Deep Learning lecture 8 Sequence Modeling (1)

Yi Wu, IIIS

Spring 2025

Apr-7

Today's Topic

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

- Basic techniques for modeling natural language

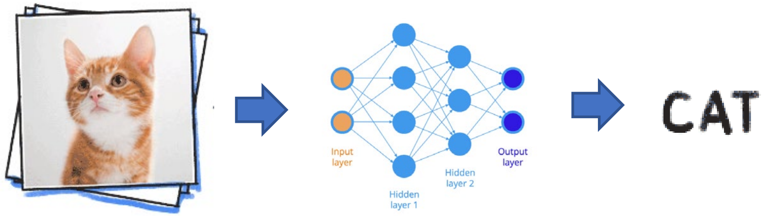
Story So Far

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

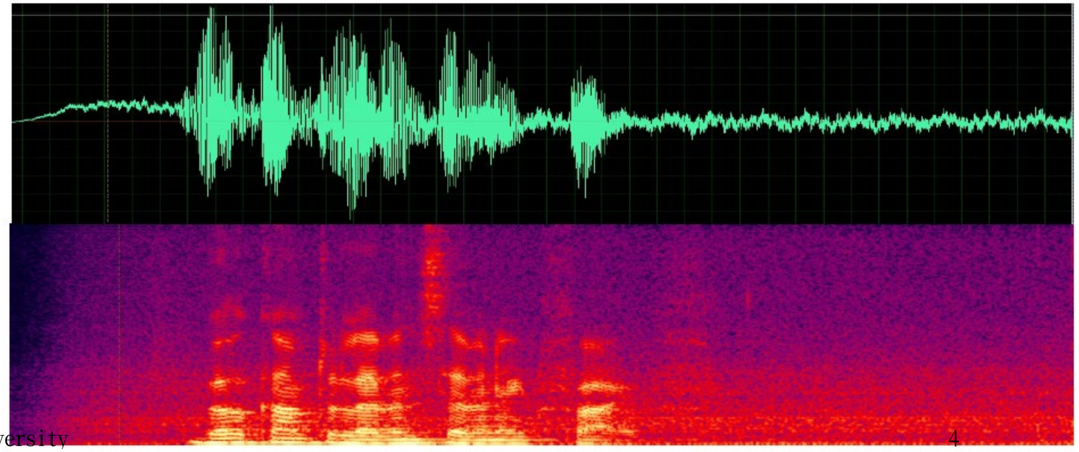
- Generative Models (Lec. 4~7)
  - Energy-based models (contrastive divergence + MCMC)
  - VAE (variational inference)
  - GAN (neural loss function)
  - Flow model (bijections)
  - Diffusion model (denoising score matching)
  - Trade-offs between expressiveness, inference and training

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

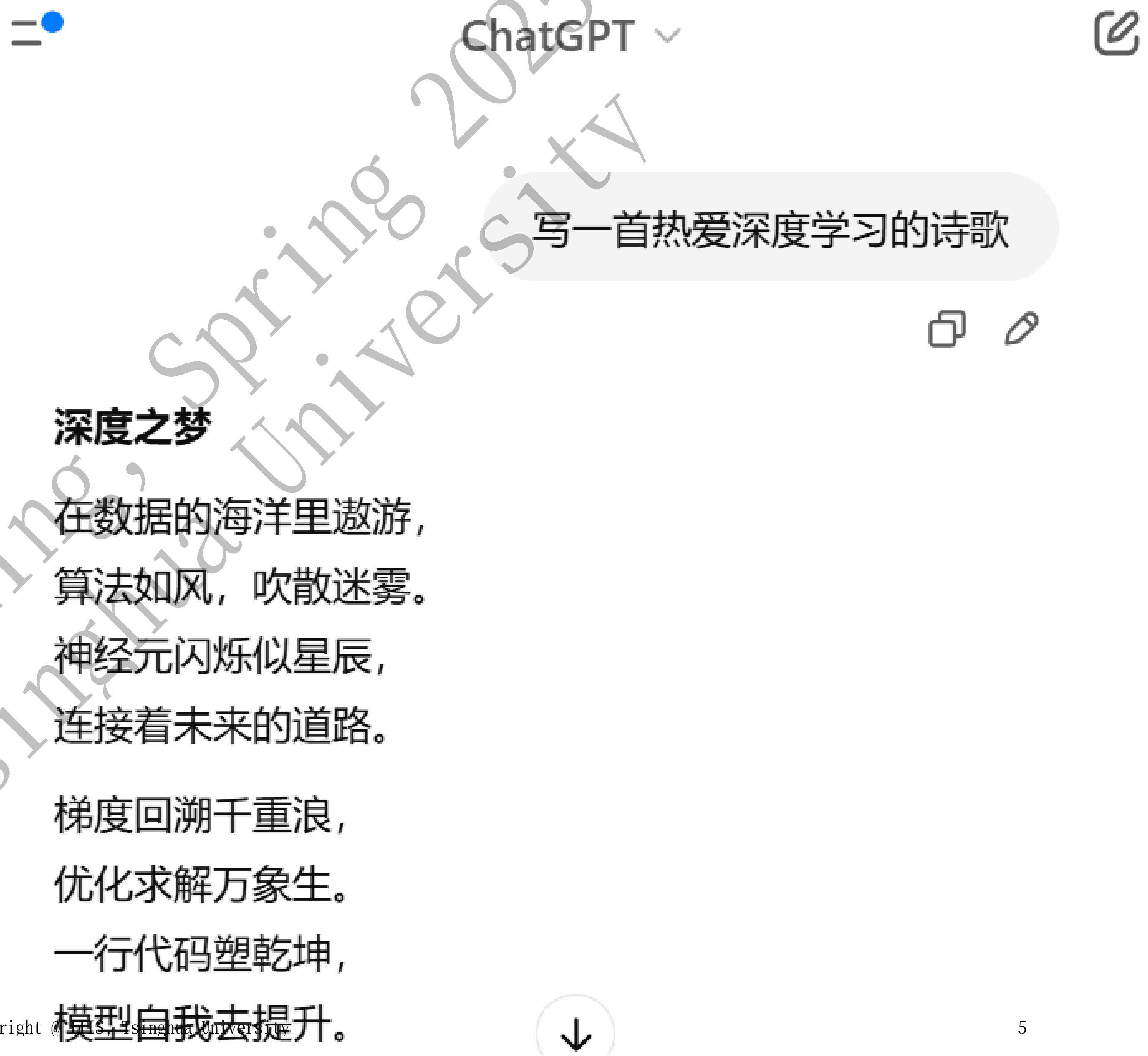


- What if the dimension of input varies a lot?
  - Finding the "welcome" (lecture 2)



Sequence Data

- Most existing discussions assu
  - E.g.: Image classification and g
    - Input image has fixed width and
    - Fixed output dimension
  - Fixed amount of network layer
- What if the dimension of inpu
  - Finding the “welcome” (lectur
  - 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

- Most existing discussions assume fixed dimensions
  - E.g.: Image classification and generation
    - Input image has fixed width and height

We need a generative model for any data dimension!

  - Generating poet
  - Machine translation

Autoregressive Model

- Goal: a tractable $p(x)$ for x of any dimension L
  - In particular, we consider sequential data $x = [x_1, x_2, \ldots, x_L]$, L may change

- Autoregressive modeling

$$p(x) = \prod_{1 \le i \le L} p(x_i | x_1 \ldots x_{i-1})$$

- Key idea: decompose a joint sequence into ordered conditionals
  - Use previous dimensions to "*predict*" the next dimension
  - Example: Gaussian auto-regressive models

$$p(x_i | x_1 \ldots x_{i-1}) \sim N(\mu_\theta(x_1 \ldots x_{i-1}), \sigma_\theta^2(x_1 \ldots x_{i-1}))$$

Autoregressive Model

- Goal: a tractable $p(x)$ for x of any dimension L
  - In particular, we consider sequential data $x = [x_1, x_2, \ldots, x_L]$, L may change
- Autoregressive modeling

$$p(x) = \prod_{1 \le i \le L} p(x_i | x_1 \ldots x_{i-1})$$

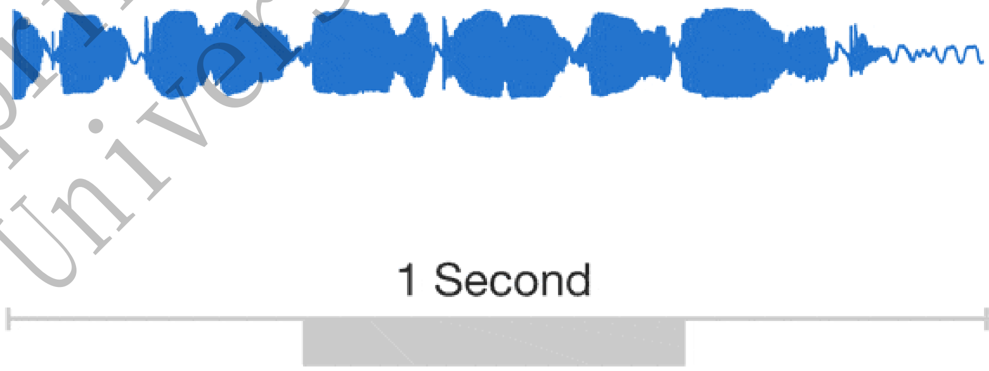
- Key idea: decompose a joint sequence into ordered conditionals
  - Use previous dimensions to "*predict*" the next dimension
  - Example: Gaussian auto-regressive models

$$p(x_i | x_1 \ldots x_{i-1}) \sim N(\mu_\theta(x_1 \ldots x_{i-1}), \sigma_\theta^2(x_1 \ldots x_{i-1}))$$

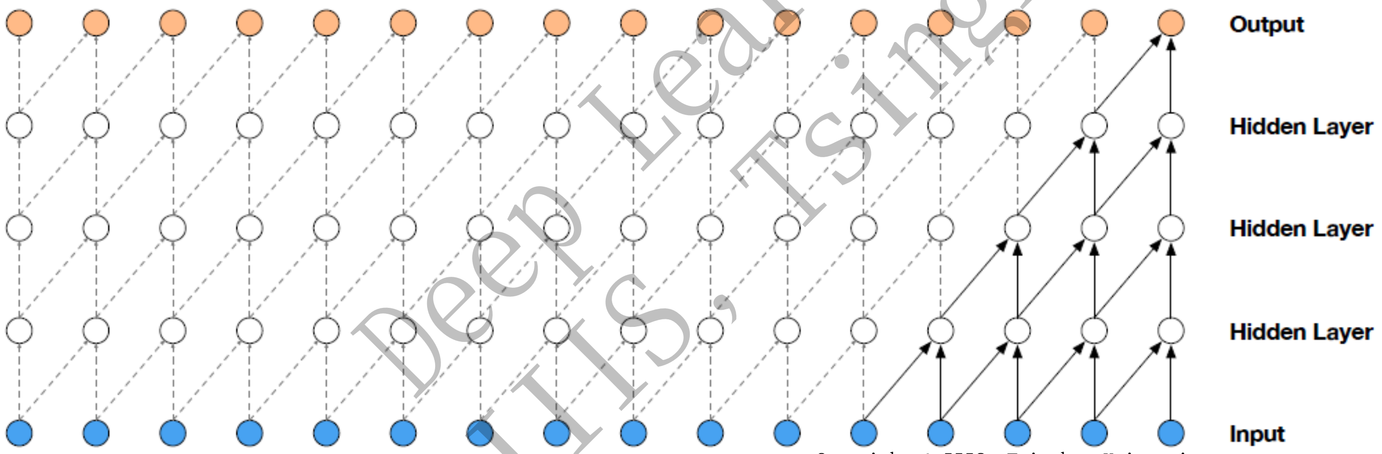
How to design these two networks???

Temporal Convolution

- WaveNet (DeepMind, 2016)
  - Goal: voice synthesis
  - $p(x) = \prod_i p(x_i | x_1, \ldots, x_{i-1})$
  - Idea: temporal convolution
    - **Q: how many layers do you need?**
    - **A: Kernel size d, we need L/d layers**



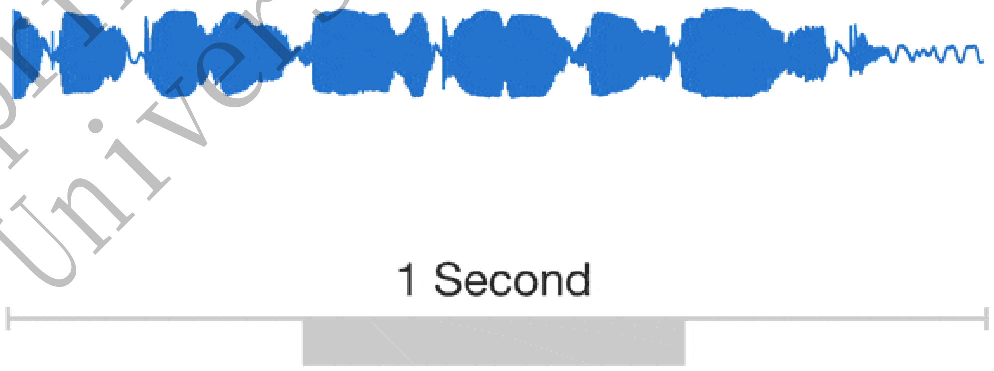
$p(x_i | x_{<i})$

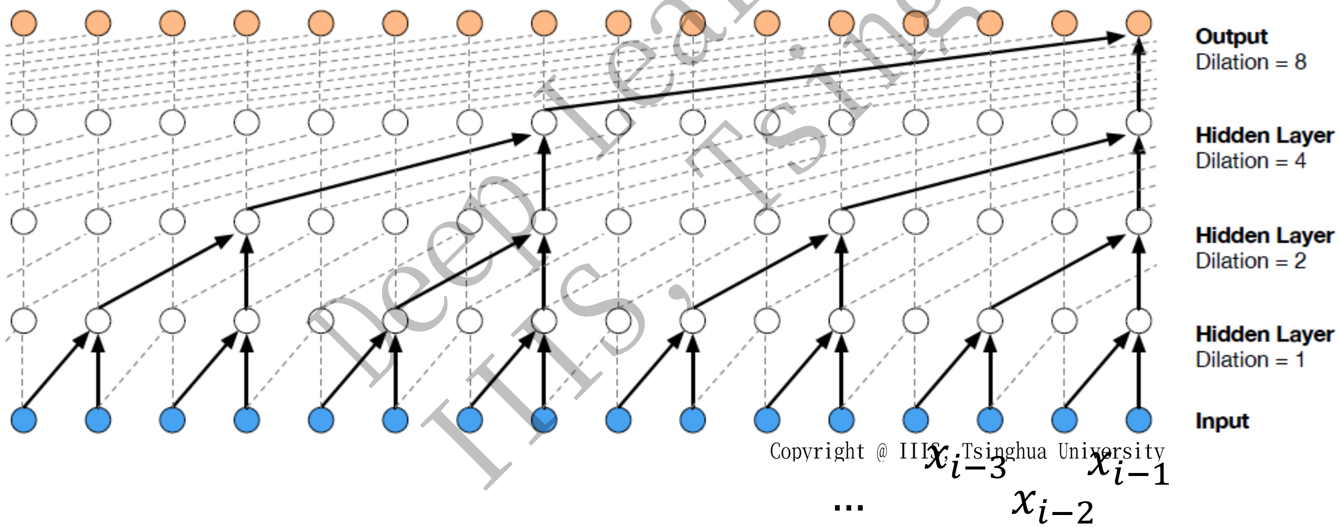


x_{i-3} x_{i-2} x_{i-1} …

Temporal Convolution

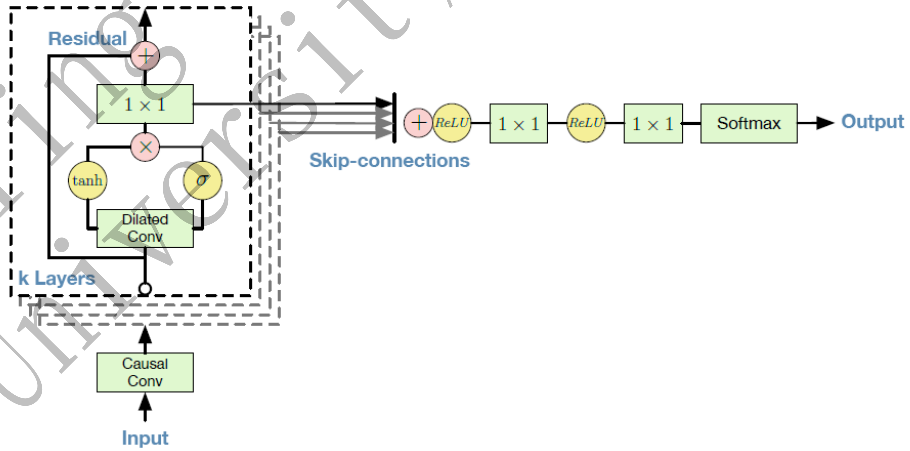
- WaveNet (DeepMind, 2016)
  - Goal: voice synthesis
  - $p(x) = \prod_i p(x_i | x_1, \ldots, x_{i-1})$
  - Idea: temporal convolution
    - **Dilated Convolution!**
    - $O(\log L)$ layers will be sufficient





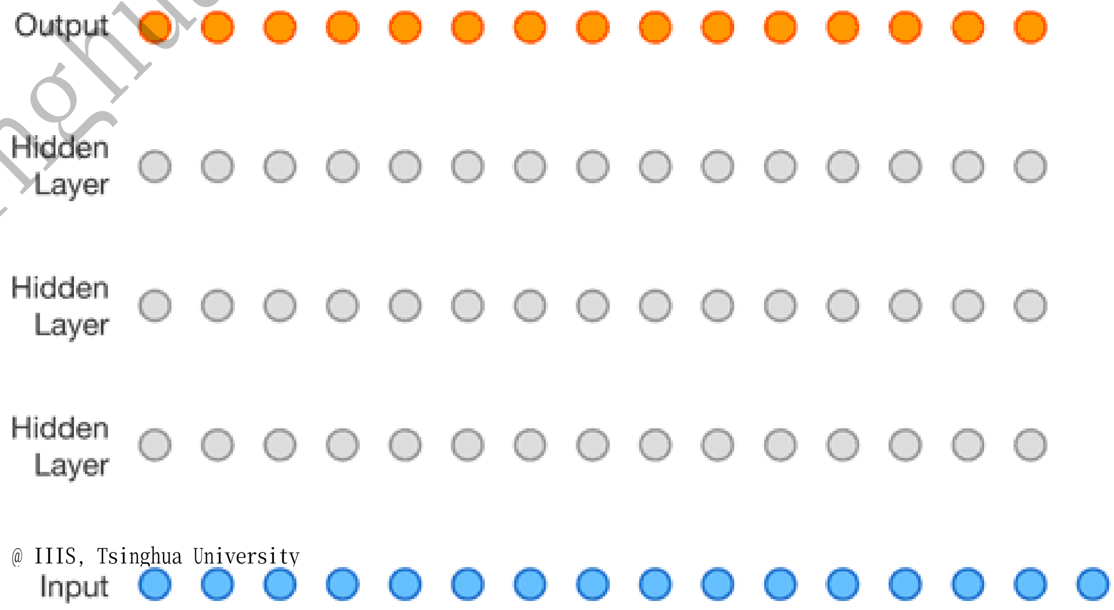
Computation Cost
- Generation
  - Sequential: $O(L)$
- Likelihood:
  - fully parallel (CNN)

Temporal Convolution



- WaveNet (DeepMind, 2016)
  - $p(x) = \prod_i p(x_i | x_1, \dots, x_{i-1})$
  - Dilated Temporal Convolution
  - And more
    - Quantization
    - Residual connection
    - Conditioned generation
      - $p(x|h)$
  - Remark
    - First deep generative model that can generate raw signals

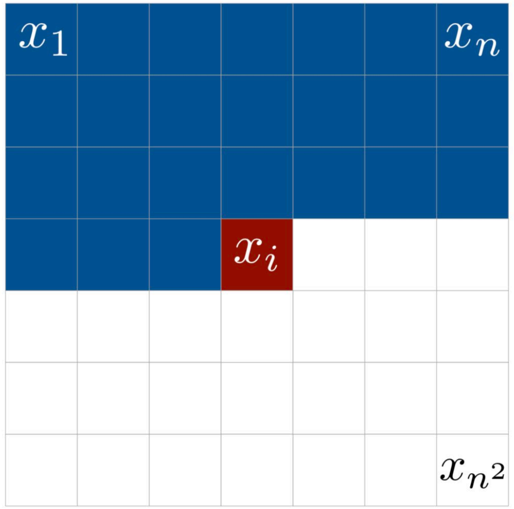
(also check newer ones Jukebox & Suno)



https://deepmind.com/blog/article/wavenet-generative-model-raw-audio
https://openai.com/blog/jukebox/
https://www.suno.ai/

Autoregressive Model for Images

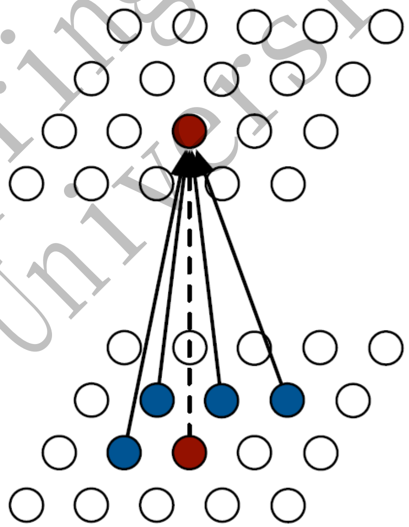
- PixelCNN (DeepMind, ICML 2016)
  - Autoregressive model over images
  - $p(x) = \prod_i^{D^2} p(x_i | x_1, \ldots, x_{i-1})$
  - CNN?
    - How to design the convolution filter?
    - Goal: the convolution filter only takes in previous values

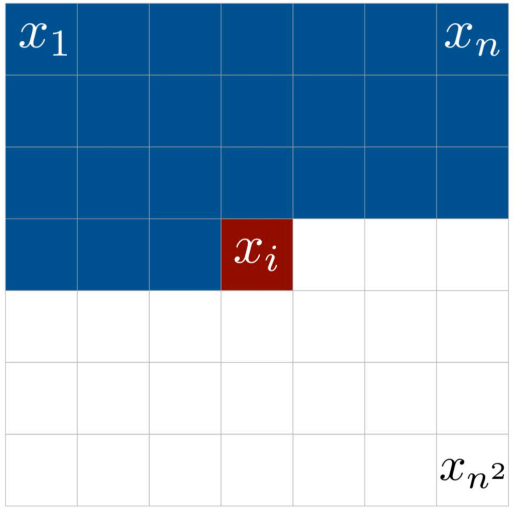


Context

Autoregressive Model for Images

- PixelCNN (DeepMind, ICML 2016)
  - Autoregressive model over images
  - $p(x) = \prod_i^{D^2} p(x_i | x_1, \dots, x_{i-1})$
  - **Masked Convolution**
    - Each pixel only takes in previous values
  - Likelihood evaluation is in perfect parallel
  - Issues?
    - Receptive fields have blind spots!

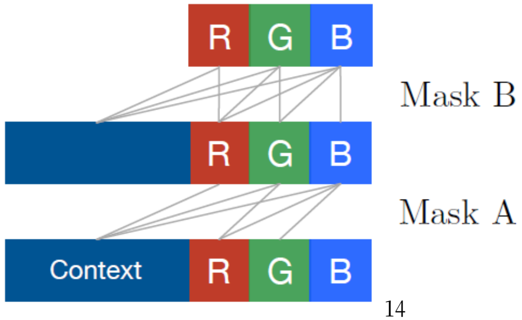


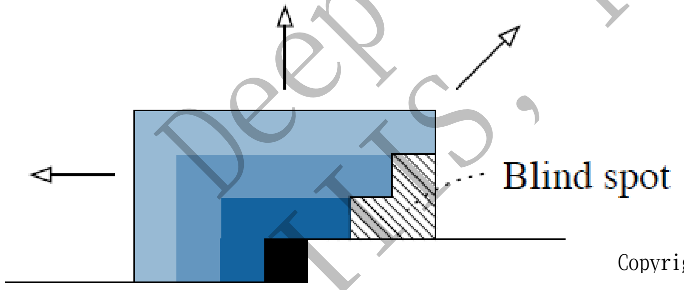


Context

1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

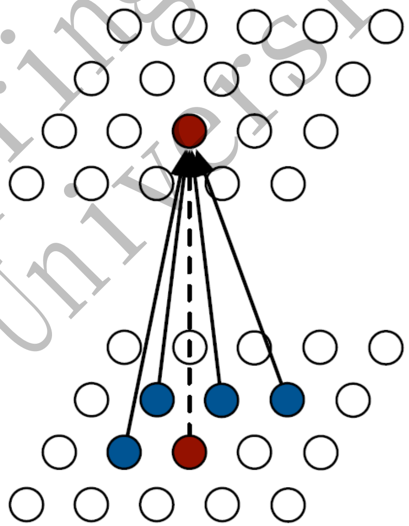
Mask

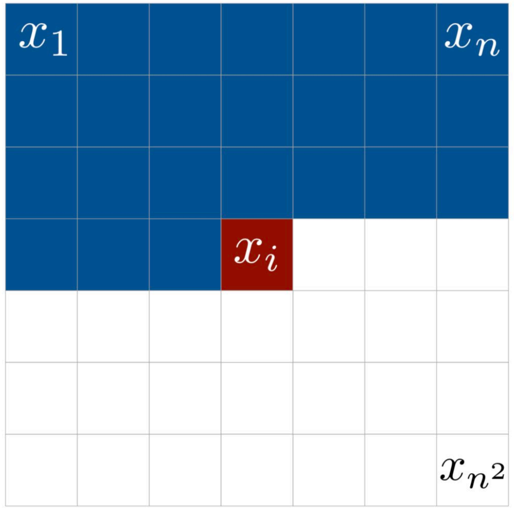




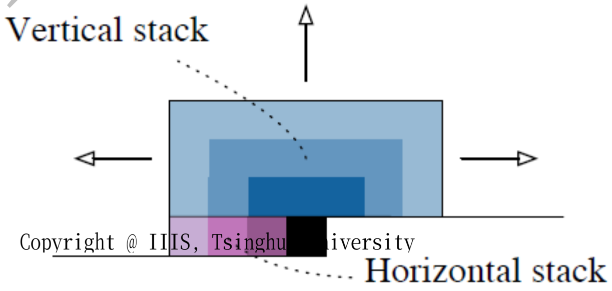
Autoregressive Model for Images

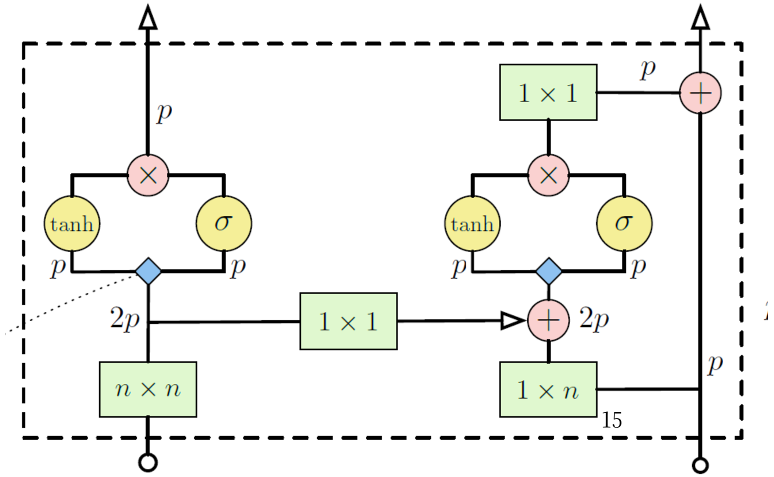
- PixelCNN (DeepMind, ICML 2016)
  - Autoregressive model over images
  - $p(x) = \prod_i^{N^2} p(x_i | x_1, \dots, x_{i-1})$
  - **Masked Convolution**
    - Each pixel only takes in previous values
  - Likelihood evaluation is in perfect parallel





- Gated PixelCNN (DeepMind, NIPS 2016)
  - Corrected receptive field (homework ☺)
  - Gated convolution
    - "Gating" technique
    - Inspired by LSTM
      - More details later

