# Vanishing Gradients, LSTMs

EE6132: Deep learning for Image Processing

Content borrowed from the course CS7015 by
Dr. Mitesh Khapra

# Issues with RNNs

Consider the following sentences (task is to fill in the blank):

- Sentence 1

  "Jane walked into the room. John walked in too. Jane said hi to ___"

# Issues with RNNs

Consider the following sentences (task is to fill in the blank):

- Sentence 1

  "Jane walked into the room. John walked in too. Jane said hi to ___"

- Sentence 2

  "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 ___"

# Issues with RNNs

Consider the following sentences (task is to fill in the blank):

- Sentence 1

  "Jane walked into the room. John walked in too. Jane said hi to ___"

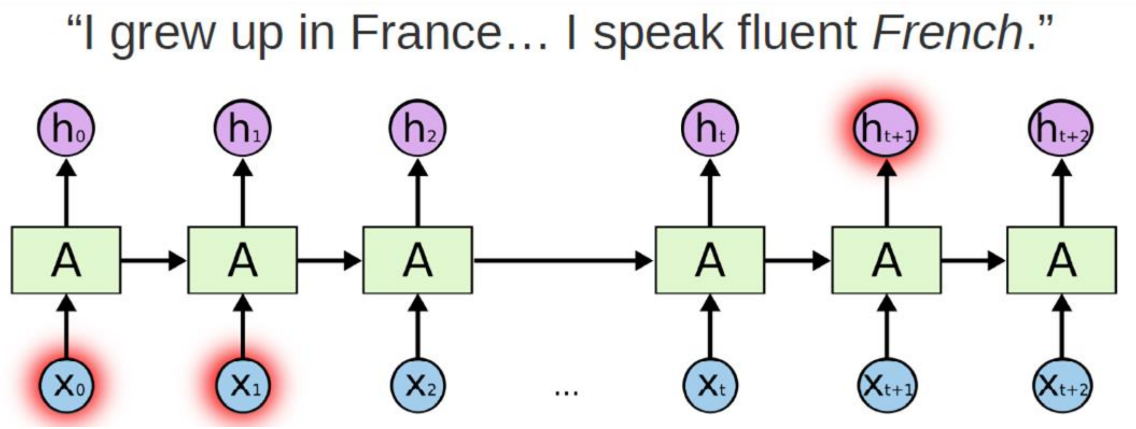
- Sentence 2

  "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 ___"

- One can tell the answer to both blank spots is most likely "John".
- It is important that the RNN predicts the next word as "John", the second person who has appeared several time-steps back in both contexts.
- But the outputs differ.

# Modeling Long Term Dependencies is an issue

- In practice, in previous example, the probability that "John" would be recognized as the next word reduces with the size of the context.
- Sensitivity of a present time step with respect to all the previous time steps is not equal.
- Eg. in cases when we need more context (from recent and distant past):
  - “I speak fluent : implies a language is the answer.
  - “I grew up in france” implies french is the answer… which is more exact.

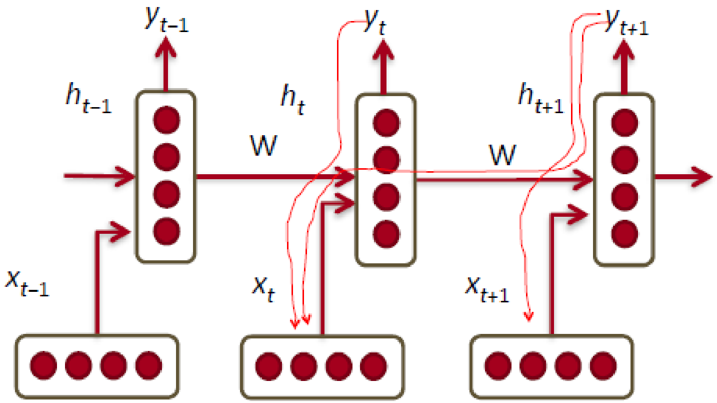


“I grew up in France… I speak fluent *French*.”

# Modeling Long Term Dependencies is an issue

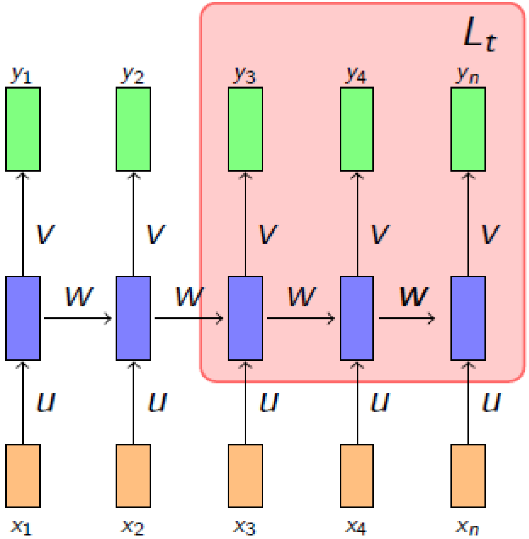
Why?

Because we are actually multiplying the same matrix at each time step during backpropagation.



