Long Short-Term Memory (LSTM)

Natural Language Processing

(based on revision of Chris Manning Lectures)

Announcement

- TA announcements (if any)...

Suggested Readings

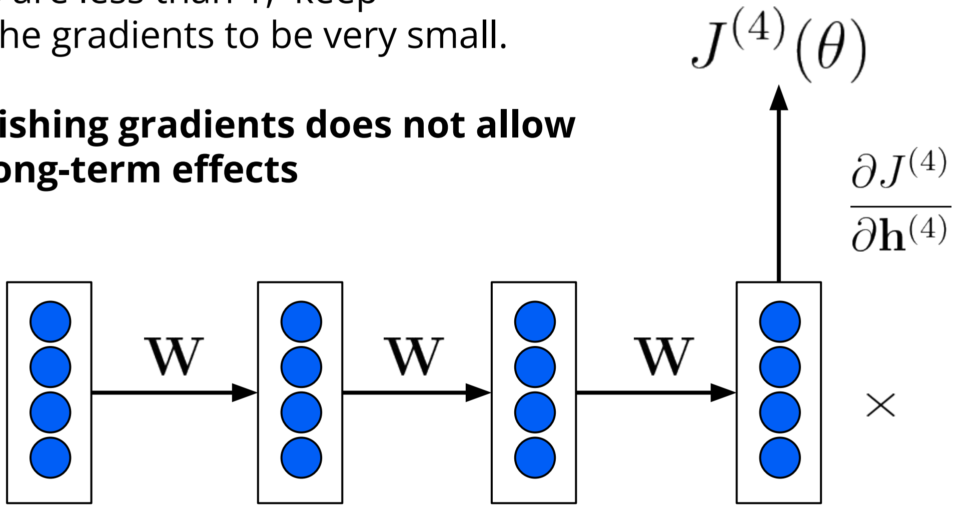
1. Sequence Modeling: Recurrent and Recursive Neural Nets (Sections 10.3, 10.5, 10.7-10.12)
2. On the difficulty of training Recurrent Neural Networks (proof of vanishing gradient problem, also about gradient clipping)
3. Understanding LSTM Networks (blog post overview)
4. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies (study how well LSTM learn syntax dependencies)
5. Long short-term memory (original LSTM paper)
6. "Learning to Forget: Continual Prediction with LSTM (actually many modern LSTM ideas come from this paper)
7. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation (original GRU paper)
8. Massive Exploration of Neural Machine Translation Architectures (studying how many layers RNN need)

Vanishing Gradients

Vanishing Gradients

Imagine if all these gradients are less than 1, keep multiplying them will cause the gradients to be very small.

The real problem is that **vanishing gradients does not allow the network to learn any long-term effects**



$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \times \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \times \frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}}$$

Example: Effect of vanishing gradients

- **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 ________"*

- To learn from this training example, the RNN-LM needs to model the dependency between "tickets" on the 7th step and the target word "tickets" at the end.

- But if gradient is small, the model can't learn this dependency. So, the model is unable to predict similar long-distance dependencies at test time.

Example: Effect of vanishing gradients

- **LM task**: "*The writer of the books _______*" (possible words: is, are)
- **Correct answer**: *The writer of the books is planning a sequel*
- **Syntactic** recency: The writer of the book 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 [Linzen et al. 2016]

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

How about exploding gradient?

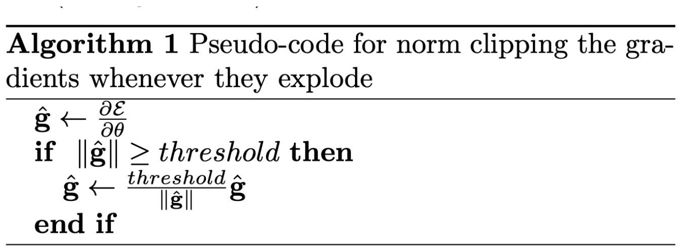
- If the gradient becomes too big, the SGD update can easily overshoot:

$$\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla_{\theta} J(\theta)$$

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

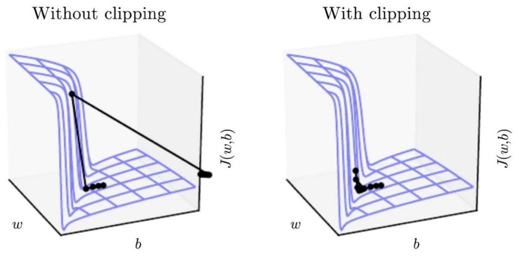
Gradient Clipping [Pascanu et al., PMLR 2013]

Idea: if the norm of the gradient is greater than some threshold, scale it down before applying update.



