



LG Advanced Data Scientists Program

Deep Learning

[3: Recurrent Neural Nets (Part 2)]

Prof. Sungroh Yoon

Electrical & Computer Engineering | Seoul National University

© 2020 Sungroh Yoon. this material is for educational uses only. some contents are based on the material provided by other paper/book authors and may be copyrighted by them.

(last compiled at 21:21:00 on 2020/02/09)

Outline

Challenge: Learning Long-Term Dependencies

Gated Recurrent Neural Networks

Summary

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](#)

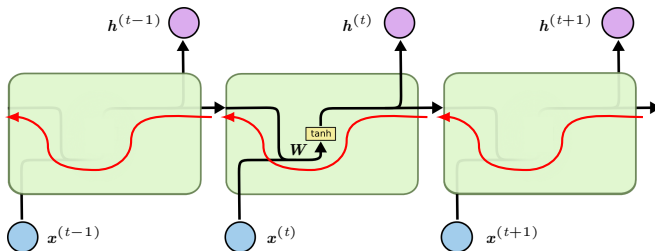
Challenges in RNN training

- 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 h path involves
 - ▶ many factors of W and repeated \tanh



(source: cs231n, Olah)

- repeated multiplication of the ____ 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)

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

- ▶ simplified to

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^\top \mathbf{h}^{(0)}$$

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

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

- ▶ the recurrence may be simplified further to

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

- 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/exploding¹ to some extent

¹e.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 ∴  in gradient computation

↑
training cannot continue

- ▶ a quick solution: gradient clipping

- **vanishing** gradient

- ▶ can go undetected while drastically  hurting training

↑
both learning quality and speed

- ▶ heuristic solutions exist (*e.g.* IRNN)
- ▶ more general solution: need to change RNN architecture

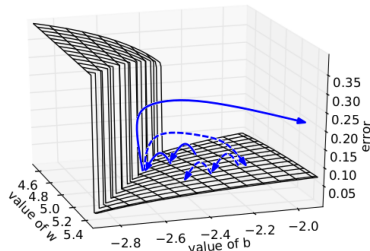
Gradient clipping

- 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

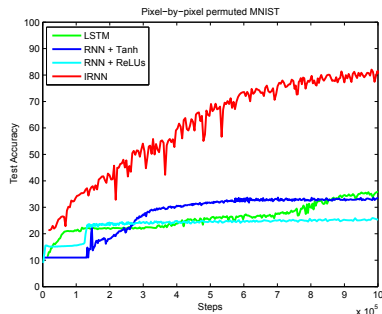
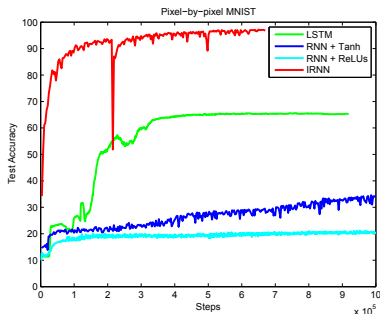
- ▶ error surface of a single hidden unit RNN
- ▶ solid lines: standard sgd
- ▶ dashes: sgd with clipping



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 W to I (instead of randomly)
 2. use _____ (instead of sigmoid)



Outline

Challenge: Learning Long-Term Dependencies

Gated Recurrent Neural Networks

Long Short-Term Memory (LSTM)

LSTM Variants

Summary

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**

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

⇒ no repeated multiplication by the **same** 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
 - ⇒ this is what gated RNNs do!

Outline

Challenge: Learning Long-Term Dependencies

Gated Recurrent Neural Networks

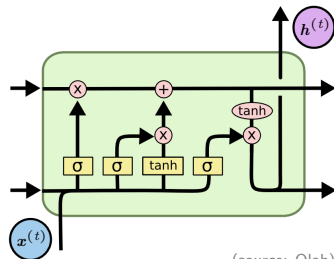
Long Short-Term Memory (LSTM)

LSTM Variants

Summary

LSTM (Hochreiter and Schmidhuber, 1997)

