

Problems in RNN (vanishing and exploding gradients):

This problem arises due to repeated multiplication of a factor in the calculation of how a word far behind in time affects the cost at a point.

- **Recall:** $\mathbf{h}^{(t)} = \sigma \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right)$
- **Therefore:** $\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} = \text{diag} \left(\sigma' \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \mathbf{W}_h$ (chain rule)
- **Consider the gradient of the loss $J^{(i)}(\theta)$ on step i , with respect to the hidden state $\mathbf{h}^{(j)}$ on some previous step j .**

$$\begin{aligned} \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} && \text{(chain rule)} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \mathbf{W}_h^{(i-j)} \prod_{j < t \leq i} \text{diag} \left(\sigma' \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) && \text{(value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \text{)} \end{aligned}$$

Note: what we actually want is dJ/dw but we talk about dJ/dh as dL/dw is represented as dJ/dh once we put in the chain rule.

Notice the \mathbf{W}_h factor, now if it is a number less than one, since it is raised to the power $(i - j)$ the resulting gradient becomes very less and we get the problem of vanishing gradient. Notice this is more prominent the further the word, (if word is further, $i - j$ increases), but this is not completely true in real world languages as a word many words ago can provide the context for gender and place and hence hold more significance than a word near where j is calculated and hence it is affected more by the word and have must have larger gradient.

The opposite happens when w_h is greater than 1 and we end up with exploding gradients.

- **Consider matrix L2 norms:**

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

- Pascanu et al showed that that if the **largest eigenvalue** of \mathbf{W}_h is **less than 1**, then the gradient $\left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} \right\|$ **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**

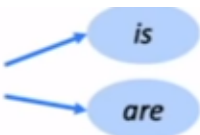


Another way to look at why this is bad is that this results in losses being affected by words near it a lot more, hence we end up with the same problem of small windows (words not affected by older words) which we are trying to solve with RNNs, (at least to a small extent)

Specific examples where vanishing gradients mess up:

1.

- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

2.

- **LM task:** *The writer of the books _____*

 - is
 - are
- **Correct answer:** *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* (correct)

- **Sequential recency:** *The writer of the books are* (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]

Why exploding gradient is a problem ?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_{\theta} J(\theta)}_{\text{gradient}}$$

- This can cause **bad updates**: we take too large a step and reach a bad parameter configuration (with large loss)
- In the worst case, this will result in **Inf** or **NaN** in your network (then you have to restart training from an earlier checkpoint)

If this happens, it is not possible to train it this way, What we should do is try to retrain from a previous checkpoint.

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

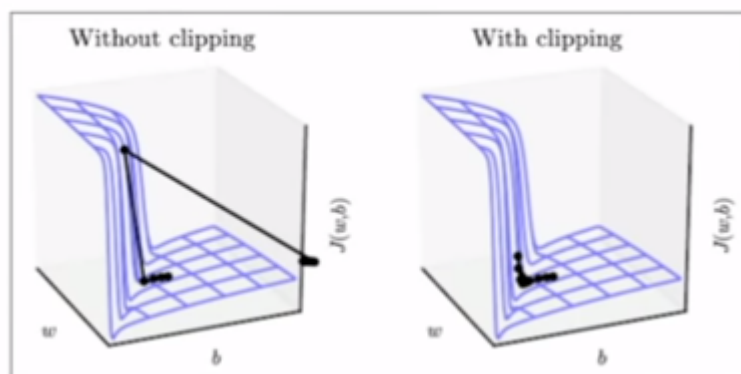
Algorithm 1 Pseudo-code for norm clipping

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

- **Intuition**: take a step in the same direction, but a smaller step

Important point is that the step is now still in the same direction, but only with a smaller size.

