Recurrent Neural Networks

CS 6956: Deep Learning for NLP

THE UNIVERSITY OF UTAH

Overview

1. Modeling sequences
2. Recurrent neural networks: An abstraction
3. Usage patterns for RNNs
4. BiDirectional RNNs
5. A concrete example: The Elman RNN
6. The vanishing gradient problem
7. Long short-term memory units

Overview

1. Modeling sequences
2. Recurrent neural networks: An abstraction
3. Usage patterns for RNNs
4. BiDirectional RNNs
5. A concrete example: The Elman RNN
6. The vanishing gradient problem
7. *Long short-term memory units*

Non-linear state updates

The state updates for the vanilla RNN were non-linear

Next state $\mathbf{s}_t = g(\mathbf{s}_{t-1}\mathbf{W}_S + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$

The non-linear activation can lead to vanishing or exploding gradients

Vanishing if absolute values of elements of $\nabla g(\mathbf{s}_{t-1}\mathbf{W}_S + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$ are strictly less than 1

Exploding if absolute values of elements of $\nabla g(\mathbf{s}_{t-1}\mathbf{W}_S + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$ are strictly more than 1

Linear
~~Non-linear~~ state updates?

The state updates for the vanilla RNN were non-linear

Next state $\mathbf{s}_t = g(\mathbf{s}_{t-1}\mathbf{W}_S + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$

The non-linear activation can lead to vanishing or exploding gradients

What if the cell state was updated linearly at each time step?

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathrm{update}_\mathrm{t}$$

Linear
~~Non-linear~~ state updates?

The state updates for the vanilla RNN were non-linear

Next state $\mathbf{s}_t = g(\mathbf{s}_{t-1}\mathbf{W}_S + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$

The non-linear activation can lead to vanishing or exploding gradients

What if the cell state was updated linearly at each time step?

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathrm{update}_t$$

No cascading multiplications due to the activations

Naïve linear updates

$$\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + \mathrm{update}_\mathrm{t}$$

No cascading multiplications due to the activations

How do we know what the update is?

We can calculate it as a function of the input and the state using a small neural network

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathrm{update}_\mathrm{t}$$

No cascading multiplications due to the activations

How do we know what the update is?

We can calculate it as a function of the input and the state using a small neural network

$$\mathrm{update}_t = g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

This addresses vanishing gradient problem. Why?

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

This addresses vanishing gradient problem. Why?

We can write the state explicitly as

$$\mathbf{s}_t = \sum_{i=1}^{t-1} g(\mathbf{s}_{i-1}\mathbf{W}_s + \mathbf{x}_i\mathbf{W}_I + \mathbf{b})$$

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

This addresses vanishing gradient problem. Why?

We can write the state explicitly as

$$\mathbf{s}_t = \sum_{i=1}^{t-1} g(\mathbf{s}_{i-1}\mathbf{W}_s + \mathbf{x}_i\mathbf{W}_I + \mathbf{b})$$

There is a term that directly relates $\mathbf{s}_t$ and $\mathbf{x}_0$without any cascading activations.

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

This addresses vanishing gradient problem. Why?

We can write the state explicitly as

$$\mathbf{s}_t = \sum_{i=1}^{t-1} g(\mathbf{s}_{i-1}\mathbf{W}_s + \mathbf{x}_i\mathbf{W}_I + \mathbf{b})$$

There is a term that directly relates $\mathbf{s}_t$ and $\mathbf{x}_0$without any cascading activations.

Why might this approach fail?

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Why might this fail?

Consider what happens when we are starting to train

- The parameters are random

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Why might this fail?

Consider what happens when we are starting to train

- The parameters are random
  - ↳They can lead to random states

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Why might this fail?

Consider what happens when we are starting to train

The parameters are random

↳They can lead to random states

↳…tend to be difficult to recover from

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Why might this fail?

Consider what happens when we are starting to train

The parameters are random

↳They can lead to random states

↳…tend to be difficult to recover from

↳…tend to lead to overflow

Naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Why might this fail?

Consider what happens when we are starting to train

The parameters are random

↳They can lead to random states

↳…tend to be difficult to recover from

↳…tend to lead to overflow

What we need: More control of how states and inputs interact

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \boxed{g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})}$$

Need better control of input-state interactions

- *Controlled state updates*
  - How much of a computed update should be saved for the next time step?

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

- *Controlled state updates*
  - How much of a computed update should be saved for the next time step?
  - **Intuition**: Think of the state as a memory and think of the update as a write to the memory.

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \boxed{g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})}$$

Need better control of input-state interactions

- *Controlled state updates*
  - How much of a computed update should be saved for the next time step?
  - **Intuition**: Think of the state as a memory and think of the update as a write to the memory.
  - Goal: Depending on what our current state and the current input is, choose what part of the update to add to the state

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

- *Controlled state reading*
  - What part of the previous state should be used to make decisions about the current input?

