

Lecture 9 - Networks for Sequential Data RNNs & LSTMs

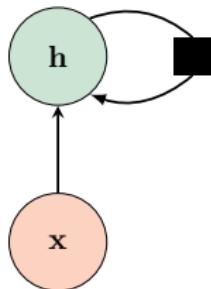
DD2424

April 23, 2018

Recurrent Neural Networks (RNNs)

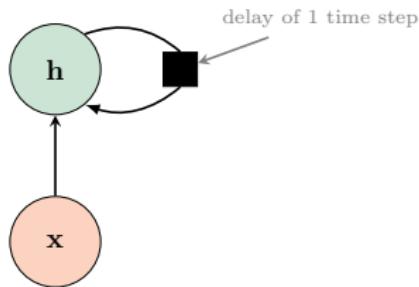
- RNNs are a family of networks for processing sequential data.
- A RNN applies the same function recursively when traversing network's graph structure.
- RNN encodes a sequence $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\tau$ into fixed length hidden vector \mathbf{h}_τ .
- The size of \mathbf{h}_τ is independent of τ .
- Amazingly flexible and powerful high-level architecture.

RNN with no outputs



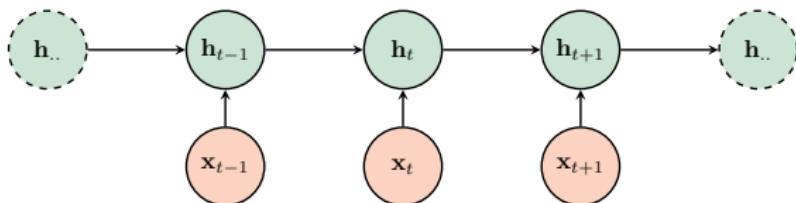
- Graph displays processing of information for each time step.
- Information from input **x** is incorporated into state **h**.

RNN with no outputs



- Graph displays processing of information for each time step.
- Information from input x is incorporated into state h .
- State h is passed forward in time.

RNN with no outputs



Unrolled visualization of the RNN

RNN: How hidden states generated

- Most **recurrent neural networks** have a function f

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta})$$

that defines their hidden state over time where

- \mathbf{h}_t is the hidden state at time t (a vector)
- \mathbf{x}_t is the input vector at time t
- $\boldsymbol{\theta}$ is the parameters of f .

- $\boldsymbol{\theta}$ remains constant as t changes.

Apply the **same function** with the **same parameter values**
at each iteration.

RNN: How hidden states generated

- Most **recurrent neural networks** have a function f

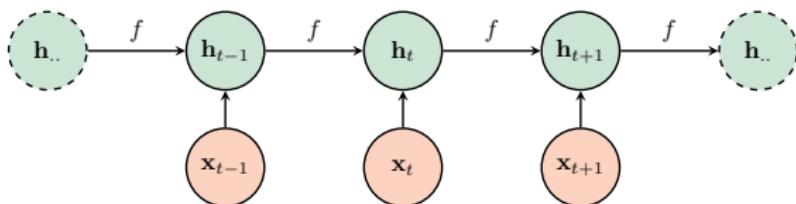
$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta})$$

that defines their hidden state over time where

- \mathbf{h}_t is the hidden state at time t (a vector)
 - \mathbf{x}_t is the input vector at time t
 - $\boldsymbol{\theta}$ is the parameters of f .
- $\boldsymbol{\theta}$ remains constant as t changes.

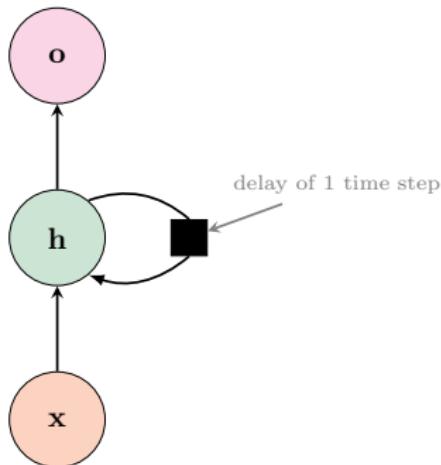
Apply the **same function** with the **same parameter values**
at each iteration.

RNN with no outputs



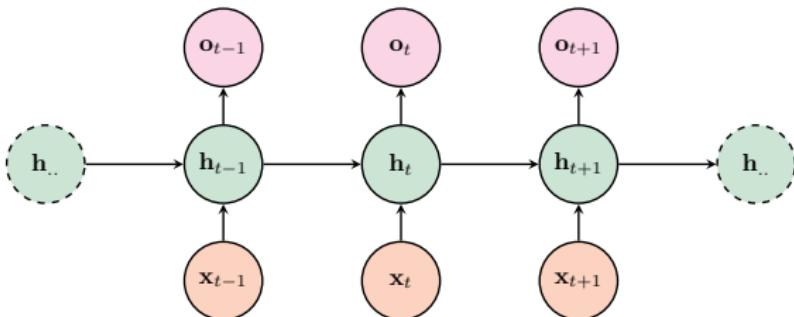
Unrolled visualization of the RNN

RNN with outputs



- Usually also predict an output vector at each time step

RNN with outputs

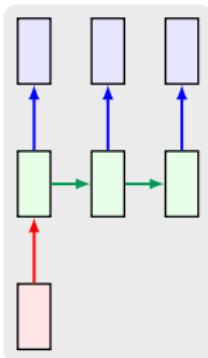


Unrolled visualization of the RNN

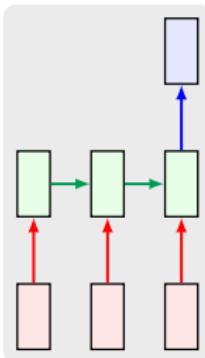
- Usually also predict an output vector at each time step

Use cases of RNNs

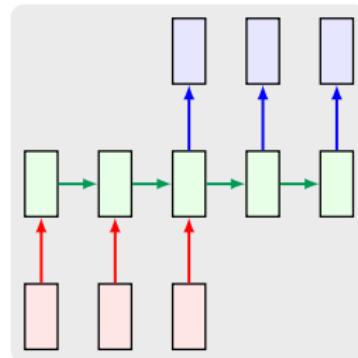
one to many



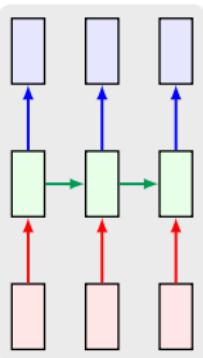
many to one



many to many

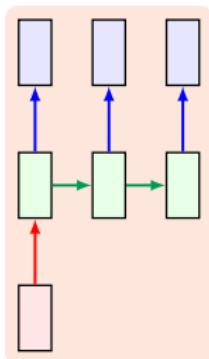


many to many

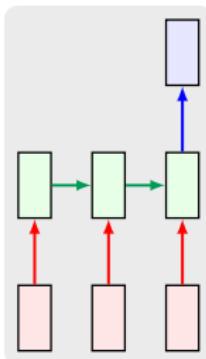


Use cases of RNNs

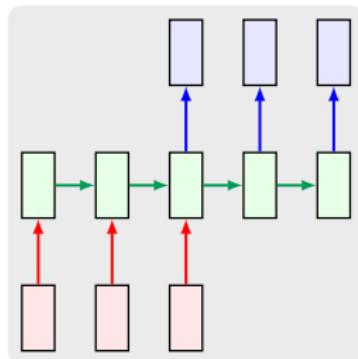
one to many



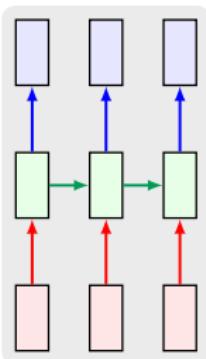
many to one



many to many



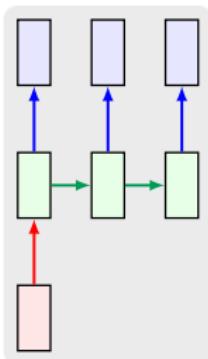
many to many



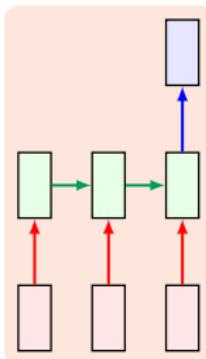
Example - Image captioning: image → sequence of words

Use cases of RNNs

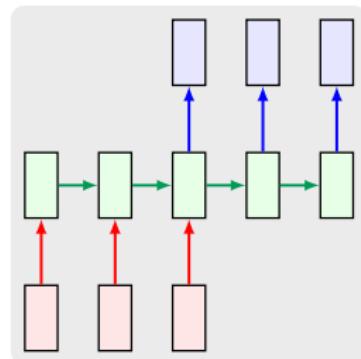
one to many



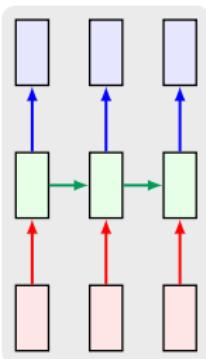
many to one



many to many



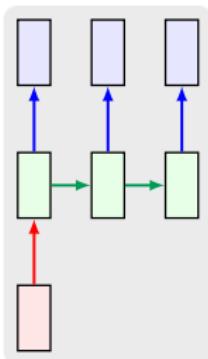
many to many



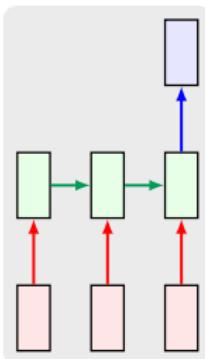
Example - Sentiment Classification: seq. of words → sentiment

Use cases of RNNs

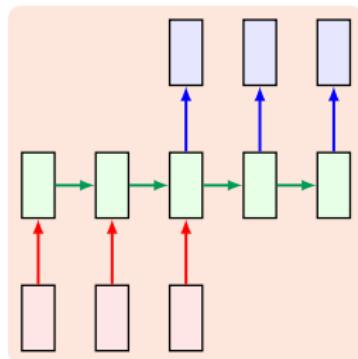
one to many



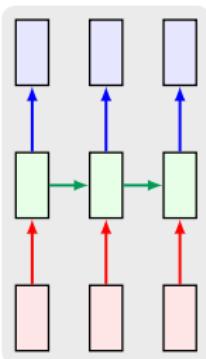
many to one



many to many



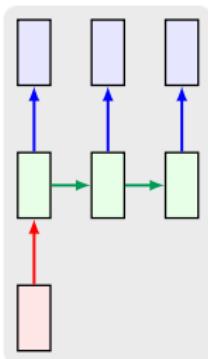
many to many



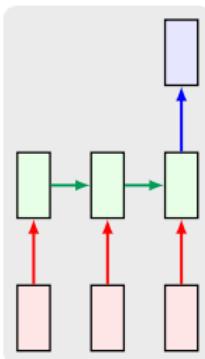
Example - Machine Translation: seq. of words → seq. of words

Use cases of RNNs

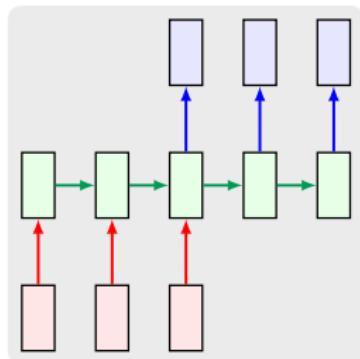
one to many



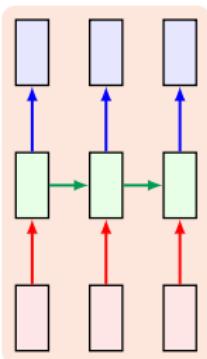
many to one



many to many



many to many



Example - Video classification on frame level

- The state consists of a single **hidden** vector \mathbf{h}_t :
- Initial hidden state \mathbf{h}_0 is assumed given.
- For $t = 1, \dots, \tau$ the RNN equations are

$$\mathbf{a}_t = W\mathbf{h}_{t-1} + U\mathbf{x}_t + \mathbf{b}$$

$$\mathbf{h}_t = \tanh(\mathbf{a}_t)$$

$$\mathbf{o}_t = V\mathbf{h}_t + \mathbf{c}$$

$$\mathbf{p}_t = \text{softmax}(\mathbf{o}_t)$$

Network's input

- \mathbf{h}_0 initial hidden state has size $m \times 1$
- \mathbf{x}_t input vector at time t has size $d \times 1$

