Deep Learning for Computer Vision

LSTMs and GRUs

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Review: Questions

Question

- How to tackle the vanishing gradient problem in RNNs with solutions that don't change the architecture?

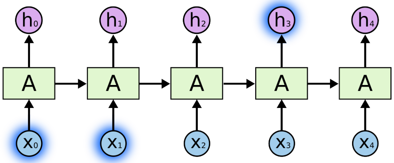
Review: Questions

Question

- How to tackle the vanishing gradient problem in RNNs with solutions that don't change the architecture? **Use ReLU; Regularization; Better initialization of weights; Use only short time sequences**

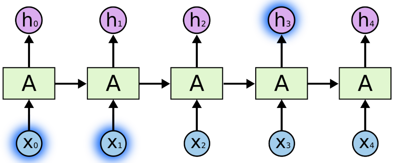
Long-Term Dependencies: The Problem

- RNNs connect previous information to present task which:
  - may be enough for predicting the next word for "*the clouds are in the sky*"

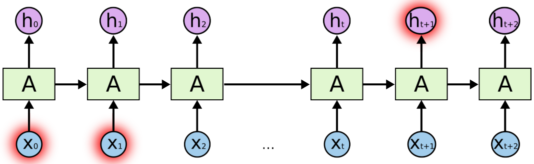


Long-Term Dependencies: The Problem

- RNNs connect previous information to present task which:
  - may be enough for predicting the next word for "*the clouds are in the sky*"



  - may not be enough when more context is needed: "*I grew up in France ... I speak fluent French*"



Credit: Christopher Olah

Recall: Training RNNs

- Recommended method: Backprop Through Time (BPTT)

Recall: Training RNNs

- Recommended method: Backprop Through Time (BPTT)
- Limitations of BPTT
  - Vanishing Gradients
  - Exploding Gradients
- How to overcome by changes in RNN architecture?

Recall: Training RNNs

- Recommended method: Backprop Through Time (BPTT)
- Limitations of BPTT
  - Vanishing Gradients
  - Exploding Gradients
- How to overcome by changes in RNN architecture?
  - **LSTMs** (1997)
  - **GRUs** (2014)

Long Short-Term Memory (LSTM)[1]

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the long-term dependency problem

Credit: Christopher Manning, Stanford Univ

[1] Hochreiter and Schmidhuber, Long Short-Term Memory, Neural Computation, 1997

Long Short-Term Memory (LSTM)[1]

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the long-term dependency problem
- On step t, there is a **hidden state** $h^{(t)}$ and a **cell state** $c^{(t)}$

Credit: Christopher Manning, Stanford Univ

[1] Hochreiter and Schmidhuber, Long Short-Term Memory, Neural Computation, 1997

Long Short-Term Memory (LSTM)[1]

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the long-term dependency problem
- On step t, there is a **hidden state** $h^{(t)}$ and a **cell state** $c^{(t)}$
  - The cell **stores long-term information**

Credit: Christopher Manning, Stanford Univ

[1] Hochreiter and Schmidhuber, Long Short-Term Memory, Neural Computation, 1997

Long Short-Term Memory (LSTM)[1]

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the long-term dependency problem
- On step t, there is a **hidden state** $h^{(t)}$ and a **cell state** $c^{(t)}$
  - The cell **stores long-term information**
  - The LSTM can **erase**, **write** and **read** information from the cell

Credit: Christopher Manning, Stanford Univ

[1] Hochreiter and Schmidhuber, Long Short-Term Memory, Neural Computation, 1997

Long Short-Term Memory (LSTM)[1]

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the long-term dependency problem
- On step t, there is a **hidden state** $h^{(t)}$ and a **cell state** $c^{(t)}$
  - The cell **stores long-term information**
  - The LSTM can **erase**, **write** and **read** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**

Credit: Christopher Manning, Stanford Univ

[1] Hochreiter and Schmidhuber, Long Short-Term Memory, Neural Computation, 1997

Long Short-Term Memory (LSTM)[1]

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the long-term dependency problem
- On step t, there is a **hidden state** $h^{(t)}$ and a **cell state** $c^{(t)}$
  - The cell **stores long-term information**
  - The LSTM can **erase**, **write** and **read** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between