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

- *Controlled state reading*
  - What part of the previous state should be used to make decisions about the current input?
  - **Intuition**: Think of the state as a memory. Maybe to process certain inputs, we don't need to access all the memory

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

- *Controlled state reading*
  - What part of the previous state should be used to make decisions about the current input?
  - **Intuition**: Think of the state as a memory. Maybe to process certain inputs, we don't need to access all the memory
  - Goal: control what part of the state gets read

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

- *Controlled forgetting*
  - [Gers et al 2000]: Why should the state be remembered forever?

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

- *Controlled forgetting*
  - [Gers et al 2000]: Why should the state be remembered forever?
  - **Intuition**: Think of the state as a memory. We need a mechanism that allows unnecessary memories to be erased

Fixing naïve linear updates

$$\mathbf{s}_t = \mathbf{s}_{t-1} + g(\mathbf{s}_{t-1}\mathbf{W}_s + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Need better control of input-state interactions

- *Controlled forgetting*
  - [Gers et al 2000]: Why should the state be remembered forever?
  - **Intuition**: Think of the state as a memory. We need a mechanism that allows unnecessary memories to be erased
  - Goal: Control what part of the state gets erased

Design goals for controlling state updates

We want mechanisms for:

1. Depending on what our current state and the current input is, choose what part of the update to add to the state

2. Control what part of the state gets read

3. Control what part of the state gets erased

The answer: Gating

Everything we are dealing with is a vector

- Our goal is to selectively read, write and erase elements of a vector

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

$$\begin{bmatrix} 0.4 \\ 0.1 \\ -3.2 \\ -1.11 \\ 4.9 \\ -0.21 \\ 0.4 \end{bmatrix}$$

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

$$\begin{bmatrix} 0.4 \\ 0.1 \\ \boxed{-3.2} \\ -1.11 \\ \boxed{4.9} \\ -0.21 \\ 0.4 \end{bmatrix} \dashrightarrow \begin{bmatrix} 0 \\ 0 \\ -3.2 \\ 0 \\ 4.9 \\ 0 \\ 0 \end{bmatrix}$$

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

We can multiply each element with an 0 or 1 as required

$$\begin{bmatrix} 0.4 \\ 0.1 \\ -3.2 \\ -1.11 \\ 4.9 \\ -0.21 \\ 0.4 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -3.2 \\ 0 \\ 4.9 \\ 0 \\ 0 \end{bmatrix}$$

Element-wise product

or

Hadamard product

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

We can multiply each element with an 0 or 1 as required

$$\begin{bmatrix} 0.4 \\ 0.1 \\ -3.2 \\ -1.11 \\ 4.9 \\ -0.21 \\ 0.4 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -3.2 \\ 0 \\ 4.9 \\ 0 \\ 0 \end{bmatrix}$$

This binary mask acts as a ***gate*** that decides what information to keep and what to erase

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

We can multiply each element with an 0 or 1 as required

$$\begin{bmatrix} 0.4 \\ 0.1 \\ -3.2 \\ -1.11 \\ 4.9 \\ -0.21 \\ 0.4 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -3.2 \\ 0 \\ 4.9 \\ 0 \\ 0 \end{bmatrix}$$

This binary mask acts as a ***gate*** that decides what information to keep and what to erase

Question: How do we get these gate values?

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

We can multiply each element with an 0 or 1 as required

$$\begin{bmatrix} 0.4 \\ 0.1 \\ -3.2 \\ -1.11 \\ 4.9 \\ -0.21 \\ 0.4 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -3.2 \\ 0 \\ 4.9 \\ 0 \\ 0 \end{bmatrix}$$

This binary mask acts as a ***gate*** that decides what information to keep and what to erase

Question: How do we get these gate values?

Answer: A neural network predicts it

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

We can multiply each element with an 0 or 1 as required

$$\begin{bmatrix} 0.4 \\ 0.1 \\ -3.2 \\ -1.11 \\ 4.9 \\ -0.21 \\ 0.4 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -3.2 \\ 0 \\ 4.9 \\ 0 \\ 0 \end{bmatrix}$$

The bad news: If the gate values were produced by a model, then it will not be differentiable – all these values are discrete

The answer: Gating

Everything we are dealing with is a vector

Our goal is to selectively read, write and erase elements of a vector

Example: Suppose we want to read only the shaded elements

We can multiply each element with an 0 or 1 as required

$$\begin{bmatrix} 0.4 \\ 0.1 \\ -3.2 \\ -1.11 \\ 4.9 \\ -0.21 \\ 0.4 \end{bmatrix} \odot \begin{bmatrix} 0.1 \\ 0.01 \\ 0.5 \\ 0.1 \\ 0.9 \\ 0.2 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.04 \\ 0.001 \\ -1.6 \\ 0.111 \\ 4.41 \\ -0.042 \\ 0.016 \end{bmatrix}$$

Instead of producing 0's or 1's, gate values are allowed to be between zero and one. That is, they are the output of a sigmoid

Gated architectures

- A large family of models
  - Two commonly used members
    - Long Short-Term Memory (LSTM)
    - Gated Recurrent Unit (GRU)
  - And many variants

