Recurrent Neural Networks

CS114B Lab 11

Kenneth Lai

April 23, 2021

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?
- Generative approach: Hidden Markov Models

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?
- Generative approach: Hidden Markov Models
- Discriminative approaches:
  - Conditional random fields
  - Structured perceptrons

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?
- Generative approach: Hidden Markov Models
- Discriminative approaches:
  - Conditional random fields
  - Structured perceptrons
  - Neural networks

Sequence Labeling

- Discriminative approaches:

Sequence Labeling

- Discriminative approaches:
  - At each time step, use local features to compute local scores, and use the Viterbi algorithm to make predictions for the whole sentence

Sequence Labeling

- Discriminative approaches:
  - At each time step, use local features to compute local scores, and use the Viterbi algorithm to make predictions for the whole sentence
    - Conditional random fields
    - Structured perceptrons

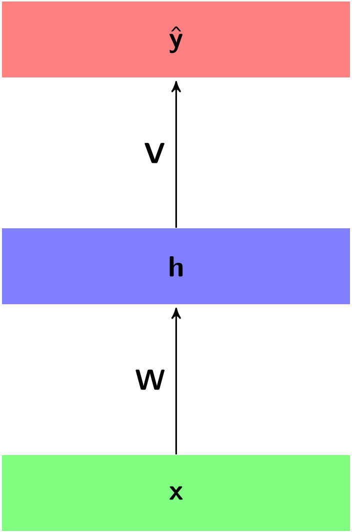
Sequence Labeling

- Discriminative approaches:
  - At each time step, use local features to compute local scores, and use the Viterbi algorithm to make predictions for the whole sentence
    - Conditional random fields
    - Structured perceptrons
  - Use features from other time steps, but make independent predictions at each time step

Sequence Labeling

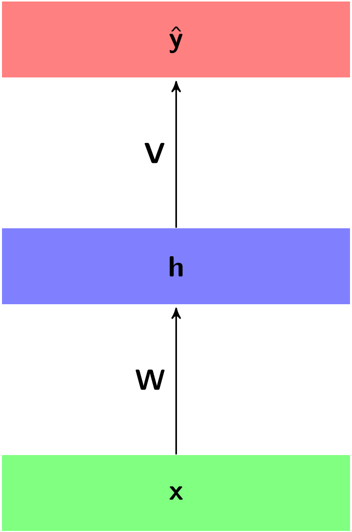
- Discriminative approaches:
  - At each time step, use local features to compute local scores, and use the Viterbi algorithm to make predictions for the whole sentence
    - Conditional random fields
    - Structured perceptrons
  - Use features from other time steps, but make independent predictions at each time step
    - Neural networks

Feedforward Neural Networks



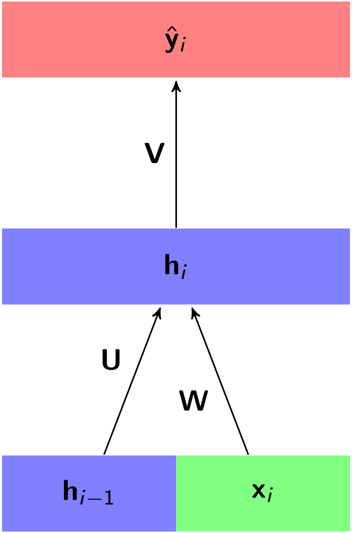
- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{h} = g(\mathbf{x} \cdot \mathbf{W})$
- $\hat{\mathbf{y}} = g(\mathbf{h} \cdot \mathbf{V})$

Feedforward Neural Networks

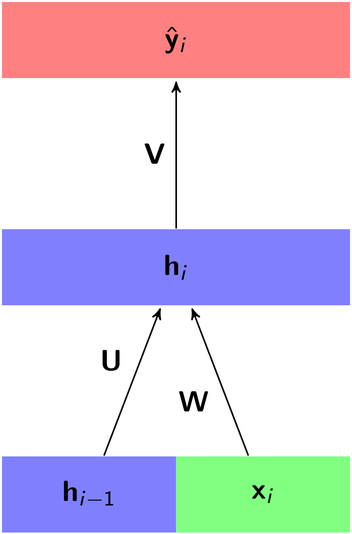


- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{h} = g(\mathbf{x} \cdot \mathbf{W})$
- $\hat{\mathbf{y}} = g(\mathbf{h} \cdot \mathbf{V})$
  - We will assume that the dummy feature 1 is part of $\mathbf{x}$ and $\mathbf{h}$, and that the bias term is part of $\mathbf{W}$ and $\mathbf{V}$, etc.

Neural Networks for Sequence Labeling

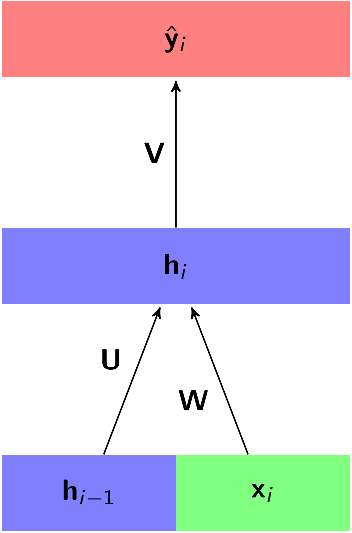


Neural Networks for Sequence Labeling



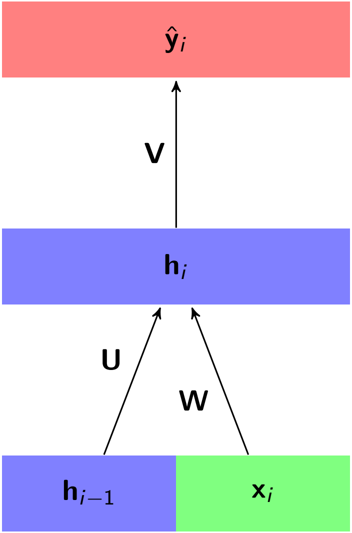
- At each time i, the input to the neural network consists of:

Neural Networks for Sequence Labeling



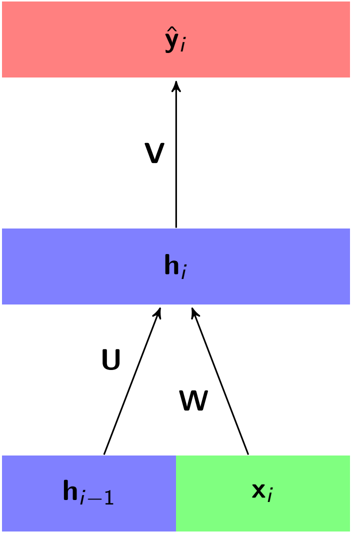
- At each time i, the input to the neural network consists of:
    - Current word (or other input) vector $\mathbf{x}_i$

Neural Networks for Sequence Labeling



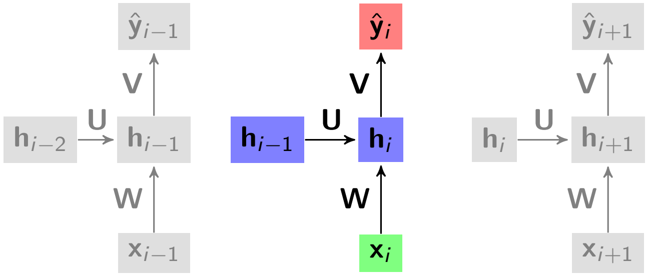
- At each time i, the input to the neural network consists of:
  - Current word (or other input) vector $\mathbf{x}_i$
  - History/(past) context vector $\mathbf{h}_{i-1}$

