M2177.003100
Deep Learning

[10: Recurrent Neural Nets (Part 2)]

Electrical and Computer Engineering
Seoul National University

(last compiled at 21:08:00 on 2019/11/03)

Outline

References

- *Deep Learning* by Goodfellow, Bengio and Courville ▸ Link
  - Chapter 10 Sequence Modeling: Recurrent and Recursive Nets
- online resources:
  - *Understanding LSTM Networks* ▸ Link
  - *The Unreasonable Effectiveness of RNNs* ▸ Link
  - *The fall of RNN/LSTM* ▸ Link
  - *Stanford CS224n: NLP with Deep Learning* ▸ Link
  - *Deep Learning Specialization (coursera)* ▸ Link
  - *Stanford CS231n: CNN for Visual Recognition* ▸ Link

Challenges in RNN training

- if well trained, RNN can learn dependencies across hundreds of steps!
  - but very hard to ____ in practice
- basic problem: gradients propagated over many stages tend to either
  - vanish (most of the time) or
  - explode (rarely, but with much damage to optimization)
- difficulty with long-term dependencies arises from
  - exponentially smaller weights given to long-term interactions
- example: predicting the next word

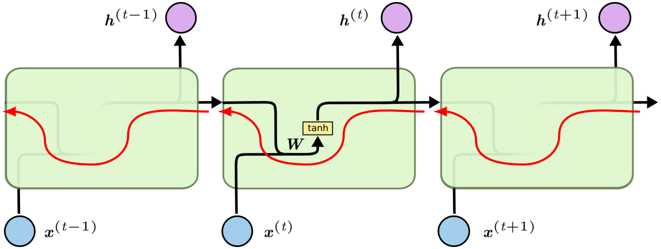
  (a) `Jane walked into the room. John walked in too. Jane said hi to __________.`

  (b) `Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to __________.`

(source: Socher)

Gradient flow in vanilla RNN

- backprop of gradient through the $\boldsymbol{h}$ path involves
  - many factors of $\boldsymbol{W}$ and repeated tanh



(source: cs231n, Olah)

- repeated multiplication of the ____ $\boldsymbol{W}$
  - largest singular value $> 1 \Rightarrow$ exploding gradients
  - largest singular value $< 1 \Rightarrow$ vanishing gradients

Analyzing recurrence relation

- consider recurrence relation (no activation/input for simplicity)

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^\top \boldsymbol{h}^{(t-1)}$$

  - simplified to

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^t\right)^\top \boldsymbol{h}^{(0)}$$

- if $\boldsymbol{W}$ admits an eigendecomposition (with orthogonal $\boldsymbol{Q}$)

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$$

  - the recurrence may be simplified further to

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^\top \boldsymbol{\Lambda}^t \boldsymbol{Q}\boldsymbol{h}^{(0)}$$

- each eigenvalue λ: raised to the power of t
  - if $|\lambda| < 1 \Rightarrow$ decay to zero
  - if $|\lambda| > 1 \Rightarrow$ explode

- this problem: particular to RNN (assume scalar weight w for simplicity)

  (1) RNN: multiply the same weight w by itself many times
  - product: w^t

  $\Rightarrow$ vanish or explode depending on magnitude of w

  (2) non-RNN: use ______ weight $w^{(t)}$ at each time step
  - product: $\prod_t w^{(t)}$

  $\Rightarrow$ careful scaling can avoid vanishing/exploding[1] to some extent

[1] *e.g.* random $w^{(t)}$ (0 mean, variance v) $\rightarrow$ variance of product: $O(v^n) \rightarrow$ set $v = \sqrt[n]{v^*}$ (v^*: desired bias)

Implications

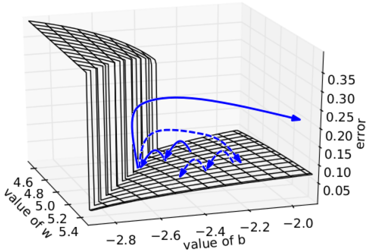
- exploding gradient
  - easy to detect $\because \underbrace{\qquad\qquad}_{\substack{\uparrow \\ \text{training cannot continue}}}$ in gradient computation
  - a quick solution: gradient clipping
- vanishing gradient
  - can go undetected while drastically $\underbrace{\text{hurting training}}_{\substack{\uparrow \\ \text{both learning quality and speed}}}$
  - heuristic solutions exist (*e.g.* IRNN)
  - more general solution: need to change RNN architecture