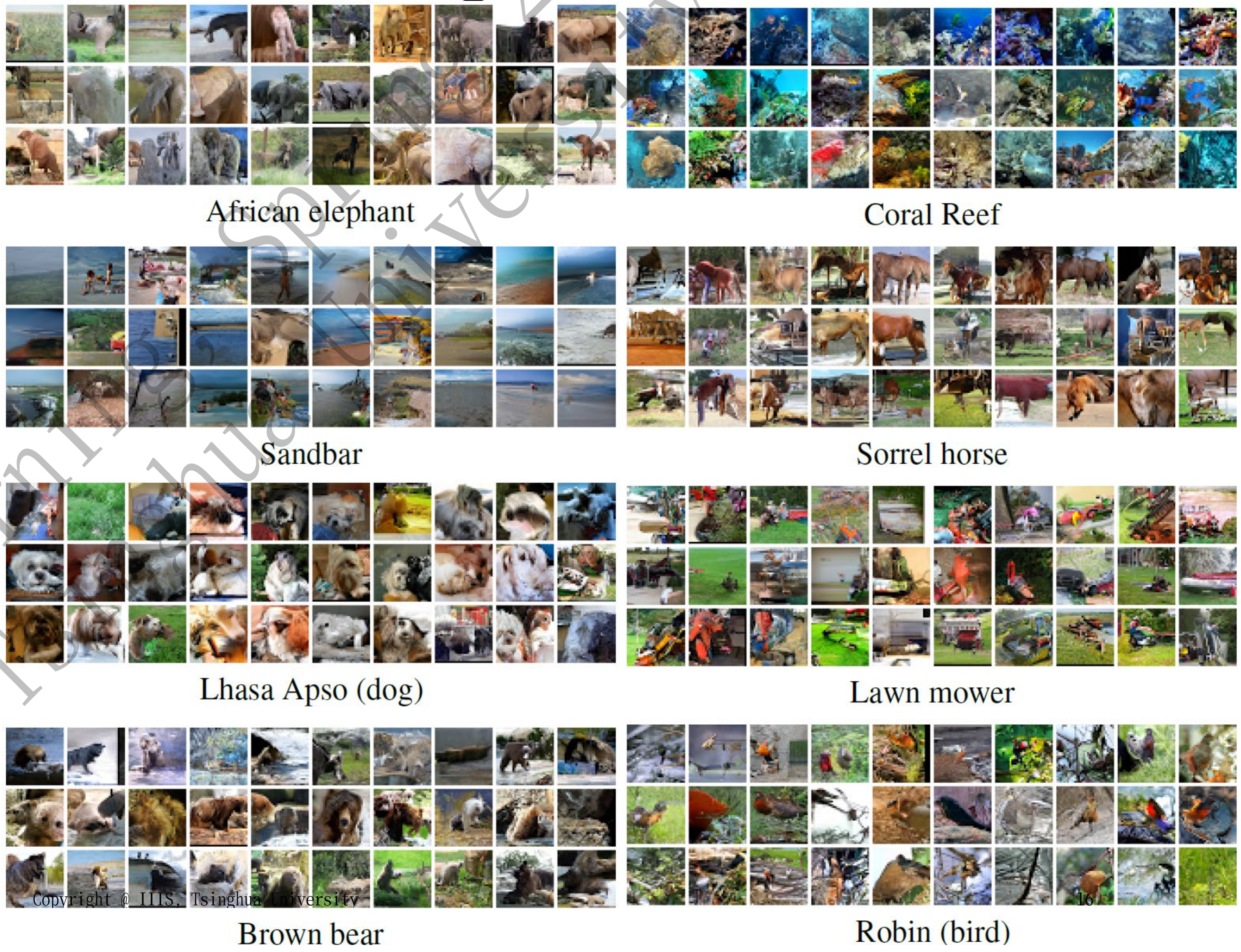




Autoregressive Model for Images

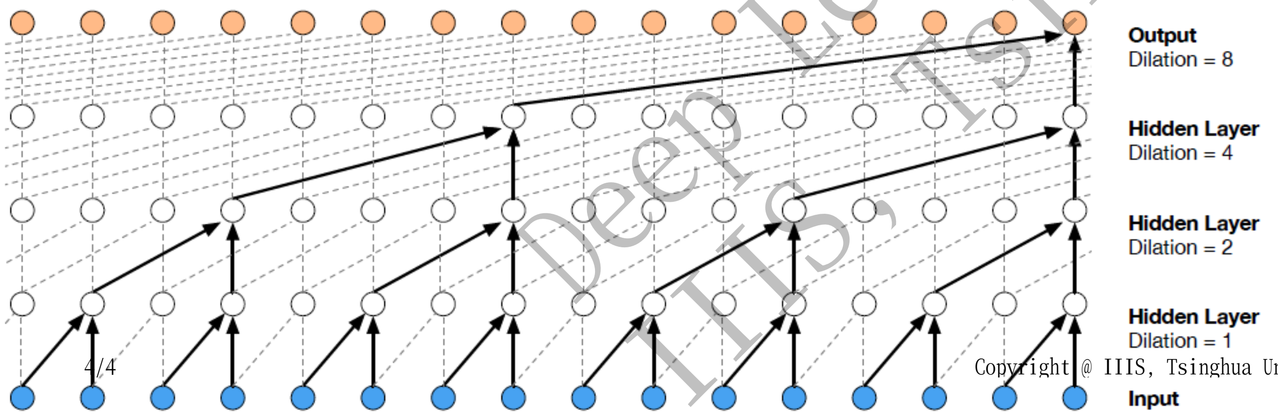
- Conditioned generation

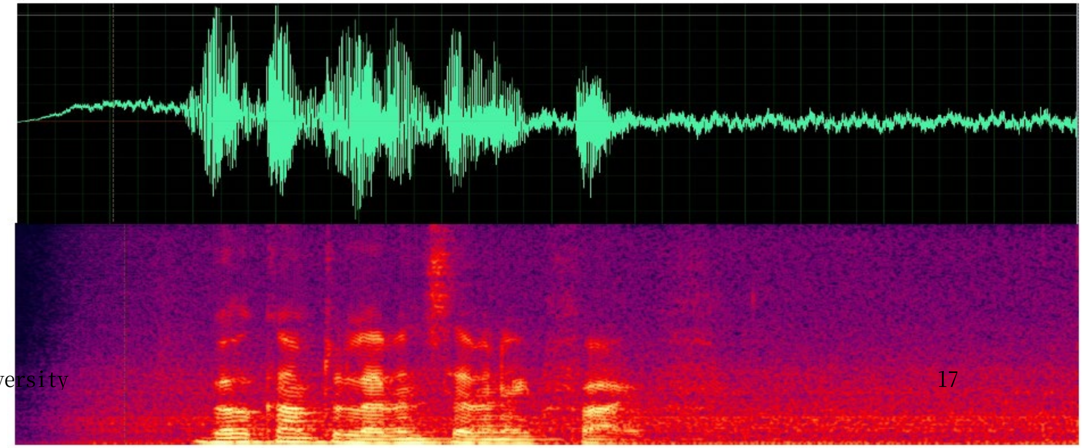
Gated PixelCNN



Sequence Data (Recap)

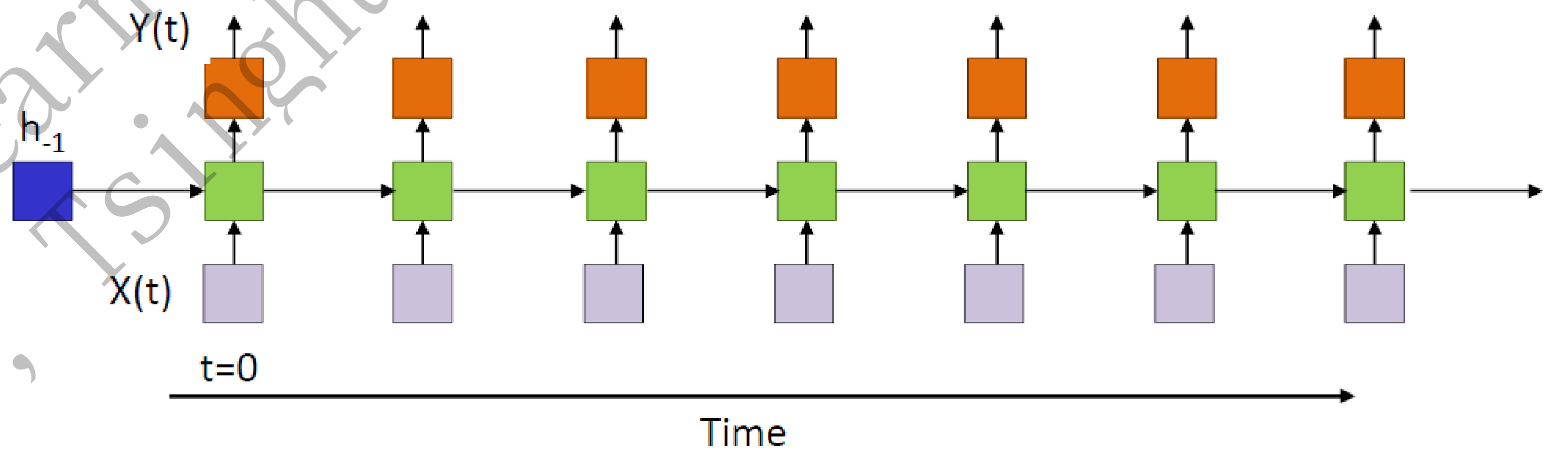
- Finding or synthesizing the “Welcome” voice
    - Input data $x = [x_1 \dots x_L]$, L may vary
    - Autoregressive modeling: $p(x) = \prod_i p(x_i | x_1 \dots x_{i-1})$
- Improved Temporal Convolution: Dilated ConvNet (WaveNet)
    - Parameter size is fixed, but need $O(\log L)$ layers to cover the entire sequence
        - This is a network of **varying/unbounded depth** for arbitrarily long sequences





State-Space Model

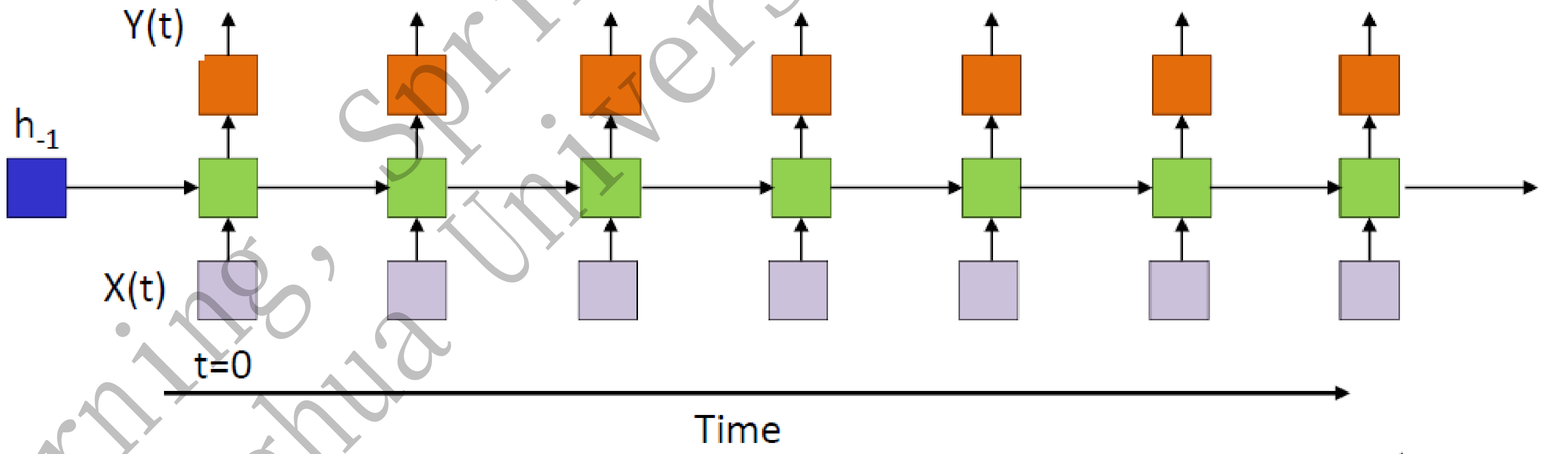
- Finding the “Welcome”
  - Input sequence data $X_1 \ldots X_L$, L may vary (X_i can be a general vector)
  - Whether the voice contains “Welcome”
- Goal: a fixed-size model for arbitrarily long sequences
- State-Space Model
  - h_t: (hidden) state
  - X_t: input
  - Y_t: output
  - $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
  - h_{-1}: initial state



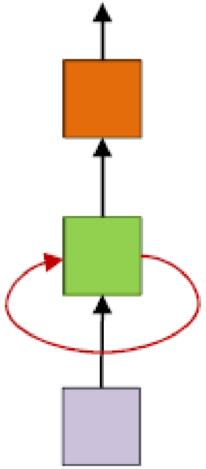
Key idea: compress any prefix sequence $[x_0 \ldots x_t]$ into a fixed dimension vector h_t

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

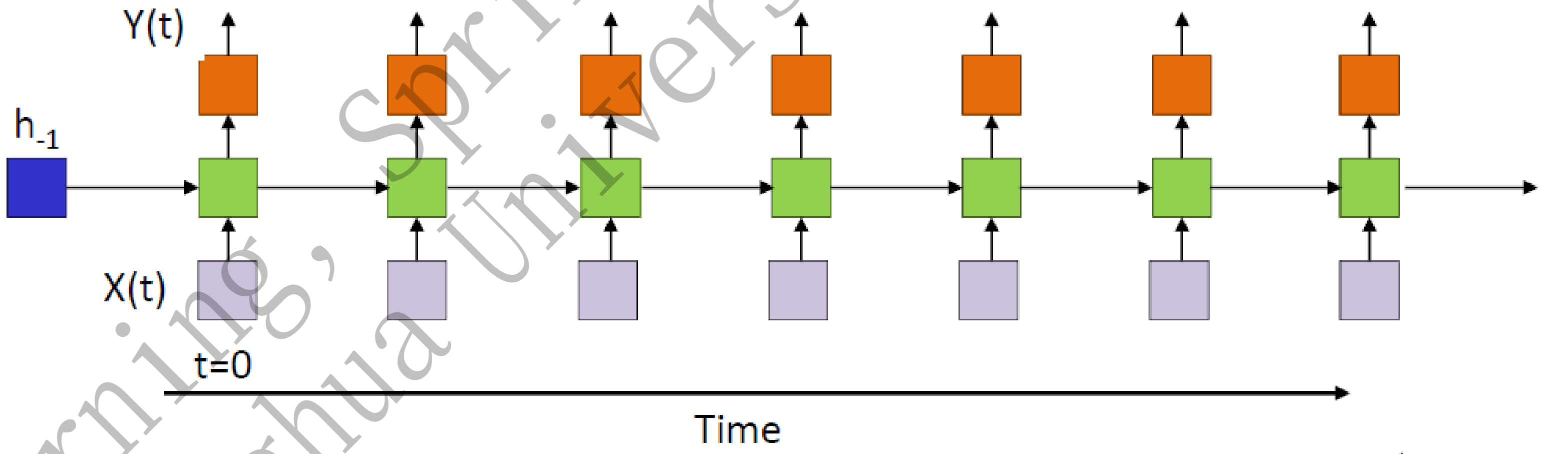


- Same neural network across all the columns!
  - Simplified drawing (loops implies recurrence)
  - h_t: a vector that summarizes all past inputs (also called "memory")
  - h_{-1} affects the whole network (typically set to zero)
  - Y_t is computed over $X_0, \dots, X_t$
  - X_t affect all the outputs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

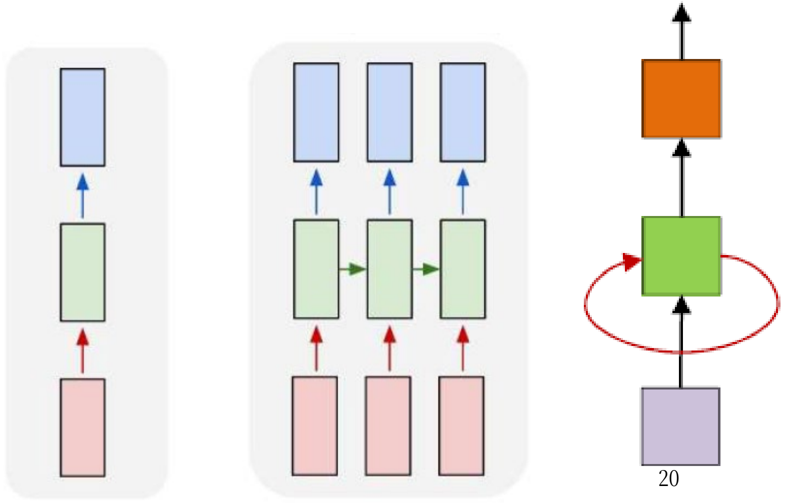


Recurrent Neural Network

- 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

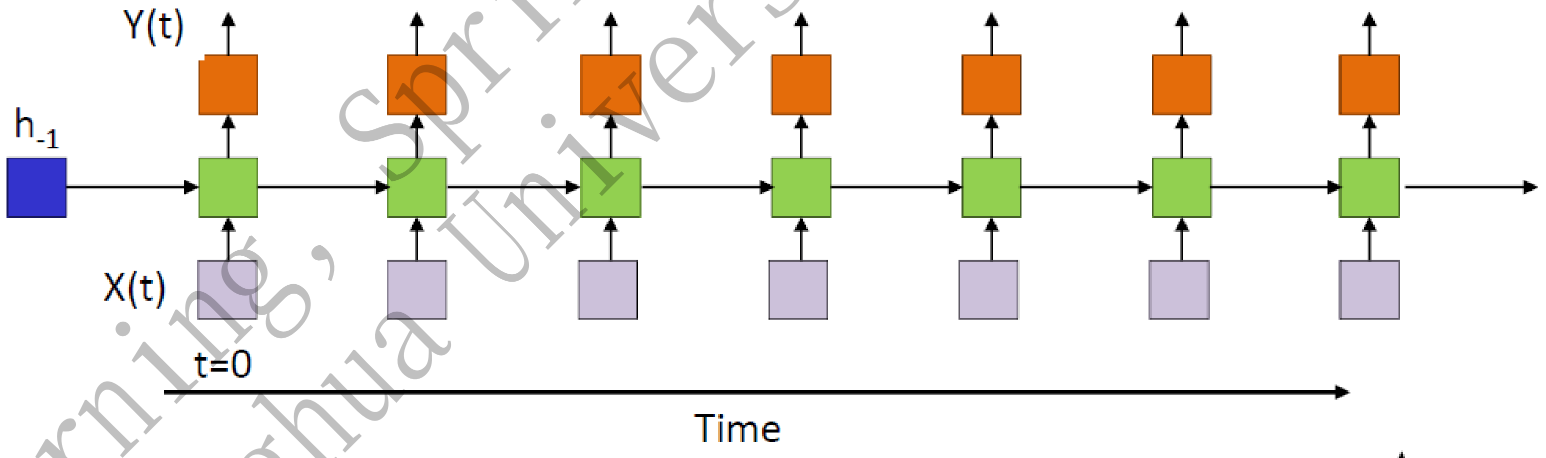


- Same neural network across all the columns!
  - MLP v.s. RNN
    - RNN can be viewed as repeatedly applying MLPs
    - $h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$
    - $Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$
    - f_1, f_2 are activations (e.g., Sigmoid, tanh,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



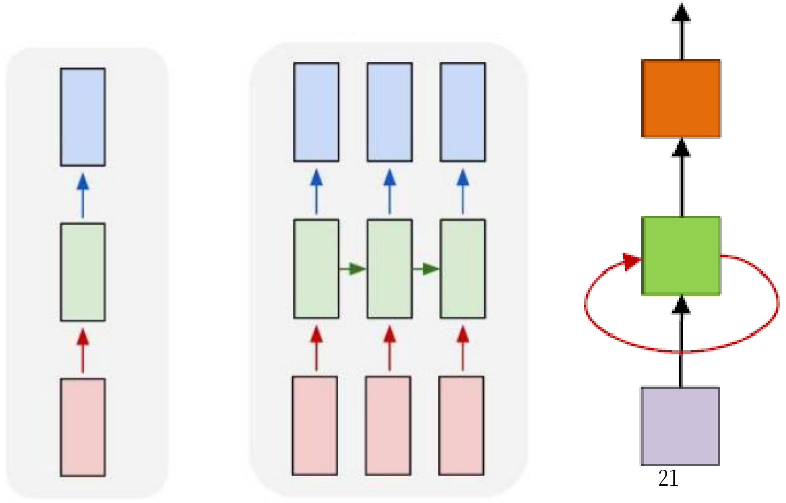
Recurrent Neural Network

- 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



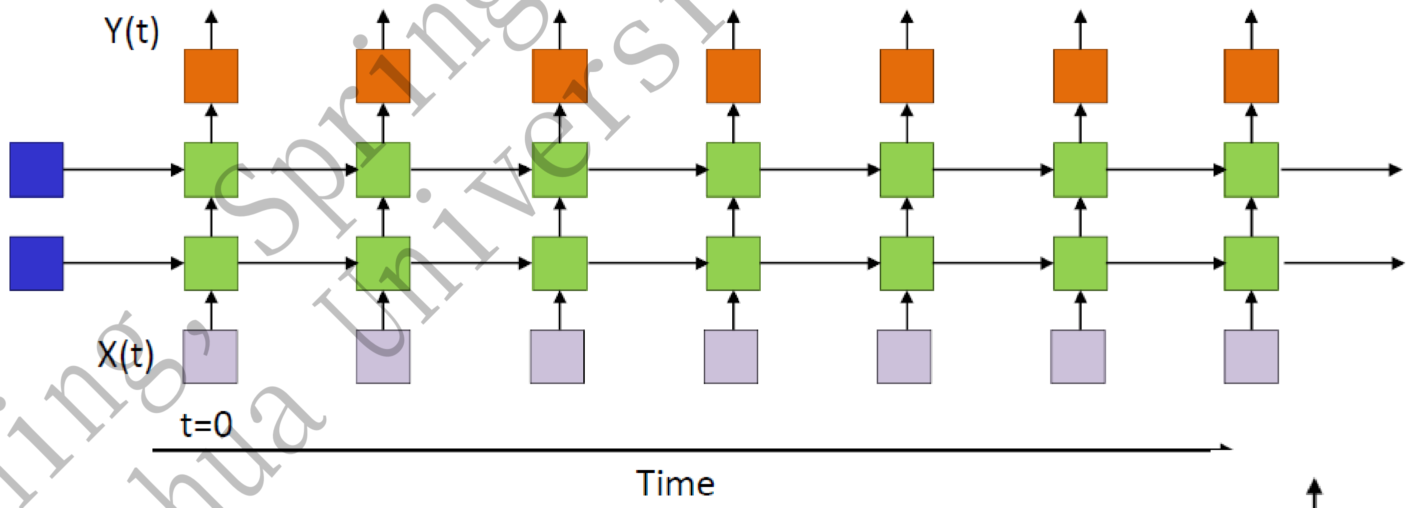
- Same neural network across all the columns!
  - MLP v.s. RNN
    - RNN can be viewed as repeatedly applying MLPs
    - $h_t = f_1\left(W^{(1)} \cdot X_t + W^{(11)} \cdot h_{t-1} + b^{(1)}\right)$
    - $Y_t = f_2\left(W^{(2)} h_t + b^{(2)}\right)$
    - f_1, f_2 are activations (e.g., Sigmoid, tanh,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

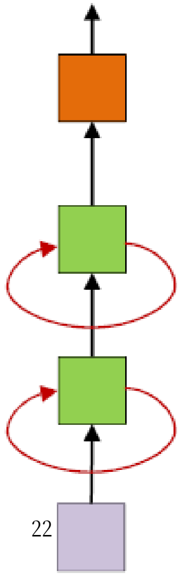
Recurrent weights!



Recurrent Neural Network

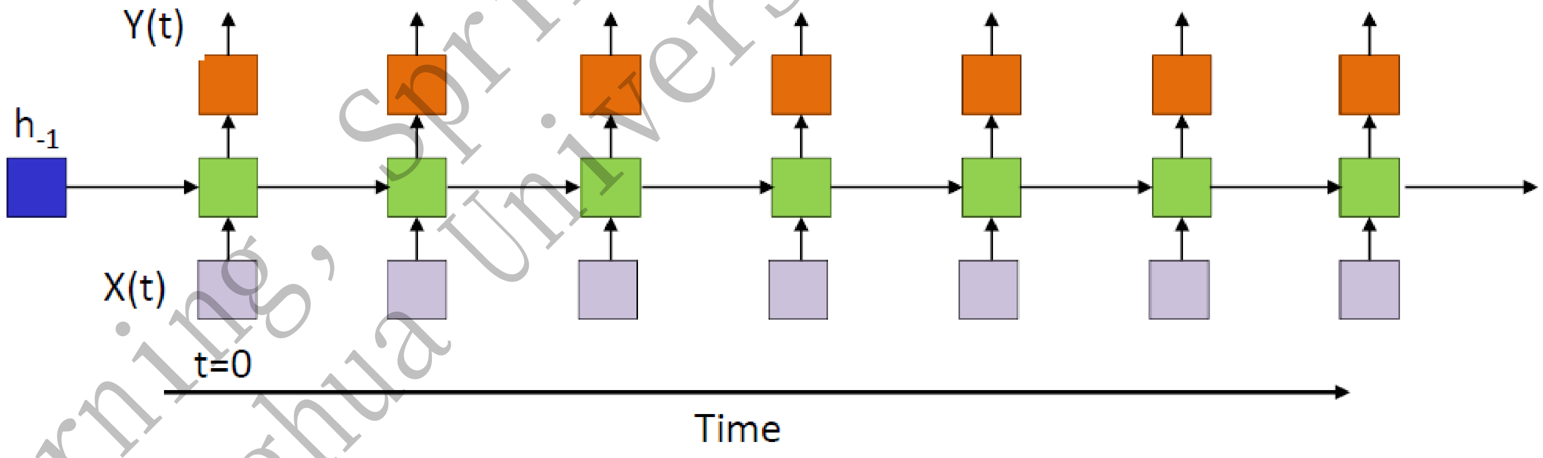
- Stack K layers of RNNs!
  - Multi-layer RNN
- State-Space Model
  - $h_t^{(k)}$: (hidden) states
  - X_t: input; Y_t: output
  - $h_t^{(1)} = f_1^{(1)}\left(h_{t-1}^{(1)}, X_t; \theta\right)$
  - $h_t^{(k)} = f_1^{(k)}\left(h_{t-1}^{(k)}, h_t^{(k-1)}; \theta\right)$
  - $Y_t = f_2\left(h_t^{(K)}; \theta\right)$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





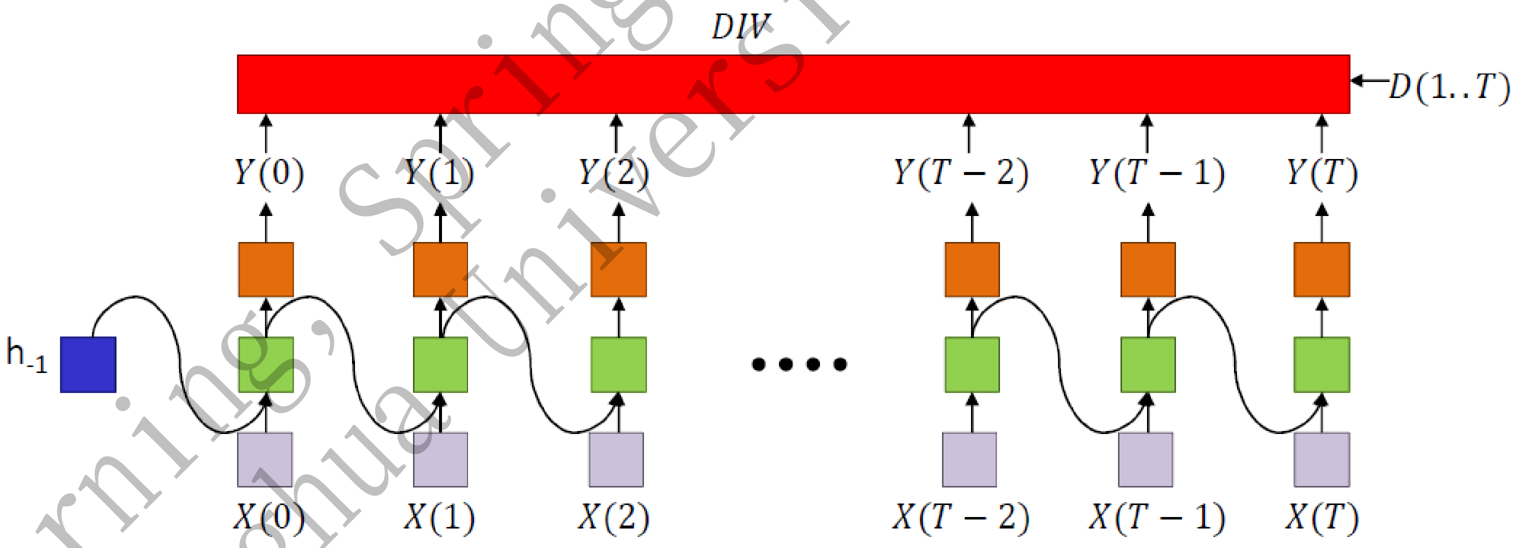
Recurrent Neural Network

- 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

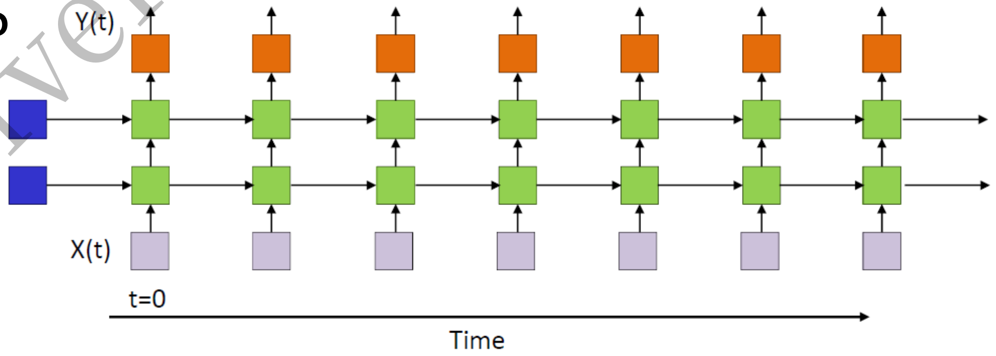
Recurrent Neural Network

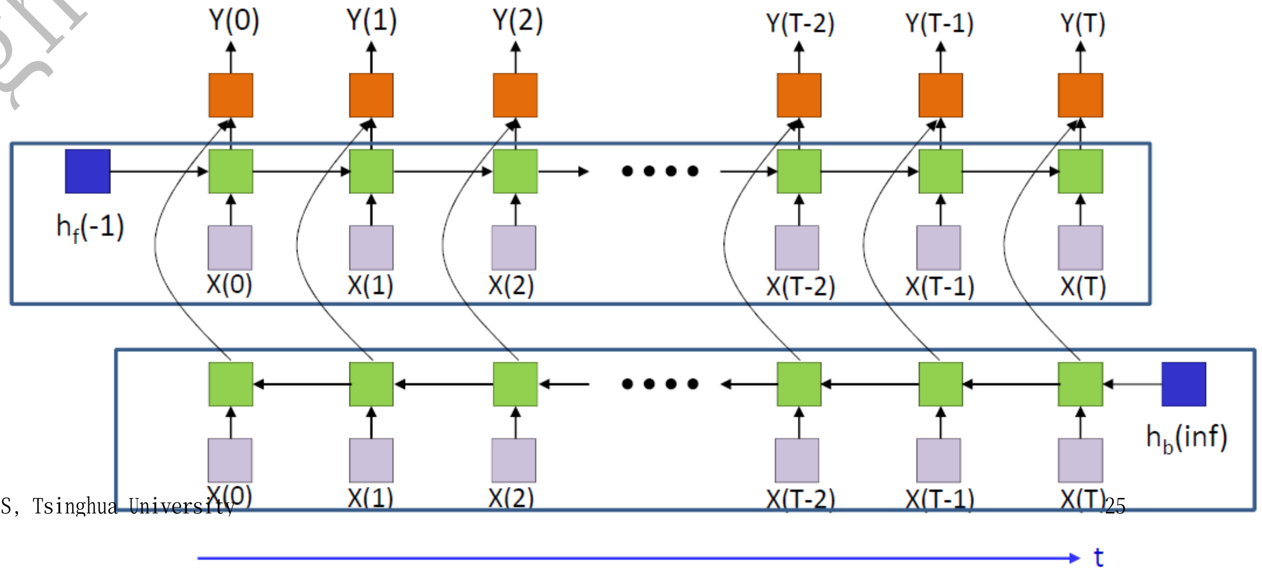


- State-Space Model
  - h_t: (hidden) state
  - X_t: input; Y_t: output
  - $h_t = f_1(h_{t-1}, X_t; \theta)$
  - $Y_t = f_2(h_t; \theta)$
  - h_{-1}: initial state
- How to train an RNN?
  - Assume we have paired input and target sequence $\{(X_t, D_t)\}_t$
  - We can define the loss function $L(\theta) = \sum_t Div(Y_t, D_t)$
    - Goal: learn the best parameter $\theta^\star$ via gradient descent
    - Backpropagation through time (BPTT)!
      - Forward pass from $t = 0 \rightarrow L$, backward pass $t = L \rightarrow 0$
      - Pay attention to gradient accumulation for recurrent weights!