- The state consists of a single **hidden** vector \mathbf{h}_t :
- Initial hidden state \mathbf{h}_0 is assumed given.
- For $t = 1, \dots, \tau$ the RNN equations are

$$\mathbf{a}_t = W\mathbf{h}_{t-1} + U\mathbf{x}_t + \mathbf{b}$$

$$\mathbf{h}_t = \tanh(\mathbf{a}_t)$$

$$\mathbf{o}_t = V\mathbf{h}_t + \mathbf{c}$$

$$\mathbf{p}_t = \text{softmax}(\mathbf{o}_t)$$

Network's output and hidden vectors

- \mathbf{a}_t hidden state at time t of size $m \times 1$ before non-linearity
- \mathbf{h}_t hidden state at time t of size $m \times 1$
- \mathbf{o}_t output vector (of unnormalized log probabilities for each class) at time t of size $C \times 1$
- \mathbf{p}_t output probability vector at time t of size $C \times 1$

- The state consists of a single **hidden** vector \mathbf{h} :
- Initial hidden state \mathbf{h}_0 is assumed given.
- For $t = 1, \dots, T$ the RNN equations are

$$\mathbf{a}_t = \mathcal{W}\mathbf{h}_{t-1} + \mathcal{U}\mathbf{x}_t + \mathbf{b}$$

$$\mathbf{h}_t = \tanh(\mathbf{a}_t)$$

$$\mathbf{o}_t = \mathcal{V}\mathbf{h}_t + \mathbf{c}$$

$$\mathbf{p}_t = \text{softmax}(\mathbf{o}_t)$$

Parameters of the network

- \mathcal{W} weight matrix of size $m \times m$ applied to \mathbf{h}_{t-1} (hidden-to-hidden connection)
- \mathcal{U} weight matrix of size $m \times d$ applied to \mathbf{x}_t (input-to-hidden connection)
- \mathbf{b} bias vector of size $m \times 1$ in equation for \mathbf{a}_t
- \mathcal{V} weight matrix of size $C \times m$ applied to \mathbf{a}_t (hidden-to-output connection)
- \mathbf{c} bias vector of size $C \times 1$ in equation for \mathbf{o}_t

Simple RNN example: Character-level language model

Vocabulary: $\{h, e, l, o\}$, **Example training sequence:** “hello”

Simple RNN example: Character-level language model

Vocabulary: $\{h, e, l, o\}$, **Example training sequence:** “hello”

Input layer:

1
0
0
0

0
1
0
0

0
0
1
0

0
0
1
0

Input chars:

“h”

“e”

“l”

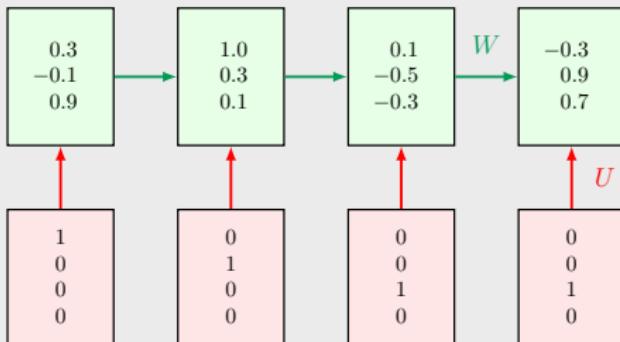
“l”

Simple RNN example: Character-level language model

Vocabulary: $\{h, e, l, o\}$, **Example training sequence:** “hello”

$$\mathbf{h}_t = \tanh (\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

Hidden layer:



Input layer:

Input chars:

“h”

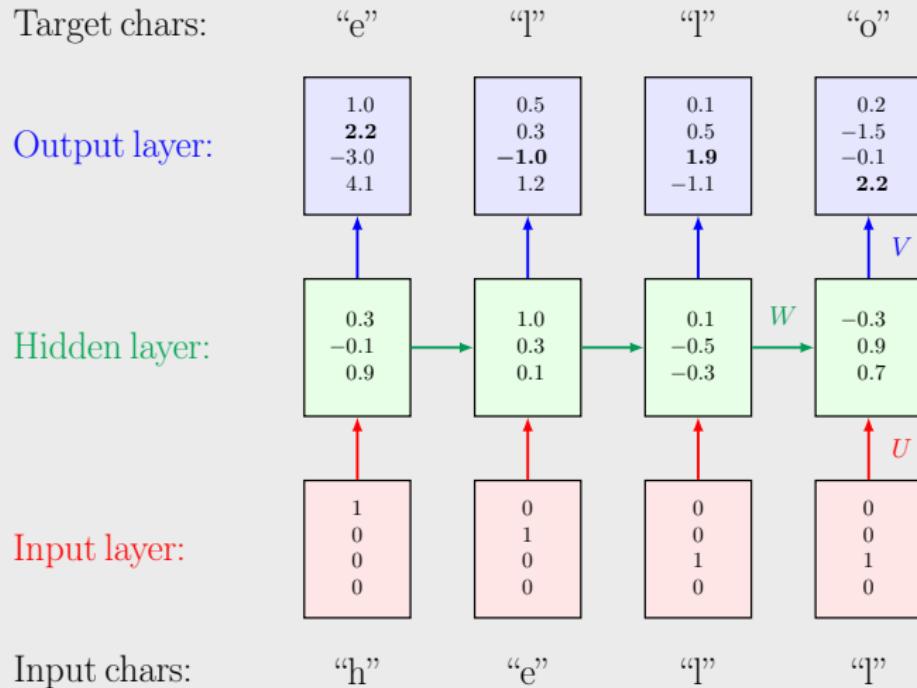
“e”

“l”

“l”

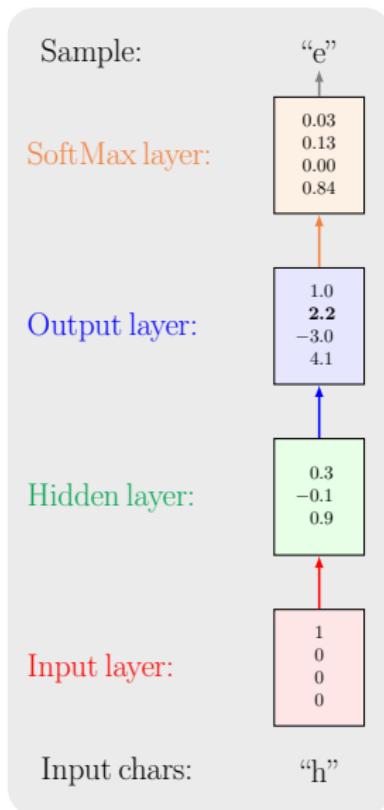
Simple RNN example: Character-level language model

Vocabulary: $\{h, e, l, o\}$, **Example training sequence:** “hello”



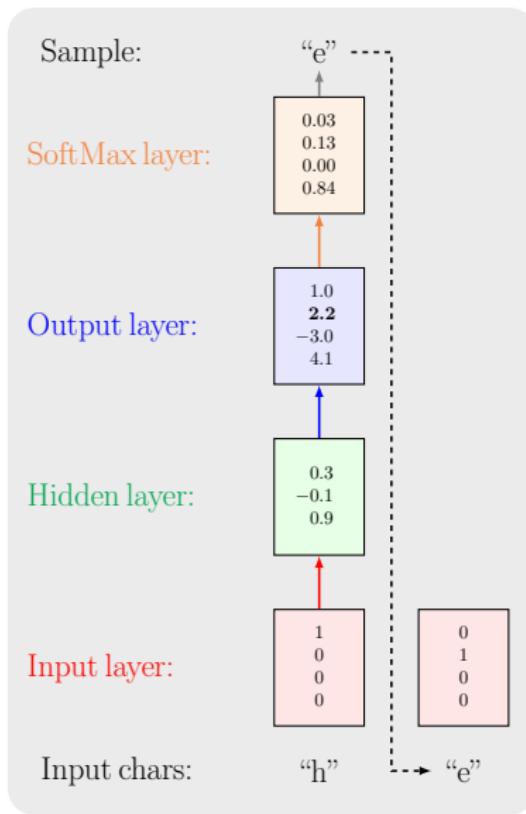
Simple RNN example: Character-level language model

Vocab: {*h, e, l, o*}, **Test time:** sample characters one at a time, feed back to model.



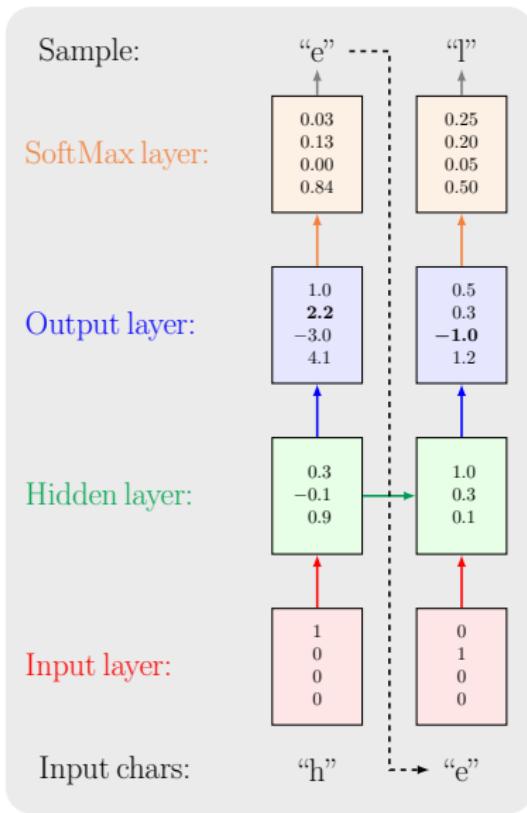
Simple RNN example: Character-level language model

Vocab: $\{h, e, l, o\}$, **Test time:** sample characters one at a time, feed back to model.



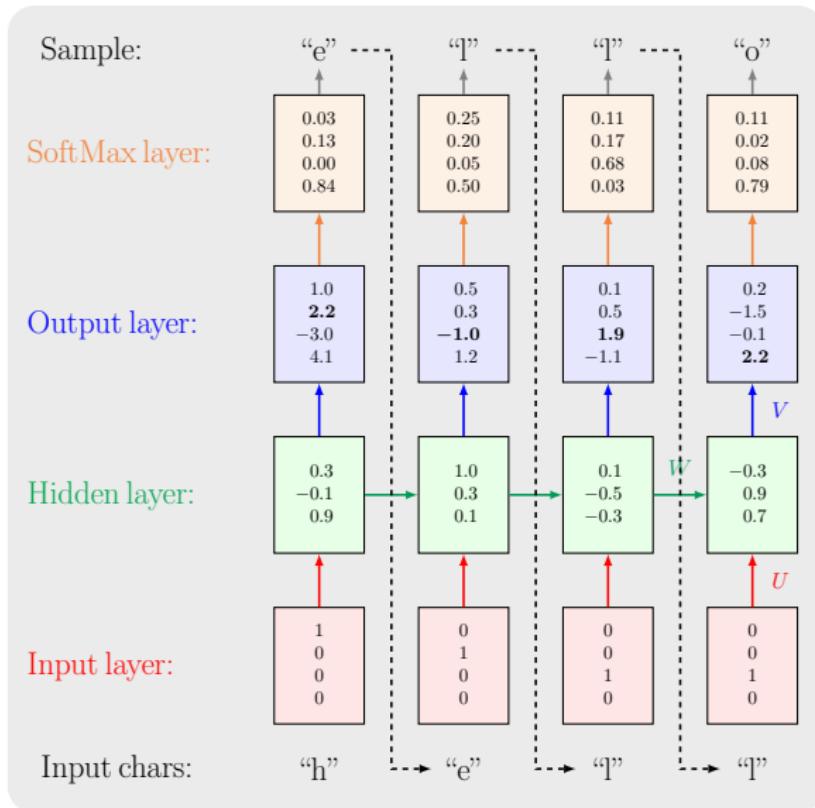
Simple RNN example: Character-level language model

Vocab: {*h, e, l, o*}, **Test time:** sample characters one at a time, feed back to model.