Credit: Christopher Manning, Stanford Univ

[1] Hochreiter and Schmidhuber, Long Short-Term Memory, Neural Computation, 1997

Long Short-Term Memory (LSTM)[1]

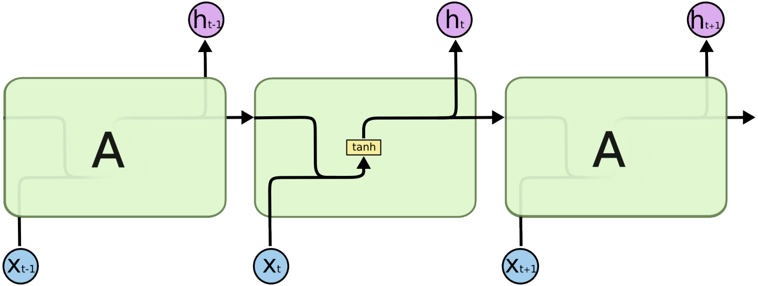
- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the long-term dependency problem
- On step t, there is a **hidden state** $h^{(t)}$ and a **cell state** $c^{(t)}$
  - The cell **stores long-term information**
  - The LSTM can **erase**, **write** and **read** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between
  - The gates are dynamic, their value is computed based on current context

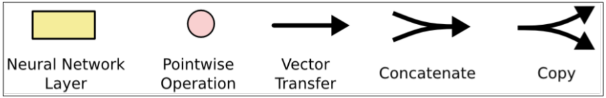
Credit: Christopher Manning, Stanford Univ

[1] Hochreiter and Schmidhuber, Long Short-Term Memory, Neural Computation, 1997

LSTMs

- All RNNs have the form of a **chain of repeating modules** of neural network
- Repeating module in a vanilla RNN is a single layer with `tanh` activation

