CS224n: NLP with Deep Learning

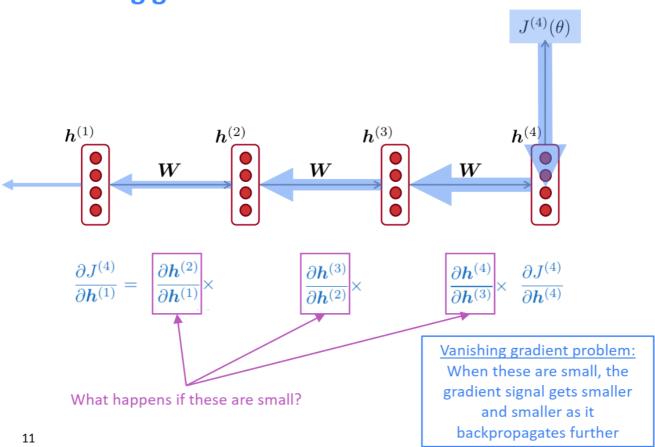
Lecture 7: Vanishing Gradients and Fancy RNN

Vanishing Gradient Problem

Intuition

• When the gradients $\frac{\partial h^{(i)}}{\partial h^{(i-1)}}$ are small, the overall gradient is going to get smaller and smaller, as we go backwards

Vanishing gradient intuition



• If the largest eigenvalue of W_h is < 1:

Proof Sketch

Vanishing gradient proof sketch

• Recall:
$$oldsymbol{h}^{(t)} = \sigma \left(oldsymbol{W}_h oldsymbol{h}^{(t-1)} + oldsymbol{W}_x oldsymbol{x}^{(t)} + oldsymbol{b}_1
ight)$$

• Therefore:
$$\frac{\partial m{h}^{(t)}}{\partial m{h}^{(t-1)}} = \mathrm{diag}\left(\sigma'\left(m{W}_hm{h}^{(t-1)} + m{W}_xm{x}^{(t)} + m{b}_1
ight)\right)m{W}_h$$
 (chain rule)

• Consider the gradient of the loss $J^{(i)}(\theta)$ on step i, with respect to the hidden state $h^{(j)}$ on some previous step j.

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j < t \le i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$
 (chain rule)
$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \underbrace{\boldsymbol{W}_{h}^{(i-j)}}_{j < t \le i} \operatorname{diag} \left(\sigma' \left(\boldsymbol{W}_{h} \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_{x} \boldsymbol{x}^{(t)} + \boldsymbol{b}_{1} \right) \right)$$
 (value of $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$)

If W_h is small, then this term gets vanishingly small as i and j get further apart

Vanishing gradient proof sketch

Consider matrix L2 norms:

$$\left\| \frac{\partial J^{(i)}(\boldsymbol{\theta})}{\partial \boldsymbol{h}^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\boldsymbol{\theta})}{\partial \boldsymbol{h}^{(i)}} \right\| \left\| \boldsymbol{W}_h \right\|^{(i-j)} \prod_{j < t \leq i} \left\| \operatorname{diag} \left(\sigma' \left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1 \right) \right) \right\|$$

- Pascanu et al showed that that if the largest eigenvalue of W_h is less than 1, then the gradient $\left\|\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}}\right\|$ will shrink exponentially
 - · Here the bound is 1 because we have sigmoid nonlinearity
- There's a similar proof relating a largest eigenvalue >1 to exploding gradients

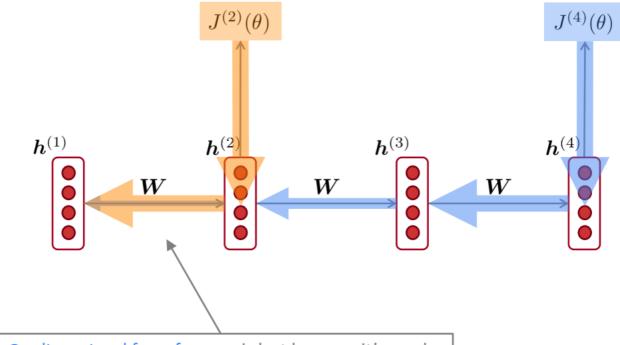
13

Source: "On the difficulty of training recurrent neural networks", Pascanu et al. 2013, http://proceedings.mlr.press/y28/pascanu13.pdf

But why is this a problem?

- The gradient from steps far away will be much smaller than gradients from close-by
- → The weights will only be updated with respect to close effects, not long-term!

Why is vanishing gradient a problem?



Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

14

Remark:

- We are studying $\frac{\partial J}{\partial h^{(i-1)}}$ because we have to calculate it to get $\frac{\partial J}{\partial W}$

Effect on RNN-LM

- Incapability of learning long-distance dependencies
- Syntactic recency (correct):
 Making dependencies to words close in their syntax
- Sequential recency (incorrect):
 Making dependencies to words close spatially

Unfortunately, RNN-LMs are better at learning sequential recency than syntactic recency: (see example below)

Effect of vanishing gradient on RNN-LM

- LM task: The writer of the books ____ are
- Correct answer: The writer of the books is planning a sequel
- Syntactic recency: The <u>writer</u> of the books <u>is</u> (correct)
- Sequential recency: The writer of the <u>books</u> <u>are</u> (incorrect)
- Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we'd like [Linzen et al 2016]

17 "Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies", Linzen et al, 2016. https://arxiv.org/pdf/1611.01368.pdf

Exploding Gradient Problem

- If our gradient is too big, the update term for our parameter becomes too big
- → Take update steps too large, and get stuck in a local optimum
- → Get Inf or Nan values

Solution: Gradient clipping

· Scale down gradients that are higher than some thereshold

LSTM

- Reason to exist = To solve the vanishing gradient problem
- A RNN with ability to learn / preserve information from many timesteps ago

What about a RNN with separate memory?

For each step t:

• a hidden state $h^{(t)}$: (n, 1)

• a cell state $c^{(t)}$: (n, 1)Stores long-term information

LSTM can erase, write and read information from cell!

This is done using gates of length n, whose values are in [0,1], thanks to our **sigmoid** friend:

- 1: open gate
- · 0: closed gate

Long Short-Term Memory (LSTM)

We have a sequence of inputs $m{x}^{(t)}$, and we will compute a sequence of hidden states $m{h}^{(t)}$ and cell states $c^{(t)}$ On timestep t:

Sigmoid function: all gate Forget gate: controls what is kept vs values are between 0 and 1 forgotten, from previous cell state $oldsymbol{f}^{(t)} = \sigma \left(oldsymbol{W}_f oldsymbol{h}^{(t-1)} + oldsymbol{U}_f oldsymbol{x}^{(t)} + oldsymbol{b}_f
ight) \ oldsymbol{i}^{(t)} = \sigma \left(oldsymbol{W}_i oldsymbol{h}^{(t-1)} + oldsymbol{U}_i oldsymbol{x}^{(t)} + oldsymbol{b}_i
ight)$ Input gate: controls what parts of the All these are vectors of same length n new cell content are written to cell Output gate: controls what parts of $oldsymbol{o}^{(t)} = \sigma \left(oldsymbol{W}_o oldsymbol{h}^{(t-1)} + oldsymbol{U}_o oldsymbol{x}^{(t)} + oldsymbol{b}_o
ight)$ cell are output to hidden state New cell content: this is the new content to be written to the cell Cell state: erase ("forget") some $oxed{ ilde{c}^{(t)} = anh\left(oldsymbol{W}_coldsymbol{h}^{(t-1)} + oldsymbol{U}_coldsymbol{x}^{(t)} + oldsymbol{b}_c
ight)}$ content from last cell state, and write ("input") some new cell content $\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$ $ightarrow oldsymbol{h}^{(t)} = oldsymbol{o}^{(t)} \circ anh oldsymbol{c}^{(t)}$ Hidden state: read ("output") some content from the cell Gates are applied using 23 element-wise product

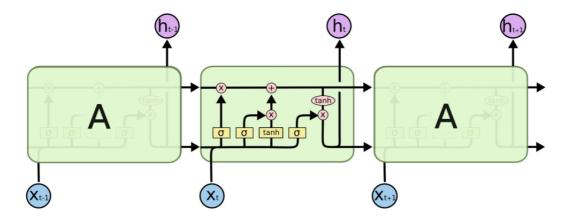
Cell state $c^{(t)}$ is the sum of:

- · previous cell state, masked by our forget gate
- · our new cell, masked by our input gate

- These cell states ~ general memory, generally not accessible from the outside
- The hidden states will be passed on to the model

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



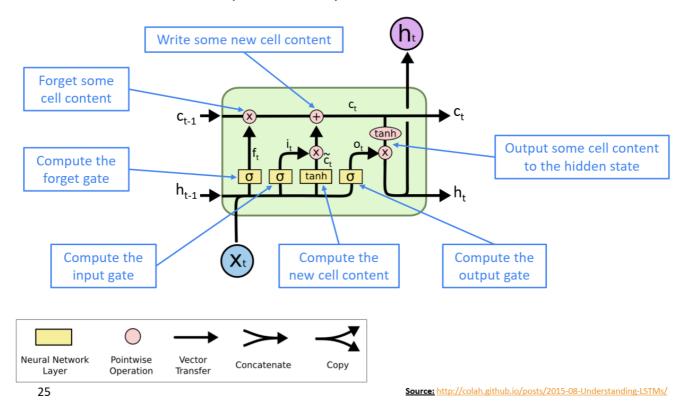


24

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



It is possible to set the forget gate so that the LSTM remembers everything from each time step

/!\ We could still have vanishing/exploding gradients, even when using LSTMs!