Gradient clipping

- a simple solution to exploding gradient
  - clip gradients to a _____ number whenever they explode

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)                # norm clipping
```

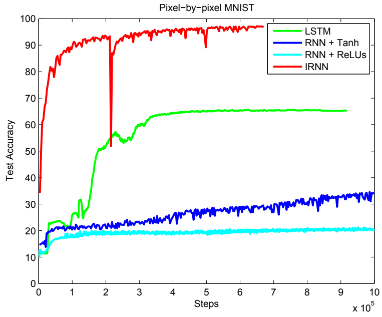
- example: effect of gradient clipping
  - error surface of a single hidden unit RNN
  - solid lines: standard sgd
  - dashes: sgd with clipping

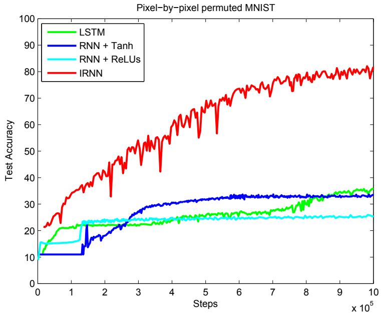


when the standard gradient descent model hits the high error wall, the gradient is pushed off to a faraway location on the decision surface; the clipping model instead pulls back the error gradient to somewhat close to the original gradient landscape (source: Pascanu, 2013)

IRNN (Le, Jaitly & Hinton, 2015)

- a heuristic to handle vanishing gradient
  1. initialize $\boldsymbol{W}$ to $\boldsymbol{I}$ (instead of randomly)
  2. use _____ (instead of sigmoid)





Outline

Gated RNNs

- very effective sequence models used in practical applications

  e.g. long short-term memory (LSTM)

  gated recurrent unit (GRU)

- create paths through time
  - these paths have derivatives that neither vanish nor explode
  - connection weights may ______ at each time step
    - not directly but through the action of gates

$$\underbrace{\boldsymbol{W}}_{\text{time-independent}} \rightarrow \underbrace{\text{gates}}_{\text{time-varying}} \rightarrow \text{gradient flow control}$$

$\Rightarrow$ no repeated multiplication by the same $\boldsymbol{W}$ any more

Idea

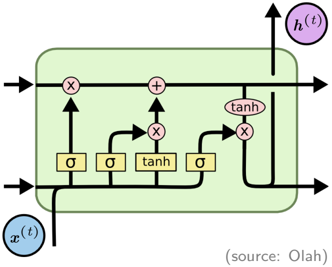
- use some units to allow the network to *accumulate* information
    - (*e.g.* evidence for a particular feature or category) over a long duration
- however, once that information has been used
    - it might be useful for the neural net to _____ the old state
- instead of manually deciding when to clear the state
    - we want the neural net to learn to decide when to do it

$\Rightarrow$ this is what gated RNNs do!

Outline

LSTM (Hochreiter and Schmidhuber, 1997)

- a core contribution of initial LSTM:
  - use self-loops to produce paths where gradient can flow for long durations
- a crucial addition: make weight on this self-loop
  - conditioned on the context rather than fixed
  - "_____" (*i.e.* controlled by another hidden unit)
  - gating action changes dynamically (based on input)
- extremely successful in many applications:
  - handwriting recognition
  - speech recognition
  - handwriting generation
  - machine translation
  - image captioning
  - many more!



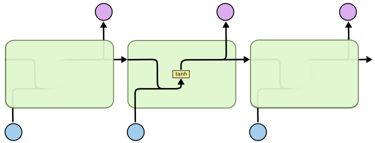
(source: Olah)

LSTM networks

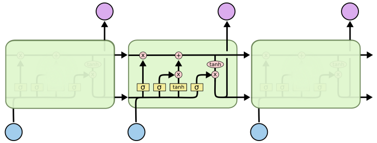
- explicitly designed to avoid long-term dependency problem
  - remembering info for long time: practically their ______ behavior