Neural Networks for Sequence Labeling

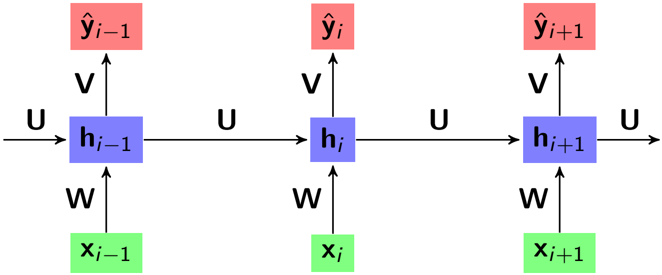


- At each time i, the input to the neural network consists of:
  - Current word (or other input) vector $\mathbf{x}_i$
  - History/(past) context vector $\mathbf{h}_{i-1}$
- $\mathbf{h}_i = g(\mathbf{x}_i \cdot \mathbf{W} + \mathbf{h}_{i-1} \cdot \mathbf{U})$

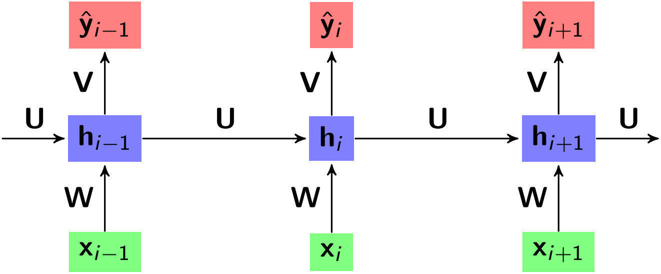
Neural Networks for Sequence Labeling



Neural Networks for Sequence Labeling

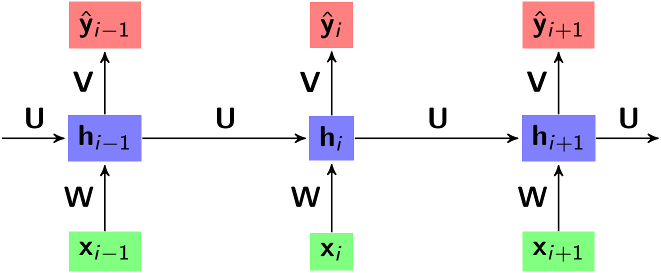


Neural Networks for Sequence Labeling



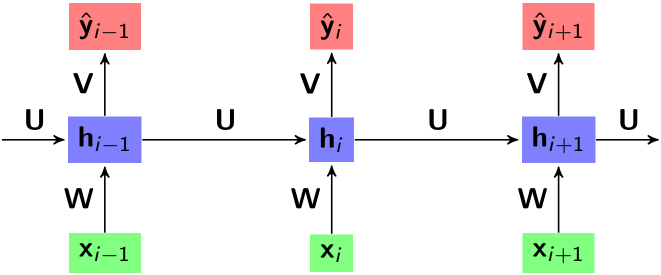
- The output of the hidden state at one time step is the history/past context input for the next time step!

Neural Networks for Sequence Labeling



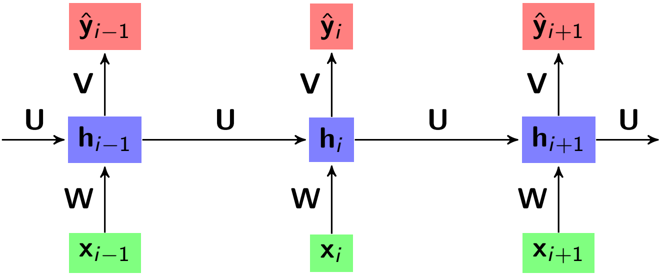
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?

Neural Networks for Sequence Labeling



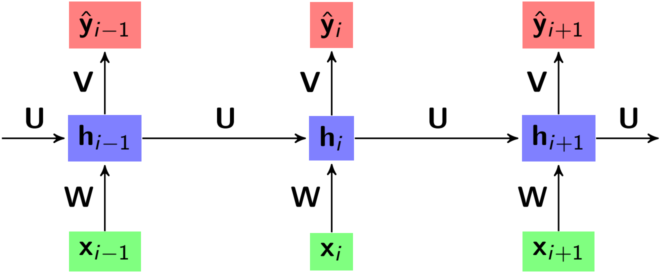
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - Previous word $\mathbf{x}_{i-1}$
  - Previous context $\mathbf{h}_{i-2}$

Neural Networks for Sequence Labeling



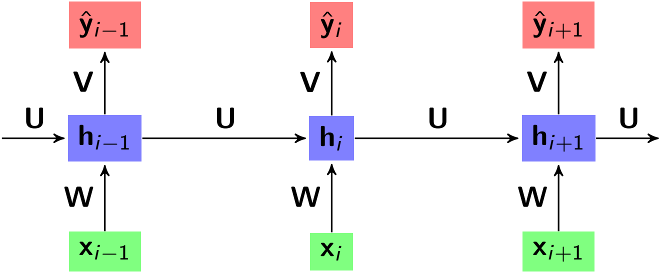
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - Previous word $\mathbf{x}_{i-1}$
  - Previous context $\mathbf{h}_{i-2}$
    - Previous previous word $\mathbf{x}_{i-2}$
    - Previous previous context $\mathbf{h}_{i-3}$

Neural Networks for Sequence Labeling



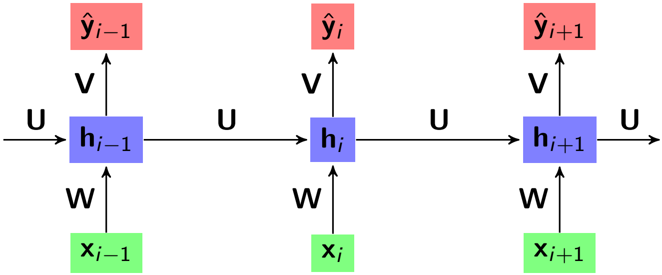
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - All previous words

Neural Networks for Sequence Labeling



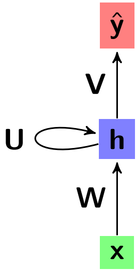
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - All previous words
  - What about previous parts of speech (as in HMMs, CRFs, structured perceptrons)?

Neural Networks for Sequence Labeling

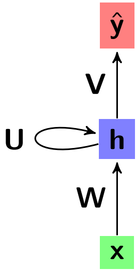


- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - All previous words
  - What about previous parts of speech (as in HMMs, CRFs, structured perceptrons)?
    - To learn more, take StatNLP in the fall!

Recurrent Neural Networks

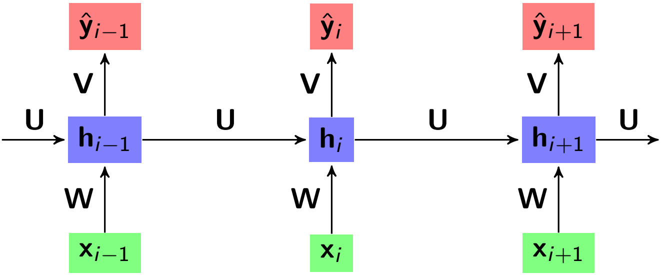


Recurrent Neural Networks

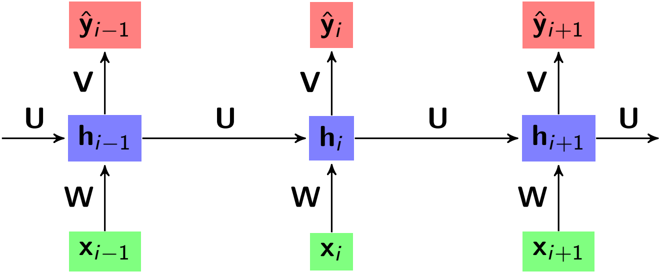


- Neural networks in which the output of a layer in one time step is input to a layer in the next time step

