Neural Sequence Models: From RNNs to Attention

COM6513/4513 Natural Language Processing

Nafise Sadat Moosavi

Readings:
Chapter 13 from Jurafsky and Martin

Attendance Monitoring/Questions?

TG-YD-HQ

Recap

Why is the chain rule essential for language modeling, and what practical assumption do n-gram models make to make it computationally feasible?

Recap

If a perfect language model (perplexity = 1) existed, would that mean it understands language?

Lecture Plan

- Neural Language Models Beyond Fixed Context
- Recurrent Neural Networks
- LSTM and GRU
- Structural Limits of Recurrence
- Attention

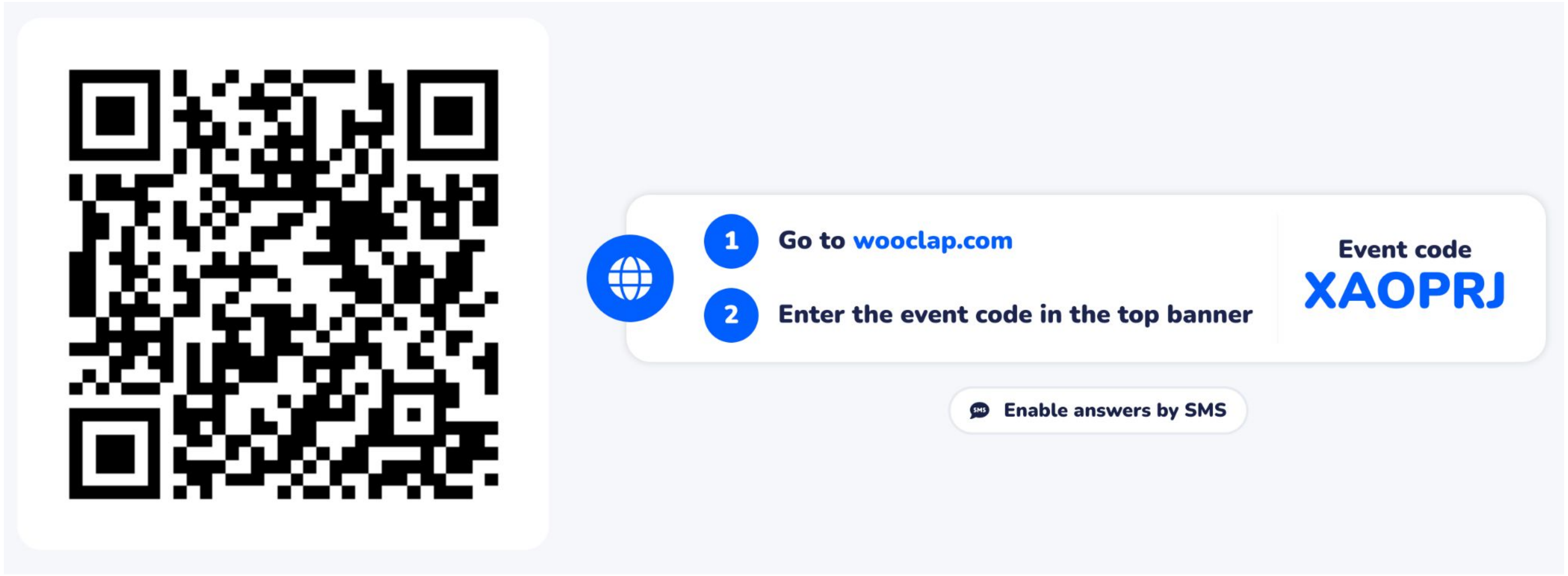
The Language Modeling Pipeline

Language Modeling System =

1. 📚 Training Data (Corpora)
2. 🔤 Input Representation (Tokenization)
3. Model Architecture
4. ⚙️ Optimization (Training Objective)
5. 📏 Evaluation
6. 🚀 Deployment / Generation

$$P(w_1, \ldots, w_T) = \prod_t P(w_t \mid w_{<t})$$

Issues of N-gram LM?



1 Go to wooclap.com

2 Enter the event code in the top banner

Event code
XAOPRJ

Enable answers by SMS

Issues of N-gram LM?

- Data sparsity
- Exponential growth of parameters
- No generalization across similar words
- Fixed length context

From Counts to Continuous Representations

N-gram Models

$$P(w_1, \ldots, w_T) = \prod_t P(w_t \mid w_{<t})$$

- Computing probabilities by counting
- Discrete one-hot word representations
- Separate parameter for each context
- No sharing across similar words
- Exponential growth: $O(V^n)$

$$P(w_t \mid w_{t-n+1:t-1}) = \frac{C(w_{t-n+1:t})}{C(w_{t-n+1:t-1})}$$

From Counts to Continuous Representations

Neural Models

$$P(w_1, \ldots, w_T) = \prod_t P(w_t \mid w_{<t})$$

- Replace counting with a learned function
- Words represented as dense vectors (embeddings)
- Parameters shared across contexts
- Generalization via distributed representations

$$P(w_t \mid \text{context}) = \text{softmax}(f_\theta(\text{context}))$$

From Counts to Continuous Representations

Neural Models

$$P(w_1, \ldots, w_T) = \prod_t P(w_t \mid w_{<t})$$

- Replace counting with a learned function
- Words represented as dense vectors
- Parameters shared across contexts
- Generalization

$$P(w_t \mid \text{context}) = \text{softmax}(f_\theta(\text{context}))$$

Moving from memorizing observed patterns to learning continuous representations that generalize

From Counts to Continuous Representations

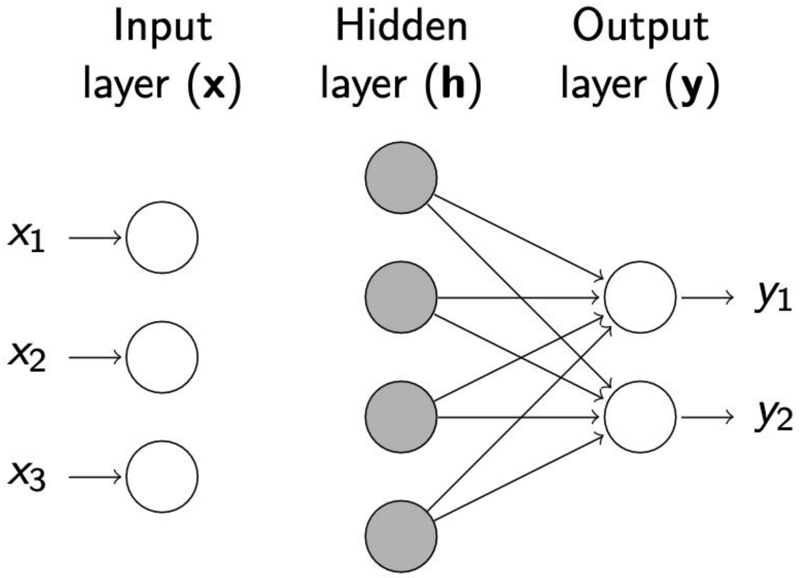
Neural Models

$$P(w_1, \ldots, w_T) = \prod_t P(w_t \mid w_{<t})$$

- Replace counting with a learned function
- Words represented as dense vectors
- Parameters shared across contexts
- Generalization

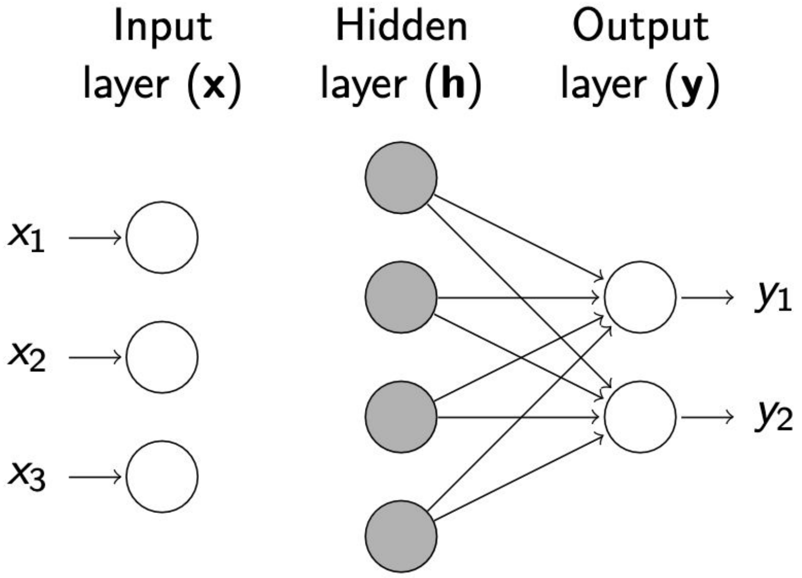
$$P(w_t \mid \text{context}) = \text{softmax}(\boxed{f_\theta}(\text{context}))$$

FF Neural Networks (recap)


Input layer ($\mathbf{x}$)
Hidden layer ($\mathbf{h}$)
Output layer ($\mathbf{y}$)
x_1
x_2
x_3
y_1
y_2

$$\mathbf{h} = g(\mathbf{x}^T\mathbf{W}_h)$$
$$\mathbf{y} = softmax(\mathbf{h}^T\mathbf{W}_o)$$
$$\mathbf{W}_o \in \mathcal{R}^{h \times y}$$

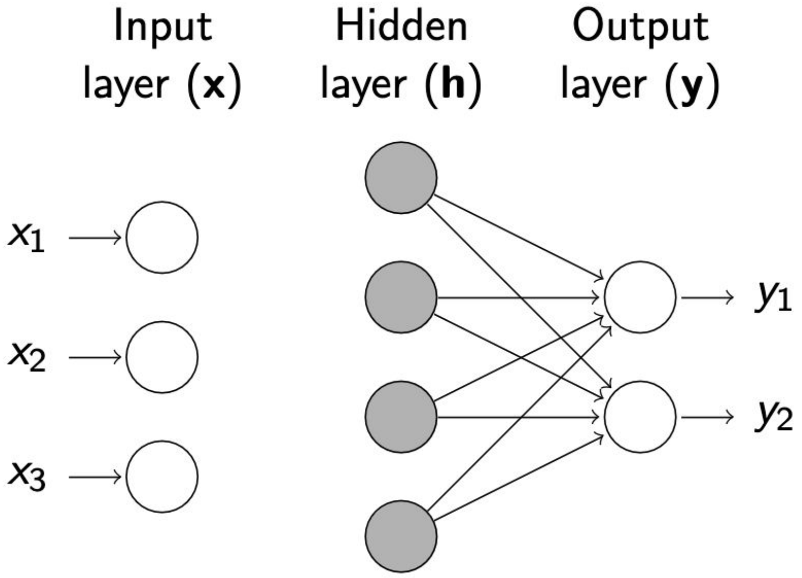
FF Neural Networks (recap)



$$\mathbf{h} = g(\mathbf{x}^T \mathbf{W}_h)$$
$$\mathbf{y} = softmax(\mathbf{h}^T \mathbf{W}_o)$$
$$\mathbf{W}_o \in \mathcal{R}^{h \times y}$$

Output dimension?

FF Neural Networks (recap)


Input layer ($\mathbf{x}$)
Hidden layer ($\mathbf{h}$)
Output layer ($\mathbf{y}$)
x_1
x_2
x_3
y_1
y_2

$$\mathbf{h} = g(\mathbf{x}^T \mathbf{W}_h)$$
$$\mathbf{y} = softmax(\mathbf{h}^T \mathbf{W}_o)$$
$$\mathbf{W}_o \in \mathcal{R}^{h \times y}$$

Why softmax?

Why Softmax

Softmax ensures:

✔ Non-negative probabilities

✔ Sum to 1

✔ Differentiable (trainable with gradient descent)

$$P(w_t = i \mid \text{context}) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Why Softmax

Enables Maximum Likelihood Estimation

Training objective:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T}\log P(w_t^{\text{true}} \mid w_{<t})$$

Training: Stochastic Gradient Descent (SGD)

```
Input: $D_{train} = \{(x_1, y_1)...(x_M, y_M)\}, D_{val} = \{(x_1, y_1)...(x_D, y_D)\},$
        $learning\ rate\ \eta,\ epochs\ e,\ tolerance\ t$
initialize $\mathbf{w}$ with zeros
for each epoch $e$ do
    randomise order in $D_{train}$
    for each $(x_i, y_i)$ in $D_{train}$ do
        update $\mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}; x_i; y_i)$
    monitor training and validation loss
    if previous validation loss $-$ current validation loss; smaller than $t$
        break
return w
```

Backpropagation (recap)

Forward Pass: Calculates activations and outputs using inputs and current weights

$$a^{[l]} = \sigma(W^{[l]}a^{[l-1]} + b^{[l]})$$

Backward Pass (Error Propagation): Computes gradient of loss w.r.t. weights using the chain rule

$$\delta^{[L]} = \frac{\partial L}{\partial a^{[L]}} \cdot \sigma'(z^{[L]}) \qquad \delta^{[l]} = (W^{[l+1]T}\delta^{[l+1]}) \cdot \sigma'(z^{[l]})$$

Weight Update: Adjusts weights in the direction that reduces loss.

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial L}{\partial W^{[l]}}$$

What FF Neural LMs Improve Over N-grams?

- ☐ Data sparsity
- ☐ Exponential growth of parameters
- ☐ No generalization across similar words
- ☐ Fixed length context

What FF Neural LMs Improve Over N-grams?

- ✓ Data sparsity
- ✓ Exponential growth of parameters
- ✓ No generalization across similar words
- ❌ Fixed length context

Fixed Context

FF LM assumes:

- Fixed window size
- Context length chosen in advance
- No memory beyond window

$$P(w_t \mid w_{t-n+1:t-1})$$

Fixed Context

Increasing window size:

- Larger input dimension
- More parameters
- More computation

$$P(w_t \mid w_{t-n+1:t-1})$$

But still finite

Doesn’t remove Markov assumption

What Do We Actually Want?