- standard RNNs: a chain of $\underbrace{\text{repeating modules}}_{\substack{\uparrow \\ \text{very simple structure (single tanh layer)}}}$

- LSTMs: also a chain of $\underbrace{\text{repeating modules}}_{\substack{\uparrow \\ \text{more complex (four interacting network layers)}}}$



standard RNN

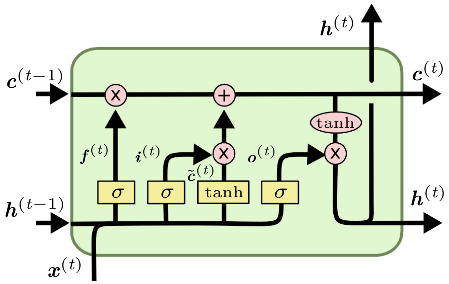


LSTM net

(source: Olah)

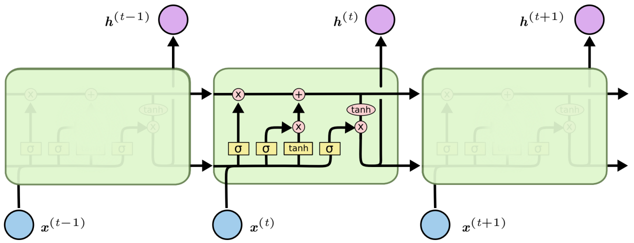
LSTM cells

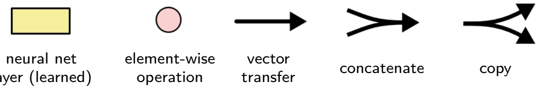
- LSTM RNNs consists of "$\underbrace{\text{LSTM cells}}_{\uparrow}$"
  - have an internal recurrence (a self-loop) besides the outer recurrence
- each LSTM cell: has the same inputs/outputs as in standard RNN
  - but has more parameters and $\underbrace{\text{a system of gating units}}_{\substack{\uparrow \\ \text{controls the flow of information}}}$



- most important component
  - ___ state $\boldsymbol{c}^{(t)}$ (a vector!)

Notation and conventions





- each line carries a vector
- $\sigma(\cdot)$: for controlling gates $(0 \leq \sigma \leq 1)$
- $\tanh(\cdot)$: for representing $(-1 \leq \tanh \leq 1)$
  - hidden state (as in vanilla RNN) and increase/decrease in cell state[2]

[2]each element itself in the cell state can be greater than $+1$ or smaller than -1

- simplified notation
  - combine hidden-hidden and input-hidden weight matrices

$$\begin{aligned}\boldsymbol{h}^{(t)} &= g(\boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_{hx}\boldsymbol{x}^{(t)} + \boldsymbol{b}_h) \\ &\triangleq g(\boldsymbol{W}_h[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_h)\end{aligned}$$

  where

$$\boldsymbol{W}_h \triangleq \begin{bmatrix} \boldsymbol{W}_{hh} & \boldsymbol{W}_{hx} \end{bmatrix} \in \mathbb{R}^{n_h \times (n_h + n_x)}$$

$$[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] \triangleq \begin{bmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t)} \end{bmatrix} \in \mathbb{R}^{(n_h + n_x) \times 1}$$

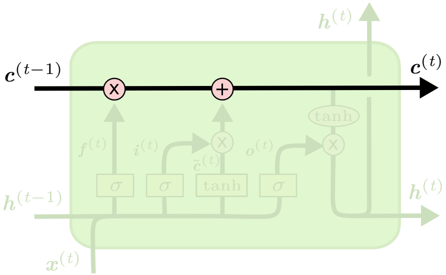
  and

$$\begin{aligned}\boldsymbol{W}_h[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] &= \begin{bmatrix} \boldsymbol{W}_{hh} & \boldsymbol{W}_{hx} \end{bmatrix} \begin{bmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t)} \end{bmatrix} \\ &= \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_{hx}\boldsymbol{x}^{(t)}\end{aligned}$$

Core idea behind LSTM

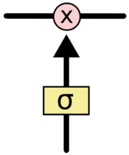
- key: cell state
  - ▸ like a conveyor belt
  - ▸ runs down the entire chain (with only minor linear interactions)
  - ⇒ information easily can just flow along the chain unchanged

- each element in cell state
  - ≈ scalar integer ______
  - ▸ incremented/decremented
    by up to one
    at each time step

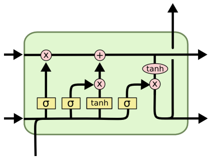