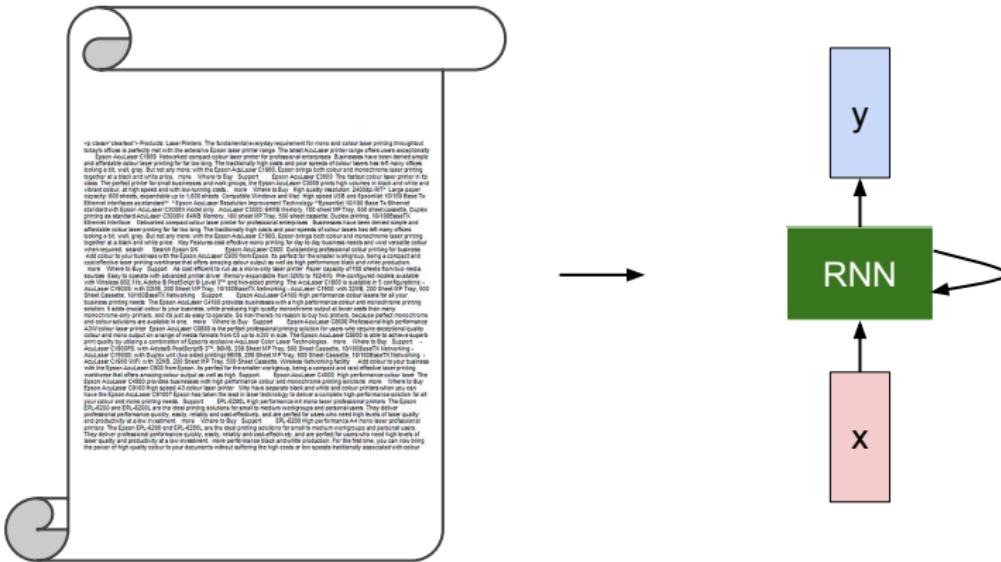


Simple RNN example: Character-level language model

Vocab: $\{h, e, l, o\}$, **Test time:** sample characters one at a time, feed back to model.



Extend this simple approach to full alphabet and punctuation characters



Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.

at first:

tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tkldg t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

↓ train more

"Tmont thithey" fomesscerliund

Keushey. Thom here

sheulke, anmerenith ol sivh I lalterthend Bleipile shuwyl fil on aseterlome coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

↓ train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly, and behs to so arwage fiving were to it belege, pavu say falling misfort how, and Gogition is so overelical and ofter.

↓ train more

"Why do what that day," replied Natasha, and wishing to himself the fact the princess, Princess Mary was easier, fed in had oftened him.

Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

VIOLA:

Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

How do we train a vanilla RNN?

Supervised learning via a loss function & mini-batch gradient descent.

Loss defined for one training sequence.

- Have a sequence $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_\tau$ of input vectors.
- For each \mathbf{x}_t in sequence have a target output \mathbf{y}_t .
- Define loss l_t between the y_t and \mathbf{p}_t for each t .
- Sum the loss over all time-steps

$$L(\mathbf{x}_1, \dots, \mathbf{x}_\tau, y_1, \dots, y_\tau, W, U, V, \mathbf{b}, \mathbf{c}) = \sum_{t=1}^{\tau} l_t$$

Loss function for a vanilla RNN

Common to use the cross-entropy loss:

$$l_t = -\log(\mathbf{y}_t^T \mathbf{p}_t)$$

thus

$$L(\mathbf{x}_{1:\tau}, y_{1:\tau}, W, U, V, \mathbf{b}, \mathbf{c}) = -\sum_{t=1}^{\tau} \log(\mathbf{y}_t^T \mathbf{p}_t)$$

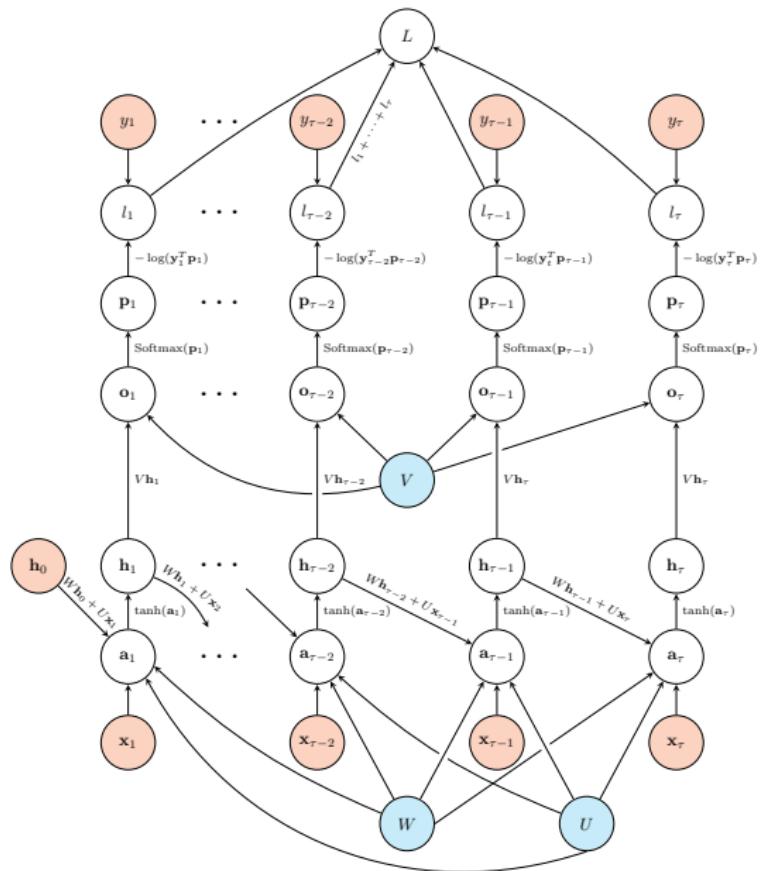
where $\mathbf{x}_{1:\tau} = \{\mathbf{x}_1, \dots, \mathbf{x}_\tau\}$ and $\mathbf{y}_{1:\tau} = \{y_1, \dots, y_\tau\}$.

- To implement mini-batch gradient descent need to compute

$$\frac{\partial L(\mathbf{x}_{1:\tau}, y_{1:\tau}, W, U, V, \mathbf{b}, \mathbf{c})}{\partial W}, \frac{\partial L(\mathbf{x}_{1:\tau}, y_{1:\tau}, W, U, V, \mathbf{b}, \mathbf{c})}{\partial U}, \dots$$

- You've guessed it, use back-prop...

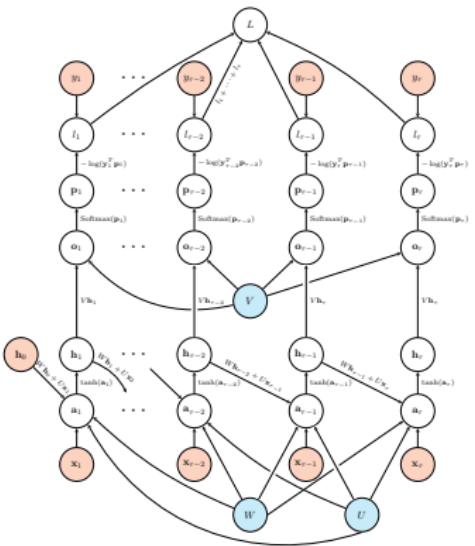
Computational Graph for vanilla RNN loss



- Loss for one labelled training sequence $\mathbf{x}_1, \dots, \mathbf{x}_\tau$
- Bias vectors have been omitted for clarity.

Back-prop for a vanilla RNN

Gradient of loss for the cross-entropy & softmax layers



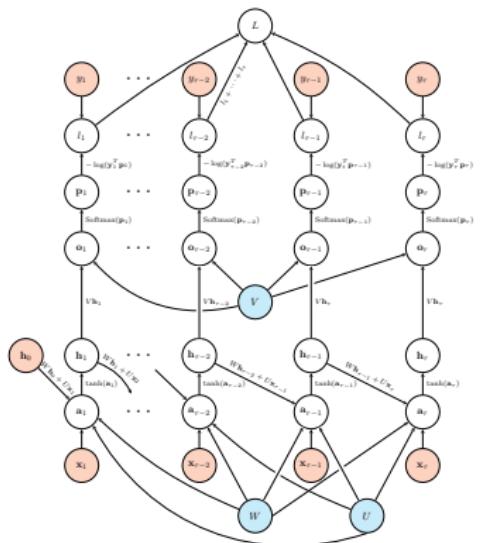
Know from prior dealings with cross-entropy loss:
for $t = 1, \dots, \tau$

$$\frac{\partial L}{\partial l_t} = 1$$

$$\frac{\partial L}{\partial \mathbf{p}_t} = \frac{\partial L}{\partial l_t} \frac{\partial l_t}{\partial \mathbf{p}_t} = -\frac{\mathbf{y}_t^T}{\mathbf{y}_t^T \mathbf{p}_t}$$

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{\partial L}{\partial \mathbf{p}_t} \frac{\partial \mathbf{p}_t}{\partial \mathbf{o}_t} = -(\mathbf{y}_t - \mathbf{p}_t)^T$$

Gradient of loss w.r.t. V



Children of node V are $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_\tau$. Thus

$$\frac{\partial L}{\partial \text{vec}(V)} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \text{vec}(V)}$$

Know

$$\begin{aligned} \mathbf{o}_t &= V \mathbf{h}_t \implies \mathbf{o}_t = (I_C \otimes \mathbf{h}_t^T) \text{vec}(V) \\ &\implies \frac{\partial \mathbf{o}_t}{\partial \text{vec}(V)} = I_C \otimes \mathbf{h}_t^T \end{aligned}$$

From previous reshapings know:

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{\tau} \mathbf{g}_t^T \mathbf{h}_t^T$$

where $\mathbf{g}_t = \frac{\partial L}{\partial \mathbf{o}_t}$.

Gradient of loss w.r.t. V

Children of node V are $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_\tau$. Thus

$$\frac{\partial L}{\partial \text{vec}(V)} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \text{vec}(V)}$$

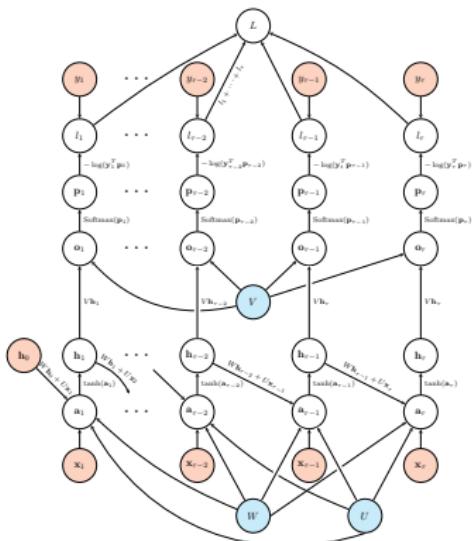
Know

$$\begin{aligned} \mathbf{o}_t &= V \mathbf{h}_t \implies \mathbf{o}_t = (I_C \otimes \mathbf{h}_t^T) \text{vec}(V) \\ &\implies \frac{\partial \mathbf{o}_t}{\partial \text{vec}(V)} = I_C \otimes \mathbf{h}_t^T \end{aligned}$$

From previous reshapings know:

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{\tau} \mathbf{g}_t^T \mathbf{h}_t^T \leftarrow \text{gradient needed for training network}$$

where $\mathbf{g}_t = \frac{\partial L}{\partial \mathbf{o}_t}$.