True Language Modeling Goal

$$P(w_t \mid w_{<t})$$

Conditioning on all previous words

What Do We Actually Want?

True Language Modeling Goal

$$P(w_t \mid w_{<t})$$

Conditioning on all previous words

- Storing the entire sequence explicitly
- Increasing parameters with sequence length

What Do We Actually Want?

True Language Modeling Goal

$$P(w_t \mid w_{<t})$$

Conditioning on all previous words

- Maintaining a running summary of everything seen so far

What Do We Actually Want?

True Language Modeling Goal

$$P(w_t \mid w_{<t})$$

Conditioning on all previous words

- Maintaining a running summary of everything seen so far

$$h_t = f(h_{t-1}, x_t)$$

Recurrent Neural Networks

Core Idea

Maintain a dynamic hidden state that summarizes the past:

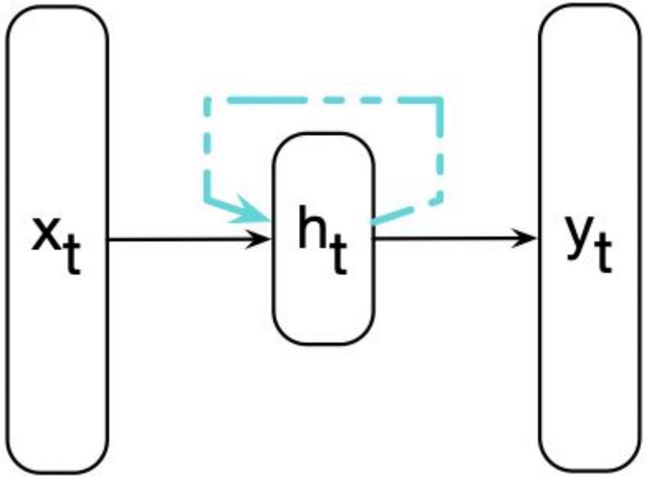
- h_t: running summary of all previous words
- Same function used at every time step
- Parameters shared across time

$$h_t = f(h_{t-1}, x_t)$$

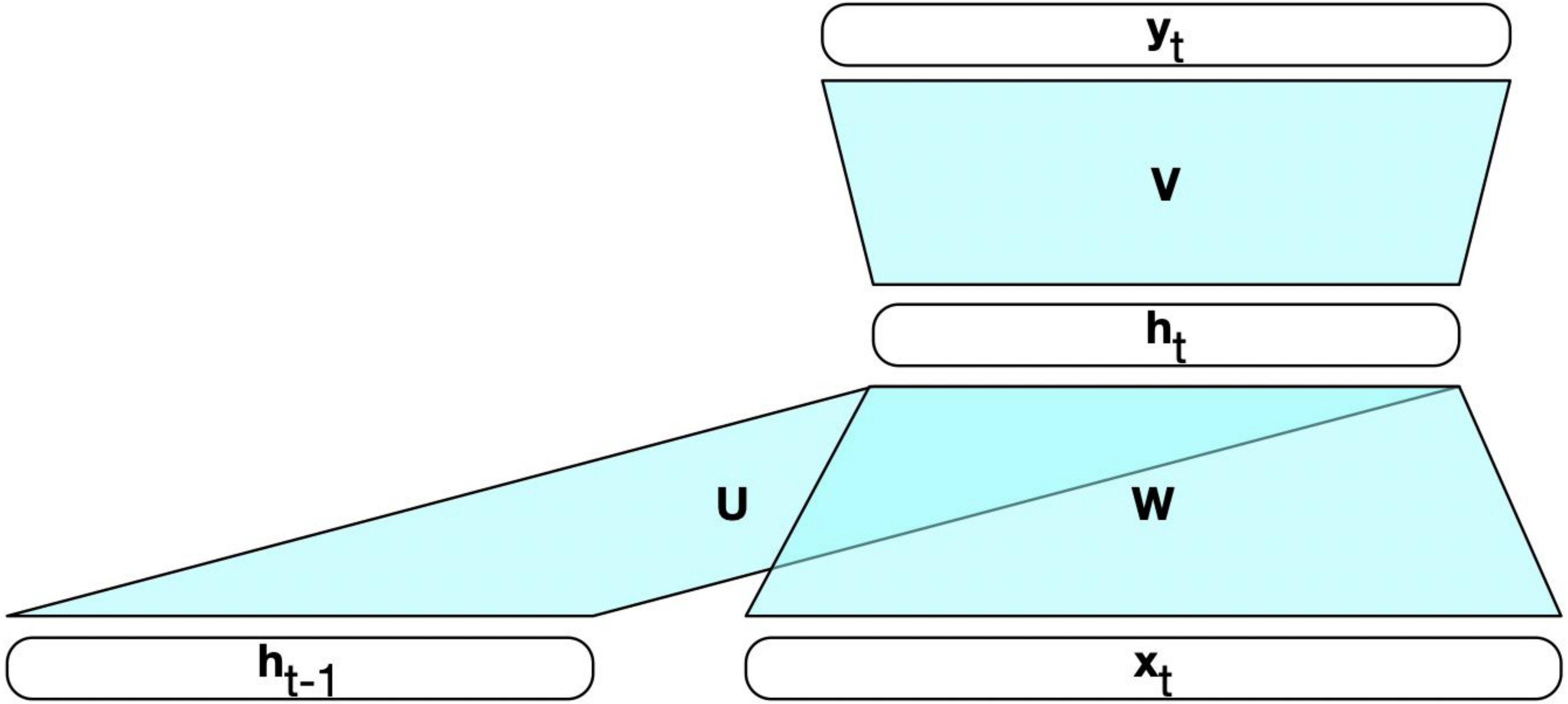
Elman Networks [Elman, 1990]

Simple recurrent networks

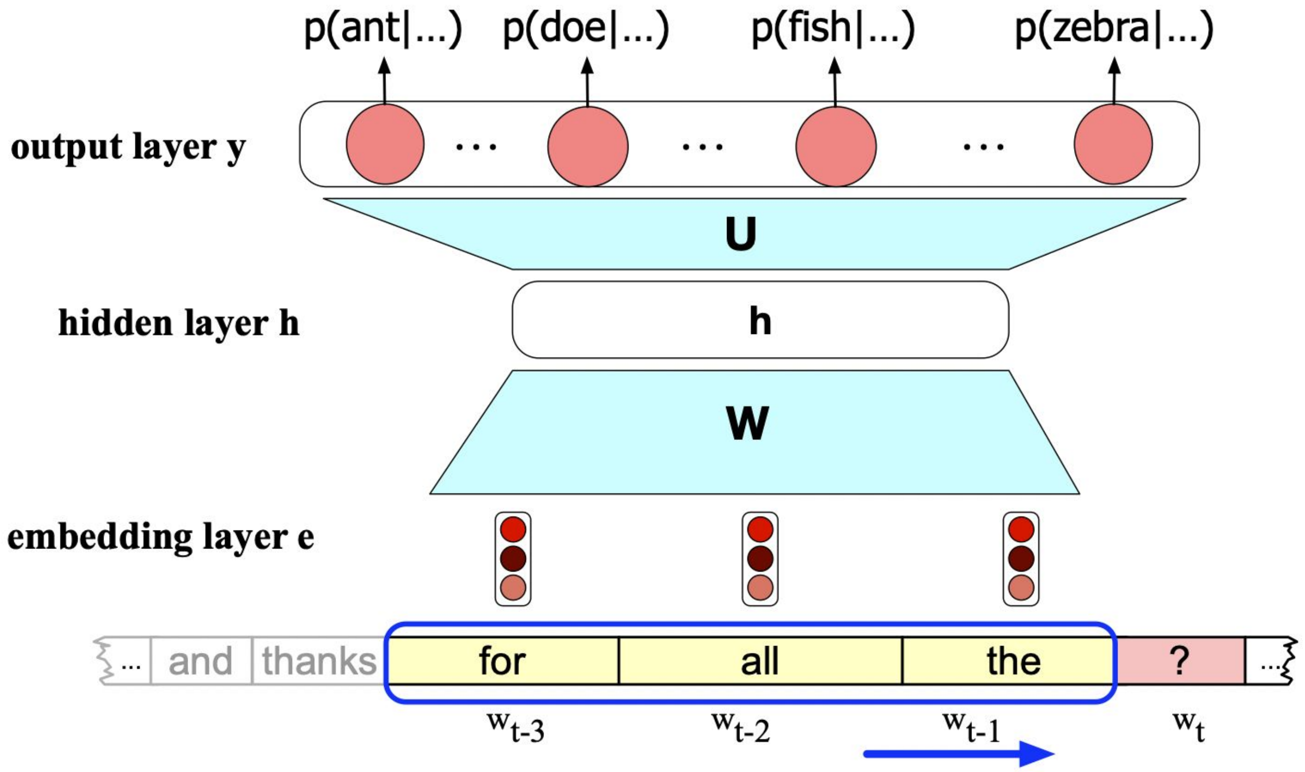
- X_t: input vector x at time t
- H_t: a form of memory with no fixed-length limit



Elman Networks [Elman, 1990]



Comparison to FFNN



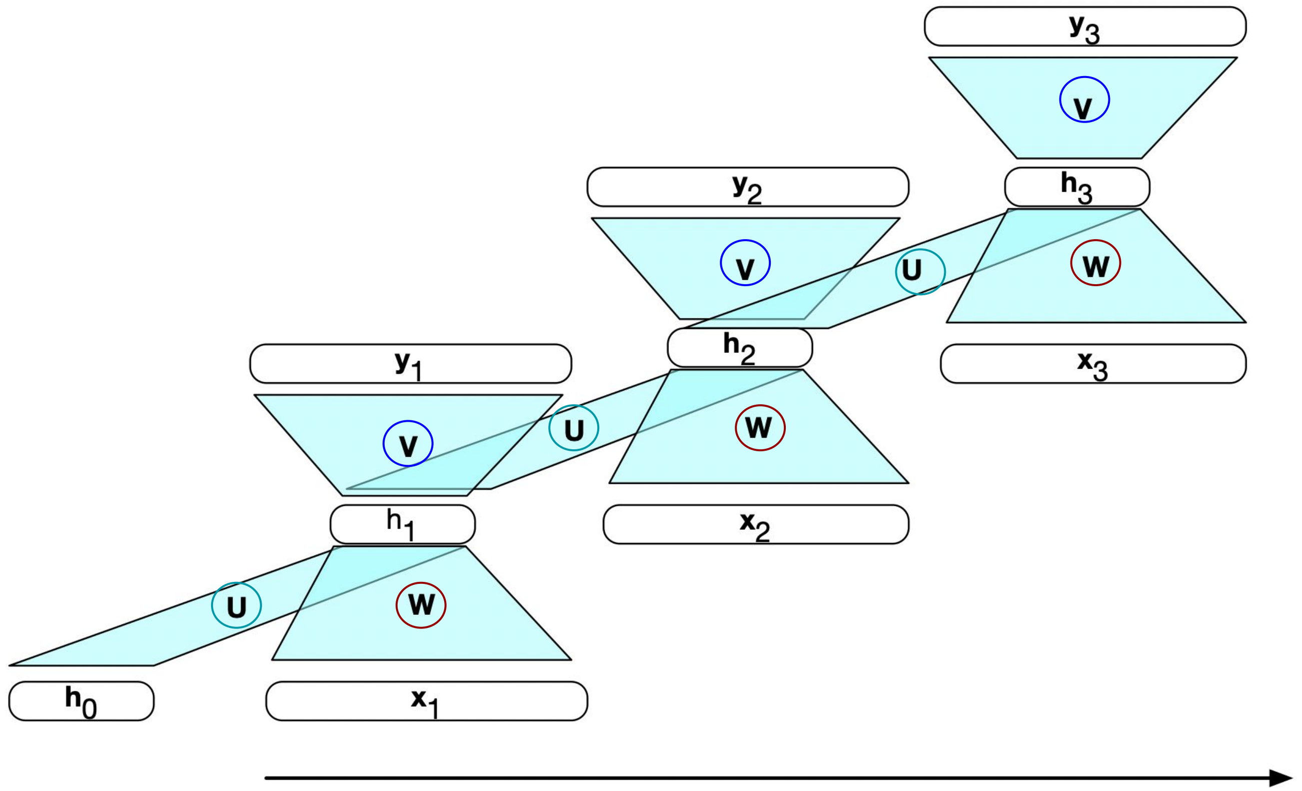
Forward inference

```
function FORWARDRNN($\mathbf{x}$, $network$) returns output sequence $\mathbf{y}$

    $\mathbf{h}^0 \leftarrow 0$
    for $i \leftarrow 1$ to LENGTH($\mathbf{x}$) do
        $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
        $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
    return $y$
```

