Natural Language Processing

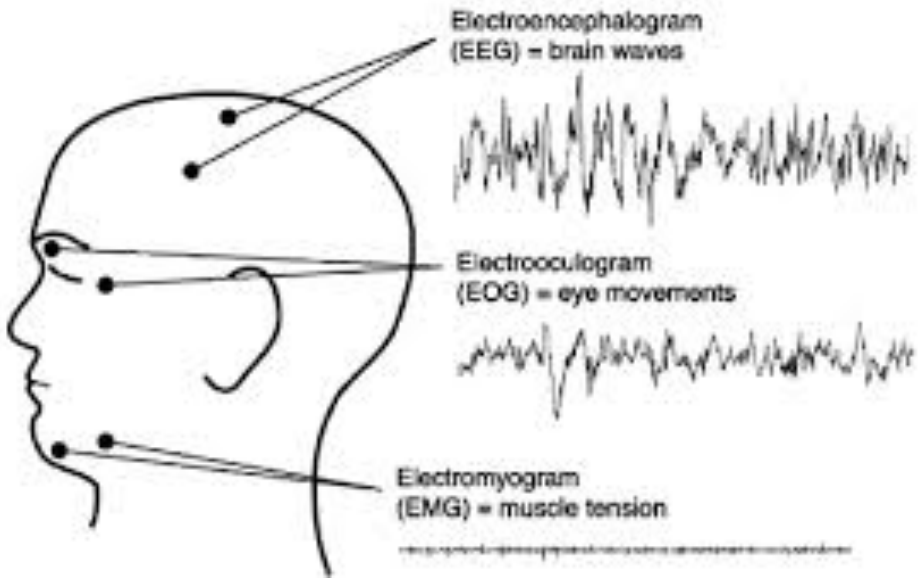
Pushpak Bhattacharyya
CSE Dept,
IIT Patna and Bombay

CNN, Attention, Summing Up

Crux of NLP-ML

- Object A: extract parts and features
- Object B which is in correspondence with A: extract parts and features
- LEARN mappings of these features and parts
- Use in NEW situations: called DECODING

Thought Reader!



"I am hungry now"

Method changes, philosophy does not!

- Classical, Statistical, DL: method changes NOT the fundamental principle

- NLP-ML works on the principle of establishing CORRESPONENCES
- Correspondence of parts
- Correspondence of features

CNN for Sentiment Analysis of Short Texts

Santos, C. N. dos, & Gatti, M. (2014). Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. In COLING-2014 (pp. 69–78)

Directly hitting the approach…

- Character to Sentence Convolutional Neural Network (CharSCNN)

- Uses two convolutional layers to extract relevant features from words and sentences of any size

Approach cntd.

- The network extracts features from the character-level up to the sentence-level
- Two convolutional layers

Initial representations (1/2)

- First layer transforms words into real-valued feature vectors (embeddings)
  - capture morphological, syntactic and semantic information
- Fixed size word vocabulary V^{wrd}
- Fixed-sized character vocabulary V^{chr}
- Given a sentence consisting of N words $\{ w_1, w_2, ..., w_N\}$, every word wn is converted into a vector $u_n = [r^{wrd}; r^{wch}]$

Initial representations (2/2)

- Two sub-vectors:
  - Word-level embedding r^{wrd} and character-level embedding r^{wch}
- While word-level embeddings are meant to capture syntactic and semantic information, character-level embeddings capture morphological and shape information.

Data set: SSTb

- Movie reviews and Twitter posts
- Stanford Sentiment Treebank (SSTb) (Socher et al., 2013b)
- Fine grained sentiment labels for 215,154 phrases in the parsetrees of 11,855 sentences

Data set: STS

- Introduced by Go et al. (2009)
- Original training set contains 1.6 million tweets
- Sample of the training data consisting of 80K (5%) randomly selected tweets.
- Development set of randomly selecting 16K

Data set particulars

Dataset	Set	# sentences / tweets	# classes
SSTb	Train	8544	5
	Dev	1101	5
	Test	2210	5
STS	Train	80K	2
	Dev	16K	2
	Test	498	3

Word embeddings used

- Word2vec: continuous bag-of-words and skip-gram (Mikolov et al., 2013)
- December 2013 snapshot of the English Wikipedia corpus
- Word must occur at least 10 times in order to be included in the vocabulary, which resulted in a vocabulary of 870,214 entries.
- Used word2vec's skip-gram method with a context window of size 9.
- Training time: 1h10min using 12 threads in a Intelr Xeonr E5-2643 3.30GHz machine

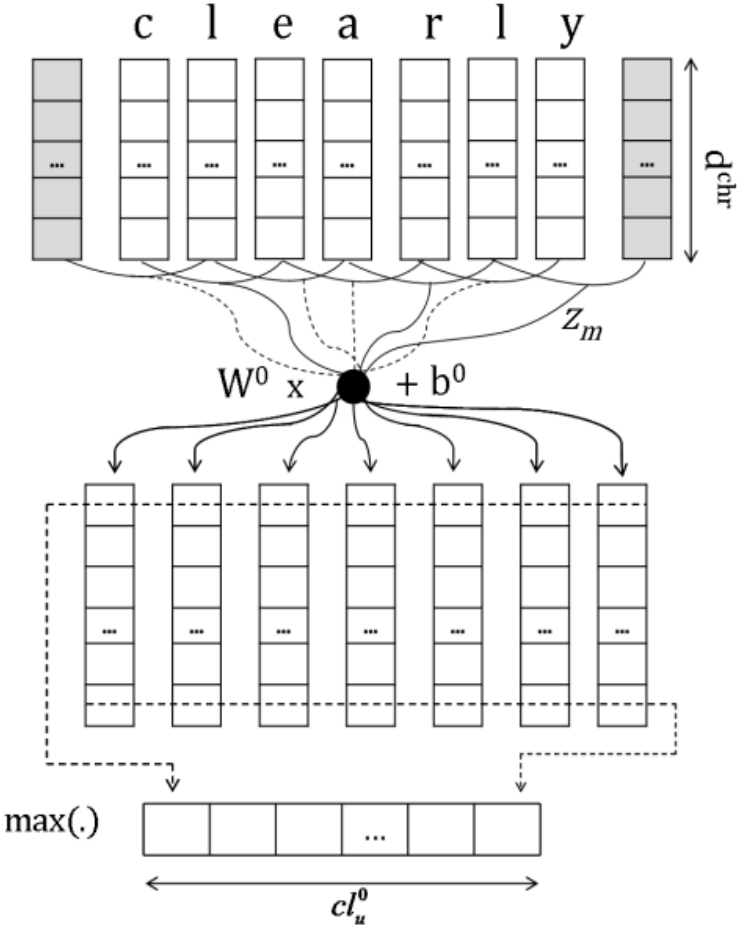
Pre-processing for word embedding

- Removal of paragraphs that are not in English
- Substitution of non-western characters for a special character
- Tokenization of the text using the tokenizer available with the Stanford POS Tagger (Manning, 2011)
- Removal of sentences that are less than 20 characters long (including white spaces) or have less than 5 tokens
- Lowercase all words and substitute each numerical digit by a 0 (e.g., 1967 becomes 0000)
- Resulting clean corpus contains about 1.75 billion tokens.

Character embedding

- Pre-training of character-level embeddings, which are initialized by randomly sampling each value from an uniform distribution: U (− r, r):$\sqrt{\frac{6}{|V^{chr}| + d^{chr}}}$.
- 94 different characters in the SSTb corpus and 453 different characters in the STS corpus
- Raw (not lowercased) words are used to construct the character vocabularies, which allows the network to capture relevant information about capitalization

Character level convolutional feature extraction



Neural network hyper parameters

Parameter	Parameter Name	SSTb	STS
d^{wrd}	Word-Level Embeddings dimension	30	30
k^{wrd}	Word Context window	5	5
d^{chr}	Char. Embeddings dimension	5	5
k^{chr}	Char. Context window	3	3
cl_u^0	Char. Convolution Units	10	50
cl_u^1	Word Convolution Units	300	300
hl_u	Hidden Units	300	300
λ	Learning Rate	0.02	0.01

Results on SSTb corpus (5-classes and binary)

Model	Phrases	Fine-Grained	Positive/Negative
CharSCNN	yes	**48.3**	**85.7**
SCNN	yes	**48.3**	85.5
CharSCNN	no	43.5	82.3
SCNN	no	43.5	82.0
RNTN (Socher et al., 2013b)	yes	45.7	85.4
MV-RNN (Socher et al., 2013b)	yes	44.4	82.9
RNN (Socher et al., 2013b)	yes	43.2	82.4
NB (Socher et al., 2013b)	yes	41.0	81.8
SVM (Socher et al., 2013b)	yes	40.7	79.4

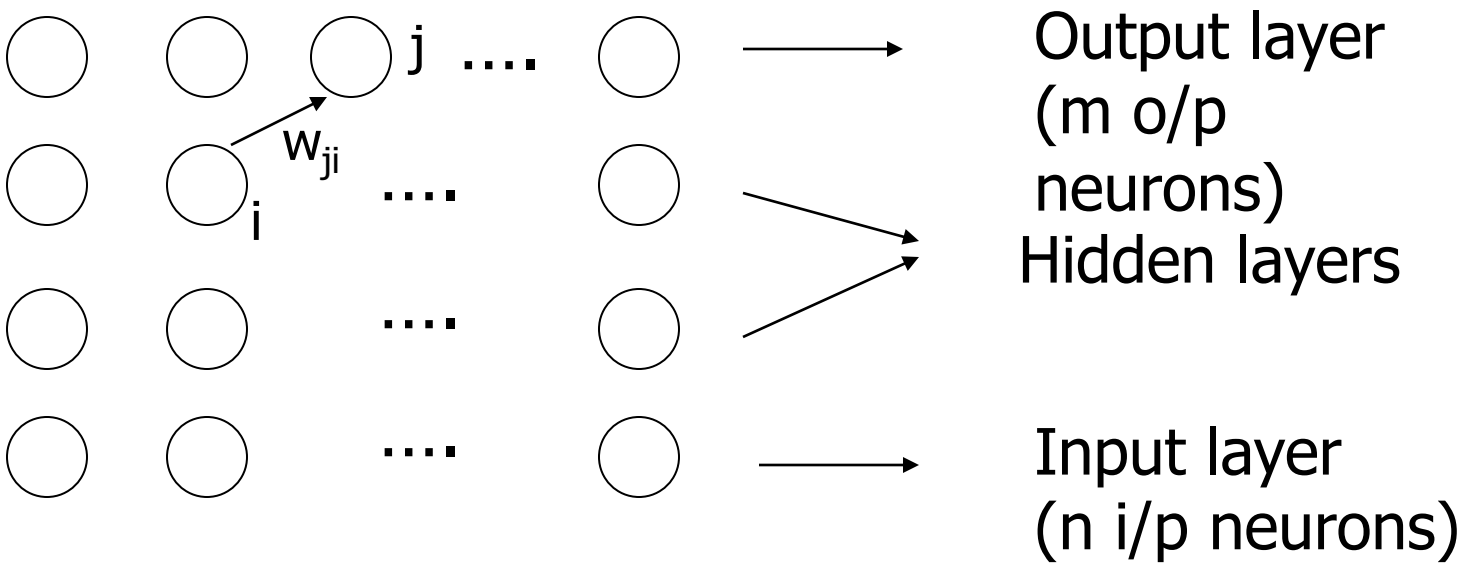
STS: binary (positive-negative)

Model	Accuracy (unsup. pre-training)	Accuracy (random word embeddings)
CharSCNN	**86.4**	81.9
SCNN	85.2	82.2
LProp (Speriosu et al., 2011)	84.7	
MaxEnt (Go et al., 2009)	83.0	
NB (Go et al., 2009)	82.7	
SVM (Go et al., 2009)	82.2	

Recap

Feedforward Network and Backpropagation

Backpropagation algorithm



- Fully connected feed forward network
- Pure FF network (no jumping of connections over layers)

General Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

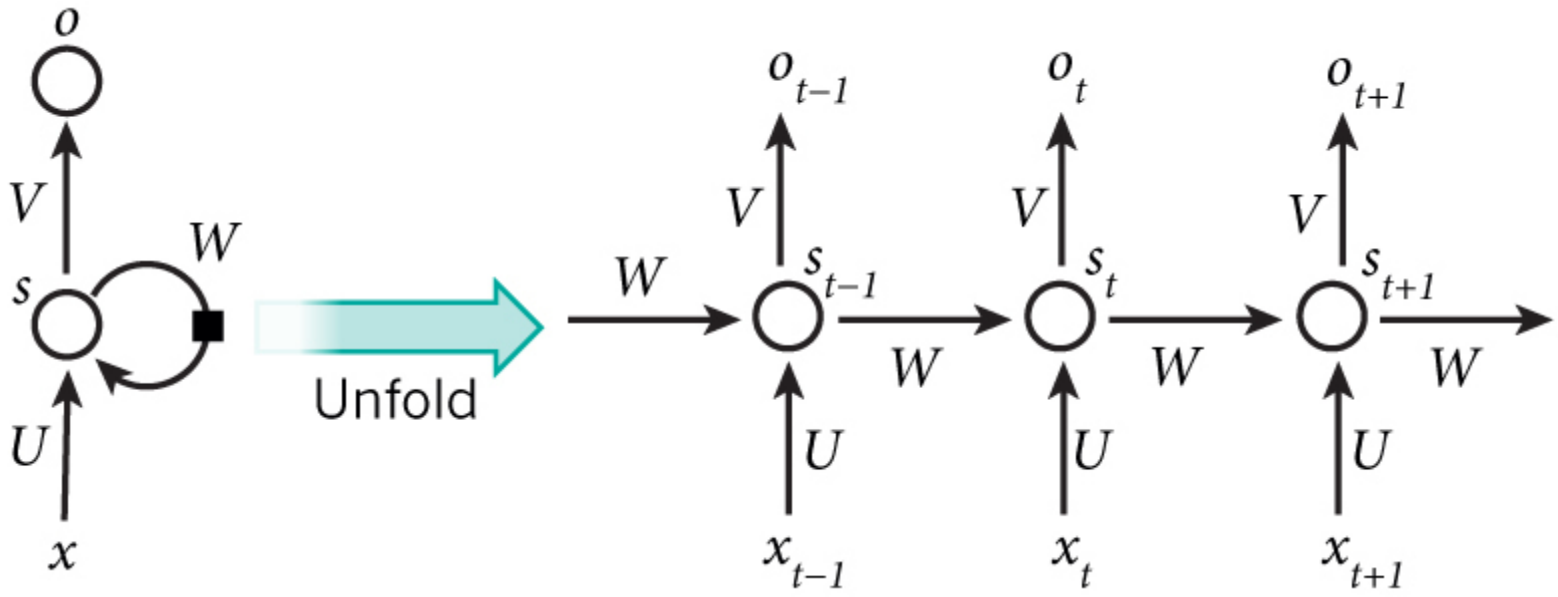
- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

