Natural Language Processing with Deep Learning

Hung-Yu Kao (高宏宇)
***I**ntelligent **K**nowledge **M**anagement Lab*

Institute of Medical Informatics,
Dept. of Computer Science and Information Engineering
National Cheng Kung University, Tainan, Taiwan

Outlines

- NLP concepts and issues
- Word Representation
- Sentence Representation
- Applications
  - Named Entity Recognition
  - Rumor Detection
  - Chatbot Training by GAN

Learning Vector Representations of Sentences

Overview

We have word vectors and we want a vector representation of a sentence*

We somehow need to compose (or encode) those words into a single vector

$$f\left(\begin{bmatrix} \cdots & \text{word}_1 & \cdots \\ & \vdots & \\ \cdots & \text{word}_T & \cdots \end{bmatrix}\right) = \overrightarrow{\text{sentence}}$$

We will consider different functions for this:

- Bag of Words
- Auto-Encoder
- Recurrent Neural Network (and variants)
- Recursive Neural Network

* It is possible to use other representations of sentences, including matrices. We only focus on vectors here.

Ideas for modeling sentence vectors / representation

1. Bag-of-Words
2. Recurrent Neural Networks (RNN)
3. Long Short-Term Memory (LSTM)
4. Gated Recurrent Units (GRU)
5. Recursive Tree-LSTMs
6. Seq2seq
7. Example Application of Sentence Vectors

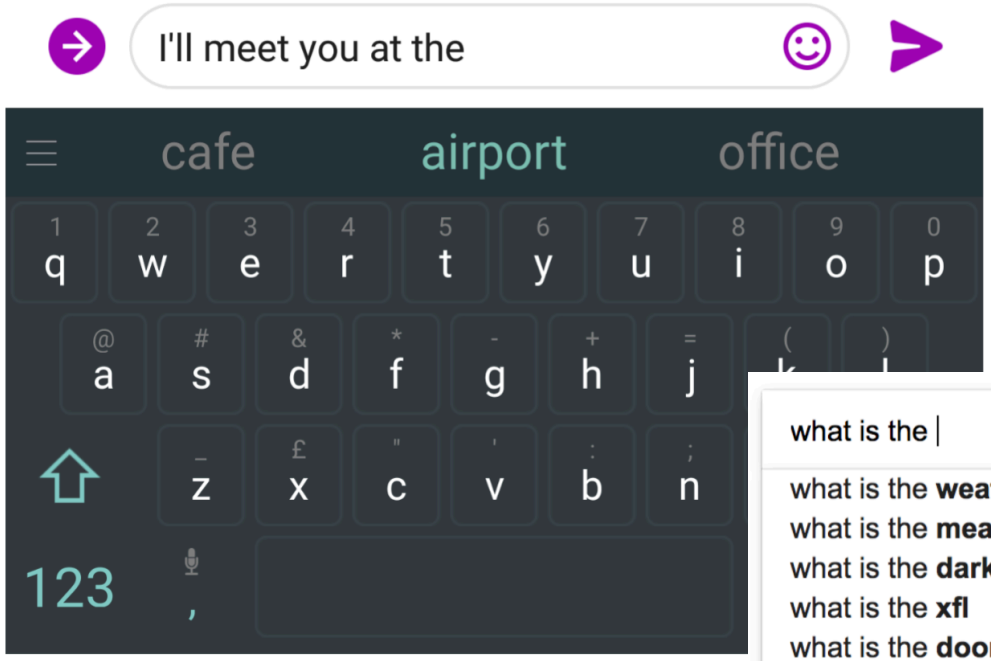
Bag of Words

We sum the word vectors

$$\overrightarrow{\text{the cat jumped}} = \begin{bmatrix} \vdots \\ \text{the} \\ \vdots \end{bmatrix} + \begin{bmatrix} \vdots \\ \text{cat} \\ \vdots \end{bmatrix} + \begin{bmatrix} \vdots \\ \text{jumped} \\ \vdots \end{bmatrix}$$

But no consideration of word order or local compositionality

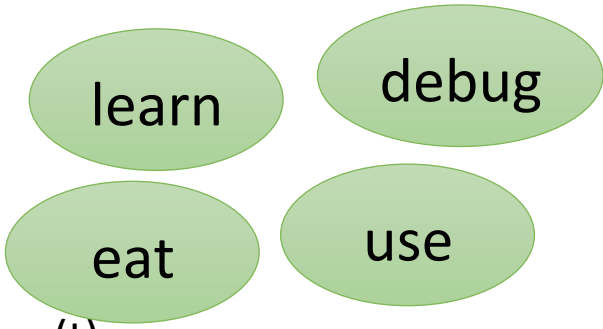
Then, let's see Language Model First



Language Model

- **Language Modeling** is the task of predicting what word comes next.

the python is good to →



- given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$

$$p(x^{(t+1)} = w_j \mid x^{(t)}, \ldots, x^{(1)})$$

- A system that does this is called a **Language Model**.

N-gram language model

- First we make a simplifying assumption $x^{(t+1)}$ depends only on the preceding *(n-1)* words

$$p(x^{(t+1)} = w_j \mid x^{(t)},\ldots,x^{(1)}) = p(x^{(t+1)} = w_j \mid \underbrace{x^{(t)},\ldots,x^{(t-n+2)}}_{\text{n-1 words}})$$

$$= \frac{p(x^{(t+1)},x^{(t)},\ldots,x^{(t-n+2)})}{p(x^{(t)},\ldots,x^{(t-n+2)})}$$

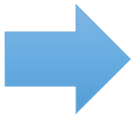
Prob of a n-gram

Prob of a (n-1)-gram

By counting them in some large corpus of text!

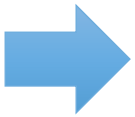
Issues of n-gram language models

If "the python is good to learn" never occurred in data? Then "learn" has probability 0



Add small count for each word

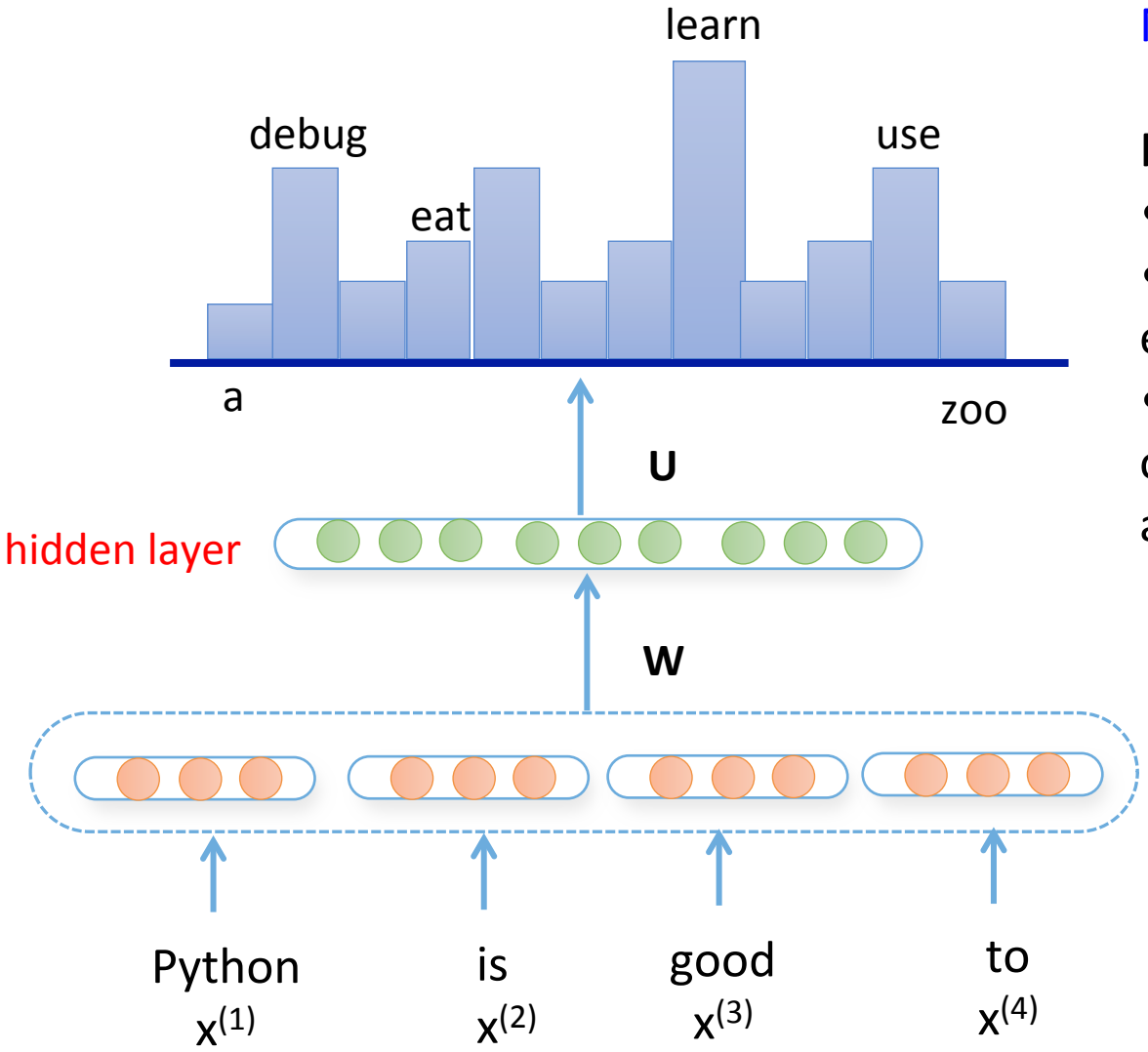
If "the python is good to" never occurred in data? Then we can't calculate probability for any word



Backoff: Condition on "python is good to" instead

Trade-off: Increasing n makes sparsity problems worse

A fixed-window neural Language Model



No sparsity problem!

Remaining **problems**:

- Enlarging window enlarges **W**
- Window can never be large enough!
- Each $x^{(i)}$ uses different rows of **W**. We don't share weights across the window.

We need a neural architecture to process any length input.

Motivation for Recurrent Neural Networks

Are these the same pattern?

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

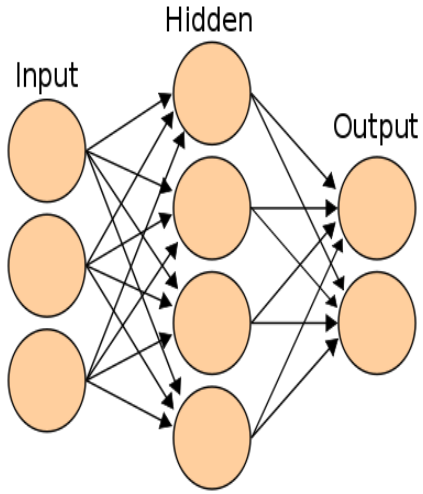
$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

There are two equally valid perspectives on these patterns:

- Geometric (considers absolute position): not the same (2, 3, 4 versus 3, 4, 5)
- Sequential (considers relative position): the same (three in a row)

Motivation for Recurrent Neural Networks

Multilayer perceptrons take a geometric perspective on the input data. They therefore require much more supervision to generalize on temporal patterns.

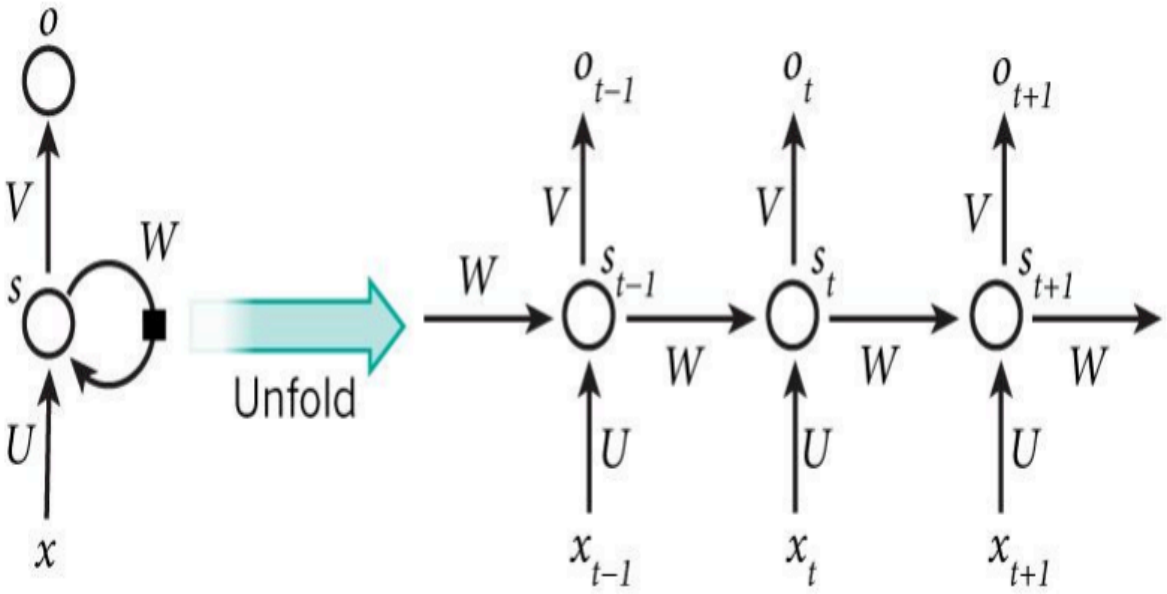


They also only deal with inputs of a fixed size (whereas sentence lengths vary)

Elman, J, (1990), "Finding Structure in Time", *Cognitive Science*, 14, 179-211.