U, V and W are shared across time

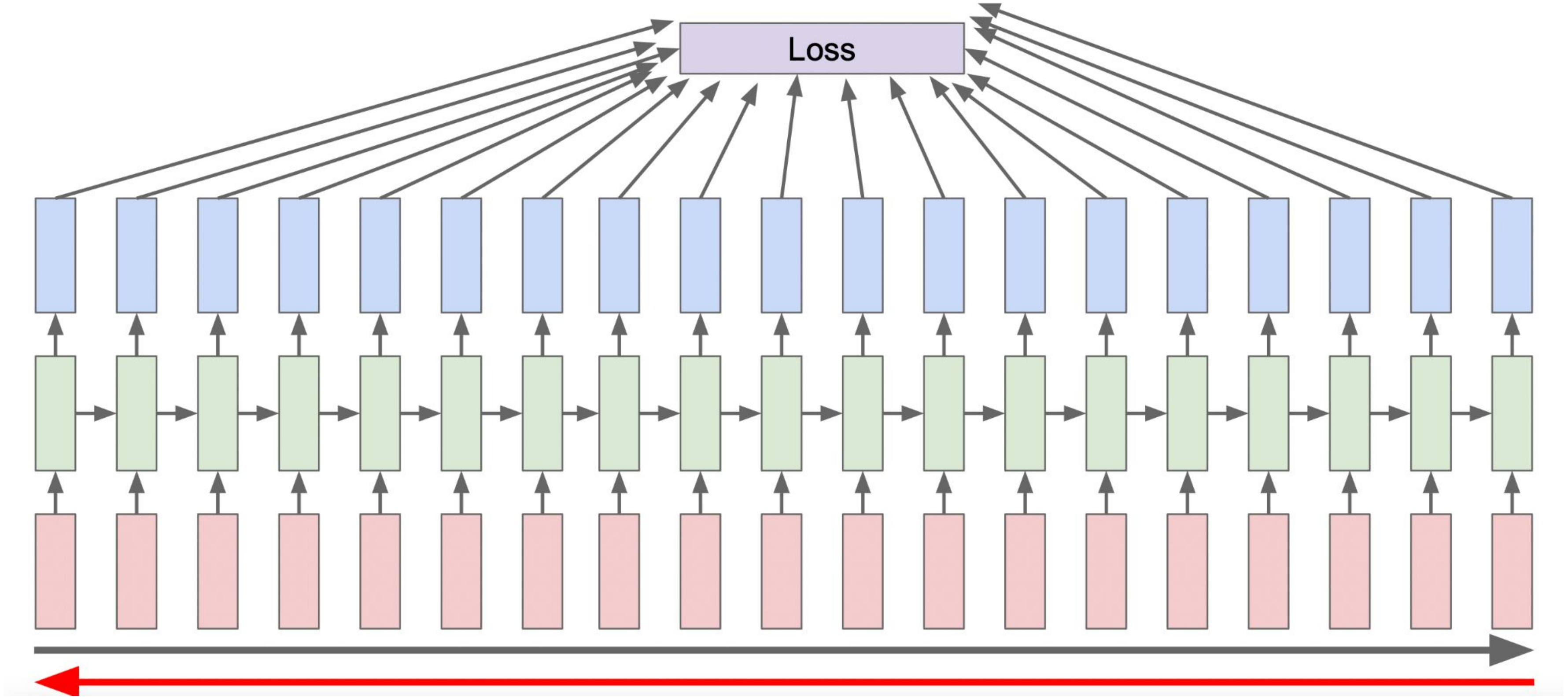
Forward Inference



Backpropagation through time (BPTT)

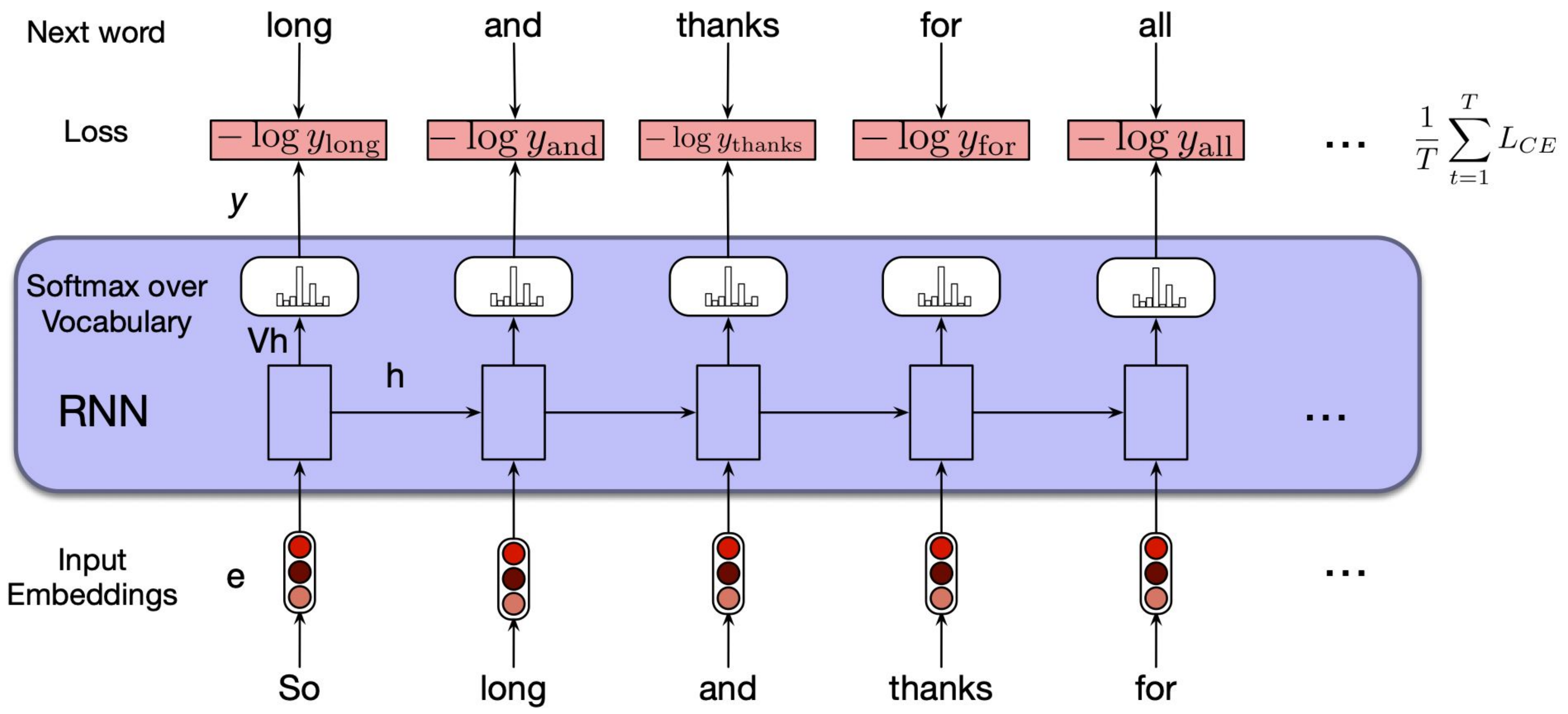
- Unfold the computational graph, and use backpropagation
- Can apply any general-purpose gradient-based techniques

Backpropagation through time (BPTT)



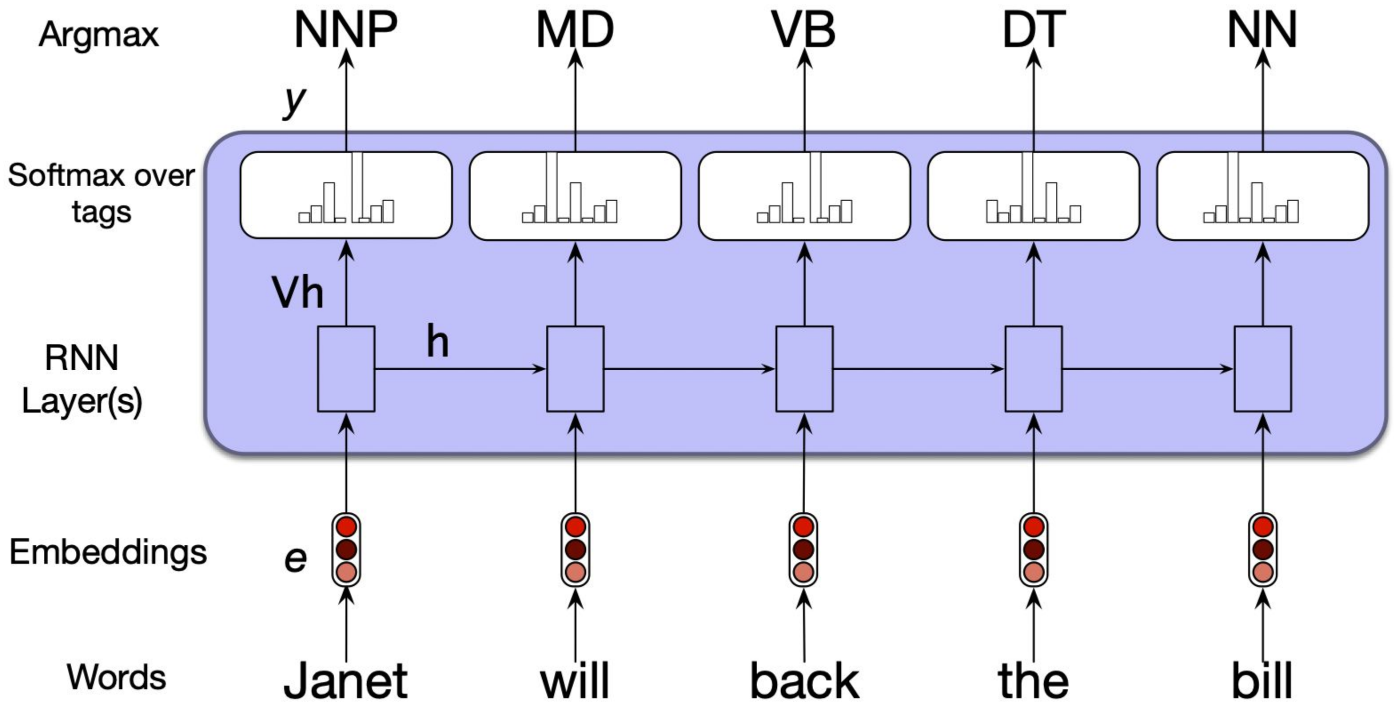
Source: http://cs231n.stanford.edu

RNN for LM



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

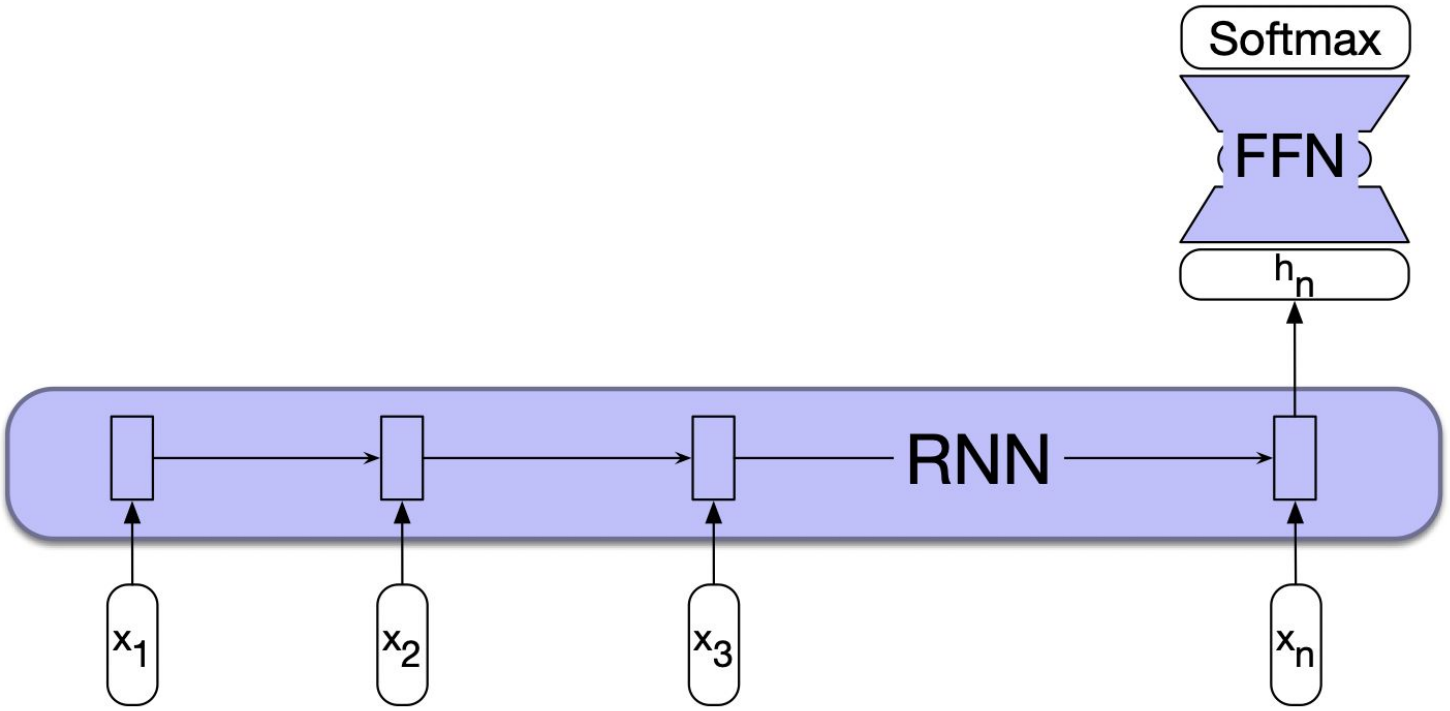
RNN for Sequence Labeling



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

RNN for Classification?