- Each time step includes a collection of gates that decide:
  - What part of the state should be read
  - What part of the state should be over-written
  - What part of the update should be saved to the state

Long Short-term Memory (LSTM) Unit

- Each recurrent unit receives two vectors from the previous one
  - Long term memory: $\mathbf{c}_{t-1}$
  - Hidden state: $\mathbf{h}_{t-1}$

- The memory is the component that is updated in the linear fashion described so far
  - The hidden state encodes a non-linearity (as we will see)

- Using the current input $\mathbf{x}_t$, the LSTM cell performs the following operations:
  1. Compute the new value of the memory $\mathbf{c}_t$
  2. Compute the new value of the hidden state $\mathbf{h}_t$
  3. Output = $\mathbf{h}_t$

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$and previous hidden state $\mathbf{h}_{t-1}$

1. Compute the update to the memory

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

1. Compute the update to the memory

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Current input

Previous hidden state

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

1. Compute the update to the memory

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

Parameters

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

1. Compute the update to the memory

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_I + \mathbf{b})$$

At a high level, this is similar to the update in the simple RNN.

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

2. Compute what part of this update should be retained
   – Called the ***input*** gate

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

2. Compute what part of this update should be retained
   - Called the ***input*** gate

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$

Different parameters from the previous ones

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

2. Compute what part of this update should be retained
   - Called the ***input*** gate

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$

Element-wise sigmoid activation – produces a vector with entries between zero and one

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$

3. Compute what part of the previous cell state should be forgotten
   – Called the ***forget*** gate

$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\boxed{\mathbf{W}_H^f} + \mathbf{x}_t\boxed{\mathbf{W}_\mathrm{I}^f} + \boxed{\mathbf{b}^f})$$

Different parameters from the previous ones

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$

3. Compute what part of the previous cell state should be forgotten
   – Called the ***forget*** gate

$$\mathbf{f} = \boxed{\sigma}(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$

Element-wise sigmoid activation – produces a vector with entries between zero and one

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$

4. Compute the updated cell state

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Element-wise product between the forget gate and the previous memory: Decides what information from the previous memory should be retained

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$

$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$

4. Compute the updated cell state

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Element-wise product between the input gate and the update computed above: Decides what information from the update should be retained

LSTM: Updating the memory

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$

4. Compute the updated cell state

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Adding these gated components gives the memory for this cell

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$

$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Now we can start computing the hidden state $\mathbf{h}_t$

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Now we can start computing the hidden state $\mathbf{h}_t$

Serves two roles:

1. Used to compute the cell update and the various gates
2. Becomes the output of the cell

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Now we can start computing the hidden state $\mathbf{h}_t$

Serves two roles:
1. Used to compute the cell update and the various gates
2. Becomes the output of the cell

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

5. Compute what part of the memory should contribute to the hidden state
   - Called the output gate

$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

5. Compute what part of the memory should contribute to the hidden state
   - Called the output gate

$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\boxed{\mathbf{W}_H^o} + \mathbf{x}_t\boxed{\mathbf{W}_\mathrm{I}^o} + \boxed{\mathbf{b}^o})$$

Different parameters
from the previous ones

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

5. Compute what part of the memory should contribute to the hidden state

– Called the output gate

$$\boxed{\mathbf{o}} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

Element-wise sigmoid activation – produces a vector with entries between zero and one

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

6. Compute the value of the hidden state

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

6. Compute the value of the hidden state

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Element-wise product of the output gate and an activated version of the memory

LSTM: Computing the hidden state

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

6. Compute the value of the hidden state

$$\boxed{\mathbf{h}_t} = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

All elements of the hidden state are between -1 and 1

LSTM: Computing the output

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$
$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

7. Output of the cell = $\mathbf{h}_t$

   We refer to the output as $\mathbf{y}_t$ in the previous lectures

LSTM: All the updates together

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$
$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Let us look at these state updates more carefully

LSTM: All the updates together

Given previous memory $\mathbf{c}_{t-1}$and previous hidden state $\mathbf{h}_{t-1}$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

LSTM: All the updates together

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

Input, forget and output gates: The differentiable gating mechanism for the LSTM cell

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

LSTM: All the updates together

Given previous memory $\mathbf{c}_{t-1}$and previous hidden state $\mathbf{h}_{t-1}$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

Compute the proposed update for the memory

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

LSTM: All the updates together

Given previous memory $\mathbf{c}_{t-1}$and previous hidden state $\mathbf{h}_{t-1}$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Update the memory as the combination of the previous memory and the update proposal

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

LSTM: All the updates together

Given previous memory $\mathbf{c}_{t-1}$ and previous hidden state $\mathbf{h}_{t-1}$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

Update the memory as the combination of the previous memory and the update proposal

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Linear update. Avoids vanishing gradient problem

LSTM: All the updates together

Given previous memory $\mathbf{c}_{t-1}$and previous hidden state $\mathbf{h}_{t-1}$

