

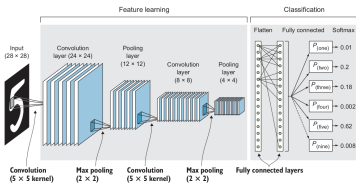
Sequence Modeling: Recurrent Neural Networks

Ju Sun

Computer Science & Engineering
University of Minnesota, Twin Cities

November 18, 2023

Recap: CNNs



(Credit: [Elgandy, 2020])

- (neuro-inspired) locality and weight sharing \implies reduced complexity (than FCNN)
- conv + pooling \implies (approx.) translation/deformation invariance (part of the learning can be avoided; see **scattering transform**)

[Bruna and Mallat, 2013, Mallat, 2016, Zarka et al., 2019])

CNNs are **not** only for images: ideal for tensors where **locality** matters

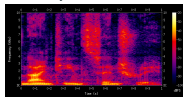
image



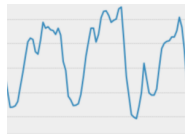
video



audio (spectrogram)



time series



Model sequences

... where directions matter

temporal sequences

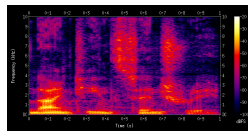
disease prognosis



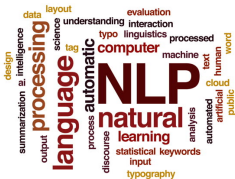
event analysis/video generation



speech to text



lexical sequences—most tasks in Natural Language Processing (NLP)



- machine translation, e.g., English \leftrightarrow Chinese
- typing/writing prediction (smart compose)
- semantic classification

Basic RNNs

Vanishing/exploding gradients

Gated RNNs

Modern RNNs

Suggested reading

Basic setup

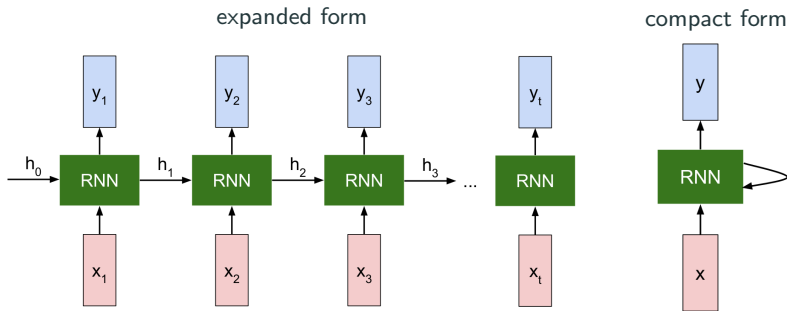
A sequence: $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \dots x_{n-1}$

A **state-space** model: h denotes the state, and state transition modeled by the **recurrence** formula

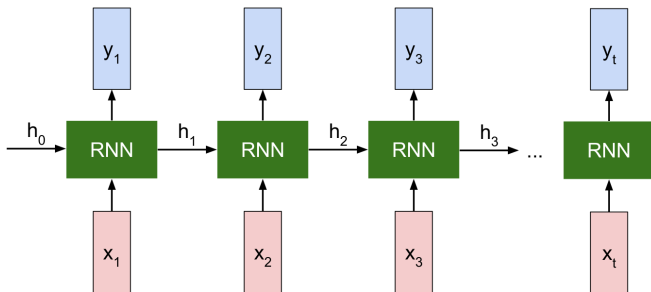
$$h_t = f_{\mathbf{w}}(h_{t-1}, x_t)$$

with **optional** output

$$y_t = g_{\mathbf{v}}(h_t)$$



A simple (vanilla) RNN



(Credit: Stanford CS231N)

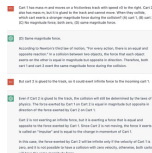
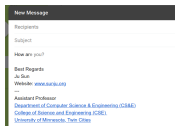
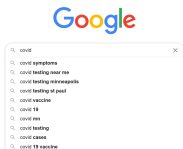
$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$

$$y_t = V_y h_t$$

W_h , W_x and V_y are shared across the sequence

A first example: language modeling

- **language modeling** is the task of predicting future words
 - ... The vaccine is effective, and COVID-19 will be **_**.
- applications: typing prediction (smart compose), machine translation, ChatGPT, etc



- (traditional) statistical formalism: given a sequence of words $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}$, compute

$$\mathbb{P} \left[\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)} \right]$$

where $\mathbf{x}^{(t+1)}$ can be any word from a vocabulary $\{\mathbf{w}_1, \dots, \mathbf{w}_N\}$, or sometimes given some text $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$

$$\mathbb{P} \left[\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)} \right] = \prod_{t=1}^T \mathbb{P} \left[\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)} \right]$$

