Neural Language Models

Large Language Models: Introduction and Recent Advances

ELL881 · AIL821

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Slides are adopted from the Stanford course 'NLP with DL' by C. Manning

Released on July 30, 2024

PyTorch Blog

PyTorch releases **torchchat** !

Torchchat is a library that facilitates the seamless and performant running of LLMs across laptops, desktops, and mobile devices.

It provides important functions such as **export**, **quantization** and **eval** in a way that's easy to understand providing an end-to-end story for those who want to build a local inference solution.

pytorch/**torchchat**

Run PyTorch LLMs locally on servers, desktop and mobile

40 Contributors · 60 Issues · 777 Stars · 32 Forks

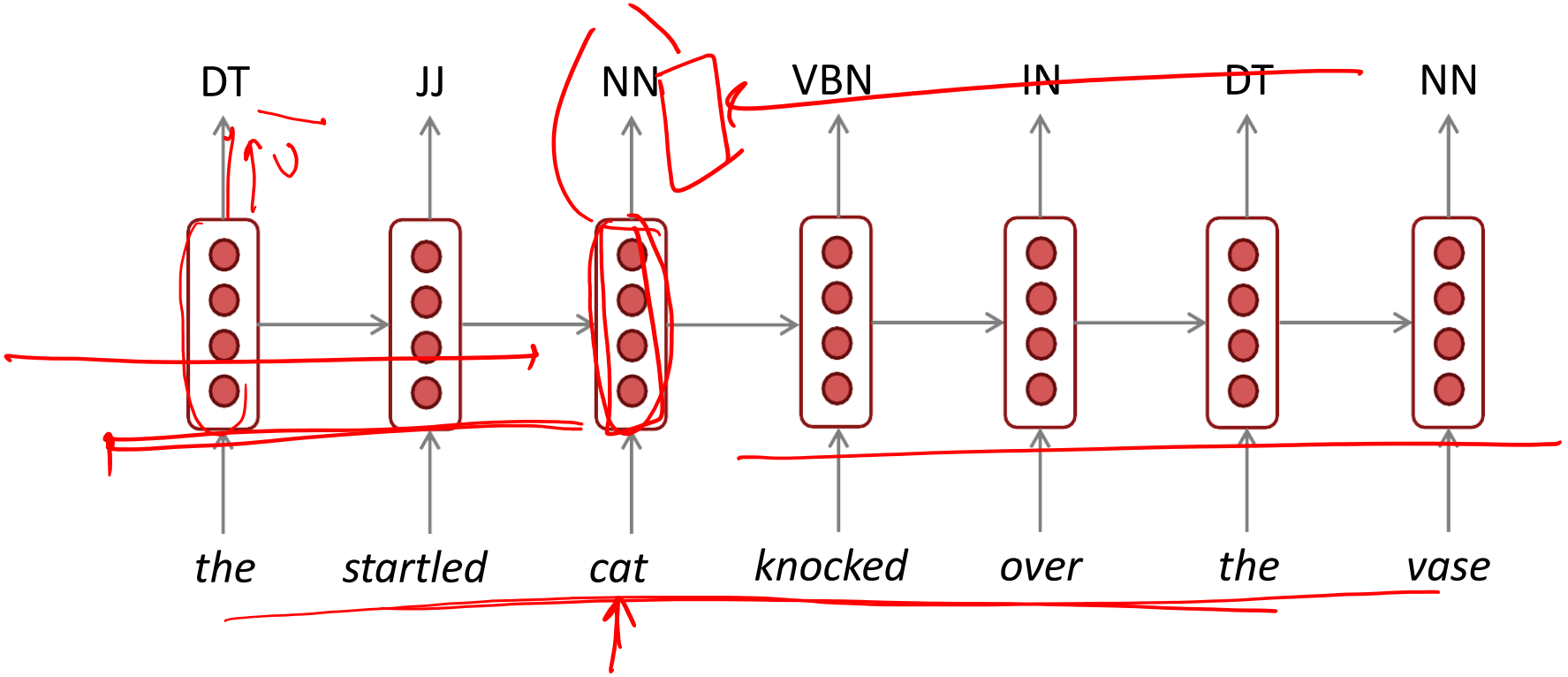
Torchchat achieves **> 8 Tokens/sec** inference rate for **Llama-3 8B Instruct** on the **Samsung Galaxy S23** and **iPhone** using 4-bit GPTQ quantization.

With it's release, **running LLMs locally on any device is going to get easier!**