RNN Language Models

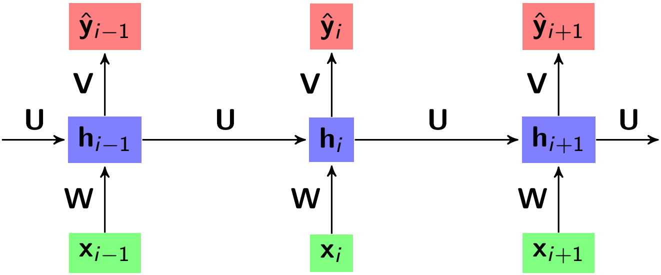


RNN Language Models



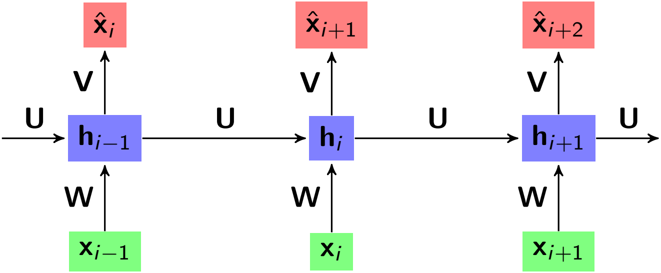
- Sequence labeling: predict current tag given current word, history

RNN Language Models



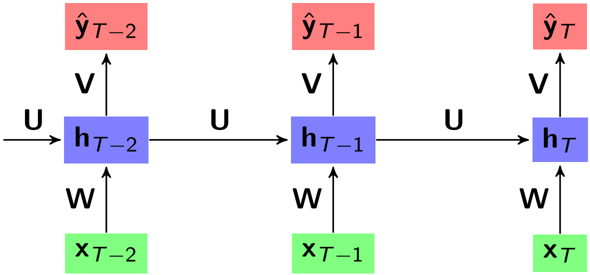
- Sequence labeling: predict current tag given current word, history
- Language modeling: predict next word given current word, history

RNN Language Models

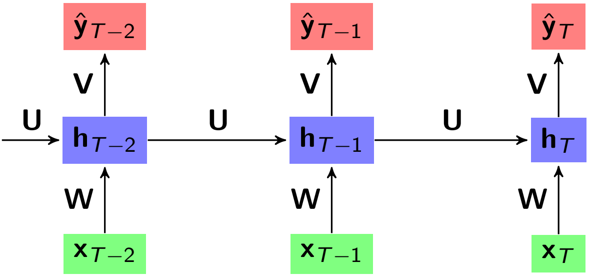


- Sequence labeling: predict current tag given current word, context
- Language modeling: predict next word given current word, context

RNNs for Text Classification

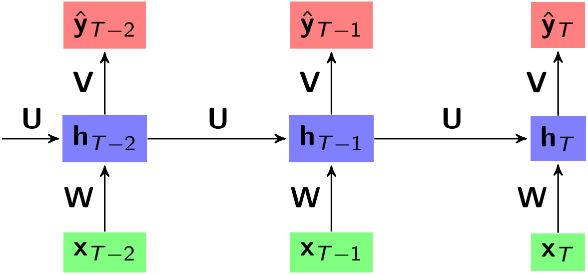


RNNs for Text Classification



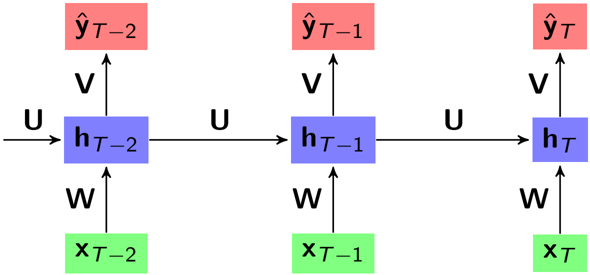
- What context information is embedded in $\mathbf{h}_T$?

RNNs for Text Classification



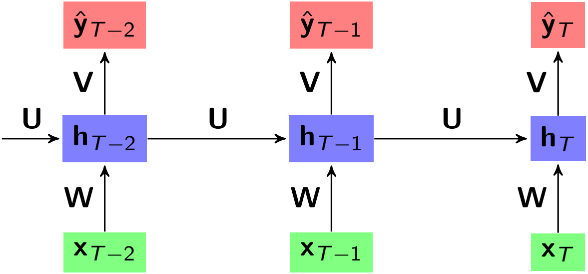
- What context information is embedded in $\mathbf{h}_T$?
  - Current word $\mathbf{x}_T$
  - Context $\mathbf{h}_{T-1}$

RNNs for Text Classification



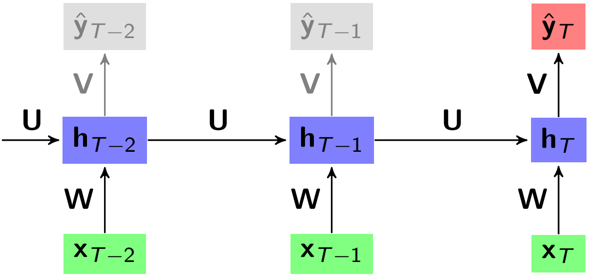
- What context information is embedded in $\mathbf{h}_T$?
  - All words (i.e. the whole text)

RNNs for Text Classification



- What context information is embedded in $\mathbf{h}_T$?
  - All words (i.e. the whole text)
- Use $\mathbf{h}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document

RNNs for Text Classification

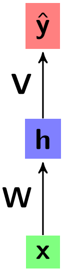


- What context information is embedded in $\mathbf{h}_T$?
  - All words (i.e. the whole text)
- Use $\mathbf{h}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document
  - Ignore other outputs

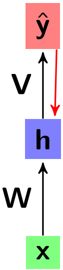
Backpropagation

- For each matrix of weights $\mathbf{W}$, starting from the output and working backwards:
  - Compute gradient $(\nabla L)^{[\mathbf{W}]}$
- For each matrix of weights $\mathbf{W}$:
  - Move in direction of negative gradient

Backpropagation

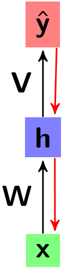


Backpropagation



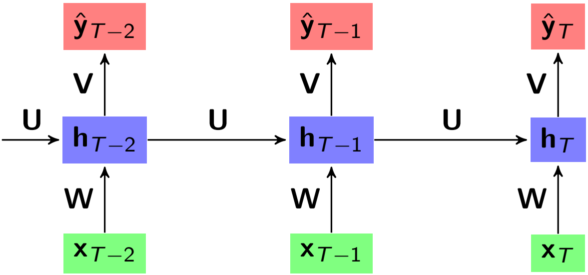
- Compute gradient $(\nabla L)^{[\mathbf{V}]}$

Backpropagation

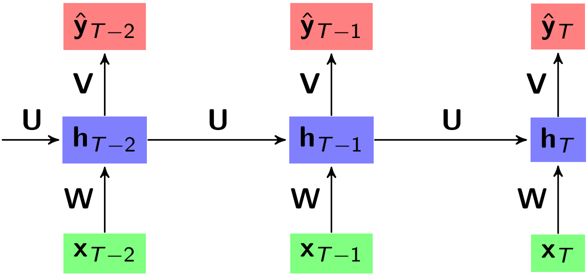


- Compute gradient $(\nabla L)^{[\mathbf{V}]}$
- Use $(\nabla L)^{[\mathbf{V}]}$ to compute gradient $(\nabla L)^{[\mathbf{W}]}$