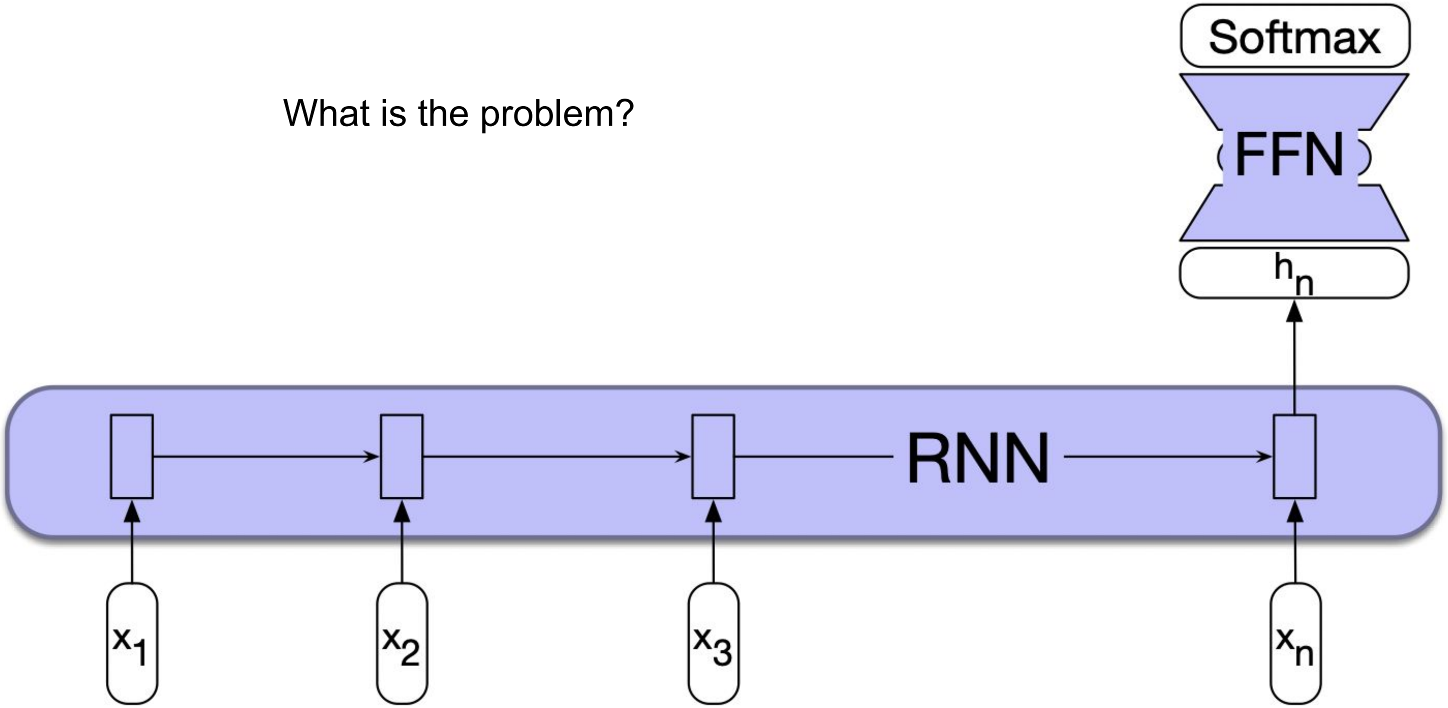
RNN for Classification



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

RNN for Classification

What is the problem?



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

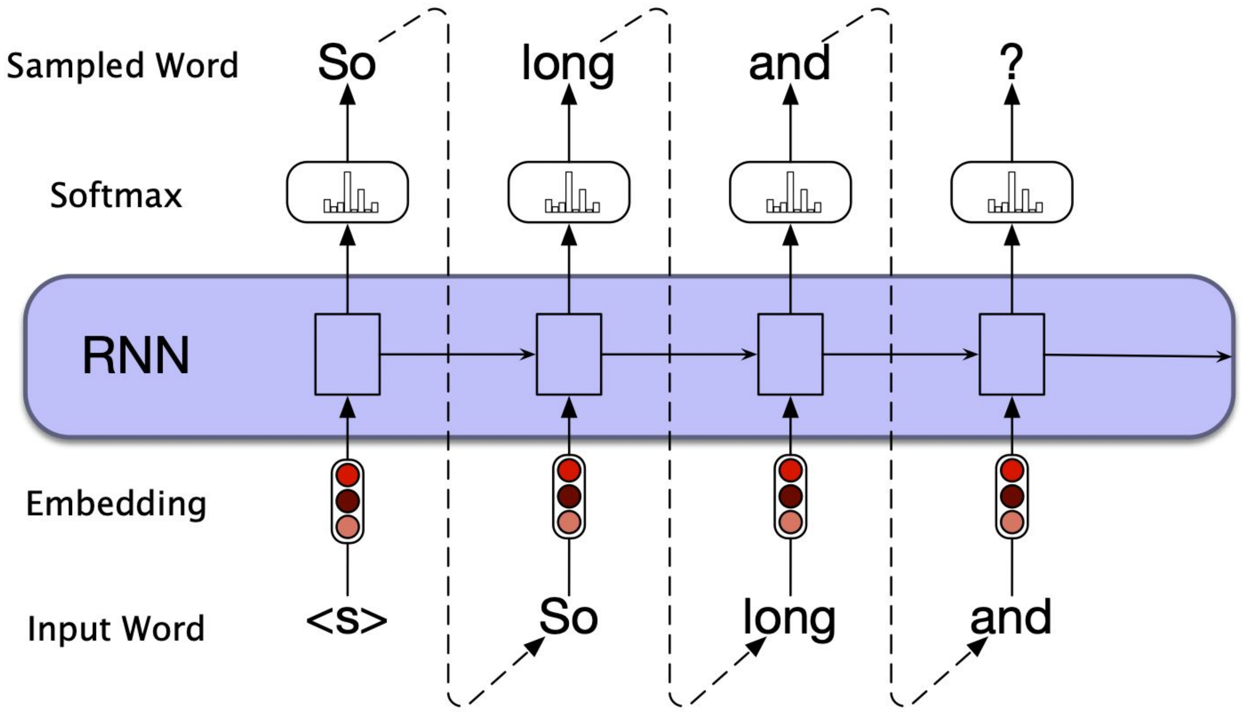
RNN for Text Generation

- “Autoregressive” text generation
  - What is the input?

RNN for Text Generation

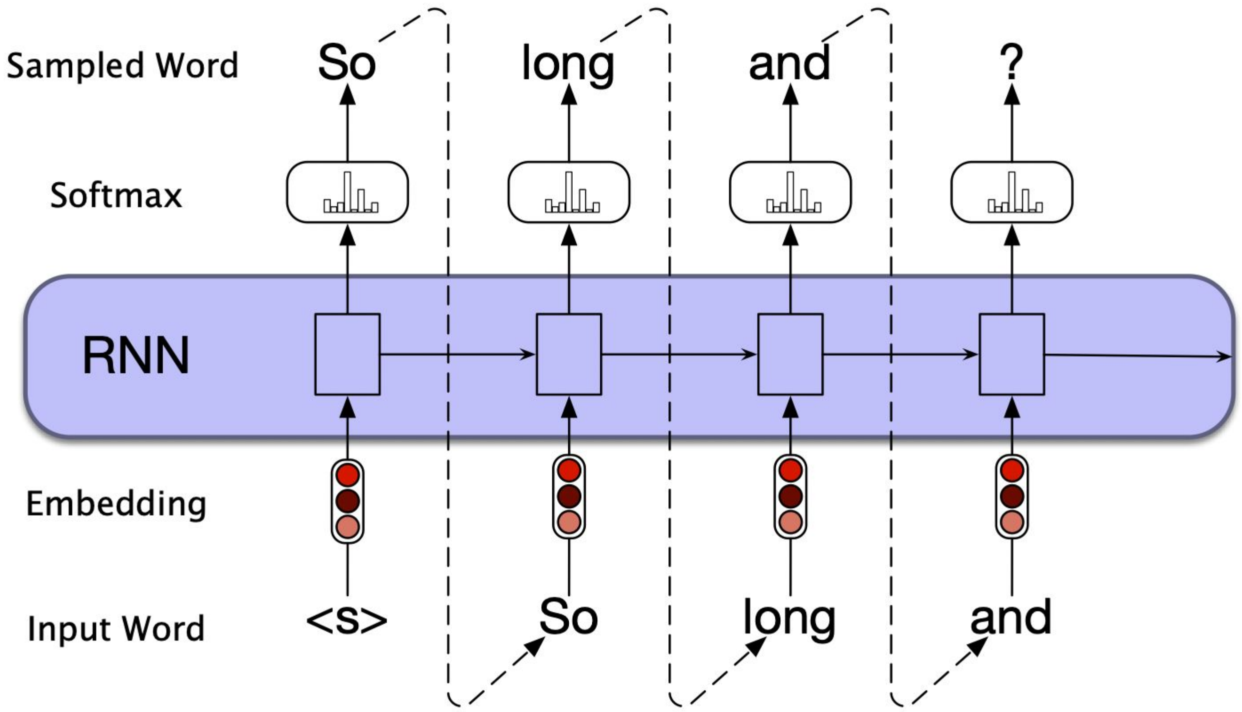
- “Autoregressive” text generation
  - Begin the sequence with a special token, e.g., <s>
  - Sample a word in the output from the softmax distribution given the start symbol
  - Use the word embedding of the generated word as the input for generating the next word
  - Continue until a max length or generating an end symbol (</s>)

RNN for Text Generation



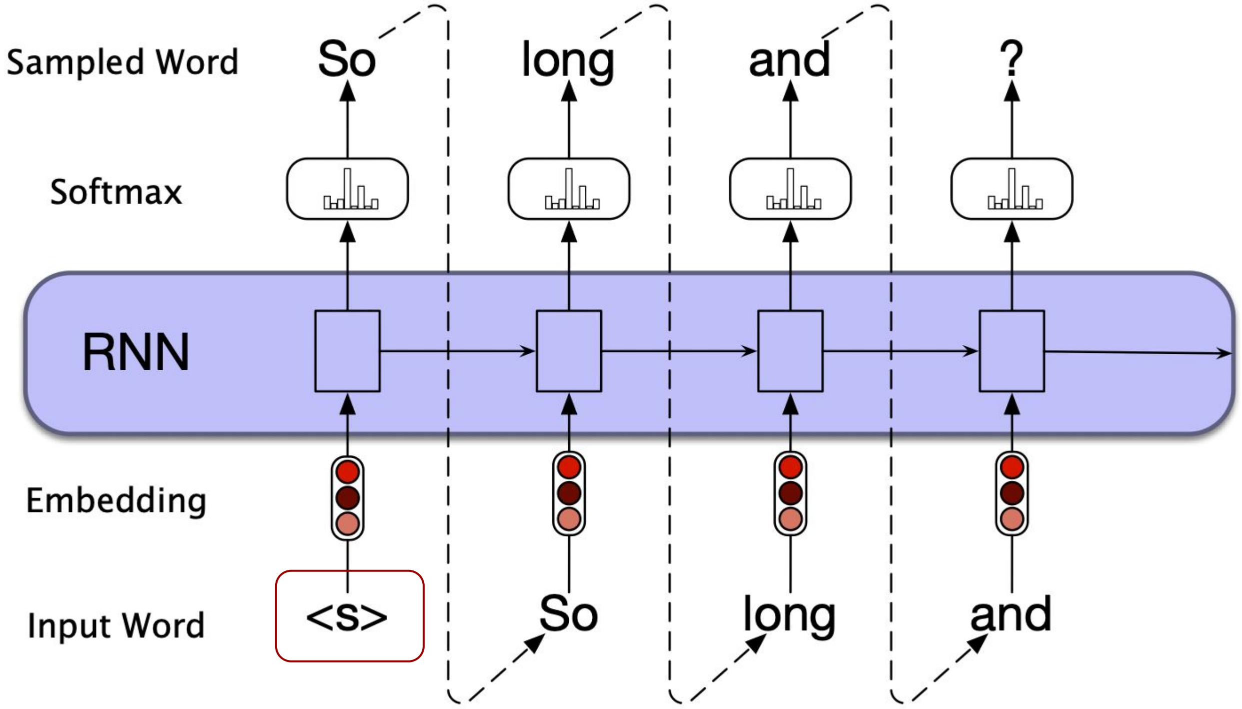
Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

How can we do better?



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

How can we do better?