Uses of RNNs

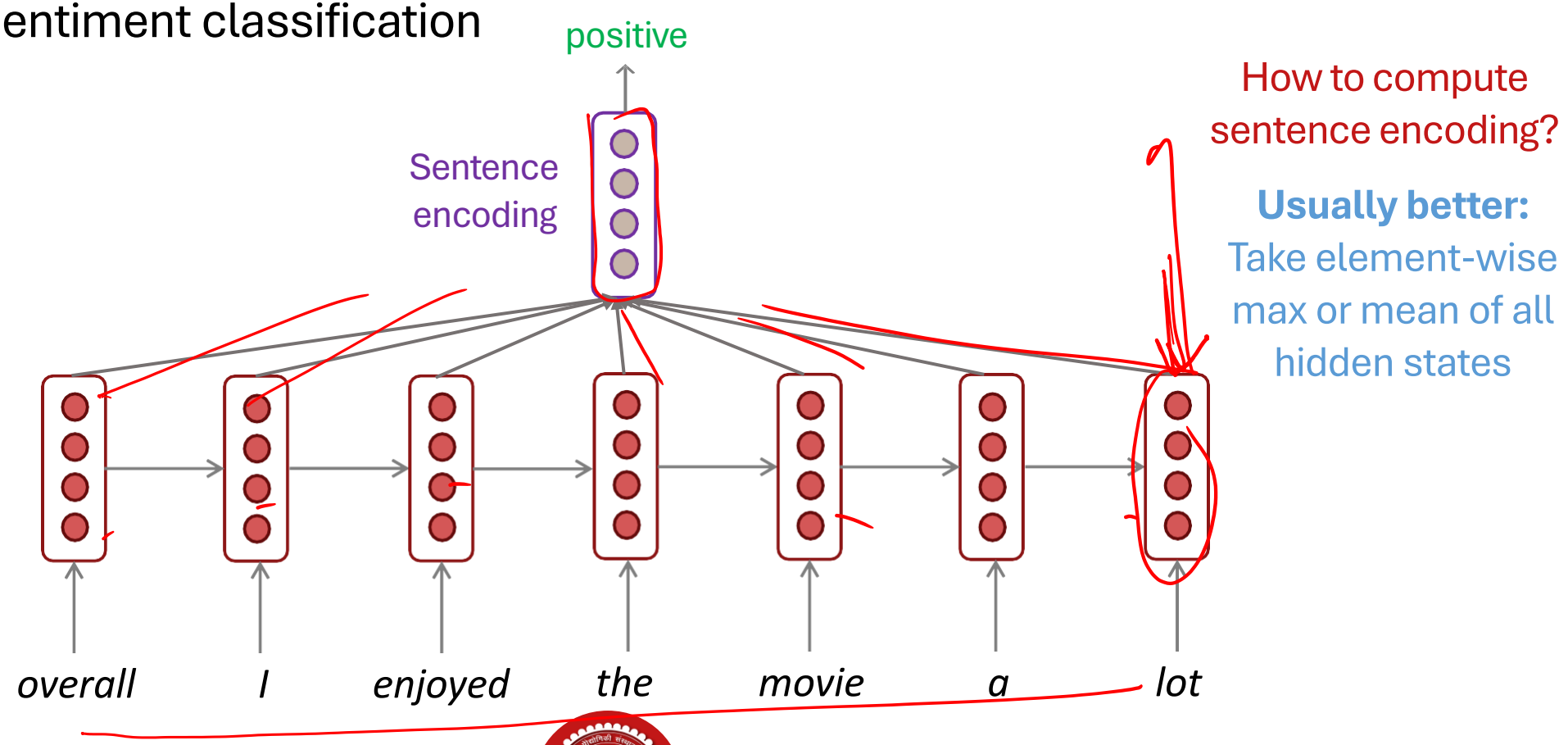
RNNs Can Be Used for Sequence Tagging

- **Example:** for part-of-speech tagging, named entity recognition



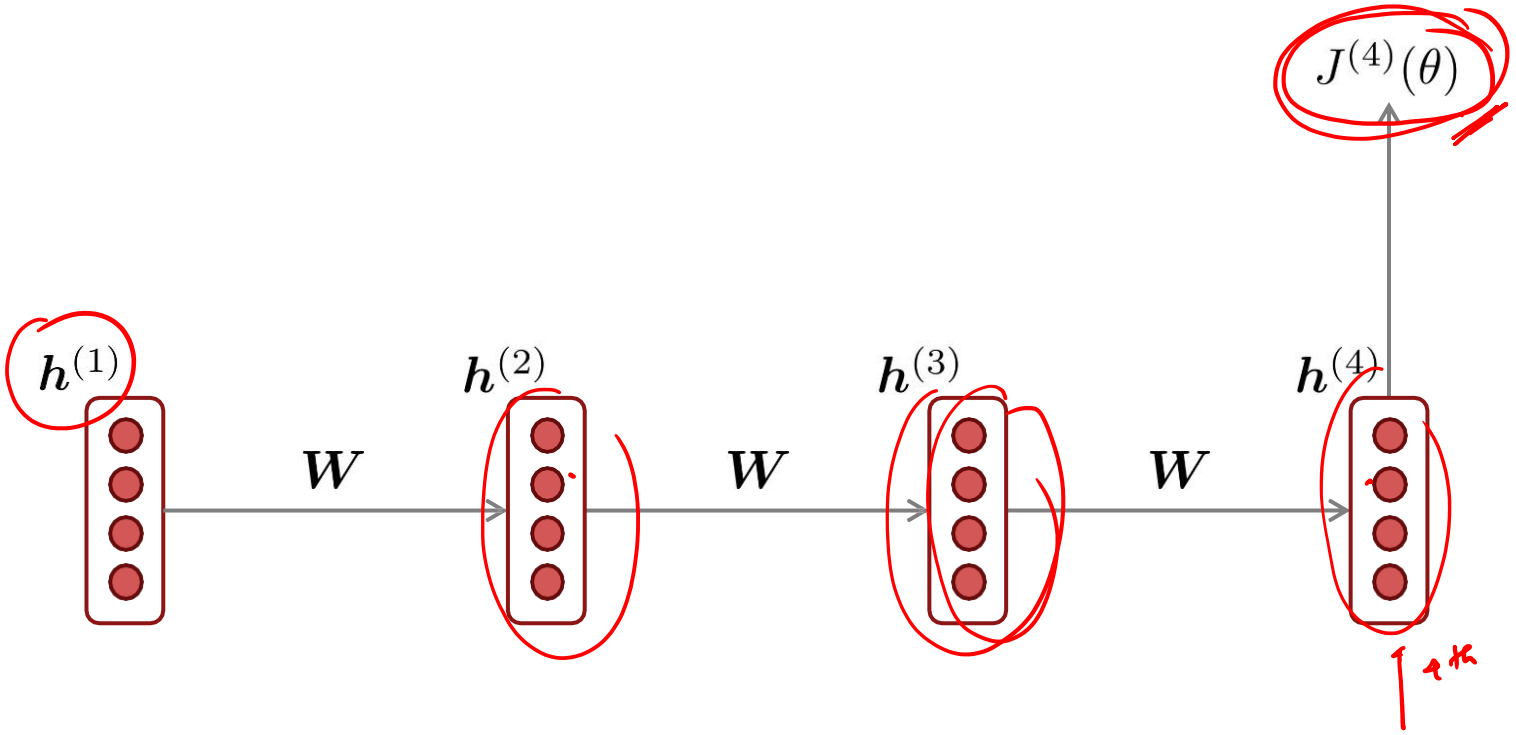
RNNs Can Be Used for Sentence Classification