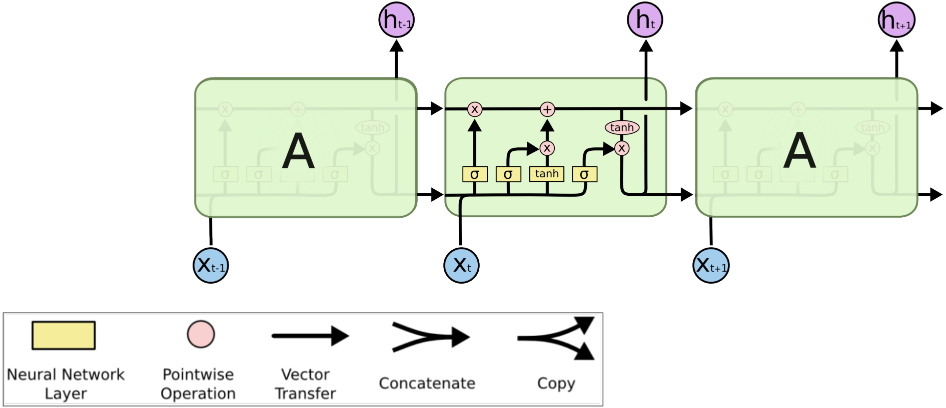




Credit: Christopher Olah

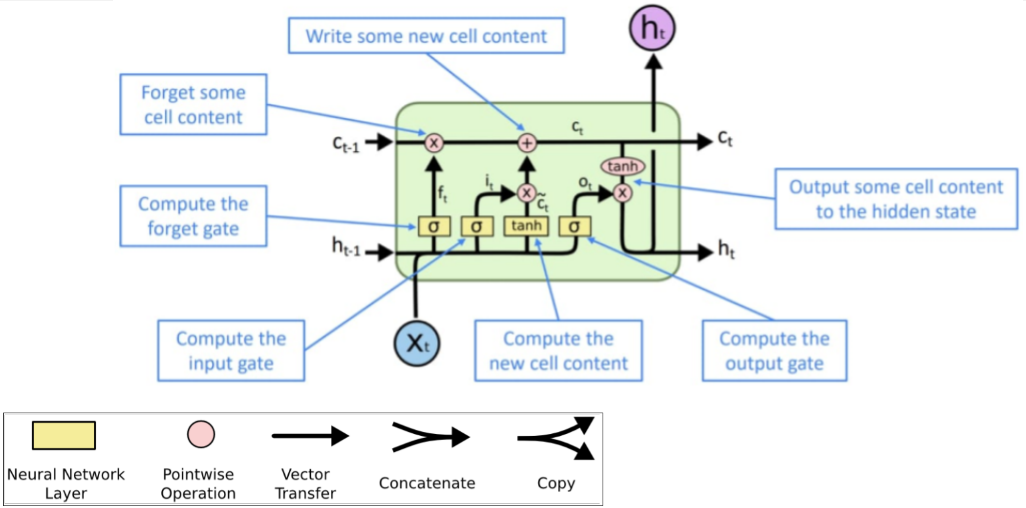
LSTMs

- All RNNs have the form of a **chain of repeating modules** of neural network
- Repeating module in an **LSTM** contains four interacting layers



Credit: Christopher Olah

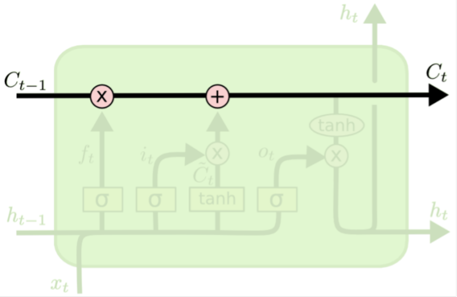
LSTMs



Credit: Christopher Manning, Stanford Univ

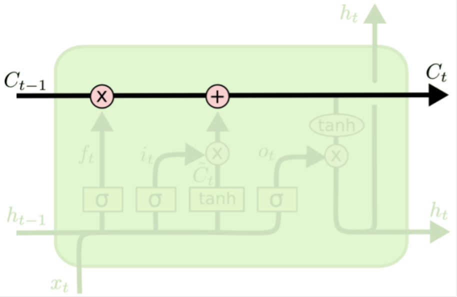
LSTMs Explained

- **Cell state (C_t):**



Credit: Christopher Olah

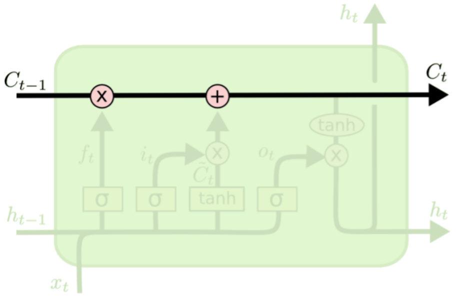
- **Cell state (C_t):**
  - Information can flow along cell state unchanged. Why is this important?



Credit: Christopher Olah

LSTMs Explained

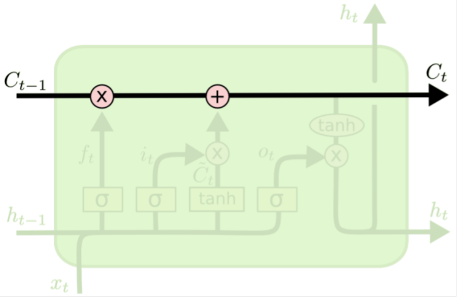
- **Cell state (C_t):**
  - Information can flow along cell state unchanged. Why is this important?
  - Ability to remove or add information to cell state, regulated by gates

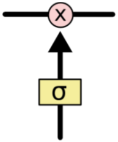


Credit: Christopher Olah

LSTMs Explained

- **Cell state (C_t):**
  - Information can flow along cell state unchanged. Why is this important?
  - Ability to remove or add information to cell state, regulated by gates
- **Gates:**

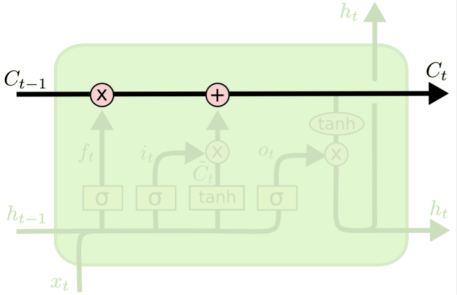


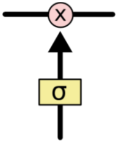


Credit: Christopher Olah

LSTMs Explained

- **Cell state (C_t):**
  - Information can flow along cell state unchanged. Why is this important?
  - Ability to remove or add information to cell state, regulated by gates
- **Gates:**
  - Composed of a sigmoid neural net layer and a pointwise multiplication operation