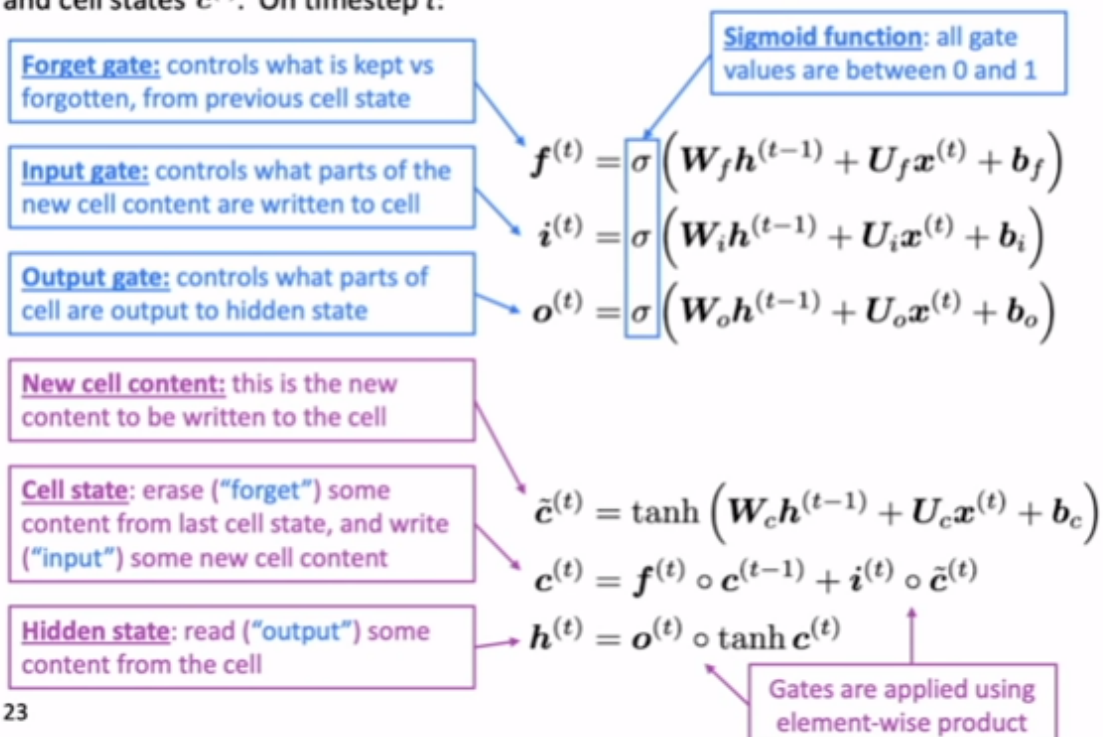
Visualization:



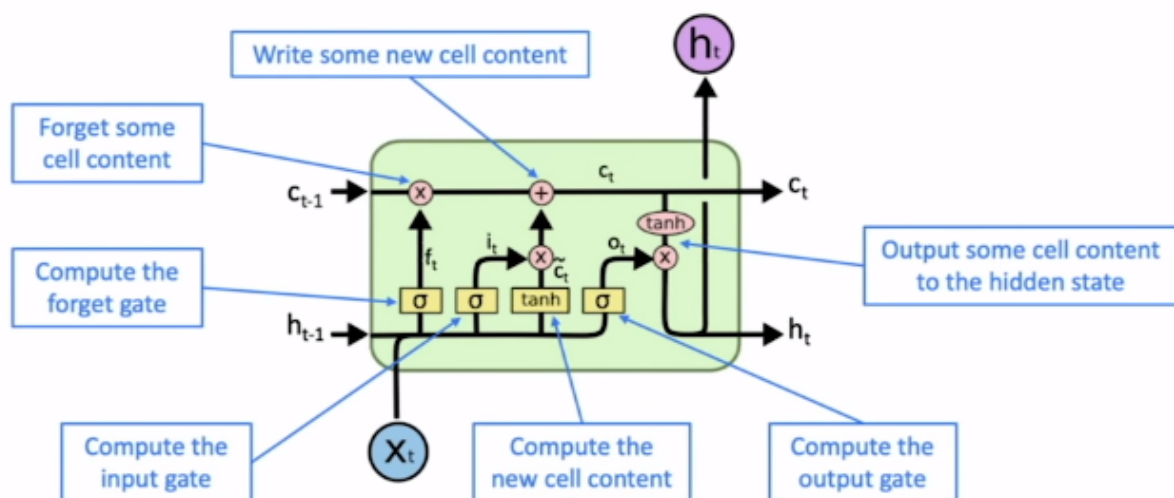
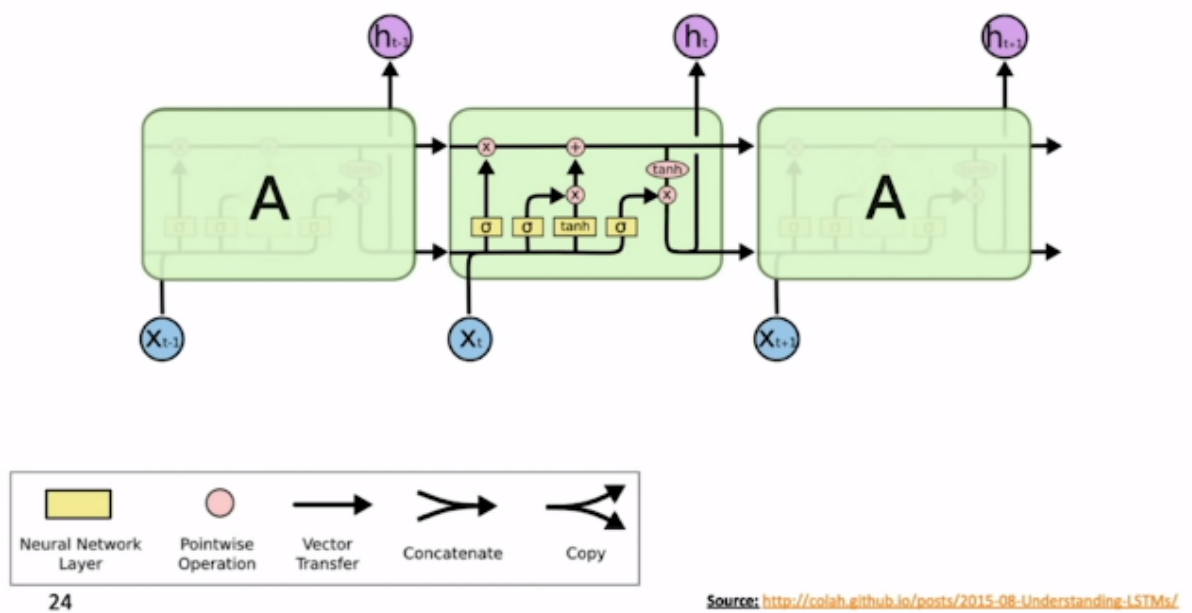
Long short-term memory (LSTM):

