

# Week 10: Recurrent Neural Networks

Instructor: Ruixuan Wang  
wangruix5@mail.sysu.edu.cn

School of Data and Computer Science  
Sun Yat-Sen University

28 April (for 02 May), 2019

① Word2vec & language modelling

② RNN & LSTM

③ RNN app examples

④ RNN structures

# Motivation: natural language processing

- Sentence or document classification (topic, sentiment)
- Topic modelling
- Translation
- Chatbots, dialogue system, assistant
- Summarization

Content and figures in this section mainly from <https://m2dsupsdclass.github.io/lectures-labs/> and <http://mccormickml.com/2016/04/27/word2vec-resources/>

# Word representation

- Words are originally represented as 1-hot vectors

# Word representation

- Words are originally represented as 1-hot vectors
- Large vocabulary of possible words
- Use of word **embeddings** as inputs in deep NLP models
- Word embeddings usually have dimensions 50 - 300

# Word representation

- Words are originally represented as 1-hot vectors
- Large vocabulary of possible words
- Use of word **embeddings** as inputs in deep NLP models
- Word embeddings usually have dimensions 50 - 300
- Then how to obtain such embedding?

# Word2vec: skip-gram model

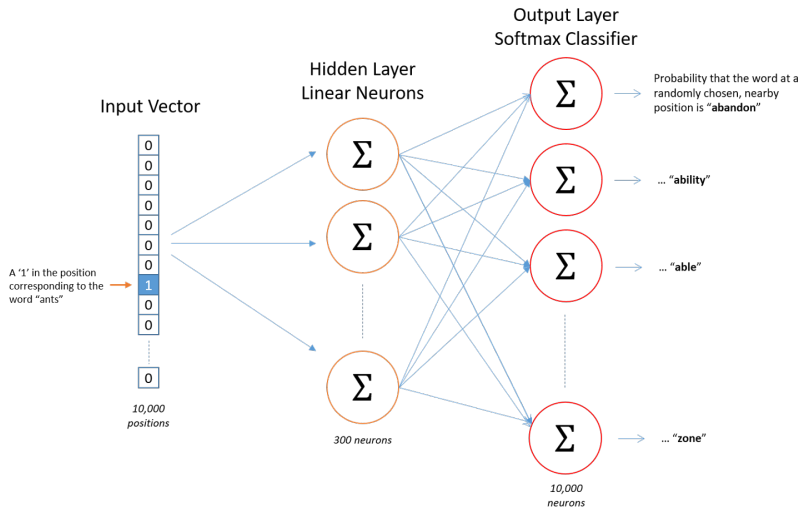
- Given central word, pred occurrence of other words in context

## Source Text

## Training Samples

The quick brown fox jumps over the lazy dog.	→	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog.	→	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog.	→	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog.	→	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

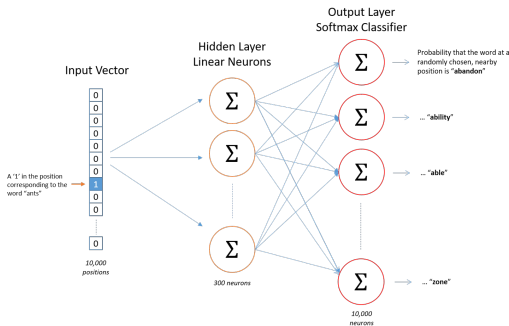
# Skip-gram model: a simple 2-layer neural network





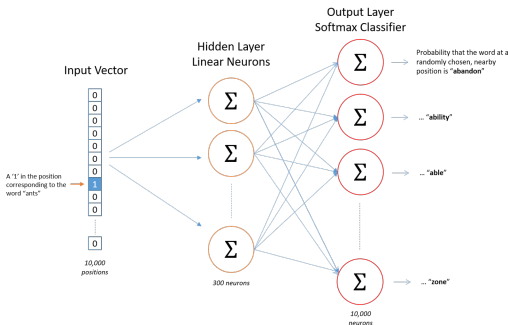
# Skip-gram model (cont')

- Two words having similar contexts: 'intelligent' and 'smart', 'ant' and 'ants', etc.



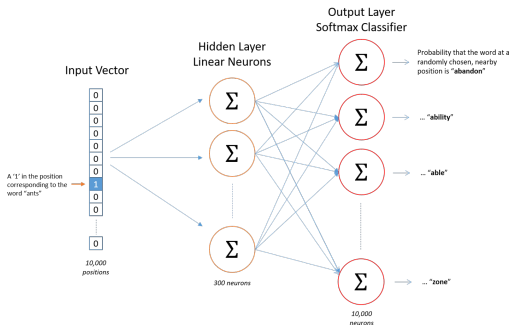
# Skip-gram model (cont')

- Two words having similar contexts: 'intelligent' and 'smart', 'ant' and 'ants', etc.
- If two words have very similar contexts, then skip-gram model needs to output similar results.