- LSTM can remove/add information to cell state
  - ▸ carefully regulated by structures called **gates**

- gates: a way to optionally let information through
  - composed of
    - a sigmoid neural net layer
    - an element-wise multiplication operation



$\Leftarrow$



- the sigmoid layer
  - describes how much of each element should be let through
  - outputs a number between zero and one

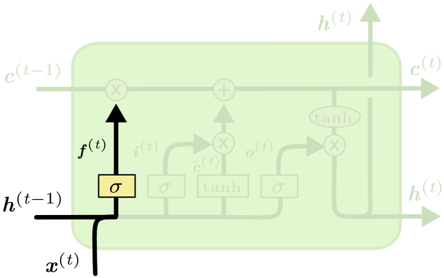
$$\underbrace{0}_{\text{“let nothing through”}} \quad \le \quad \text{output} \quad \le \quad \underbrace{1}_{\text{“let everything through”}}$$

- standard LSTM: has ____ gates (forget, input, output)

Step 1

- decide what information to throw away from cell state
  - implemented by a sigmoid layer called $\boxed{\qquad\text{gate}}$
- forget gate: for each number in cell state $\boldsymbol{c}^{(t-1)}$
  - looks at $\boldsymbol{h}^{(t-1)}$ and $\boldsymbol{x}^{(t)}$
  - outputs a number between zero and one

$$\underbrace{0}_{\text{“completely get rid of”}} \leq \quad \text{output} \quad \leq \underbrace{1}_{\text{“compeletely keep”}}$$



$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{W}_f[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_f)$$

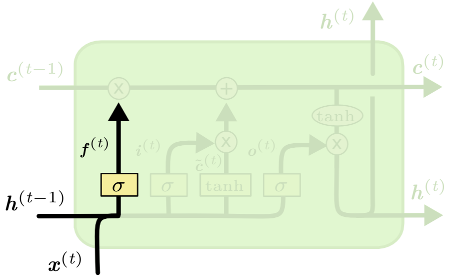
$$0 \leq \boldsymbol{f} \leq 1 \qquad \text{(gate)}$$

example: language modeling

- predict the next word based on all the previous ones

- cell state
  - might include the gender of the present subject
  - so that the correct pronouns can be used

- when we see a new subject
  - we want to forget the gender of the old subject



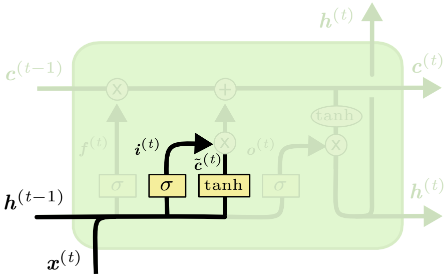
$$f^{(t)} = \sigma(W_f[h^{(t-1)}, x^{(t)}] + b_f)$$

$$0 \le f \le 1 \qquad \text{(gate)}$$

Step 2

- decide what new information to store in cell state in two substeps
  - (i) a $\underbrace{\text{sigmoid layer}}_{\substack{\uparrow \\ \text{called } \boxed{\text{ gate}}}}$ decides which values to update
  - (ii) a tanh layer creates a vector of $\underbrace{\text{new candidate values}}_{\substack{\uparrow \\ \text{called } \tilde{\boldsymbol{c}}^{(t)}\text{, which could be added to the state}}}$

  ⋇ step 3 will combine these two to create an update to the state

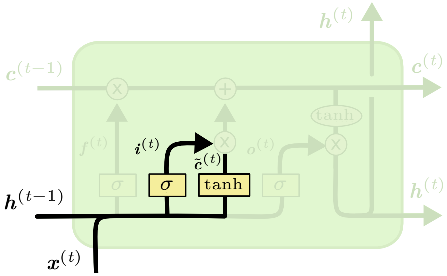


$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_i[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_i)$$
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(\boldsymbol{W}_c[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_c)$$

$$0 \le \boldsymbol{i} \le 1 \qquad \text{(gate)}$$
$$-1 \le \tilde{\boldsymbol{c}} \le 1 \qquad \text{(in/decrease)}$$

example: language modeling (continued)

- we would want to add the gender of the new subject to cell state
  - to replace the old one we are forgetting