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

$$\boxed{\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)}$$

Compute the hidden state by gating a transformed version of the memory

Parameters of the LSTM unit: W's and b's

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Peepholes

$$\mathbf{i} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^i + \mathbf{x}_t\mathbf{W}_\mathrm{I}^i + \mathbf{b}^i)$$
$$\mathbf{f} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^f + \mathbf{x}_t\mathbf{W}_\mathrm{I}^f + \mathbf{b}^f)$$
$$\mathbf{o} = \sigma(\mathbf{h}_{t-1}\mathbf{W}_H^o + \mathbf{x}_t\mathbf{W}_\mathrm{I}^o + \mathbf{b}^o)$$

All these can also include another term involving the previous memory. If so, they are called **peepholes** in the literature

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tilde{\mathbf{c}}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Inside a Long Short Term Memory unit

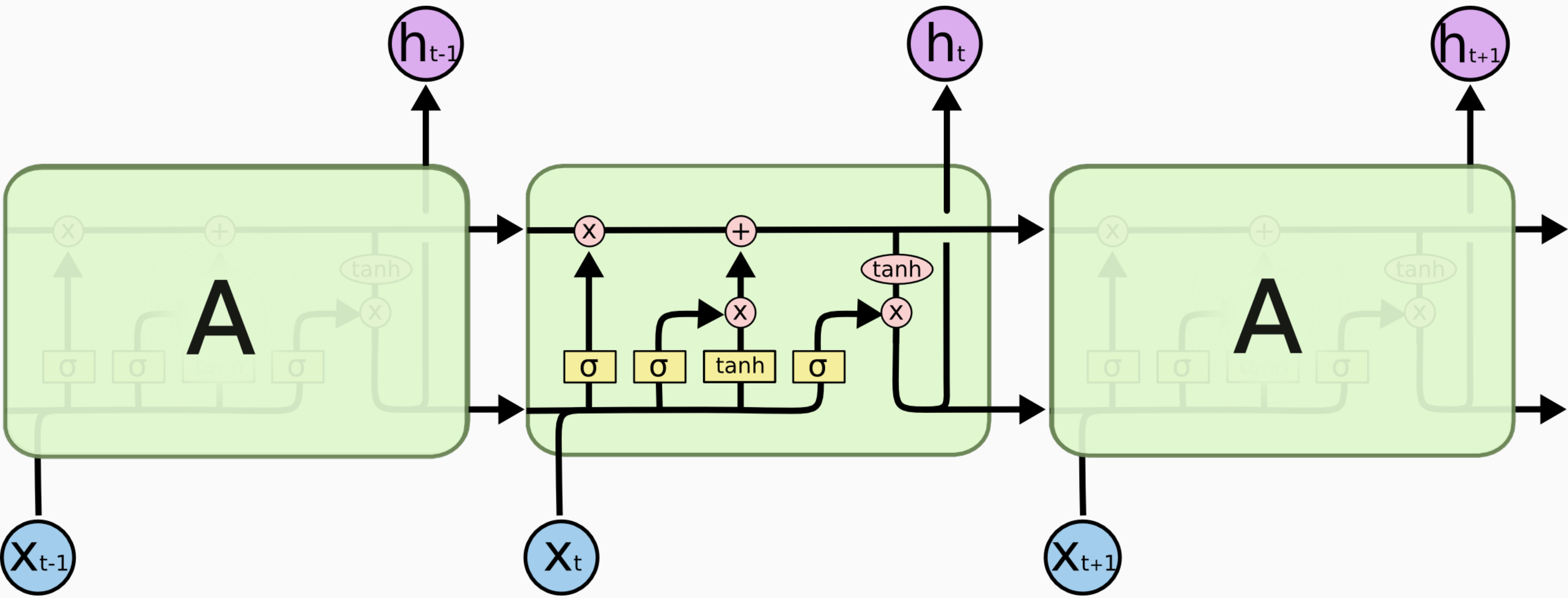


Figure from Chris Olah's blog

Let us zoom in

Cell state

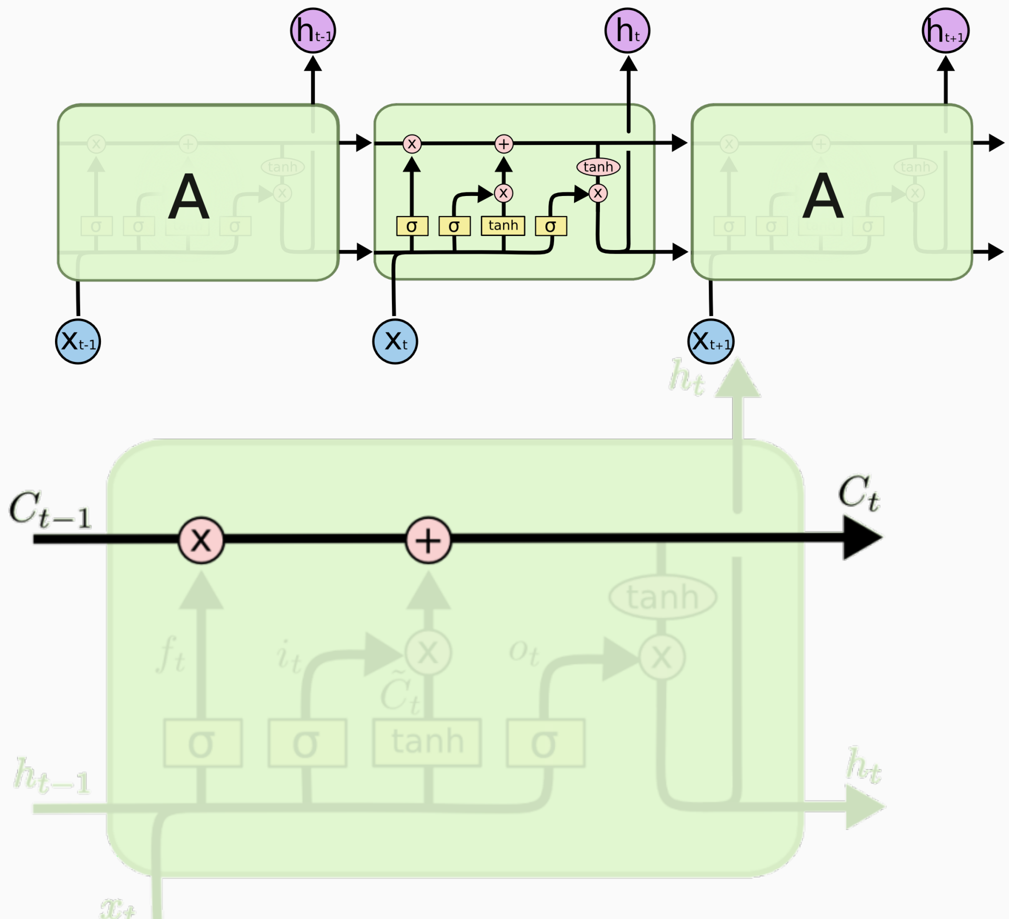
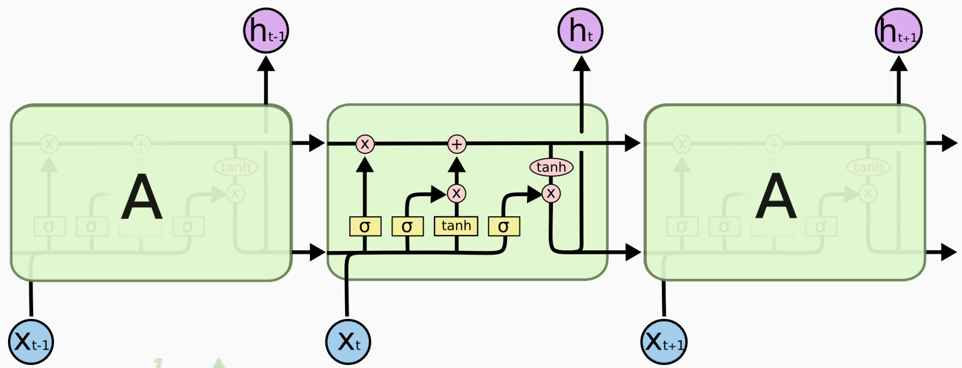
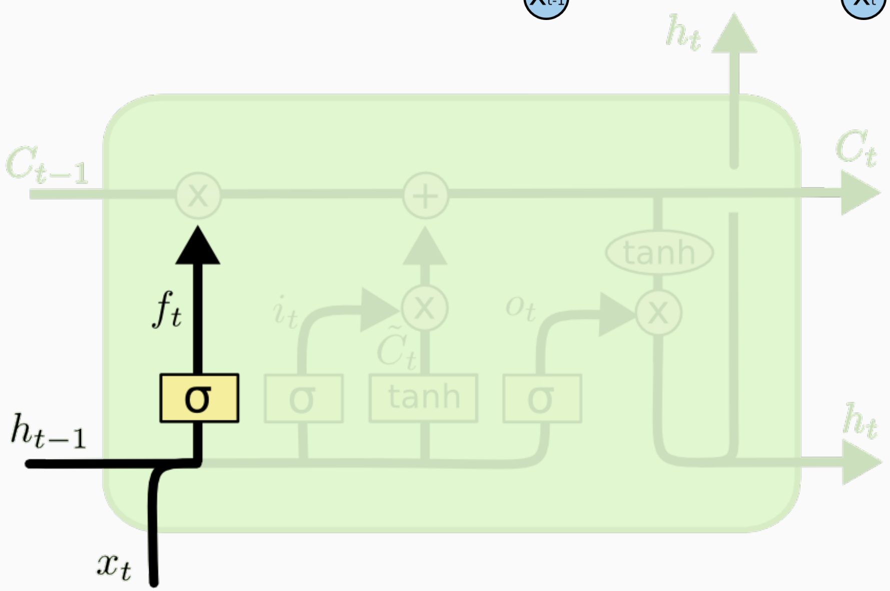