Gradient of loss w.r.t. \mathbf{h}_τ

\mathbf{h}_τ (last hidden state) has one child \mathbf{o}_τ thus

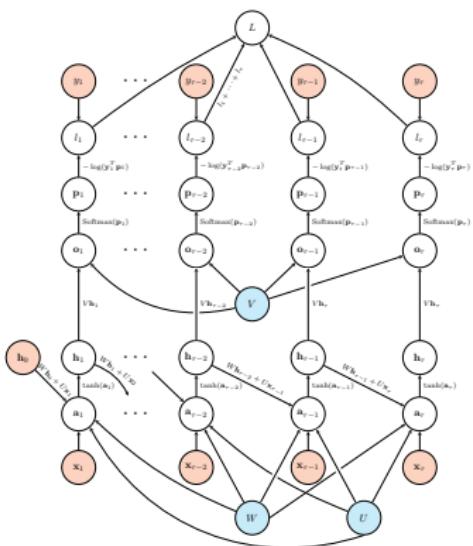
$$\frac{\partial L}{\partial \mathbf{h}_\tau} = \frac{\partial L}{\partial \mathbf{o}_\tau} \frac{\partial \mathbf{o}_\tau}{\partial \mathbf{h}_\tau}$$

Know

$$\mathbf{o}_\tau = V\mathbf{h}_\tau \implies \frac{\partial \mathbf{o}_\tau}{\partial \mathbf{h}_\tau} = V$$

Thus

$$\frac{\partial L}{\partial \mathbf{h}_\tau} = \frac{\partial L}{\partial \mathbf{o}_\tau} V$$



Gradient of loss w.r.t. \mathbf{h}_t

If $1 \leq t \leq \tau - 1$ then \mathbf{h}_t has children \mathbf{o}_t and \mathbf{a}_{t+1}

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{a}_{t+1}} \frac{\partial \mathbf{a}_{t+1}}{\partial \mathbf{h}_t}$$

Know

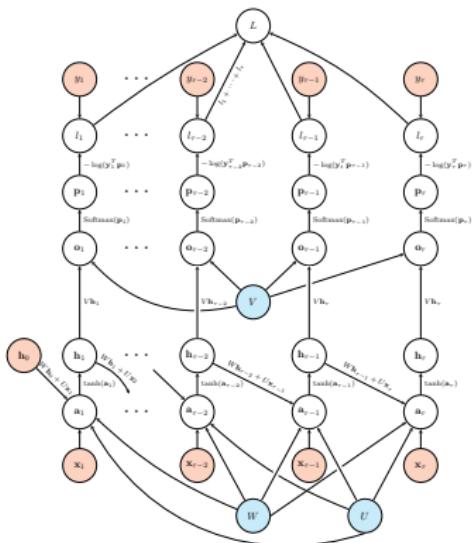
$$\mathbf{o}_t = V\mathbf{h}_t \implies \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} = V$$

and

$$\mathbf{a}_{t+1} = W\mathbf{h}_t + U\mathbf{x}_{t+1} \implies \frac{\partial \mathbf{a}_{t+1}}{\partial \mathbf{h}_t} = W$$

Thus

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{o}_t} V + \frac{\partial L}{\partial \mathbf{a}_{t+1}} W$$



Gradient of loss w.r.t. \mathbf{h}_t

If $1 \leq t \leq \tau - 1$ then \mathbf{h}_t has children \mathbf{o}_t and \mathbf{a}_{t+1}

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{a}_{t+1}} \frac{\partial \mathbf{a}_{t+1}}{\partial \mathbf{h}_t}$$

Know

$$\mathbf{o}_t = V\mathbf{h}_t \implies \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} = V$$

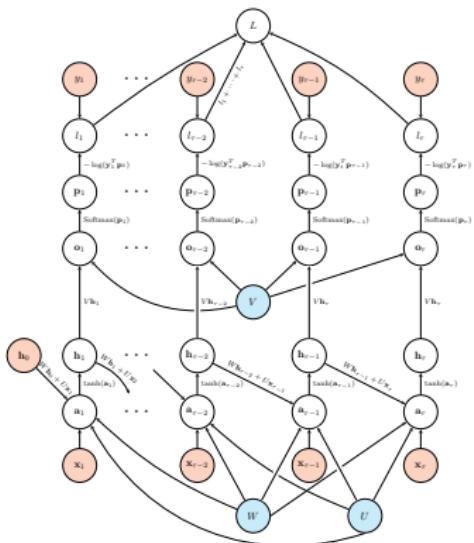
and

$$\mathbf{a}_{t+1} = W\mathbf{h}_t + U\mathbf{x}_{t+1} \implies \frac{\partial \mathbf{a}_{t+1}}{\partial \mathbf{h}_t} = W$$

Thus

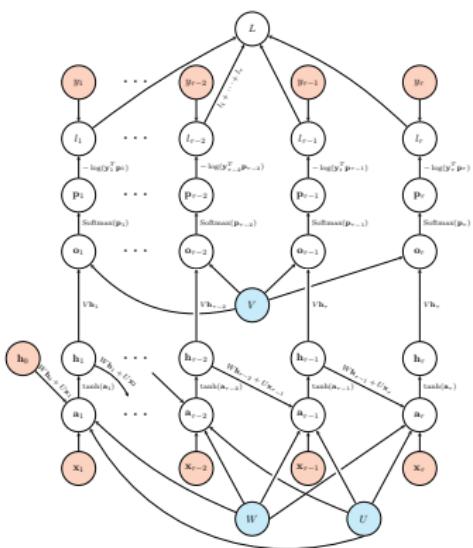
$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{o}_t} V + \frac{\partial L}{\partial \mathbf{a}_{t+1}} W$$

Have two different time steps in expression \implies
must iterate backwards in time to compute all $\frac{\partial L}{\partial \mathbf{h}_t}$



Gradient of loss w.r.t. \mathbf{a}_t

The gradient w.r.t. \mathbf{a}_t



$$\frac{\partial L}{\partial \mathbf{a}_t} = \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{a}_t}$$

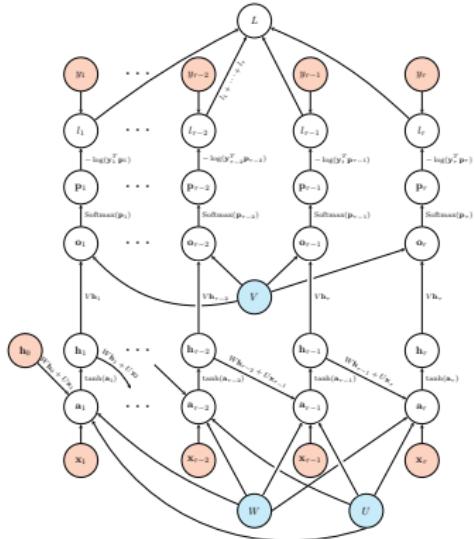
Know

$$\begin{aligned} \mathbf{h}_t = \tanh(\mathbf{a}_t) \implies \frac{\partial \mathbf{h}_t}{\partial \mathbf{a}_t} &= \text{diag} (\tanh'(\mathbf{a}_t)) \\ &= \text{diag} (1 - \tanh^2(\mathbf{a}_t)) \end{aligned}$$

Thus

$$\frac{\partial L}{\partial \mathbf{a}_t} = \frac{\partial L}{\partial \mathbf{h}_t} \text{diag} (1 - \tanh^2(\mathbf{a}_t))$$

Recursively compute gradients for all \mathbf{a}_t and \mathbf{h}_t



- Assume $\frac{\partial L}{\partial \mathbf{o}_t}$ calculated for $1 \leq t \leq \tau$.

- Calculate

$$\frac{\partial L}{\partial \mathbf{h}_\tau} = \frac{\partial L}{\partial \mathbf{o}_\tau} V \quad \& \quad \frac{\partial L}{\partial \mathbf{a}_\tau} = \frac{\partial L}{\partial \mathbf{h}_\tau} \text{diag}(1 - \tanh^2(\mathbf{a}_\tau))$$

- for $t = \tau - 1, \tau - 2, \dots, 1$

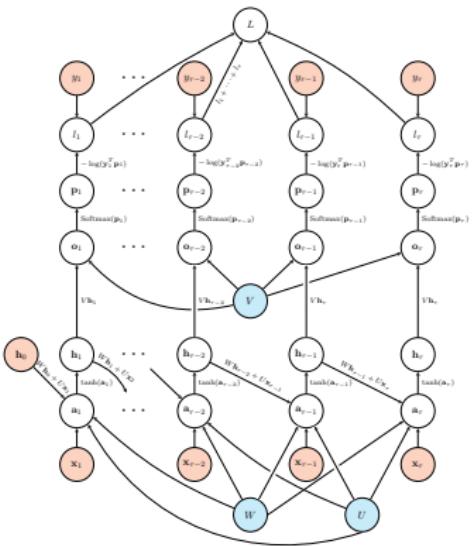
- Compute

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{o}_t} V + \frac{\partial L}{\partial \mathbf{a}_{t+1}} W$$

- Compute

$$\frac{\partial L}{\partial \mathbf{a}_t} = \frac{\partial L}{\partial \mathbf{h}_t} \text{diag}(1 - \tanh^2(\mathbf{a}_t))$$

Gradient of loss w.r.t. W



The gradient of the loss w.r.t. node W .

Children of W are $\mathbf{a}_1, \dots, \mathbf{a}_\tau$ thus

$$\frac{\partial L}{\partial \text{vec}(W)} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \mathbf{a}_t} \frac{\partial \mathbf{a}_t}{\partial \text{vec}(W)}$$

Know

$$\begin{aligned} \mathbf{a}_t &= W\mathbf{h}_{t-1} + U\mathbf{x}_t \implies \mathbf{a}_t = (I_m \otimes \mathbf{h}_{t-1}^T) \text{vec}(W) + U\mathbf{x}_t \\ &\implies \frac{\partial \mathbf{a}_t}{\partial \text{vec}(W)} = I_m \otimes \mathbf{h}_{t-1}^T \end{aligned}$$

From previous reshapings know:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{\tau} \mathbf{g}_t^T \mathbf{h}_{t-1}^T$$

where $\mathbf{g}_t = \frac{\partial L}{\partial \mathbf{a}_t}$.

Gradient of loss w.r.t. W

The gradient of the loss w.r.t. node W .