Recurrent Neural Networks

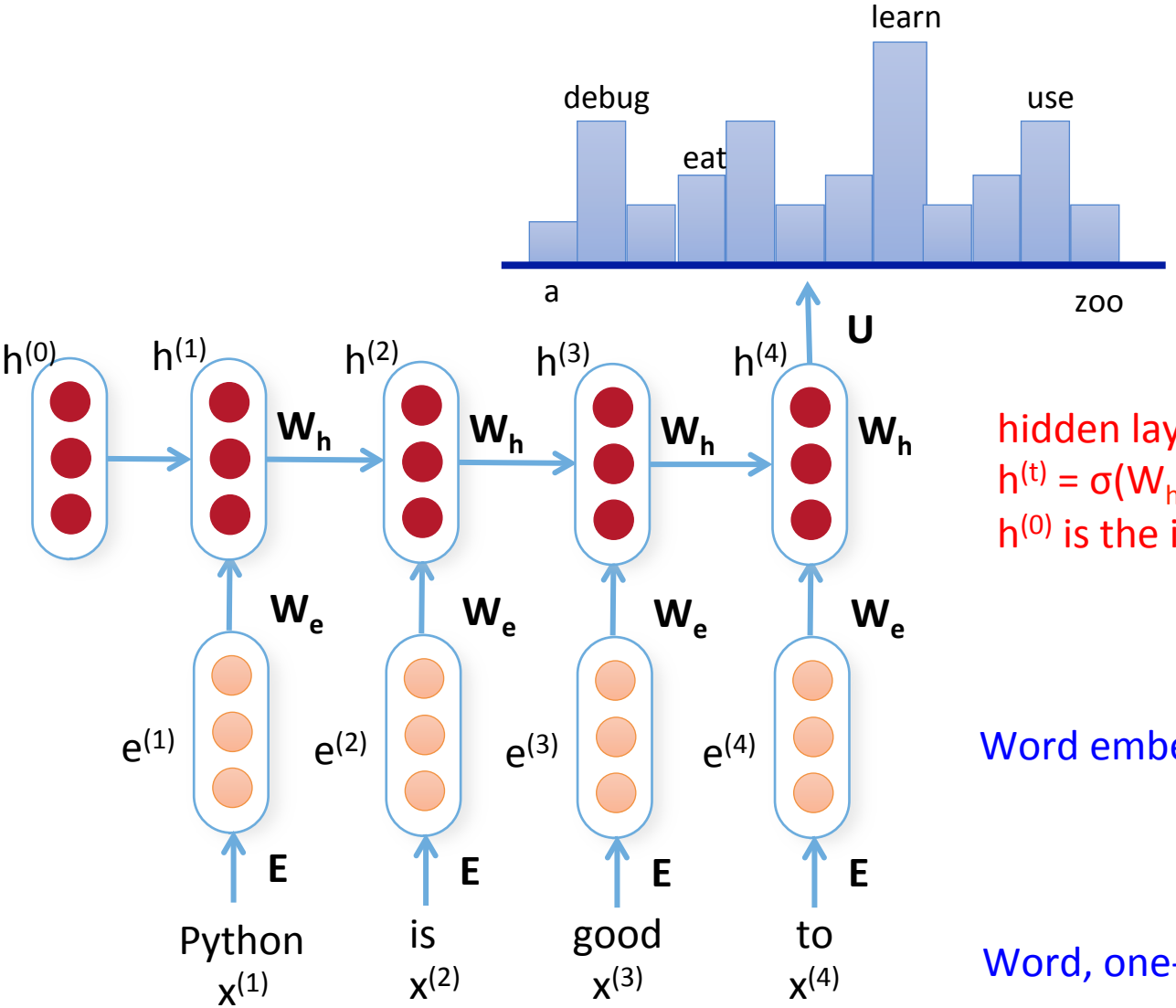
Receiving information from the previous state facilitates learning sequential patterns, allowing the network to consider the input sequentially



$$\mathbf{s}_t = \sigma\left(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1} + \mathbf{b}\right)$$

$$\mathbf{o}_t = \sigma\left(\mathbf{V}\mathbf{s}_t\right)$$

A RNN Language Model



Output distribution

$y^{(t)} = \text{softmax}(Uh^{(t)} + b_2)$

hidden layer

$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$

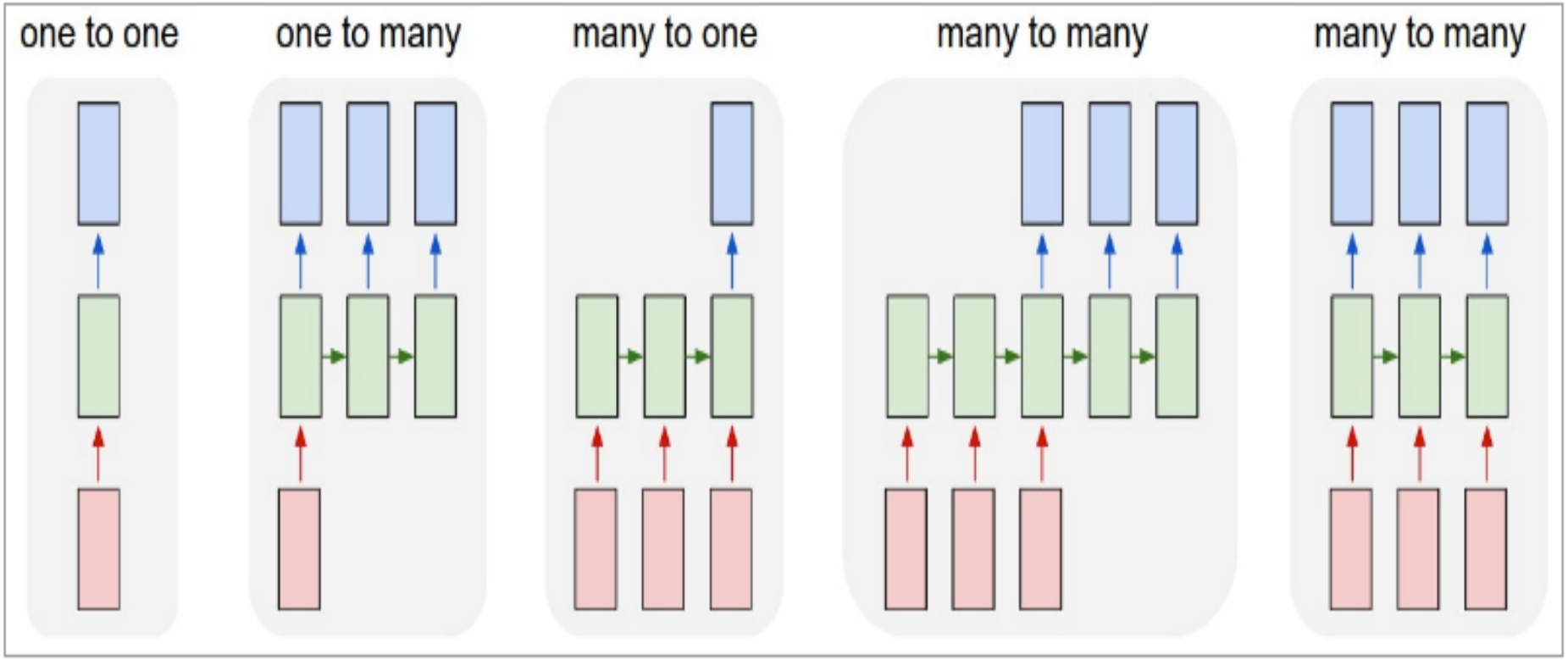
$h^{(0)}$ is the initial hidden state

Word embeddings, $e^{(t)} = Ex^{(t)}$

Word, one-hot vector

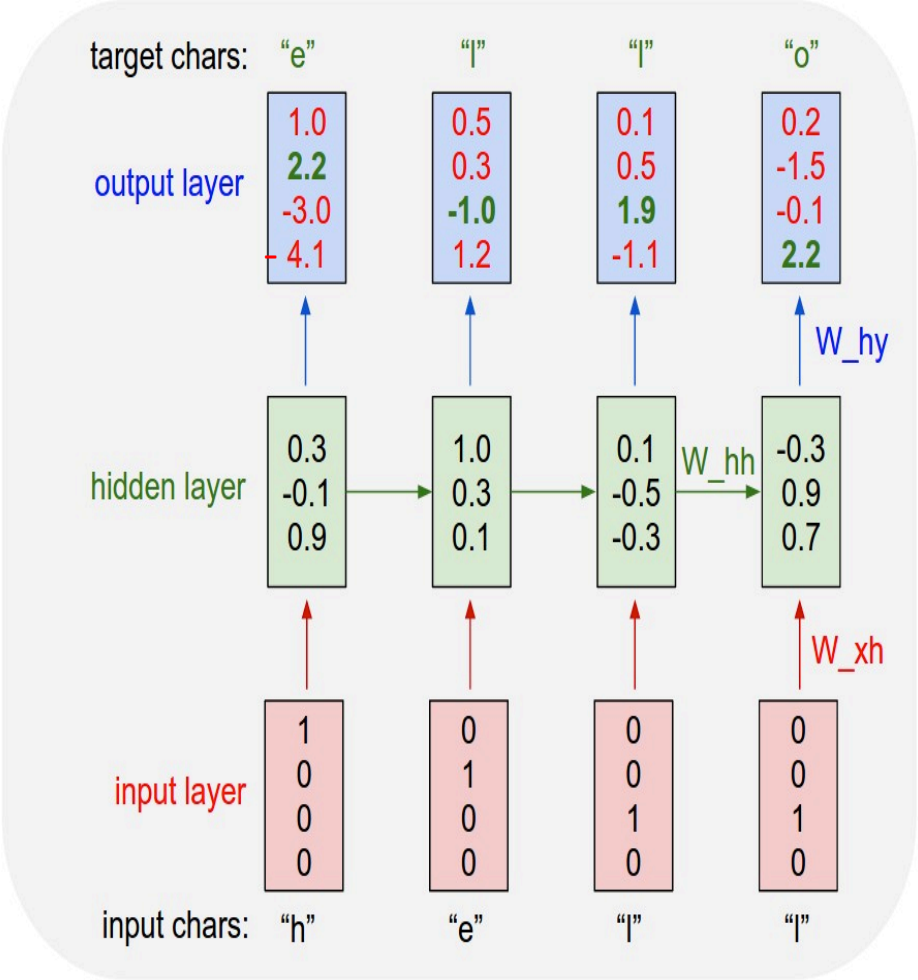
Image from Andreij Karpathy's blog post "The Unreasonable Effectiveness of Recurrent Neural Networks": http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Varying Input and Output Lengths



one to one	one to many	many to one	many to many	many to many
Fixed input Fixed output	Fixed input Sequential output	Sequential input Fixed output	Sequential input Sequential output	Synchronized sequential input and output
Image classification (without RNN)	Image captioning	Sentiment analysis	Machine translation	Frame-by-frame picture classification in video

Application: Character-Level Text Generation



* Picture from Andrej Karpathy's "The Unreasonable Effectiveness of Recurrent Neural Networks": http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Evolution of Samples While Training*

These examples taken from the same model trained on a novel.

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

Early training: starting to get an idea of words and spaces.

