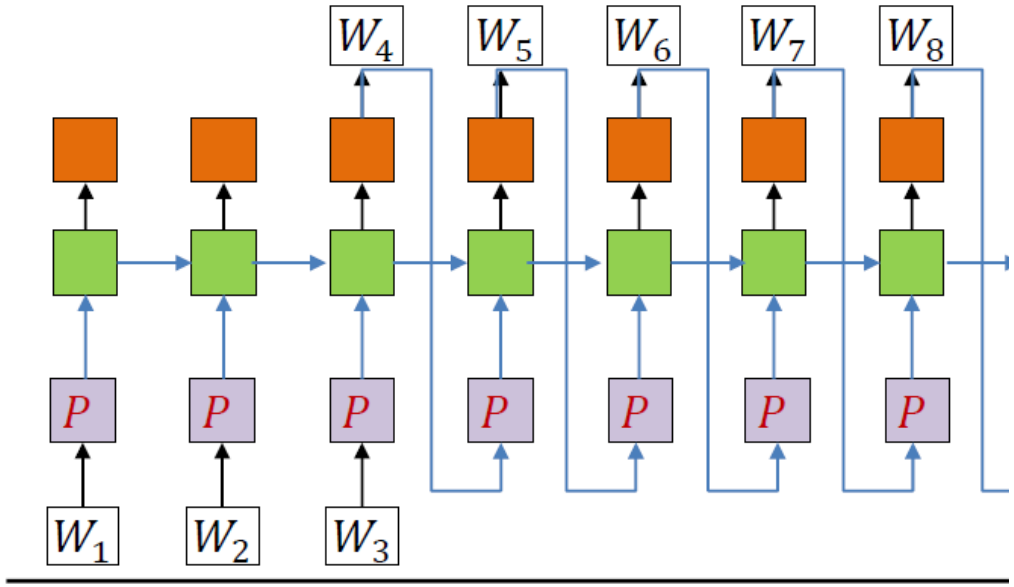
Language Model

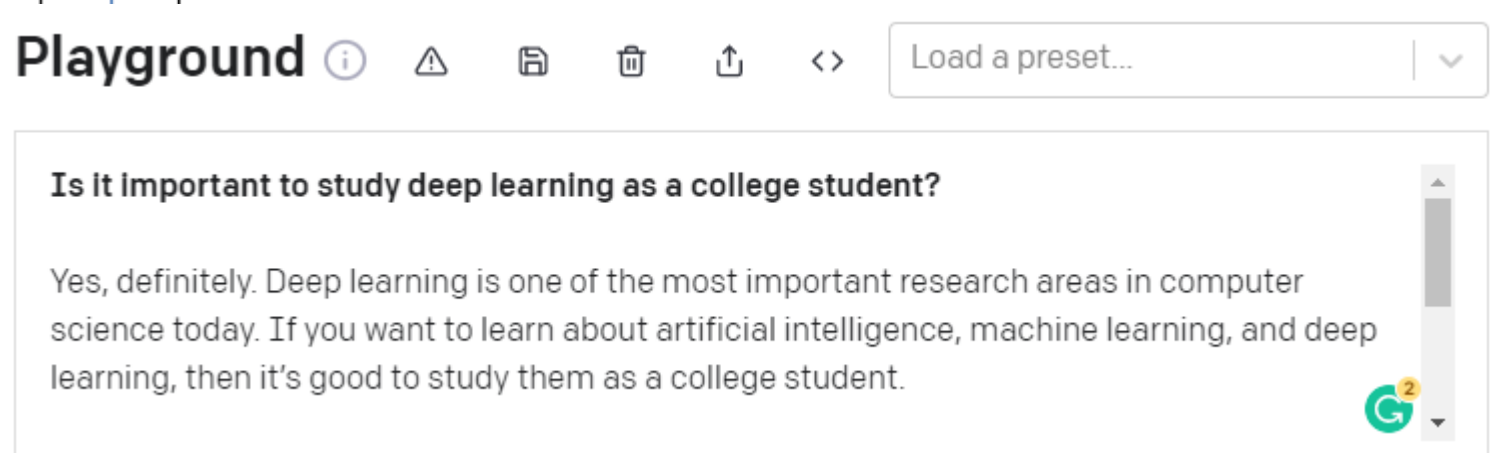
- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Language generation
  - Draw $X \sim P(X;\theta)$
  - Sample $X_1 \sim Y_1(h_0, c_0, \#;\theta)$
  - Generate X_2
  - Generate X_3
  - Repeat
- Remark
  - Ensure 1 position shift at training time!



Language Model

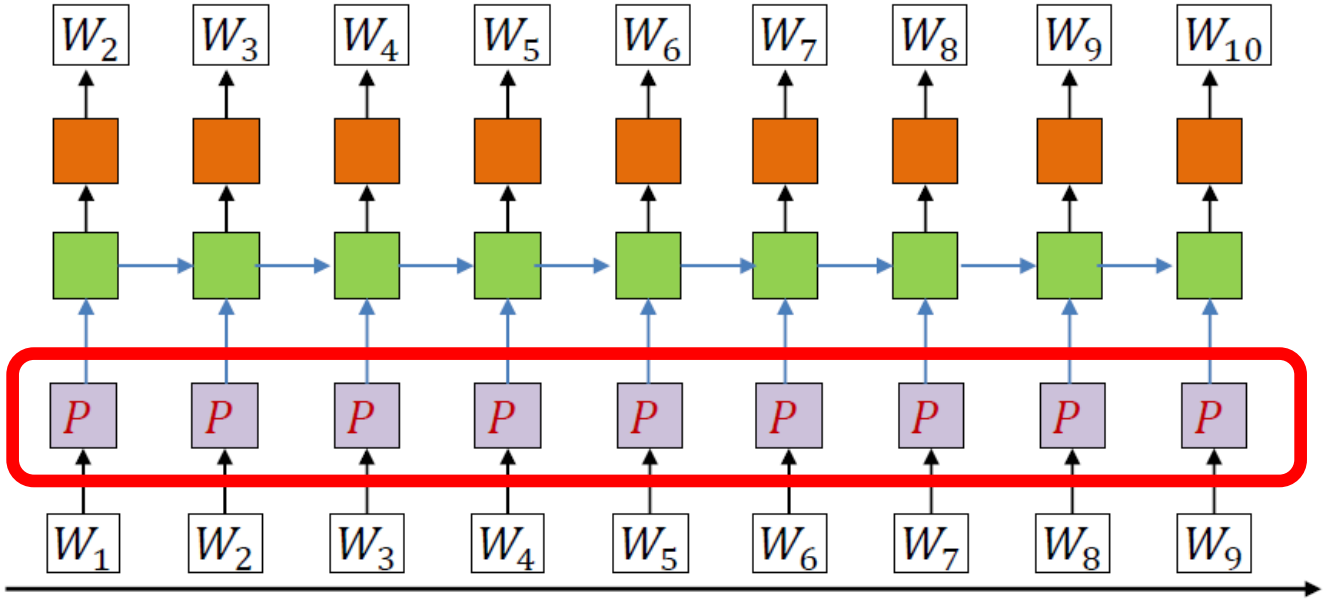
- We can also generate a language with any given prefix
  - Given $w_{1\sim3}$, we can sample $P(W|w_{1\sim3};\theta)$
  - E.g.: question answering





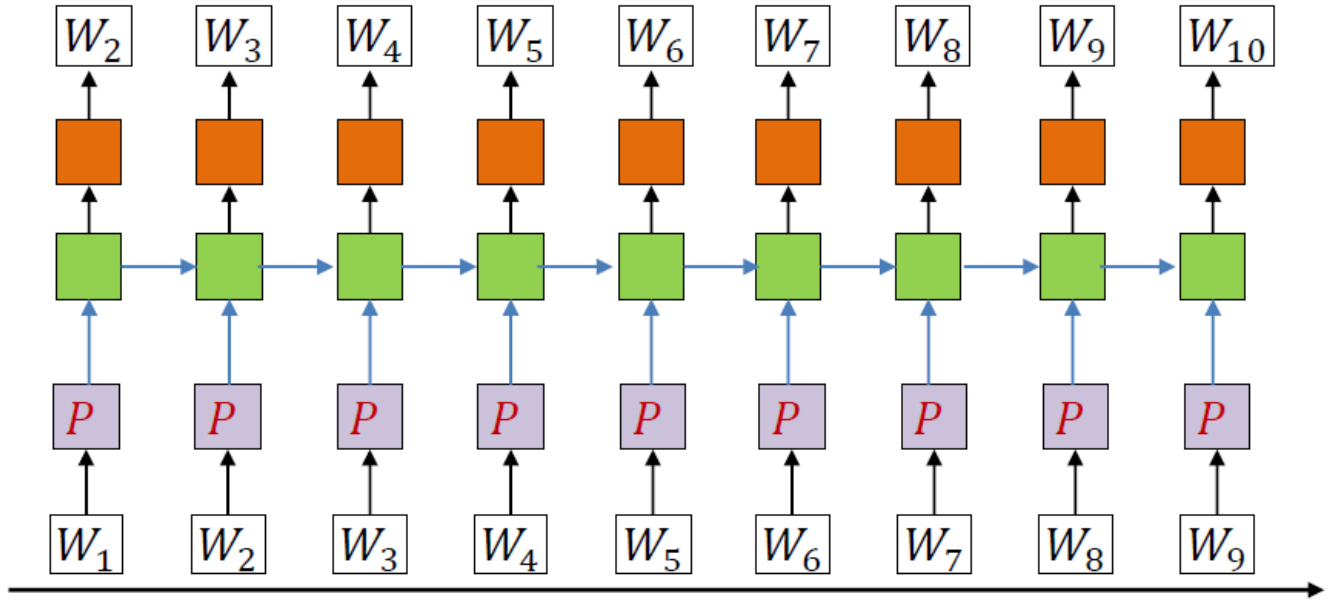
Language Model

- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- MLE Training and Autoregressive generation
- Input projection
  - “to be or not to be …”
  - w_t are discrete tokens
  - $LSTM$ requires vector input
  - Trivial solution:
    - One-hot vector
    - $X_t = [0,0,\dots,1,\dots,0]$
  - Issue?