- With RNN, the hidden states were a bottleneck: all gradients have to pass through them !!
- With LSTM, we can view the cell as a shortcut connection
 There is a potential route within the cell that doesn't make the gradient vanish!

Applications

- · Handwriting recognition
- · Speech recognition
- · Machine translation
- Parsing
- · Image captioning

However, more recently, Transformers seem to have taken over!!



Gated Recurrent Units (GRU)

- Simpler alternative to LSTMs
- We have no cell state, only hidden states
- But we still have our cool gates!

Gates

- Update gate: controls what is updated or preserved
 input & forget gates for LSTM
- Reset gate: selects which parts of the previous state are useful
- News hidden state = combination of previous hidden state, and computed new content

Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state).

<u>Update gate:</u> 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.

<u>Hidden state:</u> update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content $egin{aligned} oldsymbol{u}^{(t)} &= \sigma \left(oldsymbol{W}_u oldsymbol{h}^{(t-1)} + oldsymbol{U}_u oldsymbol{x}^{(t)} + oldsymbol{b}_u
ight) \ oldsymbol{r}^{(t)} &= \sigma \left(oldsymbol{W}_r oldsymbol{h}^{(t-1)} + oldsymbol{U}_r oldsymbol{x}^{(t)} + oldsymbol{b}_r
ight) \end{aligned}$

 $ilde{m{h}}^{(t)} = anh\left(m{W}_h(m{r}^{(t)} \circ m{h}^{(t-1)}) + m{U}_hm{x}^{(t)} + m{b}_h
ight)$ $m{h}^{(t)} = (1 - m{u}^{(t)}) \circ m{h}^{(t-1)} + m{u}^{(t)} \circ ilde{m{h}}^{(t)}$

How does this solve vanishing gradient? Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

28 "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

GRU compared to LSTM:

- Quicker
- · Less parameters
- · Similar performances

Other fixes

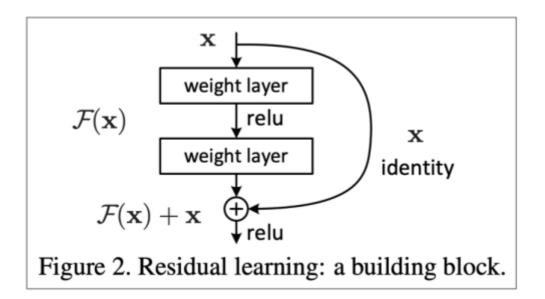
Gradient clipping

(See above)

Skip connections

- · Create direct connections from lower layers to newer layers
 - → Allowing the gradients to flow more easily

Example: ResNet



• DenseNet:

Going even further: connect every layer to each other!

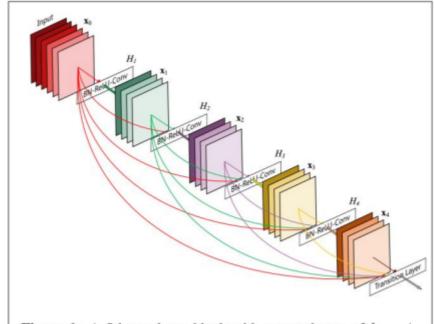


Figure 1: A 5-layer dense block with a growth rate of k=4. Each layer takes all preceding feature-maps as input.

Recap 1

Recap

- Today we've learnt:
 - Vanishing gradient problem: what it is, why it happens, and why it's bad for RNNs
 - LSTMs and GRUs: more complicated RNNs that use gates to control information flow; they are more resilient to vanishing gradients
- Remainder of this lecture:
 - Bidirectional RNNs
 - Multi-layer RNNs

Both of these are pretty simple

More fancy RNN variants

Bidirectional RNN

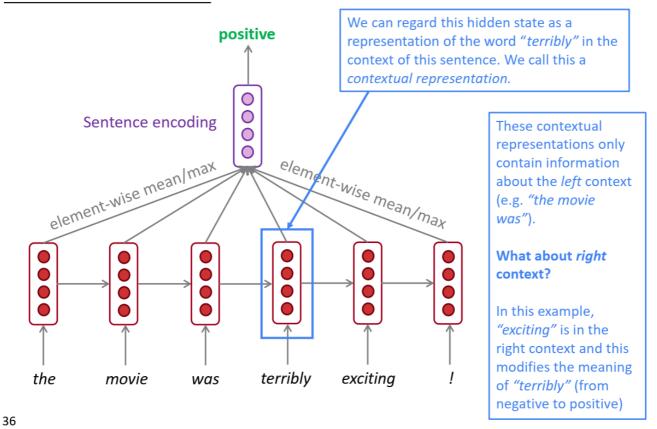
Motivation

• Our contextual representation only gets information from the left context!