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

Now it has learned to use quotes and periods.

* Andreij Karpathy's blog post "The Unreasonable Effectiveness of Recurrent Neural Networks": http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Training Time… Evolution of Samples While Training*

```
we counter. He stutn co des. His stanted out one ofler that concossions and was
to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn
```

Now we have some real words forming...

```
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 ofter.
```

Improving further...

```
"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.
```

* Andreij Kaparsky's blog post "The Unreasonable Effectiveness of Recurrent Neural Networks": http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Poem Generator

Amazon
Rekognition API

Word2Vec,
Char-RNN,
LSTM

PTT Corpus

難受的男人
如果有一天
我成名的話
他就可以經常看到我的名字即使是十年或二十年後
他也不會忘記我如果我沒有成名
他也許會把我忘掉唯一可以強橫地霸佔一個男人的記憶的
就是活得更好

Implementation Practice

Available at https://github.com/IKMLab/rnn_tutorial

Problem: Long-Term Dependencies

What if we want to learn patterns that are not adjacent? That maybe span long distances?

"My friend from Germany has spent countless hours trying to teach me to speak [what?]"

Here a language model needs to connect Germany at the 4th timestep to the 15th timestep in order to predict the right word

From the RNN update equations can you see an intuitive reason why this might be hard?

$$\mathbf{s}_t = \sigma\left(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1} + \mathbf{b}\right)$$

Problem: Long-Term Dependencies

What if we want to learn patterns that are not adjacent? That maybe span long distances?

"My friend from Germany has spent countless hours trying to teach me to speak [what?]"

Here a language model needs to connect Germany at the 4th timestep to the 15th timestep in order to predict the right word

From the RNN update equations can you see an intuitive reason why this might be hard?

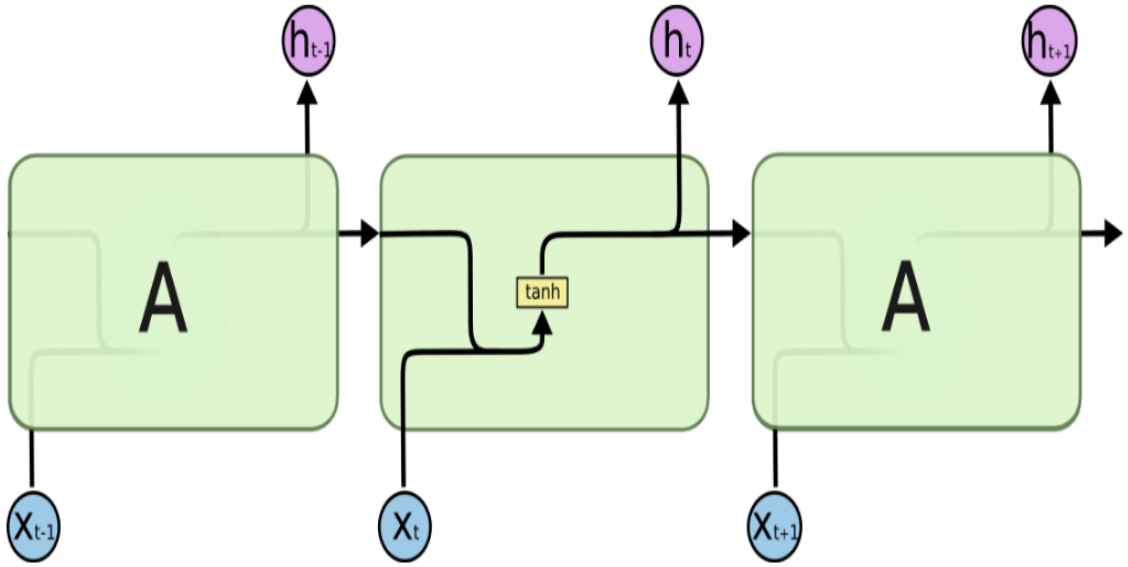
$$\mathbf{s}_t = \sigma\left(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1} + \mathbf{b}\right)$$

The memory gets a full update at each timestep - can overwrite important information over time

Long Short-Term Memory (LSTM)

Instead of a full update at each step, introduces "gating" to decide how much to update the memory, and how much of the previous memory to forget

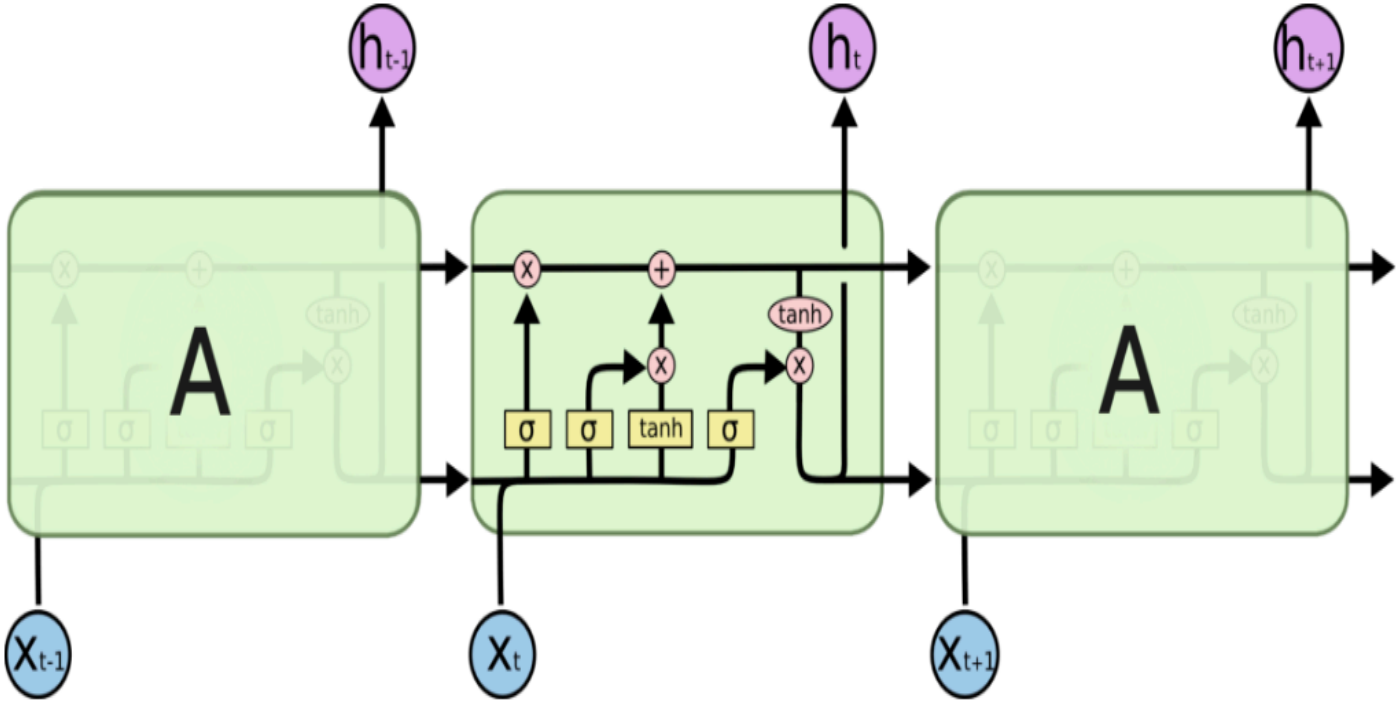
A simple RNN just has one layer (tanh or sigmoid)



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

Long Short-Term Memory (LSTM)

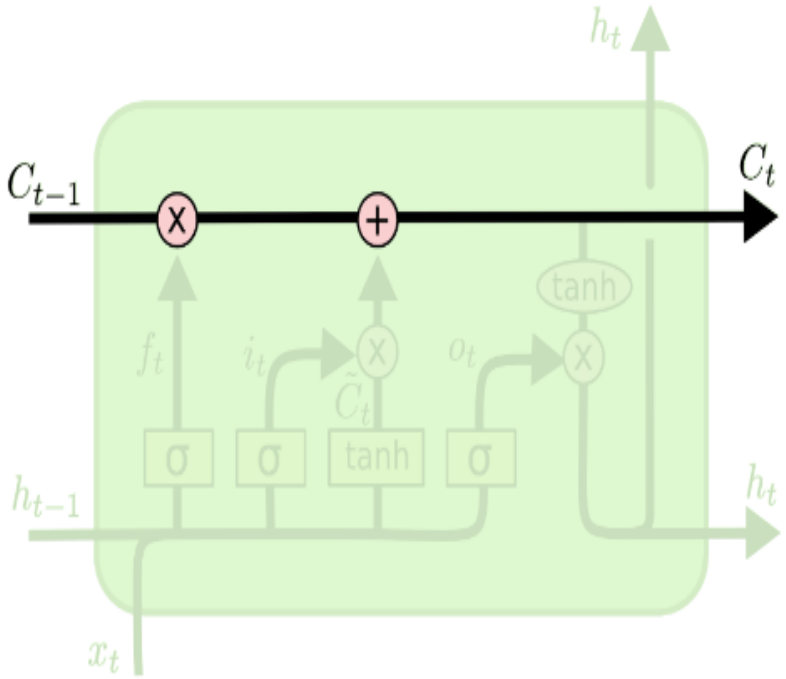
LSTMs instead have four “layers” working together to take care of selective forgetting and remembering. We will look at them in order.



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

Long Short-Term Memory (LSTM)

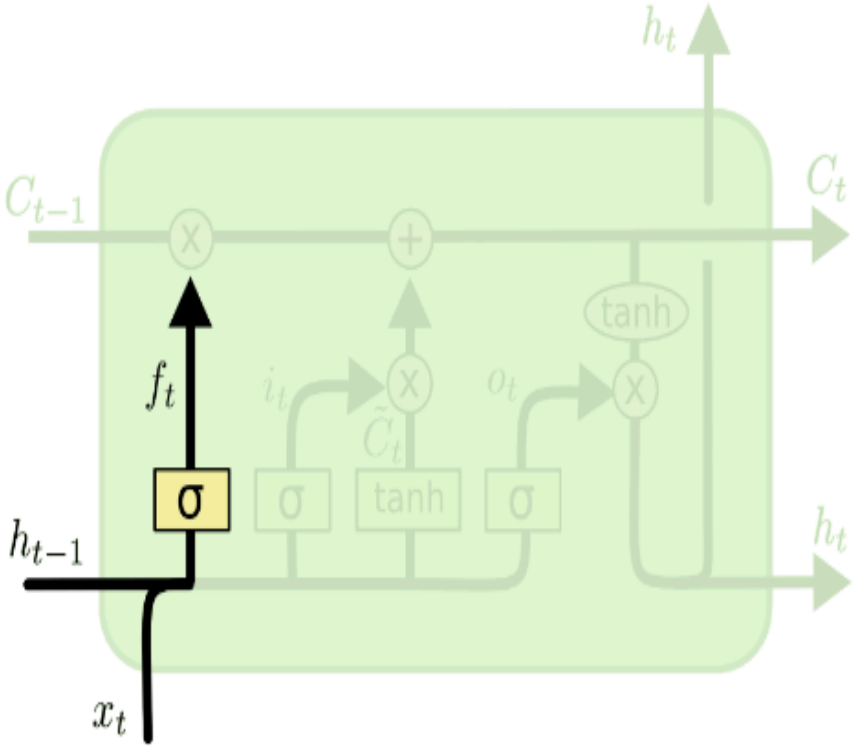
LSTMs use a cell state to keep track of information. It acts like a conveyor belt that carries information forward through time.