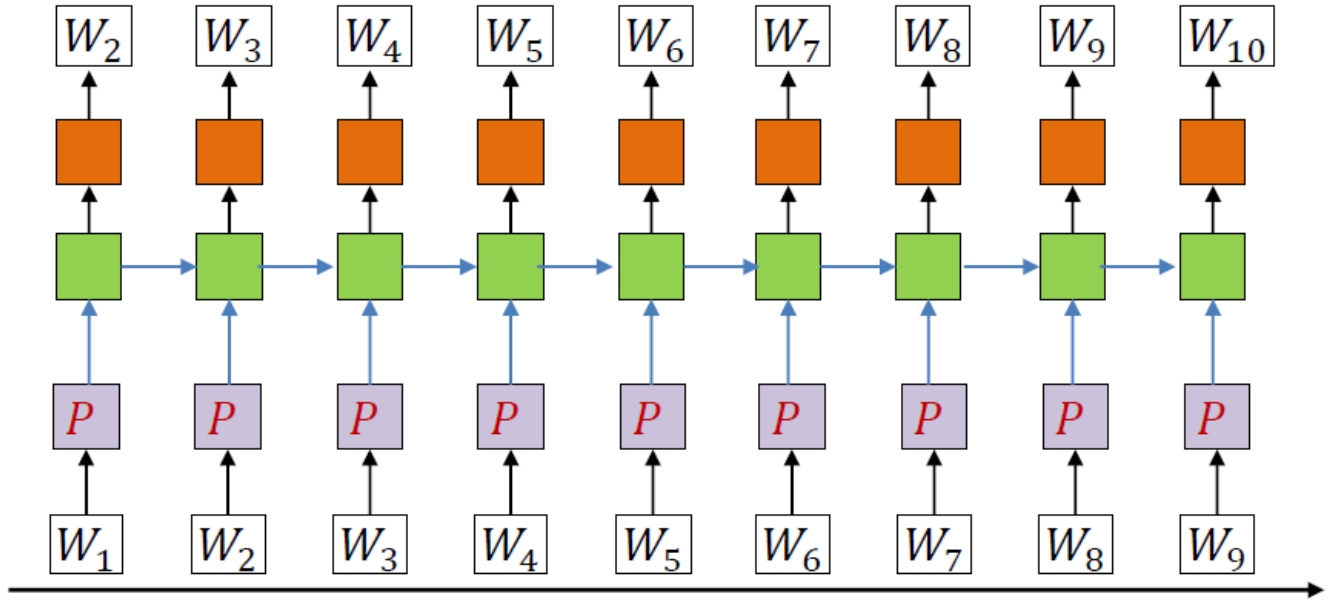
Language Model

- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Trivial input projection: one-hot encoding
  - Change a word to an ID
    - "To be or not to be …"
    - [123, 444, 8, 91, 123, 444, …]
  - No semantic meaning
    - $P(I\ live\ in\ Beijing) = 0.3$
    - $P(I\ live\ in\ Shanghai) = ?$
    - What if D has no "Shanghai"?



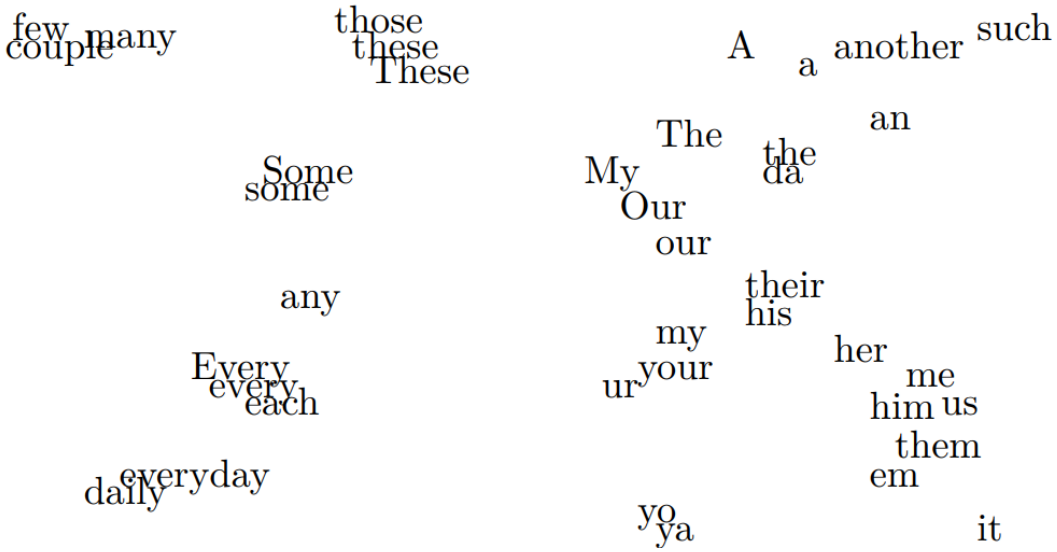
Language Model

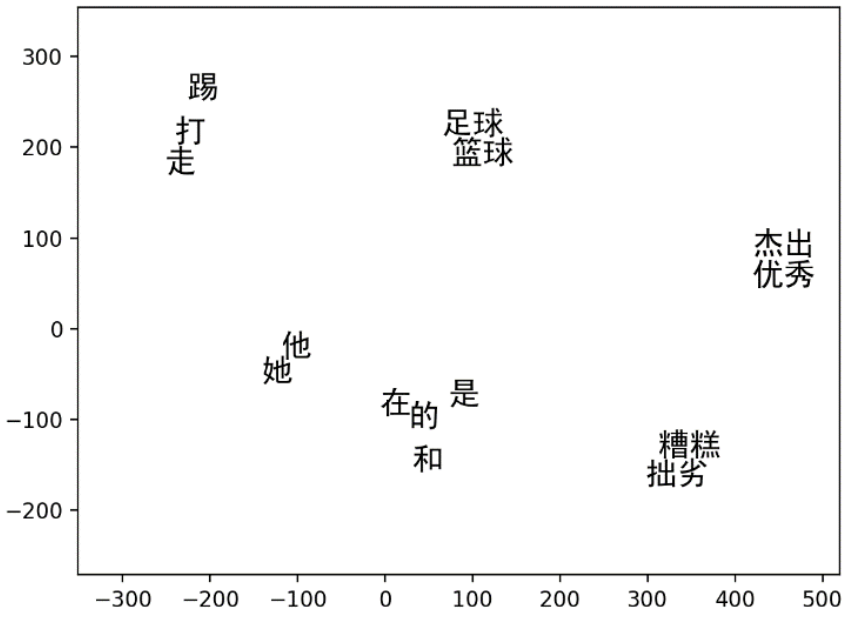
- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Goal: learn meaningful continuous representation for words
  - E.g., “Beijing”
    - It is a city in China
    - It is a noun
    - It is a capital
    - Close to “Shanghai”
    - Different from “deep”
  - Word embeddings



Word Embedding

- A semantic vector representation for words
  - Proposed in the book, “The Measurement of Meaning” 1957
    - Manually propose a few features and scores
  - Let’s learn word embeddings!





Word Embedding

- Distributional Hypothesis
  - A word's meaning is given by the words that frequently appear close-by
  - "You shall know a word by the company it keeps" (J. R. Firth 1957: 11)
- How to measure the "similarity" of two words?
  - Given word vector w_1 and w_2
  - We use cosine distance

$$D(w_1, w_2) = \cos < w_1, w_2 > = \frac{w_1^T w_2}{|w_1||w_2|}$$

- Learning objective
  - If two words are close to each other, their word embeddings have small distance
  - Otherwise, the distance should be large

Word Embedding

- Word2Vec (Mikolov, et al, Google, 2013)
  - An efficient toolbox for learning word embeddings
  - Formulation
    - Given $2k$ context (上下文) vectors $c_{-k}c_{-k+1}\dots c_{-1}?\,c_1c_2\dots c_k$
    - $P(w|c_{-k}\dots c_k)$: Predict which word should appear in the position of "?"
    - Independence assumption $P(w|c) = \prod_i P(w|c_i)$
    - $P(w|c_i)$: a softmax distribution over all words
      - There are a lot of words!!
  - We can convert multi-class classification to binary-class classification

Word Embedding

- Word2Vec (Mikolov, et al, Google, 2013)
  - An efficient toolbox for learning word embeddings
  - Simplified formulation
    - Given $2k$ context (上下文) vectors and a word w: $c_{-k}c_{-k+1} \dots c_{-1}wc_1c_2 \dots c_k$
    - $P(+|c_{-k} \dots w \dots c_k)$: the probability of w should appear with c
    - Independence assumption $P(+|w,c) = \prod_i P(+|w,c_i)$
    - $P(+|w,c_i)$: a value between 0 and 1
      - Sigmoid function over $D(w,c_i)$
      - Assume all the vectors have unit norm

$$P(+|w,c_i) = \sigma(w,c_i) = \frac{1}{1+\exp(-w^T c_i)}$$

$$P(-|w,c_i) = 1 - P(+|w,c_i)$$

$$\log P(+|w,c) = \sum_i \log P(+|w,c_i)$$

Word Embedding

- Word2Vec (Mikolov, et al, Google, 2013, NeurIPS 2023 test-of-time)
  - An efficient toolbox for learning word embeddings
  - Simplified formulation (CBOW, continuous bag-of-word model)
    - Given $2k$ context (上下文) vectors and a word w: $c_{-k}c_{-k+1} \dots c_{-1}wc_1c_2 \dots c_k$
    - $P(+|c_{-k} \dots w \dots c_k)$: the probability of w should appear with c

$$P(+|w, c_i) = \sigma(w, c_i) = \frac{1}{1 + \exp(-w^T c_i)}$$

$$P(-|w, c_i) = 1 - P(+|w, c_i)$$

$$\log P(+|w, c) = \sum_i \log P(+|w, c_i)$$

  - MLE Training!
    - Positive (w, c) pairs: all the text chunks of length $2k + 1$ from training corpus D
    - All set?
      - Identical vectors maximize the learning objective!

Word Embedding

- Word2Vec (Mikolov, et al, Google, 2013)
  - An efficient toolbox for learning word embeddings
  - Simplified formulation (CBOW, continuous bag-of-word model)
    - Given $2k$ context (上下文) vectors and a word w: $c_{-k}c_{-k+1} \dots c_{-1}wc_1c_2 \dots c_k$
    - $P(+|c_{-k} \dots w \dots c_k)$: the probability of w should appear with c

$$P(+|w, c_i) = \sigma(w, c_i) = \frac{1}{1 + \exp(-w^T c_i)}$$

$$P(-|w, c_i) = 1 - P(+|w, c_i)$$

$$\log P(+|w, c) = \sum_i \log P(+|w, c_i)$$

  - MLE Training!
    - Positive (w, c) pairs: all the text chunks of length $2k + 1$ from training corpus D
    - We need negative pairs!
      - Choose a context c, and select random negative words w'
      - *This training method is called **negative sampling***

Word Embedding

- Word2Vec (Mikolov, et al, Google, 2013)