Modern neural language modeling—1

Representing words: word embedding

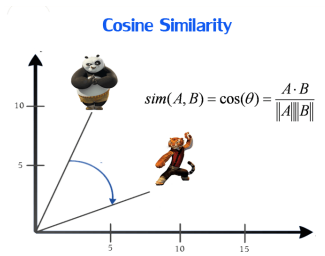
- one hot encoding

$I \mapsto [1, 0, 0, 0, \dots]$, $you \mapsto [0, 1, 0, 0, \dots]$, $we \mapsto [0, 0, 1, 0, \dots]$, \dots

- word-to-vector embedding: map words into **dense** vectors so that certain arithmetic operations are consistent with semantics



(Credit: <https://www.adityathakker.com/introduction-to-word2vec-how-it-works/>)



(Credit: <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c206>)

e.g., word2vec, BloVe, ELMo

Modern neural language modeling—2

RNN modeling: predicting the next word each time

output distribution

$$\hat{y}^{(t)} = \text{softmax}(U\mathbf{h}^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|\mathcal{V}|}$$



hidden states

$$\mathbf{h}^{(t)} = \sigma(W_h \mathbf{h}^{(t-1)} + W_e \mathbf{e}^{(t)} + \mathbf{b}_1)$$

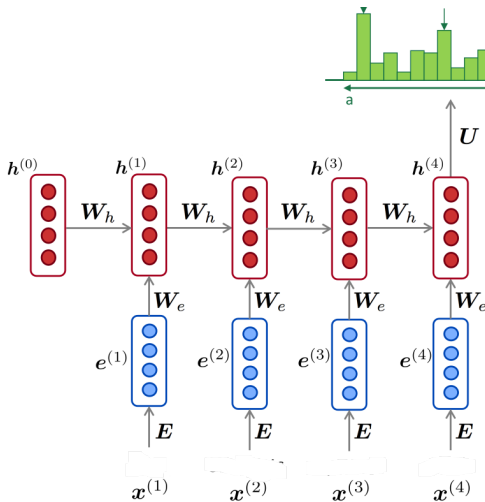
$\mathbf{h}^{(0)}$ is the initial hidden state

word embeddings

$$\mathbf{e}^{(t)} = \mathbf{E}\mathbf{x}^{(t)}$$

words / one-hot vectors

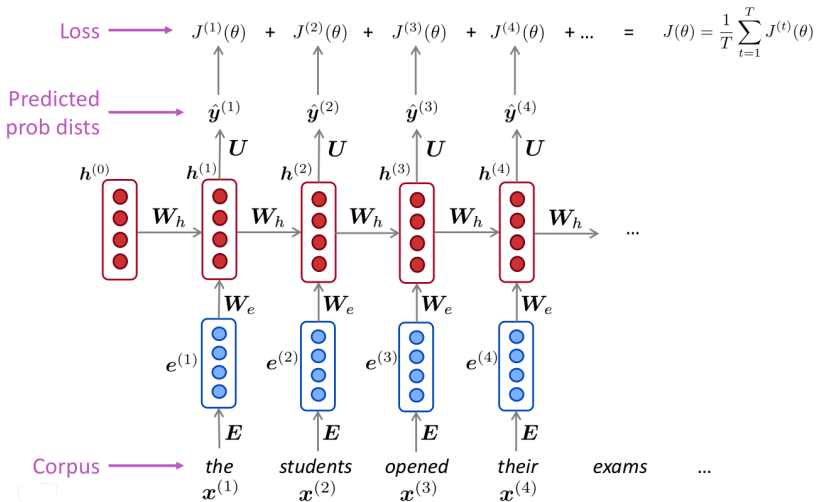
$$\mathbf{x}^{(t)} \in \mathbb{R}^{|\mathcal{V}|}$$



(Credit: adapted from Stanford CS224N)

Modern neural language modeling—3

Training the RNN model



(Credit: Stanford CS224N)

Modern neural language modeling—4

The whole training pipeline

- Step 1: collect a large corpus of text, i.e., a long sequence $\mathcal{T} = \mathbf{x}^{(1)} \rightarrow \dots \rightarrow \mathbf{x}^{(T)}$ (e.g., a sentence, a document, etc)
- Step 2: feed \mathcal{T} into the model, and compute output distribution $\hat{\mathbf{y}}^{(t)}$ for each t
- Step 3: define loss, e.g., cross entropy between $\hat{\mathbf{y}}^{(t)}$ and $\mathbf{y}^{(t)}$ (one-hot encoding of $\mathbf{x}^{(t+1)}$)