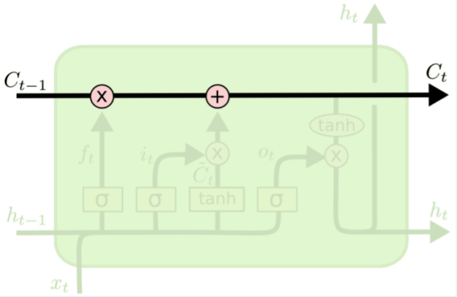


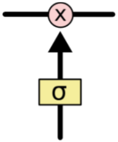


Credit: Christopher Olah

LSTMs Explained

- **Cell state (C_t):**
  - Information can flow along cell state unchanged. Why is this important?
  - Ability to remove or add information to cell state, regulated by gates
- **Gates:**
  - Composed of a sigmoid neural net layer and a pointwise multiplication operation
  - Sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through

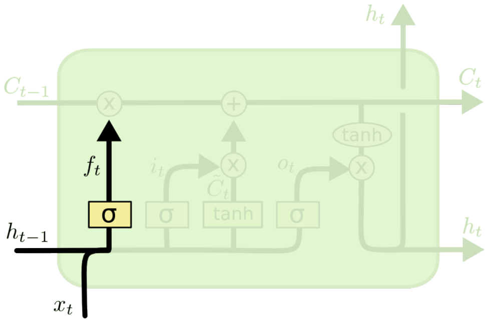




Credit: Christopher Olah

LSTMs Explained

Forget gate: controls what is kept vs what is forgotten, from previous cell state

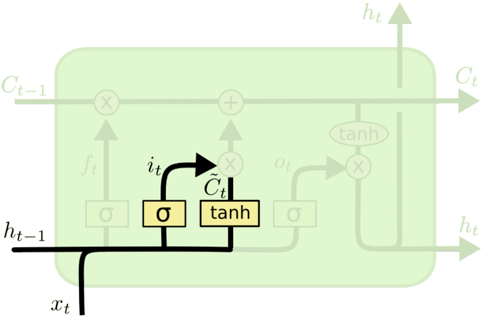


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Credit: Christopher Olah

LSTMs Explained

Input gate: decides what information to throw away from the cell state
Cell content: new content to be written to cell



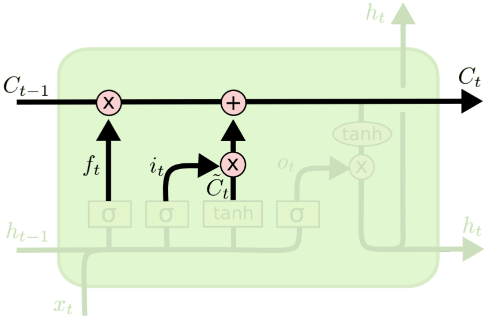
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

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

Credit: Christopher Olah

LSTMs Explained

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

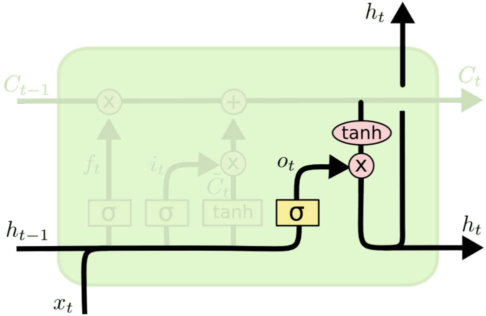


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Credit: Christopher Olah

LSTMs Explained

Output gate: controls what parts of cell are output to hidden state
Hidden state: read ("output") some content from cell



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

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

Credit: Christopher Olah

LSTMs Explained

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

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

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh(C_t)$$

LSTMs Explained

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

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

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh(C_t)$$

- What can you tell about cell state(C_t), if forget gate is set to 1 and input gate set to 0?

LSTMs Explained

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

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

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- What can you tell about cell state(C_t), if forget gate is set to 1 and input gate set to 0?
  - Information of that cell is **preserved indefinitely**

LSTMs Explained

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

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

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- What can you tell about cell state(C_t), if forget gate is set to 1 and input gate set to 0?
  - Information of that cell is **preserved indefinitely**
- What happens if you fix input gate to all 1s, forget gate to all 0s, output gate to all 1s?

