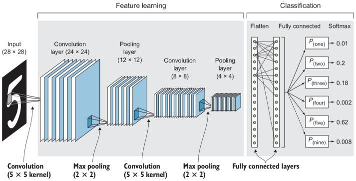
Sequence Modeling: Recurrent Neural Networks

Ju Sun
Computer Science & Engineering
University of Minnesota, Twin Cities

April 3, 2025

Recap: CNNs



(Credit: [Elgendy, 2020])

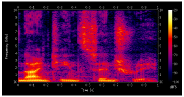
- (neuro-inspired) locality and weight sharing $\Longrightarrow$ reduced complexity (than FCNN)
- conv + pooling $\Longrightarrow$ (approx.) translation/deformation invariance (part of the learning can be avoided; see **scattering transform** [Bruna and Mallat, 2013, Mallat, 2016, Zarka et al., 2019])

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

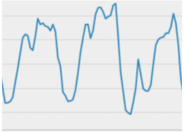
image

video

audio (spectrogram)



time series

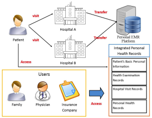


Model sequences

… where directions matter

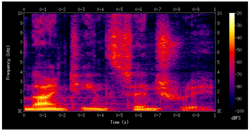
temporal sequences

disease prognosis

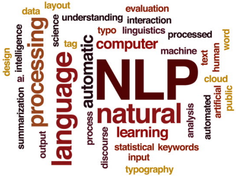


event analysis/video generation

speech to text



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



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

Outline

Basic RNNs

Vanishing/exploding gradients

Gated RNNs

Modern RNNs

Suggested reading

Basic setup

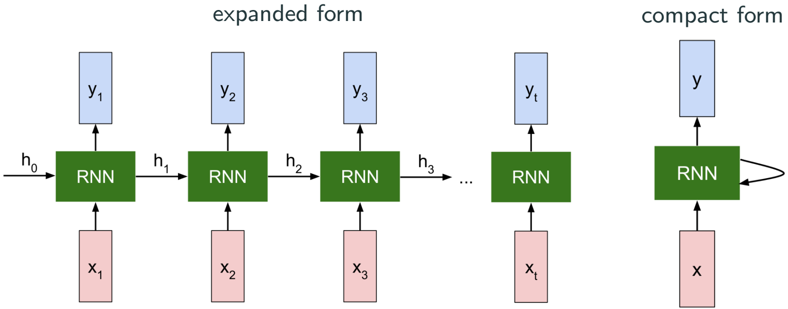
A sequence: $\boldsymbol{x}_0 \to \boldsymbol{x}_1 \to \boldsymbol{x}_2 \to \ldots \boldsymbol{x}_{n-1}$

A **state-space** model: $\boldsymbol{h}$ denotes the state, and state transition modeled by the recurrence formula

$$\boldsymbol{h}_t = f_{\boldsymbol{W}}\left(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right)$$

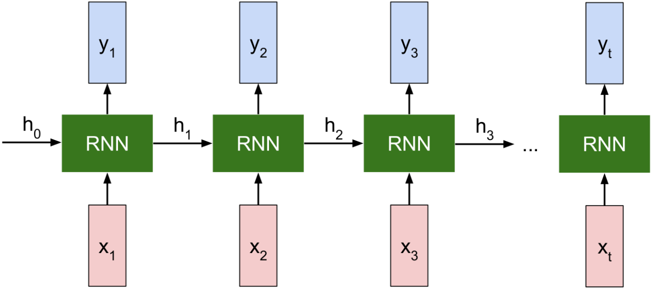
with optional output

$$\boldsymbol{y}_t = g_{\boldsymbol{V}}\left(\boldsymbol{h}_t\right)$$



(Credit: Stanford CS231N)

A simple (vanilla) RNN



(Credit: Stanford CS231N)

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_t\right)$$

$$\boldsymbol{y}_t = \boldsymbol{V}_y\boldsymbol{h}_t$$

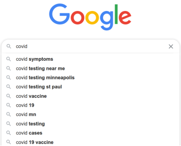
$\boldsymbol{W}_{\boldsymbol{h}}, \boldsymbol{W}_{\boldsymbol{x}}$ and $\boldsymbol{V}_{\boldsymbol{y}}$ are shared across the sequence

A first example: language modeling

- **language modeling** is the task of predicting future words

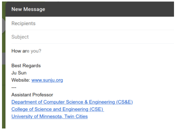
  `... The vaccine is effective, and COVID-19 will be _.`

- applications: typing prediction (smart compose), machine translation, ChatGPT, etc



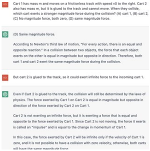
covid

covid symptoms
covid testing near me
covid testing minneapolis
covid testing st paul
covid vaccine
covid 19
covid mn
covid testing
covid cases
covid 19 vaccine



How are you?