Backpropagation Through Time

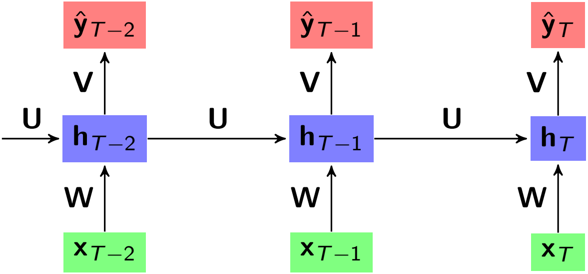


Backpropagation Through Time



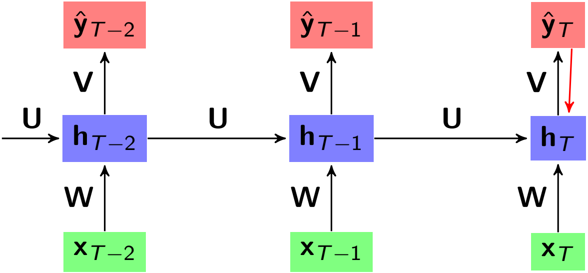
- Start at the end of the text and work backwards

Backpropagation Through Time



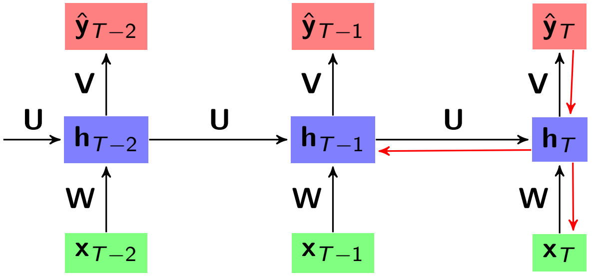
- Start at the end of the text and work backwards
  - Let $(\nabla L)_{i,j}^{[\mathbf{W}]}$ denote the part of the gradient for weight matrix $\mathbf{W}$ at time i that comes from the output at time j

Backpropagation Through Time



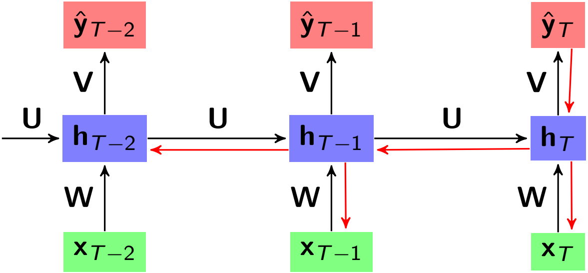
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T,T}$

Backpropagation Through Time



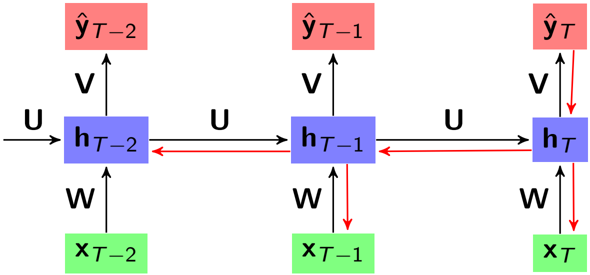
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T,T}$

Backpropagation Through Time



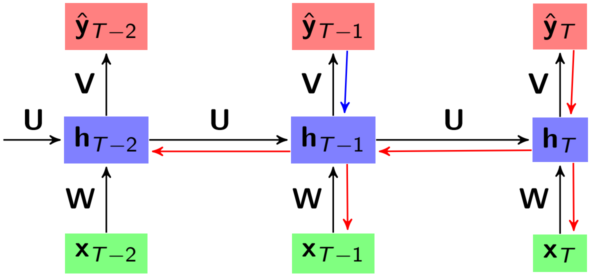
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{U}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T-1,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T-1,T}$

Backpropagation Through Time



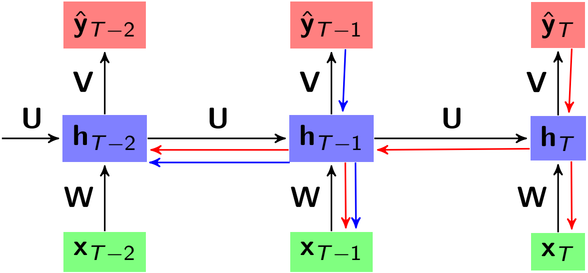
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{U}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T-1,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T-1,T}$
  - etc.

Backpropagation Through Time



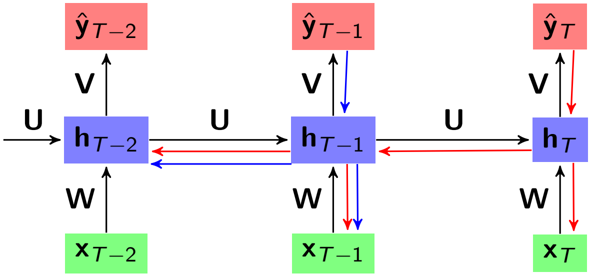
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$

Backpropagation Through Time



- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T-1,T-1}$ and $(\nabla L)^{[\mathbf{U}]}_{T-1,T-1}$

Backpropagation Through Time



- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T-1,T-1}$ and $(\nabla L)^{[\mathbf{U}]}_{T-1,T-1}$
  - etc.

Backpropagation Through Time

- The overall gradient for a weight matrix $\mathbf{W}$ is the sum of the gradients at each time i from each output $\hat{\mathbf{y}}_j$

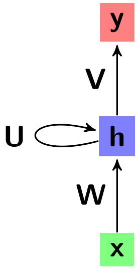
Backpropagation Through Time

- The overall gradient for a weight matrix **W** is the sum of the gradients at each time i from each output $\hat{\mathbf{y}}_j$
  - $(\nabla L)^{[\mathbf{W}]} = \sum_{j=1}^{T} \sum_{i=1}^{j} (\nabla L)_{i,j}^{[\mathbf{W}]}$

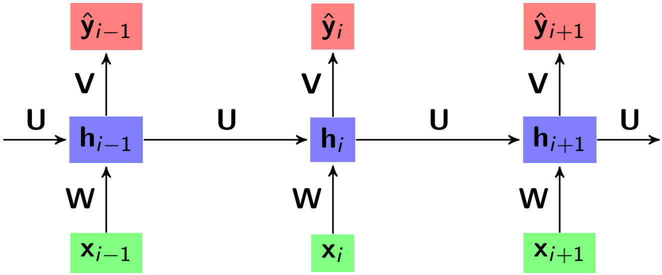
Backpropagation Through Time

- The overall gradient for a weight matrix **W** is the sum of the gradients at each time i from each output $\hat{\mathbf{y}}_j$
  - $(\nabla L)^{[\mathbf{W}]} = \sum_{j=1}^{T} \sum_{i=1}^{j} (\nabla L)_{i,j}^{[\mathbf{W}]}$
- Then move in direction of negative gradient

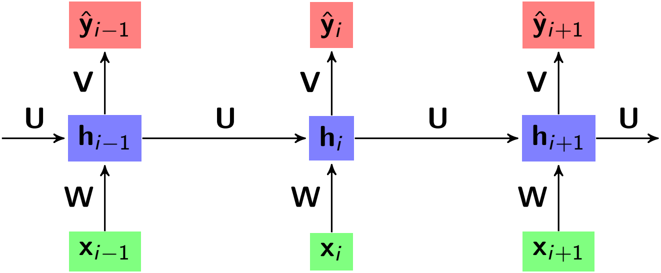
Recurrent Neural Networks



Recurrent Neural Networks

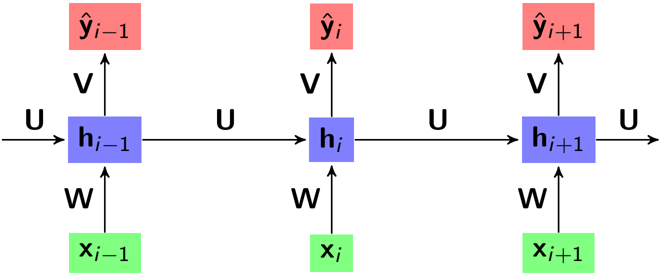


Recurrent Neural Networks



- Output $\hat{\mathbf{y}}_i$ depends on hidden state $\mathbf{h}_i$ (i.e. current word $\mathbf{x}_i$ and history/(past) context $\mathbf{h}_{i-1}$)

Recurrent Neural Networks



- Output $\hat{\mathbf{y}}_i$ depends on hidden state $\mathbf{h}_i$ (i.e. current word $\mathbf{x}_i$ and history/(past) context $\mathbf{h}_{i-1}$)
- What about future context?

