

ICON 2017 TUTORIAL ON DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Presented By
Rudra Murthy V
Kevin Patel
Shad Akhtar

Under Direction Of
Dr. Pushpak Bhattacharyya

Department of Computer Science and Engineering
IIT Bombay
Department of Computer Science and Engineering
IIT Patna



Part 1: Basics

Introduction to Neural Networks

Deep Learning for NLP

Kevin Patel

ICON 2017

December 21, 2017

Overview

- 1 Motivation
- 2 Perceptron
- 3 Feed Forward Neural Networks
- 4 Deep Learning
- 5 Conclusion

Our brains are so awesome, that we cannot replicate their
computation

Our brains are so awesome, that we cannot replicate their
computation

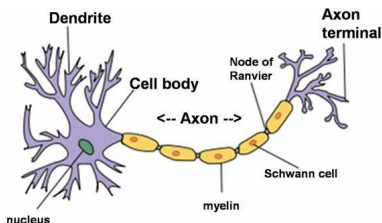
Our brains' capability is so limited, that we have failed to replicate
their computation

Purpose of Artificial Intelligence

- Human Like AI: An AI which functions like a human and has similar characteristics
- Beneficial AI: An AI which works in a fundamentally different manner, but gets the job done

The Human Brain

- Brain - a large network of neurons
- Neuron - a cell capable of receiving, processing and transmitting information via electric and chemical signals

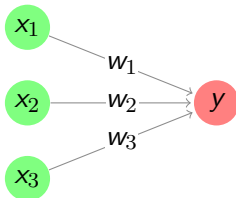


- Dendrites receive input
- Axon transmits output

Perceptron

- A simple artificial neuron Rosenblatt (1958)
- Input: one or more binary values x_i
- Output: single binary value y
- Output computed using weighted sum of inputs and a threshold
- Giving different weights to different features while making a decision

Perceptron (Contd.)



$$y = \begin{cases} 1, & \text{if } \sum w_i x_i > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Some Conventions

- Inputs also treated as neurons (no input, output is the actual value of the feature)
- Rewrite $\sum w_i x_i$ as $w \cdot x$
- Move threshold to the other side of the equation; Call it bias $b = -\text{threshold}$

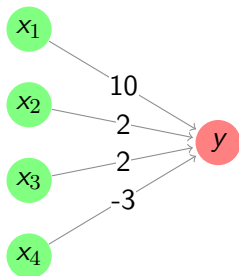
$$y = \begin{cases} 1, & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Bias

- An indication of how easy it is for a neuron to fire
- The higher the value of bias, the easier it is for the neuron to fire
- Consider it as a prior inclination towards some decision
 - The higher your initial inclination, the smaller the amount of extra push needed to finally make some decision

Perceptron Example

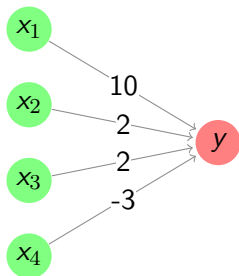
Should I go to lab?



- x_1 : My guide is here
- x_2 : Collaborators are in lab
- x_3 : The buses are running
- x_4 : Tasty tiffin in the mess
- b : My inclination towards going to the lab no matter what

Perceptron Example

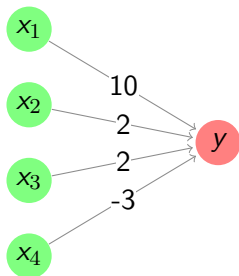
Should I go to lab?



- x_1 : My guide is here
- x_2 : Collaborators are in lab
- x_3 : The buses are running
- x_4 : Tasty tiffin in the mess
- b : My inclination towards going to the lab no matter what
- What if $b = -3$?

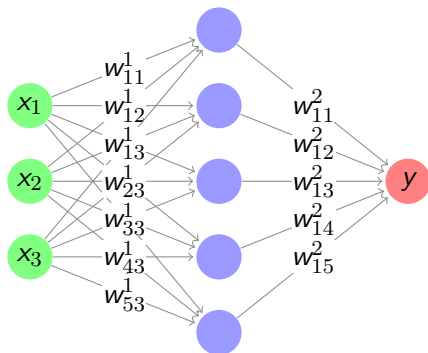
Perceptron Example

Should I go to lab?



- x_1 : My guide is here
- x_2 : Collaborators are in lab
- x_3 : The buses are running
- x_4 : Tasty tiffin in the mess
- b : My inclination towards going to the lab no matter what
- What if $b = -3$?
- What if $b = 1$?

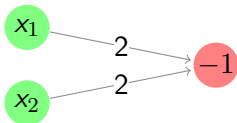
Network of Perceptrons



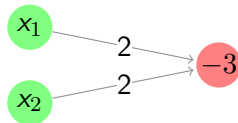
- Outputs of perceptrons fed into next layer
- Simpler decisions made at initial layers used as features for making complex decisions.

Perceptrons and Logic Gates

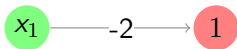
OR Gate



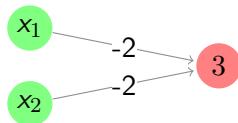
AND Gate



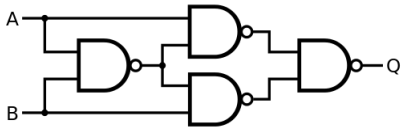
NOT Gate



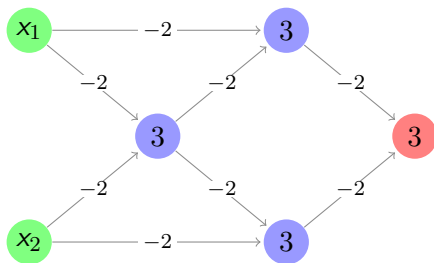
NAND Gate



Perceptrons and Logic Gates (contd.)



XOR Gate



Perceptrons and Logic Gates (contd.)

- NAND gates are universal gates
- Perceptrons can simulate NAND gates
- Therefore, Perceptrons are universal for computation
- Good News: Network of perceptrons is as capable as any other computing device
- Bad News: Is it just another fancy NAND gate?
 - Silver Lining: Learning algorithms can automatically figure out weights and biases, whereas we need to explicitly design circuits using NAND gates

Learning

- Simple weight update rule to learn parameters of a single perceptron
- Perceptron Convergence Theorem guarantees that learning will converge to a correct solution in case of linearly separable data.
- However, learning is difficult in case of network of perceptrons
 - Ideally, a learning process involves changing one of the input parameters by a small value, hoping that it will change the output by a small value.
 - For instance, in handwritten digit recognition, if the network is misclassifying 9 as 8 , then we want
 - Here, a small change in parameters of a single perceptron \Rightarrow flipped output \Rightarrow change behavior of entire network
 - Need some machinery such that gradual change in parameters lead to gradual change in output

Learning (contd.)

- If y is a function of x , then change in y *i.e.* Δy is related to change in x *i.e.* Δx as follows (Linear Approximation)

$$\Delta y \approx \frac{dy}{dx} \Delta x$$

- Example

$$f(x) = x^2$$

$$f'(x) = 2x$$

$$\begin{aligned} f(4.01) &\approx f(4) + f'(4)(4.01 - 4) \\ &= 16 + 2 \times 4 \times 0.01 \\ &= 16.08 \end{aligned}$$

Learning (contd.)

- For a fixed datapoint with two features (x_1, x_2) , the change in output of the perceptron depends on the corresponding changes in weights w_1 and w_2 and the bias b
- Thus, change in y - Δy is

$$\Delta y \approx \frac{\partial y}{\partial w_1} \Delta w_1 + \frac{\partial y}{\partial w_2} \Delta w_2 + \frac{\partial y}{\partial b} \Delta b$$

- Partial derivative ill-defined in case of perceptron, which creates hurdle for learning

Sigmoid Neurons

- Another simple artificial neuron McCulloch and Pitts (1943)
- Input: one or more real values x_i
- Output: single real value y
- Output computed by applying sigmoid function σ on the weighted sum of inputs and bias

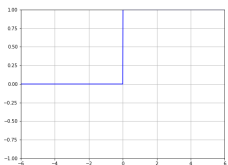
$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

- Decision making using sigmoid:
 - Given real valued output, use threshold
 - If $y > 0.5$, output 1, else 0

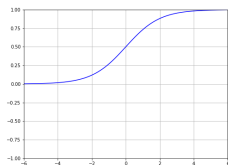
Sigmoid vs. Perceptron

- When $z = w.x + b \gg 0$
 - $e^{-z} \approx 0$
 - $\sigma(z) \approx 1$
 - Similar to perceptron producing 1 as output when z is large and positive
- When $z = w.x + b \ll 0$
 - $e^{-z} \approx \infty$
 - $\sigma(z) \approx 0$
 - Similar to perceptron producing 0 as output when z is large and negative
- Different primarily when absolute value of z is small.

Sigmoid vs. Perceptron (contd.)



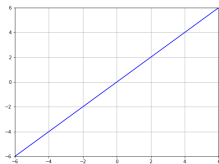
Step (Perceptron)



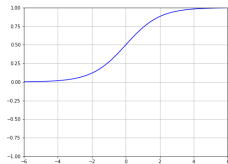
Sigmoid

- Sigmoid is continuous and differentiable over its domain
- Learning is possible via small changes in parameters and using linear approximation

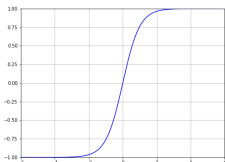
Activation Functions



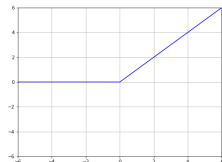
Linear



Sigmoid



Tanh

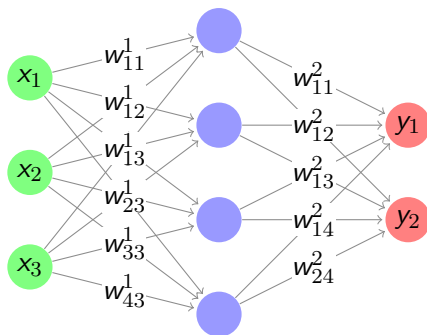


ReLU

Notations

- x : Input
- w_{ij}^l : weight from j^{th} neuron in $(l-1)^{\text{th}}$ layer to i^{th} neuron in l^{th} layer
- b_j^l : bias of the j^{th} neuron in the l^{th} layer
- z_j^l : $w^l \cdot a^{l-1} + b_j^l$
- a_j^l : $f(z_j^l)$

Neural Network Architecture



- For MNIST Digit recognition
 - Input Layer $28 \times 28 = 784$ neurons
 - Output Layer?

One-hot output vs. Binary encoded output

- Given 10 digits, we fixed 10 neurons in output layer
 - Why not 4 neurons, and generate binary representation of digits?
- The task is to observe features and learn to decide whether it is a particular digit
- Observing visual features and trying to predict, say, most significant bit, will be hard
 - Almost no correlation there

Feed Forward Computation

- Given input x
 - $z^1 = w^1 \cdot x + b^1$
 - $a^1 = \sigma(z^1)$
 - $z^l = w^l \cdot a^{l-1} + b^l$
 - $a^l = \sigma(z^l)$
 - a^L is the output, where L is the last layer
- Note that output contains real numbers (due to σ function)

Loss functions

- Consider a network with parameter setting P1

True			Predicted			Correct
0	1	0	0.3	0.4	0.3	yes
1	0	0	0.1	0.2	0.7	no
0	0	1	0.3	0.3	0.4	yes

- Number of correctly classified examples = $\frac{2}{3}$
- Classification error = $1 - \frac{2}{3} = \frac{1}{3}$
- Consider same network with parameter setting P2

True			Predicted			Correct
0	1	0	0.1	0.7	0.2	yes
1	0	0	0.3	0.4	0.3	no
0	0	1	0.1	0.2	0.7	yes

- Classification error still the same

Loss functions

- Mean Squared Error : $MSE = \frac{1}{M} \sum (y_i - t_i)^2$

True			Predicted			Correct
0	1	0	0.3	0.4	0.3	yes
1	0	0	0.1	0.2	0.7	no
0	0	1	0.3	0.3	0.4	yes

- Mean Squared Error = $(0.54 + 0.54 + 1.34)/3 = 0.81$

True			Predicted			Correct
0	1	0	0.1	0.7	0.2	yes
1	0	0	0.3	0.4	0.3	no
0	0	1	0.1	0.2	0.7	yes

- Mean Squared Error = $(0.14 + 0.14 + 0.74)/3 = 0.34$
- Indicates that second parameter setting is better

Loss functions

- Mean Cross Entropy

$$MCE = \frac{1}{M} \sum (-t_i \log y_i - (1 - t_i) \log(1 - y_i))$$

True			Predicted			Correct
0	1	0	0.3	0.4	0.3	yes
1	0	0	0.1	0.2	0.7	no
0	0	1	0.3	0.3	0.4	yes

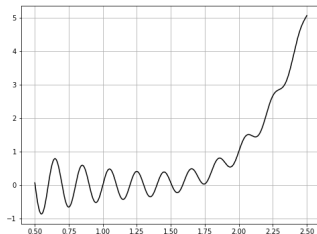
- Mean Cross Entropy = $-(\ln(0.4) + \ln(0.4) + \ln(0.1))/3 = 1.38$

True			Predicted			Correct
0	1	0	0.1	0.7	0.2	yes
1	0	0	0.3	0.4	0.3	no
0	0	1	0.1	0.2	0.7	yes

- Mean Cross Entropy = $-(\ln(0.7) + \ln(0.7) + \ln(0.3))/3 = 0.64$
- Indicates that second parameter setting is better

Minimizing Loss

- Consider a function C that depends on some parameter x as shown below:



- How to find the value of x for which C is minimum?
- Idea: Choose a random value for x , place an imaginary ball there. It will eventually lead to a valley

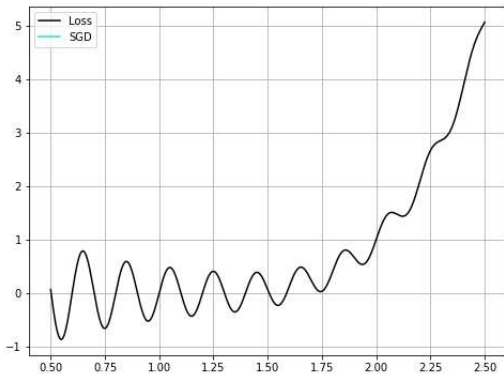
Gradient Descent

- Recall $\Delta C \approx \frac{dC}{dx} \cdot \Delta x$
- We want to change x such that C is reduced *i.e.* ΔC has to be always negative
- What if we choose $\Delta x = -\eta \frac{dC}{dx}$?

$$\begin{aligned}\Delta C &\approx \frac{dC}{dx} \cdot \Delta x \\ &= \frac{dC}{dx} \cdot \left(-\eta \frac{dC}{dx}\right) \\ &= -\eta \cdot \left(\frac{dC}{dx}\right)^2 \\ &\leq 0\end{aligned}$$

- Gradient Descent: $x_{t+1} = x_t - \eta \frac{dC}{dx}$

Gradient Descent



Back Propagation

- Effective use shown in Williams et al. (1986)
- Every neuron taking part in the decision
- Every neuron shares the blame for error
- Decision made by a neuron dependent on its weights and biases
- Thus error is caused due to these weights and biases
- Need to change weights and biases such that overall error is reduced
 - Can use gradient descent here
 - For a weight w_{ij}^k , the weight update will be

$$w_{ij}^k \leftarrow w_{ij}^k - \eta \frac{\partial C}{\partial w_{ij}^k}$$

Back Propagation (contd.)

Will use calculus chain rule to obtain the partial derivatives

- 1 First find error for each neuron on the last layer

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

- 2 Then find error for each neuron on the interior neurons

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

- 3 Update weights and biases using following gradients:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Vanishing Gradient Problem

- Observed by Hochreiter (1998)
- Important point: the $\sigma'(z^l)$ term in the previous steps
- Derivative of the activation function
- Will be multiplied at each layer during back propagation
- Example: 3 layer network

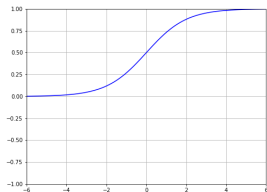
$$\delta^4 = A.\sigma'()$$

$$\delta^3 = X.\delta^4.\sigma'() = X.A.\sigma'().\sigma'()$$

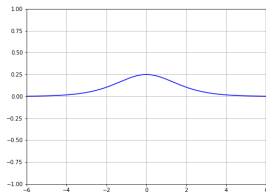
$$\delta^2 = Y.\delta^3.\sigma'() = Y.X.A.\sigma'().\sigma'().\sigma'()$$

$$\delta^1 = Z.\delta^2.\sigma'() = Z.Y.X.A.\sigma'().\sigma'().\sigma'().\sigma'()$$

Vanishing Gradient Problem (contd.)



Sigmoid



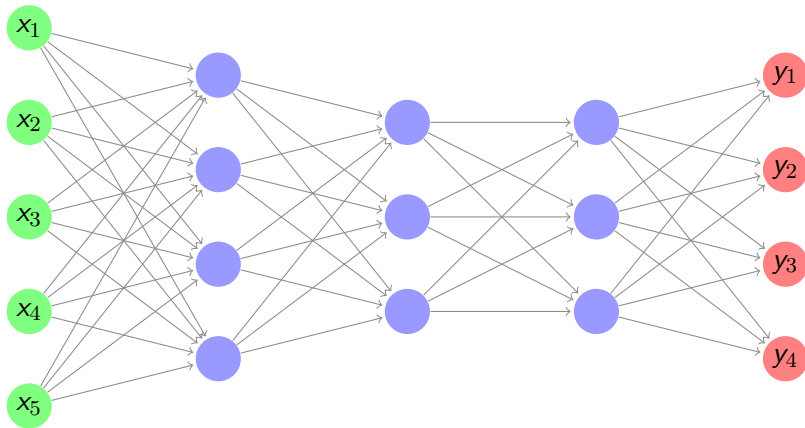
Derivative of Sigmoid

- Maximum value of sigmoid's derivative = 0.25
- $0.25^n \approx 0$ as $n \rightarrow \infty$
- Gradient tends to 0 *i.e.* *vanishes*

Deep Learning

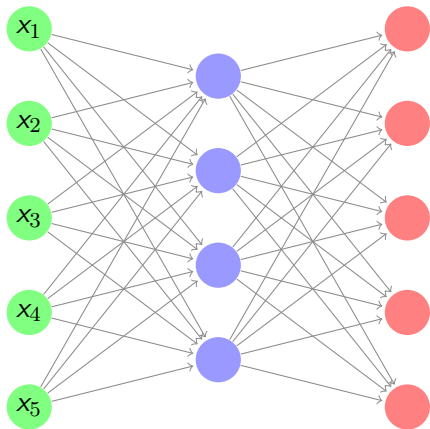
- Set of techniques and architectures that tackles such learning problems and helps to reach optimal parameters faster
- Various methods:
 - Start at near optimal values of parameters so smaller updates due to vanishing gradients is not much of a problem
 - Use better activation functions which can avoid such problems
 - Use better optimizers than standard gradient descent
 - Use novel architectures

Tackling Vanishing Gradients via Greedy Unsupervised Pretraining



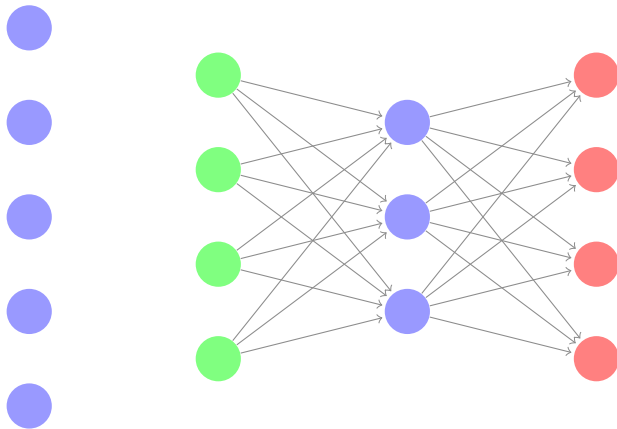
- Proposed by Bengio et al. (2007)

Tackling Vanishing Gradients via Greedy Unsupervised Pretraining



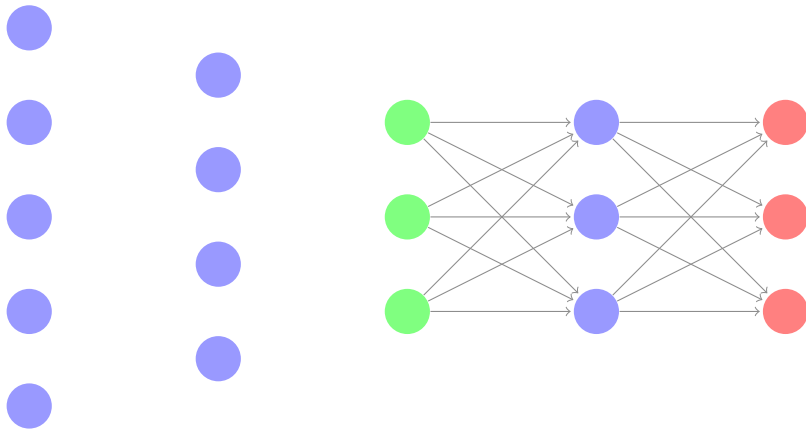
• Proposed by Bengio et al. (2007)

Tackling Vanishing Gradients via Greedy Unsupervised Pretraining



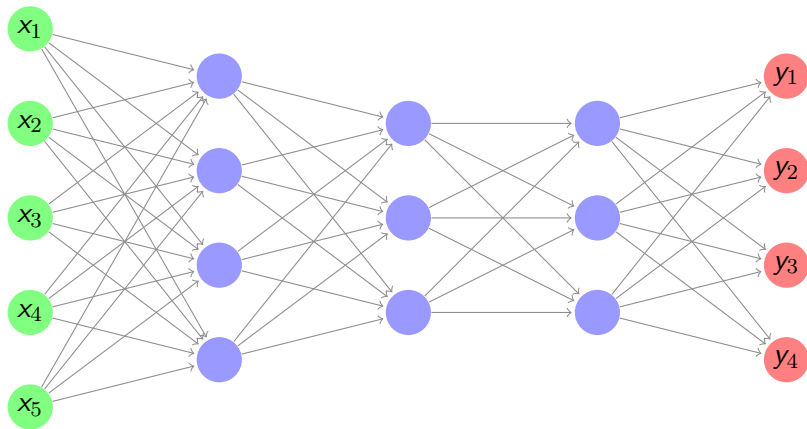
- Proposed by Bengio et al. (2007)

Tackling Vanishing Gradients via Greedy Unsupervised Pretraining



- Proposed by Bengio et al. (2007)

Tackling Vanishing Gradients via Greedy Unsupervised Pretraining



- Proposed by Bengio et al. (2007)

Tackling Vanishing Gradients via Novel Activation Functions

- Rectified Linear Unit Nair and Hinton (2010): Derivative = 1 when non-zero, else 0
- Product of derivatives does not vanish
- But once a ReLU gets to 0, it is difficult to get it to one again (Dead ReLU problem)
 - Addressed by better variants such as Leaky ReLU Maas et al. (2013), Parametric ReLU He et al. (2015) etc.

Types of Gradient Descent

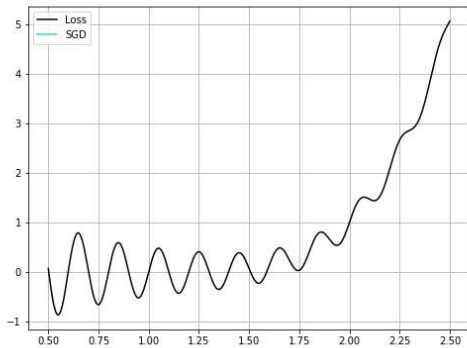
Based on the amount of data used for training

- Batch GD: all training data per update, slow, not applicable in online setting, but guaranteed to converge to global minimum for convex and local minimum for non-convex
- Stochastic GD: one training datapoint per update, fluctuates a lot, allows to jump to new and potentially better local minima, this complicates convergence, has been shown that by decreasing learning rate almost certainly converges to global in convex and local in non-convex
- Mini-batch GD: batch of n datapoints per update, best of both worlds - relatively stable convergence and can use matrix operations for batches

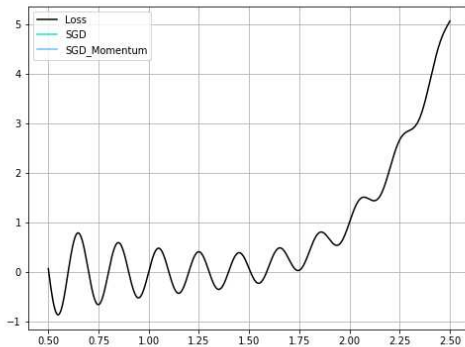
Tackling Learning Difficulties via Optimizers

- SGD mainly used for a long time
- Converges slowly
- Can get stuck in local minima

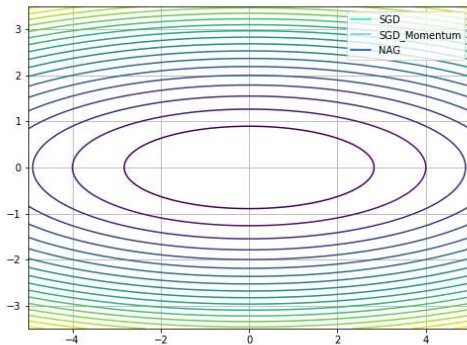
SGD



SGD + Momentum



Nesterov Accelerated Gradient



- Developed by Nesterov (1983)

Other Optimizers

- AdaGrad: Adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters Duchi et al. (2011)
- AdaDelta: Does not need an initial learning rate Zeiler (2012)
- RMSProp: Good with recurrent networks; Unpublished method from Hinton's Coursera Lectures
- Adam: Benefits of RMSProp and AdaDelta mixed with momentum tricks Kingma and Ba (2014)

Tackling Vanishing Gradients via Novel Architectures

- Novel architectures made to specific problems
- Example:
 - Derivative of activation function in LSTM is identity function is 1. Gradient does not vanish
 - Effective weight depends on forget gate activation, whose derivative is never > 1.0 . So Gradient does not explode
- Will be covered in other sessions

Conclusion

- Exciting area of research
- Heavy involvement from industry
- Many developments in each of those subareas: Activation functions, Optimizers, Architectures, *etc.*

References I

- Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.
- Hochreiter, S. (1998). Recurrent neural net learning and vanishing gradient. *International Journal Of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116.

References II

- Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.
- Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

References III

- Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. In *Doklady AN USSR*, volume 269, pages 543–547.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.
- Williams, R. J., Rumelhart, D. E., and Hinton, G. E. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–538.
- Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Thank You

Convolutional Neural Networks For NLP

Rudra Murthy

Center for Indian Language Technology,
Indian Institute of Technology Bombay

rudra@cse.iitb.ac.in

<https://www.cse.iitb.ac.in/~rudra>



*Deep Learning Tutorial.
ICON 2017, Kolkata
21th December 2017*



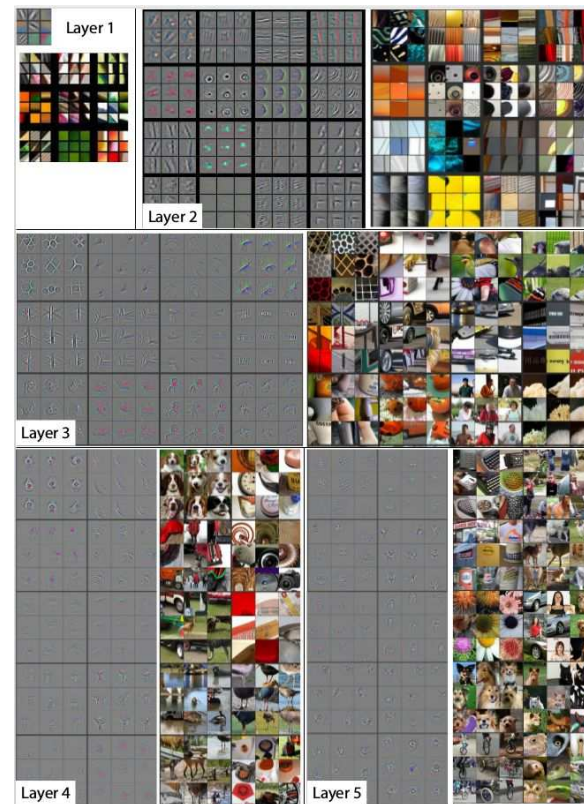
Outline

- Motivation
- What are CNNs?
- CNNs for various NLP Tasks
- Summary

Motivation

Consider the task of Image Recognition

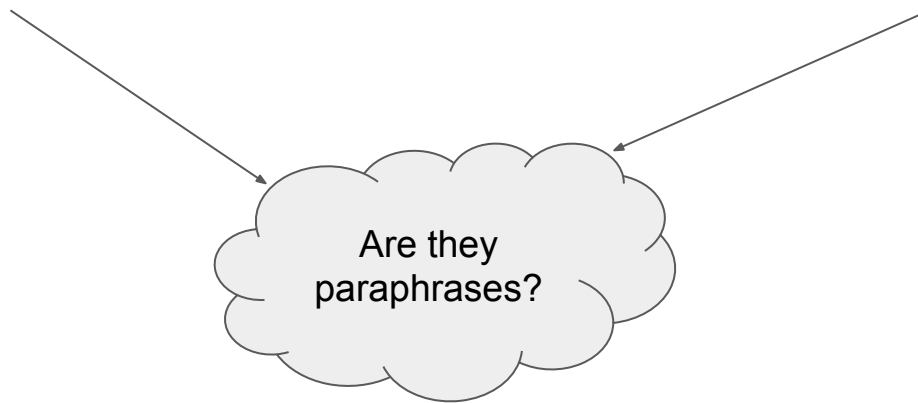
- CNNs are shown to learn hierarchical features from the data [Zeiler et.al 2014]
- Layer 1 recognizes lines and curves, layer 2 composes them to recognize simple shapes like squares, circles, *etc.*
- Layer 3 composes the output from layer 2 to recognize more complex shapes like humans, cars, *etc.*
- Two characteristics are prominent here:
 - Recognizing position-independent features
 - Composing the features to obtain more complex features



Consider the task of Paraphrase Detection

There is a Deep Learning Tutorial at ICON

DL Tutorial is scheduled @ ICON



Consider the task of Paraphrase Detection



There is a Deep Learning Tutorial at ICON

DL Tutorial is scheduled @ ICON

Consider the task of Paraphrase Detection

- Paraphrase detection can be handled at various layers of nlp layers
- At lexical layer, we compare the word overlap between the two sentences
- At Semantic layer we compare the meaning of the two sentences
- Each higher layer requires information coming from the lower layer
- We need a hierarchy of trainable feature extractors
- The lower layer feature extractor should be position independent

What are CNNs?

What is CNN?

Convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network where the individual neurons are tiled in such a way that they respond to overlapping regions in the input field. (wikipedia)

CNNs are good at learning features from the data

Specifically, we will discuss the Time-Delay Neural Network used by Collobert et.al (2011)

What is CNN?

CNN is a type of feed-forward neural network with

- Local connectivity
- Share weights/parameters across spatial positions

CNNs for various NLP tasks

Sentiment Analysis

Consider the task of sentiment analysis,

The movie is very good



The movie is very bad



Train a supervised machine learning system to predict the sentiment of the text

Consider the task of sentiment analysis,

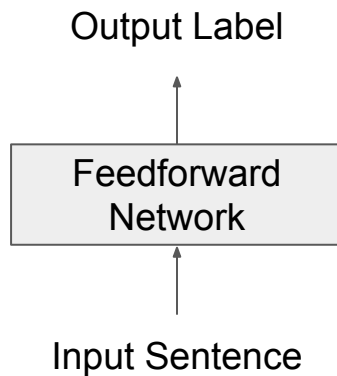
The movie is very good



The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text



Consider the task of sentiment analysis,

The movie is very good

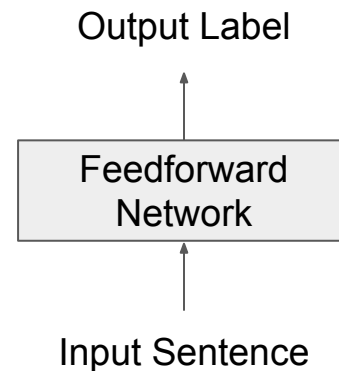


The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text

- How to represent the input sentence?
 - Bag-of-words representation
 - Disregards the word order
 - Concatenation of Word Embeddings
 - How to handle variable-length sentences?



Consider the task of sentiment analysis,

The movie is very good



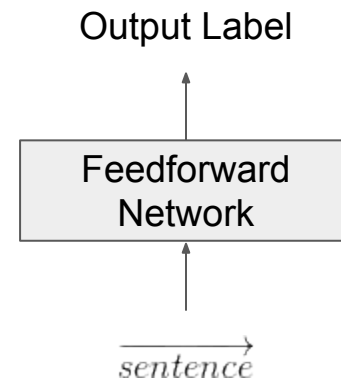
The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text

Bag of Words Representation (Average of Word Embeddings)

$$\vec{\text{sentence}} = \frac{\vec{\text{the}} + \vec{\text{movie}} + \vec{\text{is}} + \vec{\text{very}} + \vec{\text{good}}}{5}$$



Consider the task of sentiment analysis,

The movie is very good

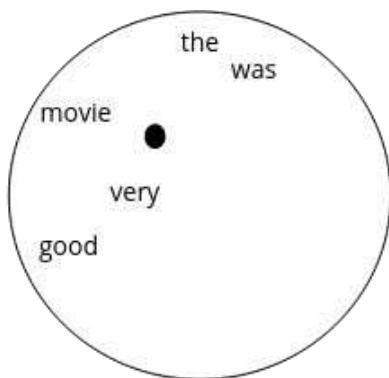


The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text

Bag of Words Representation (Average of Word Embeddings)



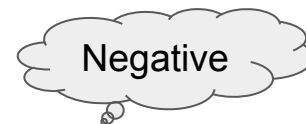
- Influence of unimportant words in the sentence changes the average embedding
- Use Tf-IDF to give importance to relevant words

Consider the task of sentiment analysis,

The movie is very good

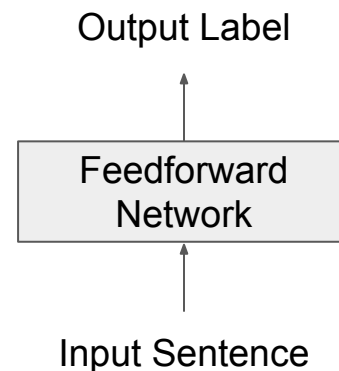


The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text

- How to represent the input sentence?
 - Concatenate the word embeddings of all words in the sentence
 - How to handle variable-length sentences?
 - Place a restriction on the sentence length



Consider the task of sentiment analysis,

The movie is very good

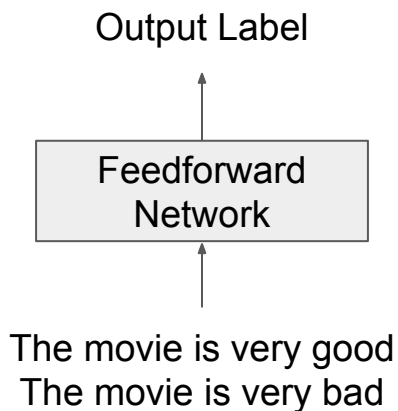


The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text

Concatenate the word embeddings



Consider the task of sentiment analysis,

The movie is very good



The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text

Concatenate the word embeddings

The
movie
is
very
good

The movie is very good



Output Label

Word
Embedding

Consider the task of sentiment analysis,

The movie is very good



The movie is very bad



Train a simple Feedforward neural network to predict the sentiment of the text

Concatenate the word embeddings

$$y = f(\overrightarrow{the}; \overrightarrow{movie}; \overrightarrow{is}; \overrightarrow{very}; \overrightarrow{good})$$

$$y = g(W \times [\overrightarrow{the}; \overrightarrow{movie}; \overrightarrow{is}; \overrightarrow{very}; \overrightarrow{good}])$$

$$y = g([W_1; W_2; W_3; W_4; W_5] \times [\overrightarrow{the}; \overrightarrow{movie}; \overrightarrow{is}; \overrightarrow{very}; \overrightarrow{good}])$$

$$y = g([W_1 \times \overrightarrow{the}; W_2 \times \overrightarrow{movie}; W_3 \times \overrightarrow{is}; W_4 \times \overrightarrow{very}; W_5 \times \overrightarrow{good}])$$

- f denotes the feedforward network here
- Let W denote the weights in the first layer
- g denotes the layers above

Concatenation of word embeddings for sentiment analysis

Let, the training data consist of instances of the form,

The ----- is very -----

The first slot is filled by words like *movie*, *camera*, and the second Slot is filled by words like *good*, *bad*, *horrible*,

$$y = f(\vec{the}; \vec{movie}; \vec{is}; \vec{very}; \vec{good})$$

$$y = g(W \times [\vec{the}; \vec{movie}; \vec{is}; \vec{very}; \vec{good}])$$

$$y = g([W_1; W_2; W_3; W_4; W_5] \times [\vec{the}; \vec{movie}; \vec{is}; \vec{very}; \vec{good}])$$

$$y = g([W_1 \times \vec{the}; W_2 \times \vec{movie}; W_3 \times \vec{is}; W_4 \times \vec{very}; W_5 \times \vec{good}])$$

- How will the model behave for the input sentence

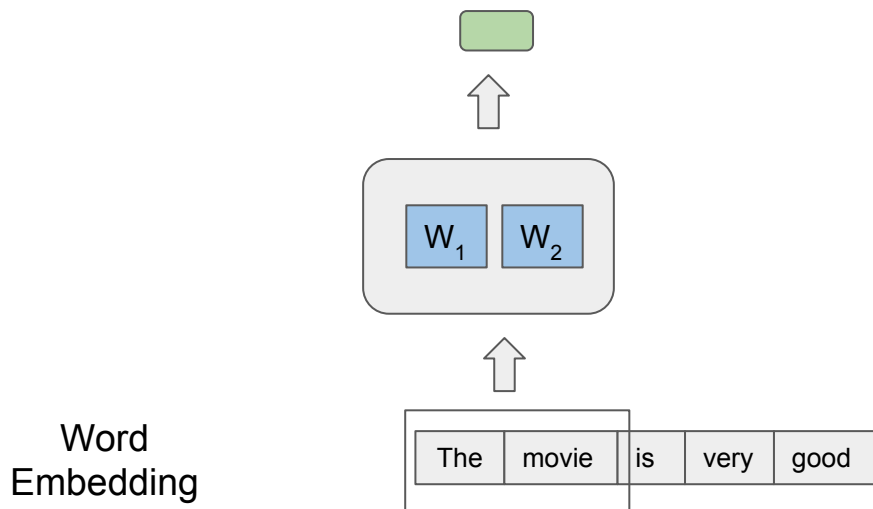
“Very good was the movie”

CNNs for Sentiment Analysis

- Simplest approach for sentiment analysis is to check if there are any sentiment bearing words
- Assign the label based on the sentiment score of the word
- This is essentially what Tf-IDF scheme tries to simulate
- Can we do this using Deep Learning?

CNNs for Sentiment Analysis

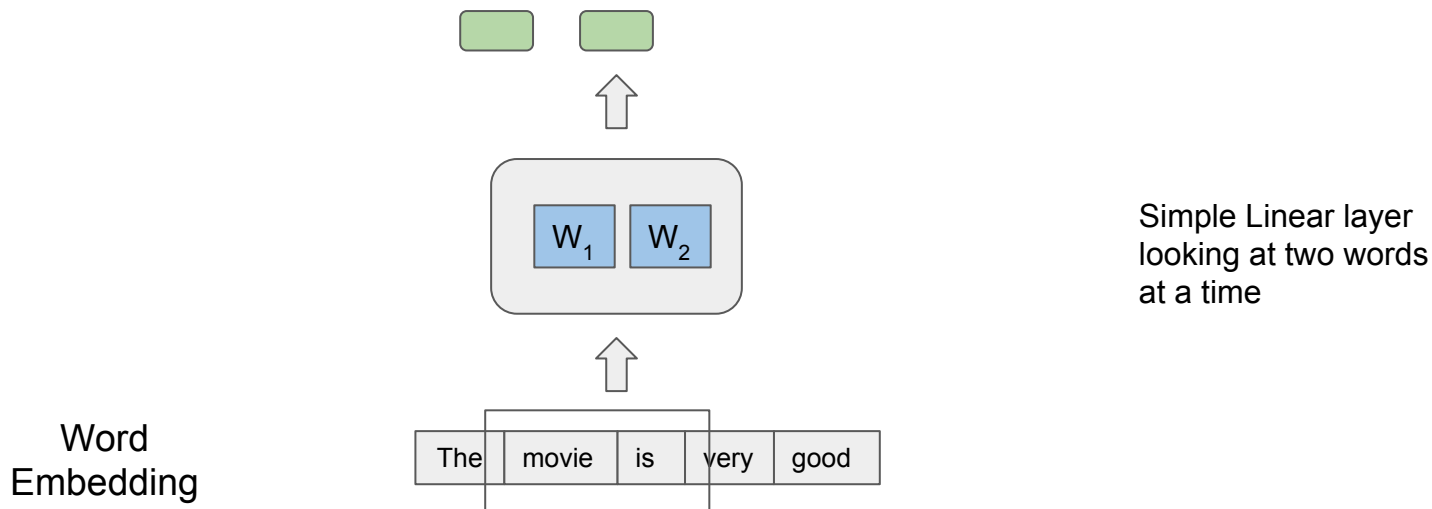
Use Feedforward neural network on consecutive *ngram* words



Simple Linear layer
looking at two words
at a time

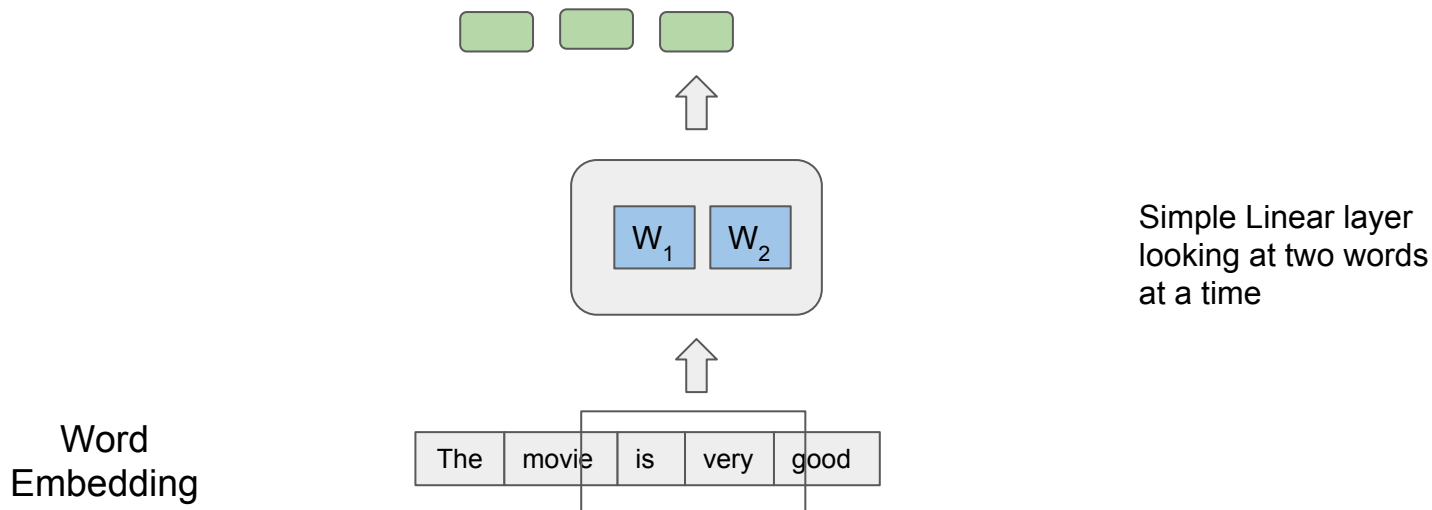
CNNs for Sentiment Analysis

Use Feedforward neural network on consecutive *ngram* words



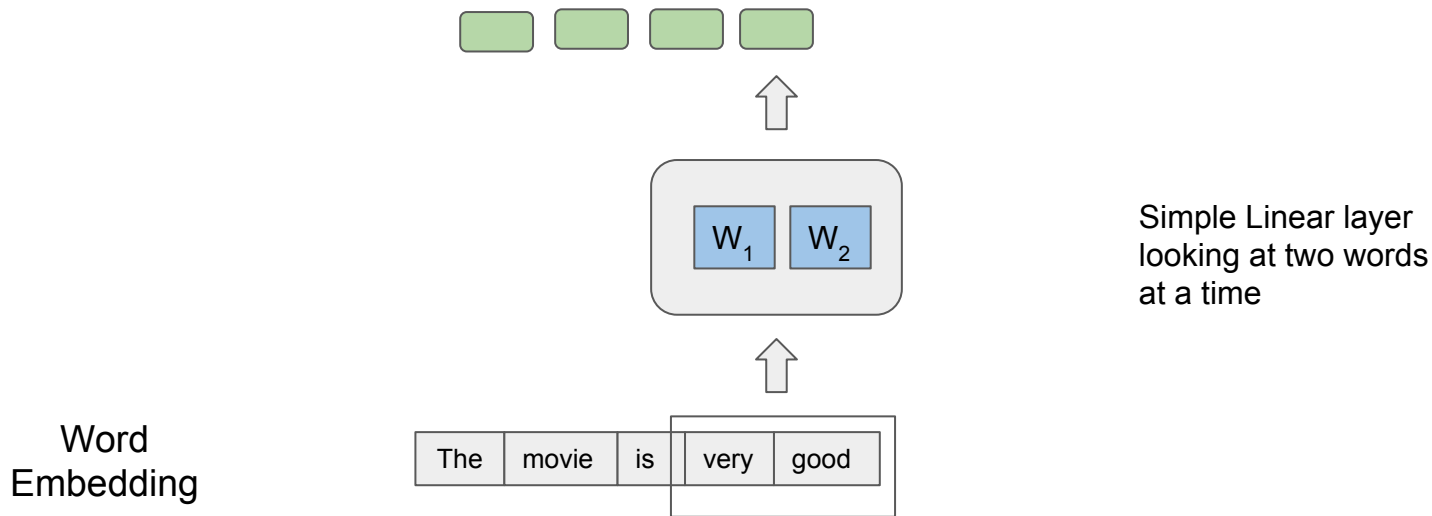
CNNs for Sentiment Analysis

Use Feedforward neural network on consecutive *ngram* words







CNNs for Sentiment Analysis

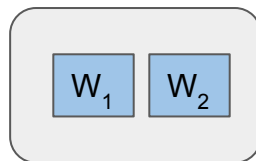
Use Feedforward neural network on consecutive *ngram* words



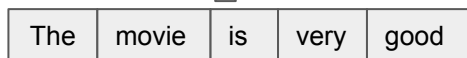
CNNs for Sentiment Analysis

Use Feedforward neural network on consecutive *ngram* words

The movie 
movie is 
is very 
very good 







Simple Linear layer
looking at two words
at a time



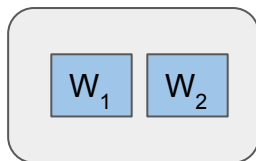
Word
Embedding

CNNs for Sentiment Analysis

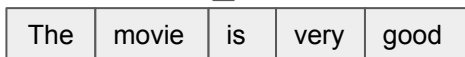
Use Feedforward neural network on consecutive *ngram* words

The movie 
movie is 
is very 
very good 

How do we go from variable length representation to a fixed length representation so that the feed-forward neural network can handle?