It leads to a problem known as **vanishing gradients**

# Solution: Truncated BPTT

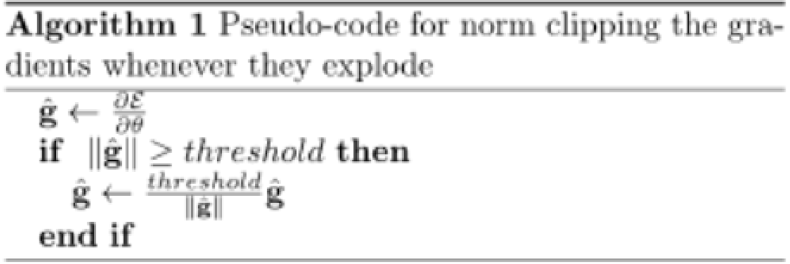


- One simple way of avoiding this is to use truncated backpropogation where we restrict the product to $\tau(< t - k)$ terms

- But this approach restricts the maximum length of the sequences it can handle.
- We can look at more elegant methods of tackling this problem.

# Solution: Gradient Clipping

A solution introduced by Mikolov is to clip gradients to a maximum value. Makes a big difference in RNNs.



**Algorithm 1** Pseudo-code for norm clipping the gradients whenever they explode

```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
**if** $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
**end if**
```

Error surface of a single hidden unit RNN,

• Solid lines:

standard gradient descent trajectories.

• Dashed lines:

gradients rescaled to fixed size.

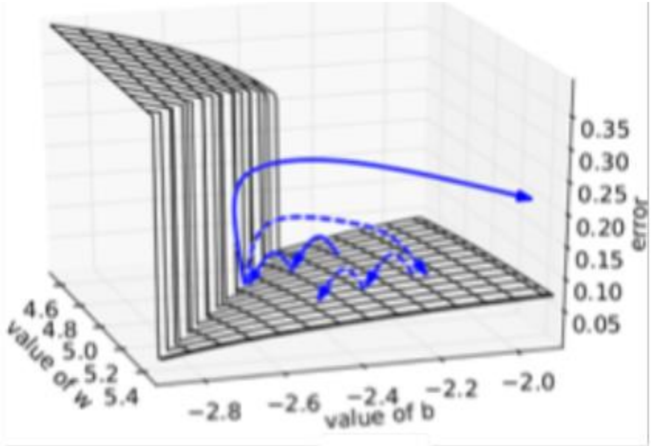
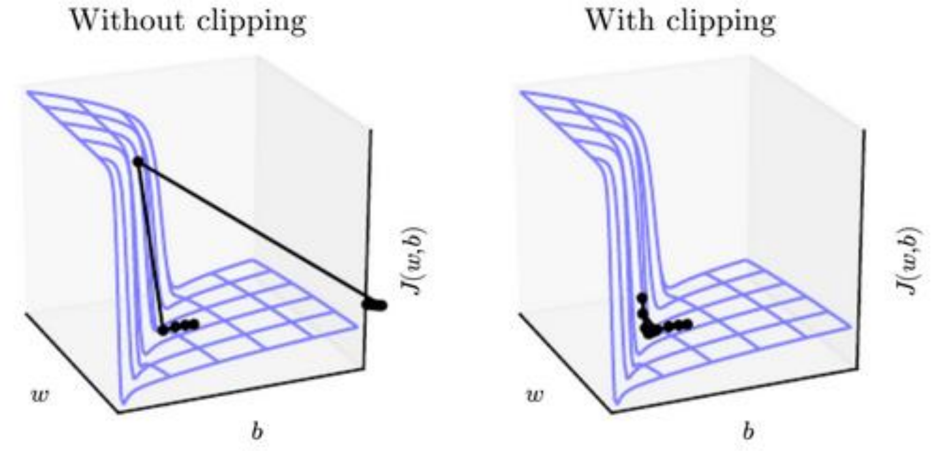


Figure from paper:
On the difficulty of training Recurrent Neural Networks, Pascanu et al. 2013

# Solution: Gradient Clipping

Gradient clipping can make gradient descent perform more reasonably in the vicinity of extremely steep cliffs.


Without clipping

With clipping

(Left Figure) Gradient descent without gradient clipping overshoots the bottom of this small ravine, then receives a very large gradient from the cliff face. The large gradient catastrophically propels the parameters outside the axes of the plot.

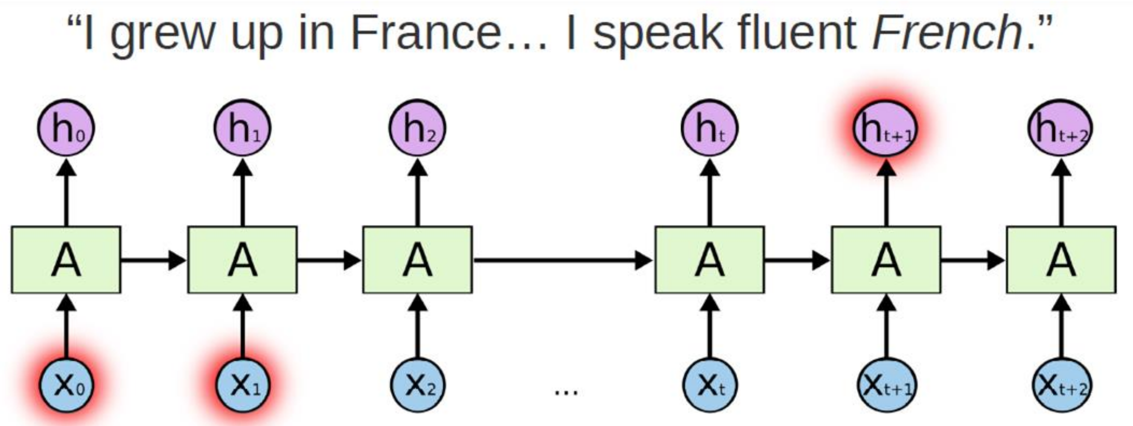
(Right Figure) Gradient descent with gradient clipping has a more moderate reaction to the cliff.