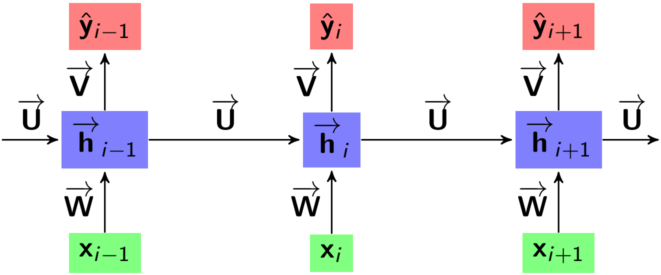
Bidirectional RNNs

- Idea: Train two RNNs: passing the input into one forward and one backward

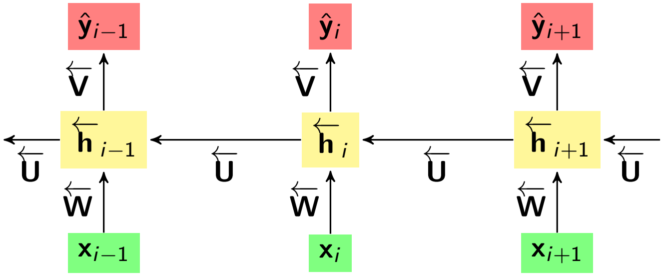
Bidirectional RNNs

- Idea: Train two RNNs: passing the input into one forward and one backward
- Output $\hat{\mathbf{y}}_i$ depends on forward hidden state $\overrightarrow{\mathbf{h}}_i$ and backward hidden state $\overleftarrow{\mathbf{h}}_i$

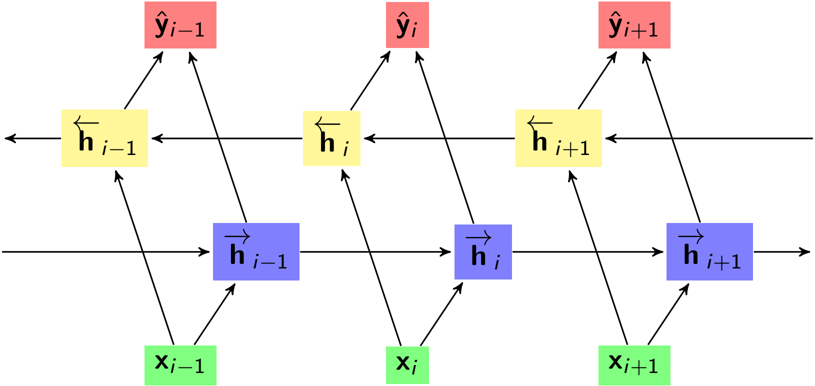
Forward RNN


$\hat{\mathbf{y}}_{i-1}$
$\hat{\mathbf{y}}_{i}$
$\hat{\mathbf{y}}_{i+1}$
$\overrightarrow{\mathbf{V}}$
$\overrightarrow{\mathbf{V}}$
$\overrightarrow{\mathbf{V}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{h}}_{i-1}$
$\overrightarrow{\mathbf{h}}_{i}$
$\overrightarrow{\mathbf{h}}_{i+1}$
$\overrightarrow{\mathbf{W}}$
$\overrightarrow{\mathbf{W}}$
$\overrightarrow{\mathbf{W}}$
$\mathbf{x}_{i-1}$
$\mathbf{x}_{i}$
$\mathbf{x}_{i+1}$

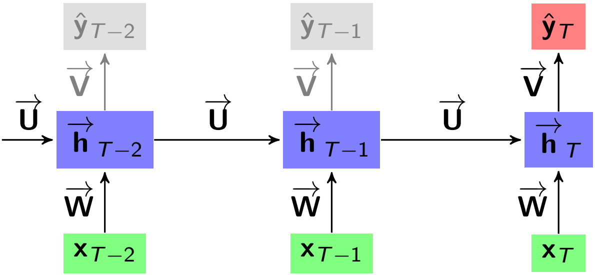
Backward RNN



Bidirectional RNN

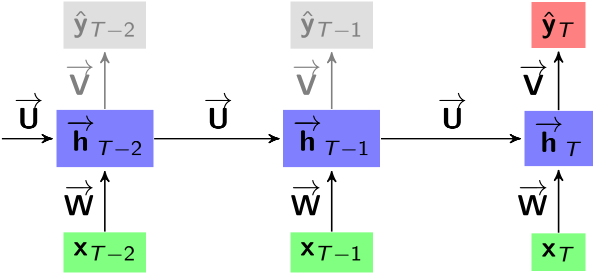


Bidirectional RNNs for Text Classification



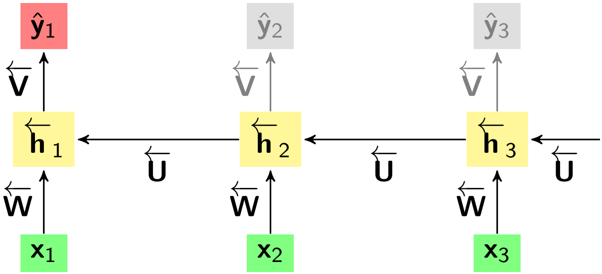
- $\overrightarrow{\mathbf{h}}_T$ encodes the whole text

Bidirectional RNNs for Text Classification



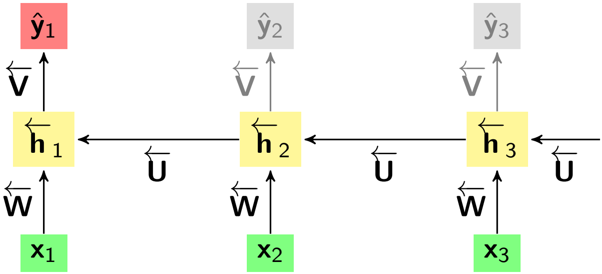
- $\overrightarrow{\mathbf{h}}_T$ encodes the whole text
  - Use $\overrightarrow{\mathbf{h}}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document

Bidirectional RNNs for Text Classification



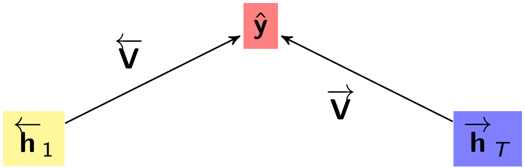
- $\overrightarrow{\mathbf{h}}_T$ encodes the whole text
  - Use $\overrightarrow{\mathbf{h}}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document
- $\overleftarrow{\mathbf{h}}_1$ also encodes the whole text

Bidirectional RNNs for Text Classification

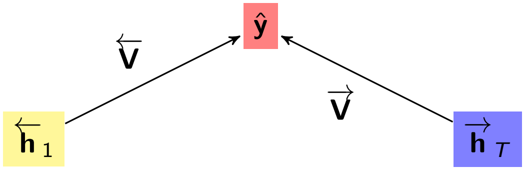


- $\overrightarrow{\mathbf{h}}_T$ encodes the whole text
  - Use $\overrightarrow{\mathbf{h}}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document
- $\overleftarrow{\mathbf{h}}_1$ also encodes the whole text
  - Use $\overleftarrow{\mathbf{h}}_1$ to predict class $\hat{\mathbf{y}}_1$ of entire document

Bidirectional RNNs for Text Classification



Bidirectional RNNs for Text Classification



- Use $\overrightarrow{\mathbf{h}}_T$ and $\overleftarrow{\mathbf{h}}_1$ to predict class $\hat{\mathbf{y}}$ of entire document