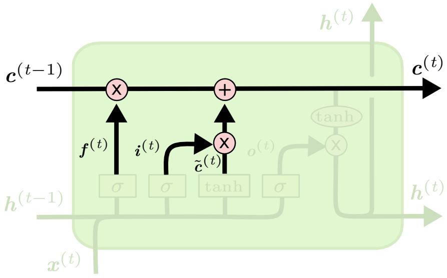
$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_i[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_i)$$
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(\boldsymbol{W}_c[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_c)$$

$0 \le \boldsymbol{i} \le 1$ (gate)

$-1 \le \tilde{\boldsymbol{c}} \le 1$ (in/decrease)

Step 3

- now it's time to update old cell state $\boldsymbol{c}^{(t-1)}$ into new cell state $\boldsymbol{c}^{(t)}$
  - step 2 already decided what to do → we just need to actually do it now



$$\underbrace{\boldsymbol{c}^{(t)}}_{\text{new state}} = \overbrace{\boldsymbol{f}^{(t)}}^{\text{how much to forget}} \odot \underbrace{\boldsymbol{c}^{(t-1)}}_{\text{old state}} + \overbrace{\boldsymbol{i}^{(t)}}^{\text{how much to update}} \odot \underbrace{\tilde{\boldsymbol{c}}^{(t)}}_{\text{new candidate values}}$$

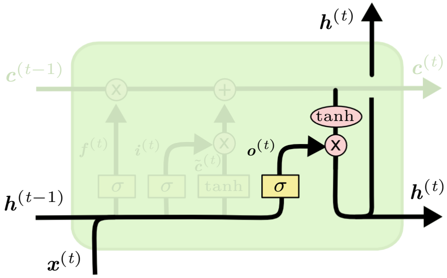
$$\begin{aligned} 0 \le \boldsymbol{f}, \boldsymbol{i} \le 1 &\qquad \text{(gates)} \\ -1 \le \tilde{\boldsymbol{c}} \le 1 &\qquad \text{(in/decrease)} \end{aligned}$$

<u>example: language modeling (continued)</u>

- we would actually drop the info about the old subject's gender
  - and add the new info as decided in step 2

Step 4

- decide what to output $\leftarrow$ based on cell state, but will be a filtered version
  (i) run a $\underbrace{\text{sigmoid layer}}_{\uparrow}$ to decide what parts of cell state to output
  called $\boxed{\ \text{gate}}$
  (ii) put cell state through tanh and multiply it by output gate
  $\Rightarrow$ we only output the parts we decided to



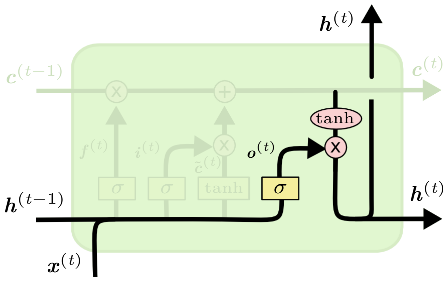
$$o^{(t)} = \sigma(W_o[h^{(t-1)}, x^{(t)}] + b_o)$$
$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

$0 \le o \le 1$ (gate)
$-1 \le h \le 1$ (hidden state)

example: language modeling (continued)

- the model just saw a subject
  - it might want to output info relevant to a matching verb

e.g. the subject is singular/plural

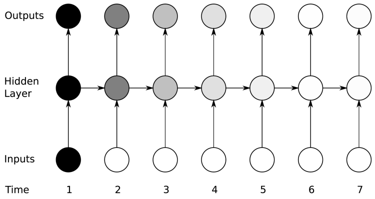


$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_o[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_o)$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)})$$

$$0 \le \boldsymbol{o} \le 1 \quad \text{(gate)}$$
$$-1 \le \boldsymbol{h} \le 1 \quad \text{(hidden state)}$$

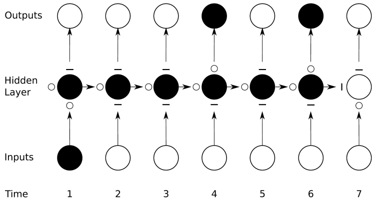
Comparison

- vanishing gradient (RNN)



The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network forgets the first inputs.

- preservation of gradient (LSTM)

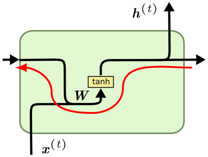


The black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are displayed below, to the left and above the hidden layer respectively. For simplicity, all gates are either entirely open (0) or closed (–). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