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

Long Short-Term Memory (LSTM)

The first step is to decide how much of the cell state to forget via a “forget gate”



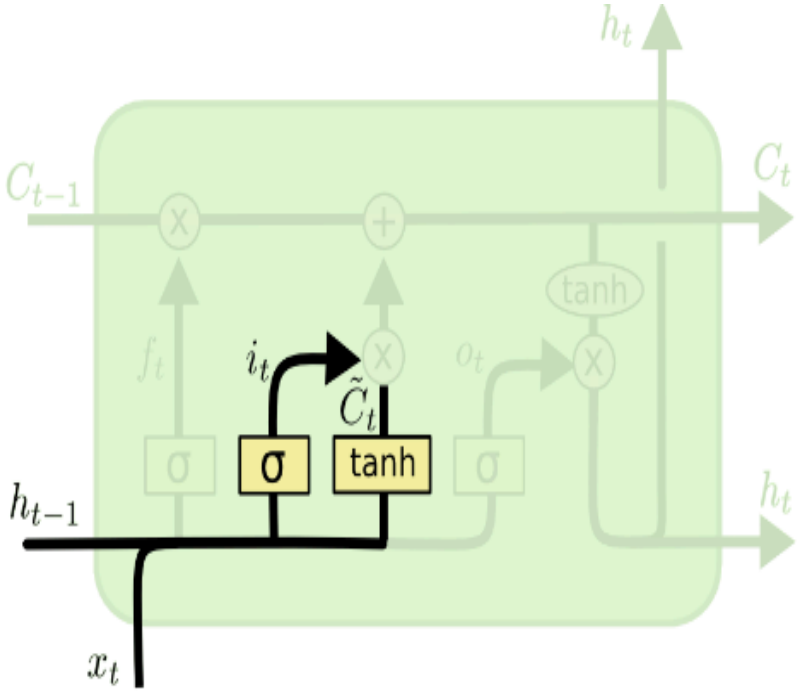
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Sigmoid returns values in [0, 1]
For each unit in the cell state we determine how much information to forget
0 means “leave” it, 1 means “forget it”

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

Long Short-Term Memory (LSTM)

Then we decide how to update the cell: calculate a "candidate update" and use an "input gate" to decide how much of it to add



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

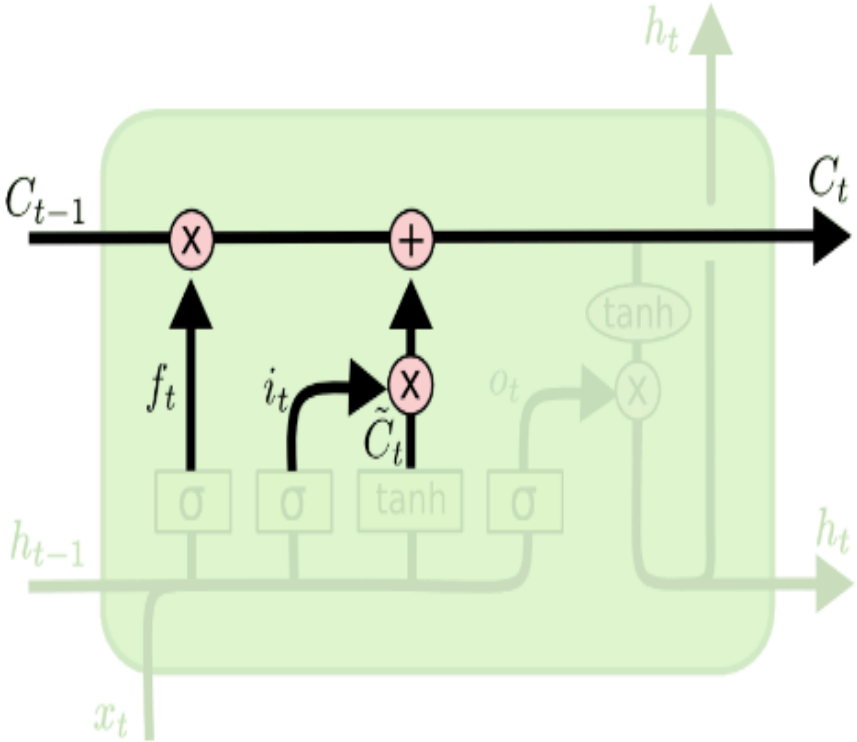
The input gate is another sigmoid and works similar to the forget gate

The candidate update is just like the vanilla RNN update

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

Long Short-Term Memory (LSTM)

Now we have our forget, candidate update, and input vectors, we can update the cell state

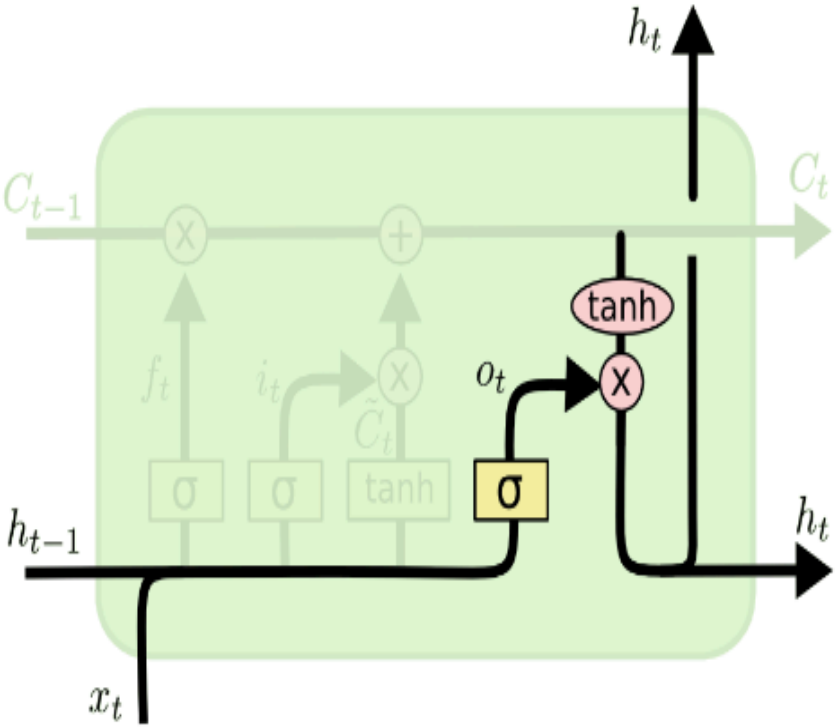


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

Long Short-Term Memory (LSTM)

Finally we output a “hidden state” for each timestep. These hidden states are consumed by other parts of a neural network.



Our hidden state could, e.g., indicate the current subject is plural to help predict the conjugation of the following verb

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

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

From BiLSTM Hidden States to Sentence Vector

Options:

- Take last hidden state
- Sum hidden states (like a contextualized bag of words)
- Pooling over hidden states: e.g. max, mean

$$\vec{s} = \max\left(\begin{bmatrix}3\\1\\4\end{bmatrix}, \begin{bmatrix}6\\3\\2\end{bmatrix}, \begin{bmatrix}1\\5\\1\end{bmatrix}\right) = \begin{bmatrix}6\\5\\4\end{bmatrix}$$

Different methods can work better on different tasks

Bi-Directional LSTM (BiLSTM)

Some language phenomena depend on the *next* word, not the previous word

For example, in Chinese we select measure words based on the noun that follows

To predict words in these positions, we can run our LSTM both ways over the text sequence

那 X 先 生 是 誰？

Sentence representations then concatenate the vectors from both forward and backward LSTM hidden state outputs, e.g. $\vec{s} = [\text{LSTM}_{\text{forward}}; \text{LSTM}_{\text{backward}}]$

Implementation Practice

Available at https://github.com/IKMLab/lstm_tutorial

RNN Implementation Practice

In this tutorial we will implement a BiLSTM encoder with max pooling over time in PyTorch. Our BiLSTM will encode sentence vectors. We will use these vectors for sentence classification on the Toxic dataset. We will compare the performance of our BiLSTM with our vanilla RNN implementation.

It is assumed you have already completed the word2vec tutorial, RNN tutorial, and are familiar with the basics of PyTorch. If not, see these links:

- https://github.com/jcjohnson/pytorch-examples
- https://github.com/IKMLab/skipgram/blob/master/Skip-Gram%20Practice.ipynb
- https://github.com/IKMLab/lstm_tutorial

Steps in this Tutorial:

1. Load pre-trained word vectors
2. Prepare sentence data for RNN processing
3. Implement a collate function
4. Learn how to use PyTorch's LSTM implementation
5. Tune hyperparameters
6. Train the model
7. Evaluate and compare results with vanilla RNN

Gated Recurrent Unit

A slightly simpler version of LSTM with less parameters.

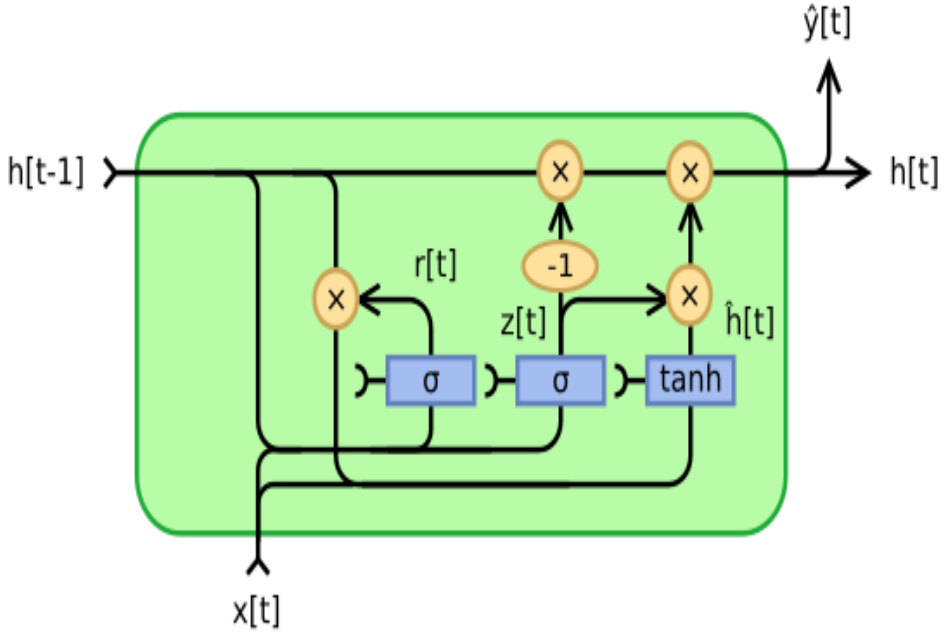
Simpler models often work better on smaller datasets.

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

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

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

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



https://en.wikipedia.org/wiki/Gated_recurrent_unit

Reset gate $\hat{\mathbf{r}}_t$ controls forgetting information

We then calculate our candidate update $\hat{\mathbf{h}}_t$

Update gate $\hat{\mathbf{z}}_t$ controls how much new information to add

New memory state uses the update gate to control the mix of new information

Recursive Tree-LSTMs -- Why Trees?

Manning 2015*:

> "...understanding novel and complex sentences crucially depends on being able to construct their meaning compositionally from smaller parts—words and multi-word expressions—of which they are constituted."

Meaning is composed hierarchically from smaller units

Human language is recursive by nature

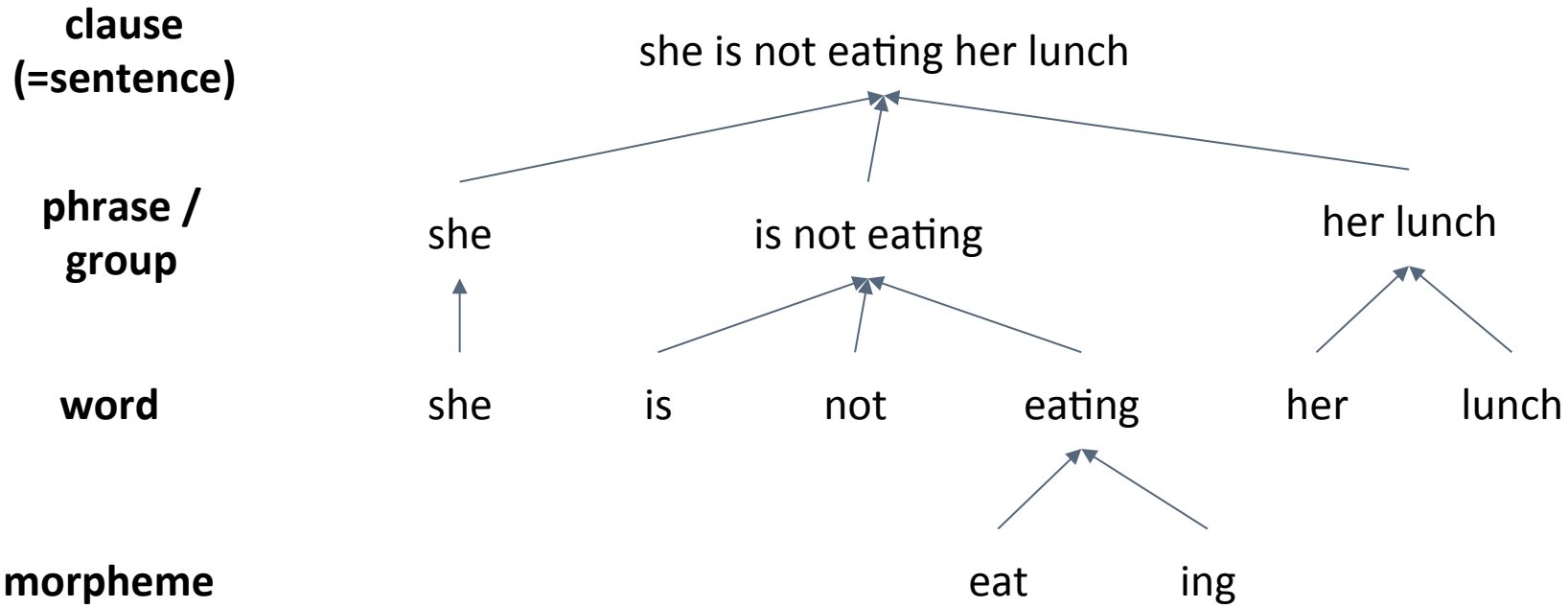
Trees can better generalize over these structures

* Manning, 2015, "Computational Linguistics and Deep Learning", Computational Linguistics.