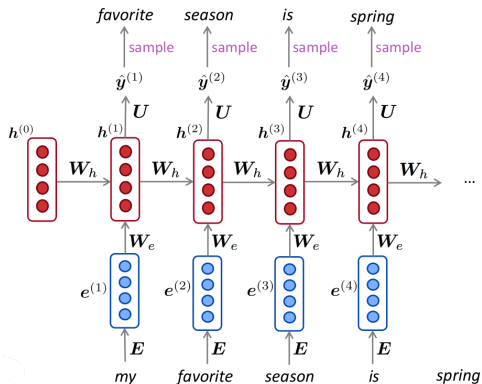
$$J^{(t)}(\boldsymbol{\theta}) = - \sum_{\mathbf{w} \in \mathcal{V}} \mathbf{y}_{\mathbf{w}}^{(t)} \log \hat{\mathbf{y}}_{\mathbf{w}}^{(t)} = - \log \hat{\mathbf{y}}_{\mathbf{x}^{(t+1)}}^{(t)}$$

- Step 4: gather and average all losses:

$$J(\boldsymbol{\theta}) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\boldsymbol{\theta})$$

- Step 5: optimization: SGD (are the summation terms iid in the objective?), etc

Test example: generate texts



(Credit: Stanford CS224N)

starting from $h^{(0)}$ and my, repeat:

- compute $\hat{y}^{(t)}$ and sample a word from the distribution
- feed the word as input to the next step

Basic RNNs

Vanishing/exploding gradients

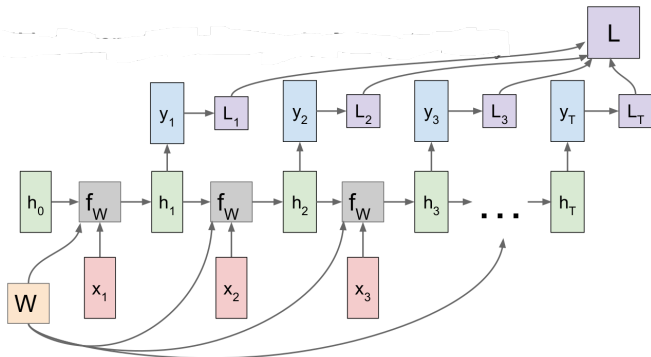
Gated RNNs

Modern RNNs

Suggested reading

How to compute gradients?

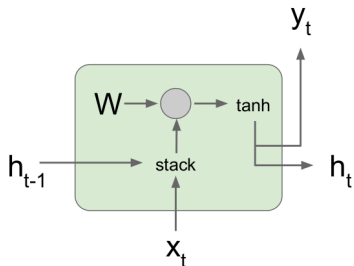
computational graph



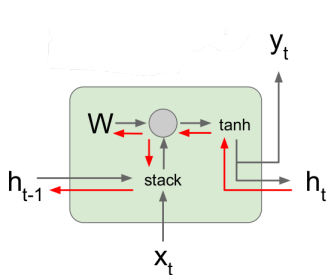
(Credit: Stanford CS231N)

- acyclic directed graph \implies auto differentiation can be applied
- W is shared across all steps!

Look into the gradient



(Credit: Stanford CS231N)



(Credit: Stanford CS231N)

$$\begin{aligned}h_t &= \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t) \\ &= \tanh\left(\mathbf{W} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix}\right) \\ \text{where } \mathbf{W} &= \begin{bmatrix} \mathbf{W}_h & \mathbf{W}_x \end{bmatrix}\end{aligned}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \text{diag}(\tanh'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)) \mathbf{W}_h$$

where

$$\tanh'(x) = 1 - \tanh^2(x)$$

Look into the gradient

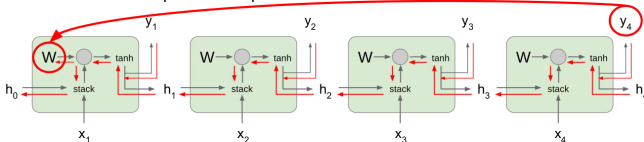
$$L = \sum_{t=1}^T L_t \implies \text{total gradient: } \frac{\partial L}{\partial \mathbf{W}} = \sum_{t=1}^T \frac{\partial L_t}{\partial \mathbf{W}}$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t) = \tanh\left(\mathbf{W} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix}\right)$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \text{diag}(\tanh'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)) \mathbf{W}_h$$

Gradients over multiple time steps:

©, MIT 2013



(Credit: Stanford CS231N)

$$\begin{aligned} \left. \frac{\partial L_t}{\partial \mathbf{W}} \right|_{\text{first block}} &= \frac{\partial L_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{W}} = \frac{\partial L_t}{\partial \mathbf{h}_t} \left(\prod_{k=2}^t \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} \right) \frac{\partial \mathbf{h}_1}{\partial \mathbf{W}} \\ &= \frac{\partial L_t}{\partial \mathbf{h}_t} \left(\prod_{k=2}^t \text{diag}(\tanh'(\mathbf{W}_h \mathbf{h}_{k-1} + \mathbf{W}_x \mathbf{x}_k)) \mathbf{W}_h \right) \frac{\partial \mathbf{h}_1}{\partial \mathbf{W}} \end{aligned}$$