LSTMs Explained

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

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

New cell content: this is the new content to be written to the cell

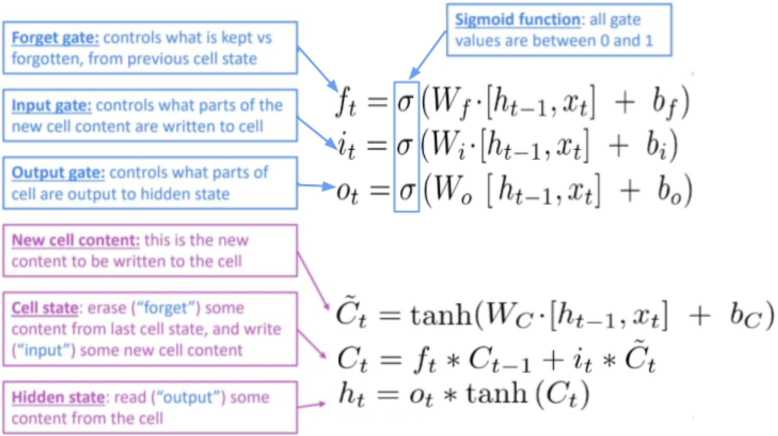
Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh(C_t)$$

- What can you tell about cell state(C_t), if forget gate is set to 1 and input gate set to 0?
  - Information of that cell is **preserved indefinitely**
- What happens if you fix input gate to all 1s, forget gate to all 0s, output gate to all 1s?
  - Almost **standard RNN**;

LSTMs Explained



- What can you tell about cell state(C_t), if forget gate is set to 1 and input gate set to 0?
  - Information of that cell is **preserved indefinitely**
- What happens if you fix input gate to all 1s, forget gate to all 0s, output gate to all 1s?
  - Almost **standard RNN**; Why almost?

LSTMs Explained

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

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

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

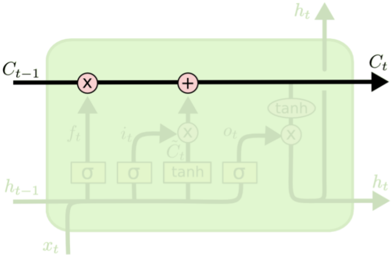
Hidden state: read ("output") some content from the cell

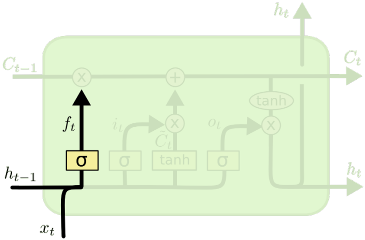
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh(C_t)$$

Credit: Christopher Olah; Christopher Manning, Stanford University

- What can you tell about cell state(C_t), if forget gate is set to 1 and input gate set to 0?
  - Information of that cell is **preserved indefinitely**
- What happens if you fix input gate to all 1s, forget gate to all 0s, output gate to all 1s?
  - Almost **standard RNN**; Why almost?
  - **Tanh** added here

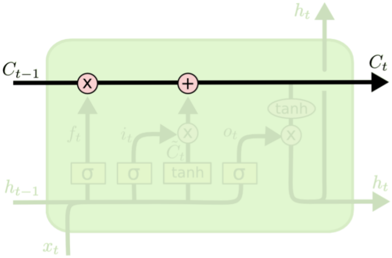
LSTM: How does it solve the vanishing gradient problem?

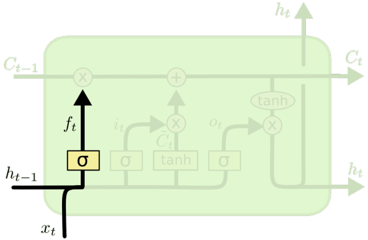




- Gradient "highway"

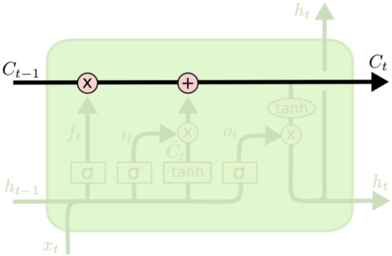
LSTM: How does it solve the vanishing gradient problem?