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
- On step t , there is a **hidden state** $h^{(t)}$ and a **cell state** $c^{(t)}$
 - Both are vectors length n
 - 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**
 - The gates are also vectors length n
 - 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 the current context**

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



Visual representation of LSTMs:



Note: using LSTMs don't guarantee that the problem of vanishing gradient disappears, but it greatly reduces the risk of it happening.

LSTMs in real world:

- In **2013-2015**, LSTMs started achieving state-of-the-art results
 - Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning
 - LSTM became the **dominant approach**
- **Now (2019)**, other approaches (e.g. **Transformers**) have become more dominant for certain tasks.
 - For example in **WMT** (a MT conference + competition):
 - In **WMT 2016**, the summary report contains "**RNN**" **44** times
 - In **WMT 2018**, the report contains "**RNN**" **9** times and "**Transformer**" **63** times

Gated Recurrent Units (GRU):

This is simpler when compared to LSTMs and has a simpler approach. In this method there is no cell state, only a hidden layer.

- 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).

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

$$u^{(t)} = \sigma(W_u h^{(t-1)} + U_u x^{(t)} + b_u)$$

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

$$r^{(t)} = \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r)$$

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

$$\tilde{h}^{(t)} = \tanh(W_h(r^{(t)} \circ h^{(t-1)}) + U_h x^{(t)} + b_h)$$

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

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

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)

LSTMs vs GRU:

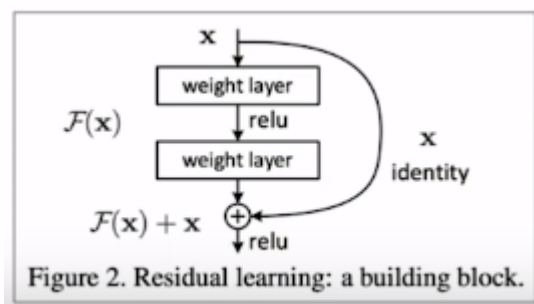
There is no definite winner when comparing LSTMs and GRU. But as discussed earlier, GRU is computationally less demanding. So the **general rule of thumb** is that: start with a LSTMs but if you want to be more efficient, use GRU. Another point to remember is that since LSTMs have more parameters than GRU they can remember more learnings and are hence more suited for larger training sets.

Is vanishing/exploding gradient only a RNN problem?

No all deep learning architecture like feed-forward and convolutional NN experience this. But it is just that RNNs are more susceptible to vanishing/ exploding gradients due to the repeated multiplication.

Residual connections (ResNet):

This is a method in which the input from steps before is added to the current step in order to protect the old data from disappearing due to squishing due to non-linearity functions. Since this method is loooooo more simple, it is obviously easier to compute.