What's wrong with the gradient?

consider $\prod_{k=2}^t \text{diag}(\tanh'(\mathbf{W}_h \mathbf{h}_{k-1} + \mathbf{W}_x \mathbf{x}_k)) \mathbf{W}_h$

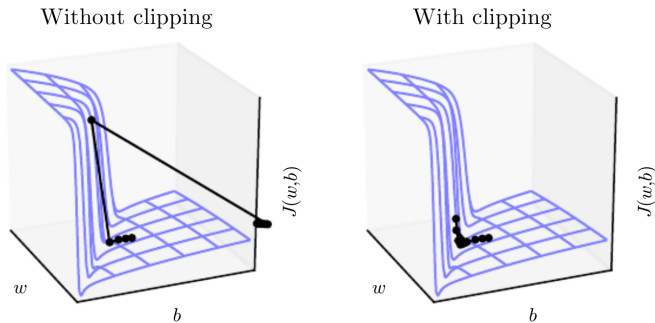
- for intuition, consider **identity** activation first, i.e., $\prod_{k=2}^t \mathbf{W}_h = \mathbf{W}_h^{t-1}$.
But $\|\mathbf{W}_h^{t-1}\|$, i.e., the largest singular value of \mathbf{W}_h^{t-1} , scales as $\|\mathbf{W}_h\|^{t-1}$
 - * when $\|\mathbf{W}_h\| > 1$, gradient **explodes** if t large
 - * when $\|\mathbf{W}_h\| < 1$, gradient **vanishes** if t large
- what happens with the tanh activation?
 - * $\tanh'(x) = 1 - \tanh^2(x) \leq 1$ —effectively always smaller
 - * we have

$$\begin{aligned} & \left\| \prod_{k=2}^t \text{diag}(\tanh'(\mathbf{W}_h \mathbf{h}_{k-1} + \mathbf{W}_x \mathbf{x}_k)) \mathbf{W}_h \right\| \\ & \leq \prod_{k=2}^t \|\text{diag}(\tanh'(\mathbf{W}_h \mathbf{h}_{k-1} + \mathbf{W}_x \mathbf{x}_k))\| \|\mathbf{W}_h\| \\ & \leq \underbrace{\prod_{k=2}^t \|\text{diag}(\tanh'(\mathbf{W}_h \mathbf{h}_{k-1} + \mathbf{W}_x \mathbf{x}_k))\|}_{\text{product of many numbers } < 1 \text{ when } t \text{ large}} \|\mathbf{W}_h\|^{t-1} \end{aligned}$$

Gradient clipping

When the gradient is too large (exploding), rescale (i.e., clip) it. Let \mathbf{g} be the gradient and $\xi > 0$ be a threshold

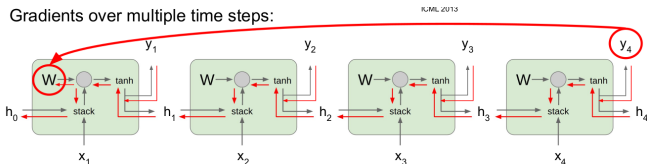
$$\hat{\mathbf{g}} = \xi \frac{\mathbf{g}}{\|\mathbf{g}\|}$$



(Credit: [Goodfellow et al., 2017])

Problem with gradient vanishing

Gradients over multiple time steps:



(Credit: Stanford CS231N)

- **gradient vanishing:** $\frac{\partial h_t}{\partial h_1}$ is (exponentially) small when t is large
⇒ earlier states have little impact on latter states, i.e., memory is short
- but we hope to use RNN to encode reasonably long-term historical/contextual information

Solution? Modify the architecture

Basic RNNs

Vanishing/exploding gradients

Gated RNNs

Modern RNNs

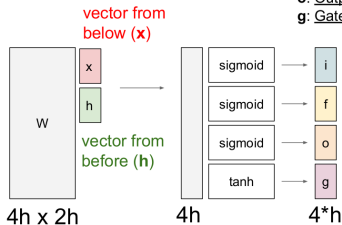
Suggested reading

Long Short-Term Memory (LSTM)

key idea: introduce a cell state c to explicitly store history, besides the hidden state h

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



i : Input gate, whether to write to cell
 f : Forget gate, Whether to erase cell
 o : Output gate, How much to reveal cell
 g : Gate gate (?), How much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

(Credit: Stanford CS231N)

where σ denotes sigmoid

f : memory controller and i : writing controller and o : output controller learned independently

Gated Recurrent Unit (GRU)

simplified version of LSTM ...