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
```



- This introduces the hyperparameter "**threshold**", anyhow, the authors didn't found much sensitivities - "*The proposed clipping is simple to implement and computationally efficient, but it does however introduce an additional hyper-parameter, namely the threshold. One good heuristic for setting this threshold is to look at statistics on the average norm over a sufficiently large number of updates. In our experiments we have noticed that for a given task and model size, training is not very sensitive to this hyperparameter and the algorithm behaves well even for rather small thresholds.*"
- Another easier method is simply to set a capping value, e.g., 10 (there is really no literature studying this value) but once again, monitoring the average gradient value is a good start.

On the difficulty of training recurrent neural networks, Pascanu et al. 2013, https://arxiv.org/abs/1211.5063

How to fix vanishing gradients?

- As hinted earlier, vanishing gradients cause the model **inability to learn long-term relationships**
- Instead of trying to fix vanishing gradients which is difficult, can we try to **preserve long-term relationships better**?
- How about a RNN with a separate **memory**?

Long Short-Term Memory

Long Short-Term Memory [Hochreiter and Schmidhuber, Neu.Comp. 1997]

A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem. Everyone cites that paper but really a crucial part of the modern LSTM is from Gersetal.(2000) :-)

Forget gate: controls what is kept vs. forgotten, from previous cell state

Input gate: controls what parts of the new cell contents are written to cell

Output gate: controls what parts of 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 content from last cell state, write ("input") some new cell state

Hidden state: read ("output") some content from the cell

$$\mathbf{f}^{(t)} = \sigma(\mathbf{W}_f\mathbf{h}^{(t-1)} + \mathbf{U}_f\mathbf{x}^{(t)} + \mathbf{b}_f)$$
$$\mathbf{i}^{(t)} = \sigma(\mathbf{W}_i\mathbf{h}^{(t-1)} + \mathbf{U}_i\mathbf{x}^{(t)} + \mathbf{b}_i)$$
$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_o\mathbf{h}^{(t-1)} + \mathbf{U}_o\mathbf{x}^{(t)} + \mathbf{b}_o)$$