# Solution: Introducing LSTM

What could be the simplest solution to handle vanishing/exploding gradient in long term dependencies?

Ideas that may work:

- Keep around memories to capture long distance dependencies (Selective read)
- Allow error messages to flow at different strengths depending on the inputs (selective forget)



"I grew up in France… I speak fluent *French*."

- Let us see an analogy for this

- Let us see an analogy for this
- We can think of the state as a fixed size memory

- Let us see an analogy for this
- We can think of the state as a fixed size memory
- Compare this to a fixed size white board that you use to record information

- Let us see an analogy for this
- We can think of the state as a fixed size memory
- Compare this to a fixed size white board that you use to record information
- At each time step (periodic intervals) we keep writing something to the board

- Let us see an analogy for this
- We can think of the state as a fixed size memory
- Compare this to a fixed size white board that you use to record information
- At each time step (periodic intervals) we keep writing something to the board
- Effectively at each time step we morph the information recorded till that time point

- Let us see an analogy for this
- We can think of the state as a fixed size memory
- Compare this to a fixed size white board that you use to record information
- At each time step (periodic intervals) we keep writing something to the board
- Effectively at each time step we morph the information recorded till that time point
- After many timesteps it would be impossible to see how the information at time step $t-k$ contributed to the state at timestep $t$

$t$

- Continuing our whiteboard analogy, suppose we are interested in deriving an expression on the whiteboard
- We follow the following strategy at each time step
- Selectively write on the board
- Selectively read the already written content
- Selectively forgot (erase) some content
- Let us look at each of these in detail

## Selective write

- There may be many steps in the derivation ut we may just skip a few
- In other words we select what to **write**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

### Selective write

- There may be many steps in the derivation ut we may just skip a few
- In other words we select what to **write**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$

## Selective write

- There may be many steps in the derivation ut we may just skip a few
- In other words we select what to **write**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$bd = 33$$

## Selective write

- There may be many steps in the derivation ut we may just skip a few
- In other words we select what to **write**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$

$$bd = 33$$

## Selective read

- While writing one step we typically read some of the previous steps we have already written and then decide what to write next

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$

$$bd = 33$$

### Selective read

- While writing one step we typically read some of the previous steps we have already written and then decide what to write next
- For example at Step 3, information from Step 2 is important

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$bd = 33$$

## Selective read

- While writing one step we typically read some of the previous steps we have already written and then decide what to write next
- For example at Step 3, information from Step 2 is important
- In other words we select what to **read**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$\begin{aligned} ac &= 5 \\ bd &= 33 \\ bd + a &= 34 \end{aligned}$$

### Selective read

- While writing one step we typically read some of the previous steps we have already written and then decide what to write next
- For example at Step 3, information from Step 2 is important
- In other words we select what to **read**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$bd = 33$$
$$bd + a = 34$$

## Selective forget

- Once the board is full, we need to delete some obsolete information

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$bd = 33$$
$$bd + a = 34$$

**Selective forget**

- Once the board is full, we need to delete some obsolete information
- But how do we decide what to delete? We will typically delete the least useful information

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$\begin{aligned} ac &= 5 \\ bd &= 33 \\ bd + a &= 34 \end{aligned}$$

## Selective forget

- Once the board is full, we need to delete some obsolete information
- But how do we decide what to delete? We will typically delete the least useful information
- In other words we select what to **forget**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$

$$bd + a = 34$$

## Selective forget

- Once the board is full, we need to delete some obsolete information
- But how do we decide what to delete? We will typically delete the least useful information
- In other words we select what to **forget**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$ac(bd + a) = 170$$
$$bd + a = 34$$

**Selective forget**

- Once the board is full, we need to delete some obsolete information
- But how do we decide what to delete? We will typically delete the least useful information
- In other words we select what to **forget**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$ac(bd + a) = 170$$
$$ad = 11$$

### Selective forget

- Once the board is full, we need to delete some obsolete information
- But how do we decide what to delete? We will typically delete the least useful information
- In other words we select what to **forget**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$\begin{aligned} ad + ac(bd + a) &= 181 \\ ac(bd + a) &= 170 \\ ad &= 11 \end{aligned}$$

## Selective forget

- Once the board is full, we need to delete some obsolete information
- But how do we decide what to delete? We will typically delete the least useful information
- In other words we select what to **forget**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