Hierarchical Composition

Text is hierarchically composed of different structures.

English “scale of rank”*:



* Halliday, M.A.K., 2004, An Introduction to Functional Grammar, Third Edition, Hodder Education.

Recursivity in Language

The embedding of a structure within another of the same type.

I have a fat cat

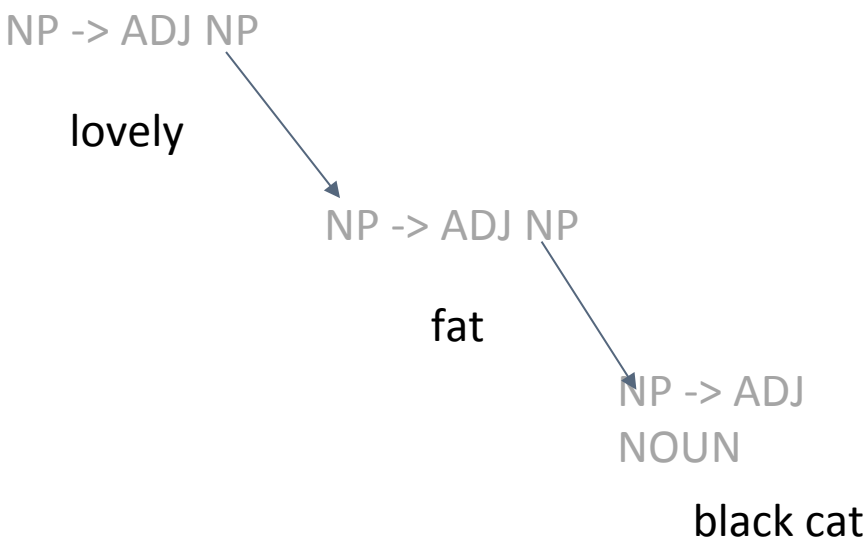
I have a fat black cat

I have a lovely fat black cat

...

NP -> ADJ NP

NP -> ADJ NOUN



* This example taken from Michael Staniek's answer to this Quora question: https://www.quora.com/What-is-recursion-recursiveness-in-linguistics

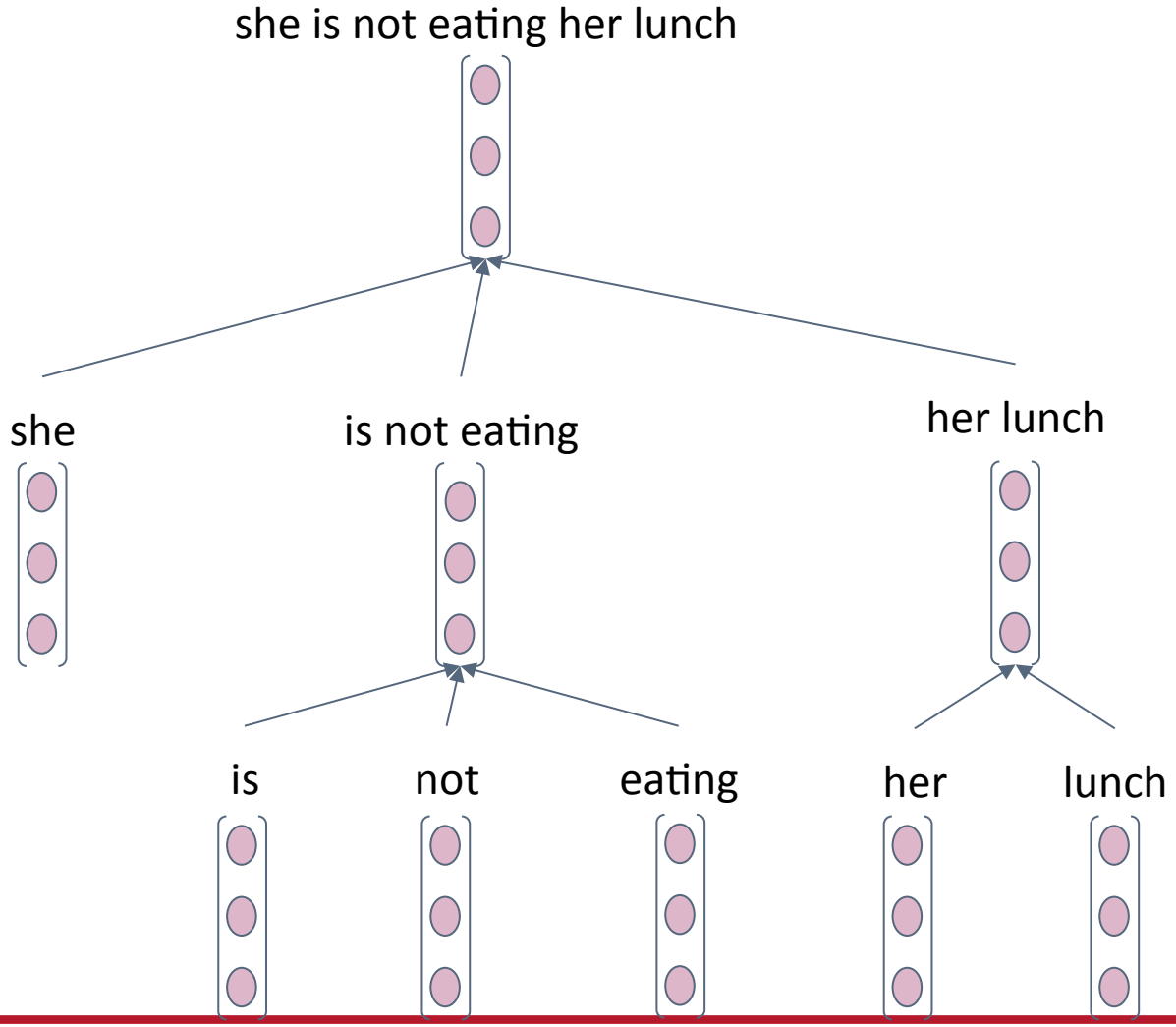
Can Linear LSTMs Handle Recursive Language?

Experiments suggest linear models trained on S-expressions can learn and generalize on recursive language structures - but not as well as tree-structured models.*

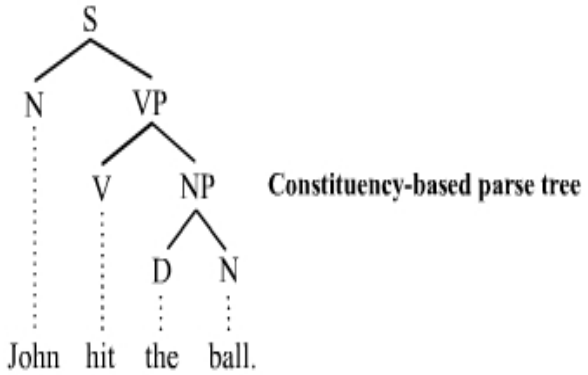
The fat cat	((The) (fat (cat)))
The fat black cat	((The) (fat (black (cat))))

* Bowman et al., 2015, Tree-Structured Composition in Neural Networks without Tree-Structured Architectures.

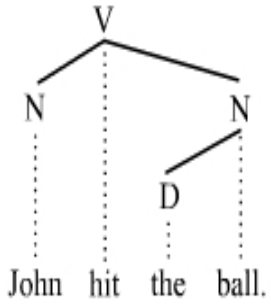
Tree-LSTM as Semantic Composition Function



Different Tree Structures



Constituency-based parse tree



Dependency-based parse tree

The leaf nodes take the word vectors as input.

Each node takes the vector corresponding to the head word as input.

Images from Wikipedia.

N-ary Tree-LSTM

$$i_j = \sigma\left(W^{(i)}x_j + \sum_{\ell=1}^{N} U_\ell^{(i)} h_{j\ell} + b^{(i)}\right), \quad (9)$$

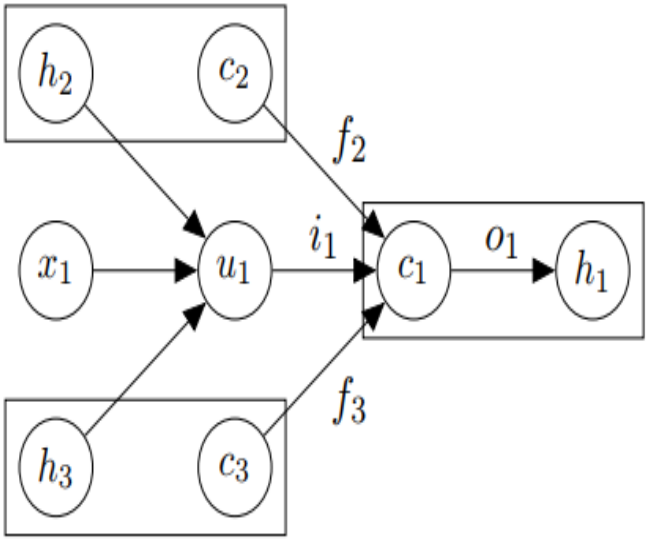
$$f_{jk} = \sigma\left(W^{(f)}x_j + \sum_{\ell=1}^{N} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right), \quad (10)$$

$$o_j = \sigma\left(W^{(o)}x_j + \sum_{\ell=1}^{N} U_\ell^{(o)} h_{j\ell} + b^{(o)}\right), \quad (11)$$

$$u_j = \tanh\left(W^{(u)}x_j + \sum_{\ell=1}^{N} U_\ell^{(u)} h_{j\ell} + b^{(u)}\right), \quad (12)$$

$$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}, \quad (13)$$

$$h_j = o_j \odot \tanh(c_j), \quad (14)$$



Focus on n=2 (binary tree)

Separate LSTM parameters for each child at each node

Updates to hidden state a controlled mixture of the two

The $\mathbf{U}_{kl}^{(f)}$ allow for interactions between left and right child forget gates, increasing flexibility

http://www.aclweb.org/anthology/P15-1150

Child-Sum Tree-LSTM

$$\tilde{h}_j = \sum_{k \in C(j)} h_k, \tag{2}$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right), \tag{3}$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right), \tag{4}$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right), \tag{5}$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right), \tag{6}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \tag{7}$$

$$h_j = o_j \odot \tanh(c_j), \tag{8}$$

- Allow for arbitrarily many children at each node (varying at each node)
- However, updates are additive over children's memory states - like a bag of words
- With more supervision (e.g. at each node) binary trees are more precise; Child-Sum Tree-LSTMs are better with less supervision due to shorter average paths to the root

http://www.aclweb.org/anthology/P15-1150

Implementation

Recursive tree encoders suffer from high computational complexity and are considerably slower than LSTMs, limiting their adoption

It is necessary to implement parallelization algorithms such as processing nodes at each level in parallel from leaves to root