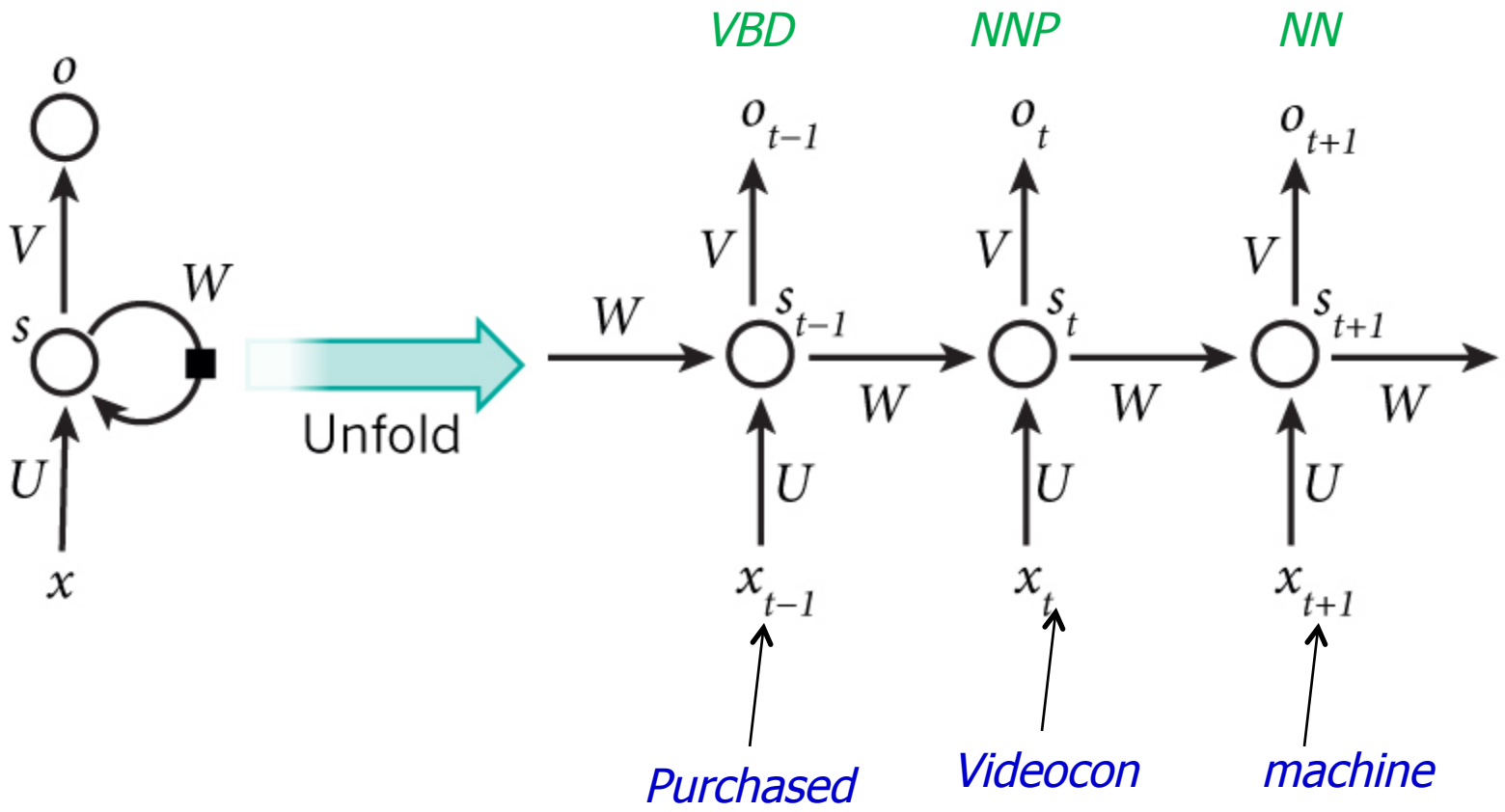
$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \text{ for hidden layers}$$

Recurrent Neural Network

Sequence processing m/c



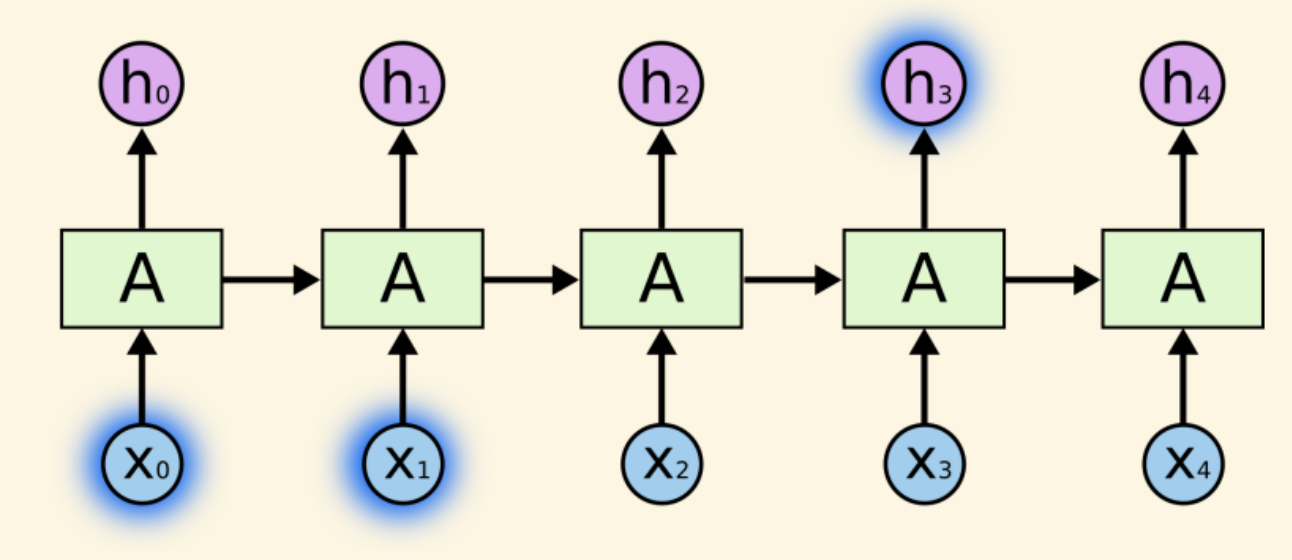
E.g. POS Tagging



LSTM

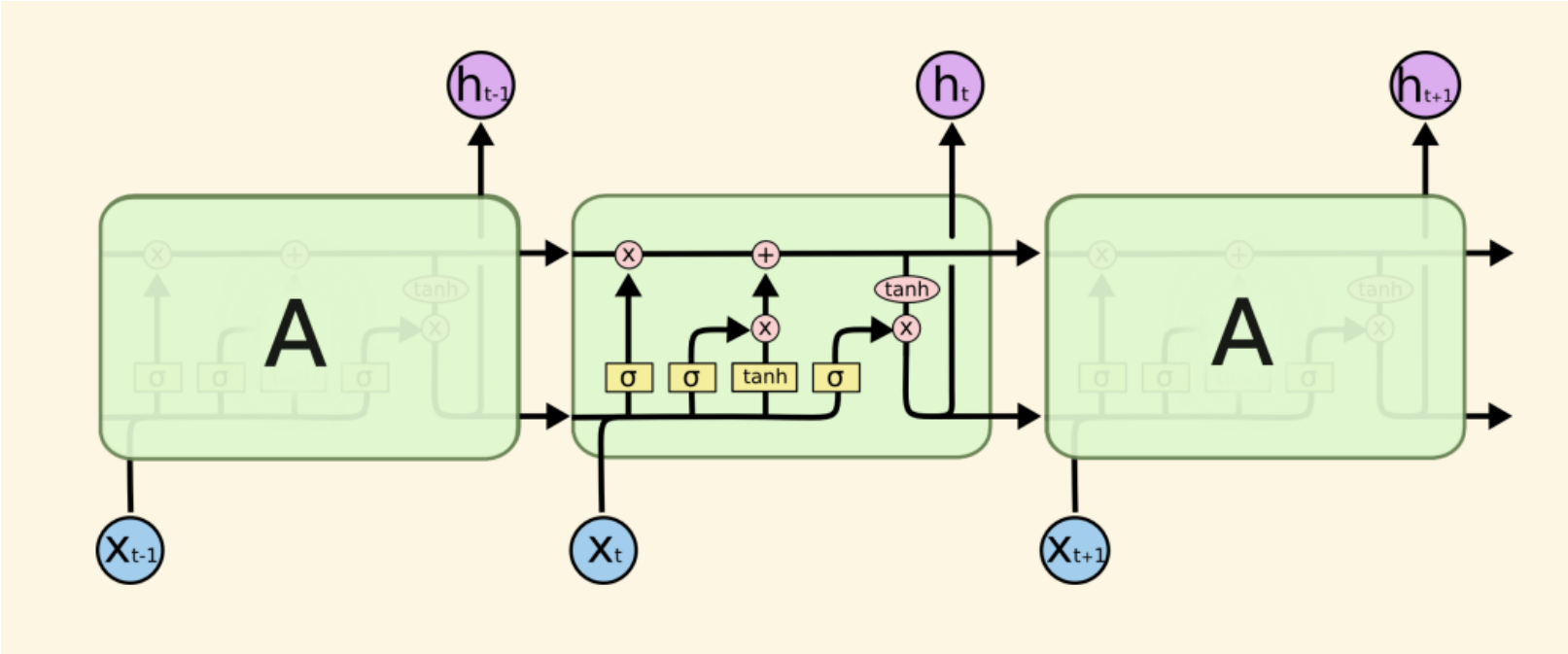
(Ack: Lecture notes of Taylor Arnold, Yale and http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM: a variation of vanilla RNN



Vanilla RNN

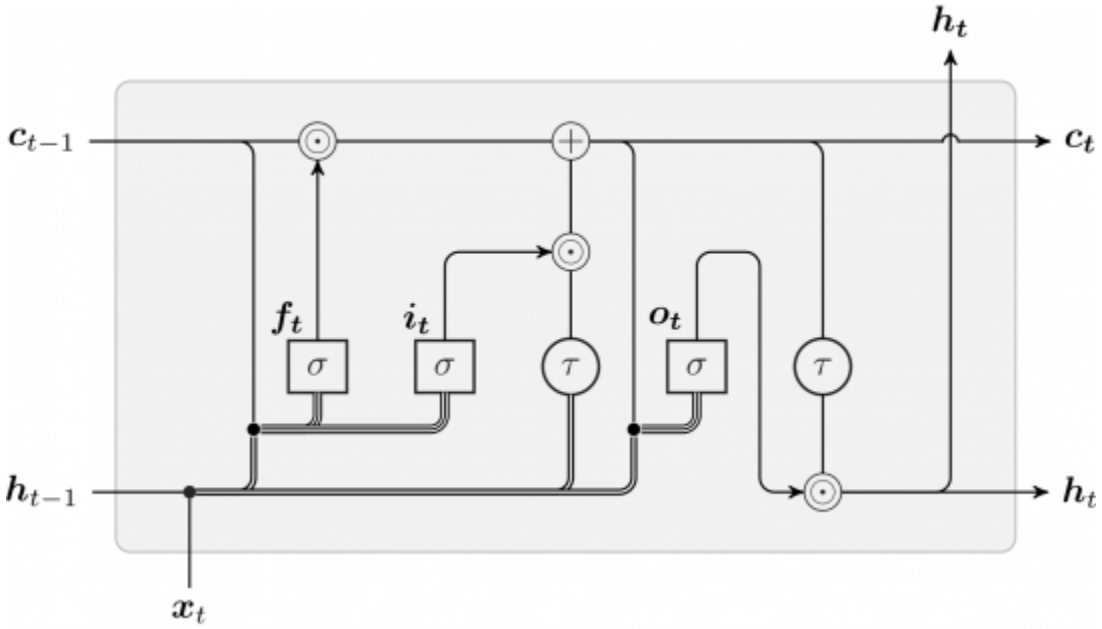
LSTM: complexity within the block



Central idea

- Memory cell maintains its state over time
- Non-linear gating units regulate the information flow into and out of the cell

A simple line diagram for LSTM



Example of Refrigerator complaint

- *Visiting service person is becoming rarer and rarer,*

  (ambiguous! 'visit to service person' OR 'visit by service person'?)