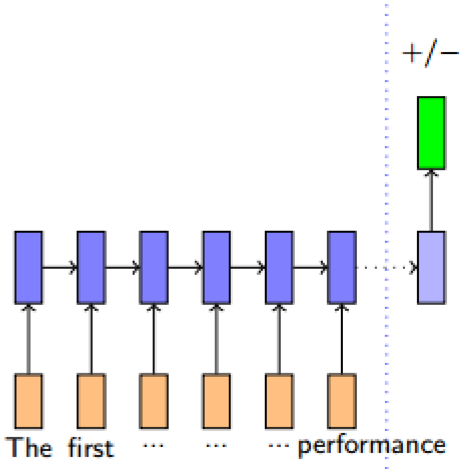
**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$\begin{aligned} ad + ac(bd + a) &= 181 \\ ac(bd + a) &= 170 \\ ad &= 11 \end{aligned}$$

- There are various other scenarios where we can motivate the need for selective write, read and forget
- For example, you could think of our brain as something which can store only a finite number of facts
- At different time steps we selectively read, write and forget some of these facts
- Since the RNN also has a finite state size, we need to figure out a way to allow it to selectively read, write and forget



**Review:** The first half of the movie was dry but the second half really picked up pace. The lead actor delivered an amazing performance

- Consider the task of predicting the sentiment (positive/negative) of a review
- RNN reads the document from left to right and after every word updates the state information
- By the time we reach the end of the document the information obtained from the first few words is completely lost
- Ideally we want to
  - **forget** the information added by stop words (a, the, etc)
  - **selectively read** the information added by previous sentiment bearing words (awesome, amazing, etc.)
  - **selectively write** new information from the current word to the state

# RNN vs LSTM

## 1. Comparison

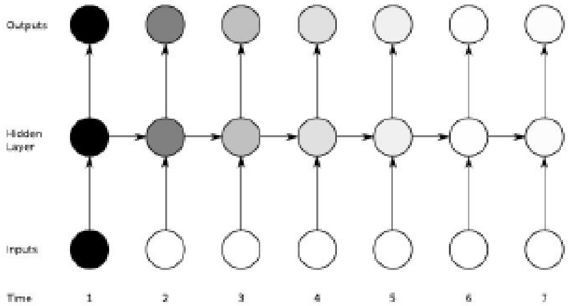


Figure 4.1: **Vanishing gradient problem for RNNs.** The shading of the nodes indicates the sensitivity over time of the network nodes to the input at time one (the darker the shade, the greater the sensitivity). The sensitivity decays exponentially over time as new inputs overwrite the activation of hidden unit and the network 'forgets' the first input.

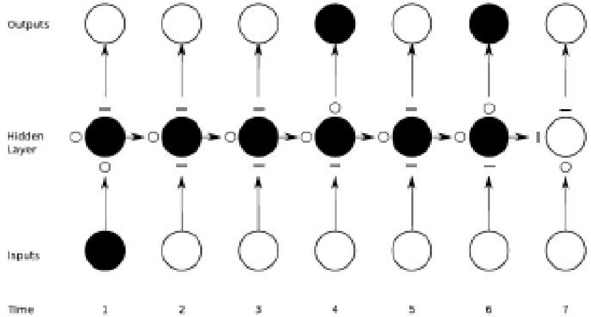


Figure 4.3: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the input unit at time one. The state of the input, forget, and output gate states are displayed below, to the left and above the hidden layer node, which corresponds to a single memory cell. For simplicity, the gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed, and the sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