$$\tilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{W}_c\mathbf{h}^{(t-1)} + \mathbf{U}_c\mathbf{x}^{(t)} + \mathbf{b}_c)$$
$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \circ \tilde{\mathbf{c}}^{(t)}$$
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh(\mathbf{c}^{(t)})$$

Sigmoid function: all gate values are between 0 and 1

All these vectors are of same length

Gates are applied using element-wise (Hadamard product)

Long short-term memory, Hochreiter and Schmidhuber, 1997. https://www.bioinf.jku.at/publications/older/2604.pdf
Learning to Forget: Continual Prediction with LSTM, Gers, Schmidhuber, and Cummins, 2000. https://dl.acm.org/doi/10.1162/089976600300015015

How does LSTM solve vanishing gradients?

- LSTM makes it easier to **preserve information over many time steps**
  - E.g., if the forget gate is set to 1 and the input gate set to 0, then the information of that cell is preserved indefinitely

- **Another intuition** - it provides several routes for the gradients to flow (very similar to **residual/skip connections**)

- Note that LSTM **does not guarantee** that there will be no vanishing/exploding gradients, but it simply alleviate the long-term dependencies problem

LSTMs: real-world success

- In **2013-2015**, LSTMs started achieving **state-of-the-art** results
  - Successful tasks including handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominanch approach for most NLP tasks

- Now (**2021**), other approaches (e.g., Transformers) have become dominant for many tasks
  - For example, in WMT (a Machine Translation conference + competition)
    - In WMT 2016, the summary report contains “RNN” 44 times
    - In WMT 2019: “RNN” 7 times, “Transformer” 105 times
    - In WMT 2020: “RNN” 0 time, “Transformer” 73 times

GRU [Cho et al., EMNLP 2014]

Proposed by Cho et al. 2014 as a simpler alternative to LSTM.

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

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

$$\mathbf{u}^{(t)} = \sigma(\mathbf{W}_u \mathbf{h}^{(t-1)} + \mathbf{U}_u \mathbf{x}^{(t)} + \mathbf{b}_u)$$
$$\mathbf{r}^{(t)} = \sigma(\mathbf{W}_r \mathbf{h}^{(t-1)} + \mathbf{U}_r \mathbf{x}^{(t)} + \mathbf{b}_r)$$

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

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