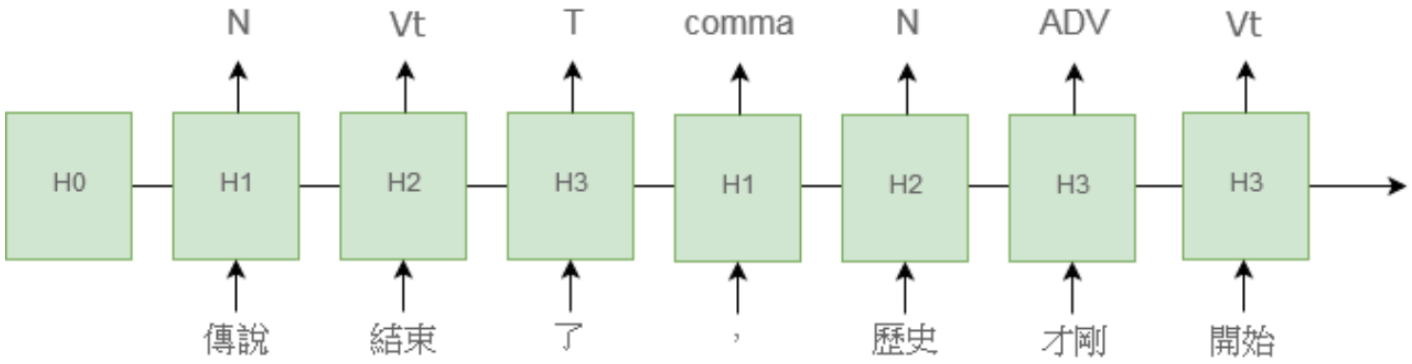
Our lab completed such an implementation:

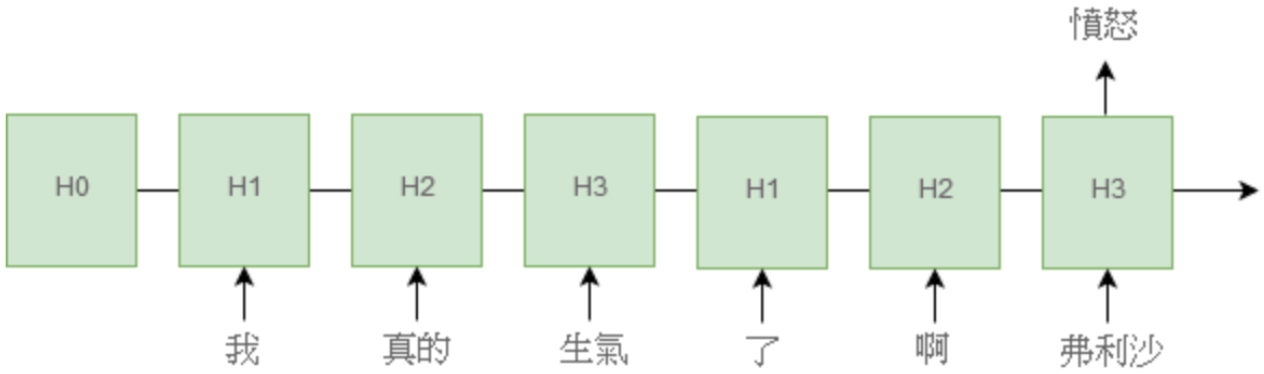
https://github.com/timniven/cstlstm

More recently, Reinforcement Learning has been explored to automatically find useful tree structures in text for downstream tasks - see https://arxiv.org/abs/1709.01121.

Sequence to Sequence

Different RNN usages

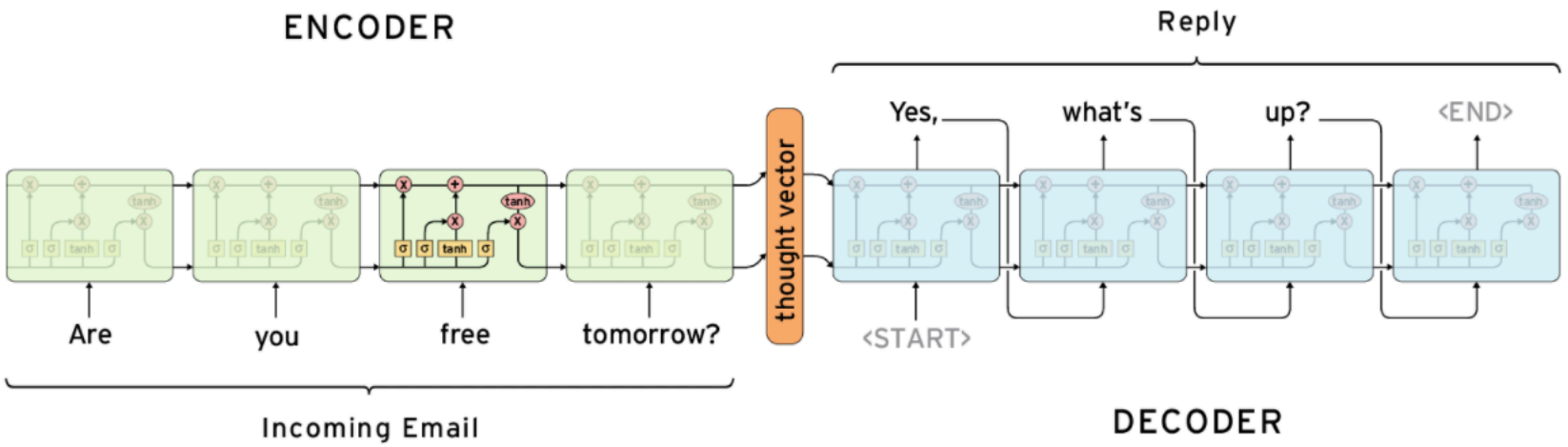




How about len(input) < len(output) ?

Figures from https://zake7749.github.io/2017/09/28/Sequence-to-Sequence-tutorial/, member in IKMlab.

Sequence to Sequence



Encoder: digest input, generate context vector

Decoder: digest context vector, generate output under previous output

Sequence to Sequence

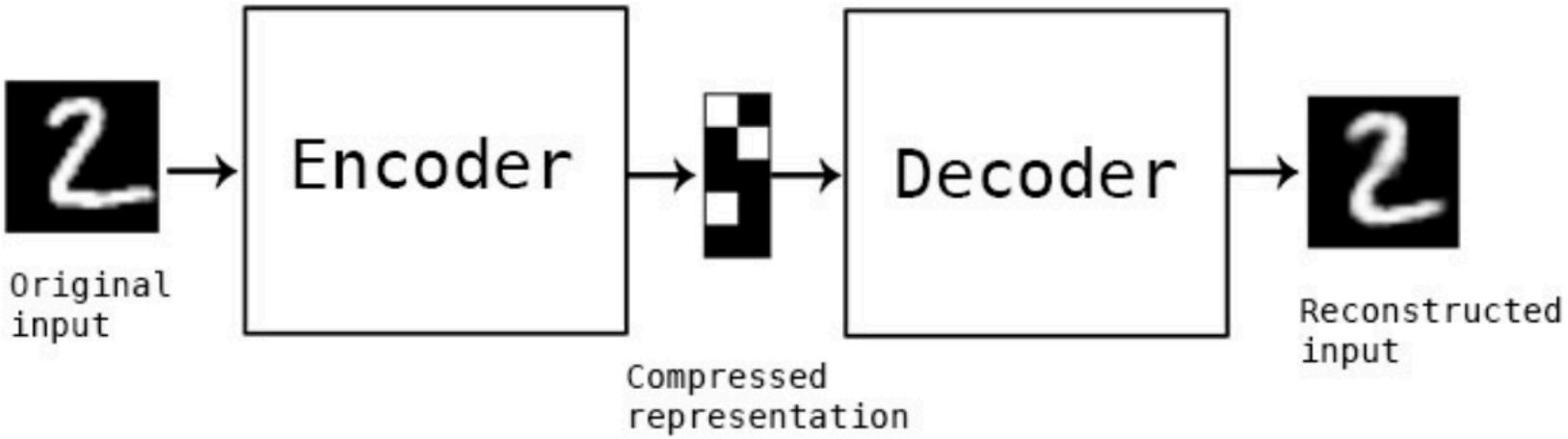
Input	Output	What's it?
English	French	Machine Translator
Question	Answer	Chatbot
Article	Summarization	Abstract system
keyword	poem	曹植+史青+寇準