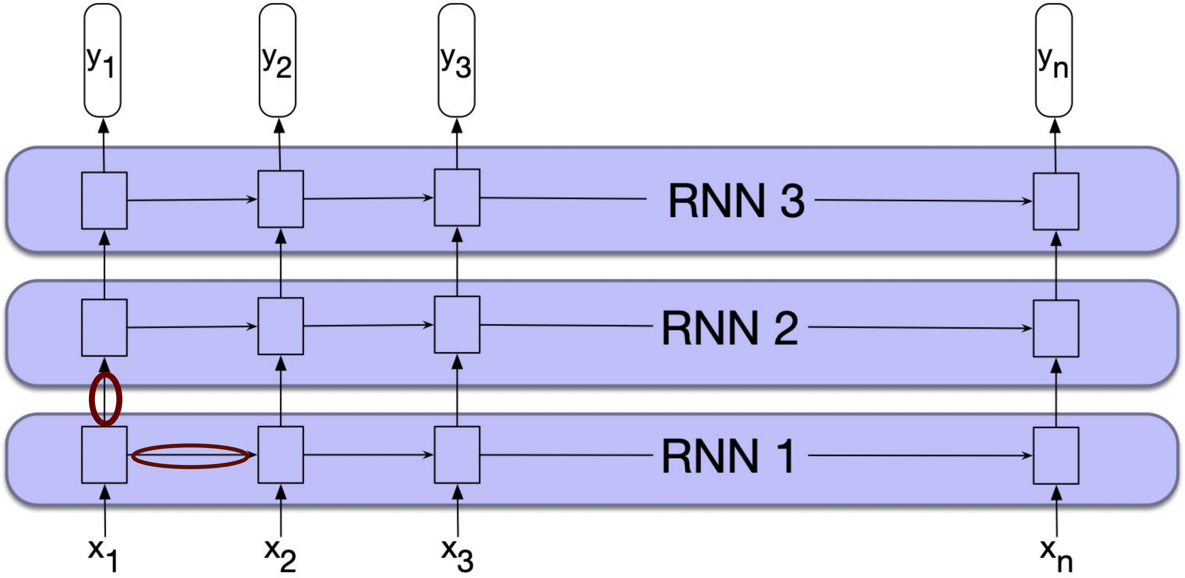


Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

RNN Variations

- Stacked RNN
- Bidirectional RNN

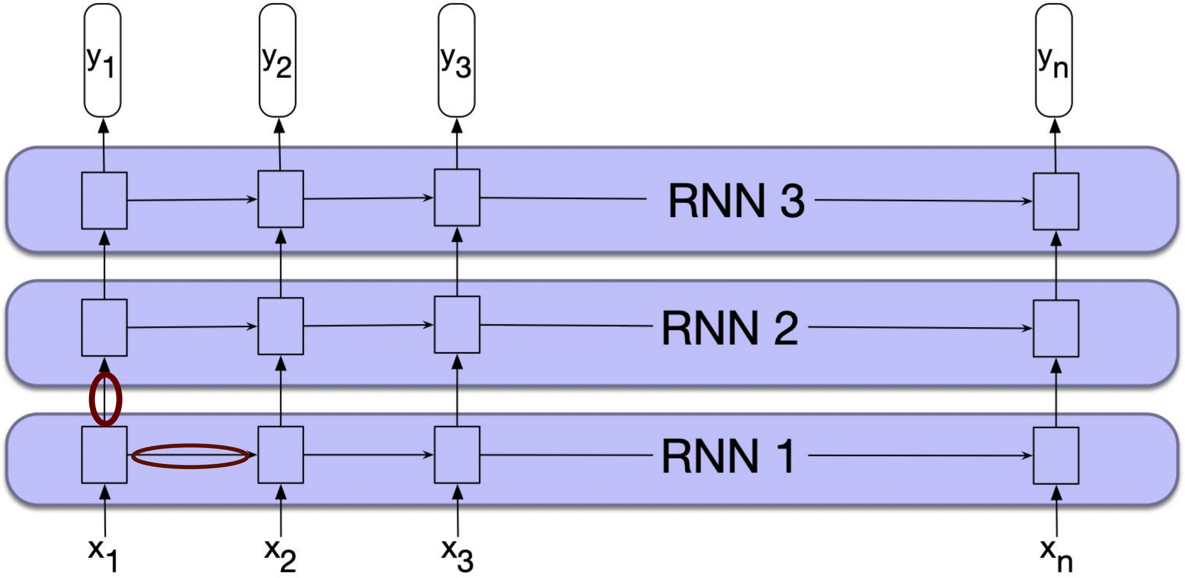
Stacked RNN



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Stacked RNN

Induces representations at differing levels of abstraction



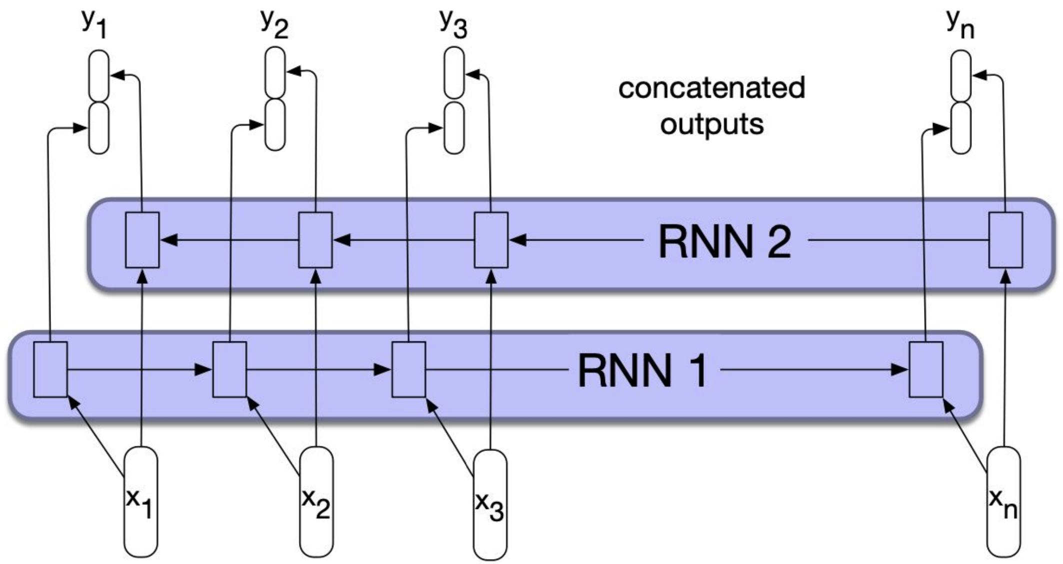
Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Bidirectional RNNs

$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

$$\mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

$$\mathbf{h}_t = [\mathbf{h}_t^f \, ; \, \mathbf{h}_t^b]$$



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

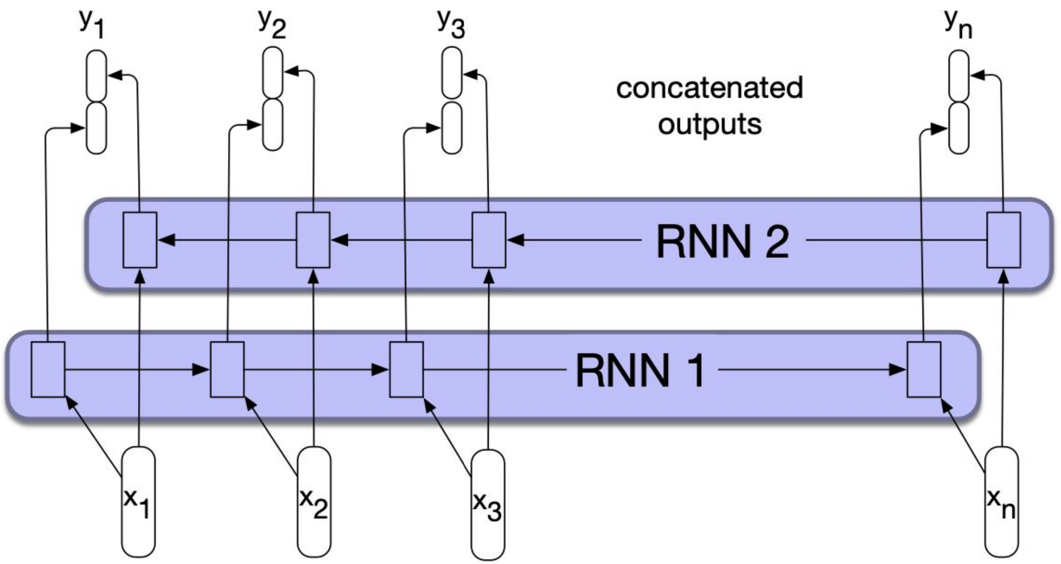
Bidirectional RNNs

Difference with Stacked RNN?

$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

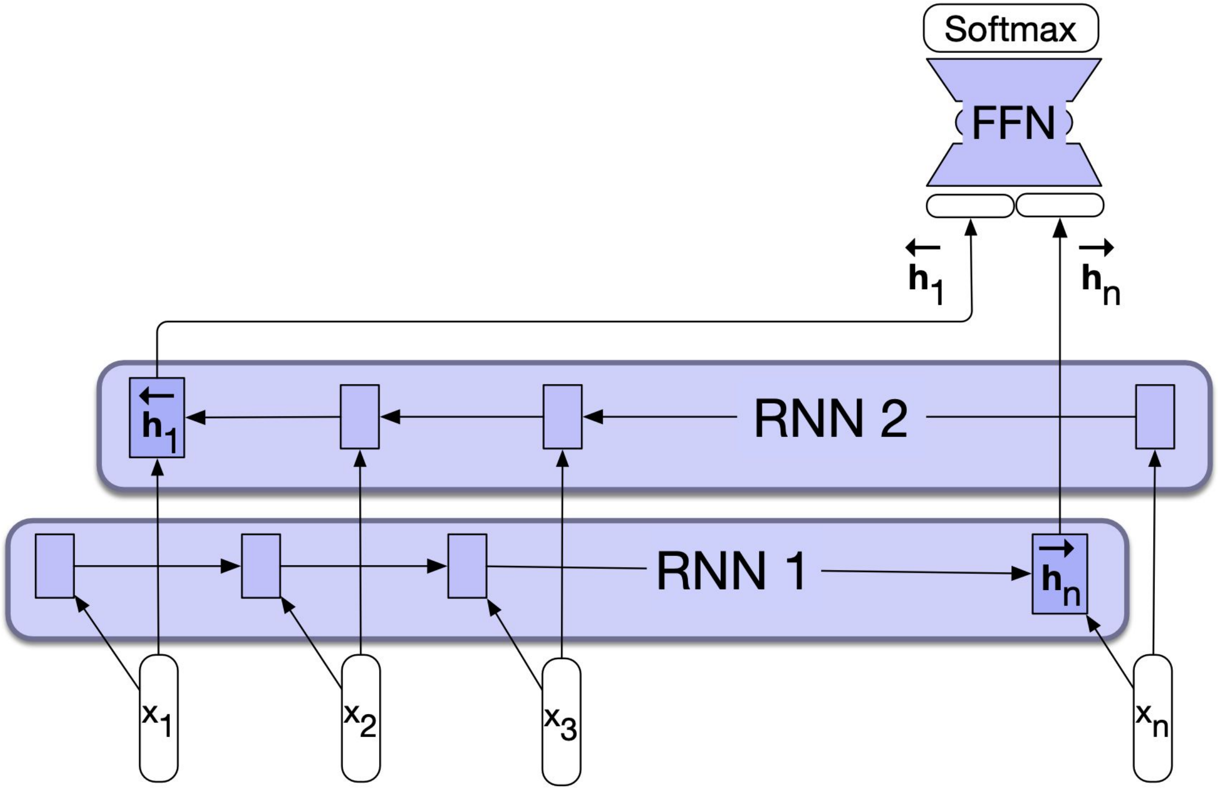
$$\mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

$$\mathbf{h}_t = [\mathbf{h}_t^f \,;\, \mathbf{h}_t^b]$$



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Bidirectional RNNs



Problem?

Problem

- Exploding or vanishing gradients for long sequences
  - Gradients can shrink exponentially when weights are small
  - Gradients grow exponentially when weights are big
  - Typical feed-forward neural nets often do not have many hidden layers
- Memory is Uncontrolled
  - Hidden state must store everything
  - No mechanism to decide what to keep or discard
- Single Vector Bottleneck
  - All past information compressed into a single vector

What Should a Better Recurrent Model Do?

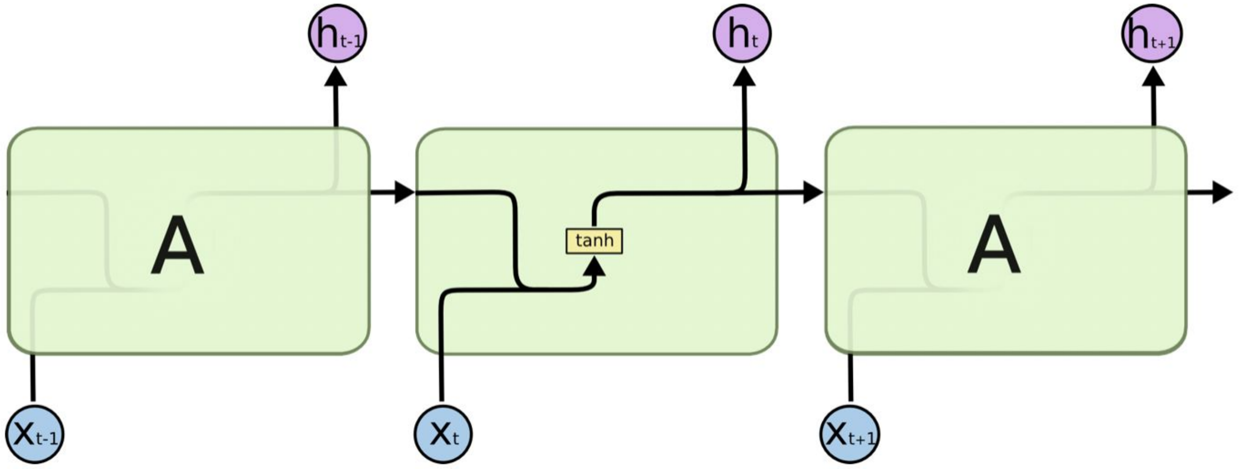
- ✔ Preserve important information over long distances
- ✔ Forget irrelevant information
- ✔ Allow gradients to flow across many time steps

LSTM (Hochreiter and Schmidhuber, 1997)

- Controlled Memory via Cell State
  - Introduces a dedicated memory vector
- Context management
  - Removing information no longer needed
  - Adding information likely to be needed later
- Gates to control the flow of information