Best Regards
Ju Sun
Website: www.sunju.org
--
Assistant Professor
Department of Computer Science & Engineering (CS&E)
College of Science and Engineering (CSE)
University of Minnesota, Twin Cities



Cart 1 has mass m and moves on a frictionless track with speed v0 to the right. Cart 2 also has mass m, but it is glued to the track and cannot move. When they collide, which cart exerts a stronger magnitude force during the collision? (A) cart 1, (B) cart 2, (C) No magnitude force, both zero, (D) same magnitude force.

(D) Same magnitude force.

According to Newton's third law of motion, "For every action, there is an equal and opposite reaction." In a collision between two objects, the force that each object exerts on the other is equal in magnitude but opposite in direction. Therefore, both cart 1 and cart 2 exert the same magnitude force during the collision.

But cart 2 is glued to the track, so it could exert infinite force to the incoming cart 1.

Even if Cart 2 is glued to the track, the collision will still be determined by the laws of physics. The force exerted by Cart 1 on Cart 2 is equal in magnitude but opposite in direction of the force exerted by Cart 2 on Cart 1.

Cart 2 is not exerting an infinite force, but it is exerting a force that is equal and opposite to the force exerted by Cart 1. Since Cart 2 is not moving, the force it exerts is called an "impulse" and is equal to the change in momentum of Cart 1.

In this case, the force exerted by Cart 2 will be infinite only if the velocity of Cart 1 is zero, and it is not possible to have a collision with zero velocity, otherwise, both carts will have the same magnitude force.

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

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

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

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

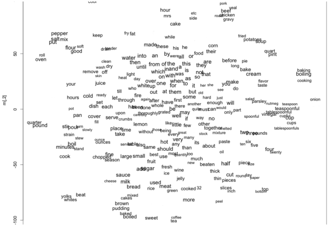
Modern neural language modeling—word embedding

Representing words: word embedding

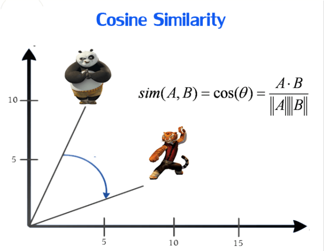
- one hot encoding

$$\text{I} \mapsto [1, 0, 0, 0, \ldots], \text{you} \mapsto [0, 1, 0, 0, \ldots], \text{we} \mapsto [0, 0, 1, 0, \ldots], \ldots$$

- 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—prediction

RNN modeling: predicting the next word each time

output distribution

$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$

hidden states

$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$

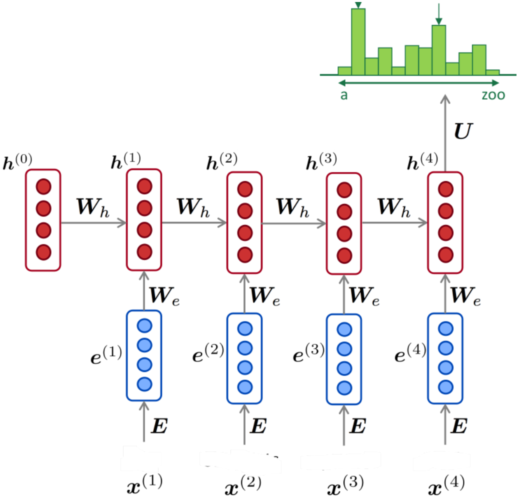
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