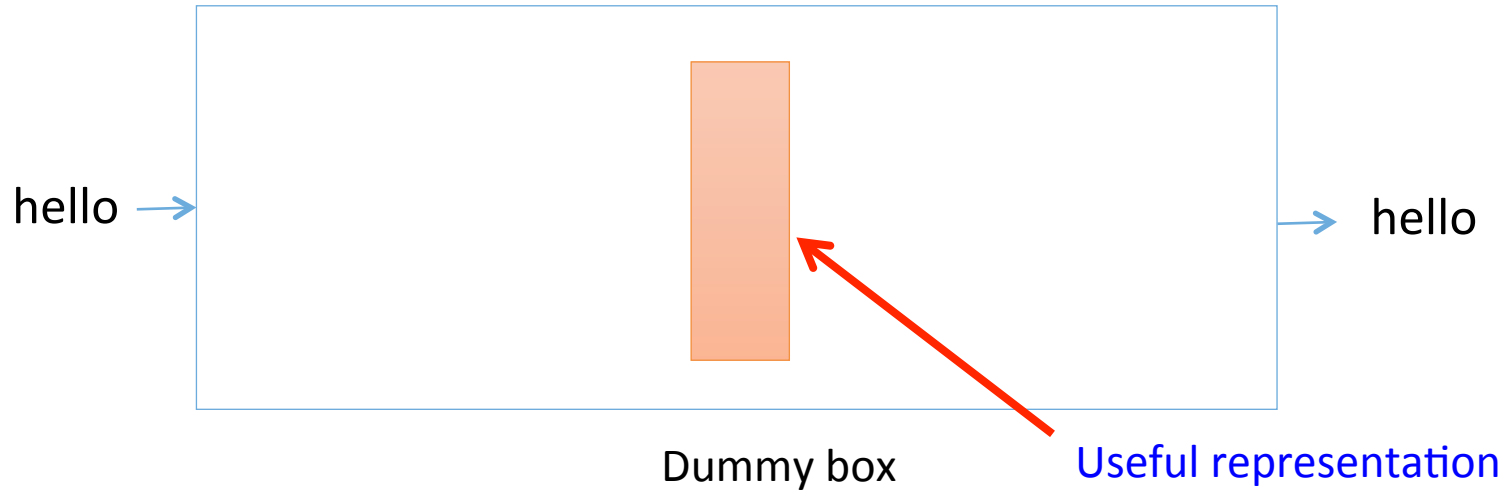
Sequence to Sequence

Use RNN + CNN as Encoder: Image Caption, Video annotation

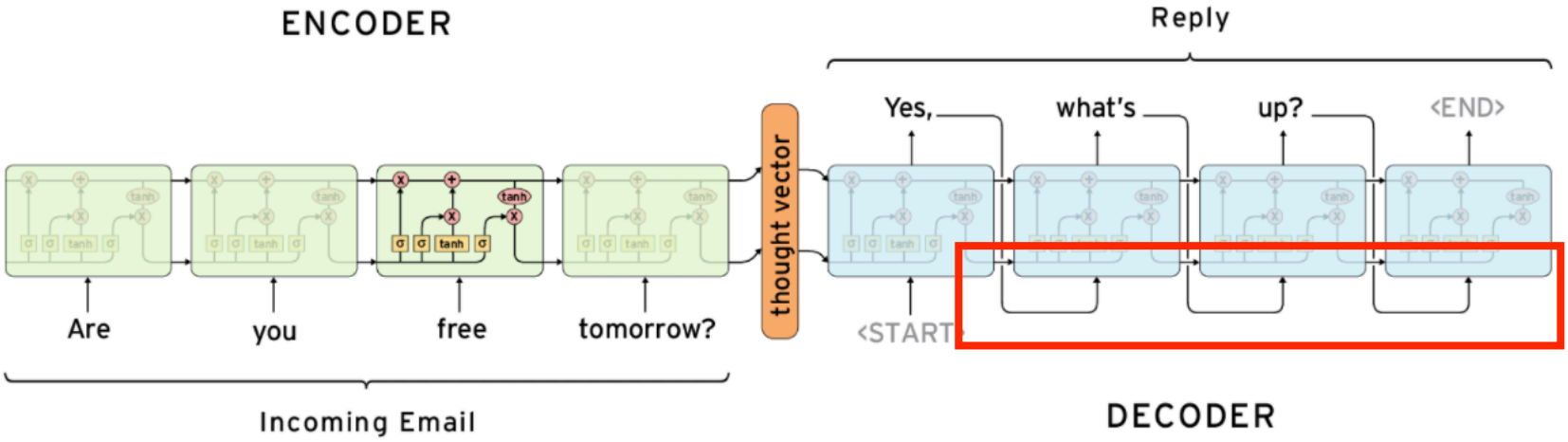
https://arxiv.org/pdf/1505.00487.pdf

Train Sequence Auto-encoder first





Issues in Decoder

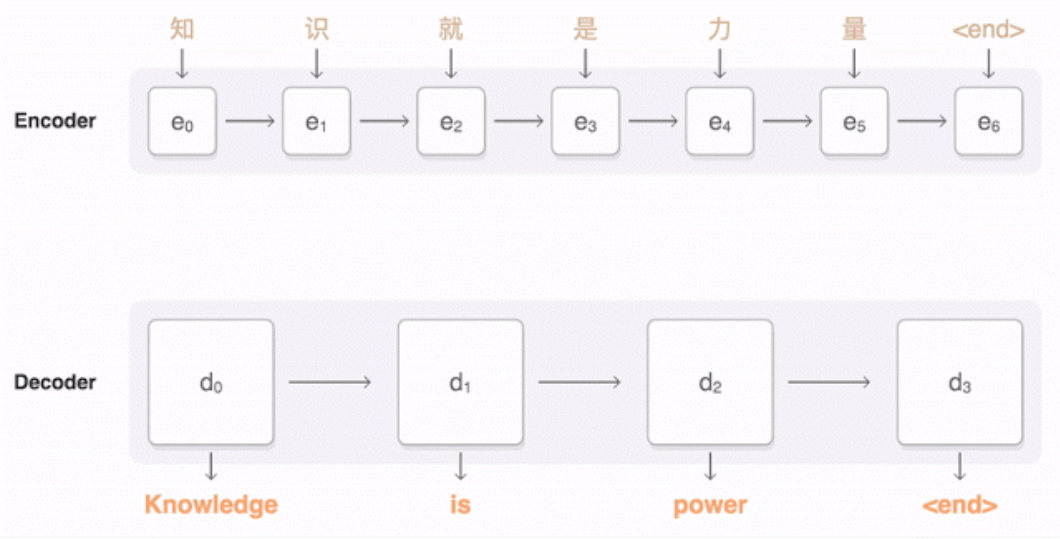


推 Zeropapa:你女友是不是鼻子尖尖的 10/25 17:44
推 k87559527: 鬍子翹翹的 10/25 17:45
噓 LPKing: 手裡還拿根釣竿? 10/25 17:46
推 yuanwu:(ˊ﹀ˋ)/ 10/25 17:49
噓 Combatant: 娘 這不是波爾嗎 10/25 17:49

use_teacher_forcing: use ground truth as input

Attention Model

Decoder use selected subset from compressed representation



From Google neural MT https://google.github.io/seq2seq/

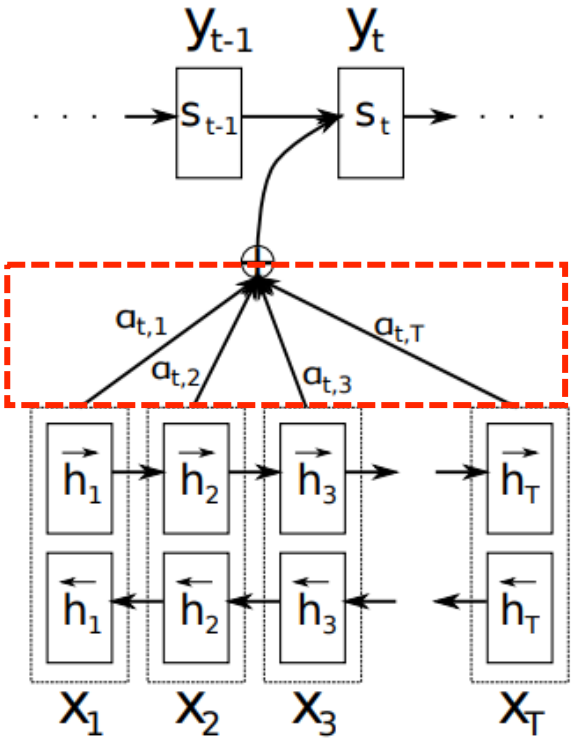


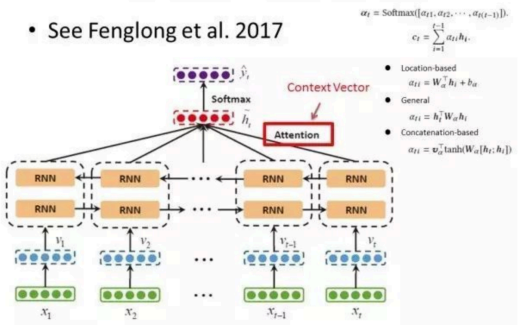
Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

https://arxiv.org/pdf/1409.0473.pdf

Attention model is hot!

Attention-based Bi-RNN

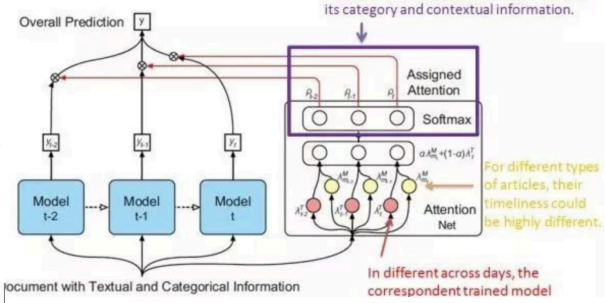
- See Fenglong et al. 2017



Dynamic Attention Deep Model

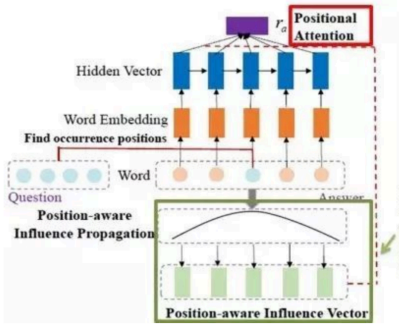
- See Huejian et al. 2017

When comes to a article at time t, we need consider its influences before according to its category and contextual information.



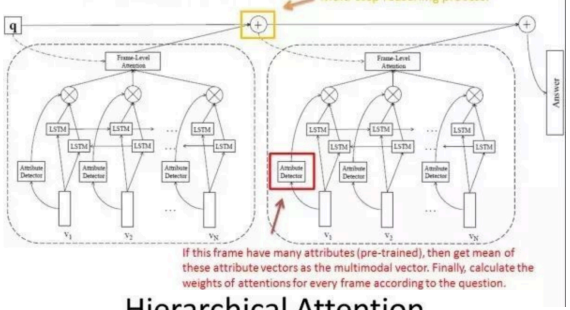
Positional Attention for QA

- See Qin et al. 2017



Attributed-Augmented Attention

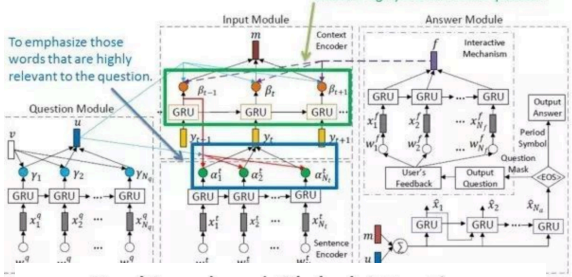
- See Yunan et al. 2017



Context-aware Attention

- See Huayu et al. 2017

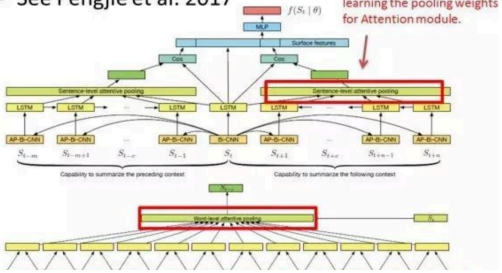
A sentence level attention mechanism is enabled to emphasize those sentences that are highly relevant to the question.



A Neural Attention Model for Summarization

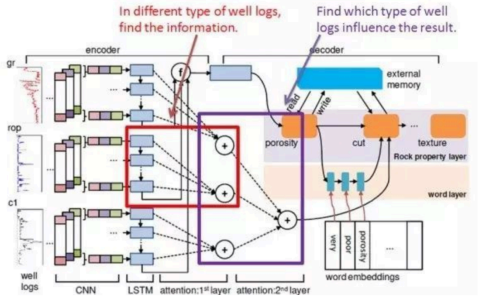
- See Pengjie et al. 2017

Use sentence relations to learning the pooling weights for Attention module.



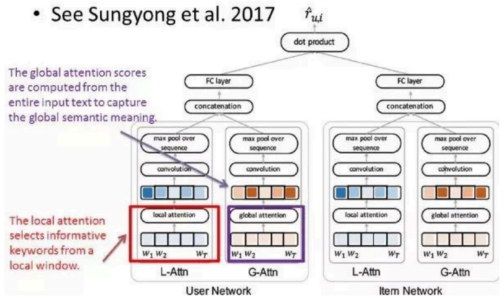
Hierarchical Attention

- See Bin et al. 2017



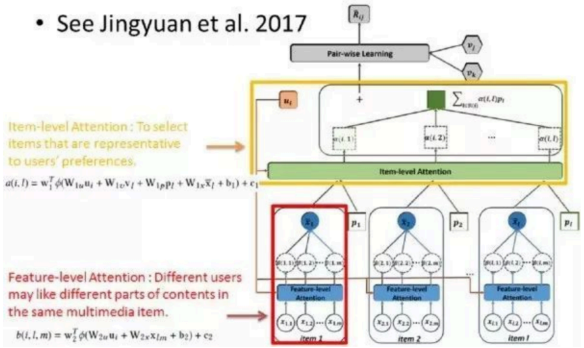
Dual Local and Global Attention

- See Sungyong et al. 2017



Item- and Component-Level Attention

- See Jingyuan et al. 2017



Example Application of Sentence Vectors

Argument Reasoning Comprehension Task

Claim: Google is not a harmful monopoly

Reason: People can choose not to use Google

Warrant: Other search engines do not re-direct to Google

Alternative: All other search engines re-direct to Google

Have to pick the correct one out of these two (binary classification)

Challenges

Tiny dataset:

- 1,210 training samples
- 326 development set samples
- No common arguments or claims across training, development, and testing datasets
- Complicated models overfit terribly on small data
- But this is a very complicated task

Challenges

High level of language understanding

"distraction" in **W** is a good thing and supports the argument

Whereas "distract" in **A** is a bad thing and supports the opposite of **C**

C	They add a lot to the piece and I look forward to reading comments
R	Comment sections have not failed
W	Comments sections are a welcome distraction from my work
A	Comments sections always distract me from my work

Challenges

Require world knowledge

Must know what a monopoly is, how consumer choice relates to monopoly, that Google is a search engine, and how web re-directs relate to search engine monopolies

Claim: Google is not a harmful monopoly

Reason: People can choose not to use Google

Warrant: Other search engines do not re-direct to Google

Alternative: All other search engines re-direct to Google

Baselines

Approach	Dev	(±)	Test	(±)
Human average			.798	.162
Human w/ training in reasoning			.909	.114
Random baseline	.473	.039	.491	.031
Language model	.617		.500	
Attention	.488	.006	.513	.012
Attention w/ context	.502	.031	.512	.014
Intra-warrant attention	**.638**	.024	.556	.016
Intra-warrant attent. w/ context	.637	.040	**.560**	.055

Table 2: Accuracy of each approach (humans and systems) on the development set and test set, respectively.

https://arxiv.org/pdf/1708.01425.pdf

How to Start?

- Pick an obvious problem to try and solve
- Start with simple models to get a feel for it

An obvious problem is the data set size restricting model usage.

Can we do something about that?

Transfer learning.