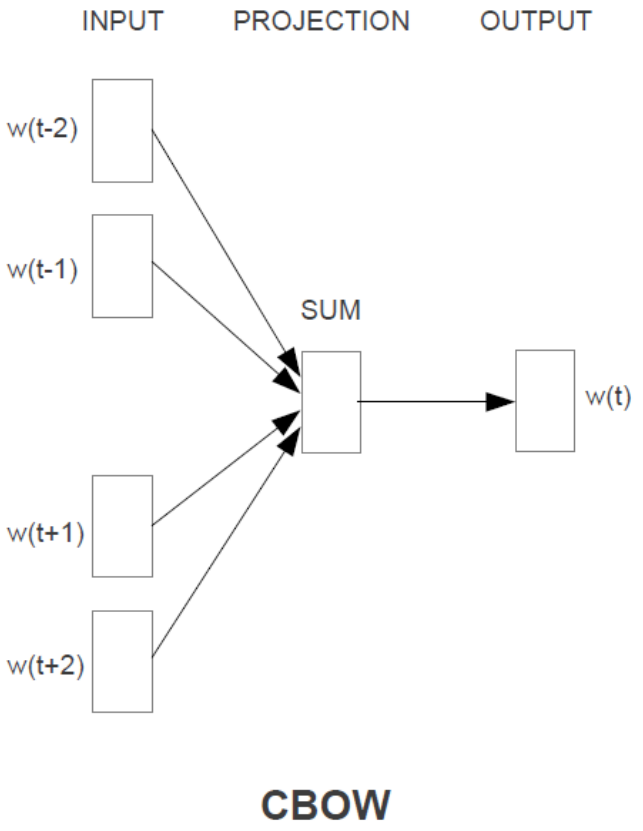
$$P(+|w, c_i) = \sigma(w, c_i) = \frac{1}{1 + \exp(-w^T c_i)}$$

$$\log P(+|w, c) = \sum_i \log P(+|w, c_i)$$

  - CBOW Training corpus D
    - For every text chunks $(c_{-k} \dots, w, \dots, c_k)$ in D
    - Collect positive data pair (c, w), add to D^+
    - Random choose a word w', add (c, w') to D^-
  - MLE Training

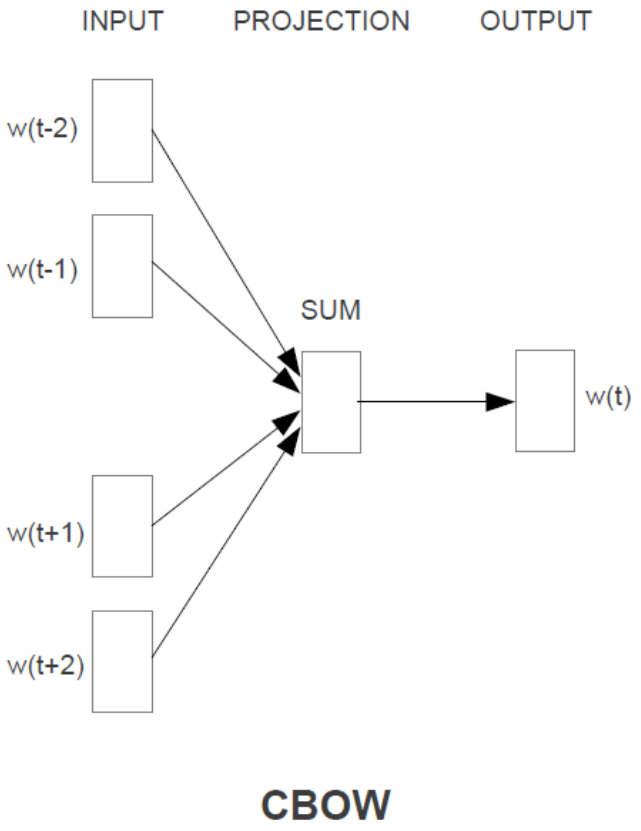
$$L(W, C) = \sum_{(c,w) \in D^+} \log P(+|w, c) + \sum_{(c,w') \in D^-} \log P(-|w', c)$$

  - Use w_i as the word embedding for the i-th word
    - We can also use $[c_i, w_i]$, or simply ignore c_i



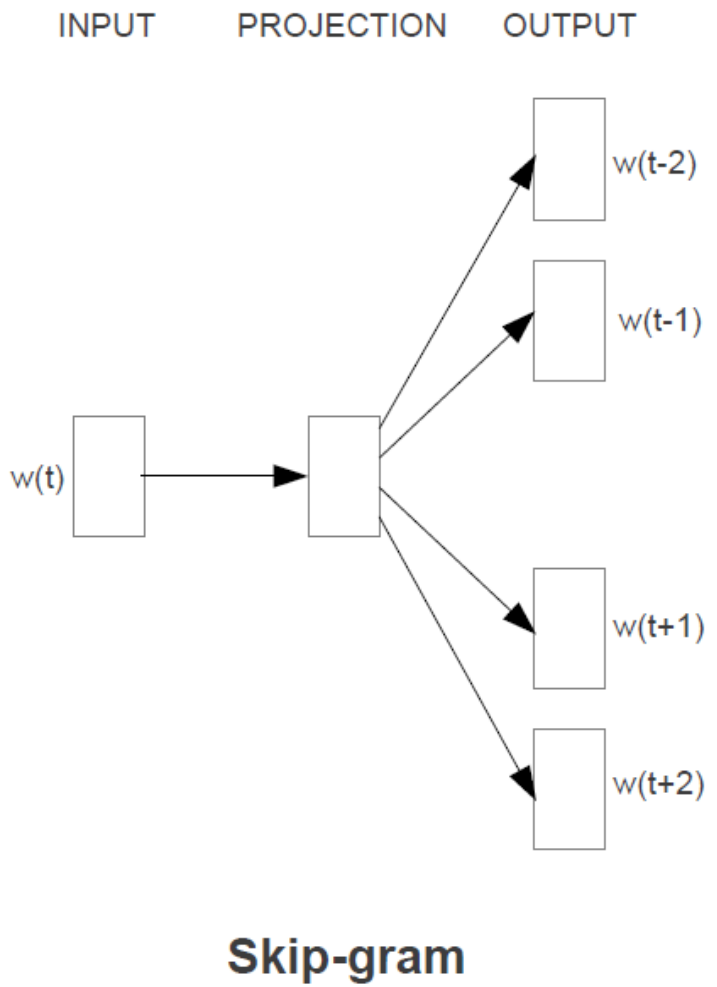
Word Embedding

- Word2Vec (Mikolov, et al, Google, 2013)
  - Continuous Bag-of-Words (CBOW)
    - Objective $\log P(w|c_i)$
    - Use contexts c to predict center word w
  - Alternative: use w to predict surrounding words c



Word Embedding

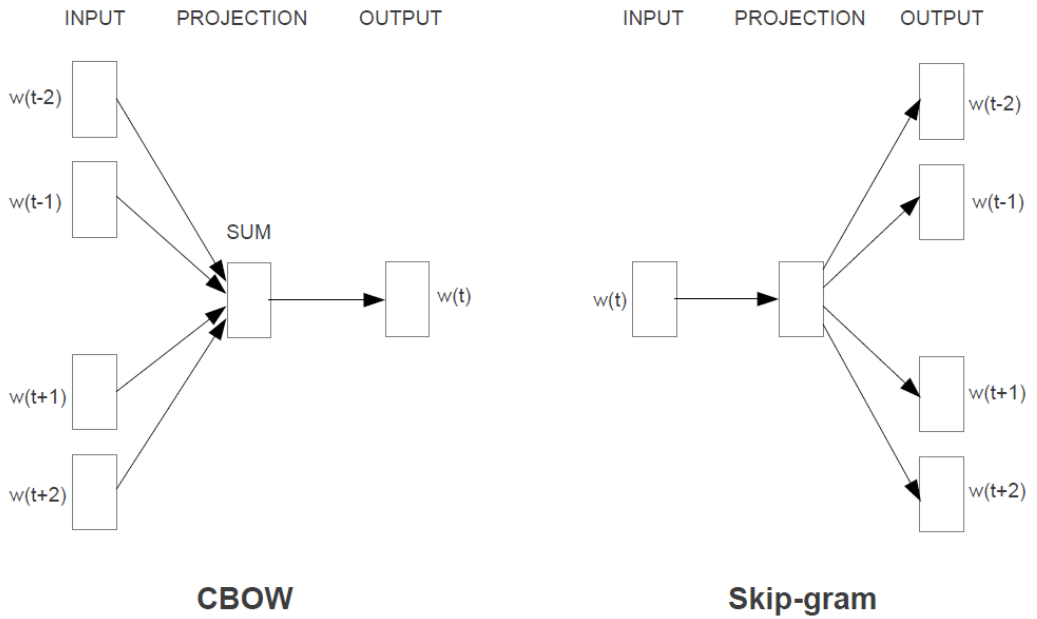
- Word2Vec (Mikolov, et al, Google, 2013)
  - Continuous Bag-of-Words (CBOW)
    - Objective $\log P(w|c_i)$
    - Use contexts c to predict center word w
  - Skip-Gram Model
    - Use a single center word c to predict $w_{-k}, \dots w_{-1}, *, w_1 \dots w_k$
    - Objective $\log P(w_i|c)$
    - Skip-Grams
      - Randomly choose sample $2R$ positions from $-k \dots -1, 1, \dots k$
    - Training Corpus D
      - For every text chunks $(w_{-k} \dots, c, \dots, w_k)$ in D
      - select a subset of $2R$ words from $\tilde{w} \subseteq w_{-k} \dots w_k$
      - Collect positive data pair $(c, \tilde{w})$, add to D^+
      - Random choose $2R$ words $\tilde{w}'$, add $(c, \tilde{w}')$ to D^-