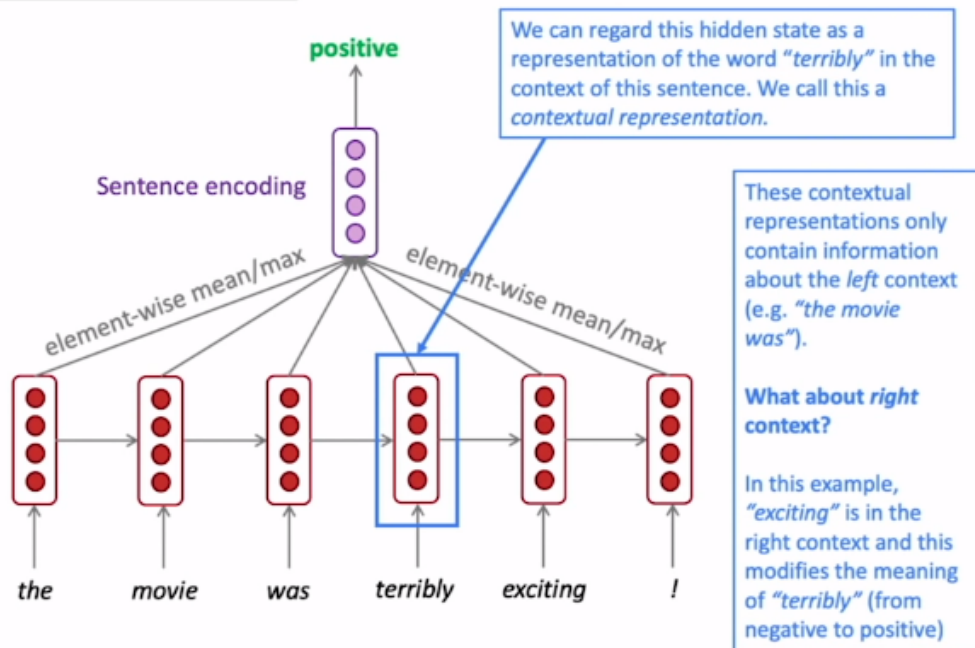
Highway connections (HighwayNet):

This is very similar to ResNet but we have an additional **dynamic** gate which is applied on top and the input from before is not simply added.

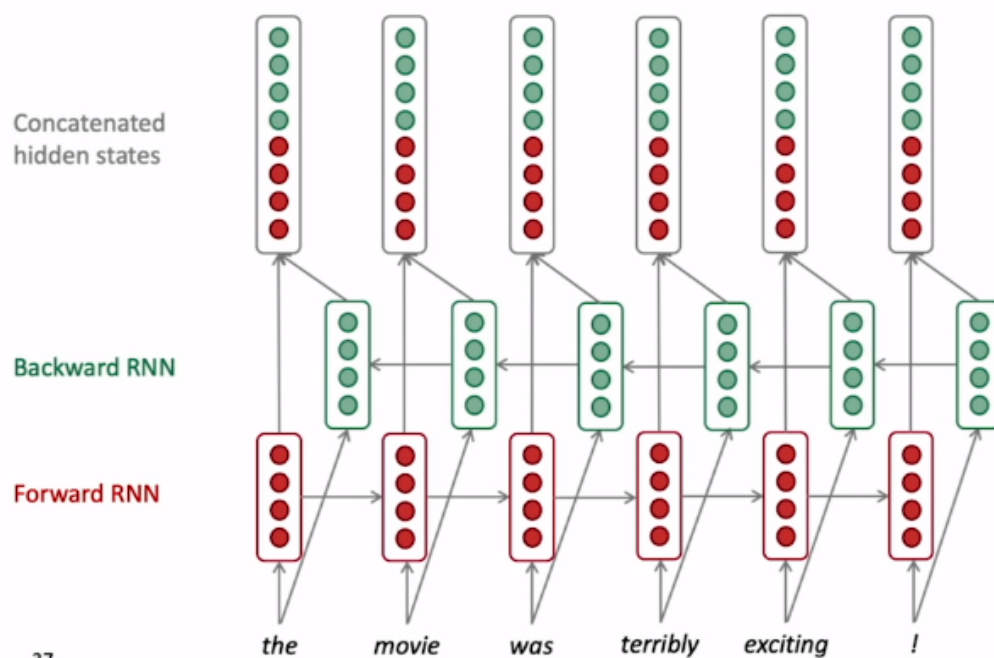
Bidirectional RNN:

Example:

Task: Sentiment Classification



36



37

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 $\vec{h}^{(t)} = \text{RNN}_{\text{FW}}(\vec{h}^{(t-1)}, x^{(t)})$