Transfer of Semantic Knowledge

- Training LSTMs on a semantically challenging task forces them to learn useful representations of the sentences
- The representations capture rich semantic features required to succeed at the task
- The encoder that learns these representations can then be transferred for use in other domains
- Pre-trained word vectors are another form of transferred semantic knowledge that improve neural network performance in NLP

Natural Language Inference

Semantically, this is a very demanding task

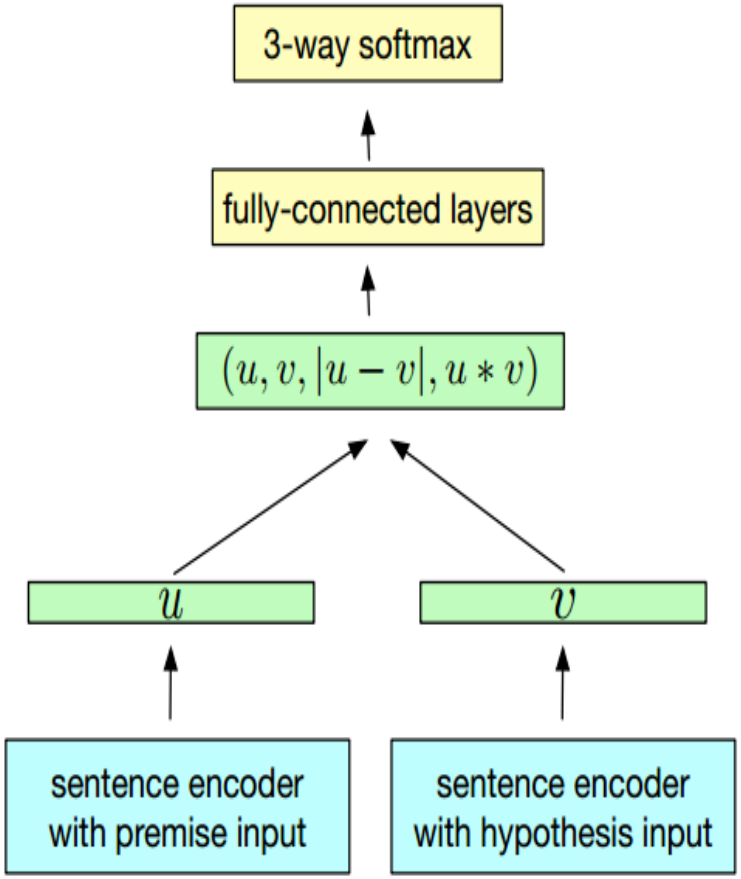
Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

Natural Language Inference

The “Siamese” encoder architecture:

- Sentences are encoded separately (but with the same encoder)
- They are then given to another neural network module for classification, usually a Multi-Layer Perceptron

This is how we pre-train encoders with natural language inference data in our experiments

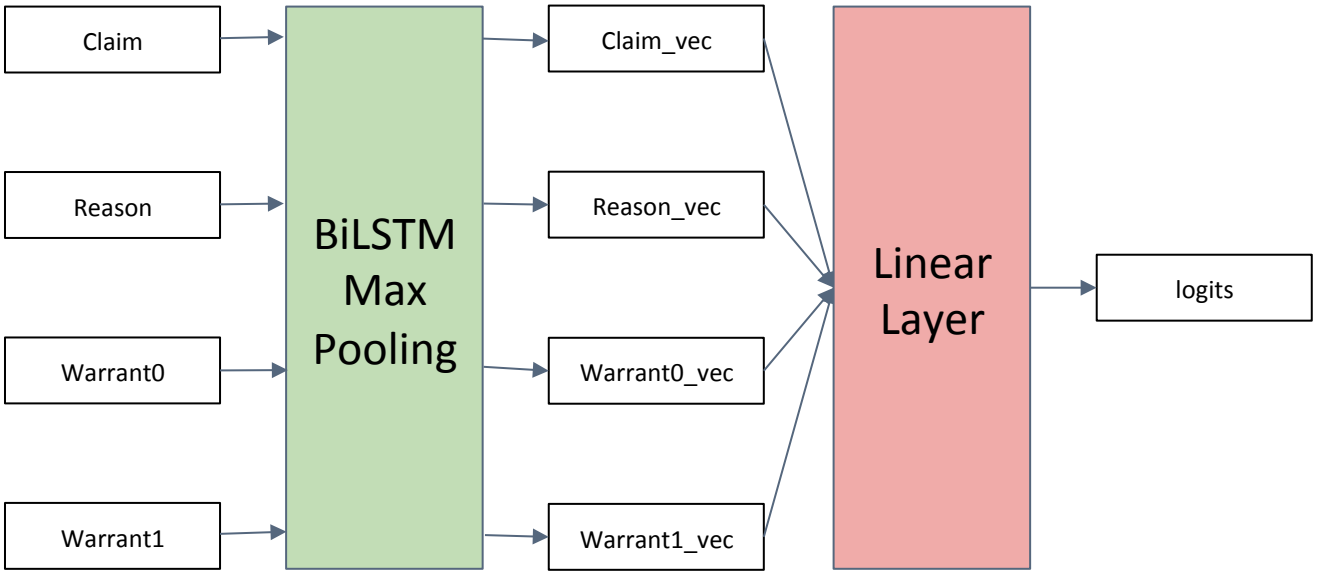


Start Simple

Conneau et al. (2017) suggest passing encoder features to a linear classifier to save parameters

This model sees a mean test set score of 0.538

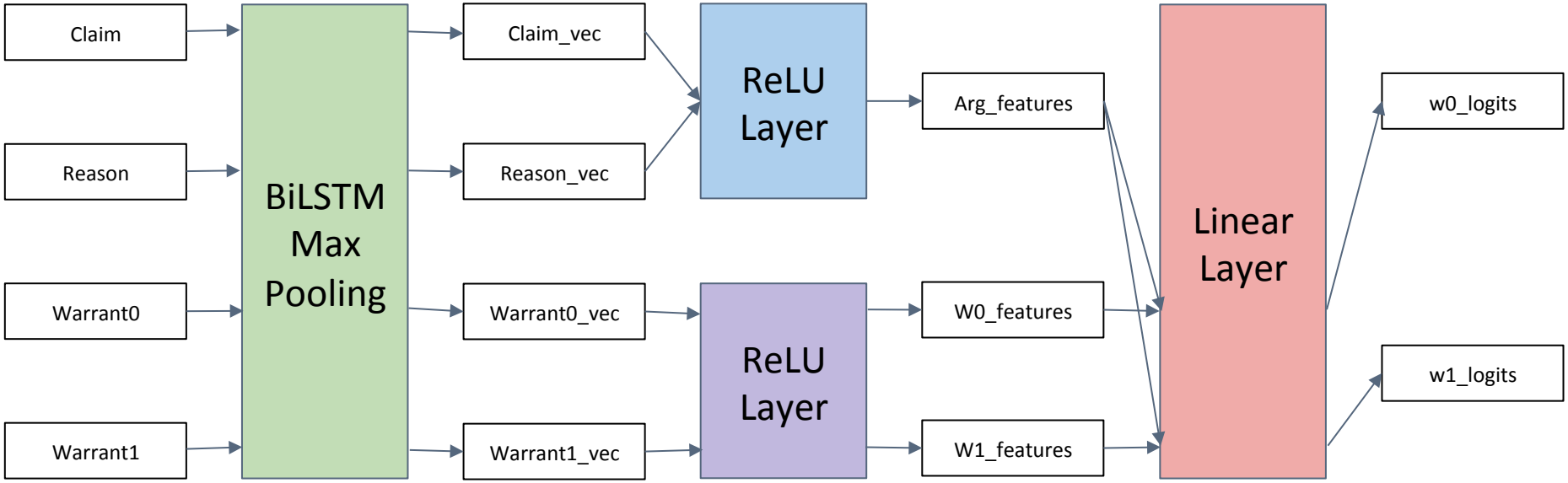
Doesn't beat the baseline



Where to Next?

- It's good to start with simple models to get a feel for the task.
- Where to go next with more complicated models?
- Deep learning is about learning representations and then performing statistical inferences over those features
- In this case, let's try learning task-specific features before the linear classifier

Our Model


Claim
Reason
Warrant0
Warrant1
BiLSTM Max Pooling
Claim_vec
Reason_vec
Warrant0_vec
Warrant1_vec
ReLU Layer
ReLU Layer
Arg_features
W0_features
W1_features
Linear Layer
w0_logits
w1_logits

Model Performance

Rank	Team/System	Accuracy
1	GIST	0.712
2	blcu_nlp	0.606
3	ECNU	0.604
4	NLITrans	0.590
5	Joker	0.586
6	YNU_Deep	0.583
7	mingyan	0.581
8	ArcNet	0.577
8	UniMelb	0.577
10	TRANSRW	0.570
11	lyb3b	0.568
12	SNU_IDS	0.565
13	ArgEns-GRU	0.556
14	ITNLP-ARC	0.552
15	YNU-HPCC	0.550
16	TakeLab	0.541
17	HHU	0.534
18	Random baseline	0.527
19	Deepfinder	0.525
20	ART	0.518
21	RW2C	0.500
22	ztangfdu	0.464

Our max score of 0.655 would put our model in second place in the Argument Reasoning Comprehension Task (SemEval 2018)

A better evaluation is the mean 0.613 over 20 runs (still second place)

The first place entry also uses transfer learning

Our code at:

https://github.com/IKMLab/arct

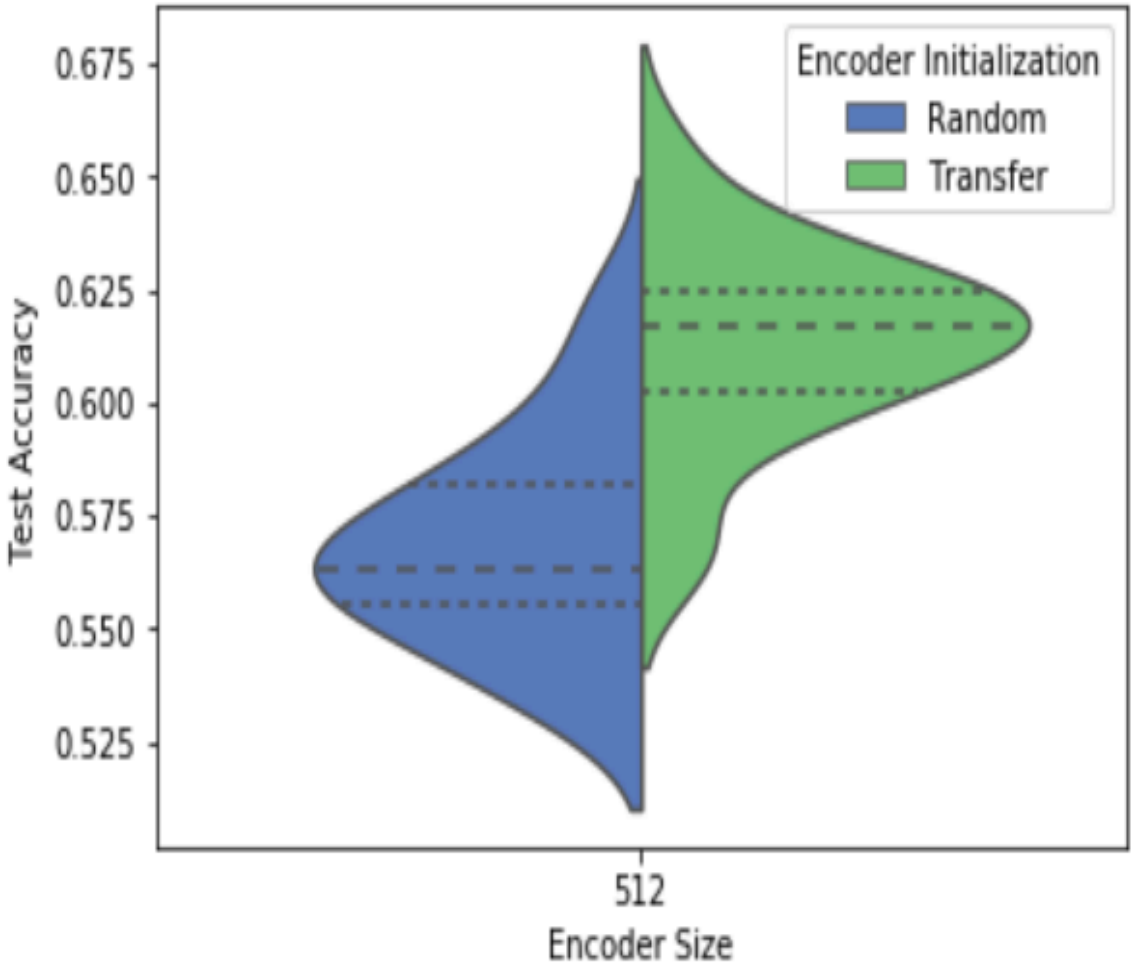
Benefits of Transfer to Our Model

Distributions come from 20 different random initializations

mean(transfer) > mean(random)

0.6131 > 0.5695

$p = 3.6 \times 10^{-7}$



Niven and Kao (2018) "NLITrans at SemEval 2018 Task 12: Transfer of Semantic Knowledge for Argument Comprehension" (NAACL 2018)