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
    - $t = 0 \dots T$
  - An RNN for backward dependencies
    - $t = T \dots 0$
  - $Y_t = f_2\left(h_t^f, h_t^b; \theta\right)$
  - BPTT for bidirectional RNN?
    - $\partial Div / \partial Y_t$ for all t
    - $t = T \dots 0$ for h_f
    - $t = 0 \dots T$ for h_b

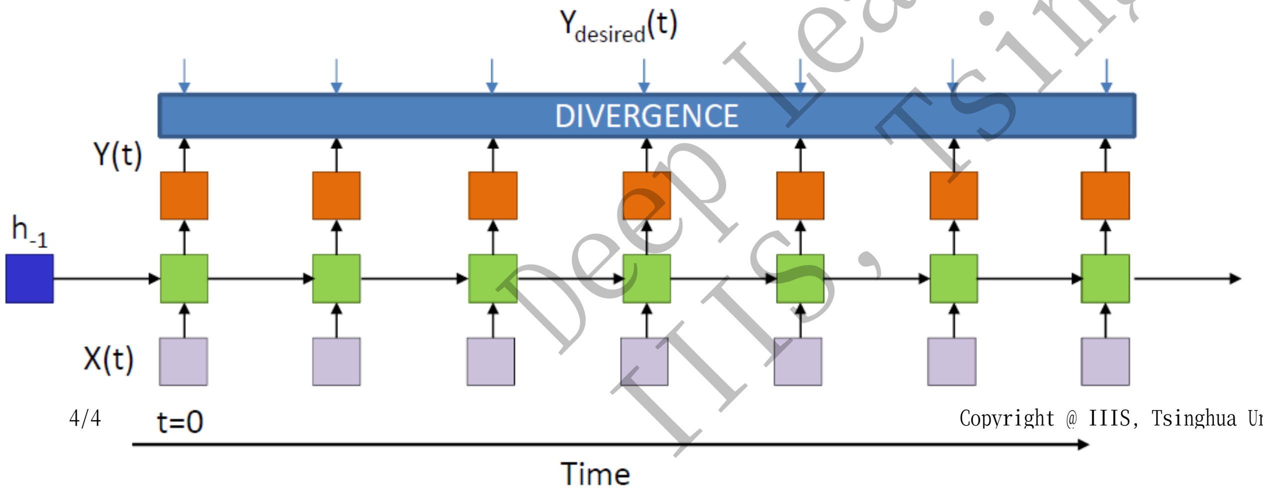


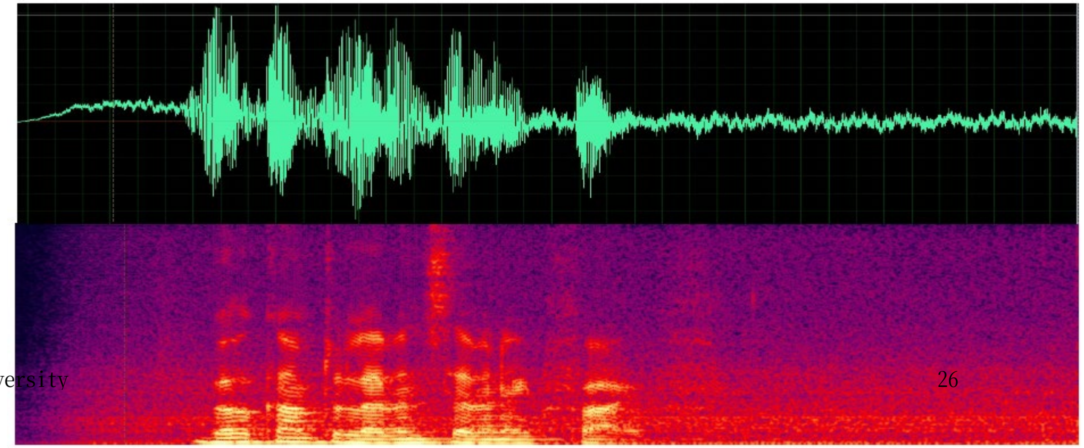


Extension

- Finding the “Welcome”
  - Input data $X_1 \dots X_L$, 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



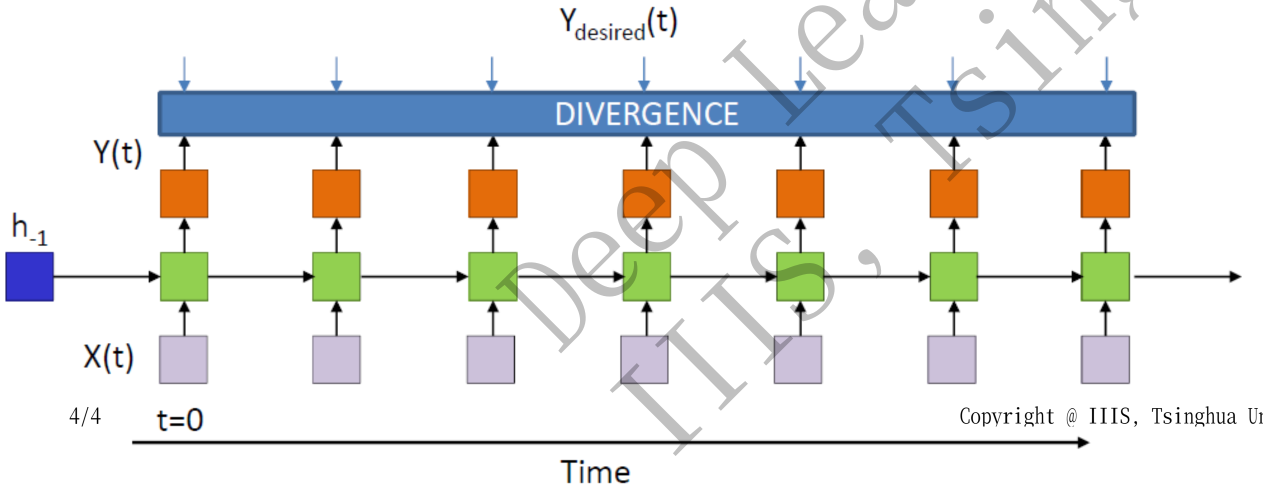


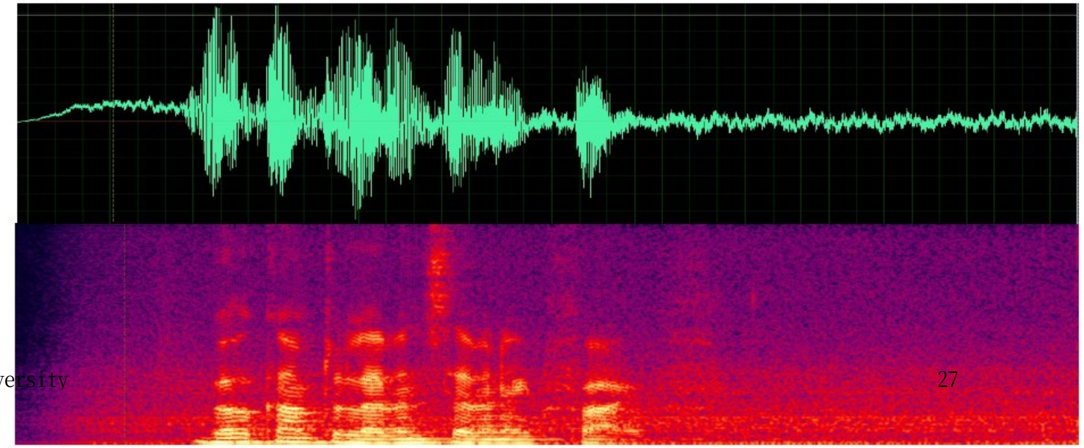
Extension

- Finding the "Welcome"
  - Input data $X_1 \dots X_L$, 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

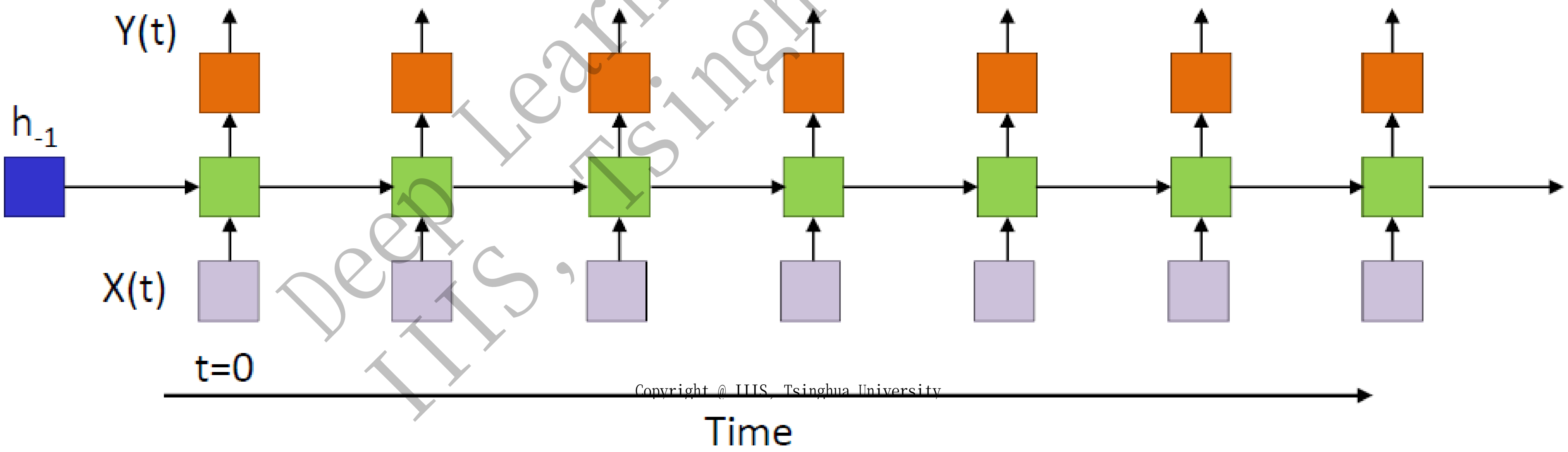
...... in theory!



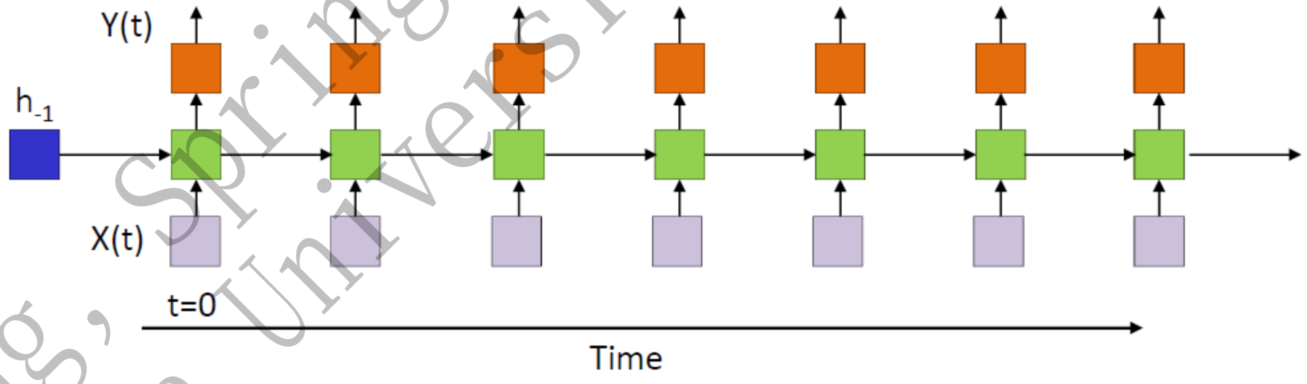


Practice Issues of RNN

- 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

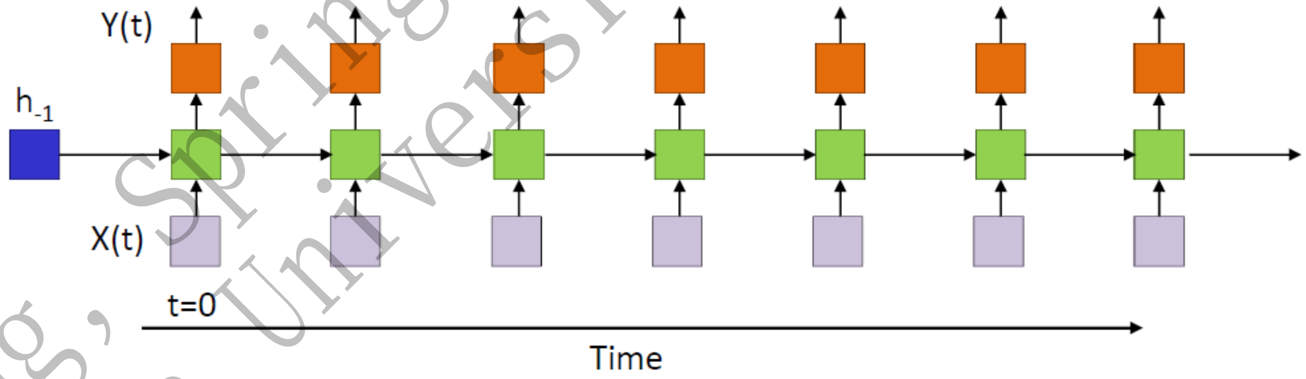


Practice Issues of RNN



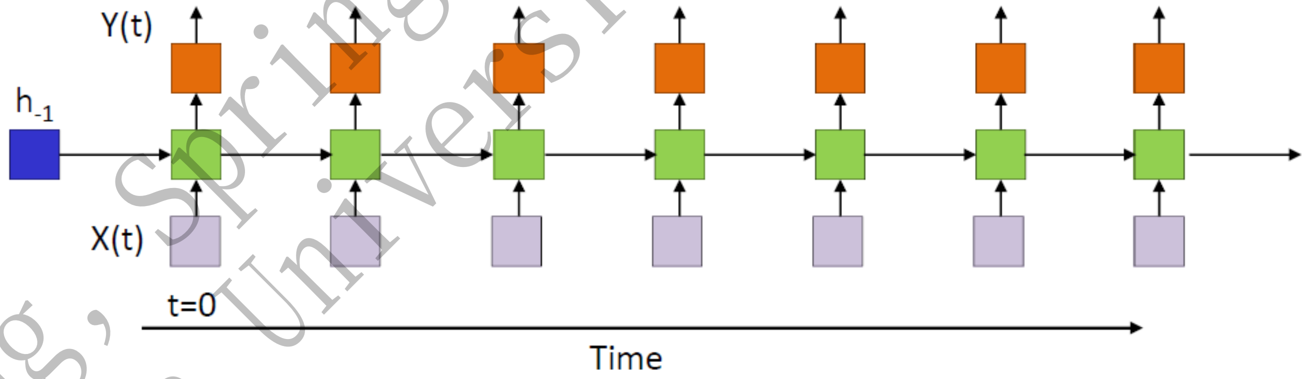
- We start with a linear RNN
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = z_t$
- Let's expand the recursions
  - $h_k = W_h \cdot h_{k-1} + W_x \cdot X_k$
  - $= W_h^2 h_{k-2} + W_h W_x \cdot X_{k-1} + W_x \cdot X_k$
  - $= W_h^3 h_{k-3} + W_h^2 W_x \cdot X_{k-2} + W_h W_x \cdot X_{k-1} +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

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

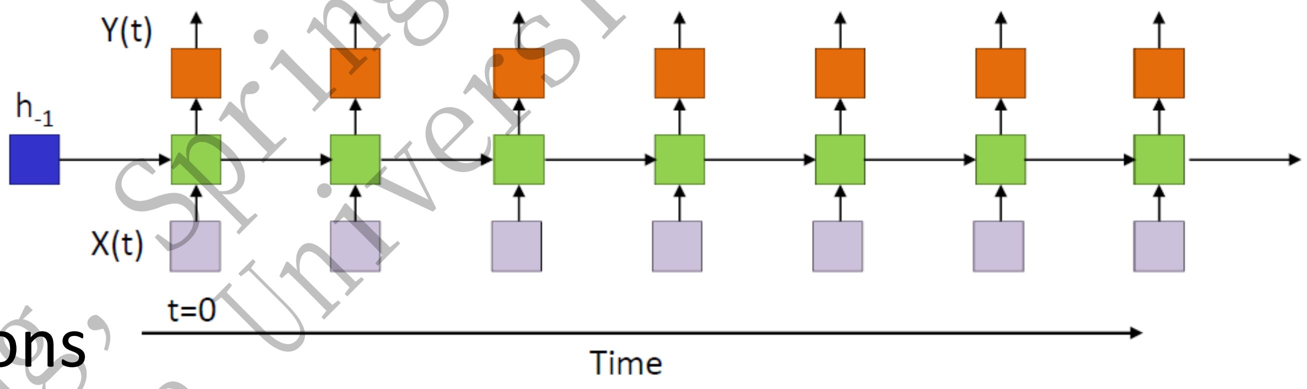
Practice Issues of RNN



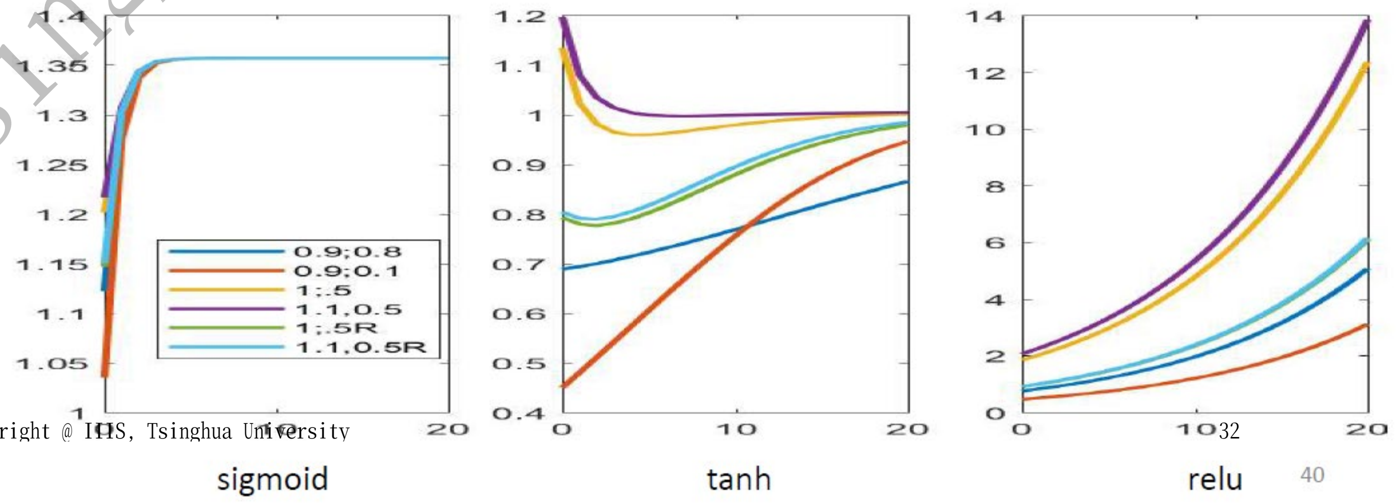
- We start with a linear RNN
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = z_t$
- Let's expand the recursions
  - $h_k = W_h^{k+1} h_{-1} + \sum_{i=0}^{k} W_h^{k-i} W_x \cdot X_i$
  - The coefficient of signal at position i is exponential over W_h
    - If $|\lambda_{\max}| > 1$, the system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

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

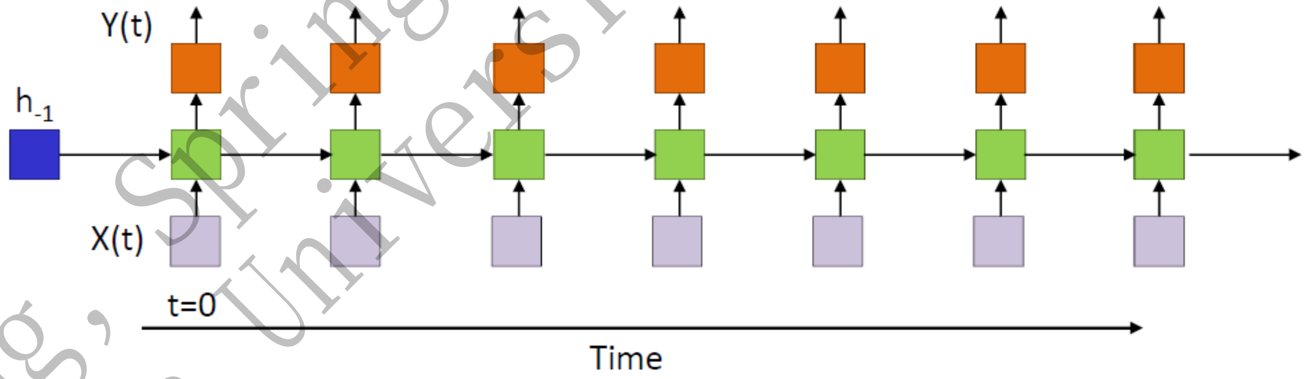


- Simulation results with activations
  - A uniform start $h_{-1} = [1,1,1,1,\ldots]/\sqrt{N}$
    - We simulate $|W_h^{k+1} h_{-1}|$ with various eigenvalues in W_h
    - Remark:
      - Tanh is preferred
      - … but still saturates
    - What about backward pass?

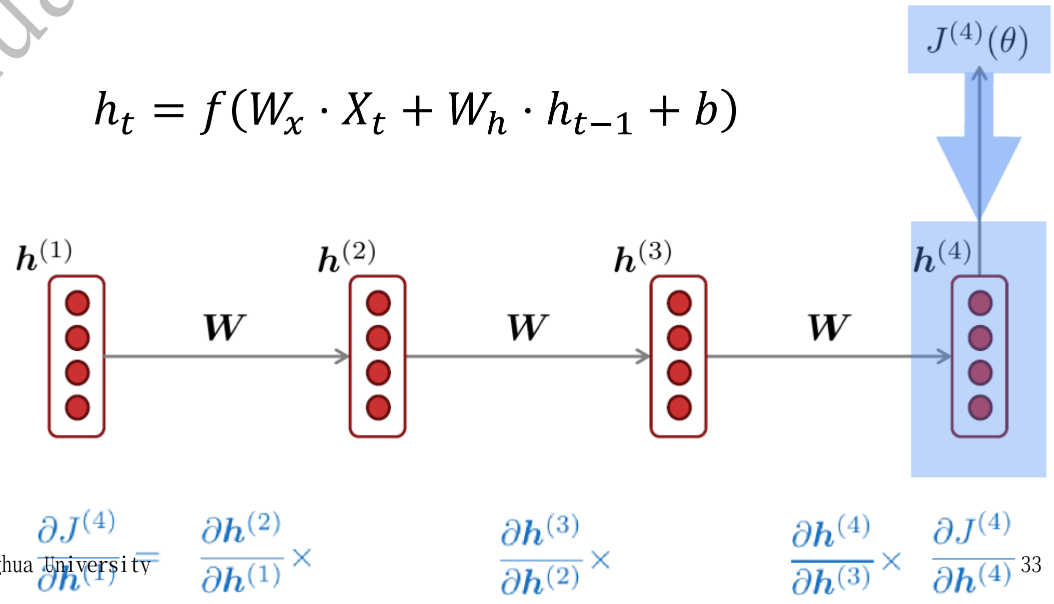


Practice Issues of RNN

- 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 $J_k(\theta) = Div(Y_k, D_k)$
  - $\frac{\partial J_k}{\partial h_0} = \frac{\partial J_k}{\partial h_k} \prod_t \frac{\partial h_t}{\partial h_{t-1}}$
  - $\propto \prod_t W_h f'(Z_t) = W_h^k \prod_t f'(Z_t)$

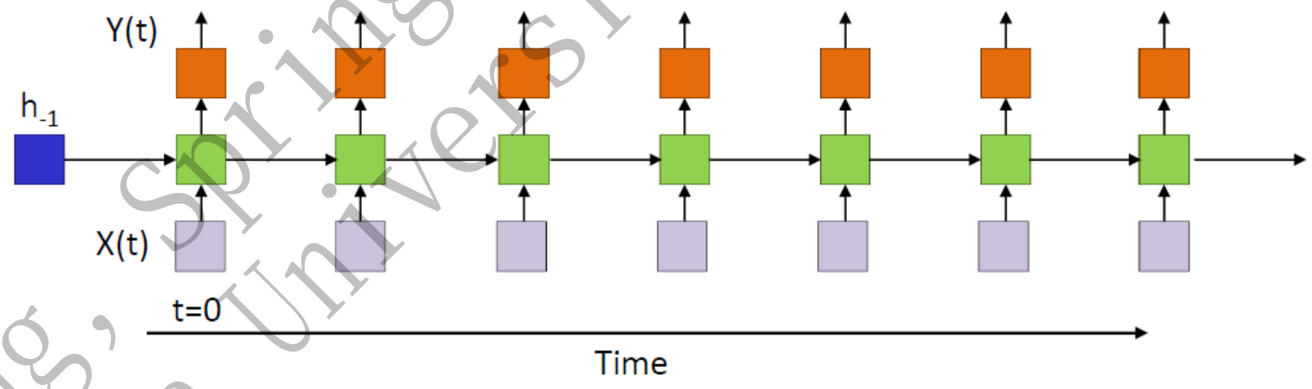


$h_t = f(W_x \cdot X_t + W_h \cdot h_{t-1} + b)$

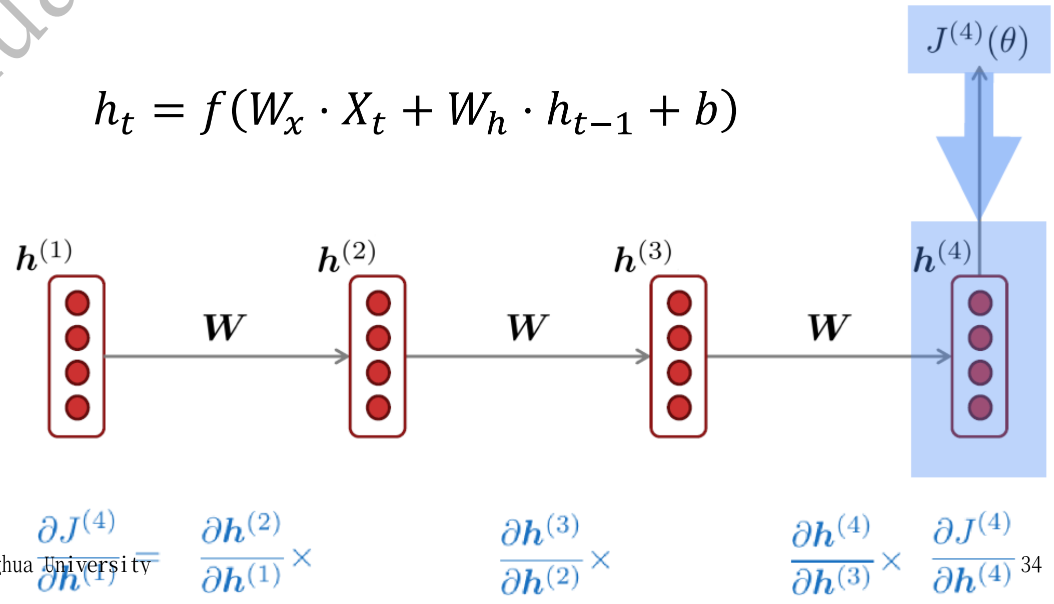


Practice Issues of RNN

- 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!