Children of W are $\mathbf{a}_1, \dots, \mathbf{a}_\tau$ thus

$$\frac{\partial L}{\partial \text{vec}(W)} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \mathbf{a}_t} \frac{\partial \mathbf{a}_t}{\partial \text{vec}(W)}$$

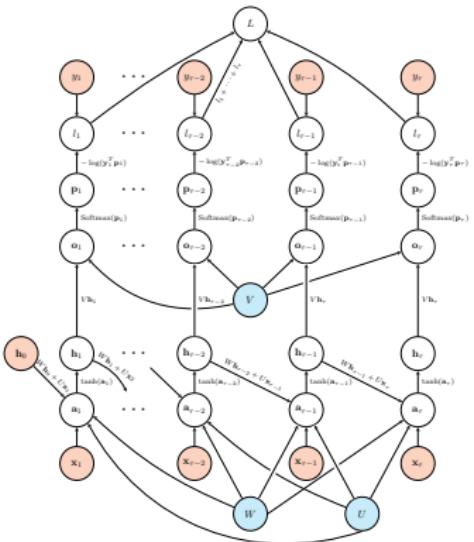
Know

$$\begin{aligned} \mathbf{a}_t &= W\mathbf{h}_{t-1} + U\mathbf{x}_t \implies \mathbf{a}_t = (I_m \otimes \mathbf{h}_{t-1}^T)\text{vec}(W) + U\mathbf{x}_t \\ &\implies \frac{\partial \mathbf{a}_t}{\partial \text{vec}(W)} = I_m \otimes \mathbf{h}_{t-1}^T \end{aligned}$$

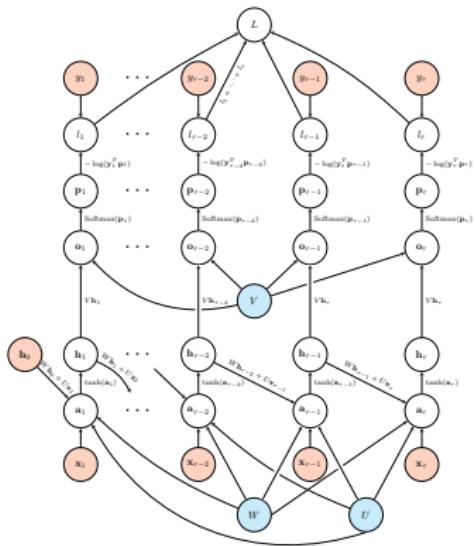
From previous reshapings know:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{\tau} \mathbf{g}_t^T \mathbf{h}_{t-1}^T \leftarrow \text{gradient needed for training network}$$

where $\mathbf{g}_t = \frac{\partial L}{\partial \mathbf{a}_t}$.



Gradient of loss w.r.t. U



The gradient of the loss w.r.t. node U .

Children of V are $\mathbf{a}_1, \dots, \mathbf{a}_\tau$ thus

$$\frac{\partial L}{\partial \text{vec}(U)} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \mathbf{a}_t} \frac{\partial \mathbf{a}_t}{\partial \text{vec}(U)}$$

Know

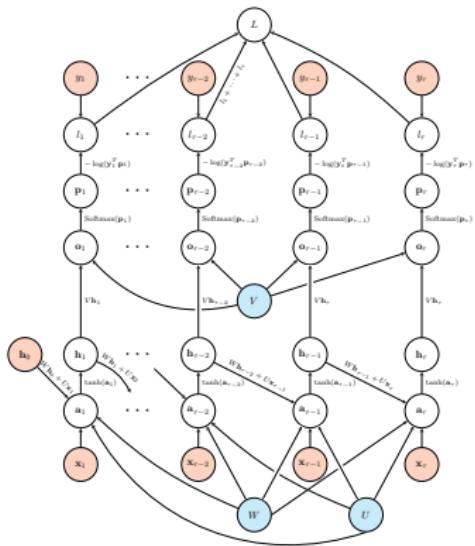
$$\begin{aligned} \mathbf{a}_t &= W\mathbf{h}_{t-1} + U\mathbf{x}_t \implies \mathbf{a}_t = W\mathbf{h}_{t-1} + (I_m \otimes \mathbf{x}_t^T)\text{vec}(U) \\ &\implies \frac{\partial \mathbf{a}_t}{\partial \text{vec}(U)} = I_m \otimes \mathbf{x}_t^T \end{aligned}$$

From previous reshapings know:

$$\frac{\partial L}{\partial U} = \sum_{t=1}^{\tau} \mathbf{g}_t^T \mathbf{x}_t^T$$

where $\mathbf{g}_t = \frac{\partial L}{\partial \mathbf{a}_t}$.

Gradient of loss w.r.t. U



The gradient of the loss w.r.t. node U .

Children of V are $\mathbf{a}_1, \dots, \mathbf{a}_\tau$ thus

$$\frac{\partial L}{\partial \text{vec}(U)} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \mathbf{a}_t} \frac{\partial \mathbf{a}_t}{\partial \text{vec}(U)}$$

Know

$$\begin{aligned} \mathbf{a}_t &= W\mathbf{h}_{t-1} + U\mathbf{x}_t \implies \mathbf{a}_t = W\mathbf{h}_{t-1} + (I_m \otimes \mathbf{x}_t^T)\text{vec}(U) \\ &\implies \frac{\partial \mathbf{a}_t}{\partial \text{vec}(U)} = I_m \otimes \mathbf{x}_t^T \end{aligned}$$

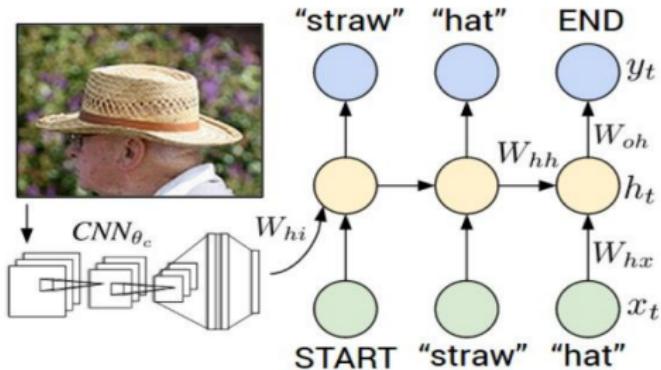
From previous reshapings know:

$$\frac{\partial L}{\partial U} = \sum_{t=1}^{\tau} \mathbf{g}_t^T \mathbf{x}_t^T \leftarrow \text{gradient needed for training network}$$

where $\mathbf{g}_t = \frac{\partial L}{\partial \mathbf{a}_t}$.

RNNs in Image Translation Application

Image Captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.

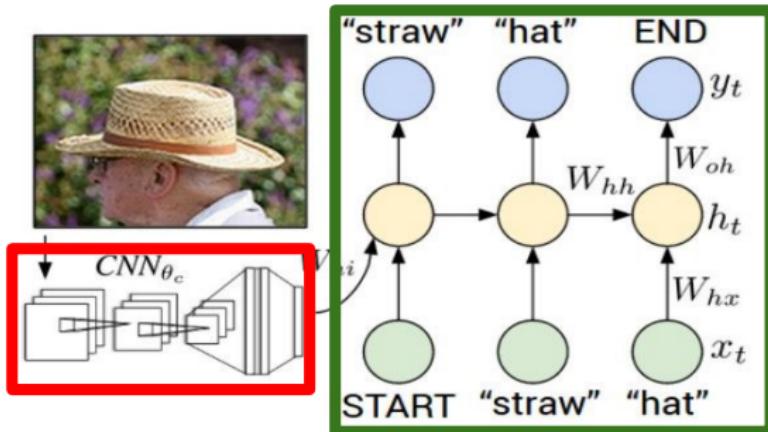
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

Show and Tell: A Neural Image Caption Generator, Vinyals et al.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.

Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network



Convolutional Neural Network

test image



image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax

image



test image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

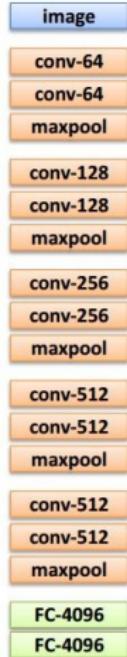
FC-4096

FC-4096

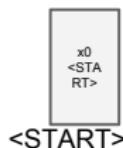
FC-1000

softmax

X

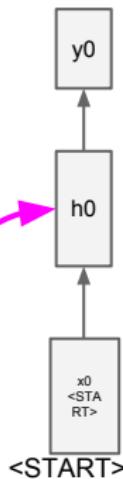


test image





test image



before:

$$h = \tanh(Wxh * x + Whh * h)$$

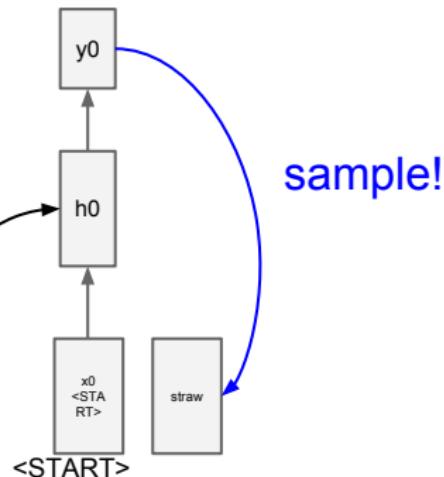
now:

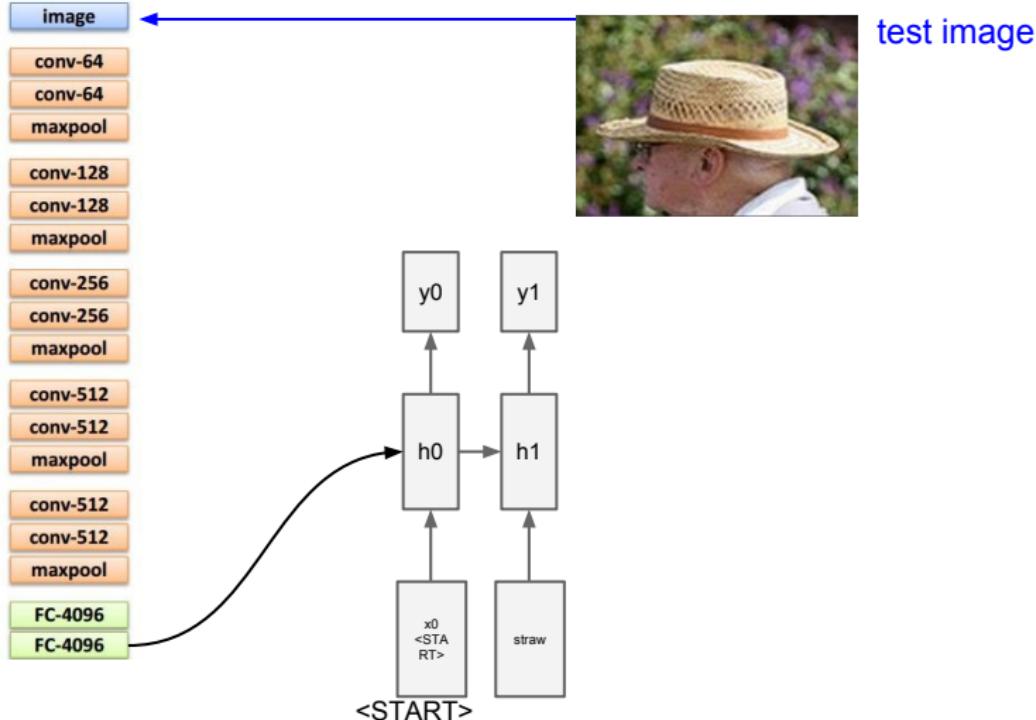
$$h = \tanh(Wxh * x + Whh * h + WiH * v)$$