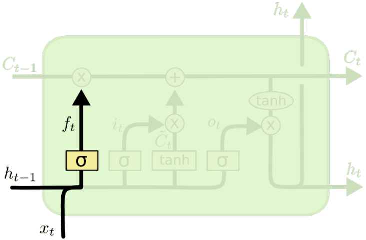




- Gradient “highway”
- Gradient at C_t passed on to C_{t-1} unaffected by any other operations, but for forget gate; why does this not matter?

LSTM: How does it solve the vanishing gradient problem?

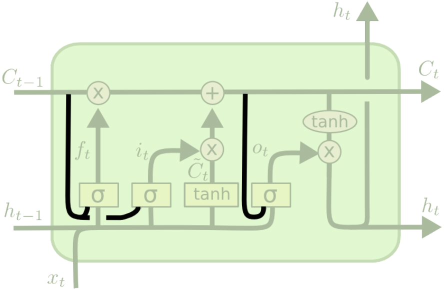




- Gradient "highway"
- Gradient at C_t passed on to C_{t-1} unaffected by any other operations, but for forget gate; why does this not matter?
- Forget gate is part of the design, it reduces the gradient where it should, does not ameliorate the gradient otherwise!

Variants of LSTM

- **LSTM with peephole connections**[2]



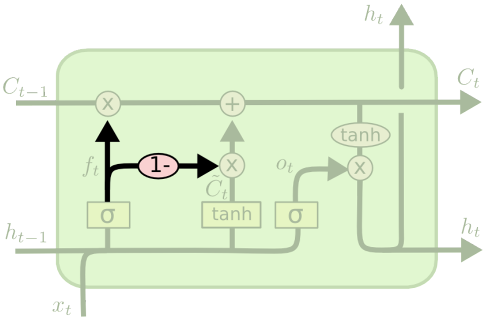
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Credit: Christopher Olah

[2] Gers and Schmidhuber, Recurrent nets that time and count, IJCNN 2000

Variants of LSTM

- **Coupled forget and input gates**
  - Instead of separately deciding what to **forget** and what to **add**, make decisions together



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

Credit: Christopher Olah

History of LSTMs

- **1997 - RTRL + BPTT (No forget gate)**
  - Hochreiter and Schmidhuber, Long Short- Term Memory, Neural Computation, 1997

History of LSTMs

- **1997 - RTRL + BPTT (No forget gate)**
  - Hochreiter and Schmidhuber, Long Short- Term Memory, Neural Computation, 1997
- **1999 – Introduced forget gate**
  - Gers Schmidhuber and Cummins, Learning to forget: Continual prediction with LSTM, ICANN 1999

History of LSTMs

- **1997 - RTRL + BPTT (No forget gate)**
  - Hochreiter and Schmidhuber, Long Short- Term Memory, Neural Computation, 1997
- **1999 – Introduced forget gate**
  - Gers Schmidhuber and Cummins, Learning to forget: Continual prediction with LSTM, ICANN 1999
- **2000 – Peephole connections**
  - Gers and Schmidhuber, Recurrent nets that time and count, IJCNN 2000

History of LSTMs

- **1997 - RTRL + BPTT (No forget gate)**
  - Hochreiter and Schmidhuber, Long Short- Term Memory, Neural Computation, 1997
- **1999 – Introduced forget gate**
  - Gers Schmidhuber and Cummins, Learning to forget: Continual prediction with LSTM, ICANN 1999
- **2000 – Peephole connections**
  - Gers and Schmidhuber, Recurrent nets that time and count, IJCNN 2000
- **2005 – Vanilla LSTM (as we know today) – Used BPTT**
  - Graves and Schmidhuber, Framewise phoneme classification with bidirectional LSTM and other neural network architectures, Neural Networks, 2005

LSTMs: Real-world success

- **2013-2015**: LSTMs started achieving **state-of-the-art results**
  - Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning

LSTMs: Real-world success

- **2013-2015**: LSTMs started achieving **state-of-the-art results**
  - Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning
- **Now (2020)**, other approaches (e.g. **Transformers**) have become more dominant for certain tasks
  - Transformers use the idea of self-attention
  - In **WMT 2019** ((a MT conference + competition), summary report contains "RNN" 7 times, "Transformer" 105 times

Credit: Christopher Manning, Stanford University

Gated Recurrent Unit (GRU)[2]