Figure from Chris Olah’s blog

Let us zoom in



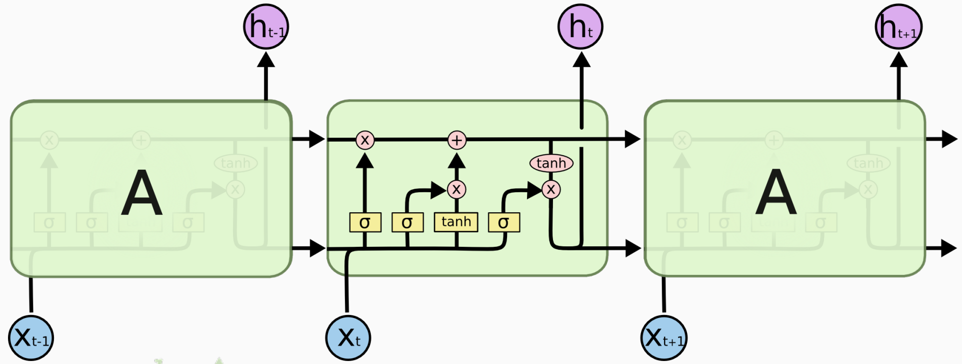


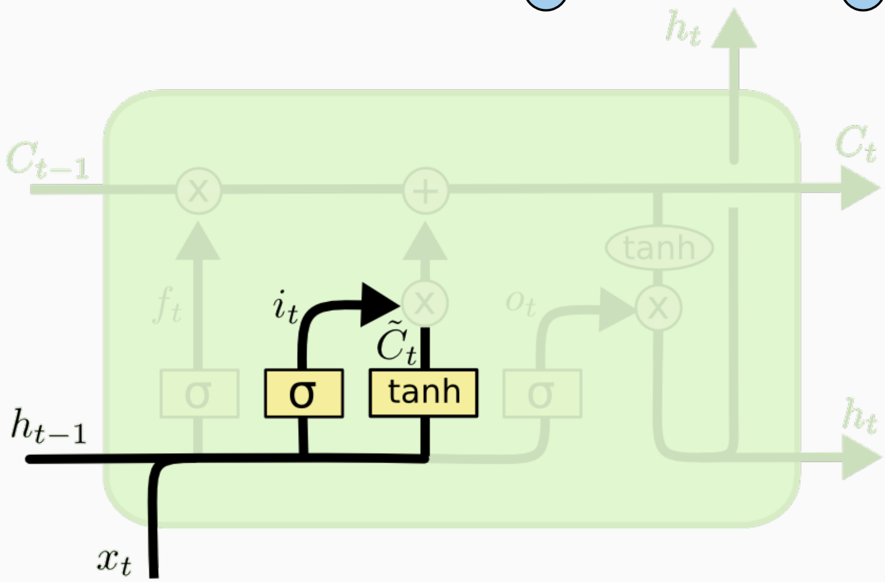
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

The forget gate: Use the current input to decide what to erase in the cell state

Figure from Chris Olah's blog

Let us zoom in





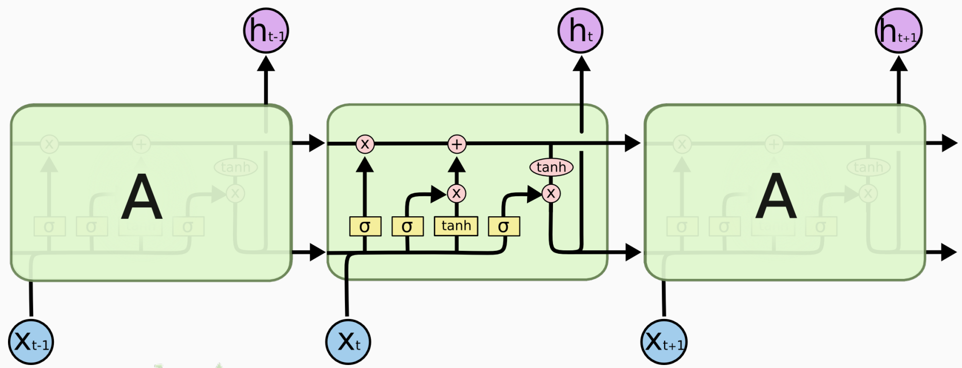
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

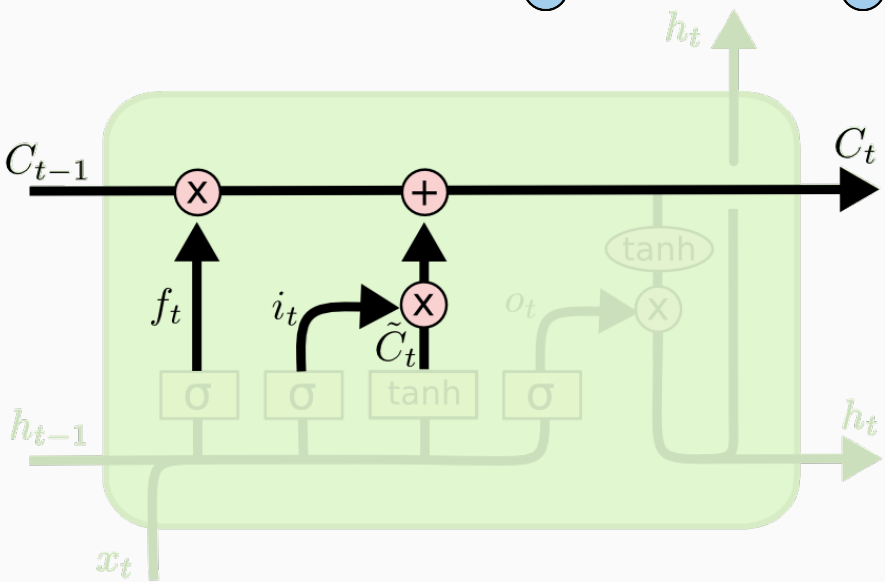
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Create a new cell state and also a gate that decides what part of the newly created cell state should be remembered