source: Graves (2012)

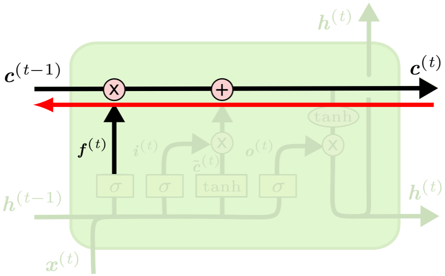
Gradient flow in LSTM

- backpropagation from $\boldsymbol{c}^{(t)}$ to $\boldsymbol{c}^{(t-1)}$ involves
  - only element-wise multiplication by $\boldsymbol{f}$
  - no matrix multiplication by $\boldsymbol{W}$



c.f. vanilla RNN

- much nicer than standard RNN for two reasons
  1. element-wise multiplication: more efficient than full matrix multiplication

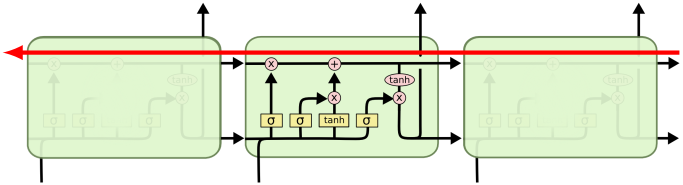


  2. multiplication by $\underbrace{\text{forget gate}}_{\uparrow}$
     potentially ______ values at every time step

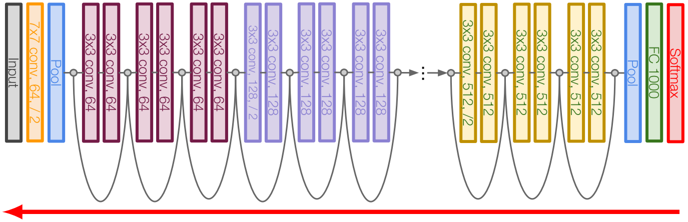
c.f. vanilla RNN: same $\boldsymbol{W}$

$\Rightarrow$ more likely to have vanishing/exploding gradient

- backprop through cell state: gradient super highway

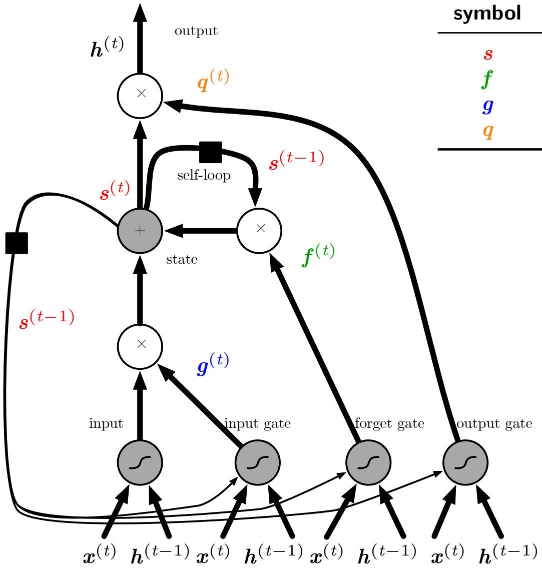


- similar to the effect of skip connections in ResNet



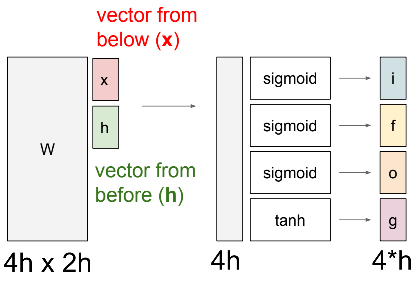
Alternative presentations

- textbook



symbol	role
$\boldsymbol{s}$	cell state
$\boldsymbol{f}$	forget gate
$\boldsymbol{g}$	input gate
$\boldsymbol{q}$	output gate

- cs231n



$$\boxed{\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

symbol	**name**	**role**
$\boldsymbol{i}$	input gate	whether to write to cell
$\boldsymbol{f}$	forget gate	whether to erase cell
$\boldsymbol{o}$	output gate	how much to reveal cell
$\boldsymbol{g}$	gate gate	how much to write to cell (*i.e.* new ________ values $\tilde{\boldsymbol{c}}^{(t)}$)

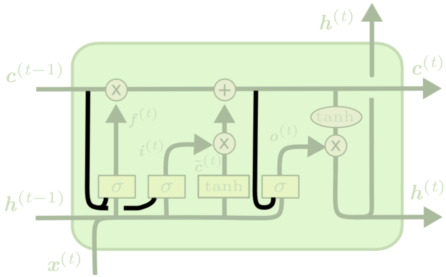
Outline

Peephole connections

- make the gate layers look at cell state

  i.e. use cell state as extra input to gates

  - need additional __________



$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{W}_f[\boldsymbol{c}^{(t-1)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_f)$$

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_i[\boldsymbol{c}^{(t-1)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_i)$$

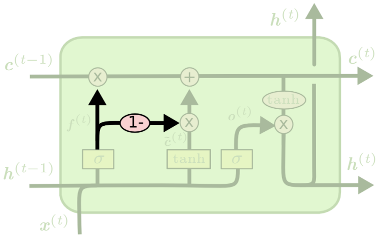
$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_o[\boldsymbol{c}^{(t)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_o)$$

  - actual peephole connections may vary from architecture to architecture (*e.g.* see page 32)

Coupling forget/input gates

- previously: separate decisions about $\underbrace{\text{what to forget}}_{\substack{\uparrow \\ \text{forget gate}}}$ and $\underbrace{\text{what to update}}_{\substack{\uparrow \\ \text{input gate}}}$

- now: make these decisions _______
  - how much to update = how much to not forget



- recall: cell state update

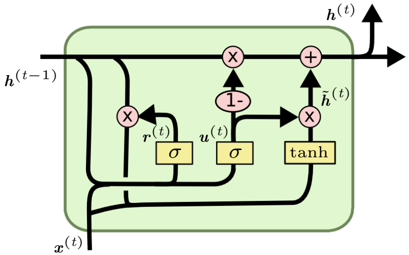
