CS772: Deep Learning for Natural Language Processing (DL-NLP)

CNN, Application in Sarcasm, Transformer

Pushpak Bhattacharyya

Computer Science and Engineering Department

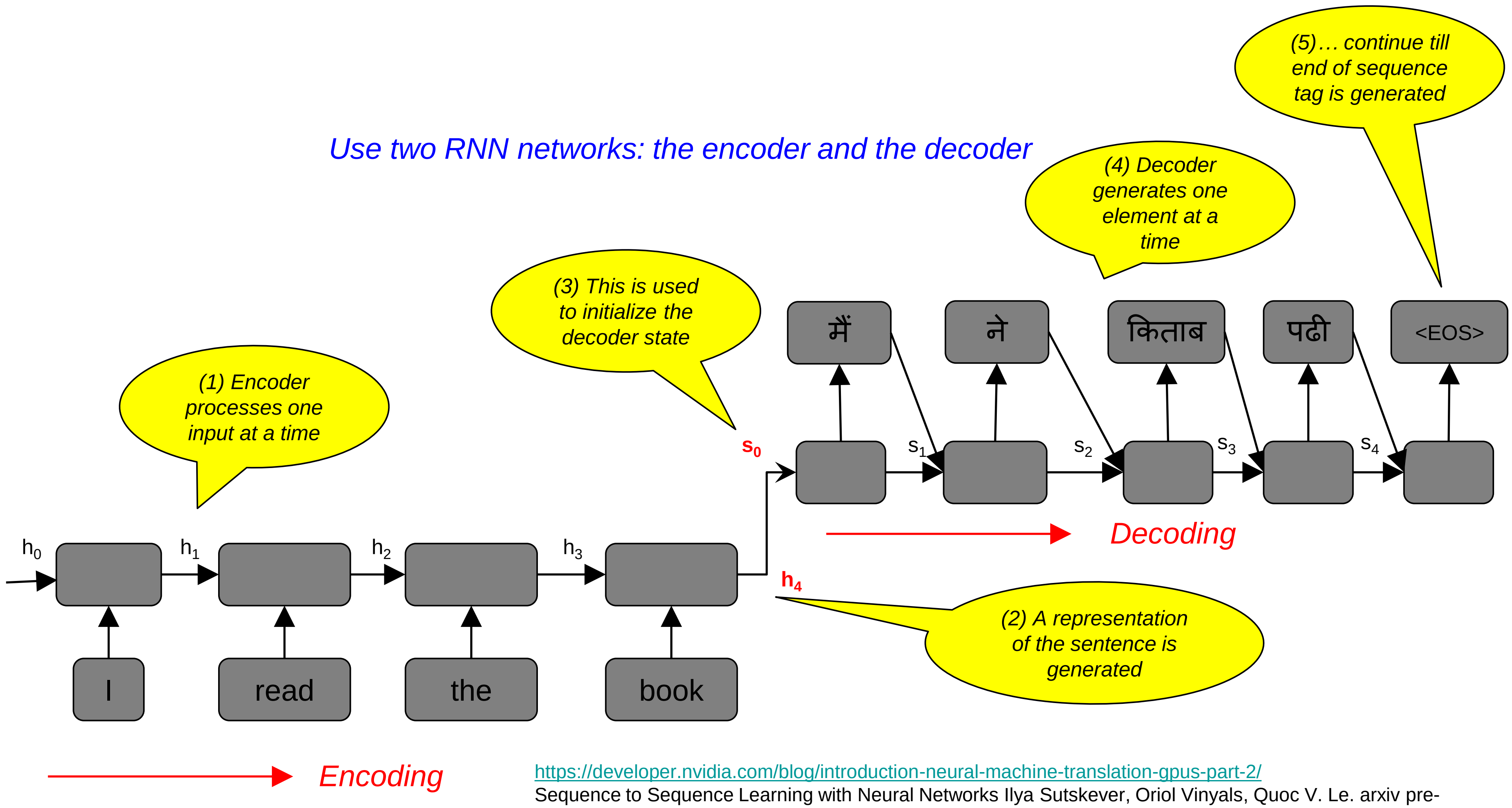
IIT Bombay

Week 8 of 27th Feb, 2023

Re-cap

Encode - Decode Paradigm Explained

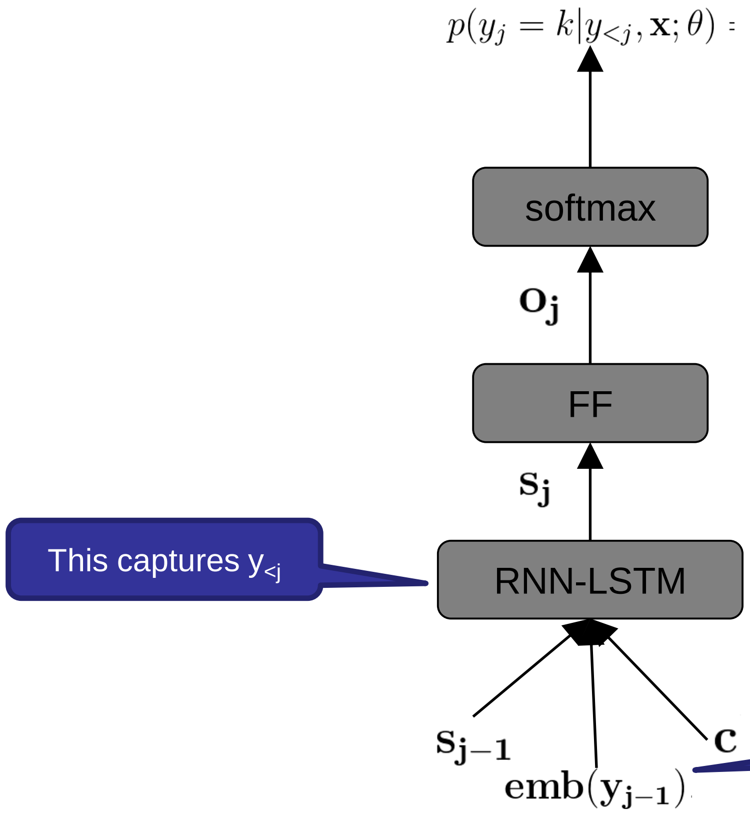
Use two RNN networks: the encoder and the decoder


(5)… continue till end of sequence tag is generated
(4) Decoder generates one element at a time
(3) This is used to initialize the decoder state
(1) Encoder processes one input at a time
(2) A representation of the sentence is generated
मैं
ने
किताब
पढी
<EOS>
s_0
s_1
s_2
s_3
s_4
h_0
h_1
h_2
h_3
h_4
I
read
the
book
Decoding
Encoding

https://developer.nvidia.com/blog/introduction-neural-machine-translation-gpus-part-2/

Sequence to Sequence Learning with Neural Networks Ilya Sutskever, Oriol Vinyals, Quoc V. Le. arxiv pre-print [link]

What is the decoder doing at each time-step?



$$softmax(o_{jk}) = \frac{\exp(o_{jk})}{\sum_{m=0}^{m=T} \exp(o_{jm})}$$

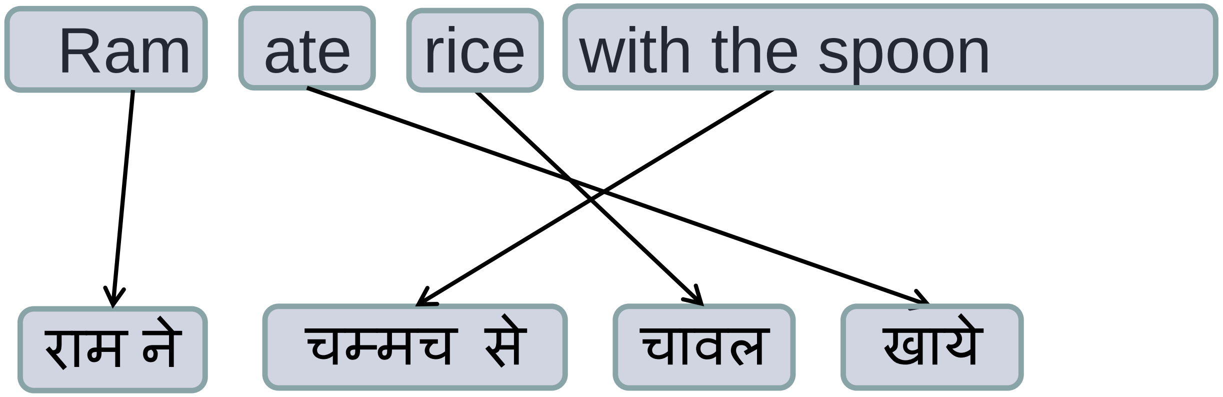
$$\mathbf{o_j} = FF(s_j)$$

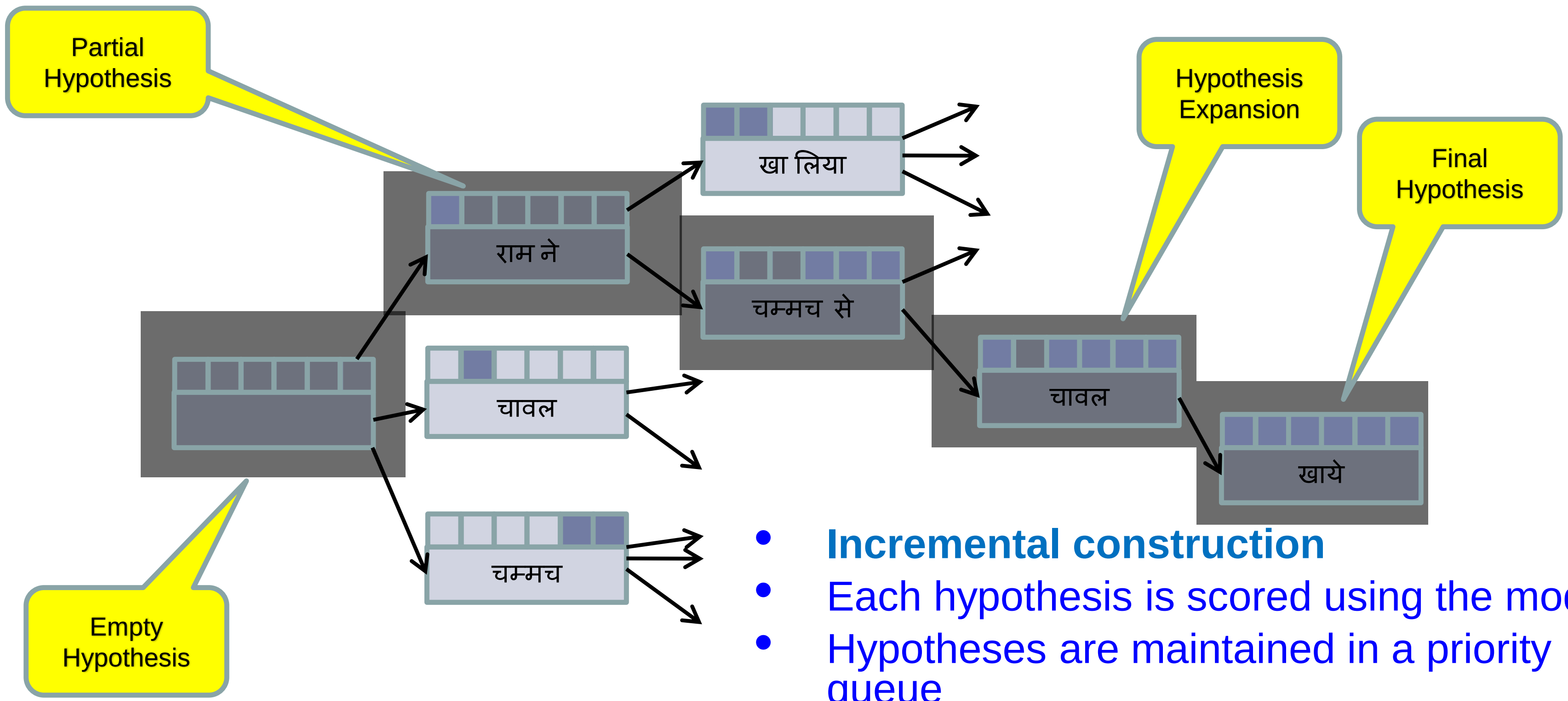
$$\mathbf{s_j} = g(\mathbf{s_{j-1}}, \mathbf{emb(y_{j-1})}, \mathbf{c})$$

This captures x, $c=h_4$

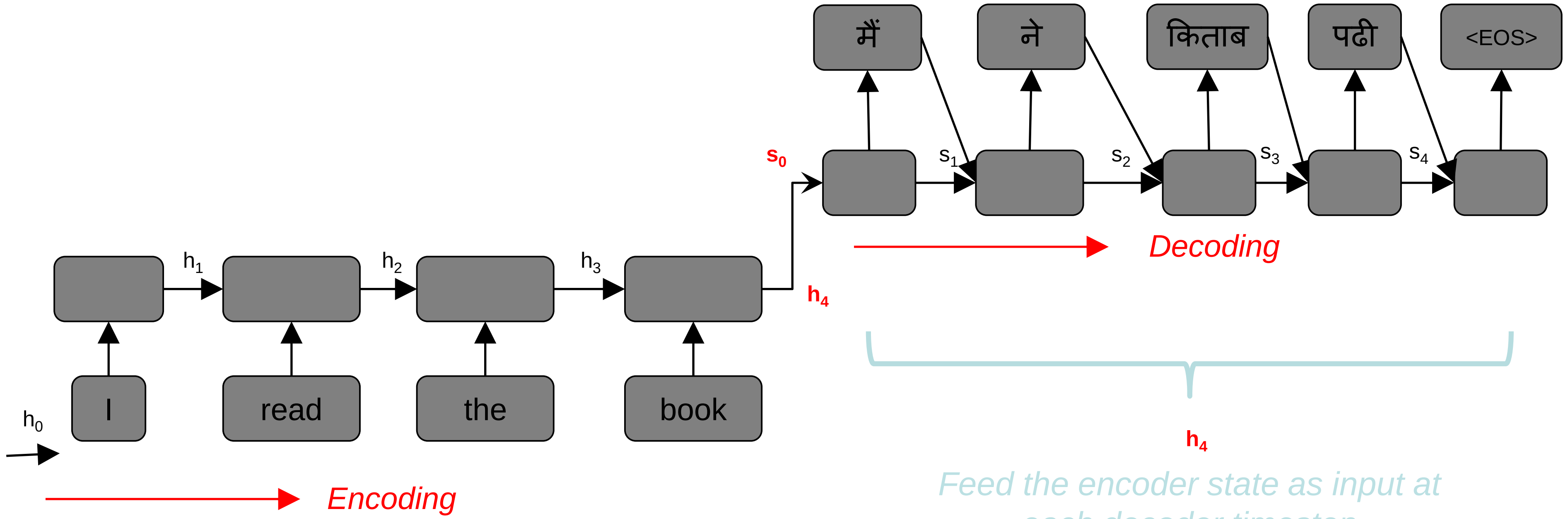
Decoding

Searching for the best translations in the space of all translations





- **Incremental construction**
- Each hypothesis is scored using the model
- Hypotheses are maintained in a priority queue



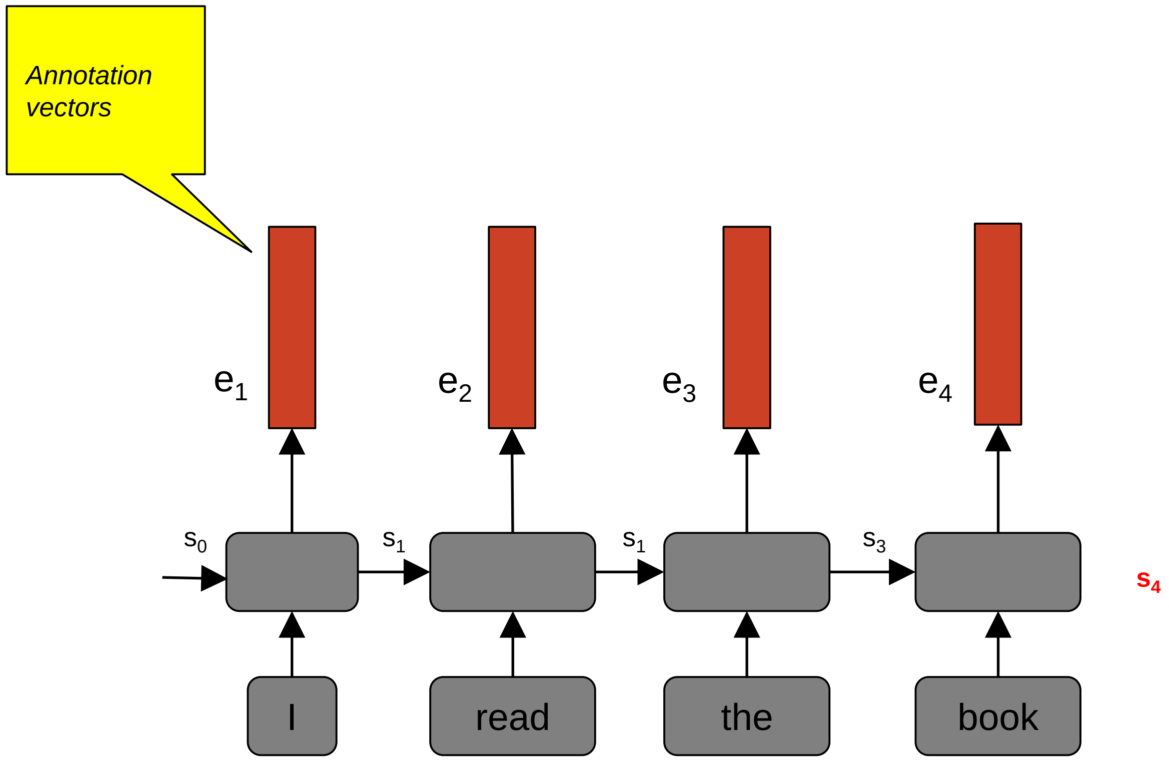
Feed the encoder state as input at each decoder timestep

The entire source sentence is represented by a single vector

Problems

- Insufficient to represent to capture all the syntactic and semantic complexities
  - *Solution: Use a richer representation for the sentences*
- Long-term dependencies: Source sentence representation not useful after few decoder time steps
  - *Solution: Make source sentence information when making the next prediction*
  - *Even better, make **RELEVANT** source sentence information available*

Encode - Attend - Decode Paradigm



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

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

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

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *ICLR 2015.*

https://developer.nvidia.com/blog/introduction-neural-machine-translation-gpus-part-3/

CNN

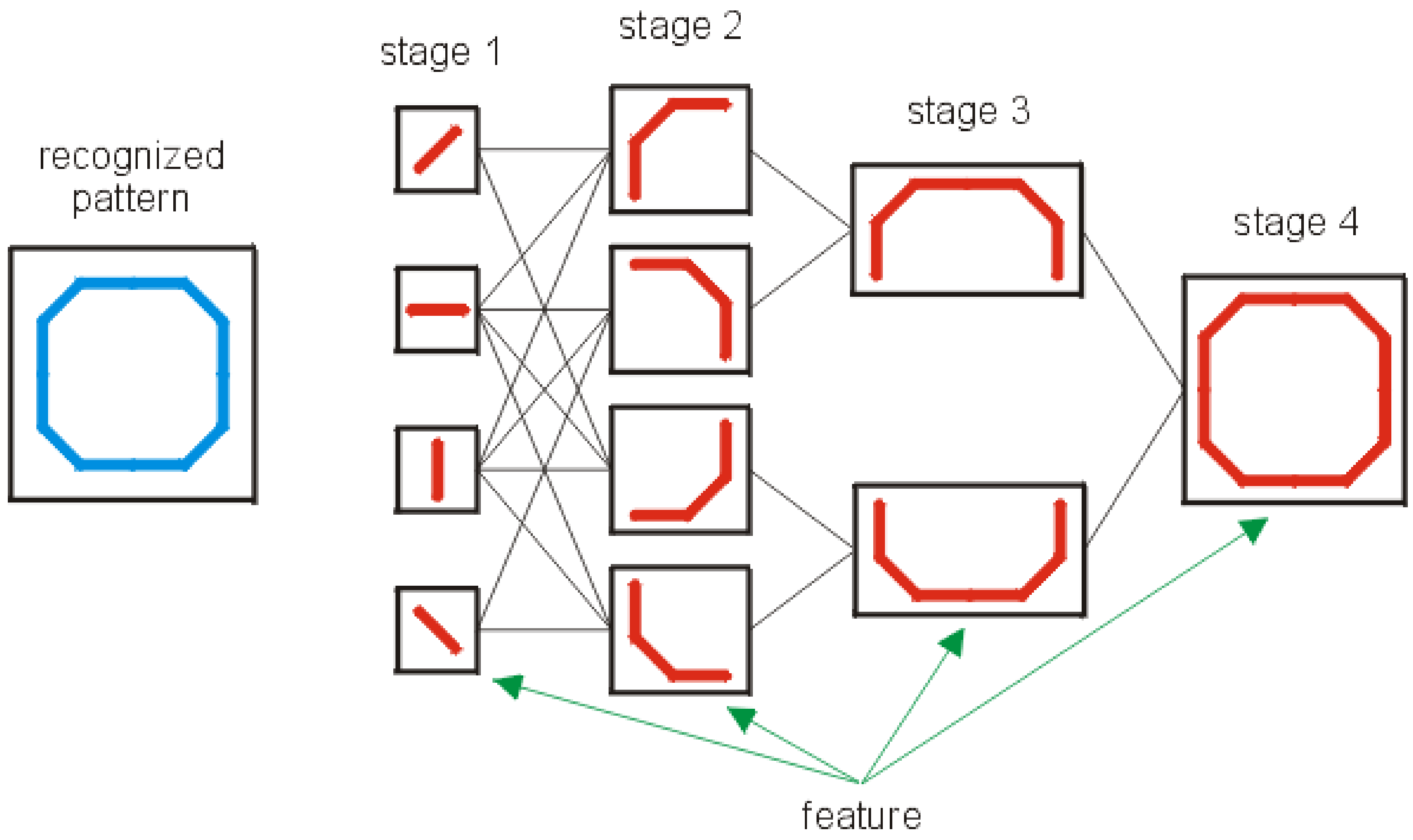
Two motivation points

- 1. Reduced number of parameters
- 2. Stepwise extraction of features
- These two are applicable to any AI situation, and not only vision and image processing

CNN= feedforward like + recurrent like!

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

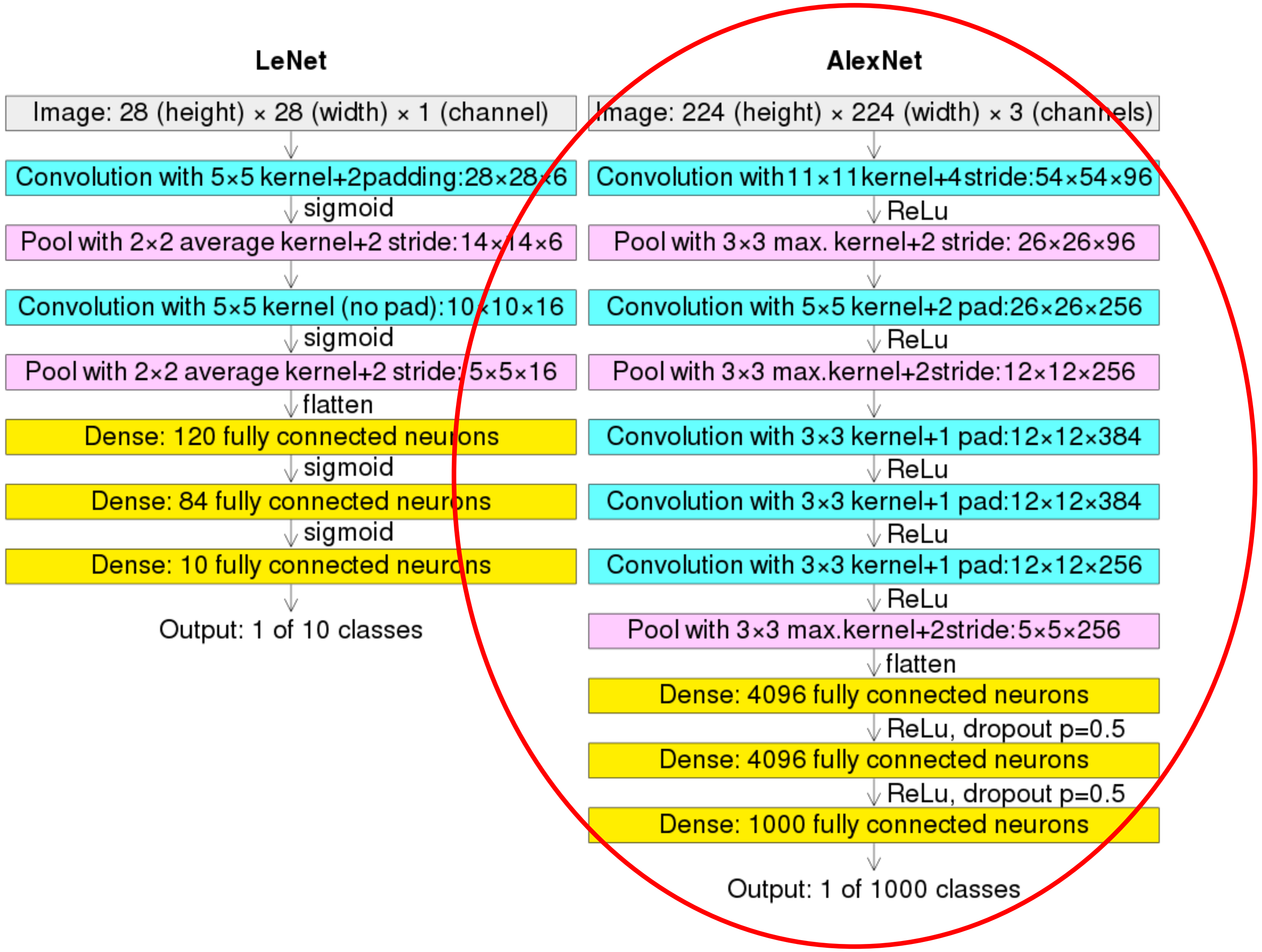
Genesis: Neocognitron (Fukusima, 1980)



Inspiration from biological processes

- Connectivity pattern between neurons resembles the organization of the animal visual cortex
- Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field
- Receptive fields of different neurons partially overlap such that they cover the entire visual field

The classic CNN (Wikipedia)

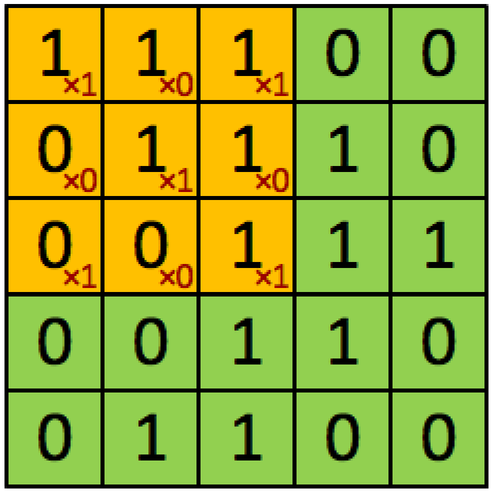

LeNet
Image: 28 (height) × 28 (width) × 1 (channel)
Convolution with 5×5 kernel+2padding:28×28×6
sigmoid
Pool with 2×2 average kernel+2 stride:14×14×6
Convolution with 5×5 kernel (no pad):10×10×16
sigmoid
Pool with 2×2 average kernel+2 stride: 5×5×16
flatten
Dense: 120 fully connected neurons
sigmoid
Dense: 84 fully connected neurons
sigmoid
Dense: 10 fully connected neurons
Output: 1 of 10 classes
AlexNet
Image: 224 (height) × 224 (width) × 3 (channels)
Convolution with 11×11 kernel+4 stride:54×54×96
ReLu
Pool with 3×3 max. kernel+2 stride: 26×26×96
Convolution with 5×5 kernel+2 pad:26×26×256
ReLu
Pool with 3×3 max.kernel+2stride:12×12×256
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×256
ReLu
Pool with 3×3 max.kernel+2stride:5×5×256
flatten
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 1000 fully connected neurons
Output: 1 of 1000 classes

Convolution

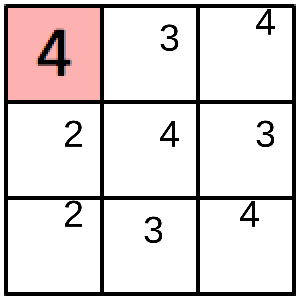
1	**0**	**1**
0	1	0
1	0	1

Filter/kernel/

feature-detector



B/W Image



Convolved Feature

4= 1.1+1.0+1.1
+0.0+1.1+1.0
+0.1+0.0+1.1

Convolution basics

Convolution: continuous and discrete

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t-\tau) d\tau$$

This is the area under the curve $f(\tau)$
weighted by $g(t-\tau)$

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f(m) g(n-m)$$

Convolution of two vectors

V_1: <0, 1, 2, 3, 4, 5, 6, 7, 8, 9>

V_2: <1, 1, 1>

$V_1 \oplus V_2 =$

<(0.1+1.1+2.1), (1.1+2.1+3.1), (2.1+3.1+4.1), (3.1+4.1+5.1), (4.1+5.1+6.1), (5.1+6.1+7.1), (6.1+7.1+8.1), (7.1+8.1+9.1)>

=<3, 6, 9, 12, 15, 18, 21, 24>

Receptive field and selective emphasis/de-emphasis

- The filter <1,1,1> given equal “emphasis” to constituents of the “receptive field” which means region of interest
- Sliding of the filter corresponds to taking different receptive fields
- By designing the filter as <0,1,0>, we emphasise the center of the receptive field

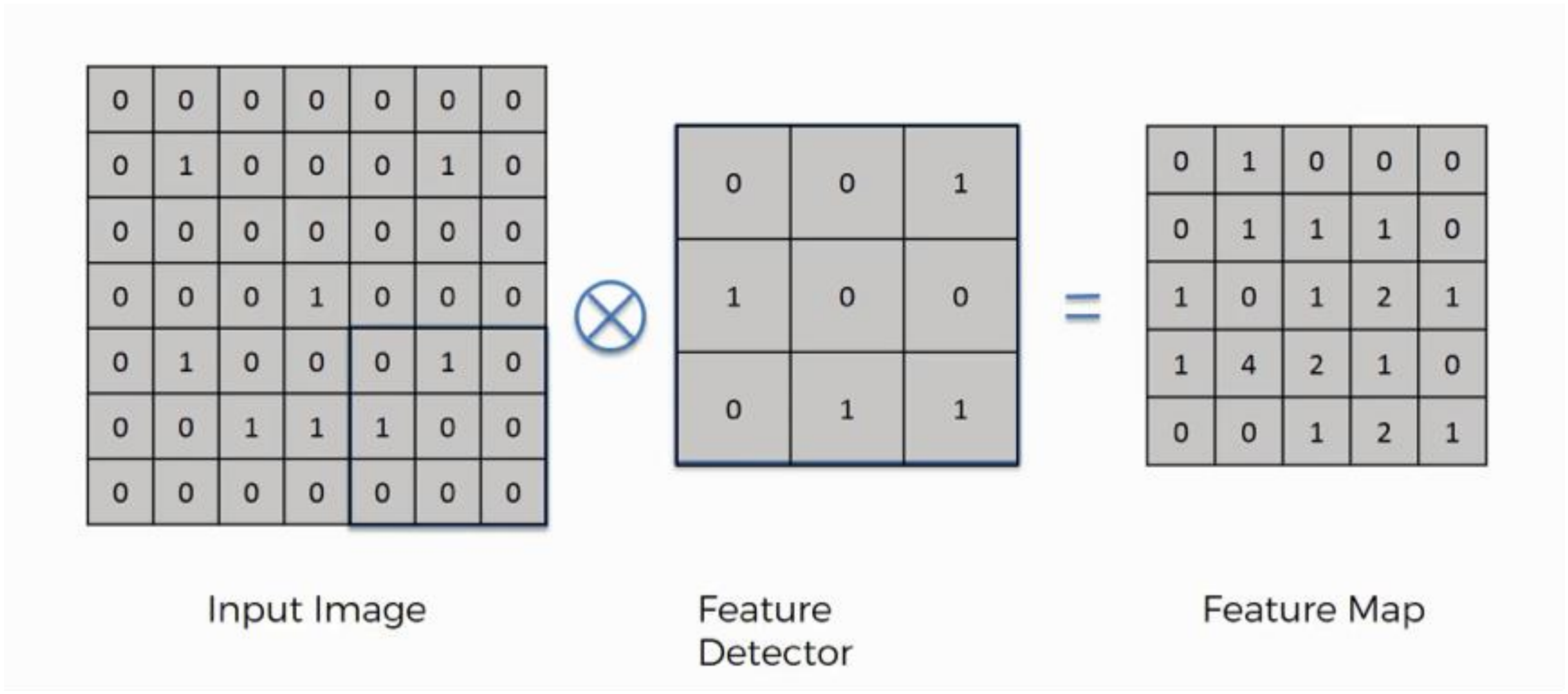
“dog” image and “cat” image

- For dog, the face is of conical shape
- For cat, the shape is round
- So, this distinguishing feature is important for classification
- The filter should have the ability of detecting this kind of feature

Interpretation of convolution

- The filter/kernel/feature_extractor highlights features and obtains those features
- The sliding achieves the effect of focussing on “region” after “region”
- This resembles sequence processing
- The filter components are **LEARNT**

Convolution as feature extractor



Input Image:

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Feature Detector:

0	0	1
1	0	0
0	1	1

Feature Map:

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

CNN architecture

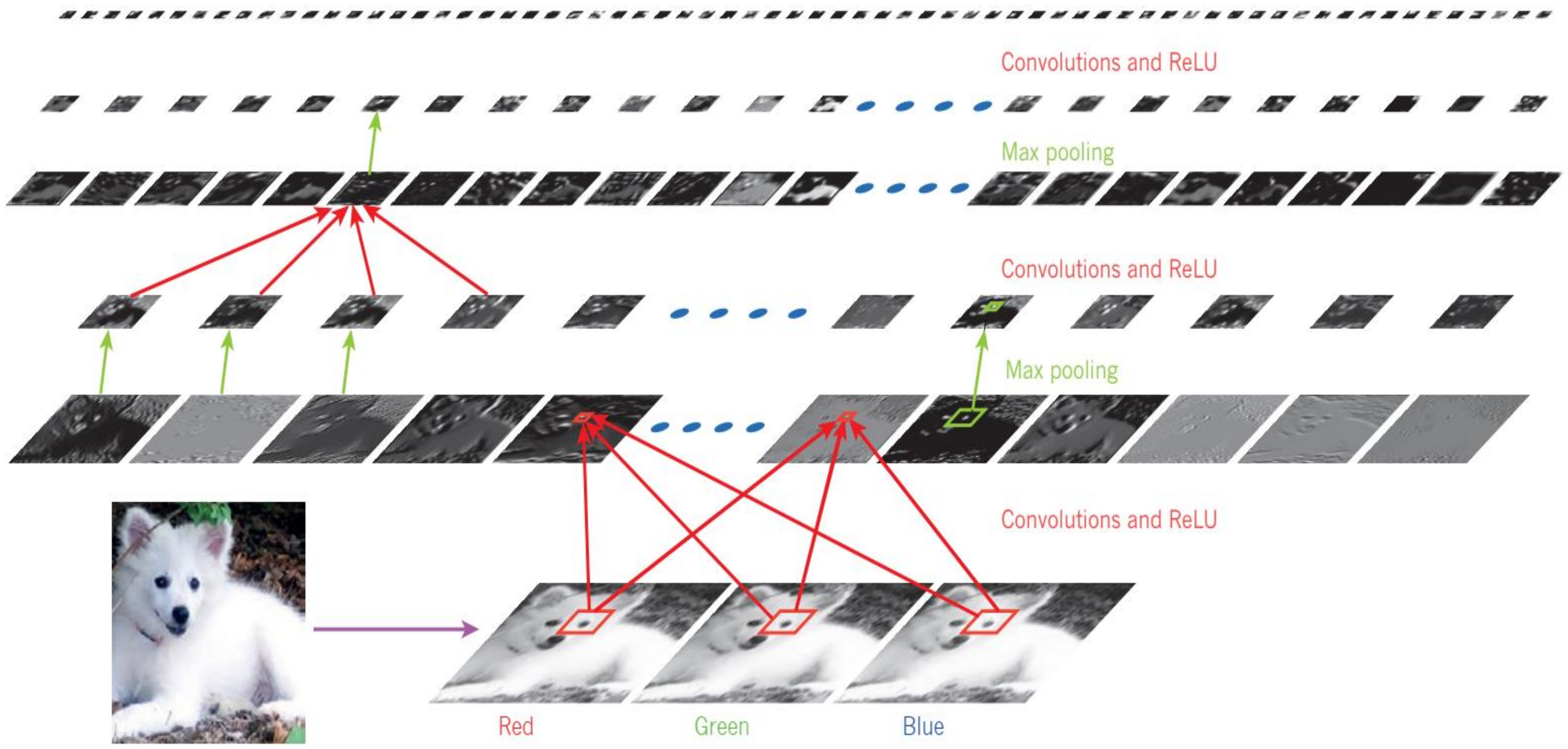
- Several layers of convolution with *tanh* or *ReLU* applied to the results
- In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That’s also called a fully connected layer, or affine layer.
- In CNNs we use convolutions over the input layer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

Key Ideas

Four key ideas that take advantage of the properties of natural signals:

- local connections,
- shared weights,
- pooling and
- the use of many layers

A typical ConvNet



Lecun, Bengio, Hinton, Nature, 2015

Why CNN became a rage: image

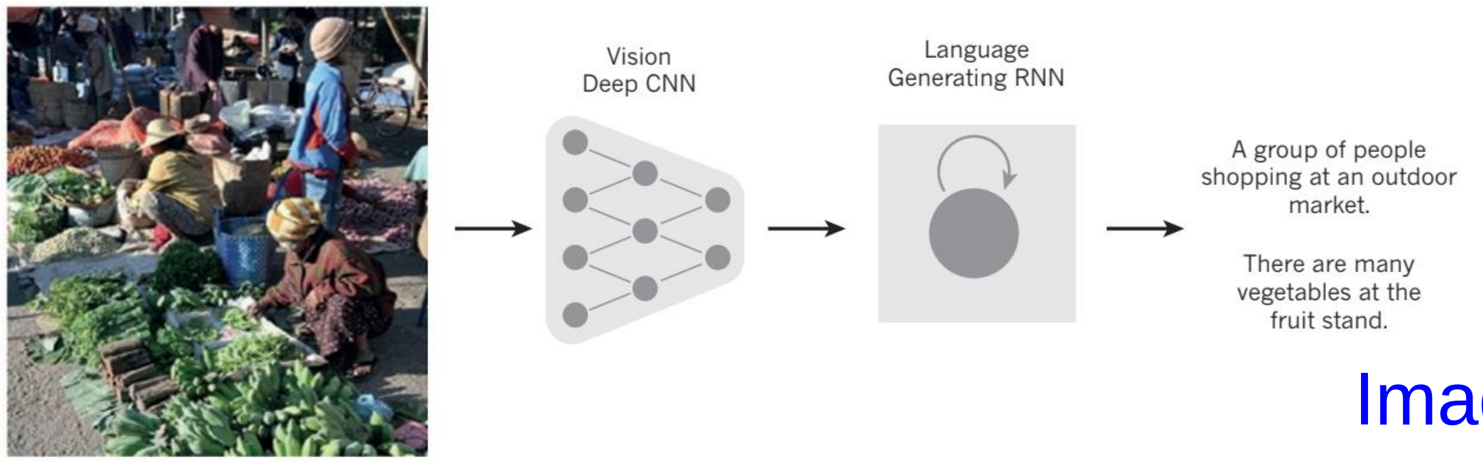
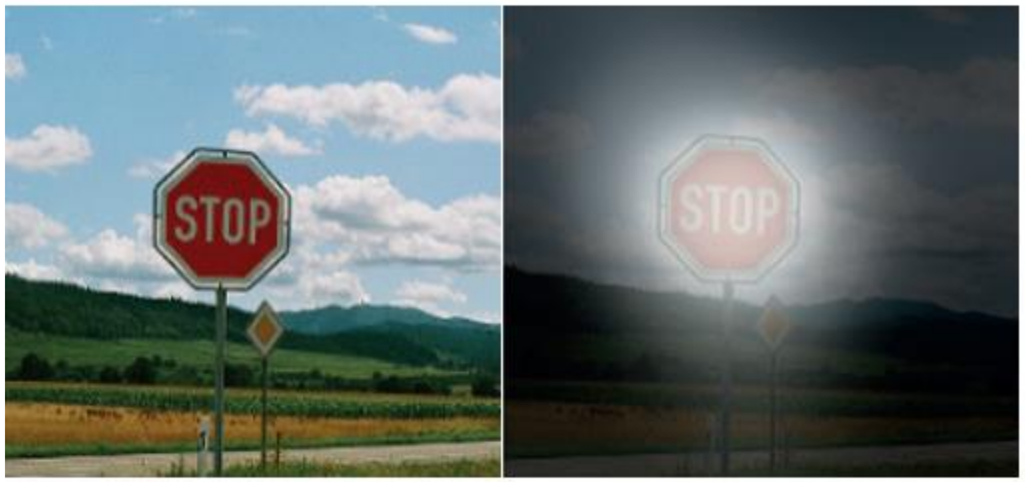

Vision
Deep CNN
Language
Generating RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Image Captioning-1



A **stop** sign is on a road with a mountain in the background

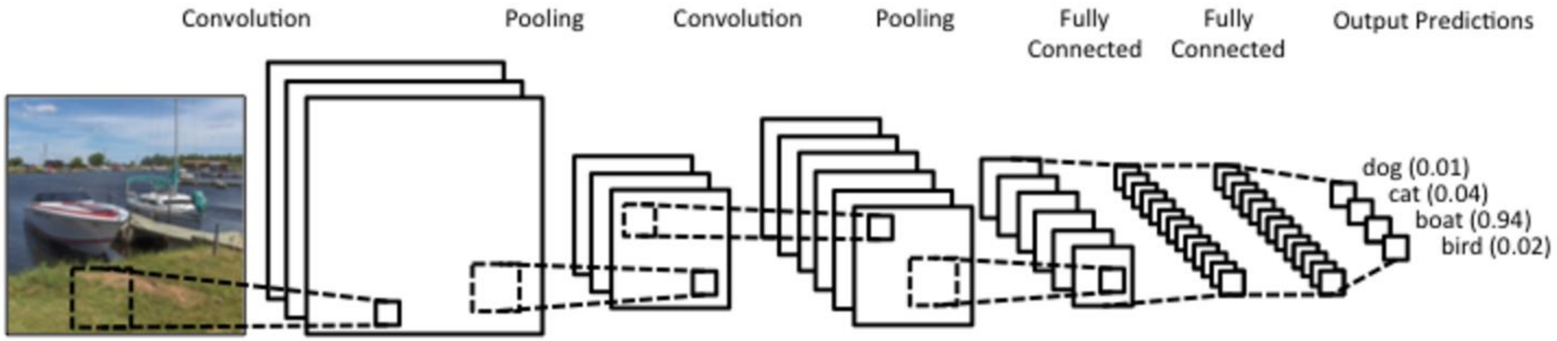
Image Captioning-2

Role of ImageNet

- Million images from the web
- 1,000 different classes
- Spectacular results!
- Almost halving the error rates of the best competing approaches1.

Learning in CNN

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



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

Pooling

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

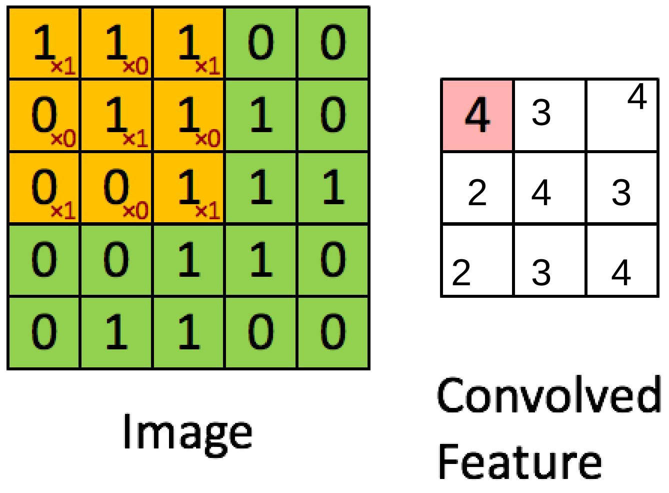
CNN for NLP

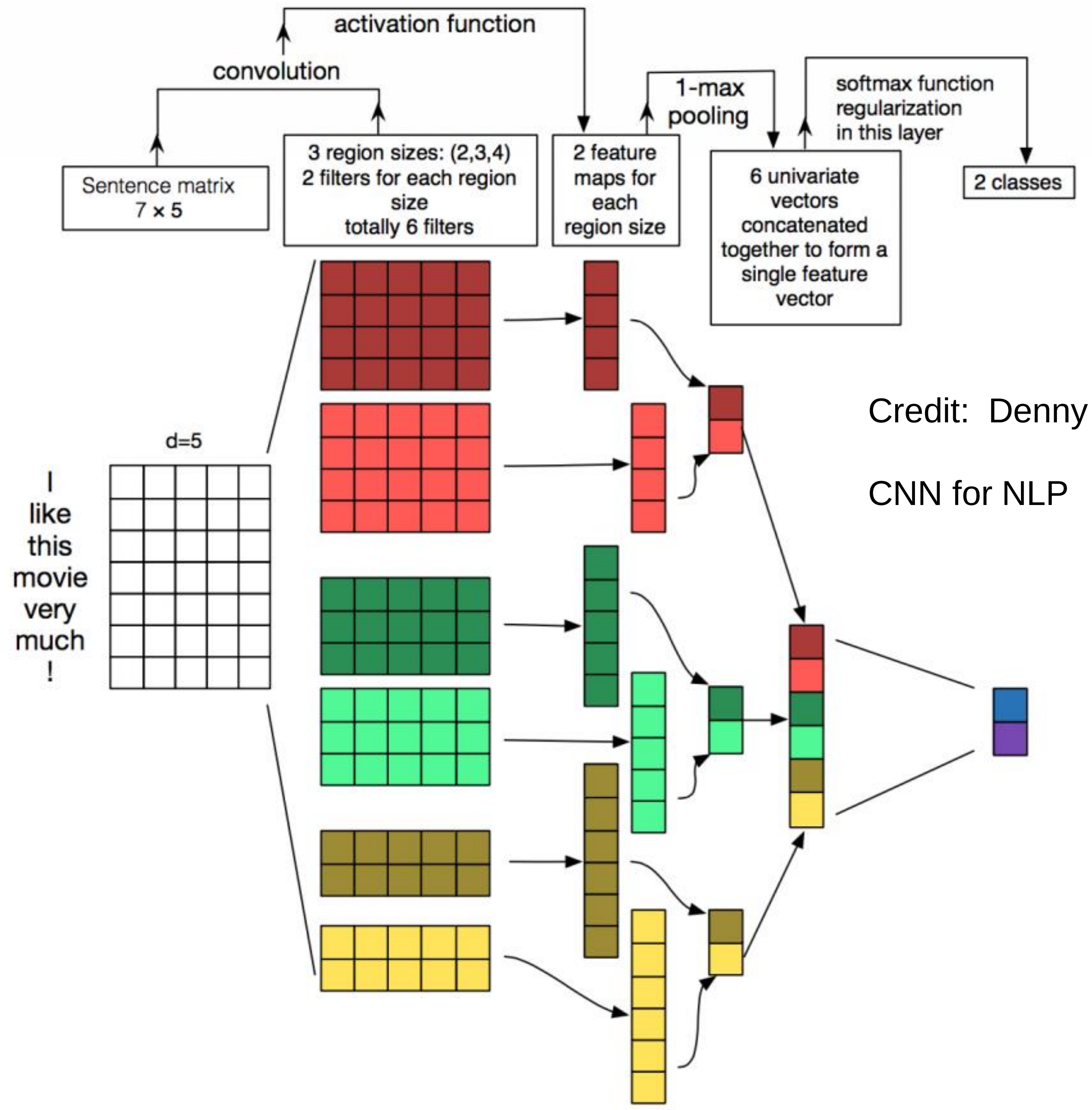
Input matrix for CNN: NLP

- “image” for NLP ←→ word vectors in the rows

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

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




activation function
convolution
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

Credit: Denny Britz

CNN for NLP

Role of multiple filters in CNN

- In the last slide- 2 filters per n-gram (n=2, 3, 4)
- In multitask learning setting, for tasks such as sentiment analysis and emotion analysis multiple filters can be used.
- Multiple filters allow multiple views and emphasis angles for each task. For instance one filter for sentiment analysis and another for emotion analysis.
- The number of filters should be equal to the number of tasks.

Role of lower order ngrams

- Lower order ngrams play an important role in vocabulary matching.
- Lower order ngrams give importance to lexical properties. For instance:
    - Unigram: I like this movie.
    - Bigram: I do not like this movie.

Role of higher order ngrams

- Higher order ngrams give emphasis to syntactic structure of the sentence and the dependencies.
- For instance:
    - Trigram: *I like this movie* (like ←→ movie)
    - Quadrigram and pentagram capture more dependencies and syntactic structure and play an important role in tasks like sentiment analysis, emotion detection, machine translation, etc.
    - Example: *John watched a movie with James yesterday in Melbourne* (Who did what to whom when and where type dependency)

CNN Hyper parameters

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

Detailing out CNN layers

Credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

CNN stages

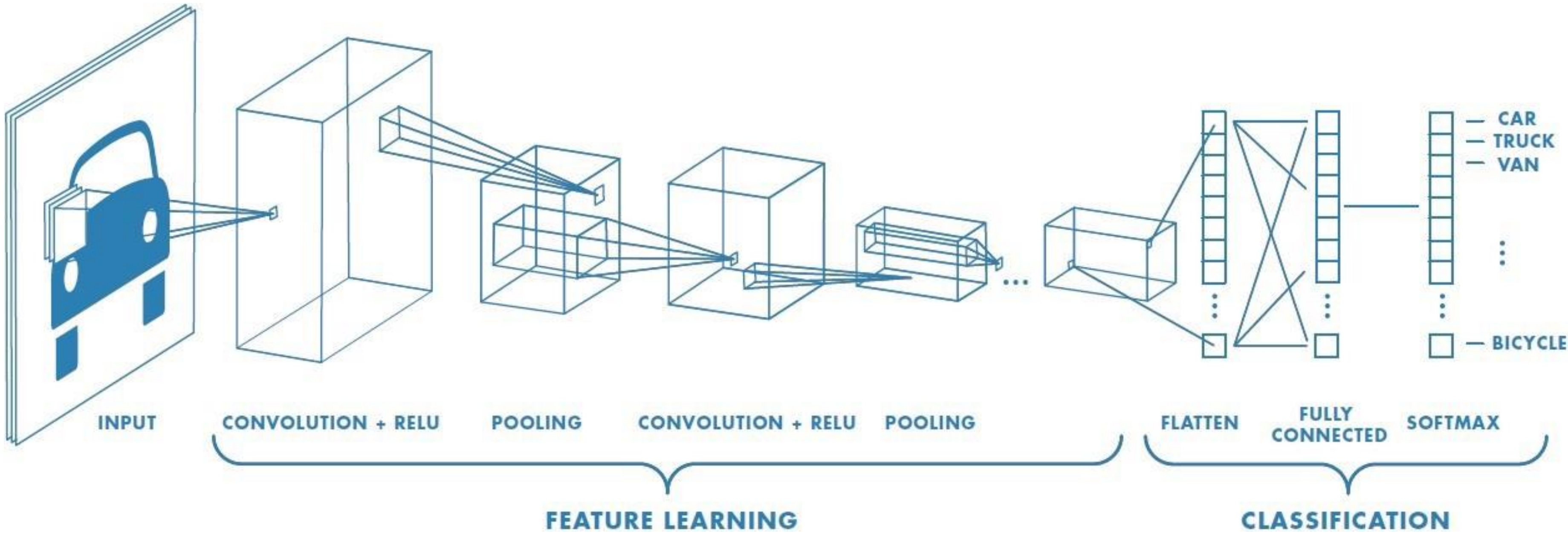


Image Credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Another depiction

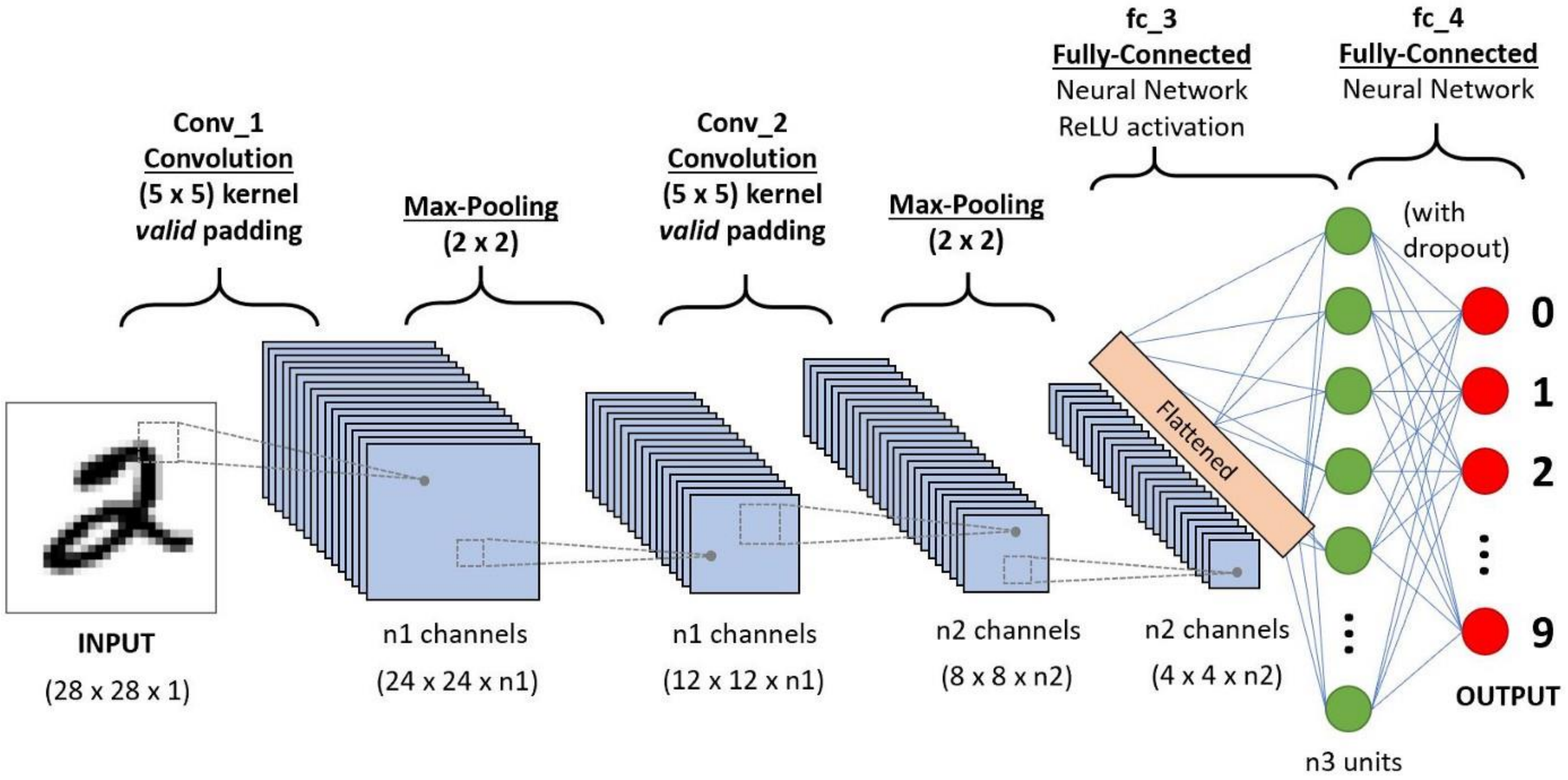
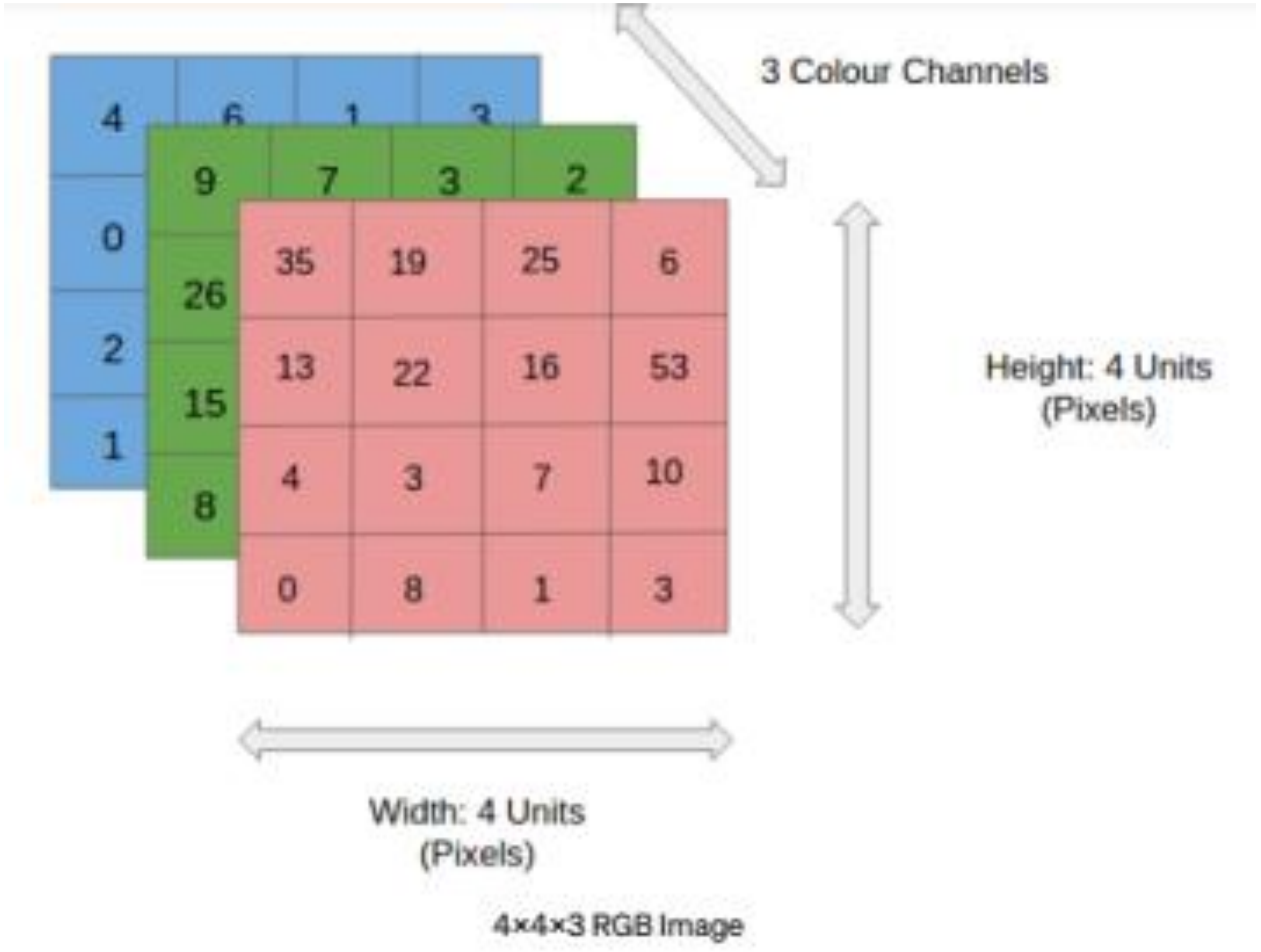
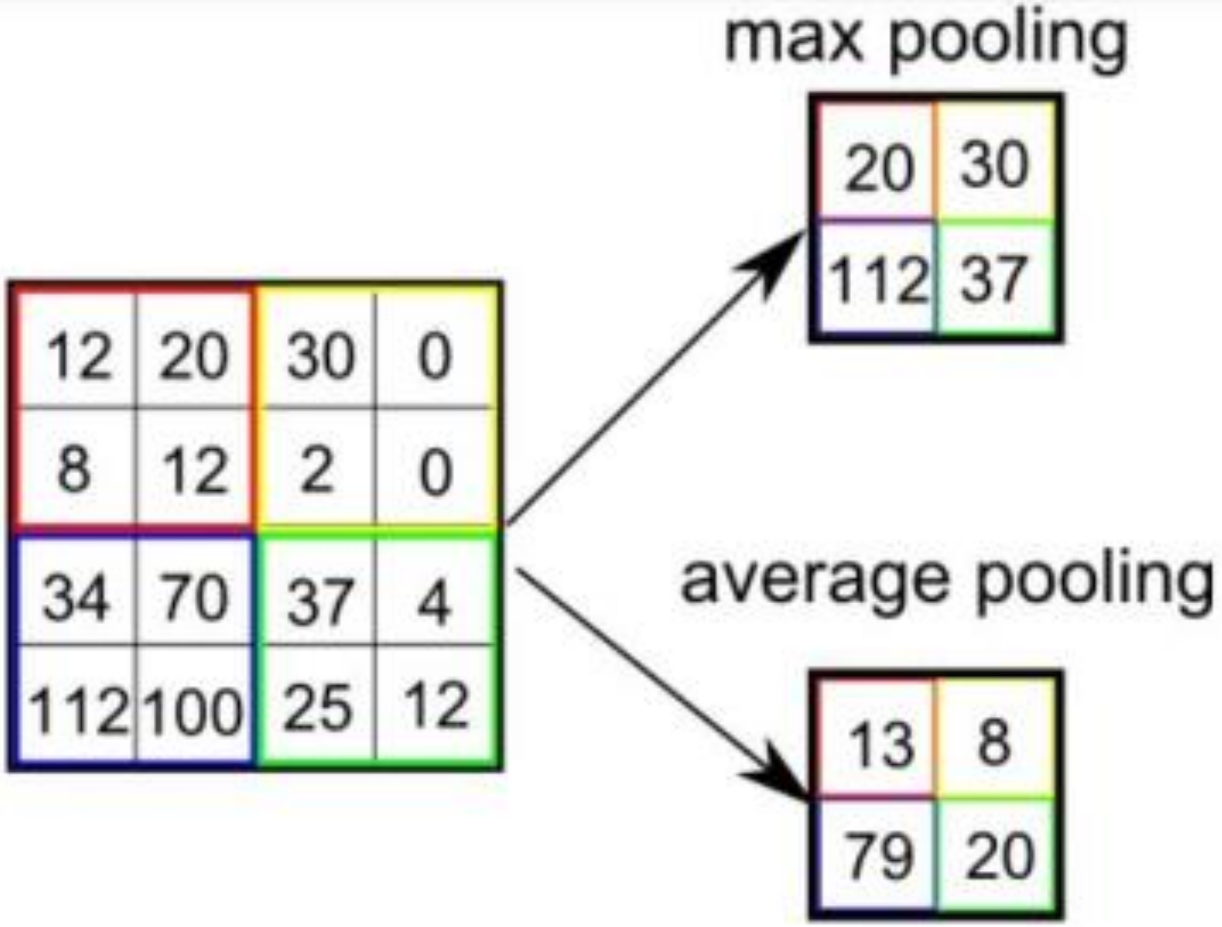


Image Credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Channelized Image

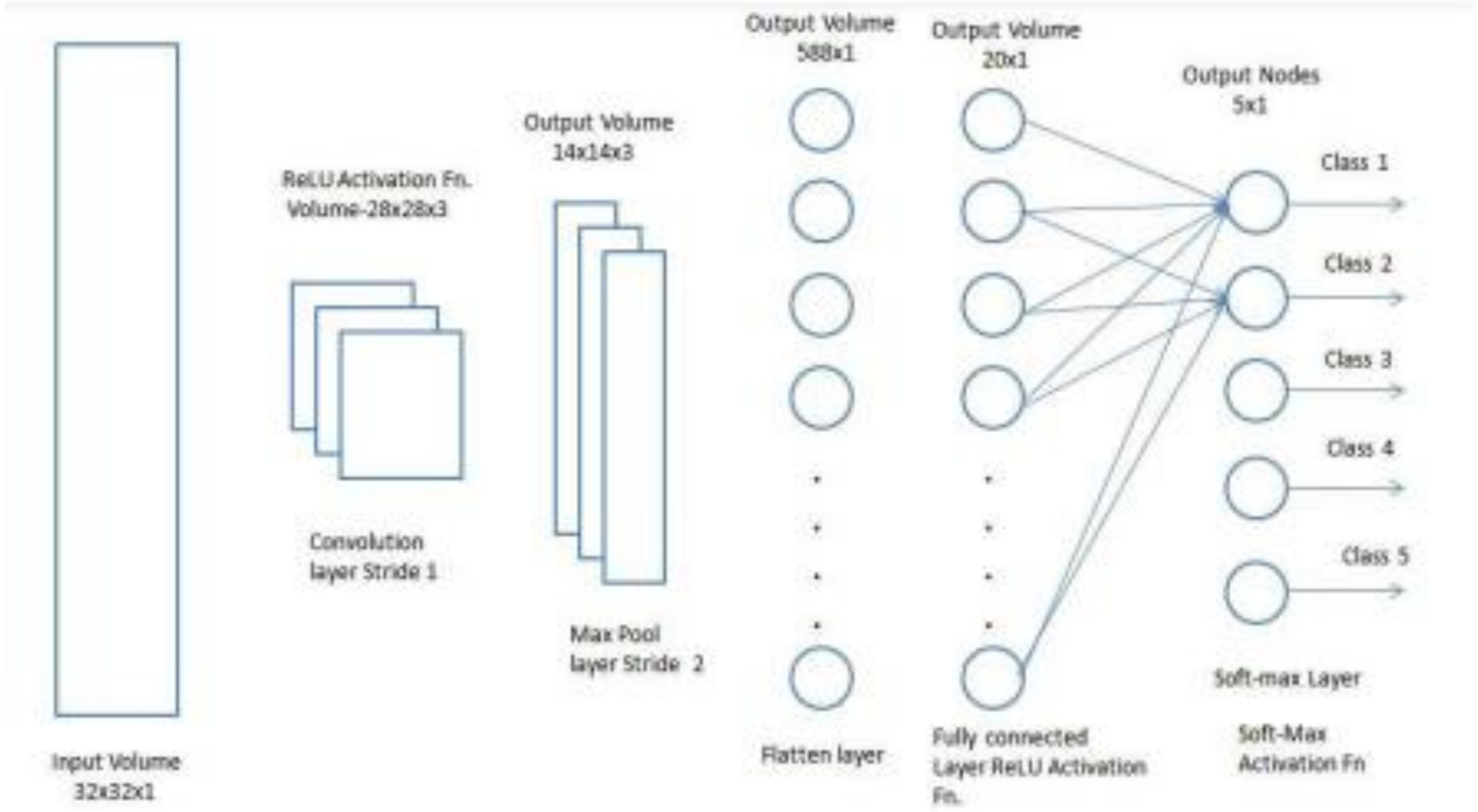


Pooling



Types of Pooling

Complete Architecture



Convolution Layer

- Input is a tensor with a shape
  - (number of inputs) x (input height) x (input width) x (input channels)
- After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape
  - (number of inputs) x (feature map height) x (feature map width) x (feature map channels).

Tensors and Vectors

- **Tensors**: vectors of vectors
- Vector, *V*: <1, 2, 3, 4, 5>
- Tensor, T1: <<1, 2, 3>, <4, 5, 6>>
- Tensor, T2: <<<1,2>, <3>>, <<4>, <5,6>>>
- **Channels**: R, G, B
- Each image consists of Red, Green and Blue channels- that is, 3 different matrices of pixel values

Pooling Layer

- “Pooling” involves sliding a two-dimensional filter over each channel of feature map
- Effect: summarizing the features
- For a feature map having dimensions $\mathbf{n_h \times n_w \times n_c}$, the output dimension after pooling is

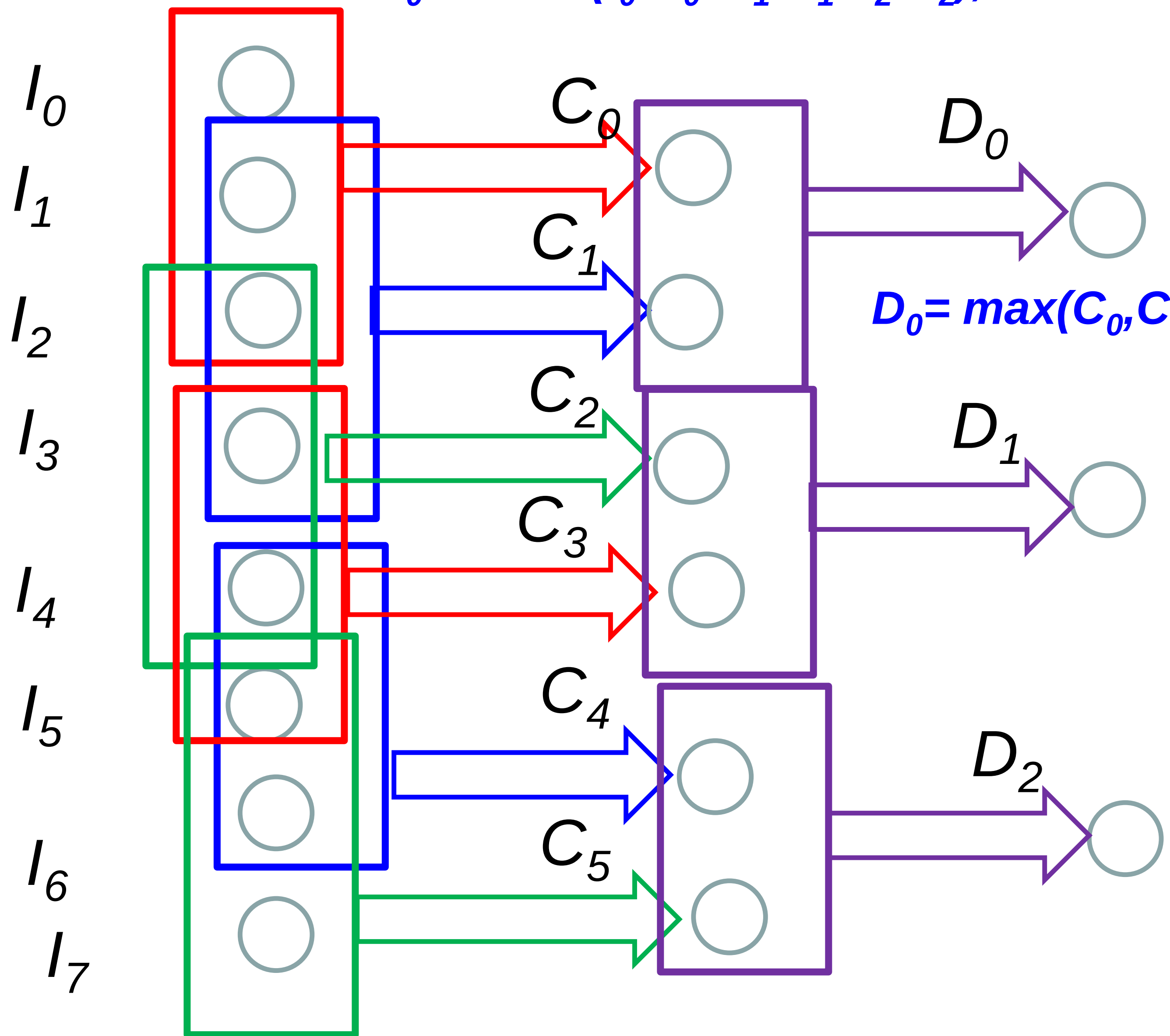
$$\left(\frac{n_h - f_h + 1}{s}\right) \cdot \left(\frac{n_w - f_w + 1}{s}\right) (.n_c)$$

where, n_h= height of feature map, n_w=width, n_c= number of channels, f_h=height of filter, f_w=width of filter, s=stride length

Learning in CNN

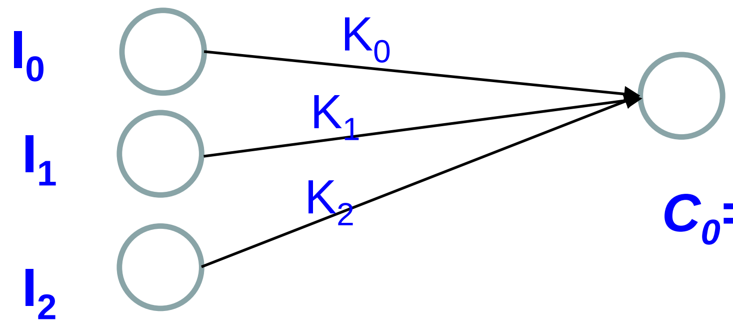
First Kernel+RELU+POOLING

C_0= RELU($I_0.K_0$+ $I_1.K_1$+$I_2.K_2$); Ks are kernel “weights”



D_0= max(C_0,C_1)

Fleshing out the details



$C_0 = RELU(I_0.K_0 + I_1.K_1 + I_2.K_2)$

Input vector I

New K_0= old K_0+sum of ΔK_0s across C_0, C_1..C_5

This addition does not violate gradient descent rule

Normal BP works

- Backpropagate from the final layer of softmax.
- When it comes to the first convolution layer, post the changes in the weights, maintaining the constraint that kernel values are parameter-shared
- Nothing special needs to be done for RELU and MAX functions

Another depiction

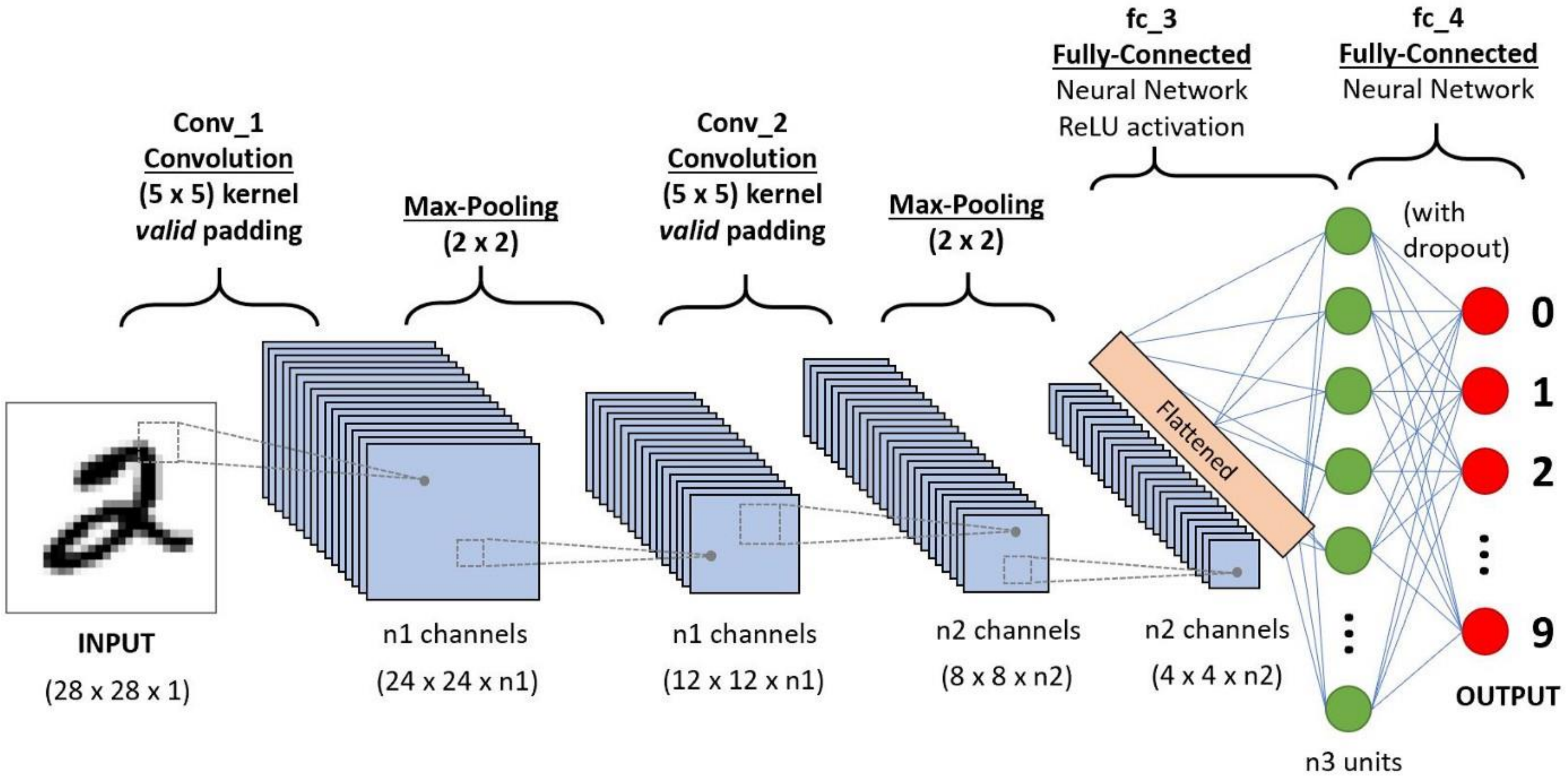


Image Credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

An application: Sarcasm Detection

Illustrates use of CNN Channels

Sarcasm Detection: a sub-problem of Sentiment and Emotion Analysis

Sentiment Analysis: The task of identifying if a certain piece of text contains any opinion, emotion or other forms of affective content.

Machine Learning based approach: classifiers and features

- SVM, KNN and Random Forest classifiers
- Sentiment-based features
  - Number of
    - positive words
    - negative words
    - highly emotional positive words,
    - highly emotional negative words.
- Positive/Negative word is said to be highly emotional if it’s POS tag is one amongst : ’JJ', ‘JJR', ‘JJS', ‘RB', ‘RBR', ‘RBS', ‘VB', ‘VBD', ‘VBG', ‘VBN', ‘VBP', ‘VBZ'.

Emotion Features

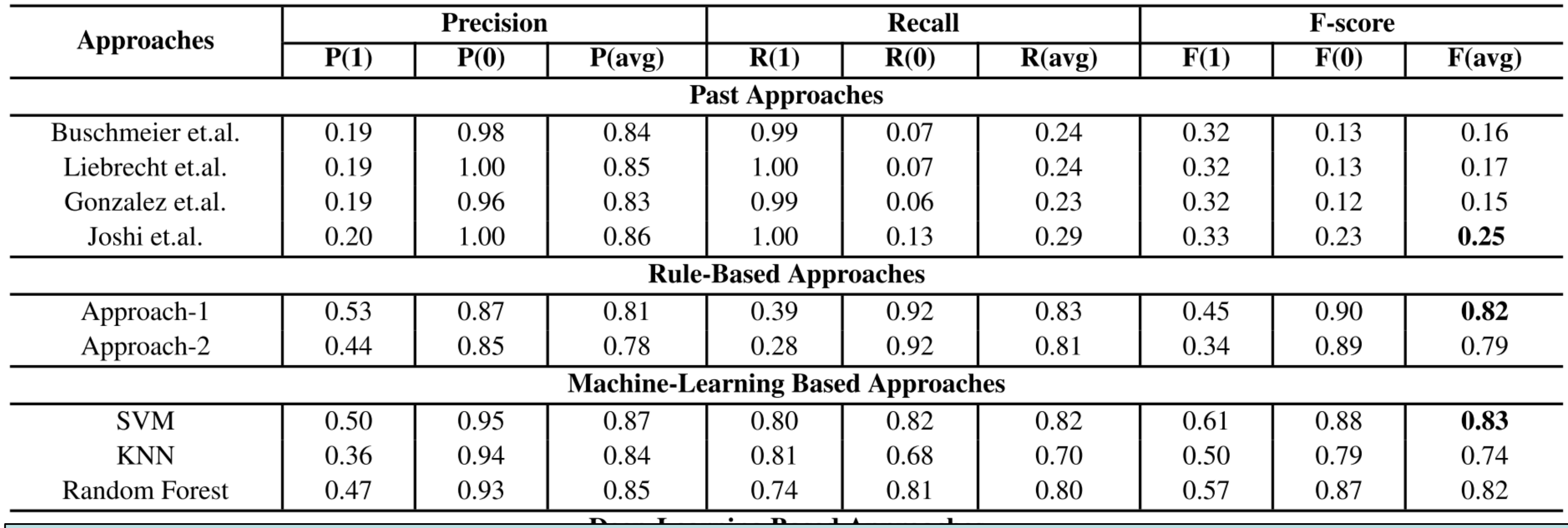
- Positive emoticon
- Negative emoticon

- Boolean features that are 1 if both positive and negative words are present in the tweet.
- Boolean features that are 1 when positive (negative) word and negative (positive) emoji are simultaneously present

Punctuation features

- number of exclamation marks.
- number of dots
- number of question mark.
- number of capital letter words.
- number of single quotations.
- Number in the tweet: This feature is simply the number present in the tweet.
- Number unit in the tweet : This feature is a one hot representation of the type of unit present in the tweet. Example of number unit can be hour, minute, etc.

Comparison of results (1: sarcastic, 0: non-sarcastic)



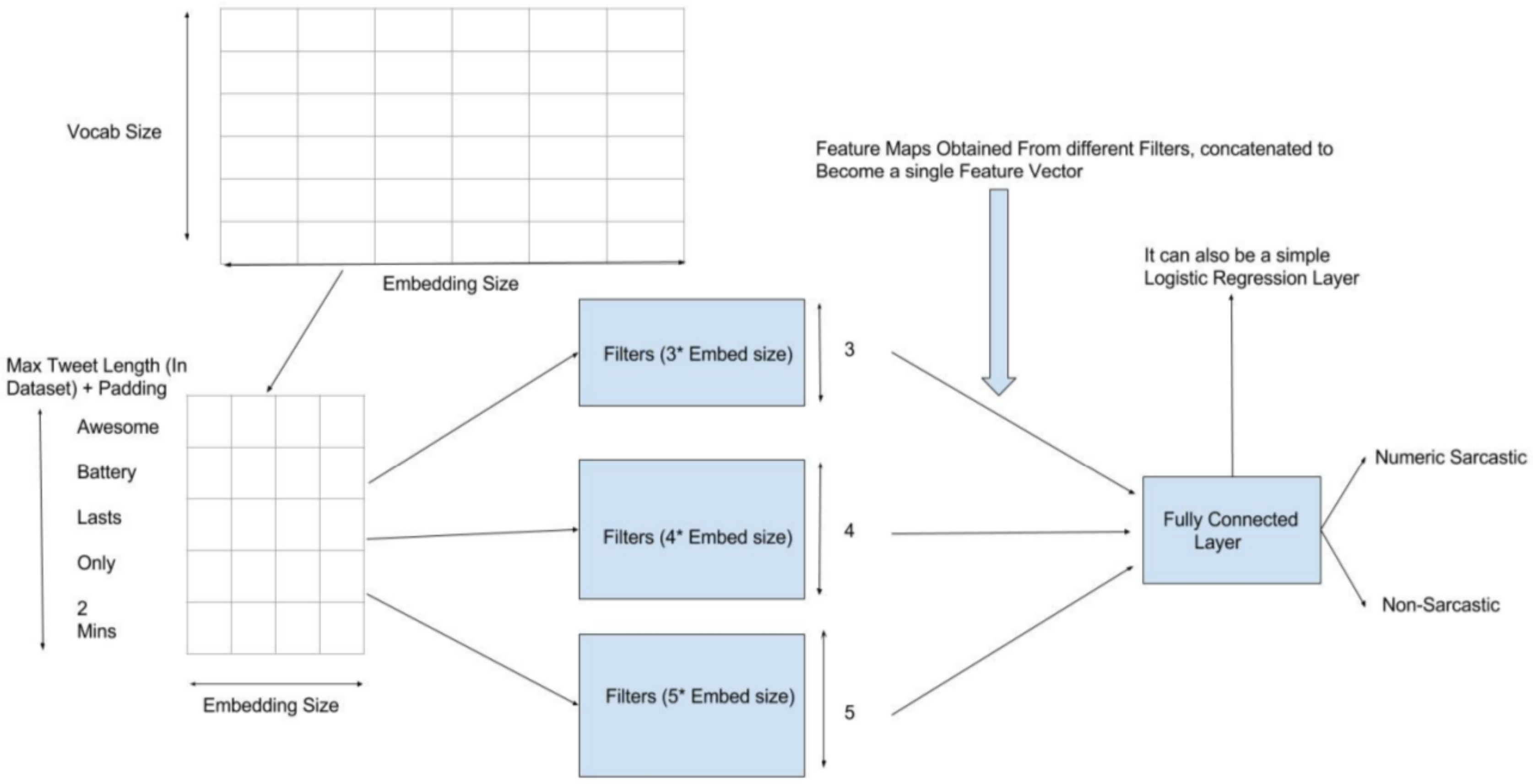
Approaches	Precision			Recall			F-score		
	P(1)	**P(0)**	**P(avg)**	**R(1)**	**R(0)**	**R(avg)**	**F(1)**	**F(0)**	**F(avg)**
Past Approaches									
Buschmeier et.al.	0.19	0.98	0.84	0.99	0.07	0.24	0.32	0.13	0.16
Liebrecht et.al.	0.19	1.00	0.85	1.00	0.07	0.24	0.32	0.13	0.17
Gonzalez et.al.	0.19	0.96	0.83	0.99	0.06	0.23	0.32	0.12	0.15
Joshi et.al.	0.20	1.00	0.86	1.00	0.13	0.29	0.33	0.23	**0.25**
Rule-Based Approaches									
Approach-1	0.53	0.87	0.81	0.39	0.92	0.83	0.45	0.90	**0.82**
Approach-2	0.44	0.85	0.78	0.28	0.92	0.81	0.34	0.89	0.79
Machine-Learning Based Approaches									
SVM	0.50	0.95	0.87	0.80	0.82	0.82	0.61	0.88	**0.83**
KNN	0.36	0.94	0.84	0.81	0.68	0.70	0.50	0.79	0.74
Random Forest	0.47	0.93	0.85	0.74	0.81	0.80	0.57	0.87	0.82

Deep Learning based

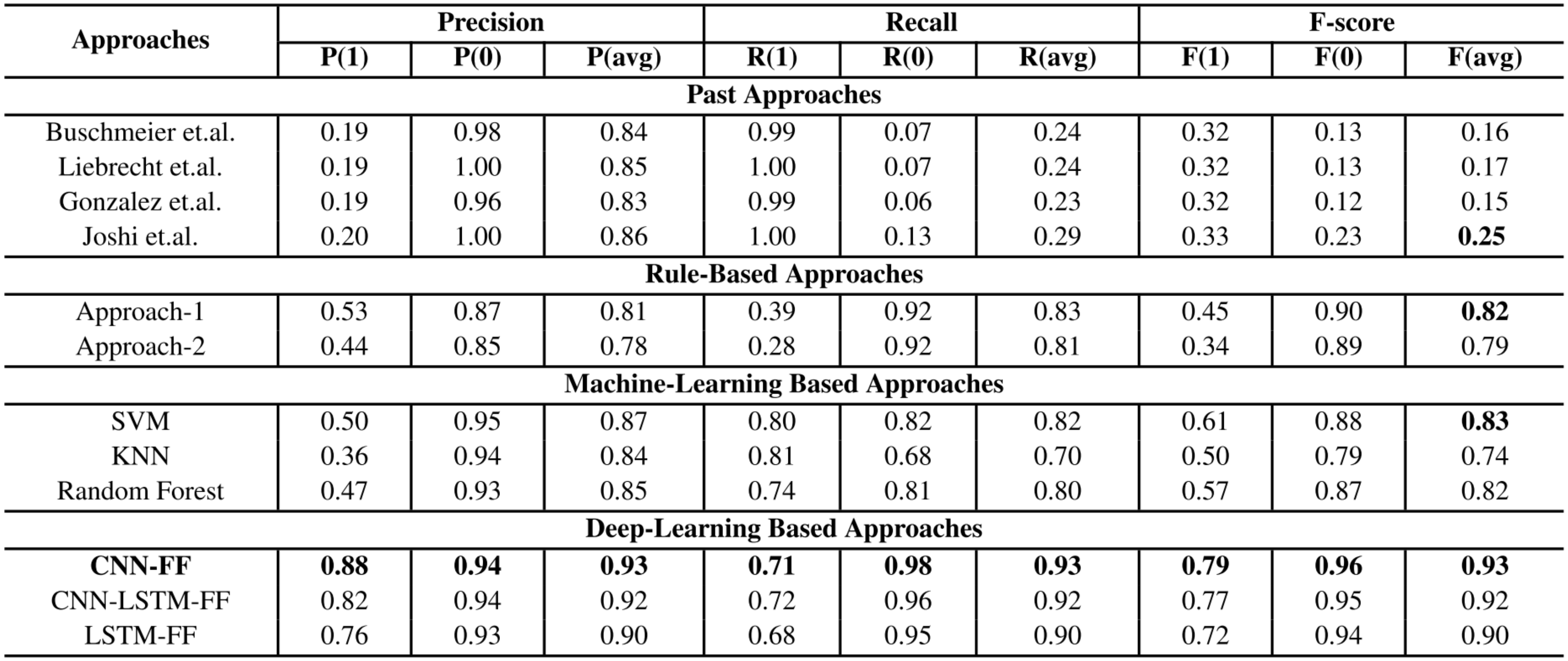
- Very little feature engg!!

- EmbeddingSize of 128
- Maximum tweet length 36 words
- Padding used
- Filters of size 3, 4, 5 used to extarct features

Deep Learning based approach: CNN-FF Model



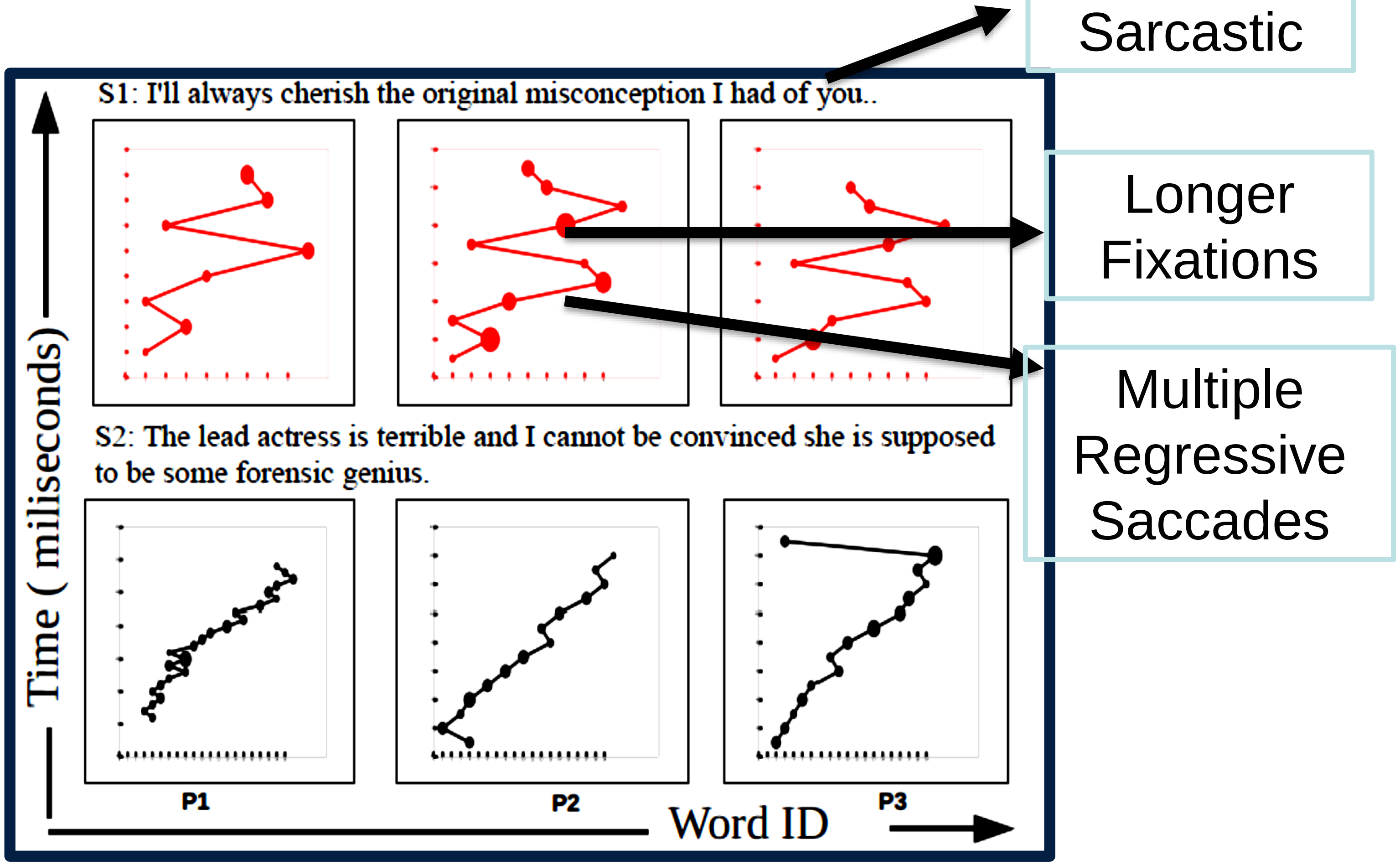
Comparison of results (1: sarcastic, 0: non-sarcastic)



Approaches	Precision			Recall			F-score		
	P(1)	P(0)	P(avg)	R(1)	R(0)	R(avg)	F(1)	F(0)	F(avg)
Past Approaches									
Buschmeier et.al.	0.19	0.98	0.84	0.99	0.07	0.24	0.32	0.13	0.16
Liebrecht et.al.	0.19	1.00	0.85	1.00	0.07	0.24	0.32	0.13	0.17
Gonzalez et.al.	0.19	0.96	0.83	0.99	0.06	0.23	0.32	0.12	0.15
Joshi et.al.	0.20	1.00	0.86	1.00	0.13	0.29	0.33	0.23	**0.25**
Rule-Based Approaches									
Approach-1	0.53	0.87	0.81	0.39	0.92	0.83	0.45	0.90	**0.82**
Approach-2	0.44	0.85	0.78	0.28	0.92	0.81	0.34	0.89	0.79
Machine-Learning Based Approaches									
SVM	0.50	0.95	0.87	0.80	0.82	0.82	0.61	0.88	**0.83**
KNN	0.36	0.94	0.84	0.81	0.68	0.70	0.50	0.79	0.74
Random Forest	0.47	0.93	0.85	0.74	0.81	0.80	0.57	0.87	0.82
Deep-Learning Based Approaches									
CNN-FF	**0.88**	**0.94**	**0.93**	**0.71**	**0.98**	**0.93**	**0.79**	**0.96**	**0.93**
CNN-LSTM-FF	0.82	0.94	0.92	0.72	0.96	0.92	0.77	0.95	0.92
LSTM-FF	0.76	0.93	0.90	0.68	0.95	0.90	0.72	0.94	0.90

back

Sentiment Annotation and Eye Movement

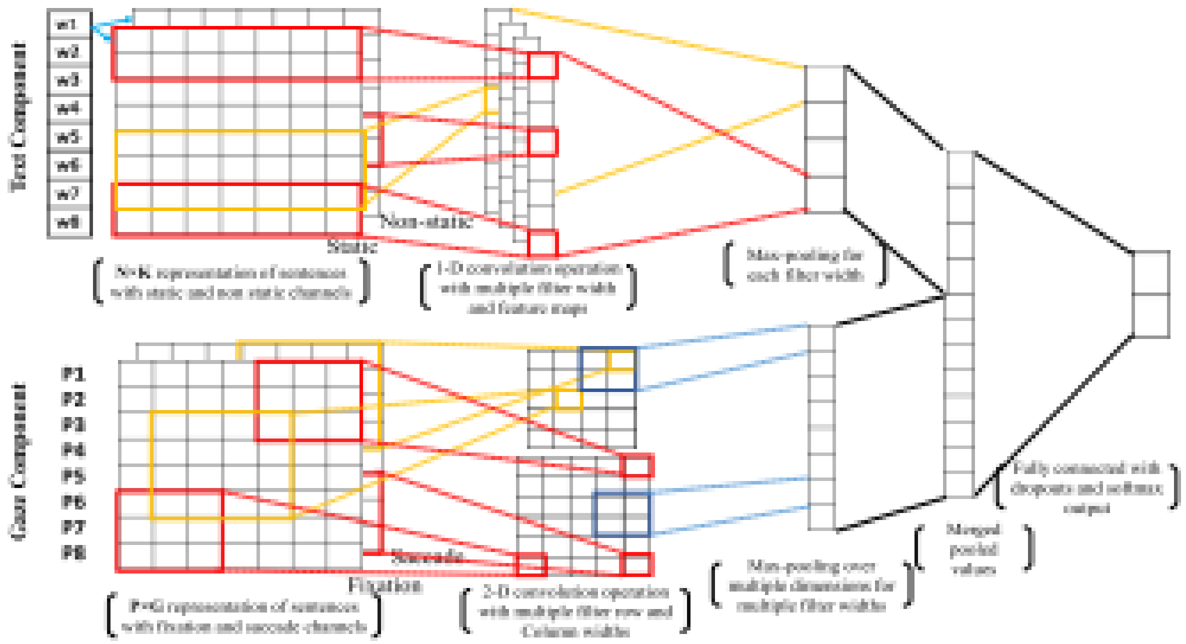


Datasets

- Two publicly available datasets released by us (Mishra et al, 2016; Mishra et al., 2014)
- **Dataset 1: (Eye-tracker: Eyelink-1000 Plus)**
  - 994 text snippets : 383 positive and 611 negative, 350 are sarcastic/ironic
  - Mixture of Movie reviews, Tweets and sarcastic/ironic quotes
  - Annotated by 7 human annotators
  - Annotation accuracy: **70%-90%** with Fleiss kappa IAA of **0.62**
- **Dataset 2: (Eye-tracker: Tobi TX300)**
  - 843 snippets : 443 positive and 400 negative
  - Annotated by 5 human subjects
  - Annotation accuracy: **75%-85%** with Fleiss kappa IAA of **0.68**

CNN Based Sarcasm Detection

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


Text Component
w1
w2
w3
w4
w5
w6
w7
w8
Non-static
Static
N×K representation of sentences with static and non static channels
1-D convolution operation with multiple filter width and feature maps
Max-pooling for each filter width
Gaze Component
P1
P2
P3
P4
P5
P6
P7
P8
Saccade
Fixation
P×G representation of sentences with fixation and saccade channels
2-D convolution operation with multiple filter row and Column widths
Max-pooling over multiple dimensions for multiple filter widths
Merged pooled values
Fully connected with dropouts and softmax output

Learning Cognitive Features from Gaze Data for Sentiment and Sarcasm Classification

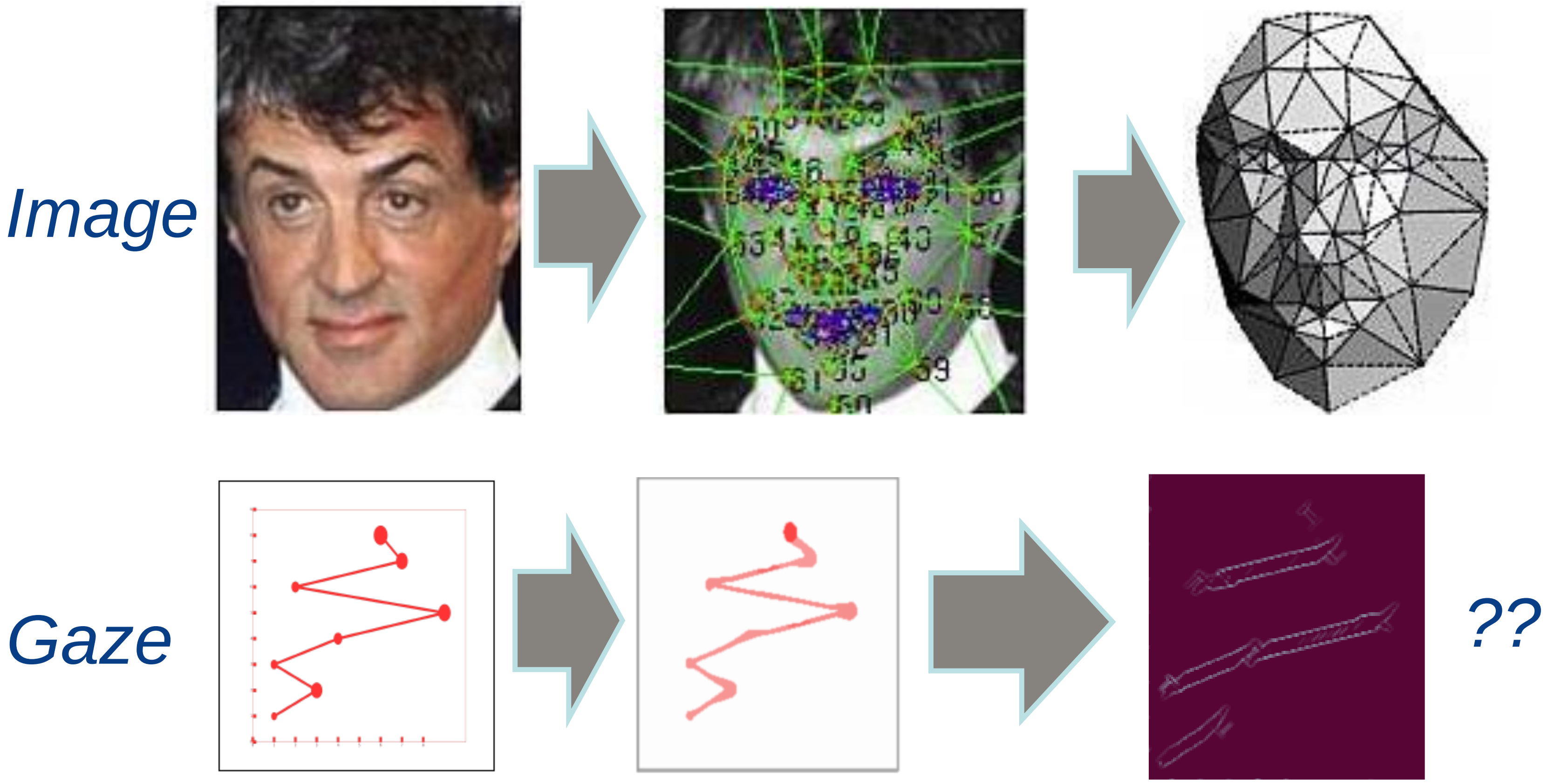
- In complex classification tasks like sentiment analysis and sarcasm detection, extraction and choice of features should be learnt
- CNN channels exploited
- CNN learns features from both gaze and text and uses them to classify the input text

Central Idea

- Learn features from Gaze sequences (fixation duration sequences and gaze-positions) and Text automatically using Deep Neural Networks.
- Deep NNs have proven to be good at learning feature representations for Image and Text classification tasks (Krizhevsky et al., 2012;Collobert et al., 2011).
- Use Convolutional Neural Network (already used for sentiment classification, Kim, 2014)

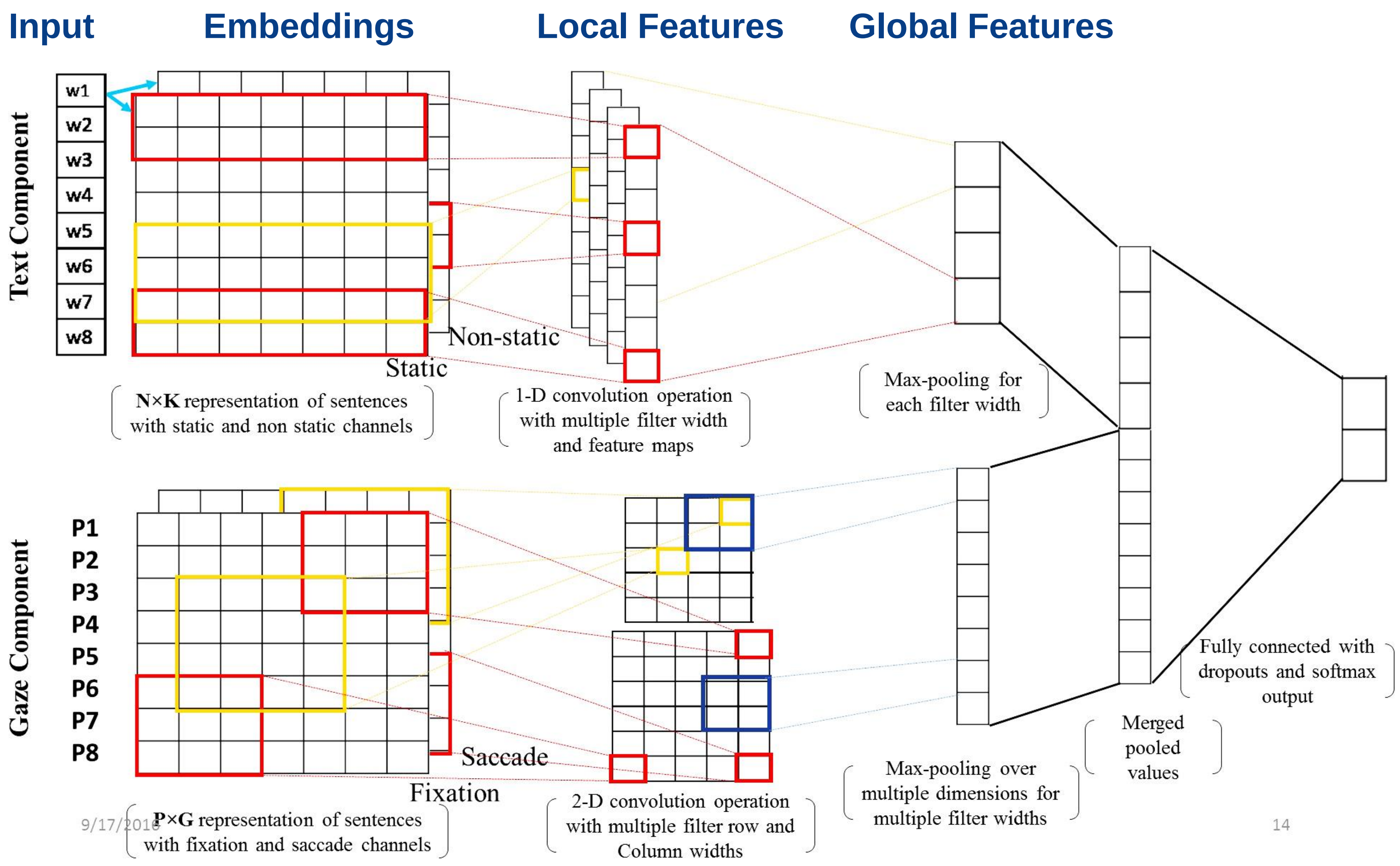
Why Convolutional NNs

- Convolutional Layers good at capturing compositionality (Lawrence et al, 1997).



Images taken from: mrulafi.blogspot.com

Neural Network Architecture


Input
Embeddings
Local Features
Global Features
Text Component
w1
w2
w3
w4
w5
w6
w7
w8
Non-static
Static
N×K representation of sentences with static and non static channels
1-D convolution operation with multiple filter width and feature maps
Max-pooling for each filter width
Gaze Component
P1
P2
P3
P4
P5
P6
P7
P8
Saccade
Fixation
P×G representation of sentences with fixation and saccade channels
2-D convolution operation with multiple filter row and Column widths
Max-pooling over multiple dimensions for multiple filter widths
Merged pooled values
Fully connected with dropouts and softmax output
9/17/2016
14

Why both Static and Non-static embedding

- Non-static embedding channel for tuning embeddings for SA/Sarcasm (e.g., produce similar embeddings for adjectives like good and excellent)
- Static embedding channel: to prevent over-tuning of embeddings due to collocation (e.g., words such as I and love are often collocated but should not share similar vector representation).

Fixation and Saccade Channels

- Fixation channel: Lexical Complexity (pertaining to length, frequency and predictability of words while annotation)
- Saccade channel: Syntactic Complexity and Incongruity

Datasets (1/2)

- Two publicly available datasets released by us (Mishra et al, 2016; Mishra et al., 2014)
- Dataset 1: (Eye-tracker: Eyelink-1000 Plus)
  - 994 text snippets : 383 positive and 611 negative, 350 are sarcastic/ironic
  - Mixture of Movie reviews, Tweets and sarcastic/ironic quotes
  - Annotated by 7 human annotators
  - Annotation accuracy: 70%-90% with Fleiss kappa IAA of 0.62

Datasets (2/2)

- Dataset 2: (Eye-tracker: Tobi TX300)
- 843 snippets : 443 positive and 400 negative
- Annotated by 5 human subjects
- Annotation accuracy: 75%-85%with Fleiss kappa IAA of 0.68

Experimental Setup: Configurations

- Text Only: (Only Text Component is Used)
  - Text_Static: Word embeddings are kept static and not updated during back propagation.
  - Text_Non-static: Embeddings are updated during back propagation.
  - Text_Multi Channel: Two channels (one taking input from static and one from dynamic embeddings) are used.

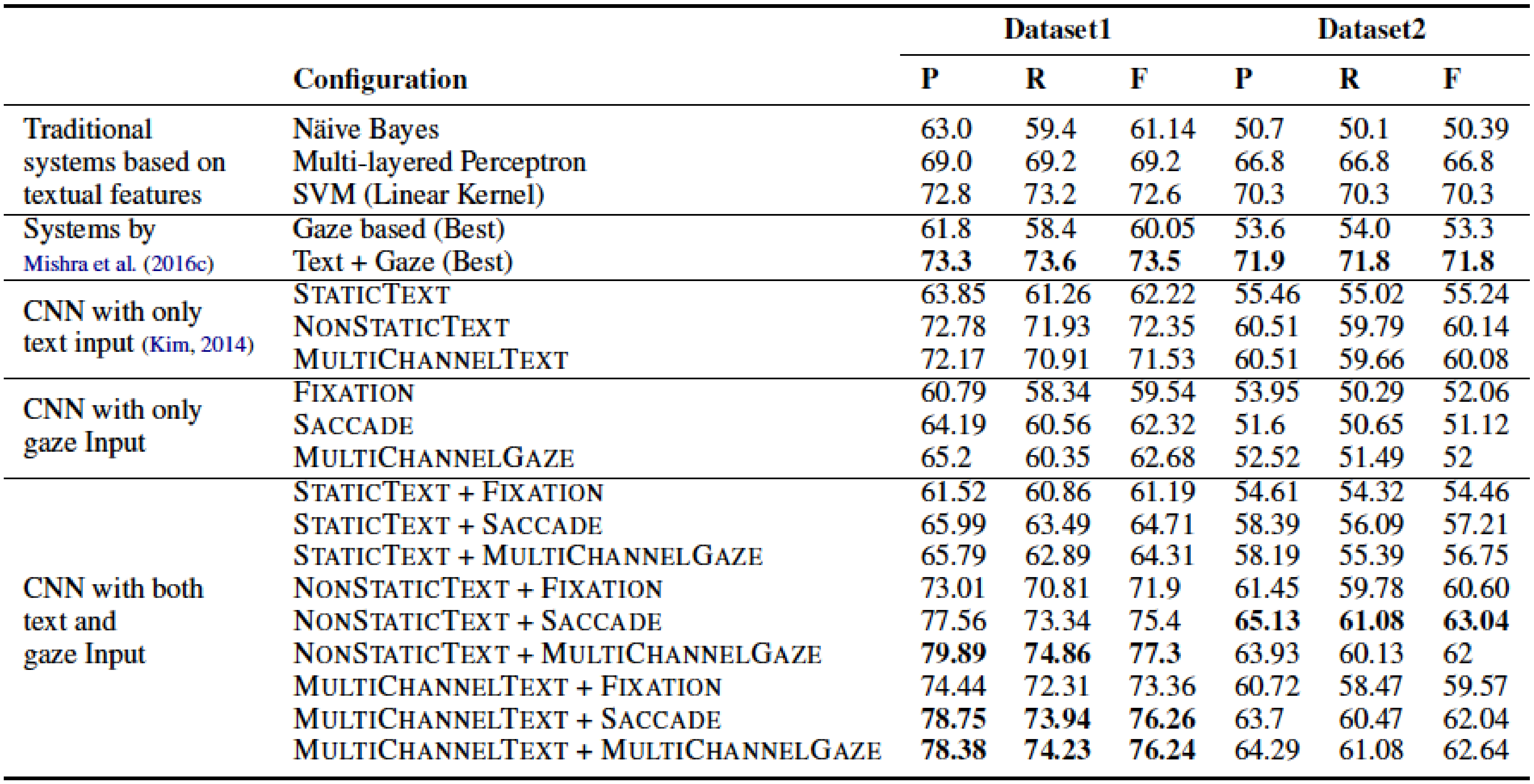
- Gaze Only: (Only Gaze Component is Used)
  - Gaze_Fixation_Duration: Sequence of fixation durations are used as input
  - Gaze_Saccade: Sequence of gaze locations (in terms of word ID used as input)
  - Gaze-Multi Channel: Two channels (one taking input from Fixation and one from saccade) are used

- Both text and Gaze (9-Configs)

Experiment Setup (Model Details)

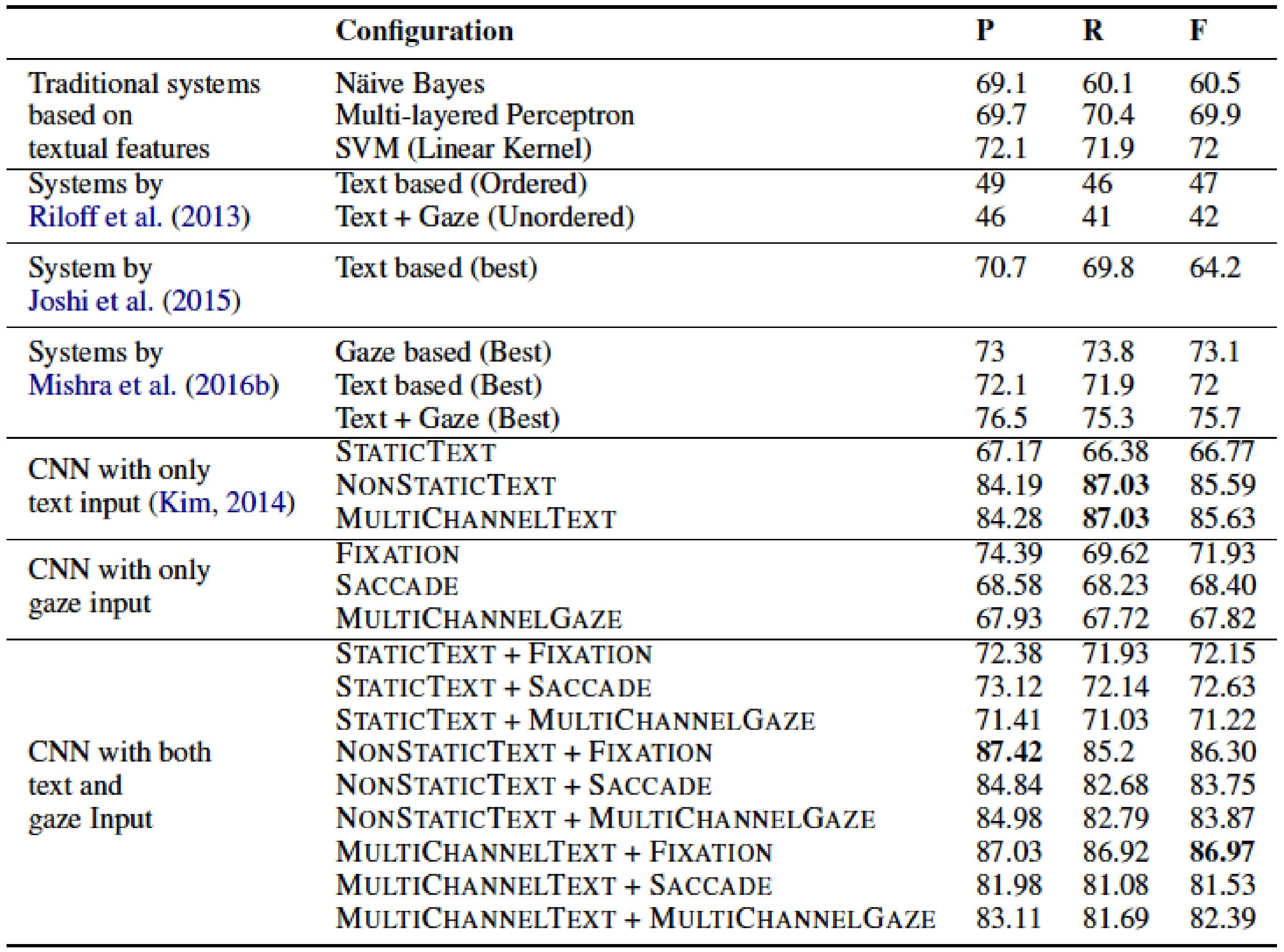
- Word Embeddings: Word2Vec (Mikolov et.al), trained on Amazon Movie Review Data, Embedding dimensions: 300
- Convolution: Filter sizes: 3,4 (Best), Number of filters used for each filter size: 150 (Better than smaller values)
- Feed-Forward: Number of hidden neurons: 150 (Better than smaller values), Dropout probability: 0.25
- Training: Number of epochs: 200 (change in loss negligible after 200 epochs), Optimizer: Adadelta, LR: 0.1

Results – Sentiment Analysis



	Configuration	Dataset1 P	Dataset1 R	Dataset1 F	Dataset2 P	Dataset2 R	Dataset2 F
Traditional systems based on textual features	Näive Bayes	63.0	59.4	61.14	50.7	50.1	50.39
	Multi-layered Perceptron	69.0	69.2	69.2	66.8	66.8	66.8
	SVM (Linear Kernel)	72.8	73.2	72.6	70.3	70.3	70.3
Systems by Mishra et al. (2016c)	Gaze based (Best)	61.8	58.4	60.05	53.6	54.0	53.3
	Text + Gaze (Best)	**73.3**	**73.6**	**73.5**	**71.9**	**71.8**	**71.8**
CNN with only text input (Kim, 2014)	StaticText	63.85	61.26	62.22	55.46	55.02	55.24
	NonStaticText	72.78	71.93	72.35	60.51	59.79	60.14
	MultiChannelText	72.17	70.91	71.53	60.51	59.66	60.08
CNN with only gaze Input	Fixation	60.79	58.34	59.54	53.95	50.29	52.06
	Saccade	64.19	60.56	62.32	51.6	50.65	51.12
	MultiChannelGaze	65.2	60.35	62.68	52.52	51.49	52
CNN with both text and gaze Input	StaticText + Fixation	61.52	60.86	61.19	54.61	54.32	54.46
	StaticText + Saccade	65.99	63.49	64.71	58.39	56.09	57.21
	StaticText + MultiChannelGaze	65.79	62.89	64.31	58.19	55.39	56.75
	NonStaticText + Fixation	73.01	70.81	71.9	61.45	59.78	60.60
	NonStaticText + Saccade	77.56	73.34	75.4	**65.13**	**61.08**	**63.04**
	NonStaticText + MultiChannelGaze	**79.89**	**74.86**	**77.3**	63.93	60.13	62
	MultiChannelText + Fixation	74.44	72.31	73.36	60.72	58.47	59.57
	MultiChannelText + Saccade	**78.75**	**73.94**	**76.26**	63.7	60.47	62.04
	MultiChannelText + MultiChannelGaze	**78.38**	**74.23**	**76.24**	64.29	61.08	62.64

Results – Sarcasm Detection



	Configuration	P	R	F
Traditional systems based on textual features	Näive Bayes	69.1	60.1	60.5
	Multi-layered Perceptron	69.7	70.4	69.9
	SVM (Linear Kernel)	72.1	71.9	72
Systems by Riloff et al. (2013)	Text based (Ordered)	49	46	47
	Text + Gaze (Unordered)	46	41	42
System by Joshi et al. (2015)	Text based (best)	70.7	69.8	64.2
Systems by Mishra et al. (2016b)	Gaze based (Best)	73	73.8	73.1
	Text based (Best)	72.1	71.9	72
	Text + Gaze (Best)	76.5	75.3	75.7
CNN with only text input (Kim, 2014)	StaticText	67.17	66.38	66.77
	NonStaticText	84.19	**87.03**	85.59
	MultiChannelText	84.28	**87.03**	85.63
CNN with only gaze input	Fixation	74.39	69.62	71.93
	Saccade	68.58	68.23	68.40
	MultiChannelGaze	67.93	67.72	67.82
CNN with both text and gaze Input	StaticText + Fixation	72.38	71.93	72.15
	StaticText + Saccade	73.12	72.14	72.63
	StaticText + MultiChannelGaze	71.41	71.03	71.22
	NonStaticText + Fixation	**87.42**	85.2	86.30
	NonStaticText + Saccade	84.84	82.68	83.75
	NonStaticText + MultiChannelGaze	84.98	82.79	83.87
	MultiChannelText + Fixation	87.03	86.92	**86.97**
	MultiChannelText + Saccade	81.98	81.08	81.53
	MultiChannelText + MultiChannelGaze	83.11	81.69	82.39

Observations (1/2)

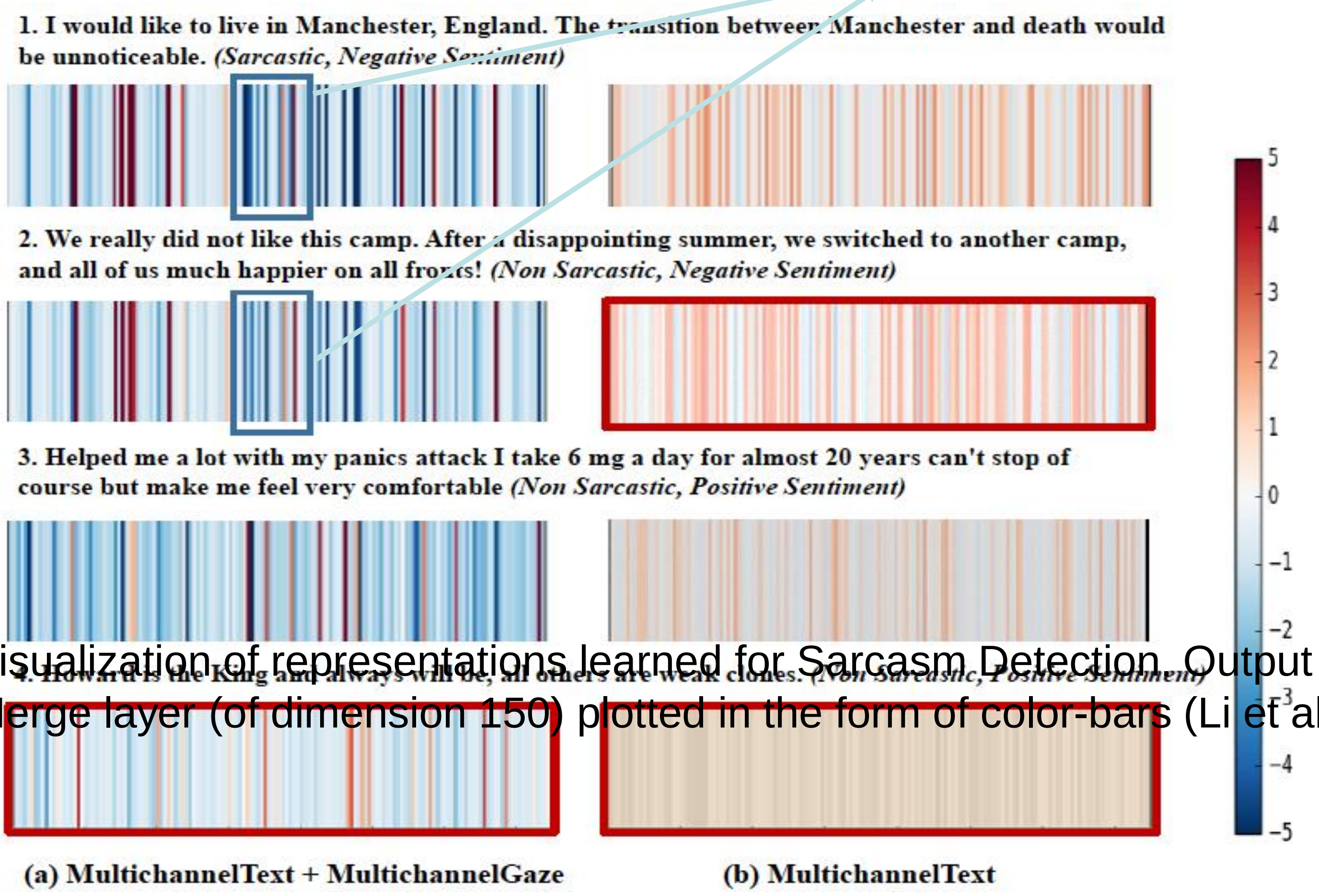
- Overfitting for SA dataset 2: Training accuracy reaches 100 within 25 epochs with validation accuracy still at around 50%. Better dropout/regularization configuration required.
- Better classification accuracy for Sarcasm detection: Clear differences between vocabulary of sarcasm and non-sarcasm classes in our dataset. Captured well by non-static embeddings.
- Effect of dimension variation: Reducing embedding dimension improves by a little margin.

Observations (2/2)

- Increasing filters beyond 180 decreases accuracy (possibly over-fits). Decreasing beyond 30 decreases accuracy.
- Effect of static / non static text channels: Better for non static (word embeddings with similar sentiment come closer in non static channels, e.g., *good* ~ *nice*
- Effect of fixation / saccade channels: Saccade channel alone handles nuances like incongruity better.
- Fixation channel does not help much, may be because of higher variance in fixation duration.

Analysis of Features Learned (1/2)

Capturing intensity variation in sarcasm VS no-sarcasm better



- Visualization of representations learned for Sarcasm Detection. Output of the Merge layer (of dimension 150) plotted in the form of color-bars (Li et al. , 2016)

Analysis of Features Learned (2/2)

- Addition of gaze information helps to generate features with more subtle differences.
- Features for the sarcastic texts exhibit more intensity than the non-sarcastic ones- perhaps capturing the notion that sarcasm typically conveys an intensified negative opinion.
- Example 4 is incorrectly classified by both the systems– lack of context?
- Addition of gaze information does not help here, as it becomes difficult for even humans to classify such texts

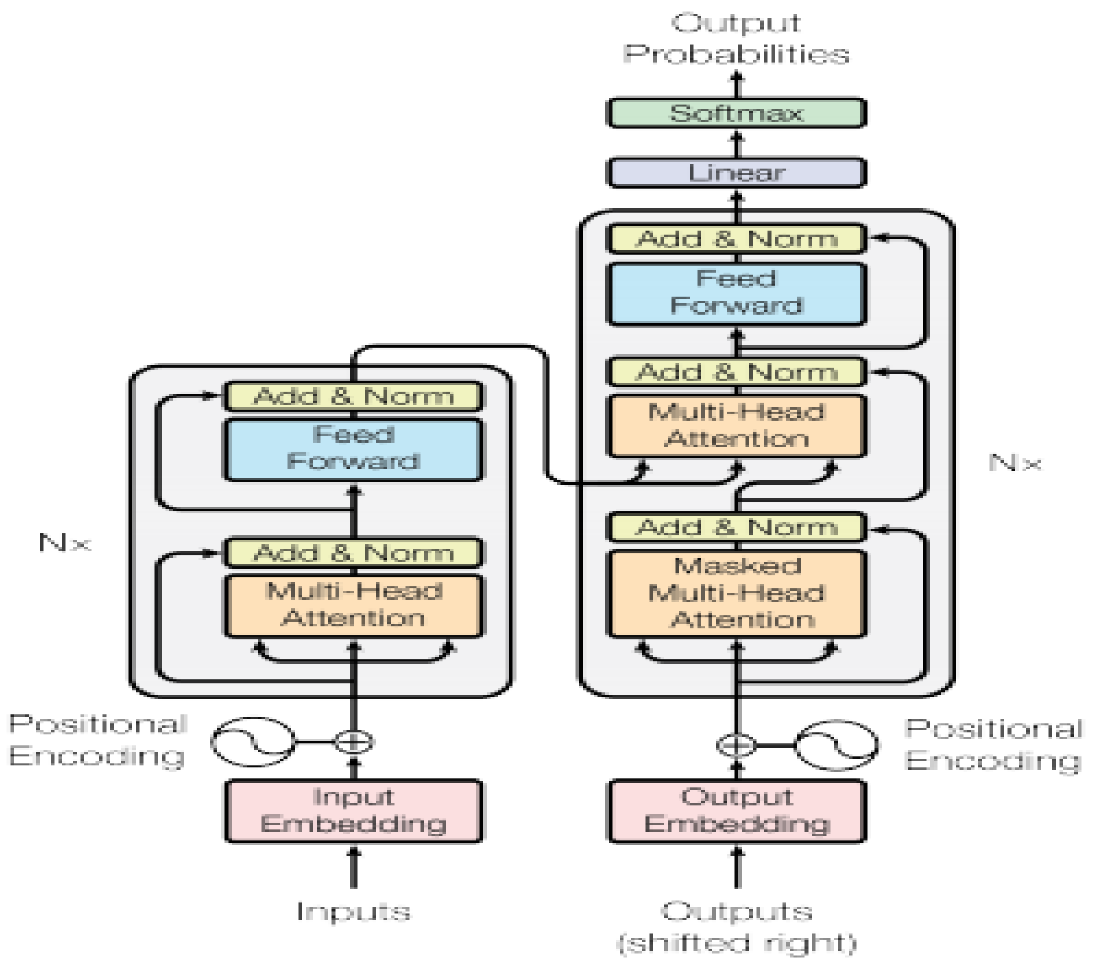
Project Idea

- CNN for multitask learning.
- For example, sentiment analysis and emotion detection.
- Dataset: IEmoCaps
- 2 filters to be used, one for each task.

Attention and Transformer

Arguably, the most important application-
MACHINE TRANSLATION

A classic diagram and a classic paper



Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need." NeurIPS (2017).

http://nlp.seas.harvard.edu/2018/04/03/attention.html
http://jalammar.github.io/illustrated-transformer/

Chronology

- IBM Models of Alignment- Brown *et al.* 1990, 1993
- Phrase Based MT- Koehn 2003
- Encoder Decoder- Sutskever *et al.* 2014, Cho *et al.* 2014
- Attention- Bahadanu *et al.* 2015
- Transformer- Vaswani *et al.* 2017

Attention

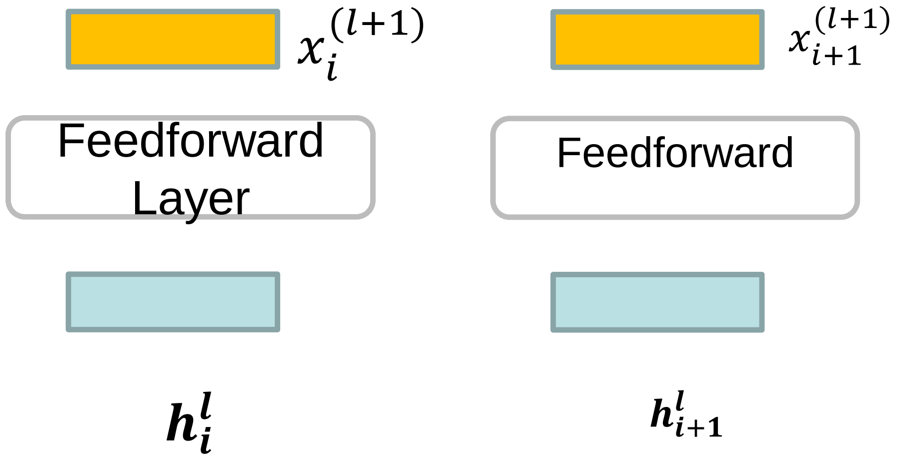
Compare every elements with all other elements

Represent the input context as *a* weighted average of input word embeddings

$$\boldsymbol{h}_i^l = \sum_{i=1}^{N} \boldsymbol{w}_i x_i$$

$$\boldsymbol{x}_i^{l+1} = \boldsymbol{FF}(h_i^l)$$

How do we compute weights ➔ Attention!



Non-recurrent ➔ this operation can be applied in parallel to all elements in the sequence

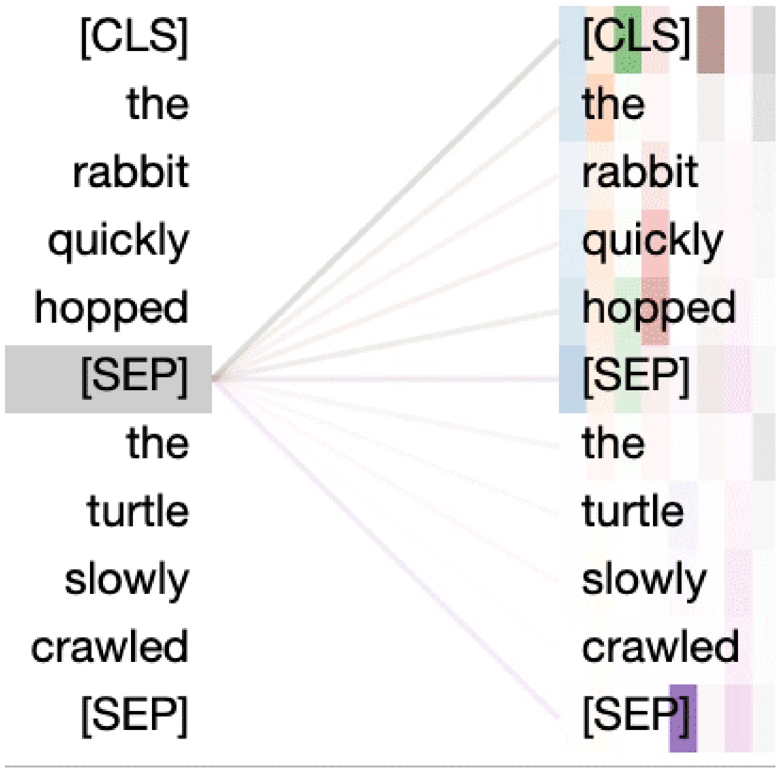
Self-Attention

Every word is compared with every other word in the same sentence

X ➔ query

$x_1, x_2\ x_3 \dots x_n$ ➔ values

Direct comparison between arbitrary words ➔ long-range dependencies can be better modelled



More computations than Recurrent models: $O(n^2)$

Important observations on self attention

- In the input sequence, pairs of words differ in their strength of association
- For example for an adjective-noun combination, adjective’s attention should be stronger for the noun than for other words in the sentence
- So the key questions are:
  - What to attend to
  - With how much attention to attend to

Attention strength vs. Co-occurrence count

- The strength of attention is not related to probability of co-occurrence.
- Co-occurrence count is used to identify words that frequently appear together, while attention strength is used to determine which words or phrases are most important in a given context.
- Attention strength is based on semantics of a sentence whereas co-occurrence is based on maximum likelihood.

Attention between pair of words

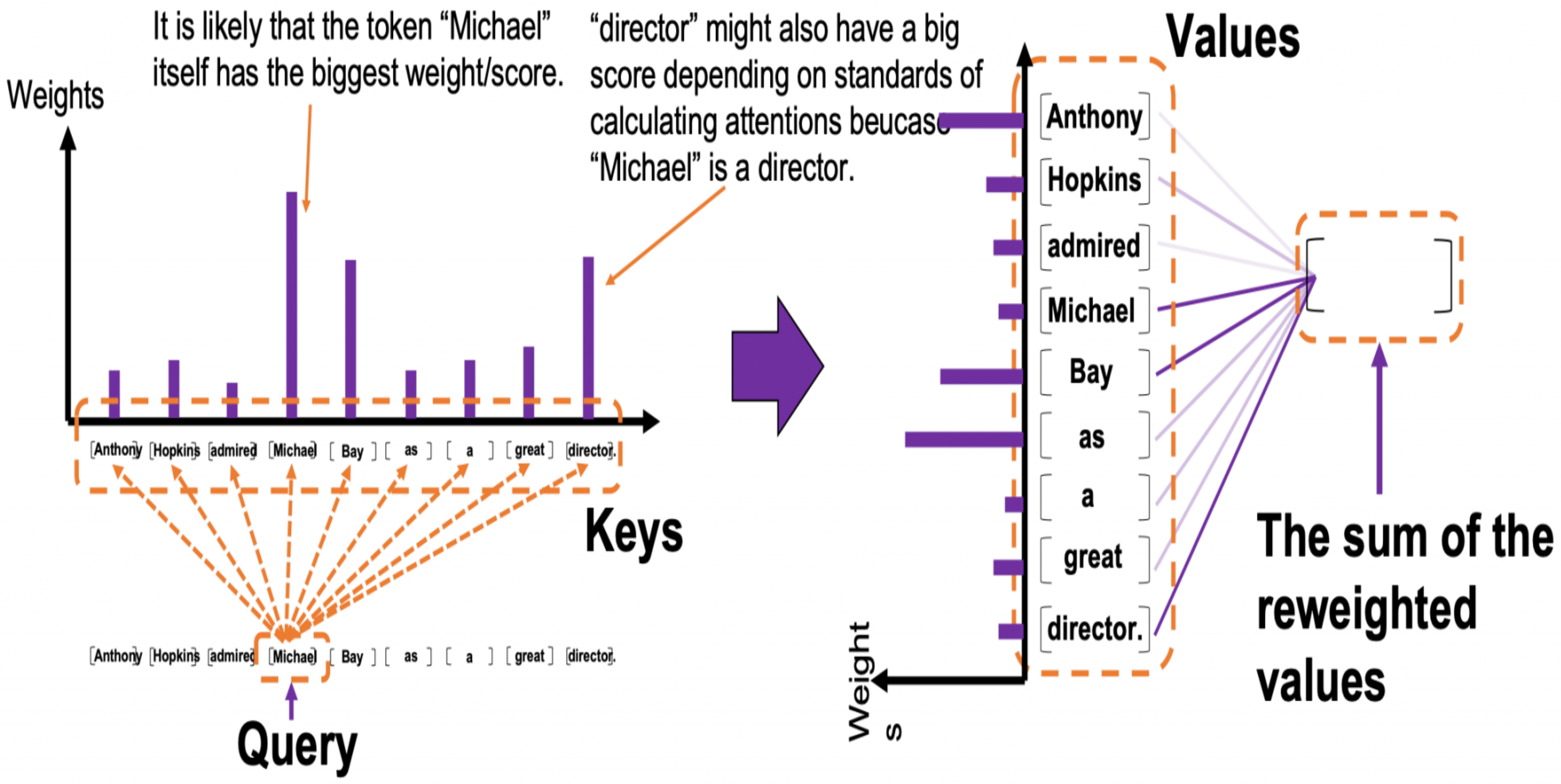
Example: ***Peter slept early.***

Peter	slept	high attention
slept	early	high attention
Peter	early	low attention

Attention that is non-self

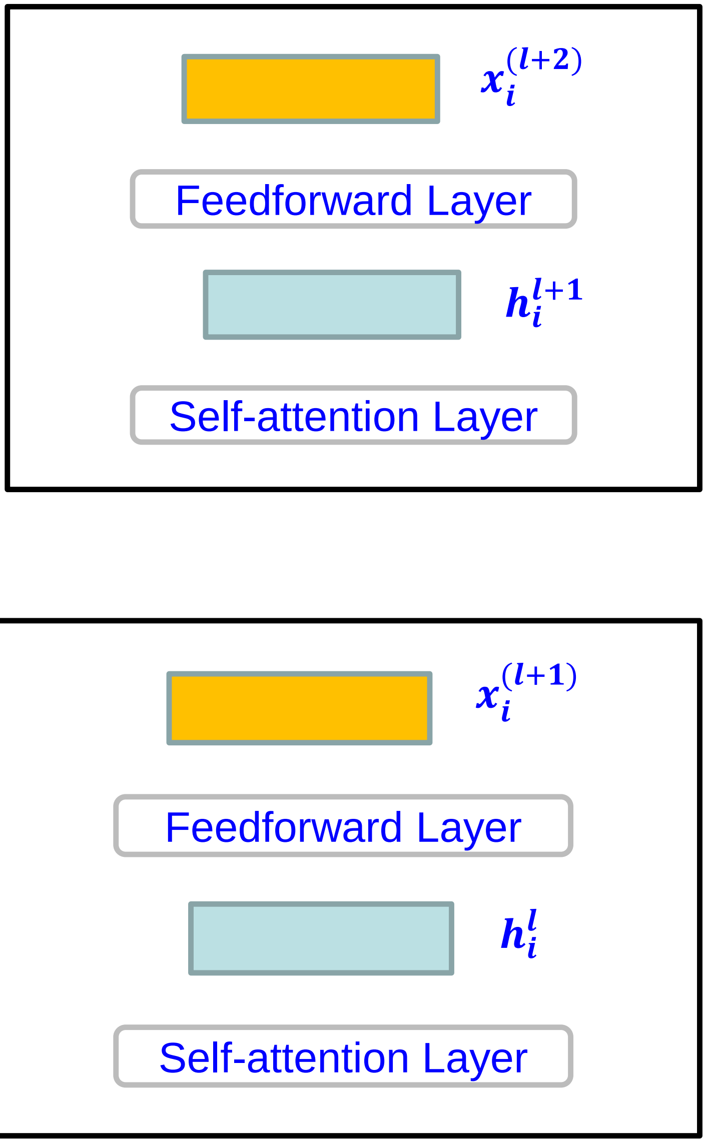
- When the decoder generates the output sequence, attention is a 2-part attention
- Each output token should attend to whatever token has been output before
- Additionally, it should attend to the tokens in the input sequence

Fundamental concepts- “Attention”, “query”, “key”, “value”


It is likely that the token “Michael” itself has the biggest weight/score.
“director” might also have a big score depending on standards of calculating attentions beucase “Michael” is a director.
Weights
Anthony
Hopkins
admired
Michael
Bay
as
a
great
director.
Keys
Query
Values
Weights
The sum of the reweighted values

Transformer Architecture

Stack self-attention blocks to create deep networks



Positional Embeddings

The ICICI bank branch is on the bank of the river

The self-attention model has no notion of position,
➔ same words will have same representations irrespective of their position/syntactic role in the sentence

Create positional embeddings that uniquely and deterministically identify a position

Add it to the word embedding at the bottom layer

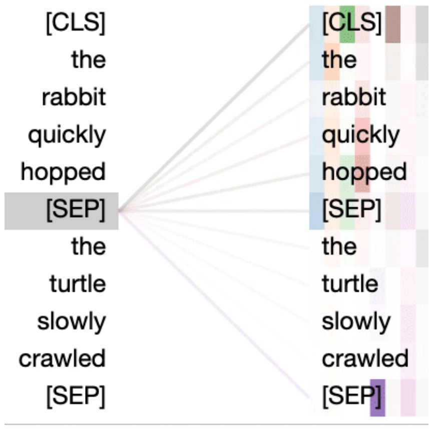
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

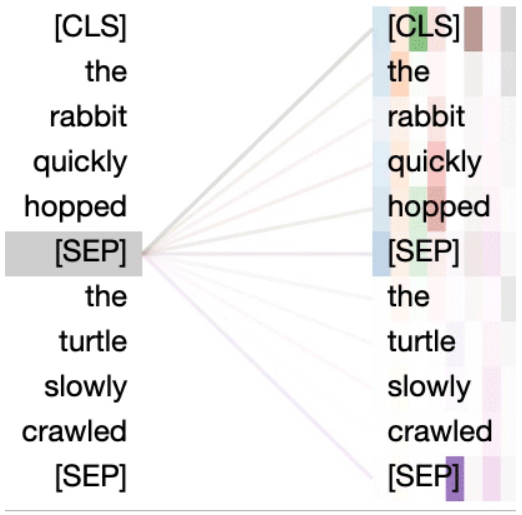
Advantage of using Positional Embedding

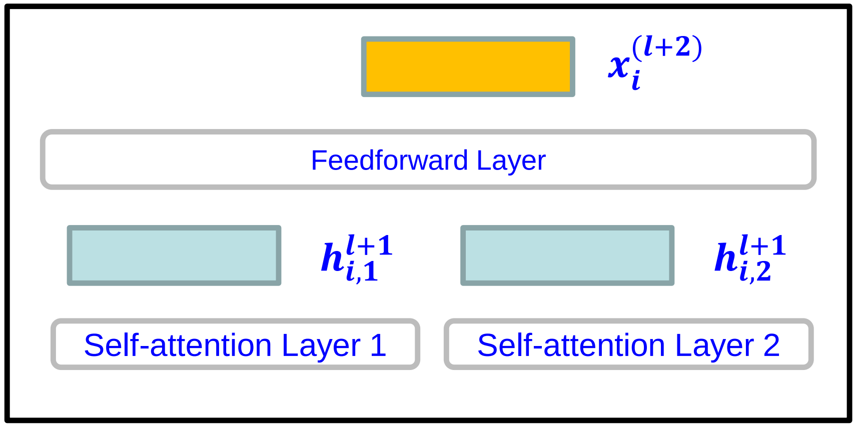
Example: **Raam ne Shyam ko dekha.**

- Using positional encoding in machine translation makes it easy for the model to learn the distinction between "**ne**" and "**ko**".
- In contrast, using SRL tags instead of positional encoding requires a large amount of data to train the model. Therefore, utilizing positional encoding has simplified the task.

Multiple self-attention heads



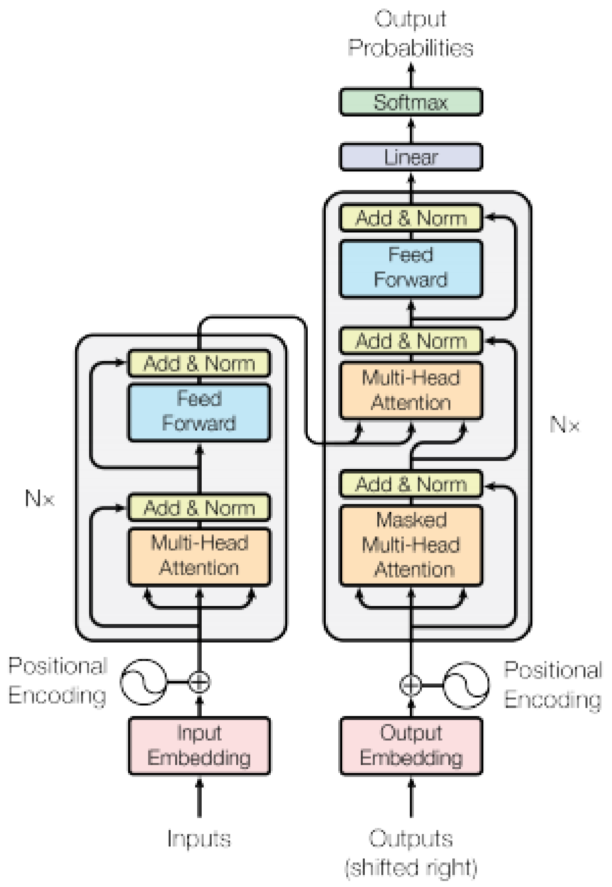




Multiple self-attention networks at each layer

Each head learns different kinds of dependencies

Putting it all together



Decoder layer also has a cross-attention layer

Decoder ➔ masking for future time-steps while computing self-attention

There are residual connections & layer-normalization between layers

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need." NeurIPS (2017).

http://nlp.seas.harvard.edu/2018/04/03/attention.html
http://jalammar.github.io/illustrated-transformer/

Transformer has led to tremendous advances in MT

Encoder architectures like BERT based on Transformer have yielded large improvements in NLU tasks

Transformer models are the de-facto standard models for many NLP tasks

Back to attention

What is “Attention”

- **Attention** enhances the important parts of the input data and fades out the rest

- The network should devote more computing power on that small part of the data that matters

Sentence-1

- Ram who is a good student and lives in London which is a large metro, will go to the University for higher studies.
- राम जो एक अच्छा छात्र है और लंदन में रहता है जो एक बड़ी मेट्रो है, उच्च अध्ययन के लिए विश्वविद्यालय जाएगा।

Sentence-2

- Sita who is a good student and lives in London which is a large metro, will go to the University for higher studies.
- सीता जो एक अच्छी छात्रा है और लंदन में रहती है जो एक बड़ी मेट्रो है, उच्च अध्ययन के लिए विश्वविद्यालय जाएगी।

Learning “Attention”

- Which part of the data is more important than others depends on the context

- Learned through training data by **gradient descent**

Two kinds of Attention

- Dot Product Attention
- Multihead Attention

Dependency Parse- Attention by Parsing

- root(ROOT-0, go-18)
- nsubj(go-18, Ram-1)
- nsubj(student-6, who-2)
- cop(student-6, is-3)
- det(student-6, a-4)
- amod(student-6, good-5)
- acl:relcl(Ram-1, student-6)
- cc(lives-8, and-7)
- conj(student-6, lives-8)
- case(London-10, in-9)
- nmod(lives-8, London-10)
- nsubj(metro-15, which-11)
- cop(metro-15, is-12)
- det(metro-15, a-13)
- amod(metro-15, large-14)
- acl:relcl(student-6, metro-15)
- aux(go-18, will-17)
- case(University-21, to-19)
- det(University-21, the-20)
- obl(go-18, University-21)
- case(studies-24, for-22)
- amod(studies-24, higher-23)
- nmod(University-21, studies-24)

Attention and Alignment

Hindi (col) --> English (row) | \/	**PIITA R (पीटर)**		**JALDII (जल्दी)**		**SOYA A (सोया)**
PETER	1		0		0
SLEPT	0		0		1
EARLY	0		1		0

Attention and Alignment

- Word alignment in statistical machine translation is analogous to cross attention in neural machine translation

- Phrase alignment uses both self attention and cross attention.

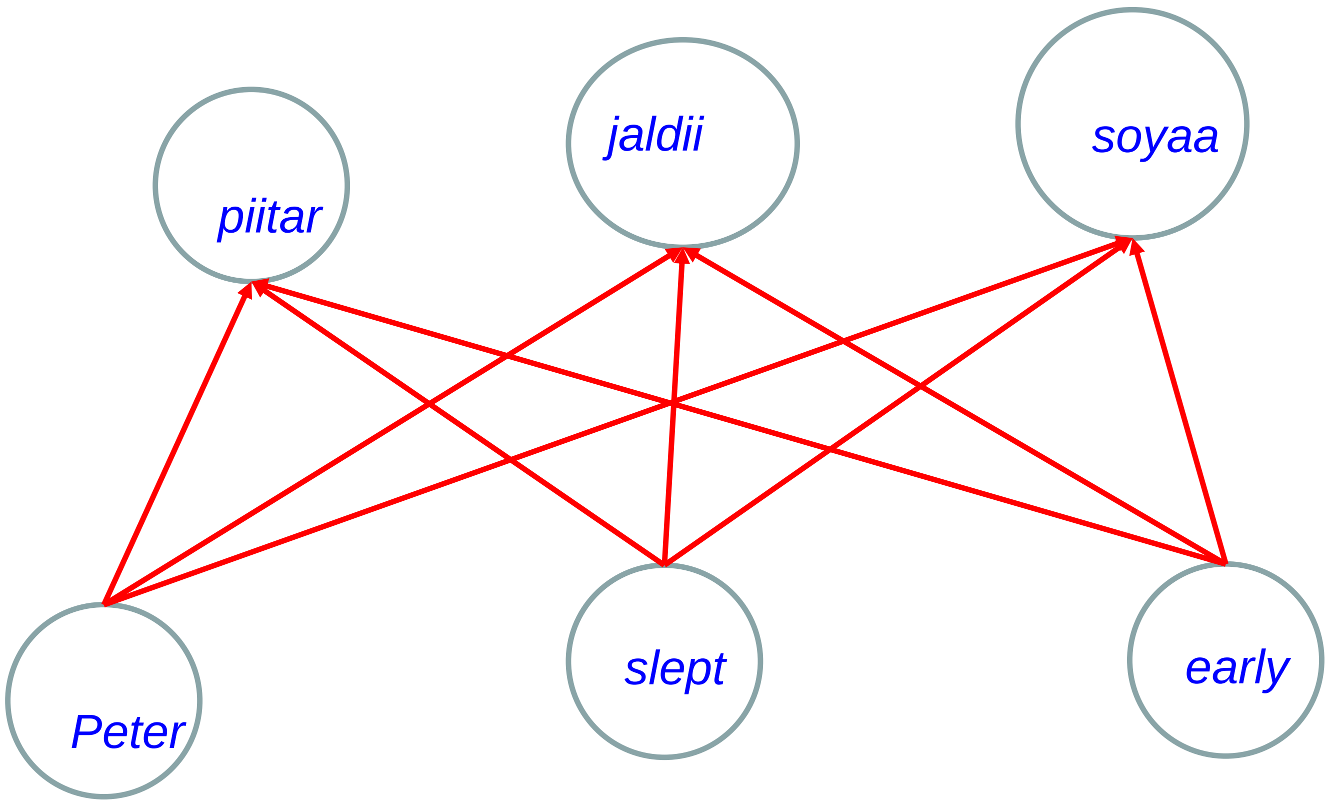
Attention: Linguistics and NLP view

- Attention was known to linguists but its measurement is the contribution of NLP

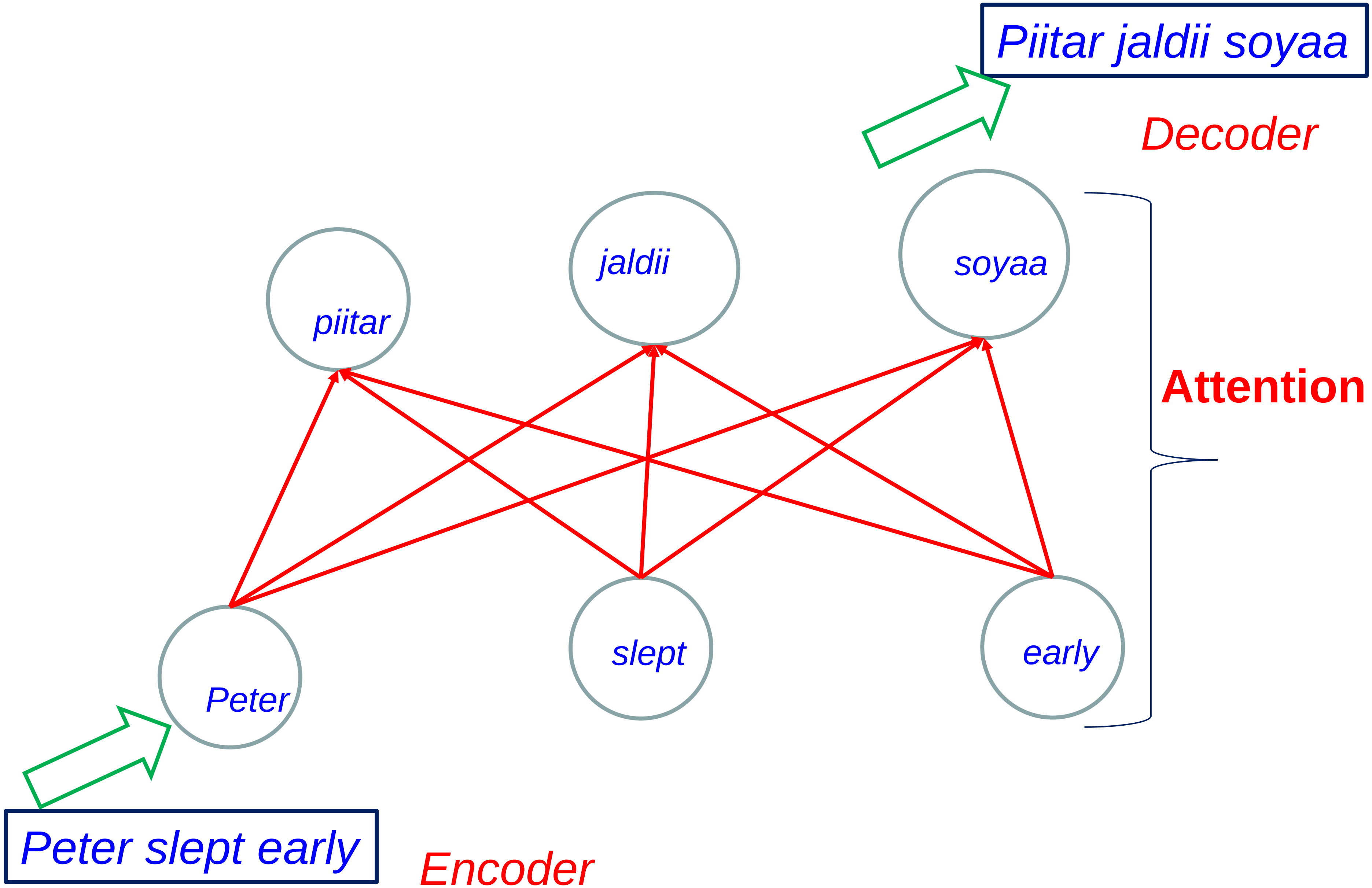
- Attention is measured in terms of probability.

FFNN for alignment:

Peter slept early → piitar jaldii soyaa

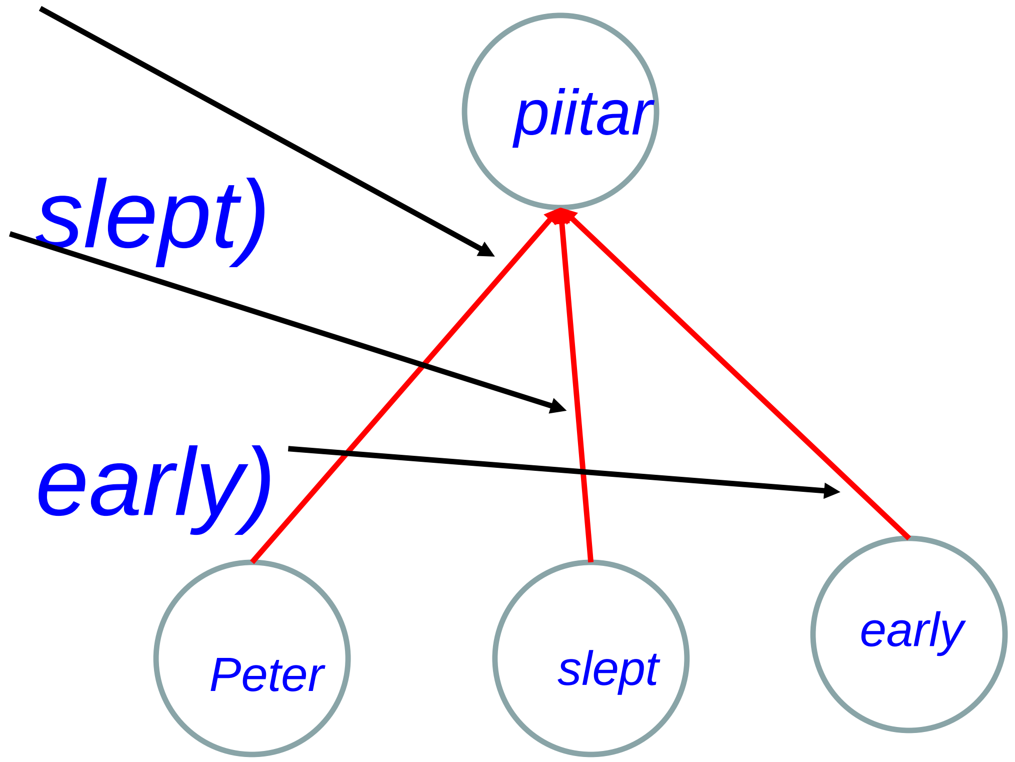


Introduce Attention Layer between Encoder and Decoder



How to learn the weights- attention weights?

- Weight (*piitar*, *peter*)
- Weight (*piitar*, *slept*)
- Weight (*piitar*, *early*)



Positional Encoding

Limitation of RNN

- Encoder-decoder RNN generates a sequence of hidden states h_t, t varying from 0 to L, where L is the sentence length.
- Each h_t is a function of previous hidden state h_{t-1} and the input at position t.
- So, to process the input at t^{th} step, the encoder or decoder has to wait for $t-1$ steps.
- This sequential nature of RNN makes the training time very large.

Inspiration from Shakespeare

- “All the world's a stage,/ And all the men and women merely players”- As You Like It- Shakespeare

- All the sentence’s a stage./And all the words and punctuations are merely players

“*children saw a big lion in the zoo in the morning*”

- main verb: *saw*;
- who (agent): *children*
- what (object): *lion*
- where (locative): *zoo*
- when (temporal): *evening*

Position Sensitivity: “Jack saw Jill” vs. “Jill saw Jack”

IF

the main verb (MV) is transitive and in past tense

THEN

the NP to the left of MV should get the ‘ ne’ postposition mark

and

The NP to the right of MV should get the ‘ko’ postposition mark

Transformer's major contribution- Positional Encoding (1/2)

- Word positions as additional disambiguation signals.
- Words influence one another by virtue of their properties and positions
- Such influences manifest in translations as morphological transformations, lexical choices, pragmatic markers and so on.
- Tenet of ML-NLP: *with sufficient data all these mutual influences can be learnt*.

Transformer’s major contribution- Positional Encoding (2/2)

- Positions are encoded as embeddings and positional embeddings are supplied along with input word embeddings.

- The training phase teaches the transformer to condition the output by paying attention to not only input words, but also their positions.

Position Vector components

- Let the *i*th component of the *t*th position vector be denoted as *pos(t,i)*, *i* varying from 0 to *(d/2)-1*. Then

$$pos(t,2i) = \sin\left(\frac{1}{10000^{\frac{2i}{d}}} t\right)$$

$$pos(t,2i+1) = \cos\left(\frac{1}{10000^{\frac{2i}{d}}} t\right)$$

Why Sine and Cosine? (1/2)

Foundational Observation-1:

Let *S* be a set of symbols. Let *P* be the set of patterns the symbols create. If |P|>|S|, then there must exist patterns in *P*, that have repeated symbols.

Why Sine and Cosine? (2/2)

Foundational Observation-2:

IF

the patterns can be arranged in a series with equal difference of values between every consecutive pair,

THEN

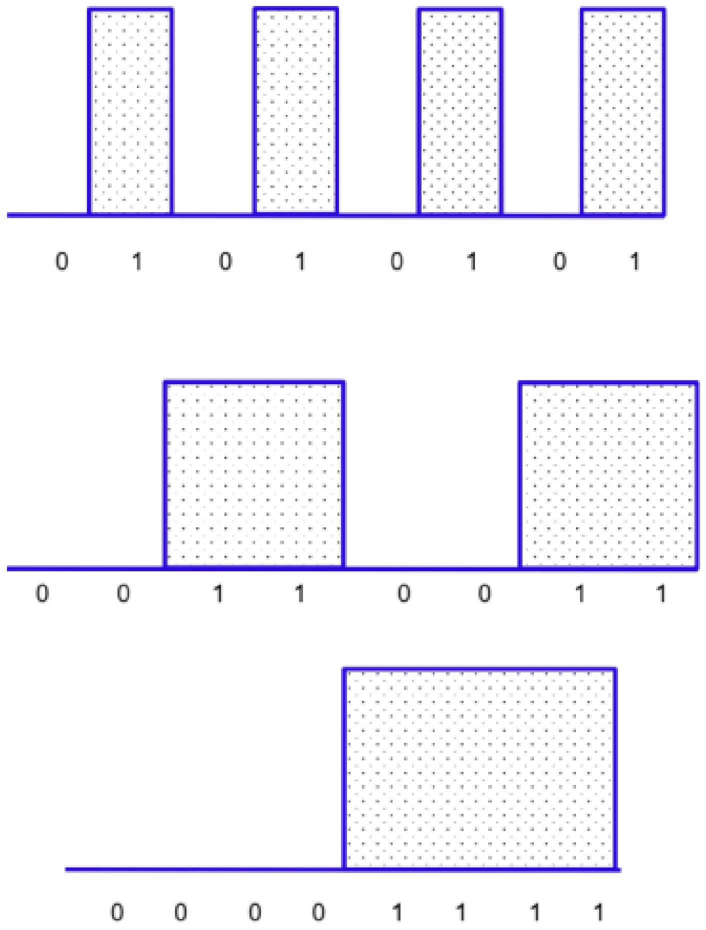
at any given position, the symbols at different positions of the pattern strings must REPEAT,

Periodicity and Decimal Integers

- 10 symbols called digits: *0, 1, 2, 3, 4, 5, 6, 7, 8, 9*.
- In the sorted list of integers in ascending order, the string length of the integer goes on increasing
- The digits repeat after every 10 numbers in the lowest significant position, after every 100 numbers in the next lowest position, after every 1000 numbers in the next to next lowest and so on.

Binary Numbers

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



Challenges in designing PEs

- Cannot append decimal integers as position values; words later in the sentence will dominate, by the force of their positions being large integers
- Cannot normalize too: Word relations changing with the length of sentences- linguistically untenable
- "*Oh, what a beautiful day!!*"- which expresses (i) delight, (ii) the nature of the '*day*' being '*beautiful*', (iii) '*Oh*', being an exclamatory prefix to the rest of the phrase and so on, should be invariant with respect to the sentence length

Binary values also will not do!

- *0*s will contribute nothing, and *1*s will influence completely.
- Such *black-and-white* (0-1) hard decisions go against the grain of NLP whose other name is ambiguity.
- A language object represented by a vector must allow *soft choices* in its components, preferably represented by values in the closed range [0,1].

Criteria PEs should satisfy

- Should be *added* component by component to the word vector.
- Components should range from 0 to 1, both included.
- Components should be periodic, since they represent consecutive integers.
- Ingenious on the part of the creators of transformers to spot that *sine* and *cosine* functions meet the above requirements.

Example: “Jack saw Jill”

- Three positions indexed as 0, 1 and 2. Assume word vector dimension *d* to be
1/(10

$$pos\_vector('Jack') =< pos(0.0), pos(0.1), pos(0.0), pos(0.2), pos(0.3) >$$
$$=< \sin(0), \cos(0), \sin(0), \cos(0) >$$
$$=< 0,1,0,1 >$$

$$pos\_vector('saw') =< pos(1.0), pos(1.1), pos(1.2), pos(1.3) >$$
$$= \left\langle \sin\left(\frac{1}{10^{\frac{2.0}{4}}}\right), \cos\left(\frac{1}{10^{\frac{2.0}{4}}}\right), \sin\left(\frac{1}{10^{\frac{2.1}{4}}}\right), \cos\left(\frac{1}{10^{\frac{2.1}{4}}}\right)\right\rangle$$
$$= \left\langle \sin(1), \cos(1), \sin\left(\frac{1}{10^{0.5}}\right), \cos\left(\frac{1}{10^{0.5}}\right)\right\rangle$$

$$pos\_vector('Jill') =< pos(2.0), pos(2.1), pos(2.2), pos(2.3) >$$
$$=< \left\langle \sin(2), \cos(2), \sin\left(\frac{2}{10^{0.5}}\right), \cos\left(\frac{2}{10^{0.5}}\right)\right\rangle$$

Statistical Alignment Learning

Non-neural

EM for word alignment from sentence alignment: example

English				**French**			
(1)	three	rabbits		(1)	trois	lapins	
	a	b			w	x	
(2)	rabbits	of	Grenoble	(2)	lapins	de	Grenoble
	b	c	d		x	y	z

Initial Probabilities: each cell denotes *t(a←→w), t(a←→x) etc.*

	a	b	c	d
w	1/4	1/4	1/4	1/4
x	1/4	1/4	1/4	1/4
y	1/4	1/4	1/4	1/4
z	1/4	1/4	1/4	1/4

Example of expected count

$C[w \leftrightarrow a;\ (a\ b) \leftrightarrow (w\ x)]$

$$= \frac{t(w \leftrightarrow a)}{t(w \leftrightarrow a) + t(w \leftrightarrow b)} \times \#(a \text{ in } `a\ b') \times \#(w \text{ in } `w\ x')$$

$$= \frac{1/4}{1/4 + 1/4} \times 1 \times 1 = 1/2$$

“counts”

a b ↔ ***w x***	a	b	c	d
w	1/2	1/2	0	0
x	1/2	1/2	0	0
y	0	0	0	0
z	0	0	0	0

b c d ↔ ***x y z***	a	b	c	d
w	0	0	0	0
x	0	1/3	1/3	1/3
y	0	1/3	1/3	1/3
z	0	1/3	1/3	1/3

Revised probability: example

$$t_{revised}(a \leftarrow\rightarrow w)$$

$$= \frac{1/2}{(1/2+1/2+0+0)_{(a\ b)\leftarrow\rightarrow(w\ x)} + (0+0+0+0)_{(b\ c\ d)\leftarrow\rightarrow(x\ y\ z)}}$$

Revised probabilities table

	a	b	c	d
w	1/2	1/2	0	0
x	1/4	5/12	1/6	1/6
y	0	1/3	1/3	1/3
z	0	1/3	1/3	1/3

“revised counts”

a b ↔ ***w x***	a	b	c	d
w	1/2	3/8	0	0
x	1/2	5/8	0	0
y	0	0	0	0
z	0	0	0	0

b c d ↔ ***x y z***	a	b	c	d
w	0	0	0	0
x	0	5/9	1/3	1/3
y	0	2/9	1/3	1/3
z	0	2/9	1/3	1/3

Re-Revised probabilities table

	a	b	c	d
w	1/2	1/2	0	0
x	3/16	**85/144**	1/9	1/9
y	0	1/3	1/3	1/3
z	0	1/3	1/3	1/3

Continue until convergence; notice that (b,x) binding gets progressively stronger;
b=rabbits, x=lapins

Derivation of EM based Alignment Expressions

$$V_E = \text{vocalbulary of language } L_1 \text{ (Say English)}$$

$$V_F = \text{vocabulary of language } L_2 \text{ (Say Hindi)}$$

E¹ *what is in a name ?*

नाम में क्या है ?

naam meM kya hai ?

F¹

name in what is ?

E² *That which we call rose, by any other name will smell as sweet.*

जिसे हम गुलाब कहते हैं, किसी और नाम से पुकारने पर भी उसकी खुशबू समान मीठा होगी

F² *Jise hum gulab kahte hai, aur bhi kisi naam se uski khushbu samaan mitha hogii*

That which we rose say , any other name by its smell as sweet

That which we call rose, by any other name will smell as sweet.

Vocabulary mapping

Vocabulary

V_E	V_F
what , is , in, a , name , that, which, we , call ,rose, by, any, other, will, smell, as, sweet	*naam, meM, kya, hai, jise, ham, gulab, kahte, aur, bhi, kisi, bhi, uski, khushbu, saman, mitha, hogii*

Key Notations

English vocabulary : V_E
French vocabulary : V_F
No. of observations / sentence pairs : S
Data D which consists of S observations looks like,

$$e^1{}_1, e^1{}_2, \dots, e^1{}_{l^1} \Leftrightarrow f^1{}_1, f^1{}_2, \dots, f^1{}_{m^1}$$
$$e^2{}_1, e^2{}_2, \dots, e^2{}_{l^2} \Leftrightarrow f^2{}_1, f^2{}_2, \dots, f^2{}_{m^2}$$
.....
$$e^s{}_1, e^s{}_2, \dots, e^s{}_{l^s} \Leftrightarrow f^s{}_1, f^s{}_2, \dots, f^s{}_{m^s}$$
.....
$$e^S{}_1, e^S{}_2, \dots, e^S{}_{l^S} \Leftrightarrow f^S{}_1, f^S{}_2, \dots, f^S{}_{m^S}$$

No. words on English side in s^{th} sentence : l^s
No. words on French side in s^{th} sentence : m^s
$index_E(e^s{}_p)$ =Index of English word $e^s{}_p$in English vocabulary/dictionary
$index_F(f^s{}_q)$ =Index of French word $f^s{}_q$in French vocabulary/dictionary

(Thanks to Sachin Pawar for helping with the maths formulae processing)

Hidden variables and parameters

Hidden Variables (Z) :
Total no. of hidden variables $= \sum_{s=1}^{S} l^s \, m^s$ where each hidden variable is as follows:
$z_{pq}^{s} = 1$, if in s^{th} sentence, p^{th} English word is mapped to q^{th} French word.
$z_{pq}^{s} = 0$, otherwise

Parameters (*Θ*) :
Total no. of parameters $= |\, V_E| \times |V_F|$, where each parameter is as follows:
$P_{i,j}$ = Probability that i^{th} word in English vocabulary is mapped to j^{th} word in Hindi vocabulary

Likelihoods

Data Likelihood *L(D; Θ)* :

$$L(D;\Theta) = \prod_{s=1}^{S}\prod_{p=1}^{l^s}\prod_{q=1}^{m^s}\left(P_{index_E(e_p^s),index_F(f_q^s)}\right)^{z_{pq}^s}$$

Data Log-Likelihood LL(D; Θ) :

$$LL(D;\Theta) = \sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} z_{pq}^s \log\left(P_{index_E(e_p^s),index_F(f_q^s)}\right)$$

Expected value of Data Log-Likelihood E(LL(D; Θ)) :

$$E(LL(D;\Theta)) = \sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} E(z_{pq}^s) \log\left(P_{index_E(e_p^s),index_F(f_q^s)}\right)$$

Constraint and Lagrangian

$$\sum_{j=1}^{|V_F|} P_{i,j} = 1 \ , \forall i$$

$$\sum_{s=1}^{S} \sum_{p=1}^{l^s} \sum_{q=1}^{m^s} E(z_{pq}^s) \, log\left(P_{index_E(e_p^s), index_F(f_q^s)}\right) - \sum_{i=1}^{|V_E|} \lambda_i \left(\sum_{j=1}^{|V_F|} P_{i,j} - 1 \right)$$

P

P_{ij} is “asymmetric” in the sense that the dictionary mapping is obtained by “looking” from the English side, i^{th} English word mapping to SOME HIndi word; we can “look” from the Hindi side too; Then we take the average of P_{ij} and P_{ji}

Aligners like GIZA++, Moses, Berkley etc. do this

Differentiating wrt *P_{ij}*

$$\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} \delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j}\left(\frac{E(z_{pq}^s)}{P_{i,j}}\right) - \lambda_i = 0$$

$$P_{i,j} = \frac{1}{\lambda_i}\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} \delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j}\,E(z_{pq}^s)$$

$$\sum_{j=1}^{|V_F|} P_{i,j} = 1 = \sum_{j=1}^{|V_F|}\frac{1}{\lambda_i}\sum_{s=1}^{S}\sum_{p=1}^{l^s}\sum_{q=1}^{m^s} \delta_{index_E(e_p^s),i}\,\delta_{index_F(f_q^s),j}\,E(z_{pq}^s)$$

Final E and M steps

M-step

$$P_{i,j} = \frac{\sum_{s=1}^{S} \sum_{p=1}^{l^s} \sum_{q=1}^{m^s} \delta_{index_E(e_p^s),i}\, \delta_{index_F(f_q^s),j} E(z_{pq}^s)}{\sum_{j=1}^{|V_F|} \sum_{s=1}^{S} \sum_{p=1}^{l^s} \sum_{q=1}^{m^s} \delta_{index_E(e_p^s),i}\, \delta_{index_F(f_q^s),j} E(z_{pq}^s)}, \forall i,j$$

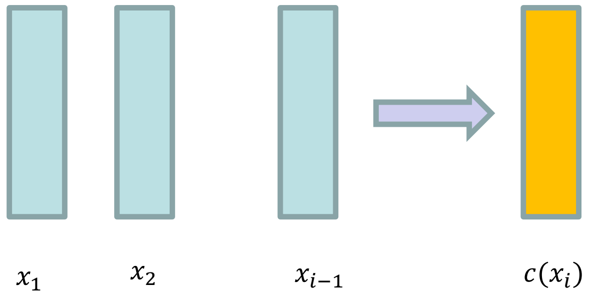
E-step

$$E(z_{pq}^s) = \frac{P_{index_E(e_p^s),index_F(f_q^s)}}{\sum_{q'=1}^{m^s} P_{index_E(e_p^s),index_F(f_{q'}^s)}}, \forall s,p,q$$

Back to RNN

Recurrent Neural Networks: two key Ideas

1. Summarize context information into a single vector



$$c(x_i) = F(x_1, x_2, \ldots, x_{i-1})$$

$$P(x_i | c(x_i))$$

Function G requires all context inputs at once

How does RNN address this problem?

Nature of $\boldsymbol{P}(.)$

n-gram LM: look-up table

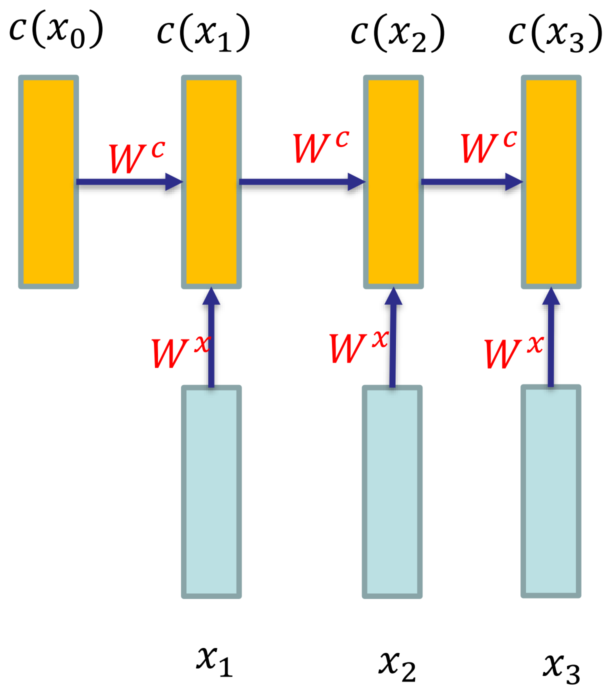
FF LM: $c(x_i)$ = $G(x_{i-1}, x_{i-2})$ (trigram LM)

RNN LM: $c(x_i)$ = $F(x_1, x_2, \ldots, x_{i-1})$ (unbounded context)

Two Key Ideas (cntd)

2. Recursively construct the context

$$c(x_i) = F(c(x_{i-1}), x_i)$$



We just need two inputs to construct the context vector:

- Context vector of previous timestep
- Current input

The context vector ➔ state/hidden state/contextual representation

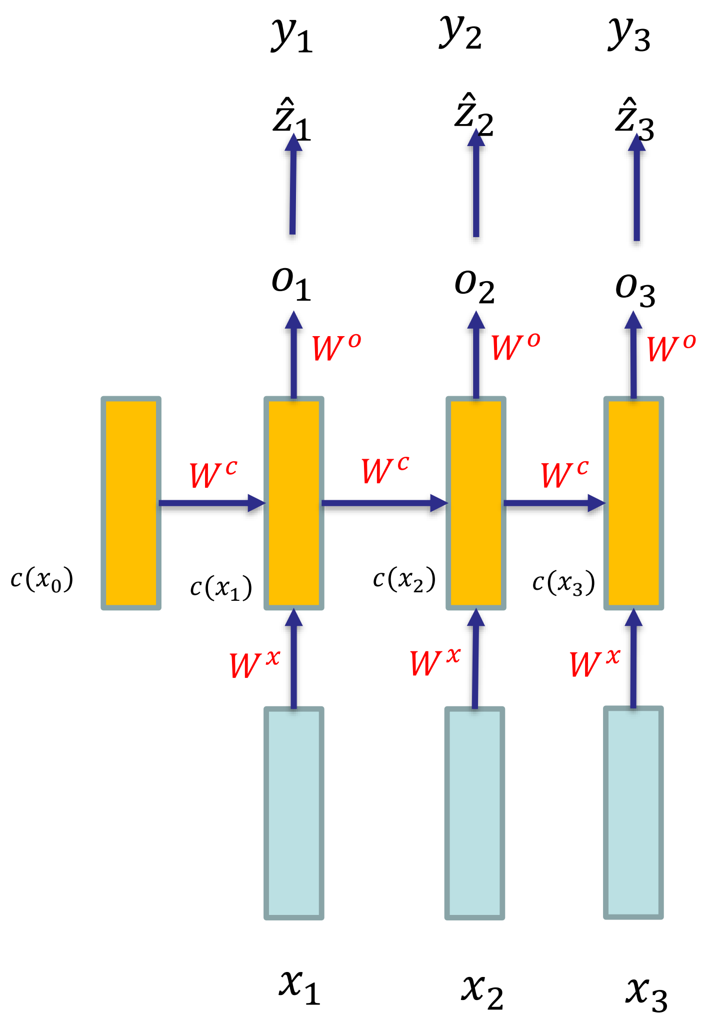
$F(.)$ can be implemented as

$$c(x_i) = \sigma(W^c c(x_{i-1}) + W^x x_i + b_1)$$

Like a feed-forward network

Generate output give the current input and state/context

W^o=wt. for output layer;
W^c= wt. for generating next state (context);
W^x= wt. for the input layer



$$o(x_i) = W^o c(x_i) + b_2$$

We are generally interested in categorical outputs

$$\hat{z}_i = softmax(o(x_i)) \\ = P(y_i | ctx(x_i))$$

$$\widehat{z_i^w} = P(y_i = w | ctx(x_i))$$

The same parameters are used at each time-step

Model size does not depend on sequence length

Long range context is modeled

Sequence Labelling Task

Input Sequence: $(x_1\ \ x_2\ \ x_3\ \ x_4 \ldots\ldots x_i \ldots\ldots x_N)$

Output Sequence: $(y_1\ \ y_2\ \ y_3\ \ y_4 \ldots\ldots y_i \ldots\ldots y_N)$

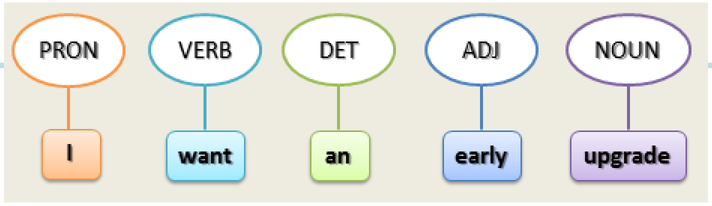
Input and output sequences have the same length

Variable length input

Output contains categorical labels

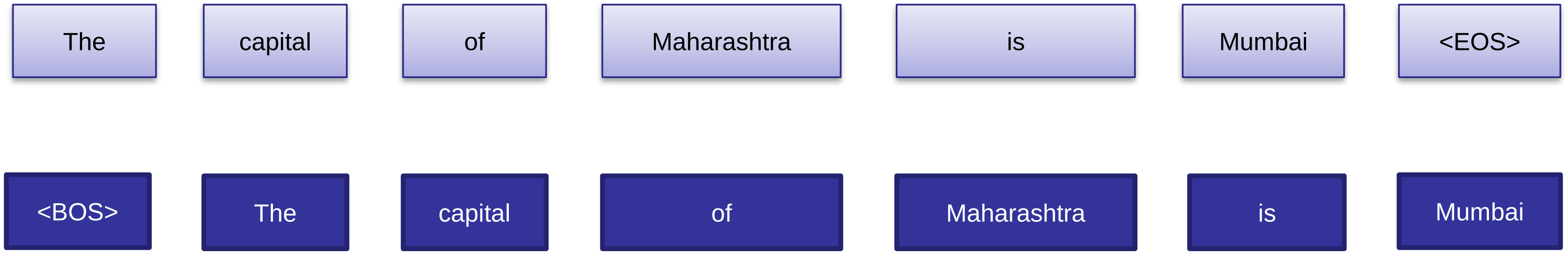
Output at any time-step typically depends on neighbouring output labels and input elements

Part-of-speech tagging



Recurrent Neural Network is a powerful model to learn sequence labelling tasks

How do we model language modeling as a sequence labeling task?



The output sequence is one-time step ahead of the input sequence

Training Language Models

Input: large monolingual corpus

- Each example is a tokenized sentence (sequence of words)
- At each time step, predict the distribution of the next word given all previous words
- Loss Function:
    - Minimize cross-entropy between actual distribution and predicted distribution
    - Equivalently, maximize the likelihood

At a single time-step:

$$J_i(\theta) = CE(z_i, \hat{z}_i) = -\sum_{w \in V} z_i^w \log \widehat{z_i^w} = -\log \widehat{z_i^L}$$

where $y_i = L$

Average over time steps for example n:

$$J^n(\theta) = \frac{1}{T}\sum_{i=1}^{T} J_i(\theta)$$

How do we learn model parameters?
More on that later!

Average over entire corpus:

$$J(\theta) = \frac{1}{N}\sum_{k=1}^{N} J^n(\theta)$$

Evaluating Language Models

How do we evaluate quality of language models?

↓

Evaluate the ability to predict the next word given a context

↓

Evaluate the probability of a testset of sentences

Standard test sets exist for evaluating language models: Penn Treebank, Billion Word Corpus, WikiText

Evaluating LM (cntd.)

- Ram likes to play -----
  - Cricket: high probability, low entropy, low perplexity (relatively very high frequency for ‘like to play cricket’)
  - violin: -do- (relatively high frequency possibility for ‘like to play violin’
  - Politics: moderate probability, moderate entropy, moderate perplexity (relatively moderate frequency ‘like to play politics’
  - milk: almost 0 probability, very high entropy, very high perplexity (relatively very low possibility for ‘like to play milk’

So an LM that predicts ‘milk’ is bad!

Language Model Perplexity

Perplexity: $\exp\big(J(\theta)\big)$

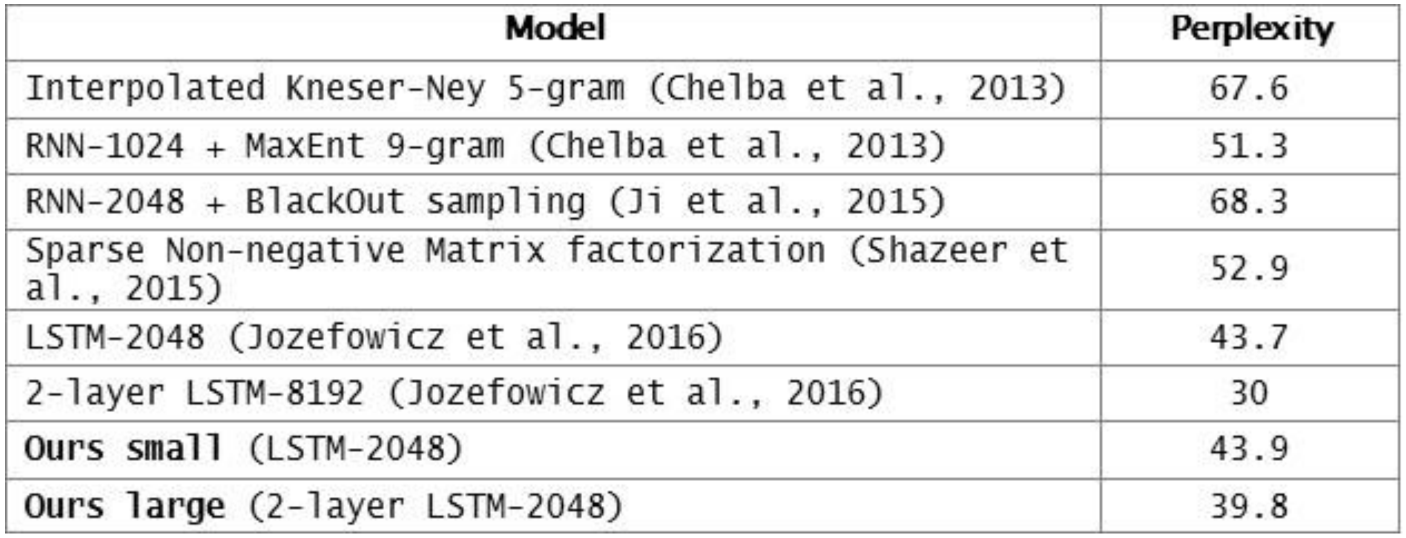
$J(\theta)$ is cross-entropy on the test set

Cross-entropy is measure of difference between actual and predicted distribution

Lower perplexity and cross-entropy is better

Training objective matches evaluation metric *n-gram*

RNN variants



Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

RNN models outperform n-gram models

A special kind of RNN network – LSTM- does even later ➔ we will see that soon

Appendix

Digression

Phrase Based SMT (PBSMT) and distortion

Governing equation

$$e_{best} = \arg\max_e P(e \mid f) = \arg\max_e [P(f \mid e) P_{LM}(e)]$$

where *e* and *f* have their usual meaning of output and input respectively; the translation with the highest score is e_{best}. $P(f|e)$ and $P_{LM}(e)$ are the translation model and language model, respectively.

Modelling *P(f|e)*

$$P(\overline{f}_1^I \mid \overline{e}_1^I) = P(\overline{f_1}, \overline{f_2}, \ldots, \overline{f_I} \mid \overline{e_1}, \overline{e_2}, \ldots, \overline{e_I})$$

$$= \prod_{i=1}^{I} \Phi(\overline{f_i} \mid \overline{e_i})\, d(start_i - end_{i-1} - 1)$$

LHS is the probability of sequence of *I* phrases in the sentence *f*, given *I* phrases in sentence *e*. *Φ* is called the *phrase translation probability* and *d(.)* is the *distortion probability.*

Distortion Probability

- $d(start_i - end_{i-1} - 1)$
- $start_i$: starting position of the translation of the i^{th} phrase of *e* in *f*
- end_{i-1}: end position of the translation of the $(i-1)^{th}$ phrase of *e* in *f*

- The quantity $start_i - end_{i-1} - 1$ is a measure of the distance between the translation of i^{th} phrase and the translation of the $(i-1)^{th}$ phrase of *e* as they appear as the end^{th} and $start^{th}$ phrase in *f*.

- It is, thus, also a measure of the *reordering* of phrases induced by the translation.

Need for phrases (linguistic phrases and non-linguistic "phrases")

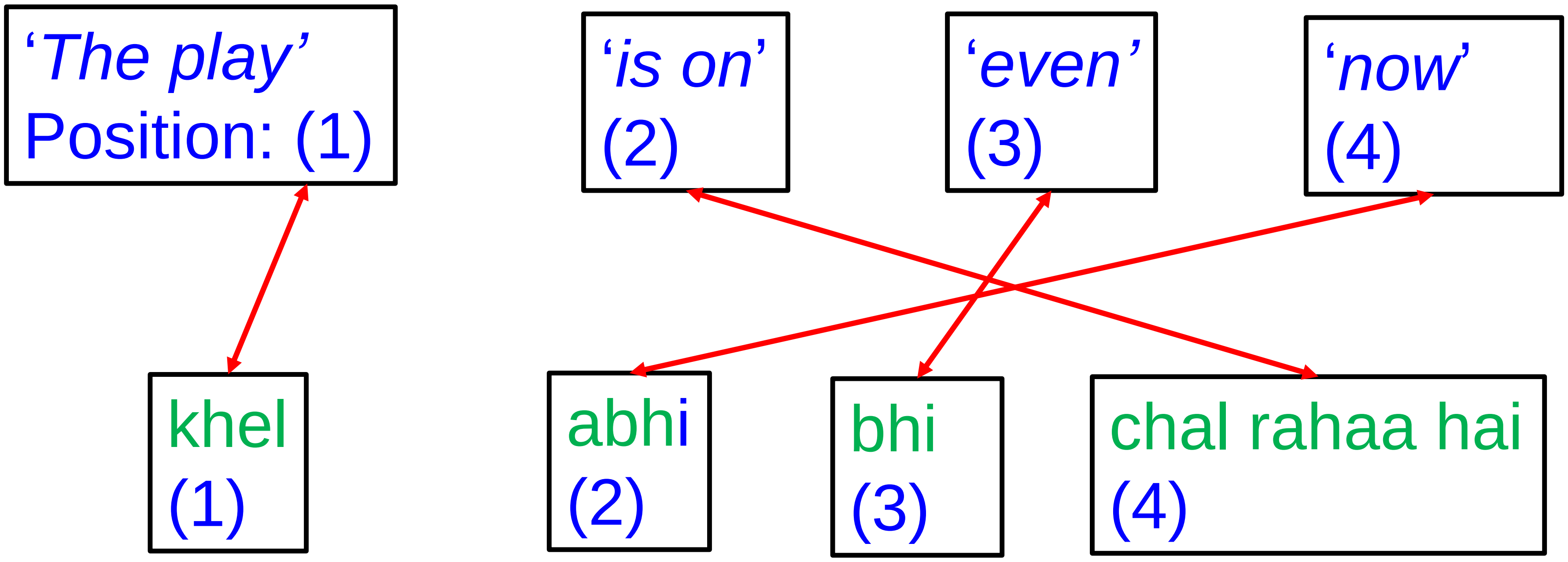
- *"The play is on"* ←→ *"khel chal rahaa hai"*
- *"is on"* ←→ *"chal rahaa hai"*
- IMP: treat '*is*' and '*on*' together and NOT separately

- Otherwise, '*on*' might map to '*rahaa*' which will take away some probability mass of '*on*' onto Hindi word mappings like 'on' ←→ {'par', 'upar', …}
- May produce non-fluent translations like

  "*the book is on the table*" ←→ "*kitaab mej rahaa hai*" instead of "*kitaab mej par hai*"

Back to distortion

- *"The play is on even now"* ←→ *"khel abhii bhii chal rahaa hai"*
- Mappings:



'The play' Position: (1)
'is on' (2)
'even' (3)
'now' (4)
khel (1)
abhi (2)
bhi (3)
chal rahaa hai (4)