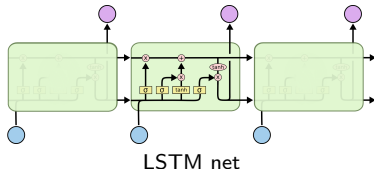
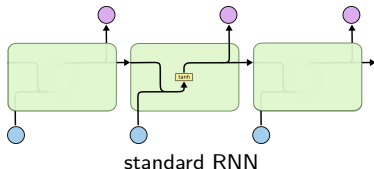
- a core contribution of initial LSTM:
 - ▶ use **self-loops** to produce paths where gradient can flow for long durations
- a crucial addition: make **weight** on this self-loop
 - ▶ conditioned on the context rather than fixed
 - ▶ “ ” (*i.e.* controlled by another hidden unit)
 - ▶ gating action changes 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

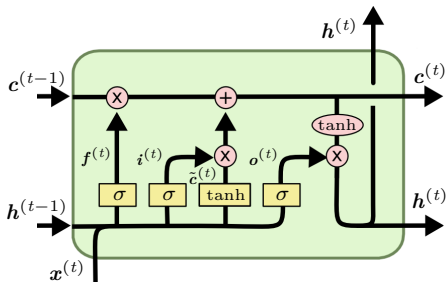
- explicitly designed to avoid long-term dependency problem
 - ▶ remembering info for long time: practically their _____ behavior
- standard RNNs: a chain of repeating modules
 - ↑
very simple structure (**single** tanh layer)
- LSTMs: also a chain of repeating modules
 - ↑
more complex (**four** interacting network layers)



(source: Olah)

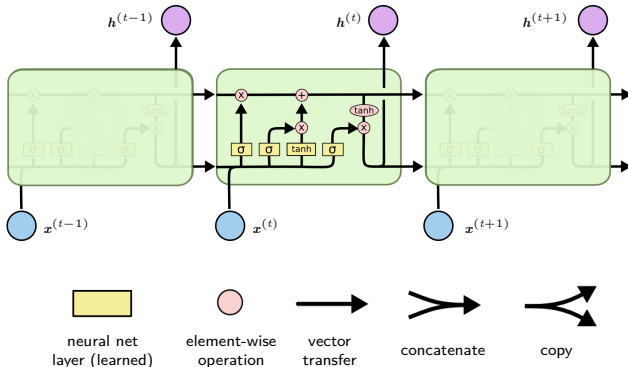
LSTM cells

- LSTM RNNs consists of “LSTM cells”
 - ▶ 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 a **system of gating units**
 - ↑
controls the flow of information



- most important component
 - ▶ **state $c^{(t)}$** (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

²each 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

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

where

$$\begin{aligned}\mathbf{W}_h &\triangleq [\mathbf{W}_{hh} \quad \mathbf{W}_{hx}] \in \mathbb{R}^{n_h \times (n_h + n_x)} \\ [\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] &\triangleq \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(t)} \end{bmatrix} \in \mathbb{R}^{(n_h + n_x) \times 1}\end{aligned}$$

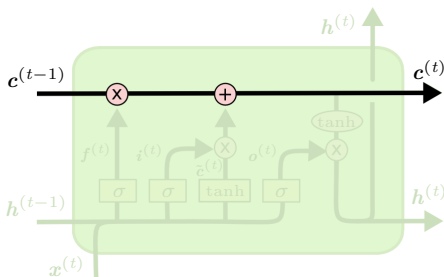
and

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

Core idea behind LSTM

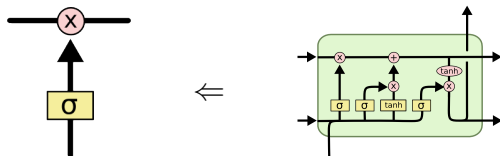
- key: **cell state**
 - ▶ like a conveyor belt
 - ▶ runs down the entire chain (with only minor linear interactions)
 - ⇒ information easily can just flow along the chain unchanged

- 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



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

$$\underbrace{0}_{\text{"let nothing through"}} \leq \text{output} \leq \underbrace{1}_{\text{"let everything through"}}$$