RNN

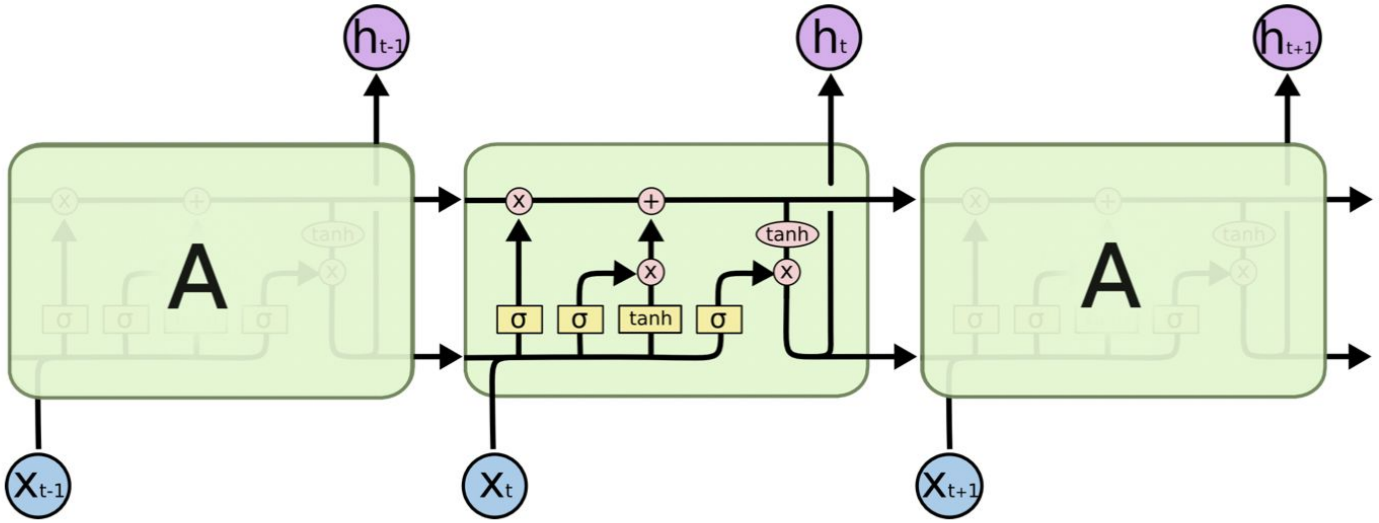
The repeating module in a standard RNN contains a single layer



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

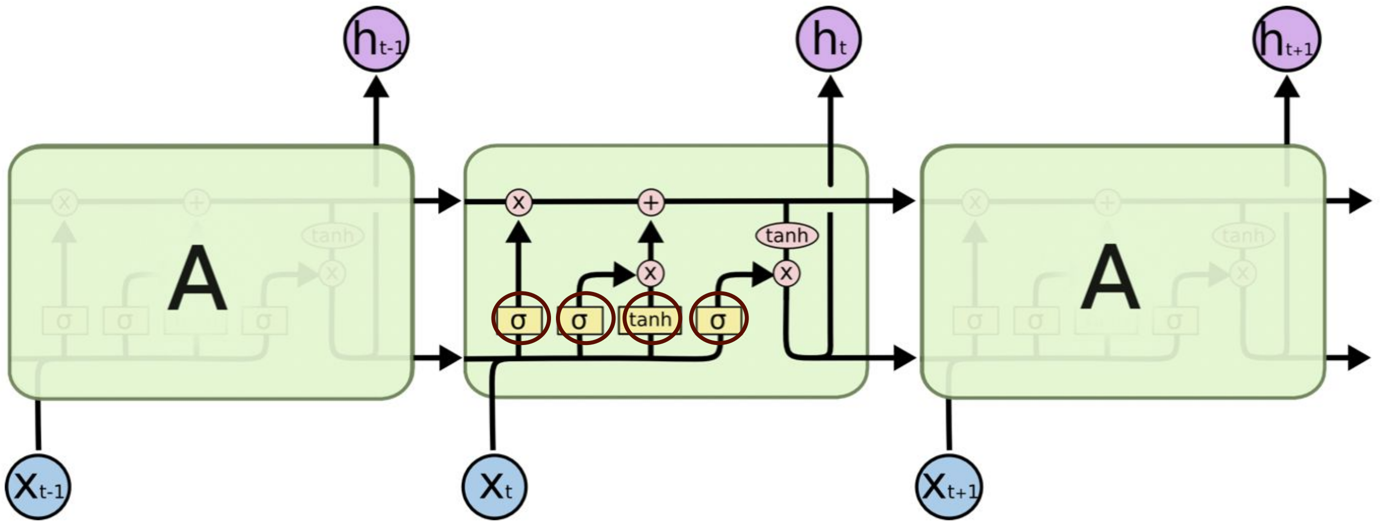
The repeating module in an LSTM contains four interacting layers



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

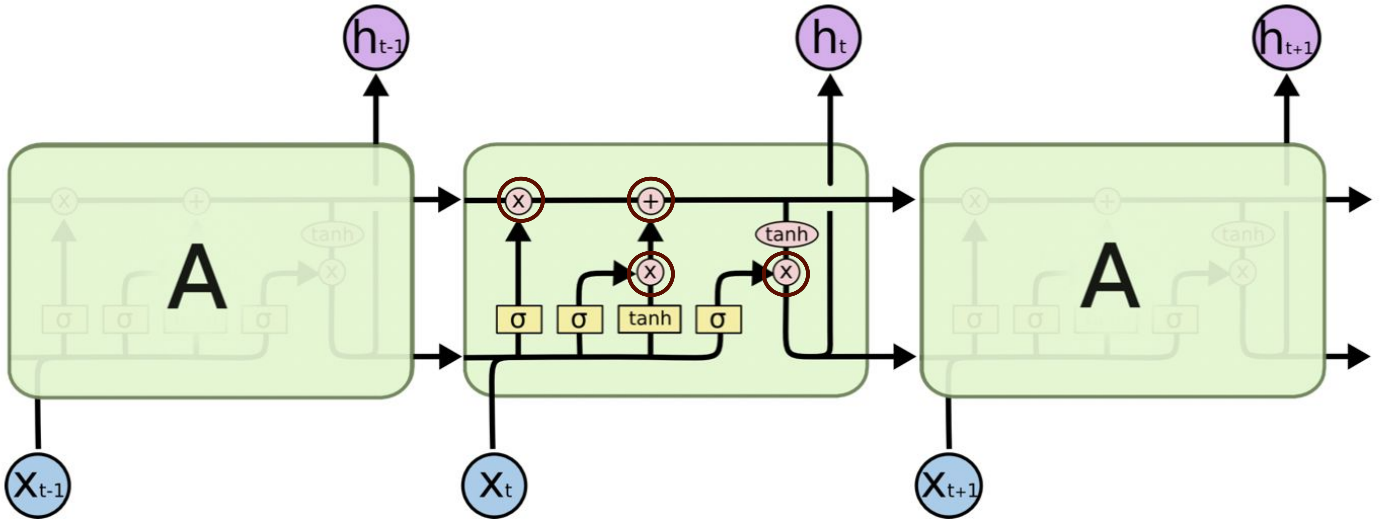
Neural network layers



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

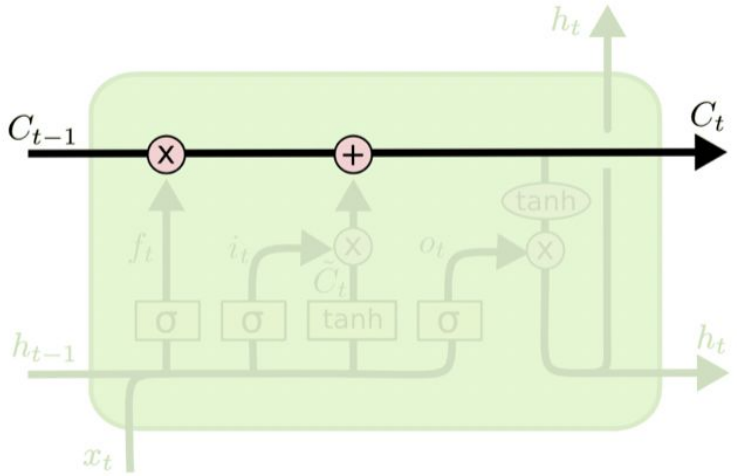
LSTM

Pointwise operations



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

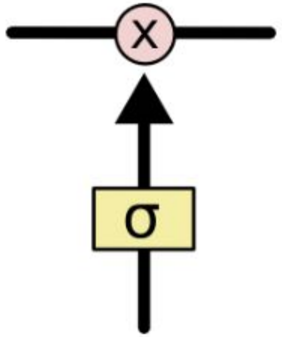
Cell state



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gates

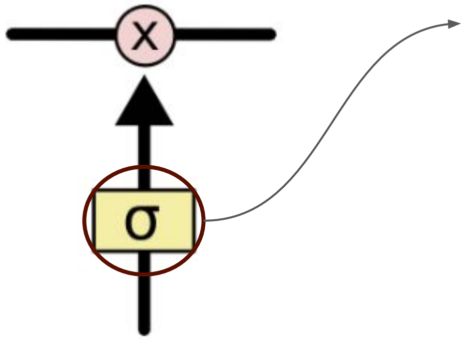
- Optionally let information through
- Sigmoid neural net layer + a pointwise multiplication operation



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gates

- Optionally let information through
- Sigmoid neural net layer + a pointwise multiplication operation

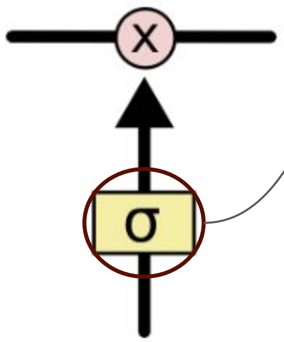


values between zero and one

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gates

- Optionally let information through
- Sigmoid neural net layer + a pointwise multiplication operation



values between zero and one

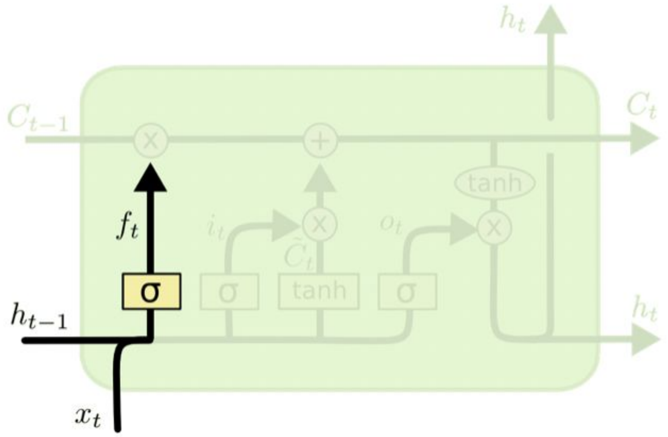
how much of each component should be let through

0: let nothing through
1: let everything through

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget Gate Layer

What information to throw away from the cell state

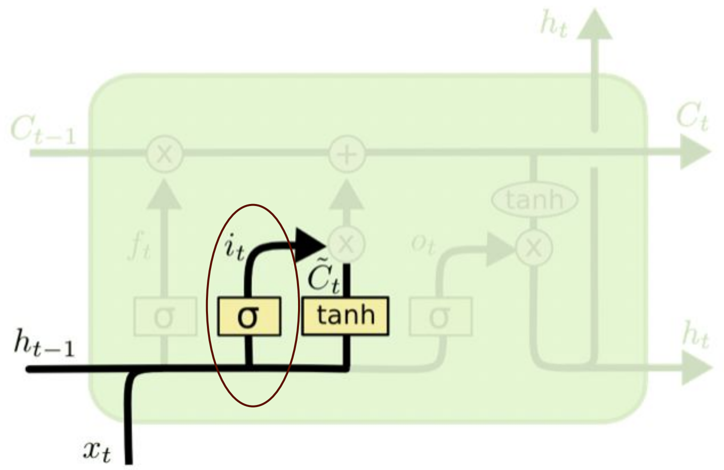


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Input Gate Layer

which values to update



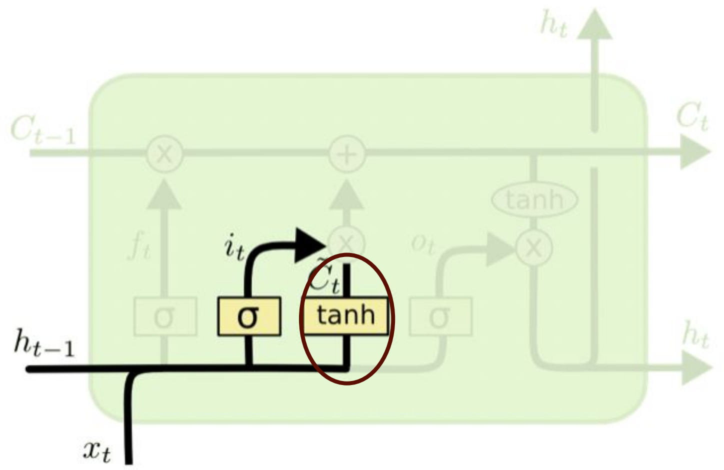
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Input Gate Layer

vector of new candidate values that could be added to the state (candidate memory update)