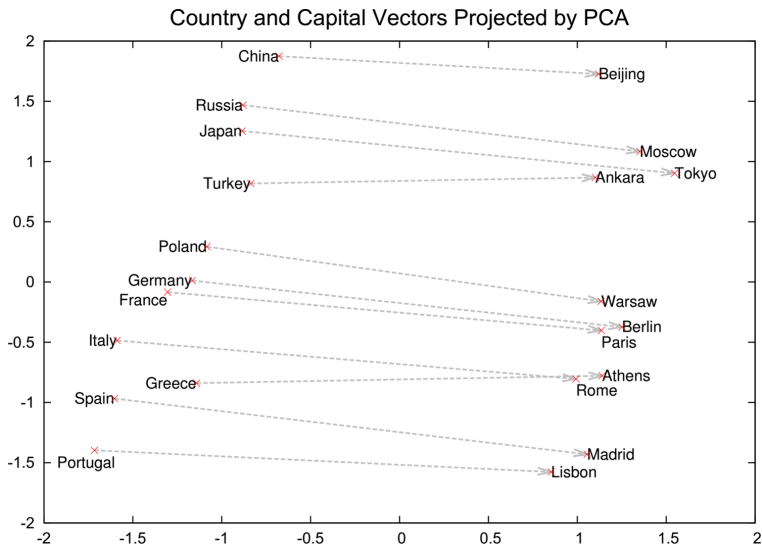


# Skip-gram model (cont')

- Two words having similar contexts: 'intelligent' and 'smart', 'ant' and 'ants', etc.
- If two words have very similar contexts, then skip-gram model needs to output similar results.
- Then the skip-gram network is motivated to learn similar word vectors (at hidden layer) for these similar words!

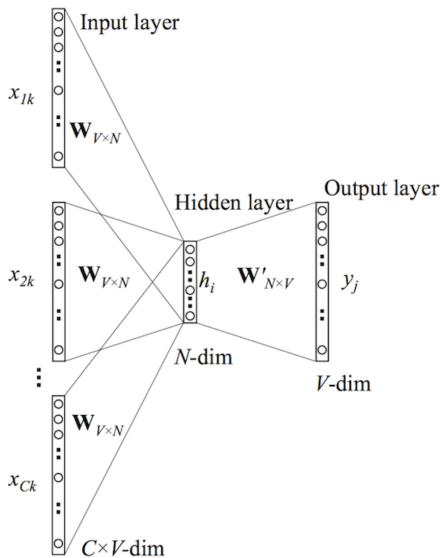


# Skip-gram model: vector space is semantic



# Word2vec: continuous bag of words (CBOW) model

- ‘Reverse’ of skip-gram
- $C$  context words as input
- Center word as output
- Hidden layer: average over  $C$  embeddings, hence ‘bag of words’
- Training: again with cross-entropy loss



# It only begins...

Word2vec is just for word representation.

How to capture meaning of sentence/paragraph?

# It only begins...

Word2vec is just for word representation.

How to capture meaning of sentence/paragraph?

We need consider order of words in text!

# Language models

Language models: assign a probability to a sequence of words, such that plausible sequences have higher probabilities, e.g.,

- $p(\text{'I like apples'}) > p(\text{'I sit apples'})$
- $p(\text{'I like apples'}) > p(\text{'like I apples'})$



# Language models

Language models: assign a probability to a sequence of words, such that plausible sequences have higher probabilities, e.g.,

- $p(\text{'I like apples'}) > p(\text{'I sit apples'})$
- $p(\text{'I like apples'}) > p(\text{'like I apples'})$

Auto-regressive modelling of sequence  $(\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_n)$ :

$$p(\mathbf{w}_0) \cdot p(\mathbf{w}_1 | \mathbf{w}_0) \dots p(\mathbf{w}_n | \mathbf{w}_{n-1}, \mathbf{w}_{n-2}, \dots, \mathbf{w}_0)$$

# Language models

Language models: assign a probability to a sequence of words, such that plausible sequences have higher probabilities, e.g.,

- $p(\text{'I like apples'}) > p(\text{'I sit apples'})$
- $p(\text{'I like apples'}) > p(\text{'like I apples'})$

Auto-regressive modelling of sequence  $(\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_n)$ :

$$p(\mathbf{w}_0) \cdot p(\mathbf{w}_1 | \mathbf{w}_0) \dots p(\mathbf{w}_n | \mathbf{w}_{n-1}, \mathbf{w}_{n-2}, \dots, \mathbf{w}_0)$$

- $p(\cdot)$  can be a neural network!
- $p(\cdot)$  could capture meaning of sequential information