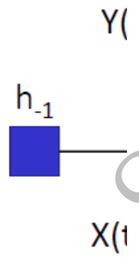


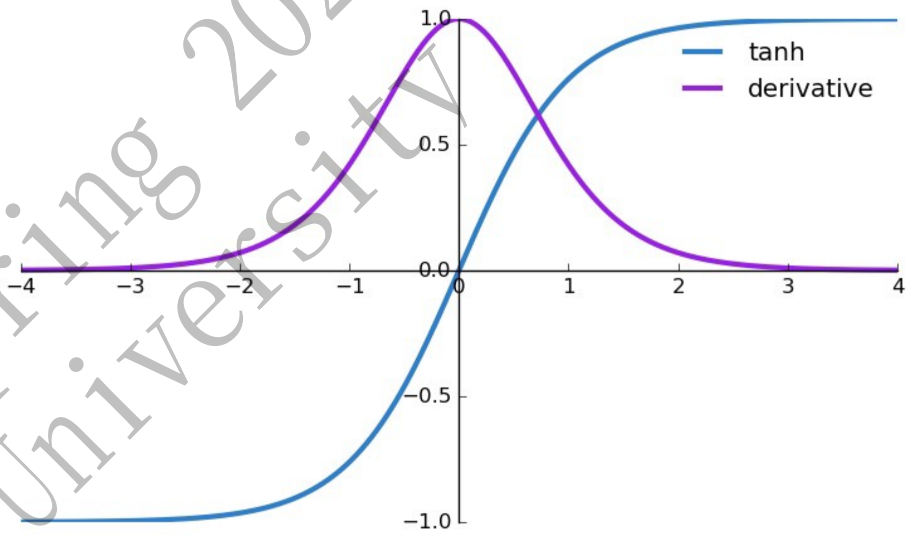
$h_t = f(W_x \cdot X_t + W_h \cdot h_{t-1} + b)$



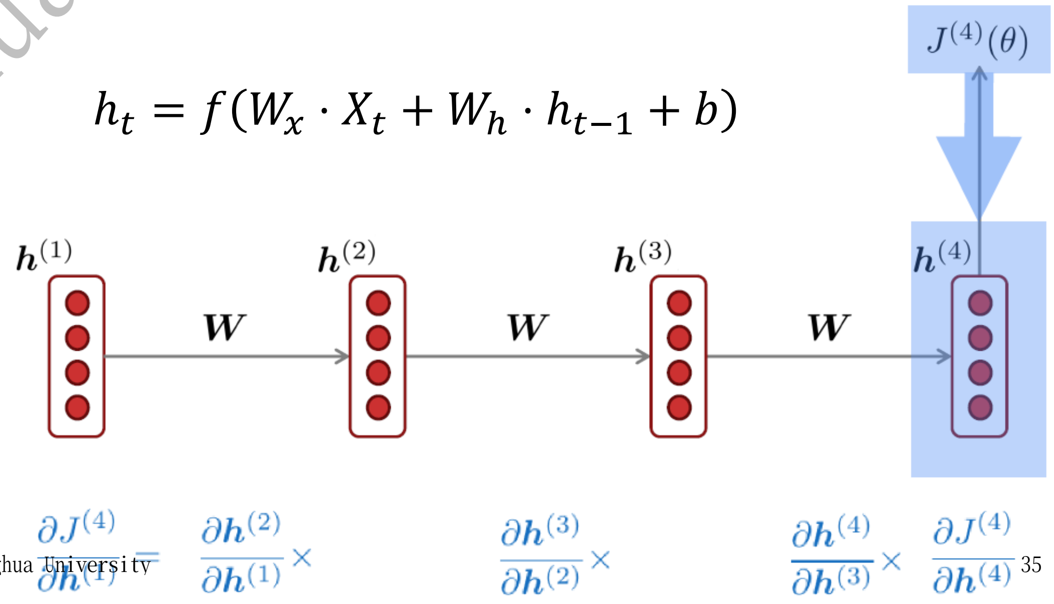
Practice Issues of RNN

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



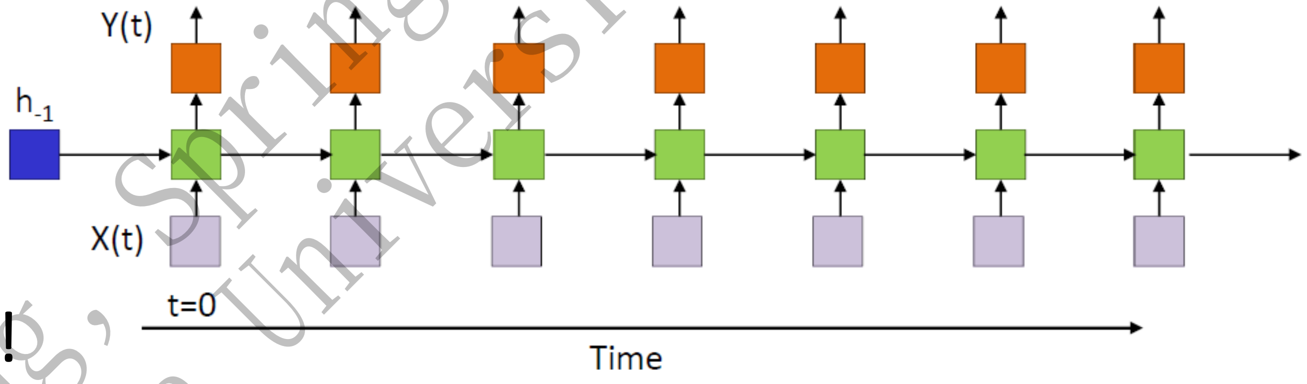


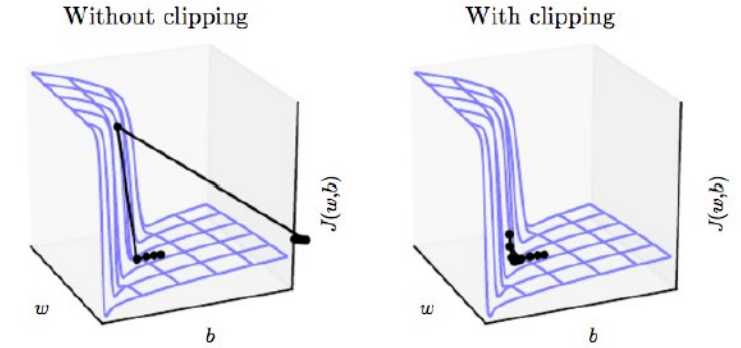
$h_t = f(W_x \cdot X_t + W_h \cdot h_{t-1} + b)$



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



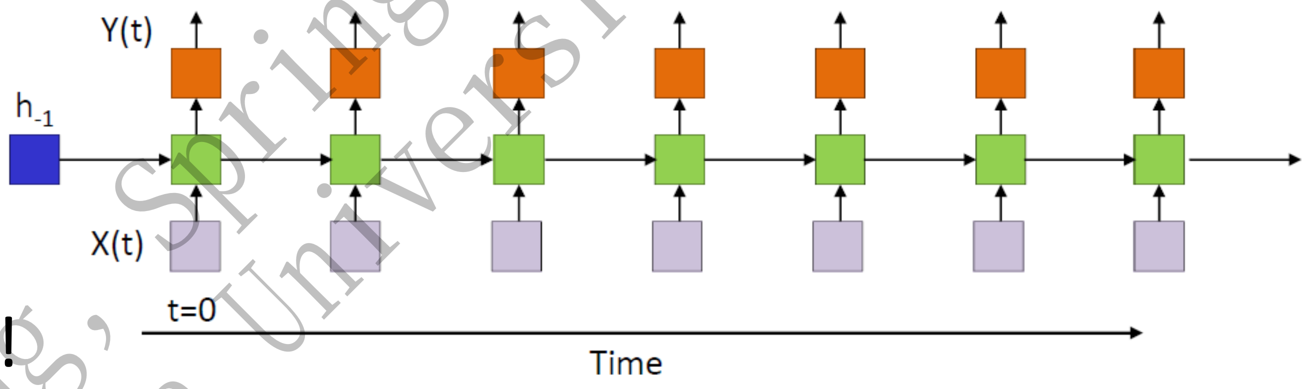


Algorithm 1 Pseudo-code for norm clipping

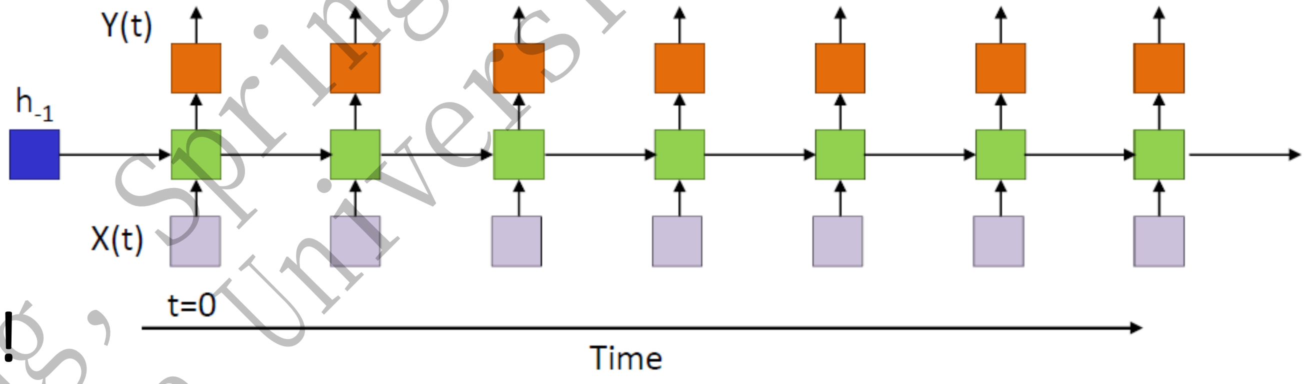
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Practice Issues of RNN

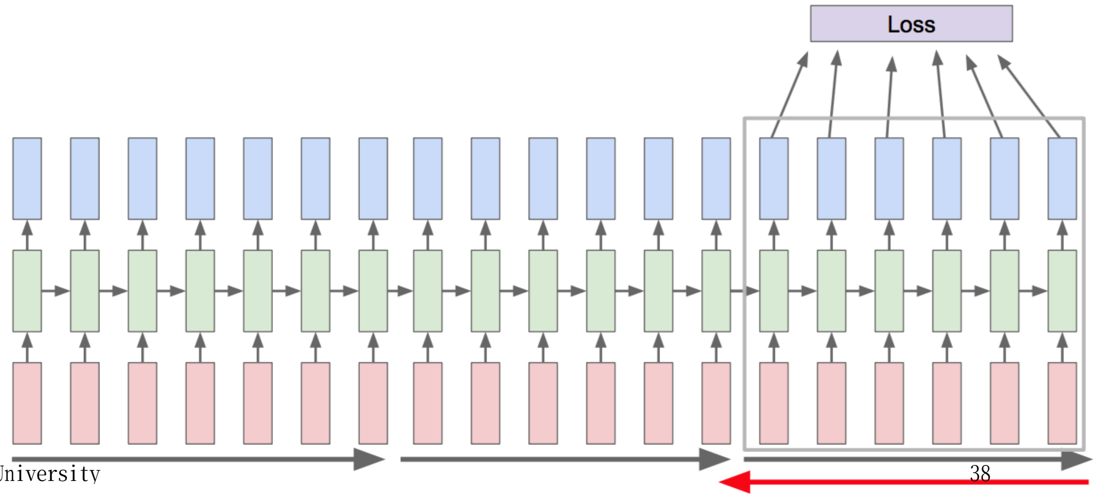
- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
  - Identity initialization
    - Make sure the weight matrix is initialized to have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



Practice Issues of RNN

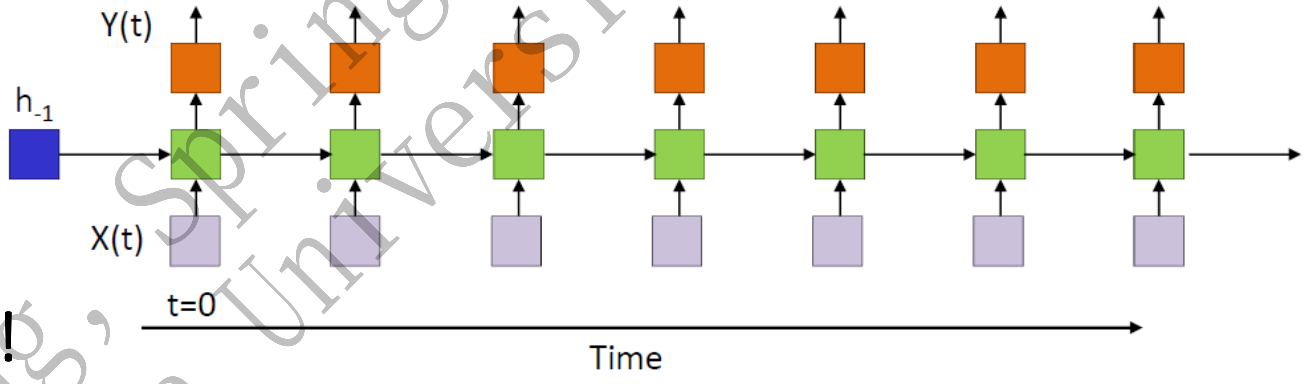


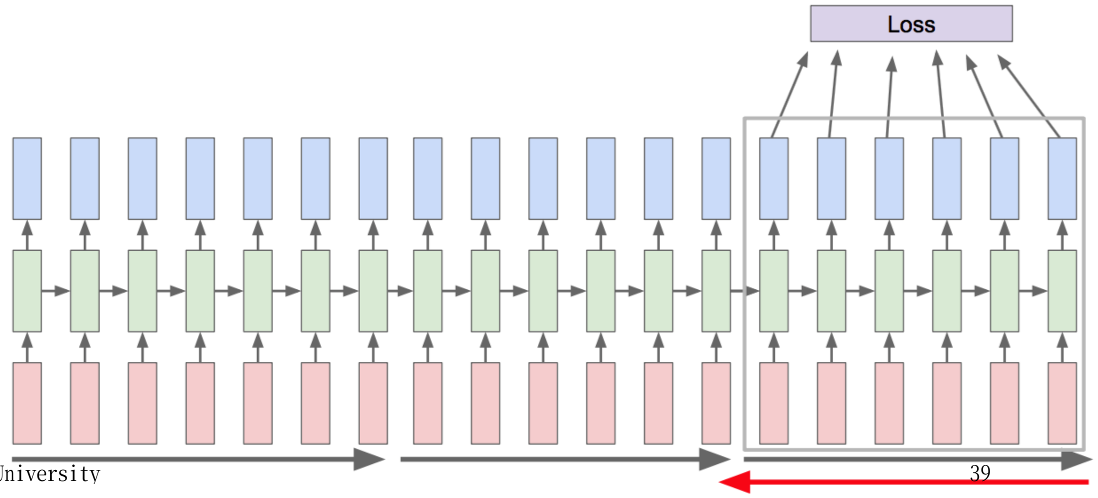
- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
  - Identity initialization
  - Truncated Backprop Through Time
    - Only backpropagate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



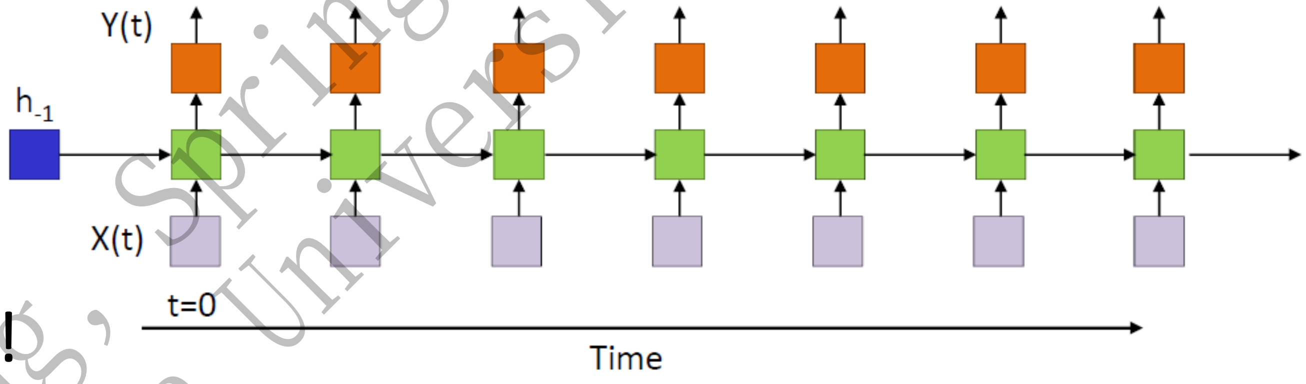
Practice Issues of RNN

- 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

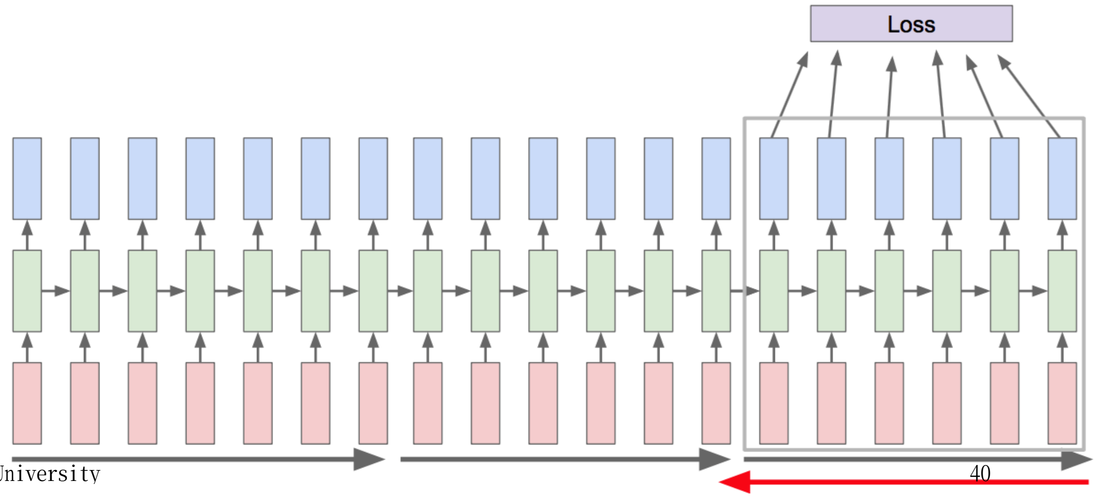




Practice Issues of RNN

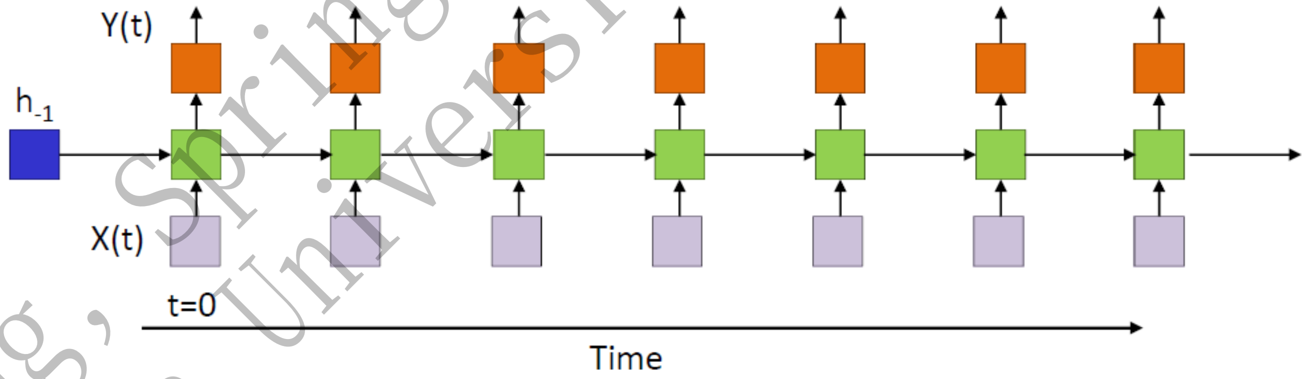


- RNN with non-linearity
  - $z_t = W_h \cdot h_{t-1} + W_x \cdot X_t$
  - $h_t = f(z_t)$
- Gradient Explosion & Vanishing!
  - Training instability and long-term dependency
- Tricks for explosion
  - Gradient clipping
  - Identity initialization
  - Truncated Backprop Through Time
- What about memory?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

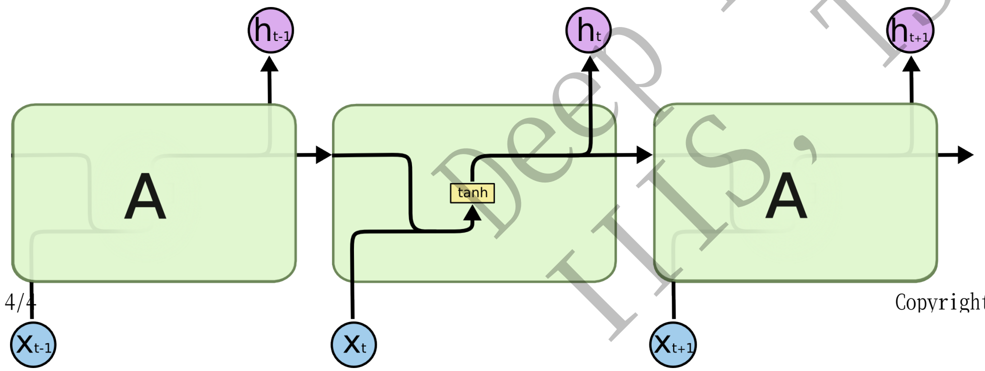


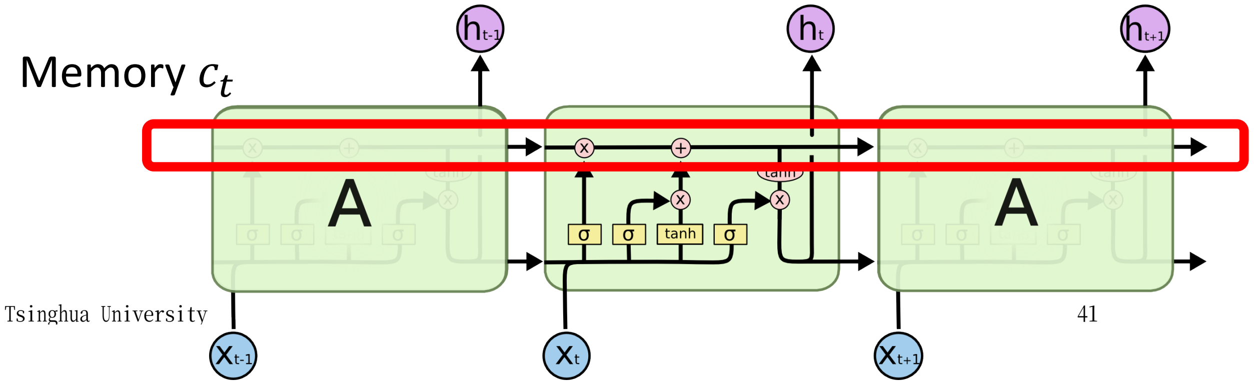
Preserve Long-Term Memory

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



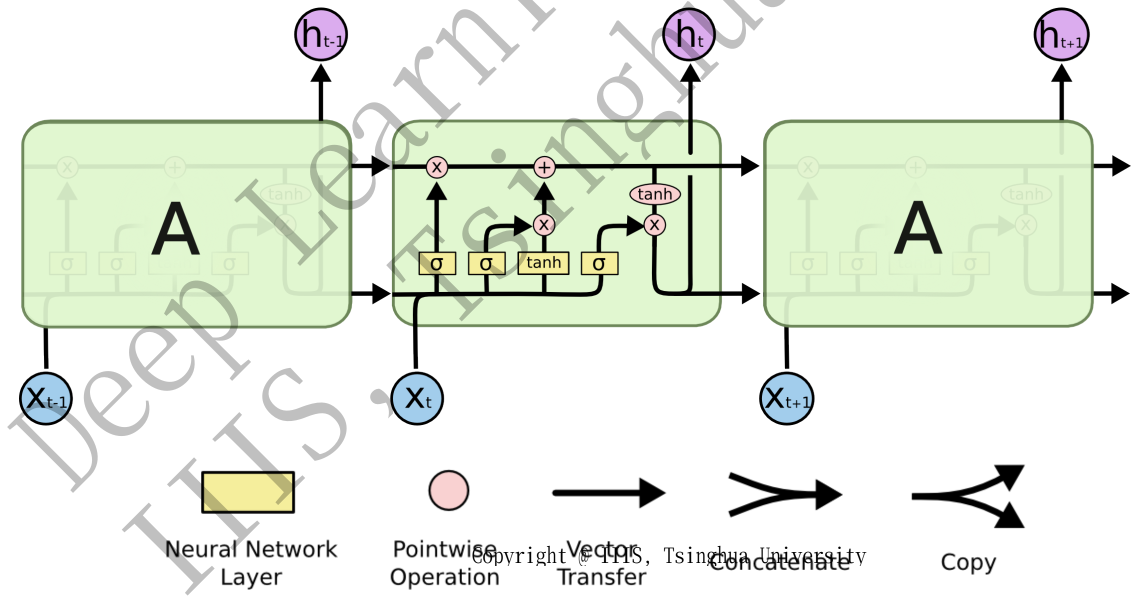
- It is difficult for RNN to preserve long-term memory
  - The hidden state h_t is constantly being written (short-term memory)
  - Let's keep a separate cell for maintaining long-term memory