Figure from Chris Olah's blog

Let us zoom in



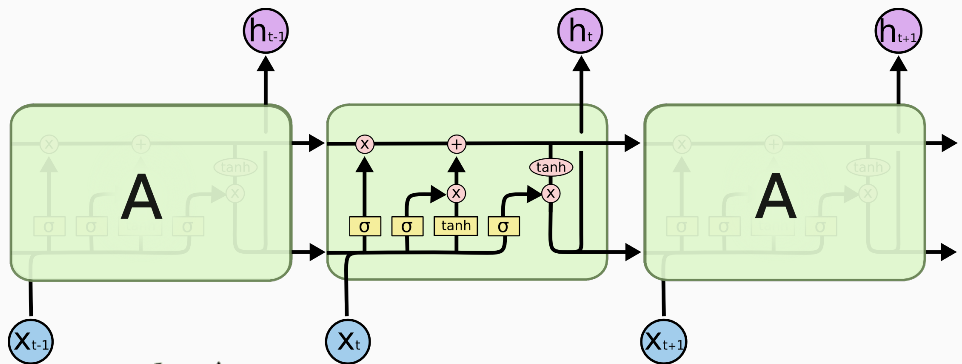


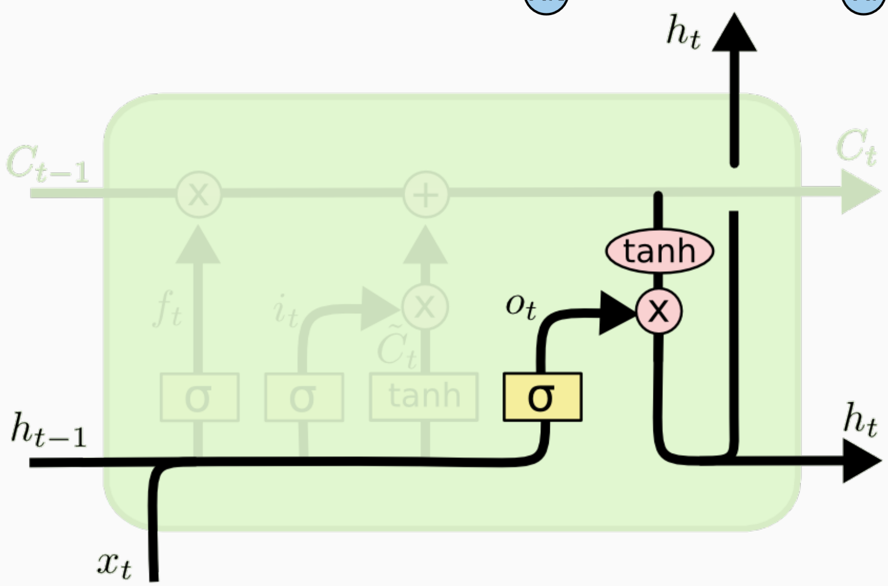
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

New cell state = remaining part of previous state + newly computed information

Figure from Chris Olah's blog

Let us zoom in





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

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

Finally, output = filtered version of the new cell state

Figure from Chris Olah's blog

Why LSTMs?

- The LSTM cell is one of the most commonly used building blocks in deep learning for NLP
  - Avoids the vanishing and exploding gradient problem, and empirically successful
- … but can be complicated
  - Requires a large number of parameters
- Do we need all this complexity?
  - Are there other simpler gated architectures that avoid the vanishing gradient problem?