V



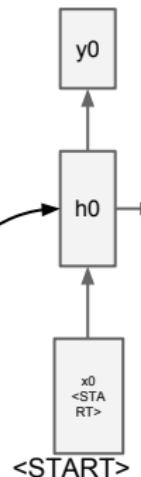
test image



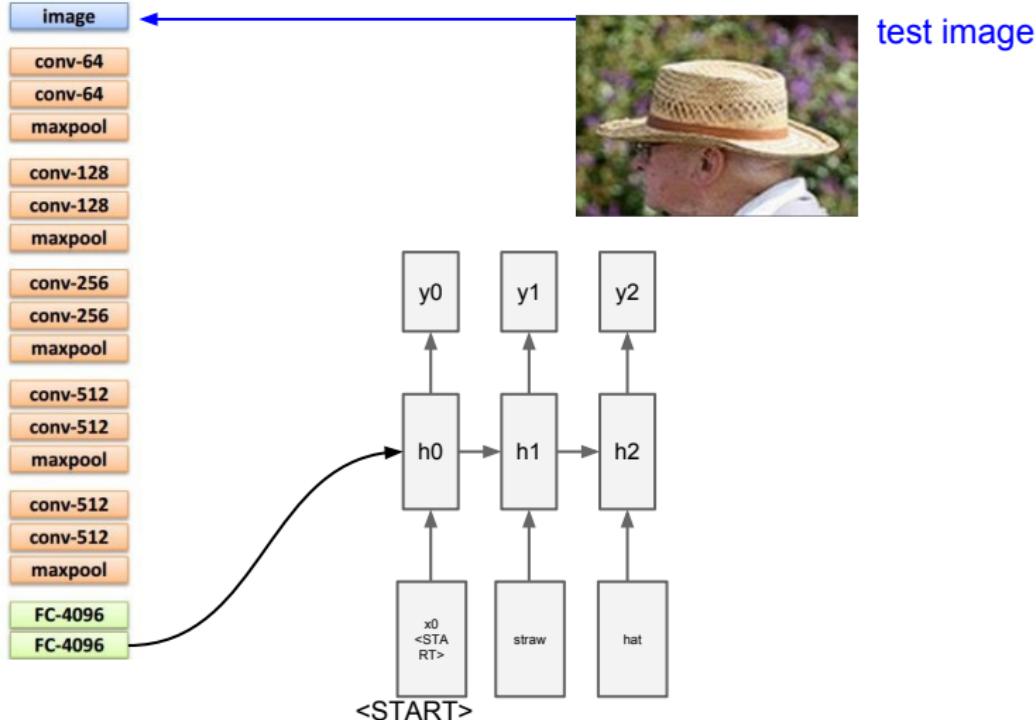


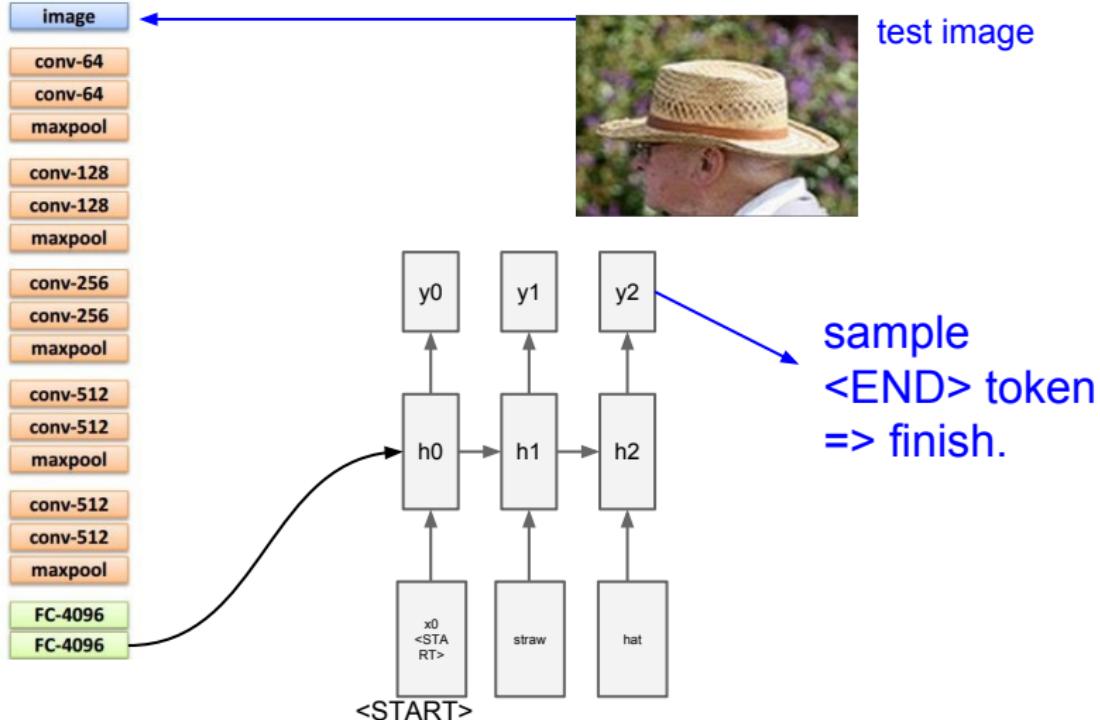


test image



sample!







"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."



"a woman holding a teddy bear in front of a mirror."



"a horse is standing in the middle of a road."

Evaluation of text translation results

- Tricky to do automatically!
- Ideally want humans to evaluate
 - What do you ask?
 - Can't use human evaluation for validating models - too slow and expensive.
- Use standard machine translation metrics instead
 - BLEU
 - ROUGE CIDEER
 - Meteor

Image Sentence Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.



Microsoft COCO
[Tsung-Yi Lin et al. 2014]
mscoco.org

currently:
~120K images
~5 sentences each

Problem of exploding and vanishing gradients in an RNN

Focus on gradient of loss w.r.t. W

- Take a closer look at

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{\tau} \mathbf{g}_t^T \mathbf{h}_{t-1}^T \quad \text{where } \mathbf{g}_t = \frac{\partial L}{\partial \mathbf{a}_t}$$

- $\Rightarrow \frac{\partial L}{\partial W}$ depends on \mathbf{h}_{t-1} and $\frac{\partial L}{\partial \mathbf{a}_t}$ for $t = 1, \dots, \tau$.
- Let's take a closer look at $\frac{\partial L}{\partial \mathbf{a}_t}$

- Denote

$$\mathbf{g}_{\mathbf{o}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \quad \text{and} \quad D(\mathbf{a}_t) = \text{diag}(1 - \tanh^2(\mathbf{a}_t))$$

- Remember

$$\frac{\partial L}{\partial \mathbf{h}_\tau} = \mathbf{g}_{\mathbf{o}_\tau} V \implies \frac{\partial L}{\partial \mathbf{a}_\tau} = \mathbf{g}_{\mathbf{o}_\tau} V D(\mathbf{a}_\tau)$$

and for $t = \tau - 1, \tau - 2, \dots, 1$:

$$\frac{\partial L}{\partial \mathbf{h}_t} = \mathbf{g}_{\mathbf{o}_t} V + \frac{\partial L}{\partial \mathbf{a}_{t+1}} W \quad \text{and} \quad \frac{\partial L}{\partial \mathbf{a}_t} = \frac{\partial L}{\partial \mathbf{h}_t} D(\mathbf{a}_t)$$

- Then you can show by recursive substitution that

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=1}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

and

$$\frac{\partial L}{\partial \mathbf{a}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=0}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

- Denote

$$\mathbf{g}_{\mathbf{o}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \quad \text{and} \quad D(\mathbf{a}_t) = \text{diag}(1 - \tanh^2(\mathbf{a}_t))$$

- Then you can show by recursive substitution that

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=1}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

and

$$\frac{\partial L}{\partial \mathbf{a}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \underbrace{\left(\prod_{k=0}^{j-t} D(\mathbf{a}_{t+k}) \right)}_{\text{likely has small values on diagonal}} W^{j-t}$$

Why? Each matrix $D(\mathbf{a}_{t+k})$ has $\tanh'(\mathbf{a}_{t+k})$ on its diagonal and $0 \leq \tanh'(a) \leq 1$. Thus $(\tanh'(a))^{j-t+1}$ is highly likely to have a small value even for not too large $j - t + 1$.

$\implies \frac{\partial L}{\partial \mathbf{a}_t}$ only depends on first few entries in the sum.

- Denote

$$\mathbf{g}_{\mathbf{o}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \quad \text{and} \quad D(\mathbf{a}_t) = \text{diag}(1 - \tanh^2(\mathbf{a}_t))$$

- Then you can show by recursive substitution that

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=1}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

and

$$\frac{\partial L}{\partial \mathbf{a}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=0}^{j-t} D(\mathbf{a}_{t+k}) \right)$$

W^{j-t}
potentially has very large or small values

Why?

- Remember W has size $m \times m$.
- Assume W is diagonalizable.
- Let its eigen-decomposition be

$$W = Q\Lambda Q^T$$

where Q is orthogonal and Λ is a diagonal matrix containing the eigenvalues of W .

- Then

$$W^n = Q\Lambda^n Q^T$$

- Let $\lambda_1, \dots, \lambda_m$ be the e-values of W . Thus
 - If $|\lambda_i| > 1 \implies \lambda_i^n$ will explode as n increases.
 - If $|\lambda_i| < 1 \implies \lambda_i^n \rightarrow 0$ as n increases.

- Denote

$$\mathbf{g}_{\mathbf{o}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \quad \text{and} \quad D(\mathbf{a}_t) = \text{diag}(1 - \tanh^2(\mathbf{a}_t))$$

- Then you can show by recursive substitution that

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=1}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

and

$$\frac{\partial L}{\partial \mathbf{a}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=0}^{j-t} D(\mathbf{a}_{t+k}) \right)$$

W^{j-t}
potentially has very large or small values

Thus for sufficiently large $j - t$ either entries in W^{j-t} can explode or vanish.

- Denote

$$\mathbf{g}_{\mathbf{o}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \quad \text{and} \quad D(\mathbf{a}_t) = \text{diag}(1 - \tanh^2(\mathbf{a}_t))$$

- Then you can show by recursive substitution that

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=1}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

and

$$\frac{\partial L}{\partial \mathbf{a}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=0}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

- If W^{j-t} explodes for $j - t > N \implies \frac{\partial L}{\partial \mathbf{a}_t}$ explodes $\implies \frac{\partial L}{\partial W}$ explodes.
- If W^{j-t} vanishes for $j - t > N$
 - $\implies \frac{\partial L}{\partial \mathbf{a}_t}$ only has contributions from nearby $\mathbf{g}_{\mathbf{o}_{t'}}$, where $t \leq t' \leq t + N$
 - $\implies \frac{\partial L}{\partial W}$ is based on aggregation of gradients from subsets of temporally nearby states.

- Denote

$$\mathbf{g}_{\mathbf{o}_t} = \frac{\partial L}{\partial \mathbf{o}_t} \quad \text{and} \quad D(\mathbf{a}_t) = \text{diag}(1 - \tanh^2(\mathbf{a}_t))$$

- Then you can show by recursive substitution that

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=1}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

and

$$\frac{\partial L}{\partial \mathbf{a}_t} = \sum_{j=t}^{\tau} \mathbf{g}_{\mathbf{o}_j} V \left(\prod_{k=0}^{j-t} D(\mathbf{a}_{t+k}) \right) W^{j-t}$$

- If W^{j-t} explodes for $j - t > N \implies \frac{\partial L}{\partial \mathbf{a}_t}$ explodes $\implies \frac{\partial L}{\partial W}$ explodes.
- If W^{j-t} vanishes for $j - t > N$
 - $\implies \frac{\partial L}{\partial \mathbf{a}_t}$ only has contributions from nearby $\mathbf{g}_{\mathbf{o}_{t'}}$, where $t \leq t' \leq t + N$
 - $\implies \frac{\partial L}{\partial W}$ is based on aggregation of gradients from subsets of temporally nearby states.
 - \implies Cannot learn long-range dependencies between states.

Solution to Exploding & Vanishing Gradients

Easy solution to exploding gradients

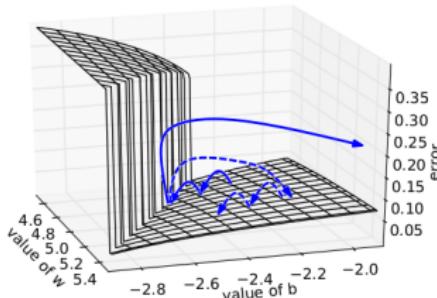
- **Gradient clipping**

Let $G = \frac{\partial L}{\partial W}$ then

$$G = \begin{cases} \frac{\theta}{\|G\|} G & \text{if } \|G\| \geq \theta \\ G & \text{otherwise} \end{cases}$$

where θ is some sensible threshold.