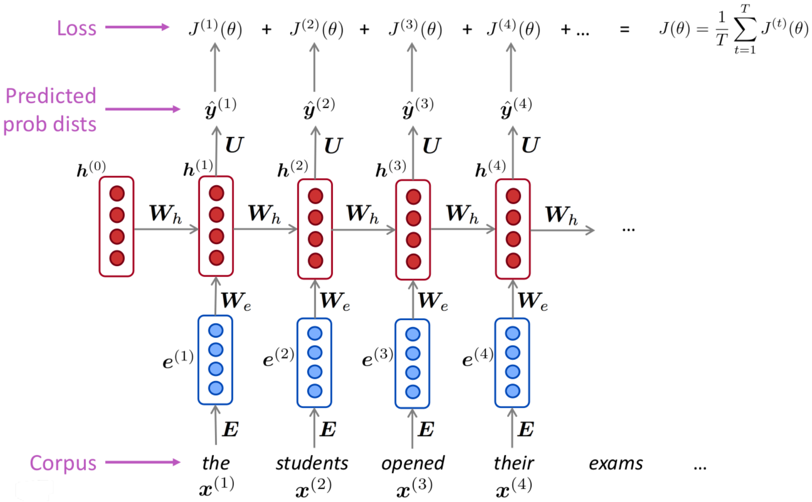
$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



(Credit: adapted from Stanford CS224N)

Modern neural language modeling—training

Training the RNN model



(Credit: Stanford CS224N)

Modern neural language modeling—whole pipeline

The whole training pipeline

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

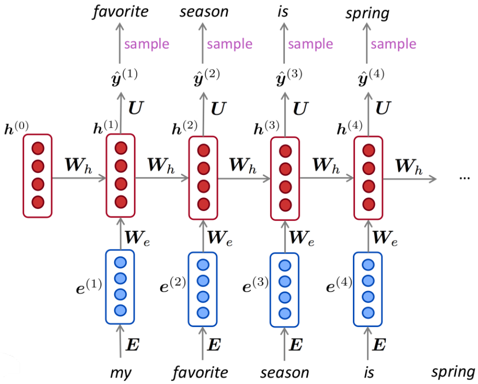
$$J^{(t)}\left(\boldsymbol{\theta}\right) = -\sum_{\boldsymbol{w} \in \mathcal{V}} \boldsymbol{y}_{\boldsymbol{w}}^{(t)} \log \widehat{\boldsymbol{y}}_{\boldsymbol{w}}^{(t)} = -\log \widehat{\boldsymbol{y}}_{\boldsymbol{x}^{(t+1)}}^{(t)}$$

- Step 4: gather and average all losses:

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

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

Outline

Basic RNNs

Vanishing/exploding gradients

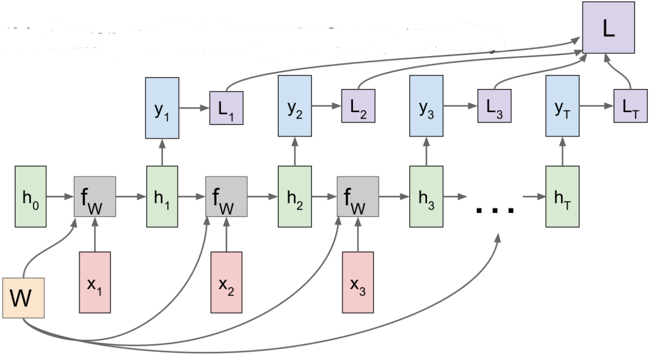
Gated RNNs

Modern RNNs

Suggested reading

How to compute gradients?

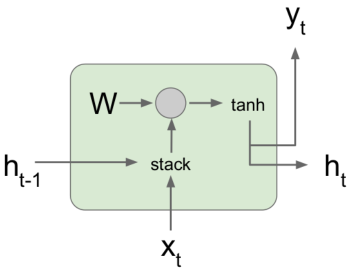
computational graph



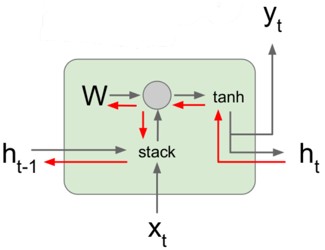
(Credit: Stanford CS231N)

- acyclic directed graph $\Longrightarrow$ auto differentiation can be applied
- $\boldsymbol{W}$ is shared across all steps!