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

Step 1

- decide what information to **throw away** from cell state

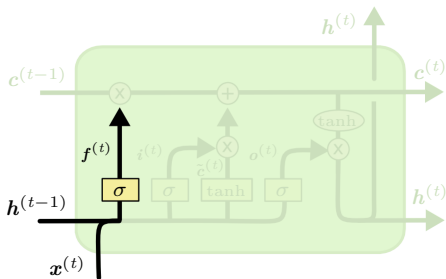
► implemented by a sigmoid layer called gate

- forget gate: for each number in cell state $c^{(t-1)}$

► looks at $h^{(t-1)}$ and $x^{(t)}$

► outputs a number between zero and one

$\underbrace{0}_{\text{"completely get rid of"}} \leq \text{output} \leq \underbrace{1}_{\text{"completely keep"}}$

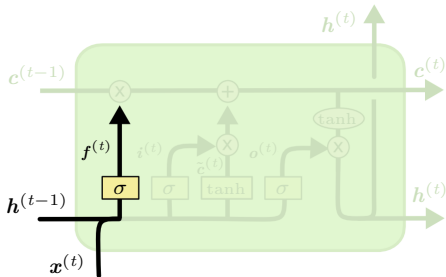


$$f^{(t)} = \sigma(W_f[h^{(t-1)}, x^{(t)}] + b_f)$$

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

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



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

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

Step 2

- decide what **new** information to store in cell state in two substeps

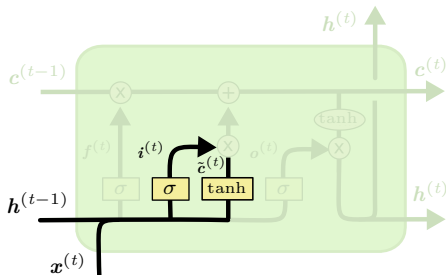
(i) a sigmoid layer decides which values to update



(ii) a tanh layer creates a vector of **new candidate values**

↑
called $\tilde{c}^{(t)}$, which could be added to the state

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



$$i^{(t)} = \sigma(W_i[h^{(t-1)}, x^{(t)}] + b_i)$$

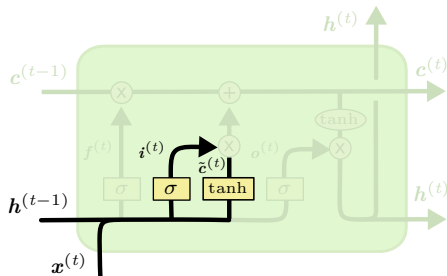
$$\tilde{c}^{(t)} = \tanh(W_c[h^{(t-1)}, x^{(t)}] + b_c)$$

$$0 \leq i \leq 1 \quad (\text{gate})$$

$$-1 \leq \tilde{c} \leq 1 \quad (\text{in/decrease})$$

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



$$i^{(t)} = \sigma(W_i[h^{(t-1)}, x^{(t)}] + b_i)$$

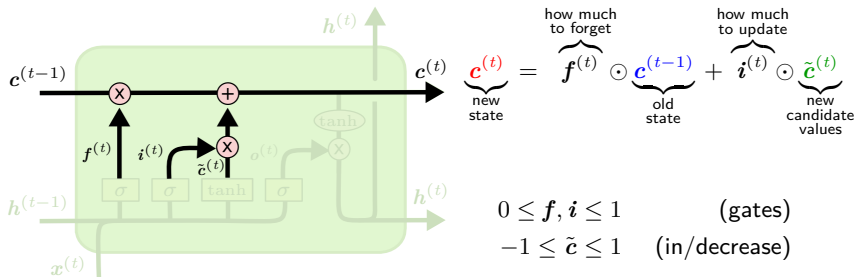
$$\tilde{c}^{(t)} = \tanh(W_c[h^{(t-1)}, x^{(t)}] + b_c)$$

$$0 \leq i \leq 1 \quad (\text{gate})$$

$$-1 \leq \tilde{c} \leq 1 \quad (\text{in/decrease})$$

Step 3

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



example: language modeling (continued)