# LSTM Overview

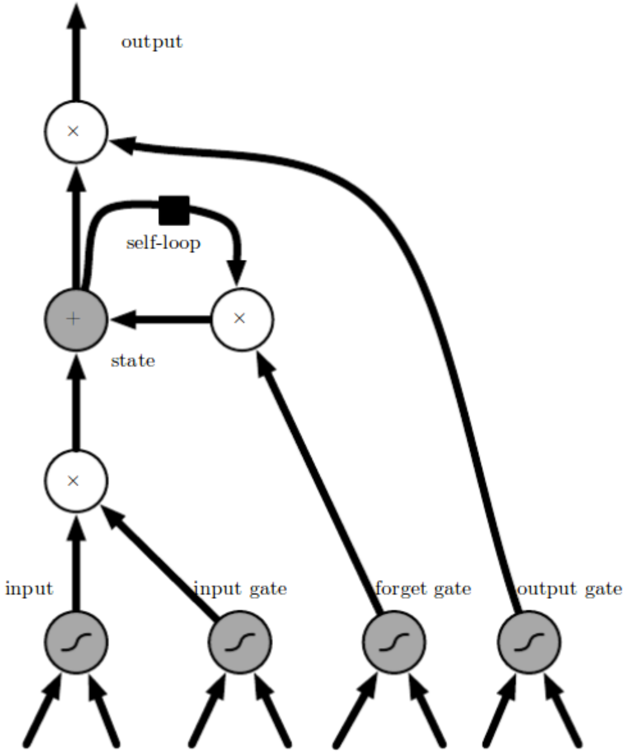