$$\tilde{\mathbf{h}}^{(t)} = \tanh(\mathbf{W}_h(\mathbf{r}^{(\mathbf{t})} \circ \mathbf{h}^{(t-1)}) + \mathbf{U}_h \mathbf{x}^{(t)} + \mathbf{b}_h)$$
$$\mathbf{h}^{(t)} = (1 - \mathbf{u}^{(t)}) \circ \mathbf{h}^{(t-1)} + \mathbf{u}^{(t)} \circ \tilde{\mathbf{h}}^{(t)}$$

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

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

LSTM vs. GRU (Gated Recurrent Units)

- Many other variants but LSTM and GRU seem most widely-used
- GRU is **quicker to compute**
- No conclusive evidence which one is better
- LSTM is a **good default choice**
- **Rule of thumb**: start with LSTM; switch to GRU if you want something more efficient (e.g., for real-time)

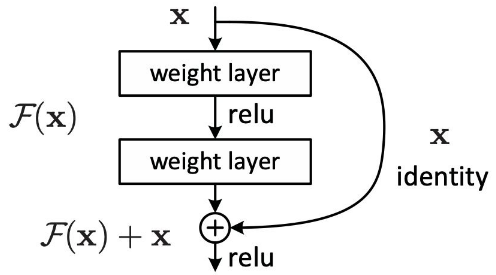
Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **very deep** ones. (but not much as RNN though!)
    - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
    - Thus, lower layers are learned very slowly (hard to train)

- **Solution**: lots of new deep feedforward/convolutional architectures that add more **direct connections** (thus allowing the gradient to flow)

For example:

- Residual/skip connection a.k.a "ResNet"
- The identity connection preserves information
- This makes deep networks much easier to train



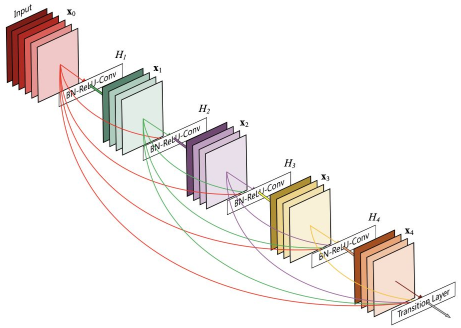
"Deep Residual Learning for Image Recognition", He et al, 2015. https://arxiv.org/pdf/1512.03385.pdf

Is vanishing/exploding gradient just a RNN problem?

Many other more **aggressive** residual connections

DenseNet

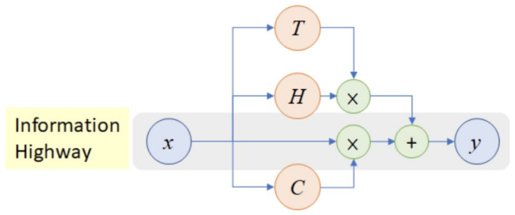
- Directly connect each layer to all future layers!



"Densely Connected Convolutional Networks", Huang et al, 2017. https://arxiv.org/pdf/1608.06993.pdf

HighwayNet

- Similar to residual connections, but the identity connection vs. the transformation layer is controlled by a dynamic gate
- Inspired by LSTMs, but can be applied to deep feedforward/CNN



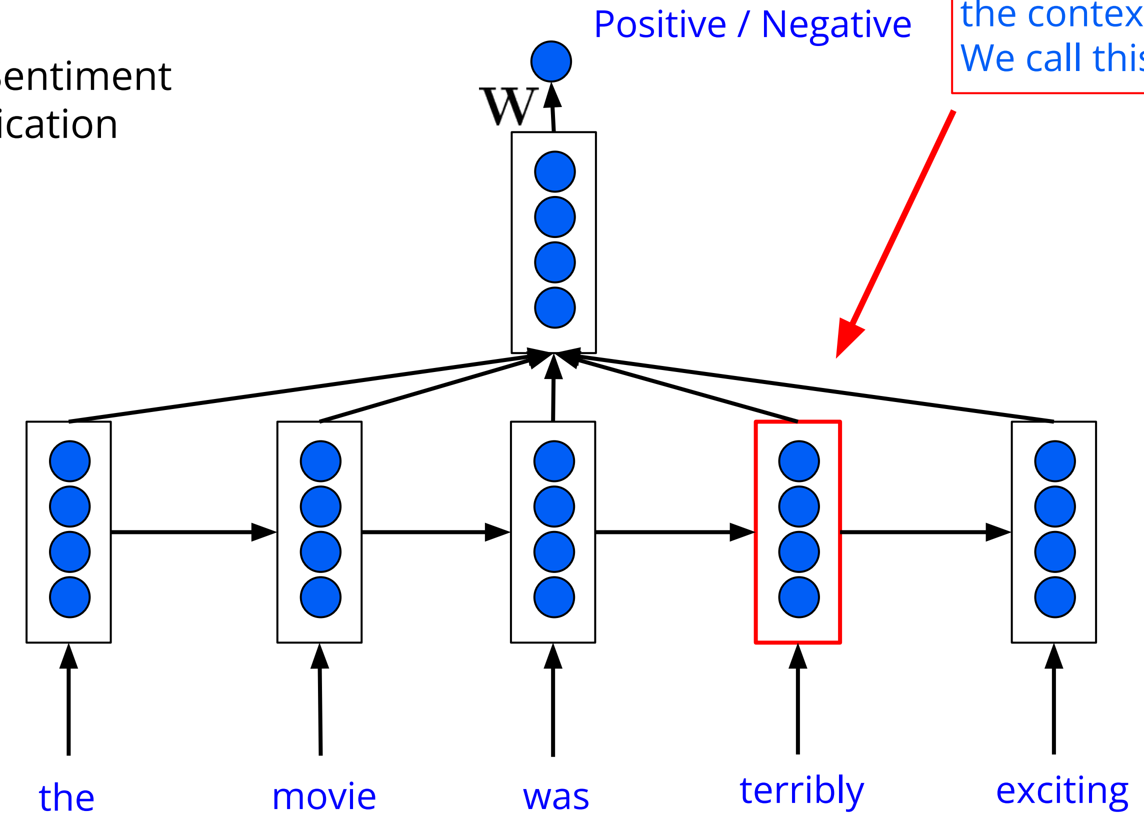
Highway Circuit

"Highway Networks", Srivastava et al, 2015. https://arxiv.org/pdf/1505.00387.pdf

Bidirectionality and Multi-Layer RNNs

Why we need bidirectionality?

Task: Sentiment Classification



We can regard this hidden state as a representation of the word "terribly" in the context of this sentence.