Context and Long-Distance Dependencies

- $\mathbf{h}_{i-1}$ encodes the (past, in a forward RNN) context $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$

Context and Long-Distance Dependencies

- $\mathbf{h}_{i-1}$ encodes the (past, in a forward RNN) context $\mathbf{x}_1, ..., \mathbf{x}_{i-1}$
  - But mostly $\mathbf{x}_{i-1}$, less $\mathbf{x}_{i-2}$, even less $\mathbf{x}_{i-3}$, ..., very little $\mathbf{x}_1$

Context and Long-Distance Dependencies

- $\mathbf{h}_{i-1}$ encodes the (past, in a forward RNN) context $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$
  - But mostly $\mathbf{x}_{i-1}$, less $\mathbf{x}_{i-2}$, even less $\mathbf{x}_{i-3}$, ..., very little $\mathbf{x}_1$
- Context is local

Context and Long-Distance Dependencies

- Example: subject-verb agreement

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for "was" is mostly "airline"

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for “was” is mostly “airline”

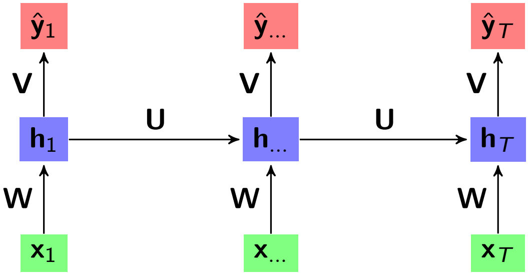
Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for “was” is mostly “airline”
  - The context for “were” is mostly “cancelling”, “was”, “airline”

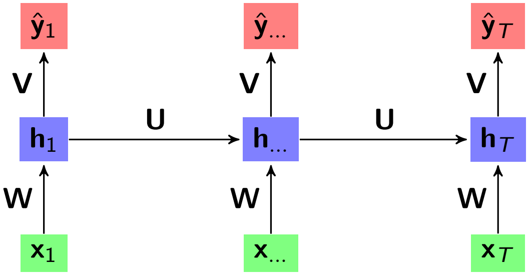
Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for "was" is mostly "airline"
  - The context for "were" is mostly "cancelling", "was", "airline"
    - Very little "flights"

Vanishing and Exploding Gradients

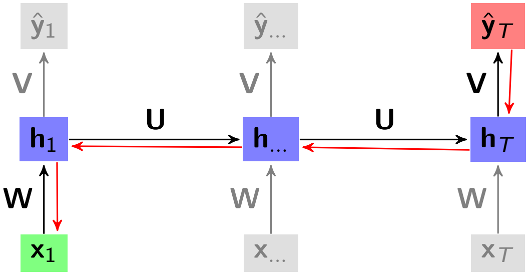


Vanishing and Exploding Gradients



- What is $(\nabla L)^{[\mathbf{W}]}_{1,T}$?

Vanishing and Exploding Gradients



- What is $(\nabla L)_{1,T}^{[\mathbf{W}]}$?

Vanishing and Exploding Gradients

- For all layers i:
  - $(\nabla L)^{[i]} = (\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}$

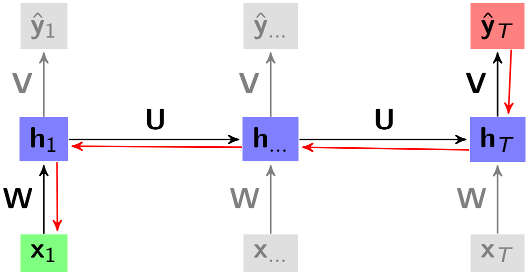
Vanishing and Exploding Gradients

- For all layers i:
  - $(\nabla L)^{[i]} = (\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}$
    - For simplicity, we will assume that the minibatch size $m = 1$

Vanishing and Exploding Gradients

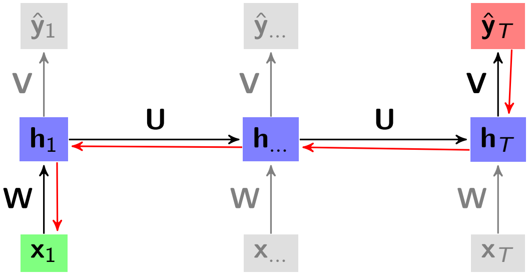
- For all layers i:
  - $(\nabla L)^{[i]} = (\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}$
    - For simplicity, we will assume that the minibatch size $m = 1$
- For an output layer $\mathcal{L}$:
  - $\delta^{[\mathcal{L}]} = \hat{\mathbf{y}} - \mathbf{y}$
- For a hidden layer i:
  - $\delta^{[i]} = (\delta^{[i+1]} \cdot (\mathbf{W}^{[i+1]})^T) \odot g'(\mathbf{z}^{[i]})$

Vanishing and Exploding Gradients



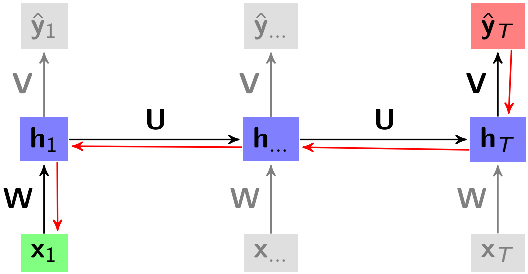
$$(\nabla L)_{1,T}^{[\mathbf{W}]} = \mathbf{x}^T \cdot \delta^{[\mathbf{h}_1]}$$

Vanishing and Exploding Gradients



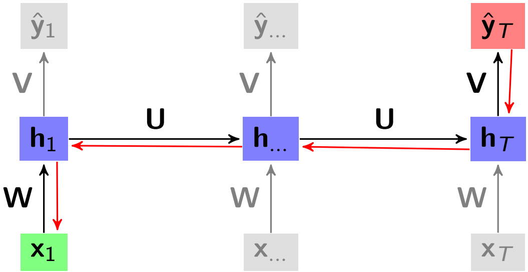
$$
\begin{aligned}
(\nabla L)^{[\mathbf{W}]}_{1,T} &= \mathbf{x}^T \cdot \delta^{[\mathbf{h}_1]} \\
&= \mathbf{x}^T \cdot ((\delta^{[\mathbf{h}_2]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]}))
\end{aligned}
$$

Vanishing and Exploding Gradients



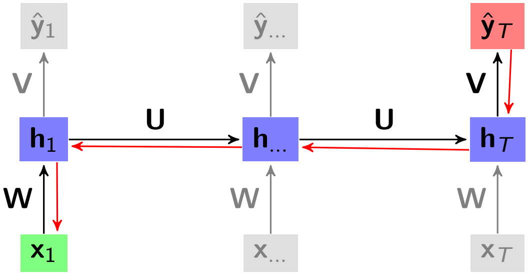
$$
\begin{aligned}
(\nabla L)^{[\mathbf{W}]}_{1,T} &= \mathbf{x}^T \cdot \delta^{[\mathbf{h}_1]} \\
&= \mathbf{x}^T \cdot ((\delta^{[\mathbf{h}_2]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]})) \\
&= \mathbf{x}^T \cdot ((((\delta^{[\mathbf{h}_3]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_2]})) \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]}))
\end{aligned}
$$

Vanishing and Exploding Gradients



$$
\begin{aligned}
(\nabla L)^{[\mathbf{W}]}_{1,T} &= \mathbf{x}^T \cdot \delta^{[\mathbf{h}_1]} \\
&= \mathbf{x}^T \cdot ((\delta^{[\mathbf{h}_2]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]})) \\
&= \mathbf{x}^T \cdot ((((\delta^{[\mathbf{h}_3]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_2]})) \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]})) \\
&= \dots
\end{aligned}
$$