  …

- *and I am regretting/appreciating my decision to have bought the refrigerator from this company*

  (appreciating → 'to'; regretting → 'by')

Possibilities

- ‘Visiting’: ‘visit to’ or ‘visit by’ (ambiguity, syntactic opacity)
- Problem: solved or unsolved (not known, semantic opacity)
- ‘Appreciating’/’Regretting’: transparent; available on the surface

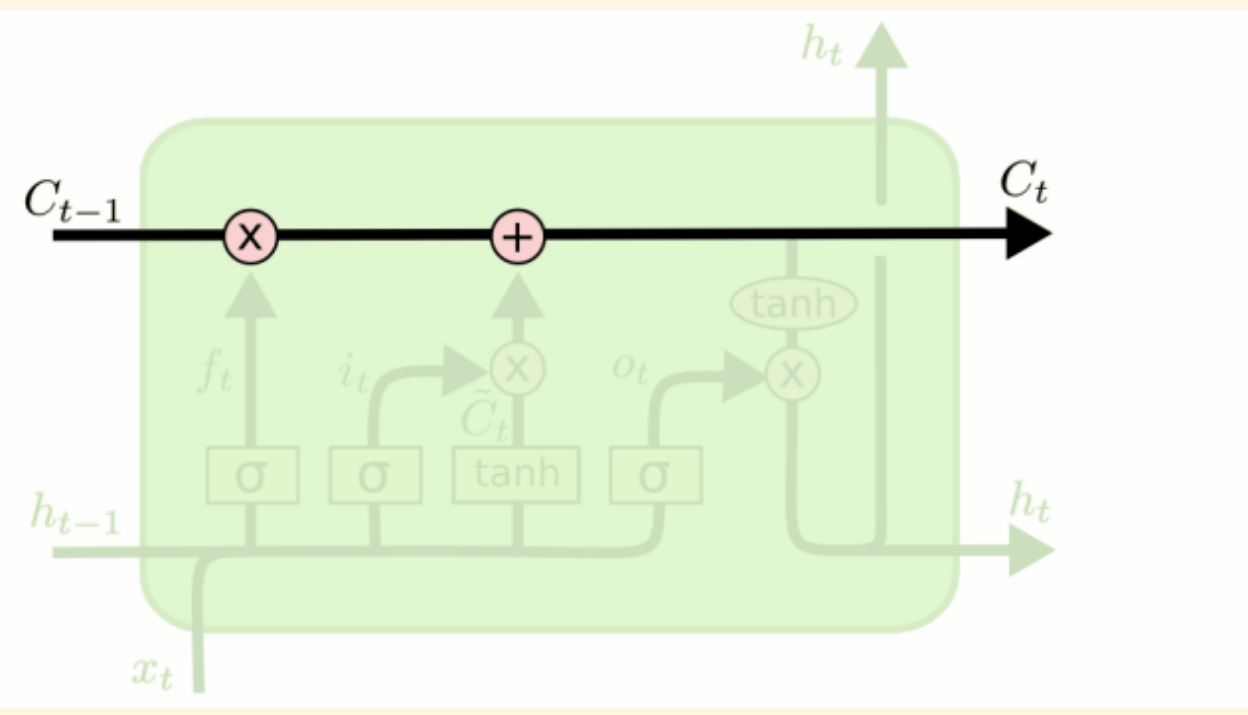
4 possibilities (states)

Clue-1	Clue-2	Problem	Sentiment
Visit to service person	*Appreciating*	solved	Positive
Visit to service person	*Appreciating*	Not solved	Not making sense! Incoherent
Visit to service person	*Regretting*	solved	May be reverse sarcasm
Visit to service person	*Regretting*	Not solved	Negative

4 possibilities (states)

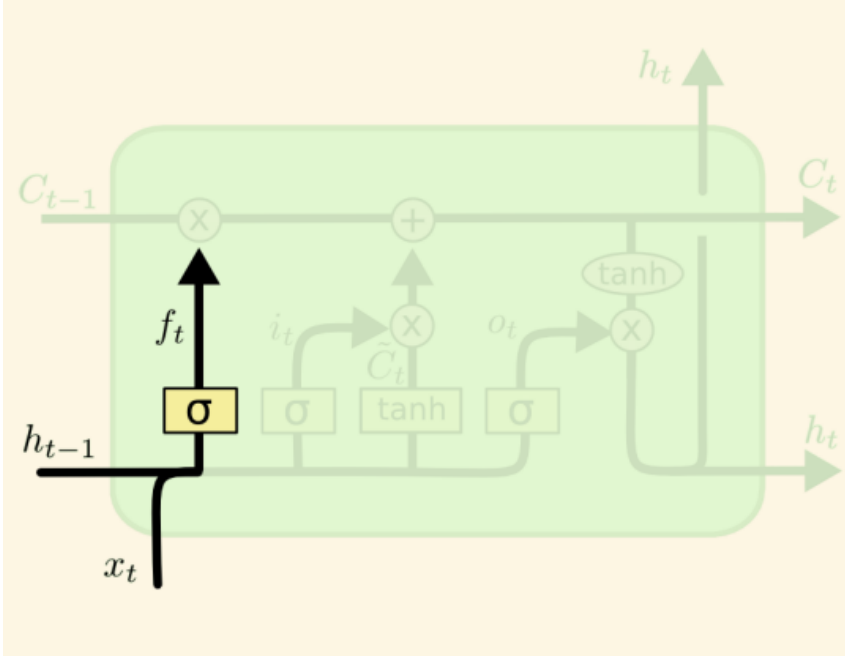
Clue-1	Clue-2	Problem	Sentiment
Visit by service person	*Appreciating*	solved	Positive
Visit by service person	*Appreciating*	Not solved	May be sarcastic
Visit by service person	*Regretting*	solved	May be reverse sarcasm
Visit by service person	*Regretting*	Not solved	Negative

LSTM constituents: Cell State



The first and foremost component- the controller of flow of information

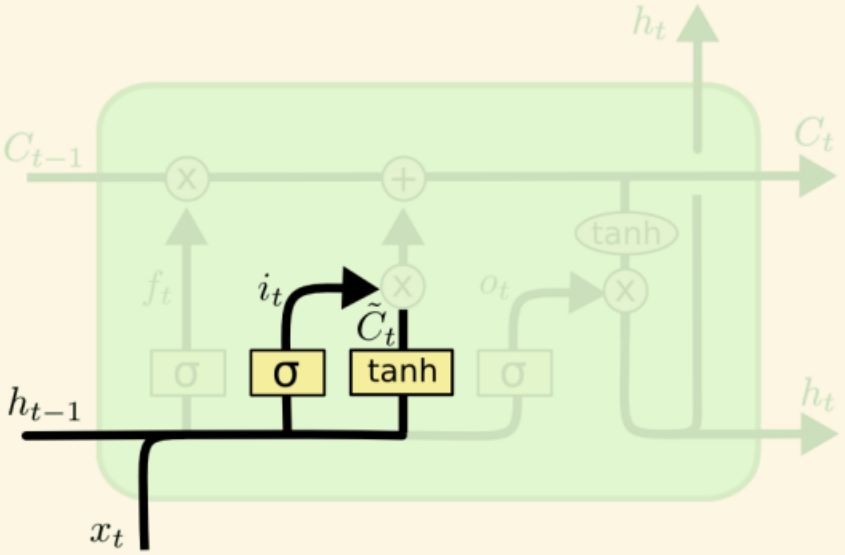
LSTM constituents- Forget Gate



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

Helps forget irrelevant information. Sigmoid function. Output is between 0 and 1. Because of product, close to 1 will be full pass, close to 0 no pass

LSTM constituents: Input gate

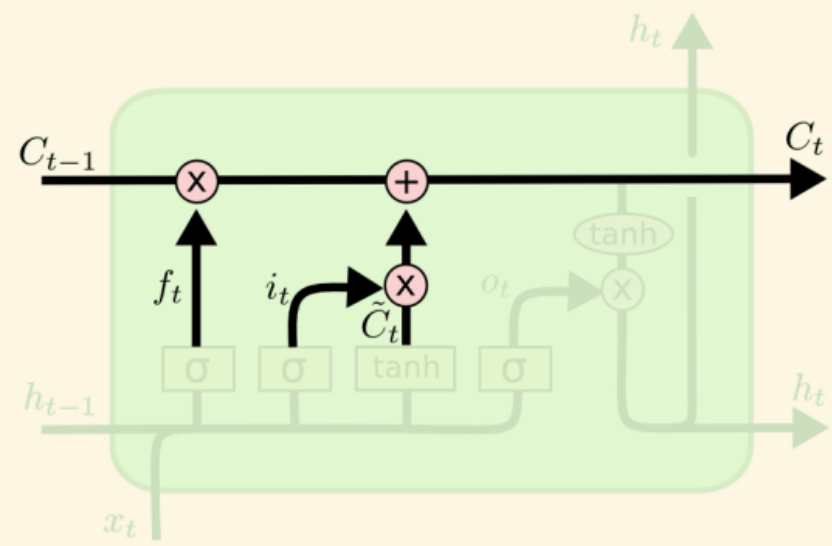


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

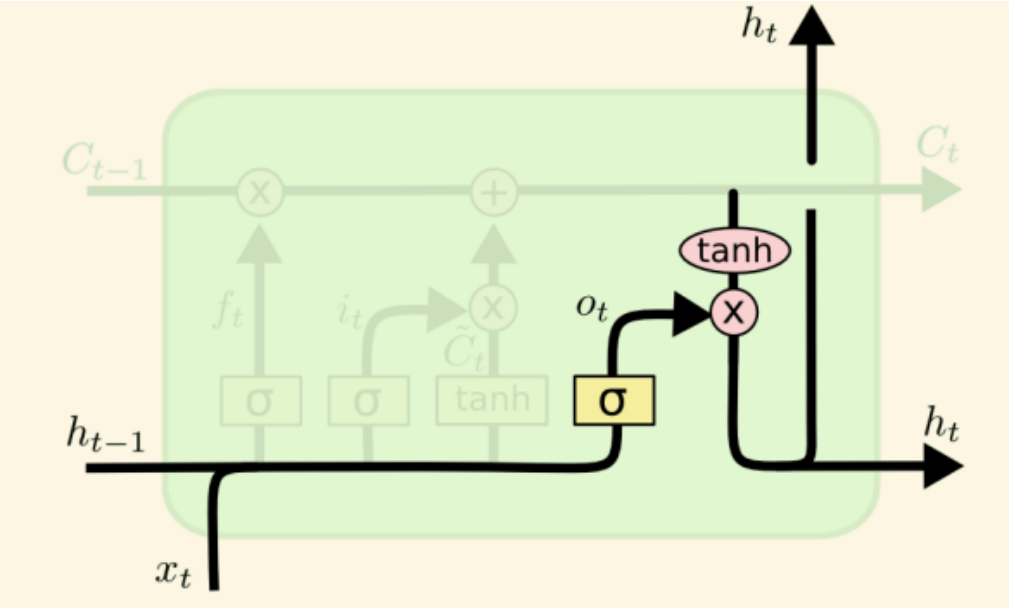
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

tanh produces a cell state vector; multiplied with input gate which again 0-1 controls what and how much input goes FOWARD

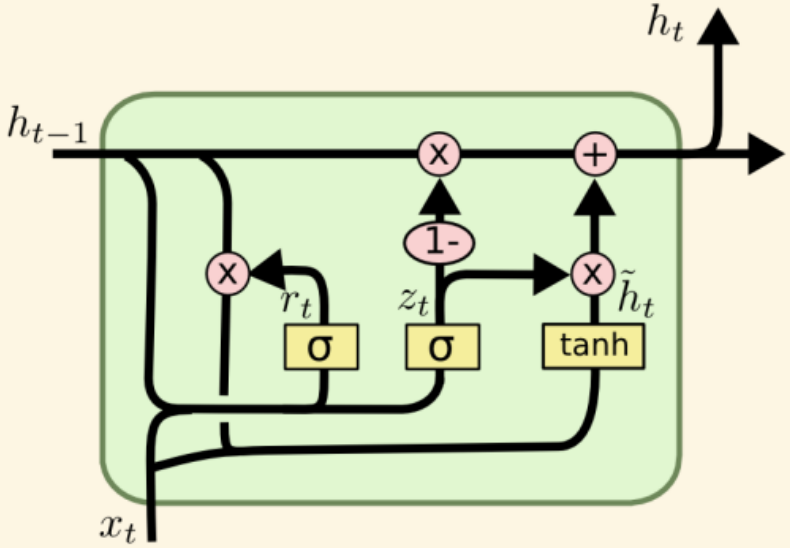
Cell state operation



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


h_t
C_{t-1}
C_t
tanh
f_t
i_t
$\tilde{C}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
x_t

Finally



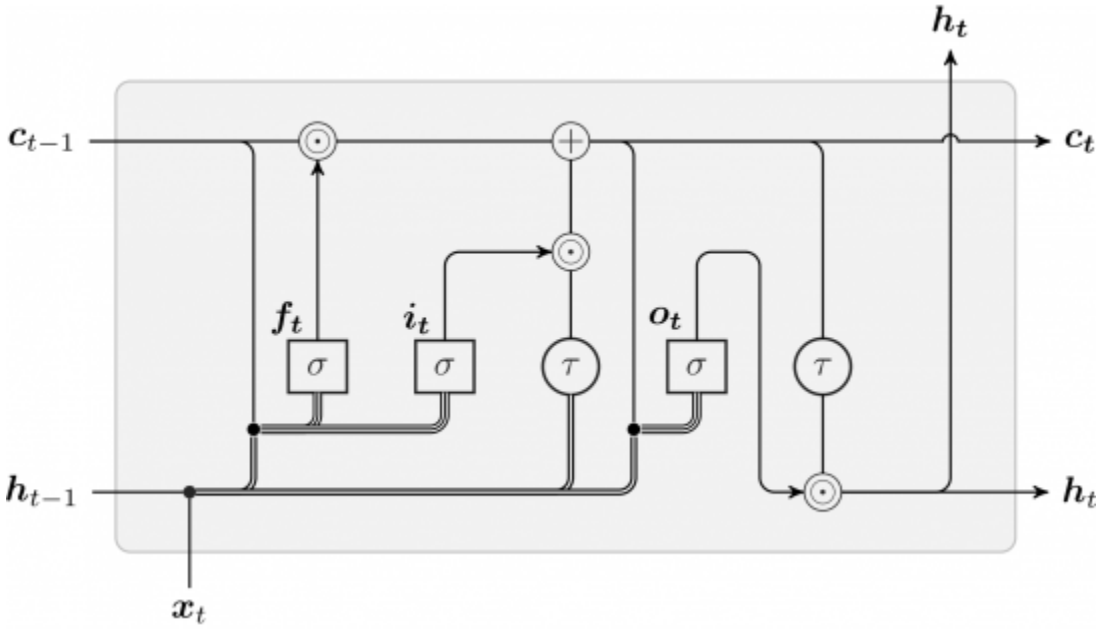
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