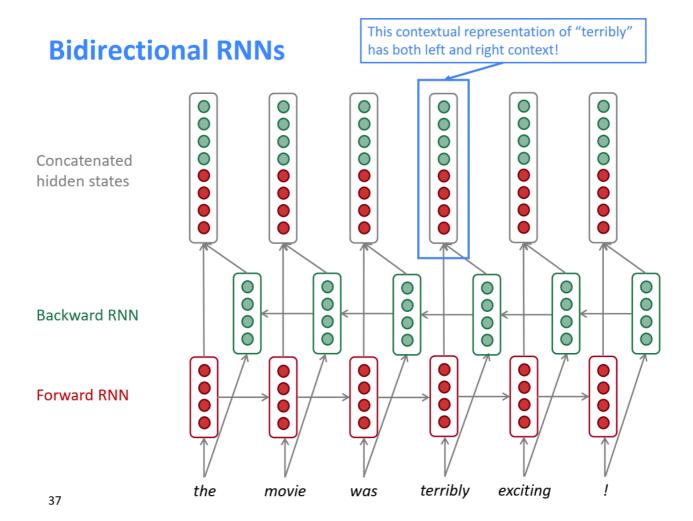
We might want to get context coming from the right direction as well!

Bidirectional RNNs: motivation

Task: Sentiment Classification



- We have 2 parallel RNN: one for each direction
- Hidden states = [Hidden state from left, Hidden state from right]



Notations

Bidirectional RNNs

On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a vanilla, LSTM or GRU computation.

Forward RNN
$$\overrightarrow{\boldsymbol{h}}^{(t)} = \overline{\text{RNN}_{\text{FW}}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \overline{\text{RNN}_{\text{BW}}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Concatenated hidden states $\overleftarrow{\boldsymbol{h}}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

38

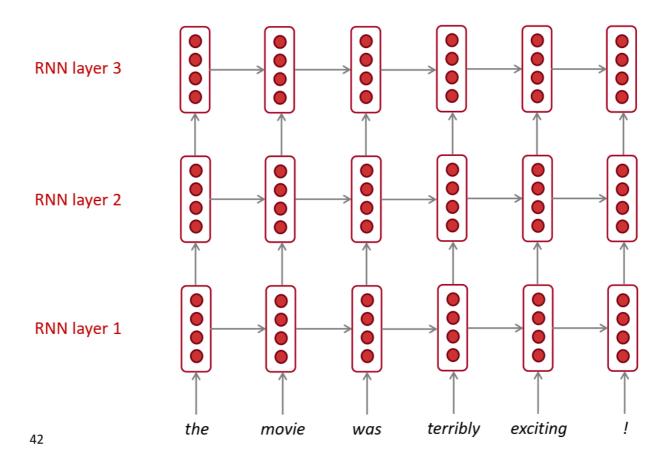
- The 2 RNNs are trained together, not separately
- Not useful for Language Modelling (as the Right context is missing, by definition!)

Multi-layer RNN

- · Using multiple RNNs
- · Allows to compute more complex representations
- Hidden states of layer i are the inputs to layer i+1

Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*

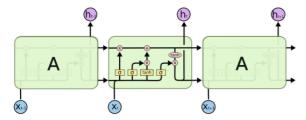


- RNNs have to be computed sequentially
 - \rightarrow Expensive to compute, can't go too deep

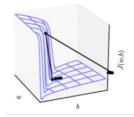
Recap 2: Practical Takeaways

In summary

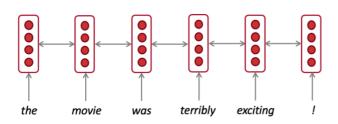
Lots of new information today! What are the practical takeaways?



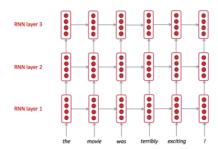
1. LSTMs are powerful but GRUs are faster



2. Clip your gradients



3. Use bidirectionality when possible



4. Multi-layer RNNs are powerful, but you might need skip/dense-connections if it's deep

44