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

Step 4

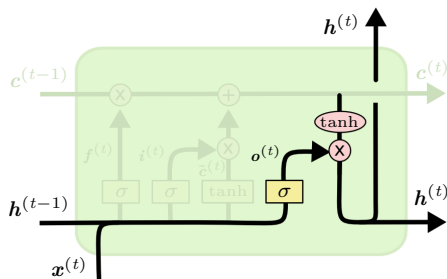
- decide what to output \leftarrow based on cell state, but will be a filtered version

(i) run a sigmoid layer to decide **what parts of cell state to output**

↑
called gate

(ii) put cell state through tanh and multiply it by output gate

\Rightarrow we only output the parts we decided to



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

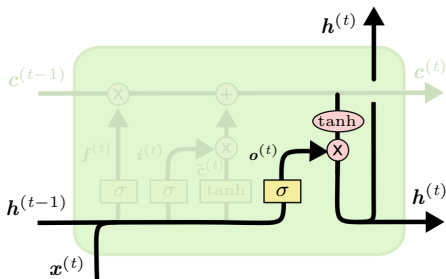
$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

$$0 \leq o \leq 1 \quad (\text{gate})$$

$$-1 \leq h \leq 1 \quad (\text{hidden state})$$

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



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

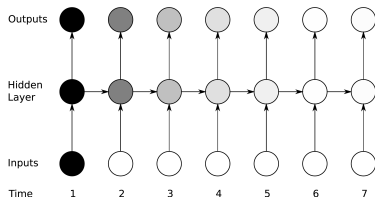
$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

$$0 \leq o \leq 1 \quad (\text{gate})$$

$$-1 \leq h \leq 1 \quad (\text{hidden state})$$

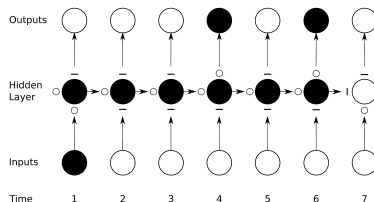
Comparison

- 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)

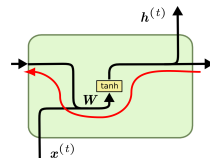
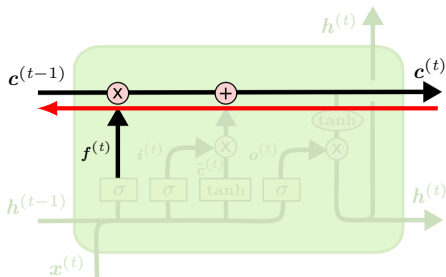
Gradient flow in LSTM

- backpropagation from $c^{(t)}$ to $c^{(t-1)}$ involves

- ▶ only **element-wise** multiplication by f
- ▶ no matrix multiplication by W

- much nicer than standard RNN for two reasons

1. element-wise multiplication: **more efficient** than full matrix multiplication



c.f. vanilla RNN

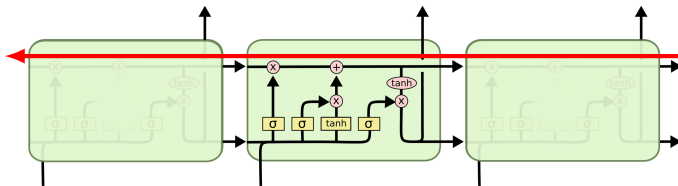
2. multiplication by **forget gate**

↑
potentially _____
values at every time step

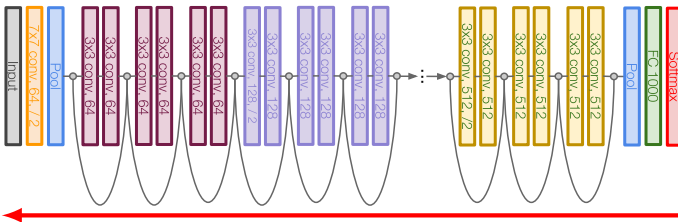
c.f. vanilla RNN: same W

⇒ more likely to have
vanishing/exploding
gradient

- backprop through **cell state**: gradient super highway



- similar to the effect of **skip connections** in ResNet



Outline

Challenge: Learning Long-Term Dependencies

Gated Recurrent Neural Networks

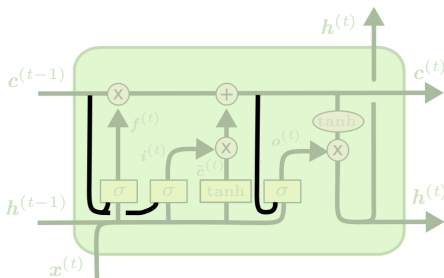
Long Short-Term Memory (LSTM)