- A simple heuristic first introduced by Thomas Mikolov.



Dashed arrow shows what happens when the gradient is rescaled to a fixed size when its norm is above a threshold.

Easy partial solutions to vanishing gradients

- **Solution 1:** Initialize W as the identity matrix as opposed a random initialization.
- **Solution 2:** Use ReLU instead of tanh as the non-linear activation function.

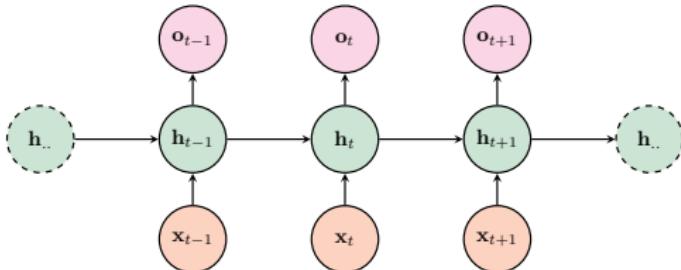
Easy partial solutions to vanishing gradients

- **Solution 1:** Initialize W as the identity matrix as opposed a random initialization.
- **Solution 2:** Use ReLU instead of tanh as the non-linear activation function.

Still hard for an RNN to capture long-term dependencies.

Long-Short-Term-Memories (LSTMs) - capturing long-range dependencies

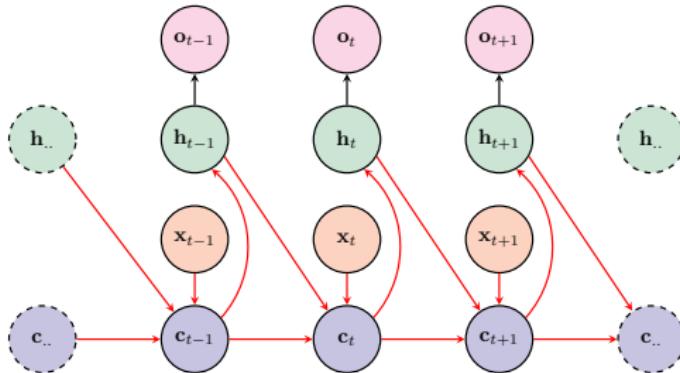
LSTMs Core Idea: Introduce a memory cell



High-level graphic of an RNN

- LSTMs similar to RNN but they introduce a memory cell state c_t .

LSTMs Core Idea: Introduce a memory cell



High-level graphic of a LSTM

- LSTMs similar to RNN but they introduce a memory cell state \mathbf{c}_t .
- LSTMs have the ability to remove or add information to \mathbf{c}_t regulated by structures called gates based on context.
- Update of \mathbf{c}_t designed so gradient flows through these nodes backward in time easily.
- \mathbf{c}_t then controls what information from \mathbf{h}_{t-1} and \mathbf{x}_t and \mathbf{c}_{t-1} should be used to generate \mathbf{h}_t .

- LSTMs (Hochreiter & Schmidhuber, 1997) better at capturing long term dependencies.
- Introduces gates to calculate $\mathbf{h}_t, \mathbf{c}_t$ from $\mathbf{c}_{t-1}, \mathbf{h}_{t-1}$ and \mathbf{x}_t .
- Formal description of a LSTM unit:

$$\mathbf{i}_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t) \quad \text{Input gate}$$

$$\mathbf{f}_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t) \quad \text{Forget gate}$$

$$\mathbf{o}_t = \sigma(W_o \mathbf{h}_{t-1} + U_o \mathbf{x}_t) \quad \text{Output/Exposure gate}$$

$$\tilde{\mathbf{c}}_t = \tanh(W_c \mathbf{h}_{t-1} + U_c \mathbf{x}_t) \quad \text{New memory cell}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad \text{Final memory cell}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where

- $\sigma(\cdot)$ is the sigmoid function and
- \odot denotes element by element multiplication.

LSTM basic unit to help with visualization

- Let

$$W_{\text{all}} = \begin{pmatrix} W_f \\ W_i \\ W_o \\ W_c \end{pmatrix} \quad U_{\text{all}} = \begin{pmatrix} U_f \\ U_i \\ U_o \\ U_c \end{pmatrix}$$

- Begin to compute all the gates and the new memory cell

$$\mathbf{a}_t = W_{\text{all}} \mathbf{h}_{t-1} + U_{\text{all}} \mathbf{x}_t$$

- Extract the individual gate vectors and the new memory cell

$$\mathbf{f}_t = \sigma(E_1 \mathbf{a}_t)$$

$$\mathbf{i}_t = \sigma(E_2 \mathbf{a}_t)$$

$$\mathbf{o}_t = \sigma(E_3 \mathbf{a}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(E_4 \mathbf{a}_t)$$

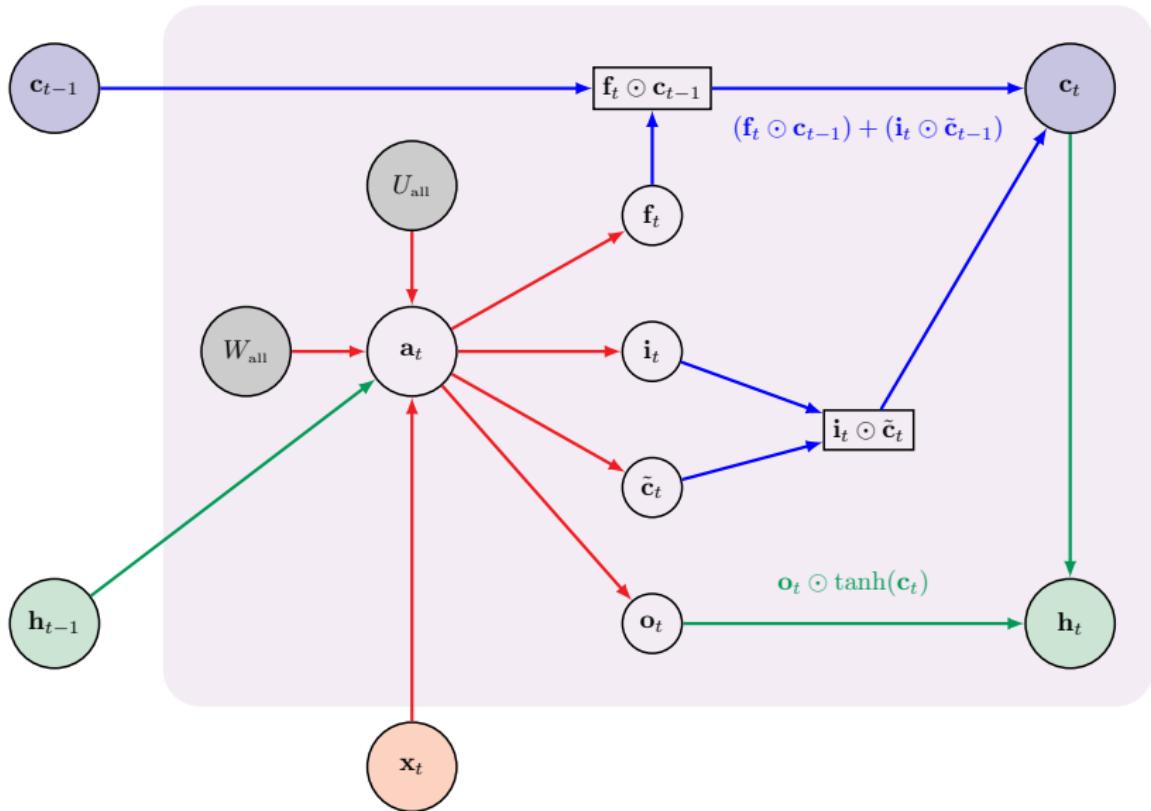
where

$$E_1 = (I_m \quad 0_{m \times m} \quad 0_{m \times m} \quad 0_{m \times m}) ,$$

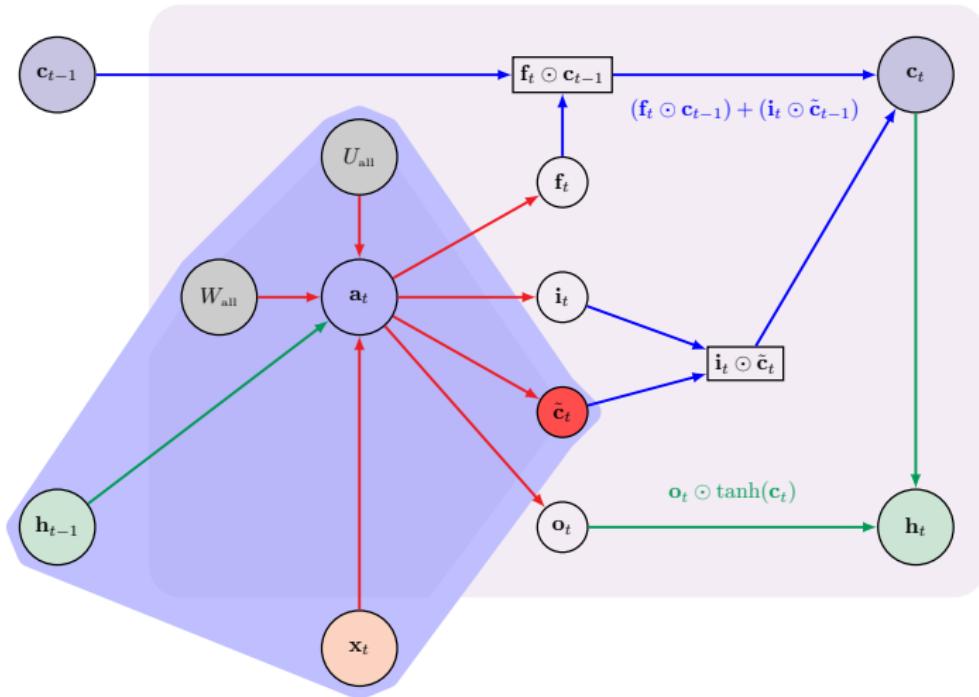
$$E_2 = (0_{m \times m} \quad I_m \quad 0_{m \times m} \quad 0_{m \times m}) ,$$