Gated Recurrent Units (GRUs)

[Cho et al 2014]

- An attempt at simplifying the LSTM cell

- What do we need?
  - We need a linear update of the cell states
  - We want a gating mechanism to control how to interpolate between the previous state and the proposed update
  - We want a gate to control what part of the previous state should be read

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

1. Compute the values of two gates

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

1. Compute the values of two gates
   - Reset gate to decide what part of the previous state should be read to compute the update

$$\mathbf{r} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^r + \mathbf{x}_t\mathbf{W}_\mathrm{I}^r + \mathbf{b}^r)$$

Similar to the gates in the LSTM cell: uses element-wise sigmoid

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

1. Compute the values of two gates
   – Reset gate to decide what part of the previous state should be read to compute the update

$$\mathbf{r} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^r + \mathbf{x}_t\mathbf{W}_\mathrm{I}^r + \mathbf{b}^r)$$

   – Update gate to decide how to interpolate between the previous cell state and the proposed update

$$\mathbf{z} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^z + \mathbf{x}_t\mathbf{W}_\mathrm{I}^z + \mathbf{b}^z)$$

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

1. Compute the values of two gates
   - Reset gate to decide what part of the previous state should be read to compute the update
   $$\mathbf{r} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^r + \mathbf{x}_t\mathbf{W}_\mathrm{I}^r + \mathbf{b}^r)$$
   - Update gate to decide how to interpolate between the previous cell state and the proposed update
   $$\mathbf{z} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^z + \mathbf{x}_t\mathbf{W}_\mathrm{I}^z + \mathbf{b}^z)$$

2. Compute the proposed update
$$\tilde{\mathbf{s}} = \tanh((\mathbf{r} \odot \mathbf{s}_{t-1})\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

1. Compute the values of two gates
   - Reset gate to decide what part of the previous state should be read to compute the update

$$\mathbf{r} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^r + \mathbf{x}_t\mathbf{W}_\mathrm{I}^r + \mathbf{b}^r)$$

   - Update gate to decide how to interpolate between the previous cell state and the proposed update

$$\mathbf{z} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^z + \mathbf{x}_t\mathbf{W}_\mathrm{I}^z + \mathbf{b}^z)$$

2. Compute the proposed update

$$\tilde{\mathbf{s}} = \tanh((\mathbf{r} \odot \mathbf{s}_{t-1})\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

Use the reset gate to selectively read the previous state

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

1. Compute the values of two gates
   – Reset gate to decide what part of the previous state should be read to compute the update
   $$\mathbf{r} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^r + \mathbf{x}_t\mathbf{W}_\mathrm{I}^r + \mathbf{b}^r)$$
   – Update gate to decide how to interpolate between the previous cell state and the proposed update
   $$\mathbf{z} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^z + \mathbf{x}_t\mathbf{W}_\mathrm{I}^z + \mathbf{b}^z)$$
2. Compute the proposed update
   $$\tilde{\mathbf{s}} = \tanh((\mathbf{r} \odot \mathbf{s}_{t-1})\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$
3. Compute the new cell state
   $$\boldsymbol{s}_t = (1 - \mathbf{z}) \odot \mathbf{s}_{t-1} + \mathbf{z} \odot \tilde{\mathbf{s}}$$

Gated recurrent unit

Given the previous cell state $\mathbf{s}_{t-1}$ and current input $\mathbf{x}_t$

1. Compute the values of two gates
   – Reset gate to decide what part of the previous state should be read to compute the update

$$\mathbf{r} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^r + \mathbf{x}_t\mathbf{W}_\mathrm{I}^r + \mathbf{b}^r)$$

   – Update gate to decide how to interpolate between the previous cell state and the proposed update

$$\mathbf{z} = \sigma(\mathbf{s}_{t-1}\mathbf{W}_H^z + \mathbf{x}_t\mathbf{W}_\mathrm{I}^z + \mathbf{b}^z)$$

2. Compute the proposed update

$$\tilde{\mathbf{s}} = \tanh((\mathbf{r} \odot \mathbf{s}_{t-1})\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

3. Compute the new cell state

$$\boldsymbol{s}_t = (1 - \mathbf{z}) \odot \mathbf{s}_{t-1} + \mathbf{z} \odot \tilde{\mathbf{s}}$$

Linear interpolation between the previous state and the current proposal

LSTM extensions: Peephole connections

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

The proposed update to the memory depends on the previous $\mathbf{h}_{t-1}$, but not on the previous $\boldsymbol{c}_{t-1}$

- Same for all the gates as well

LSTM extensions: Peephole connections

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

The proposed update to the memory depends on the previous $\mathbf{h}_{t-1}$, but not on the previous $\boldsymbol{c}_{t-1}$

- Same for all the gates as well

Peepholes: All the state updates depend on both $\mathbf{h}_{t-1}$ and $\mathbf{c}_{t-1}$

$$\tilde{\mathbf{c}} = \tanh(\mathbf{h}_{t-1}\mathbf{W}_H + \mathbf{c}_{t-1}\mathbf{W}_\mathrm{P} + \mathbf{x}_t\mathbf{W}_\mathrm{I} + \mathbf{b})$$

Empirical observations

- LSTM and GRU are only two ways to use gates to avoid vanishing and exploding gradients
- Which one is better? Are there other variants that may be even better?

Empirical observations

- LSTM and GRU are only two ways to use gates to avoid vanishing and exploding gradients
- Which one is better? Are there other variants that may be even better?

- [Jozefowicz et al 2015]: An empirical comparison of about 10,000 different variants of this idea on three different tasks
  - There are some minor variants of GRU that appear to be better
  - It appears that GRU slightly outperforms the LSTM
  - LSTM with a forget gate bias set to 1 is also nearly as good