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

f: memory controller and **i**:
writing controller and **o**: output
controller learned *independently*
long-term memory when **f** = 1

GRU: no cell state

u: **update gate**, control state update

r: **reset gate**, control how previous state affects
new content

g: new content

$$\begin{bmatrix} u \\ r \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \end{bmatrix} \left(W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \right)$$

$$g = \tanh(W_h(r \odot h_{t-1}) + W_x x_t + b_g)$$

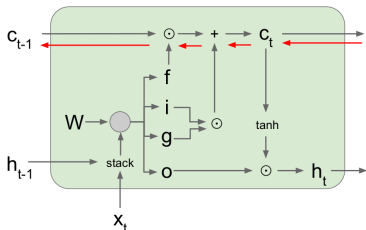
$$h_t = u \odot h_{t-1} + (1 - u) \odot g$$

f, **i**, **o** are merged

long-term memory when **u** = 1 and **r** = 1

LSTM is more flexible and powerful but less efficient in speed

Do they save the vanishing gradient?



(Credit: Stanford CS231N)

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

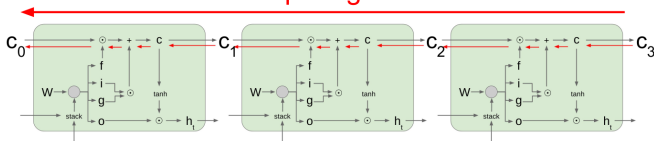
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

(Credit: Stanford CS231N)

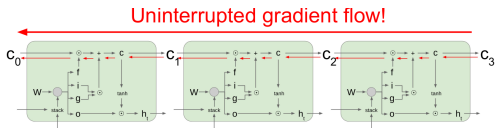
$$\frac{\partial c_t}{\partial c_{t-1}} = \text{diag}(f) \text{ — no multiplication by } W$$

Uninterrupted gradient flow!

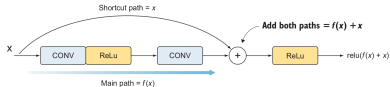


(Credit: Stanford CS231N)

Look familiar?



(Credit: Stanford CS231N)



a residual block (Credit: [Elgendy, 2020])

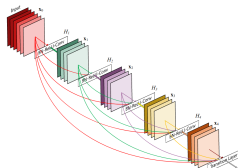


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

(Credit: [Huang et al., 2016])

They are all skip-connections! Similarly for GRU.

- skip connections allow better modeling of long-distance dependency
- but no guarantee of solving the grad vanishing/explosion problem

Do we need to modify the architecture?

problem: W_h can have singular values other than 1

solution: ensure all singular values are 1 $\implies W_h$ is orthogonal

$$\min_{W_h, W_x} L(W), \text{ s. t. } W_h \text{ orthogonal, i.e., } W_h^T W_h = I$$

Good empirical performance, but cost is high for large-scale problems. See, e.g., [Arjovsky et al., 2016, Lezcano-Casado and Martínez-Rubio, 2019]

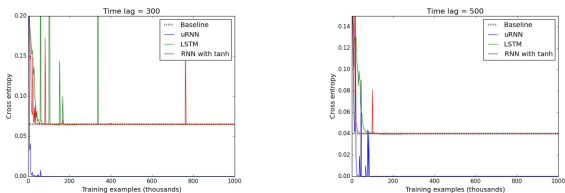


Figure 1. Results of the copying memory problem for time lags of 100, 200, 300, 500. The LSTM is able to beat the baseline only for 100 times steps. Conversely the uRNN is able to completely solve each time length in very few training iterations, without getting stuck at the baseline.

(Credit: [Arjovsky et al., 2016])

(see demo based on PyGRANSO

https://ncvx.org/examples/D3_orthogonal_rnn.html)

Basic RNNs

Vanishing/exploding gradients

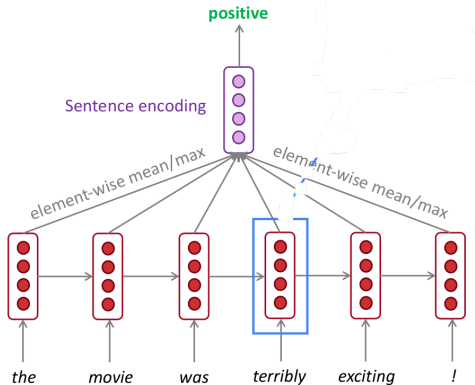
Gated RNNs

Modern RNNs

Suggested reading

Context is important!