- Proposed in 2014 as a simpler alternative to LSTM

[2]Chung et al, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, NeurIPS-W 2014

Gated Recurrent Unit (GRU)[2]

- Proposed in 2014 as a simpler alternative to LSTM
- Combines forget and input gates into a single **update gate**

[2]Chung et al, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, NeurIPS-W 2014

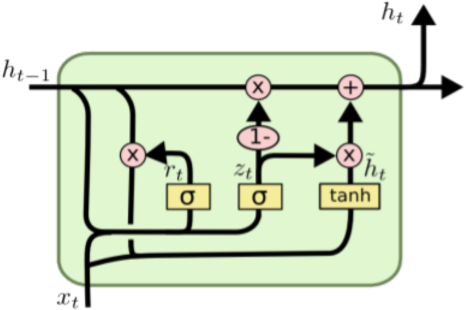
Gated Recurrent Unit (GRU)[2]

- Proposed in 2014 as a simpler alternative to LSTM
- Combines forget and input gates into a single **update gate**
- Merges cell state and hidden state

[2] Chung et al, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, NeurIPS-W 2014

Gated Recurrent Unit (GRU)[2]

- Proposed in 2014 as a simpler alternative to LSTM
- Combines forget and input gates into a single **update gate**
- Merges cell state and hidden state



[2] Chung et al, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, NeurIPS-W 2014

Gated Recurrent Unit (GRU)[2]

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

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

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

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

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

[2] Chung et al, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, NeurIPS-W 2014

Gated Recurrent Unit (GRU)[2]

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

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

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

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

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

- What happens if reset gate is set to all 1s and update gate to all 0s?

[2] Chung et al, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, NeurIPS-W 2014

GRUs vs LSTMs: Summary

- Input and forget gates of LSTMs are coupled by an update gate in GRUs; reset gate (GRUs) is applied directly to previous hidden state

GRUs vs LSTMs: Summary

- Input and forget gates of LSTMs are coupled by an update gate in GRUs; reset gate (GRUs) is applied directly to previous hidden state
- GRU has two gates, an LSTM has three gates; what does this tell you?

GRUs vs LSTMs: Summary

- Input and forget gates of LSTMs are coupled by an update gate in GRUs; reset gate (GRUs) is applied directly to previous hidden state
- GRU has two gates, an LSTM has three gates; what does this tell you? **Lesser parameters to learn!**

GRUs vs LSTMs: Summary

- Input and forget gates of LSTMs are coupled by an update gate in GRUs; reset gate (GRUs) is applied directly to previous hidden state
- GRU has two gates, an LSTM has three gates; what does this tell you? **Lesser parameters to learn!**
- In GRUs:
  - No internal memory (c_t) different from exposed hidden state
  - No output gate as in LSTMs

GRUs vs LSTMs: Summary

- Input and forget gates of LSTMs are coupled by an update gate in GRUs; reset gate (GRUs) is applied directly to previous hidden state
- GRU has two gates, an LSTM has three gates; what does this tell you? **Lesser parameters to learn!**
- In GRUs:
  - No internal memory (c_t) different from exposed hidden state
  - No output gate as in LSTMs
- LSTM a good default choice (especially if data has long-range dependencies, or if training data is large); Switch to GRUs for speed and fewer parameters

Homework

Readings

- Deep Learning book: Sections 10.1-10.7, 10.10-10.11
- Understanding LSTM Networks
- Illustrated Guide to LSTMs and GRUs: A step by step explanation
- (Optional) Recurrent Neural Network Tutorial– Implementing a GRU/LSTM RNN with Python and Theano
- (Optional) Training LSTMs using BPTT: Alex Graves' book on RNN (Sec 4.6, pg 36-38)

Questions

- How does GRU address vanishing gradients?

References

[1] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997), pp. 1735–1780.

[2] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *ICML*. 2013.

[3] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555.

[4] *Li, Fei-Fei; Johnson, Justin; Serena, Yeung; CS 231n - Convolutional Neural Networks for Visual Recognition (Spring 2019)*. URL: http://cs231n.stanford.edu/2019/ (visited on 08/28/2020).

[5] *Manning, Christopher, CS 224n Natural Language Processing with Deep Learning (Winter 2019)*. URL: http://cs224n.stanford.edu/ (visited on 08/28/2020).