We call this a contextual representation.

These contextual representations only contain information about the left context (e.g., "the movie was").

What about right context?

In this example, "exciting" is in the right context and this modifies the meaning of "terribly" (from negative to positive)

Bidirectional RNNs

Forward RNN:

$$\overrightarrow{\mathbf{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\mathbf{h}}^{(t-1)}, \mathbf{x}^{(t)})$$

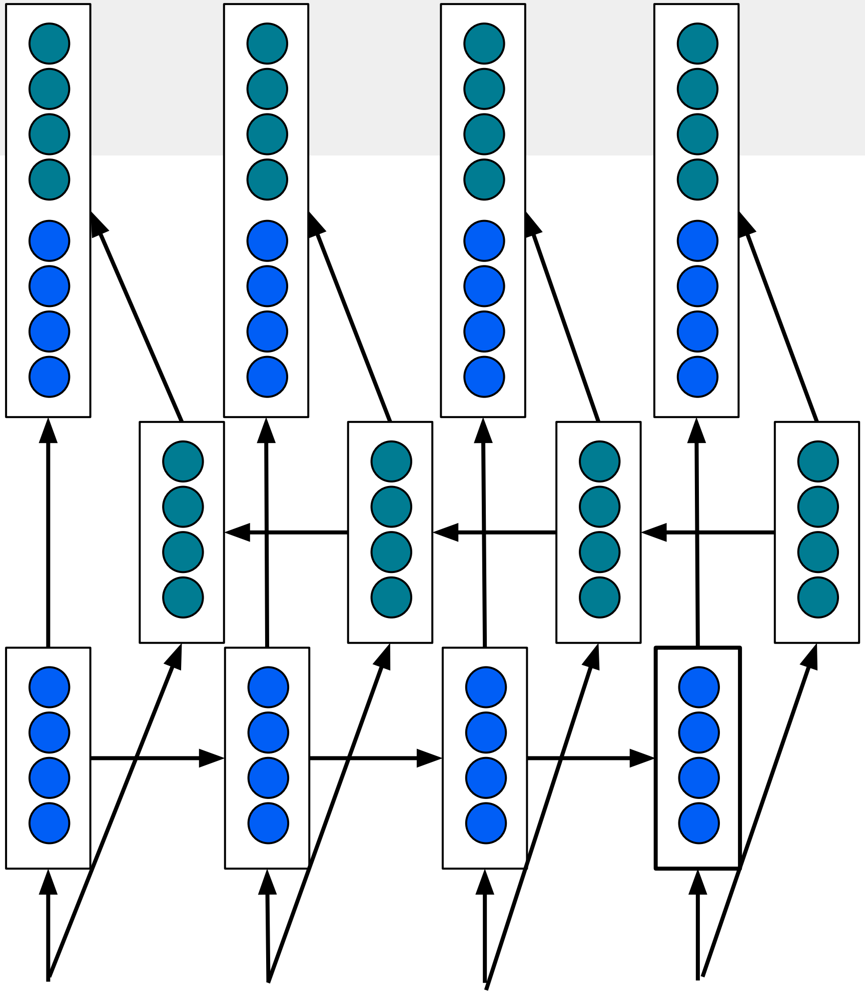
Backward RNN:

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

Concatenated hidden states:

$$\mathbf{h}^{(t)} = [\overrightarrow{\mathbf{h}}^{(t)}; \overleftarrow{\mathbf{h}}^{(t)}]$$

Note that this "RNN" can refer to vanilla RNN or LSTM or GRU. Also note that a bidirectional arrow is used instead to simplify the diagram.



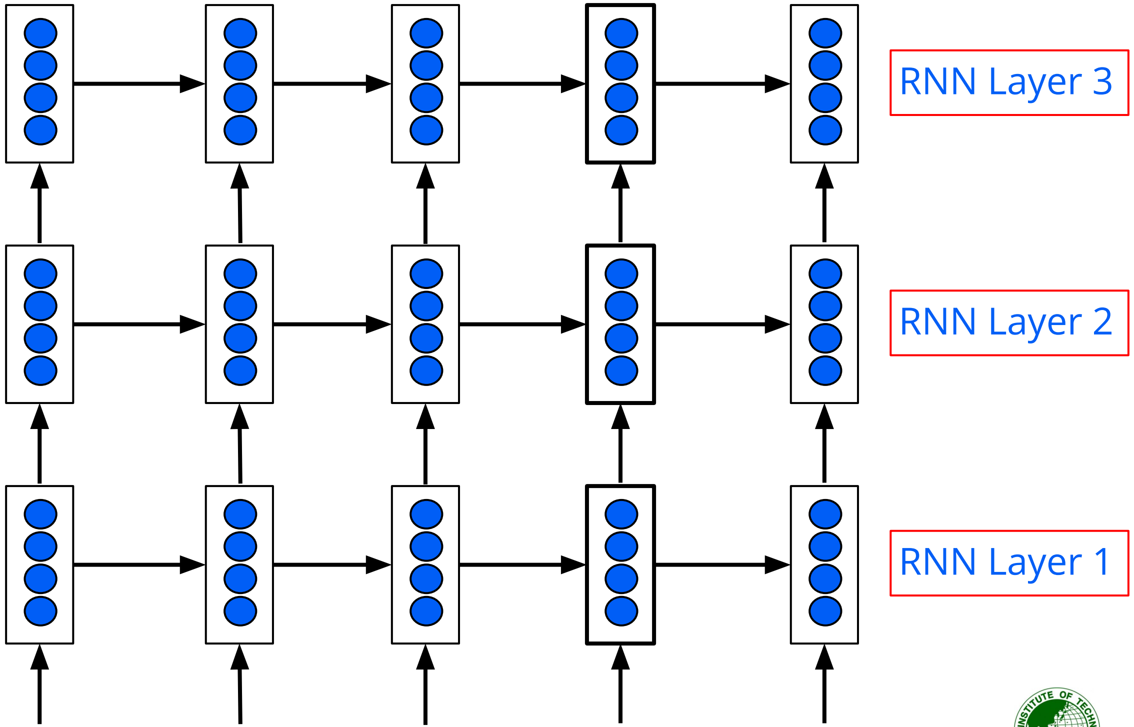
Bidirectional RNNs

- Bidirectional RNN should always be used when possible
- Note that bidirectional RNN are ONLY applicable if you have access to the **entire input sequence**
  - For example, **not applicable to LM**, because LM only have left context

- For example, **BERT** (Bidirectional Encoder Representations from Transformers) is a powerful **pretrained contextual representation system built on bidirectionality**.
  - Coming up in our future lectures!

Multi-layer RNNs (Stacked RNNs)

- **High-performing RNNs** are often multi-layer
  - But are not as deep as convolutional or feedforward
- Works well with **residual / skip connections**
- Also called **Stacked RNNs.**
- Lower layers focus on lower level features, e.g., syntax, and higher layers focus on semantics
- In a 2017 paper, Britz et al. find that for Neural Machine Translation, **2 to 4 layers** are good for the RNN encoder and **4 layers** is best for the RNN decoder.
- **Transformer** like BERT has **12 or 24 layers** variants



"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

Summary

- **LSTMs** are powerful
  - Still used till today!
  - Use **bidirectionality** when possible (almost always)
  - **Multi-layer RNNs** are powerful, but make sure to add skip connection
- **Clip** your gradients (almost always)
- Use **skip connections** (almost always)
  - You will see it in **Transformers** and in almost any modern architectures