Survey on the attention based RNN model and its applications in computer vision

Feng Wang f.wang-6@student.tudelft.nl
Pattern Recognition Lab
EEMCS
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

D.M.J. Tax D.M.J.Tax@tudelft.nl
Pattern Recognition Lab
EEMCS
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Contents

Abstract

The recurrent neural networks (RNN) can be used to solve the sequence to sequence problem, where both the input and the output have sequential structures. Usually there are some implicit relations between the structures. However, it is hard for the common RNN model to fully explore the relations between the sequences. In this survey, we introduce some attention based RNN models which can focus on different parts of the input for each output item, in order to explore and take advantage of the implicit relations between the input and the output items. The different attention mechanisms are described in detail. We then introduce some applications in computer vision which apply the attention based RNN models. The superiority of the attention based RNN model is shown by the experimental results. At last some future research directions are given.

1. Introduction

In this section, we discuss the nature of attention based recurrent neural network (RNN) model. What does "attention" mean? What are the applications of the attention based model in computer vision area? What is the attention based RNN model? What are the attention mechanisms introduced in this survey? What are the advantages of using attention based RNN model? These are the questions we would like to answer in this section.

1.1 What does "attention" mean?

In psychology, limited by the processing bottlenecks, humans tend to selectively concentrate on a part of the information, and at the same time ignore other perceivable information. The above mechanism is usually called attention [3]. For example, in human visual processing, although human's eye has the ability to receive a large visual field, usually only a small part is fixated on. The reason is different areas of retina have different magnitude of processing ability, which is usually referred as acuity. And only a small area of the retina, fovea, has the greatest acuity. To allocate the limited visual processing resources, one needs to firstly choose a particular part of the visual field, and then focuses on it. For example, when humans are reading, usually the words to be read at the particular moment are attended and processed. As a result, there are two main aspects of attention:

- Decide which part of the input needs to be focused on.
- Allocate the limited processing resources to the important part.

The definition of attention introduced in psychology is very abstract and intuitive, so the idea is borrowed and widely used in computer science area, although some techniques do not contain exactly the word "attention". For example, a CPU will only load the data needed for computing instead of loading all data available, which can be seen as a naïve application of the attention. Furthermore, to make the data loading more effective, the *multilevel storage* design in computer architecture is employed, i.e., the allocation of computing resources is aided by the structure of multi caches, main memory, and storage device as shown in Figure 1. In a big picture, all of them make up an attention model.

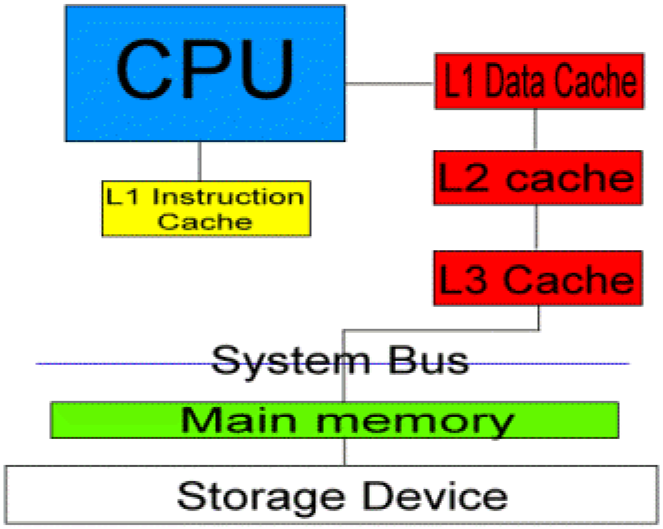


Figure 1: The CPU cache system. We can see that there is a multi-level structure between the data computing unit (CPU), and the data permanent storage device (Storage Device). The figure is taken from [1].

1.2 What are the applications of the attention based model in computer vision?

As mentioned above, the attention mechanism plays an important role in human visual processing. Some researchers also bring the attention into computer vision area. As in the human perception, a computer vision system should also focus on the important part of the input image, instead of giving all pixels the same weights. A simple and widely used method is extracting local image features from the input. Usually the local image features can be points, lines, corners, or small image patches, and then some measurements are taken from the patches and converted into descriptors. Obviously, the distribution of local features' positions in a natural image is not uniform, which makes the system give different weights to different parts of the input image, and satisfies the definition of attention.