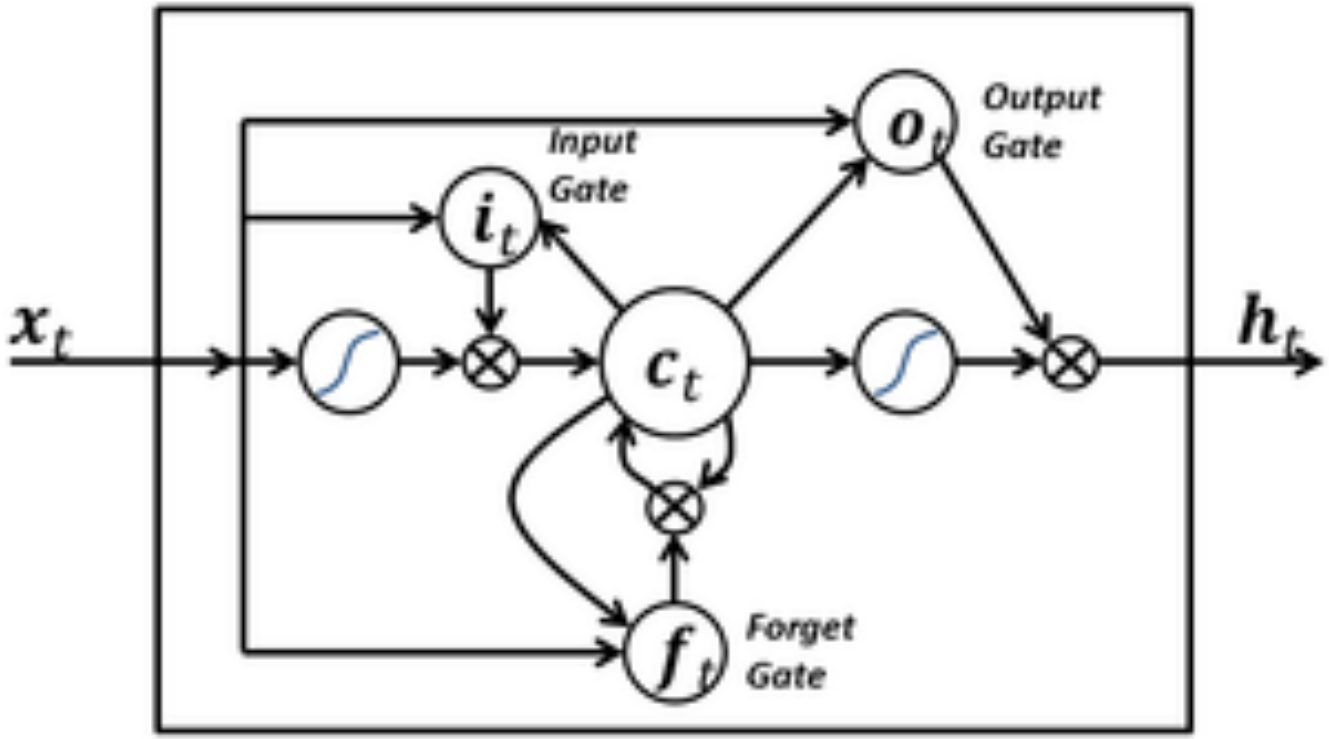
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Better picture (the one we started with)



Another picture



Training of LSTM

Many layers and gates

- Though complex, in principle possible to train
- Gates are also *sigmoid* or *tanh* networks

- Remember the FUNDAMENTAL backpropagation rule

General Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \quad \text{for hidden layers}$$

LSTM tools

- *Tensorflow, Ocropus, RNNlib etc.*
- Tools do everything internally
- Still insights and concepts are inevitable

LSTM applications

Many applications

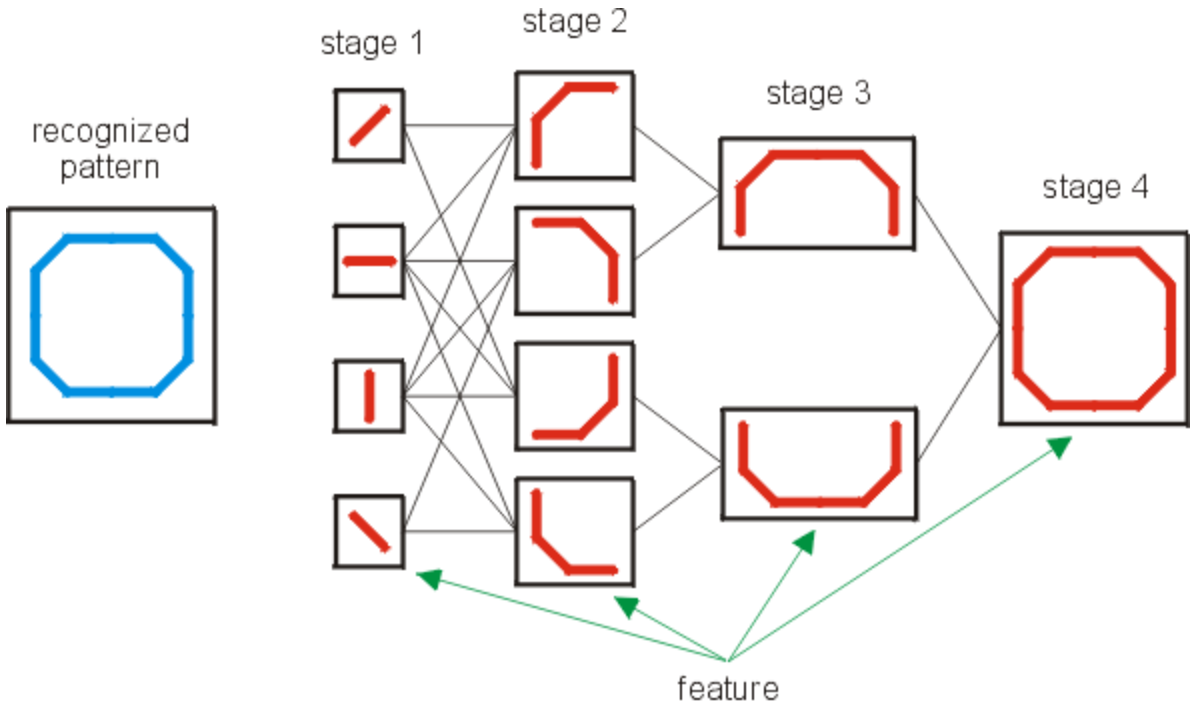
- Language modeling (The tensorflow tutorial on PTB is a good place to start Recurrent Neural Networks) character and word level LSTM's are used
- Machine Translation also known as sequence to sequence learning (https://arxiv.org/pdf/1409.3215.pdf)
- Image captioning (with and without attention, https://arxiv.org/pdf/1411.4555v...)
- Hand writing generation (http://arxiv.org/pdf/1308.0850v5...)
- Image generation using attention models (https://arxiv.org/pdf/1502.04623...)
- Question answering (http://www.aclweb.org/anthology/...)
- Video to text (https://arxiv.org/pdf/1505.00487...)

Convolutional Neural Network (CNN)

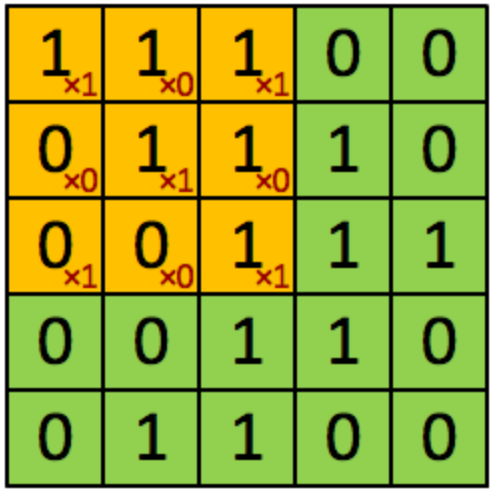
CNN= feedforward + recurrent!

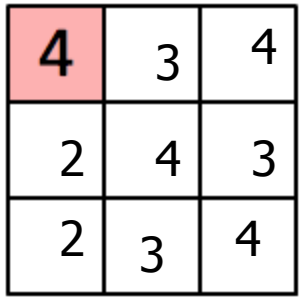
- Whatever we learnt so far in FF-BP is useful to understand CNN
- So also is the case with RNN (and LSTM)
- Input divided into regions and fed forward
- Window slides over the input: input changes, but 'filter' parameters remain same
- That is RNN

Remember Neocognitron



Convolution





- Matrix on the left represents an black and white image.
- Each entry corresponds to one pixel, 0 for black and 1 for white (typically it's between 0 and 255 for grayscale images).
- The sliding window is called a *kernel, filter,* or *feature detector.*
- Here we use a 3×3 filter, multiply its values element-wise with the original matrix, then sum them up.
- To get the full convolution we do this for each element by sliding the filter over the whole matrix.

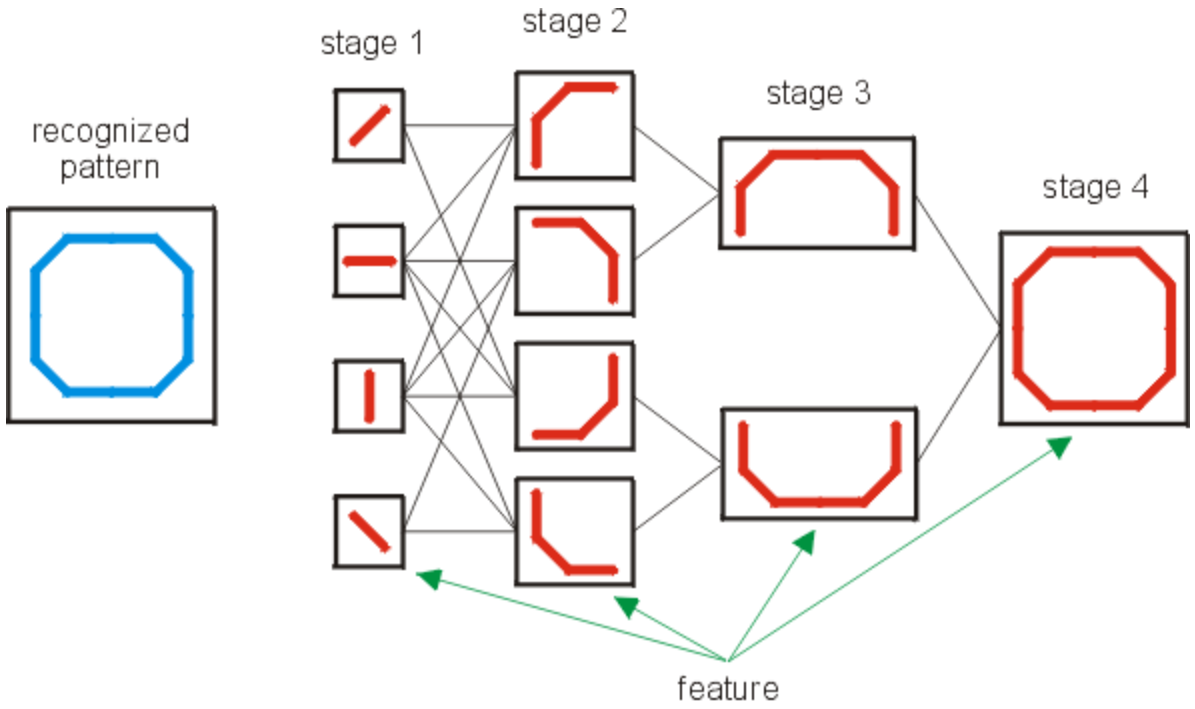
CNN architecture

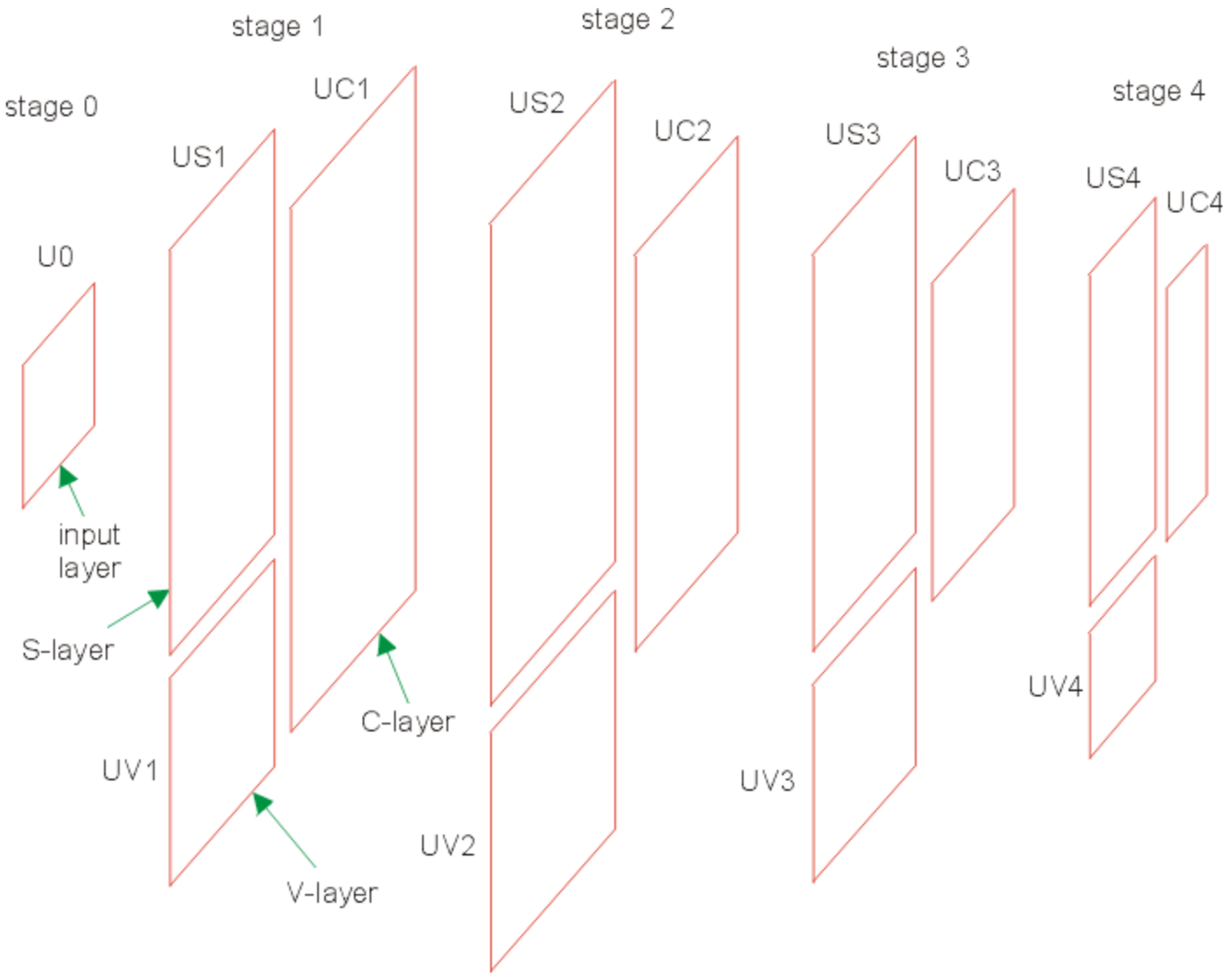
- Several layers of convolution with *tanh* or *ReLU* applied to the results
- In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That's also called a fully connected layer, or affine layer.
- In CNNs we use convolutions over the input layer to compute the output.
- This results in local connections, where each region of the input is connected to a neuron in the output

Learning in CNN

- **Automatically learns the values of its filters**
- For example, in Image Classification learn to
  - detect edges from raw pixels in the first layer,
  - then use the edges to detect simple shapes in the second layer,
  - and then use these shapes to deter higher-level features, such as facial shapes in higher layers.
  - The last layer is then a classifier that uses these high-level features.