# Conditional language models for NLP problems

Translation:  $p(\textit{Target}|\textit{Source})$

- Source (Chinese): 'wo xi huan ping guo'
- Target (English): 'I like apples'
- Model the output word by word:

$$p(\mathbf{w}_0|\textit{Source}) \cdot p(\mathbf{w}_1|\mathbf{w}_0, \textit{Source}) \dots$$

# Conditional language models for NLP problems

Translation:  $p(\textit{Target}|\textit{Source})$

- Source (Chinese): 'wo xi huan ping guo'
- Target (English): 'I like apples'
- Model the output word by word:

$$p(\mathbf{w}_0|\textit{Source}) \cdot p(\mathbf{w}_1|\mathbf{w}_0, \textit{Source}) \dots$$

Question answering / Dialogue:  $p(\textit{Answer}|\textit{Question}, \textit{Context})$

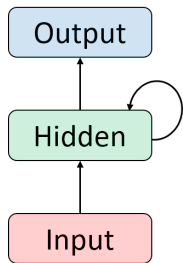
- Context:
  - 'John puts two apples on the table.'
  - 'Tom adds three more apples.'
  - 'Tom leaves to study in the library.'
- Question: 'How many apples are there?'
- Answer: 'There are five apples.'

# Then...

What neural networks can represent  $p(\cdot|\cdot)$ ?

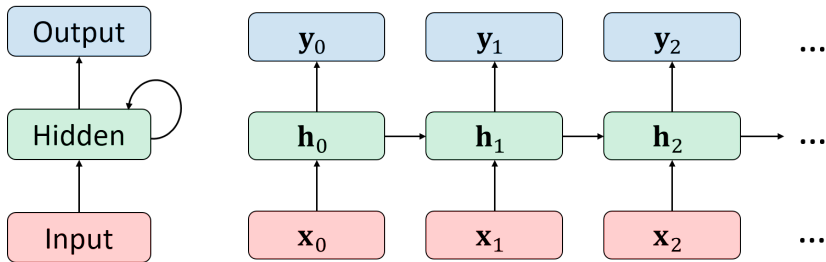
# Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.



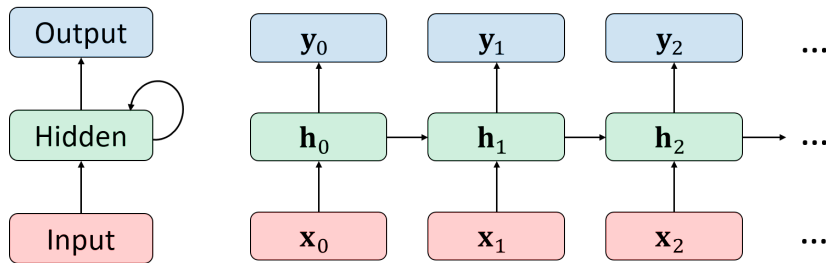
# Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.
- Unroll to process an input sequence  $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots)$



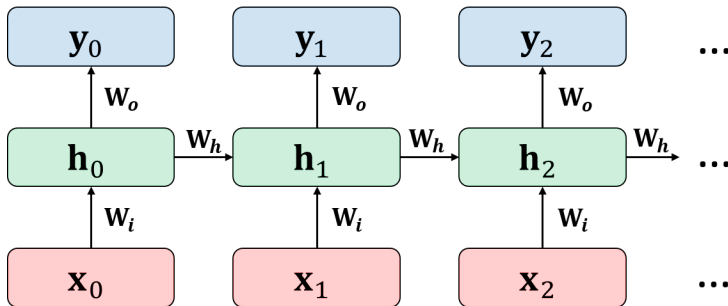
# Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.
- Unroll to process an input sequence  $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots)$
- RNN is a DEEP neural network model





# RNN basics



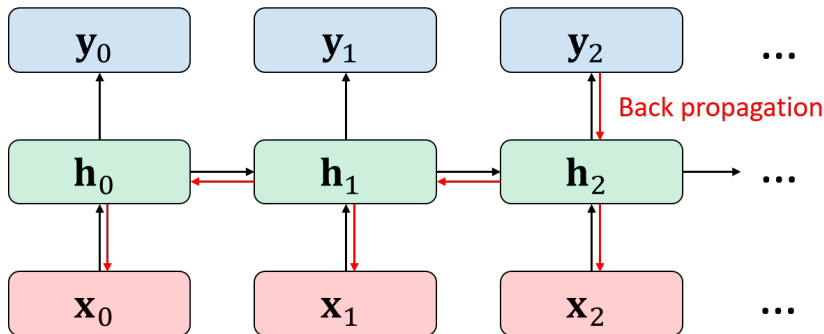
$$\mathbf{h}_t = g(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t + \mathbf{b}_h)$$

$$\mathbf{y}_t = \sigma(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o)$$

- $g(\cdot)$ : activation function, often  $\tanh$ ;  $\sigma(\cdot)$ : softmax function
- Same functions (model parameters) used at every time step!