$$\underbrace{\boldsymbol{c}^{(t)}}_{\substack{\text{new}\\\text{state}}} = \overbrace{\boldsymbol{f}^{(t)}}^{\substack{\text{how much}\\\text{to forget}}} \odot \underbrace{\boldsymbol{c}^{(t-1)}}_{\substack{\text{old}\\\text{state}}} + \overbrace{\boldsymbol{i}^{(t)}}^{\substack{\text{how much}\\\text{to update}}} \odot \underbrace{\tilde{\boldsymbol{c}}^{(t)}}_{\substack{\text{new}\\\text{candidate}\\\text{values}}}$$

- coupled gating

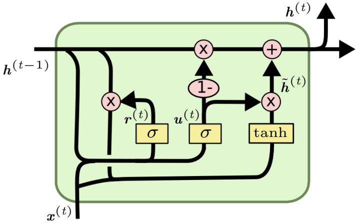
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + (1 - \boldsymbol{f}^{(t)}) \odot \tilde{\boldsymbol{c}}^{(t)}$$

Gated recurrent unit (GRU) (Cho et al., 2014)

- major changes over LSTM: simplification
  - combines forget and input gates into a single "update gate"
  - introduces "reset gate"
  - merges cell state and hidden state into one state
- potential advantages over LSTM
  - fewer __________ $\Rightarrow$ sometimes easier training/better performance



$$\boldsymbol{u}^{(t)} = \sigma(\boldsymbol{W}_u[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}])$$
$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}_r[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}])$$
$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{W}[\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}])$$
$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \odot \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \odot \tilde{\boldsymbol{h}}^{(t)}$$



$$\boldsymbol{u}^{(t)} = \sigma(\boldsymbol{W}_u[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}]) \tag{1}$$

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}_r[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}]) \tag{2}$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{W}[\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}]) \tag{3}$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \odot \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \odot \tilde{\boldsymbol{h}}^{(t)} \tag{4}$$

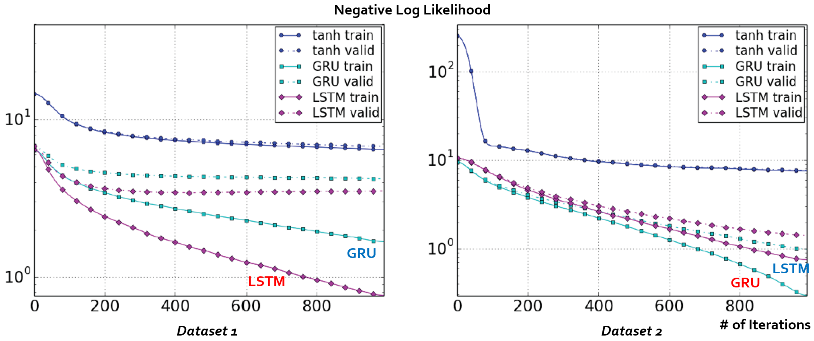
eq	**meaning**
(1)	update gate $\boldsymbol{u}^{(t)}$ is learned from old state $\boldsymbol{h}^{(t-1)}$ and current input $\boldsymbol{x}^{(t)}$
(2)	reset gate $\boldsymbol{r}^{(t)}$ is learned from old state $\boldsymbol{h}^{(t-1)}$ and current input $\boldsymbol{x}^{(t)}$
(3)	• new candidate value $\tilde{\boldsymbol{h}}^{(t)}$ reflects old state $\boldsymbol{h}^{(t-1)}$ and current input $\boldsymbol{x}^{(t)}$ • reset gate $\boldsymbol{r}^{(t)}$ controls how much old state $\boldsymbol{h}^{(t-1)}$ is used • current input $\boldsymbol{x}^{(t)}$ is fully used regardless of $\boldsymbol{r}^{(t)}$
(4)	• current state $\boldsymbol{h}^{(t)}$ combines old state $\boldsymbol{h}^{(t-1)}$ and new candidate value $\tilde{\boldsymbol{h}}^{(t)}$ • update gate $\boldsymbol{u}^{(t)}$ controls how to ___ these two

Variants and comparisons