Remember Neocognitron

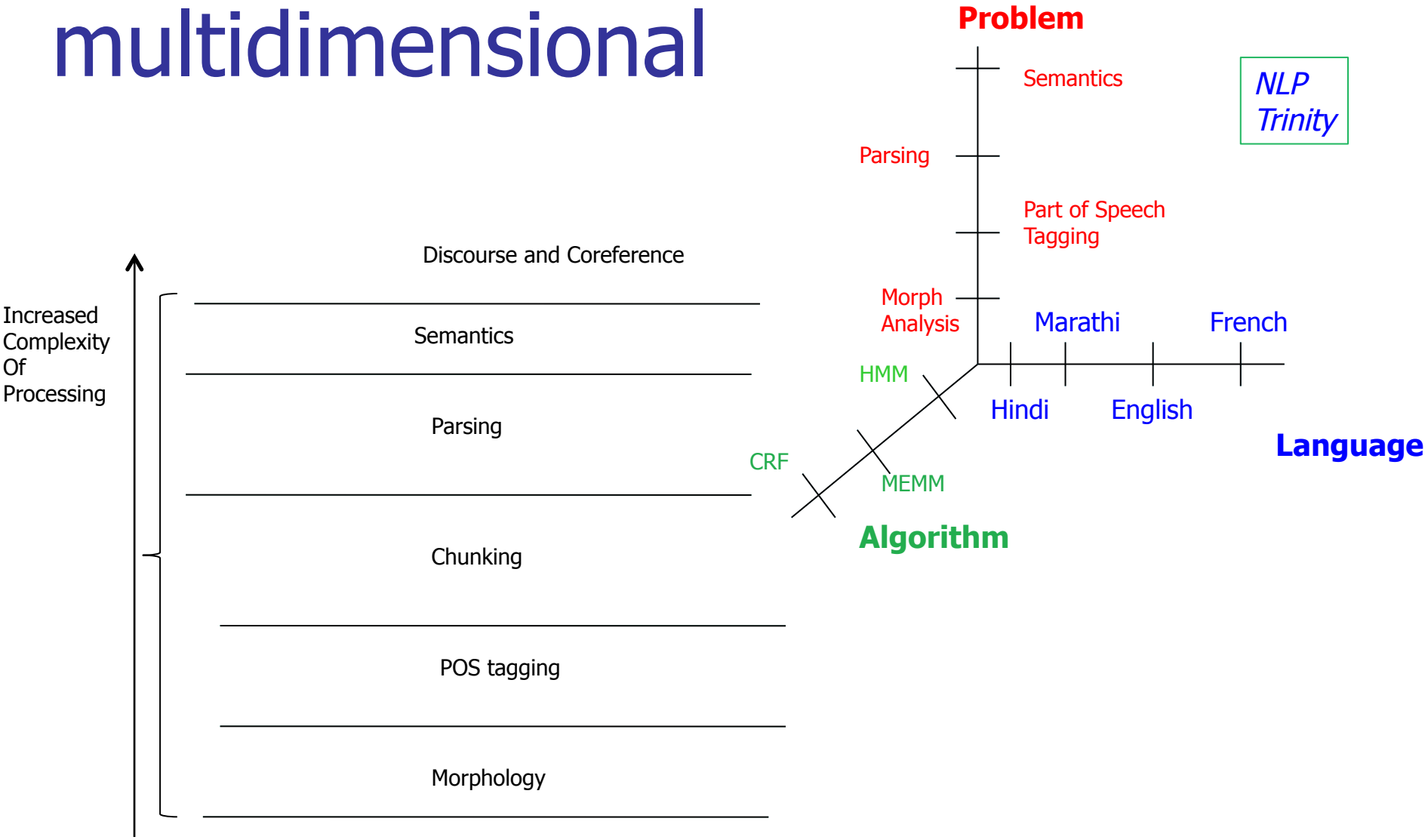



stage 0
stage 1
stage 2
stage 3
stage 4
U0
US1
UC1
US2
UC2
US3
UC3
US4
UC4
input layer
S-layer
C-layer
UV1
V-layer
UV2
UV3
UV4

What about NLP and CNN?

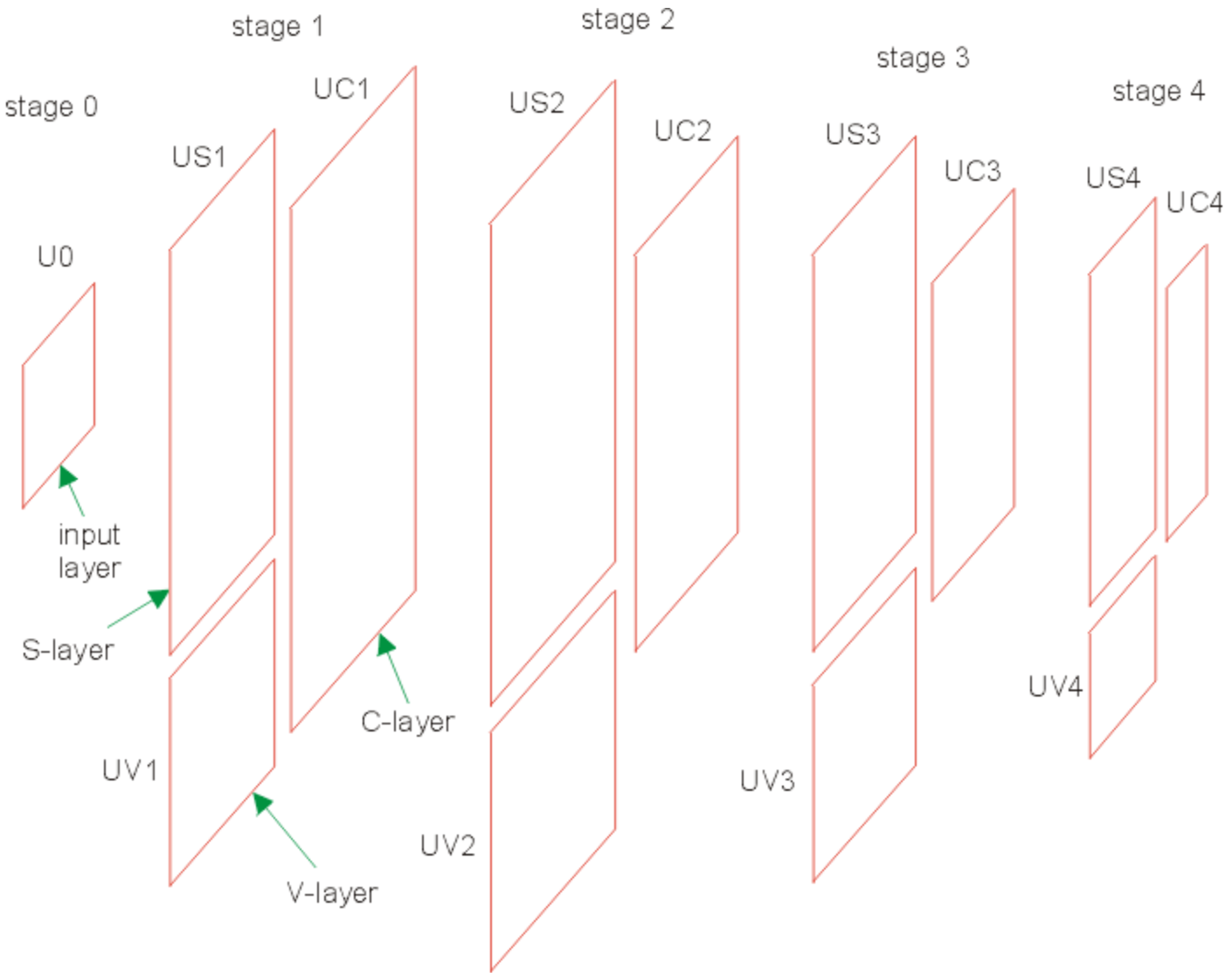
- Natural Match!
- NLP happens in layers

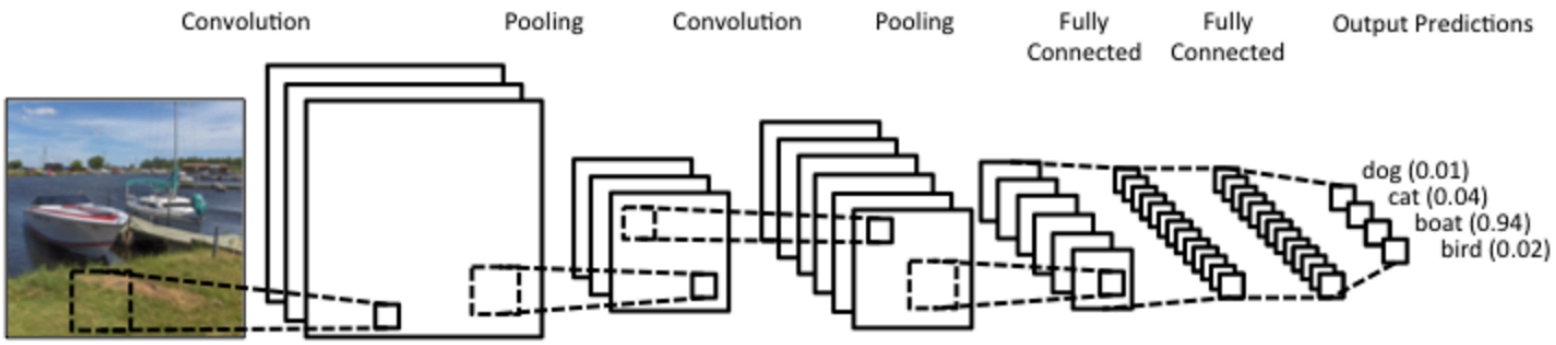
NLP: multilayered, multidimensional



NLP layers and CNN

- Morph layer →
- POS layer →
- Parse layer →
- Semantics layer


stage 0
stage 1
stage 2
stage 3
stage 4
U0
US1
UC1
US2
UC2
US3
UC3
US4
UC4
input
layer
S-layer
C-layer
UV1
V-layer
UV2
UV3
UV4



http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Pooling

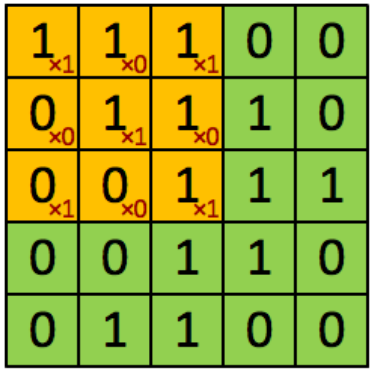
- Gives invariance in translation, rotation and scaling
- Important for image recognition
- Role in NLP?