- **Example:** for sentiment classification

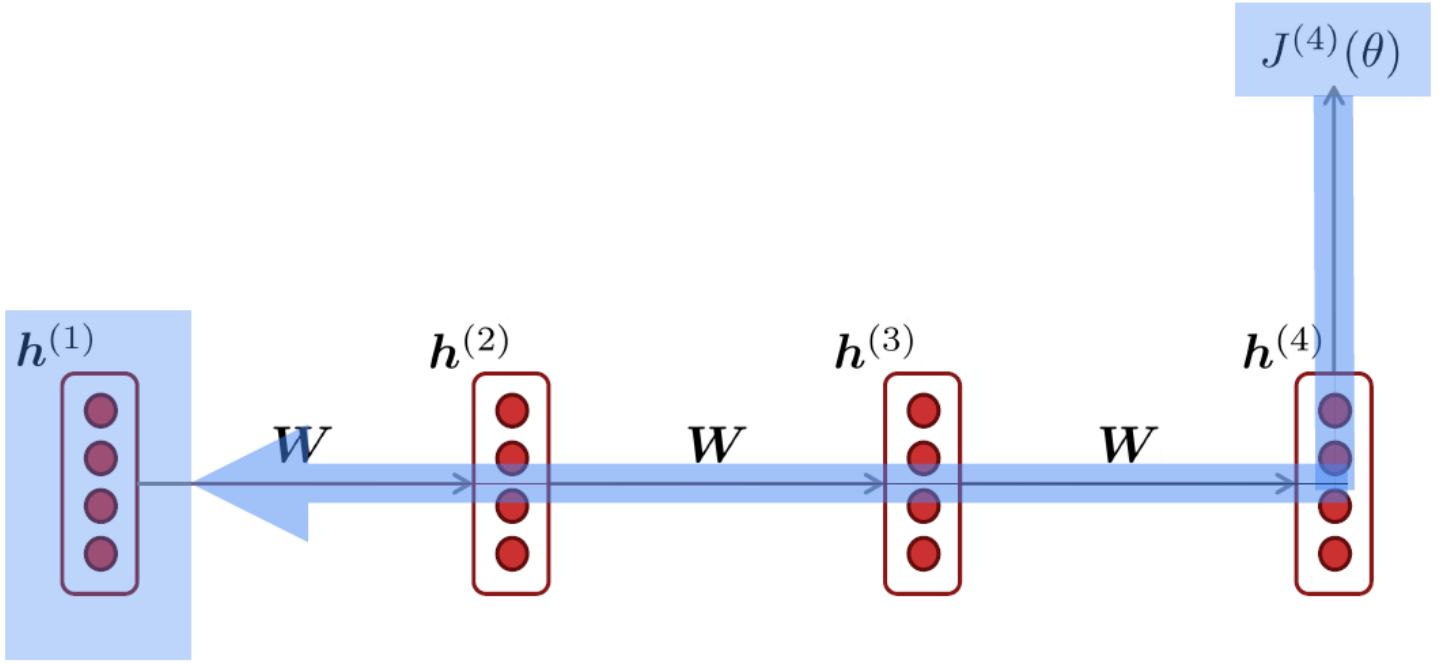


Problems with RNNs

Vanishing and Exploding Gradients

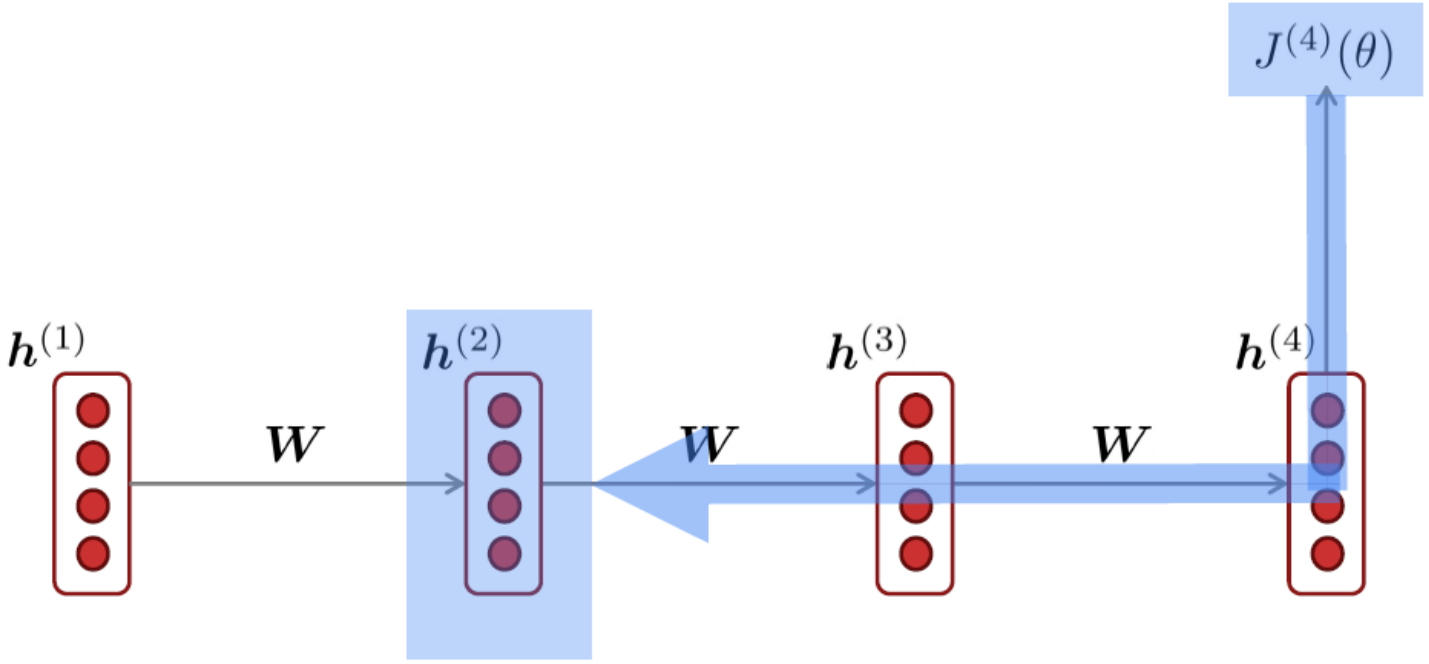


Vanishing Gradient Intuition