Saliency detection is another typical example directly motivated by human perception. As mentioned above, humans have ability to detect salient objects/regions rapidly and then attend on by moving the line of sight. This capability is also studied by the computer vision community, and many saliency object/region detection algorithms are proposed, e.g. [2, 24, 35]. However, most of the saliency detection methods only use the low level image features, e.g., contrast, edge, intensity, etc., which makes them bottom-up, stimulus-driven, and fixed. So they cannot capture the task specific semantic information. Furthermore, the detections of different saliency objects/regions in an image are individually and independent, which is obviously unlike the humans, where the next attended region of an image usually is influenced by the previous perceptions.

The sliding window paradigm [11, 52] is another model which matches the essence of attention. It is common that the interested object only occupies a small region of the input image, and sometimes there are some limitations in the computational capabilities, so the sliding window is widely used in many computer vision tasks, e.g., object detection, object tracking, etc. However, since the size, shape and location of a window can be assigned arbitrarily, in theory there are infinite possible windows for an input image. Lots of work is devoted to reducing the number of windows to be evaluated, and some of them make notable speedups compared to the naïve method. But most of the window reducing algorithms are specifically designed for object detection or object tracking tasks, and the lack of universality makes them difficult to be applied in other tasks. Besides, most of the sliding window methods do not have the ability to take full advantage of the past processed data in future prediction.

1.3 What is the attention based RNN model?

Note that the *attention* is just a mechanism, or a methodology, so there is no strict definition in mathematics what it is. For example, the local image features, saliency detection, sliding window methods introduced above, or some recently proposed object detection methods like [19], all employ the attention mechanism in different mathematical forms. On the other hand, as illustrated later, the RNN is a specific type of neural network with a specific mathematical description. The *attention based RNN models* in this survey refers in particular to the RNN models which are designed for sequence to sequence problems with the attention mechanism. Besides, all the systems in this survey are *end-to-end* trainable, where the parameters for both the attention module and the common RNN model should be learned simultaneously. Here we will give a general explanation of what the RNN is and what the attention based RNN model is, and the mathematical details are left to be described later.

1.3.1 Recurrent neural network (RNN)

Recently, the convolutional neural network (CNN) is very popular in the computer vision community. But the biggest problem for the CNN is that it only accepts a fixed length input vector and gives a fixed-length output vector. So it cannot handle the data with rich structures, especially the sequences. That is the reason why RNN is interesting, which can not only operate over sequences of input vectors, but also generate sequences of output vectors. Figure 2 gives an abstract structure of a RNN unit.

The blue rectangles are the input vectors in a sequence, and the yellow rectangles are the output vectors in a sequence. By holding a hidden state (green rectangles in Figure 2), the RNN is able to process the sequence data. The dashed arrow indicates sometimes the output vectors do not have to be generated. The details of the structure of the neural network, the recurrent neural network, how the hidden state works, and how the parameters are learned will be described later.

1.3.2 The RNN for sequence to sequence problem

1.3.2.1. Sequence to sequence problem

As mentioned above, the RNN unit holds a hidden state which empowers it to process sequential data with variable sizes. In this survey, we focus on the RNN models for sequence to sequence problem. The sequence to sequence problem is defined as to map the input sequence to the output sequence, which is a very general definition. Here the *input sequence*

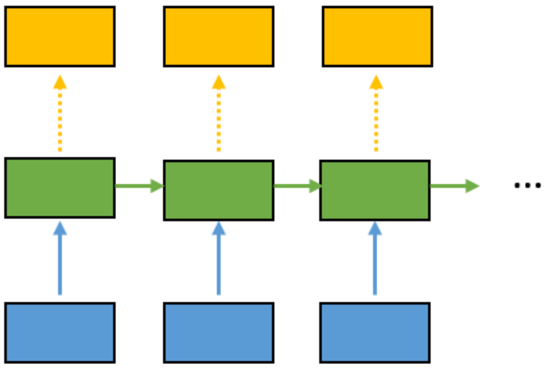


Figure 2: An abstract structure of a RNN unit, where each rectangle represents a vector. The blue rectangles are the input vectors, the yellow rectangles are the output vectors, and the green rectangles are the hidden states. The dashed arrow indicates that the output is optional.

does not refer strictly to a sequence of items, otherwise for different sequence to sequence problems, it could be in different forms. For example:

- For the machine translation task, one usually needs to translate some sentences in the source language to the target language. In this case, the input sequence is a natural language sentence, where each item in the sequence is the word in the sentence. The order of the items in the sequence is critical.
- In the video classification task, a video clip usually should be assigned a label. In this case, the input sequence is a list of frames of the video, where each frame is an image. And the order is critical.
- The object detection task requires the model to detect some objects in an input image. In this case, the input sequence is only an image which can be seen as a list of objects. But obviously, it is not trivial to extract those objects from the image without additional data or information. So for the object detection task, the input of the model is just a single feature map (an image).