# RNN training

- Multiply same matrix at each time step during forward prop
- Inputs from many time steps ago can affect output  $y_t$
- Multiply the same matrix at each time step during backprop



# RNN training: gradient exploding/vanishing

- Training RNN is hard
- Similar but simpler RNN formulation:

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i\mathbf{x}_t$$

$$\mathbf{y}_t = \sigma(\mathbf{W}_o\mathbf{h}_t)$$

# RNN training: gradient exploding/vanishing

- Training RNN is hard
- Similar but simpler RNN formulation:

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i\mathbf{x}_t \\ \mathbf{y}_t &= \sigma(\mathbf{W}_o\mathbf{h}_t)\end{aligned}$$

- Total loss is the sum of loss over all time steps, then

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{t=1}^T \frac{\partial L_t}{\partial \mathbf{W}}$$

# RNN training: gradient exploding/vanishing

- Training RNN is hard
- Similar but simpler RNN formulation:

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i\mathbf{x}_t \\ \mathbf{y}_t &= \sigma(\mathbf{W}_o\mathbf{h}_t)\end{aligned}$$

- Total loss is the sum of loss over all time steps, then

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{t=1}^T \frac{\partial L_t}{\partial \mathbf{W}}$$

- With chain rule:

$$\frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

# RNN training: gradient exploding/vanishing

- So far:

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i\mathbf{x}_t$$

$$\frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

# RNN training: gradient exploding/vanishing

- So far:

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i\mathbf{x}_t$$

$$\frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

- With chain rule again

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{j=k+1}^t \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} = \prod_{j=k+1}^t \mathbf{W}^T \text{diag} [g'(\mathbf{h}_{j-1})]$$

# RNN training: gradient exploding/vanishing

- So far:

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i\mathbf{x}_t$$

$$\frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

- With chain rule again

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{j=k+1}^t \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} = \prod_{j=k+1}^t \mathbf{W}^T \text{diag} [g'(\mathbf{h}_{j-1})]$$

$$\text{If } \left\| \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| \leq \|\mathbf{W}^T\| \|\text{diag} [g'(\mathbf{h}_{j-1})]\| \leq \beta_W \beta_h$$



# RNN training: gradient exploding/vanishing

- So far:

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i\mathbf{x}_t$$

$$\frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

- With chain rule again

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{j=k+1}^t \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} = \prod_{j=k+1}^t \mathbf{W}^T \text{diag} [g'(\mathbf{h}_{j-1})]$$

$$\text{If } \left\| \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| \leq \|\mathbf{W}^T\| \|\text{diag} [g'(\mathbf{h}_{j-1})]\| \leq \beta_W \beta_h$$

$$\text{Then } \left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| = \prod_{j=k+1}^t \left\| \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

# RNN training: gradient exploding/vanishing

- When  $\beta_W \beta_h > 1$ ,

$$\left\| \frac{\partial L_t}{\partial \mathbf{W}} \right\| \gg 1$$

# RNN training: gradient exploding/vanishing

- When  $\beta_W \beta_h > 1$ ,

$$\left\| \frac{\partial L_t}{\partial \mathbf{W}} \right\| \gg 1$$

causing gradient exploding!

# RNN training: gradient exploding/vanishing

- When  $\beta_W \beta_h > 1$ ,

$$\left\| \frac{\partial L_t}{\partial \mathbf{W}} \right\| \gg 1$$

causing gradient exploding!

- Trick: gradient clipping

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

- Gradient clipping well solved gradient exploding!

# RNN training: vanishing gradient is a problem

- When  $\beta_W \beta_h < 1$ , vanishing  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$  and

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i \mathbf{x}_t$$

$$\frac{\partial L_t}{\partial \mathbf{W}_i} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \text{diag}[\mathbf{x}_k]$$

would cause  $\mathbf{x}_k$  from previous time step  $k$  not to affect update of  $\mathbf{W}_i$  at time step  $t$ .

# RNN training: vanishing gradient is a problem

- When  $\beta_W \beta_h < 1$ , vanishing  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$  and

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i \mathbf{x}_t$$

$$\frac{\partial L_t}{\partial \mathbf{W}_i} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \text{diag}[\mathbf{x}_k]$$

would cause  $\mathbf{x}_k$  from previous time step  $k$  not to affect update of  $\mathbf{W}_i$  at time step  $t$ .

- In other words, prediction error at time step  $t$  would not tell a far-away previous step  $k$  to change during backprop.