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \; ?$$

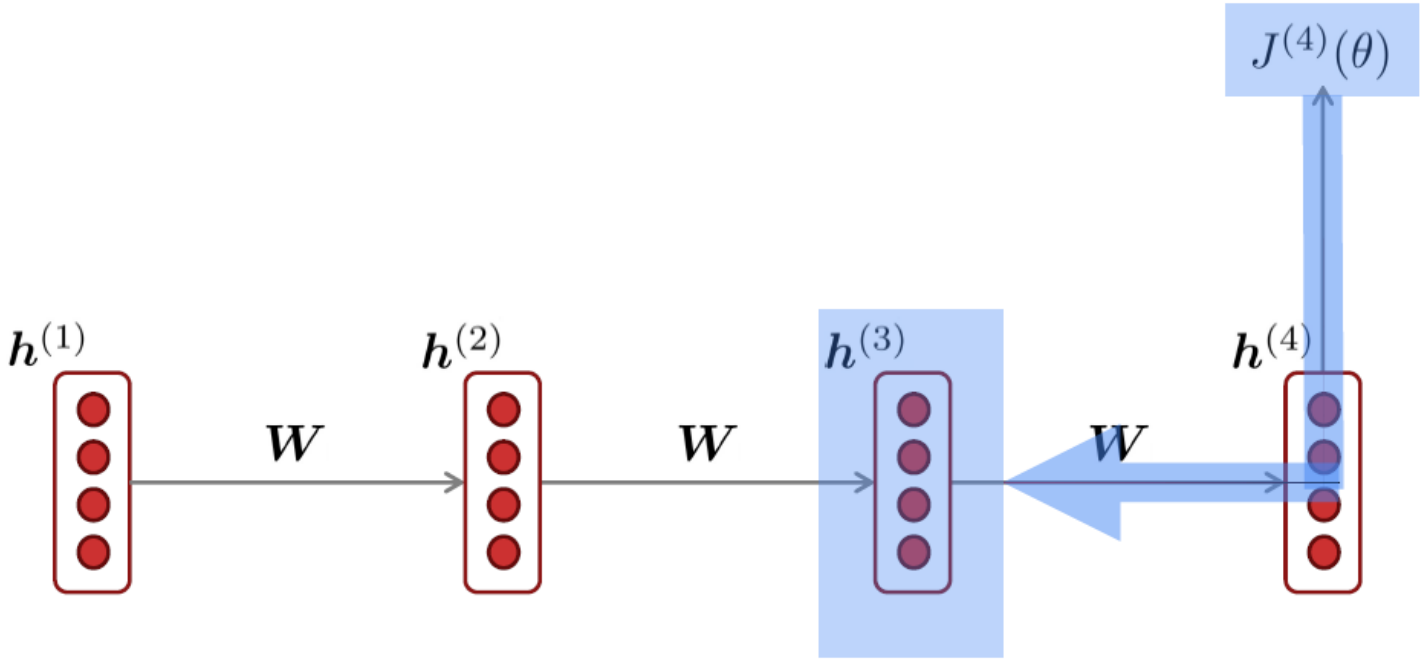
Vanishing Gradient Intuition



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

Chain Rule!