Similarly, the output sequence also does not always contain explicit items, but in this survey, we only focus on the problems where the output sequence has explicit items, although sometimes the number of items in the sequence is one. Besides, no matter the input is a feature map or it contains some explicit items, we always use ”input sequence” to represent them. And the term ”item” is used to describe the item in the sequence no matter it is contained by the sequence explicitly or implicitly.

1.3.2.2. The structure of the RNN model for sequence to sequence problem

Under the condition that the lengths of the input and output sequences can vary, it is not possible for a common RNN to directly construct the corresponding relationships between the items in the input and output sequences without any additional information. As a

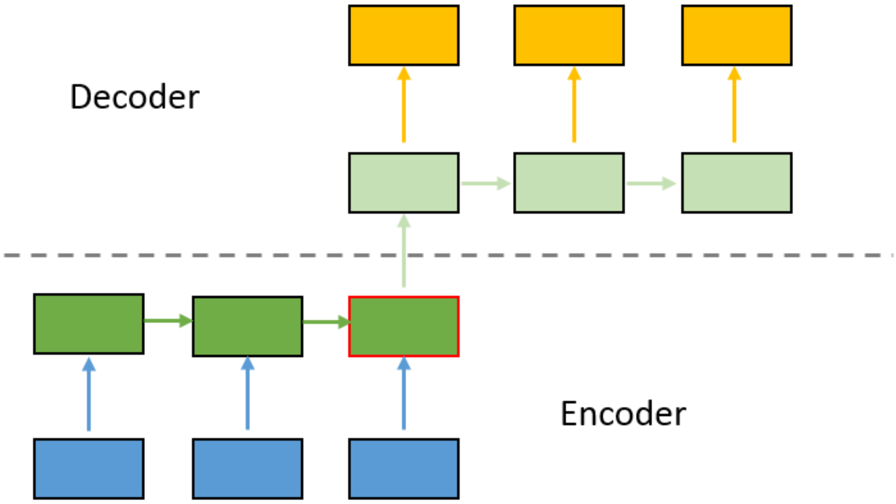


Figure 3: The encoder and the decoder. The rectangle surrounded by the red box is the intermediate code. The blue rectangles represent the input items, the dark green rectangles are the hidden states of the encoder, the shallow green rectangles represent the hidden states of the decoder, and the yellow rectangles are the output items.

result, the RNN model needs to firstly learn and absorb all (or a part of) useful information from all of the input items, and then make the predictions. So it is natural to divide the whole system into two parts: the encoder, and the decoder [10, 47]. The encoder encodes all (or a part of) the information of the input data to an intermediate code, and the decoder employs this code in generating the output sequence. In some sense, all neural networks can be cut into two parts in any position, the first half is called the encoder, and the second half is treated as the decoder, so here the encoder and decoder are both neural networks.

Figure 3 gives a visual illustration of a RNN model for sequence to sequence problem. The blue rectangles represent the input items, the dark green rectangles are the hidden states of the encoder, the shallow green rectangles represent the hidden states of the decoder, and the yellow rectangles are the output items. Although there are three input items and three output items in Figure 3, actually the number of input and output items can be arbitrary number and do not have to be the same. And the rectangle surrounded by the red box in Figure 3 serves as the ”middleman” between the input and the output which is the intermediate code generated by the encoder. In Figure 3 both the encoder and the decoder are recurrent networks, while with different types of the input sequence mentioned above, the encoder does not have to be recurrent all the time. For example, when the input is a feature map, a neural network without recurrence usually is applied as the encoder.

When dealing with a supervised learning problem, during training, usually the ground truth output sequence corresponds to an input sequence is available. With the predicted output sequence, one can calculate the log-likelihood between the items which have the same indices in the predicted and the ground truth output sequences. The sum of the log-likelihood of all items usually is set as the objective function of the supervised learning problem. And then usually the gradient decent/ascent is used to optimize the objective in order to learn the value of the parameters of the model.

1.3.3 Attention based RNN model

For sequence to sequence problems, usually there are some corresponding relations between the items in the output and input sequences. However for a common RNN model introduced above, the length of the intermediate code is fixed, which prevents the model giving different weights to different items in an input sequence explicitly, so all items in the input sequence has the same importance no matter which output item is attempted to be predicted. This inspires the researchers to add the attention mechanism into the common RNN model.