Simple Linear layer
looking at two words
at a time



Word
Embedding

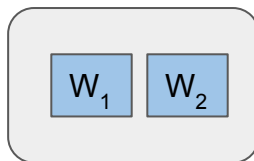
CNNs for Sentiment Analysis

Use Feedforward neural network on consecutive *ngram* words

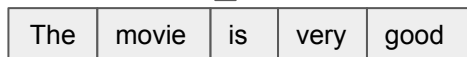
The movie 
movie is 
is very 
very good 

How do we go from variable length representation to a fixed length representation so that the feed-forward neural network can handle?

Ideally we have to choose the phrase “very good”, so weightage have to be given to this phrase



Simple Linear layer looking at two words at a time



Word
Embedding

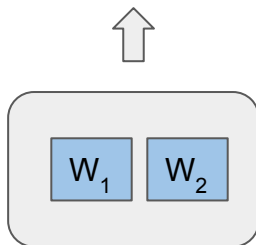
CNNs for Sentiment Analysis

Use Feedforward neural network on consecutive *ngram* words

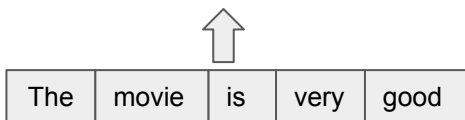


How do we go from variable length representation to a fixed length representation so that the feed-forward neural network can handle?

Ideally we have to choose the phrase "very good", so weightage have to be given to this phrase



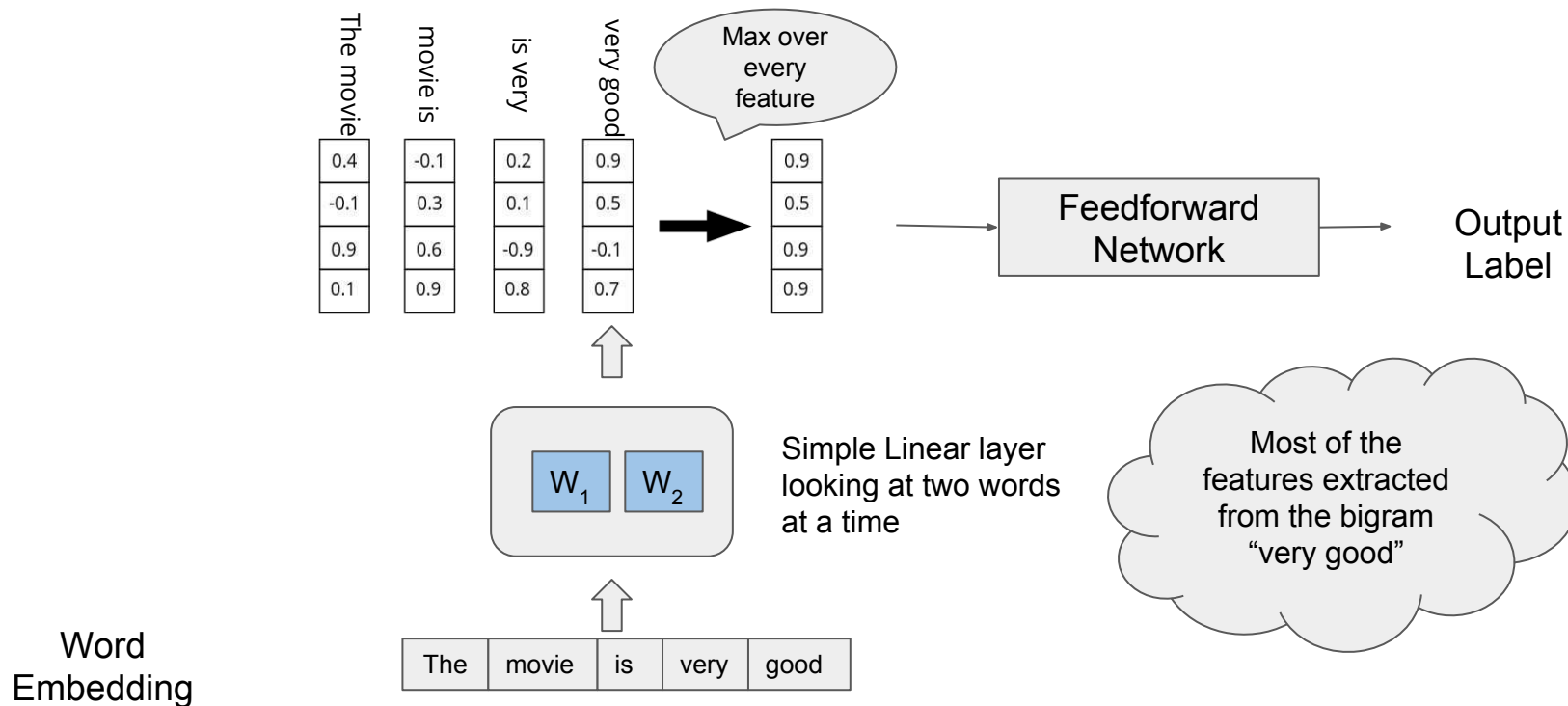
Simple Linear layer looking at two words at a time



Word
Embedding

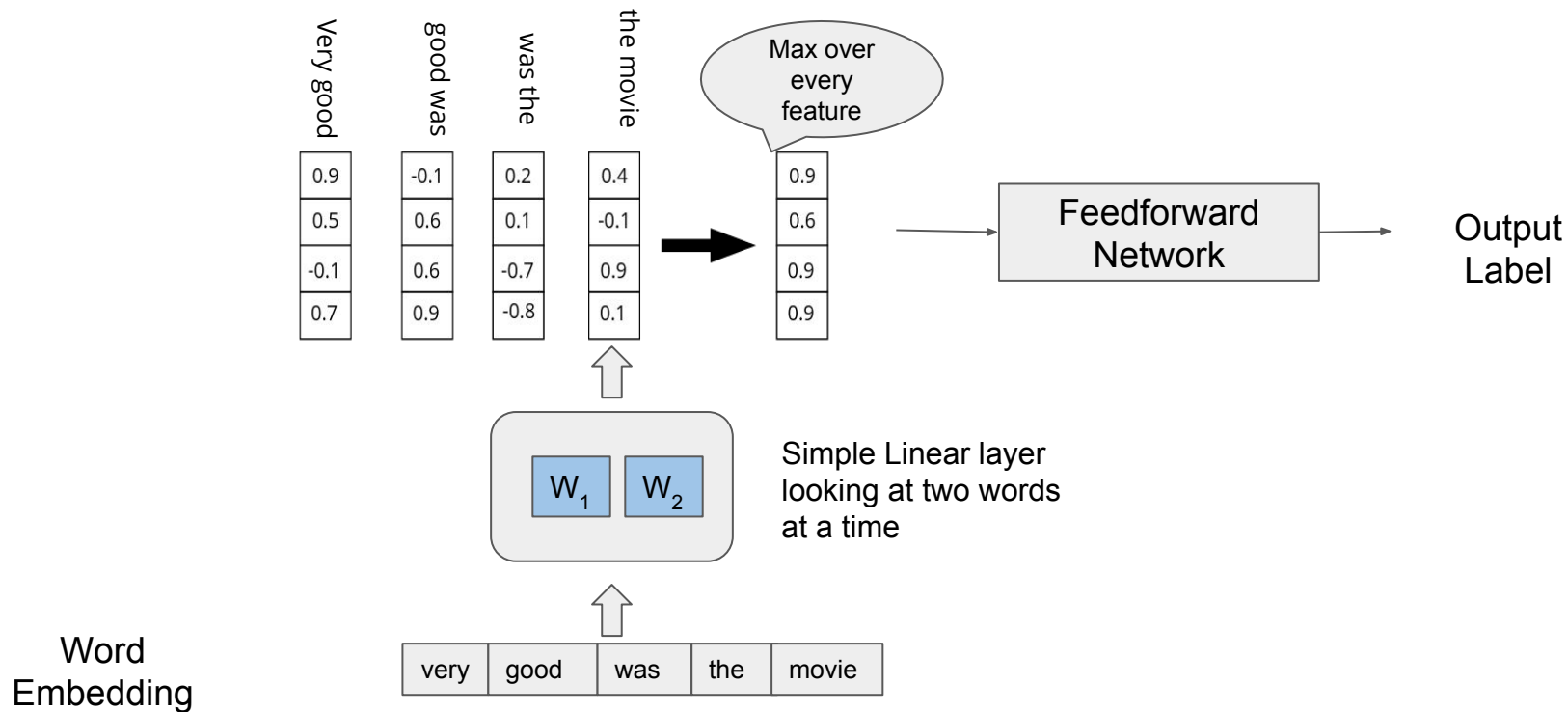
CNNs for Sentiment Analysis

Use Feedforward neural network on consecutive *ngram* words



CNNs for Sentiment Analysis

Use Feedforward neural network on consecutive *ngram* words



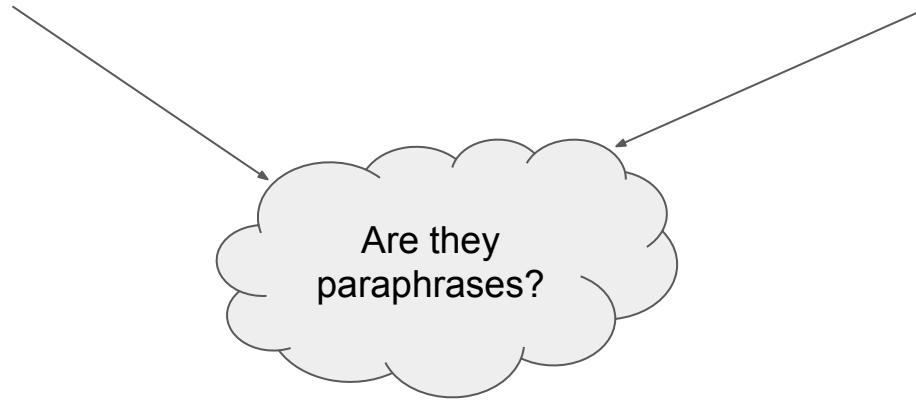
CNNs for various NLP tasks

Paraphrase Detection

Paraphrase Detection

There is a Deep Learning Tutorial at ICON

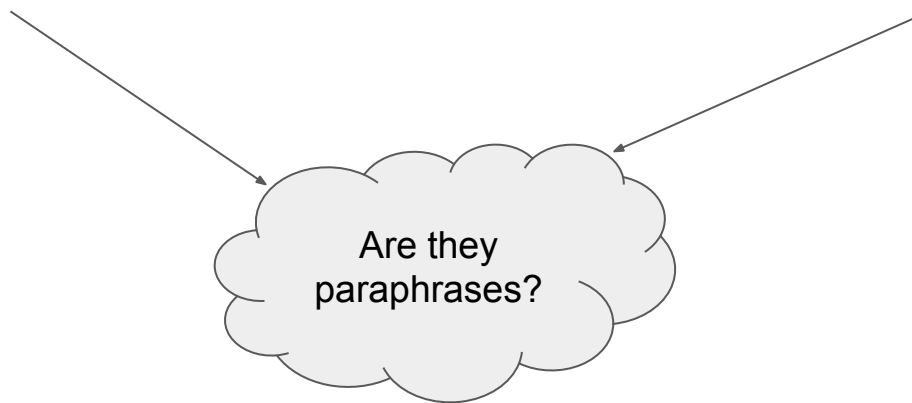
Deep Learning Tutorial is scheduled at ICON



Paraphrase Detection

There is a Deep Learning Tutorial at ICON

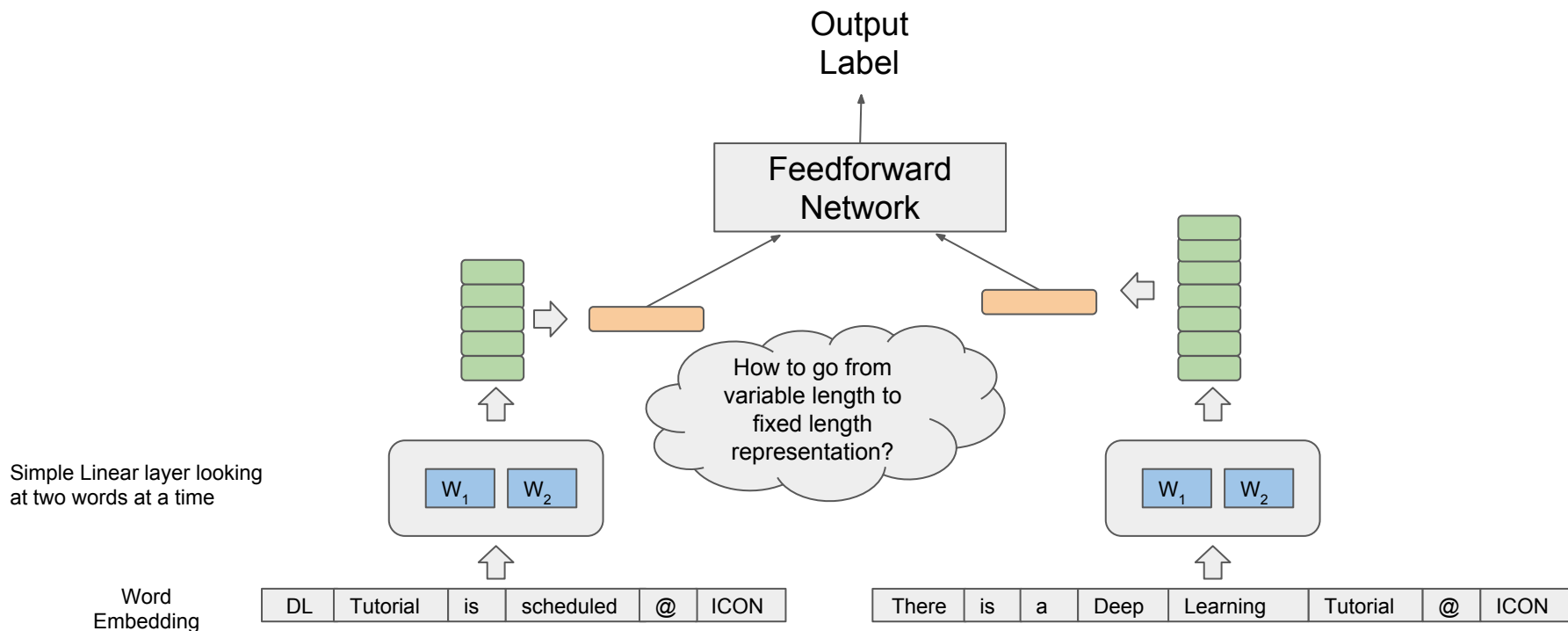
Deep Learning Tutorial is scheduled at ICON



Simplest approach for paraphrase detection using CNNs

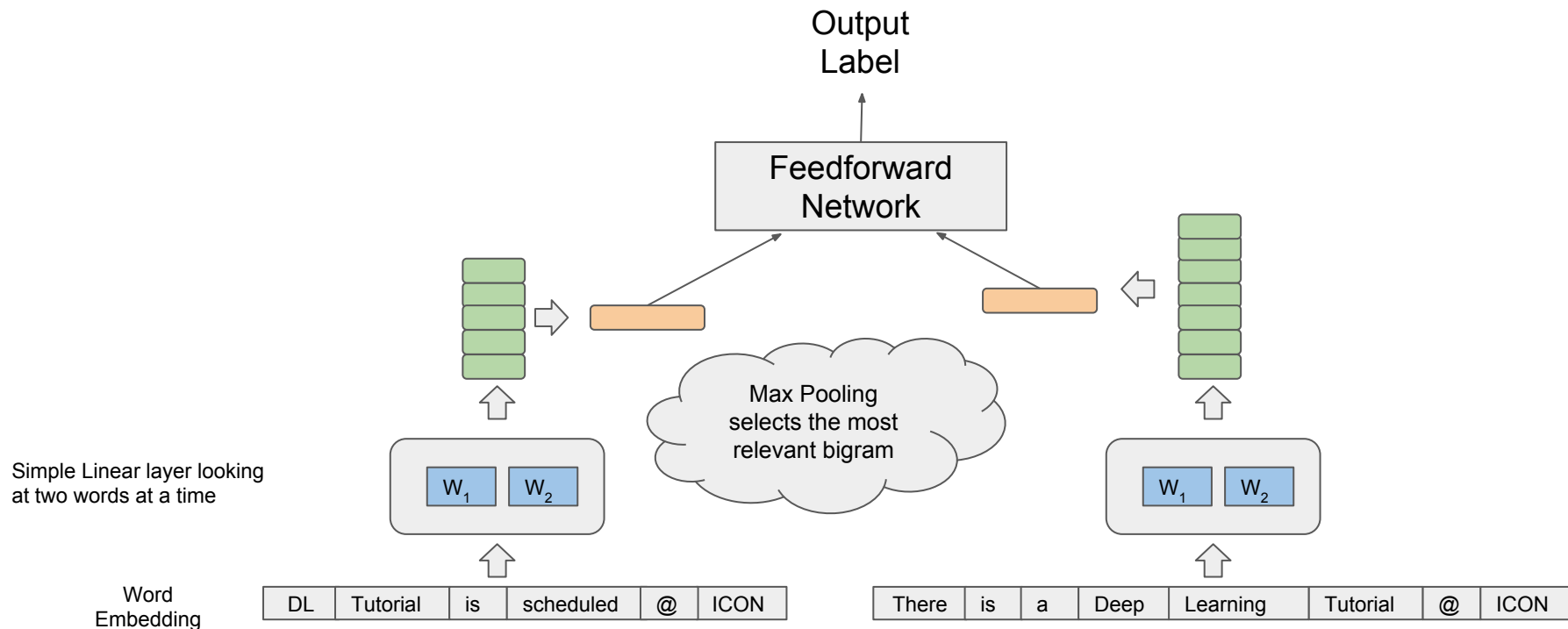
CNNs for Paraphrase Detection

Use Feedforward neural network on consecutive *ngram* words



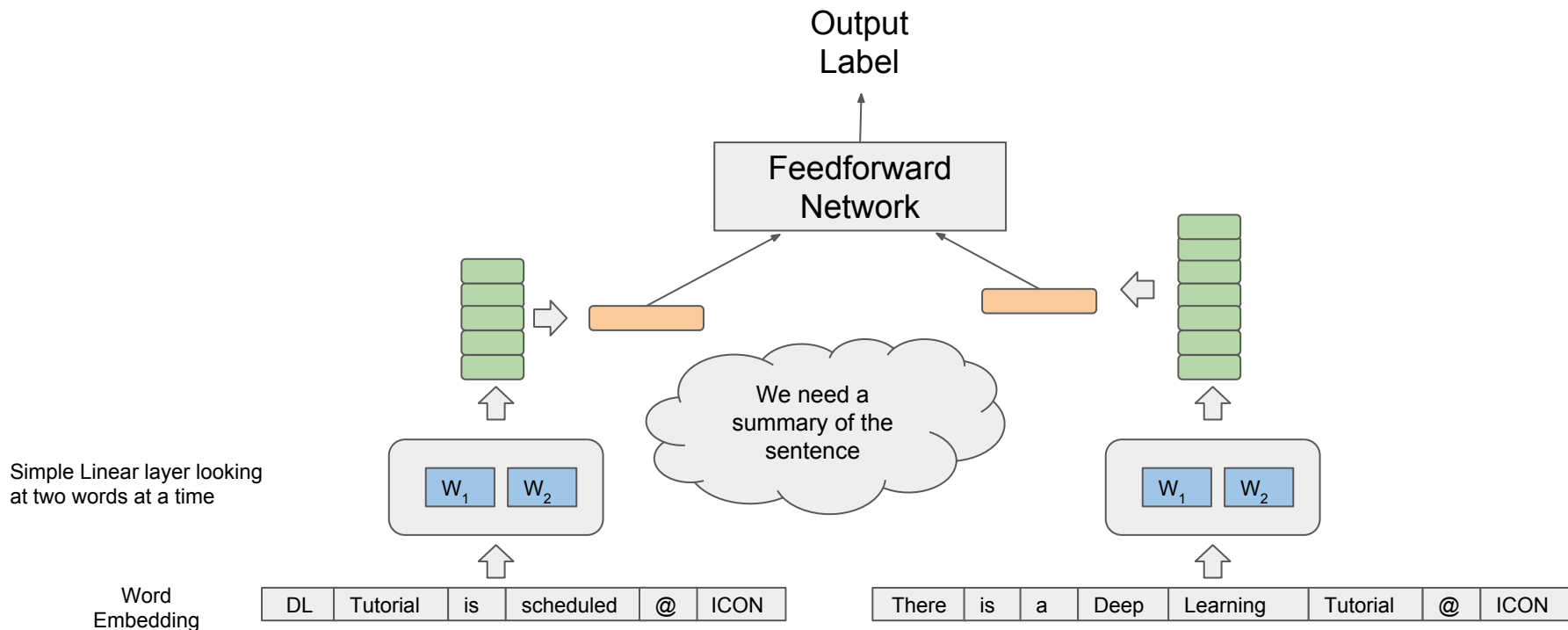
CNNs for Paraphrase Detection

Use Feedforward neural network on consecutive *ngram* words



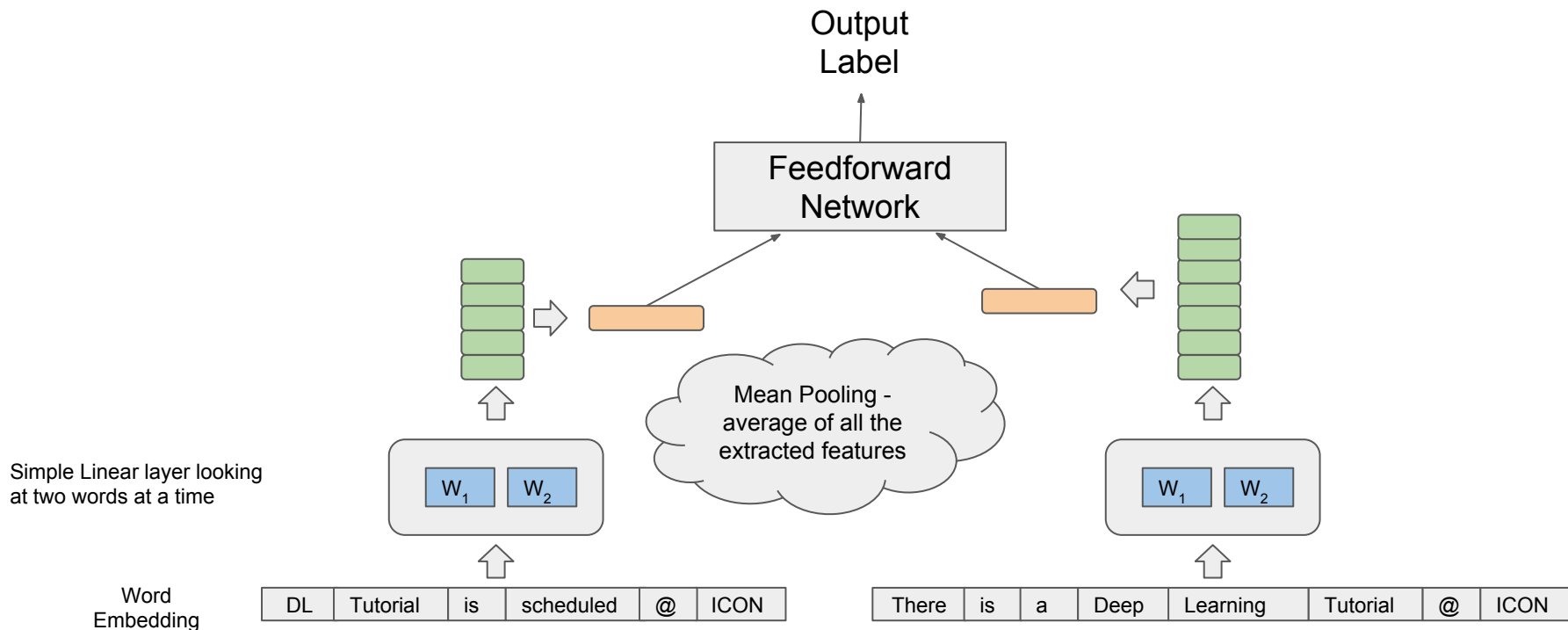
CNNs for Paraphrase Detection

Use Feedforward neural network on consecutive *ngram* words



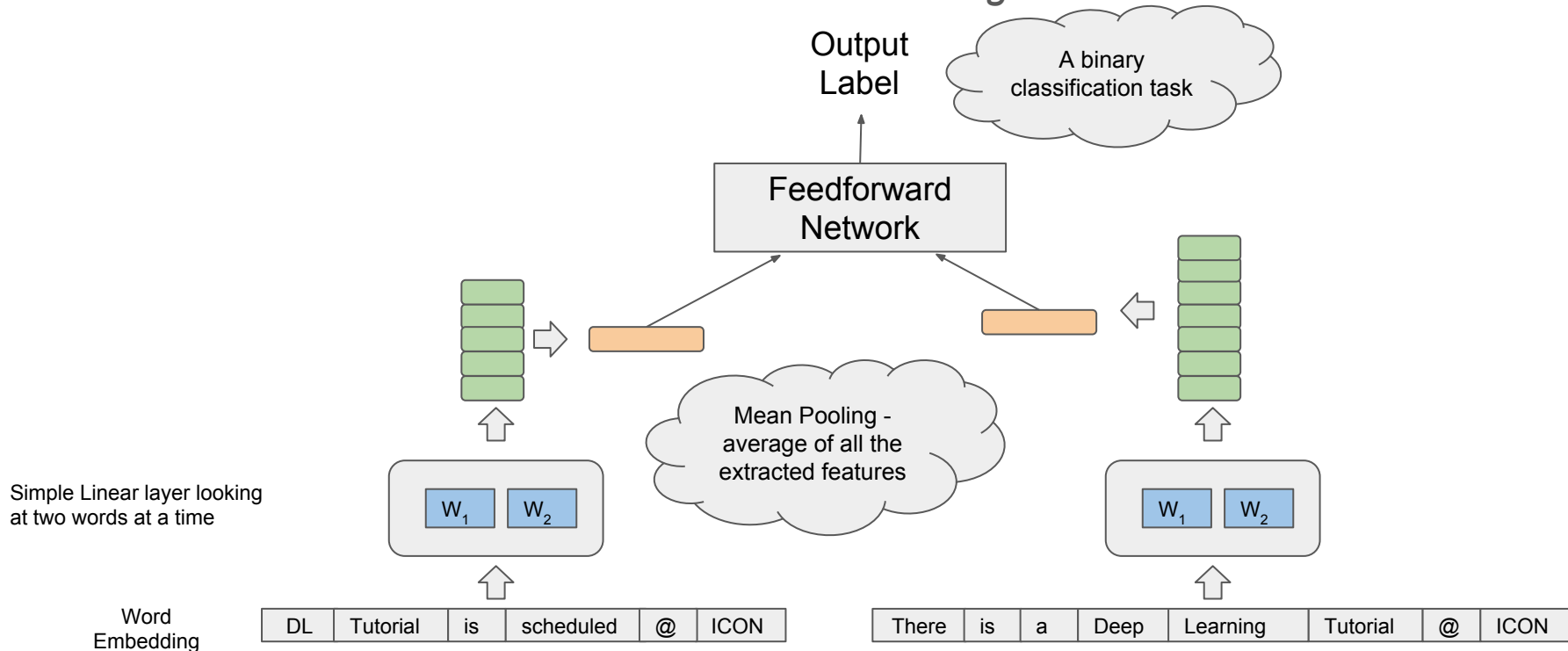
CNNs for Paraphrase Detection

Use Feedforward neural network on consecutive *ngram* words



CNNs for Paraphrase Detection

Use Feedforward neural network on consecutive *ngram* words



CNNs for Paraphrase Detection

- The architecture presented is too simple
- We can have parallel CNNs each looking at ngrams of specific length
- CNNs can be thought of performing composition operation
- We can have hierarchy of CNNs, which composes words to form simple phrases, simple phrases to form complex phrases, complex phrases to sentences, sentences to paragraphs,

CNNs in Torch

- First let us create word embedding matrix
 - `embed = nn.LookupTableMaskZero(sourceDictionarySize, embeddingDimension)`
- Given any word we send it through LookupTable to get the corresponding word embeddings
 - `outputWordEmbed = embed:forward(inputWords)`
- Now let us Create CNN module
 - `cnn = nn.Sequential()`
 - `cnn:add(embed)`
 - `cnn:add(nn.TemporalConvolution(embeddingDimension , filterSize, nGrams))`
 - `cnn:add(nn.Tanh())` -- *Optional non-linearity*
 - `cnn:add(nn.Max(1))`

Thank You

Questions?

References

- Zeiler, Matthew D. and Fergus, Rob (2014). Visualizing and Understanding Convolutional Networks. Computer Vision -- ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I. Springer International Publishing
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. J. Mach. Learn. Res.,
- dos Santos, C., Guimaraes, V., Niterói, R., and de Janeiro, R. (2015). Boosting named entity recognition with neural character embeddings. Proceedings of NEWS 2015 The Fifth Named Entities Workshop, page 9.
- Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. CoRR, abs/1508.01991

References

- Lample, G., Ballesteros, M., Kawakami, K., Subramanian, S., and Dyer, C. (2016). Neural architectures for named entity recognition. In In proceedings of NAACL-HLT (NAACL 2016)., San Diego, US

Recurrent Neural Network (RNN)

Md Shad Akhtar

Research Scholar

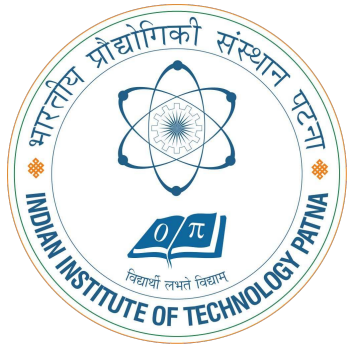
AI-NLP-ML Group

Department of Computer Science & Engineering

Indian Institute of Technology Patna

shad.pcs15@iitp.ac.in

<https://iitp.ac.in/~shad.pcs15/>



Outline

- Recurrent Neural Network (RNN)
 - Training of RNNs
 - BPTT
 - Visualization of RNN through Feed-Forward Neural Network
 - Usage
 - Problems with RNNs
- Long Short Term Memory (LSTM)
- Attention Mechanism

Recurrent Neural Network (RNN)

Basic definition:

A neural network with feedback connections.

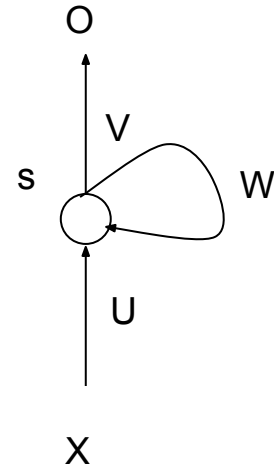
X: Input

O: Output

S: Hidden state

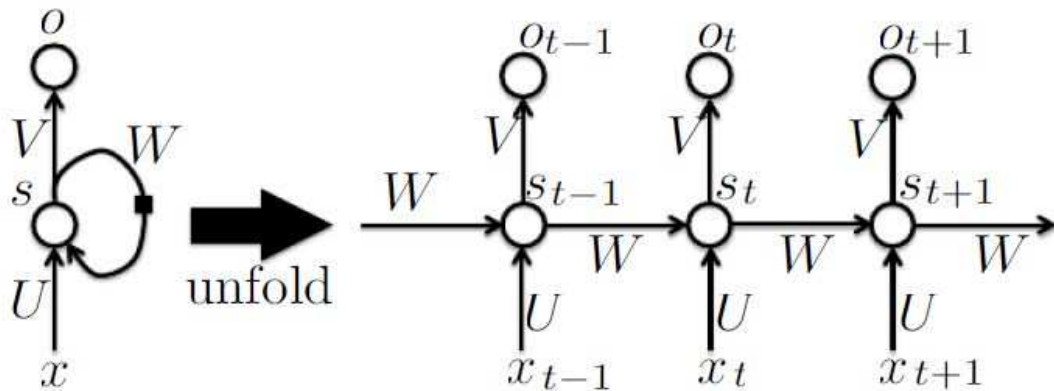
Weights: [U,V,W]

Learned during training



Recurrent Neural Network (RNN)

- Enable networks to do temporal processing
- Good at learning sequences
- Acts as memory unit



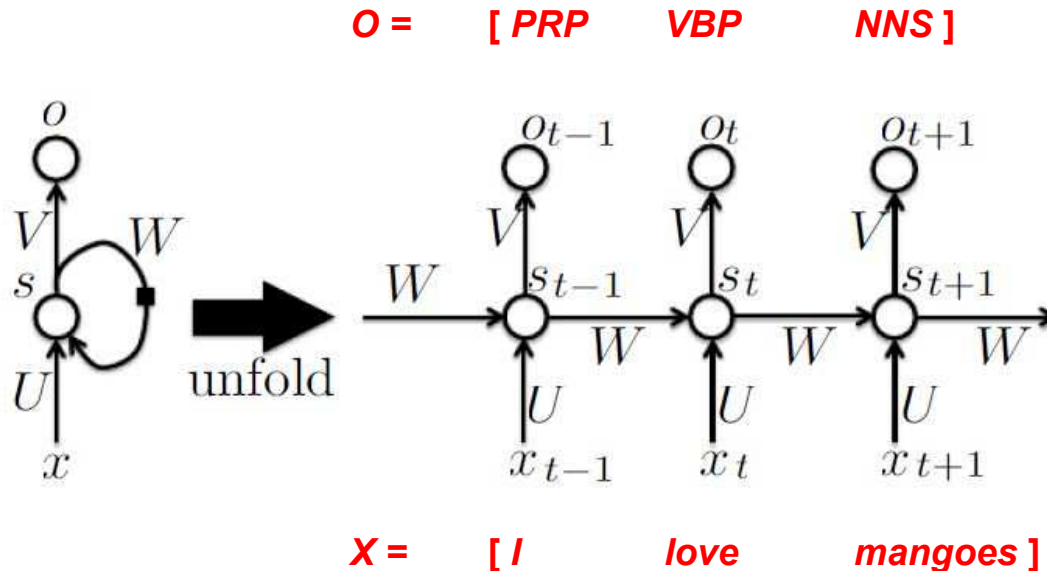
Memory

$$a_t = b + \mathbf{W} s_{t-1} + U x_t$$
$$s_t = \tanh(a_t)$$
$$o_t = c + V s_t$$
$$p_t = \text{softmax}(o_t)$$

RNN - Example 1

Part-of-speech tagging:

- Given a sentence X , tag each word its corresponding grammatical class.



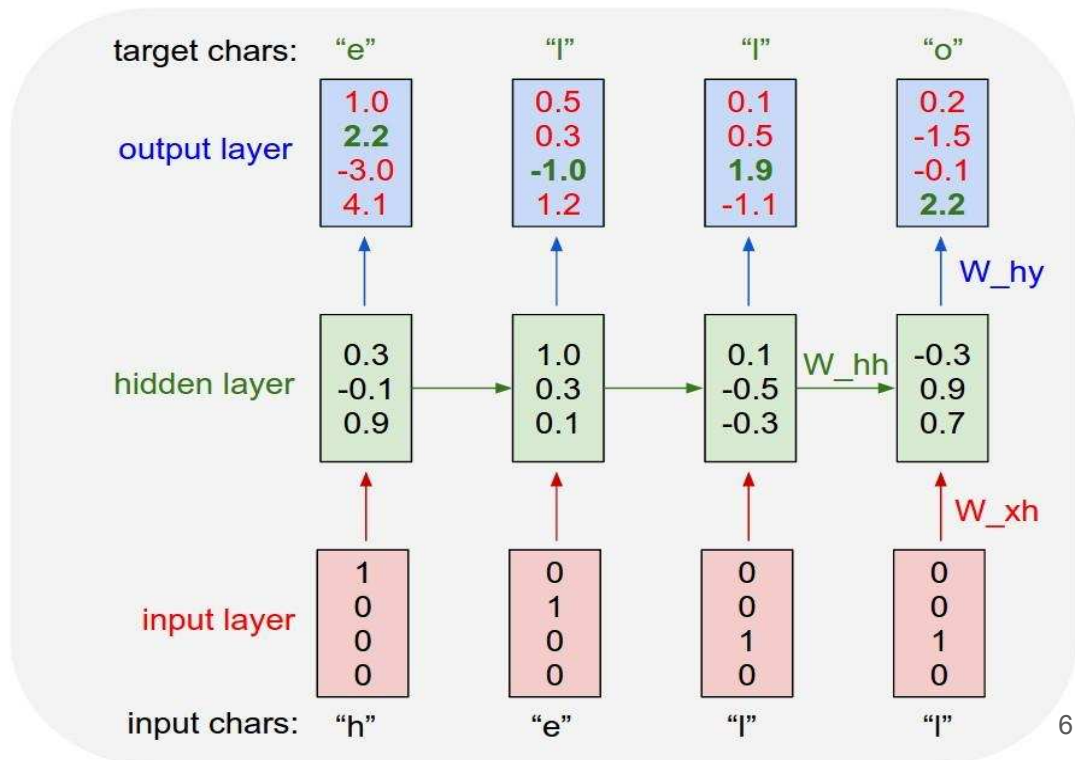
RNN - Example 2

Character level language model:

- Given previous and current characters, predict the next character in the sequence.

Let

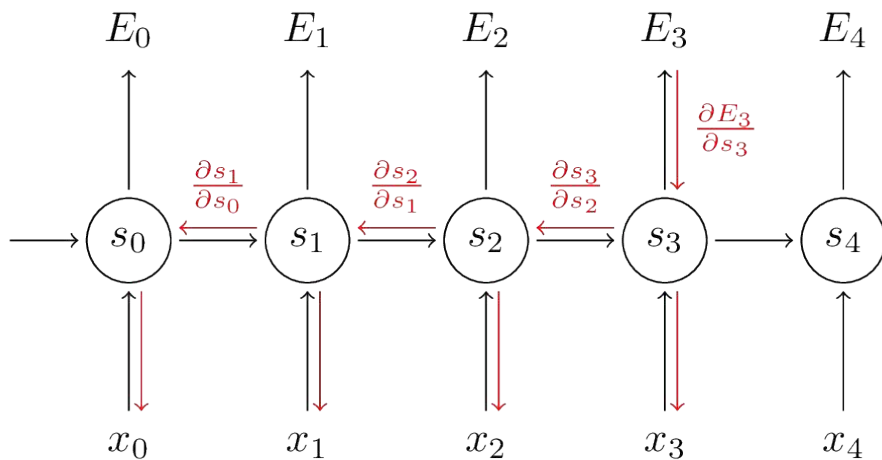
- Vocabulary:** [h,e,l,o]
- One-hot representations**
 - $h = [1\ 0\ 0\ 0]$
 - $e = [0\ 1\ 0\ 0]$
 - $l = [0\ 0\ 1\ 0]$
 - $o = [0\ 0\ 0\ 1]$



Training of RNNs

How to train RNNs?

- Typical FFN
 - Backpropagation algorithm
- RNNs
 - A variant of backpropagation algorithm namely **Back-Propagation Through Time (BPTT)**.

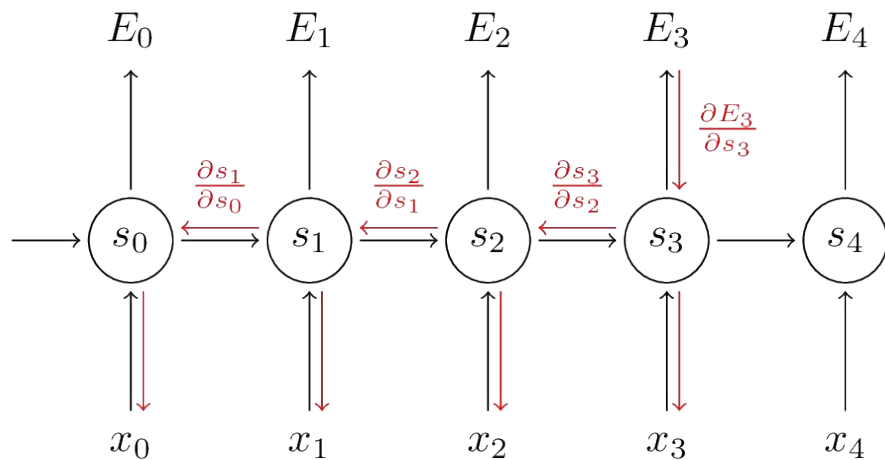


BackPropagation Through Time (BPTT)

Error for an instance = Sum of errors at each time step of the instance

Gradient of error

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$



BackPropagation Through Time (BPTT)

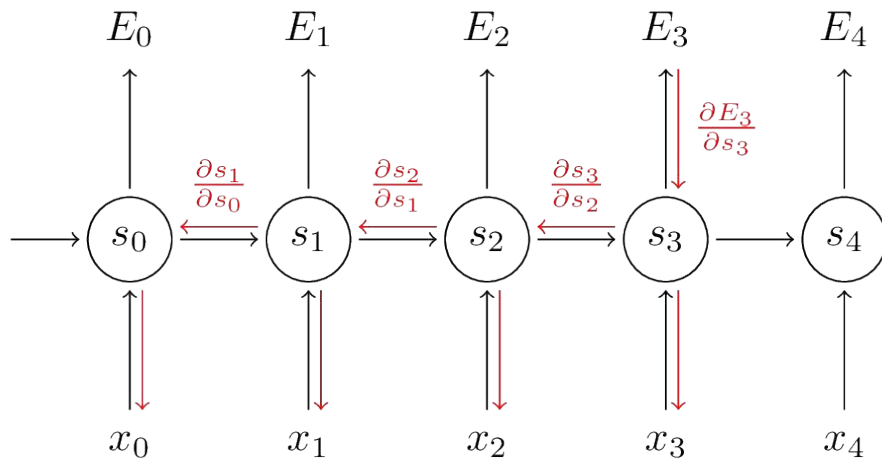
For V

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V}$$

For W (Similarly for U)

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

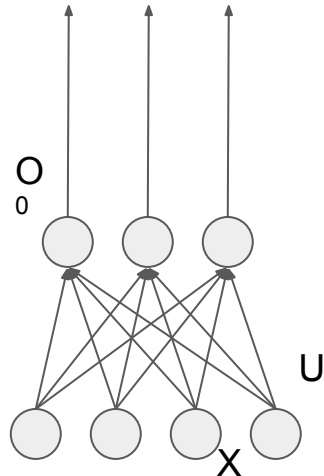


Visualization of RNN through Feed-Forward Neural Network

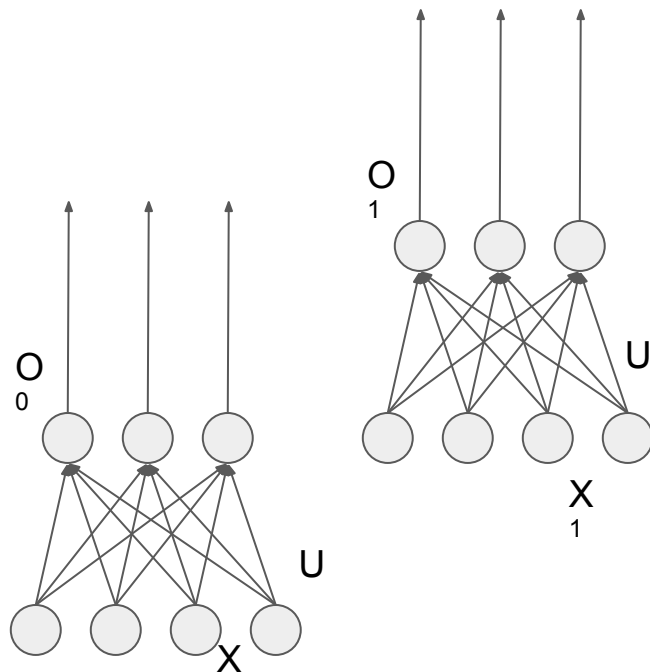
Problem, Data and Network Architecture

- Problem:
 - I/p sequence (X) : X^0, X^1, \dots, X^T
 - O/p sequence (O) : O^0, O^1, \dots, O^T
- Representation of data:
 - I/p dimension : 4
 - $X^0 \rightarrow 0\ 1\ 1\ 0$
 - O/p dimension : 3
 - $O^0 \rightarrow 0\ 0\ 1$
- Network Architecture
 - Number of neurons at I/p layer : 4
 - Number of neurons at O/p layer : 3
 - Do we need hidden layers?
 - If yes, number of neurons at each hidden layers

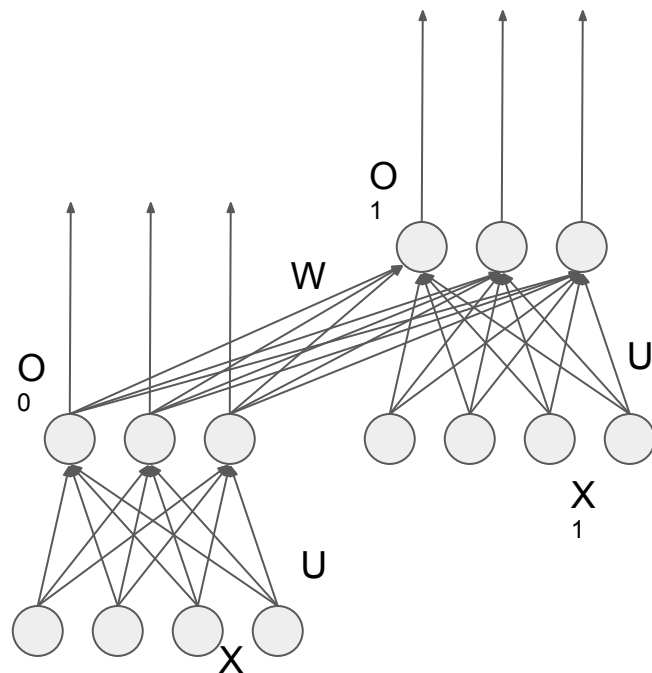
Network @ $t = 0$



Network @ $t = 1$

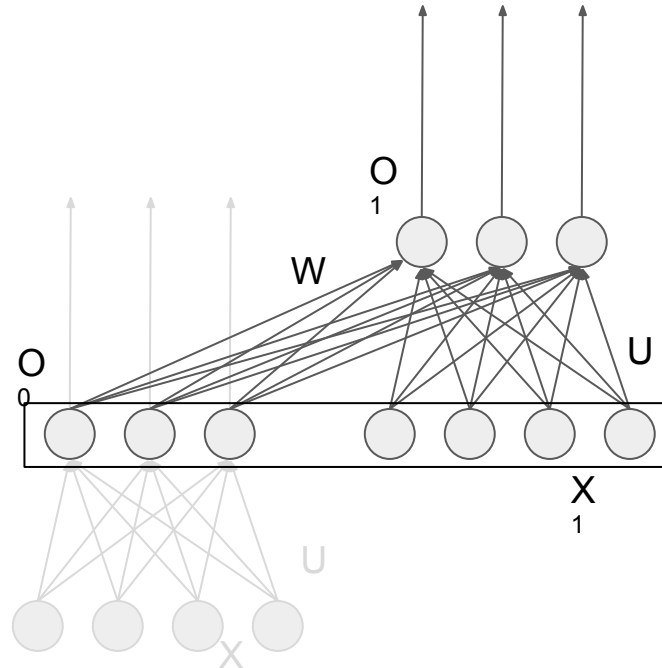


Network @ $t = 1$



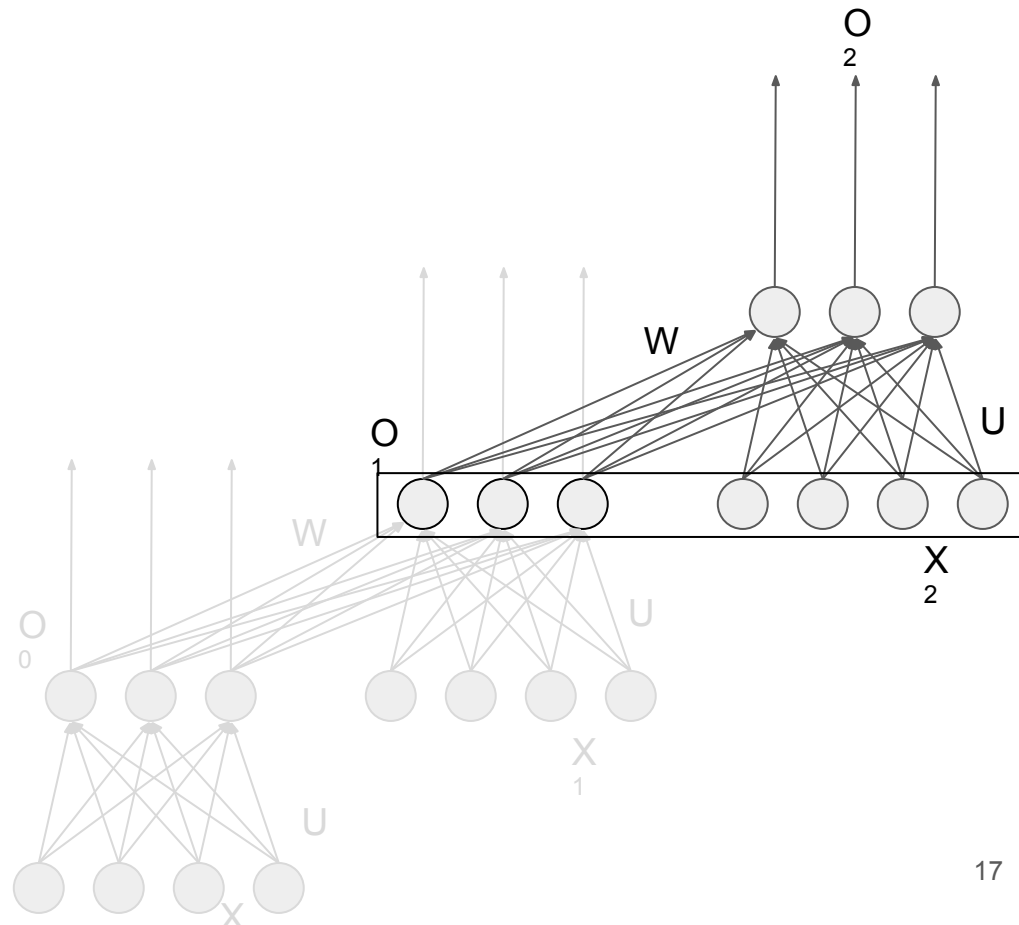
Network @ $t = 1$

$$\begin{aligned} O^1 &= f(W \cdot O^0 + U \cdot X^1) \\ &= f([W, U] \cdot [O^0, x^1]) \end{aligned}$$

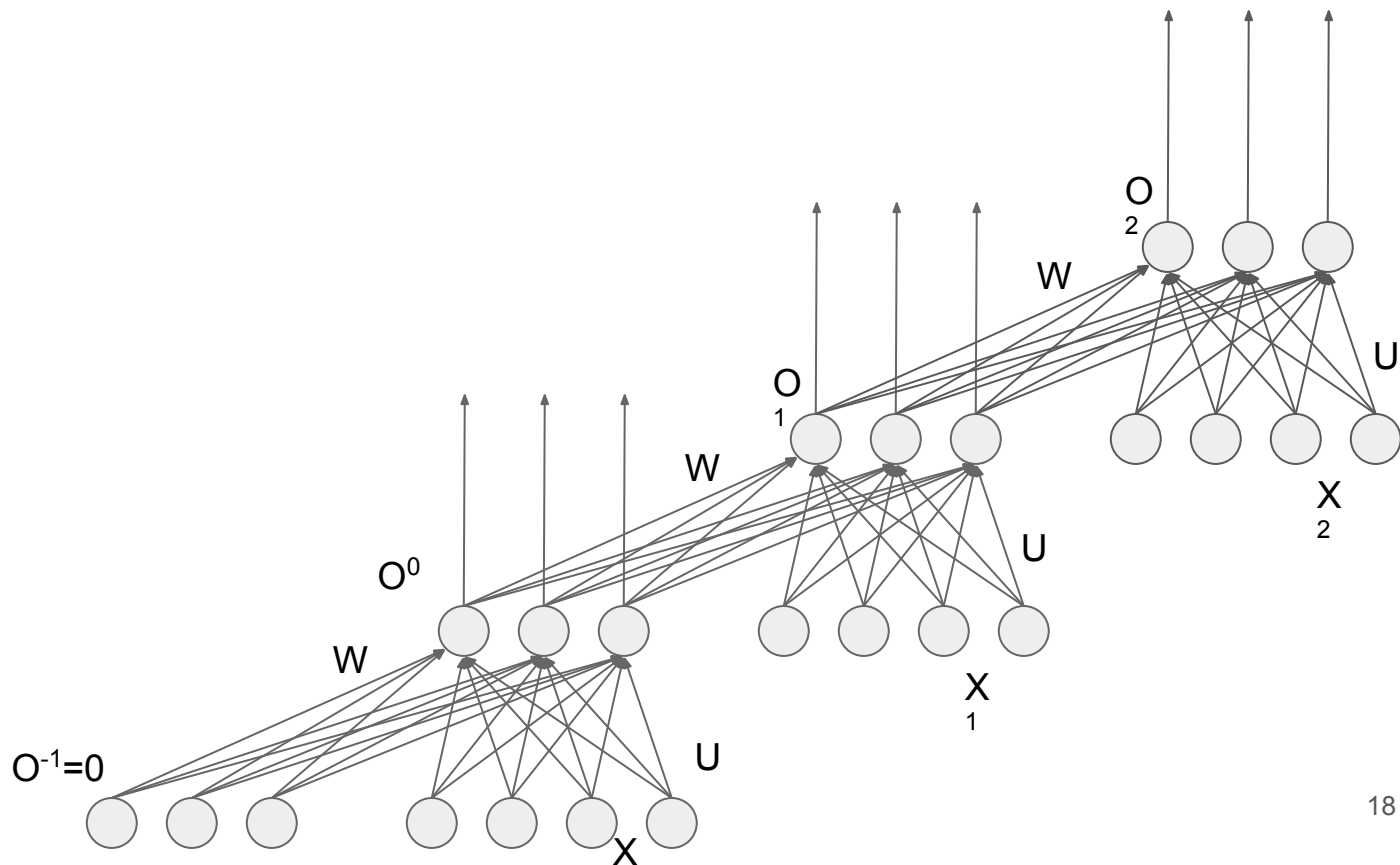


Network @ $t = 2$

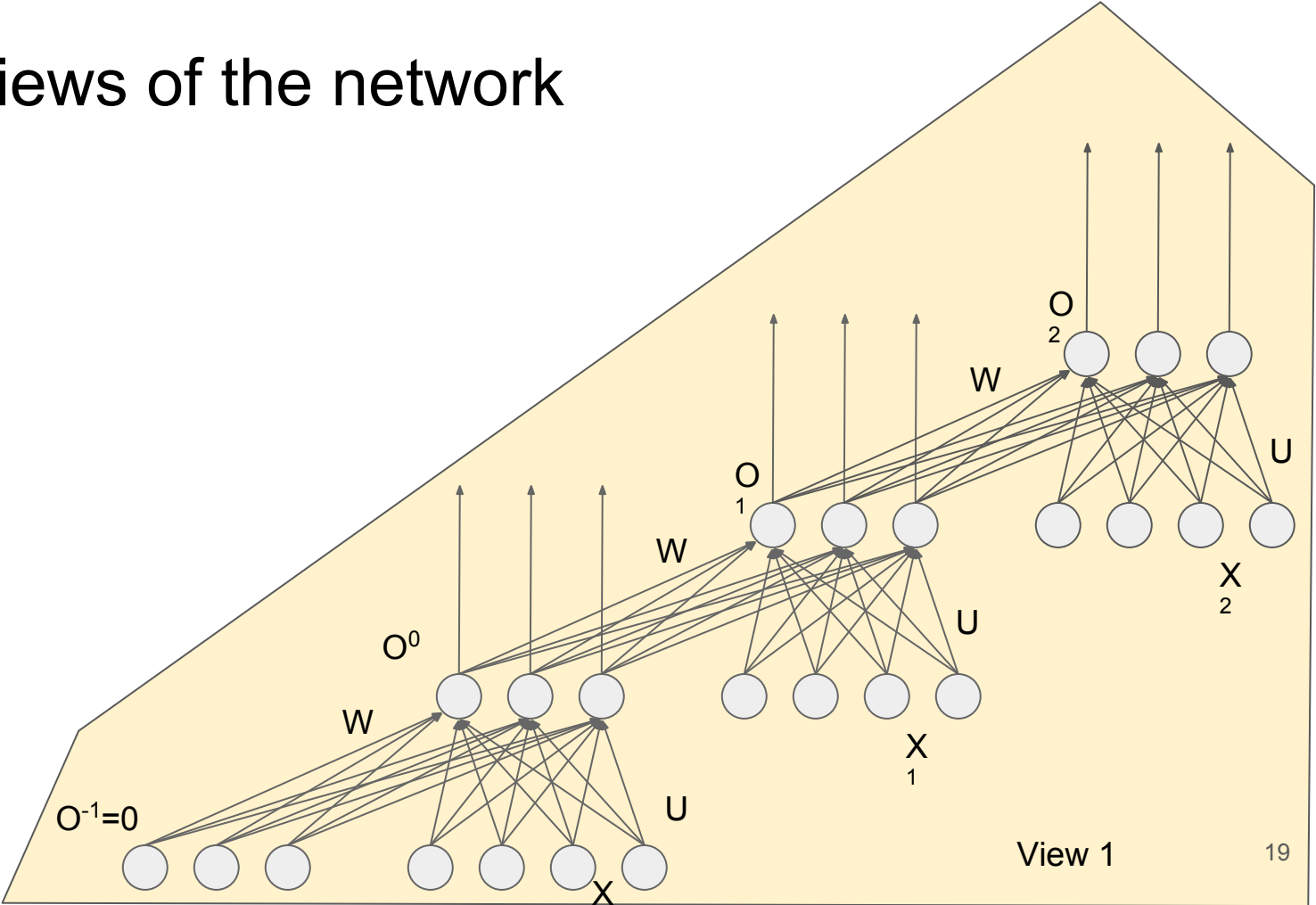
$$O^2 = f(W \cdot O^1 + U \cdot X^2)$$
$$= f([W, U] \cdot [O^1, x^2])$$



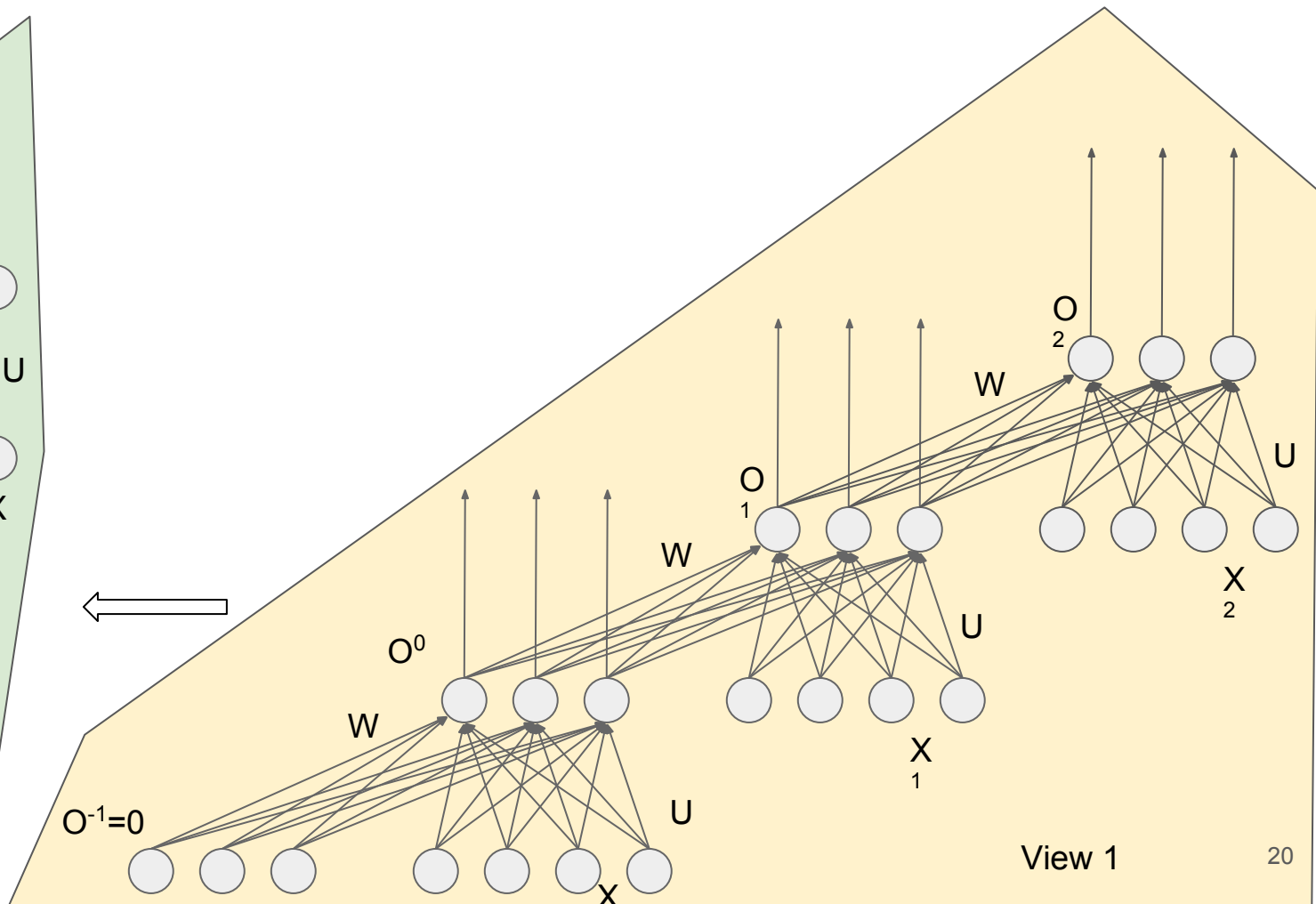
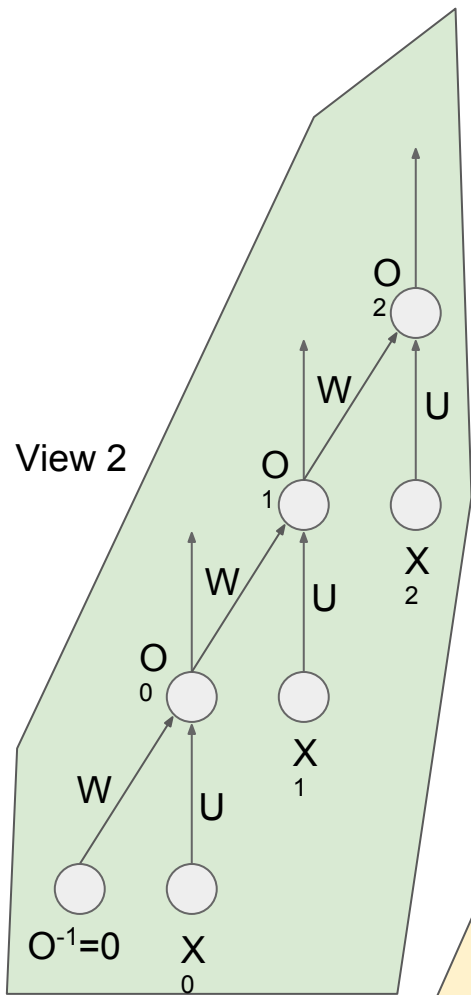
Complete Network



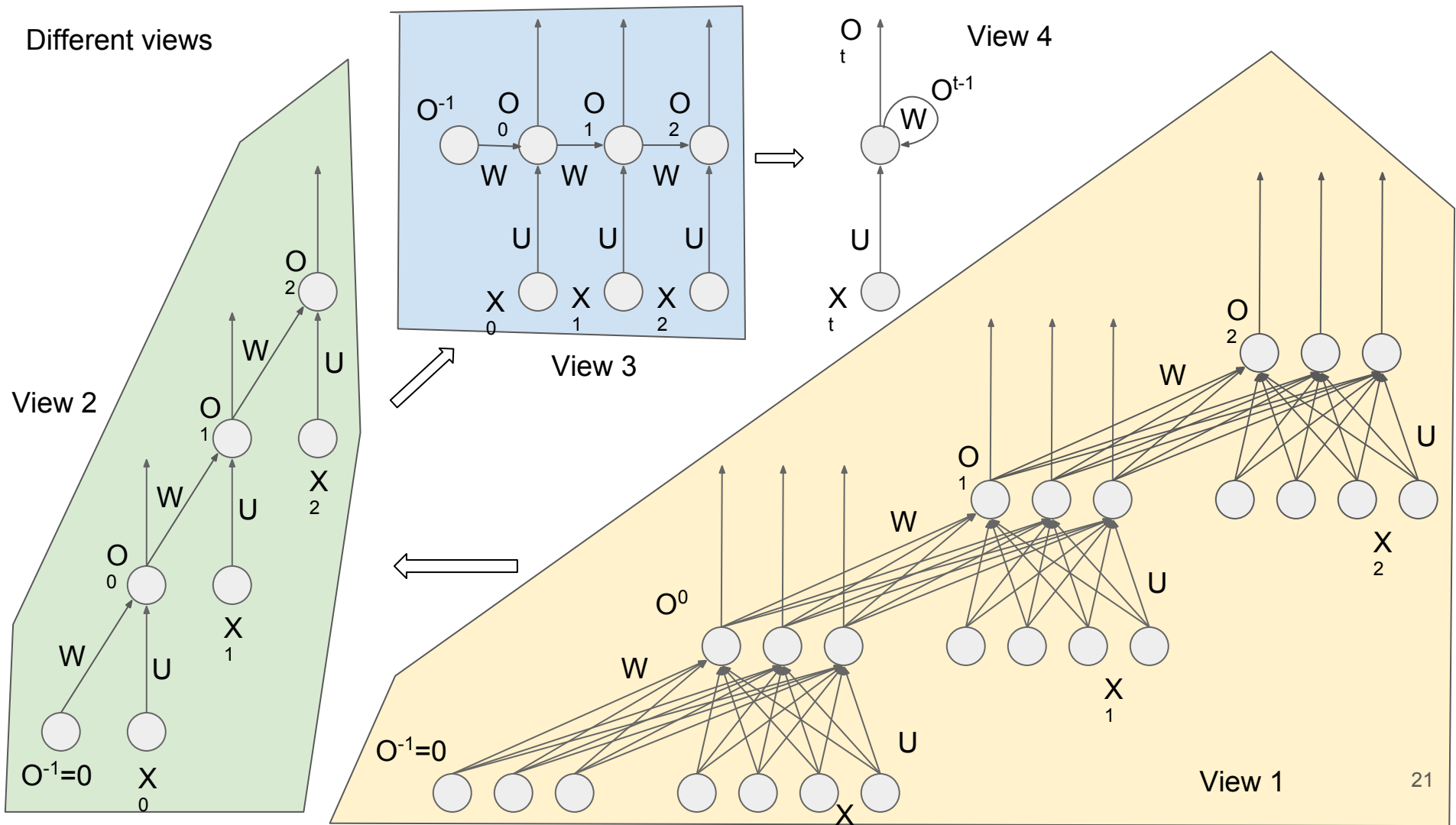
Different views of the network



Different views



Different views



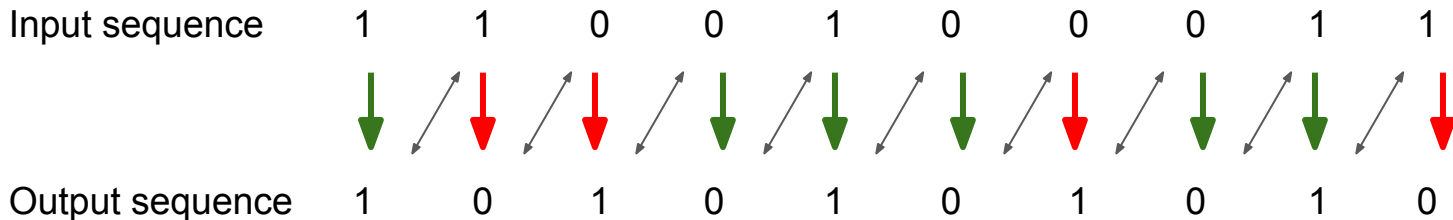
When to use RNNs

Usage

- Depends on the problems that we aim to solve.
- Typically good for sequence processings.
- Some sort of memorization is required.

Bit reverse problem

- Problem definition:
 - **Problem 1:** Reverse a binary digit.
 - $0 \rightarrow 1$ and $1 \rightarrow 0$
 - **Problem 2:** Reverse a sequence of binary digits.
 - $0101001 \rightarrow 1010110$
 - Sequence: Fixed or Variable length
 - **Problem 3:** Reverse a sequence of bits over time.
 - $0101001 \rightarrow 1010110$
 - **Problem 4:** Reverse a bit if the current i/p and previous o/p are same.



Data

Let

- **Problem 1**

- I/p dimension: **1 bit**

O/p dimension: **1 bit**

- **Problem 2**

- Fixed

- I/p dimension: **10 bit**

O/p dimension: **10 bit**

- Variable: Pad each sequence upto max sequence length: **10**

- Padding value: **-1**

- I/p dimension: **10 bit**

O/p dimension: **10 bit**

- **Problem 3 & 4**

- Dimension of each element of I/p (X) : **1 bit**

- Dimension of each element of O/p (O) : **1 bit**

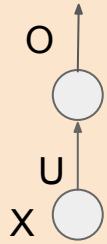
- Sequence length : **10**

Network Architecture

No. of I/p neurons = I/p dimension
 No. of O/p neurons = O/p dimension

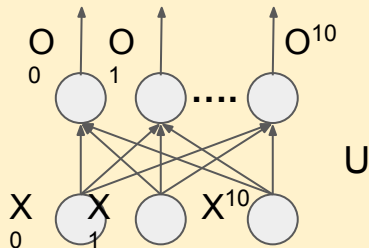
Problem 1:

- I/p neurons = 1
- O/p neurons = 1



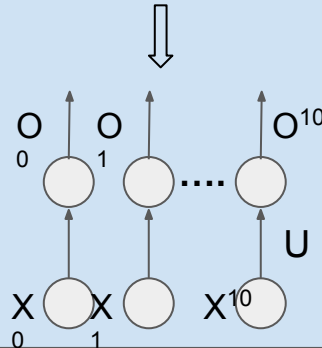
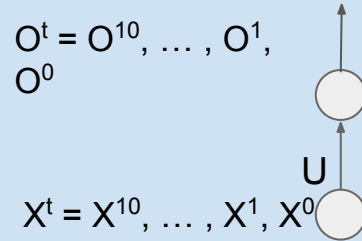
Problem 2: Fixed & Variable

- I/p neurons = 10
- O/p neurons = 10



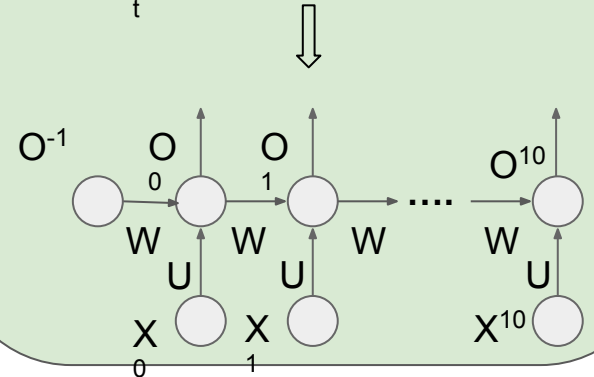
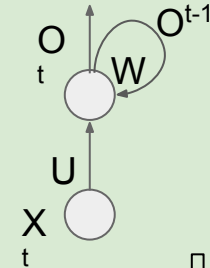
Problem 3:

- I/p neurons = 1
- O/p neurons = 1
- Seq len = 10



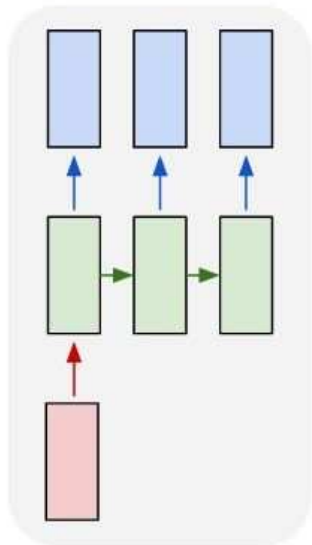
Problem 4:

- I/p neurons = 1
- O/p neurons = 1
- Seq len = 10



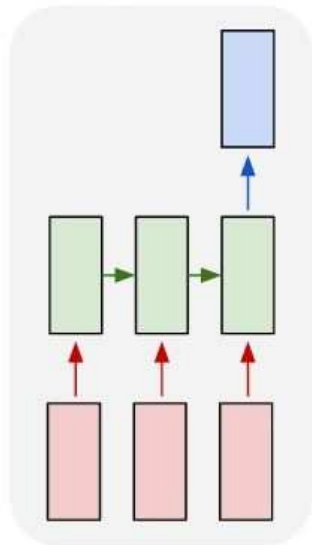
Different configurations of RNNs

one to many



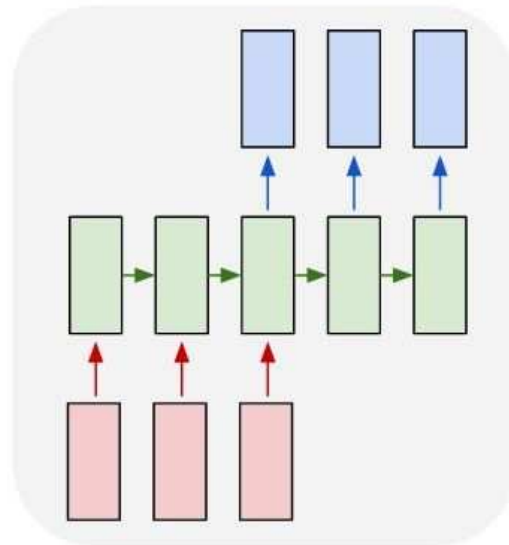
**Image
Captioning**

many to one



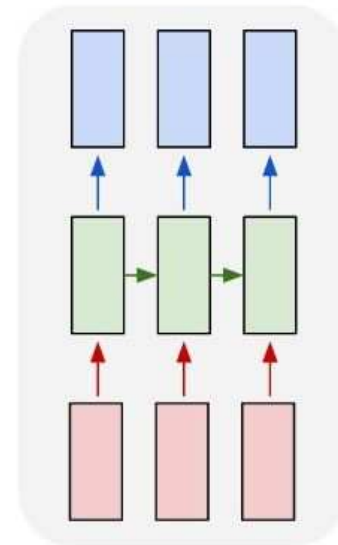
**Sentiment
Analysis**

many to many



**Machine
Translation**

many to many

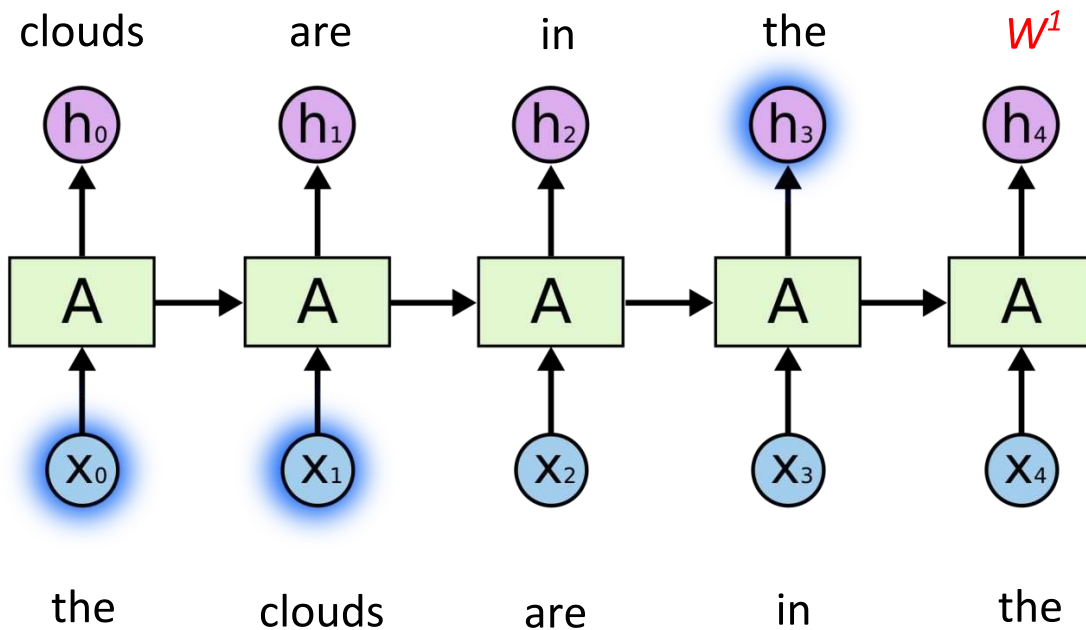


**Language
modelling**

Problems with RNNs

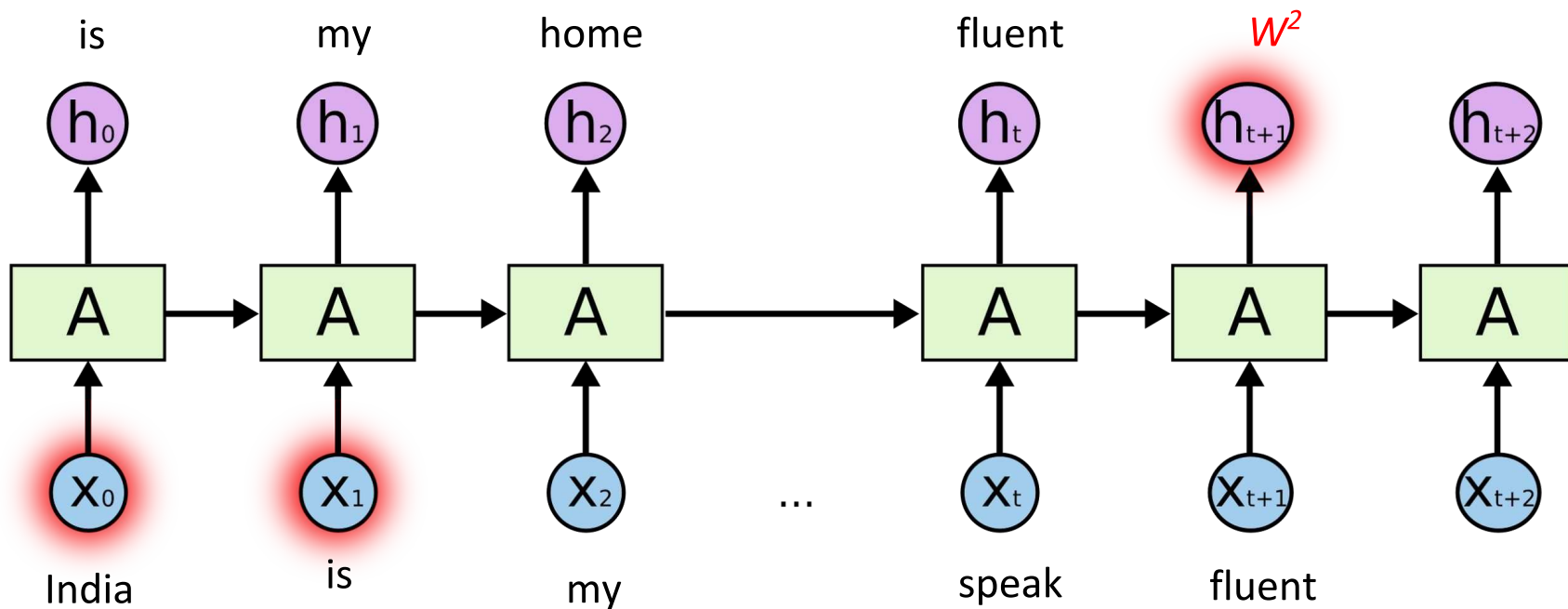
Language modelling: Example - 1

- “the clouds are in the *sky*”



Language modelling: Example - 2

- “India is my home country. I can speak fluent *Hindi*.”



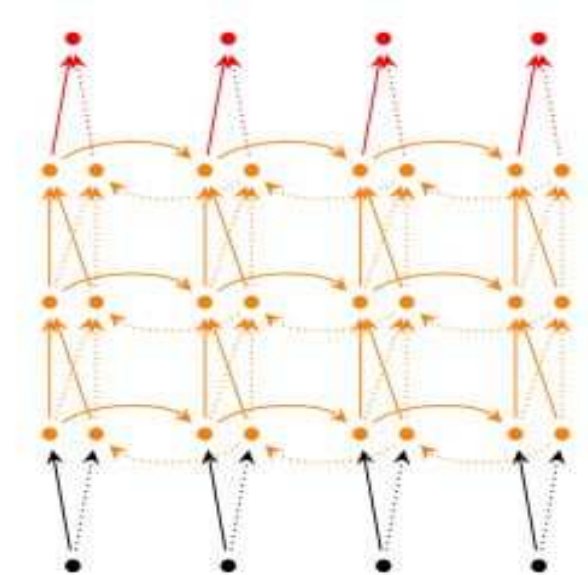
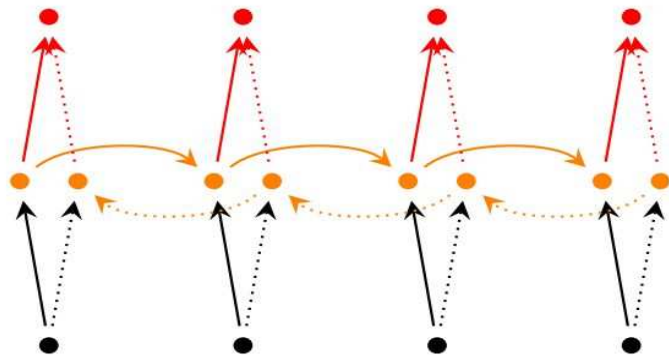
Vanishing/Exploding gradients

- Cue word for the prediction
 - Example 1: **sky** → **clouds** [3 units apart]
 - Example 2: **hindi** → **India** [9 units apart]

- As the sequence length increases, it becomes hard for RNNs to learn “long-term dependencies.”
 - **Vanishing gradients:** If weights are small, gradient shrinks exponentially. Network stops learning.
 - **Exploding gradients:** If weights are large, gradient grows exponentially. Weights fluctuate and become unstable.

RNN extensions

- Bi-directional RNN
- Deep (Bi-directional) RNN



Long Short Term Memory (LSTM)

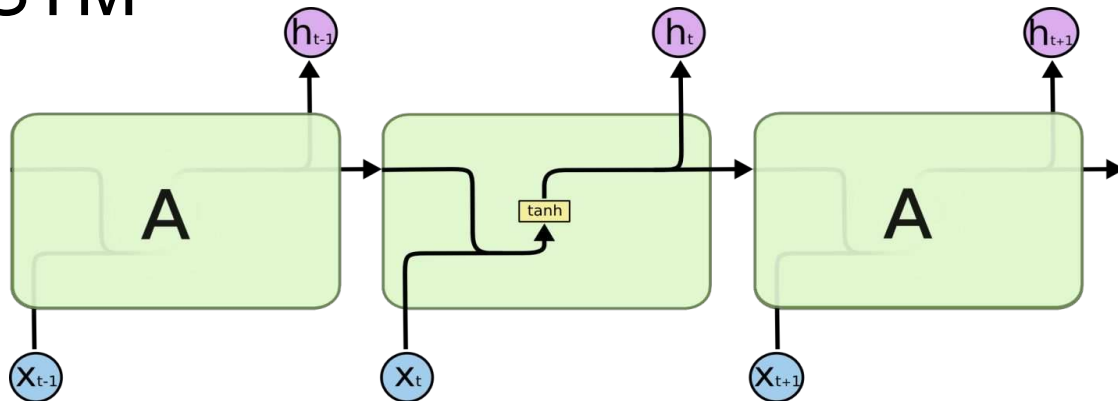
Hochreiter & Schmidhuber (1997)

LSTM

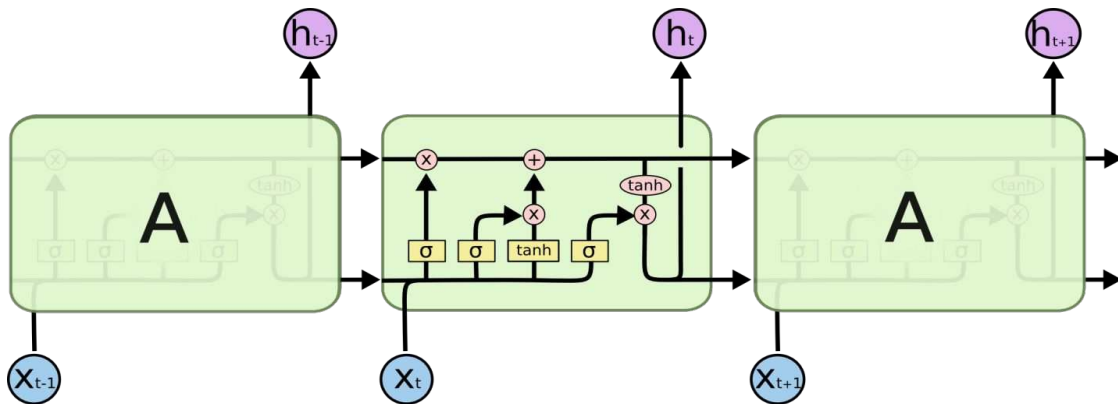
- A variant of simple RNN (Vanilla RNN)
- Capable of learning long dependencies.
- Regulates information flow from recurrent units.

Vanilla RNN vs LSTM

Vanilla RNN cell

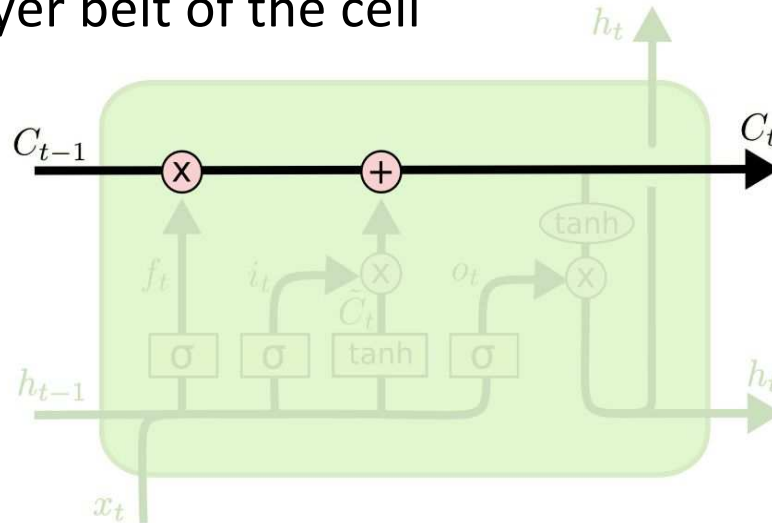


LSTM cell



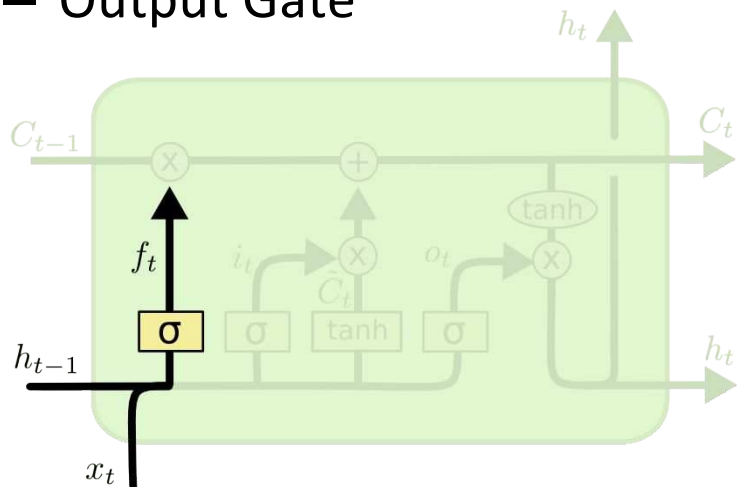
LSTM cell

- LSTM removes or adds information to the cell state, carefully regulated by structures called gates.
- Cell state: Conveyor belt of the cell



LSTM gates

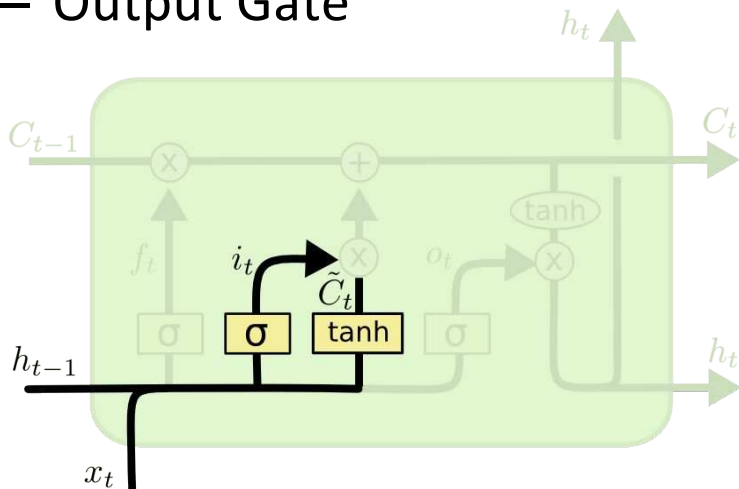
- Each LSTM unit comprises of three gates.
 - Forget Gate: Amount of memory it should forget.
 - Input Gate
 - Output Gate



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

LSTM gates

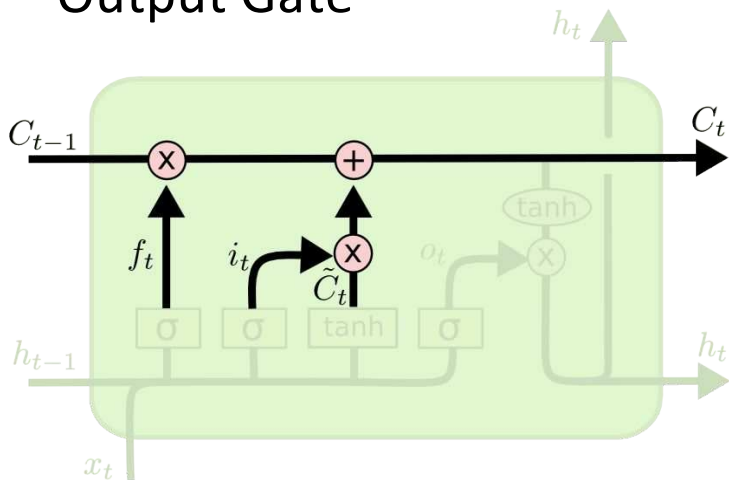
- Each LSTM unit comprises of three gates.
 - Forget Gate
 - **Input Gate: Amount of new information it should memorize.**
 - Output Gate



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

LSTM gates

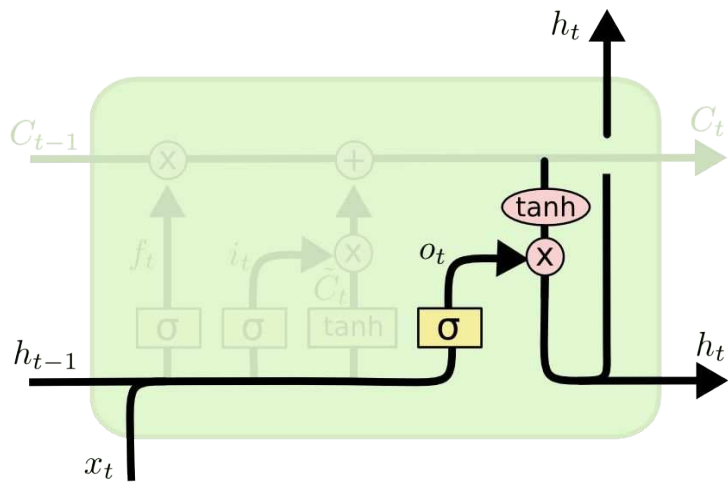
- Each LSTM unit comprises of three gates.
 - Forget Gate: Amount of memory it should forget.
 - Input Gate: Amount of new information it should memorize.
 - Output Gate



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

LSTM gates

- Each LSTM unit comprises of three gates.
 - Forget Gate
 - Input Gate
 - **Output Gate: Amount of information it should pass to next unit.**



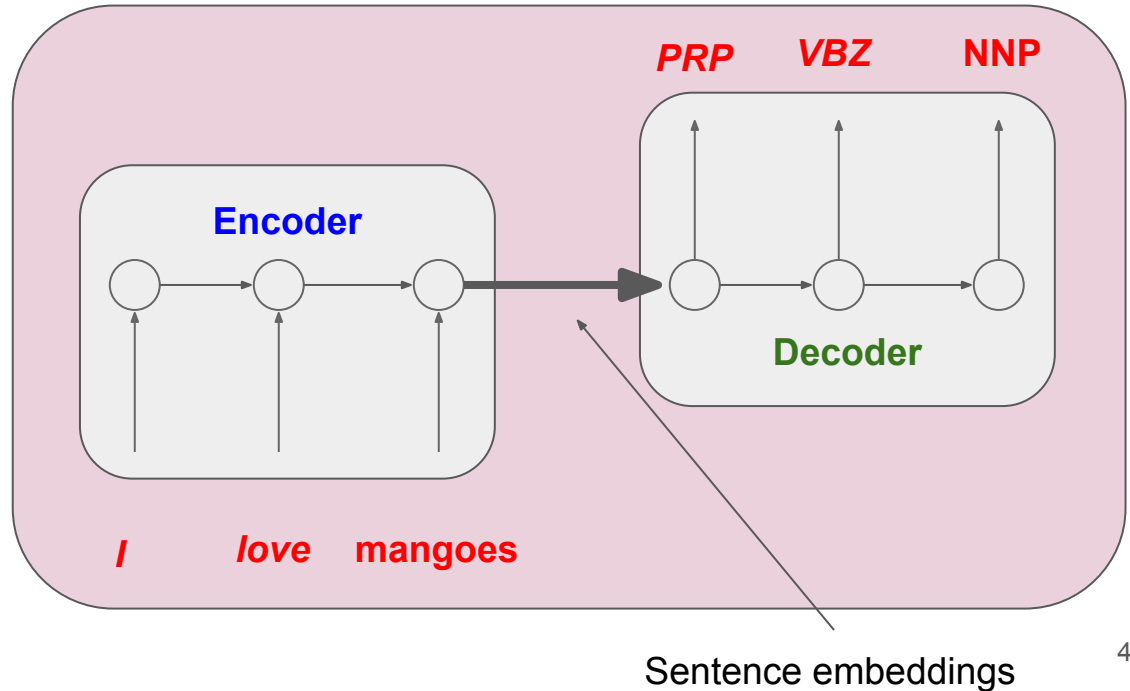
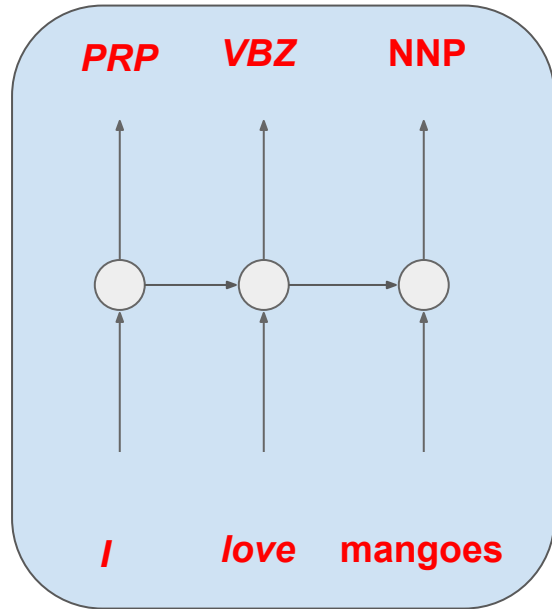
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

Sequence to sequence transformation with Attention Mechanism

Sequence labeling v/s Sequence transformation

- PoS Tagging



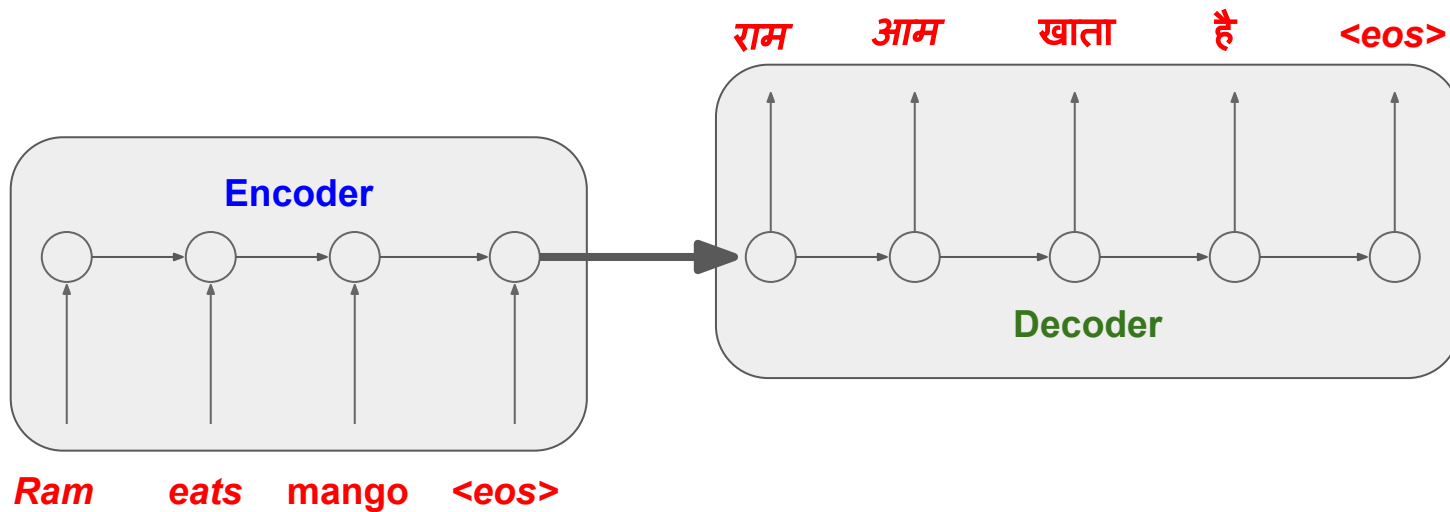
Why sequence transformation is required?

- For many application length of I/p and O/p are not necessarily same. E.g. Machine Translation, Summarization, Question Answering etc.
- For many application length of O/p is not known.
- Non-monotone mapping: Reordering of words.
- Applications like PoS tagging, Named Entity Recognition does not require these capabilities.

Encode-Decode paradigm

- English-Hindi Machine Translation

- Source sentence: 3 words
- Target sentence: 4 words
- Second word of the source sentence maps to 3rd & 4th words of the target sentence.
- Third word of the source sentence maps to 2nd word of the target sentence



Problems with Encode-Decode paradigm

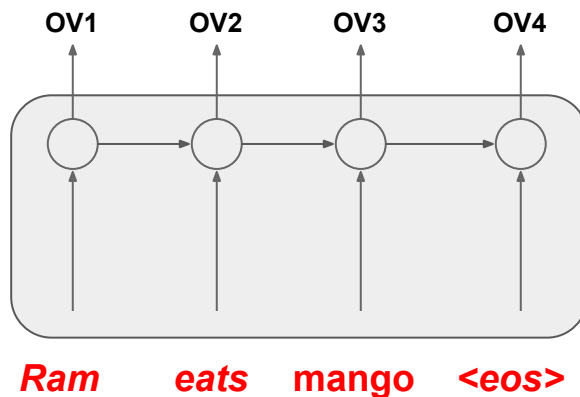
- Encoding transforms the entire sentence into a single vector.
- Decoding process uses this sentence representation for predicting the output.
 - Quality of prediction depends upon the quality of sentence embeddings.
- After few time steps decoding process may not properly use the sentence representation due to long-term dependency.
- To improve the quality of predictions we can
 - Improve the quality of sentence embeddings **'OR'**
 - Present the source sentence representation for prediction at each time step. **'OR'**
 - Present the RELEVANT source sentence representation for prediction at each time step.

Solutions

- To improve the quality of predictions we can
 - Improve the quality of sentence embeddings **'OR'**
 - Present the source sentence representation for prediction at each time step. **'OR'**
 - Present the RELEVANT source sentence representation for prediction at each time step.
 - *Encode - Attend - Decode* (Attention mechanism)

Attention Mechanism

- Represent the source sentence by the set of **output vectors** from the encoder.
- Each **output vector** (OV) at time t is a contextual representation of the input at time t .

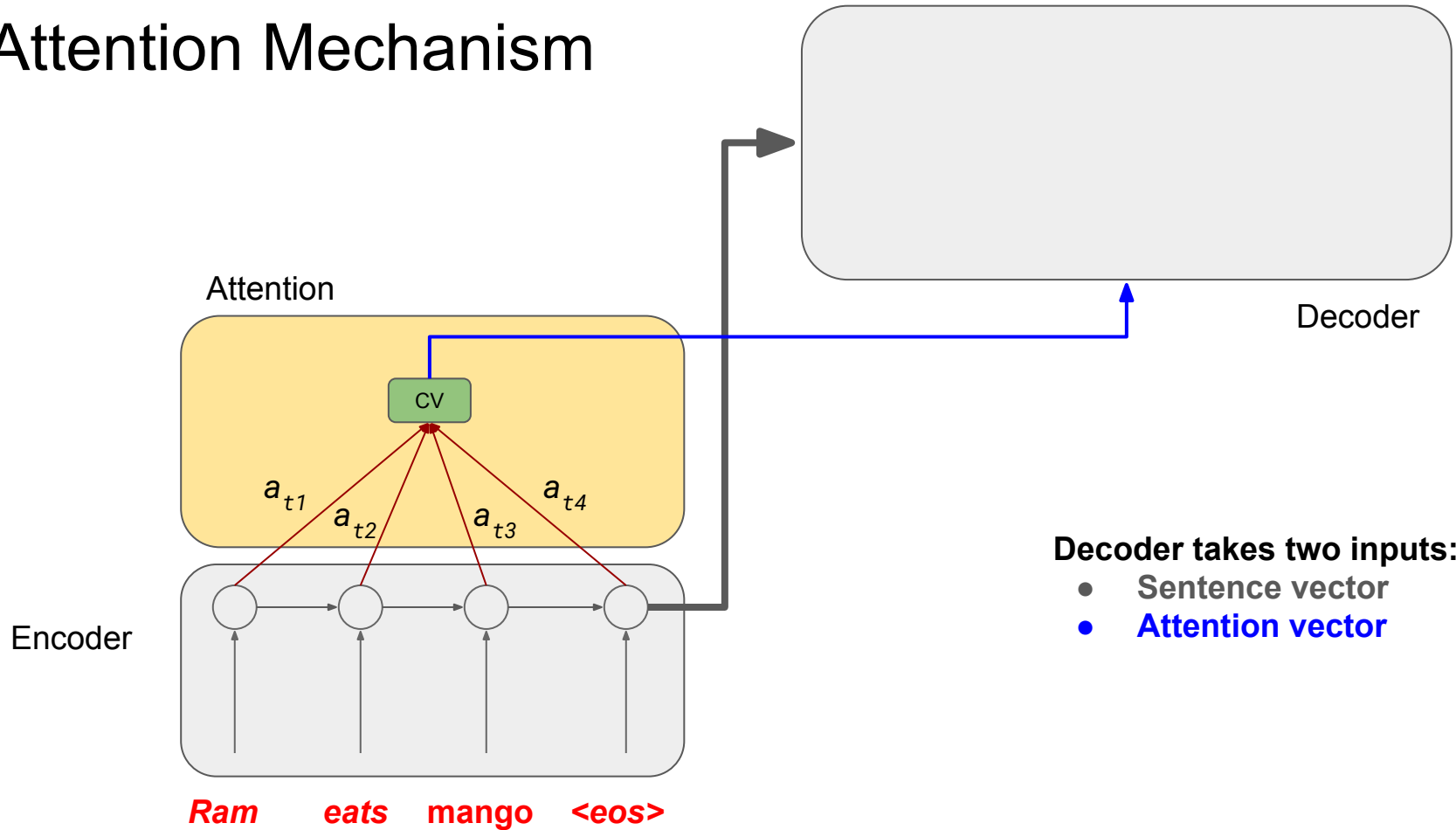


Attention Mechanism

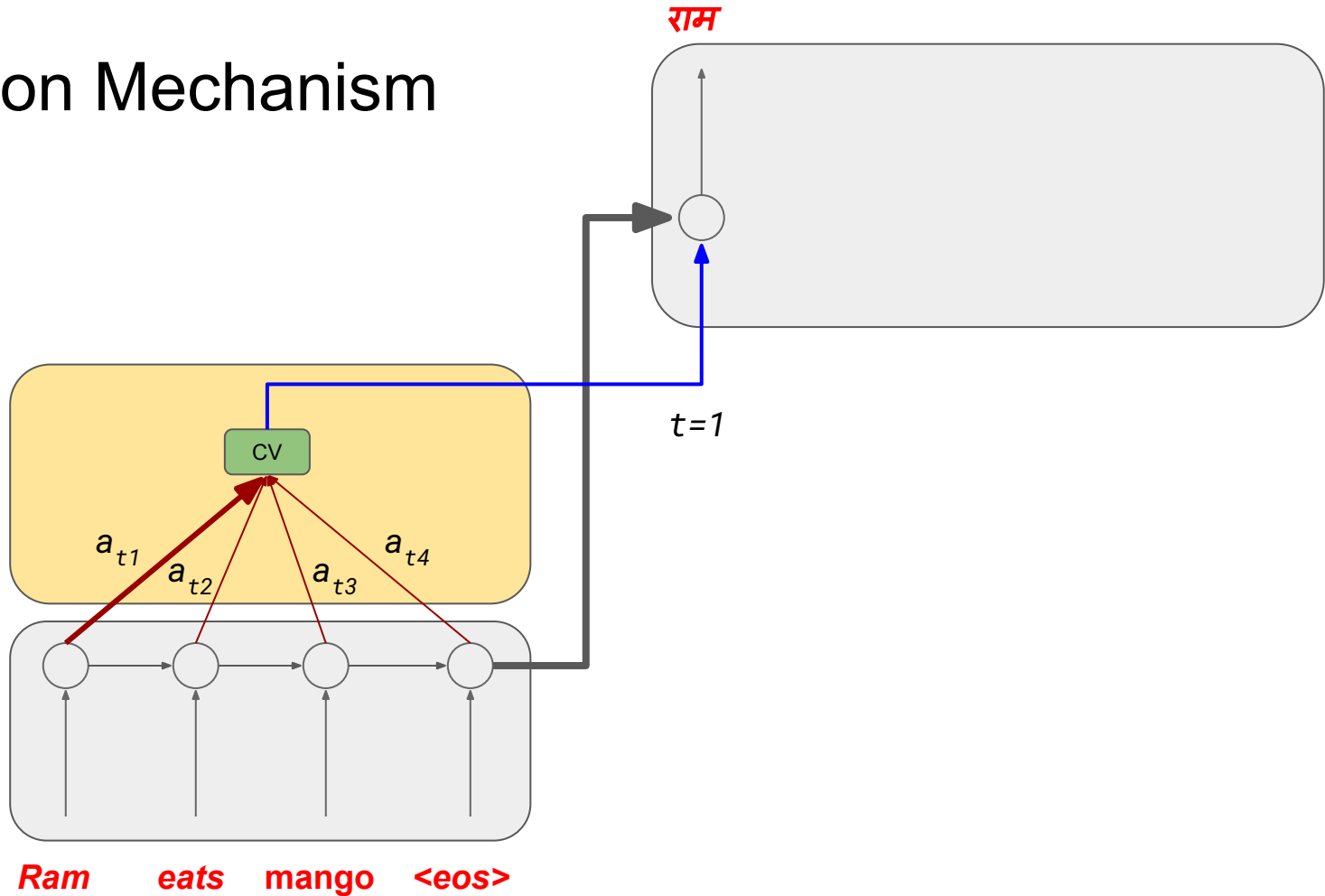
- Each of these output vectors (OVs) may not be equally relevant during decoding process at time t .
- Weighted average of the output vectors can resolve the relevancy.
 - Assign more weights to an output vector that needs more **attention** during decoding at time t .
- The weighted average **context vector (CV)** will be the input to decoder along with the sentence representation.
 - $CV_i = \sum a_{ij} \cdot OV_j$

where a_{ij} = weight of the j^{th} OV

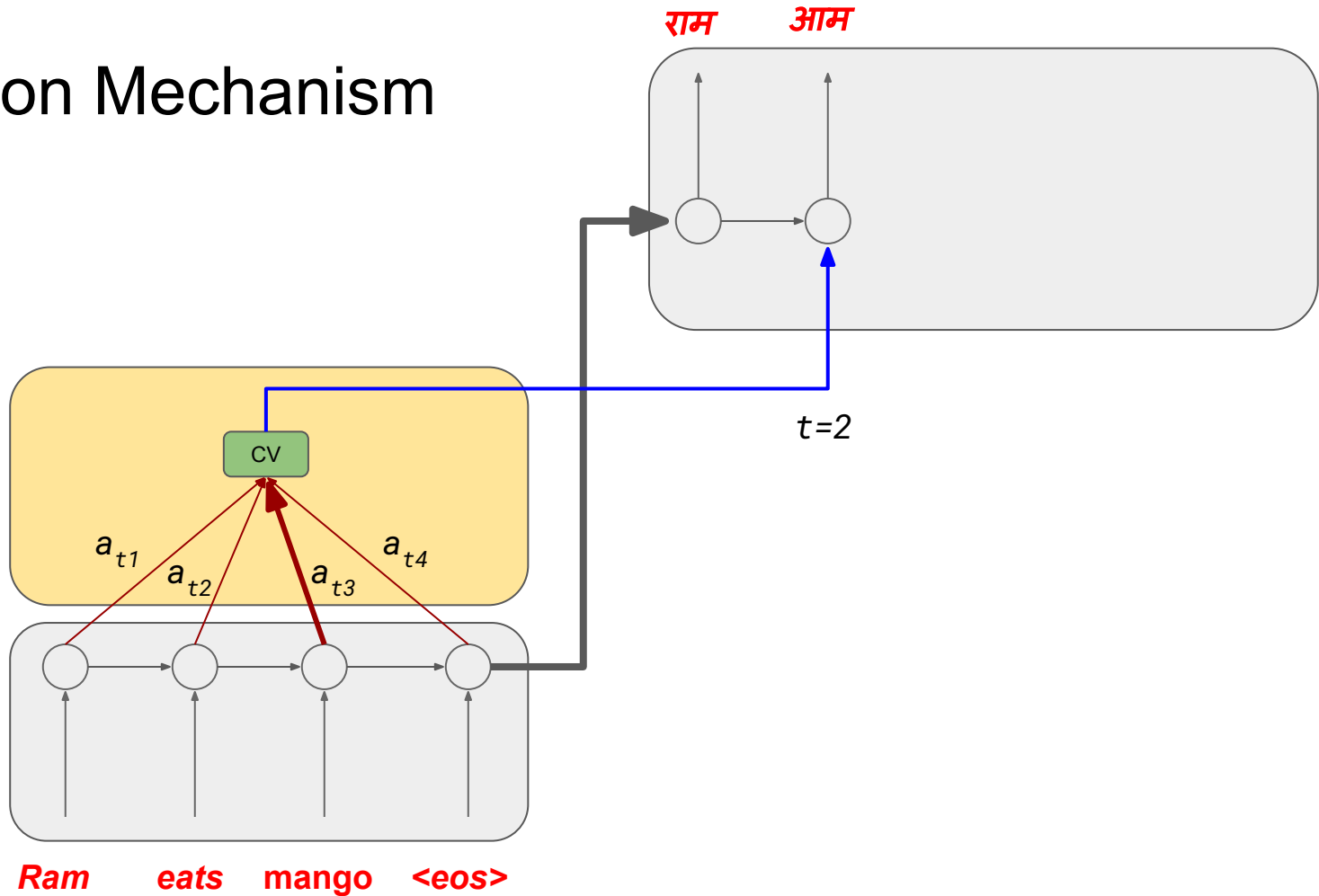
Attention Mechanism



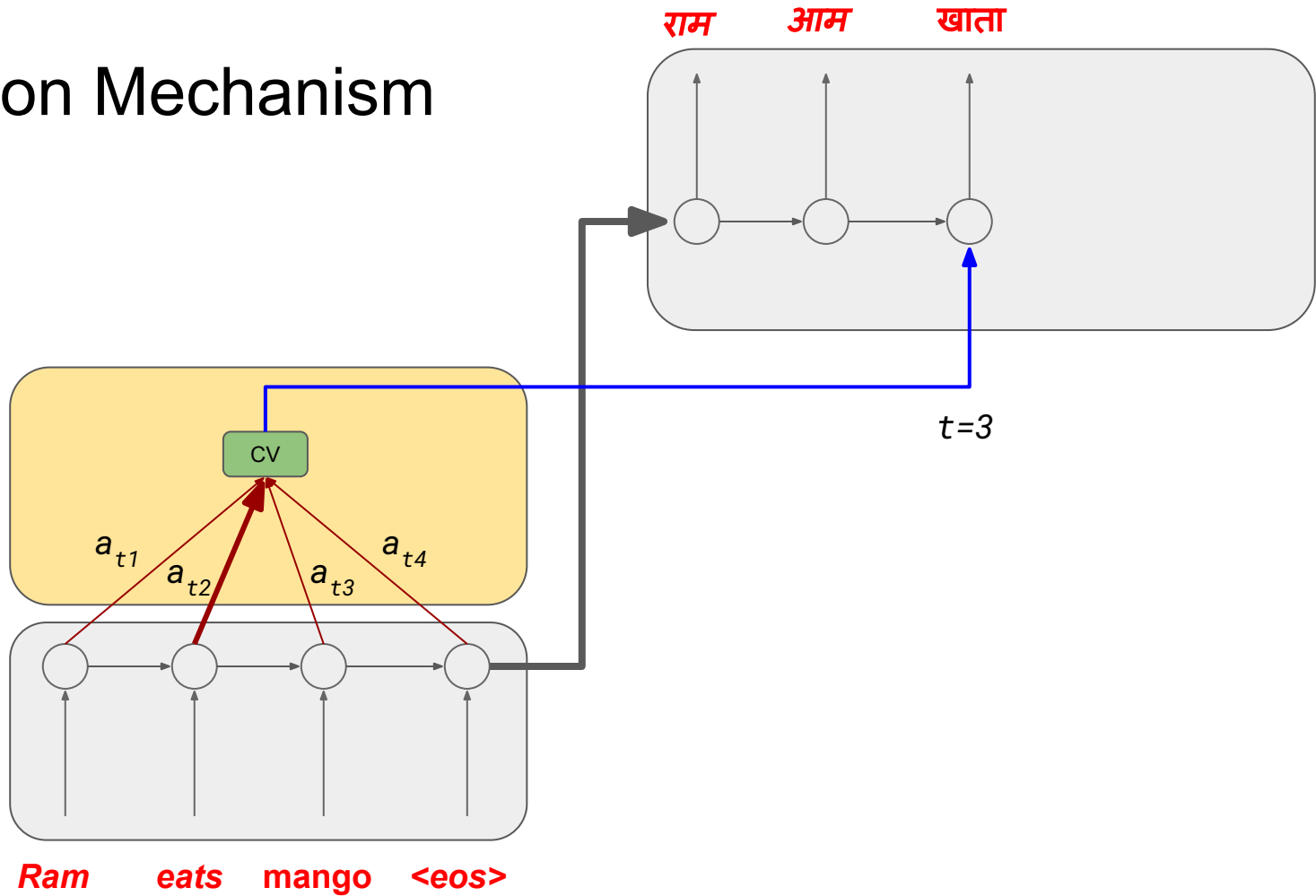
Attention Mechanism



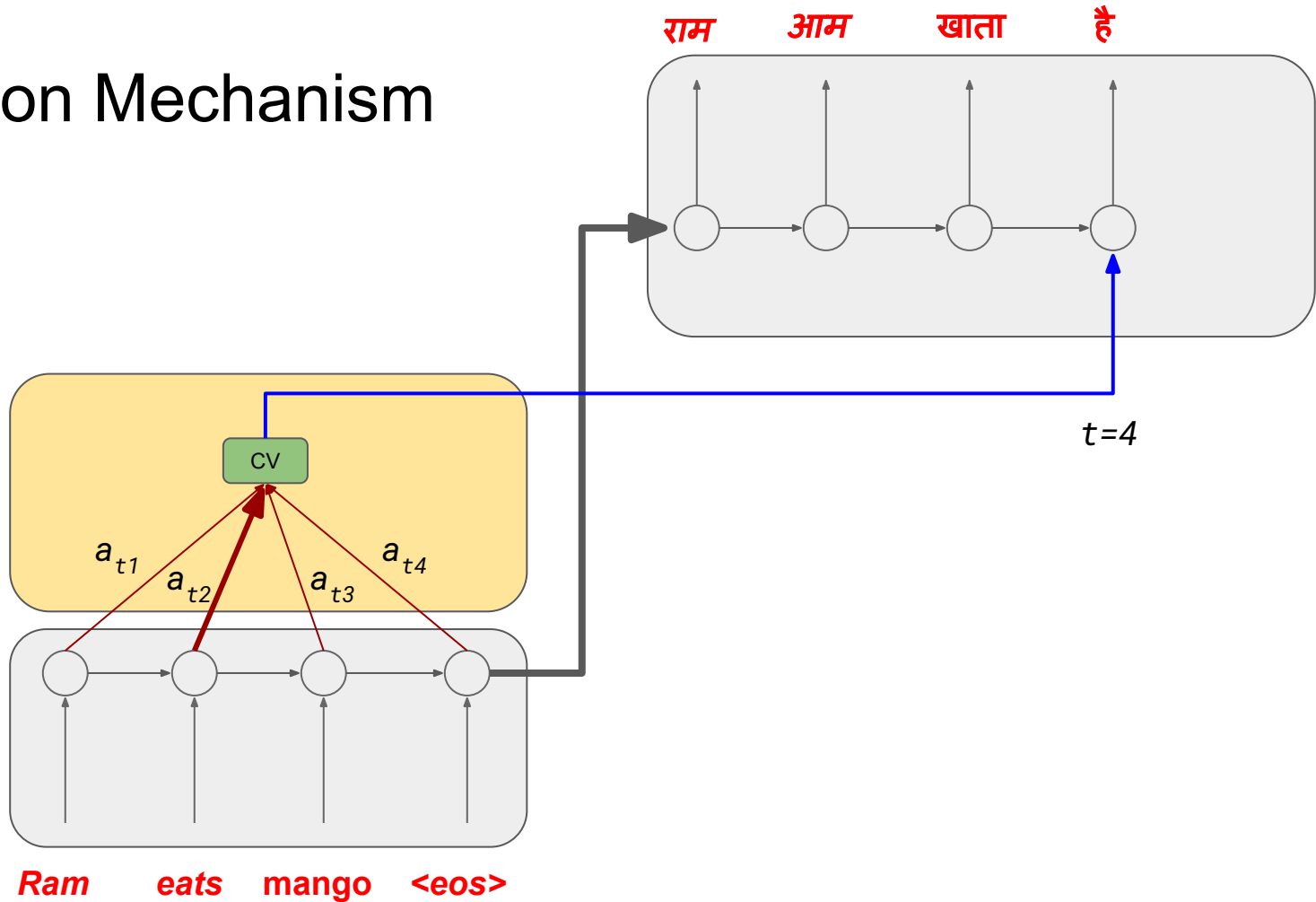
Attention Mechanism



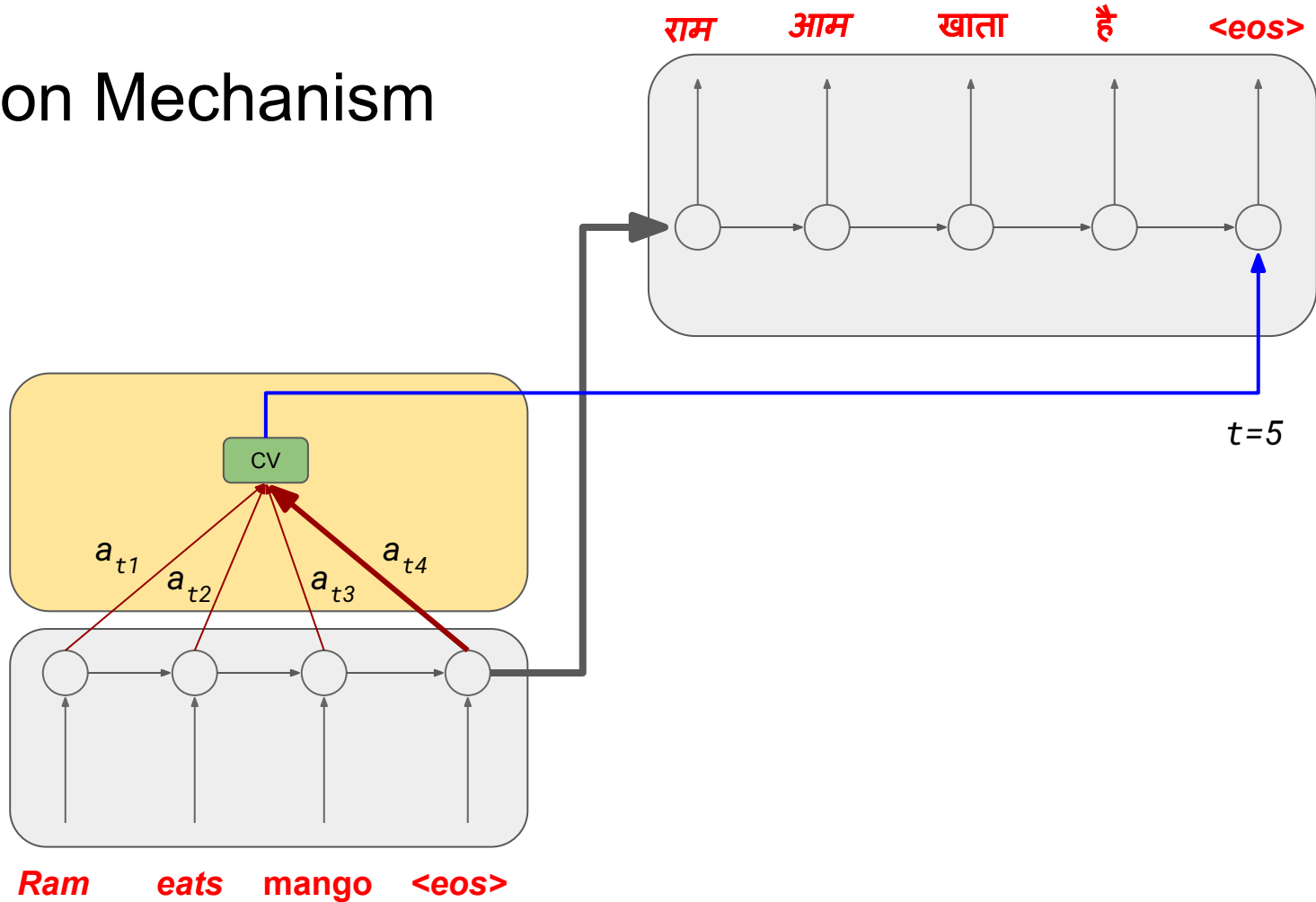
Attention Mechanism



Attention Mechanism



Attention Mechanism



Few good reads..

- Denny Britz; Recurrent Neural Networks Tutorial, Part 1-4
<http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>
- Andrej Karpathy; The Unreasonable Effectiveness of Recurrent Neural Networks
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- Chris Olah; Understanding LSTM Networks
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Thank You!

AI-NLP-ML Group, Department of CSE, IIT Patna (<http://www.iitp.ac.in/~ai-nlp-ml/>)

Research Supervisors:

- Prof. Pushpak Bhattacharyya
- Dr. Asif Ekbal
- Dr. Sriparna Saha

Siamese Networks For NLP

Rudra Murthy

Center for Indian Language Technology,
Indian Institute of Technology Bombay

rudra@cse.iitb.ac.in

<https://www.cse.iitb.ac.in/~rudra>



*Deep Learning Tutorial.
ICON 2017, Kolkata
21th December 2017*



Outline

- Motivation
- What are Siamese Networks?
- Siamese Networks and NLP
- Summary

Motivation

Motivation

- Given a (input, target) pairs, the goal is to learn a discriminative function which maps the input pattern to target label
- Some tasks like Face recognition have large number of classes
- Some classes have very less number of samples
- Supervised systems require large number of examples for each category

Usually approaches like k-nearest neighbors are used, where the test instance is compared with prototypes of each class

Motivation

Usually approaches like k-nearest neighbors are used, where the test instance is compared with prototypes of each class

- To use something like a nearest neighbour we need to define a **distance metric**
- Distance metric like *Euclidean distance* works well for well-defined features
- Given two facial images of the same person with slightly different orientation, euclidean distance fails

Siamese Networks

Goal: *Find a function f , which maps the input patterns to a target space such that Euclidean distance approximates the semantic distance in the input space [Chopra et.al 2005]*

- Euclidean space as a distance metric might fail in the input space
- However, if we map the input to some semantic space where the Euclidean distance is well-defined
- *The task is to learn the mapping from the data*

Siamese Networks

Goal: Find a function f , which maps the input patterns to a target space such that Euclidean distance approximates the semantic distance in the input space [Chopra et.al 2005]

There is a Deep Learning Tutorial at ICON

DL Tutorial is scheduled @ ICON

[1 1 1 1 1 1 1 1 0 0 0]

[0 1 0 0 0 1 0 1 1 1 1]

Euclidean Distance

2.8284

Siamese Networks

Goal: Find a function f , which maps the input patterns to a target space such that Euclidean distance approximates the semantic distance in the input space [Chopra et.al 2005]

There is a Deep Learning Tutorial at ICON

DL Tutorial is scheduled @ ICON

[1 1 1 1 1 1 1 1 0 0 0]

[0 1 0 0 0 1 0 1 1 1 1]

Euclidean Distance

2.8284

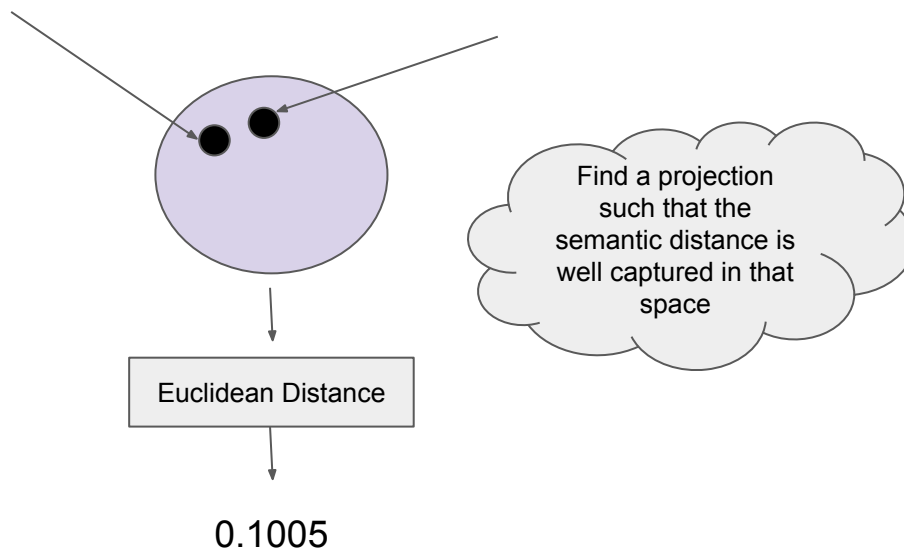
Accounts for
lexical overlap
between the
two sentences

Siamese Networks

Goal: Find a function f , which maps the input patterns to a target space such that Euclidean distance approximates the semantic distance in the input space [Chopra et.al 2005]

There is a Deep Learning Tutorial at ICON

DL Tutorial is scheduled @ ICON



What are Siamese Networks?

Siamese Networks

Goal: Find a function f , which maps the input patterns to a target space such that Euclidean distance approximates the semantic distance in the input space [Chopra et.al 2005]

Given examples X_1 and X_2 input examples, find a function parameterized by W such that,

$$E_W(X_1, X_2) = ||G_W(X_1) - G_W(X_2)||$$

The distance is small for examples from same category and large for examples from different category

Siamese Networks

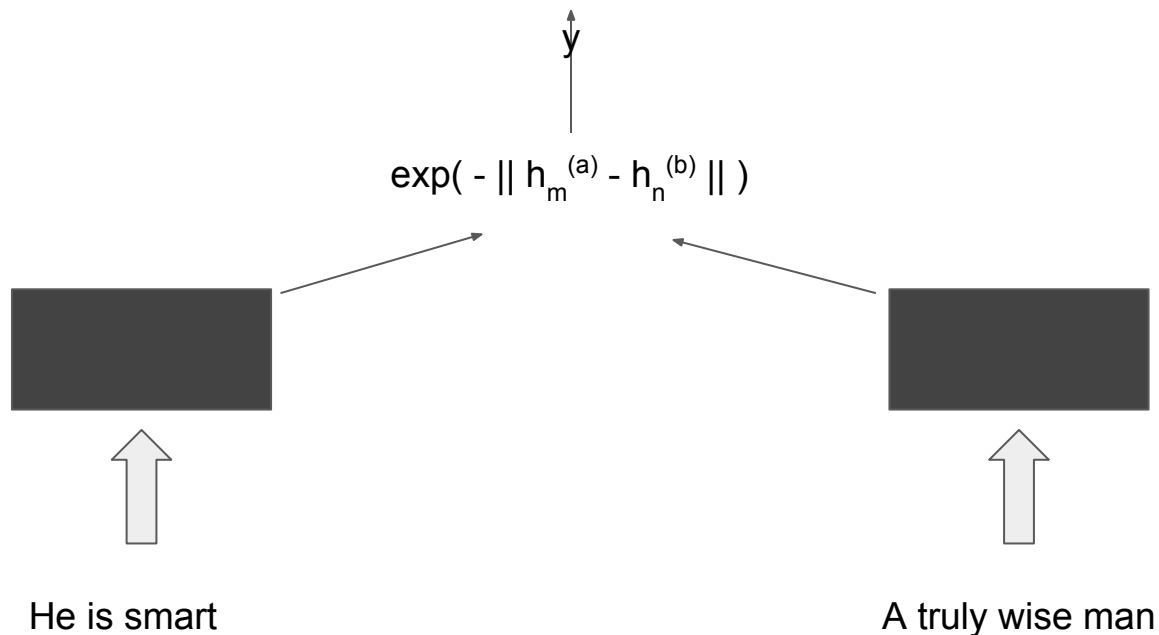
Testing [Chopra et.al 2005]

- The network would learn a function which projects the inputs patterns to a space where semantic distance between instances are well-defined
- We can obtain semantic vectors for all instances belonging to a particular category
- Assume the semantic vector forms a multivariate normal density
- Construct a model for every category by taking the mean semantic vector and the variance-covariance matrix
- For every test instance, we get a probability score
- Define imposter instances (negative instances) and calculate the probability score for all imposter instances and take their average
- The final prediction depends on how well the test image is far to the imposter instances

Applications to NLP Tasks

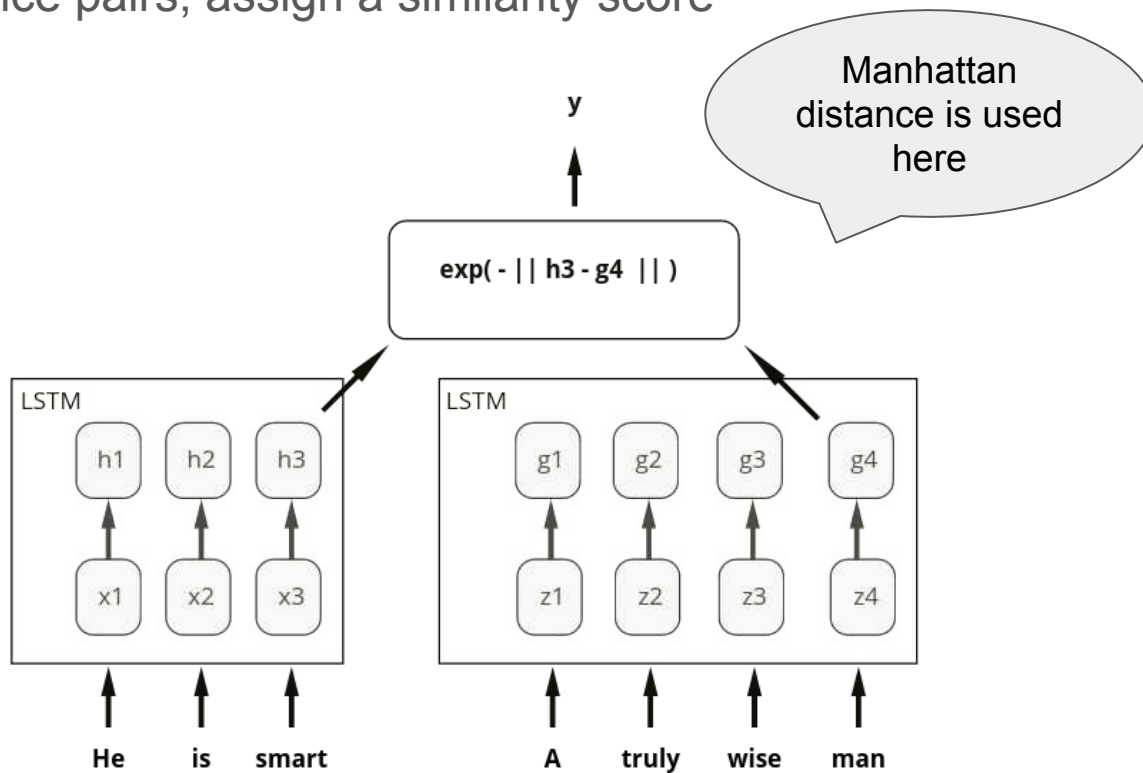
Sentence Similarity [Mueller et.al 2016]

Task: Given sentence pairs, assign a similarity score



Sentence Similarity [Mueller et.al 2016]

Task: Given sentence pairs, assign a similarity score



Sentence Similarity [Mueller et.al 2016]

- The model was trained on Sentence Similarity task using SICK dataset
- The dataset contains 9927 sentence pairs each annotated with relatedness label $\in [1,5]$
- Each pair was judged by 10 annotators and the average relatedness was taken as the similarity judgement

Sentence Similarity [Mueller et.al 2016]

- The model was trained on Sentence Similarity task using SICK dataset
- The dataset contains 9927 sentence pairs each annotated with relatedness label $\in [1,5]$
- Each pair was judged by 10 annotators and the average relatedness was taken as the similarity judgement

Sentence Similarity [Mueller et.al 2016]

Network Architecture

- Google pre-trained word embeddings were used
- Words in the sentences were randomly replaced by one of their synonyms from Wordnet
- The LSTM uses 50-dimensional hidden representations
- Adadelta with gradient clipping were used for training

Sentence Similarity [Mueller et.al 2016]

Results

Method	r	ρ	MSE
Illinois-LH (Lai and Hockenmaier 2014)	0.7993	0.7538	0.3692
UNAL-NLP (Jimenez et al. 2014)	0.8070	0.7489	0.3550
Meaning Factory (Bjerva et al. 2014)	0.8268	0.7721	0.3224
ECNU (Zhao, Zhu, and Lan 2014)	0.8414	–	–
Skip-thought+COCO (Kiros et al. 2015)	0.8655	0.7995	0.2561
Dependency Tree-LSTM (Tai, Socher, and Manning 2015)	0.8676	0.8083	0.2532
ConvNet (He, Gimpel, and Lin 2015)	0.8686	0.8047	0.2606
MaLSTM	0.8822	0.8345	0.2286

Table 2: Test set Pearson correlation (r), Spearman’s ρ , and mean squared error for the SICK semantic textual similarity task. The first group of results are top SemEval 2014 submissions and the second group are recent neural network methods (best result from each paper shown).

Sentence Similarity [Mueller et.al 2016]

Results

Ranking by Dependency Tree-LSTM Model	Tree	M
a woman is slicing potatoes		
a woman is cutting potatoes	4.82	4.87
potatoes are being sliced by a woman	4.70	4.38
tofu is being sliced by a woman	4.39	3.51
a boy is waving at some young runners from the ocean		
a group of men is playing with a ball on the beach	3.79	3.13
a young boy wearing a red swimsuit is jumping out of a blue kiddies pool	3.37	3.48
the man is tossing a kid into the swimming pool that is near the ocean	3.19	2.26
two men are playing guitar		
the man is singing and playing the guitar	4.08	3.53
the man is opening the guitar for donations and plays with the case	4.01	2.30
two men are dancing and singing in front of a crowd	4.00	2.33

Table 3: Most similar sentences (from 1000-sentence sub-sample) in the SICK test data according to the Tree-LSTM. Tree / M denote relatedness (with the sentence preceding each group) predicted by the Tree-LSTM / MaLSTM.

Sentence Similarity [Mueller et.al 2016]

Entailment Classification

- Applied the model on SemEval 2014 textual entailment task
- The obtained representation from the LSTM layer is used as features to a classification system
- The LSTM layer was already trained on the sentence similarity task
- Given a pair of sentences, first the representations $h_m^{(a)}$ and $h_n^{(b)}$ are obtained from the LSTM layer
- The element-wise absolute difference $|h_m^{(a)} - h_n^{(b)}|$ and element-wise dot product $h_m^{(a)} \odot h_n^{(b)}$ are used as features to a radial basis SVM classifier
- The LSTM layer parameters are not updated during training

Sentence Similarity [Mueller et.al 2016]

Entailment Classification Results

Method	Accuracy
Illinois-LH (Lai and Hockenmaier 2014)	84.6
ECNU (Zhao, Zhu, and Lan 2014)	83.6
UNAL-NLP (Jimenez et al. 2014)	83.1
Meaning Factory (Bjerva et al. 2014)	81.6
Reasoning-based n-best (Lien and Kouylekov 2015)	80.4
LangPro Hybrid-800 (Abzianidze 2015)	81.4
SNLI-transfer 3-class LSTM (Bowman et al. 2015)	80.8
MaLSTM features + SVM	84.2

Table 4: Test set accuracy for the SICK semantic entailment classification. The first group of results are top SemEval 2014 submissions and the second are more recently proposed methods.

Thank You

Questions?

References

- Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "Siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS'93)*, San Francisco, CA, USA
- Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* - IEEE Computer Society, Washington, DC, USA
- Mueller, J and Thyagarajan, A. Siamese Recurrent Architectures for Learning Sentence Similarity. Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)

Part 2: Word Embeddings

Word Embeddings

Deep Learning for NLP

Kevin Patel

ICON 2017

December 21, 2017

Outline

- 1 Introduction
- 2 Word Embeddings
 - Count Based Embeddings
 - Prediction Based Embeddings
 - Multilingual Word Embeddings
 - Interpretable Word Embeddings
- 3 Evaluating Word Embeddings
 - Intrinsic Evaluation
 - Extrinsic Evaluation
 - Evaluation Frameworks
 - Visualizing Word Embeddings

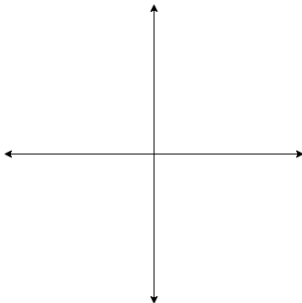
- 4 Discussion on Lower Bounds
- 5 Applications of Word Embeddings
 - Are Word Embeddings Useful for Sarcasm Detection?
 - Iterative Unsupervised Most Frequent Sense Detection using Word Embeddings
- 6 Conclusion

Layman(ish) Intro to ML

- In simple terms, Machine Learning comprises of
 - Representing data in some numeric form
 - Learning some function on that representation

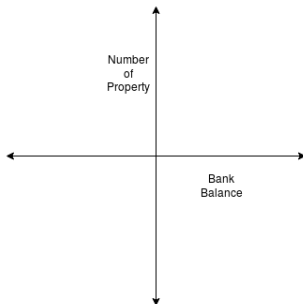
Layman(ish) Intro to ML

- In simple terms, Machine Learning comprises of
 - Representing data in some numeric form
 - Learning some function on that representation



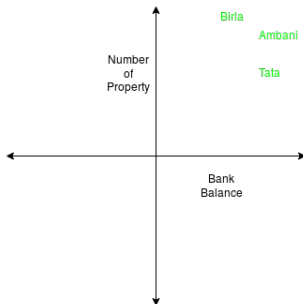
Layman(ish) Intro to ML

- In simple terms, Machine Learning comprises of
 - Representing data in some numeric form
 - Learning some function on that representation



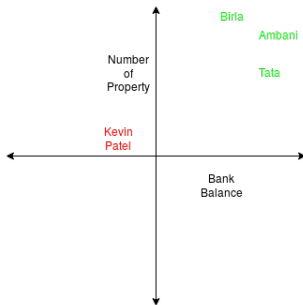
Layman(ish) Intro to ML

- In simple terms, Machine Learning comprises of
 - Representing data in some numeric form
 - Learning some function on that representation



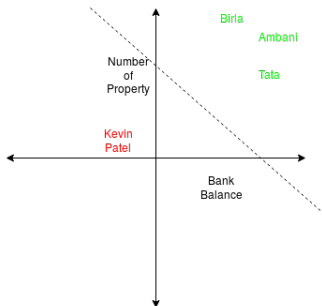
Layman(ish) Intro to ML

- In simple terms, Machine Learning comprises of
 - Representing data in some numeric form
 - Learning some function on that representation



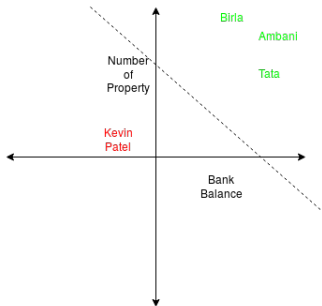
Layman(ish) Intro to ML

- In simple terms, Machine Learning comprises of
 - Representing data in some numeric form
 - Learning some function on that representation



Layman(ish) Intro to ML

- In simple terms, Machine Learning comprises of
 - Representing data in some numeric form
 - Learning some function on that representation



- How to place words to learn, say, Binary Sentiment Classification?
 - Good: Positive
 - Awesome: Positive
 - Bad: Negative

Representations for Learning Algorithms

- Detect whether the following image is dog or not?



- Basic idea: feed raw pixels as input vector
- Works well:
 - Inherent structure in the image
- Detect whether a word is a dog or not?

Labrador

- Nothing in spelling of *labrador* that can connect it to *dog*
- Need a representation of *labrador* which indicates that it is a dog

Local Representations

- Information about a particular item located solely in the corresponding representational element (dimension)
- Effectively one unit is turned on in a network, all the others are off
- No sharing between represented data
- Each feature is independent
- No generalization on the basis of similarity between features

Distributed Representations

- Information about a particular item distributed among a set of (not necessarily) mutually exclusive representational elements (dimensions)
 - One item spread over multiple dimensions
 - One dimension contributing to multiple items
- A new input is processed similar to samples in training data which were similar
 - Better generalization

Distributed Representations: Example

Number	Local Representation	Distributed Representation
0	1 0 0 0 0 0 0 0	0 0 0
1	0 1 0 0 0 0 0 0	0 0 1
2	0 0 1 0 0 0 0 0	0 1 0
3	0 0 0 1 0 0 0 0	0 1 1
4	0 0 0 0 1 0 0 0	1 0 0
5	0 0 0 0 0 1 0 0	1 0 1
6	0 0 0 0 0 0 1 0	1 1 0
7	0 0 0 0 0 0 0 1	1 1 1

Word Embeddings : Intuition

- Word Embeddings: distributed vector representations of words such that the similarity among vectors correlate with semantic similarity among the corresponding words

Given that $\text{sim}(\text{dog}, \text{cat})$ is more than $\text{sim}(\text{dog}, \text{furniture})$,
 $\cos(\vec{\text{dog}}, \vec{\text{cat}})$ is greater than $\cos(\vec{\text{dog}}, \vec{\text{furniture}})$

- Such similarity information uncovered from context

Word Embeddings : Intuition

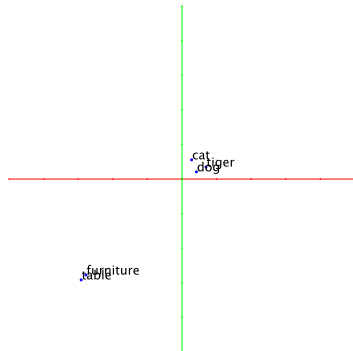
- Word Embeddings: distributed vector representations of words such that the similarity among vectors correlate with semantic similarity among the corresponding words

Given that $\text{sim}(\text{dog}, \text{cat})$ is more than $\text{sim}(\text{dog}, \text{furniture})$,
 $\cos(\vec{\text{dog}}, \vec{\text{cat}})$ is greater than $\cos(\vec{\text{dog}}, \vec{\text{furniture}})$

- Such similarity information uncovered from context
- Consider the following sentences:
 - I like sweet food .
 - You like spicy food .
 - They like *xyzabc* food .
- What is *xyzabc* ?
- Meaning of words can be inferred from their neighbors (context) and words that share neighbors
 - Neighbors of *xyzabc*: $\{\textit{like}, \textit{food}\}$
 - Words that share neighbors of *xyzabc*: $\{\textit{sweet}, \textit{spicy}\}$

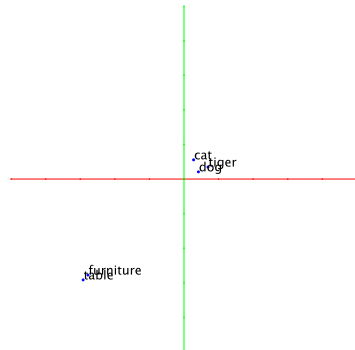
Modelling Meaning via Word Embeddings

- Geometric metaphor of meaning (Sahlgren, 2006):
 - Meanings are locations in semantic space, and semantic similarity is proximity between the locations.



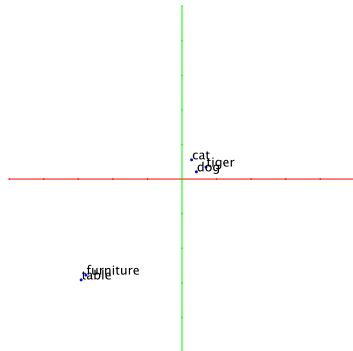
Modelling Meaning via Word Embeddings

- Geometric metaphor of meaning (Sahlgren, 2006):
 - Meanings are locations in semantic space, and semantic similarity is proximity between the locations.
- Distributional Hypothesis (Harris, 1970)
 - Words with similar distributional properties have similar meanings



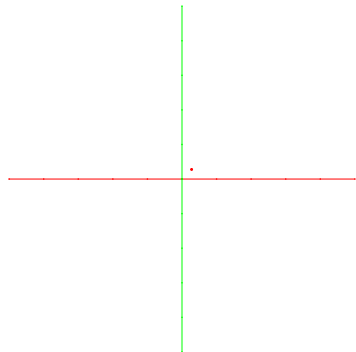
Modelling Meaning via Word Embeddings

- Geometric metaphor of meaning (Sahlgren, 2006):
 - Meanings are locations in semantic space, and semantic similarity is proximity between the locations.
- Distributional Hypothesis (Harris, 1970)
 - Words with similar distributional properties have similar meanings
 - **Only differences in meaning can be modelled**



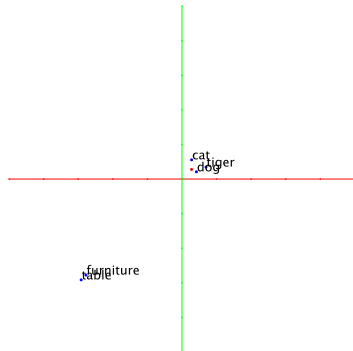
Modelling Meaning via Word Embeddings

- Geometric metaphor of meaning (Sahlgren, 2006):
 - Meanings are locations in semantic space, and semantic similarity is proximity between the locations.
- Distributional Hypothesis (Harris, 1970)
 - Words with similar distributional properties have similar meanings
 - **Only differences in meaning can be modelled**



Modelling Meaning via Word Embeddings

- Geometric metaphor of meaning (Sahlgren, 2006):
 - Meanings are locations in semantic space, and semantic similarity is proximity between the locations.
- Distributional Hypothesis (Harris, 1970)
 - Words with similar distributional properties have similar meanings
 - **Only differences in meaning can be modelled**



Entire Vector vs. Individual dimensions

- Only proximity in the entire space is represented
- No phenomenological correlations with dimensions of high-dimensional space (in majority of algorithms)
 - Those models who do have some correlations, are known as *interpretable models*

Modelling Meaning via Word Embeddings

- Co-occurrence matrix (Rubenstein and Goodenough, 1965)
 - A mechanism to capture distributional properties
 - Rows of co-occurrence matrix can be directly considered as word vectors
- Neural Word Embeddings
 - Vector representations learnt using neural networks - Bengio et al. (2003); Collobert and Weston (2008a); Mikolov et al. (2013b)

Co-occurrence Matrix

- Originally proposed by Schütze (1992)
- Foundation of count based approaches that follow
- Automatic derivation of vectors
- Collect co-occurrence counts in a matrix
- Rows or columns are the vectors of corresponding word
- If counting in both directions, matrix is symmetrical
- If counting in one side, matrix is asymmetrical, and is known as directional co-occurrence

Co-occurrence Matrix (contd.)

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>

Co-occurrence Matrix

word	<>	I	like	love	hate	rate	rats	cats	dogs	bats
<>	0	4	0	0	0	0	1	1	1	1
I	4	0	1	1	1	1	0	0	0	0
like	0	1	0	0	0	0	0	1	0	0
love	0	1	0	0	0	0	0	0	1	0
hate	0	1	0	0	0	0	1	0	0	0
rate	0	1	0	0	0	0	0	0	0	1
rats	1	0	0	0	1	0	0	0	0	0
cats	1	0	1	0	0	0	0	0	0	0
dogs	1	0	0	1	0	0	0	0	0	0
bats	1	0	0	0	0	1	0	0	0	0

Word Embeddings

Count Based Embeddings

LSA

- Latent Semantic Analysis
- Originally developed as Latent Semantic Indexing (LSI) (Dumais et al., 1988)
- Adapted for word-space models
- Developed to tackle inability of models of co-occurrence matrices to handle synonymy
 - Query about *hotels* cannot retrieve results about *motels*
- Words and Documents dimensions \rightarrow Latent dimensions
 - Uses Singular Value Decomposition (SVD) for dimensionality reduction

LSA (contd.)

- Words-by-documents matrix
- Entropy based weighting of co-occurrences

$$f_{ij} = (\log(TF_{ij}) + 1) \times \left(1 - \left(\sum_j \left(\frac{p_{ij} \log p_{ij}}{\log D}\right)\right)\right) \quad (1)$$

where D is number of documents, TF_{ij} is frequency of term i in document j , f_i is frequency of term i in document collection, and $p_{ij} = \frac{TF_{ij}}{f_i}$

- Truncated SVD to reduce dimensionality
- Cosine measure to compute vector similarities

HAL

- Hyperspace Analogous to Language (Lund and Burgess, 1996a)
- Developed specifically for word representations
- Uses directional co-occurrence

HAL (contd.)

- Directional word by word matrix
- Distance weighting of the co-occurrences
- Concatenation of row-column vectors
- Dimensionality reduction optional
 - Discard low variant dimensions
- Normalization of vectors to unit length
- Similarities computed through either Manhattan or Euclidean distance

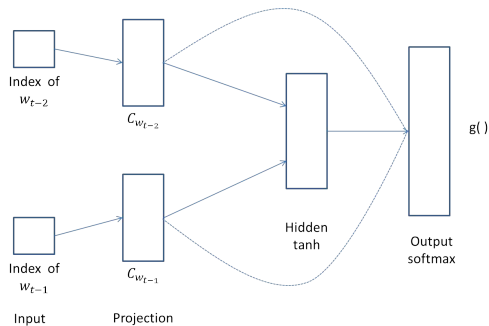
Word Embeddings

Prediction Based Embeddings

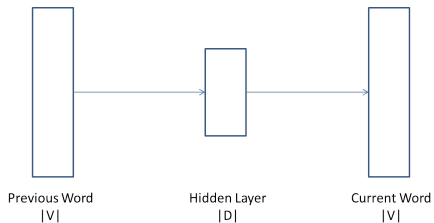
NNLM

- Neural Network Language Model
- Proposed by Bengio et al. (2003)
- Predict word given context
- Word Vectors learnt as a by-product of language modelling

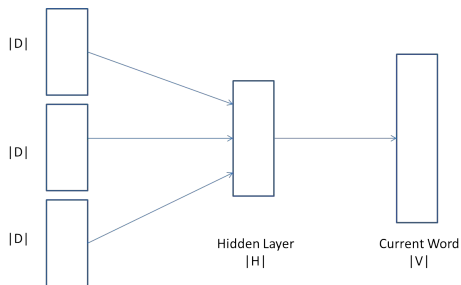
NNLM: Original Model



NNLM: Simplified (1)

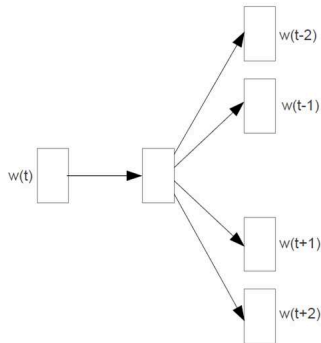


NNLM: Simplified (2)



Skip Gram

- Proposed by Mikolov et al. (2013b)
- Predict Context given word



Skip Gram (contd.)

- Given a sequence of training words w_1, w_2, \dots, w_T , maximize

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t) \quad (2)$$

- where

$$p(w_o | w_l) = \frac{\exp(u_{w_o}^T v_{w_l})}{\sum_{w=1}^W \exp(u_w^T v_{w_l})} \quad (3)$$

Global Vectors (GloVe)

- Proposed by Pennington et al. (2014)
- Predict Context given word
- Similar to Skip-gram, but objective function is different

$$J = \sum_{i,j=1}^V f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \quad (4)$$

- where X_{ij} can be likelihood of i_{th} and j^{th} word occurring together, and f is a weightage function

Tuning word embeddings

- Techniques which intend to tune already trained word embeddings to various tasks using additional information
- Ling et al. (2015) improve quality of word2vec for syntactic tasks such as POS
 - Take word positioning into account
 - Structured Skip-Gram and Continuous Windows: available as `wang2vec`
- Levy and Goldberg (2014) use dependency parse trees
 - Linear windows capture broad topical similarities, and dependency context captures functional similarities
- Patel et al. (2017) use medical code hierarchy to improve medical domain specific word embeddings

Word Embeddings

Multilingual Word Embeddings

Objective

- English data \gg Data for other languages
- Language independent phenomenon learnt on English should be applicable to other languages
- Solution via word embeddings:
 - Project words of different languages into a common subspace
- Goal of multilingual word embeddings: Shared subspace for all languages
- Neural MT learns such embeddings implicitly by optimizing the MT objective
- We shall discuss explicit models
 - These models are for MT what word2vec, GloVe are for NLP
 - Much lower cost of training as compared to Neural MT
- Applications: Machine Translation, Automated Bilingual Dictionary Generation, Cross-lingual Information Retrieval, *etc.*

Types of Cross-lingual Embeddings

- Based on the underlying approaches:
 - Monolingual mapping
 - Cross-lingual training
 - Joint optimization
- Based on the resource used:
 - Word-aligned data
 - Sentence-aligned data
 - Document-aligned data
 - Lexicon
 - No parallel data

Monolingual Mapping

- Learning in two step:
 - Train separate embeddings w_e and w_f on large monolingual corpora of corresponding languages e and f
 - Learn transformations g_1 and g_2 such that $w_e = g_1(w_f)$ and $w_f = g_2(w_e)$
- Transformations learnt using bilingual word mappings (lexicon)

Monolingual Mapping (contd.)

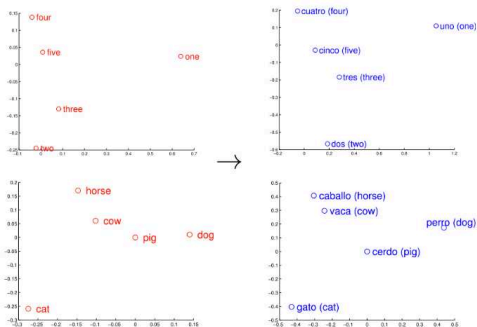
- Linear Projection proposed by Mikolov et al. (2013a)

- Learn matrix W s.t

$$w_e \approx W \cdot w_f$$

which minimizes

$$\sum_{i=1}^n \|W w_f - w_e\|^2$$



- We adapted this method for automatic synset linking in multilingual wordnets (accepted at GWC 2018)

Monolingual Mapping (Contd.)

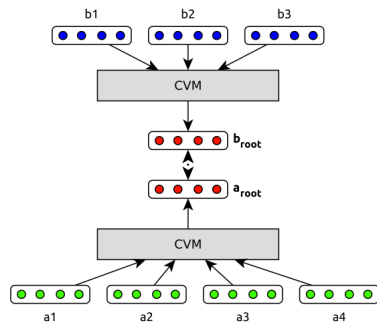
- Linear projection (Mikolov et al., 2013a): Lexicon
- Projection via CCA (Faruqui and Dyer, 2014b): Lexicon
- Alignment-based projection (Guo et al., 2015): Word-aligned data
- Adversarial auto-encoder (Barone, 2016): No parallel data

Cross Linguual Training

- Goal: optimizing cross-lingual objective
- Mainly rely on sentence alignments
- Require parallel corpus for training

Cross Linguual Training (contd.)

- Bilingual Compositional Sentence Model proposed by Hermann and Blunsom (2013)
- Train two models to produce sentence representations of aligned sentences in two languages
- Minimize distance between sentence representations of aligned sentences



Cross Lingual Training (contd.)

- Bilingual compositional sentence model (Hermann and Blunsom, 2013): Sentence-aligned data
- Distributed word alignment (Kočiský et al., 2014): Sentence-aligned data
- Translation-invariant LSA (Huang et al., 2015): Lexicon
- Inverted Indexing on Wikipedia (Søgaard et al., 2015): Document-aligned data

Joint Optimization

- Jointly optimize both monolingual M and cross-lingual Ω constraints
- Objective: minimize $M_{l_1} + M_{l_2} + \lambda \cdot \Omega_{l_1 \rightarrow l_2} + \Omega_{l_2 \rightarrow l_1}$
where λ decides weightage of cross-lingual constraints

Joint Optimization (contd.)

- Multitask Language Model proposed by Klementiev et al. (2012):
 - Train neural language model (NNLM)
 - Jointly optimize monolingual maximum likelihood (M) with word alignment based MT regularization term (Ω)

Joint Optimization (contd.)

- Multi-task language model (Klementiev et al., 2012):
Word-aligned data
- Bilingual skip-gram (Luong et al., 2015): Word-aligned data
- Bilingual bag-of-words without alignment (Gouws et al., 2015): Sentence-aligned data
- Bilingual sparse representations (Vyas and Carpuat, 2016):
Word-aligned data

Word Embeddings

Interpretable Word Embeddings

Interpretability and Explainability

- A model is interpretable if a human can make sense out of it
- Example: Decision trees
- Interpretable models enable one to explain the performance of the system and tune it accordingly
- However, in practice, interpretable models generally perform poor compared to other systems

Interpretable Word Embeddings

- Dimensions interpretable by ordering words based on value

Word	d205	Word	d272
iguana	0.599371	thigh	0.875286
bueller	0.584335	knee	0.872282
chimpanzee	0.577834	shoulder	0.866209
wasp	0.556845	elbow	0.857403
chimp	0.553980	wrist	0.852959
hamster	0.534810	ankle	0.851555
giraffe	0.532316	groin	0.841347
unicorn	0.529533	forearm	0.837988
caterpillar	0.528376	leg	0.836661
baboon	0.526324	pelvis	0.777564
gorilla	0.521590	neck	0.758420
tortoise	0.519941	spine	0.754774
sparrow	0.516842	torso	0.707458
lizard	0.515716	hamstring	0.701921
cockroach	0.505015	buttocks	0.689092
crocodile	0.491139	knees	0.676485
alligator	0.486275	ankles	0.658485
moth	0.471682	jaw	0.653126
kangaroo	0.469284	biceps	0.650972
toad	0.463514	hips	0.647000

Examples from NNSE embeddings Murphy et al. (2012)

NNSE

- Non Negative Sparse Embeddings proposed by Murphy et al. (2012)
- Word embeddings interpretable and cognitively plausible
- Performs a mixture of topical and taxonomical semantics
- Computation
 - Dependency co-occurrence adjusted with PPMI (to normalize for word frequency) and reduced with sparse SVD
 - Document co-occurrence adjusted with PPMI and reduced with sparse SVD
 - Their union factorized using a variant of non-negative sparse coding
- Resulting word embeddings have both topical neighbors (*judge* is near to *prison*) and taxonomical neighbors (*judge* is near to *referee*)
- Code unavailable, embeddings available at <http://www.cs.cmu.edu/~bmurphy/NNSE/>

OIWE

- Online Interpretable Word Embeddings proposed by Luo et al. (2015)
- Main idea: apply sparsity to skip gram
- Achieve sparsity by setting to 0 any dimensions of a vector that falls below 0
- Propose two techniques to do this via gradient descent
- They outperform NNSE at word intrusion task
- Code available on Github at <https://github.com/SkTim/OIWE>

Evaluating Word Embeddings

Intrinsic Evaluation

Word Pair Similarity

- Evaluates generalizability of word embeddings
- One of the most widely used evaluations
- Many datasets available: WS353, RG65, MEN, SimLex, SCWS, *etc.*

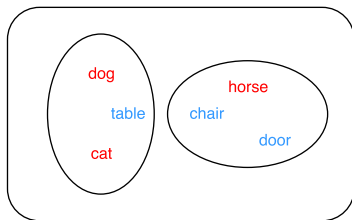
Word1	Word2	Human Score	Model1 Score	Model2 Score
street	street	10.00	1.0	1.0
street	avenue	8.88	0.04	0.38
street	block	6.88	0.14	0.26
street	place	6.44	0.21	0.18
street	children	4.94	-0.08	0.15
Spearman Correlation			0.6	1.0

Word Analogy task

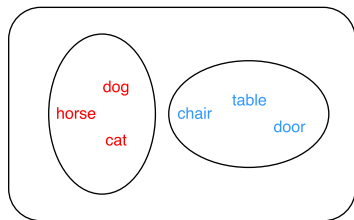
- Proposed by Mikolov et al. (2013b)
- Try to answer the question
man is to woman as king is to ?
- Often discussed in media

Categorization

- Evaluates the ability of embeddings to form proper clusters
- Given sets of words with different labels, try to cluster them, and check the correspondence between clusters and sets.
- The *purer* the cluster, the better is the embeddings
- Datasets available: Bless, Battig, *etc.*



E1



E2

Word Intrusion Detection

- Proposed by Murphy et al. (2012)
- Provides a way to interpret dimensions
- Most approaches do not report results on this task
 - Experiments done by us suggest many of them are not interpretable

Word Intrusion Detection (contd.)

- The approach:
 - 1 Select a dimension
 - 2 Reverse sort all vectors based on this dimension
 - 3 Select top 5 words
 - 4 Select a word, which is in bottom half of this list, and is in top 10 percentile in some other columns
 - 5 Give a random permutation of these 6 words to a human evaluator
 - Example: {bathroom, closet, attic, balcony, quickly, toilet}
 - 6 Check precision

Evaluating Word Embeddings

Extrinsic Evaluation

Extrinsic Evaluations

- Evaluating word embeddings on downstream NLP tasks such as Part of speech tagging, Named Entity Recognition, *etc.*
- Makes more sense as we ultimately want to use embeddings for such tasks
- However, performance does not solely rely on embeddings
 - Improvement/Degradation could be due to other factors such as network architecture, hyperparameters, *etc.*
- If an embedding E_1 is better than another embedding E_2 when used with some network architecture for NER, does that mean E_1 will be better for all architectures of NER?

Evaluations on Unified Architectures

- Unified architectures such as Collobert and Weston (2008b) used for extrinsic evaluations
- For different tasks, the architecture remains same, except the last layer, where the output neurons are changed according to the task at hand
- If an embedding E_1 is better than another embedding E_2 on all tasks on such a unified architecture, then we can expect it to be truly better

Evaluating Word Embeddings

Evaluation Frameworks

WordVectors.org

- Proposed by Faruqui and Dyer (2014a)
- A web interface for evaluating a collection of word pair similarity datasets on your embeddings available at <http://wordvectors.org/>
- Also provides visualization for common sets of words like (Male,Female) and (Antonym,Synonym) pairs

VecEval

- Proposed by Nayak et al. (2016)
- A web based tool for performing extrinsic evaluations
<http://www.veceval.com/>
- Claimed to support six different tasks: POS, NER, Chunking, Sentiment Analysis, Natural Language Inference, Question Answering
- Has never worked for me
- Web interface ~~no longer available~~ inactive, code available on Github at <https://github.com/NehaNayak/veceval>

Anago

- A Keras implementation of sequence labelling based on Lample et al. (2016)'s architecture
- Can perform POS, NER, SRL, *etc.*
- Used in our lab for extrinsic evaluation
- Code available on Github at <https://github.com/Hironsan/anago>

Evaluating Word Embeddings

Visualizing Word Embeddings

Visualizing Word Embeddings

- Various ways to visualize word embeddings: PCA, Isomap, tSNE, *etc.*, available in scikit-learn

```
from sklearn import decomposition, manifold
vis = decomposition.TruncatedSVD(n_components=2) - PCA
E_vis = vis.fit_transform( E)
plot E_vis here
```

- Check out http://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html for many methods applied to MNIST visualization

Related Work

- Baroni et al. (2014): Neural word embeddings are better than traditional methods such as LSA, HAL, RI (Landauer and Dumais, 1997; Lund and Burgess, 1996b; Sahlgren, 2005)
- Levy et al. (2015): Superiority of neural word embeddings not due to the embedding algorithm, but due to **certain design choices and hyperparameters optimizations**
 - Varies other hyperparameters; keeps number of dimensions = 500
- Schnabel et al. (2015); Zhai et al. (2016); Ghannay et al. (2016): No justification for chosen number of dimensions in their evaluations
- Melamud et al. (2016): Optimal number of dimensions different for different evaluations of word embeddings

Why Dimensions matter?: A Practical Example

- Various app developers want to utilize word embeddings
- Example memory limit for app: 200 MB
- Size of Google Pre-trained vectors file: 3.4 GB
- Natural thought process: decrease dimensions
 - To what value? 100? 50? 20?
- Depends on the words/entities we want to place in the space

Number of Dimensions and Equidistant points

Number of Dimensions and Equidistant points

- Number of dimensions of a vector space imposes a restriction on the number of equidistant points it can have
- Given that distance is euclidean, if the number of dimensions $\lambda = N$, then maximum number of equidistant points E in the corresponding space is $N + 1$ (Swanepoel, 2004)
- Given that distance is cosine, no closed form solution exists

Dimensions λ and max. no. of equiangular lines E (Barg and Yu, 2014)

λ	E	λ	E
3	6	18	61
4	6	19	76
5	10	20	96
6	16	21	126
$7 \leq n \leq 13$	28	22	176
14	30	23	276
15	36	$24 \leq n \leq 41$	276
16	42	42	288
17	51	43	344

Objective

Problem Statement

Does the number of pairwise equidistant words enforce a lower bound on the number of dimensions for word embeddings?

- 'Equidistance' determined using co-occurrence matrix
- Plan of Action:
 - Verify using a toy corpus
 - Evaluate on actual corpus

Motivation (1/4)

- Consider the following toy corpus:

<>I like cats <>I love dogs <>I hate rats <>I rate bats <>

- Corresponding co-occurrence matrix:

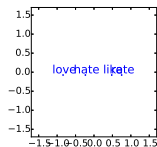
word	<>	I	like	love	hate	rate	rats	cats	dogs	bats
like	0	1	0	0	0	0	0	1	0	0
love	0	1	0	0	0	0	0	0	1	0
hate	0	1	0	0	0	0	1	0	0	0
rate	0	1	0	0	0	0	0	0	0	1

- Distance between any pair of words = $\sqrt{2}$
- The words form a regular tetrahedron

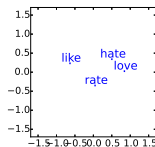
Motivation (2/4)

Mean and Std Dev of Mean of a point's distance with other points

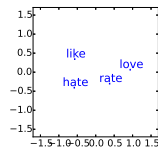
Dimension	Mean	Stddev
1	0.94	0.94
2	1.77	0.80
3	2.63	0.10



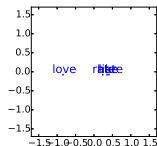
1d(Before)



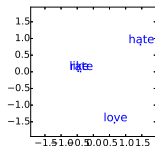
2d(Before)



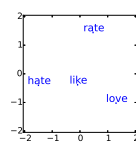
3d(Before)



1d(After)



2d(After)



3d(After)

Motivation (3/4)

Hypothesis:

- If the learning algorithm of word embeddings does not get enough dimensions, then it will fail to uphold the equality constraint
 - Standard deviation of the mean of all pairwise distances will be higher
- As we increase the dimension, the algorithm will get more degrees of freedom to model the equality constraint in a better way
 - There will be statistically significant changes in the standard deviation
- Once the lower bound of dimensions is reached, the algorithm gets enough degrees of freedom.
 - From this point onwards, even if we increase dimensions, there will not be any statistically significant difference in the standard deviation

Motivation (4/4)

Dim	$\bar{\sigma}$	P-value	Dim	$\bar{\sigma}$	P-value
7	0.358		12	0.154	0.0058
8	0.293	0.0020	13	0.111	0.0001
9	0.273	0.0248	14	0.044	0.0001
10	0.238	0.0313	15	0.047	0.3096
11	0.189	0.0013	16	0.054	0.1659

Avg standard deviation ($\bar{\sigma}$) for 15 pairwise equidistant words (along with two tail p-values of Welch's unpaired t-test for statistical significance)

Approach (1/5)

1. Compute the word \times word co-occurrence matrix from the corpus

$\langle \rangle$ I like cats $\langle \rangle$ I love dogs $\langle \rangle$ I hate rats $\langle \rangle$ I rate bats $\langle \rangle$

word	$\langle \rangle$	I	like	love	hate	rate	rats	cats	dogs	bats
$\langle \rangle$	0	4	0	0	0	0	1	1	1	1
I	4	0	1	1	1	1	0	0	0	0
like	0	1	0	0	0	0	0	1	0	0
love	0	1	0	0	0	0	0	0	1	0
hate	0	1	0	0	0	0	1	0	0	0
rate	0	1	0	0	0	0	0	0	0	1
rats	1	0	0	0	1	0	0	0	0	0
cats	1	0	1	0	0	0	0	0	0	0
dogs	1	0	0	1	0	0	0	0	0	0
bats	1	0	0	0	0	1	0	0	0	0

Approach (2/5)

2. Create the corresponding word \times word cosine similarity matrix

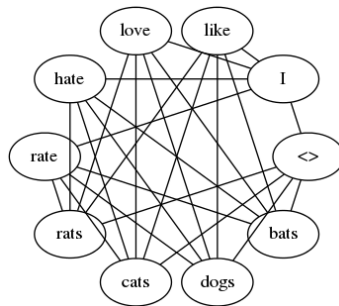
$\langle \rangle$ I like cats $\langle \rangle$ I love dogs $\langle \rangle$ I hate rats $\langle \rangle$ I rate bats $\langle \rangle$

	$\langle \rangle$	I	like	love	hate	rate	rats	cats	dogs	bats
$\langle \rangle$	1.0	0.0	0.8	0.8	0.8	0.8	0.0	0.0	0.0	0.0
I	0.0	1.0	0.0	0.0	0.0	0.0	0.8	0.8	0.8	0.8
like	0.8	0.0	1.0	0.5	0.5	0.5	0.0	0.0	0.0	0.0
love	0.8	0.0	0.5	1.0	0.5	0.5	0.0	0.0	0.0	0.0
hate	0.8	0.0	0.5	0.5	1.0	0.5	0.0	0.0	0.0	0.0
rate	0.8	0.0	0.5	0.5	0.5	1.0	0.0	0.0	0.0	0.0
rats	0.0	0.8	0.0	0.0	0.0	0.0	1.0	0.5	0.5	0.5
cats	0.0	0.8	0.0	0.0	0.0	0.0	0.5	1.0	0.5	0.5
dogs	0.0	0.8	0.0	0.0	0.0	0.0	0.5	0.5	1.0	0.5
bats	0.0	0.8	0.0	0.0	0.0	0.0	0.5	0.5	0.5	1.0

Approach (3/5)

3. For each similarity value s_k , create a graph, where the words are nodes, and an edge between node i and node j if $\text{sim}(i, j) = s_k$

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>

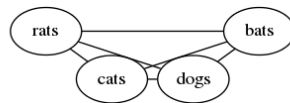
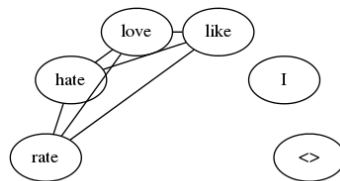


Sim=0.0

Approach (3/5)

3. For each similarity value s_k , create a graph, where the words are nodes, and an edge between node i and node j if $\text{sim}(i, j) = s_k$

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>

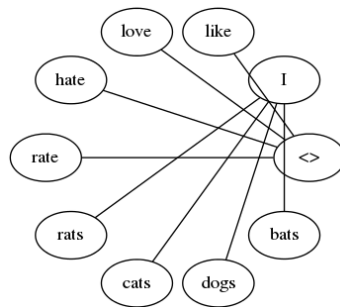


Sim=0.5

Approach (3/5)

3. For each similarity value s_k , create a graph, where the words are nodes, and an edge between node i and node j if $\text{sim}(i, j) = s_k$

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>

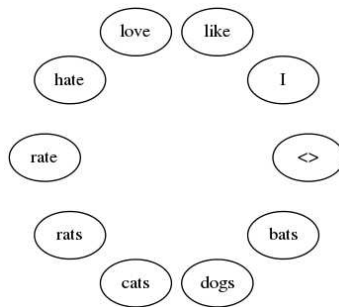


Sim=0.8

Approach (3/5)

3. For each similarity value s_k , create a graph, where the words are nodes, and an edge between node i and node j if $\text{sim}(i, j) = s_k$

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>



Sim=1.0

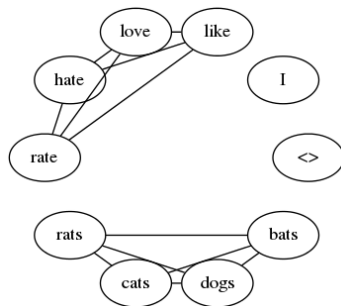
Approach (4/5)

4. Find maximum clique on this graph. The number of nodes in this clique is the maximum number of pairwise equidistant points

$$E_k$$

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>

Sim	E_k
0.5	4



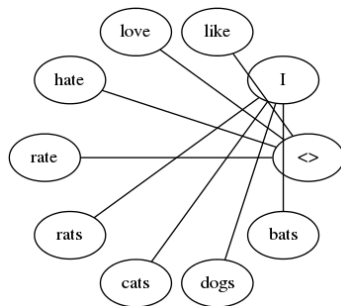
Approach (4/5)

4. Find maximum clique on this graph. The number of nodes in this clique is the maximum number of pairwise equidistant points

$$E_k$$

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>

Sim	E_k
0.5	4
0.8	0



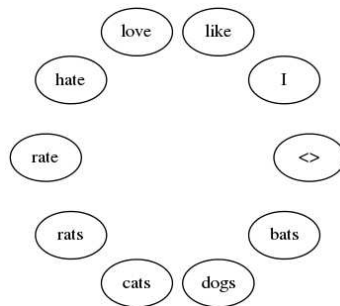
Approach (4/5)

4. Find maximum clique on this graph. The number of nodes in this clique is the maximum number of pairwise equidistant points

$$E_k$$

<> I like cats <> I love dogs <> I hate rats <> I rate bats <>

Sim	E_k
0.5	4
0.8	0
1.0	0



Approach (5/5)

5. Reverse lookup E_k to get the number of dimension λ

$\langle \rangle$ I like cats $\langle \rangle$ I love dogs $\langle \rangle$ I hate rats $\langle \rangle$ I rate bats $\langle \rangle$

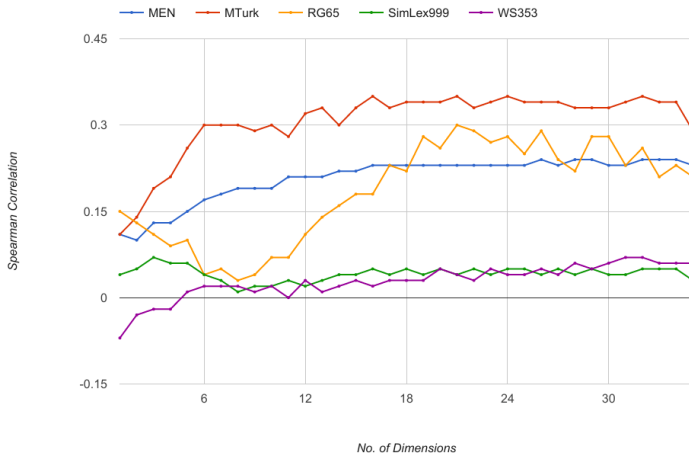
Sim	E_k
0.5	4
0.8	0
1.0	0
Max	4

λ	E	λ	E
3	6	18	61
4	6	19	76
5	10	20	96
6	16	21	126
$7 \leq n \leq 13$	28	22	176
14	30	23	276
15	36	$24 \leq n \leq 41$	276
16	42	42	288
17	51	43	344

Evaluation

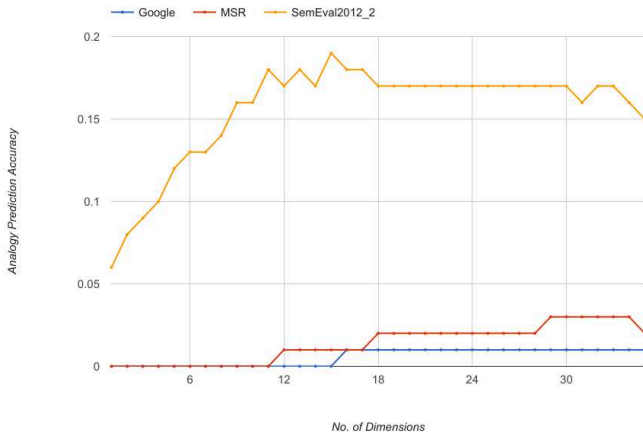
- Used Brown Corpus
- Found **19** as lower bound using our approach
- Context window: 1 to the left and 1 to the right
- Number of dimensions: 1 to 35
- 5 randomly initialized models for each configuration (average results reported)
- Intrinsic Evaluation
 - Word Pair Similarity: Predicting $sim(w_a, w_b)$ using corresponding word embeddings
 - Word Analogy: Finding missing w_d in the relation: a is to b as c is to d
 - Categorization: Checking the purity of clusters formed by word embeddings

Results



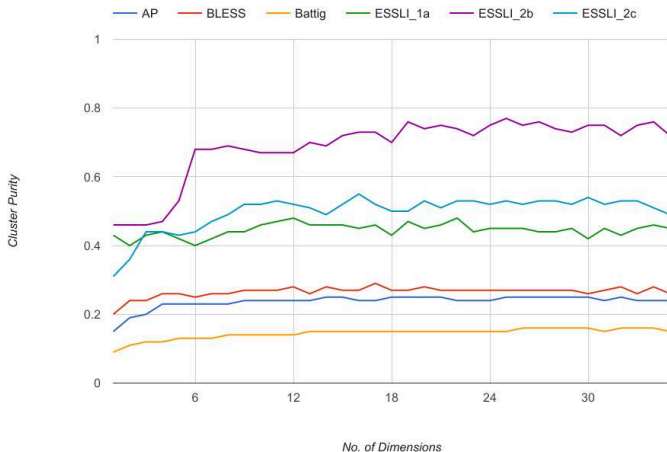
Performance for Word Pair Similarity task with respect to number of dimensions

Results



Performance for Word Analogy task with respect to number of dimensions

Results



Performance for Categorization task with respect to number of dimensions

Analysis

- Found lower bound consistent with experimental evaluation

Poltawa	snakestrike	burnings	Tsar's
miswritten	brows	maintained	South-East
far-famed	27%	non-dramas	octagonal
boatyards	U-2	Devol	mourners
Hearing	sideshow	third-story	upcoming
pram	dolphins	Croydon	neuromuscular
Gladius	pvt	littered	annoying
vuhranduh	athletes	eraser	provincialism
Daly	wreaths	villain	suspicious
nooks	fielder	belly	Gogol's
interchange	two-to-three	resemble	discounted
kidneys	Hangman's	commend	accordion
summarizing	optimality	Orlando	Leamington
swift	Taras-Tchaikovsky	puts	groomed
spit	firmer	rosy-fingered	Bechhofer
campfire	Tomas		

Set of pairwise equiangular points (vectors) from Brown corpus

Limitations

- The Max Clique finding component of the approach
 - Renders approach intractable for larger corpora
 - Need to find an alternative

Applications of Word Embeddings

Are Word Embeddings Useful for Sarcasm Detection?

Problem Statement

- Detect whether a sentence is sarcastic or not?
 - Especially among those sentences which do not contain sentiment bearing words
- Example: A woman needs a man just like a fish needs a bicycle

Motivation

- Similarity measure among word embeddings a proxy for measuring contextual incongruity
- Example: A woman needs a man just like a fish needs a bicycle

$$\textit{similarity}(\textit{man}, \textit{woman}) = 0.766$$

$$\textit{similarity}(\textit{fish}, \textit{bicycle}) = 0.131$$

- Imbalance in similarities above an indication of contextual incongruity

Approach

- Gist of the approach is adding similarity of word embeddings based features, such as
 - Maximum similarity between all pairs of words in a sentence
 - Minimum similarity between all pairs of words in a sentence

Evaluation

- 1 **Liebrecht et al. (2013)**: They consider unigrams, bigrams and trigrams as features.
- 2 **González-Ibáñez et al. (2011)**: Two sets of features: unigrams and dictionary-based.
- 3 **Buschmeier et al. (2014)**:
 - Hyperbole (captured by 3 positive or negative words in a row)
 - Quotation marks and ellipsis
 - Positive/Negative Sentiment words followed by an exclamation or question mark
 - Positive/Negative Sentiment Scores followed by ellipsis ('...')
 - Punctuation, Interjections, and Laughter expressions.
- 4 **Joshi et al. (2015)**: In addition to unigrams, they use features based on implicit and explicit incongruity
 - Implicit incongruity features - patterns with implicit sentiment , extracted in a pre-processing step.
 - Explicit incongruity features - number of sentiment flips, length of positive and negative sub-sequences and lexical polarity.

Results

Word Embedding	Average F-score Gain
LSA	0.453
Glove	0.651
Dependency	1.048
Word2Vec	1.143

Average gain in F-scores for the four types of word embeddings; These values are computed for a subset of these embeddings consisting of words common to all four

Applications of Word Embeddings

Iterative Unsupervised Most Frequent Sense Detection using
Word Embeddings

WordNet

- Groups synonymous words into *synsets*
- Synset example:
 - Synset ID: 02139199
 - Synset Members: { bat, chiropteran }
 - Gloss: nocturnal mouselike mammal with forelimbs modified to form membranous wings and anatomical adaptations for echolocation by which they navigate
 - Example: Bats are creatures of the night.
- Relations with other synsets (hypernym/hyponym: parent/child, meronym/holonym: part/whole)

Introduction

- Word Sense Disambiguation (WSD) : one of the relatively hard problems in NLP
 - Both supervised and unsupervised ML explored in literature
- Most Frequent Sense (MFS) baseline: strong baseline for WSD
 - Given a WSD problem instance, simply assign the most frequent sense of that word
- Ignores context
- Really strong results
 - Due to skew in sense distribution of data
- Computing MFS:
 - Trivial for sense-annotated corpora, which is not available in large amounts.
 - Need to learn from raw data

Problem Statement

Problem Statement

Given a raw corpus, estimate most frequent sense of different words in that corpus

- Bhingardive et al. (2015) showed that pretrained word embeddings can be used to compute most frequent sense
- Our work further strengthens the claim by Bhingardive et al. (2015) that word embeddings indeed capture most frequent sense
- Our approach outperforms others at the task of MFS extraction
- To compute MFS using our approach:
 - 1 Train word embeddings on the raw corpus.
 - 2 Apply our approach on the trained word embeddings.

Intuition

- Strive for consistency in assignment of senses to maintain semantic congruity
- Example:
 - If *cricket* and *bat* co-occur a lot, then *cricket* taking *insect* sense and *bat* taking reptile sense is less likely

Intuition

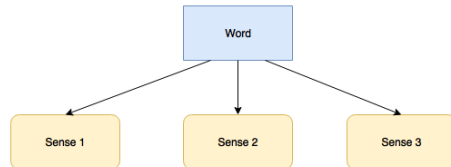
- Strive for consistency in assignment of senses to maintain semantic congruity
- Example:
 - If *cricket* and *bat* co-occur a lot, then *cricket* taking *insect* sense and *bat* taking reptile sense is less likely
 - If *cricket* and *bat* co-occur a lot, and *cricket*'s MFS is *sports*, then *bat* taking reptile sense is extremely unlikely
- Key point: solve easy words, then use them for difficult words
In other words, iterate over degree of polysemy from 2 onward

Algorithm

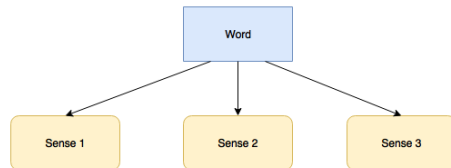


Word

Algorithm



Algorithm

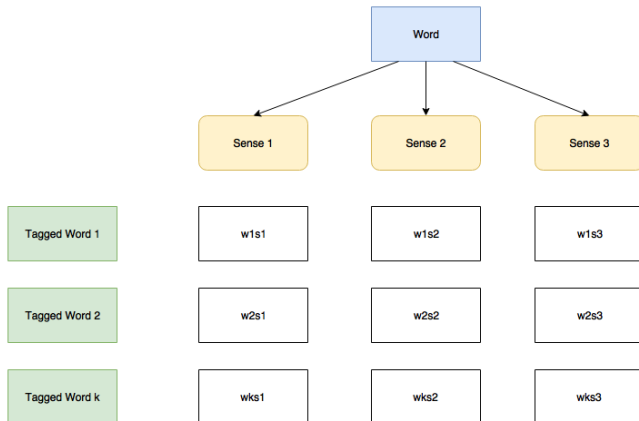


Tagged Word 1

Tagged Word 2

Tagged Word k

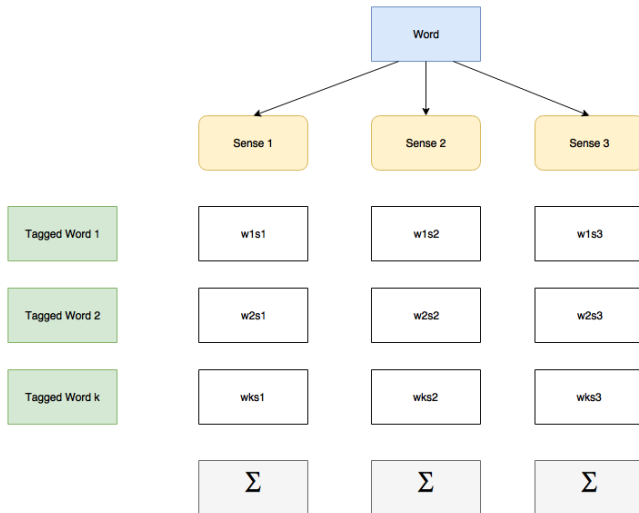
Algorithm



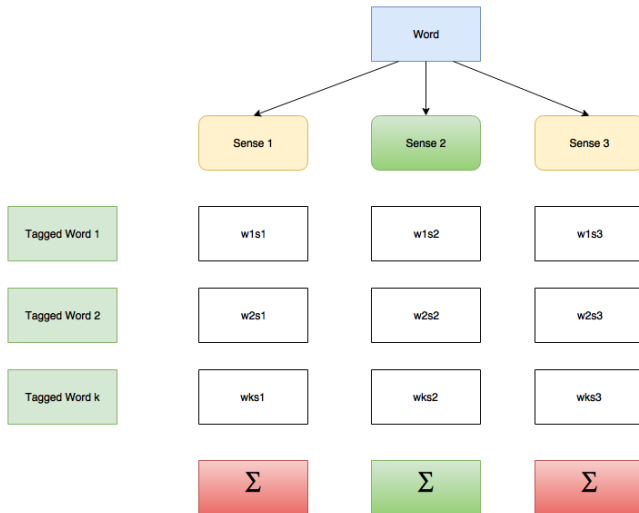
Algorithm

- $w_i s_j$ is vote for s_j due to w_i
- Two components
 - Wordnet similarity between $\text{mfs}(w_i)$ and s_j
 - Embedding space similarity between w_i and current word

Algorithm



Algorithm



Parameters

- K
- Similarity Measure
- Unweighted (no vector space component) vs. Weighted

Evaluation

- Two setups:
 - Evaluating MFS as solution for WSD
 - Evaluating MFS as a classification task

MFS as solution for WSD

Method	Senseval2	Senseval3
Bhingardive(reported)	52.34	43.28
SemCor(reported)	59.88	65.72
Bhingardive	48.27	36.67
Iterative	63.2	56.72
SemCor	67.61	71.06

Accuracy of WSD using MFS (Nouns)

MFS as solution for WSD (contd.)

Method	Senseval2	Senseval3
Bhingardive(reported)	37.79	26.79
Bhingardive(optimal)	43.51	33.78
Iterative	48.1	40.4
SemCor	60.03	60.98

Accuracy of WSD using MFS (All Parts of Speech)

MFS as classification task

Method	Nouns	Adjectives	Adverbs	Verbs	Total
Bhingardive	43.93	81.79	46.55	37.84	58.75
Iterative	48.27	80.77	46.55	44.32	61.07

Percentage match between predicted MFS and WFS

MFS as classification task (contd.)

	Nouns (49.20)	Verbs (26.44)	Adjectives (19.22)	Adverbs (5.14)	Total
Bhingardive	29.18	25.57	26.00	33.50	27.83
Iterative	35.46	31.90	30.43	47.78	34.19

Percentage match between predicted MFS and true SemCor MFS. Note that numbers in column headers indicate what percent of total words belong to that part of speech

Analysis

- Better than Bhingardive et al. (2015); not able to beat SemCor and WFS.
 - There are words for which WFS doesn't give *proper* dominant sense. Consider the following examples:
 - *tiger* - an audacious person
 - *life* - characteristic state or mode of living (social life, city life, real life)
 - *option* - right to buy or sell property at an agreed price
 - *flavor* - general atmosphere of place or situation
 - *season* - period of year marked by special events
 - Tagged words ranking very low to make a significant impact. For example:
 - While detecting MFS for a bisemous word, the first monosemous neighbour actually ranks 1101
 - *i.e.* a 1000 polysemous words are closer than this monosemous word.
 - Monosemous word may not be the one who can influence the MFS.

Summary

- Proposed an iterative approach for unsupervised most frequent sense detection using word embeddings
- Similar trends, yet better overall results from Bhingardive et al. (2015)
- Future Work
 - Apply approach to other languages

Conclusion

- Discussed why we need word embeddings
- Briefly looked at classical word embeddings
- Discussed a few cross-lingual word embeddings and interpretable word embeddings
- Mentioned evaluation mechanisms and tools
- Argued on existence of lower bounds for number of dimensions of word embeddings
- Discussed some in-house applications

Thank You

References

- Barg, A. and Yu, W.-H. (2014). New bounds for equiangular lines. *Contemporary Mathematics*, 625:111–121.
- Barone, A. V. M. (2016). Towards cross-lingual distributed representations without parallel text trained with adversarial autoencoders. *arXiv preprint arXiv:1608.02996*.
- Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247. Association for Computational Linguistics.
- Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

References

- Bhingardive, S., Singh, D., V, R., Redkar, H., and Bhattacharyya, P. (2015). Unsupervised most frequent sense detection using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1238–1243, Denver, Colorado. Association for Computational Linguistics.
- Buschmeier, K., Cimiano, P., and Klinger, R. (2014). An impact analysis of features in a classification approach to irony detection in product reviews. In *Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 42–49.

References

- Collobert, R. and Weston, J. (2008a). A unified architecture for natural language processing: deep neural networks with multitask learning. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 160–167. ACM.
- Collobert, R. and Weston, J. (2008b). A unified architecture for natural language processing: deep neural networks with multitask learning. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 160–167. ACM.
- Dumais, S. T., Furnas, G. W., Landauer, T. K., Deerwester, S., and Harshman, R. (1988). Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–285. ACM.

References

- Faruqui, M. and Dyer, C. (2014a). Community evaluation and exchange of word vectors at wordvectors.org. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Baltimore, USA. Association for Computational Linguistics.
- Faruqui, M. and Dyer, C. (2014b). Improving vector space word representations using multilingual correlation. Association for Computational Linguistics.
- Ghannay, S., Favre, B., Estève, Y., and Camelin, N. (2016). Word embedding evaluation and combination. In Chair), N. C. C., Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).

References

- González-Ibáñez, R., Muresan, S., and Wacholder, N. (2011). Identifying sarcasm in twitter: a closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 581–586. Association for Computational Linguistics.
- Gouws, S., Bengio, Y., and Corrado, G. (2015). Bilbowa: Fast bilingual distributed representations without word alignments. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 748–756.
- Guo, J., Che, W., Yarowsky, D., Wang, H., and Liu, T. (2015). Cross-lingual dependency parsing based on distributed representations. In *ACL (1)*, pages 1234–1244.
- Harris, Z. S. (1970). *Distributional structure*. Springer.

References

- Hermann, K. M. and Blunsom, P. (2013). Multilingual distributed representations without word alignment. *arXiv preprint arXiv:1312.6173*.
- Huang, K., Gardner, M., Papalexakis, E., Faloutsos, C., Sidiropoulos, N., Mitchell, T., Talukdar, P. P., and Fu, X. (2015). Translation invariant word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1084–1088.
- Joshi, A., Sharma, V., and Bhattacharyya, P. (2015). Harnessing context incongruity for sarcasm detection. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 2, pages 757–762.

References

- Klementiev, A., Titov, I., and Bhattacharai, B. (2012). Inducing crosslingual distributed representations of words. *Proceedings of COLING 2012*, pages 1459–1474.
- Kočiský, T., Hermann, K. M., and Blunsom, P. (2014). Learning bilingual word representations by marginalizing alignments. *arXiv preprint arXiv:1405.0947*.
- Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.
- Landauer, T. K. and Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *PSYCHOLOGICAL REVIEW*, 104(2):211–240.

References

- Levy, O. and Goldberg, Y. (2014). Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 302–308.
- Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.
- Liebrecht, C., Kunneman, F., and van den Bosch, A. (2013). The perfect solution for detecting sarcasm in tweets# not. *Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA*.

References

- Ling, W., Dyer, C., Black, A. W., and Trancoso, I. (2015). Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304.
- Lund, K. and Burgess, C. (1996a). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208.
- Lund, K. and Burgess, C. (1996b). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208.

References

- Luo, H., Liu, Z., Luan, H., and Sun, M. (2015). Online learning of interpretable word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1687–1692.
- Luong, T., Pham, H., and Manning, C. D. (2015). Bilingual word representations with monolingual quality in mind. In *VS@ HLT-NAACL*, pages 151–159.
- Melamud, O., McClosky, D., Patwardhan, S., and Bansal, M. (2016). The role of context types and dimensionality in learning word embeddings. In Knight, K., Nenkova, A., and Rambow, O., editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1030–1040. The Association for Computational Linguistics.

References

- Mikolov, T., Le, Q. V., and Sutskever, I. (2013a). Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.
- Murphy, B., Talukdar, P., and Mitchell, T. (2012). *Learning Effective and Interpretable Semantic Models using Non-Negative Sparse Embedding*, pages 1933–1949. Association for Computational Linguistics.

References

- Nayak, N., Angeli, G., and Manning, C. D. (2016). Evaluating word embeddings using a representative suite of practical tasks. *ACL 2016*, page 19.
- Patel, K., Patel, D., Golakiya, M., Bhattacharyya, P., and Birari, N. (2017). Adapting pre-trained word embeddings for use in medical coding. In *BioNLP 2017*, pages 302–306, Vancouver, Canada,. Association for Computational Linguistics.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014)*, 12.
- Rubenstein, H. and Goodenough, J. B. (1965). Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633.

References

- Sahlgren, M. (2005). An introduction to random indexing. In *In Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*.
- Sahlgren, M. (2006). *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, Institutionen för lingvistik.
- Schnabel, T., Labutov, I., Mimno, D., and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 298–307.
- Schütze, H. (1992). Dimensions of meaning. In *Supercomputing'92., Proceedings*, pages 787–796. IEEE.

References

- Søgaard, A., Agić, Ž., Alonso, H. M., Plank, B., Bohnet, B., and Johannsen, A. (2015). Inverted indexing for cross-lingual nlp. In *The 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015)*.
- Swanepoel, K. J. (2004). Equilateral sets in finite-dimensional normed spaces. In *Seminar of Mathematical Analysis*, volume 71, pages 195–237. Secretariado de Publicaciones, Universidad de Sevilla, Seville.
- Vyas, Y. and Carpuat, M. (2016). Sparse bilingual word representations for cross-lingual lexical entailment. In *HLT-NAACL*, pages 1187–1197.

References

Zhai, M., Tan, J., and Choi, J. D. (2016). Intrinsic and extrinsic evaluations of word embeddings. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16*, pages 4282–4283. AAAI Press.

Web References

- <http://www.indiana.edu/~gasser/Q530/Notes/representation.html>

Sentiment Analysis

Md Shad Akhtar

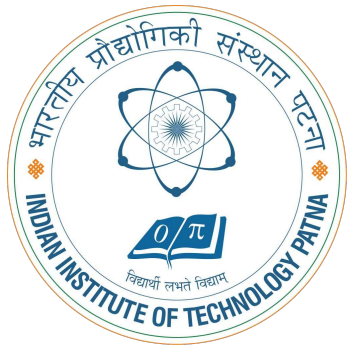
Research Scholar

AI-NLP-ML Group

Department of Computer Science & Engineering
Indian Institute of Technology Patna

shad.pcs15@iitp.ac.in

<https://iitp.ac.in/~shad.pcs15/>



Sentiment Analysis

- Sentiment analysis aims to identify the orientation of opinion in a piece of text.



Why do we need Sentiment Analysis?

- *What others think* has always been an important piece of information.
- *Overwhelming amounts of information* on one topic: Manually reading or analysing all data is very inefficient.
- *Baised/Fake* reviews
- An example
 - Mr. X needs to buy a phone. He was browsing amazon.in and found 1000 reviews for a particular phone.

Scenario 1:

- Let there are **850 negative**, **100 positive** and **50 neutral** reviews
- Sentiment → **Negative**.

What if all the 100 positive reviews are at the top?

Scenario 2:

- Let there are **420 negative**, **480 positive** and **100 neutral** reviews.
- Sentiment → **Positive**

What if few of the reviews (e.g. 100) are fake?

Challenges

- Similar lexical features but different sentiments
 - यह मूवी अच्छी नहीं है। (This movie is not good.)
 - कोई भी मूवी इस से अच्छी नहीं हो सकती। (No movie can be better than this.)
- Different style of writing but same sentiment
 - सैमसंग का फ़ोन बहुत ही बेकार है। (Samsung phone is extremely useless.)
 - सैमसंग फ़ोन पर मेरे पैसे बरबाद हो गए। (My money was wasted on Samsung phone.)
 - सैमसंग से अच्छा मैं आईफ़ोन खरीद लेता। (I could have bought Iphone instead of Samsung.)

Levels of sentiment analysis

- **Document level**
 - Overall sentiment of a document
 - A document consists of many sentences/paragraphs
- **Sentence level**
 - Sentiment of a stand alone sentence
- **Phrase level**
 - Sentiment towards a given phrase in a sentence
- **Aspect level**
 - Sentiment towards an attribute/feature/aspect of a sentence.
 - Aspect is an attribute or component of the product that has been commented on in a review
 - **Sentiment targets** helps us to understand the sentiment analysis problem better.

Increasing level of
information



Sentence level v/s Aspect level

- **Sentence level**
 - इसकी बैटरी शानदार है, लेकिन कैमरा बहुत ही खराब है। (Its battery is awesome but camera is very poor.)
 - Sentiment: Both *positive* and *negative* → *conflict*
- **Aspect level**
 - इसकी बैटरी शानदार है, लेकिन कैमरा बहुत ही खराब है। (Its battery is awesome but camera is very poor.)
 - Aspect terms and their sentiments:
 - बैटरी (battery) → positive
 - कैमरा (camera) → negative

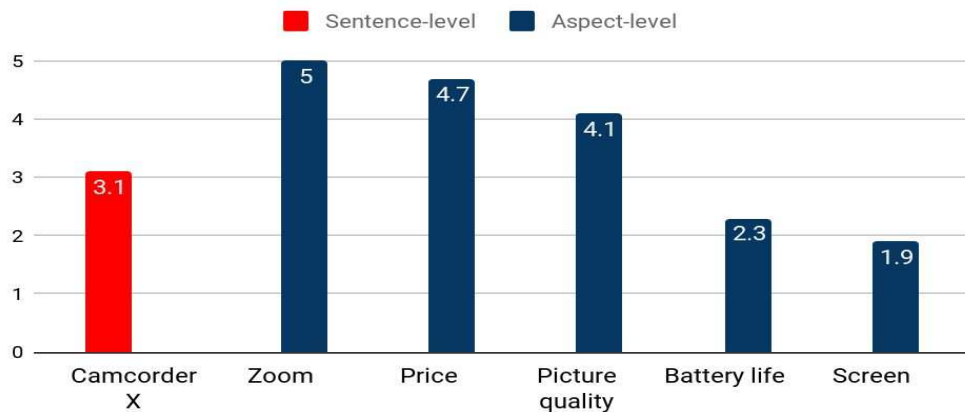
Sentence level v/s Aspect level

- Informed decision

- **Camcorder X**

- The **zoom** is excellent, but the **LCD** is blurry.
- Great value for the **price**.
- Although the **display** is poor the **picture quality** is amazing.
- **Batteries** drain pretty quickly.
- I love this camera but for short **battery life** is definitely a pain.

Rating



Sentiment Analysis: A boarder view

Sentiment Analysis

Levels

- Coarse-grained
- Fine-grained (Aspect)

Languages

- English
- Hindi
- French

Domains

- E-commerce
- Social Media
- Financial

Techniques

- Feature Engg.
- Deep learning

Application

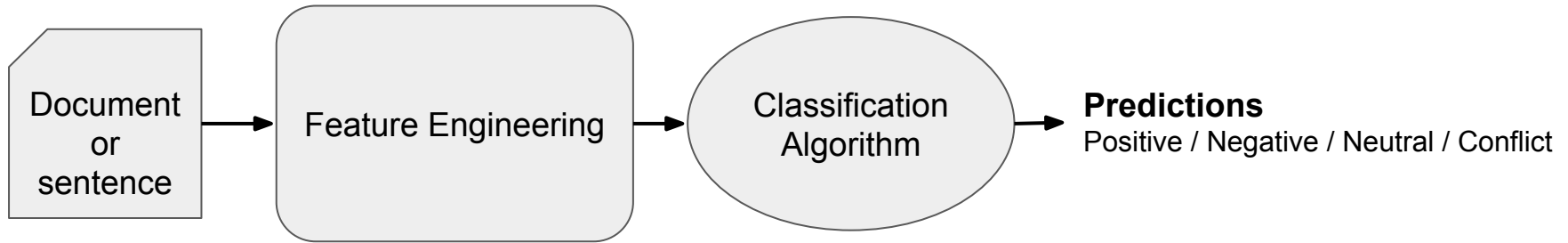
- E-comm
- Politics
- Security

Related problems

- Subjective
- Sarcasm
- Rumour

Pre-deep learning approaches for Sentiment Analysis (Traditional ML approaches)

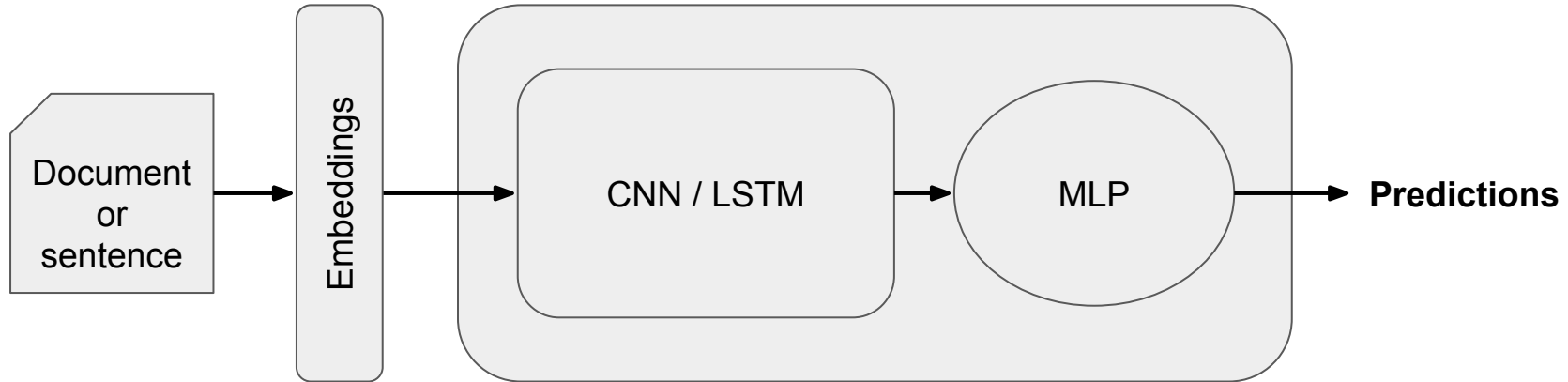
A traditional classification pipeline



- Ngrams
- Presence or Absence of cue words
- Lexicons
- SVM
- Decision Tree

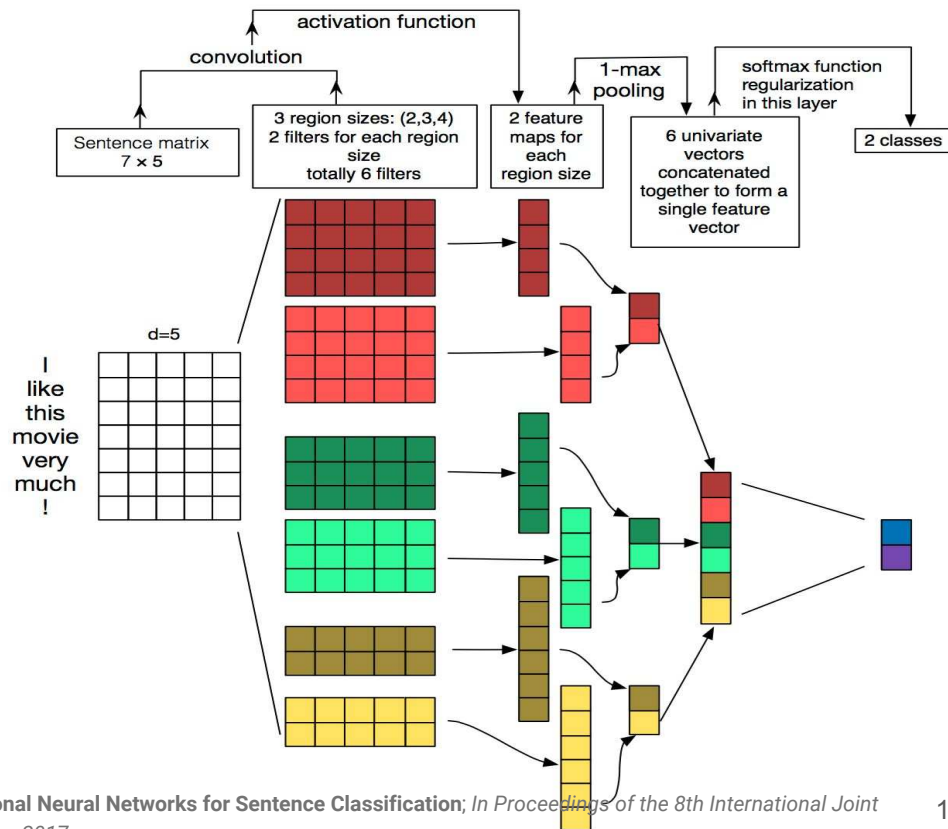
Deep Architectures for Sentiment Analysis

A typical deep learning solution

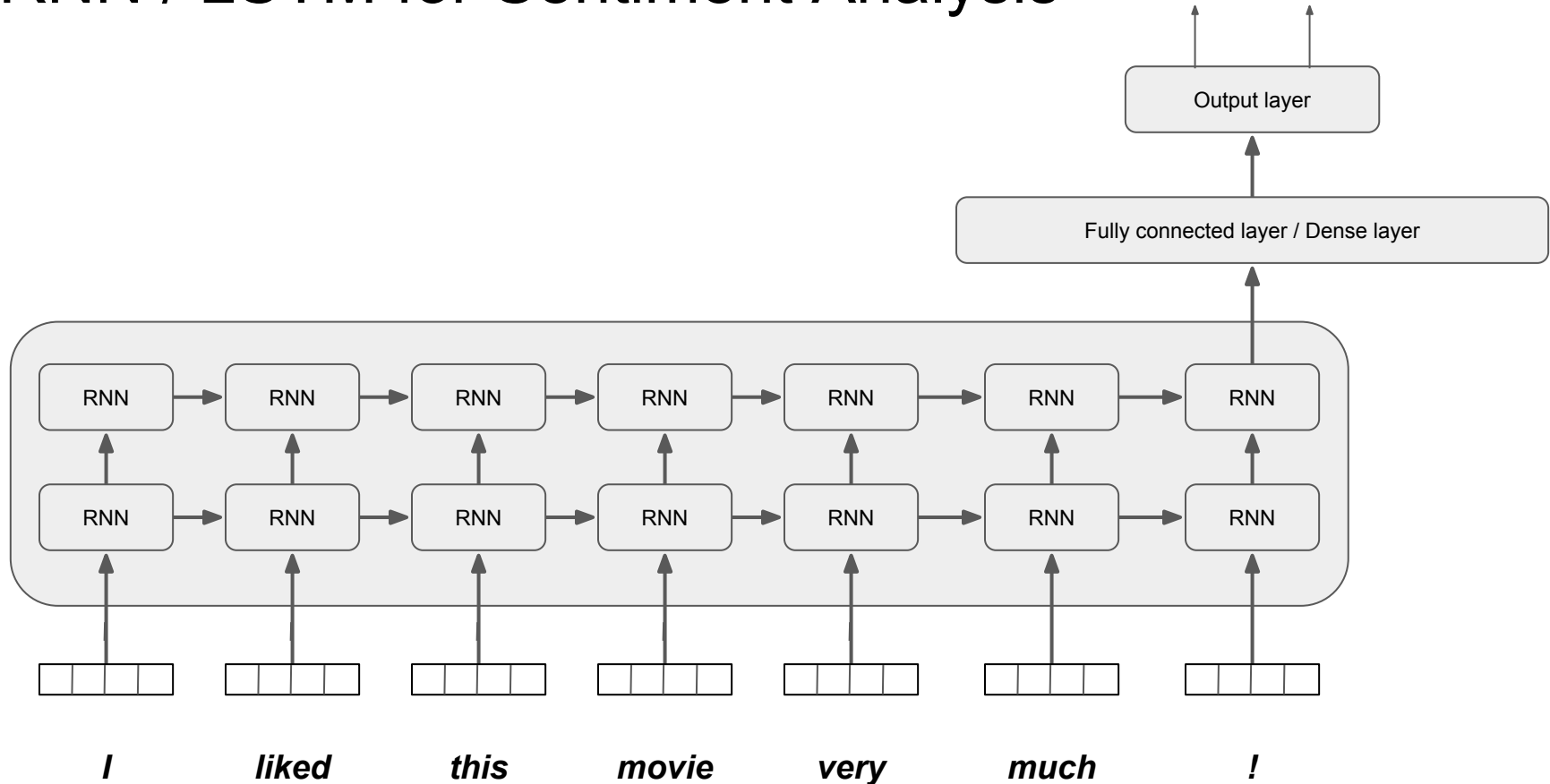


A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

1. Sentence matrix
 - a. embeddings of words
2. Convolution filters
 - a. Total 6 filters; Two each of size 2, 3 & 4.
 - b. 6 feature maps for each filter
3. Pooling
 - a. 1-max pooling
4. Concatenate the max-pooled vector
5. Classification
 - a. Softmax



RNN / LSTM for Sentiment Analysis



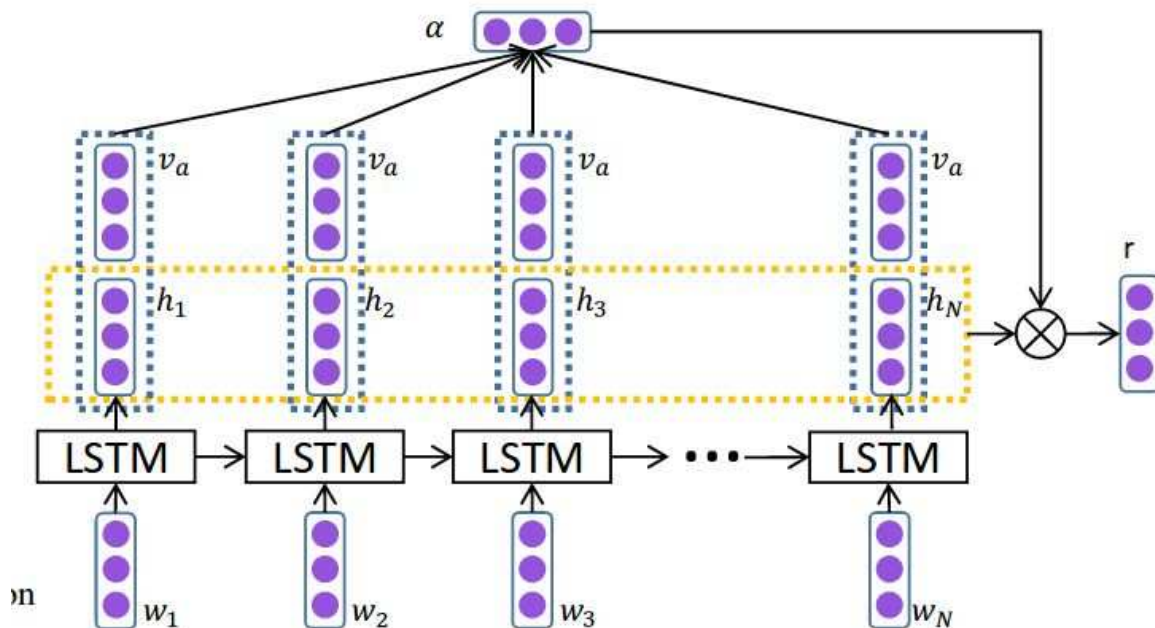
Attention-based LSTM for Aspect-level Sentiment Classification

Staffs are not that friendly, but the taste covers all.

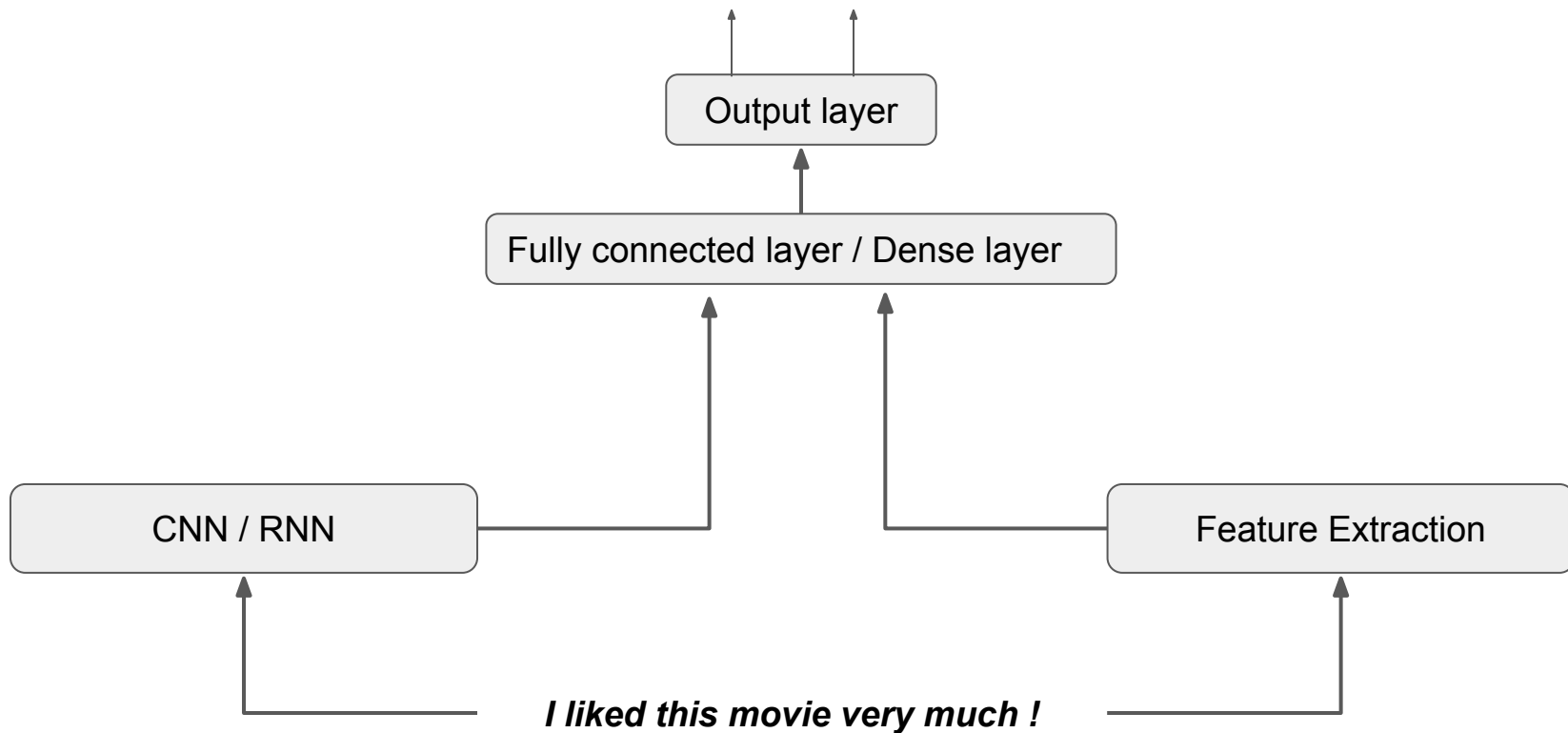
- Aspects / Aspect Category
 - Service** → *negative*
 - Food** → *positive*

Instances and attentions:

- {*Staffs are not that friendly, but the taste covers all, **Service***}
- {*Staffs are not that friendly, but the taste covers all, **Food***}

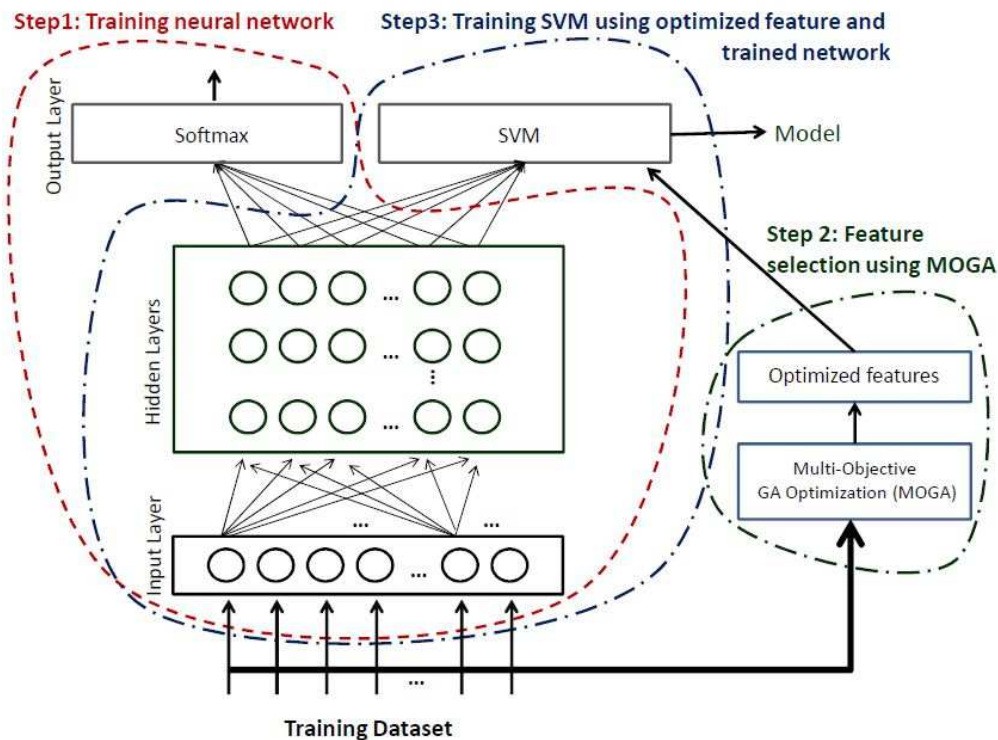


CNN / RNN with extra features



A Hybrid Deep Learning Architecture for Sentiment Analysis

1. Training of a typical **convolutional neural network (CNN)**
 - Obtain weight matrix
2. A **multi-objective GA based optimization technique (MOGA)** for extracting the optimized set of features
 - Two objectives
 - *Accuracy* (maximize)
 - *Num of features* (minimize)
 - Optimized feature set
3. Training of **SVM with non-linear kernel** utilizing the network trained in first step and optimized features

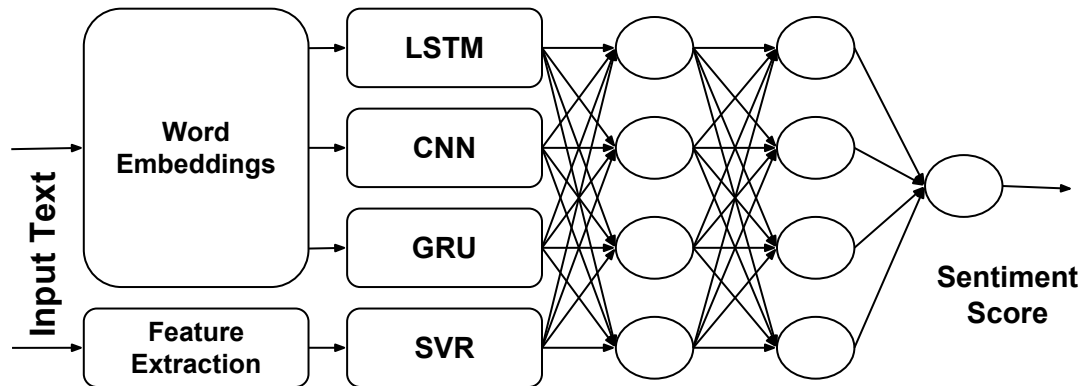


A Multilayer Perceptron based Ensemble Technique for Fine-grained Financial Sentiment Analysis

- Given a financial tweet, predict its sentiment score *w.r.t.* a company.

E.g. best stock: \$WTS +15%

- Company/Cashtag: *WTS*
 - Sentiment: *Positive*
 - Intensity score: *0.857*
- Trained
 - Three DL methods:
 - LSTM, CNN & GRU
 - One feature driven
 - SVR
 - Tf-idf, lexicons
 - Performance:
 - Numerically → Similar
 - Qualitative → Contrasting



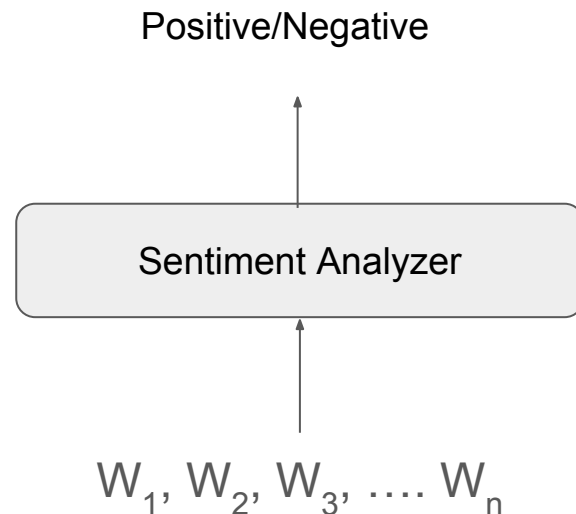
Designing a Sentiment Analyzer

Problem definition

- Sentiment Analysis:
 - Given a sentence predicts its sentiment class.

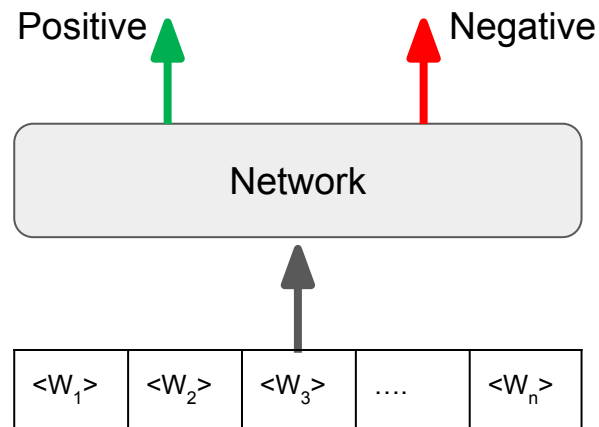
● I/P: $W_1, W_2, W_3, \dots, W_n$

● O/P: positive / negative



Data Representation

- Convert sequence of words into some numeric representation (i.e. word embeddings)
 - Let dimension of each word be 300
- Let label representation be
 - Positive $\rightarrow 0$
 - Negative $\rightarrow 1$



Word vector of 300 dimension each

Data Representation - Train/Test file

Sentence (sequence of word embeddings)					Label (Positive: 0, Negative: 1)
<300 dim vector for W_1 >	<300 dim vector for W_2 >	<300 dim vector for W_3 >	<300 dim vector for W_n >	0
<300 dim vector for W_1 >	<300 dim vector for W_2 >	<300 dim vector for W_3 >	<300 dim vector for W_n >	1
<300 dim vector for W_1 >	<300 dim vector for W_2 >	<300 dim vector for W_3 >	<300 dim vector for W_n >	1
<300 dim vector for W_1 >	<300 dim vector for W_2 >	<300 dim vector for W_3 >	<300 dim vector for W_n >	0
<300 dim vector for W_1 >	<300 dim vector for W_2 >	<300 dim vector for W_3 >	<300 dim vector for W_n >	1
...					
<300 dim vector for W_1 >	<300 dim vector for W_2 >	<300 dim vector for W_3 >	<300 dim vector for W_n >	0

Implementation of a typical NN: Basic steps

1. Import necessary libraries
2. Design Network
3. Compile Network
4. Prepare/Load training data
5. Train the network
6. Evaluate the network
 - a. Prepare/Load testing data
 - b. Predict o/p
 - c. Print test and its prediction

Keras: A deep learning API

- Python based API
- Wrapper that support three packages at the backend
 - Theano (supports ended recently)
 - TensorFlow
 - CNTK
- <https://keras.io/>

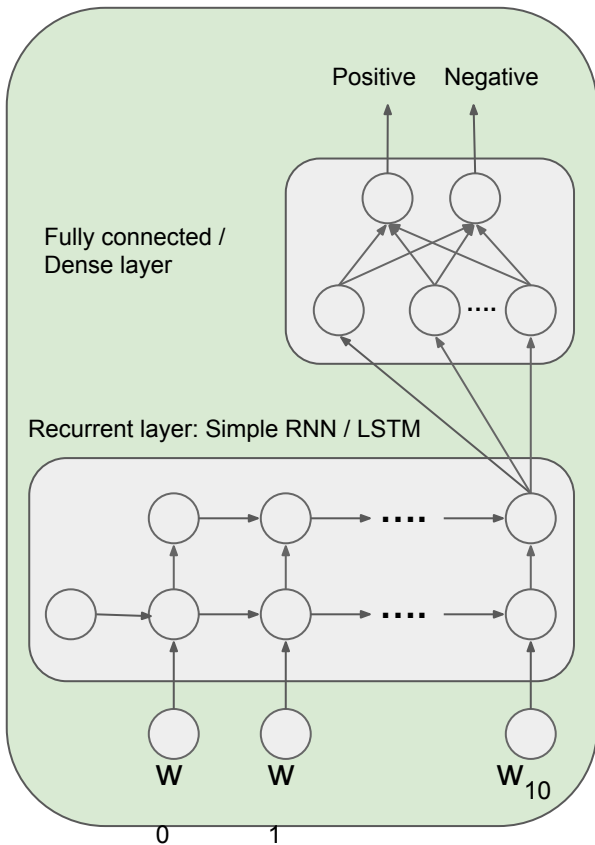
Implementation using Keras: **Import necessary libraries**

1/6

```
import numpy # Numpy for mathematical ops
import keras # Keras main library
from keras.models import Sequential # Model type
from keras.layers import SimpleRNN, LSTM # Recurrent unit
from sklearn import metrics # For evaluation
```

Implementation using Keras: Design Network

2/6



```
numInNeurons = 300
numOutNeurons = 2
numHiddentUnitsRecurrent = 100
numHiddentUnitsDense = 70
seqLength = 10

model = Sequential()      # Instantiate sequential network
# Add a SimpleRNN Layer.
# input_dim is required only in the first layer of the network.
model.add(SimpleRNN(numHiddentUnits,
input_shape=(seqLength, numInNeurons),
return_sequences=true, activation='sigmoid'))
model.add(SimpleRNN(numHiddentUnits, activation='sigmoid'),
return_sequences=false)
model.add(Dense(numHiddentUnitsDense, activation='tanh'))
# If we need to add more layers we have to call model.add() again.
model.add(Dense(numOutNeurons, activation='softmax'))
```

Implementation using Keras: Compile the network

3/6

```
model.compile(optimizer='sgd', loss='mse')
```

```
# Validate the network. If any issues (dimension mismatch etc.) are found, they will  
be reported.
```

```
# Optimization algorithm is stochastic gradient descent
```

```
# Loss is mean squared error
```

```
# At this point network is ready for training
```

Implementation using Keras: Prepare/Load training data

4/6

```
data_train = np.loadtxt(open('train_data.txt','r'))
label_train = np.loadtxt(open('train_label.txt','r'))

numTrainInst = 1000           # Number of instances in training data

data_train = data_train.reshape(numTrainInst, seqLength, numInNeurons)

# Input file has 'numTrainInst', each instance has 'seqLength' and each unit has
dimension 'numInNeurons'.
```

Implementation using Keras: Train the network

5/6

```
model.fit(X, O, epochs=5, validation_split=0.2) # Train network for 5 epochs
```

Epoch 1/5

800/800 [=====] - 0s - loss: 0.4118 - val_loss: 0.4520

Epoch 2/5

800/800 [=====] - 0s - loss: 0.4116 - val_loss: 0.4517

Epoch 3/5

800/800 [=====] - 0s - loss: 0.4114 - val_loss: 0.4514

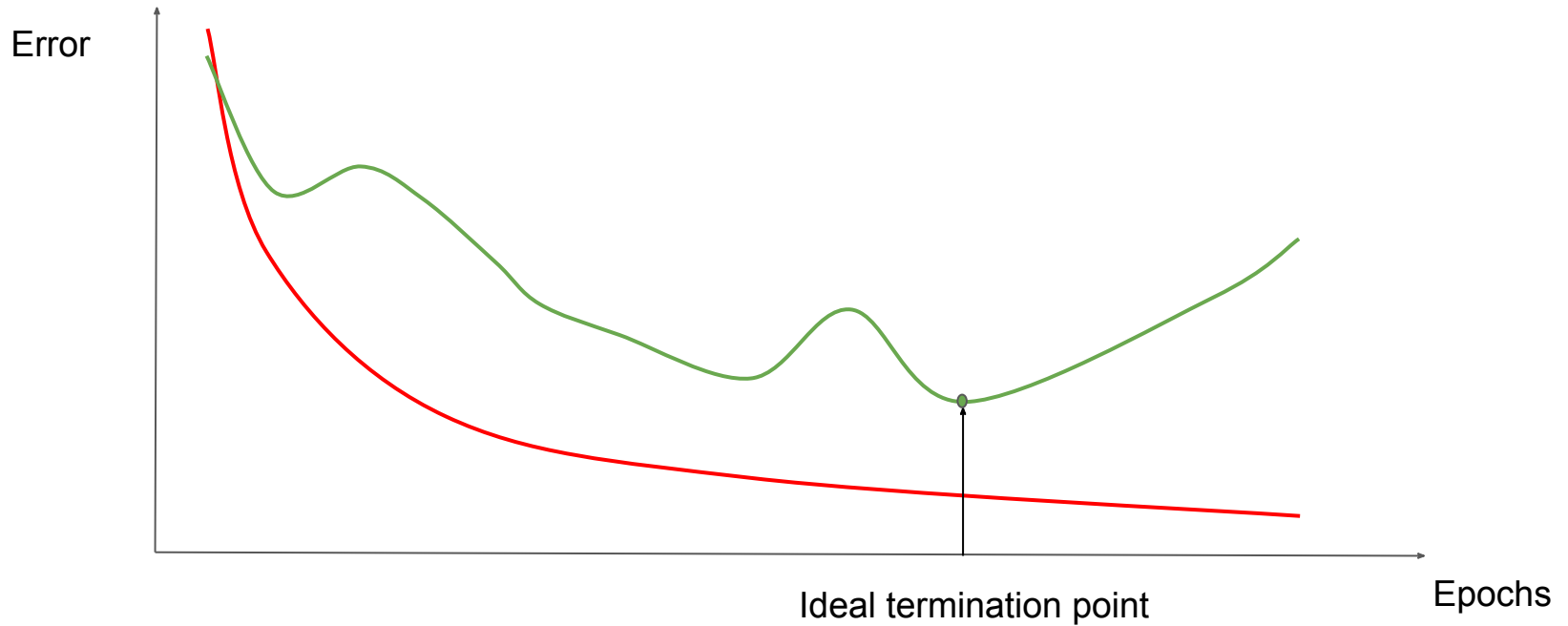
Epoch 4/5

800/800 [=====] - 0s - loss: 0.4112 - val_loss: 0.4512

Epoch 5/5

800/800 [=====] - 0s - loss: 0.4110 - val_loss: 0.4509

Training v/s Validation loss



Implementation using Keras: Evaluate the network

6/6

a. Prepare the test data

```
data_test = np.loadtxt(open('test_data.txt', 'r'))
label_test = np.loadtxt(open('test_label.txt', 'r'))
numTestInstances = 100
data_test = data_test.reshape(numTestInstances, seqLength, numInNeurons)
```

b. Compute predictions

- Predict probabilities

```
pred_probability = model.predict(data_test) # predict o/p prob.
```

- Predict o/p

```
prediction = model.predict_classes(data_test) # predict o/p
```

Implementation using Keras: Evaluate the network..(contd) 6/6

- c. Print test, probability and its prediction

```
print ('Test instances:', data_test)
print ('Actual:', label_test)
print ('Prediction:', prediction)
print ('Prediction Probabilities:', pred_probability)
```

- d. Evaluate predictions

```
print('Accuracy: ' + metrics.accuracy_score(label_test , pred_test))
```


Thank You!

AI-NLP-ML Group, Department of CSE, IIT Patna (<http://www.iitp.ac.in/~ai-nlp-ml/>)

Research Supervisors:

- Prof. Pushpak Bhattacharyya
- Dr. Asif Ekbal
- Dr. Sriparna Saha

Named Entity Recognition Using Deep Learning

Rudra Murthy

Center for Indian Language Technology,
Indian Institute of Technology Bombay

rudra@cse.iitb.ac.in

<https://www.cse.iitb.ac.in/~rudra>



*Deep Learning Tutorial.
ICON 2017, Kolkata
21th December 2017*



Outline

- What is NER?
- Traditional ML Approaches
- Motivating Deep Learning
- Deep Learning Solutions
- Summary

Introduction

What is Named Entity Recognition?

- The task of identifying **person** names, **location** names, **organization** names and other **miscellaneous entities** in a given piece of text.
- Example:
 - **Malinga** omitted from squad for **Pakistan** ODIs

Malinga will be tagged as **Person** and **Pakistan** as **Location** entity

You thought NER was trivial



more awesome pictures at THEMETAPICTURE.COM

Challenges

- Named Entities are ambiguous
 - I went to Washington
 - I met Washington
- Named Entities form an open class
 - Box8
 - Alphabet
 - .
 - .

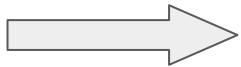
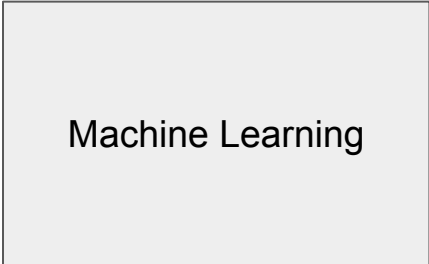
Challenges

List of Unique/Crazy Person Names

- Ahmiracle
- Anna...
- I'munique
- Baby Girl
- Abcde
- North West
- Melanomia
- Heaven Lee
- Tu Morrow
- Moxie Crimefighter
- Abstinence
- Apple
- Facebook
- Danger
- Colon
- Mercury Constellation Starcuiser
- Pilot Inspektor
- Rage
- Billion
- Audio Science
- Sadman
- Hashtag

Traditional ML Approaches

Vince's	Person
maiden	O
test	O
fifty	O
keeps	O
England	Misc
ticking	O
Mumbai	Misc
drop	O
Nayar	Person
.	.

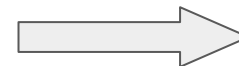


NER Model

Vince's	Person
maiden	O
test	O
fifty	O
keeps	O
England	Team
ticking	O
Mumbai	Team
drop	O
Nayar	Person
.	.



Machine Learning
*learn probabilities over words



NER Model

$P(\text{Person} \mid \text{Vince's}) = ?$
 $P(\text{Location} \mid \text{Vince's}) = ?$
 $P(\text{Team} \mid \text{Vince's}) = ?$
 $P(O \mid \text{Vince's}) = ?$

Problem Formulation

Given a word sequence (w_1, w_2, \dots, w_n) find the most probable tag sequence (y_1, y_2, \dots, y_n)

i.e, find the most probable entity label for every word in the sentence

Best Tag
Sequence

$$y^* = P(y_1, y_2, \dots, y_n | w_1, w_2, \dots, w_n)$$

Why sequence labeling and not classification task?

Sequence labeling performs better at identifying named entity phrases

Problem Formulation (CRF)

Given a word sequence (w_1, w_2, \dots, w_n) find the most probable tag sequence (y_1, y_2, \dots, y_n)

$$P(\mathbf{y} | \mathbf{w}) = \exp\left(\sum_{i=1}^n \sum_k \lambda_k f_k(y_i, y_{i-1}, \mathbf{x})\right)$$

Here, $f_k(y_t, y_{t-1}, \mathbf{x})$ is a feature function whose weights λ_k needs to be learned during training

The feature function is used to define various features

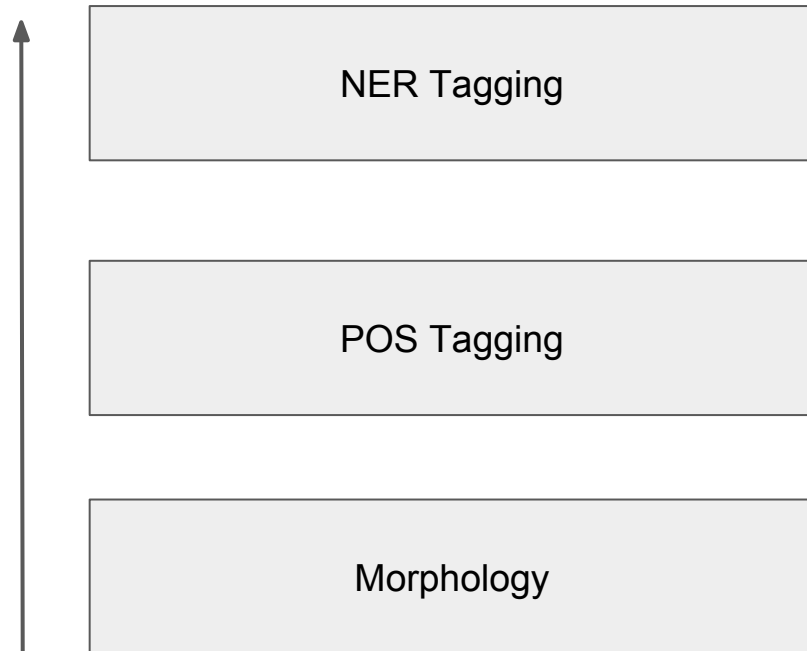
Typical Features

- **Word Features**
- **Subword Features**
- **Context Words**
- POS Tag
- Gazetteers
- Suffix Gazetteers
- Handcrafted Features
 - Does the word begin with an uppercase character?
 - Contains any digits?
 - Contains special characters?

Why Deep Learning?

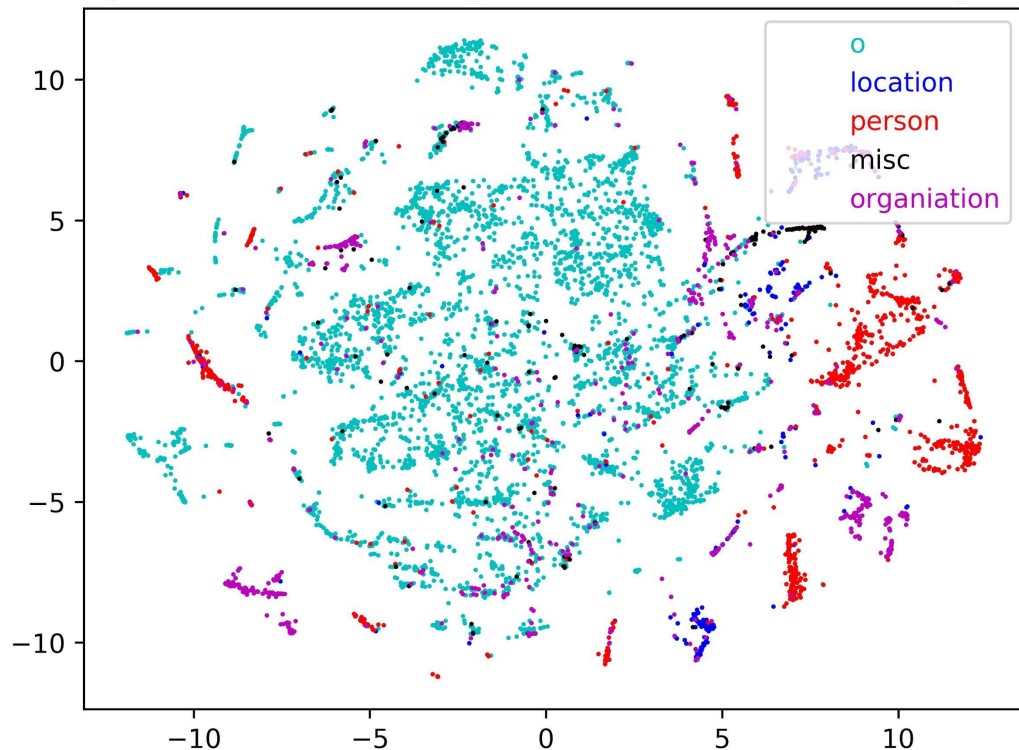
Why Deep Learning?

- Neural networks provide an hierarchical architecture
- Lower layers of the network can discover subword features
- Layers above it can be used to discuss word specific features
- The higher layer can use the information coming from lower layers to identify named entities



Word Embeddings

2-d plot of word embeddings annotated with named entity tags



- Plot of word Spectral word embedding for words from English CoNLL 2003 test data
- Choose the most frequent named tag for every word
- We observe named entities of the same type forming a cluster in the embedding space

Deep Learning Solutions

- We have looked at various neural network architectures
- What are the important features for NER?
- What neural network architectures can we use to make the model learn these features?

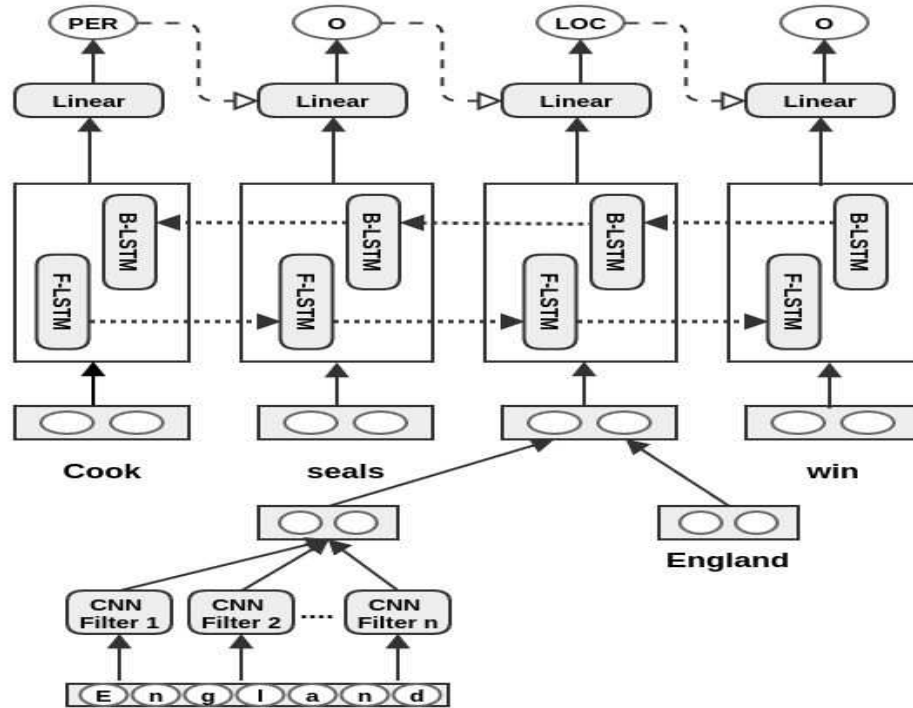
Deep Learning Model for NER Timeline

Model	Subword		Word	
	CNN	Bi-LSTM	CNN	Bi-LSTM
Hammerton [2003]				✓
Collobert et al. [2011]			✓	
dos Santos et al. [2015]	✓		✓	
Huang et al. [2015]				✓
Chiu and Nichols [2016]	✓			✓
Murthy and Bhattacharyya [2016]	✓			✓
Lample et al. [2016]		✓		✓
Ma and Hovy [2016]	✓			✓
Yang et al. [2017]		✓		✓

Deep Learning Model for NER [Murthy and Bhattacharyya [2016]]

- Given a dataset D consisting of tagged sentences
 - Let $X = \{x_1, x_2, \dots, x_n\}$ be the sequence of words in a sentence
 - Let $Y = \{y_1, y_2, \dots, y_n\}$ be the sequence of corresponding tags
- Goal is to maximize the likelihood of the tag sequence given the word sequence
 - maximize $P(Y | X)$
 - maximize $P(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_n)$
 - maximize $\prod_{i=1}^n P(y_i | x_1, x_2, \dots, x_n, y_{i-1})$
- We maximize the log-likelihood for every tag sequence in the training data

Deep Learning Architecture for NER



Deep Learning Architecture for NER

- The input to the model is words and the character sequence forming the word
- One-hot representation of the word is sent through a Lookup Table
- Lookup Table is initialized with pre-trained embeddings
- Additionally character sequence is fed to CNN to extract sub-word features
- The word embeddings and sub-word features are concatenated to get final word representation
- This representation is fed to a Bi-LSTM layer which disambiguates the word (w.r.t NER task) in the sentence
- Finally, the output from Bi-LSTM model is fed to softmax layer which predicts the named entity label

Word Embeddings

- Word embeddings represent words using d -dimensional real valued vector
- Word embeddings exhibit the property that named entities tend to form a cluster in the embedding space
- Providing word embedding features as input is more informative compared to the one-hot representation
- Word embeddings are updated during training

Subword Features

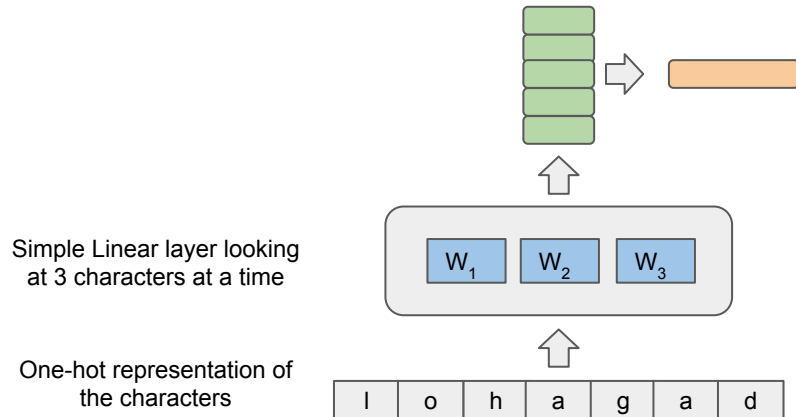
- We use multiple CNNs of varying width to extract sub-word features
- Every character is represented using one-hot vector representation
- The input is a matrix with i^{th} row indicating the one-hot vector of i^{th} character in the word
- The output of CNN is fed to max-pooling layer
- We extract 15-50 features from the CNN for every word
- This forms the sub-word features for the word

Subword Features

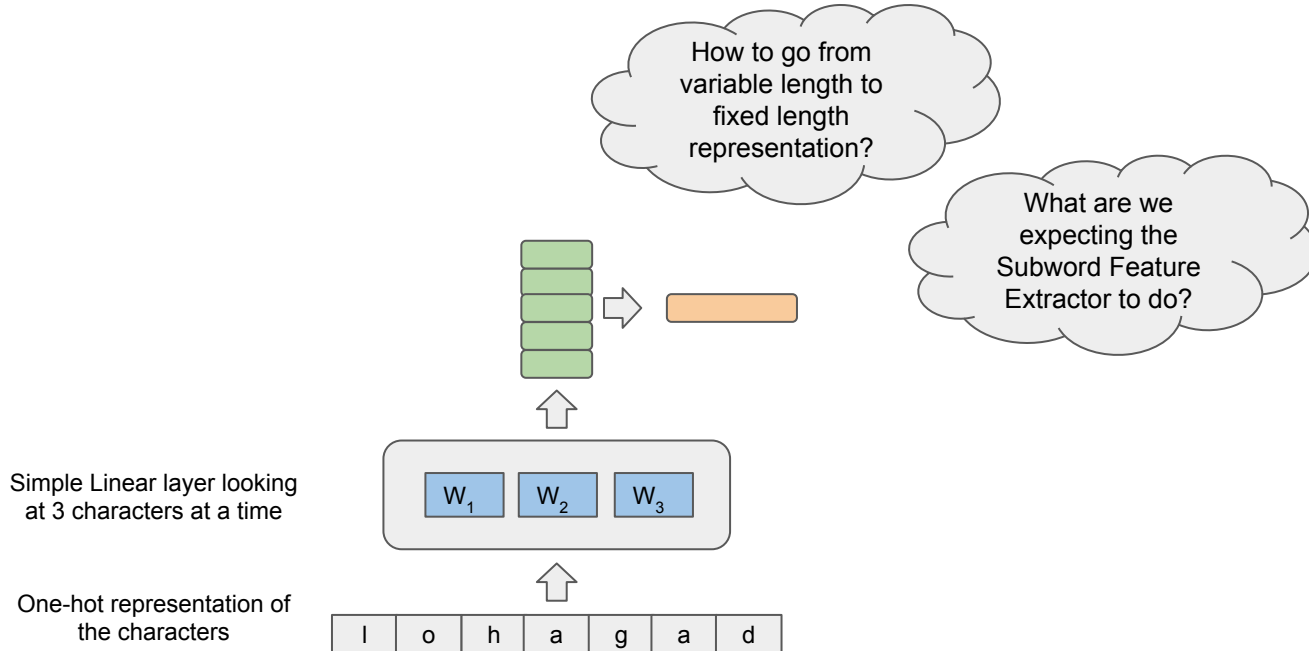
- This module should be able to discover various subword features
- The feature could be capitalization feature, affix features, presence of digits *etc.*

CNNs to Extract Subword Features

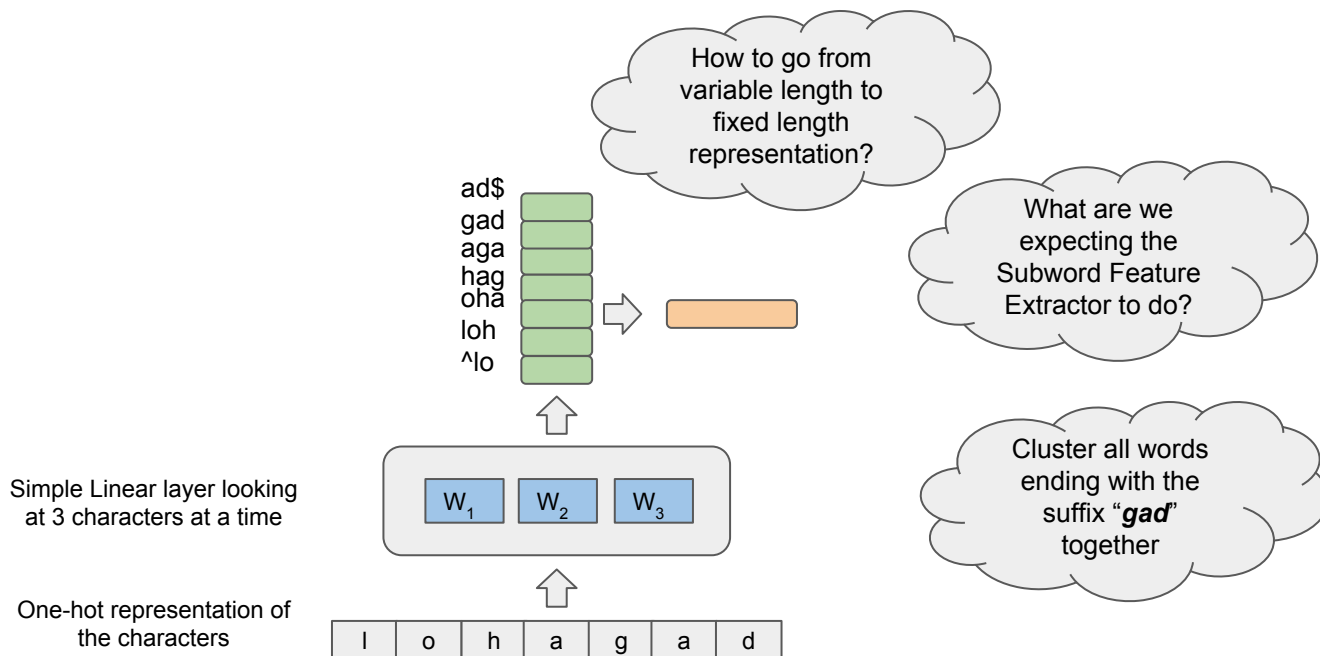
How to go from variable length to fixed length representation?



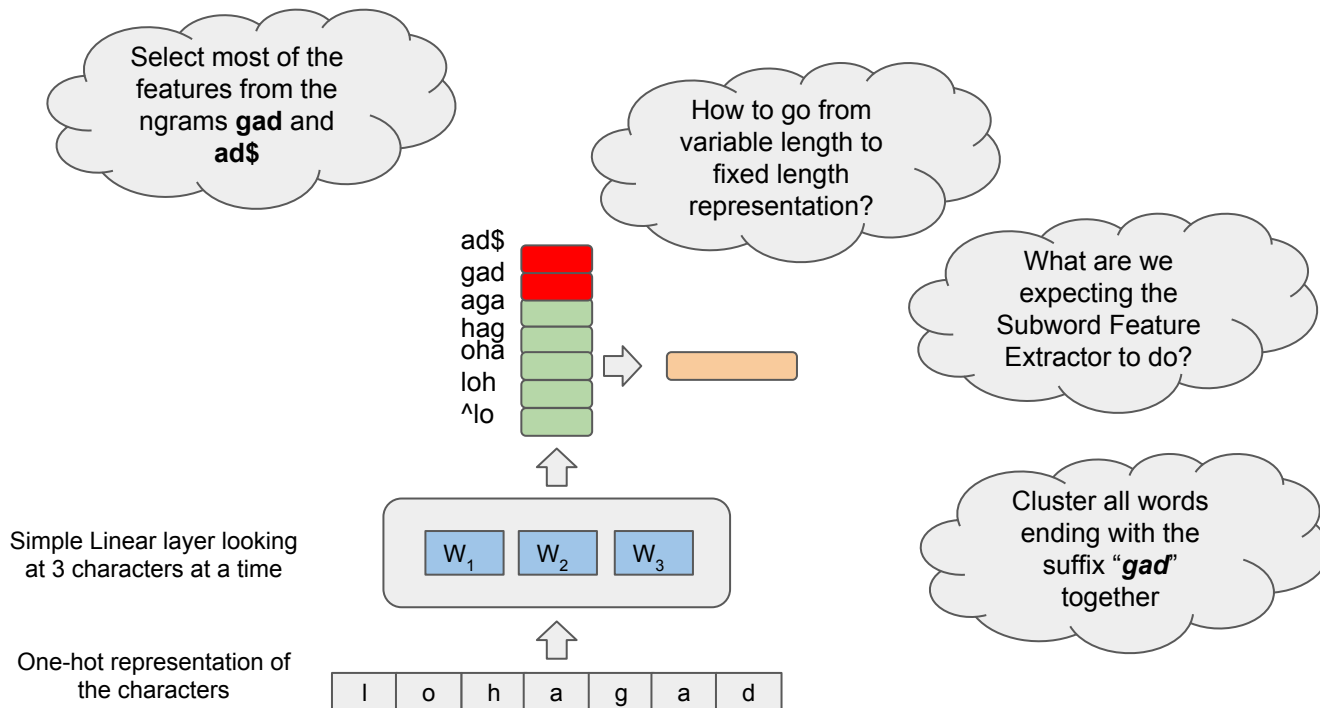
CNNs to Extract Subword Features



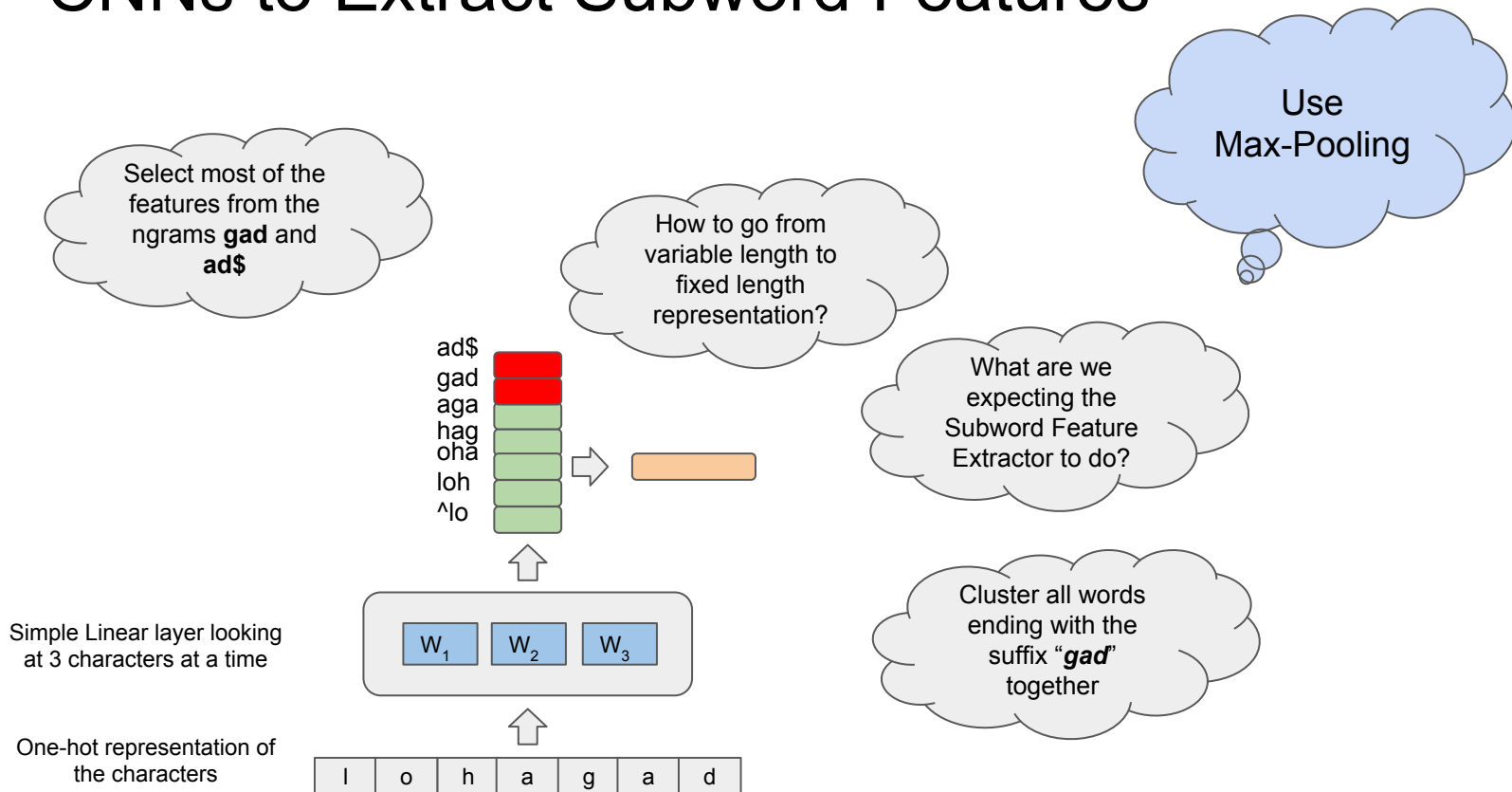
CNNs to Extract Subword Features



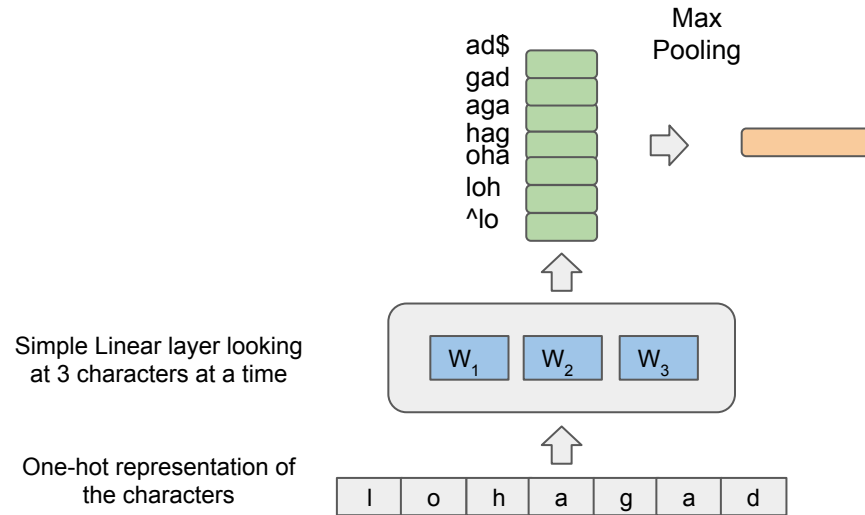
CNNs to Extract Subword Features



CNNs to Extract Subword Features



CNNs to Extract Subword Features



CNNs to Extract Subword Features

- Use CNNs to extract various sub-word features
- By extracting, we mean word with similar features to be closer in the feature space
- The features could be *capitalization features, similar suffixes, similar prefixes, all time expressions, etc.*
- This is similar to say suffix embeddings except that the suffix pattern is discovered by the model

Subword Features



- Plot of subword features for different Marathi words
- We observe that the CNN was able to cluster words based on their suffixes
- The CNN model was able to cluster words with similar suffixes

Bi-LSTM Layer

- We have observed that both word embeddings and subword features are able to cluster similar words together
- All location names forming a cluster in word embedding space
- All words with similar suffixes forming a cluster in the sub-word feature space
- This acts as a proxy feature for suffix features used in traditional ML methods
- Till now we have looked at only global features
- What about local features like contextual features?

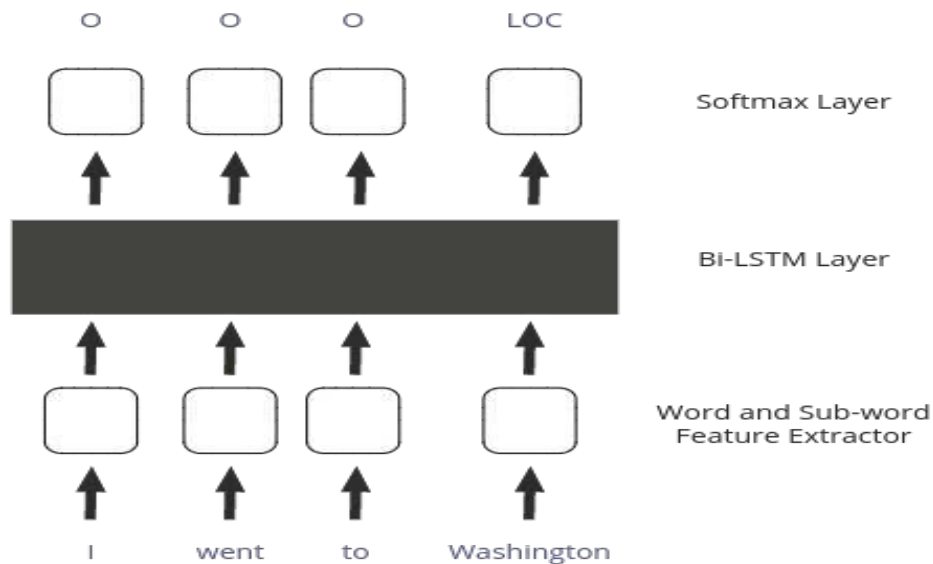
Bi-LSTM Layer

- The word embeddings and extracted sub-word features give global information about the word
- Whether a word is named entity or not depends on the specific context in which it is used
- For example,
 - I went to Washington
 - I met Washington
- The Bi-LSTM layer is responsible for disambiguation of the word in a sentence
- Here the disambiguation is w.r.t named entity tags

Bi-LSTM Layer

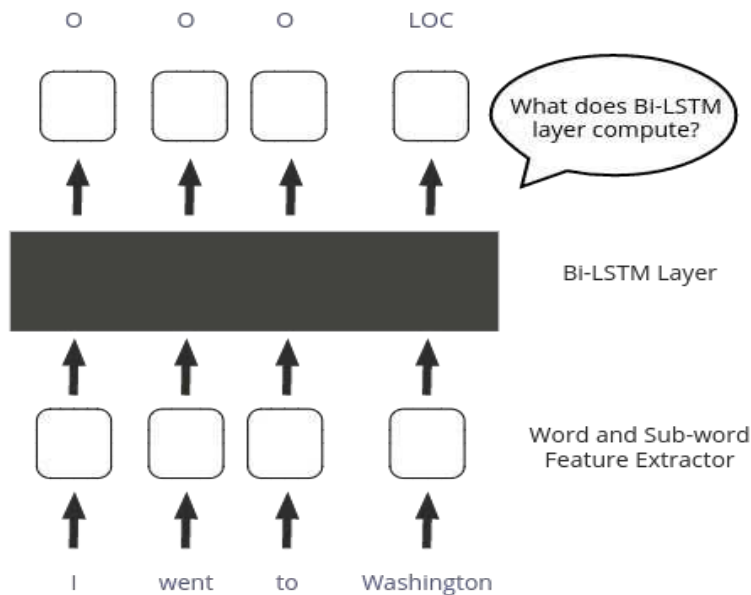
- Given a sequence of words, $\{x_1, x_2, \dots, x_n\}$ the Bi-LSTM layer employs two LSTM modules
- The forward LSTM module reads a sequence from left-to-right and disambiguates the word based on left context
- The forward LSTM extracts a set of features based on current word representation and previous word's forward LSTM output
 - $h_f^i = f(x_i, h_f^{i-1})$
- Similarly, backward LSTM reads sequence from right to left and disambiguates the word based on right context
 - $h_b^i = f(x_i, h_b^{i+1})$

What does Bi-LSTM Layer compute?



Revisiting the Deep Learning Architecture for NER

What does Bi-LSTM Layer compute?



Revisiting the Deep Learning Architecture for NER

Bi-LSTM Layer

- The Bi-LSTM layer extracts a set of features for every word in the sentence
- We will now call this representation as **instance-level representation**
- Consider the sentence snippets,
 - वर्तमान में उत्तर प्रदेश के जौनसार बावर क्षेत्र ...
 - Currently Jaunsar Bavar area of Uttar Pradesh ...
 - ... भावजूद भी कोई प्रोत्साहित (संतोषजनक) उत्तर प्राप्त नहीं हुआ
 - even after that no satisfactory answer was obtained
- The word उत्तर will now have two instance-level representations one for first sentence and the other for second sentence
- We will now query the nearest neighbors for उत्तर using instance-level representations from both sentences

B-LSTM Layer

Word Embedding			Sentence 1			Sentence 2		
Neighbors	Score	Tag	Neighbors	Score	Tag	Neighbors	Score	Tag
प्रदेश	0.8722	-	उत्तरी	0.9088	LOC	उत्तर	0.9183	O
पश्चिम	0.8596	-	उत्तर	0.9033	LOC	उत्तर	0.9155	O
मध्य	0.8502	-	तिब्बत	0.8669	LOC	उत्तर	0.9137	O
पूरब	0.8432	-	शिमला	0.8641	LOC	उत्तर	0.9125	O
अरुणाचल	0.8430	-	किन्नौर	0.8495	LOC	उत्तर	0.9124	O

- The table shows the nearest neighbors (using cosine similarity) for the ambiguous word उत्तर using instance-level representation
- In sentence 1, the nearest neighbors are all location entities
- In sentence 2, we observe different instances of उत्तर appearing as nearest neighbors
 - All the instances of उत्तर takes the answer meaning as in ... कि उत्तर देने वाले व्यक्ति ..

Analyzing Bi-LSTM Layer

Sentence 2			
Neighbors	Score	Tag	Sentence
उत्तर	0.9183	O	कि उत्तर देने वाले व्यक्ति
उत्तर	0.9155	O	अनुसार उत्तर दिये परन्तु
उत्तर	0.9137	O	उसे उत्तर देने में
उत्तर	0.9125	O	सही उत्तर की संभावना
उत्तर	0.9124	O	एक भी उत्तर ना दे

- In sentence 2, we observe different instances of उत्तर appearing as nearest neighbors
 - All the instances of उत्तर takes the answer meaning as in ... कि उत्तर देने वाले व्यक्ति ..

Softmax Layer (Linear + Softmax)

- The output from Bi-LSTM module and correct previous tag is fed as input to Softmax layer
- The correct previous tag is crucial in identifying the named entity phrase boundaries
- During testing, we do not have previous tag information
- We use beam search to find the best possible tag sequence

Results

- We perform the NER experiments on the following set of languages

Language	Dataset
English	CoNLL 2003 Shared Task
Spanish	CoNLL 2002 Shared Task
Dutch	CoNLL 2002 Shared Task
Hindi	IJCNLP 2008 Shared Task
Bengali	
Telugu	
Marathi	In-House Data

Results

- The following Table shows the F1-Score obtained using the Deep Learning system

Language	F1-Score
English	90.94
Spanish	84.85
Dutch	85.20
Hindi	59.80
Marathi	61.78
Bengali	43.24
Telugu	21.11

Demo

Thank You

Questions?

References

- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*,
- dos Santos, C., Guimaraes, V., Niterói, R., and de Janeiro, R. (2015). Boosting named entity recognition with neural character embeddings. *Proceedings of NEWS 2015 The Fifth Named Entities Workshop*, page 9.
- Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991
- Lample, G., Ballesteros, M., Kawakami, K., Subramanian, S., and Dyer, C. (2016). Neural architectures for named entity recognition. In *In proceedings of NAACL-HLT (NAACL 2016).*, San Diego, US

References

- Gillick, Dan and Brunk, Cliff and Vinyals, Oriol and Subramanya, Amarnag "Multilingual Language Processing From Bytes." *In proceedings of NAACL-HLT (NAACL 2016).*, San Diego, US.
- Murthy, Rudra and Bhattacharyya, Pushpak "Complete Deep Learning Solution For Named Entity Recognition." *CICLING 2016, Konya, Turkey*
- Murthy, Rudra and Khapra, Mitesh and Bhattacharyya, Pushpak "Sharing Network Parameters for Crosslingual Named Entity Recognition" *CoRR*, *abs/1607.00198*

Sequence to Sequence Learning Using Deep Learning

Anoop Kunchukuttan

Center for Indian Language Technology,
Indian Institute of Technology Bombay

anoopk@cse.iitb.ac.in

<https://www.cse.iitb.ac.in/~anoopk>



*Deep Learning Tutorial.
ICON 2017, Kolkata
21th December 2017*



Outline

- Introduction
- Neural Machine Translation
- Summary

Outline

- Introduction
- Neural Machine Translation
- Summary

Introduction

- Typical machine learning approaches deals with fixed input or variable-length input and fixed output
 - Image Classification
 - Sentiment Analysis
 - ...
- Certain tasks require the machine learning approach to generate variable-length outputs
 - Summary Generation
 - Machine Translation
 - Image Descriptions
- We will look at Deep Learning Approaches for such tasks, specifically we focus on Machine Translation

What is Machine Translation?

Automatic conversion of text/speech from one natural language to another



Be the change you want to see in the world

वह परिवर्तन बनो जो संसार में देखना चाहते हो

Outline

- Introduction
- Neural Machine Translation
- Summary

Neural Machine Translation

SMT, Rule-based MT and Example based MT manipulate **symbolic representations of knowledge**

Every word has an atomic representation,
which can't be further analyzed

No notion of similarity or relationship between words

- Even if we know the translation of `home`, we can't translate `house` if it an OOV

home	0	1	0	0	0
water	1	0	1	0	0
house	2	0	0	1	0
tap	3	0	0	0	1

Difficult to represent new concepts

- We cannot say nothing about 'mansion' if it comes up at test time
- Creates problems language model as well \Rightarrow whole are of smoothing exists to overcome this problem

Neural Network techniques work with **distributed representations**

Every word is represented by a vector of numbers

- No element of the vector represents a particular word
- The word can be understood with all vector elements
- Hence distributed representation
- But less interpretable

Can define similarity between words

- Vector similarity measures like cosine similarity
- Since representations of `home` *and* `house`, we may be able to translate `house`

home
water
house
tap

0.5	0.6	0.7
0.2	0.9	0.3
0.55	0.58	0.77
0.24	0.6	0.4

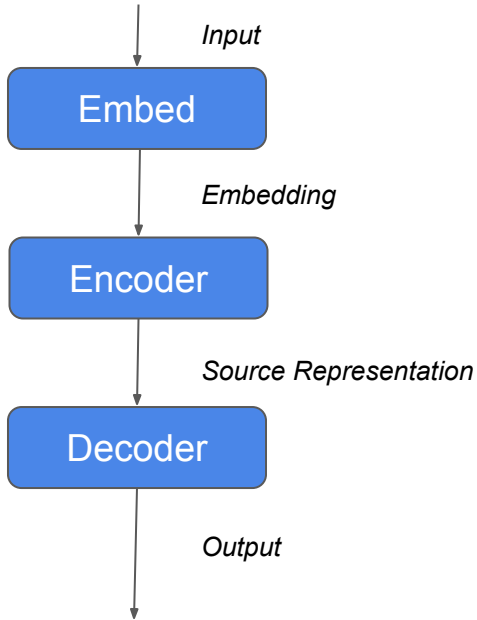
Word vectors
or embeddings

New concepts can be represented using a vector with different values

Symbolic representations are **continuous representations**

- **Generally computationally more efficient** to work with continuous values
- Especially optimization problems

Encode - Decode Paradigm



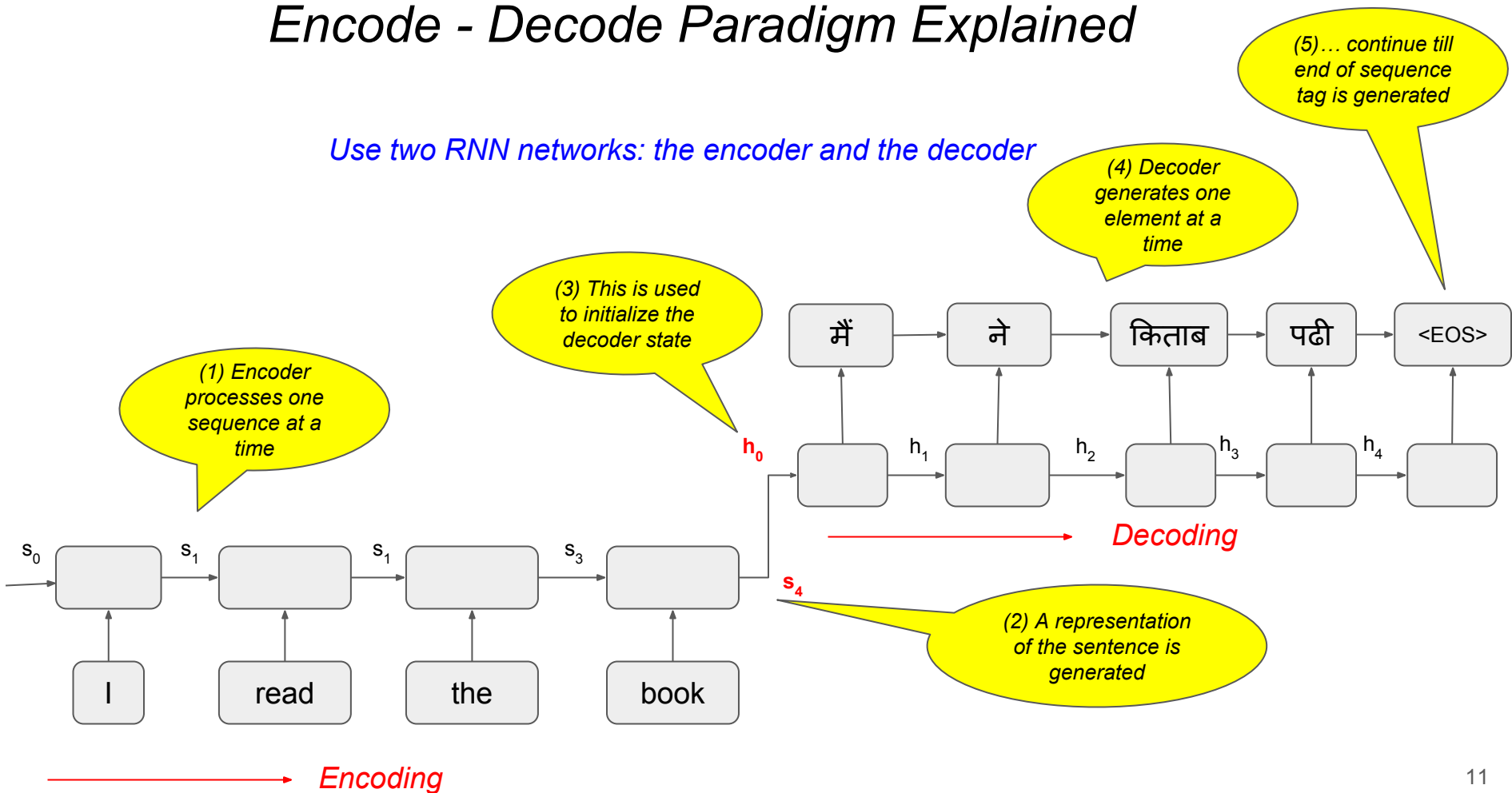
Entire input sequence is processed before generation starts
⇒ In PBSMT, generation was piecewise

The input is a sequence of words, processed one at a time

- *While processing a word, the network needs to know what it has seen so far in the sequence*
- *Meaning, know the history of the sequence processing*
- *Needs a special kind of neural: **Recurrent neural network unit** which can keep state information*

Encode - Decode Paradigm Explained

Use two RNN networks: the encoder and the decoder



This approach reduces the entire sentence representation to a single vector

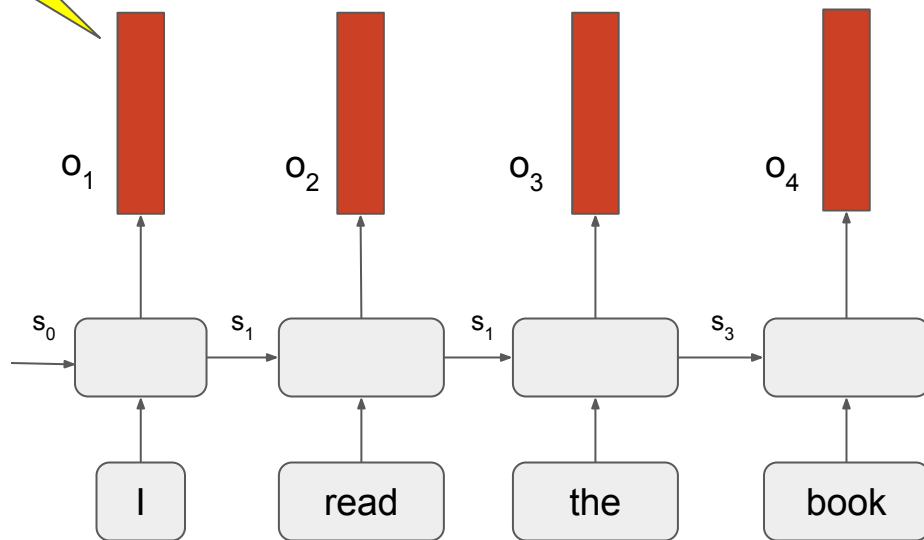
Two problems with this design choice:

- A single vector 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 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

Annotation vectors



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

Note: in the encoder-decode paradigm, we ignore the encoder outputs

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

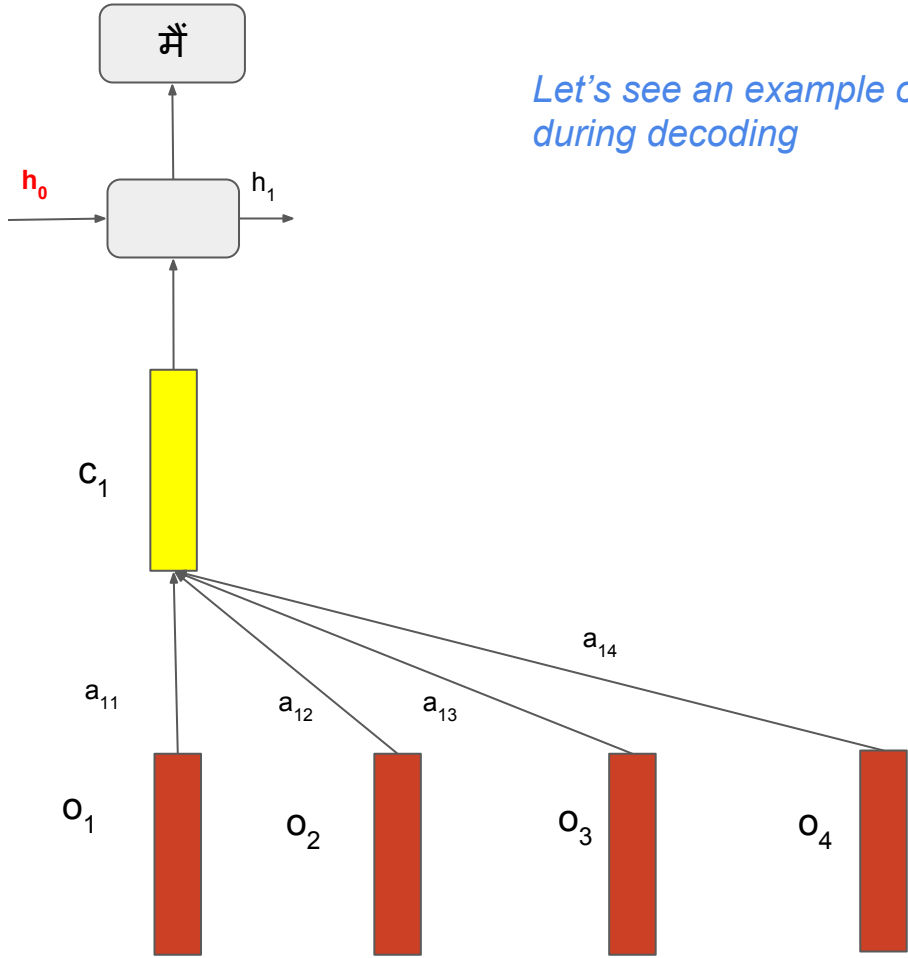
eg. To generate the 3rd target word, the 3rd annotation vector (hence 3rd source word) is most important

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**

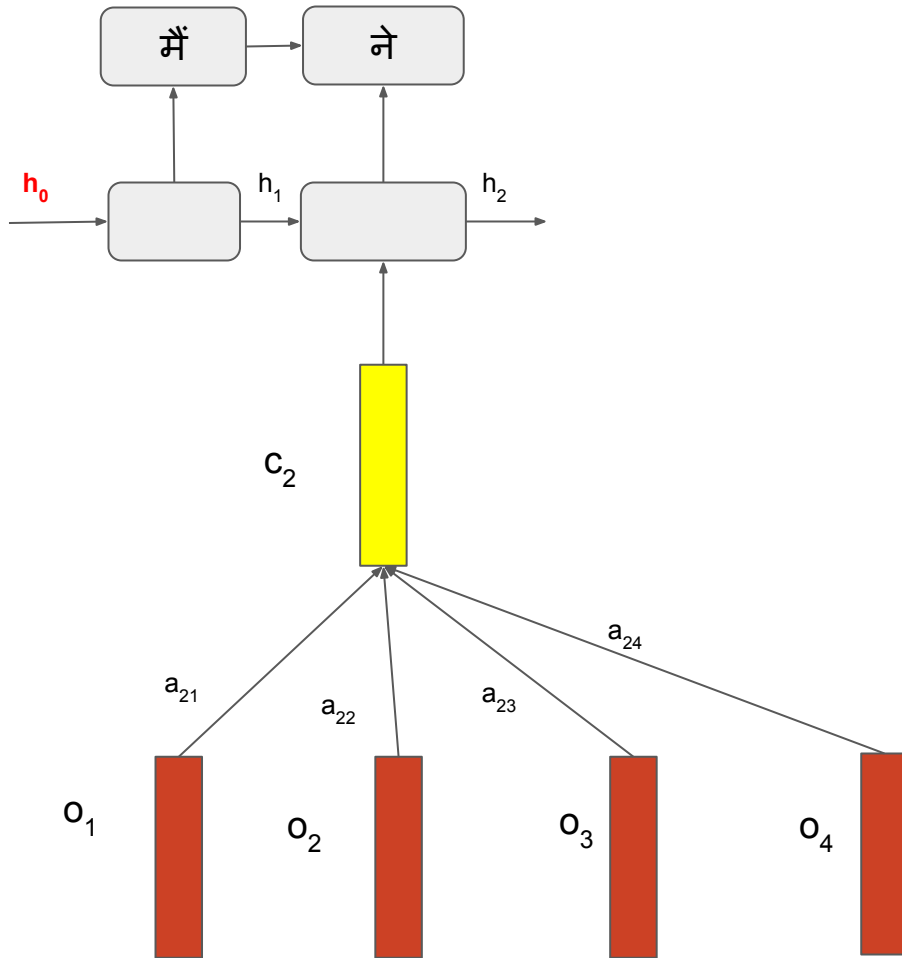
This averaged **context vector** is an input to the decoder

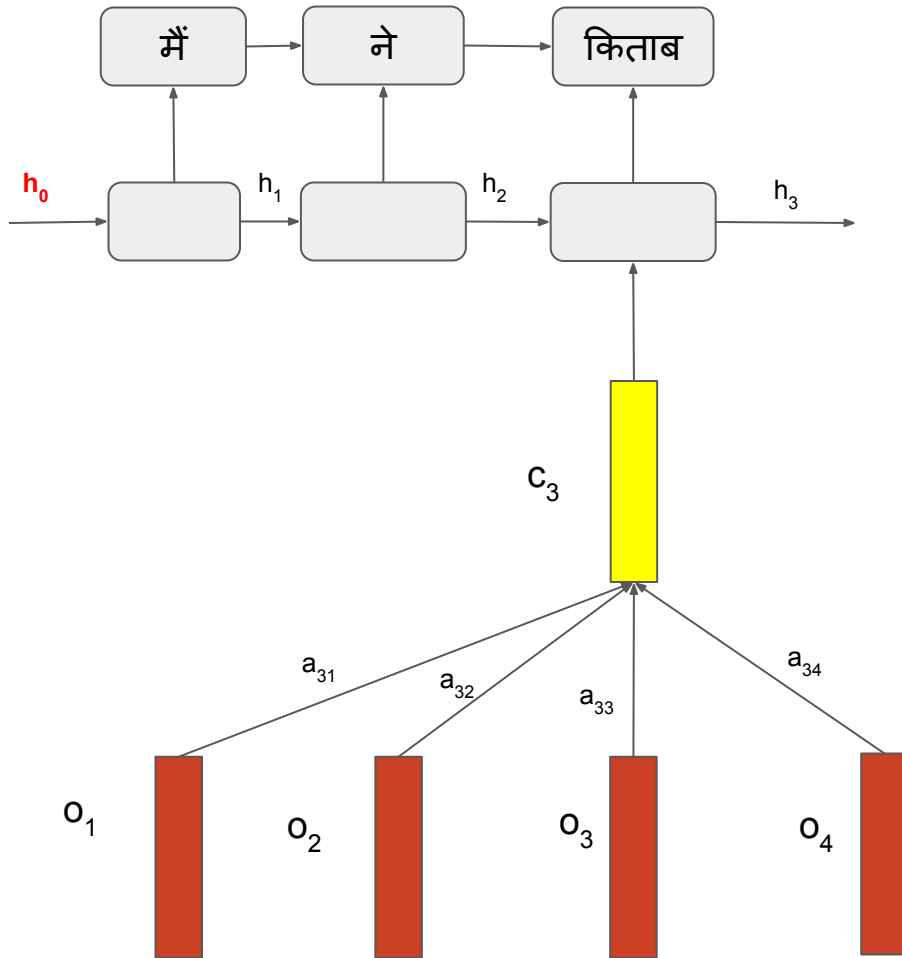
Let's see an example of how the **attention mechanism** works during decoding

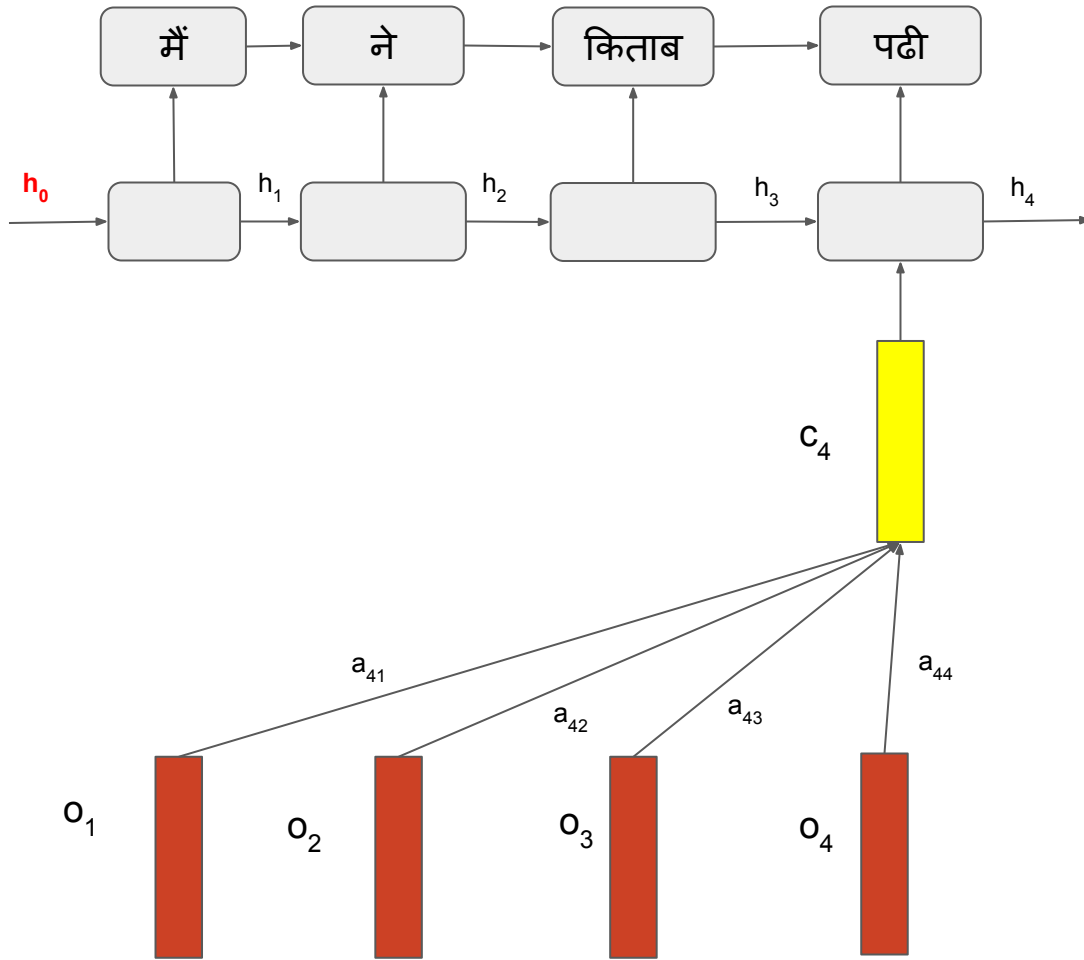


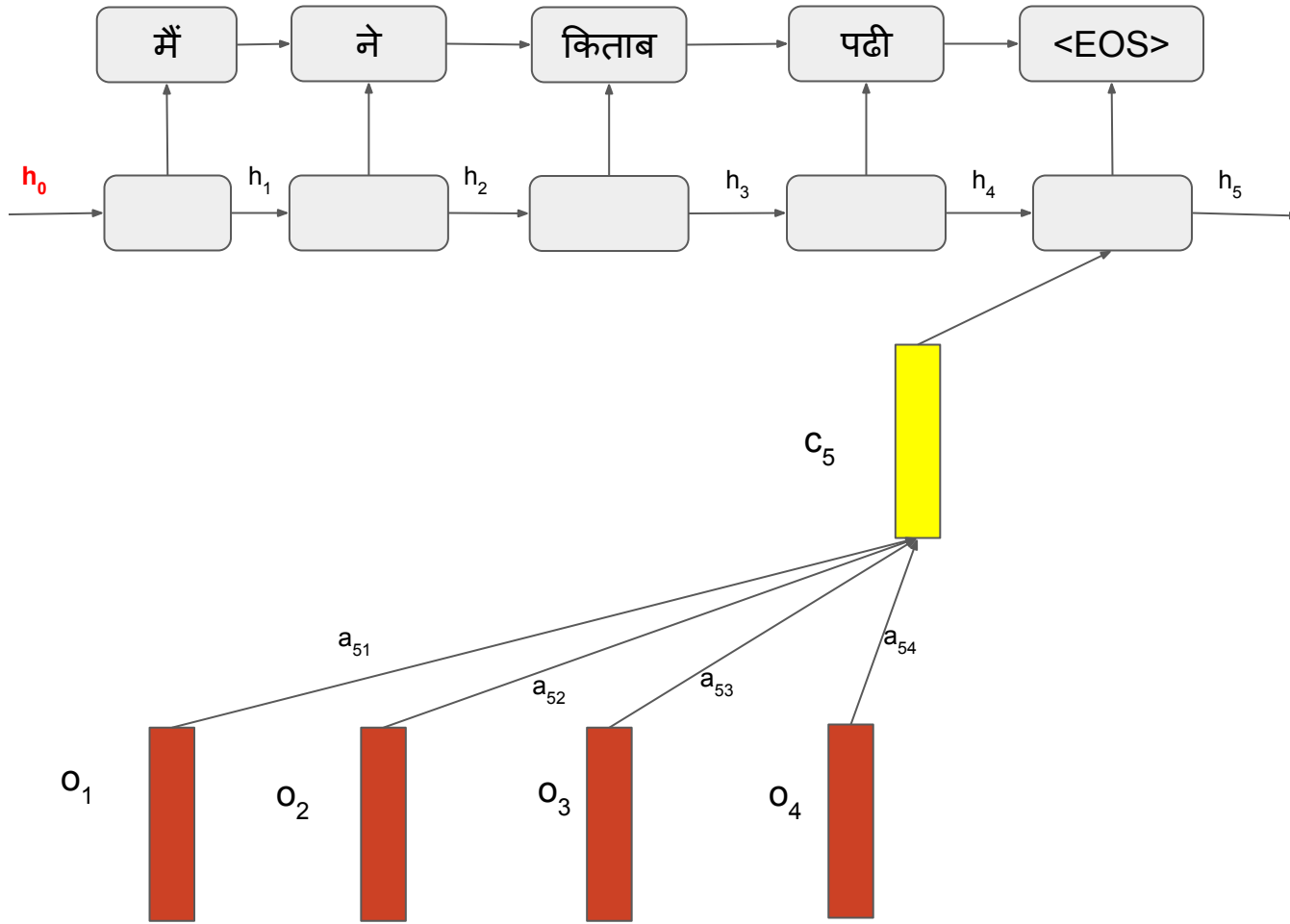
$$c_i = \sum_{j=1}^n a_{ij} o_j$$

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









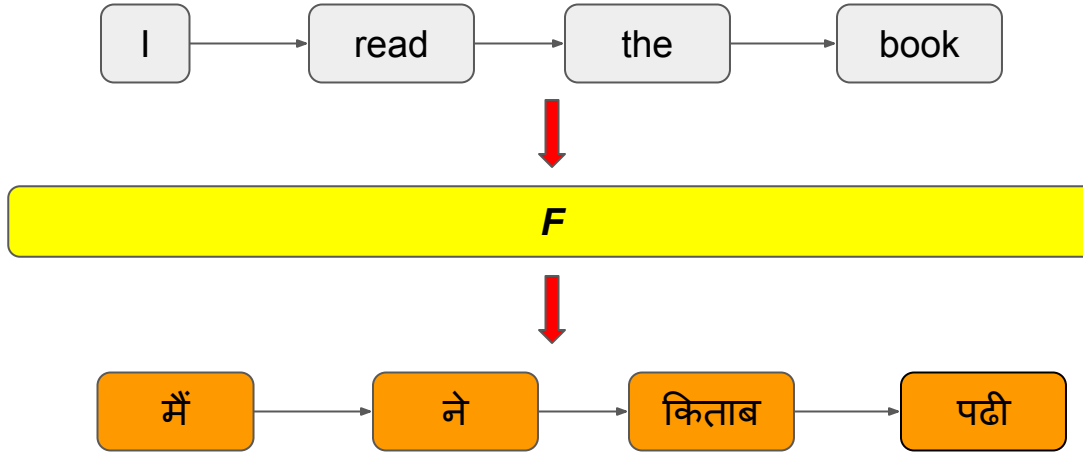
*But we do not know the attention weights?
How do we find them?*

Let the training data help you decide!!

Idea: Pick the attention weights that maximize the translation accuracy
(more precisely, decrease training data loss)

- *Note ⇒ no separate language model*
- *Neural MT generates fluent sentences*
- *Quality of word order is better*
- *No combinatorial search required for evaluating different word orders:*
 - *Decoding is very efficient compared to PBSMT*
- *Exciting times ahead!*

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 very general framework

Sequence to Sequence transformation is a very general framework

Many other problems can be expressed as sequence to sequence transformation

- *Summarization: Article \Rightarrow Summary*
- *Question answering: Question \Rightarrow Answer*
- *Image labelling: Image \Rightarrow Label*
- *Transliteration: character sequence \Rightarrow character sequence*

Resources for Reading

Books & Articles

- Machine Translation. Pushpak Bhattacharyya (book)
- Neural Machine Translation. Kyunghyun Cho (online)
 - <https://devblogs.nvidia.com/paralleforall/introduction-neural-machine-translation-with-gpus/>

Thank You!

<https://www.cse.iitb.ac.in/~anoopk>

Question Answering

Md Shad Akhtar

Research Scholar

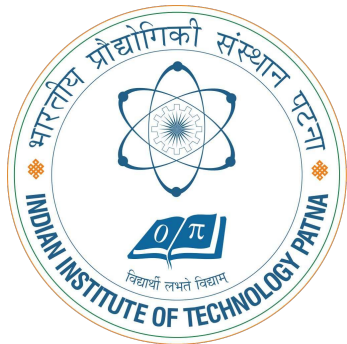
AI-NLP-ML Group

Department of Computer Science & Engineering

Indian Institute of Technology Patna

shad.pcs15@iitp.ac.in

<https://iitp.ac.in/~shad.pcs15/>



Thanks to *Deepak Gupta*, 1
Research Scholar, IIT Patna

Question Answering

- System that automatically answer questions posed by humans in natural language.



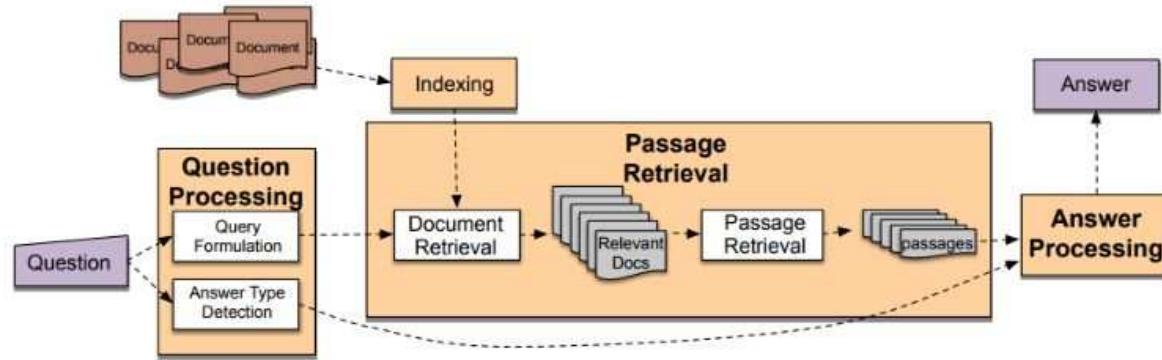
Why Question Answering?

- Conversational Agents
 - Ask it questions. Tell it to do things. Facebook (M), Apple (Siri), Google Assistant
- Biomedical and clinical QA
 - There is a need of system that accepts the queries from medical practitioners in natural language and returns the answers quickly and efficiently.
- E-commerce sites
 - E-commerce sites can exploit the user reviews to provide the answers of various user questions.

Categories of QA

- Factoid Questions
 - Question that can be answered with simple facts expressed in short text answers.
 - **Question:** *Who is the author of the book Wings of Fire?*
 - **Answer:** *A. P. J, Abdul Kalam*
- List Questions
 - Question requires multiple facts to be returned in answer to a question.
 - **Question:** *What are the islands in India?*
 - **Answer:** *Andaman Island, Nicobar Island, Labyrinth Island, Barren Island*
- Descriptive Questions
 - The answer can be short descriptive from a single sentence to multiple but limited (2-3) sentences. It can also be the long descriptive where answer can be a paragraph with meaning full information of the natural language query.
 - **Question:** *What is Greenhouse effect?*
 - **Answer:** *The analogy used to describe the ability of gases in the atmosphere to absorb heat from the earth's surface.*

A typical IR based QA



- Question Processing
 - Type of entity the answer should consists of (person, location, time, etc.).
 - The query specifies the keywords that should be used for the IR system to use in searching
- Passage Retrieval
 - Collect the document against the query for documents.
- Answer Processing
 - To extract a specific answer from the passage.
 - Use information about the expected answer type to find the answer

Sub-problems of QA

- Semantic Question matching

- Given a natural language question Q and set of candidate questions CQ , the task is to rank each of the question $Q_{cq} \in CQ$ according to their semantic similarity to the question Q ,
 - *Q1: What are the best ways to lose weight?*
 - *Q2: How can a person reduce weight?*
 - *Q3: What are the effective weight loss plans?*

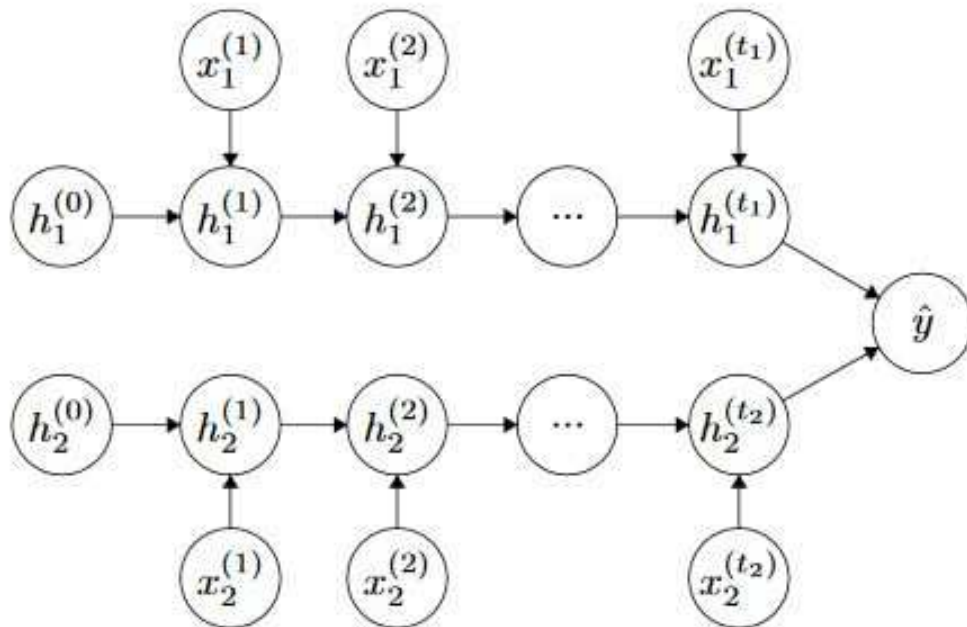
- Answer Triggering

- Given a question and a set of answer candidates, **answer triggering** determines if the candidate set contains any correct answers. If yes, it then outputs a correct one.
 - *Q: How big is BMC software in houston?*
 - *A1: BMC Software Inc. is an American company specializing in Business Service Management (BSM) software.*
 - *A2: Employing over 6,000, BMC is often credited with pioneering the BSM concept as a way to help better align IT operations with business needs.*

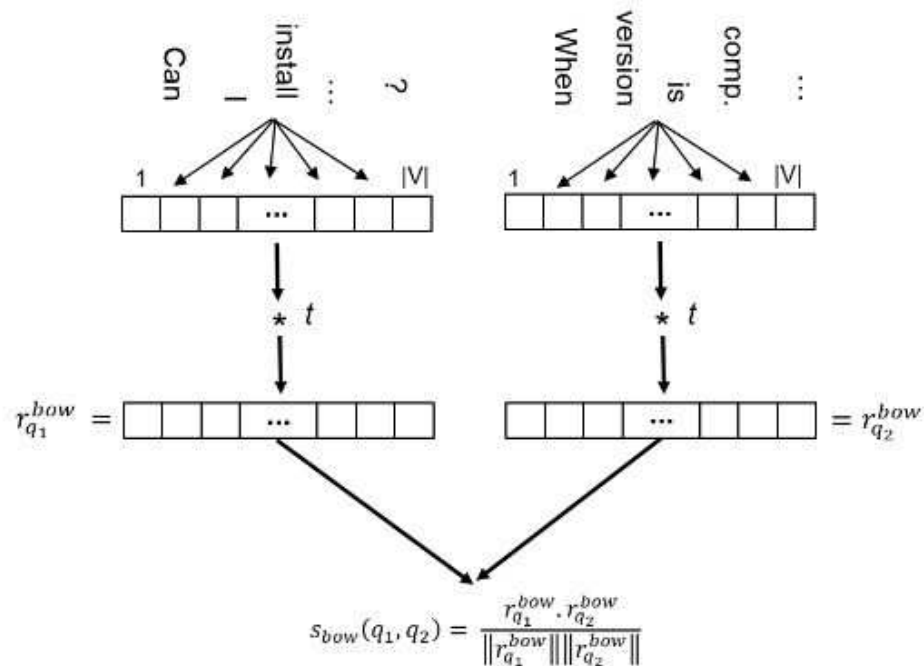
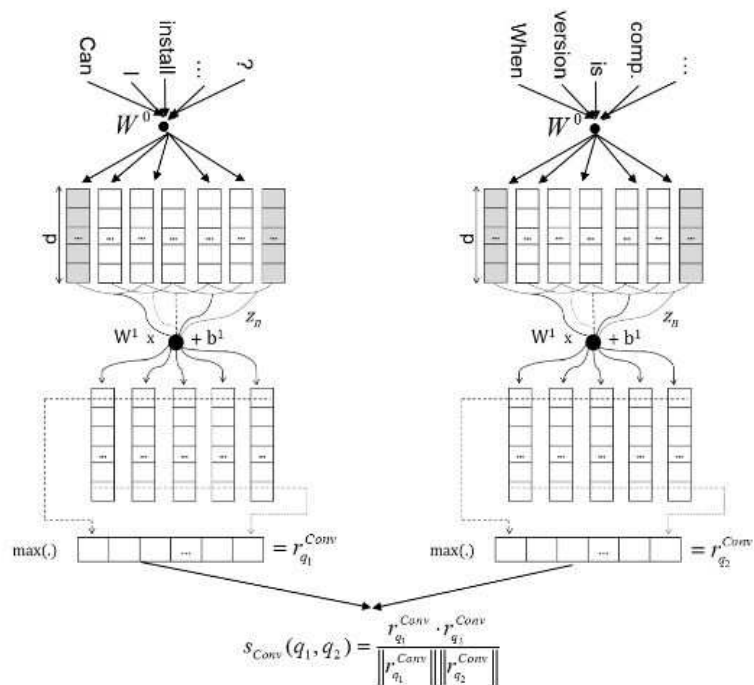
Problem formulation: Semantic Question Matching / Answer Triggering

- Let
 - Question is q
 - Candidate questions are $CQ = \{cq_1, cq_2, \dots, cq_n\}$
 - Candidate answers are $CA = \{ca_1, ca_2, \dots, ca_n\}$
- Question - Question pairing
 - $\langle q, cq_1 \rangle, \langle q, cq_2 \rangle, \dots, \langle q, cq_n \rangle$
- Question - Answer pairing
 - $\langle q, ca_1 \rangle, \langle q, ca_2 \rangle, \dots, \langle q, ca_n \rangle$
- Decision
 - Binary - Yes/No
 - Similarity score - a continuous value in the range 0 to 1.

Detecting Duplicate Questions with Deep Learning - Siamese Network



Learning Hybrid Representations to Retrieve Semantically Equivalent Questions



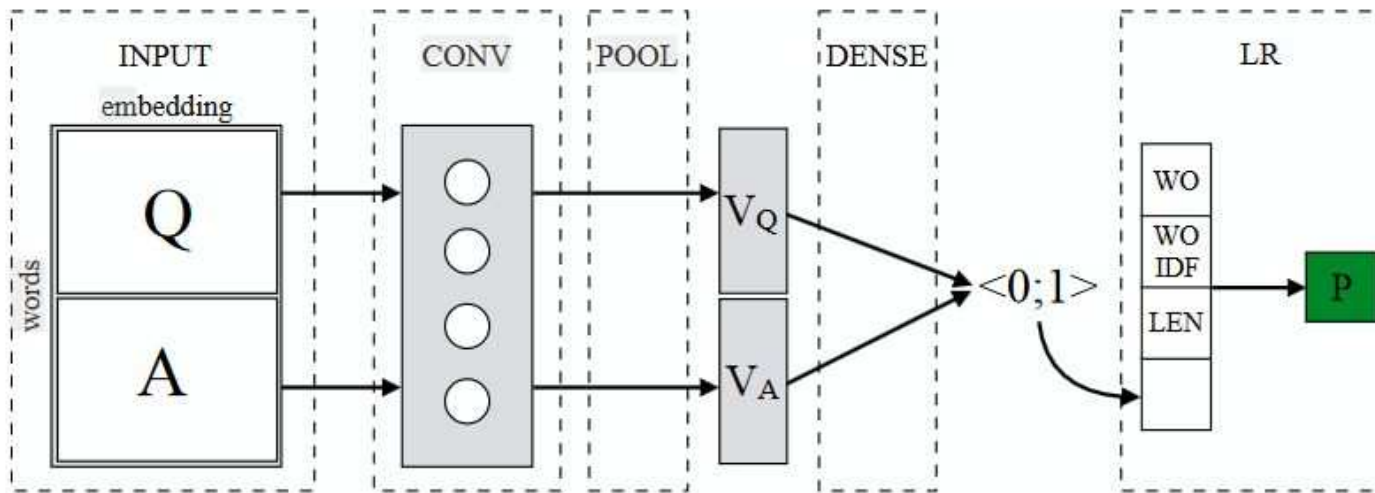
Learning Hybrid Representations to Retrieve Semantically Equivalent Questions

$$s_{bow}(q_1, q_2) = \frac{r_{q_1}^{bow} \cdot r_{q_2}^{bow}}{\|r_{q_1}^{bow}\| \|r_{q_2}^{bow}\|}$$

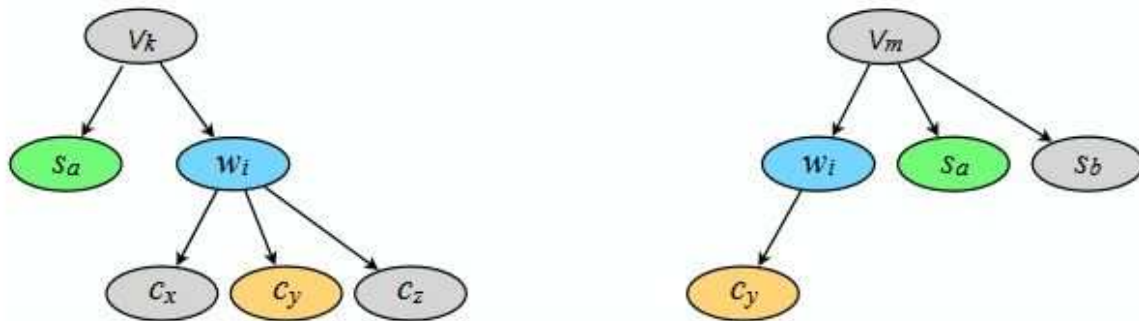
$$s_{conv}(q_1, q_2) = \frac{r_{q_1}^{conv} \cdot r_{q_2}^{conv}}{\|r_{q_1}^{conv}\| \|r_{q_2}^{conv}\|}$$

$$s(q_1, q_2) = \beta_1 * s_{bow}(q_1, q_2) + \beta_2 * s_{conv}(q_1, q_2)$$

SelQA: A New Benchmark for Selection-based Question Answering



SelQA: A New Benchmark for Selection-based Question Answering



- For all $w \in T$, where T is common words in Q & A.
 - P_q & $P_a \rightarrow$ parents of w in D_q & D_a
 - S_q & $S_a \rightarrow$ siblings of w in D_q & D_a
 - C_q & $C_a \rightarrow$ children of w in D_q & D_a
- Three features: $f(P_q, P_a)$, $f(S_q, S_a)$ and $f(C_q, C_a)$

Thank You!

AI-NLP-ML Group, Department of CSE, IIT Patna (<http://www.iitp.ac.in/~ai-nlp-ml/>)

Research Supervisors:

- Prof. Pushpak Bhattacharyya
- Dr. Asif Ekbal
- Dr. Sriparna Saha