# RNN training: vanishing gradient is a problem

- When  $\beta_W \beta_h < 1$ , vanishing  $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$  and

$$\mathbf{h}_t = \mathbf{W}g(\mathbf{h}_{t-1}) + \mathbf{W}_i \mathbf{x}_t$$

$$\frac{\partial L_t}{\partial \mathbf{W}_i} = \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \text{diag}[\mathbf{x}_k]$$

would cause  $\mathbf{x}_k$  from previous time step  $k$  not to affect update of  $\mathbf{W}_i$  at time step  $t$ .

- In other words, prediction error at time step  $t$  would not tell a far-away previous step  $k$  to change during backprop.
- Vanishing gradient makes RNN unable to capture long-term relationship between items far away from each other!

# Long short-term memory (LSTM)

- LSTM as basic unit of RNN reduces gradient vanishing



# Long short-term memory (LSTM)

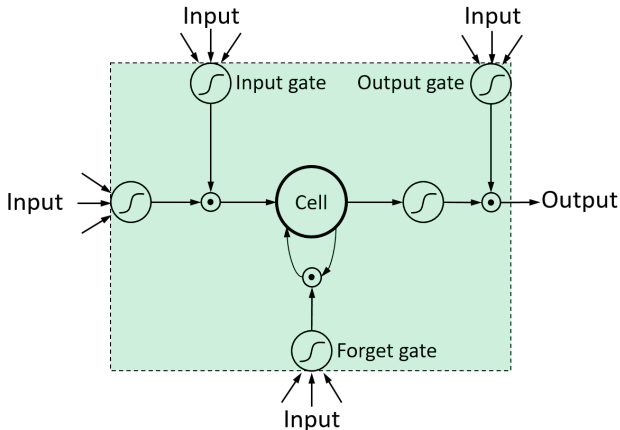
- LSTM as basic unit of RNN reduces gradient vanishing
- 'short-term memory': a small amount of information
- 'long': information can last for a long period of time

# Long short-term memory (LSTM)

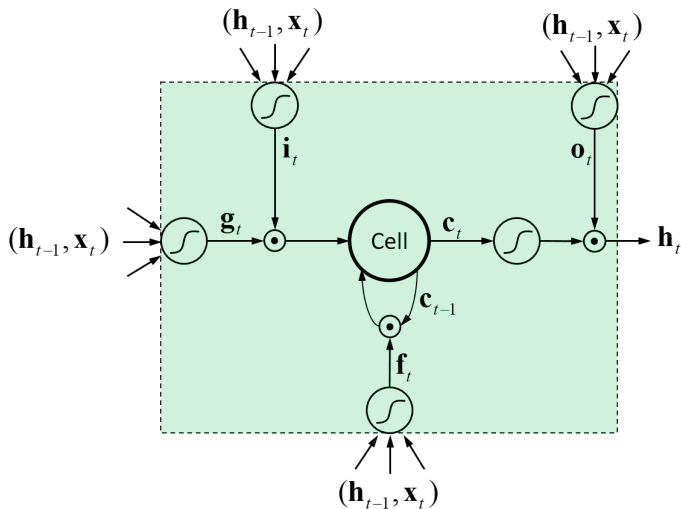
- LSTM as basic unit of RNN reduces gradient vanishing
- 'short-term memory': a small amount of information
- 'long': information can last for a long period of time
- LSTM: cell, input gate, output gate, (un)forget gate
- Cell for 'remembering' values over arbitrary time steps, hence the word 'memory' in LSTM
- Gates as regulators of the flow of signals through LSTM

# LSTM

- Input gate: whether/how much to write to cell
- Output gate: how much to reveal cell
- Forget gate: whether/how much to erase cell



## LSTM



## LSTM

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

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

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

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

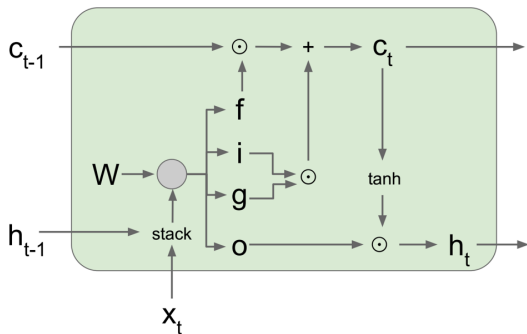
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$

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

- $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ : input, forget, and output gate;  $\sigma$ : sigmoid function
- $\mathbf{g}_t$ : new signal to update cell
- $\mathbf{c}_t$ : updated cell;  $\mathbf{h}_t$ : new hidden state
- Well chosen activation function (tanh) is critical
- Three times more parameters than RNN

## LSTM

- An alternative diagram representation of LSTM



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

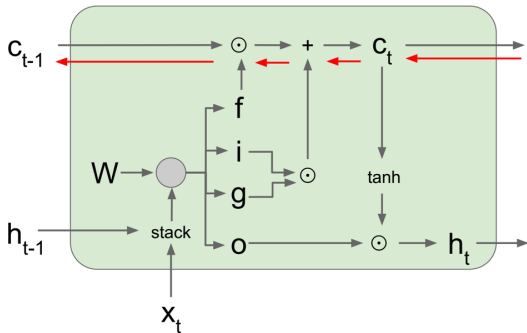
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Figures and content here and in the next 9 slide mainly from Stanford CS231n Lecture 10, 2017