Besides, the encoding-decoding process can also be regarded as a compression process: firstly the input sequence is compressed into an intermediate code, and then the decoder decompresses the code to get the output. The biggest problem is that no matter what kind of input the model gets, it always compresses the information into a fixed length code, and then the decoder uses this code to decompress all items of a sequence. From an information theory perspective, this is not a good design since the length of the compressed code should have a linear relationship to the amount of information the inputs contain. As in the real file compression, the length of the compressed file is proportional to the amount of information the source files contain. There are some straightforward solutions to solve the problem mentioned above:

从压缩的观点看，*encoder-decoder*模型是对输入信息进行压缩； 但是传统*encoder-decoder*模型的问题在于，不论输入序列多长，压缩向量的长度却保持不变。

1. Build an extremely non-linear and complicated encoder model such that it can store large amount of information into the intermediate code.

2. Just use a long enough code.

3. Stay with the fixed length code, but let the encoder only encode a part of the input items which are needed for the current decoding.

Because the input sequence can be infinite long (or contain infinite amount of information), the first solution does not solve the problem essentially. Needless to say, it is even a more difficult task to estimate the amount of information some data contains. Based on the same reason, solution 2 is not a good choice either. In some extreme cases, even if the amount of information is finite, the code should still be large enough to store all possible input-output pairs, which is obviously not practical.

Solution 3 also cannot solve the problem when the amount of the input information needed for the current decoding is infinite. But usually a specific item in the output sequence only corresponds to a part of the input items. In this case, solution 3 is more practical and can prevent the problem in some degrees compared to other solutions if the needed input items can be ”located” correctly. Furthermore, it essentially has the same insights as the multilevel storage design pattern as shown in Figure 1. The RNN model which has the addressing ability is called the attention based RNN model.

With the attention mechanism, the RNN model is able to assign different weights to different parts of items in the input sequence, and consequently, the inherent corresponding relations between items in input and output sequences can be captured and exploited explicitly. Usually, the attention module is an additional neural network which is connected to the original RNN model whose details will be introduced later.

1.4 What are the attention mechanisms introduced in this survey?

In this survey, we introduce four types of attention mechanisms, which are: item-wise soft attention, item-wise hard attention, location-wise hard attention, and location-wise soft attention. Here we only give a brief and high-level illustration of them, and leave the mathematical descriptions in the next chapter.

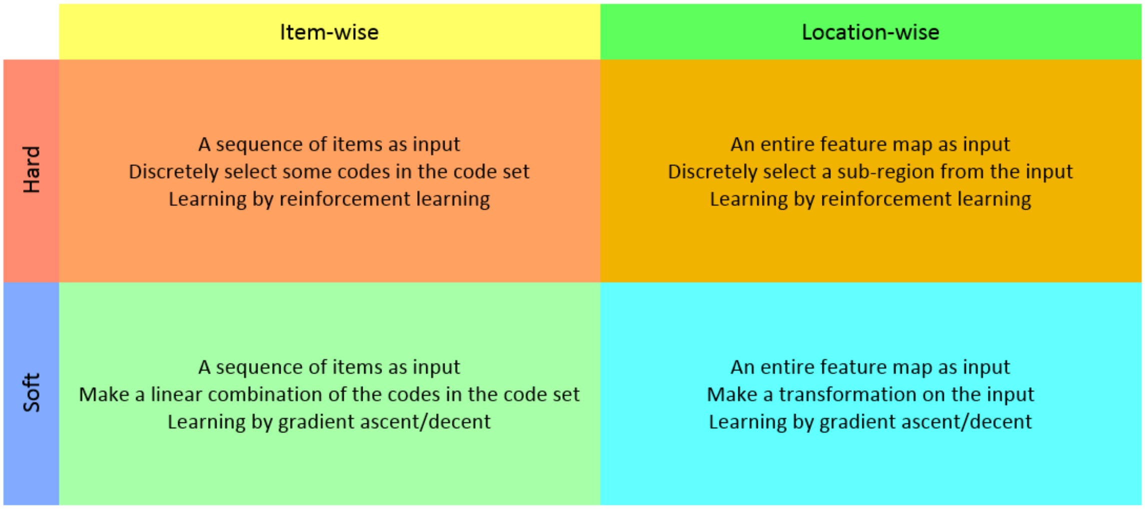
For the *item-wise* and the *location-wise*, as mentioned above, the forms of the input sequence are different for different tasks. The item-wise mechanism requires that the input is a sequence containing explicit items, or one has to add an additional pre-processing step to generate a sequence of items from the input. However the location-wise mechanism is designed for the input which is a single feature map. The term ”location” is used because in most cases this kind of input is an image, and when treating it as a sequence of objects, all the objects can be pointed by their locations.