Look into the gradient



(Credit: Stanford CS231N)



(Credit: Stanford CS231N)

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

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

where $\boldsymbol{W} = \begin{bmatrix}\boldsymbol{W}_h \; \boldsymbol{W}_x\end{bmatrix}$

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t\right)\right) \boldsymbol{W}_h$$

where

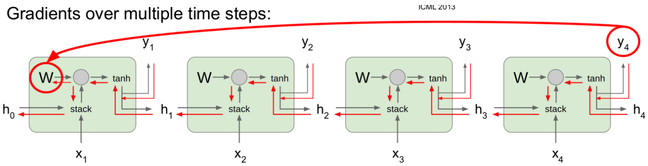
$$\tanh'(x) = 1 - \tanh^2(x)$$

Look into the gradient

$L = \sum_{t=1}^{T} L_t \Longrightarrow$ total gradient: $\frac{\partial L}{\partial \boldsymbol{W}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \boldsymbol{W}}$

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

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}} \boldsymbol{h}_{t-1} + \boldsymbol{W}_{\boldsymbol{x}} \boldsymbol{x}_t\right)\right) \boldsymbol{W}_h$$



(Credit: Stanford CS231N)

$$\left.\frac{\partial L_t}{\partial \boldsymbol{W}}\right|_{\text{first block}} = \frac{\partial L_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\boldsymbol{W}} = \frac{\partial L_t}{\partial \boldsymbol{h}_t} \left(\prod_{k=2}^{t} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{h}_{k-1}}\right) \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}$$

$$= \frac{\partial L_t}{\partial \boldsymbol{h}_t} \left(\prod_{k=2}^{t} \operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}} \boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}} \boldsymbol{x}_k\right)\right) \boldsymbol{W}_h\right) \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}$$

What's wrong with the gradient?

consider $\prod_{k=2}^{t} \operatorname{diag}\left(\tanh'\left(\boldsymbol{W_h h}_{k-1} + \boldsymbol{W_x x}_k\right)\right) \boldsymbol{W}_h$

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

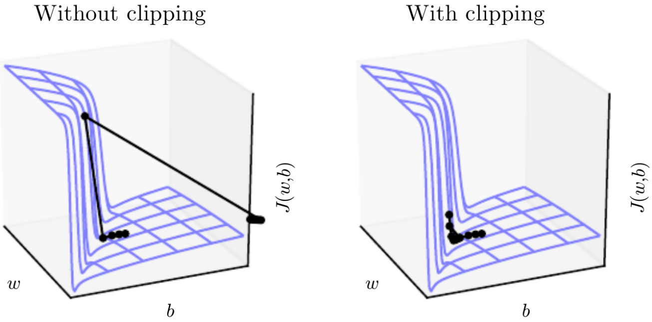
$$\begin{aligned}
&\left\| \prod_{k=2}^{t} \operatorname{diag}\left(\tanh'\left(\boldsymbol{W_h h}_{k-1} + \boldsymbol{W_x x}_k\right)\right) \boldsymbol{W}_h \right\| \\
\le &\prod_{k=2}^{t} \left\|\operatorname{diag}\left(\tanh'\left(\boldsymbol{W_h h}_{k-1} + \boldsymbol{W_x x}_k\right)\right)\right\|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

gradient vanishing is more common

Gradient clipping

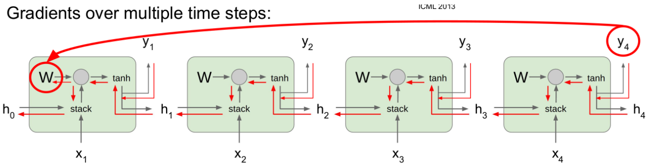
When the gradient is too large (exploding), rescale (i.e., clip) it. Let $\boldsymbol{g}$ be the gradient and $\xi > 0$ be a threshold

$$\widehat{\boldsymbol{g}} = \xi \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$$



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

Problem with gradient vanishing



(Credit: Stanford CS231N)

- **gradient vanishing**: $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_1}$ is (exponentially) small when t is large
  $\Longrightarrow$ 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