Skip-gram

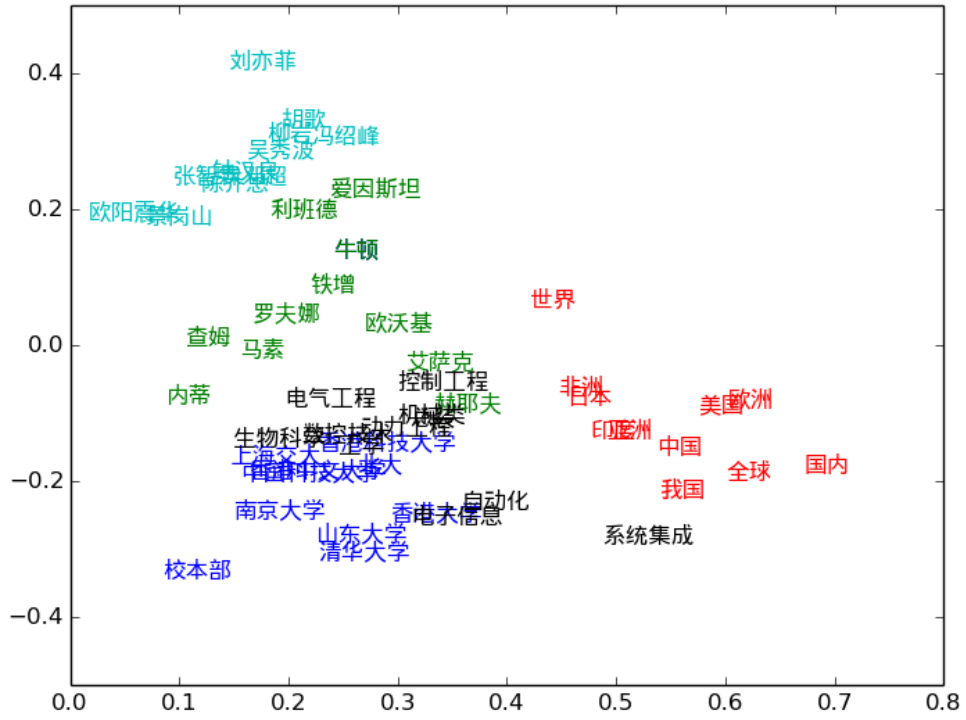
Word Embedding

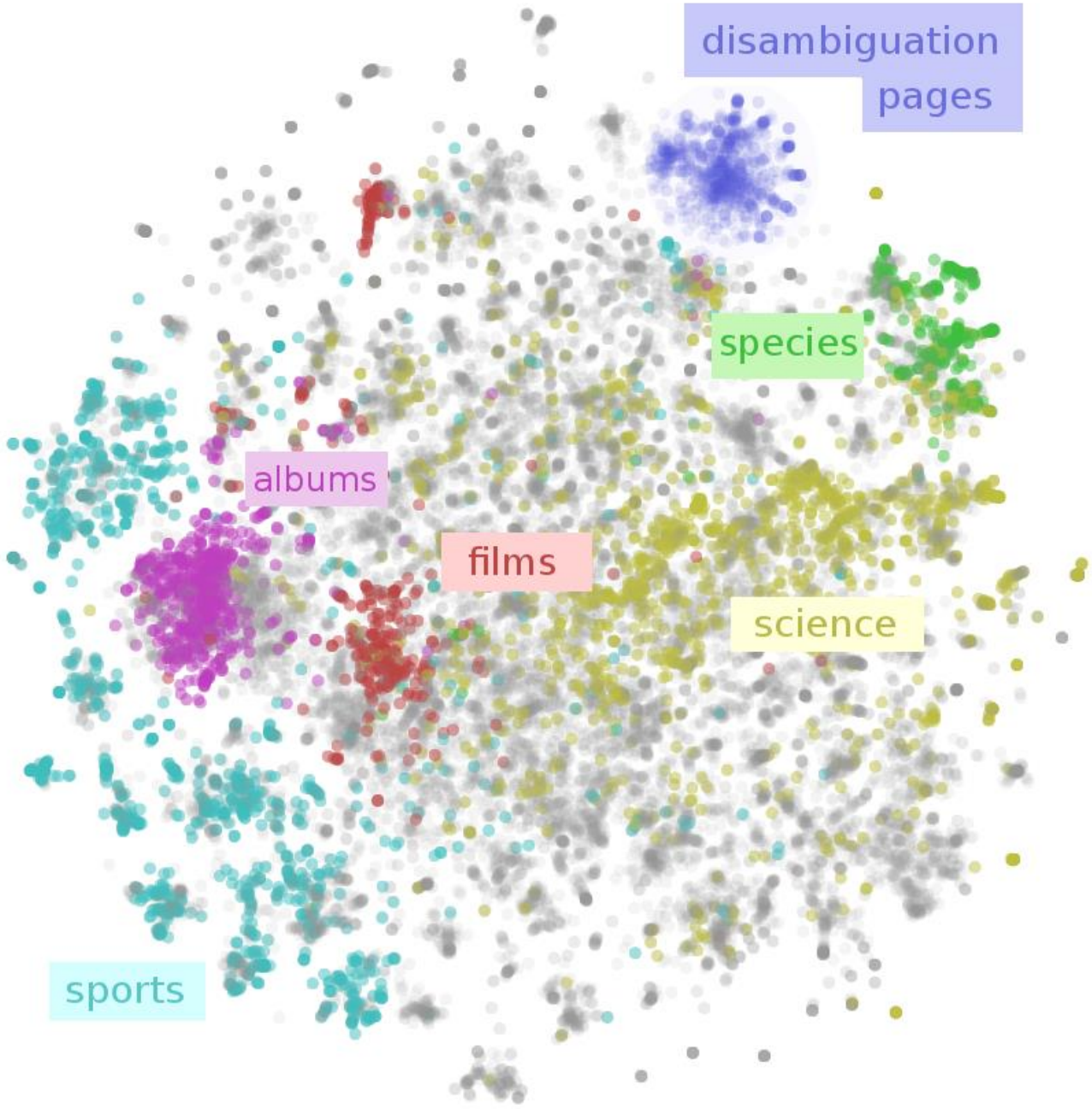
- Word2Vec (Mikolov, et al, Google, 2013)
  - Continuous Bag-of-Words (CBOW)
    - Use contexts c to predict center word w
  - Skip-Gram Model
    - Use a single center word c to predict $w_{-k}, \dots w_{-1}, *, w_1 \dots w_k$
- Remark
  - CBOW trains faster than Skip-Gram
  - Skip-Gram is a harder problem
    - Harder to overfit
  - Skip-Gram performs better
    - Particularly for rare words



Word Embedding

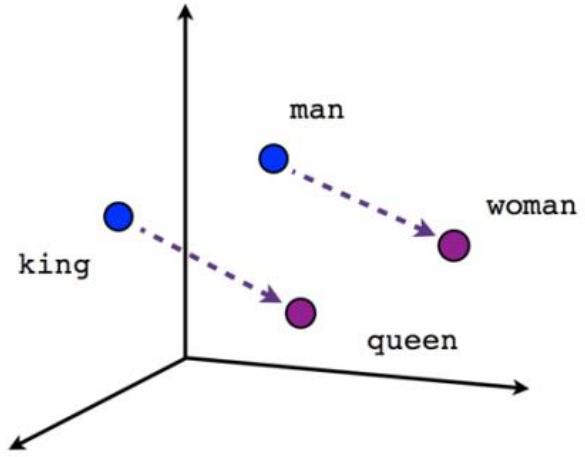
- Word2Vec Visualization
  - t-SNE projection in 2D
  - Similar topics cluster together



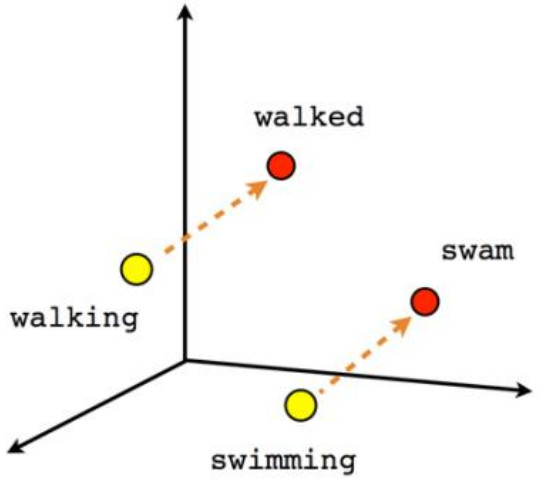


Word Embedding

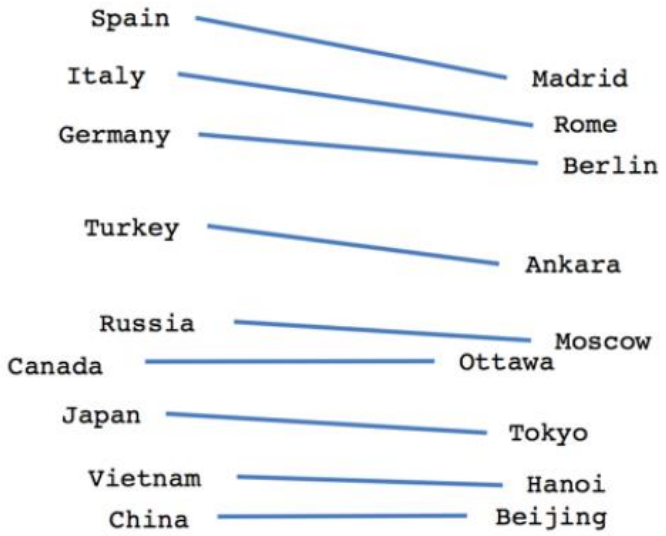
- Word2Vec Vector Arithmetic
  - Emergent analogies
    - king – man + woman ≈ queen
    - Beijing – China + France ≈ Paris



Male-Female



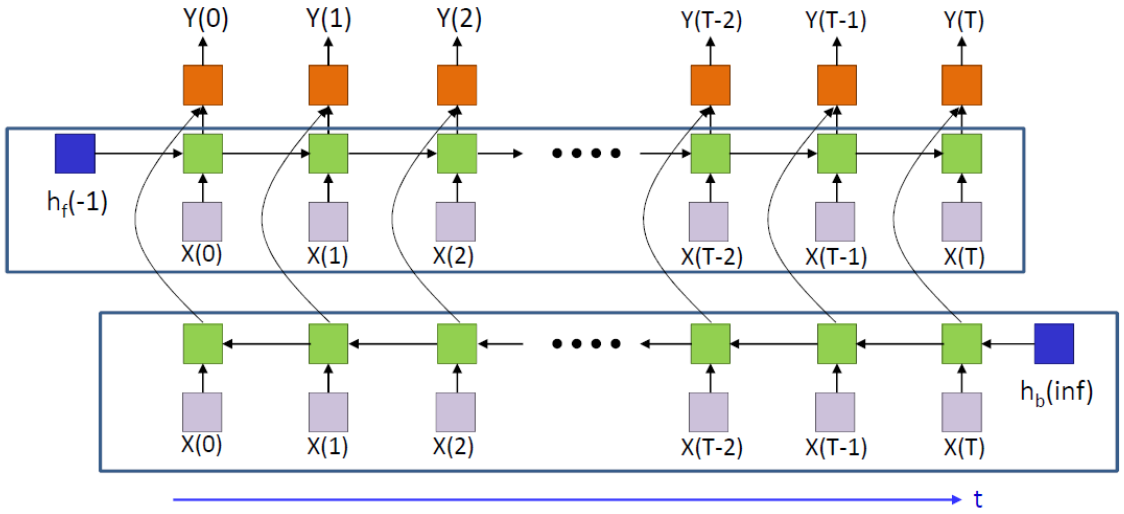
Verb tense

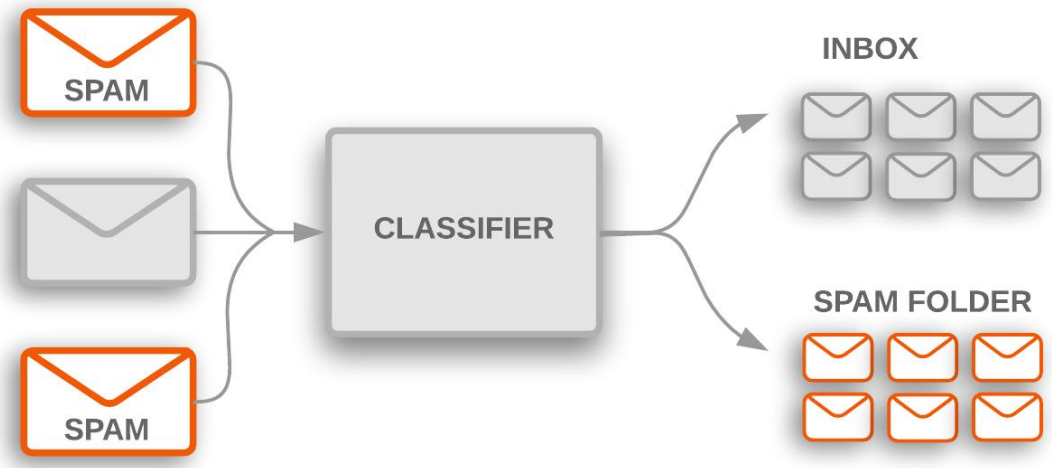


Country-Capital

LSTM Applications

- Pre-processing
  - Collect a large corpus and learn word embeddings (word2vec)
- Text classification
  - Bi-directional LSTM and then run Softmax on final hidden states





LSTM Applications

- Pre-processing
  - Collect a large corpus and learn word embeddings (word2vec)
- Text classification
  - Bi-directional LSTM and then run Softmax on final hidden states
- Text generation
  - For the specific training domain, learn an autoregressive model $P(X)$

LSTM Applications

- Pre-processing
  - Collect a large corpus and learn word embeddings (word2vec)
- Text classification
  - Bi-directional LSTM and then run Softmax on final hidden states
- Text generation
  - For the specific training domain, learn an autoregressive model $P(X)$
- Text correction (文本改错)
  - MCMC over $P(X)$ to improve X
    - $-\log P(\text{文学是一种医术形式}) = 1484.5$
    - $-\log P(\text{文学是一种艺术形式}) = 234.5$

LSTM Applications

- English v.s. 中文
  - Word v.s. character
    - We typically use word models for English & character model for Chinese
    - A huge number of words in English! (“pneumonoultramicroscopicsilicovolcanoconiosi”)
      - Many practical tricks
        - Efficient Softmax operator
        - Use <unk> for very rare words
      - Dictionary is much smaller in Chinese (your homework ☺)
  - 分词 word segmentation
    - An issue in Chinese if you want to use word model: 中关村北大街
  - 词条化 word tokenization
    - A 11-year-old boy; 12345*54321=670592745 **(still critical in modern LMs)**
  - 词干化 stemming
    - Has → have; running → run
    - Apples → apple

Computation Techniques

- Language Model Learning
  - MLE learning: $P(X_t|X_{i<t})$
  - The expensive Softmax operator
    - Objective $P_\theta(w|h) = \mathrm{softmax}(w^T h) \propto \exp(w^T h)$ for $w \in V$
      - h is the hidden state of LSTM language model output
      - Equivalent: $P_\theta(w|h) \propto u_\theta(w,h)$, $u_\theta(w,h)$ is exponential logit for w given h
  - Loss
    - $L(w;\theta) = \log P_\theta(w) = \log u_\theta(w) - \log Z = \log u_\theta(w) - \log\left(\sum_{w'} u_\theta(w')\right)$
    - Partition function Z
    - Monte Carlo Estimate!
      - $\nabla L(w;\theta) = \nabla \log u_\theta(w) - \mathrm{E}_{w' \sim P_\theta}[\nabla \log u_\theta(w')]$
      - How to sample?
        - *Note: this is a categorical distribution over words…*

Computation Techniques

- Language Model Learning
  - Softmax Objective for word w and context h
    - $P_\theta(w|h) \propto u_\theta(w,h)$, $u_\theta(w,h)$ the exponential logit for w given h
    - $L_{MLE}(w|h;\theta) = \log P_\theta(w,h) = \log u_\theta(w,h) - \log\left(\sum_{w'} u_\theta(w',h)\right)$
  - Overall MLE loss over the entire corpus
    - $\tilde{P}(w|h)$: data distribution over the courpus
    - $\nabla L_{MLE}(\theta|h) = \sum_w \tilde{P}(w|h) \nabla \log u_\theta(w,h) - \frac{1}{Z_\theta} \sum_{w'} \nabla u_\theta(w',h)$
    - $= \sum_w \tilde{P}(w|h) \nabla \log u_\theta(w,h) - \mathrm{E}_{w' \sim P_\theta}[\nabla \log u_\theta(w',h)]$
    - $= \sum_w \left(\tilde{P}(w|h) - P_\theta(w|h;\theta)\right) \nabla \log u_\theta(w,h)$
    - Evaluating $P_\theta(w|h;\theta)$ is expensive
      - Same formulation as a multi-class classification problem
      - *Can we convert it to binary classification (like negative sampling) with desired gradients?*

Computation Techniques

- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - A binary-classification simplification loss with mathematical guarantees
    - Gradient of NCE loss can be shown to be equivalent to MLE loss
  - Objective $P(+|w,h)$: whether w should follow context h
    - 1 positive samples (w,h) and k random negative samples $(\overline{w},h)$ with $\overline{w}\sim q$
      - q can be uniform (similar to negative sampling) or other prior (e.g., word frequency)
    - Positive data: $P(+,w|h)=\frac{1}{k+1}\tilde{P}(w|h)$
    - Negative data: $P(-,\overline{w}|h)=\frac{k}{k+1}q(\overline{w})$
    - $P(+|w,h)=\frac{\tilde{P}(w|h)}{k\cdot q(w)+\tilde{P}(w|h)}$
    - $P(-|\overline{w},h)=\frac{k\cdot q(\overline{w})}{k\cdot q(\overline{w})+\tilde{P}(\overline{w}|h)}$

Computation Techniques

- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - A binary-classification simplification loss with mathematical guarantees
    - Gradient of NCE loss can be shown to be equivalent to MLE loss
  - Objective $P(+|w,h)$: whether w should follow context h
    - 1 positive samples (w,h) and k random negative samples $(\bar{w},h)$ with $\bar{w}\sim q$
      - q can be uniform (similar to negative sampling) or other prior (e.g., word frequency)
    - Positive data: $P(+,w|h)=\frac{1}{k+1}\tilde{P}(w|h)$
    - Negative data: $P(-,\bar{w}|h)=\frac{k}{k+1}q(\bar{w})$
    - $P(+|w,h)=\frac{\tilde{P}(w|h)}{k\cdot q(w)+\tilde{P}(w|h)}$ NCE objective (will be further explained)
    - $P(-|\bar{w},h)=\frac{k\cdot q(\bar{w})}{k\cdot q(\bar{w})+\tilde{P}(\bar{w}|h)}$
      - *Remark: we will prove that NCE **gradient** is the same as MLE gradient,*
      - *… and this is just smart Math. ☺*

Computation Techniques

- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - Binary classification with k negative samples
    - Positive data: $P(+, w|h) = \frac{1}{k+1}\tilde{P}(w|h)$
    - Negative data: $P(-, \overline{w}|h) = \frac{k}{k+1}q(\overline{w})$
    - $P(+|w, h) = \frac{\tilde{P}(w|h)}{k \cdot q(w) + \tilde{P}(w|h)}$
    - Trick#1: replace $\tilde{P}$ with learning model P_θ
    - $L(h; \theta) = \sum_w \tilde{P}(w|h) \log\big(P_\theta(+|w, h)\big) + k \cdot E_{\overline{w} \sim q}[\log P_\theta(-|\overline{w}, h)]$
      - *The binary classification loss to optimize the NCE objective*

Computation Techniques

- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - Binary classification with k negative samples
    - Positive data: $P(+, w|h) = \frac{1}{k+1}\tilde{P}(w|h)$
    - Negative data: $P(-, \bar{w}|h) = \frac{k}{k+1}q(\bar{w})$
    - $P(+|w,h) = \frac{\tilde{P}(w|h)}{k \cdot q(w) + \tilde{P}(w|h)}$
    - Trick#1: replace $\tilde{P}$ with learning model P_θ
    - $L(h;\theta) = \sum_w \tilde{P}(w|h) \log\big(P_\theta(+|w,h)\big) + k \cdot E_{\bar{w}\sim q}[\log P_\theta(-|\bar{w},h)]$
    - $= \sum_w \tilde{P}(w|h) \log\big(P_\theta(+|w,h)\big) + \sum_{1\le i\le k, \bar{w}_i} q(\bar{w}_i) \log P_\theta(-|\bar{w}_i,h)$

Computation Techniques

- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - Binary classification with k negative samples
    - Positive data: $P(+, w|h) = \frac{1}{k+1}\tilde{P}(w|h)$
    - Negative data: $P(-, \bar{w}|h) = \frac{k}{k+1}q(\bar{w})$
    - $P(+|w,h) = \frac{\tilde{P}(w|h)}{k \cdot q(w) + \tilde{P}(w|h)}$
    - Trick#1: replace $\tilde{P}$ with learning model P_θ
    - $L(h;\theta) = \sum_w \tilde{P}(w|h) \log\big(P_\theta(+|w,h)\big) + k \cdot E_{\bar{w}\sim q}[\log P_\theta(-|\bar{w},h)]$
    - $= \sum_w \tilde{P}(w|h) \log\big(P_\theta(+|w,h)\big) + \sum_{1\le i\le k, \bar{w}_i} q(\bar{w}_i) \log P_\theta(-|\bar{w}_i,h)$
    - $= \sum_w \tilde{P}(w|h) \log\left(\frac{P_\theta(w|h)}{k \cdot q(w) + P_\theta(w|h)}\right) + \sum_{1\le i\le k, \bar{w}_i} q(\bar{w}_i) \log\left(\frac{k \cdot q(\bar{w}_i)}{k \cdot q(\bar{w}_i) + P_\theta(\bar{w}_i|h)}\right)$

Computation Techniques

- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - Binary classification with k negative samples
    - $P(+|w,h) = \frac{\tilde{P}(w|h)}{k \cdot q(w) + \tilde{P}(w|h)}$
    - Trick#1: replace $\tilde{P}$ with learning model P_θ
    - $L(h;\theta) = \sum_w \tilde{P}(w|h) \log\left(\frac{P_\theta(w|h)}{k \cdot q(w) + P_\theta(w|h)}\right) + \sum_{1 \le i \le k, \bar{w}_i} q(\bar{w}_i) \log\left(\frac{k \cdot q(\bar{w}_i)}{k \cdot q(\bar{w}_i) + P_\theta(\bar{w}_i|h)}\right)$

Computation Techniques

- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - Binary classification with k negative samples
    - $P(+|w,h) = \frac{\tilde{P}(w|h)}{k \cdot q(w) + \tilde{P}(w|h)}$
    - Trick#1: replace $\tilde{P}$ with learning model P_θ
    - $L(h;\theta) = \sum_w \tilde{P}(w|h) \log\left(\frac{P_\theta(w|h)}{k \cdot q(w) + P_\theta(w|h)}\right) + \sum_{1 \le i \le k, \bar{w}_i} q(\bar{w}_i) \log\left(\frac{k \cdot q(\bar{w}_i)}{k \cdot q(\bar{w}_i) + P_\theta(\bar{w}_i|h)}\right)$
    - Trick#2: assume $\hat{Z}(h) = 1$ for any h, which also encourages $Z_h \approx 1$ (self-normalized)
      - Further read: https://arxiv.org/abs/1206.6426 (Mnih and Teh, ICML 2012) & https://arxiv.org/abs/1410.8251
    - $L^k_{NCE}(h;\theta) = \sum_w \tilde{P}(w|h) \log\left(\frac{u_\theta(w,h)}{k \cdot q(w) + u_\theta(w,h)}\right) + \sum_{1 \le i \le k, \bar{w}_i} q(\bar{w}_i) \log\left(\frac{k \cdot q(\bar{w}_i)}{k \cdot q(\bar{w}_i) + u_\theta(\bar{w}_i,h)}\right)$

Computation Techniques

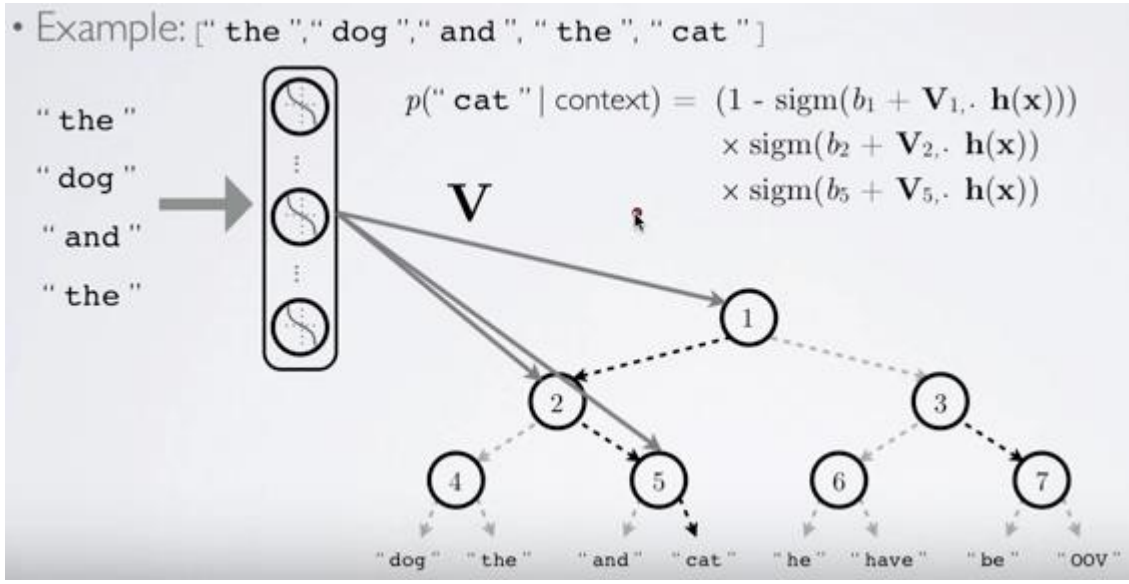
- Noise-Contrastive Estimation (Gutmann & Hyvarinen, AISTATS 2010)
  - Binary classification with k negative samples
    - $P(+|w,h) = \frac{\tilde{P}(w|h)}{k \cdot q(w) + \tilde{P}(w|h)}$
    - Trick#1: replace $\tilde{P}$ with learning model P_θ
    - $L(h;\theta) = \sum_w \tilde{P}(w|h) \log\left(\frac{P_\theta(w|h)}{k \cdot q(w) + P_\theta(w|h)}\right) + \sum_{1 \le i \le k, \bar{w}_i} q(\bar{w}_i) \log\left(\frac{k \cdot q(\bar{w}_i)}{k \cdot q(\bar{w}_i) + P_\theta(\bar{w}_i|h)}\right)$
    - Trick#2: assume $\hat{Z}(h) = 1$ for any h, which also encourages $Z_h \approx 1$ (self-normalized)
    - $L^k_{NCE}(h;\theta) = \sum_w \tilde{P}(w|h) \log\left(\frac{u_\theta(w,h)}{k \cdot q(w) + u_\theta(w,h)}\right) + \sum_{1 \le i \le k, \bar{w}_i} q(\bar{w}_i) \log\left(\frac{k \cdot q(\bar{w}_i)}{k \cdot q(\bar{w}_i) + u_\theta(\bar{w}_i,h)}\right)$
    - Remark: when $k = |V|$ and q is uniform, NCE becomes Negative Sampling
      - $L_{NS} = \sum_{(c,w) \in D} \log \frac{1}{\exp(-w^T c) + 1} + \sum_{c \in D, \tilde{w} \sim V} \log \frac{\exp(-\tilde{w}^T c)}{\exp(-\tilde{w}^T c) + 1}$
  - Claim: as $k \to \infty$, $\nabla L^k_{NCE}(h;\theta) = \nabla L_{MLE}(h;\theta)$
    - Your homework ☺ (assume $Z_h = 1$)

Computation Techniques

- Language Model Learning
  - NCE loss for efficient softmax operator with mathematical guarantees
    - $L^k_{NCE}(h;\theta) = \sum_w \tilde{P}(w|h) \log\left(\frac{u_\theta(w,h)}{k\cdot q(w)+u_\theta(w,h)}\right) + \sum_{1\le i\le k,\bar{w}_i} q(\bar{w}_i) \log\left(\frac{k\cdot q(\bar{w}_i)}{k\cdot q(\bar{w}_i)+u_\theta(\bar{w}_i,h)}\right)$
    - Special case: q is uniform
    - $L^k_{NCE}(h;\theta) = \sum_w \tilde{P}(w|h) \log\left(\frac{u_\theta(w,h)}{k/|V|+u_\theta(w,h)}\right) + \sum_{1\le i\le k,\bar{w}\sim V} \log\left(\frac{k/|V|}{k/|V|+u_\theta(\bar{w},h)}\right)$
    - *Note: NCE is **different from** negative sampling (except uniform q and $k = |V|$)*
  - Remark
    - In practice, $k\sim30$ works reasonably well for small corpus
    - Trade-off between performance and efficiency
    - At test time we still need to run full Softmax
      - Some further readings
        - https://ruder.io/word-embeddings-softmax/

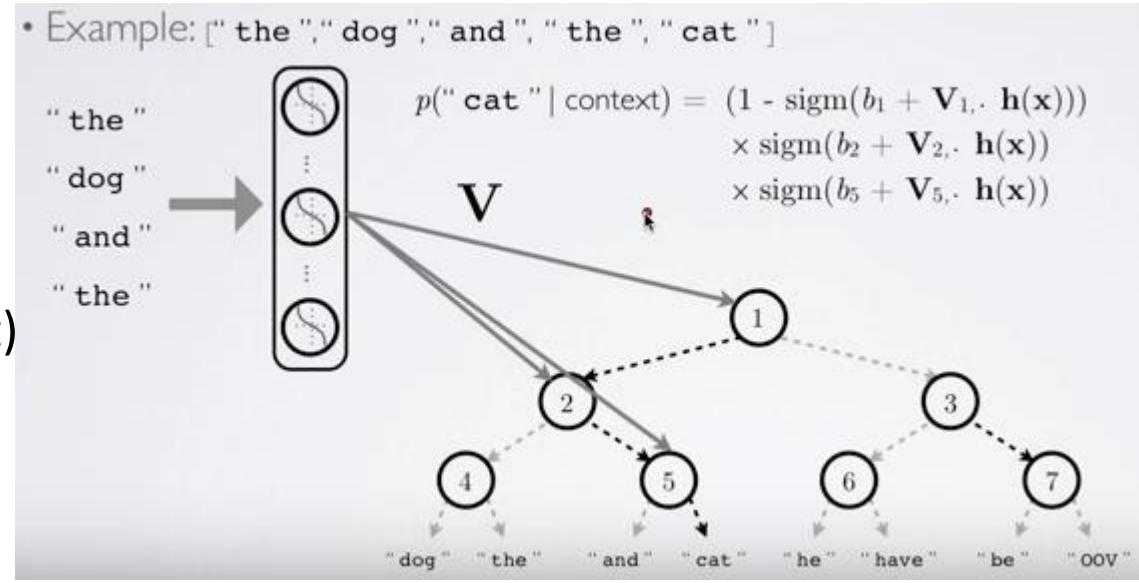
Computation Techniques

- Language Model Learning
  - NCE loss for efficient softmax operator with mathematical guarantees
    - Monte-Carlo estimates of MLE gradients
    - *Non-sampling methods?*
  - Hierarchical Softmax
    - Build a binary tree: $O(V) \rightarrow O(\log V)$
    - For node j, $P\left(\text{left}\middle|n_j, h\right) = \sigma(n_j^T h)$
    - $P(w) = \prod_j \sigma(h^T n_j)$
    - #Params = $2V$
    - $2V$ operators to calculate all probabilities
    - Remark:
      - Each word has different frequency
      - *Optimal tree structure?*



Computation Techniques

- Language Model Learning
  - NCE loss for efficient softmax operator with mathematical guarantees
    - Monte-Carlo estimates of MLE gradients
    - Non-sampling methods?
  - Hierarchical Softmax
    - Computation cost $H = \sum_w P(w)I(w)$
      - H is also referred to as entropy
    - $P(w)$: the frequency of word w
    - $I(w)$: the tree depth (or information content)
      - In a complete binary tree, $I(w) = \log_2 V$
    - The optimal tree structure is Huffman tree!
    - We can also utilize semantic information
      - E.g., A Scalable Hierarchical Distributed Language Model. Mnih & Hinton, NIPS2008
      - https://papers.nips.cc/paper/2008/file/1e056d2b0ebd5c878c550da6ac5d3724-Paper.pdf



Advanced Techniques

- Language Model Learning
  - NCE loss: noise contrastive loss
    - Sampling-based gradient approximation
  - Hierarchical Softmax
    - Non-sampling, tree-based probability computation
  - Remark
    - Use full softmax when possible for the best performance (GPU memory allowed)
- Q: what if we want the *best* output?
  - $X = \arg\max_X P(X)$
  - Greedy solution: for each $P(X_t|X_{i<t})$, select the optimal X_t
    - *Optimal?*

Advanced Techniques

- Language Model Inference
  - Goal: find $X^\star = \arg\max_X P(X) = \prod_t P(X_t|X_{i<t})$
  - Greedy Solution:
    - For each t, $X_t^\star = \arg\max_{X_t} P(X_t|X_{i<t}^\star)$ (i.e., keep the best partial candidate)
  - Better Solution: Beam Search
    - Idea: keep top K candidates for each t
      - K is called the beam size (in practice $k = 5\sim10$)
    - At each time step t, compute K^2 expansions and keep the top K for $t+1$
      - For each candidate $\tilde{X}_{i<t}$, find the top-K X_t based on $P(X_t|\tilde{X}_{i<t})$
      - Rank K^2 candidates by their partial probability $P(\tilde{X}_{i\leq t})$
    - No guarantee to find the optimal solution
      - Trade-off between accuracy (exhaustive search) and efficiency (greedy)

Advanced Techniques

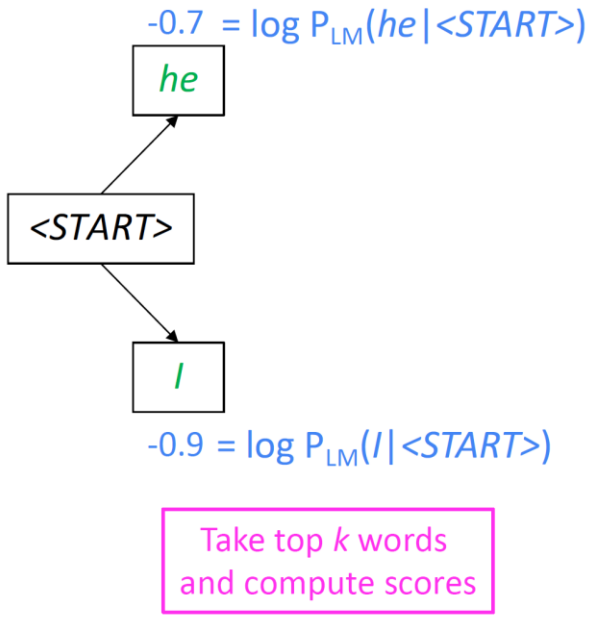
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$

<*START*>

Calculate prob
dist of next word

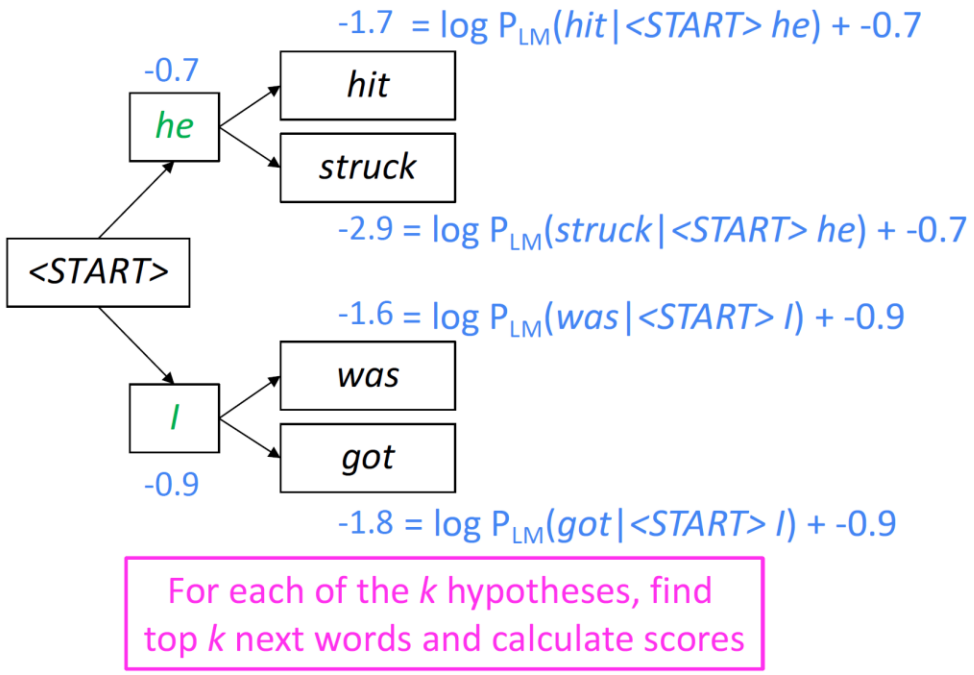
Advanced Techniques

- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



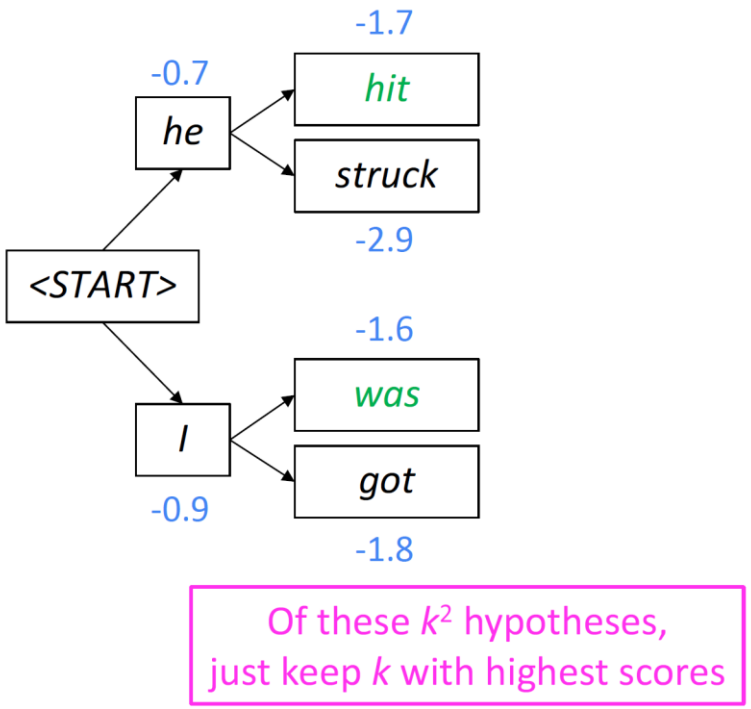
Advanced Techniques

- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



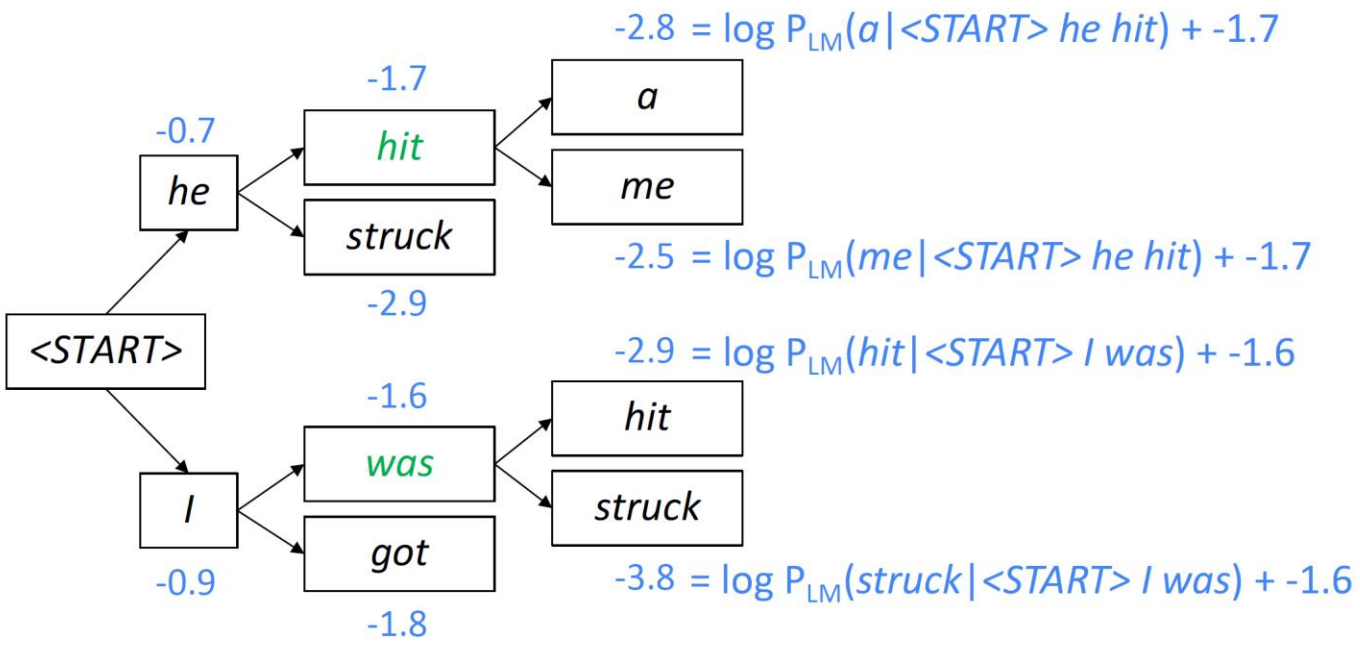
Advanced Techniques

- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



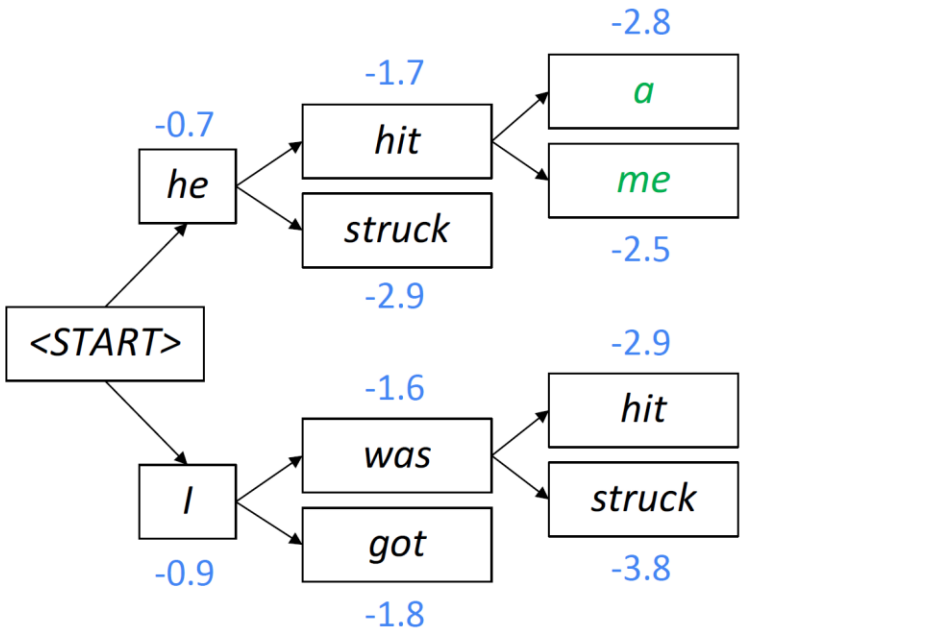
Advanced Techniques

- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



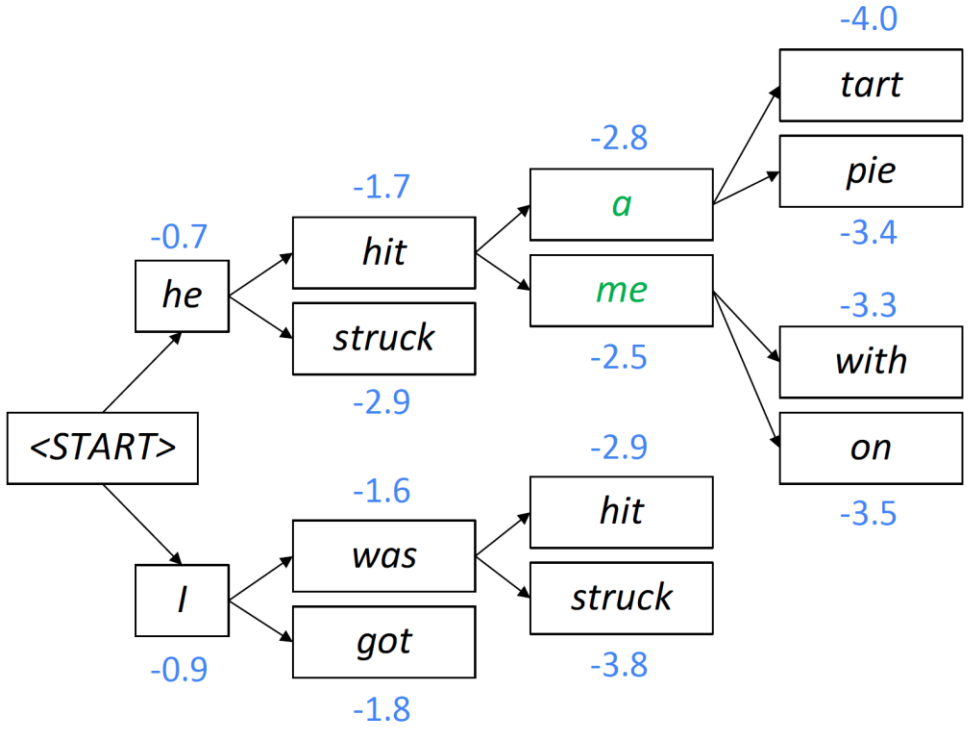
Advanced Techniques

- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



Advanced Techniques

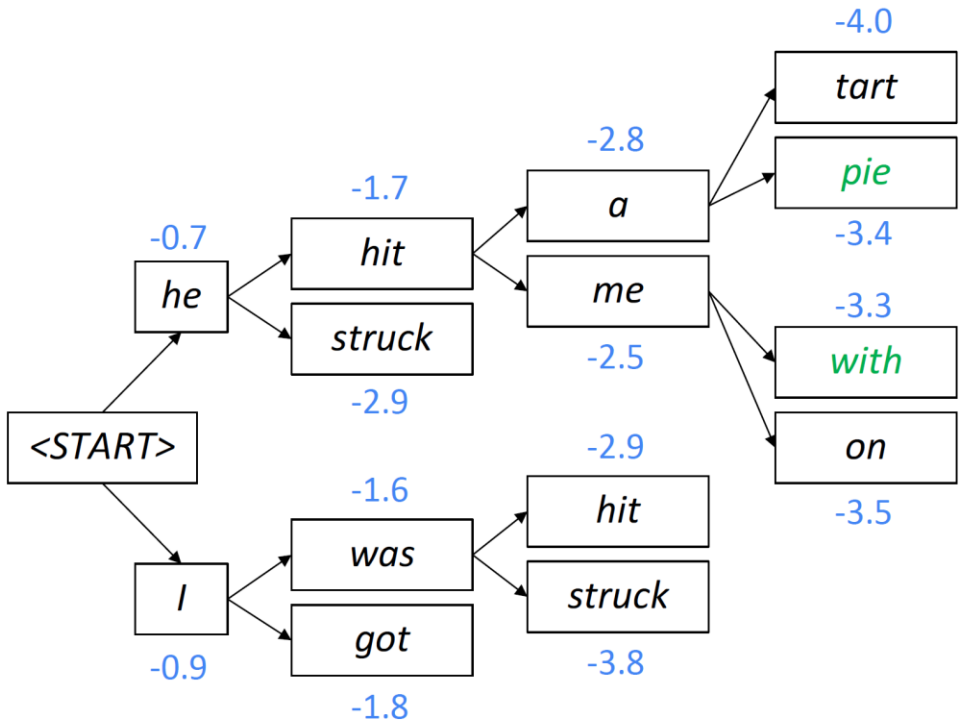
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Advanced Techniques

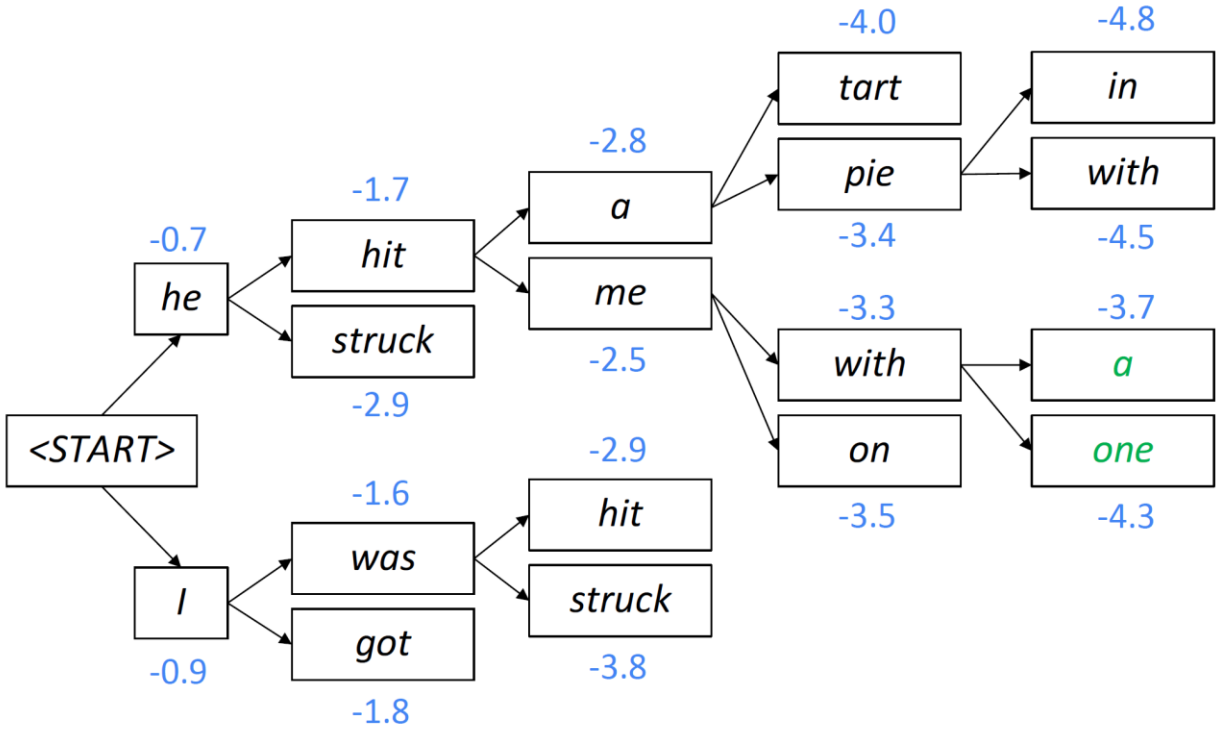
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



Of these k^2 hypotheses,
just keep k with highest scores

Advanced Techniques

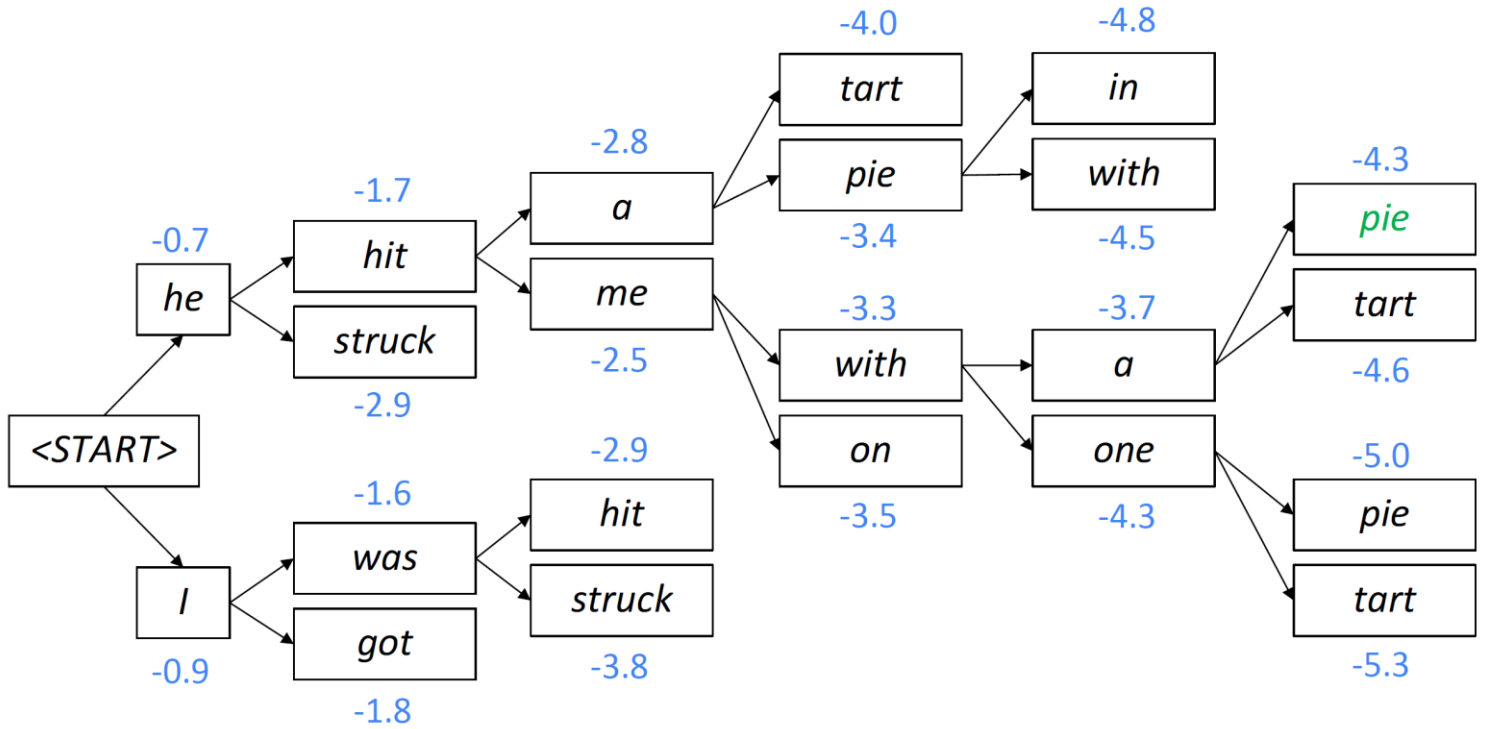
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



Of these k^2 hypotheses,
just keep k with highest scores

Advanced Techniques

- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



This is the top-scoring hypothesis!