# Why LSTM can reduce gradient vanishing

- Additive path between  $c_t$  and  $c_{t-1}$



Backpropagation from  $c_t$  to  $c_{t-1}$  only elementwise multiplication by  $f$ , no matrix multiply by  $W$

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

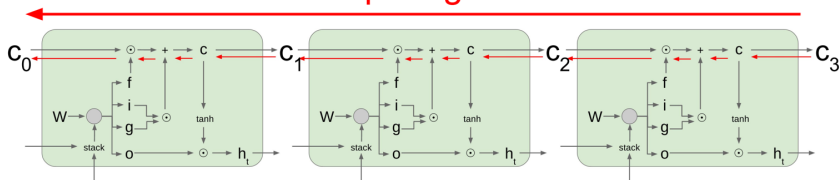
$$c_t = f \odot c_{t-1} + i \odot g$$

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

# Why LSTM can reduce gradient vanishing

- Gradient signal can easily back propagate through multiple time steps (if forget gate is open)

Uninterrupted gradient flow!

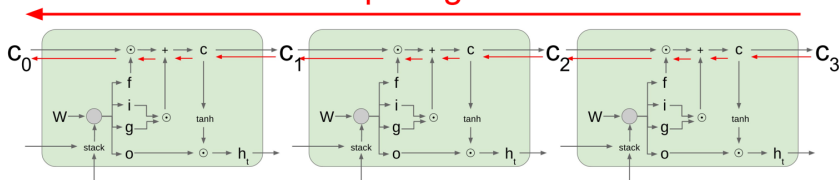




# Why LSTM can reduce gradient vanishing

- Gradient signal can easily back propagate through multiple time steps (if forget gate is open)
- Reminder: skip connections in ResNet

Uninterrupted gradient flow!



# Gated recurrent unit (GRU)

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

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

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

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \hat{\mathbf{h}}_t$$

- One gate less than LSTM, so fewer parameters
- No 'cell', only hidden vector  $\mathbf{h}_t$  passed to next unit
- No systematic difference between GRU and LSTM
- People tend to use LSTM more

# Vanilla RNN for language modeling

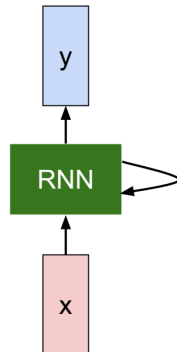
- Predict next character based on previous characters

## THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,  
 That thereby beauty's rose might never die,  
 But as the ripener should by time decease,  
 His tender heir might bear his memory;  
 But thou, contracted to thine own bright eyes,  
 Feed'st thy light's flame with self-substantial fuel,  
 Making a famine where abundance lies,  
 Thyself thy foe, to thy sweet self too cruel:  
 Thou that art now the world's fresh ornament,  
 And only herald to the gaudy spring,  
 Within thine own buduriest thy content,  
 And tender churl mak'st waste in niggarding:  
 Pity the world, or else this glutton be,  
 To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,  
 And dig deep trenches in thy beauty's field,  
 Thy youth's proud livery so gazed on now,  
 Will be a tatter'd weed of small worth held:  
 Then being asked, where all thy beauty lies,  
 Where all the treasure of thy lusty days;  
 To say, within thine own deep sunken eyes,  
 Were an all-eating shame, and thriftless praise.  
 How much more praise deserv'd thy beauty's use,  
 If thou couldst answer 'This fair child of mine  
 Shall sum my count, and make my old excuse,'  
 Proving his beauty by succession thine!  
 This were to be new made when thou art old,  
 And see thy blood warm when thou feel'st it cold.



# Language modeling

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

train more

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y 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 beloge, pavu say falling misfort  
how, and Gogition is so overelical and offer.

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.

# Language modeling

- With latex source code: predict next character based on previous characters

The Stacks Project: open source algebraic geometry textbook

The Stacks Project

[home](#) [about](#) [tags explained](#) [tag lookup](#) [browse](#) [search](#) [bibliography](#) [recent comments](#) [blog](#) [add slogans](#)

**Browse chapters**

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	2. Conventions	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	3. Set Theory	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	4. Categories	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	5. Topology	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	6. Sheaves on Spaces	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	7. Sites and Sheaves	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	8. Stacks	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	9. Fields	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >
	10. Commutative Algebra	<a href="#">online</a>	<a href="#">tex</a> ↻	<a href="#">pdf</a> >

**Parts**

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

**Statistics**

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

<http://stacks.math.columbia.edu/>

The stacks project is licensed under the [GNU Free Documentation License](#)

# Language modeling

- After training a RNN, generate latex doc, then render it

For  $\bigoplus_{i=1, \dots, m}$  where  $\mathcal{L}_{m_*} = 0$ , hence we can find a closed subset  $\mathcal{H}$  in  $\mathcal{H}$  and any sets  $\mathcal{F}$  on  $X$ ,  $U$  is a closed immersion of  $S$ , then  $U \rightarrow T$  is a separated algebraic space.