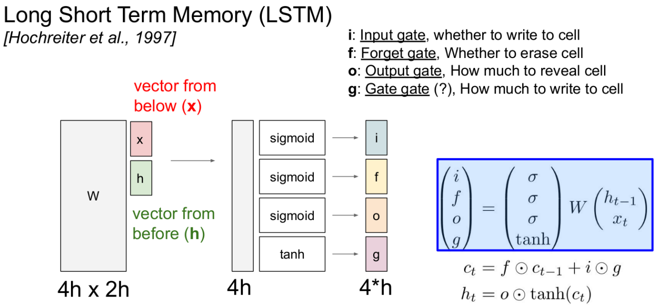
Solution? Modify the architecture

Outline

- Basic RNNs
- Vanishing/exploding gradients
- Gated RNNs
- Modern RNNs
- Suggested reading

Long Short-Term Memory (LSTM)

key idea: introduce a cell state $\boldsymbol{c}$ to explicitly store history, besides the hidden state $\boldsymbol{h}$



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

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

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

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

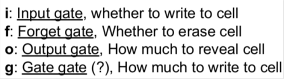
(Credit: Stanford CS231N)

where σ denotes sigmoid

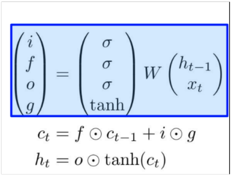
f: memory controller and i: writing controller and o: output controller learned *independently*

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



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

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

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

$\boldsymbol{f}$: memory controller and $\boldsymbol{i}$: writing controller and $\boldsymbol{o}$: output controller learned *independently*

long-term memory when $\boldsymbol{f} = \mathbf{1}$

GRU: no cell state

$\boldsymbol{u}$: **update gate**, control state update

$\boldsymbol{r}$: **reset gate**, control how previous state affects new content

$\boldsymbol{g}$: new content

$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{r} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \end{bmatrix} \left(\boldsymbol{W} \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix} \right)$$

$$\boldsymbol{g} = \tanh\left(\boldsymbol{W}_{\boldsymbol{h}}\left(\boldsymbol{r} \odot \boldsymbol{h}_{t-1}\right) + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_t + \boldsymbol{b}_{\boldsymbol{g}}\right)$$

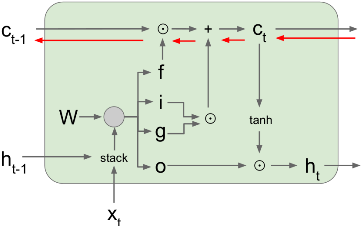
$$\boldsymbol{h}_t = \boldsymbol{u} \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}) \odot \boldsymbol{g}$$

$\boldsymbol{f}$, $\boldsymbol{i}$, $\boldsymbol{o}$ are merged

long-term memory when $\boldsymbol{u} = \mathbf{1}$ and $\boldsymbol{r} = \mathbf{1}$

LSTM is more flexible and powerful but less efficient in speed

Do they save the vanishing gradient?



(Credit: Stanford CS231N)

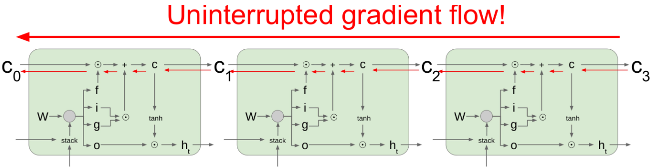
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

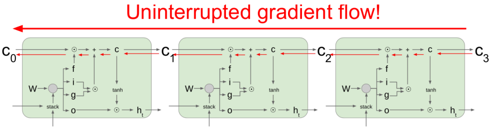
(Credit: Stanford CS231N)

$\frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{c}_{t-1}} = \operatorname{diag}(\boldsymbol{f})$ — no multiplication by $\boldsymbol{W}$

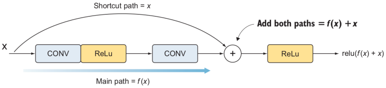


(Credit: Stanford CS231N)

Look familiar?



(Credit: Stanford CS231N)



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

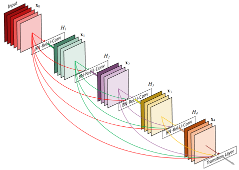


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

(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: $\boldsymbol{W_h}$ can have singular values other than 1

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

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

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

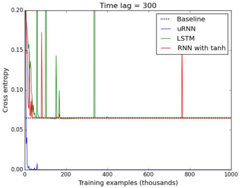


Figure 1. Results of the copying memory problem for time lags of 100, 200, 300, 500. The LSTM is able to beat the baseline only for 100 times steps. Conversely the uRNN is able to completely solve each time length in very few training iterations, without getting stuck at the baseline.