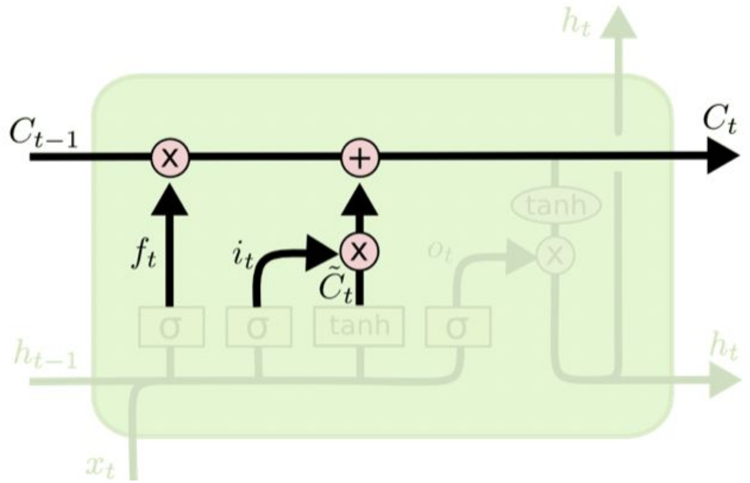


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Updating Old Cell State

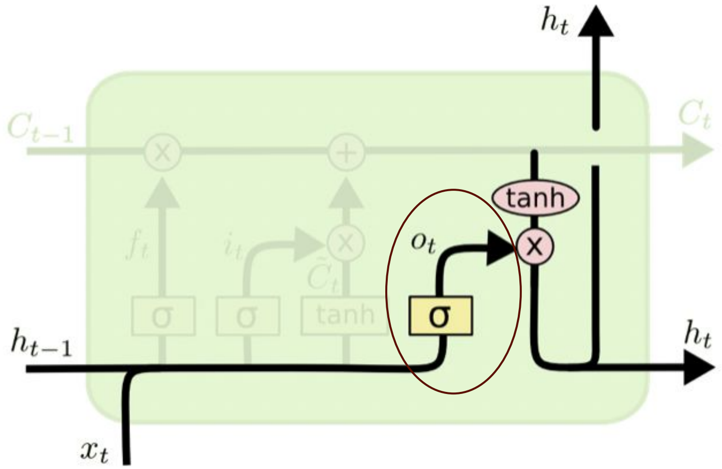


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Output

what parts of the cell to output



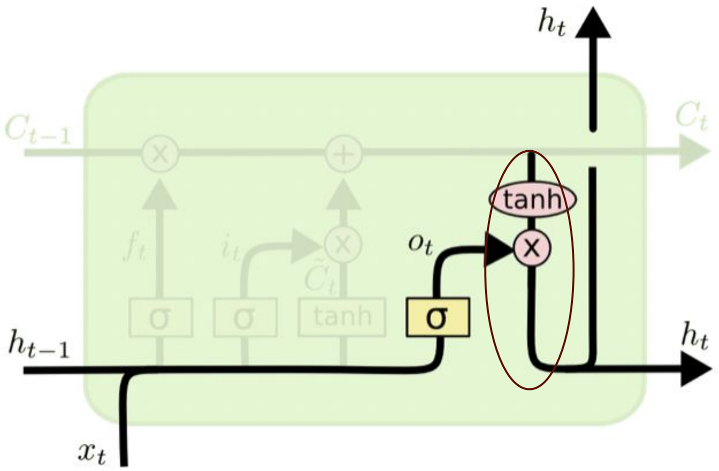
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Output

tanh (cell state): pushing values to [-1, 1]
multiply cell state by the output gate



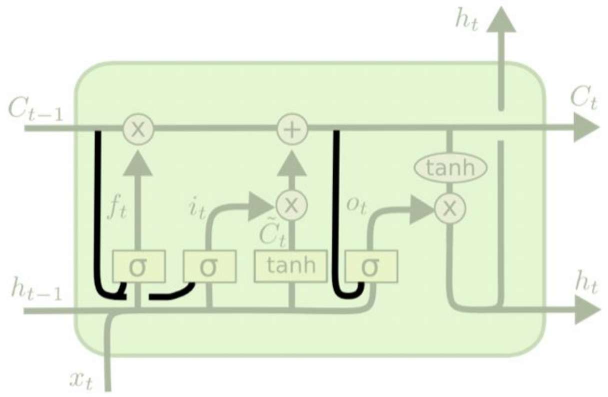
$$o_t = \sigma \left(W_o \left[h_{t-1}, x_t \right] + b_o \right)$$

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

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Variants

Gers & Schmidhuber (2000)
letting the gate layers look at the cell state

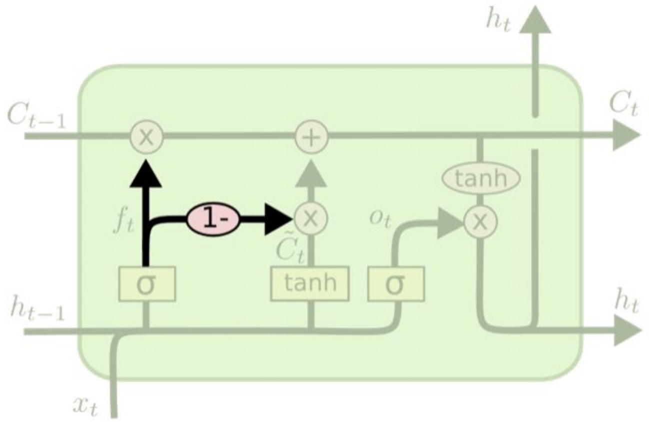


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Variants

coupled forget and input gates

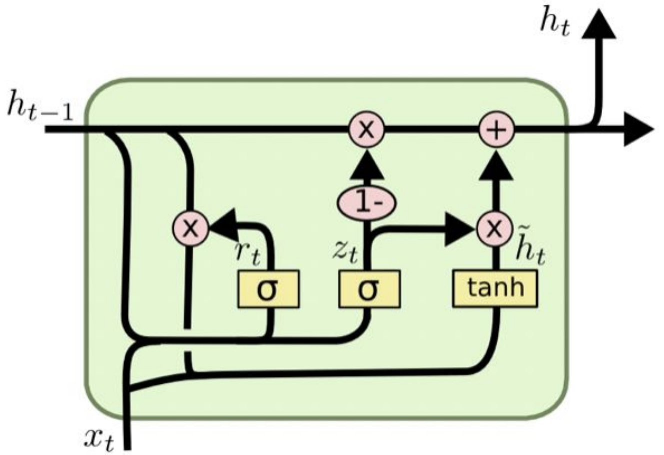


$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Variants

Gated Recurrent Unit (GRU) [Cho, et al., 2014]



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

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

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

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

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Why Gated RNNs Still Matter?

Even though Transformers dominate:

Gated RNNs introduced three foundational ideas:

✔ Additive state updates (stabilize gradients)

✔ Gated information flow (learn what to keep/forget)

✔ Learned dynamic memory

From LSTM to Residual Connections

LSTM cell update

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

- Preserve previous state and add controlled update

Transformer Residual Connection

$$x_{l+1} = x_l + \mathrm{Attention}(x_l)$$

- Preserve previous representation
- Add learned modification

From LSTM to Residual Connections

- Stabilize gradients
- Enable deep networks
- Prevent information collapse

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

$$x_{l+1} = x_l + \mathrm{Attention}(x_l)$$

From Gates to Gated MLP Blocks

In LSTM, gates decide:

- What to keep
- What to update

Some modern LLMs replace the FFN with a Gated FFN (SwiGLU activation)

- Llama, Qwen, Mistral, Gemma

$$\mathrm{FFN}(x) = W_2\,\mathrm{ReLU}(W_1 x)$$

$$(W_1 x) \odot \mathrm{Activation}(W_2 x)$$

From Memory Control to Mixture-of-Experts

LSTM Gates Route Information

- control memory flow

$$\text{Output} = \sum_k g_k(x) \cdot \text{Expert}_k(x)$$

Mixture-of-Experts (MoE)

- g_k are learned routing weights
- control compute flow

Recurrence Revisited: State-Space Models

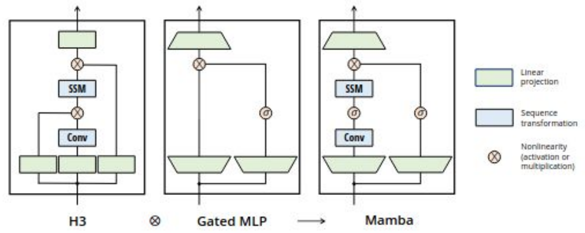
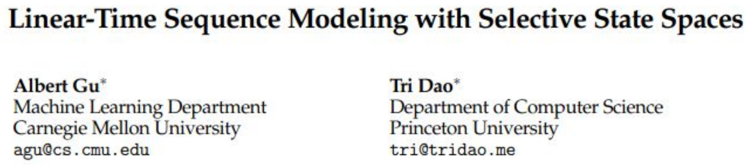


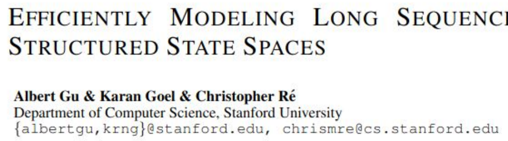
Figure 2: (**Architecture**.) Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogenously. Compared to the H3 block, Mamba replaces the first multiplicative gate with an activation function. Compared to the MLP block, Mamba adds an SSM to the main branch. For σ we use the SiLU / Swish activation (Hendrycks & Gimpel, 2016; Ramachandran et al., 2017).



Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu[*]
Machine Learning Department
Carnegie Mellon University
agu@cs.cmu.edu

Tri Dao[*]
Department of Computer Science
Princeton University
tri@tridao.me



EFFICIENTLY MODELING LONG SEQUENCES WITH STRUCTURED STATE SPACES

Albert Gu & Karan Goel & Christopher Ré
Department of Computer Science, Stanford University
{albertgu,krng}@stanford.edu, chrismre@cs.stanford.edu

Transformers dominate
But research is rediscovering structured recurrence (for efficiency)

The Structural Limitation of Recurrence

The Structural Limitation of Recurrence

All information must pass through a single evolving state

- Information from token 1 must pass through many steps
- Compression bottleneck
- Sequential dependency

The Structural Limitation of Recurrence

If we increase the hidden dimension of an RNN, do we eliminate the compression bottleneck?

Why or why not?

The Structural Limitation of Recurrence

All information must pass through a single evolving state

- Information from token 1 must pass through many steps
- Compression bottleneck
- Sequential dependency

Do we really need to compress the past into one vector?

Attention

Instead of:

State evolution

Let each token:

- Look at all others
- Select relevant information
- Compute a weighted combination

$$h_i = \sum_j \alpha_{ij} x_j$$

Attention

Consider a sequence of length n

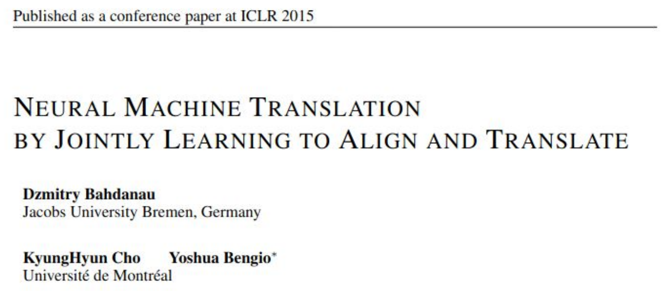
How many transformations must information from the first token undergo before it can influence the last token in:

An RNN?

A self-attention layer?

Attention

First Successful Use of Attention



Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

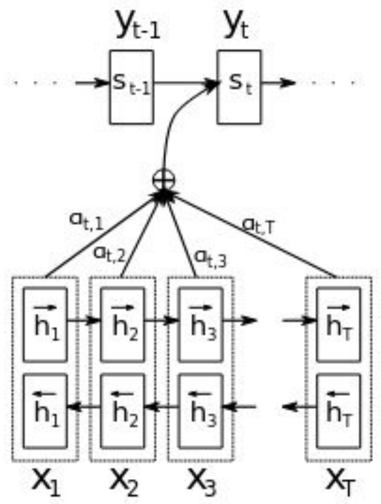
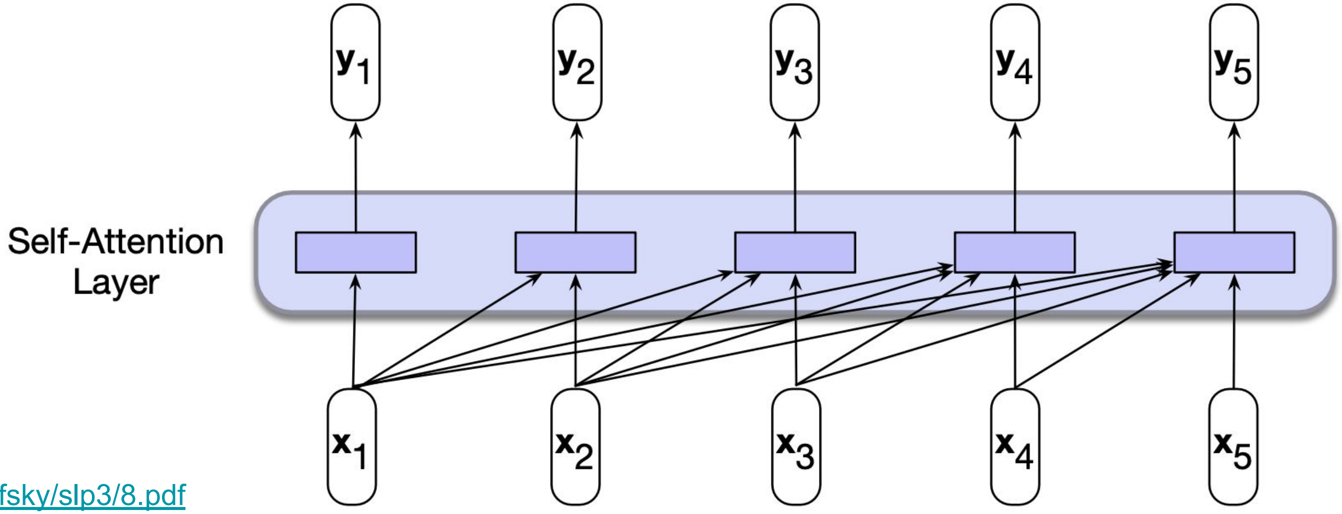

y_{t-1}
y_t
s_{t-1}
s_t
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
X_1 X_2 X_3 X_T

Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Self-Attention?