As described above, the item-wise method works on ”item-level”. After being fed into the encoder, each item in the input sequence has an individual code, and all of them make up a code set. During decoding, at each step the item-wise soft attention just calculates a weight for each code in the code set, and then makes a linear combination of them. The combined code is treated as the input of the decoder to make the current prediction. The only difference lies in the item-wise hard attention is that it instead makes a hard decision and stochastically picks some (usually one) codes from the code set according to their weights as the intermediate code fed into the decoder.

The location-wise attention mechanism directly operates on an entire feature map. At each decoding step, the location-wise hard attention mechanism discretely picks a sub-region from the input feature map and feeds it to the encoder to generate the intermediate code. And the location of the sub-region to be picked is calculated by the attention module. The location-wise soft attention still accepts the entire feature map as input at each step while otherwise makes a transformation on it in order to highlight the interesting parts instead of discretely picking a sub-region.

As mentioned above, usually the attention mechanism is implemented as an additional neural network connected to the raw RNN. The whole model should still be *end-to-end*, where both the raw RNN and the attention module are learned simultaneously. When it comes to *soft* and *hard*, for the soft attention, the attention module is differentiable with respect to the inputs, so the whole system can still be updated by gradient ascent/decent. However, since the hard attention mechanism makes hard decisions and gives discrete selections of the intermediate code, the whole system is not differentiable with respect to its inputs anymore. Then some techniques from the reinforcement learning are used to solve learning problem.

In summary, the attention mechanisms are designed to help the model select better intermediate codes for the decoder. Figure 4 gives a visual illustration of the four attention mechanisms which are about to be introduced in this survey.



	Item-wise	Location-wise
Hard	A sequence of items as input Discretely select some codes in the code set Learning by reinforcement learning	An entire feature map as input Discretely select a sub-region from the input Learning by reinforcement learning
Soft	A sequence of items as input Make a linear combination of the codes in the code set Learning by gradient ascent/decent	An entire feature map as input Make a transformation on the input Learning by gradient ascent/decent

Figure 4: Four attention mechanisms.

1.5 What are the advantages of using attention based RNN model?

Here we give some general advantages of the attention based RNN model. The detailed comparison between the attention based RNN model and the common RNN model will be illustrated later. Advantages:

- First of all, as its name indicates, the attention based RNN model is able to learn to assign weights to different parts of the input instead of treating all input items equally, which can provide the inherent relations between the items the in input and output sequence. This is not only a way to boost the performance of some tasks, but it is also a powerful tool of visualization compared to the common RNN model.

- The hard attention model does not need to process all items in the input sequence, instead, sometimes only processes the interested ones, so it is very useful for some tasks with only partially observable environments, like game playing, which usually cannot be handled by the common RNN model.

- Not limited in computer vision area, the attention based RNN model is also suitable for all kinds of sequence related problems. For example:

  - Applications in natural language processing (NLP), e.g., machine translation [6, 37], machine comprehension [22], sentence summarization [43], word representation [33, 34].
  - Bioinformatics [46].
  - Speech recognition [8, 36].
  - Game play [38].
  - Robotics.

1.6 Organization of this survey

In Section 2, we describe the general mathematical form of the attention based RNN model, especially four types of the attention mechanisms: item-wise soft attention, item-wise hard attention, location-wise hard attention, and location-wise soft attention. Next, we introduce some typical applications of the attention based RNN model in computer vision area in Section 3. At last, we summarize the insights of the attention based RNN model, and discuss some potential challenges and future research directions.

2. The attention based RNN model

One can have a general idea of what the attention based RNN is from the last section. In this section, we firstly give a short introduction of the neural network, followed by an abstract mathematical description of the RNN. And then four attention mechanisms on RNN model are analyzed in detail.