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

(see demo based on PyGRANSO `https://ncvx.org/examples/D3_orthogonal_rnn.html`)

Outline

Basic RNNs

Vanishing/exploding gradients

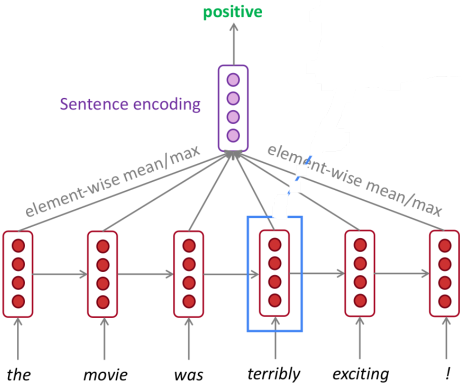
Gated RNNs

Modern RNNs

Suggested reading

Context is important!

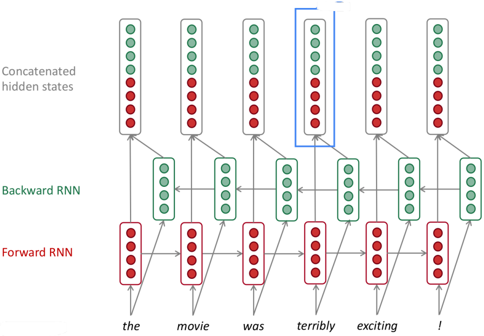
sentiment classification



(Credit: adapted from Stanford CS224N)

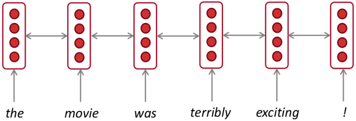
- the state vectors are **contextual representation** of the input words
- but to tell sentiment, “exciting”, which is to the right of “terribly” is crucial

Bidirectional RNNs



(Credit: Stanford CS224N)

simplified schematic



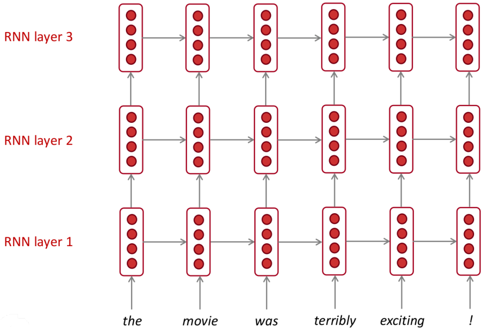
(Credit: Stanford CS224N)

- 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!

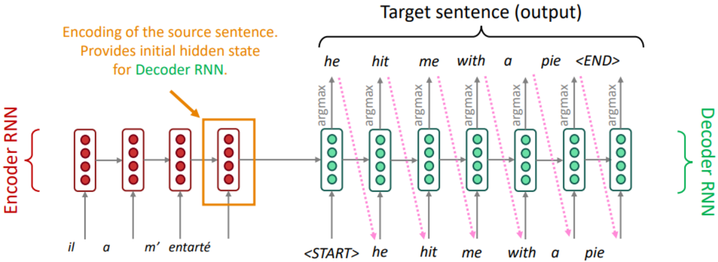


(Credit: Stanford CS231N)

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

Sequence to sequence models (Seq2Seq)

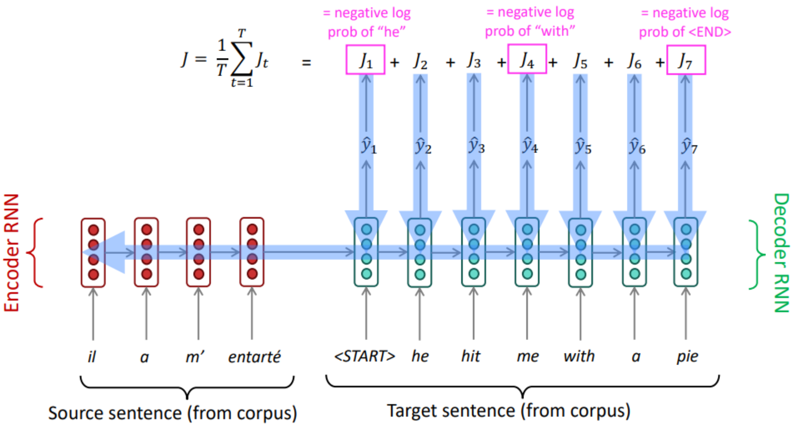
machine translation, image-to-text, speech-to-text, etc