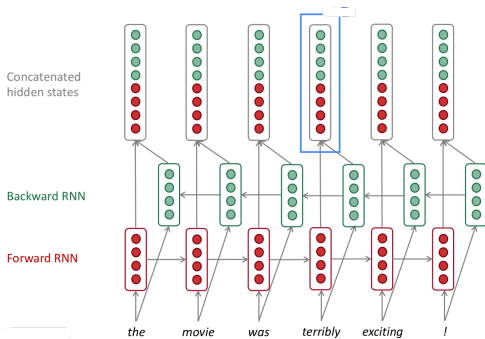
sentiment classification



(Credit: adapted from Stanford CS224N)

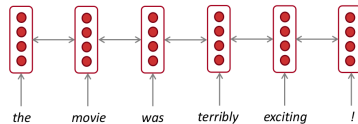
- the state vectors are **contextual representation** of the input words
- but to tell sentiment, "exciting", which is to the right of "terribly" is crucial

Bidirectional RNNs



(Credit: Stanford CS224N)

simplified schematic



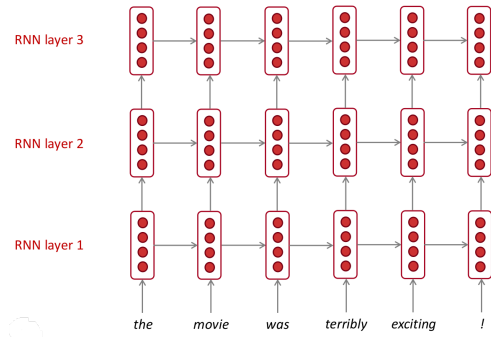
(Credit: Stanford CS224N)

- both left and right contexts are now encoded!
- applicable when the full sequence is available

Deep RNNs

hidden state h can be thought of representation, and so far we only have one layer

Go deeper for more powerful representation learning!

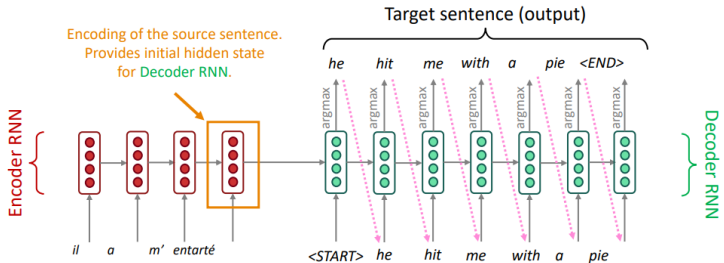


(Credit: Stanford CS231N)

multi-layer RNNs or stacked RNNs. Typically only few layers (much less than that of CNNs)

Sequence to sequence models (Seq2Seq)

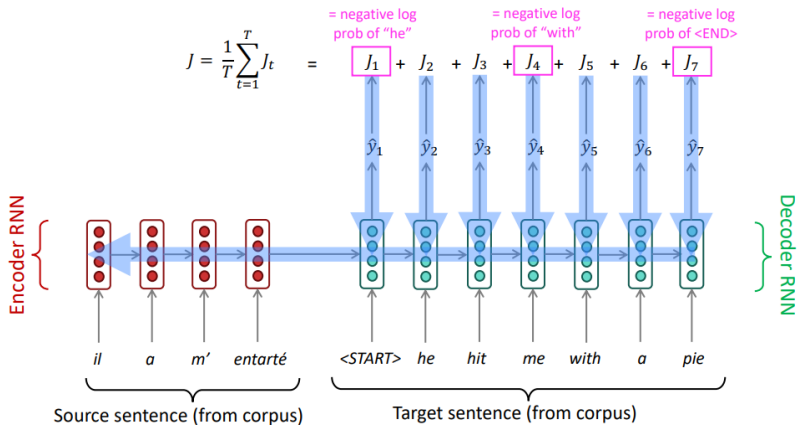
machine translation, image-to-text, speech-to-text, etc



(Credit: Stanford CS231N)

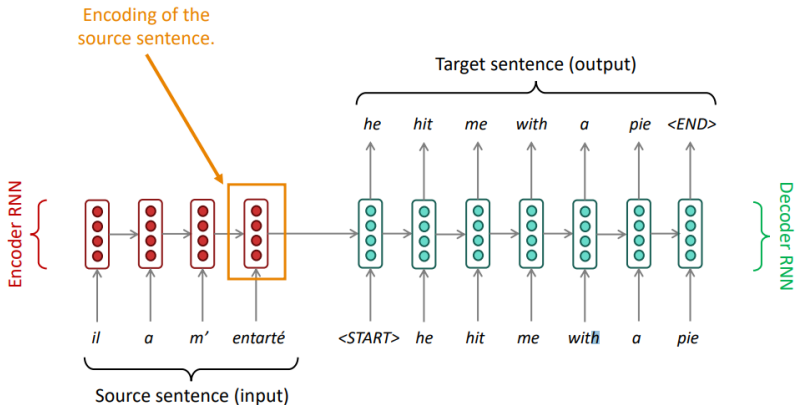
- Falls under encoder-decoder models
- Encoder RNN translate source into an **encoding**
- Decoder RNN is a language model generates output sentence based on the **encoding**

Seq2Seq—training



(Credit: Stanford CS231N)

Seq2Seq—the information bottleneck problem



(Credit: Stanford CS231N)