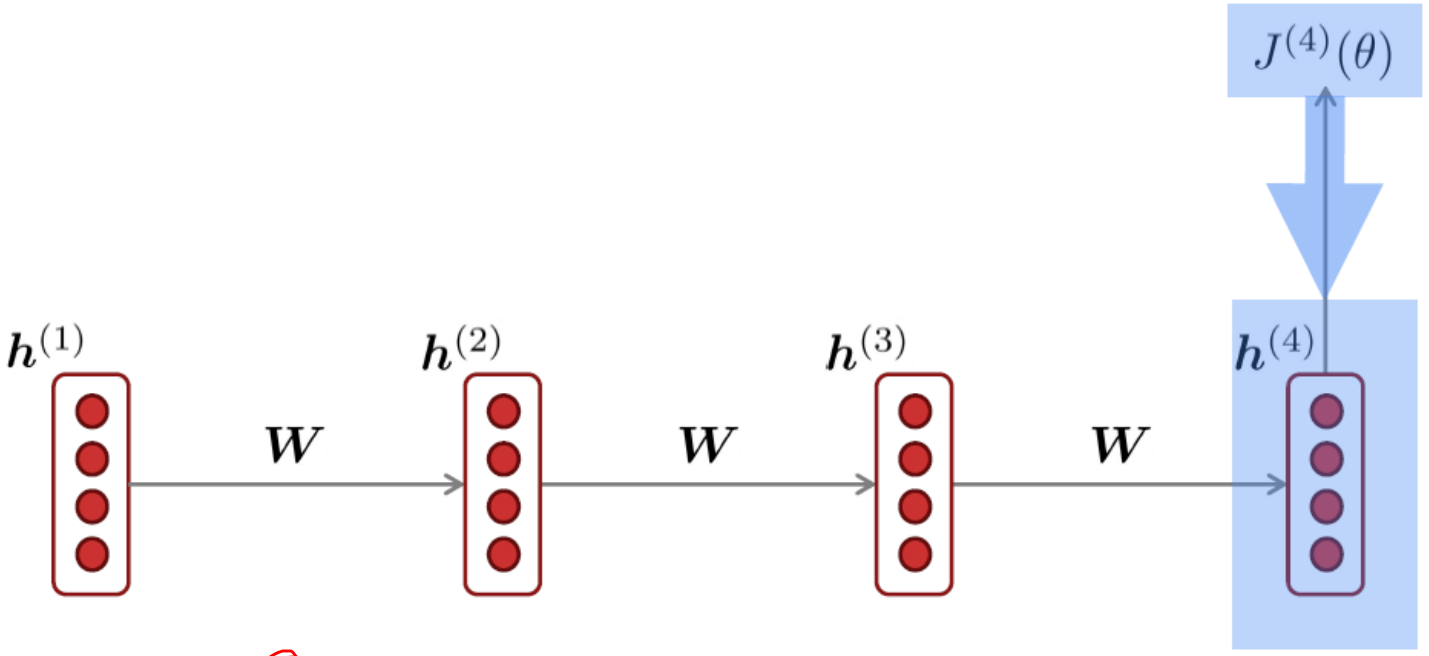
Vanishing Gradient Intuition



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

Chain Rule!

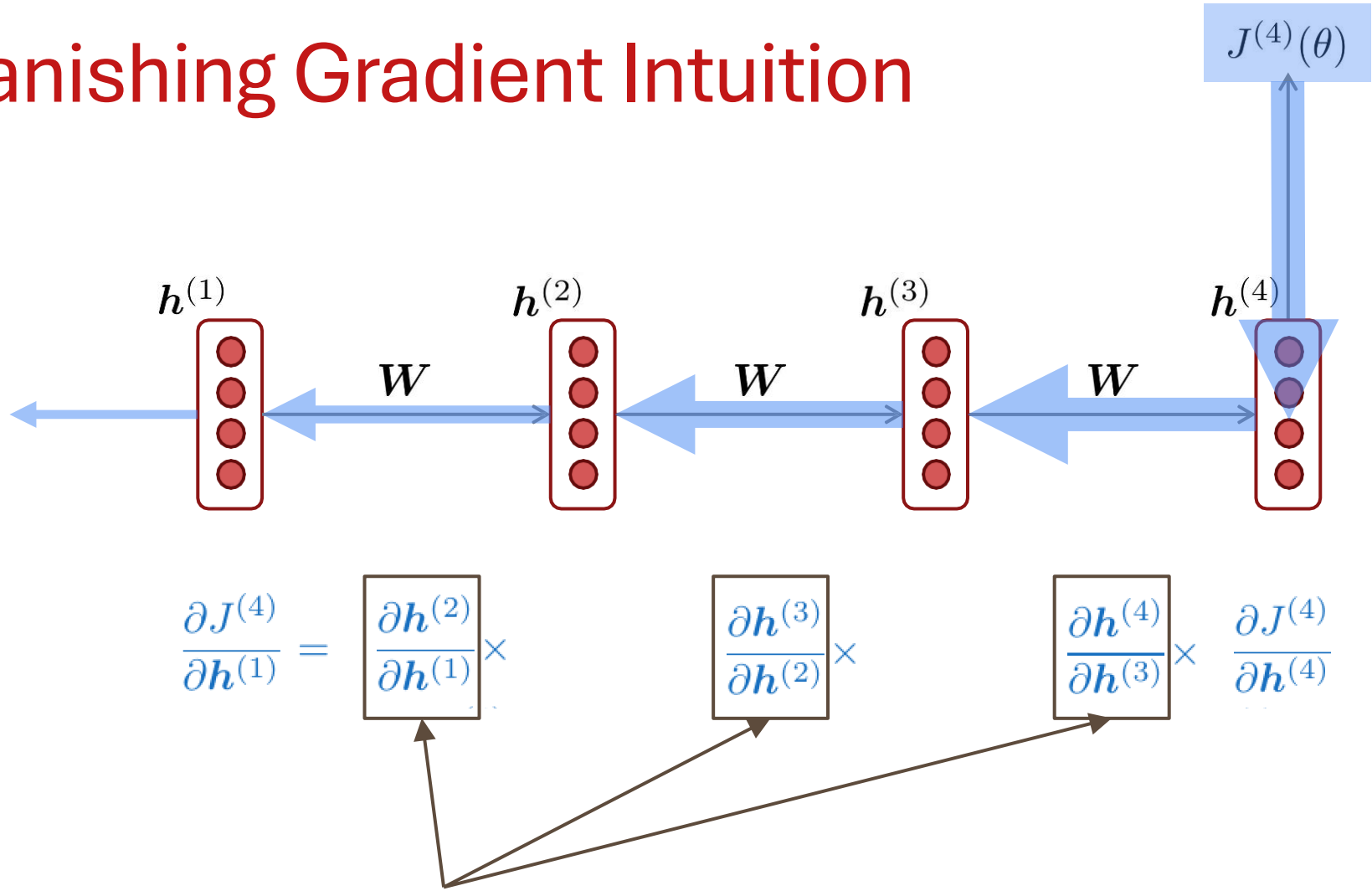
Vanishing Gradient Intuition



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

Chain Rule!