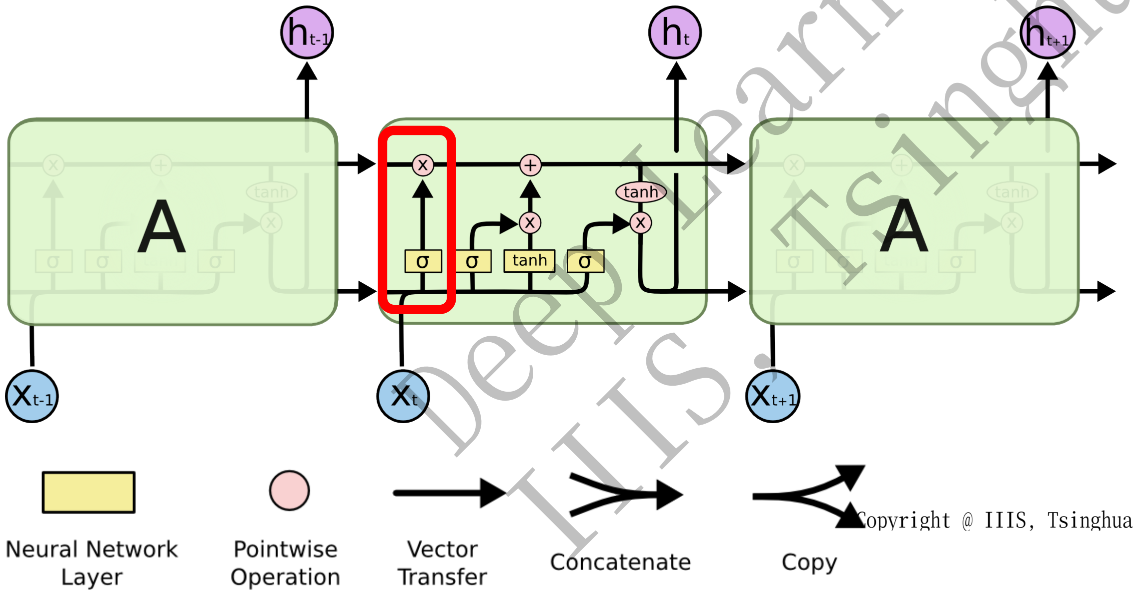
Long Short-Term Memory Network

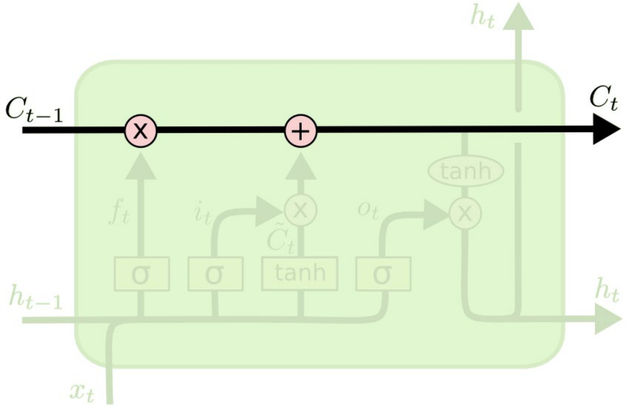
- LSTM (Hochreiter & Schmidhuber, 1997)
  - A special RNN architecture for learning long-term dependencies
  - σ: layer with sigmoid activation
  - Let's walk through the architecture
    - Diagrams from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

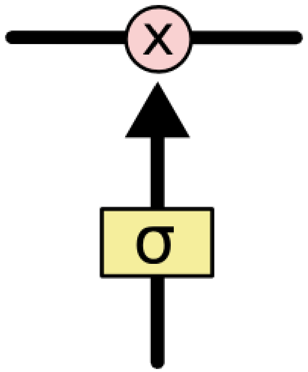


Long Short-Term Memory Network

- 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

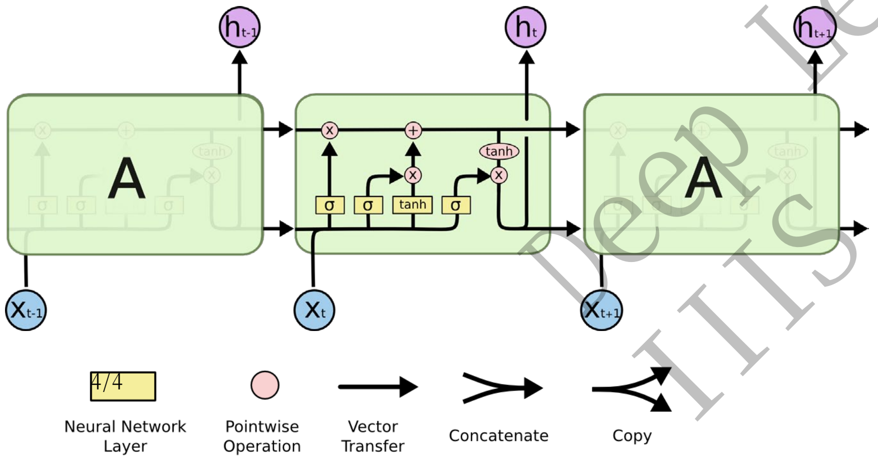


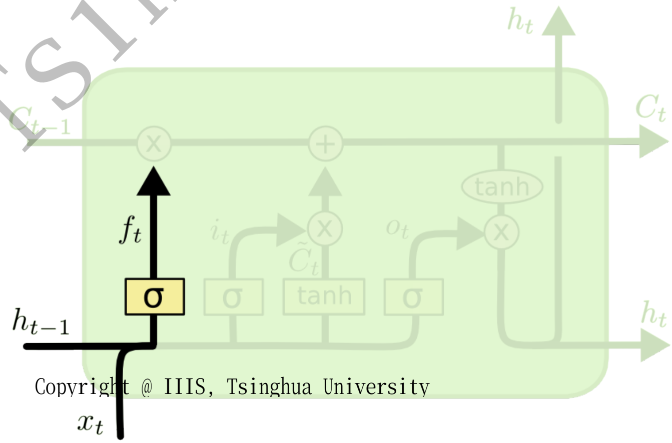




Long Short-Term Memory Network

- 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 $c_t(i)$

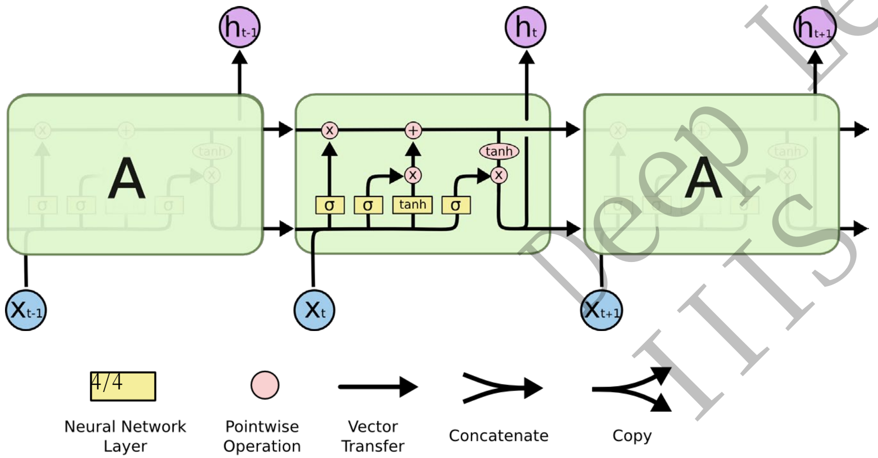


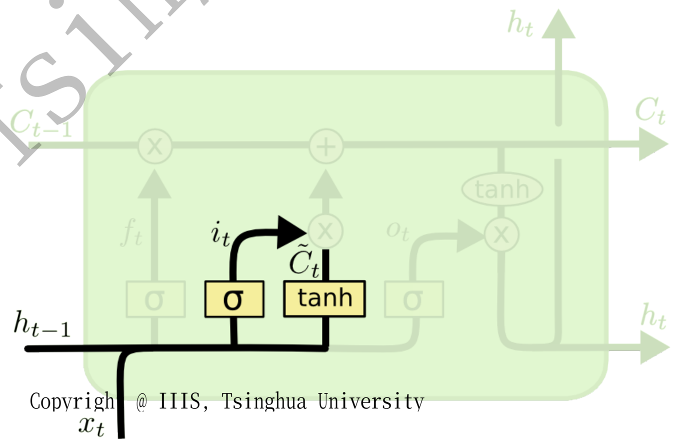


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

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
      - $i_t(j) \to 1$: we want to keep the information in $\tilde{c}_t(j)$ to update memory
      - $i_t(j) \to 0$: $\tilde{c}_t(j)$ should not contribute to memory



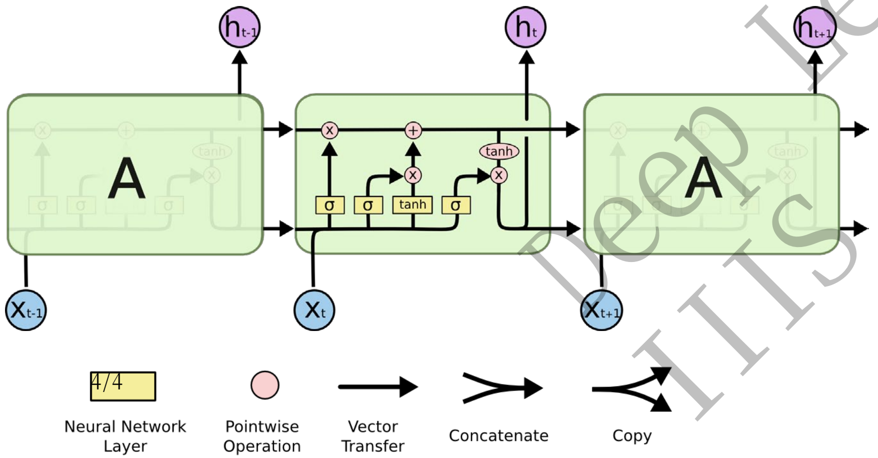


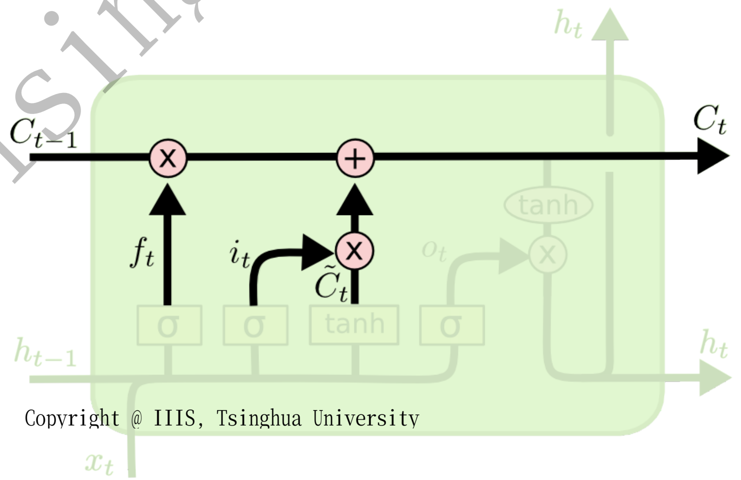
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

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

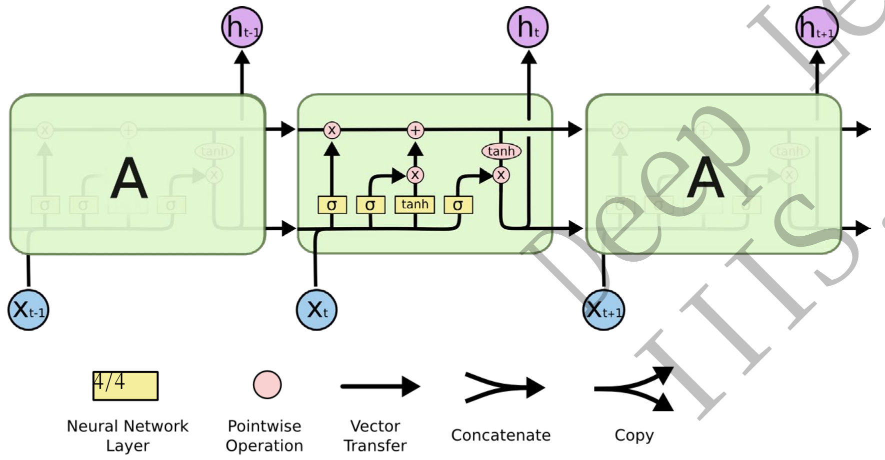


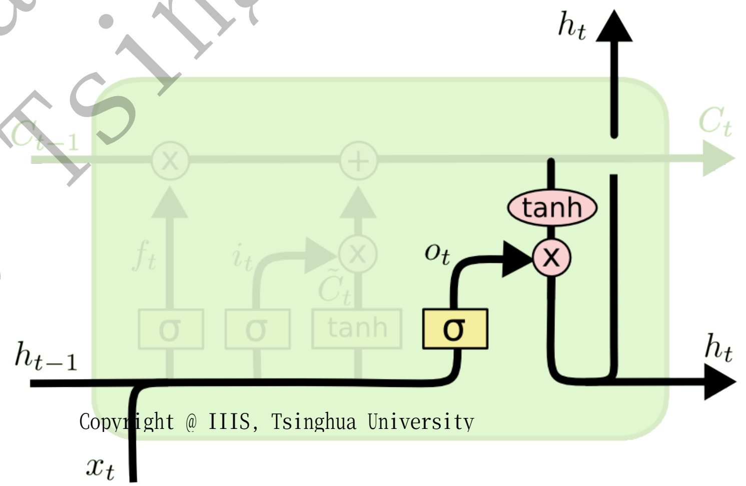


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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
      - $o_t(j) \to 1$: $\tanh(c_t(j))$ is critical for next state
      - $o_t(j) \to 0$: $\tanh(c_t(j))$ does not worth reporting



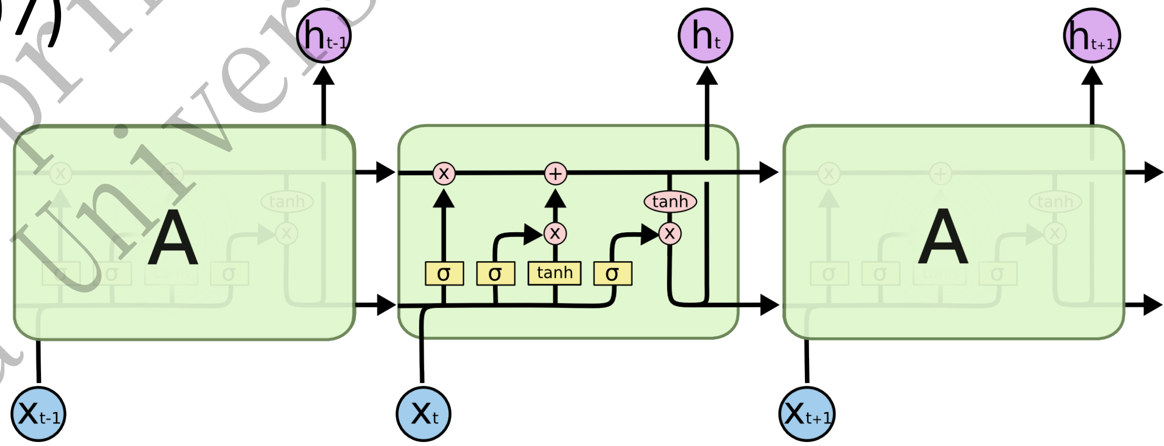


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

Long Short-Term Memory Network

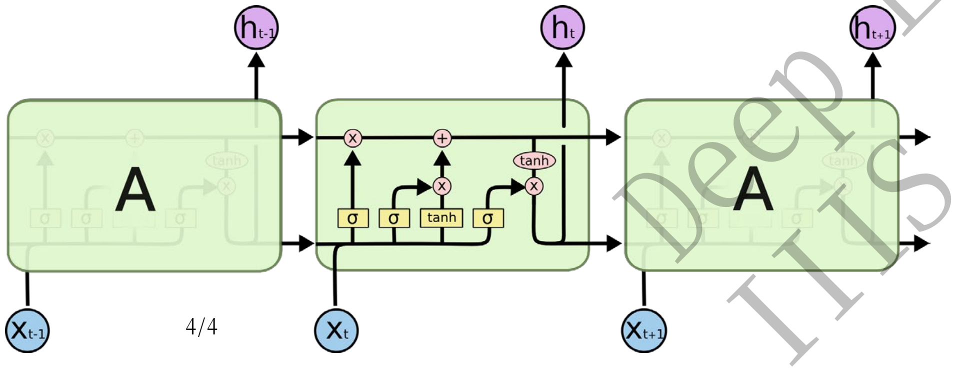
- LSTM (Hochreiter & Schmidhuber, 1997)
  - $h_t = o_t \odot \tanh(c_t)$
  - $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
  - $Y_t = g(h_t)$
  - Uninterrupted gradient flow!
    - No more matrix multiplication for c_t
    - In practice: ~100 timesteps of memory instead of ~7

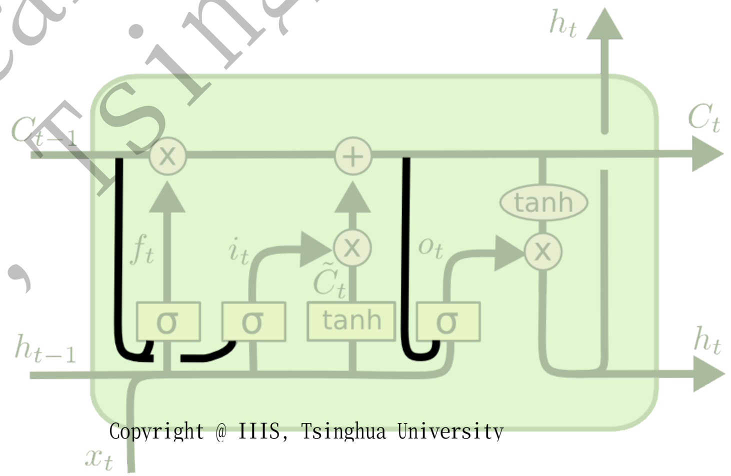


- Remark
  - LSTM does not have guarantees for gradient explosion/vanishing
  - An architecture that makes learning long-term dependency easier
  - LSTMs is the dominant approach for sequence modeling from 2013~2016

LSTM Variants

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

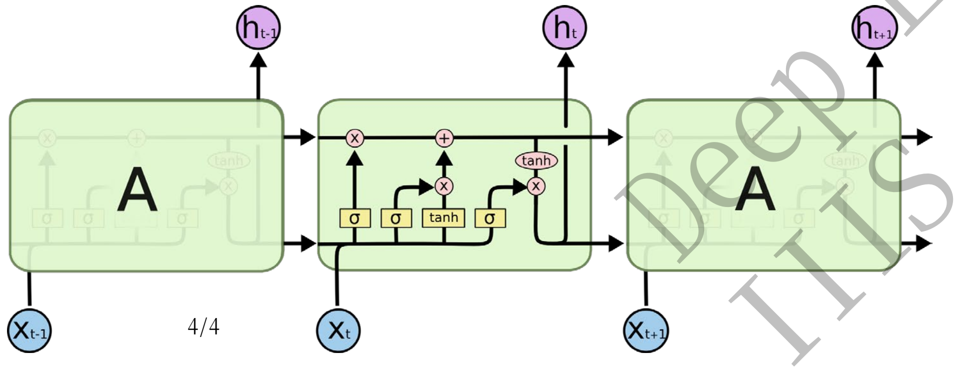


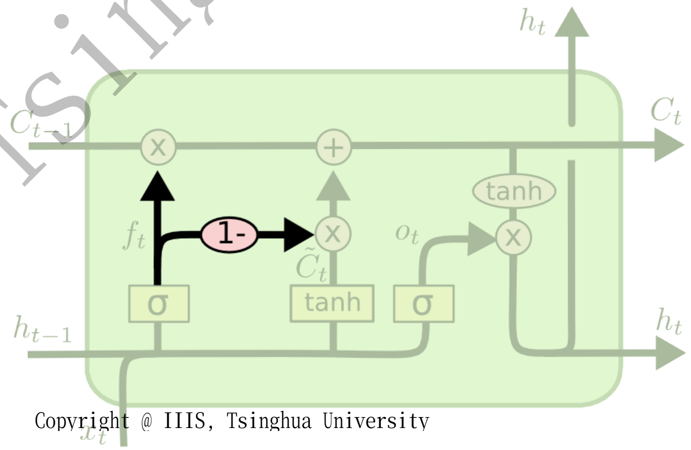


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

LSTM Variants

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

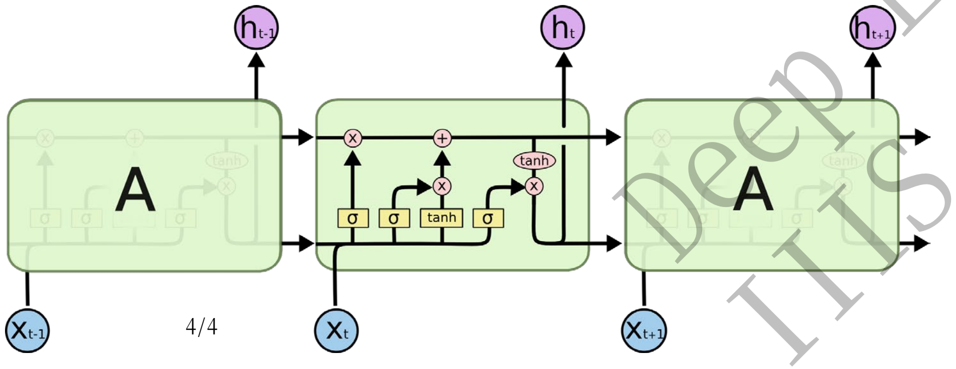


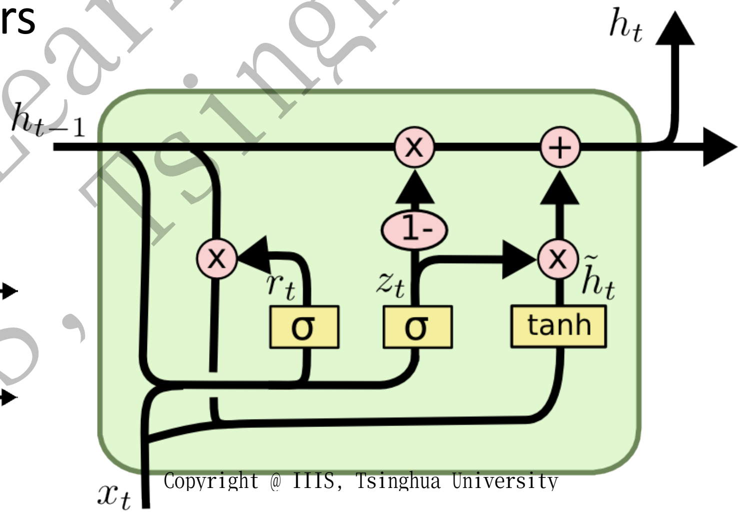


$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

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





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

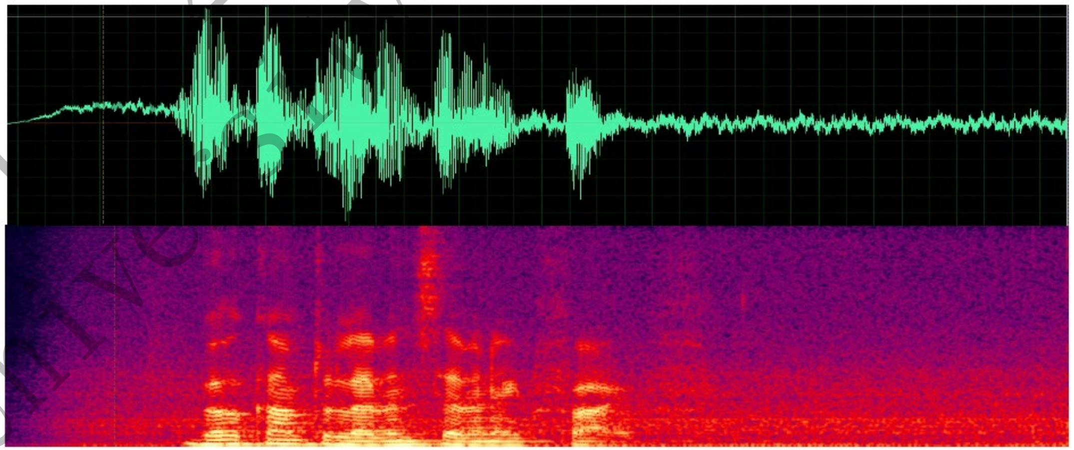
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

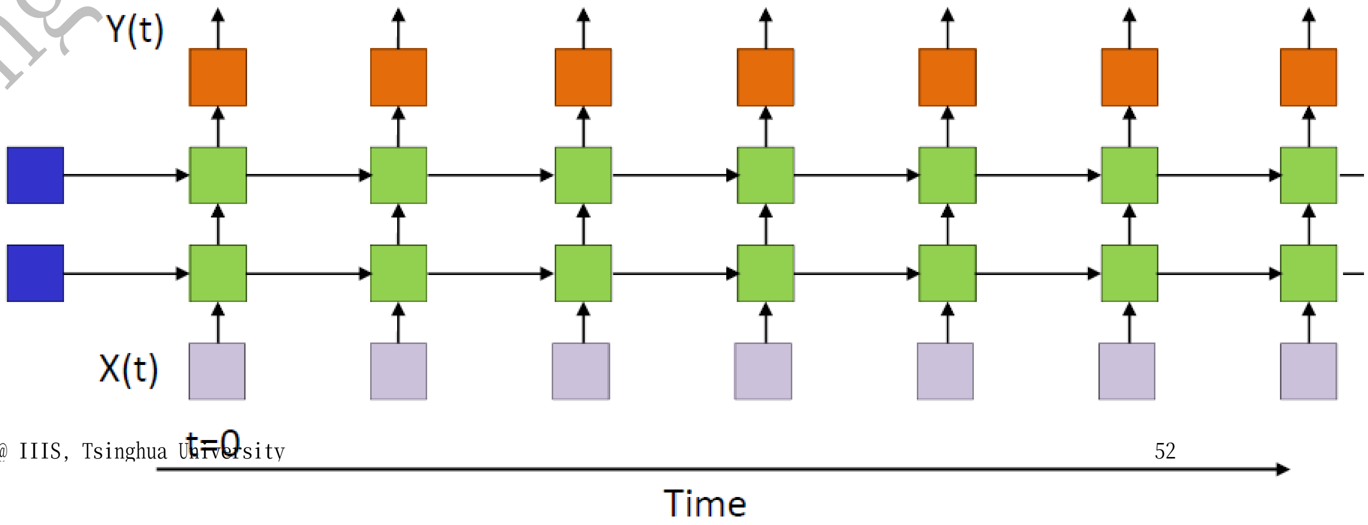
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

4/4

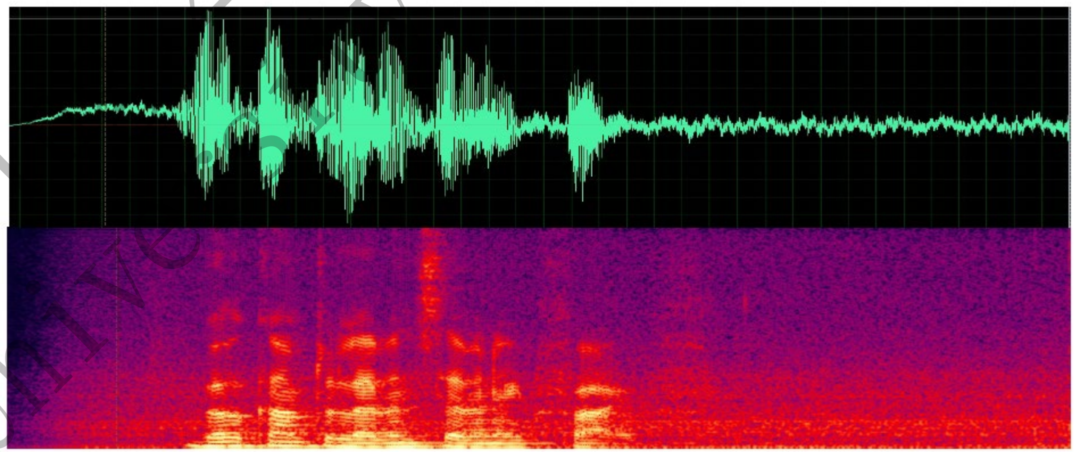
LSTM Applications



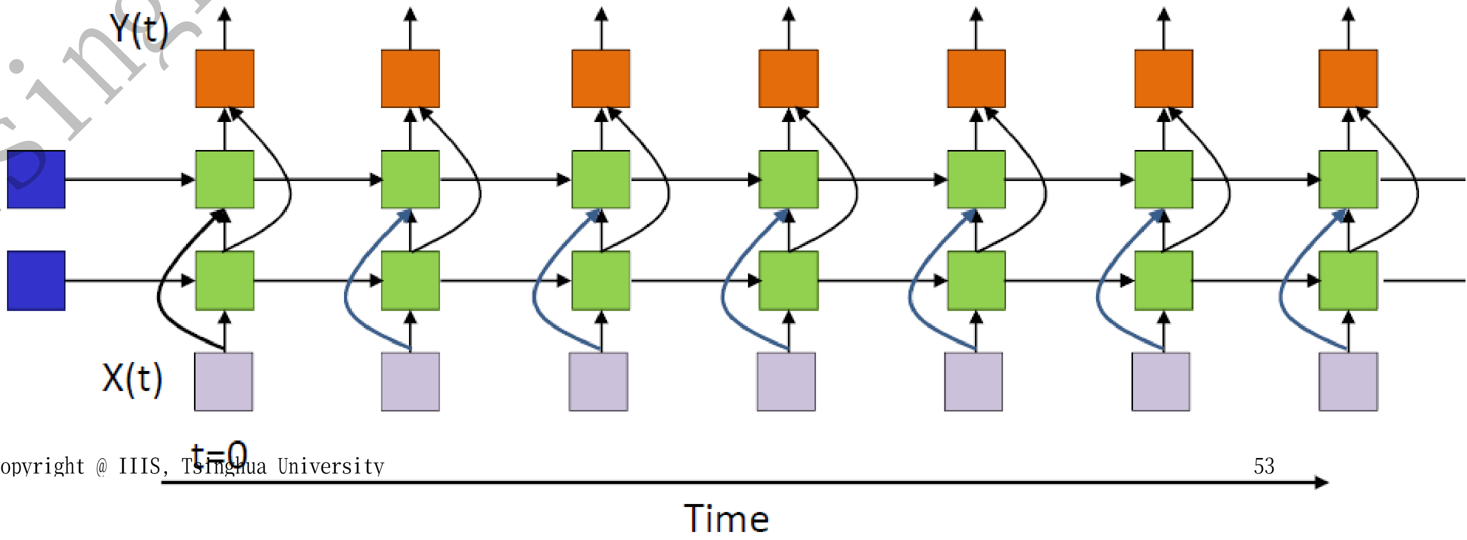
- Finding the "Welcome"
  - Input data $X_1 \dots X_L$, L may vary
  - Whether the voice contains "Welcome"
- Solution
  - Multi-layer LSTM and max-pooling over Y_t
    - Sometimes also just use h_T to compute output for simplicity