(Credit: Stanford CS231N)

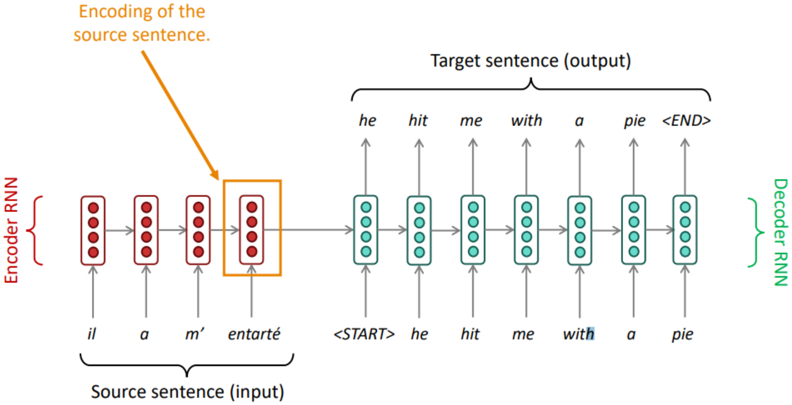
- Falls under encoder-decoder models
- Encoder RNN translate source into an encoding
- Decoder RNN is a language model generates output sentence based on the encoding

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

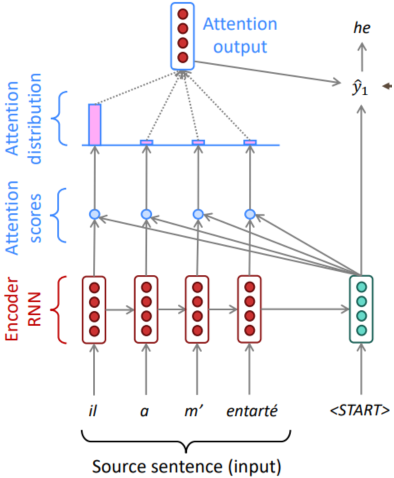
Problem: the encoding has to capture all info of the source to be effective
Solution: make each target state dependent on all source states

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

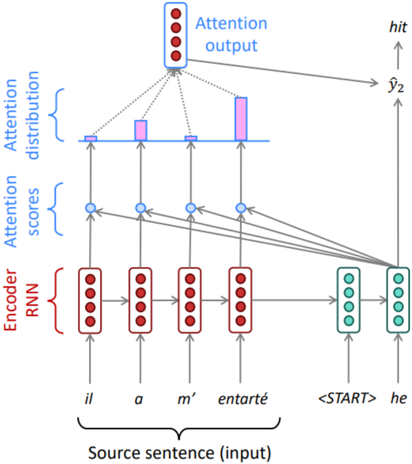
- Idea 1: concatenate, i.e., form $[\boldsymbol{s}_1; \ldots; \boldsymbol{s}_N; \boldsymbol{t}_i]$ as the new state vector for the current target step. What's wrong?
- Idea 2: sum and concatenate, i.e., $[\frac{1}{N}\sum_{j=1}^{N} \boldsymbol{s}_j; \boldsymbol{t}_i]$. What's wrong?
- Idea 3: weighted sum and concatenate, i.e., $[\sum_{j=1}^{N} w_j \boldsymbol{s}_j; \boldsymbol{t}_i]$
  - What weights? Emphasize those most relevant to $\boldsymbol{t}_i$
  - Set $w_j = \mathrm{similarity}(\boldsymbol{s}_j, \boldsymbol{t}_i)$: **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 $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N \in \mathbb{R}^h$, and target vector $\boldsymbol{t}$, to obtain selective summary (e.g., weighted summation) of $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N \in \mathbb{R}^h$

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

Many possibilities:

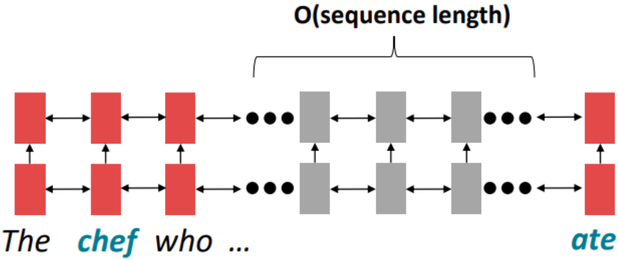
- dot-product attention: $\widehat{w_j} = \langle \boldsymbol{s}_j, \boldsymbol{t} \rangle$ (Is it better to normalize this or rescale it by the dimension factor?)
- multiplicative attention: $\widehat{w_j} = \langle \boldsymbol{s}_j, \boldsymbol{W}\boldsymbol{t} \rangle$
- “additive attention”: $\widehat{w_j} = \boldsymbol{v}^{\intercal} \sigma \left(\boldsymbol{W}_1 \boldsymbol{s}_j + \boldsymbol{W}_2 \boldsymbol{t} \right)$

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

$$w_j = \frac{\exp\left(\widehat{w_j}\right)}{\sum_k \exp\left(\widehat{w_k}\right)}$$

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

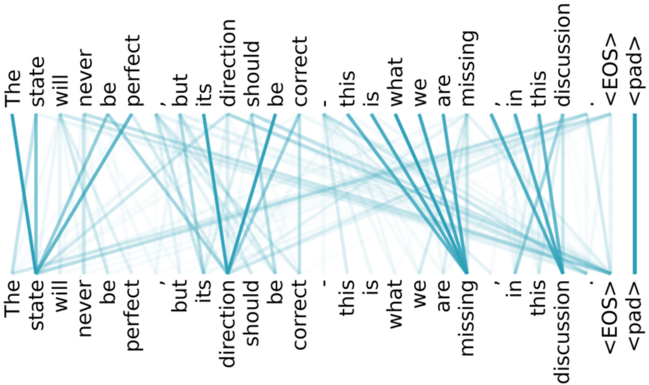
Problems with RNNs



(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

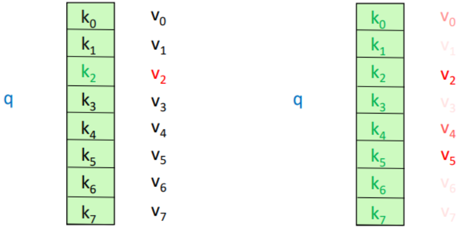
Solution—self-attention



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 $\boldsymbol{x}_i$, we have:

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

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

Self-attention in matrix notation

Assume $\boldsymbol{X}$ collects all input words, each one a row:

- Compute queries, keys, and values

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

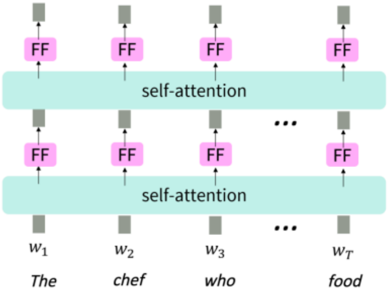
- Calculate attention scores between query and all keys: $\boldsymbol{E} = \boldsymbol{Q}\boldsymbol{K}^\intercal$
- softmax normalization to each row: $\boldsymbol{A} = \mathrm{softmax}(\boldsymbol{E})$
- output the weighted sum of values $\boldsymbol{A}\boldsymbol{V}$

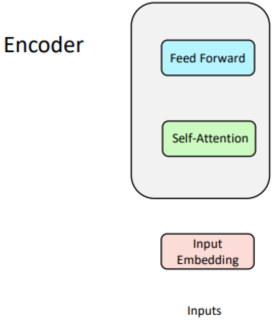
$$\mathrm{output} = \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\intercal)\boldsymbol{V}$$

Adding in nonlinearity

Equation for Feed Forward Layer

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





First step toward the Transformer!

Outline

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-
- Attention is all you need: Discovering the Transformer paper
  https://towardsdatascience.com/
  attention-is-all-you-need-discovering-the-transformer-paper-73e5f

References i

[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 PUBN.

[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*.

References ii

[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*.