- many more variants of LSTM can be designed
- however, several investigations found
  - no variant would clearly beat both of LSTM/GRU across a wide range of tasks (Greff *et al.*, 2015; Jozefowicz *et al.*, 2015)
- Greff *et al.* (2015) found:
  - a crucial ingredient is the forget gate
- Jozefowicz *et al.* (2015) found:
  - adding a bias of 1 to LSTM _____ gate
    - makes LSTM as strong as the best of explored architectural variants

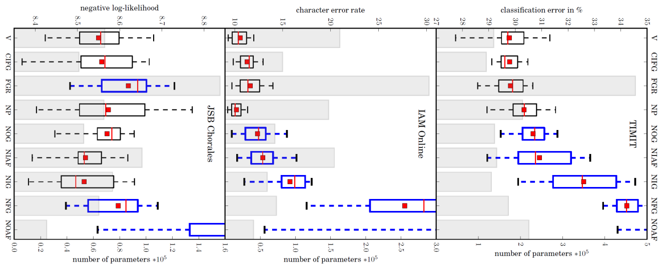
"No clear winner" (Chung, 2014)

- but GRU/LSTM-RNNs certainly outperform traditional RNNs



LSTM: a search space odyssey (Greff, 2015)

- large-scale analysis of eight LSTM variants
  - speech/handwriting recognition, polyphonic music modeling
  - 5400 experimental runs (≈ 15 years of CPU time)
- result: “no variants can improve on standard LSTM significantly”
  - most critical: ______ gate & output activation function



Toward optimal RNN architectures (Jozefowicz, 2015)

- objectives
  - determine whether LSTM is optimal or better one exists
  - better understand the role of individual components
- results from evaluating over 10,000 different architectures
  - found an architecture that outperforms LSTM/GRU
  - fine print: "on some but not all tasks"
- another finding: adding a bias of ___ to LSTM forget gate
  - closes the gap between LSTM and GRU

Outlook

- memory-augmented neural net (MANN)
  - memory networks: learn Q&A tasks
  - neural Turing machines: learn algorithms

“the fall of RNN/LSTM” ▸ Link

- critical problems of RNN/LSTM
  - often very difficult to train
  - hardware acceleration issue: memory-bandwidth bound
- (arguably better) alternatives
  - temporal convolutional networks (TCN) ▸ Link
  - pervasive attention (2D CNN for seq-to-seq prediction) ▸ Link
  - the Transformer (machine translation with self-attention; no RNN/CNN)
    - “________ is all you need” ▸ Link

Outline

Summary

- challenge of training RNN: learning long-term dependency
  - main cause: vanishing gradient and/or exploding gradient
  - heuristic solutions: gradient clipping, IRNN (limited applicability)
  - generic solution requires changes in architecture $\Rightarrow$ gated RNN

- gated RNN: long short-term memory (LSTM), gated recurrent unit (GRU)
  - create paths through time without vanishing/exploding gradients
  - gates: control how much "open" the signal flows $(0 \leq \sigma \leq 1)$
  - cell state: stores system state like a counter $(-1 \leq \tanh \leq 1)$
  - backprop path through cell state: mainly element-wise multiplication

  $\Rightarrow$ helpful to maintain strong gradient flows (similar to ResNet idea)

- further extensions and recent criticism
  - memory-augmented neural nets: learn Q&A tasks or algorithms
  - "attention is all you need"