2.1 Neural network

A neural network is a data processing and computational model consists of multiple neurons which connect to each other. The basic component of all neural network is the neuron whose name indicates its inspiration from the human neuron cell. Ignoring the biological implication, a neuron in the neural network accepts one or more inputs, reweighs the inputs and sums them, and finally gives one output by passing the sum through an activation function. Let the input to the neuron be $\boldsymbol{i} =< i_1, i_2, \ldots, i_n >$, the corresponding weights be $\boldsymbol{w} =< w_1, w_2, \ldots, w_n >$, and the output be o. A visual example of a neuron is given in Figure 5, where f is the activation function. And we have:

$$o = f\left(\sum_{k=1}^{n}(w_k i_k) + w_0\right) \tag{1}$$

where w_0 is the bias parameter. If we add $i_0 = 1$ into $\boldsymbol{i}$ and rewrite the previous equation in vector form, then

$$o = f\left(\sum_{k=0}^{n}(w_k i_k)\right) = f\left(\boldsymbol{w}^T \boldsymbol{i}\right) \tag{2}$$

In summary, a neuron just applies an activation function on the linear transformation of the input parameterized by the weights. Usually the activation functions are designed to be non-linear in order to import non-linearity into the model. There are many different forms of activation functions, i.e., sigmoid, tanh, ReLU [40], PReLU [21].

A neural network is usually constructed by the connections of many neurons, and the most popular structure is layer by layer connection as shown in Figure 6. In Figure 6, each circle represents a neuron, and each arrow represents a connection between neurons. The outputs of all neurons in a layer in Figure 6 are connected to the next layer, which can be categorized as fully-connected layer. In summary, if we treat the neural network as a black box and see it from outside, a simple mathematical form can be obtained:

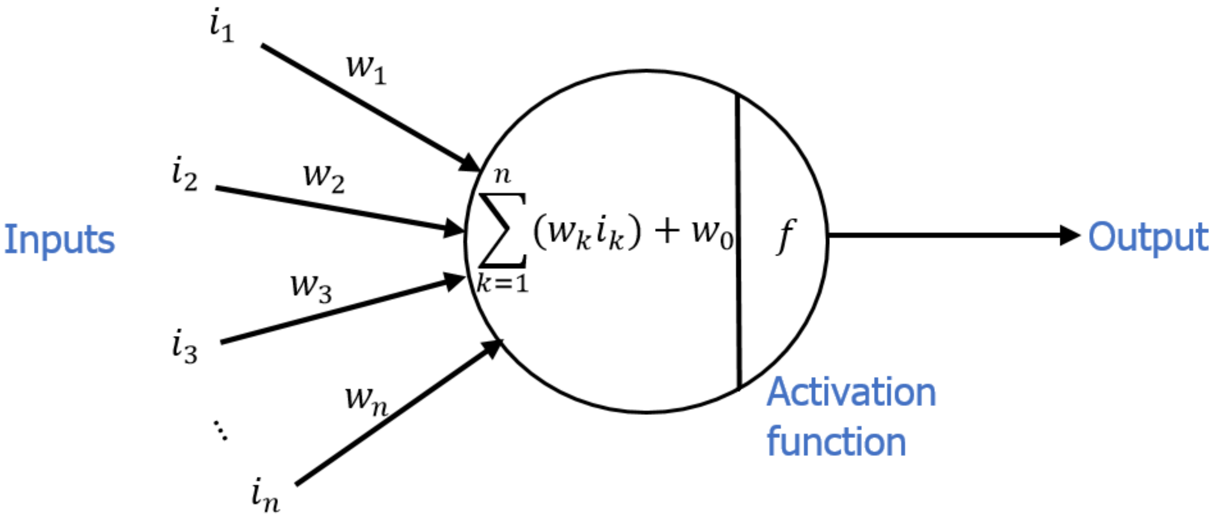


Figure 5: A neuron.

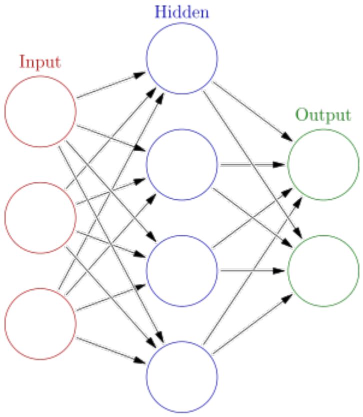


Figure 6: A neural network, where each circle represents a neuron, and each arrow represents a connection. Here the ”connection” means the output of a neuron is used as the input for another neuron. The figure is taken from [55].

$$\boldsymbol{o} = \phi_W(\boldsymbol{i}) \tag{3}$$

where $\boldsymbol{i}$ is the input, $\boldsymbol{o}$ is the output, and ϕ_W is a particular model consists of many neurons parameterized by W. One property of the model ϕ_W is that it is usually differentiable with respect to W and $\boldsymbol{i}$. For more details about the neural network, one can refer to [44].