LSTM Variants

Summary

Peephole connections

- make the gate layers look at cell state
 - i.e.* use **cell state as** extra input to gates
 - ▶ need additional _____



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

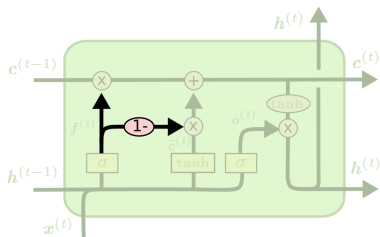
$$i^{(t)} = \sigma(W_i[\mathbf{c}^{(t-1)}, h^{(t-1)}, x^{(t)}] + b_i)$$

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

- ▶ actual peephole connections may vary from architecture to architecture

Coupling forget/input gates

- previously: separate decisions about $\underbrace{\text{what to forget}}_{\text{forget gate}}$ and $\underbrace{\text{what to update}}_{\text{input gate}}$
- now: make these decisions _____
 - ▶ how much to **update** = how much to **not forget**



- recall: cell state update

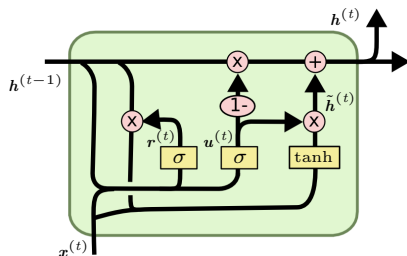
$$\underbrace{c^{(t)}}_{\text{new state}} = \underbrace{f^{(t)}}_{\text{how much to forget}} \odot \underbrace{c^{(t-1)}}_{\text{old state}} + \underbrace{i^{(t)}}_{\text{how much to update}} \odot \underbrace{\tilde{c}^{(t)}}_{\text{new candidate values}}$$

- coupled gating

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

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

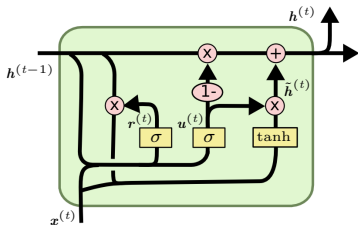


$$\mathbf{u}^{(t)} = \sigma(W_u[h^{(t-1)}, x^{(t)}])$$

$$\mathbf{r}^{(t)} = \sigma(W_r[h^{(t-1)}, x^{(t)}])$$

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

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



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

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

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

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

eq	meaning
(1)	update gate $\mathbf{u}^{(t)}$ is learned from old state $\mathbf{h}^{(t-1)}$ and current input $\mathbf{x}^{(t)}$
(2)	reset gate $\mathbf{r}^{(t)}$ is learned from old state $\mathbf{h}^{(t-1)}$ and current input $\mathbf{x}^{(t)}$
(3)	<ul style="list-style-type: none"> new candidate value $\tilde{\mathbf{h}}^{(t)}$ reflects old state $\mathbf{h}^{(t-1)}$ and current input $\mathbf{x}^{(t)}$ reset gate $\mathbf{r}^{(t)}$ controls how much old state $\mathbf{h}^{(t-1)}$ is used current input $\mathbf{x}^{(t)}$ is fully used regardless of $\mathbf{r}^{(t)}$
(4)	<ul style="list-style-type: none"> current state $\mathbf{h}^{(t)}$ combines old state $\mathbf{h}^{(t-1)}$ and new candidate value $\tilde{\mathbf{h}}^{(t)}$ update gate $\mathbf{u}^{(t)}$ controls how to ___ these two

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

Challenge: Learning Long-Term Dependencies

Gated Recurrent Neural Networks

Summary

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”