Vanishing Gradient Intuition



Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Why is Vanishing Gradient a Problem?

Effect of Vanishing Gradient on RNN-LM

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

How to Fix the Vanishing Gradient Problem?

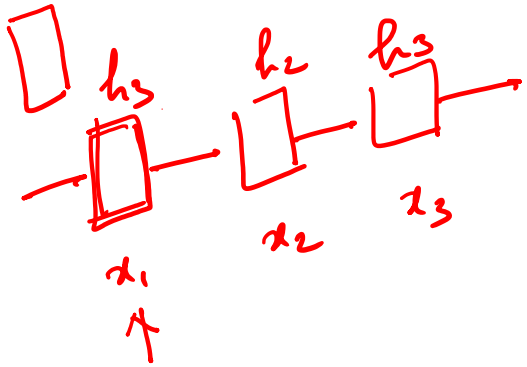
- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*.
- In a vanilla RNN, the hidden state is constantly being rewritten

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

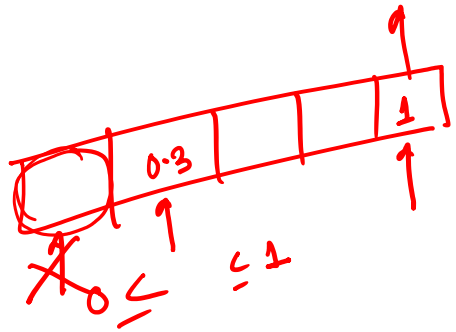
- How about an RNN with separate memory which is added to?
  - LSTMs
- And then: Creating more direct and linear pass-through connections in model
  - Attention, residual connections, etc.

LSTMs & GRUs

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 (gates are calculated things whose values are probabilities)
  - 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

LSTM

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:

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

Input gate: controls what parts of the new cell content 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, and write ("input") some new cell content

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

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

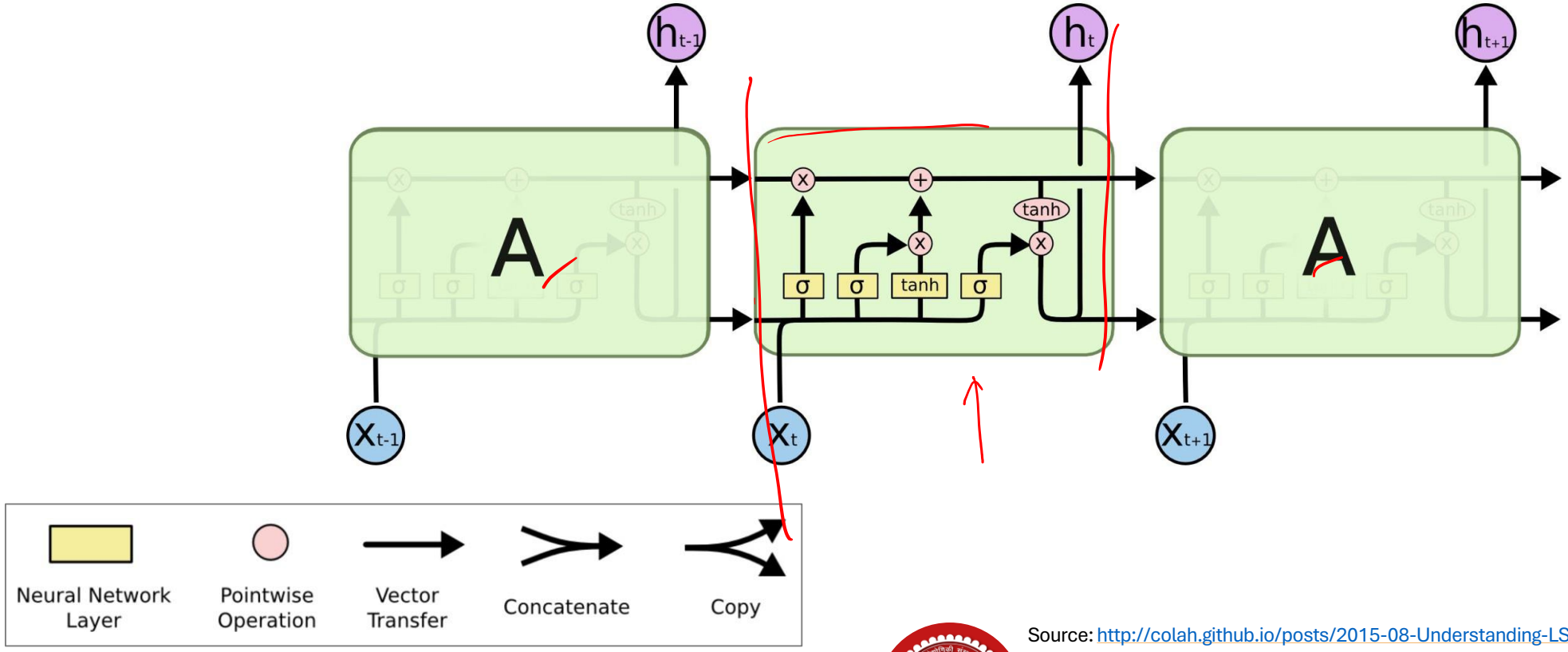
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



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

How Does LSTM Solve Vanishing Gradients?