etc

LSTMs basic unit

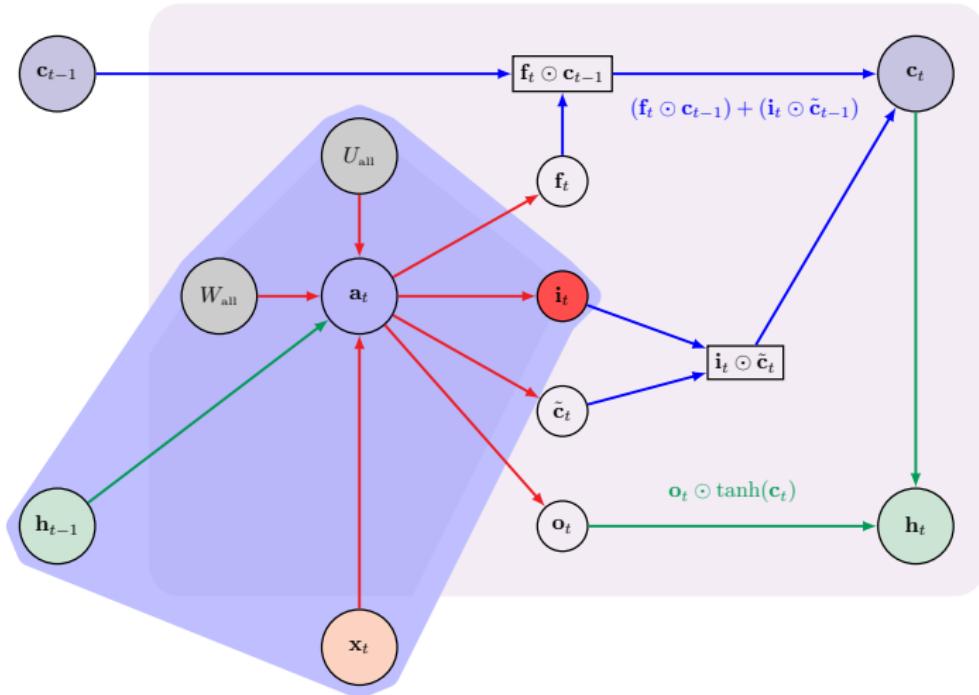


LSTMs basic unit



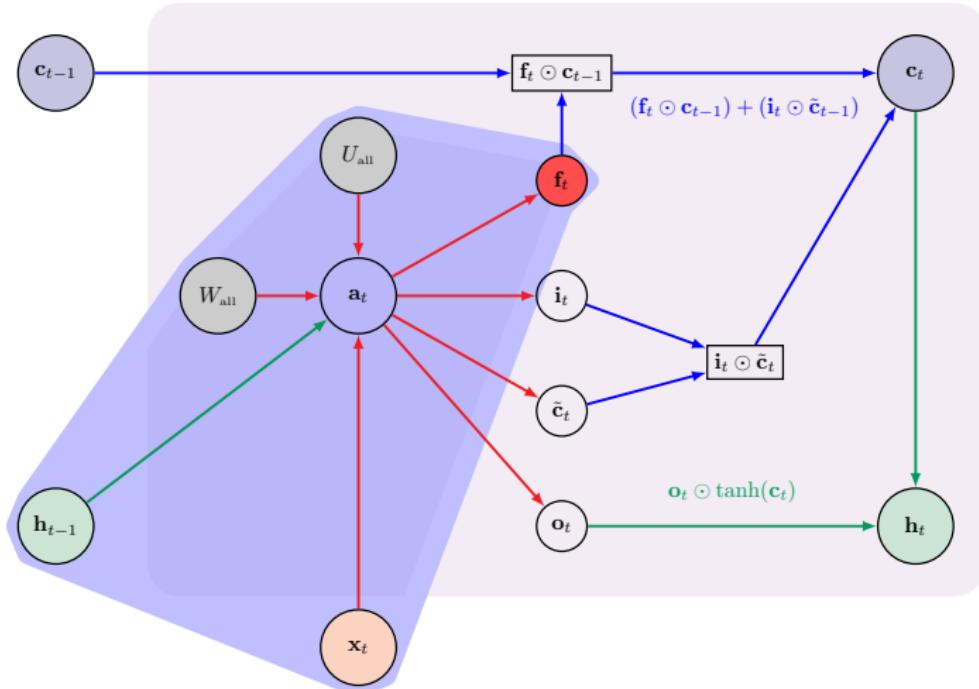
New temporary memory: Use x_t and h_{t-1} to generate new memory that includes aspects of x_t .

LSTMs basic unit



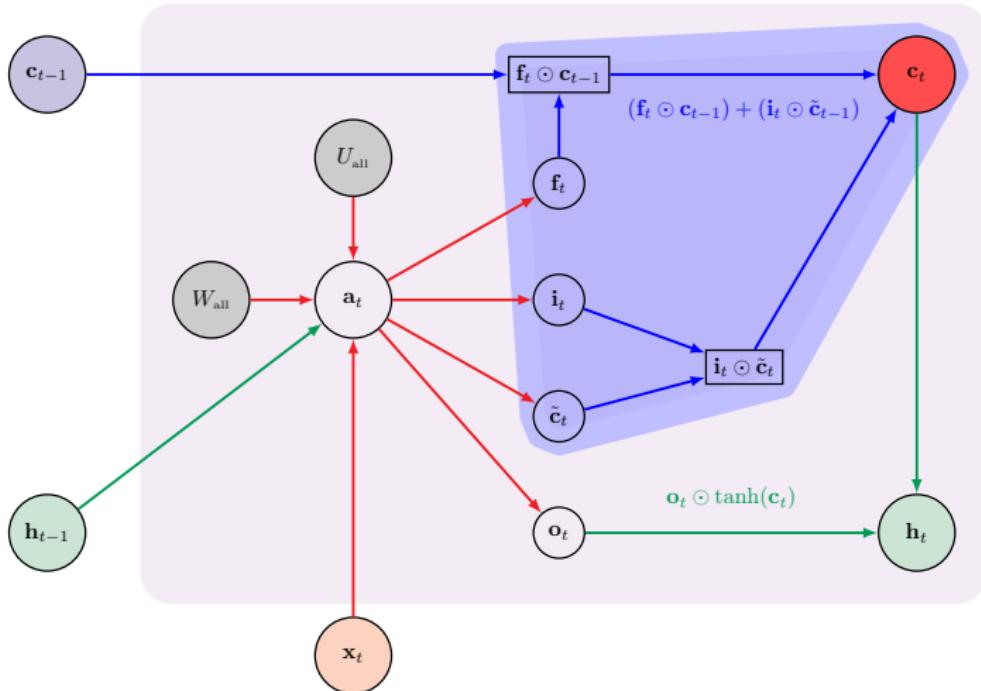
Input gate: Use x_t and h_{t-1} to determine whether the temporary memory \tilde{c}_t is worth preserving.

LSTMs basic unit



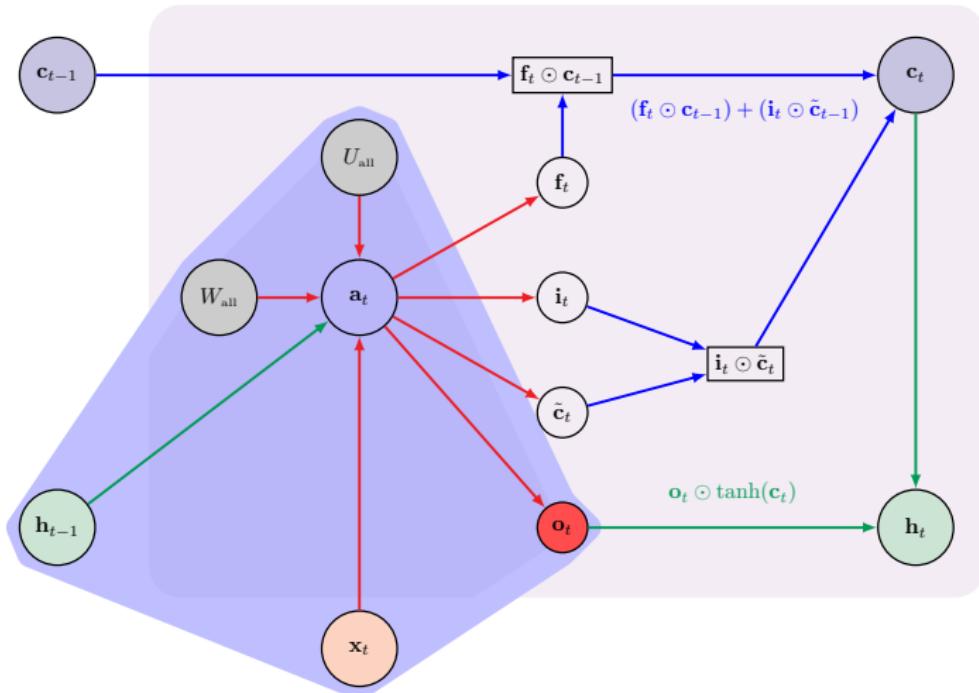
Forget gate: Assess which parts of past memory cell c_{t-1} should be included in c_t .

LSTMs basic unit



Updated memory state: Use the forget and input gates to combine the new temporary memory and the current memory cell state to get c_t .

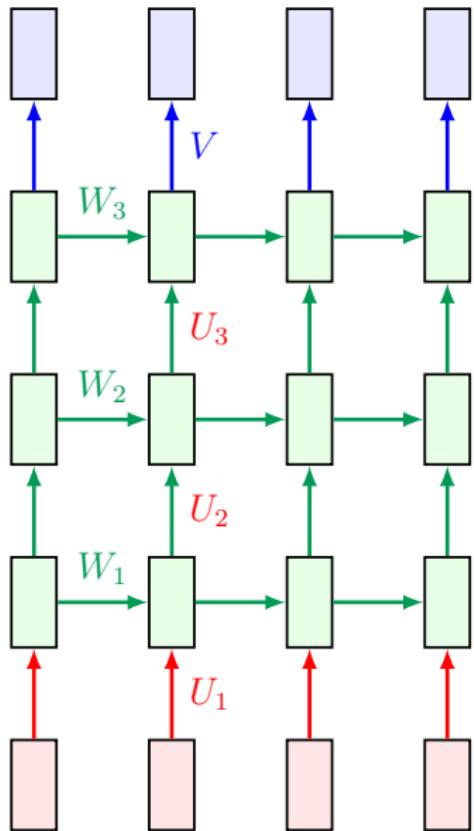
LSTMs basic unit



Output gate: Decides which part of c_t should be exposed to h_t .

Can go deep with LSTMs & RNNs

Multi-layer RNN



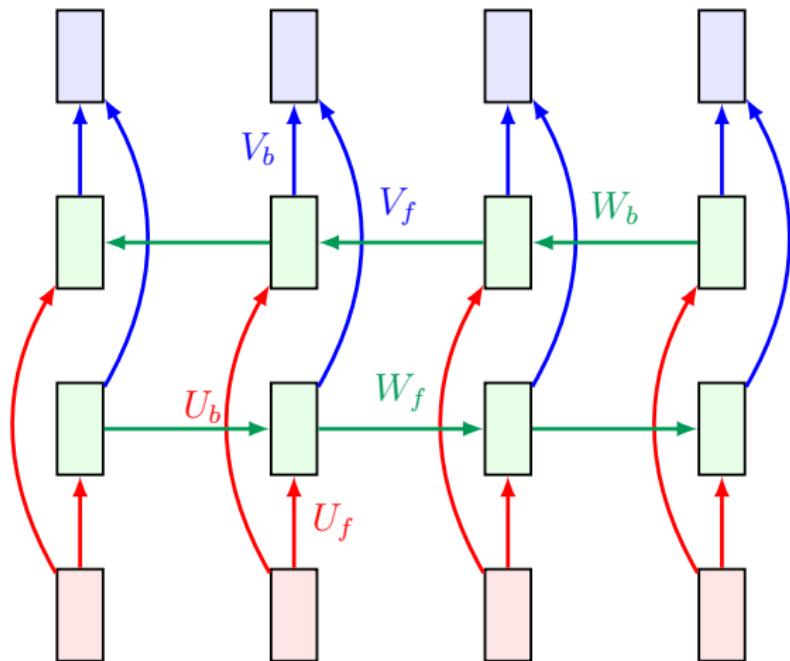
for $l = 1, 2, 3$

$$\mathbf{h}_t^{(l)} = \tanh \left(W_l \mathbf{h}_{t-1}^{(l)} + U_l \mathbf{h}_t^{(l-1)} + \mathbf{b}_l \right)$$

where $h_t^{(0)} = \mathbf{x}_t$

Can go forward & backward in time with LSTMs & RNNs

Bidirectional RNN



- Have separate RNNs in each direction and then jointly make prediction.

- RNNs allow a lot of flexibility in architecture design
- Backward flow of gradients in RNN can explode or vanish.
- Vanilla RNNs are simple but find it hard to learn long-term dependencies.
- Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Common to use LSTMs: their additive interactions improve gradient flow