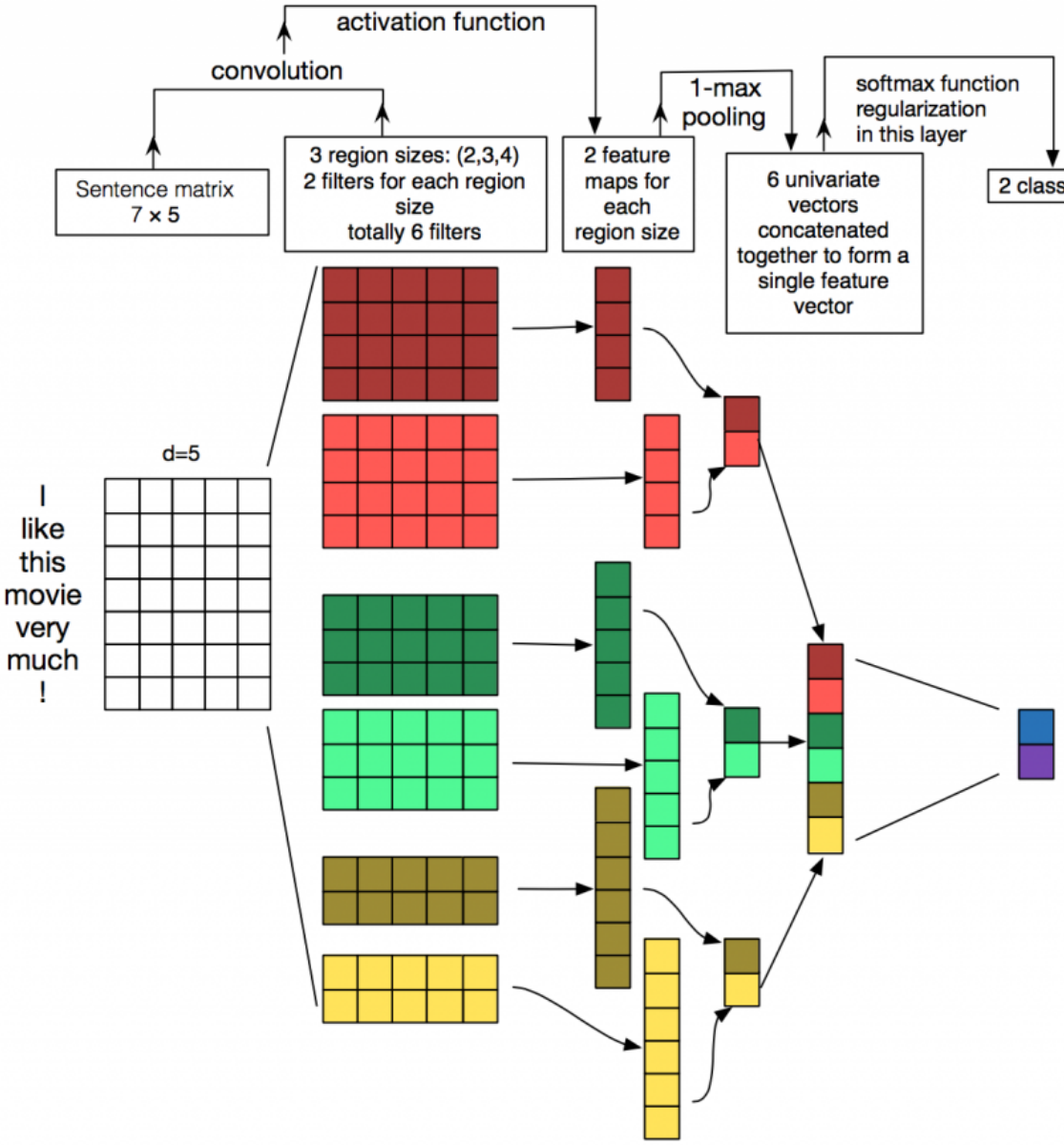
Input matrix for CNN: NLP

- "image" for NLP ←→ word vectors
- in the rows

- For a 10 word sentence using a 100-dimensional Embedding,

- we would have a 10×100 matrix as our input





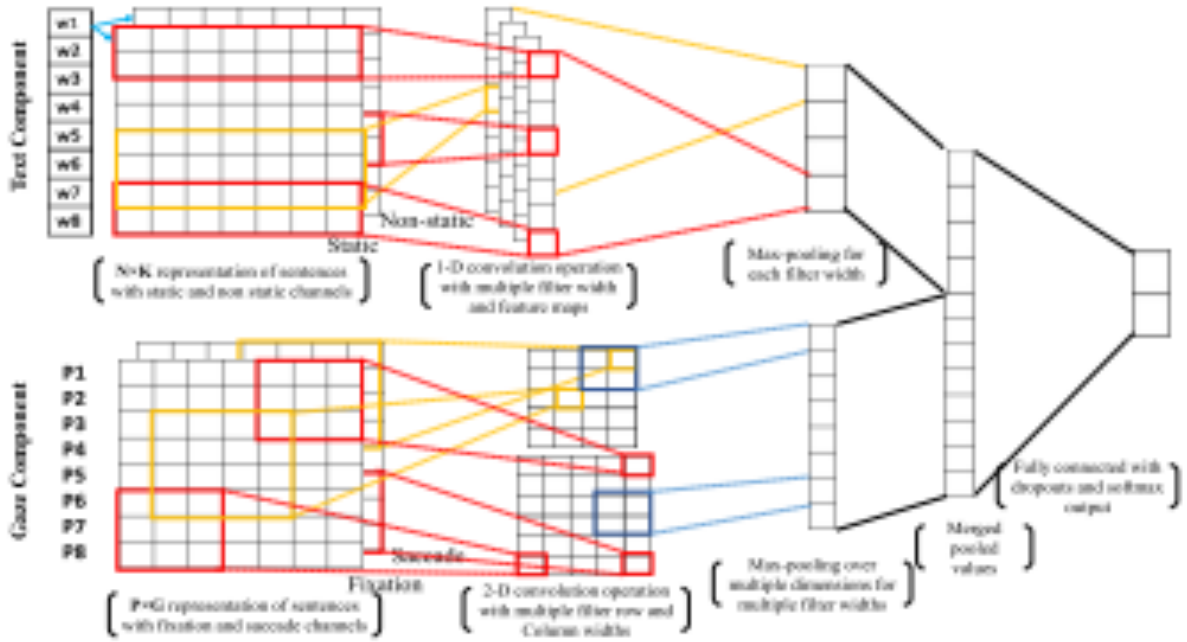
Credit: Denny Britz

CNN for NLP

CNN Hyper parameters

- Narrow width vs. wide width
- Stride size
- Pooling layers
- Channels

Abhijit Mishra, Kuntal Dey and Pushpak Bhattacharyya,
Learning Cognitive Features from Gaze Data for Sentiment and Sarcasm Classification Using Convolutional Neural Network, **ACL 2017**, Vancouver, Canada, July 30-August 4, 2017.



Learning Cognitive Features from Gaze Data for Sentiment and Sarcasm Classification

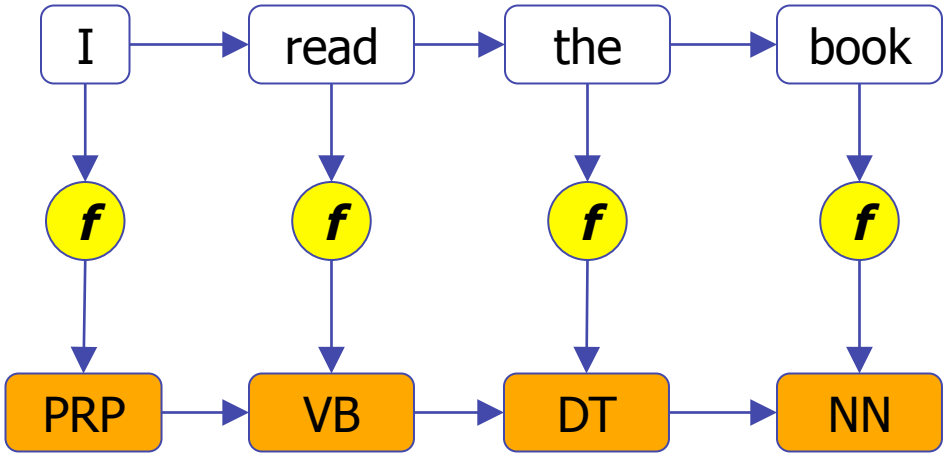
- In complex classification tasks like sentiment analysis and sarcasm detection, even the extraction and choice of features should be delegated to the learning system
- CNN learns features from both gaze and text and uses them to classify the input text

Illustration of attention: DL-POS

Acknowledgement: Anoop Kunchukuttan, PhD Scholar, IIT Bombay

So far we are seen POS tagging as a sequence labelling task

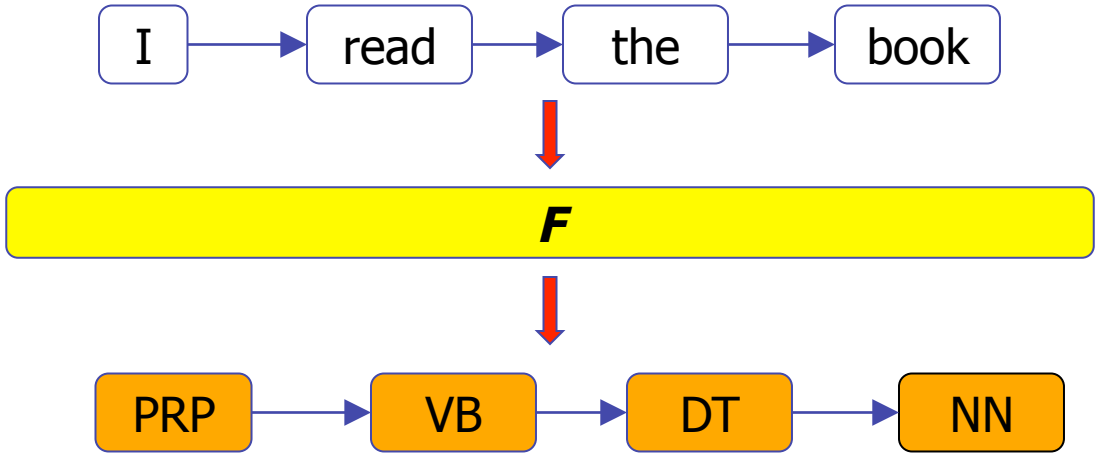
*For every element, predict the tag/label (using function **f**)*



- Length of output sequence is same as input sequence
- Prediction of tag at time *t* can use only the words seen till time *t*

We can also look at POS tagging as a sequence to sequence transformation problem

*Read the entire sequence and predict the output sequence (using function **F**)*



- Length of output sequence need not be the same as input sequence
- Prediction at any time step *t* has access to the entire input
- A more general framework than sequence labelling

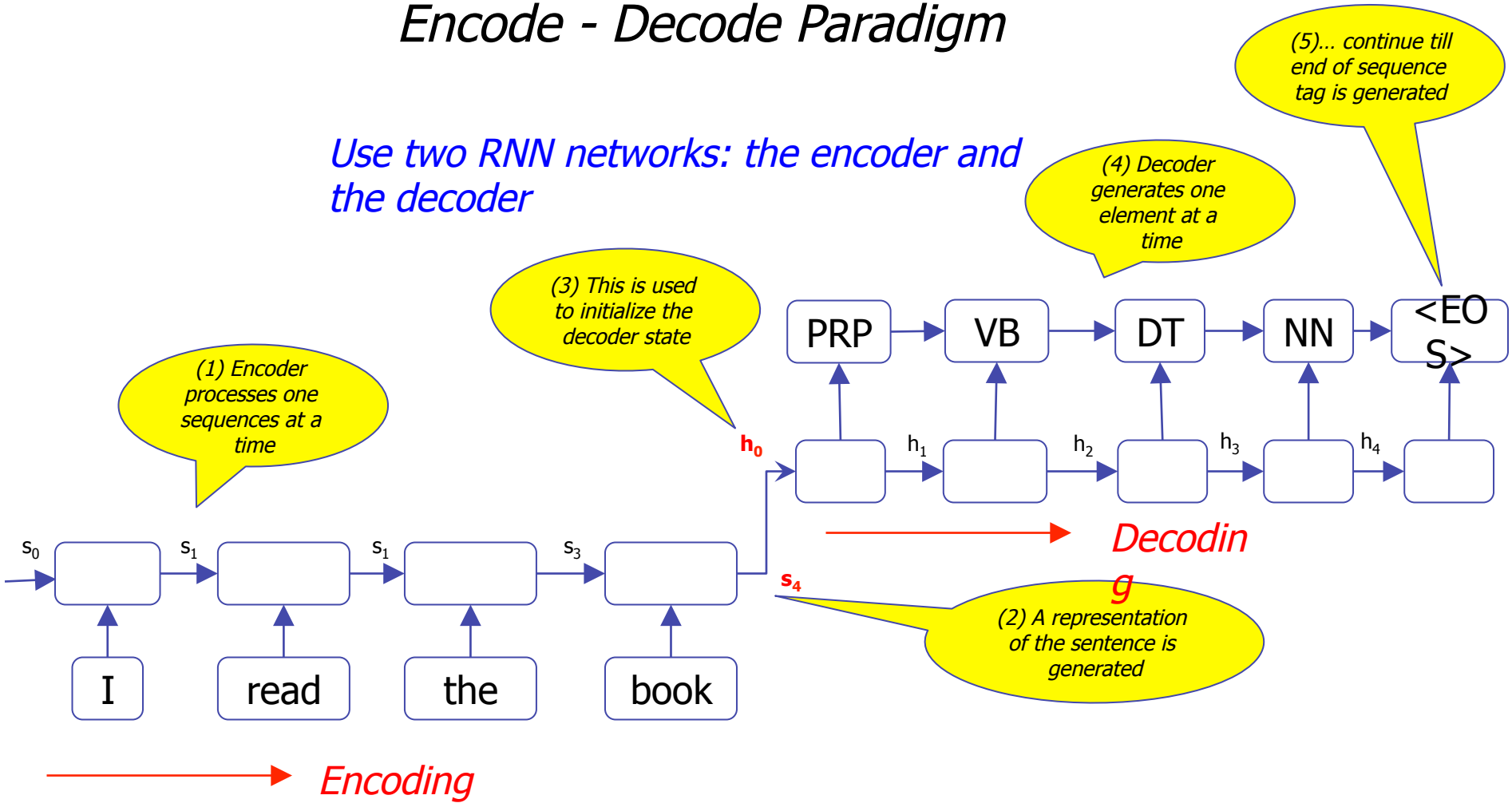
Sequence to Sequence transformation is a more general framework than sequence labelling

- Many other problems can be expressed as sequence to sequence transformation
  - *e.g. machine translation, summarization, question answering, dialog*
- Adds more capabilities which can be useful for problems like MT:
  - many → many mappings: insertion/deletion to words, one-one mappings
  - non-monotone mappings: reordering of words
- For POS tagging, these capabilites are not required

How does a sequence to sequence model work? Let's see two paradigms

Encode - Decode Paradigm

Use two RNN networks: the encoder and the decoder



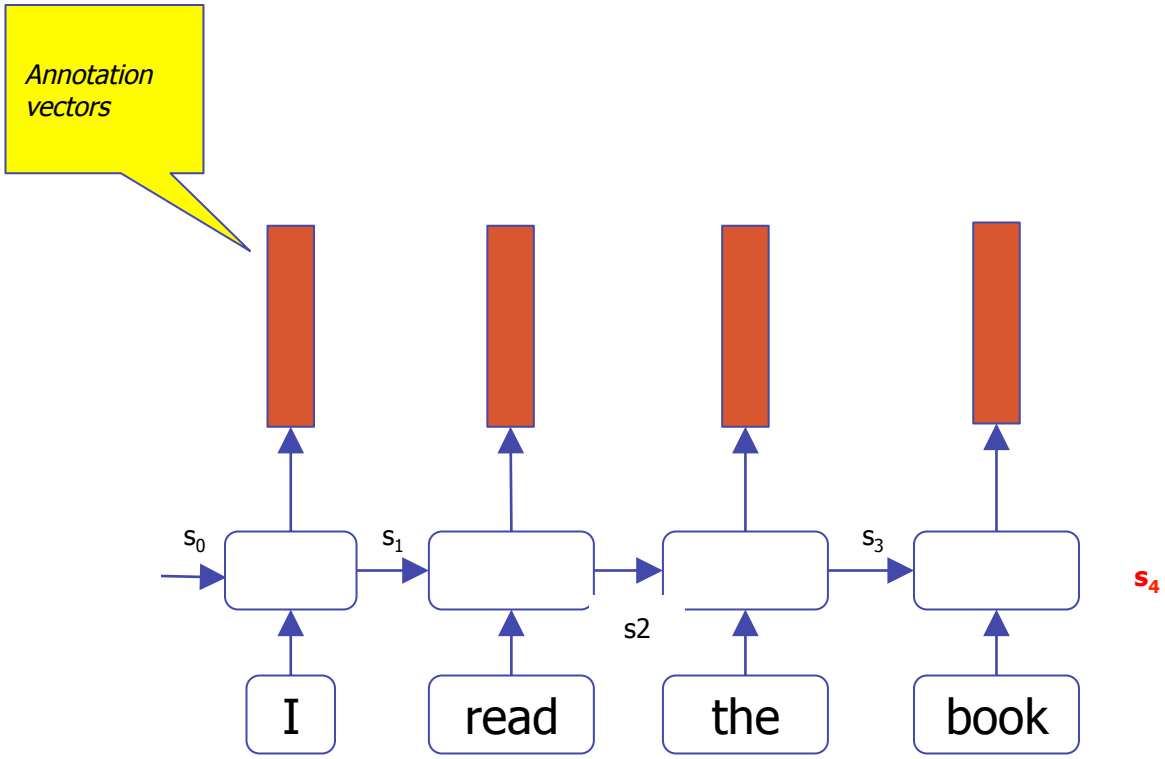
This approach reduces the entire sentence representation to a single vector

Two problems with this design choice:

- This is not sufficient to represent to capture all the syntactic and semantic complexities of a sentence
  - *Solution: Use a richer representation for the sentences*
- Problem of capturing long term dependencies: The decoder RNN will not be able to able to make use of source sentence representation after a few time steps
  - *Solution: Make source sentence information when making the next prediction*
  - *Even better, make **RELEVANT** source sentence information available*

These solutions motivate the next paradigm

Encode - Attend - Decode Paradigm



Represent the source sentence by the **set of output vectors** from the encoder

Each output vector at time *t* is a contextual representation of the input at time *t*

Let's call these encoder output vectors ***annotation vectors***

How should the decoder use the set of annotation vectors while predicting the next character?