Problem: the encoding has to capture all info of the source to be effective

Solution: make each target state dependent on **all source states**

Seq2Seq—the information bottleneck problem

Problem: the encoding has to capture all info of the source to be effective

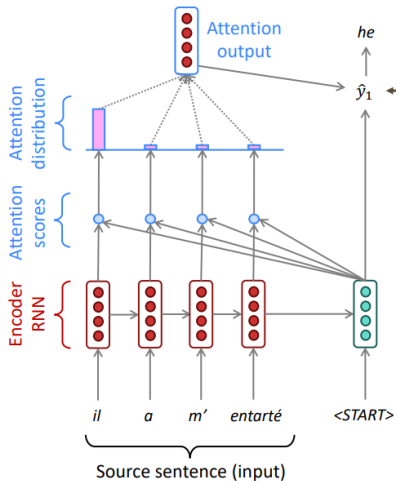
Solution: make each target state dependent on **all source states**

Assume source state vectors $\mathbf{s}_1, \dots, \mathbf{s}_N \in \mathbb{R}^h$, and current target state vector \mathbf{t}_i

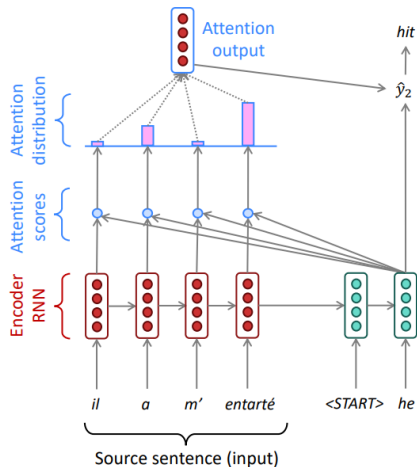
- Idea 1: concatenate, i.e., form $[\mathbf{s}_1; \dots; \mathbf{s}_N; \mathbf{t}_i]$ as the new state vector for the current target step. What's wrong?
- Idea 2: sum and concatenate, i.e., $[\frac{1}{N} \sum_{j=1}^N \mathbf{s}_j; \mathbf{t}_i]$. What's wrong?
- Idea 3: weighted sum and concatenate, i.e., $[\sum_{j=1}^N w_j \mathbf{s}_j; \mathbf{t}_i]$
 - * What weights? Emphasize those most relevant to \mathbf{t}_i
 - * Set $w_j = \text{similarity}(\mathbf{s}_j, \mathbf{t}_i)$: **attention mechanism**

Attention is about measuring (nonlinear) correlation/similarity

Attention in Seq2Seq models



(Credit: Stanford CS231N)



(Credit: Stanford CS231N)

Attention in a nutshell

Assume source vectors $\mathbf{s}_1, \dots, \mathbf{s}_N \in \mathbb{R}^h$, and target vector \mathbf{t} , to obtain **selective summary** (e.g., **weighted summation**) of $\mathbf{s}_1, \dots, \mathbf{s}_N \in \mathbb{R}^h$

$$\sum_{j=1}^N w_j \mathbf{s}_j \quad \text{where } w_j = \text{similarity}(\mathbf{s}_j, \mathbf{t})$$

Many possibilities:

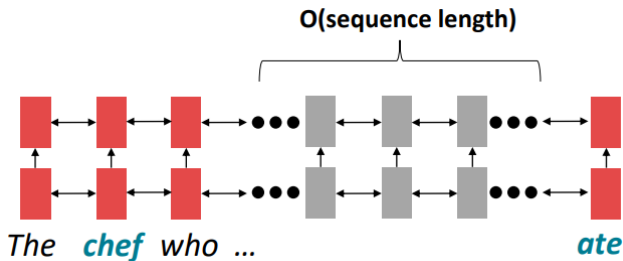
- dot-product attention: $\widehat{w}_j = \langle \mathbf{s}_j, \mathbf{t} \rangle$ (Is it better to normalize this or rescale it by the dimension factor?)
- multiplicative attention: $\widehat{w}_j = \langle \mathbf{s}_j, \mathbf{W}\mathbf{t} \rangle$
- "additive attention": $\widehat{w}_j = \mathbf{v}^\top \sigma(\mathbf{W}_1 \mathbf{s}_j + \mathbf{W}_2 \mathbf{t})$

Afterward, pass the whole weight vector $[w_1, \dots, w_N]$ through softmax to turn it into a valid distribution

$$w_j = \frac{\exp(\widehat{w}_j)}{\sum_k \exp(\widehat{w}_k)}$$

Attention is not only for Seq2Seq or RNNs, it is to: calculate a weighted sum of a bunch of (source) vectors, with the weights dependent on a target/query vector