- The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
  - For example, if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
  - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
  - In practice, you get about 100 timesteps rather than about 7
- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.
- There are also alternative ways of creating more direct and linear pass-through connections in models for long distance dependencies.

Is Vanishing/Exploding Gradient Just an RNN Problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional neural networks), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (i.e., are hard to train)
- **Another solution:** lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

For example:

- Residual connections aka "ResNet"
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train

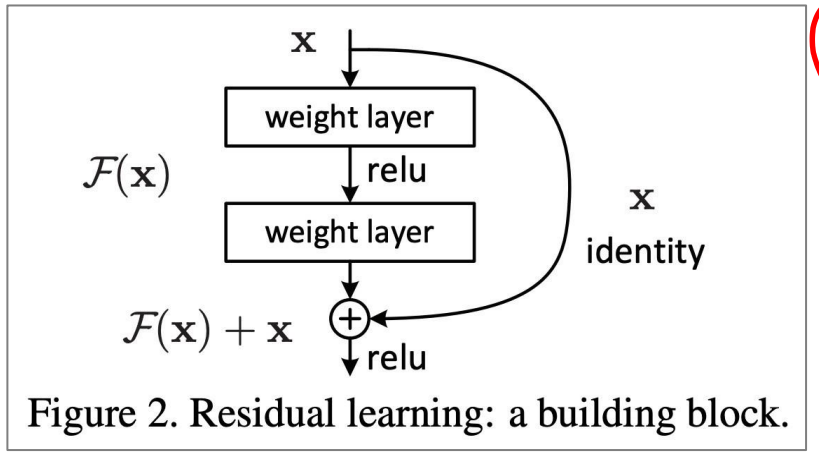
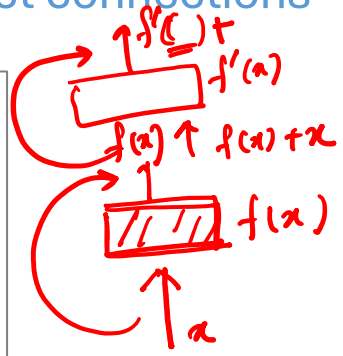
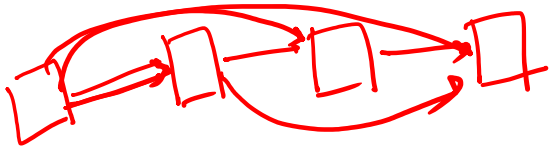


Figure 2. Residual learning: a building block.



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



Is Vanishing/Exploding Gradient Just an RNN Problem?

Other Methods:

- Dense connections aka "DenseNet"
- Directly connect each layer to all future layers!

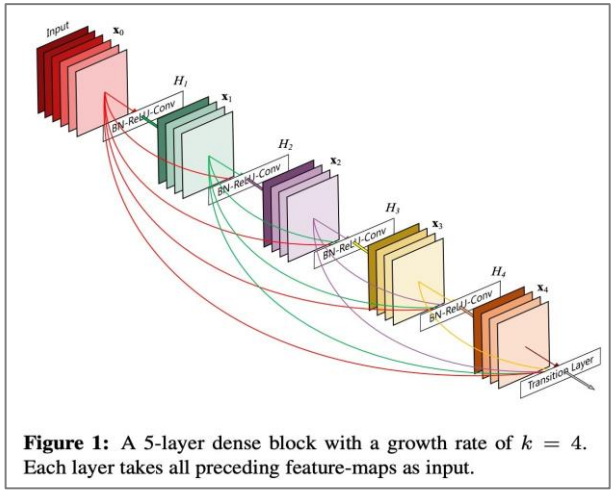
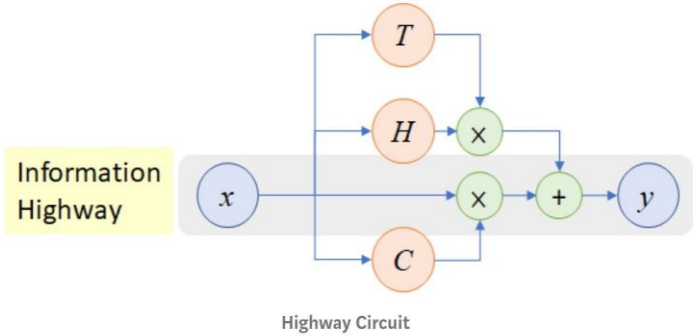


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- Highway connections aka "HighwayNet"
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
- Inspired by LSTMs, but applied to deep feedforward/convolutional networks



Highway Circuit

Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix.

Gated Recurrent Units (GRUs)

- 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).

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)

Update gate: 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.

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

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

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

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

"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

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used.
- The biggest difference is that GRU is quicker to compute and has fewer parameters.
- There is no conclusive evidence that one consistently performs better than the other.
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data).
- **Rule of thumb:** start with LSTM, but switch to GRU if you want something more efficient.