Key Insight:

(1)Not all annotation vectors are equally important for prediction of the next element

(2)The annotation vector to use next depends on what has been generated so far by the decoder

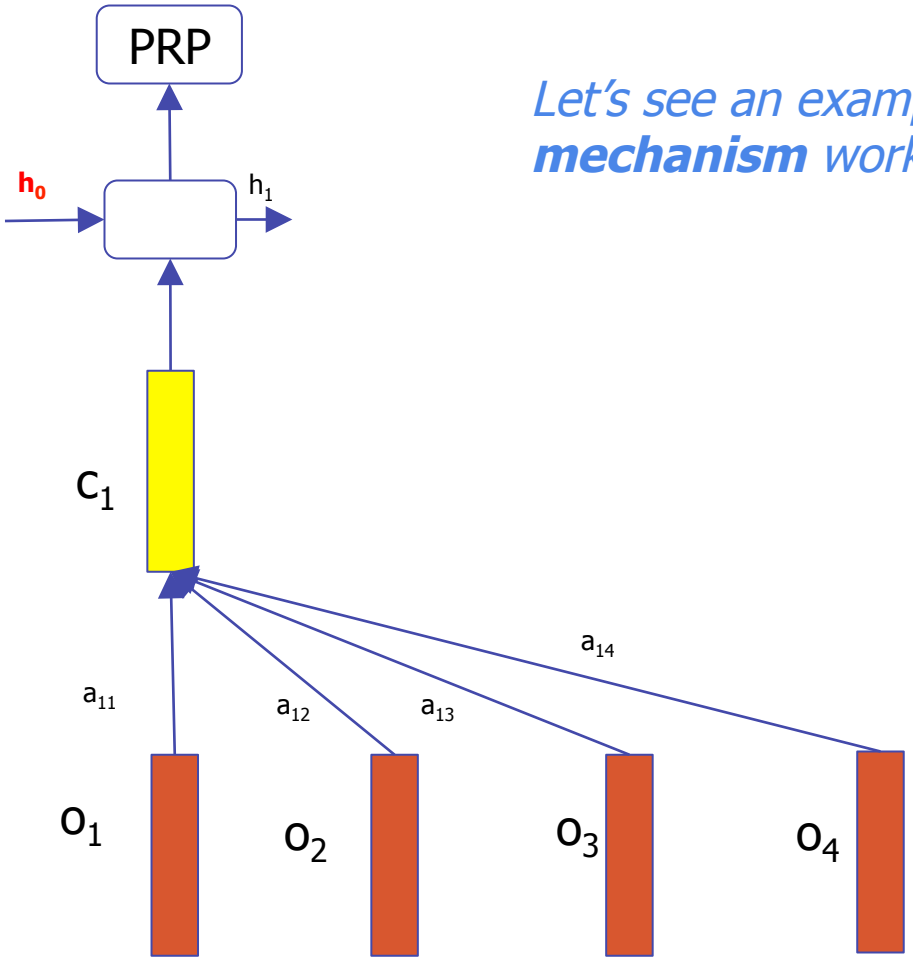
eg. To generate the 3rd POS tag, the 3rd annotation vector (hence 3rd word) is most important

One way to achieve this:
Take a weighted average of the annotation vectors, with more weight to annotation vectors which need more **focus or attention**

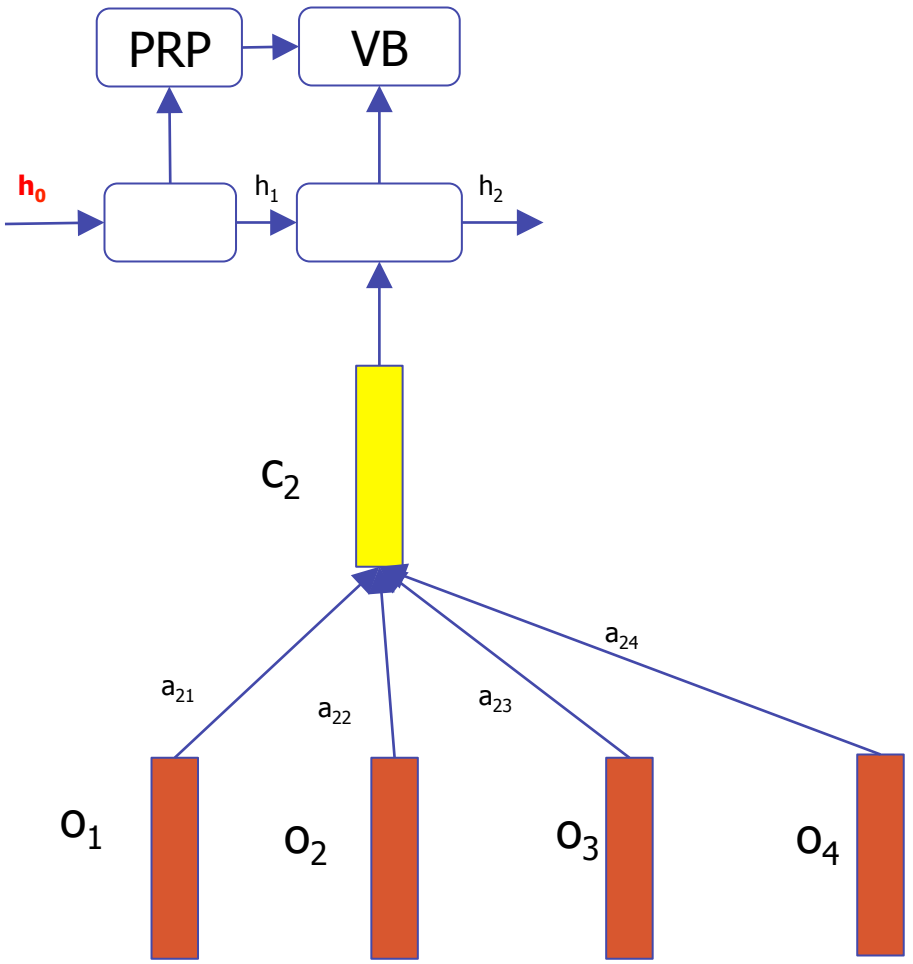
This averaged ***context vector*** is an input to the decoder

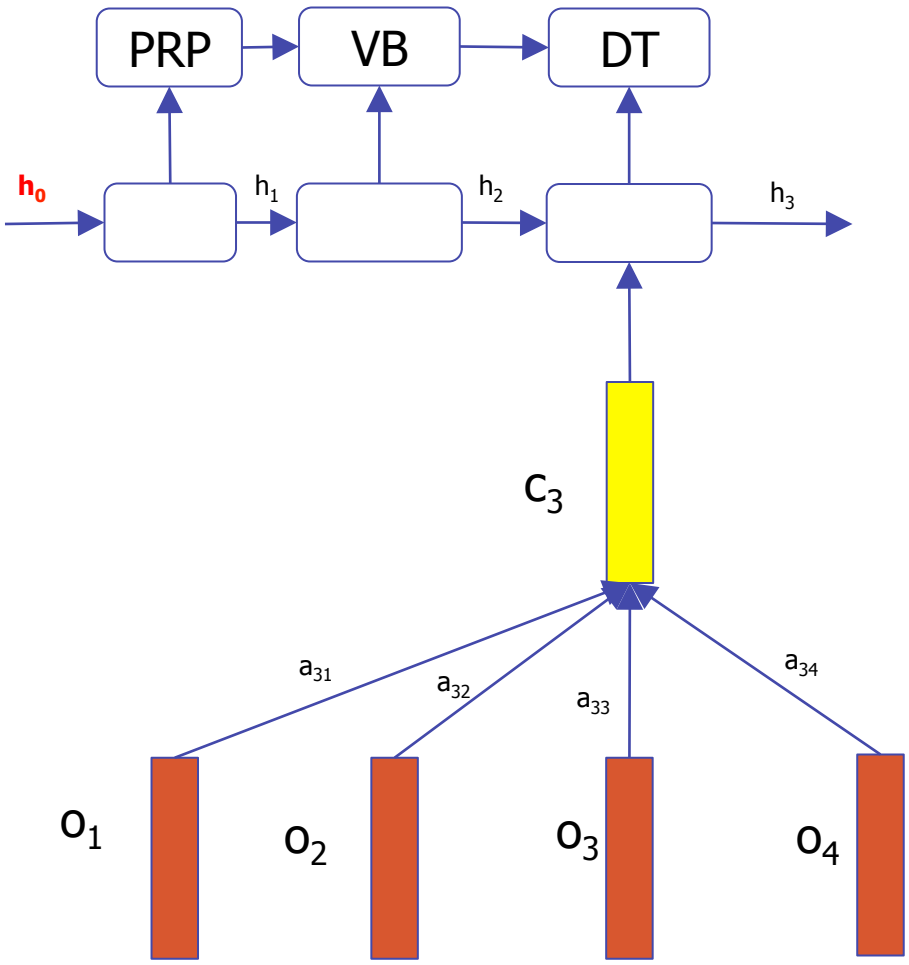
$$c_i = \sum_{j=1}^{n} a_{ij} o_j$$

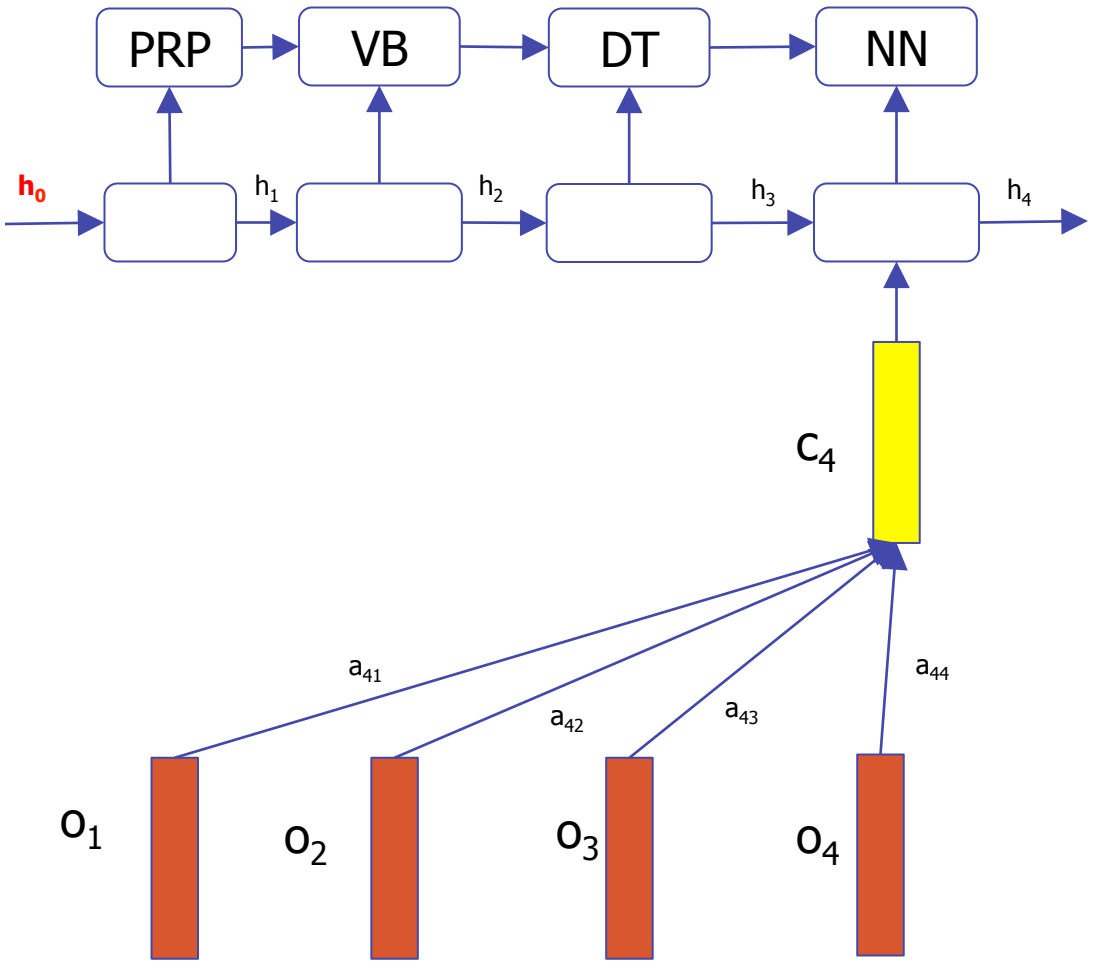
For generation of i^{th} output character:
c_i : context vector
a_{ij} : annotation weight for the j^{th} annotation vector
o_j: j^{th} annotation vector