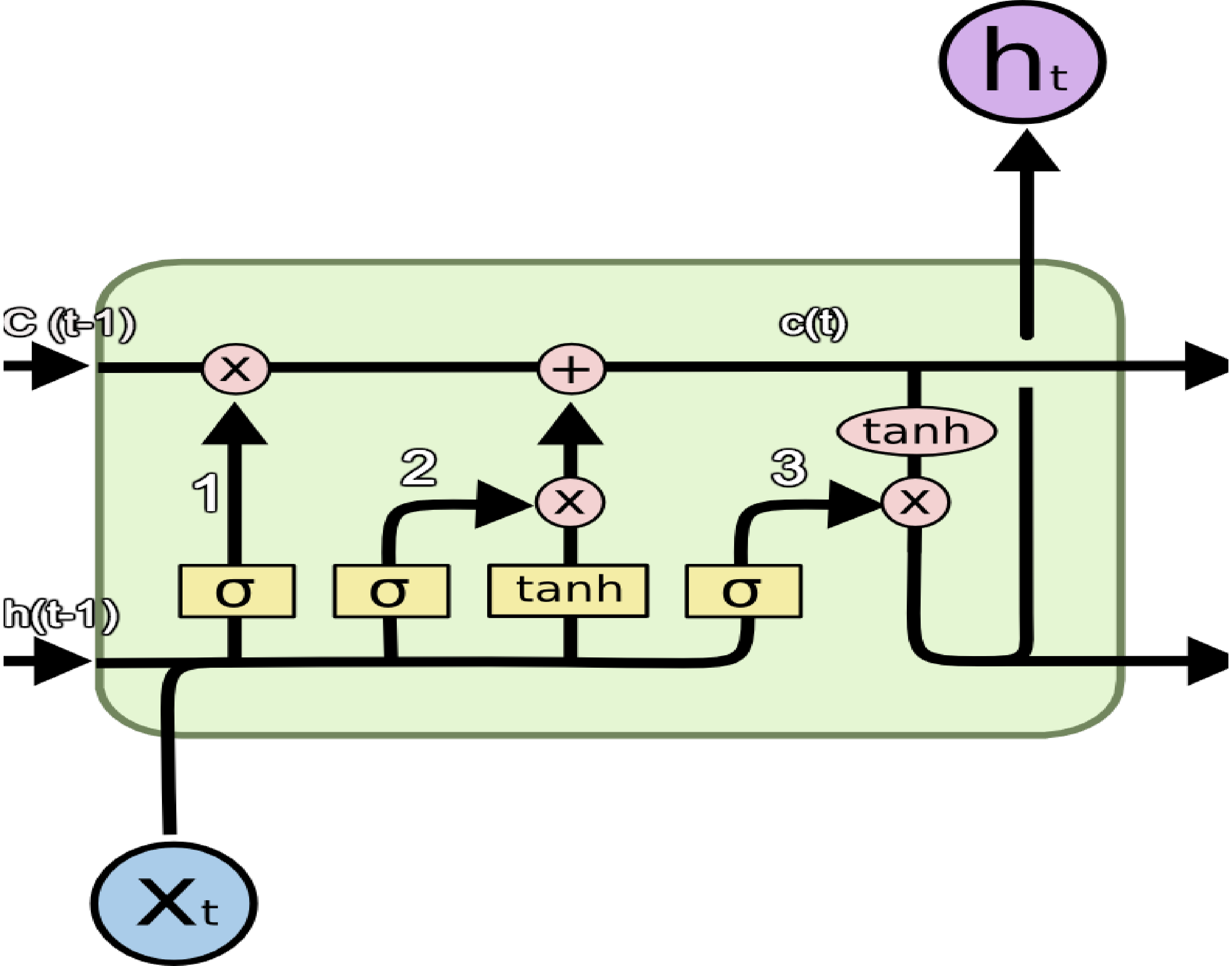
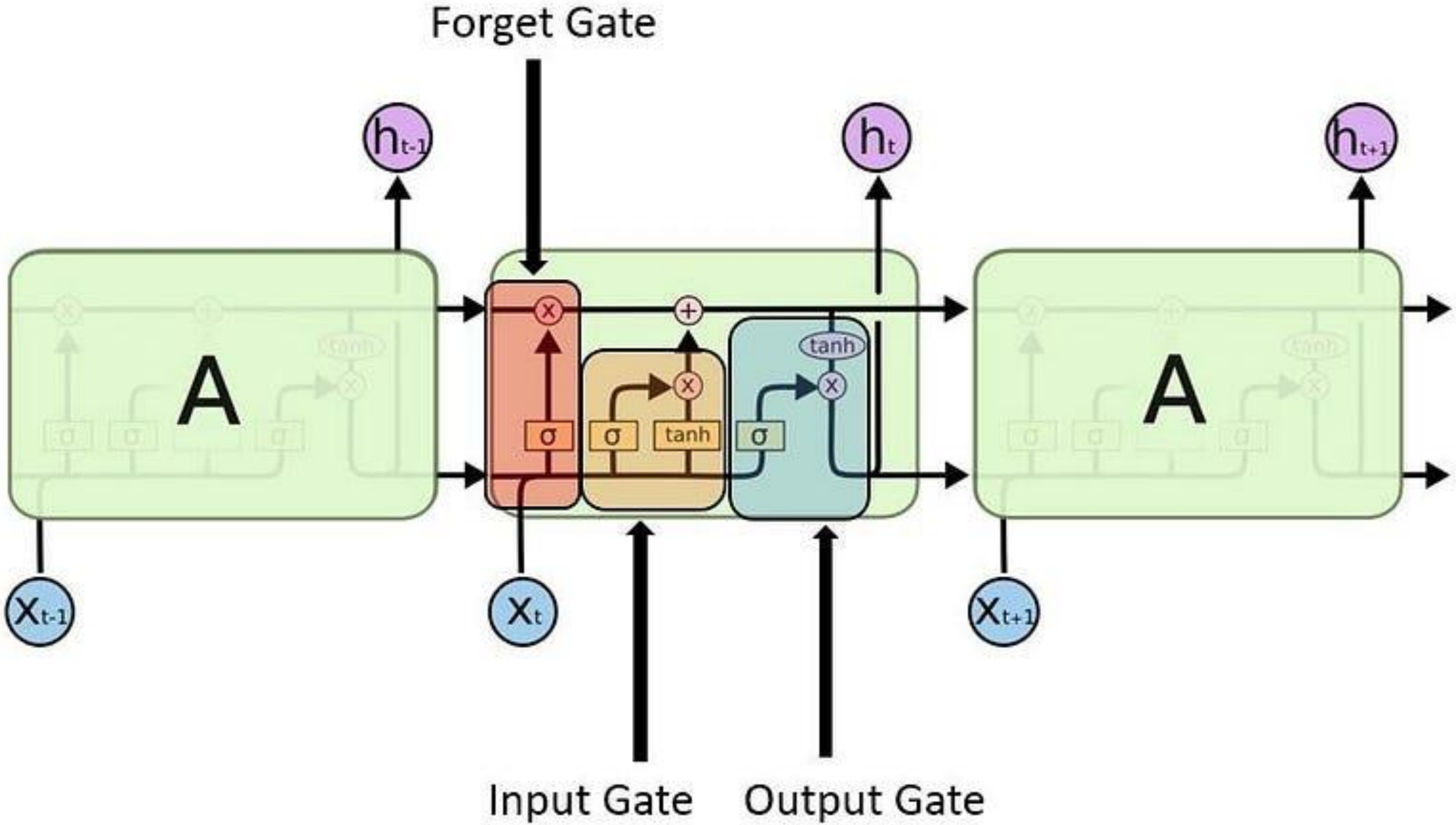