*Proof.* Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparably in the fibre product covering we have to prove the lemma generated by  $\coprod Z \times_U U \rightarrow V$ . Consider the maps  $M$  along the set of points  $\text{Sch}_{\text{fppf}}$  and  $U \rightarrow U$  is the fibre category of  $S$  in  $U$  in Section, ?? and the fact that any  $U$  affine, see Morphisms, Lemma ?? . Hence we obtain a scheme  $S$  and any open subset  $W \subset U$  in  $\text{Sh}(G)$  such that  $\text{Spec}(R') \rightarrow S$  is smooth or an

$$U = \bigcup U_i \times_S U_i$$

which has a nonzero morphism we may assume that  $f_i$  is of finite presentation over  $S$ . We claim that  $\mathcal{O}_{X,x}$  is a scheme where  $x, x', s' \in S'$  such that  $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}_{X',x'}$  is separated. By Algebra, Lemma ?? we can define a map of complexes  $\text{GL}_{S'}(x'/S'')$  and we win.  $\square$

To prove study we see that  $\mathcal{F}|_U$  is a covering of  $\mathcal{X}'$ , and  $\mathcal{T}_i$  is an object of  $\mathcal{F}_{X|S}$  for  $i > 0$  and  $\mathcal{F}_n$  exists and let  $\mathcal{F}_i$  be a presheaf of  $\mathcal{O}_X$ -modules on  $\mathcal{C}$  as a  $\mathcal{F}$ -module. In particular  $\mathcal{F} = U/\mathcal{F}$  we have to show that

$$\bar{M}^* = \mathcal{I}^* \otimes_{\text{Spec}(k)} \mathcal{O}_{S_n} - i_n^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)_{\text{fppf}}^{\text{opp}}, (\text{Sch}/S)_{\text{fppf}}$$

and

$$V = \Gamma(S, \mathcal{O}) \rightarrow (U, \text{Spec}(A))$$

is an open subset of  $X$ . Thus  $U$  is affine. This is a continuous map of  $X$  is the inverse, the groupoid scheme  $S$ .

*Proof.* See discussion of sheaves of sets.  $\square$

The result to prove any open covering follows from the less of Example ?? . It may replace  $S$  by  $X_{\text{spaces, étale}}$  which gives an open subspace of  $X$  and  $T$  equal to  $S_{Zar}$ , see Descent, Lemma ?? . Namely, by Lemma ?? we see that  $R$  is geometrically regular over  $S$ .

**Lemma 0.1.** Assume (3) and (3) by the construction in the description.

Suppose  $X = \lim |X|$  (by the formal open covering  $X$  and a single map  $\text{Proj}_X(A) = \text{Spec}(B)$  over  $U$  compatible with the complex

$$\text{Set}(A) = \Gamma(X, \mathcal{O}_{X, \mathcal{O}_X}).$$

When in this case of to show that  $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$  is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If  $T$  is surjective we may assume that  $T$  is connected with residue fields of  $S$ . Moreover there exists a closed subspace  $Z \subset X$  of  $X$  where  $U$  in  $X'$  is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1)  $f$  is locally of finite type. Since  $S = \text{Spec}(R)$  and  $Y = \text{Spec}(R)$ .

*Proof.* This is form all sheaves of sheaves on  $X$ . But given a scheme  $U$  and a surjective étale morphism  $U \rightarrow X$ . Let  $U \cap U = \coprod_{i=1, \dots, n} U_i$  be the scheme  $X$  over  $S$  at the schemes  $X_i \rightarrow X$  and  $U = \lim_i X_i$ .  $\square$

The following lemma surjective retrocomposes of this implies that  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{x_{\dots, 0}}$ .

**Lemma 0.2.** Let  $X$  be a locally Noetherian scheme over  $S$ ,  $E = \mathcal{F}_{X/S}$ . Set  $I = \mathcal{J}_1 \subset \mathcal{I}_n$ . Since  $\mathcal{I}^n \subset \mathcal{I}^n$  are nonzero over  $i_0 \leq \mathfrak{p}$  is a subset of  $\mathcal{J}_{n,0} \circ \bar{A}_2$  works.