Bidirectional RNNs

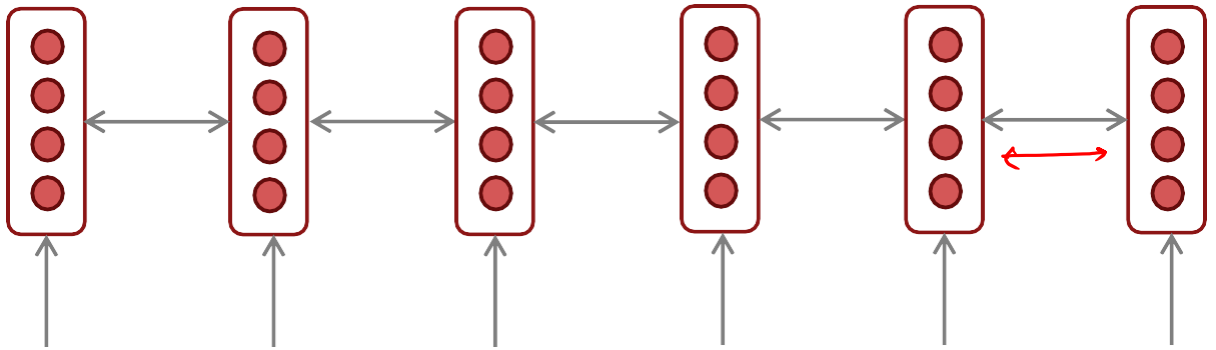
On timestep *t* :

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

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

Generally, these two RNNs have separate weights

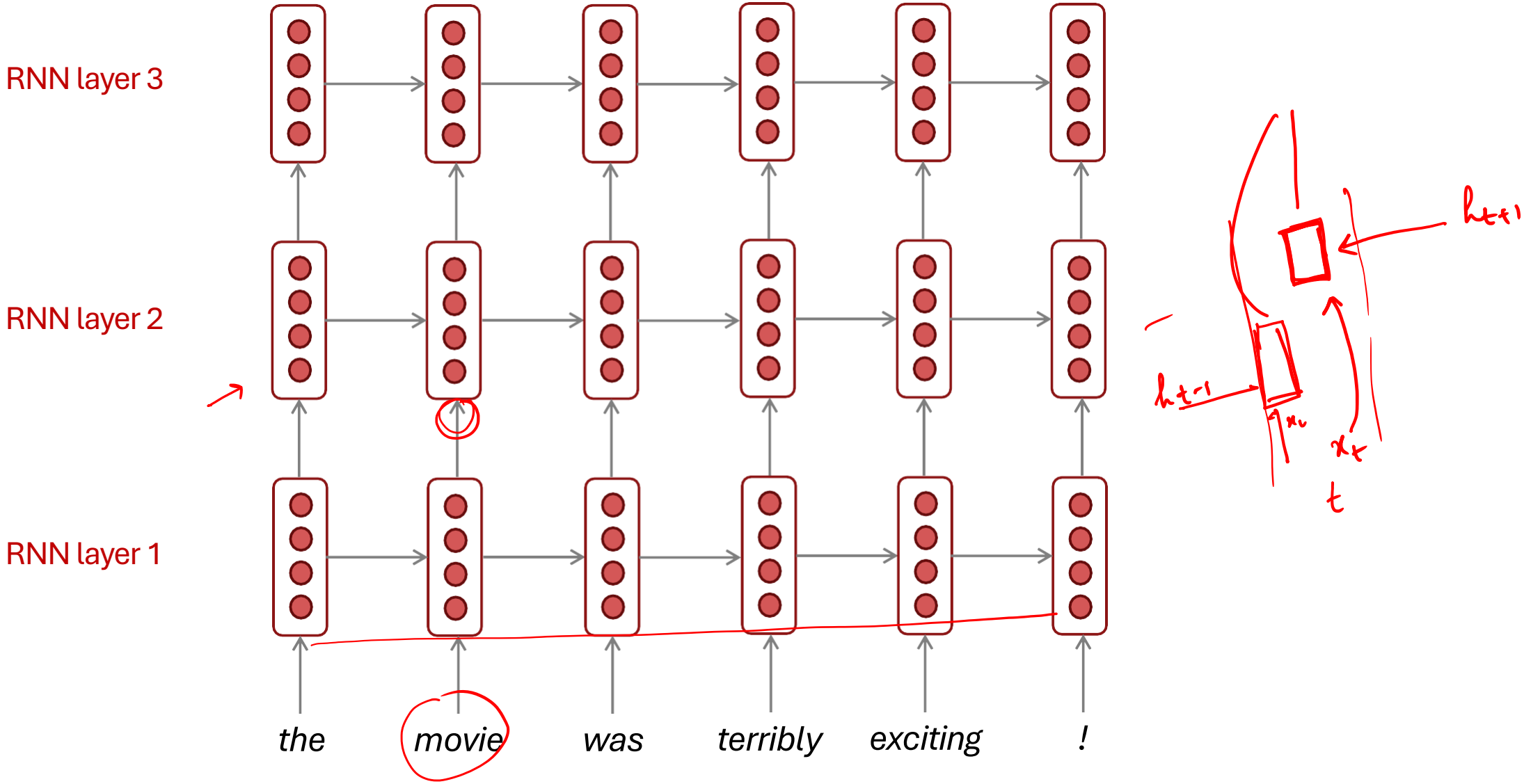
Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$



Multi-layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps)
- We can also make them “deep” in another dimension by applying multiple RNNs – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called *stacked RNNs*.

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



LSTMs: Real-world Success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks
- Now (2019–2024), Transformers have become dominant for all tasks
  - For example, in WMT (a Machine Translation conference + competition):
    - In WMT 2014, there were 0 neural machine translation systems (!)
    - In WMT 2016, the summary report contains "RNN" 44 times (and these systems won)
    - In WMT 2019: "RNN" 7 times, "Transformer" 105 times

Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
Source: "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf
Source: "Findings of the 2019 Conference on Machine Translation (WMT19)", Barrault et al. 2019, http://www.statmt.org/wmt18/pdf/WMT028.pdf