Figure 10.16: Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as an extra input to the gating units. The black square indicates a delay of a single time step.

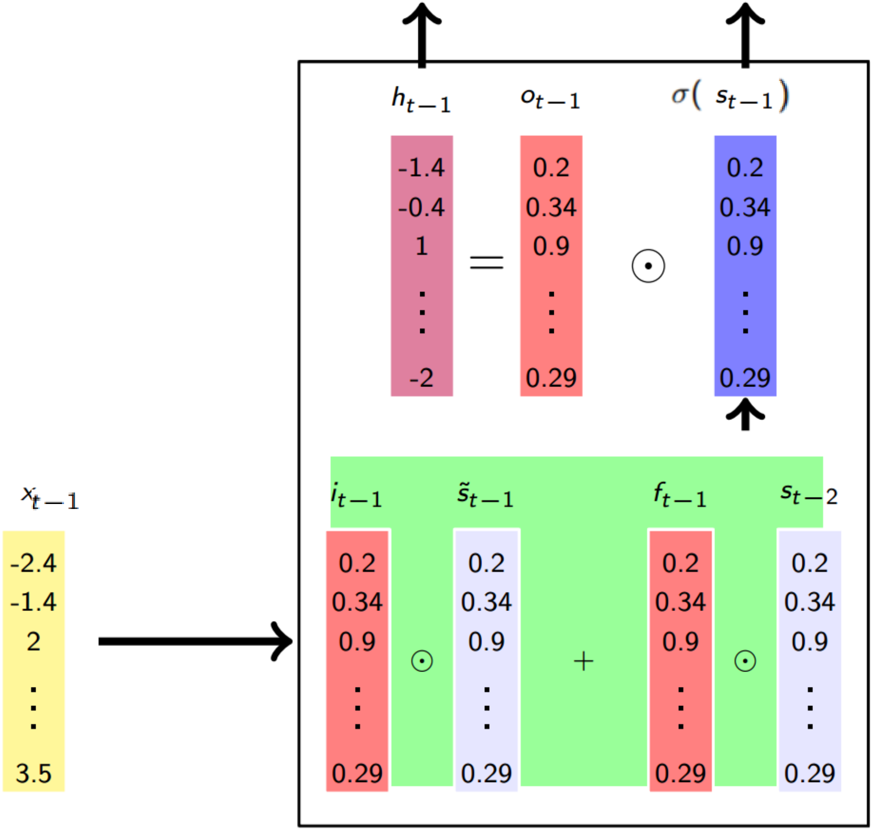

h
t
C (t-1)
c(t)
x
+
tanh
1
2
3
x
x
σ
σ
tanh
σ
h(t-1)
X
t



Forget Gate

Input Gate

Output Gate



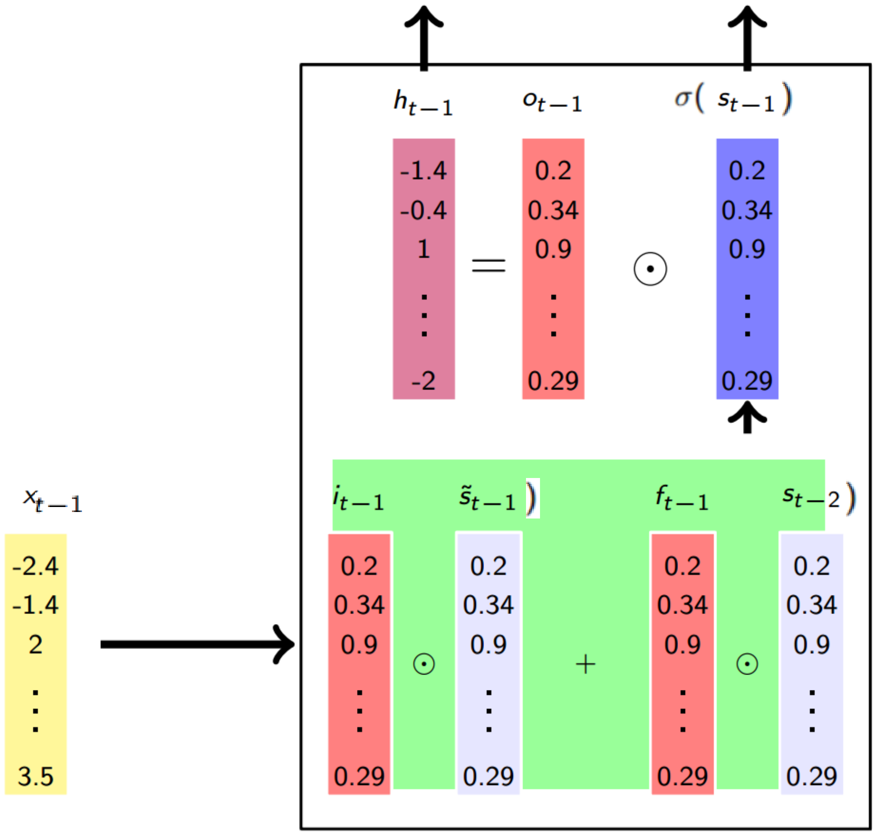
- We now have the full set of equations for LSTMs

**Gates:**

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

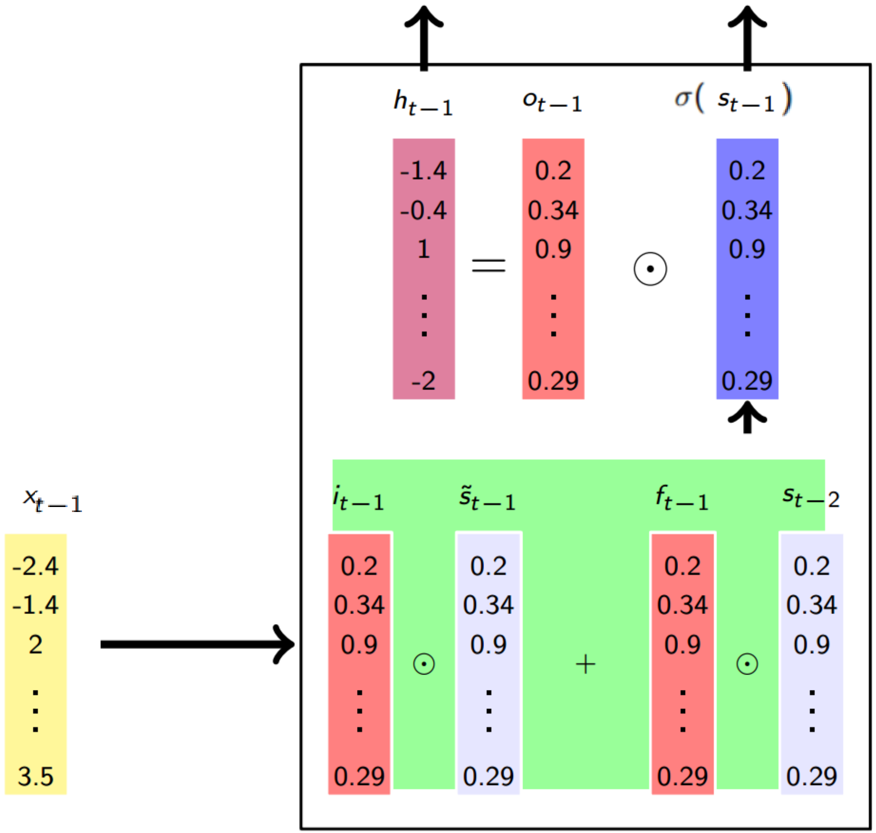
**States:**

$$\tilde{s_t} = \sigma(W h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s_t}$$
$$h_t = o_t \odot \sigma(s_t)$$
$$rnn_{out} = h_t$$



## Intuition: Why LSTMs work?

- During forward propagation the gates control the flow of information
- They prevent any irrelevant information from being written to the state
- Similarly during backward propagation they control the flow of gradients
- It is easy to see that during backward pass the gradients will get multiplied by the gate