The convolutional neural network (CNN) is a specific type of neural network with structures designed for image inputs [30], which usually consists of multiple convolutional layers followed by a few fully-connected layers. The convolutional layer can take a region of a 2D feature map as input, and this is the reason why it is suitable for image data. Recently the CNN is one of the most popular model as candidate for image related problems. A deep CNN model trained on some large image classification dataset (e.g., ImageNet [12]) has good generalization power/ability and can be easily transferred to other tasks. For more details about CNN, one can refer to [44].

2.2 A general RNN unit

As its name shows, the RNN is also a type of neural network, which therefore consists of a certain number of neurons as all kinds of neural network. The ”recurrent” in the RNN indicates that it performs the same operations for each item in the input sequence, and at the same time, keeps a *memory* of the processed items for future operations. Let the input sequence be $I = (\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_T)$. At each step t, the operation performed by a general RNN unit can be described by:

$$\begin{bmatrix} \hat{\boldsymbol{o}}_t \\ \boldsymbol{h}_t \end{bmatrix} = \phi_W \left(\boldsymbol{i}_t, \boldsymbol{h}_{t-1} \right) \tag{4}$$

where $\hat{\boldsymbol{o}}_t$ is the predicted output vector at time t, and $\boldsymbol{h}_t$ is the hidden state at time t. ϕ_W is a neural network parameterized by W which takes the t-th input item ($\boldsymbol{i}_t$), and the previous hidden state $\boldsymbol{h}_{t-1}$ as inputs. Here it is clear that the hidden state $\boldsymbol{h}$ performs as a memory to store the previous processed information. From Equation (4), we can see the definition of a RNN unit is quite general, because there is no specific requirements for the structure of the network ϕ_W. People have already proposed hundreds or even thousands of different structures which are out of the scope of this paper. Now two widely used structures are LSTM (long-short term memory) [23] and GRU (gated recurrent units) [9].

2.3 The model for sequence to sequence problem

As shown in Figure 3, the model for sequence to sequence problem can be generally divided into two parts: the encoder, and the decoder, where both of them are neural networks. Let the input sequence be X, and the output sequence be Y. As mentioned above, in some tasks the input sequence does not consists of explicit items, for which we use X to represent the input sequence, and otherwise $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T)$. Besides, in this survey, only the output sequence containing explicit items is considered, i.e., $Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{T'})$ is the output sequence. The lengths of the input and the output sequences do not have to be the same.

2.3.1 Encoder

As mentioned above, the core task of an encoder is to encode all (or a part of) the input items to an intermediate code to be decoded by the decoder. In the common RNN for sequence to sequence problem, the model does not try to figure out the corresponding relations between the items in the input and output sequences, and usually all items in the input sequence are compressed into a single intermediate code. So for the encoder,

$$\boldsymbol{c} = \phi_{W_{enc}} \left(X \right) \tag{5}$$

where $\phi_{W_{enc}}$ represents the encoder neural network parameterized by W_{enc}, $\boldsymbol{c}$ is the intermediate code to be used by the decoder. Here the encoder can be any neural network, but its type usually depends on the input data. For example, when the input X is treated as a single feature map, usually a neural network without recurrence is used as the encoder, like a CNN for the image input. But it is also common to set encoder as a recurrent network when the input is a sequence of data, e.g., a video clip, a natural language sentence, a human speech clip.

2.3.2 Decoder

If the length of the output sequence is larger than 1, in most cases the decoder is recurrent, because the decoder at least needs to have the knowledge what it has already predicted to prevent the repeated prediction. Especially in the RNN model with the attention mechanism which will be introduced later, the weights of the input items are assigned with the guidance of the past predictions, so in this case, the decoder should be able to store some history information. As a result, in this survey we only consider the cases where the decoder is an RNN.

As mentioned above, in a common RNN, the decoder accepts the intermediate code $\boldsymbol{c}$ as input, and at each step j generates the predicted output $\hat{\boldsymbol{y}}_j$, and the hidden state $\boldsymbol{h}_j$ by

$$\begin{bmatrix} \hat{\boldsymbol{y}}_j \\ \boldsymbol{h}_j \end{bmatrix} = \phi_{W_{dec}} \left(\boldsymbol{c}, \hat{\boldsymbol{y}}_{j-1}, \boldsymbol{h}_{j-1} \right) \tag{6}$$

where $\phi_{W_{dec}}$ represents the decoder recurrent neural network parameterized by W_{dec}, which accepts the code $\boldsymbol{c}$, the previous hidden state $\boldsymbol{h}_{j-1}$, and the previous predicted output $\hat{\boldsymbol{y}}_{j-1}$ as input. After T' steps of decoding, a sequence of predicted outputs $\hat{Y} = (\hat{\boldsymbol{y}}_1, \hat{\boldsymbol{y}}_2, \ldots, \hat{\boldsymbol{y}}_{T'})$ is obtained.