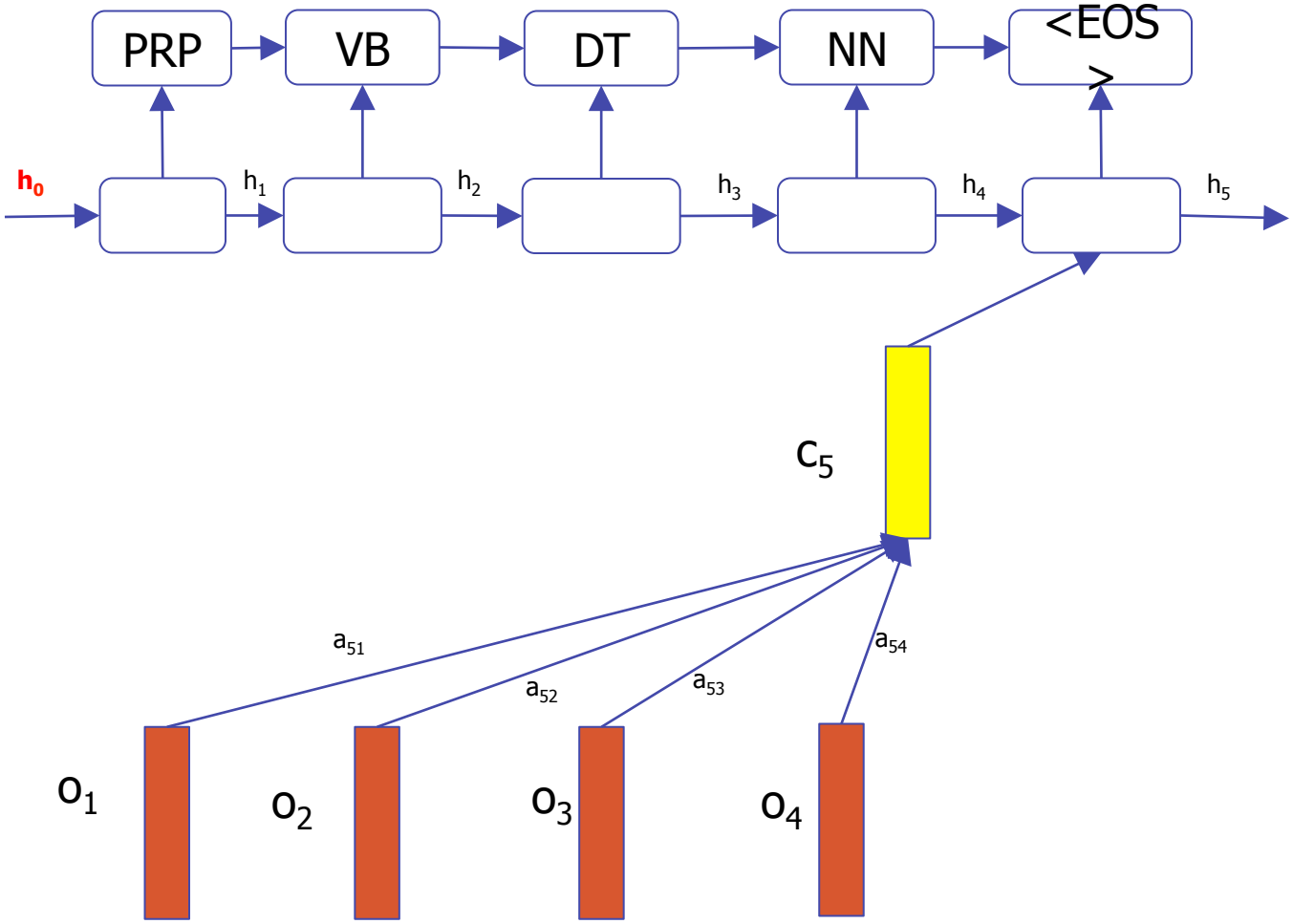


*Let's see an example of how the **attention mechanism** works*


PRP
VB
h0
h1
h2
C2
a21
a22
a23
a24
O1
O2
O3
O4


PRP
VB
DT
h0
h1
h2
h3
C3
a31
a32
a33
a34
O1
O2
O3
O4


PRP
VB
DT
NN
h0
h1
h2
h3
h4
C4
a41
a42
a43
a44
O1
O2
O3
O4


PRP
VB
DT
NN
<EOS>
h0
h1
h2
h3
h4
h5
C5
a51
a52
a53
a54
O1
O2
O3
O4

But we do not know the attention weights?
How do we find them?

Let the training data help you decide!!

Idea: Pick the attention weights that maximize the POS tagging accuracy

(more precisely, decrease training data loss)

Have an attention function that predicts the attention weights:

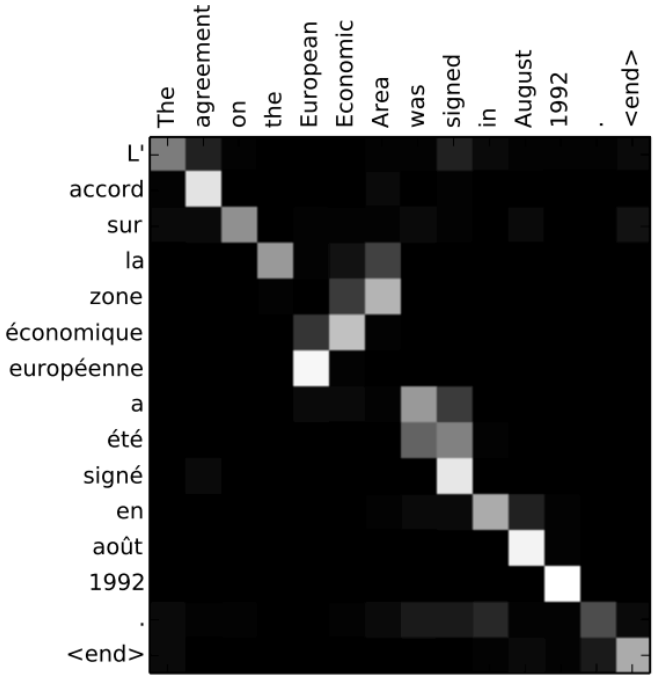
$$a_{ij} = \mathbf{A}(o_j, h_i; \mathbf{o})$$

A could be implemented as a feedforward network which is a component of the overall network

Then training the attention network with the rest of the network ensures that the attention weights are learnt to minimize the translation loss

OK, but do the attention weights actually show focus on certain parts?

Here is an example of how attention weights represent a soft alignment for machine translation



Let's go back to the encoder. What type of encoder cell should we use there?

- Basic RNN: models sequence history by maintaining state information
  - But, cannot model long range dependencies
- LSTM: can model history and is better at handling long range dependencies

The RNN units model only the sequence seen so far, cannot see the sequence ahead

- Can use a bidirectional RNN/LSTM
- This is just 2 LSTM encoders run from opposite ends of the sequence and resulting output vectors are composed

Both types of RNN units process the sequence sequentially, hence parallelism is limited

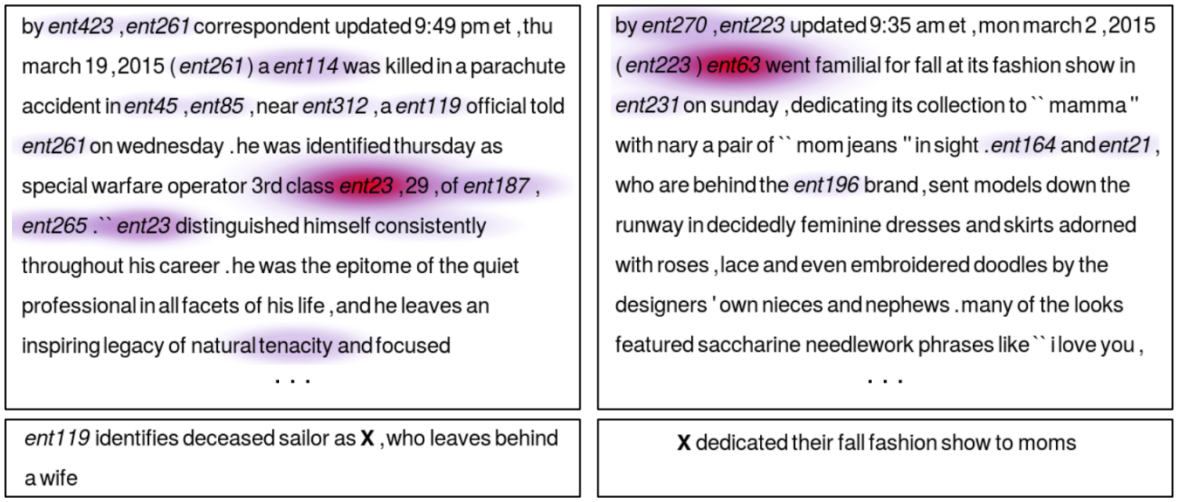
Alternatively, we can use a **CNN**

- Can operate on a sequence in parallel
- However, cannot model entire sequence history
- Model only a short local context. This may be sufficient for some applications or deep CNN layers can overcome the problem

Other applications of Attention

Teaching Machines to Read and Comprehend

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, Phil Blunsom, arxiv, 2015

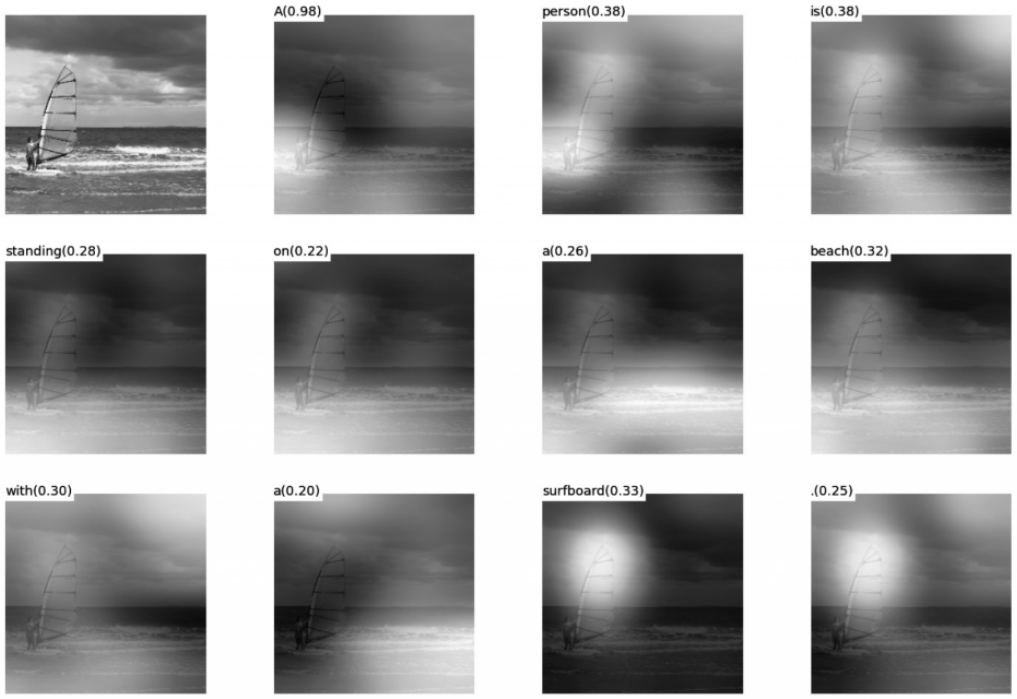


Used RNN to read a text, read a (synthetically generated) question, and then produce an answer.

By visualizing the attention matrix we can see where the networks "looks" while it tries to find the answer to the question

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio



(b) A person is standing on a beach with a surfboard.

Use a Convolutional Neural Network to "encode" the image, and a Recurrent Neural Network with attention mechanisms to generate a description.

By visualizing the attention weights, we interpret what the model is looking at while generating a word

Hands-on sessions that were done, 12-16 June, 2017

DL Sessions @ LG

Prof. Pushpak Bhattacharyya
Dr. Asif Iqbal, Dr. Sriparna
Rudramurthy, Kevin Patel

Outline

- Day 1
- Day 2
- Day 3
- Day 4

Day 1 (Theory)

- Theory: Review of feed forward neural networks and beginning of recurrent neural networks

Day 1 (Practical)

- SVMs on artificially generated datasets
  - Classifying linearly separable data with linear SVM
  - Failure to classify non-linear data (concentric circles) with linear SVM
    - Using polynomial kernel for successful classification
  - Visualizing support vectors
  - Effect of regularization parameter on number of support vectors
- Tensorflow basics
  - Along with basics of Tensorboard

Day 2 (Theory)

- Recurrent Neural Network continued

Day 2 (Practical)

- Binary addition using Feed Forward Network
  - Coded by LG Team
  - Experienced the inability of FFNs to handle arbitrary length sequences
- Binary addition using Recurrent Neural Network
  - Understanding the RNN API in tensorflow
- Sentiment Analysis Example
  - Showing shortcomings of FFNs
  - Hinting how RNNs and CNNs may help

Day 3 (Theory)

- A discussion on word embeddings
  - Focus on intuition, usage and evaluation
- A discussion on CNNs
  - Focus on their ability to extract features and achieve positional invariance

Day 3 (Practical)

- Word2vec trained on SMS corpus
- Nearest neighbors of *wife*
  - *Unbearable*
  - *Torture*
  - *BP* (Blood Pressure)
- Dimensions - (20, 100, 500) – did not change the ranking

Day 3 (Practical) (contd.)

- Sentiment Analysis Example using CNN
  - Given examples of the form 'the movie was very good', 'the movie was very awful', the network learned 'very good' and 'very awful' to be important features
  - Was able to correctly classify 'very good was the movie'
    - FFNs failed to do so

Day 3 (Practical) (contd.)

- SMS Classification using CNN
  - LG team asked to compute list of features - like ‘very good’ in the previous case – for different classes in their dataset
  - Program taking too long on CPUs. To be resumed the next day

Day 4 (Theory)

- LSTMs motivated by example
- Working of LSTMs covered

Day 4 (Practical)

- Previous day's CNN runs at LG incomplete
  - Crashed / far away from completion
  - Lack of GPUs are a limitation
- Analysis on model trained by Rudra
  - *My wife is beautiful* -> Emergency/Police
  - *His wife is beautiful* -> Emergency/Health
  - *Someone's wife is beautiful* -> TODO

Day 4 (Practical) (contd.)

- Analysis revealed shortcomings of data
  - *Wife*'s occurrence more than 40% in Police category
  - 0 times in General category
  - Wrong priors thus introduced in the model
  - Need more data for General category to address this
- Suggested multi-step classification
  - First, classify whether the SMS belongs to general category or not
  - If not general, then classify into subclasses such as Police, Fire, *etc.*

Day 4 (Practical) (contd.)

- Suggested gathering dummy data for General category
  - Clean current SMS corpus
  - POS tag the data
  - Form a vocab of content words
  - For each word in vocab, extract K sentences from Wikipedia. Consider these sentences as General category sms

Day 4 (Practical) (contd.)

- Results of Rudra's models

Model	Accuracy
CNN	60%
RNN	65%
RNN + Max Pooling	66%

Thank You