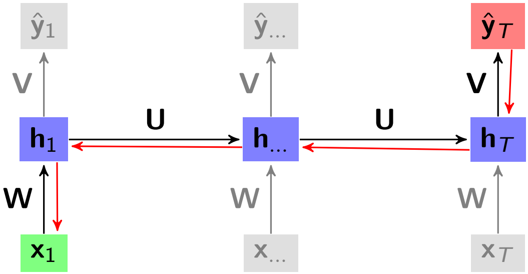
Vanishing and Exploding Gradients



$$
\begin{aligned}
(\nabla L)^{[\mathbf{W}]}_{1,T} &= \mathbf{x}^T \cdot \delta^{[\mathbf{h}_1]} \\
&= \mathbf{x}^T \cdot ((\delta^{[\mathbf{h}_2]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]})) \\
&= \mathbf{x}^T \cdot ((((\delta^{[\mathbf{h}_3]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_2]})) \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]})) \\
&= \dots
\end{aligned}
$$

- If weights/derivatives are small, vanishing gradient

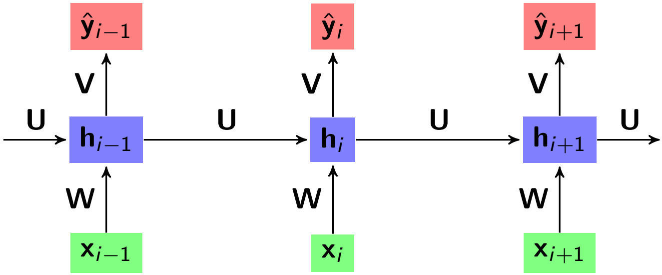
Vanishing and Exploding Gradients



$$
\begin{aligned}
(\nabla L)^{[\mathbf{W}]}_{1,T} &= \mathbf{x}^T \cdot \delta^{[\mathbf{h}_1]} \\
&= \mathbf{x}^T \cdot ((\delta^{[\mathbf{h}_2]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]})) \\
&= \mathbf{x}^T \cdot ((((\delta^{[\mathbf{h}_3]} \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_2]})) \cdot \mathbf{U}^T) \odot g'(\mathbf{z}^{[\mathbf{h}_1]})) \\
&= \dots
\end{aligned}
$$

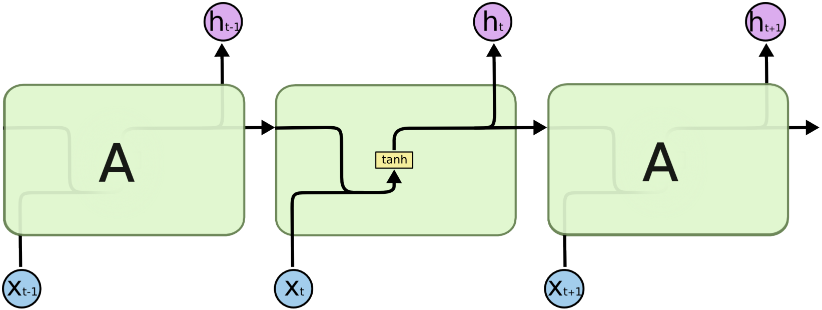
- If weights/derivatives are small, vanishing gradient
- If weights/derivatives are large, exploding gradient

Simple RNN



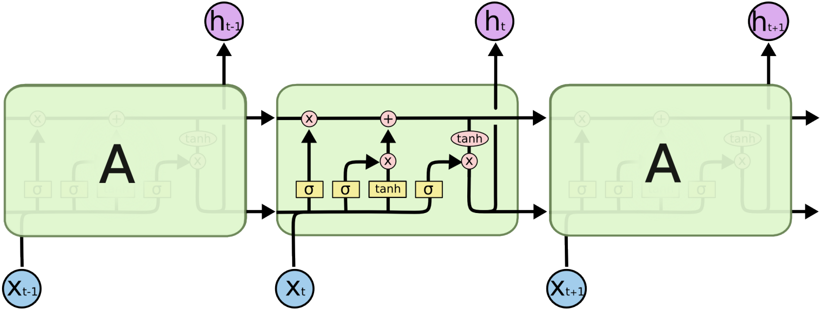
$\hat{\mathbf{y}}_{i-1}$
$\hat{\mathbf{y}}_{i}$
$\hat{\mathbf{y}}_{i+1}$
V
V
V
U
$\mathbf{h}_{i-1}$
U
$\mathbf{h}_{i}$
U
$\mathbf{h}_{i+1}$
U
W
W
W
$\mathbf{x}_{i-1}$
$\mathbf{x}_{i}$
$\mathbf{x}_{i+1}$

Simple RNN



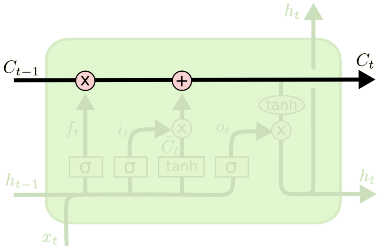
Source

Long Short-Term Memory



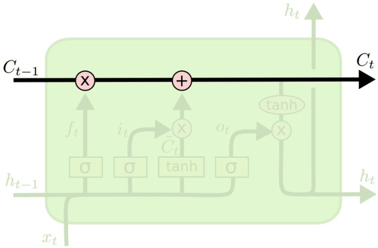
Source

Long Short-Term Memory



Source

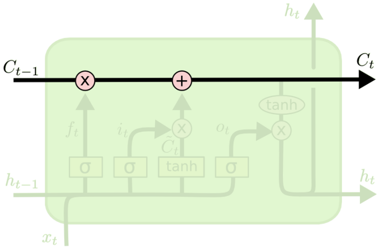
Long Short-Term Memory



Source

- Separate memory (cell) state

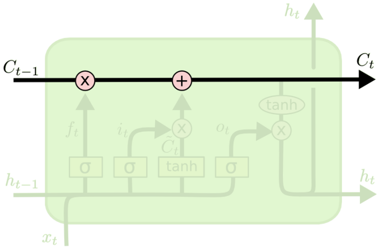
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates

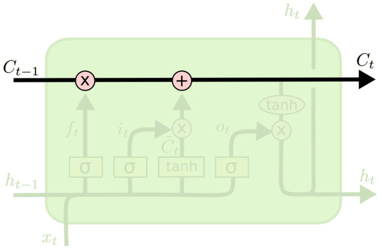
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers

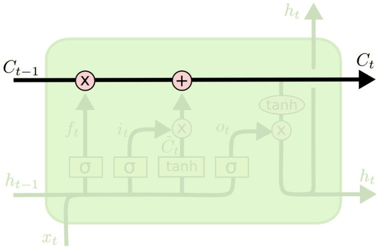
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time

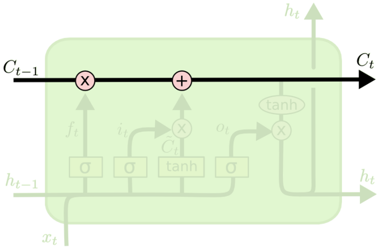
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time
    - May remember information from long ago

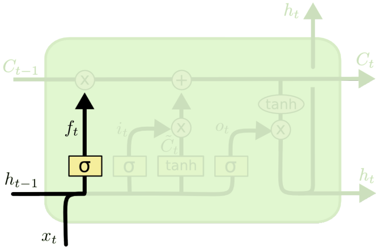
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time
    - May remember information from long ago
    - Gradients for memory don't decay with time