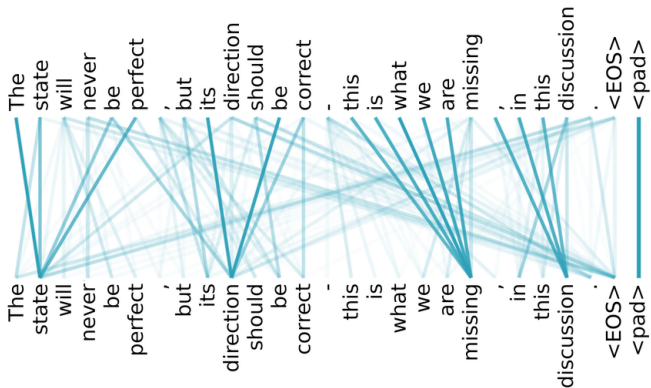
Problems with RNNs



(Credit: Stanford CS231N)

- **linear interaction distance**: challenging to encode long-range dependencies, even within the same sequence
- **resistance to parallelization**: state generation is inherently sequential—problematic for very long sequences

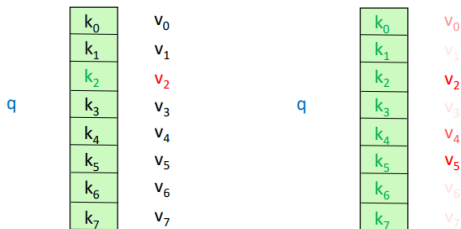
Solution—self-attention



build connections between each other: each state vector depends on all the rest

- $O(1)$ interaction distance
- two state vectors (query, key) to allow parallelization

Self-attention: a closer look



- Each word now encoded as (query, key, value) triple
- For an input x_i , we have:

$$\mathbf{q}_i = (\mathbf{W}^Q)^\top \mathbf{x}_i, \quad \mathbf{k}_i = (\mathbf{W}^K)^\top \mathbf{x}_i, \quad \mathbf{v}_i = (\mathbf{W}^V)^\top \mathbf{x}_i$$

- Calculate attention scores between query and all keys: $e_{ij} = \langle \mathbf{q}_i, \mathbf{k}_j \rangle$
- softmax normalization $w_{ij} = \exp(e_{ij}) / \sum_k \exp(e_{ik})$
- output the weighted sum of values $\sum_j w_{ij} \mathbf{v}_j$

Self-attention in matrix notation

Assume X collects all input words, each one a row:

- Compute queries, keys, and values

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

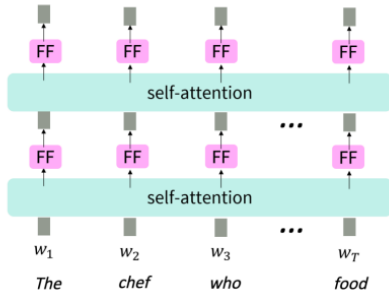
- Calculate attention scores between query and all keys: $E = QK^T$
- softmax normalization to each row: $A = \text{softmax}(E)$
- output the weighted sum of values AV

$$\text{output} = \text{softmax}(QK^T)V$$

Adding in nonlinearity

Equation for Feed Forward Layer

$$\begin{aligned} m_i &= MLP(\text{output}_i) \\ &= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2 \end{aligned}$$



Encoder



First step toward the Transformer!

Basic RNNs

Vanishing/exploding gradients

Gated RNNs

Modern RNNs

Suggested reading

Suggested reading

- Stanford CS224N
<http://web.stanford.edu/class/cs224n/index.html#schedule>
- Understanding LSTM Networks
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- A Guide to the Encoder-Decoder Model and the Attention Mechanism
<https://medium.com/better-programming/a-guide-on-the-encoder-decoder-model-and-the-attention-mechanism-4>
- Attention is all you need: Discovering the Transformer paper
<https://towardsdatascience.com/attention-is-all-you-need-discovering-the-transformer-paper-73e5f1>

- [Arjovsky et al., 2016] Arjovsky, M., Shah, A., and Bengio, Y. (2016). **Unitary evolution recurrent neural networks**. In *International Conference on Machine Learning*, pages 1120–1128.
- [Bruna and Mallat, 2013] Bruna, J. and Mallat, S. (2013). **Invariant scattering convolution networks**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886.
- [Elgendy, 2020] Elgendy, M. (2020). **Deep Learning for Vision Systems**. MANNING PUBL.
- [Goodfellow et al., 2017] Goodfellow, I., Bengio, Y., and Courville, A. (2017). **Deep Learning**. The MIT Press.
- [Huang et al., 2016] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2016). **Densely connected convolutional networks**. *arXiv:1608.06993*.
- [Lezcano-Casado and Martínez-Rubio, 2019] Lezcano-Casado, M. and Martínez-Rubio, D. (2019). **Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group**. *arXiv1901.08428*.

- [Mallat, 2016] Mallat, S. (2016). **Understanding deep convolutional networks.** *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203.
- [Zarka et al., 2019] Zarka, J., Thiry, L., Angles, T., and Mallat, S. (2019). **Deep network classification by scattering and homotopy dictionary learning.** *arXiv:1910.03561*.