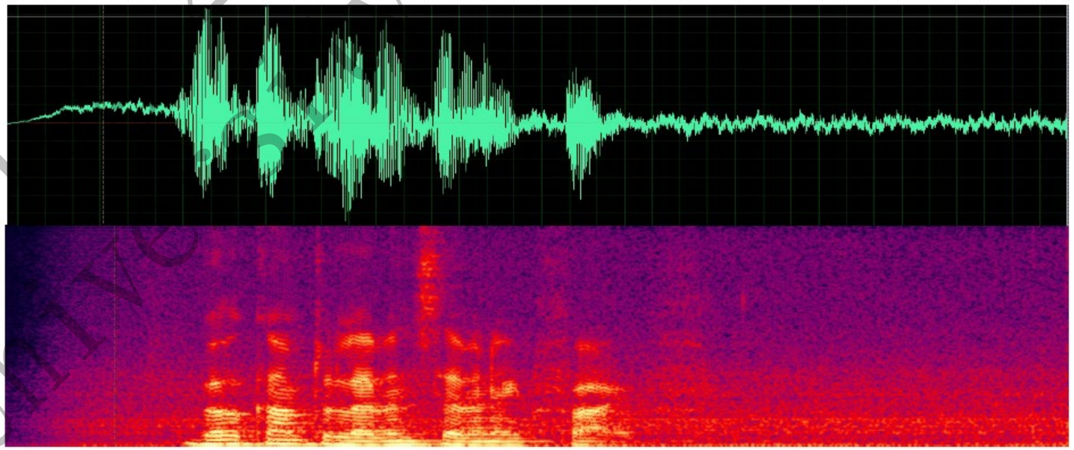
LSTM Applications



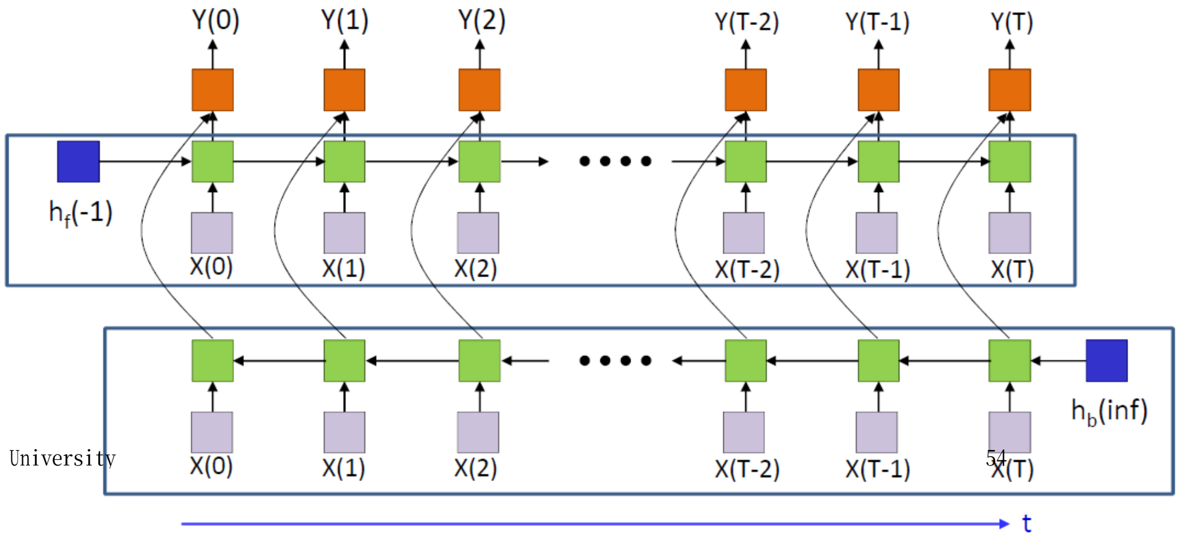
- Finding the "Welcome"
  - Input data $X_1 \dots X_L$, L may vary
  - Whether the voice contains "Welcome"
- Solution
  - Multi-layer LSTM and max-pooling over Y_t
    - Skip-connections for deeper LSTMS!



LSTM Applications

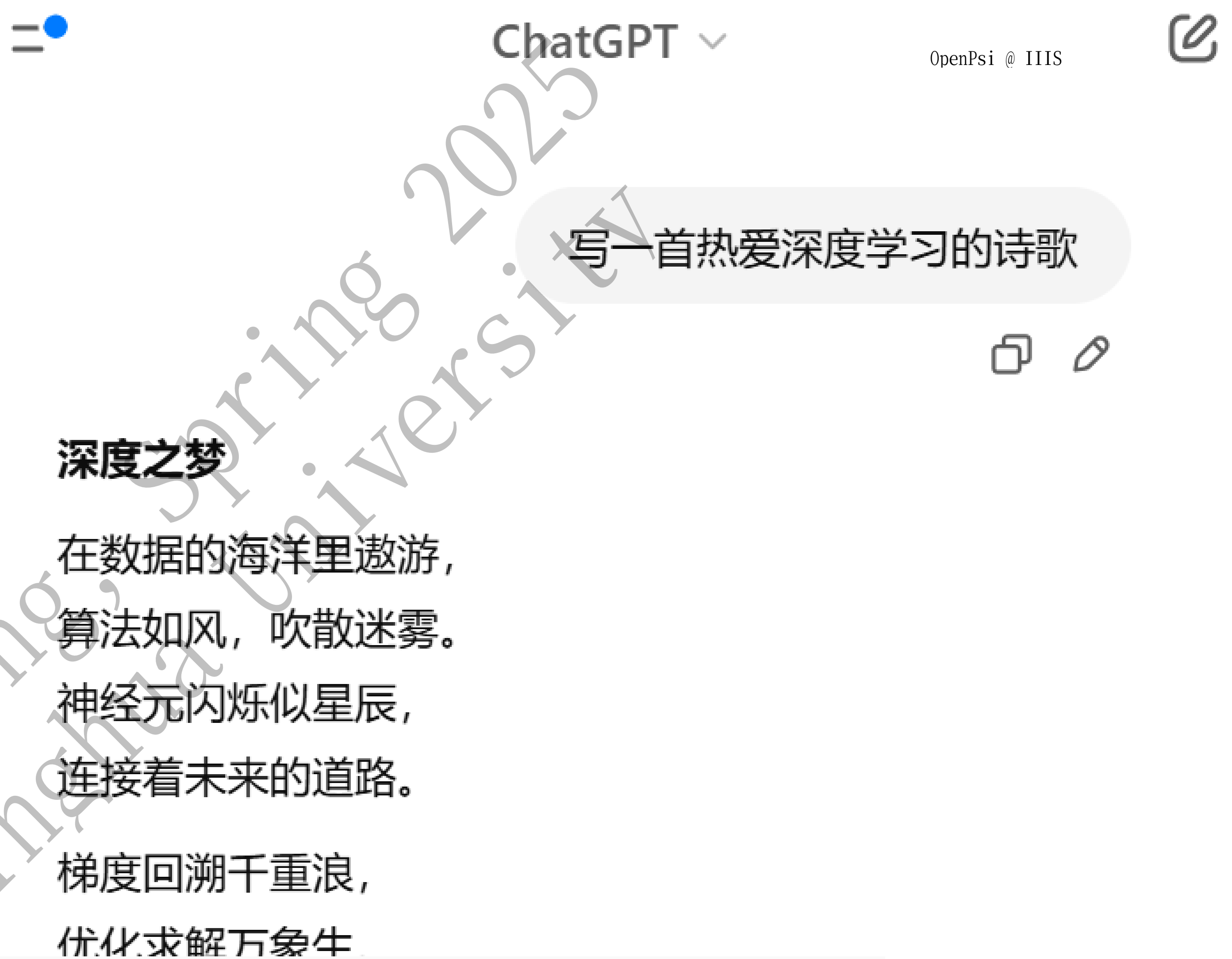


- Finding the “Welcome”
  - Input data $X_1 \dots X_L$, L may vary
  - Whether the voice contains “Welcome”
- Solution
  - Multi-layer LSTM and max-pooling over Y_t
    - Skip-connections for deeper LSTMS!
  - Bidirectional LSTMs!
    - Sometimes use $h_f(T)$ & $h_b(T)$ for output
  - Remember gradient 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.: 诗歌全集
  - Even conditional generation!
    - Next lecture

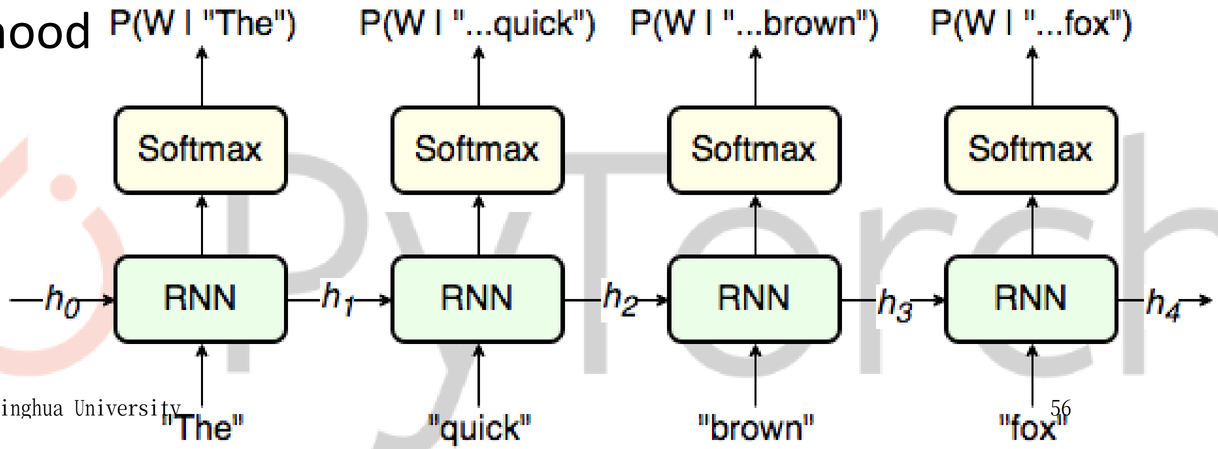


Language Model

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

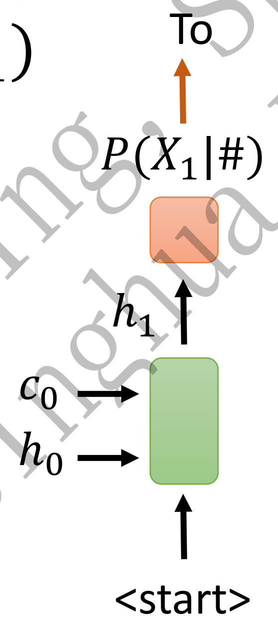
$$P(X;\theta)=\prod_{t=1}^{L}P(X_t|X_{i<t};\theta)$$

  - 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



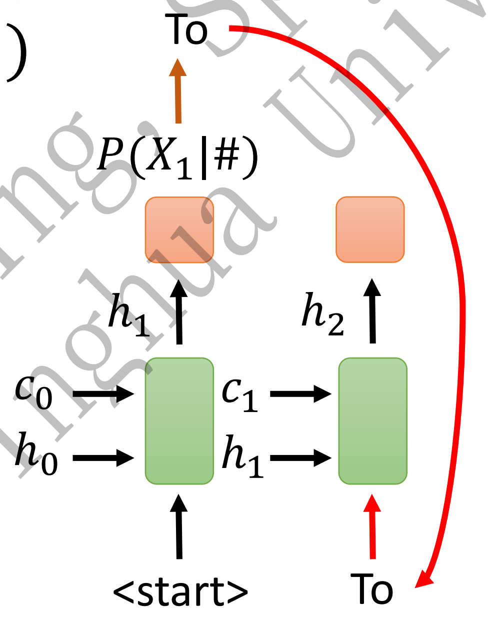
Language Model

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



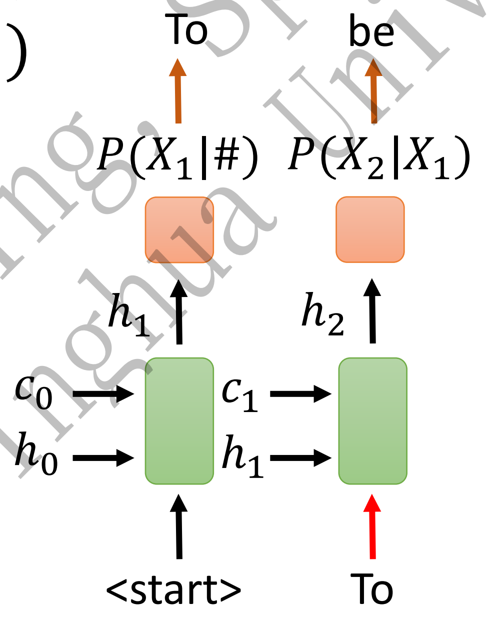
Language Model

- LSTM language model
  - $Y_t = \mathrm{softmax}(h_t) = P(X_t|X_{i<t})$
  - $h_t, c_t = \mathrm{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- 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



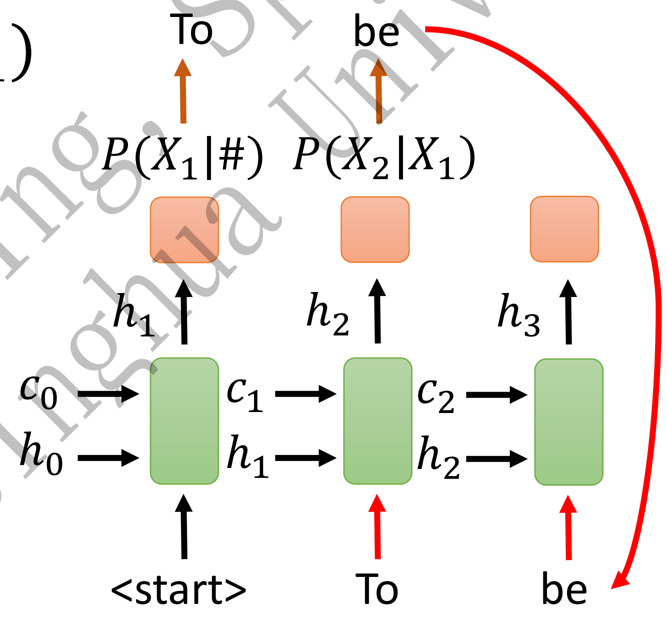
Language Model

- 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 $X_2 \sim Y_2(h_1, c_1, X_1; \theta)$



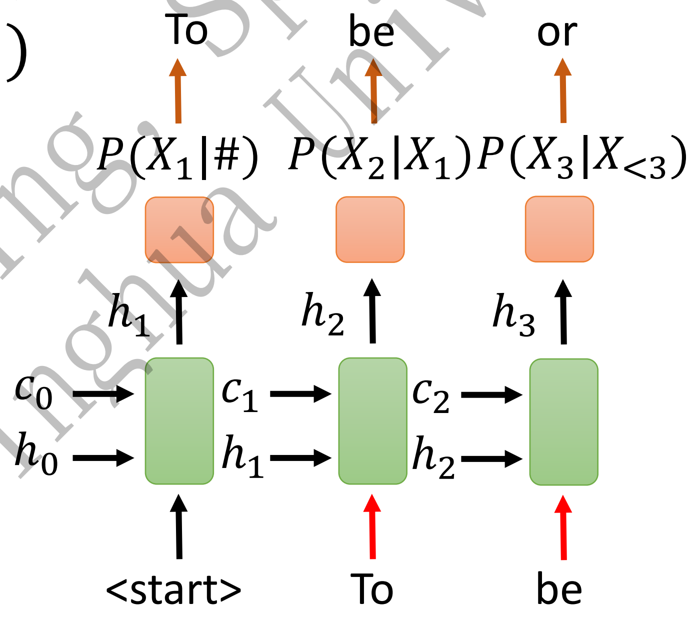
Language Model

- LSTM language model
  - $Y_t = \mathrm{softmax}(h_t) = P(X_t | X_{i<t})$
  - $h_t, c_t = \mathrm{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- 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



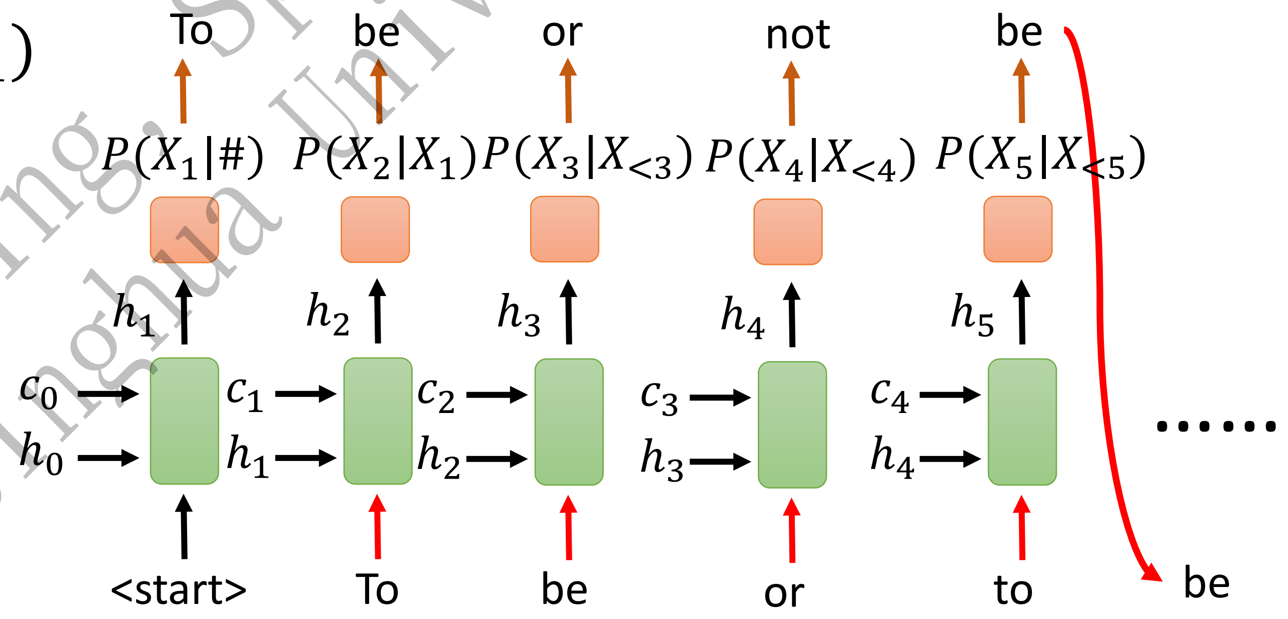
Language Model

- 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 $X_3 \sim Y_3(c_2, h_2, X_2; \theta)$



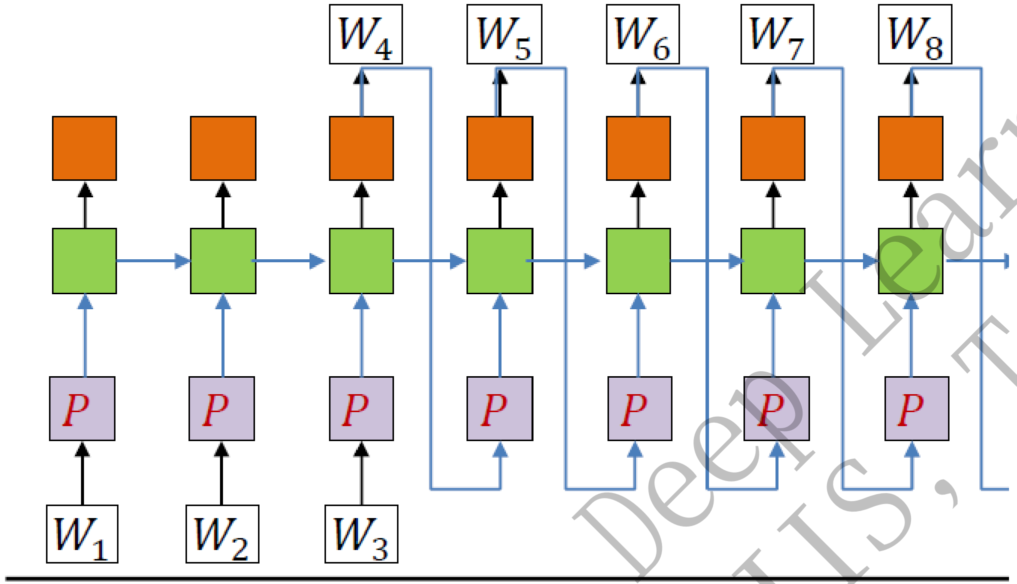
Language Model

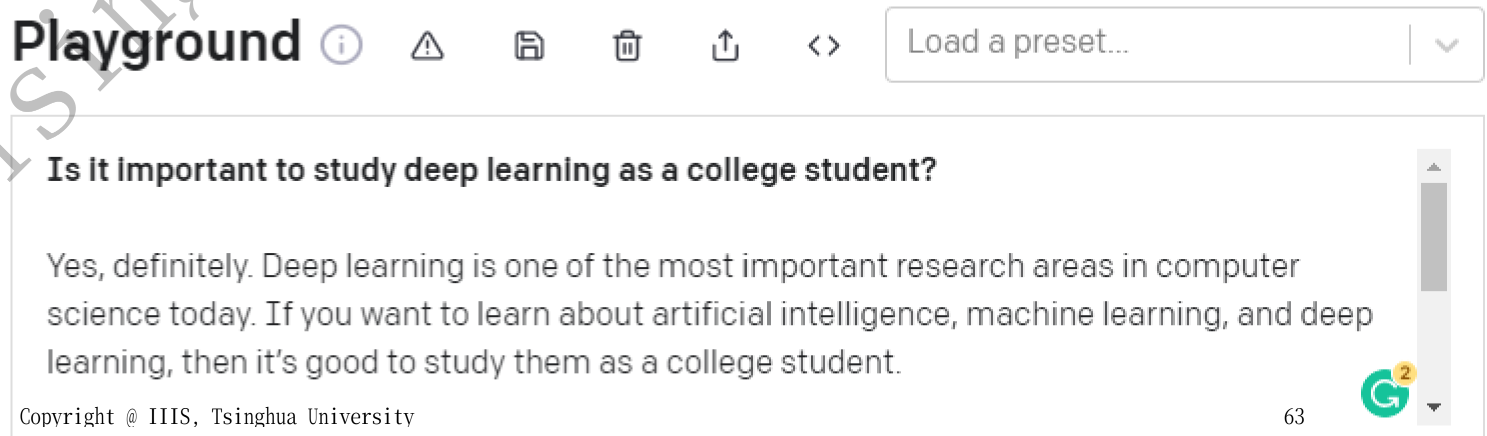
- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t | X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Language generation
  - Draw $X \sim P(X; \theta)$
  - Sample $X_1 \sim Y_1(h_0, c_0, \#; \theta)$
  - Generate X_2
  - Generate X_3
  - Repeat
- Remark
  - Ensure 1 position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



Language Model

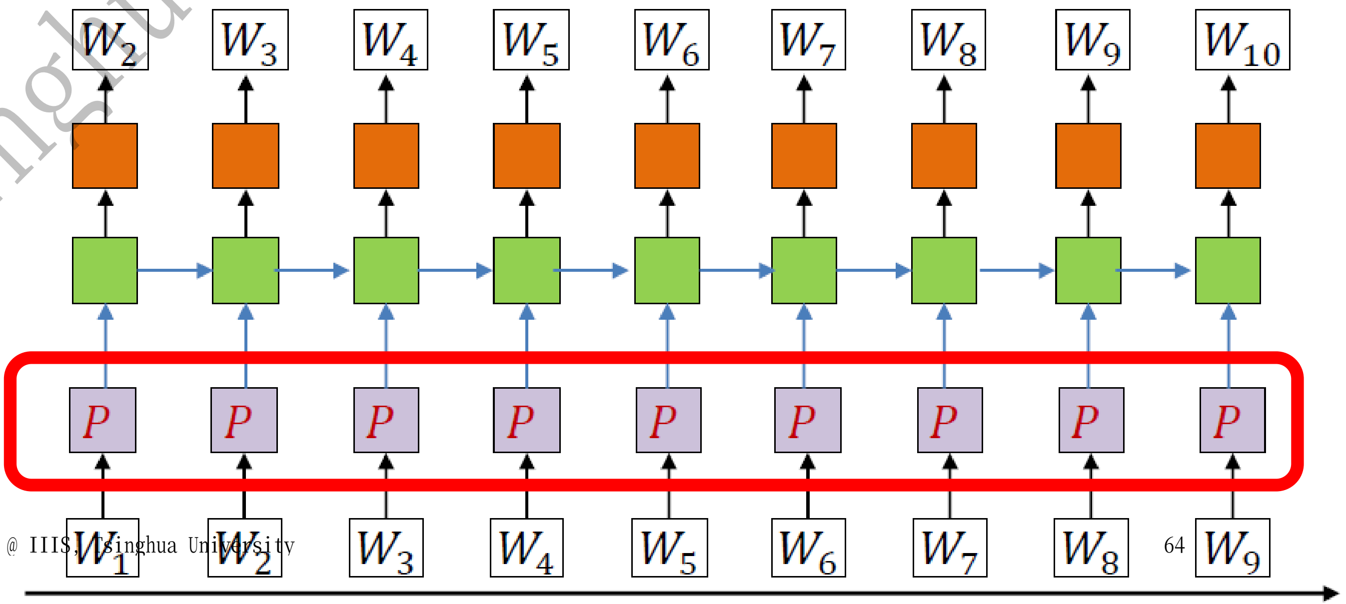
- We can also generate a language with any given prefix
  - Given $w_{1\sim3}$, we can sample $P(W|w_{1\sim3};\theta)$
  - E.g.: question answering





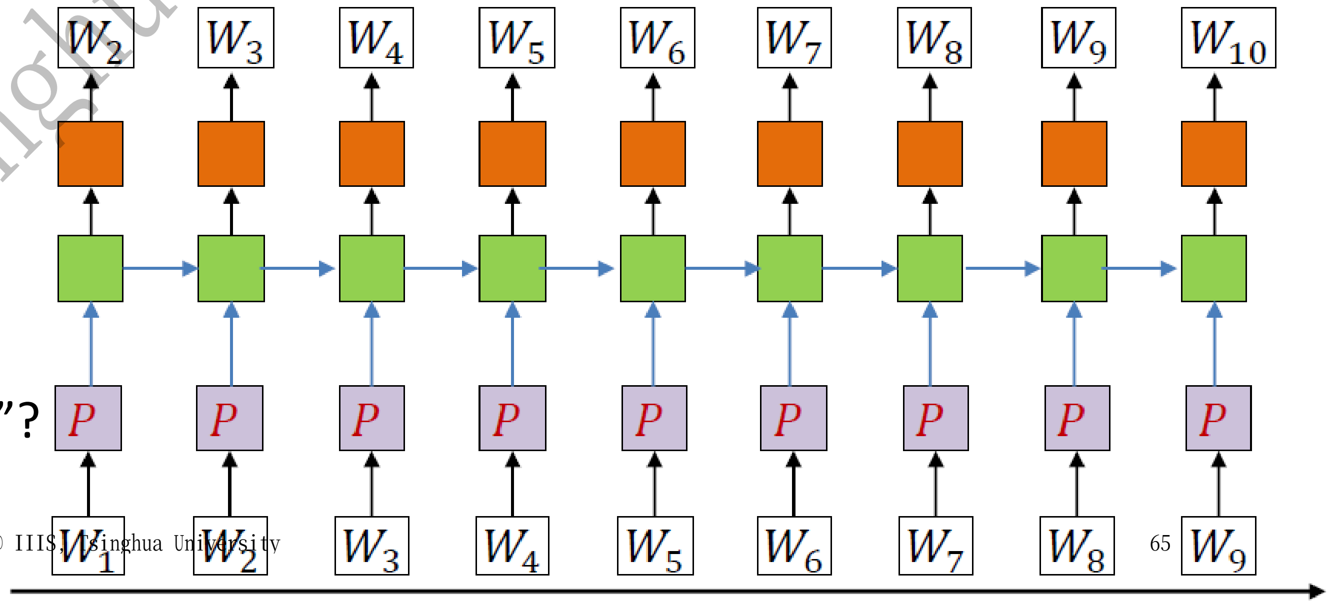
Language Model

- 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?