## Intuition: Why LSTMs work?

- If the state at time $t$ did not contribute much to the state at time $t+1$ (i.e., if $f_t \rightarrow 0$ and $o_t \rightarrow 0$ ) then during backpropagation the gradients flowing into $s_t$ will vanish
- But this kind of a vanishing gradient is fine (since $s_t$ did not contribute to $s_{t+1}$ we don't want to hold it responsible for the crimes of $s_t$)
- The key difference from vanilla RNNs is that the flow of information and gradients is controlled by the gates which ensure that the gradients vanish only when they should (i.e., when $s_{t+1}$ to contribute to $s_t$)

# Variants of LSTM

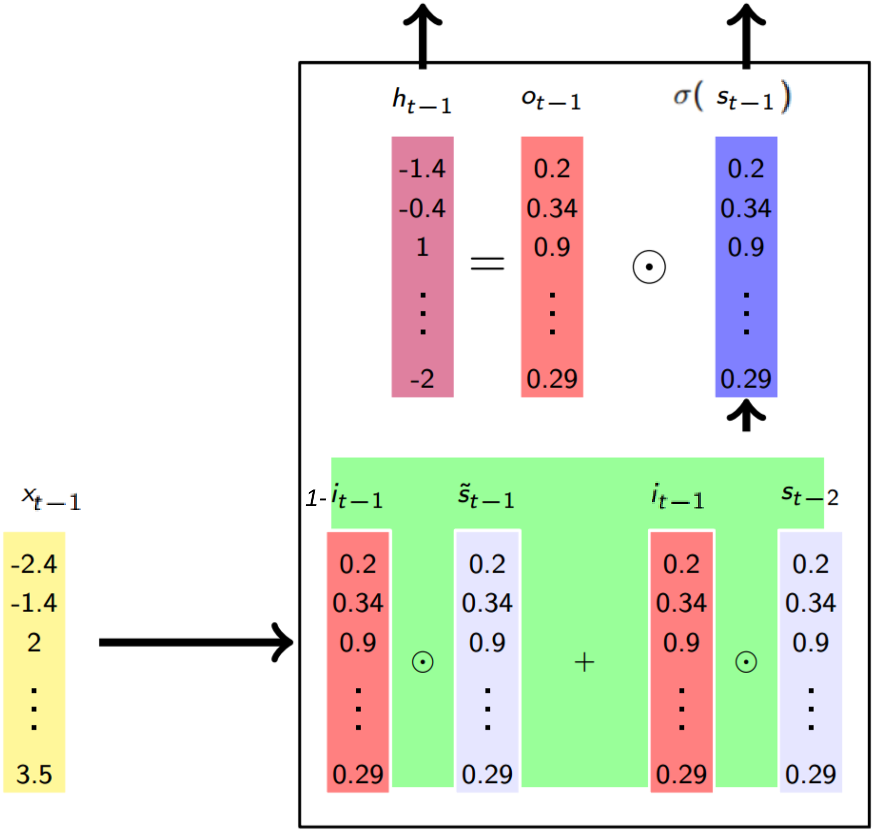
LSTM has many variants which include different number of gates and also different arrangement of gates:

- No Input Gate (NIG)
- No Forget Gate (NFG)
- No Output Gate (NOG)
- Coupled Input and Forget Gate (CIFG)

Etc.

GRU is one such frequently used variant.

# Variants of LSTM



- The full set of equations for GRUs

**Gates:**

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

**States:**

$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$
$$s_t = i_t \odot s_{t-1} + (1 - i_t) \odot \tilde{s}_t$$

- No explicit forget gate (the forget gate and input gates are tied)
- The gates depend directly on $s_{t-1}$ and not the intermediate $h_{t-1}$ as in the case of LSTMs

# Variants of LSTM

- The most commonly used LSTM architecture (vanilla LSTM) performs  reasonably well on various datasets and using any of eight possible modifications does not significantly improve the LSTM performance. (Greff et al., 2015; Jozefowicz et al., 2015).

- Certain modifications such as coupling the input and forget gates or removing peephole connections simplify LSTM without significantly hurting performance.

- The forget gate and the output activation function are the critical components of the LSTM block. While the first is crucial for LSTM performance, the second is necessary whenever the cell state is unbounded.

- Initially, if forget gates are set, then the memory decays fast. This trick suits tasks that exhibit less long range dependencies.

- Jozefowicz et al. (2015) found that adding a bias of 1 to the LSTM forget gate, a practice advocated by Gers et al. (2000), makes the LSTM as strong as the best of the explored architectural variants.

# References

1. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduc7on-to-rnns/
2. http://karpathy.github.io/2015/05/21/rnn-effec7veness/
3. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
4. http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf
5. http://jeffdonahue.com/lrcn/
6. Multiple Object Recognition with Visual Attention Jimmy Ba, Volodymyr Mnih, Koray Kavukcuoglu
7. DRAW: A Recurrent Neural Network For Image Generation Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, Daan Wierstra
8. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. Junyoung Chung, Caglar Gulcehre Kyung, Hyun Cho, Yoshua Bengio
9. An Empirical Exploration of Recurrent Network Architectures. Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever