Introduction to Attention Models

Dr. Dileep A. D.

Associate Professor,
Multimedia Analytics Networks And Systems (MANAS) Lab,
School of Computing and Electrical Engineering (SCEE),
Indian Institute of Technology Mandi, Kamand, H.P.
Email: addileep@iitmandi.ac.in

Agenda

- Neural network models
  - Fully connected neural networks
  - Convolutional neural networks
  - Recurrent neural networks
- Introduction to encoder-decoder models
- Attention Mechanism
  - Attention models in vision

2

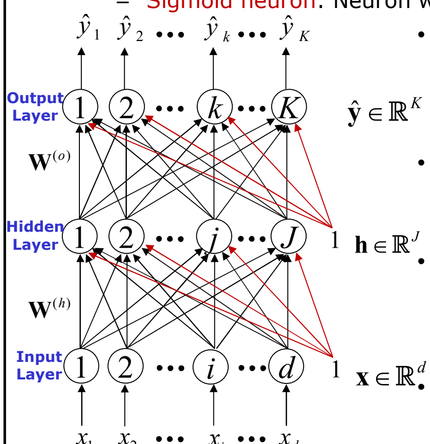
Artificial Neural Networks

- Learning method:
  - Error correction learning (Backpropagation algorithm [1])
- Structure of network:
  - Feedforward neural networks
    - Fully Connected Neural Network (FCNN),
    - Convolutional Neural Networks (CNN),
    - Auto Encoders
  - Feedback neural networks
    - Recurrent Neural Networks (RNN)
    - Long Short Term Memory (LSTM)
  - Feedforward and feedback neural networks
    - Bidirectional LSTM
    - Self Organizing Maps (SOM)

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, Parallel Distributed Processing, volume 1, pages 318-362. MIT Press, 1986.

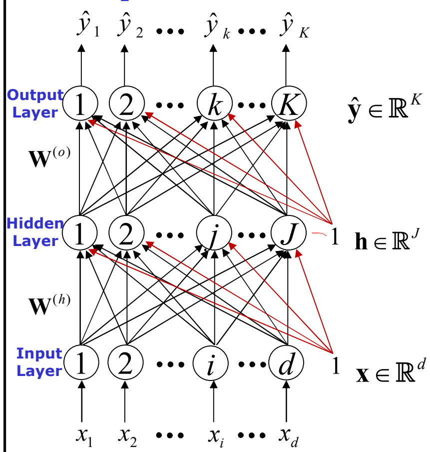
Fully Connected Neural Network (FCNN)

- Architecture of an FCNN:
- Input layer: Linear neurons
  - Linear neuron: When an input is given to a neuron and the same input comes out as output
- Hidden layers (1 or 2 or more): Sigmoidal neurons or ReLU
  - Sigmoid neuron: Neuron with sigmoid activation function



- Output layer: Sigmoidal/Softmax neurons (for pattern classification task) or Linear neuron (for regression)
- Number of neurons in input layer (d): Dimension of the data vector (number of input variables)
- Number of neurons in the output layer (K): Number of classes in classification or number of output variables
- Number of layers and neurons in each of the hidden layers are decided experimentally

Fully Connected Neural Network (FCNN)



$$\mathbf{h} = g\left(\mathbf{W}^{(h)\top}\mathbf{x} + \mathbf{w}_0^{(h)}\right)$$

$g()$ is sigmoid or ReLU

$$\hat{\mathbf{y}} = f\left(\mathbf{W}^{(o)\top}\mathbf{h} + \mathbf{w}_0^{(o)}\right)$$

$f()$ is sigmoid/softmax or Linear

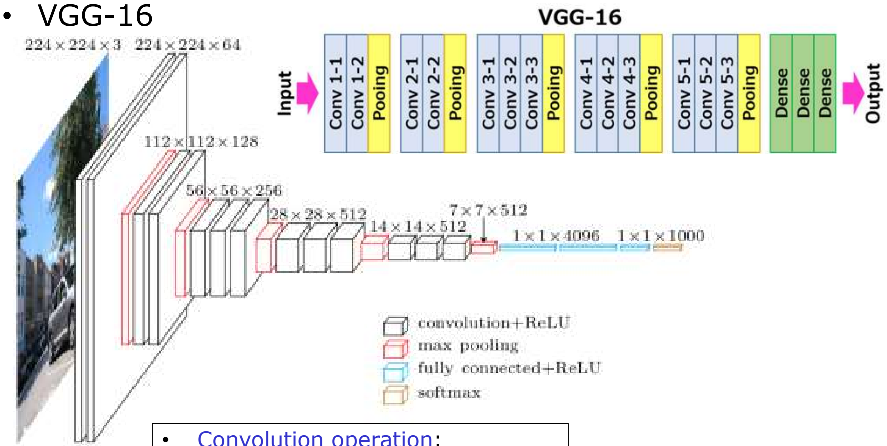
- Weights associated with all the connection between the neurons indicate the parameter of the complex nonlinear discriminant function that the network is trying to approximate
- We train the FCNN using backpropagation
  - Need to compute the gradient of loss function with respect to weight parameters

Convolutional Neural Networks (CNN)

- CNN learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier
  - The connections are much sparser
  - Weight sharing
- A CNN can be implemented as a feed-forward neural network
  - Only a few weights are active
  - Rest of the weights are zero
- Each hidden layer is the resultant of convolution operation on the previous layer
  - Rectified linear function (ReLU) is used as activation function on the out put of convolution operation
- It has alternate convolution and pooling layers
- CNN is using backpropagation by considering it as a feed-forward neural network with sparse connection and weight sharing

VGGNet [2]

- VGG-16



- Convolution operation:
  - Kernel size is 3 × 3 throughout
  - Stride is 1 and padding is 1
- Pooling operation:
  - Filter size is 2 x 2 and stride is 2

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014

CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11		

CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	

CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13

CNN as FFNN

1	2	3	4
5	**6**	7	8
9	**10**	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14		

CNN as FFNN

1	2	3	4
5	**6**	**7**	8
9	**10**	**11**	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14	h15	

CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14	h15	h16

CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14	h15	h16
h17		

CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

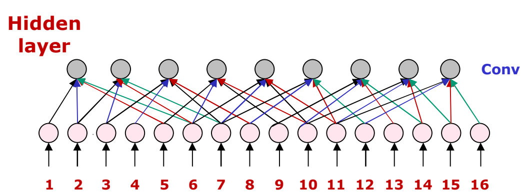
h11	h12	h13
h14	h15	h16
h17	h18	

CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.



h11	h12	h13
h14	h15	h16
h17	h18	h19

CNN as FFNN

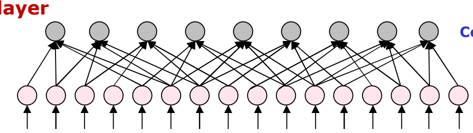
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14	h15	h16
h17	h18	h19

Hidden layer



h11	h12	h13
h14	h15	h16
h17	h18	h19

pooling

h11	

17

CNN as FFNN

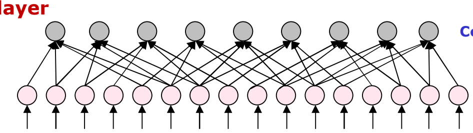
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14	h15	h16
h17	h18	h19

Hidden layer



h11	h12	h13
h14	h15	h16
h17	h18	h19

pooling

h11	h16

18

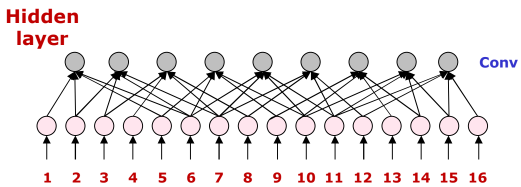
CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14	h15	h16
h17	h18	h19



h11	h12	h13
h14	h15	h16
h17	h18	h19

pooling

h11	h16
h17	

19

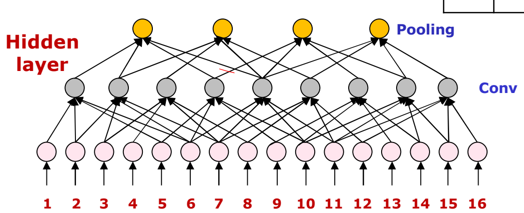
CNN as FFNN

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

h11	h12	h13
h14	h15	h16
h17	h18	h19



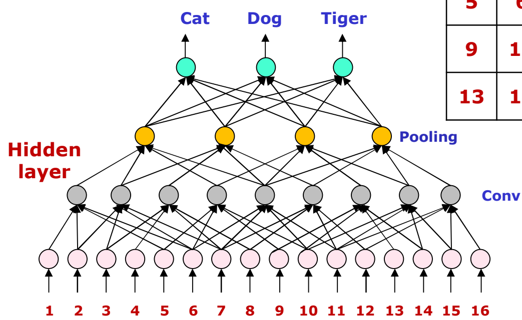
h11	h12	h13
h14	h15	h16
h17	h18	h19

pooling

h11	h16
h17	h19

20

CNN as FFNN



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

.	.
.	.

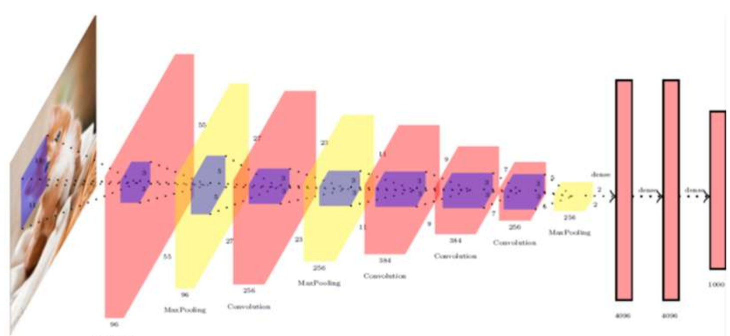
h11	h12	h13
h14	h15	h16
h17	h18	h19

h11	h12	h13
h14	h15	h16
h17	h18	h19

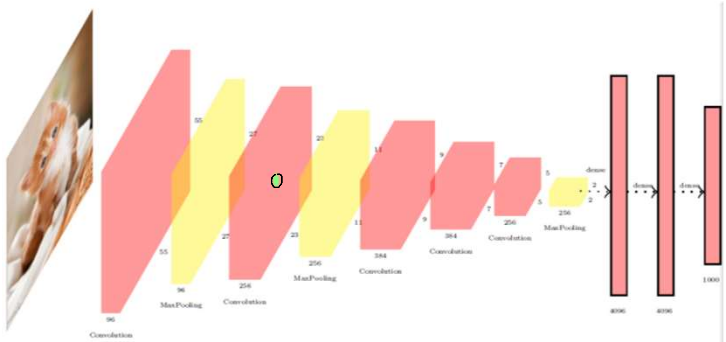
pooling

h11	h16
h17	h19

Visualizing Patches which Maximally Activate a Neuron



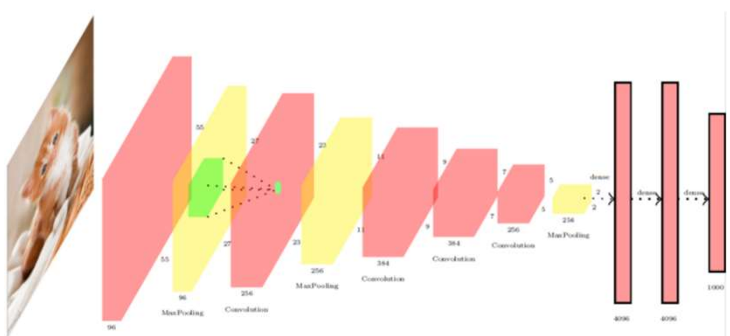
Visualizing Patches which Maximally Activate a Neuron



- Consider some neurons in a given layer of a CNN
- Feed in images to this CNN and identify the images which cause these neurons to fire

23

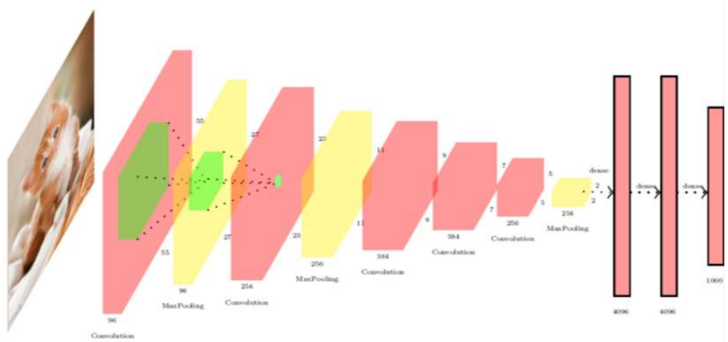
Visualizing Patches which Maximally Activate a Neuron



- Consider some neurons in a given layer of a CNN
- Feed in images to this CNN and identify the images which cause these neurons to fire
- Then trace back to the patch in the image which causes these neurons to fire

24

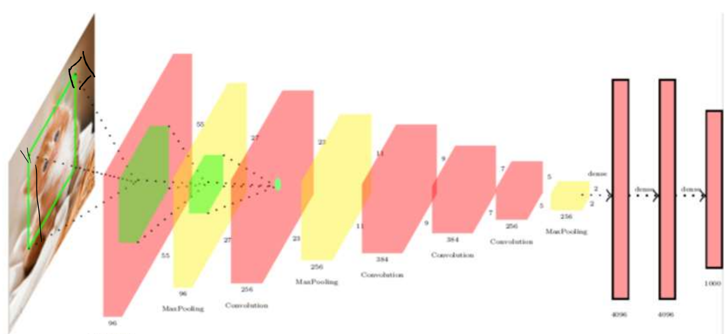
Visualizing Patches which Maximally Activate a Neuron



- Consider some neurons in a given layer of a CNN
- Feed in images to this CNN and identify the images which cause these neurons to fire
- Then trace back to the patch in the image which causes these neurons to fire

25

Visualizing Patches which Maximally Activate a Neuron



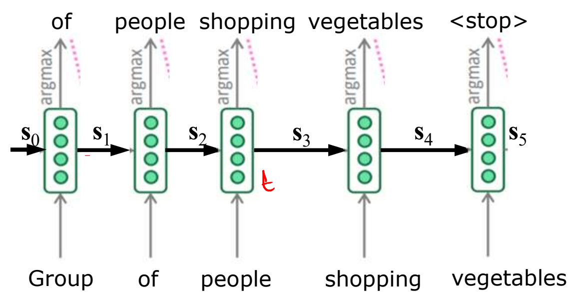
- Consider some neurons in a given layer of a CNN
- Feed in images to this CNN and identify the images which cause these neurons to fire
- Then trace back to the patch in the image which causes these neurons to fire

26

Recurrent Neural Networks (RNN)

- RNN used for sequential learning problem
  - Each input is dependent on the previous or future input
  - In many applications the input is not of a fixed size
- Consider the problem of language modelling: Natural sentence generation
  - Given $t-i$ words predict the t^{th} word

- Example: Generate a sentence – "Group of people shopping vegetables"
- A word *shopping* is predicted given the words *Group*, *of*, *people*

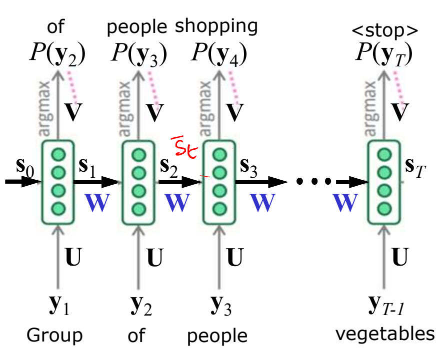


- $\mathbf{s}_t$ is the state of the network at time step t

27

Sequence Learning Problem: RNN

- Sequence learning: More formally, given $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}$ we want to find $\hat{\mathbf{y}} = \arg\max\ P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1})$
  - where $j \in \mathcal{V}$ and $\mathcal{V}$ is the set of all the words in vocabulary
- Let us denote $P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1})$ as $P(\mathbf{y}_t = j \mid (\mathbf{y})_1^{t-1})$

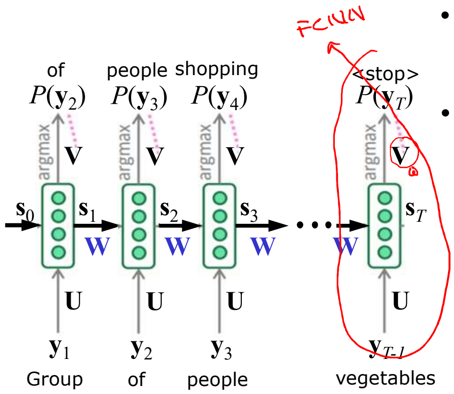


- Using RNN:
  $P(\mathbf{y}_t = j \mid (\mathbf{y})_1^{t-1}) = \mathrm{softmax}(\mathbf{V}\mathbf{s}_t + c)_j$
- $\mathbf{s}_t$ is the hidden representation at time step t
- Recurrent connections ensure that information about sequence $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}$ is embedded in $\mathbf{s}_t$
- Hence,
  $P(\mathbf{y}_t = j \mid (\mathbf{y})_1^{t-1}) = P(\mathbf{y}_t = j \mid \mathbf{s}_t)$

28

Sequence Learning Problem: RNN

- Sequence learning: More formally, given $\mathbf{y}_1$, $\mathbf{y}_2$, ..., $\mathbf{y}_{t-1}$ we want to find $\hat{\mathbf{y}} = \arg\max\ P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1})$
  - where $j \in \mathcal{V}$ and $\mathcal{V}$ is the set of all the words in vocabulary
- Let us denote $P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1})$ as $P(\mathbf{y}_t = j \mid (\mathbf{y})_1^{t-1})$

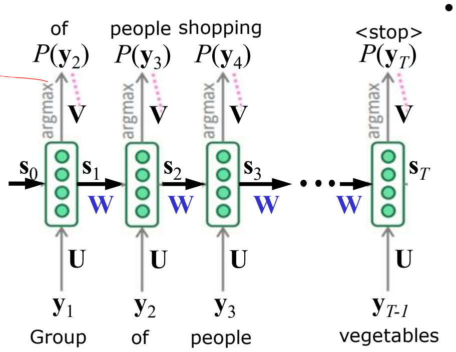


- Using RNN:
  $P(\mathbf{y}_t = j \mid \mathbf{s}_t) = \text{softmax}(\mathbf{V}\mathbf{s}_t + c)_j$
- Recurrent connections ensure that information about sequence $\mathbf{y}_1$, $\mathbf{y}_2$, ..., $\mathbf{y}_{t-1}$ is embedded in $\mathbf{s}_t$
  $\mathbf{s}_t = sigmoid(\mathbf{U}\mathbf{y}_t + \mathbf{W}\mathbf{s}_{t-1} + b)$
  $\mathbf{s}_t = RNN(\mathbf{s}_{t-1}, \mathbf{y}_t)$

29

Language Modelling Problem: Natural Sentence Generation : RNN

- Data: All sentences from any large corpus (say Wikipedia)
- Each word in the vocabulary is represented as d-dimensional word vector (example: word-to-vec)
- RNN is trained using backpropagation through time (BPTT)

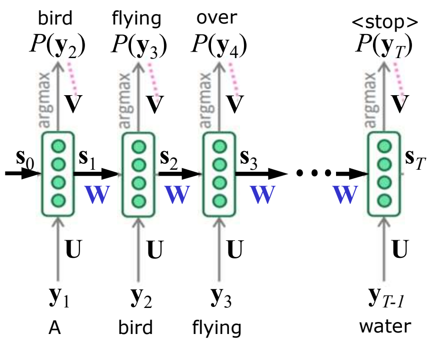


- One can also use LSTM or GRU in the place of RNN

30

Neural Image Caption Generation

- So far we have seen how to generate a sentence given previous words
- Now, we want to generate a sentence given an image

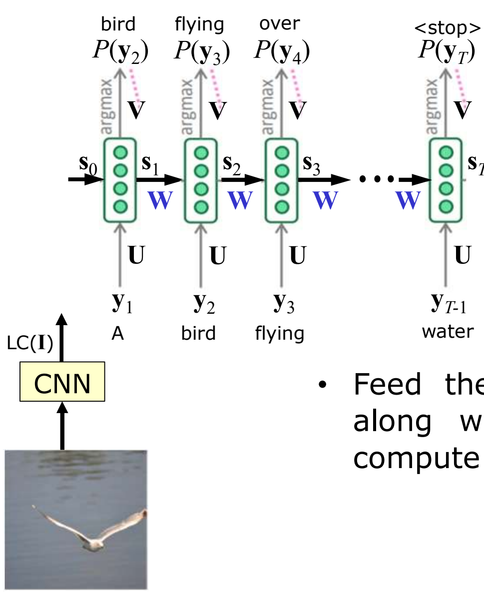


- We are now interested in $P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}, \mathbf{I})$ instead of $P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1})$
  - where $\mathbf{I}$ is an image
- Usually information in the image is encoded in a feature vector

A bird flying over a body of water

31

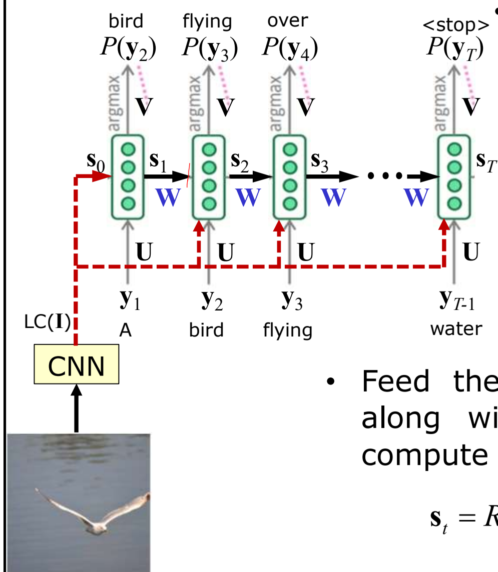
Neural Image Caption Generation



- We now model $P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}, \mathbf{I})$ as $P(\mathbf{y}_t = j \mid \mathbf{s}_t, \text{LC}(\mathbf{I}))$
  - where $\mathbf{I}$ is an image
  - LC($\mathbf{I}$) is the representation obtained from the last convolution layer of CNN
- Feed the LC($\mathbf{I}$) at every time step along with word representation to compute $\mathbf{s}_t$

32

Neural Image Caption Generation



- We now model $P(\mathbf{y}_t = j \mid \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}, \mathbf{I})$ as $P(\mathbf{y}_t = j \mid \mathbf{s}_t, \mathrm{LC}(\mathbf{I}))$
  - where $\mathbf{I}$ is an image
  - LC($\mathbf{I}$) is the representation obtained from the last convolution layer of CNN

- Feed the LC($\mathbf{I}$) at every time step along with word representation to compute $\mathbf{s}_t$

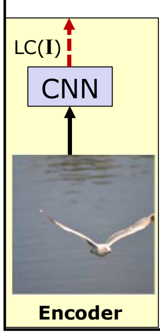
vector. repn of word at time t

$$\mathbf{s}_t = RNN(\mathbf{s}_{t-1}, [\mathbf{y}_t, \mathrm{LC}(\mathbf{I})])$$

33

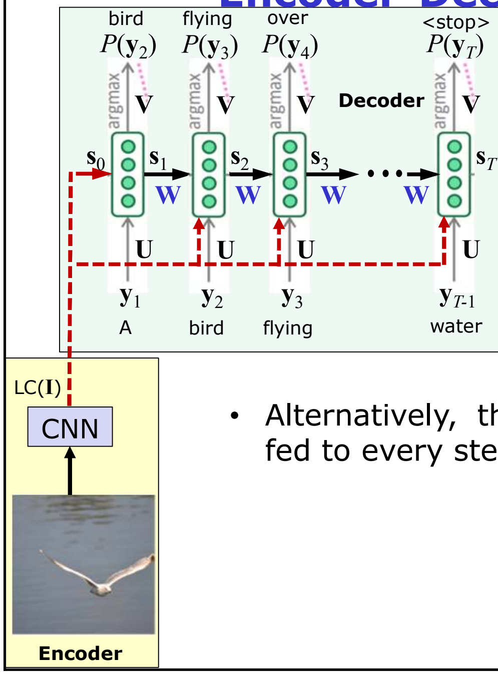
Encoder-Decoder Model

- A CNN is first used to encode the image



34

Encoder-Decoder Model



- A CNN is first used to encode the image
- A RNN is then used to decode (generate) a sentence from the encoding
- Both the encoder and decoder use a neural network

- Alternatively, the encoder's output can be fed to every step of the decoder

35

More Applications of Encoder-Decoder Models

- Machine Translation:
  - Translating sentence in one language to another
  - Encoder: RNN
  - Decoder: RNN
- Transliteration:
  - Translating the script of one language to script if another language
  - Encoder: RNN
  - Decoder: RNN
- Image Question Answering:
  - Given the image and a question (sentence), generate answer (word)
  - Encoder: CNN + RNN
  - Decoder: FCNN

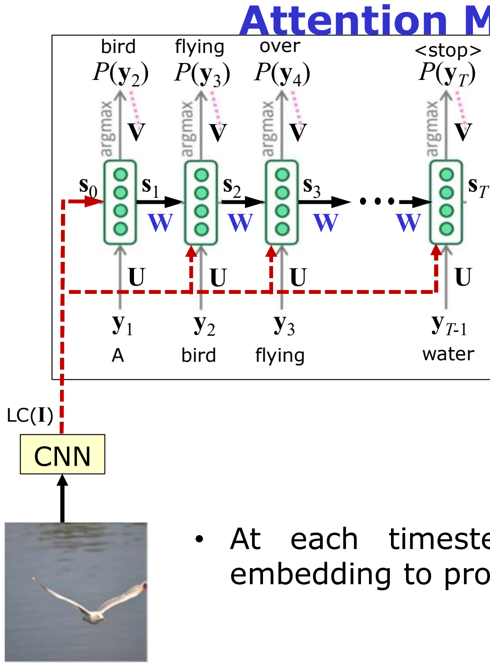
36

More Applications of Encoder-Decoder Models

- Document Summarization:
  - Generating a summery of a document
  - Encoder: RNN
  - Decoder: RNN
- Video Captioning:
  - Generate sentence given video
  - Encoder: CNN-RNN
  - Decoder: RNN
- And many more …

37

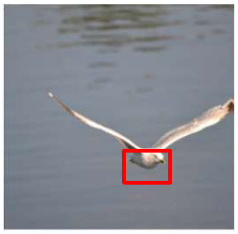
Attention Mechanism



- Encoder-decoder models can be made even more expressive by adding an "attention" mechanism
- Let us motivate the task of attention with the help of image captioning
- The encoder reads the image only once and encodes it
  - Embedding from last convolution layer of CNN
- At each timestep the decoder uses this embedding to produce a new word

38

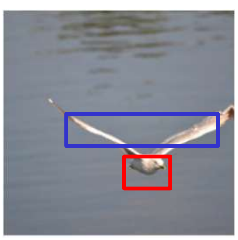
Attention Mechanism: Image Captioning



- Humans try to produce each word in the output by focusing only on certain objects (concepts) in the input image
- Example:

A bird flying over a body of water

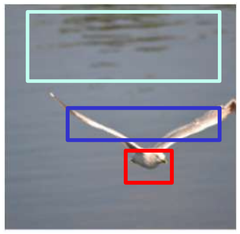
Attention Mechanism: Image Captioning



- Humans try to produce each word in the output by focusing only on certain objects (concepts) in the input image
- Example:

A bird flying over a body of water

Attention Mechanism: Image Captioning



- Humans try to produce each word in the output by focusing only on certain objects (concepts) in the input image
- Example:
- Essentially at each time step we come up with a distribution (weights) on the input concepts (objects)

A bird flying over a body of water

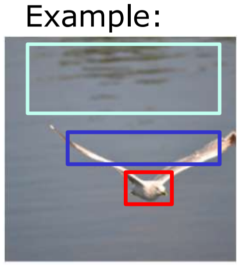
- This distribution tells us how much attention to pay to each objects in input at each time step
- Ideally, at each time step we should feed only this relevant information (i.e. encodings of relevant objects) to the decoder

- t_1: A [1 0 0]
- 0.5 0.5
- t_2: bird [1 0 0]
- t_3: flying [0 1 0]
- t_4: over [0 1 0]
- t_5: a [0 0 1]
- t_6: body [0 0 1]
- t_7: of [0 0 1]
- t_8: water [0 0 1]

41

Attention Mechanism: Image Captioning

- Humans try to produce each word in the output by focusing only on certain objects (concepts) in the input image
- Example:

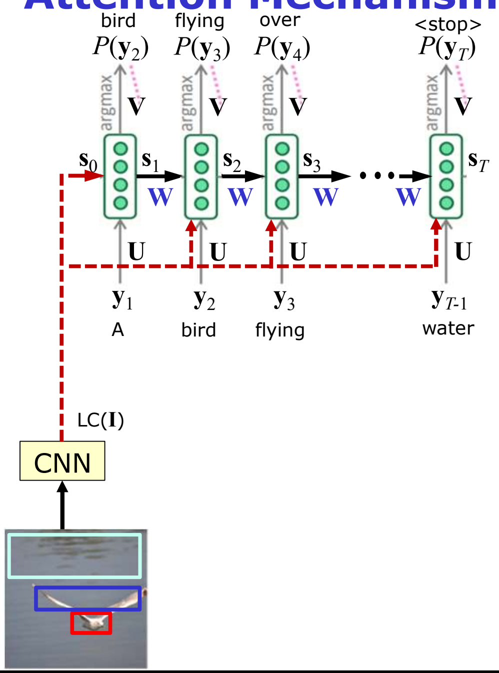


A bird flying over a body of water

A group of people sitting on a boat in the water

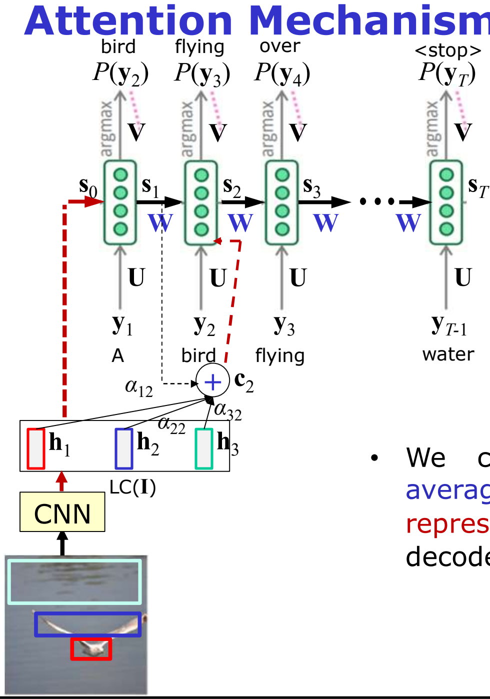
42

Attention Mechanism: Image Captioning



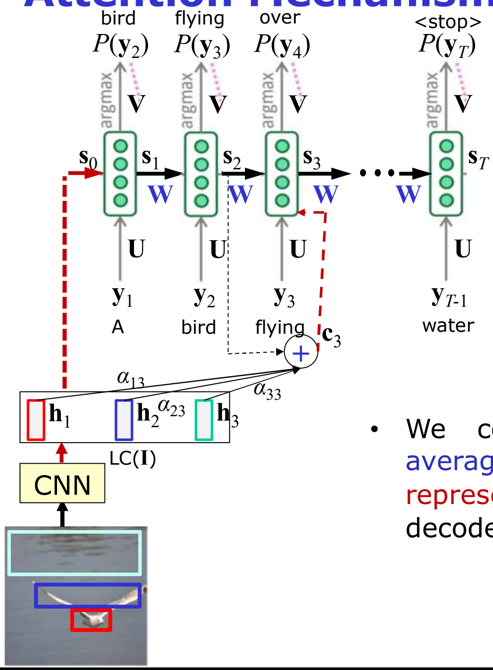
- Let us revisit the decoder that we have seen so far
- Entire image is encoded into a vector representation
- We feed this encoded representation to decoder at each time step
- Now suppose an oracle told you which location in image to focus on at a given time step t

Attention Mechanism: Image Captioning



- Let us revisit the decoder that we have seen so far
- Entire image is encoded into a vector representation
- We feed this encoded representation to decoder at each time step
- Now suppose an oracle told you which location in image to focus on at a given time step t
- We could just take a weighted average of the corresponding location representations ($\mathbf{h}_j$) and feed it to the decoder

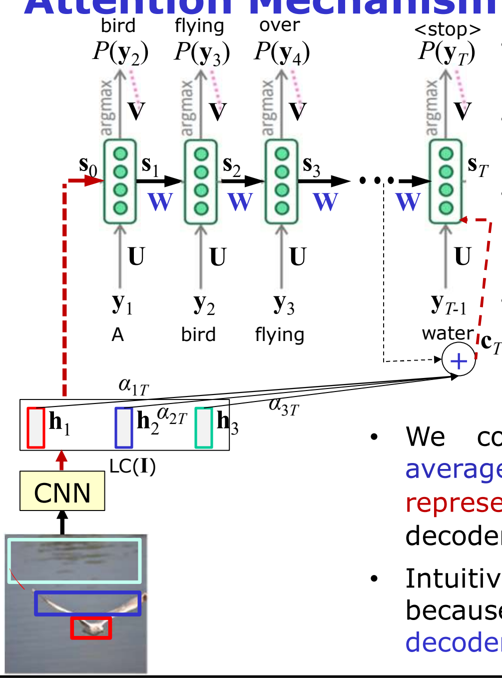
Attention Mechanism: Image Captioning



- Let us revisit the decoder that we have seen so far
- Entire image is encoded into a vector representation
- We feed this encoded representation to decoder at each time step
- Now suppose an oracle told you which location in image to focus on at a given time step t
- We could just take a weighted average of the corresponding location representations ($\mathbf{h}_j$) and feed it to the decoder

45

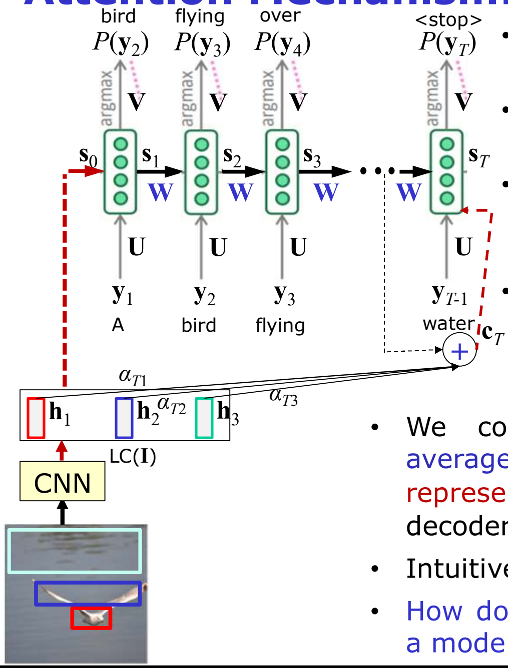
Attention Mechanism: Image Captioning



- Let us revisit the decoder that we have seen so far
- Entire image is encoded into a vector representation
- We feed this encoded representation to decoder at each time step
- Now suppose an oracle told you which location in image to focus on at a given time step t
- We could just take a weighted average of the corresponding location representations ($\mathbf{h}_j$) and feed it to the decoder
- Intuitively this should work better because we are not overloading the decoder with irrelevant information

46

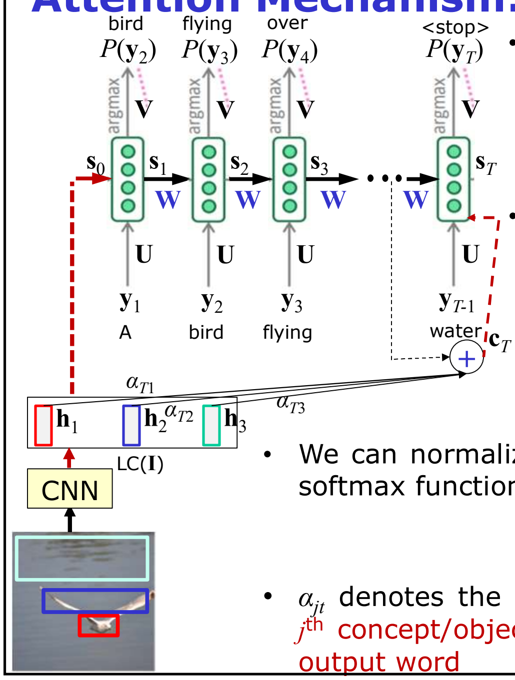
Attention Mechanism: Image Captioning



- Let us revisit the decoder that we have seen so far
- Entire image is encoded into a vector representation
- We feed this encoded representation to decoder at each time step
- Now suppose an oracle told you which location in image to focus on at a given time step t
- We could just take a weighted average of the corresponding location representations ($\mathbf{h}_j$) and feed it to the decoder
- Intuitively this should work better
- How do we convert this intuition into a model?

47

Attention Mechanism: Image Captioning



- In practice we will not have the information of how much importance
  - The machine will have to learn this from the data
- To enable this we define a function:

$$e_{jt} = f_{ATT}(\mathbf{s}_{t-1}, \mathbf{h}_j)$$

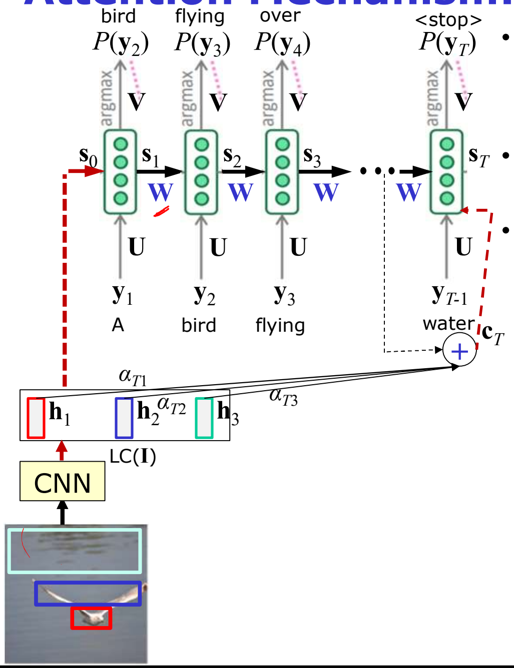
  - e_{jt} captures the importance of the j^{th} concept/object in image for decoding the t^{th} output word
- We can normalize these weights by using the softmax function

$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^{M} \exp(e_{jt})}$$

- α_{jt} denotes the probability of focusing on the j^{th} concept/object in image to produce the t^{th} output word

48

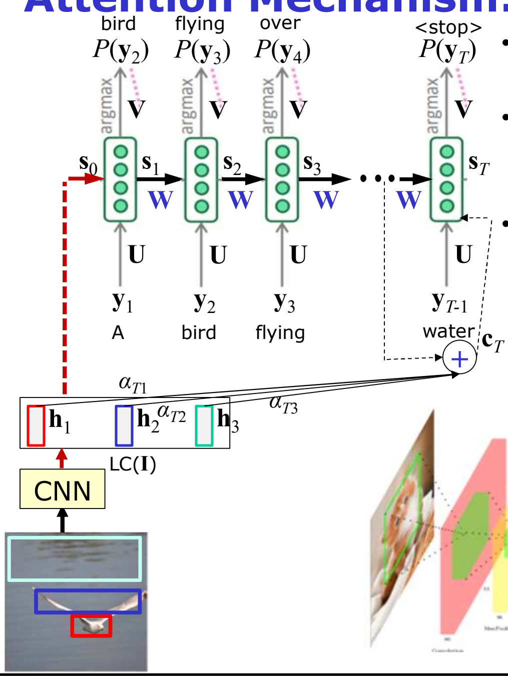
Attention Mechanism: Image Captioning



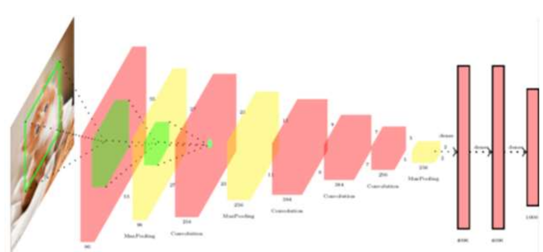
- The attention model $f_{ATT}(.)$ is thus given by
$$e_{jt} = f_{ATT}(\mathbf{s}_{t-1}, \mathbf{h}_j)$$
- This attention model is implemented using FCNN
- Input to the model is the concatenation of location representations ($\mathbf{h}_j$) in input image and previous hidden state ($\mathbf{s}_{t-1}$)

49

Attention Mechanism: Image Captioning



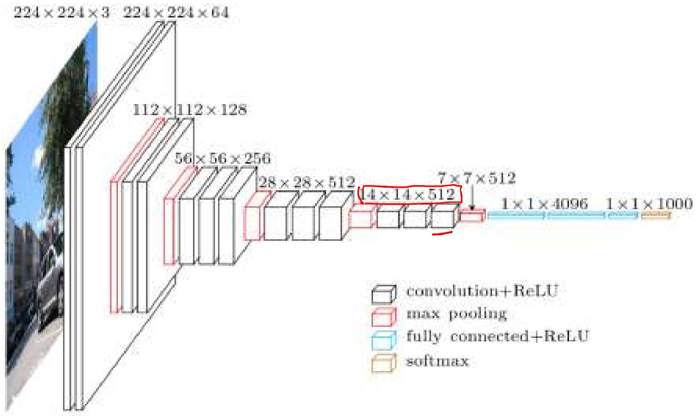
- How do we get the location information?
- Image is encoded into a last convolution layer representation
- Convolution representation has location information



50

Learning Attention Over Image Location

- Consider VGG16 network to encode image.

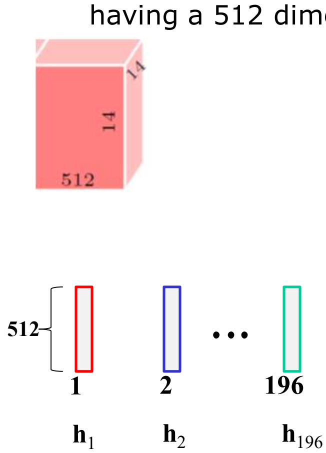


- Output of last convolution layer is a 14x14x512 feature map
- We could think of this as 196 (i.e. 14x14) locations (each having a 512 dimensional representation)

51

Learning Attention Over Image Location

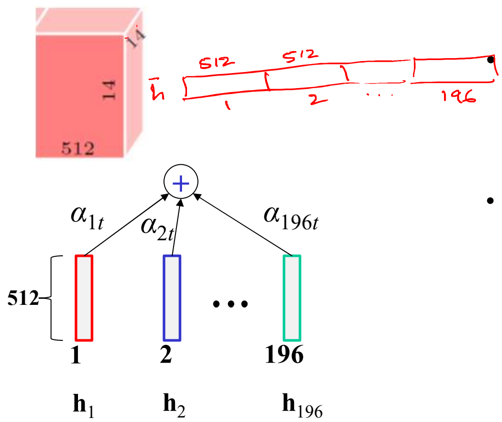
- Consider VGG16 network to encode image.
- Output of last convolution layer is a 14x14x512 feature map
- We could think of this as 196 (i.e. 14x14) locations (each having a 512 dimensional representation)



52

Learning Attention Over Image Location

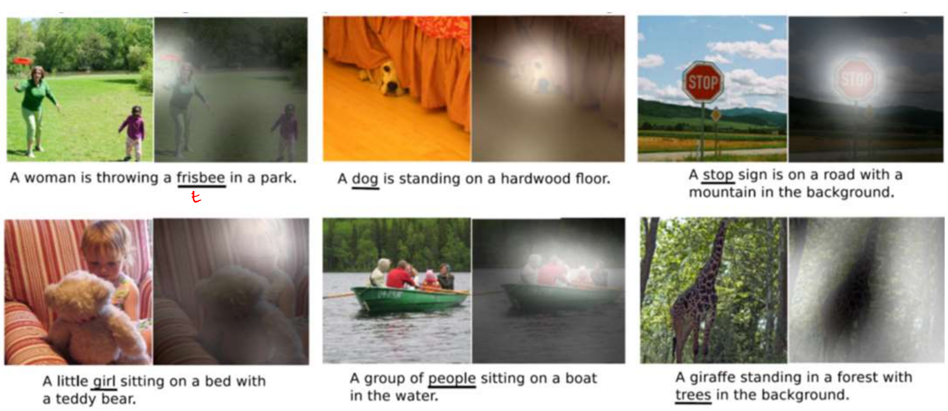
- Consider VGG16 network to encode image.
- Output of last convolution layer is a 14x14x512 feature map
- We could think of this as 196 (i.e. 14x14) locations (each having a 512 dimensional representation)



- α_{jt} denotes the amount of attention on the location (j^{th} location vector) in image to produce the t^{th} output word
- The model will then learn an attention over these locations (which in turn correspond to actual locations in the images)

53

Illustrations: Attention Over Images

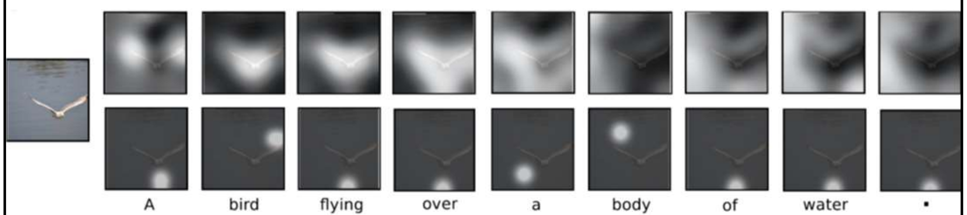


Examples of the attention-based model attending to the correct object (*white* indicates the attended regions, *underlines* indicates the corresponding word) [3]

[3] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, Yoshua Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR vol. 37, pp. 2048-2057, 2015.

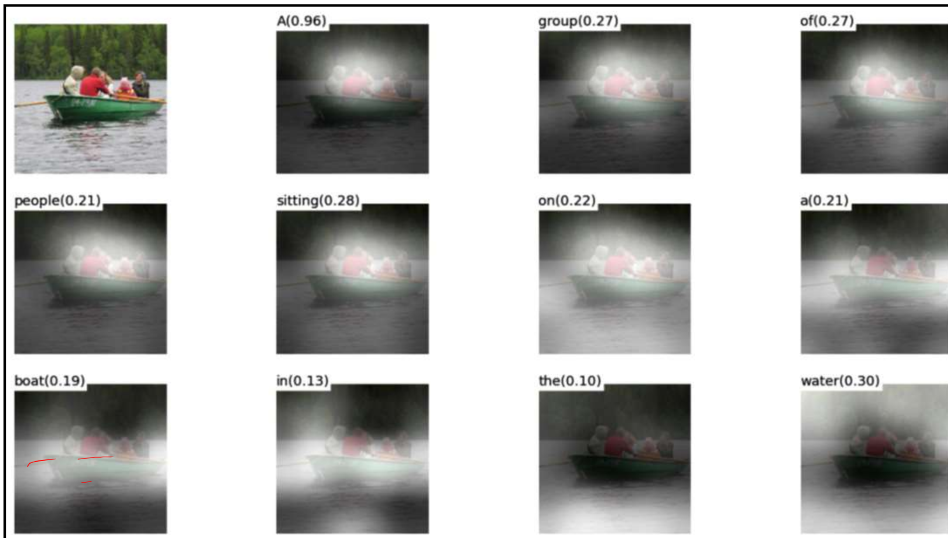
54

Illustrations: Attention Over Images



Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. (*white* indicates the attended regions) [3]

[3] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, Yoshua Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR vol. 37, pp. 2048-2057, 2015.



A group of people sitting on a boat in the water. [3]

[3] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, Yoshua Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR vol. 37, pp. 2048-2057, 2015.

Summary

- Attention mechanism in encoder-decoder models
- Encoder-decoder model
  - Encoder first used to encode the input
  - A decoder is then used to decode (generate) a output from the encoding
- Encoder-decoder models can be made even more expressive by adding an "attention" mechanism
- A model will then learn an attention over input to generate output
  - Attention is seen as probability of portion of input responsible for generating output

57

Text Books

1. Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep learning, MIT Press, Available online: http://www.deeplearningbook.org, 2016
2. Charu C. Aggarwal, Neural Networks and Deep Learning, Springer, 2018
3. B. Yegnanarayana, Artificial Neural Networks, Prentice-Hall of India, 1999.
4. Satish Kumar, Neural Networks - A Class Room Approach, Second Edition, Tata McGraw-Hill, 2013.
5. S. Haykin, Neural Networks and Learning Machines, Prentice Hall of India, 2010.
6. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
7. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Third Edition, Morgan Kaufmann Publishers, 2011.
8. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, 2009.

58