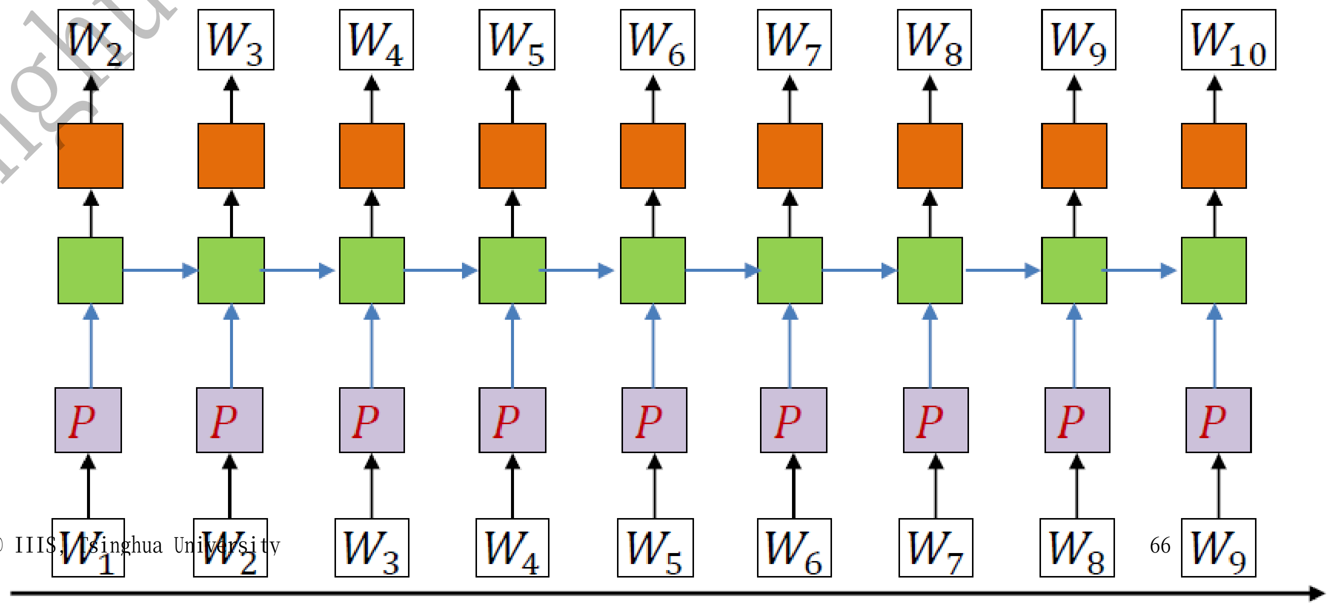
Language Model

- 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 corpus D has no "Shanghai"?



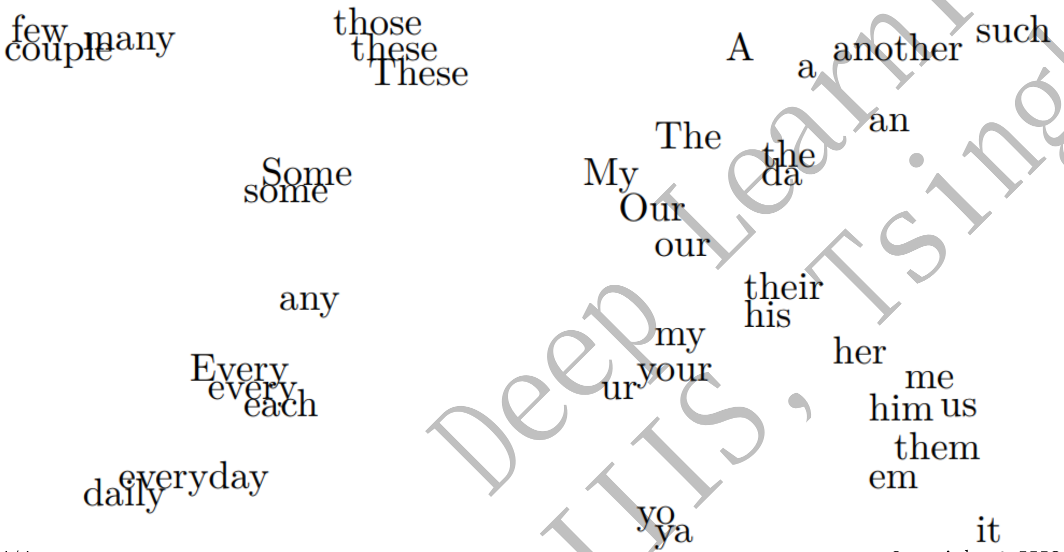
Language Model

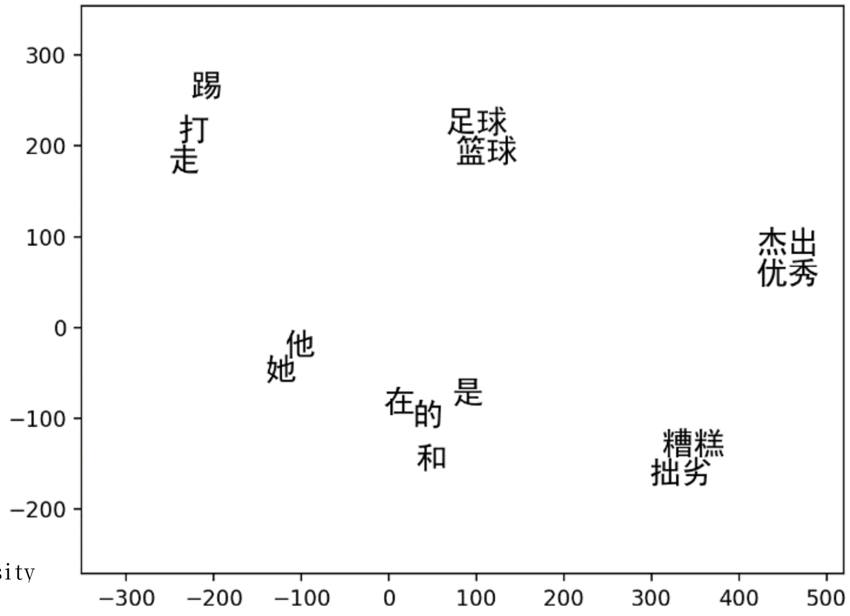
- LSTM language model
  - $Y_t = \text{softmax}(h_t) = P(X_t | X_{i<t})$
  - $h_t, c_t = \text{LSTM}(X_{t-1}, h_{t-1}, c_{t-1})$
- Goal: learn meaningful continuous representation for words
  - E.g., "Beijing"
    - It is a city in China
    - It is a noun
    - It is a capital
    - Close to "Shanghai"
    - 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, NeurIPS 2023 test-of-time)
  - An efficient toolbox for learning word embeddings
  - Formulation
    - Given $2k$ context (上下文) vectors $c_{-k} c_{-k+1} \dots c_{-1} ? \, c_1 c_2 \dots c_k$
    - $P(w|c_{-k} \dots c_k)$: 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)
  - 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: $c_{-k}c_{-k+1}\ldots c_{-1}wc_1c_2\ldots c_k$
    - $P(+|c_{-k}\ldots w\ldots c_k)$: the probability of w should appear with c

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

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

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

  - 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

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

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

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

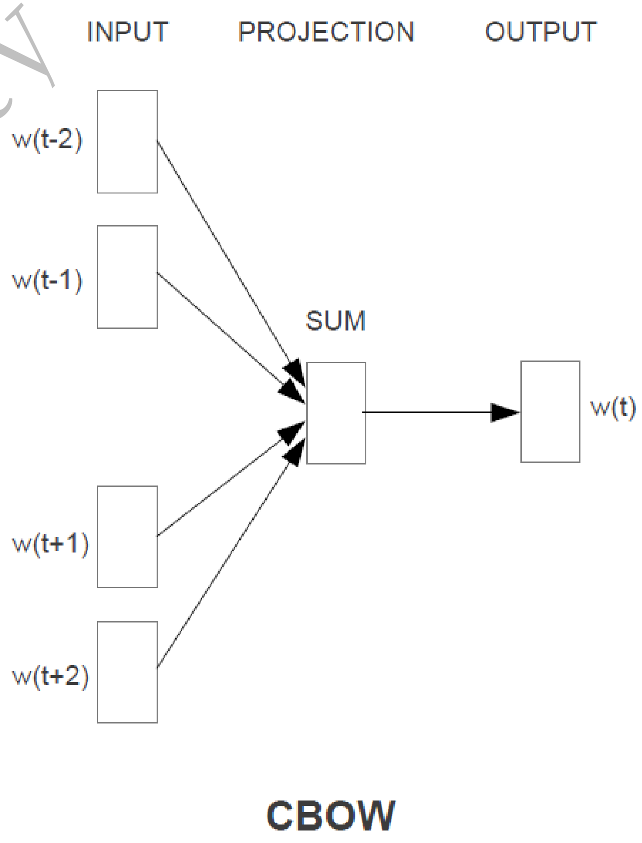
  - MLE Training!
    - Positive (w, c) pairs: all the text chunks of length $2k + 1$ from training corpus D
    - We need negative pairs!
      - Choose a context c, and select random negative words w'
      - *This training method is called* ***negative sampling***

Word Embedding

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

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

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



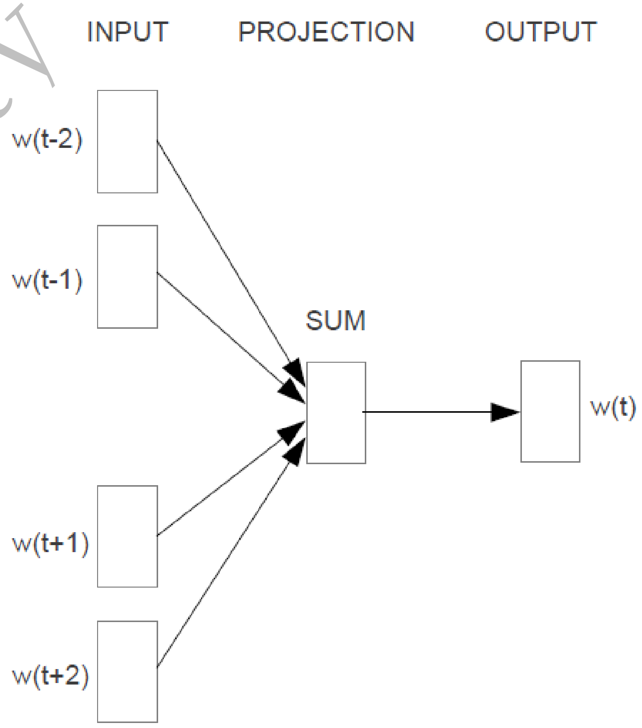
  - CBOW Training corpus D
    - For every text chunks $(c_{-k} \dots, w, \dots, c_k)$ in D
    - Collect positive data pair (c, w), add to D^+
    - Random choose a word w', add (c, w') to D^-
  - MLE Training

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

  - 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

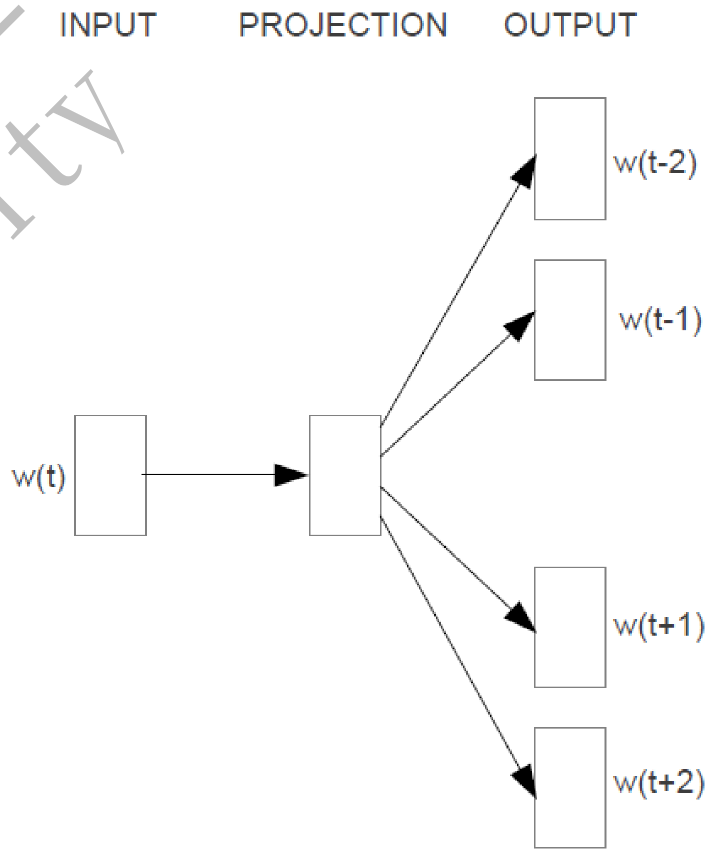
- Word2Vec (Mikolov, et al, Google, 2013)
  - Continuous Bag-of-Words (CBOW)
    - Objective $\log P(w|c_i)$
    - Use contexts c to predict center word w
  - Alternative: use w to predict surrounding words c



CBOW

Word Embedding

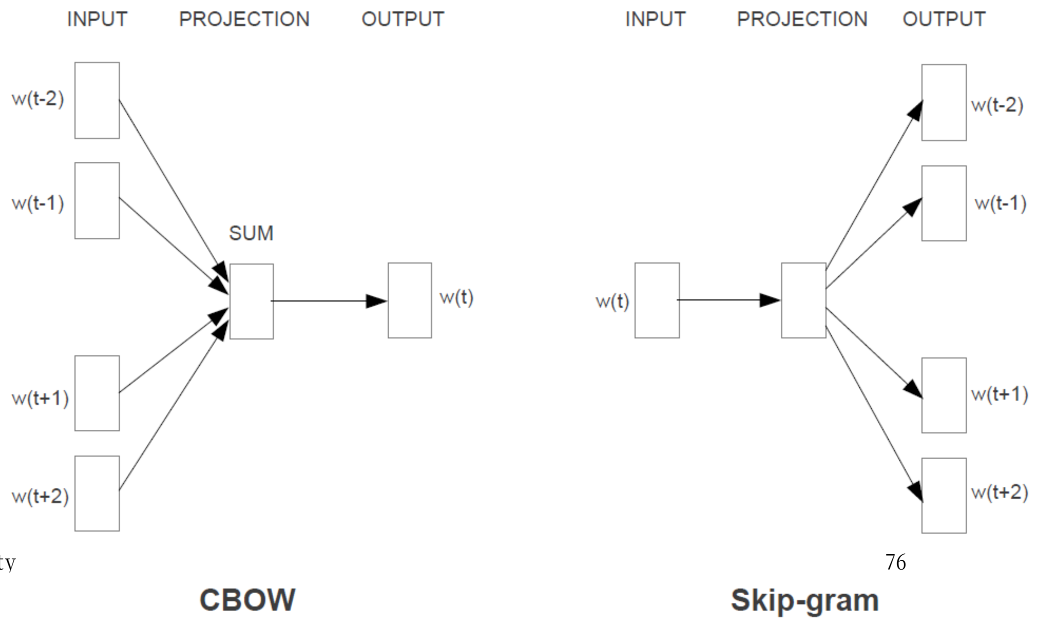
- Word2Vec (Mikolov, et al, Google, 2013)
  - Continuous Bag-of-Words (CBOW)
    - Objective $\log P(w|c_i)$
    - Use contexts c to predict center word w
  - Skip-Gram Model
    - Use a single center word c to predict $w_{-k}, \ldots w_{-1}, *, w_1 \ldots w_k$
    - Objective $\log P(w_i|c)$
    - Skip-Grams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



Skip-gram

Word Embedding

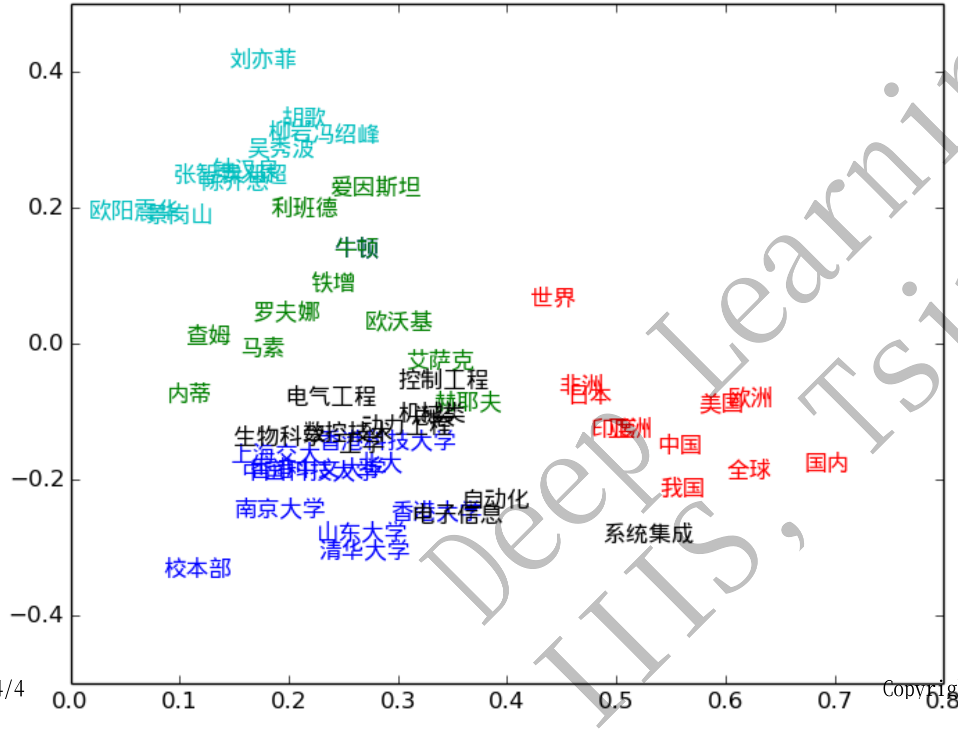
- Word2Vec (Mikolov, et al, Google, 2013)
  - Continuous Bag-of-Words (CBOW)
    - Use contexts c to predict center word w
  - Skip-Gram Model
    - Use a single center word c to predict $w_{-k}, \dots w_{-1}, *, w_1 \dots w_k$

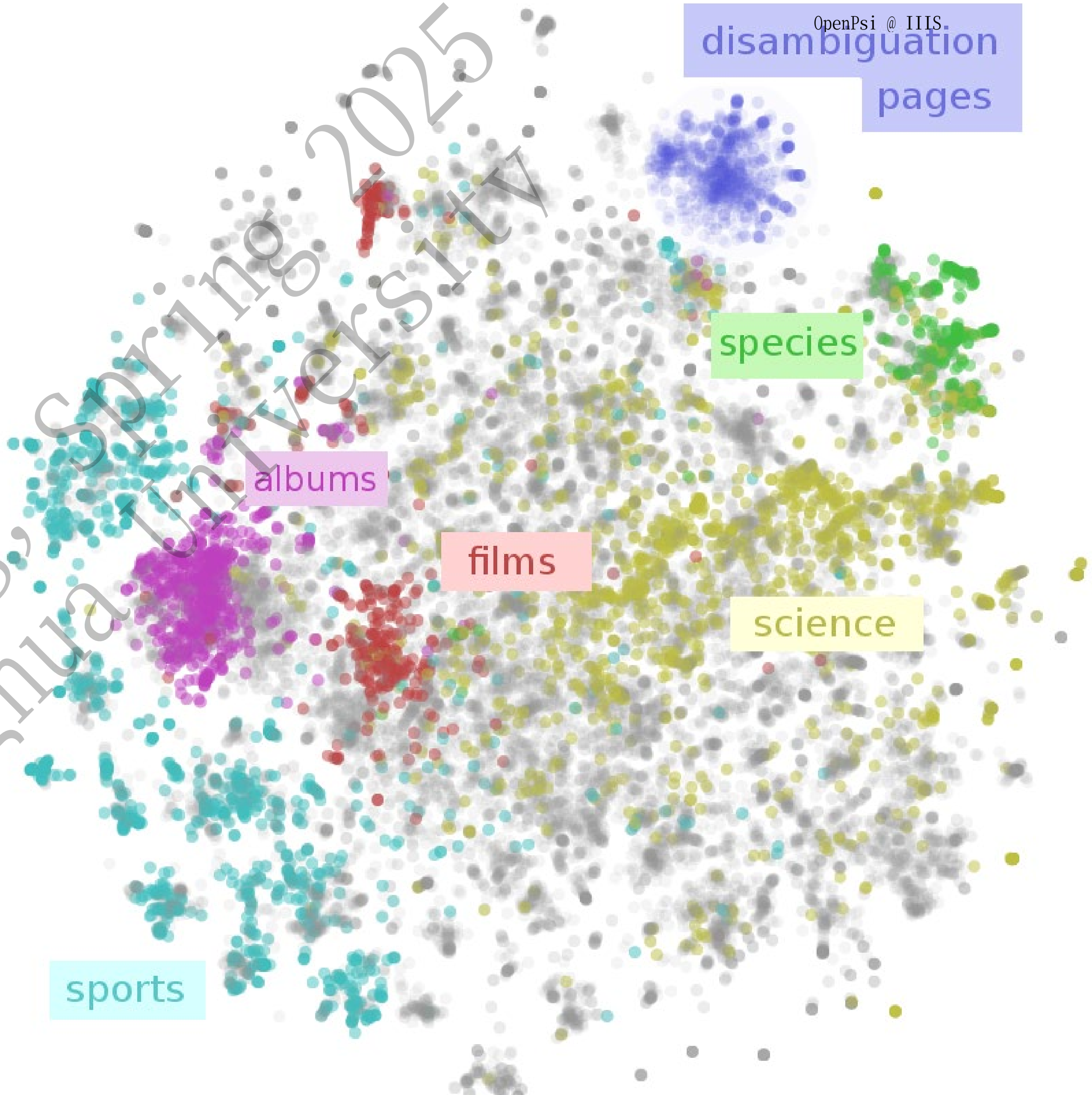


- Remark
  - CBOW trains faster than Skip-Gram
  - Skip-Gram is a harder problem
    - Harder to overfit
  - Skip-Gram performs better
    - Particularly for rare words

Word Embedding

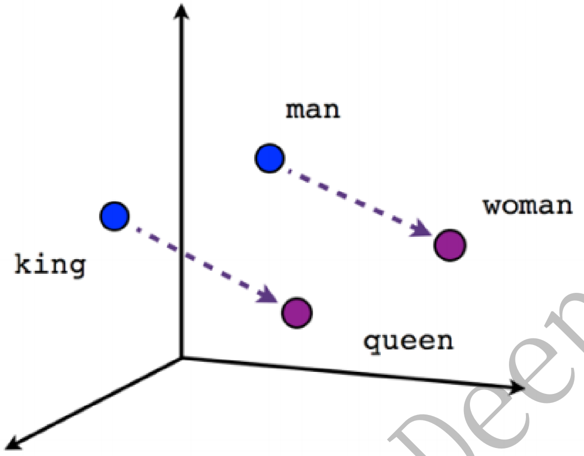
- Word2Vec Visualization
  - t-SNE projection in 2D
  - Similar topics cluster together



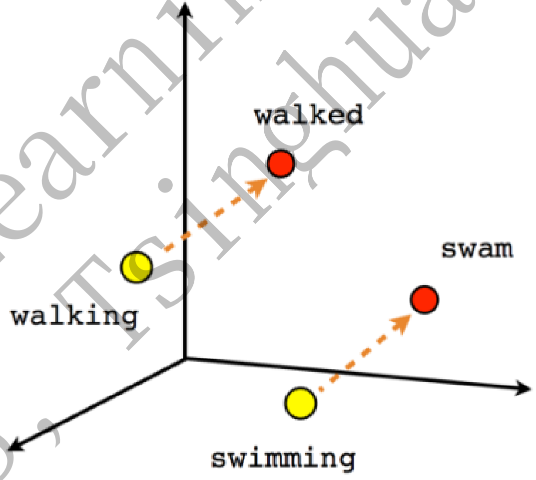


Word Embedding

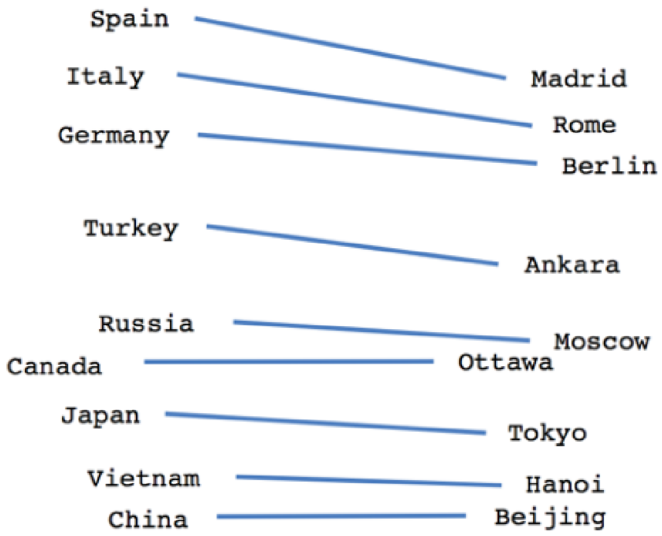
- Word2Vec Vector Arithmetic
  - Emergent analogies
    - king – man + woman ≈ queen
    - Beijing – China + France ≈ Paris



Male-Female



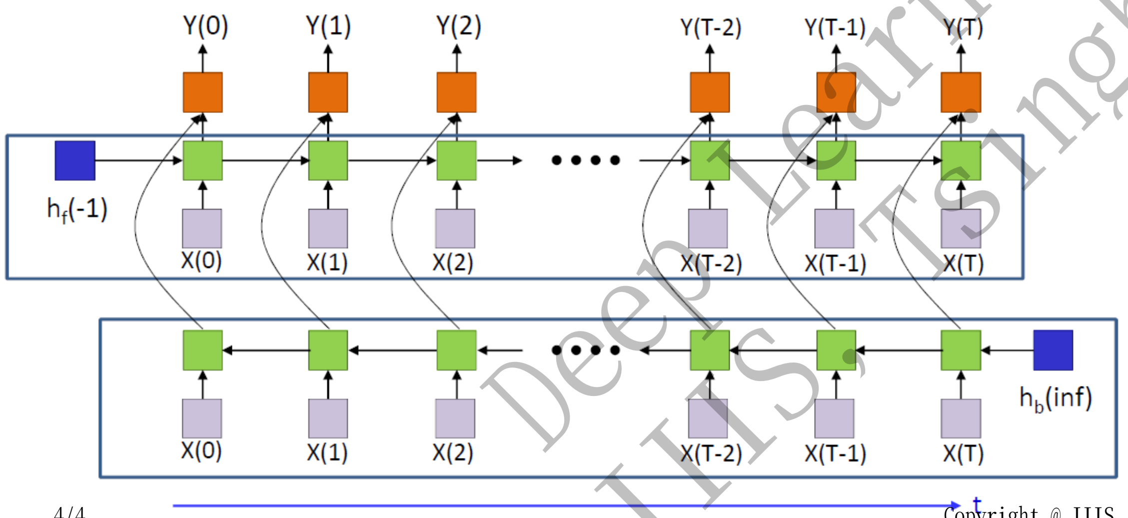
Verb tense

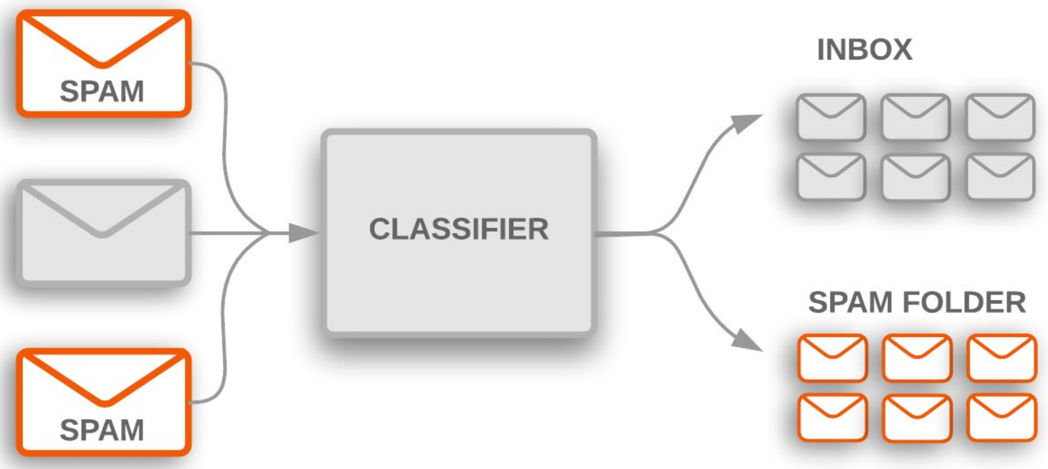


Country-Capital

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

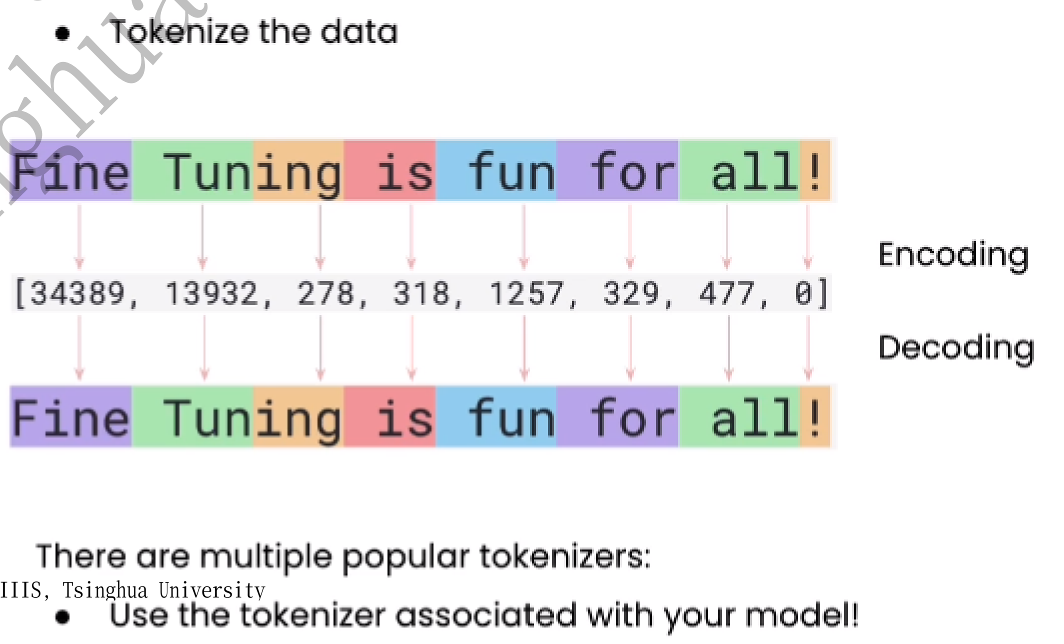
- Pre-processing
  - Collect a large corpus and learn word embeddings (word2vec)
- Text classification
  - Bi-directional LSTM and then run Softmax on final hidden states
- Text generation
  - For the specific training domain, learn an autoregressive model $P(X)$
- Text correction (文本改错)
  - MCMC over $P(X)$ 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”)
      - Use <unk> for very rare words
      - Dictionary is much smaller for pure Chinese (your homework ☺)
  - 分词 word segmentation
    - An issue in Chinese if you want to use word model: 中关村北大街
  - 词干化 stemming
    - Has → have; running → run
    - Apples → apple
  - 词条化(令牌化) tokenization
    - A 11-year-old boy; 12345*54321=670592745 **(still critical in modern LMs)**