**Lemma 0.3.** In Situation ?? . Hence we may assume  $\mathfrak{q}' = 0$ .

*Proof.* We will use the property we see that  $\mathfrak{p}$  is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_{X'}(D)$$

where  $K$  is an  $F$ -algebra where  $\delta_{n+1}$  is a scheme over  $S$ .  $\square$

# Language modeling

- With C source code: predict next character based on previous characters

The screenshot shows the GitHub repository for the Linux kernel source code, owned by torvalds. The repository has 520,037 commits, 1 branch, 420 releases, and 5,039 contributors. The current branch is 'master'. The repository is watched by 3,711 users, has 23,054 stars, and 9,141 forks. The main content area shows a list of recent commits, including a merge of 'drm-fixes' and updates to 'arch', 'block', 'crypto', 'drivers', 'firmware', 'fs', 'include', and 'init'. The right sidebar shows options to view code, pull requests (74), and pulse, along with HTTPS clone URL and options to clone on desktop or download ZIP.

# Language modeling

```

static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}

```

## Generated C code

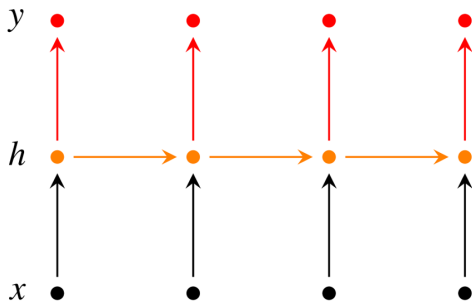


# RNN variants in structure...

RNN can have more complex structures!

# Vanilla RNN

- People may use different notations for RNN.
- $h_t$  summarizes the sentence up to time step  $t$ .
- Problem: for some tasks, it would be better to incorporate information from both preceding and following words.

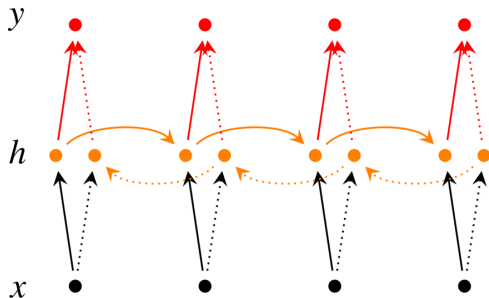


$$h_t = f(Wx_t + Vh_{t-1} + b)$$

$$y_t = g(Uh_t + c)$$

# Bidirectional RNN (BiRNN)

- Bidirectional RNN captures sequential information from both directions.
- RNN unit could be LSTM or others



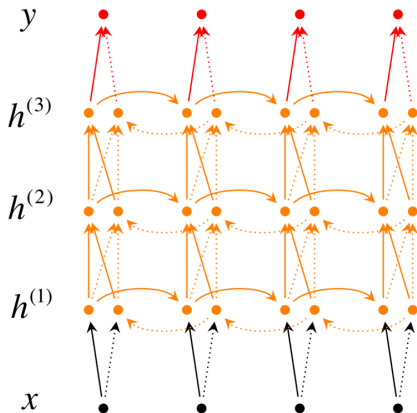
$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

# Deep bidirectional RNN

- Deep BiRNN: each layer passes an intermediate sequential representation to the next layer.



$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

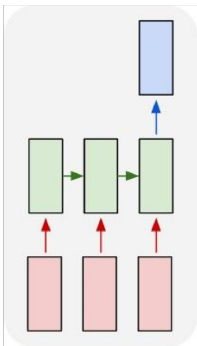
$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

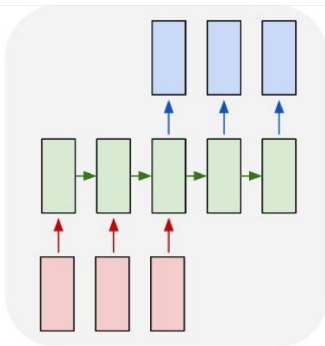
# RNN outputs

- Left: e.g., sequence of words  $\rightarrow$  sentiment
- Centre: e.g., machine translation
- Right: e.g., video classification for each frame

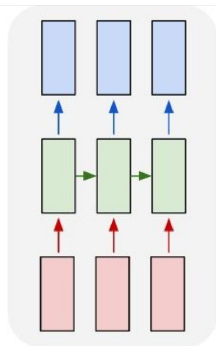
## Many to one



## Many to many



## Many to many



# Summary

- Word2vec as input to RNN models
- Gradient clipping to reduce exploding issue
- Vanilla RNN may not well capture long-term relationships
- LSTM can capture long-term relationships
- LSTM can reduce gradient vanishing issue
- Different RNN structures/outputs for different apps

## Further reading:

- Mikolov, Sutskever, Chen, Corrado, Dean, 'Distributed representations of words and phrases and their compositionality', NIPS, 2013
- Hochreiter, Sepp, Schmidhuber, 'Long short-term memory', Neural computation, 1997