Advanced Techniques

- Language Model Inference
  - Goal: find $X^\star = \arg\max_X P(X) = \prod_t P(X_t|X_{i<t})$
  - Greedy Solution:
    - For each t, $X_t^\star = \arg\max_{X_t} P(X_t|X_{i<t}^\star)$ (i.e., keep the best partial candidate)
  - Better Solution: Beam Search
    - Idea: keep top K candidates for each t
    - When to terminate (sequences may have varying lengths)?
      - We typically include a <end> token to indicate a text sequence is ended
      - $L_{\max}$ words reached or n completed sequences obtained (<end> token produced)
    - Which sequence to choose?
      - Issue: longer sequences tend to have lower scores!
      - Adjusted metric: $X^\star = \arg\max_X \frac{1}{L_X}\sum_t \log P(X_t|X_{i<t})$ (normalized by its length)

Summary

- Recurrent neural network (RNN)
  - A neural network for sequence data
  - Vanishing/exploding gradients/value
  - Gradient clipping!
- Long Short-Term Memory networks (LSTM)
  - An RNN architecture for long-term dependency
- Language Model
  - Auto-regressive model over texts & LSTM applications
  - Computation Techniques: NCE & Hierarchical Softmax
  - Beam search for the best output
- Next lecture: more advanced sequence modeling techniques

Thanks!