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”)
      - Use <unk> for very rare words
      - Dictionary is much smaller for pure
  - 分词 word segmentation
    - An issue in Chinese if you want to us
  - 词干化 stemming
    - Has → have; running → run
    - Apples → apple
  - 词条化(令牌化) tokenization
    - A 11-year-old boy; 12345*54321=67
    - **Subword tokenization**

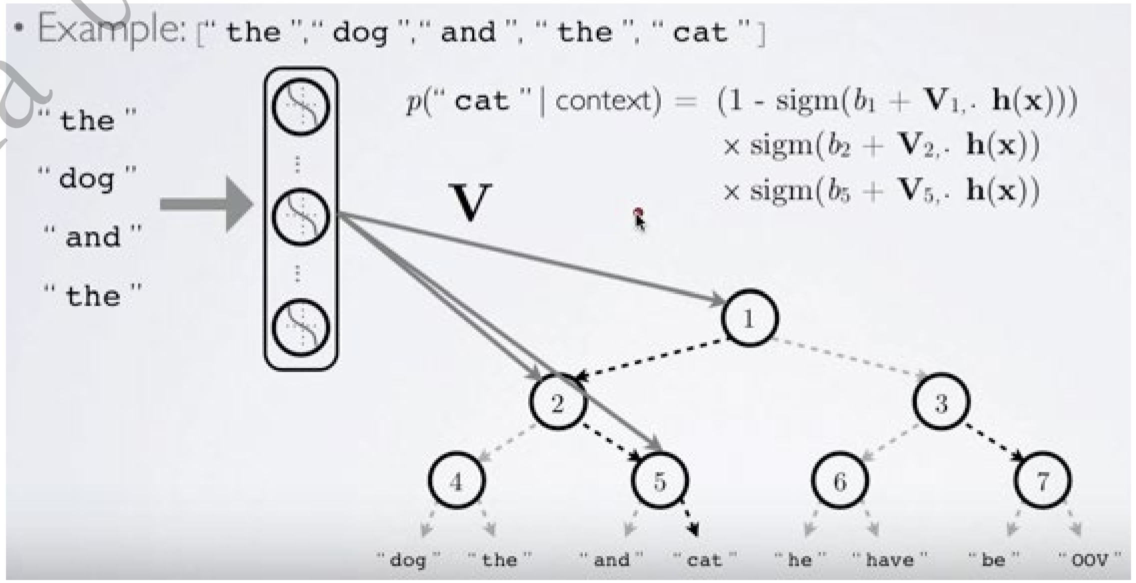


Computation Techniques

- 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…*

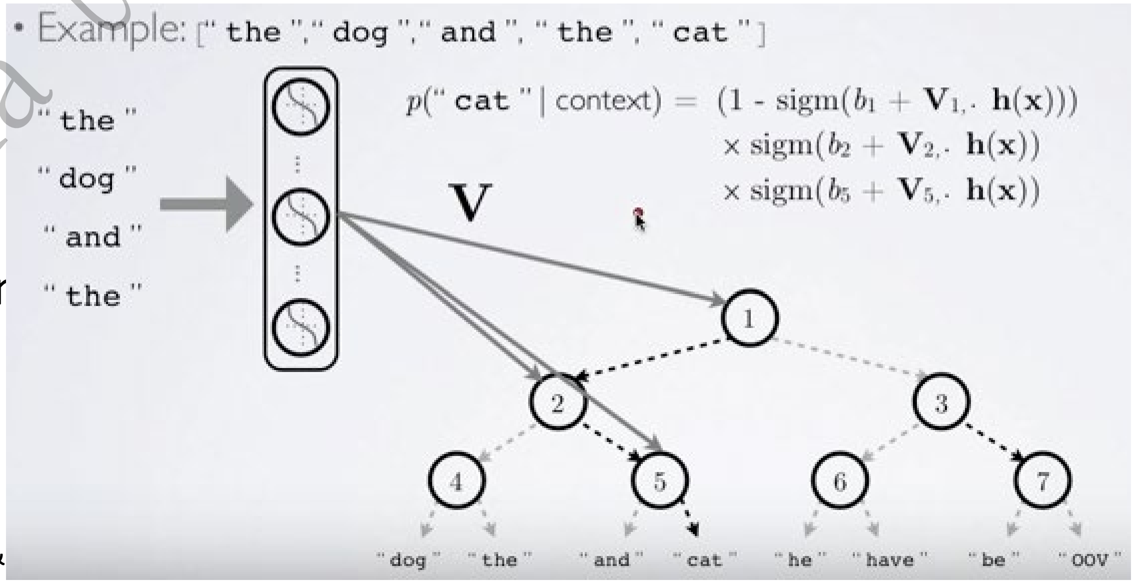
Computation Techniques

- Language Model Learning
  - MLE learning: $P(X_t|X_{i<t})$
  - The expensive Softmax operator
  - Hierarchical Softmax
    - Build a binary tree: $O(V) \to O(\log V)$
    - 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
  - MLE learning: $P(X_t|X_{i<t})$
  - The expensive Softmax operator
  - 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 conter
      - 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 &
      - https://papers.nips.cc/paper/2008/file/1e056d2b0ebd5c878c550da6ac5d3724-Paper.pdf



Advanced Techniques

- Language Model Learning
  - 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 $P(\tilde{X}_{i\le t})$
    - No guarantee to find the optimal solution
      - Trade-off between accuracy (exhaustive search) and efficiency (greedy)

Advanced Techniques

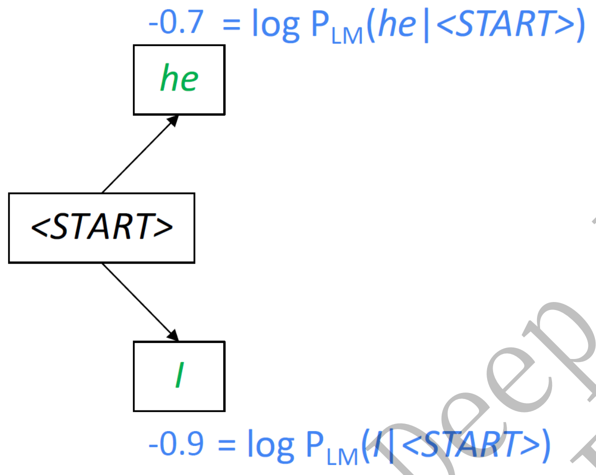
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$

<START>

Calculate prob dist of next word

Advanced Techniques

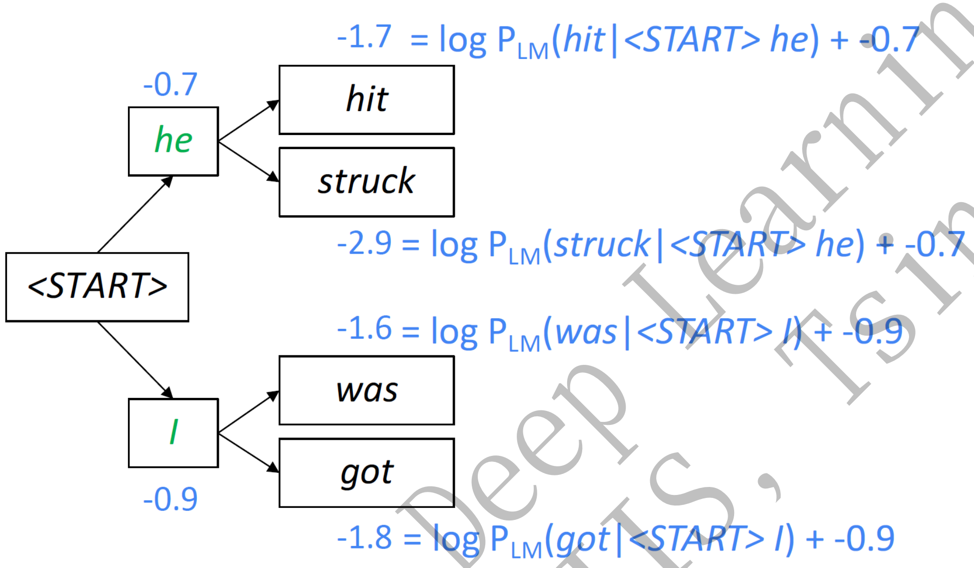
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t|X_{i<t})$



Take top *k* words and compute scores

Advanced Techniques

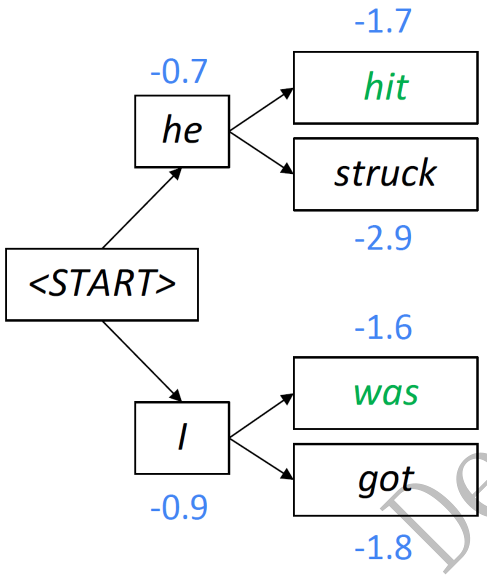
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



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

Advanced Techniques

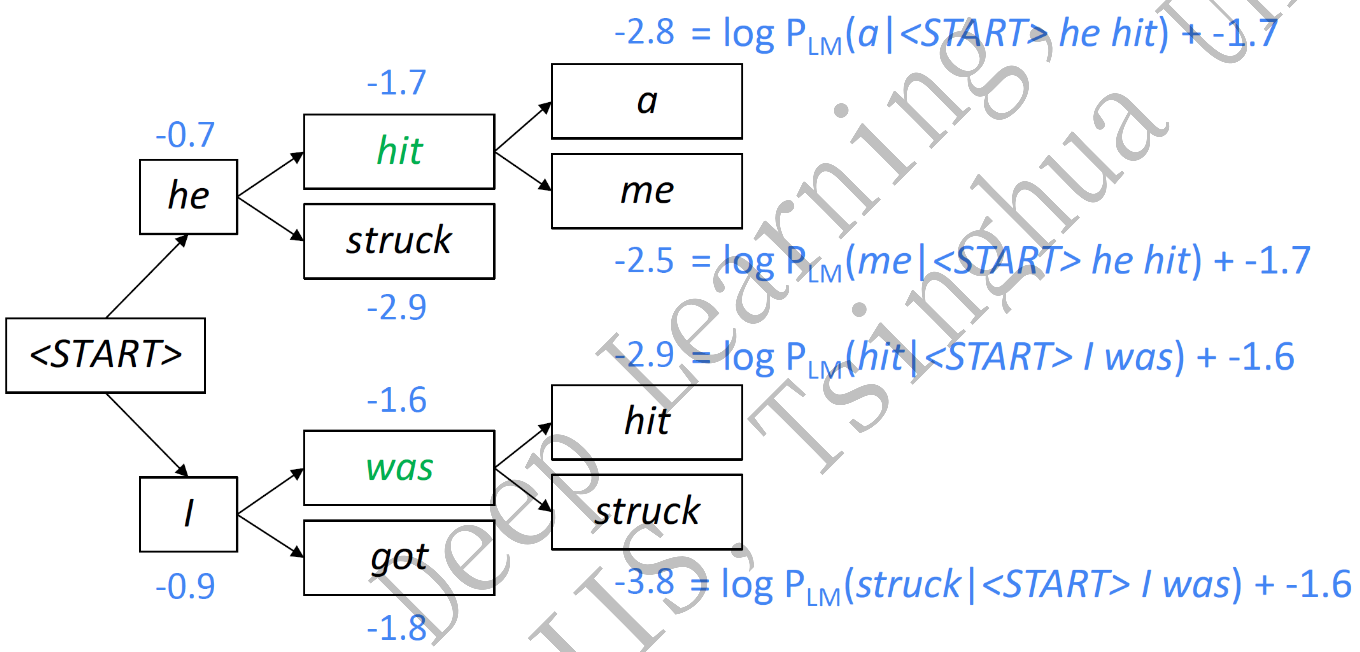
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



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

Advanced Techniques

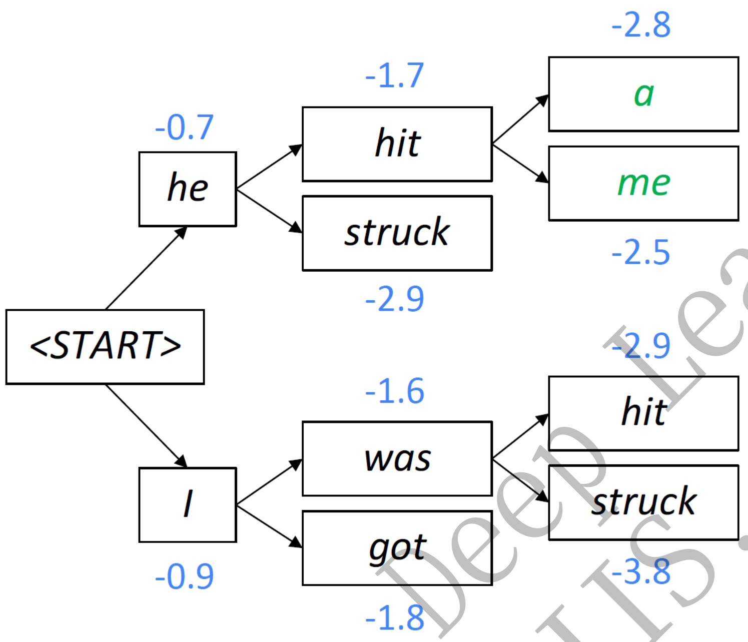
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



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

Advanced Techniques

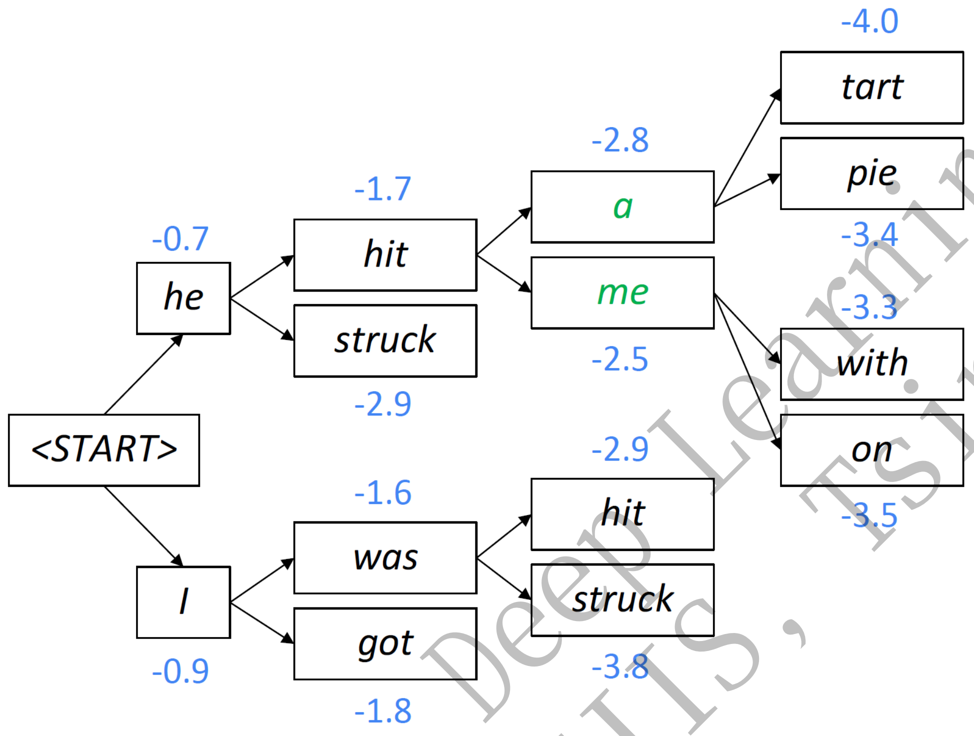
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



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

Advanced Techniques

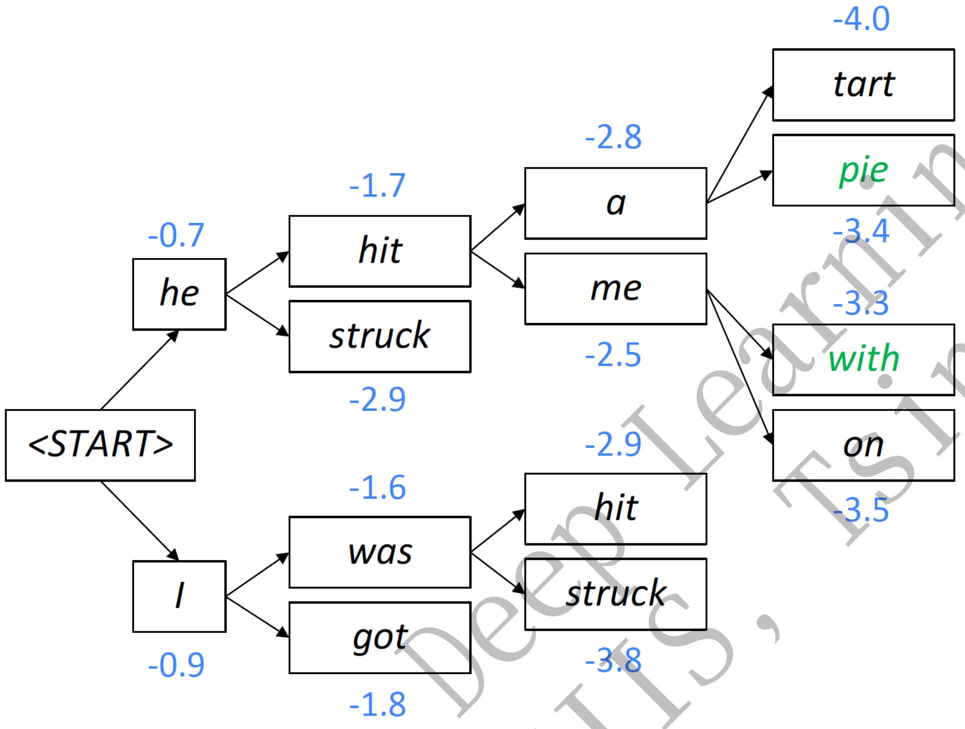
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



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

Advanced Techniques

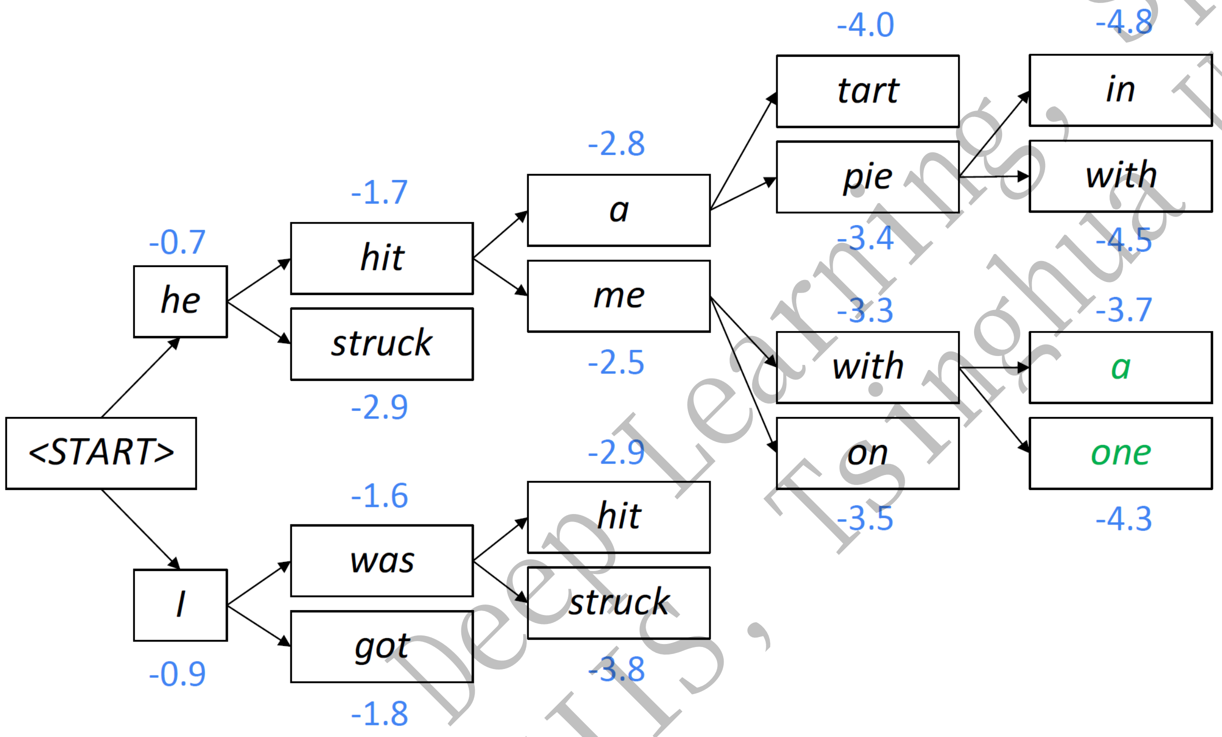
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t | X_{i<t})$



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

Advanced Techniques

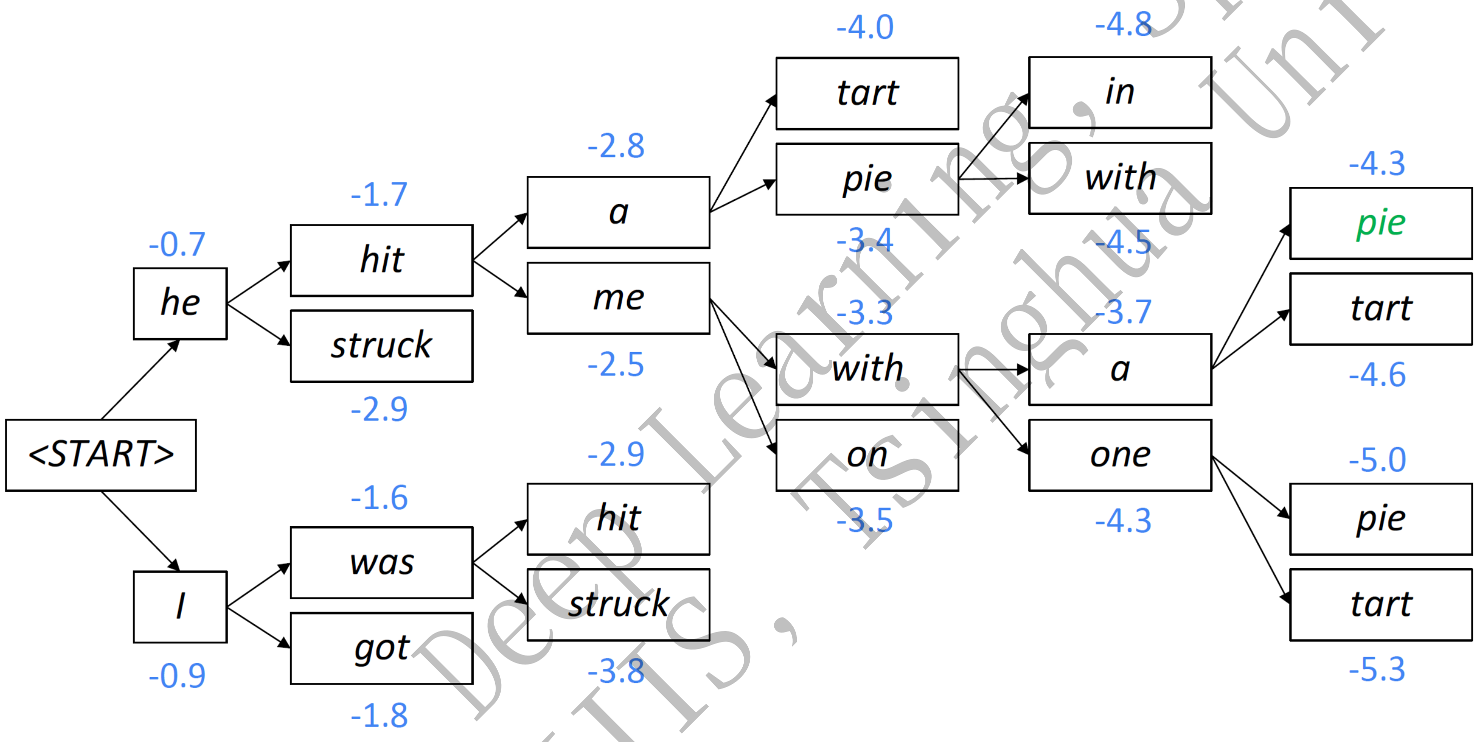
- Beam Search
  - Example: $K = 2$, score = $\log P(X) = \sum_t \log P(X_t|X_{i<t})$



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)

Advanced Techniques

- Language Model Learning
  - The expensive softmax operator
- Language Model Inference
  - Beam search for the best generated sequence
    - You can also include a temperature parameter in score if you want diverse texts
- Improving the word representation
  - So far, we assume a static (pretrained) embedding
  - Issue: the same word in different contexts may have different meaning
    - Teddy **bear** v.s. I cannot **bear** him any more
    - A nice **weather** v.s. I'm under the **weather** today
    - **苟**富贵，毋相忘 v.s. **苟**全性命于乱世
  - Word embeddings should be context-aware!

Advanced Techniques

- Deep contextualized word representations (EMNLP2018)
  - Idea: a word feature should be related to the whole contexts
    - Including both previous words and future words
  - ELMo
    - (Optional) Use word2vec to pretrain static word embeddings w_s
    - Train a (stacked) bidirectional RNN language model $g_f(w, h)$ and $g_b(w, h)$ use w_s
    - Fix the RNN model g_f and g_b
    - For a sequence for a specific task, for the t-th word
      - Run g_f and g_b on the sentence to get h_t^f and h_t^b
      - Use $[w_t, h_t^f, h_t^b]$ as embedding
  - Remark:
    - Bidirectional LSTM is critical!

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Advanced Techniques

- Language Model Learning
  - The expensive softmax operator
- Language Model Inference
  - Beam search for the best generated sequence
    - You can also include a temperature parameter in score if you want diverse texts
- Contextualized Word Embedding
  - ELMo: use contexts to compute features of a word

- More techniques in your NLP course ☺

Summary

- Recurrent neural network (RNN) for sequence data
  - Vanishing/exploding gradients/value
- Long Short-Term Memory networks (LSTM)
  - An RNN architecture for long-term dependency
- Language Model
  - Auto-regressive model over texts & LSTM applications
  - Word2vec for word representation
  - Hierarchical Softmax for more efficient softmax
  - Beam search for the best output
  - Elmo for contextualized representation
- Next lecture: more advanced sequence modeling techniques

Thanks!