Self-attention

- Mapping input sequences to output sequences of the same length
- Has access to all of the inputs up to and including x_i for processing x_i
- Computations for each item is independent of all the other computations

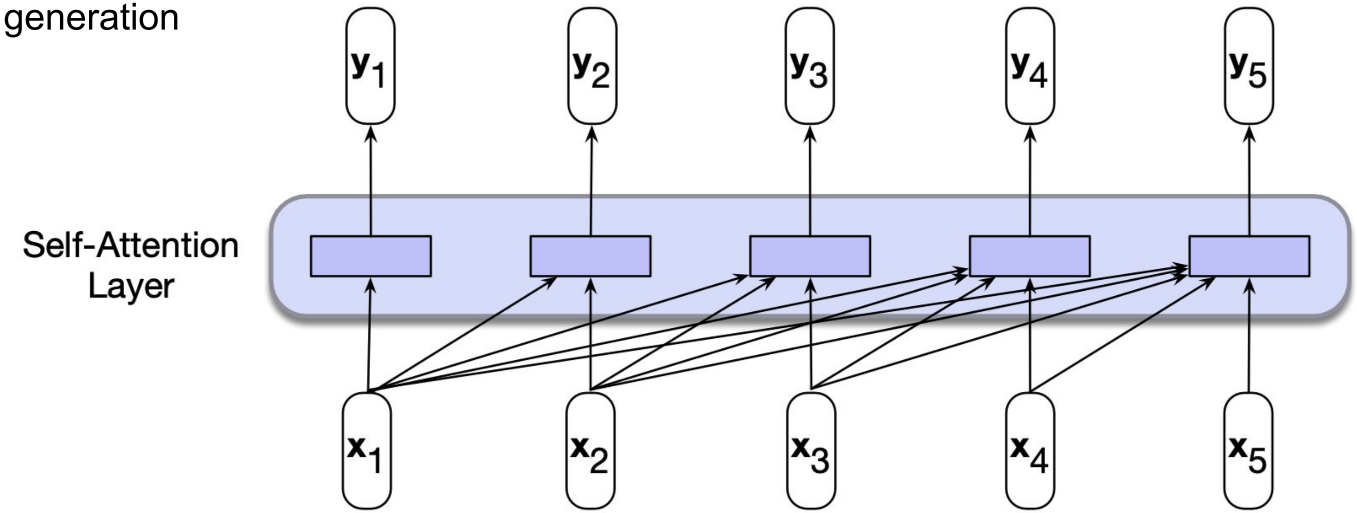


Source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Self-attention

- Mapping input sequences to output sequences of the same length
- Has access to all of the inputs up to and including x_i for processing x_i
- Computations for each item is independent of all the other computations

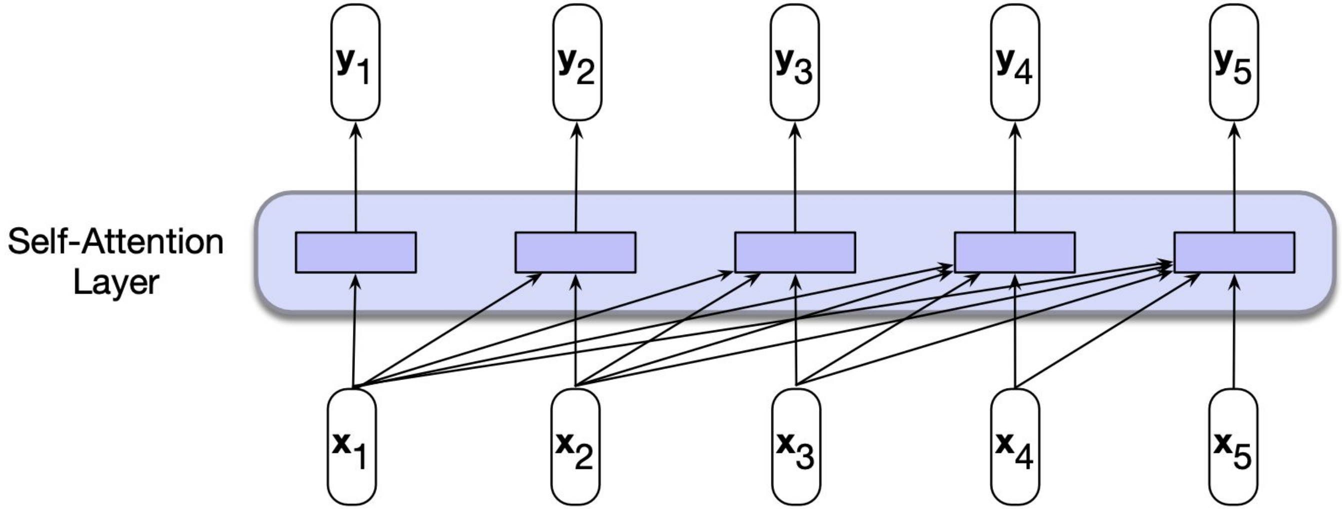
LMs and autoregressive generation



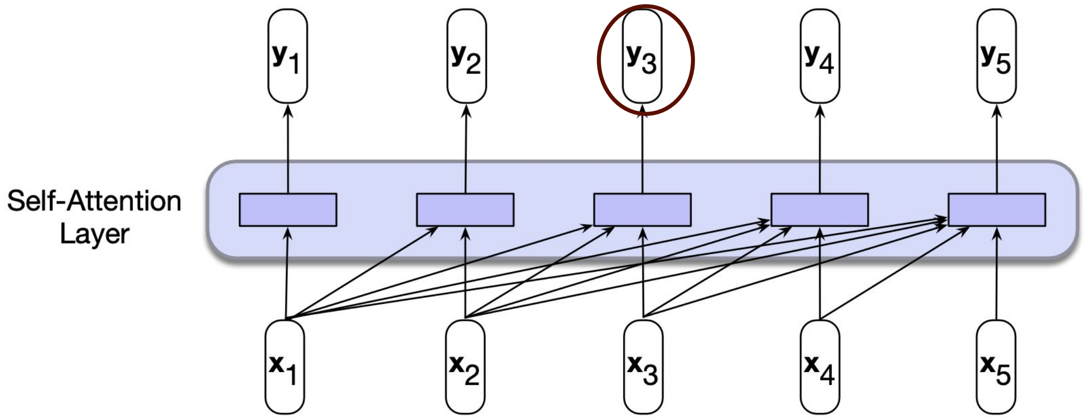
Self-attention

- Mapping input sequences to output sequences of the same length
- Has access to all of the inputs up to and including x_i for processing x_i
- Computations for each item is independent of all the other computations

Parallel computations

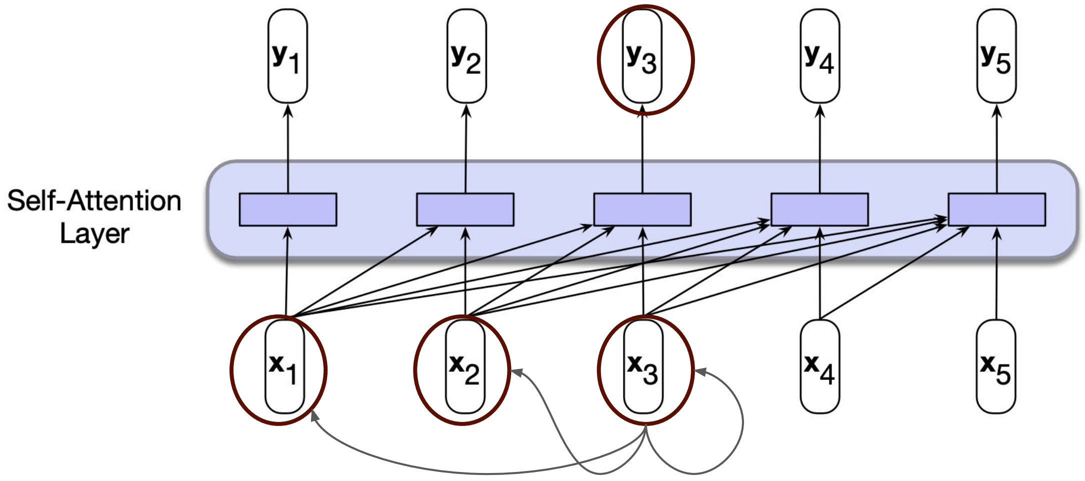


Self-attention



Self-attention

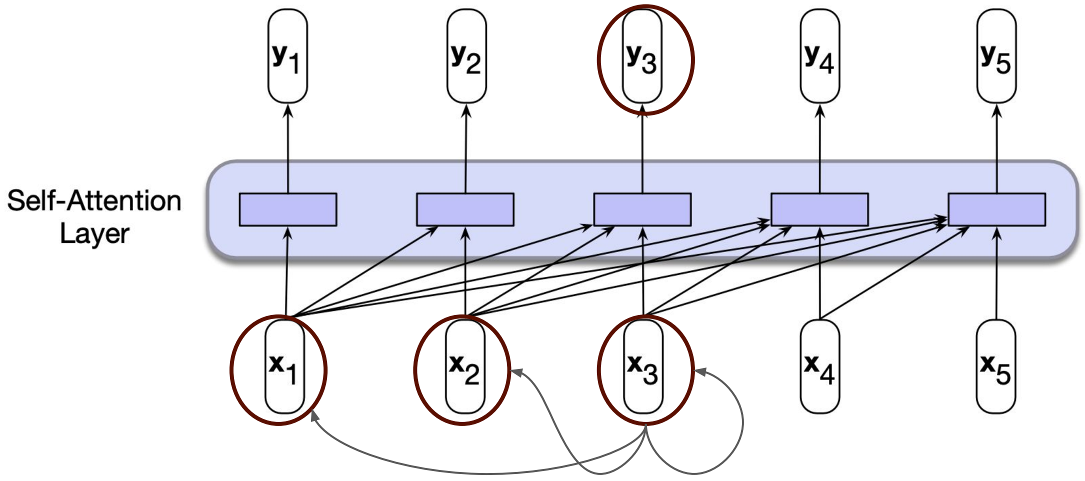
comparing an item of interest to a collection of other items to reveal their relevance in the given context



Self-attention

comparing an item of interest to a collection of other items to reveal their relevance in the given context

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$



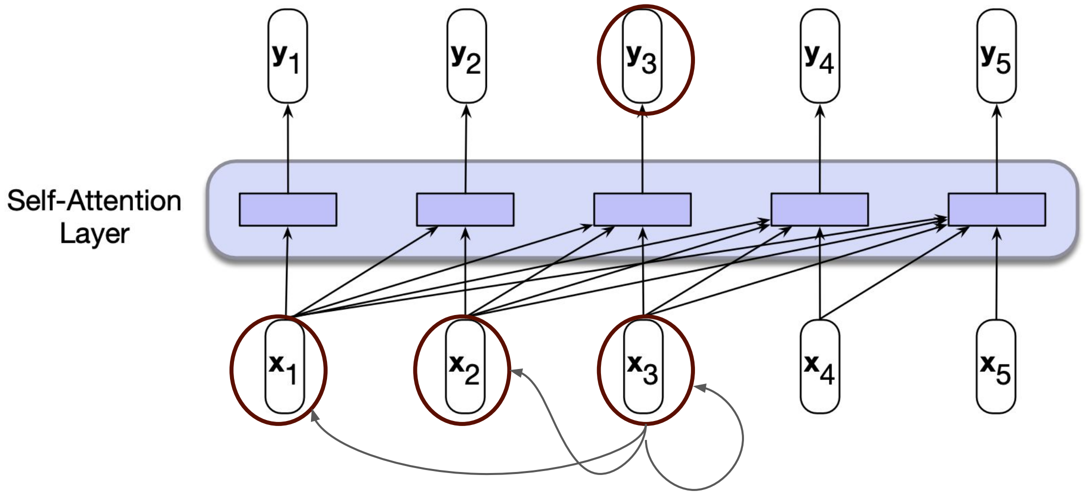
Self-attention

comparing an item of interest to a collection of other items to reveal their relevance in the given context

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

normalization to get probability distribution

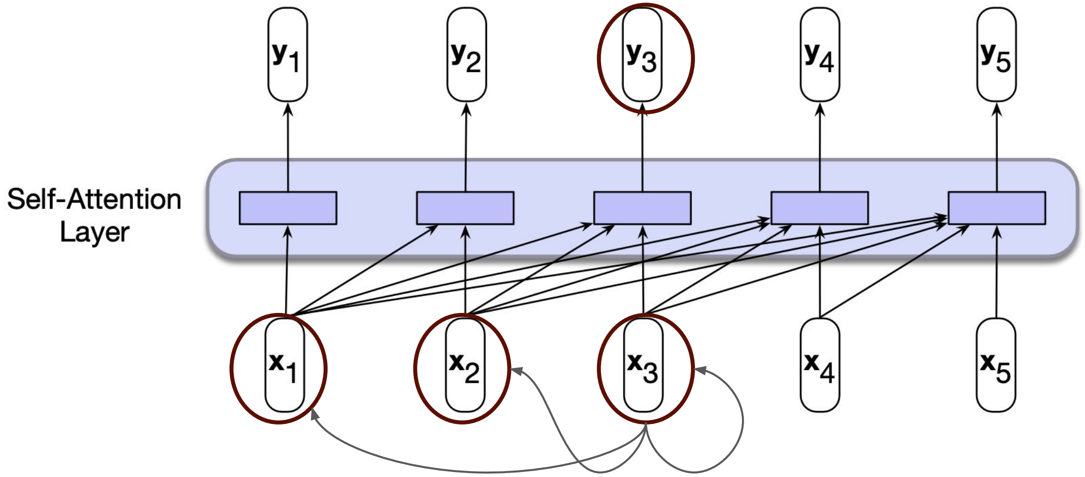
$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \;\; \forall j \leq i\end{aligned}$$



Self-attention

weighted sum using this distribution

$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij} \mathbf{x}_j$$



Question

Are RNNs and Self-Attention Aware of Position?

Consider a sentence:

“The cat chased the mouse.”

Would the model know if we permute the words?

“Mouse the chased cat the.”

Wrap-Up

Model	**Core Idea**	**Limitation**
N-gram	Fixed window counts	
FFNN LM	Distributed representations	
RNN	Dynamic state	
LSTM/GRU	Gated memory	
Attention	Dynamic context access	

Wrap-Up

Model	Core Idea	Limitation
N-gram	Fixed window counts	No generalization
FFNN LM	Distributed representations	Fixed context size
RNN	Dynamic state	Hard to train, compression
LSTM/GRU	Gated memory	Still sequential, compression
Attention	Dynamic context access	Quadratic cost

Questions?