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

Generally, these two RNNs have separate weights

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

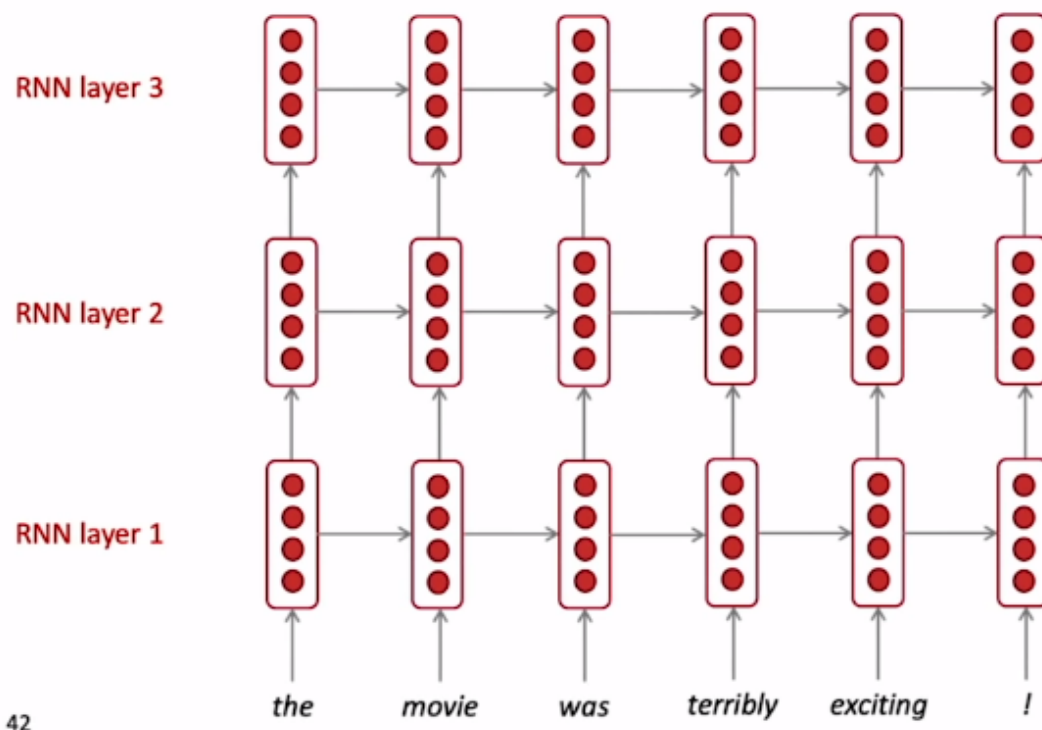
We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

You cannot bidirectional RNNs for Language modelling (obviously as you have only the left context). But **if you have the full input sequence then you must by default use bidirectional RNNs as they are very powerful.**

BERT (Bidirectional Encoder Representations form transformers) is a very powerful pretrained contextual representation system built on bidirectionality.

Multi layer RNNs:

Multi layer RNNs also called as stacked RNNs are just multiple RNNs. RNNs are already deep as they are processed over many timestamps, but they can be made deeper by making them a multi-layer RNN. Lower RNNs should compute lower level features like syntax and higher RNNs should calculate something like semantics.

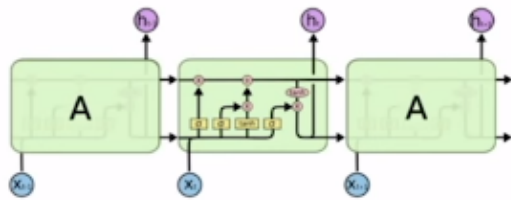


This is a unidirectional RNN but they can be bidirectional RNNs also.

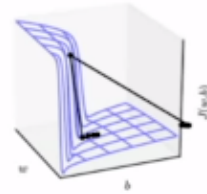
Note: hidden layer of one RNN is input of RNN above it.

Note: 2 to 4 layers are best for encoder RNNs and 4 layers is best for decoder RNNs. however if you add skip connections / dense connections, you can do upto 8 layers efficiently

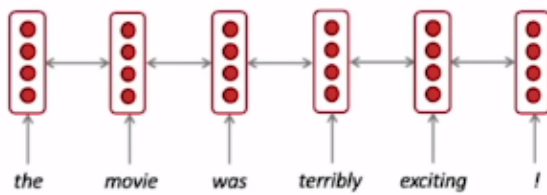
Summary:



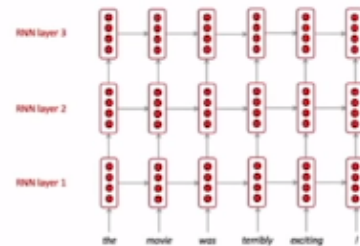
1. LSTMs are powerful but GRUs are faster



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