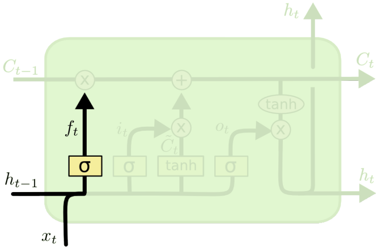
Forget Gate



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

Source

Forget Gate

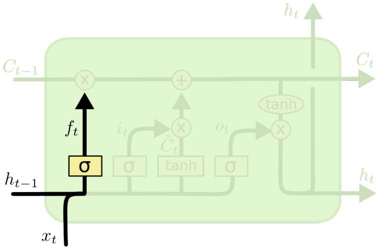


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Source

- Neural network layer with logistic activation function

Forget Gate

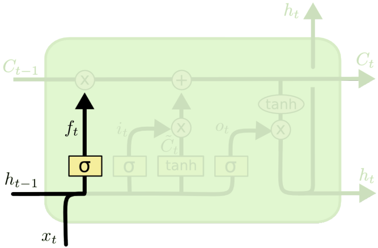


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Source

- Neural network layer with logistic activation function
- Element-wise multiplication of forget gate output with memory state

Forget Gate

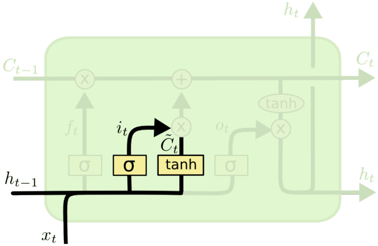


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Source

- Neural network layer with logistic activation function
- Element-wise multiplication of forget gate output with memory state
  - Mask: What parts of memory to forget/remember?

Input Gate

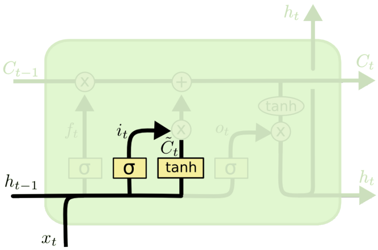


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$

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

Source

Input Gate

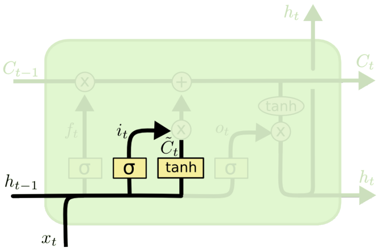


$$i_t = \sigma\left(W_i\cdot[h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C\cdot[h_{t-1}, x_t] \ + \ b_C)$$

Source

- Two parts

Input Gate

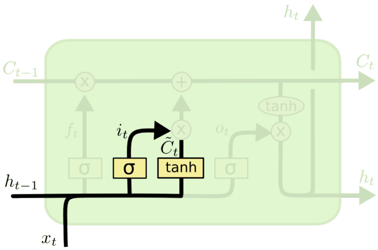


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Source

- Two parts
  1. Candidate choice

Input Gate



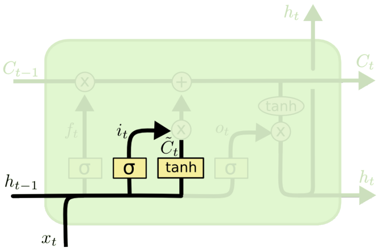
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Source

- Two parts
  1. Candidate choice
     - Logistic activation function

Input Gate



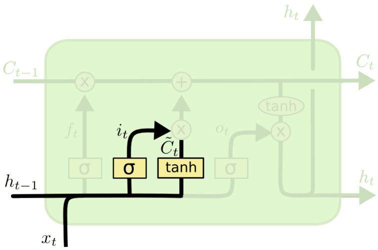
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Source

- Two parts
  1. Candidate choice
     - Logistic activation function
     - What parts of memory to update?

Input Gate



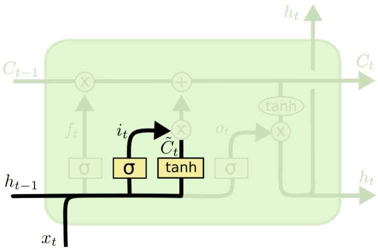
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Source

- Two parts
  1. Candidate choice
     - Logistic activation function
     - What parts of memory to update?
  2. Candidate values

Input Gate



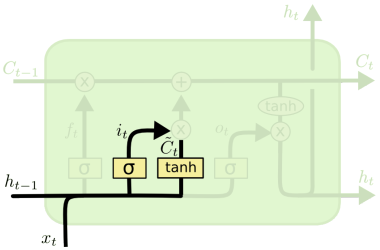
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Source

- Two parts
  1. Candidate choice
     - Logistic activation function
     - What parts of memory to update?
  2. Candidate values
     - Tanh activation function

Input Gate



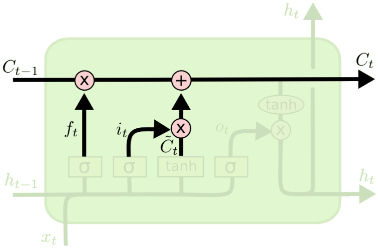
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Source

- Two parts
  1. Candidate choice
     - Logistic activation function
     - What parts of memory to update?
  2. Candidate values
     - Tanh activation function
     - How much to update them by?

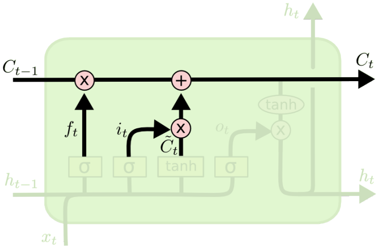
Input Gate



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

Source

Input Gate

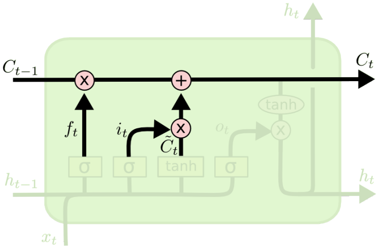


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source

- Element-wise multiplication of two outputs

Input Gate

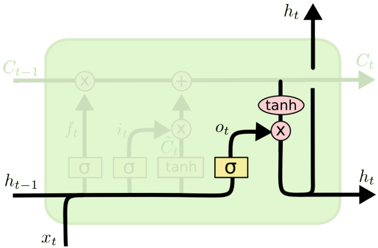


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source

- Element-wise multiplication of two outputs
- Then element-wise addition with memory state

Output Gate

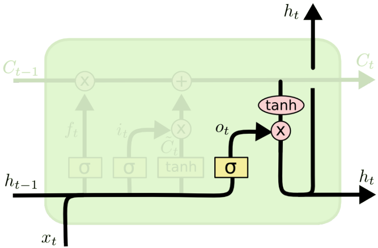


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Source

Output Gate

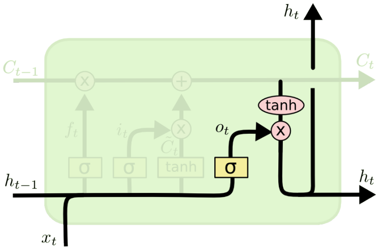


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Source

- Logistic activation function

Output Gate



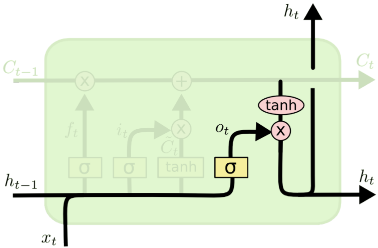
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Source

- Logistic activation function
  - What parts of memory to output?

Output Gate



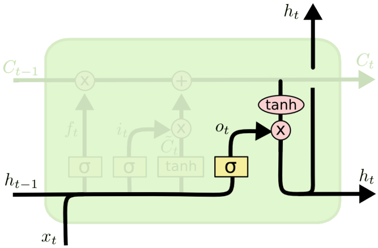
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

Source

- Logistic activation function
  - What parts of memory to output?
- Element-wise multiplication with tanh of memory state

Output Gate



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

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

Source

- Logistic activation function
  - What parts of memory to output?
- Element-wise multiplication with tanh of memory state
  - This is the "hidden layer output" that gets passed on to the output layer/next time step