2.3.3 Learning

Like many other supervised learning problems, the supervised sequence to sequence problem is also optimized by maximizing the log-likelihood. With the input sequence X, the ground truth output sequence Y, and the predicted output sequence $\hat{Y}$, the log-likelihood for the j-th item $\boldsymbol{y}_j$ in the output sequence Y is

$$L_j\left(X, Y, \theta\right) = \log p\left(\boldsymbol{y}_j | X, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{j-1}, \theta\right) = \log p\left(\boldsymbol{y}_j | \hat{\boldsymbol{y}}_j, \theta\right) \tag{7}$$

where $\theta = [W_{enc}, W_{dec}]$ represents all learnable parameters in the model, and $p()$ calculates the likelihood of $\boldsymbol{y}_j$ based on $\hat{\boldsymbol{y}}_j$. For example, in a classification problem where $\boldsymbol{y}_j$ indicates the index of the label, the $p\left(\boldsymbol{y}_j | \hat{\boldsymbol{y}}_j, \theta\right) = \text{softmax}(\hat{\boldsymbol{y}}_j)_{\boldsymbol{y}_j}$. The objective function is then set as the sum of the log-likelihood:

$$L(X, Y, \theta) = \sum_{j=1}^{T'} L_j(X, Y, \theta) = \sum_{j=1}^{T'} \log p\left(\boldsymbol{y}_j | X, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{j-1}, \theta\right) = \sum_{j=1}^{T'} \log p\left(\boldsymbol{y}_j | \hat{\boldsymbol{y}}_j, \theta\right) \tag{8}$$

For the common encoder-decoder system, both the encoder and decoder network are differentiable with respect to their parameters, W_{enc}, and W_{dec}, respectively, so the objective function is differentiable with respect to θ. As a result, one can update the parameters θ by gradient ascent to maximize the sum of the log-likelihood.

2.4 Attention based RNN model

In this section, we give a comprehensive mathematical description of the four attention mechanisms. As mentioned above, essentially the attention module helps the encoder calculate a better intermediate code $\boldsymbol{c}$ for the decoder. So in this section, the decoder part is identical as shown in Section 2.3.2, but the encoder network $\phi_{W_{enc}}$ may not always be the same as in Equation (5). In order to make the illustration clear, we still keep the term $\phi_{W_{enc}}$ to represent the encoder.

2.4.1 Item-wise soft attention

The item-wise soft attention requires the input sequence X contains some explicit items $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T$. Instead of extracting only one code $\boldsymbol{c}$ from the input X, the encoder of the item-wise soft attention RNN model extracts a set of codes C from X:

$$C = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{T''}\} \tag{9}$$

where the size of C does not have to be the same as X, which means one can use multiple input items to calculate a single code or extract multiple codes from one input item. For simplicity, we just set $T'' = T$. So

$$\boldsymbol{c}_t = \phi_{W_{enc}}(\boldsymbol{x}_t) \text{ for } t \in (1, 2, \ldots, T) \tag{10}$$

where $\phi_{W_{enc}}$ is the encoder neural network parameterized by W_{enc}. Note here that the encoder is different from the encoder in Equation (5), because the encoder in the item-wise soft attention model only takes one item of the sequence as input.

During the decoding, the intermediate code $\boldsymbol{c}$ fed into the decoder is calculated by the attention module, which accepts all individual codes C, and the decoder's previous hidden state $\boldsymbol{h}_{j-1}$ as input. At the decoding step j, the intermediate code is:

$$\boldsymbol{c} = \boldsymbol{c}^j = \phi_{W_{att}}(C, \boldsymbol{h}_{j-1}) \tag{11}$$ [1]

where $\phi_{W_{att}}$ represents the attention module parameterized by W_{att}. Actually, as long as the attention module $\phi_{W_{att}}$ is differentiable with respect to W_{att}, it satisfies the requirements of the soft attention. Here we only introduce the first proposed item-wise soft attention model in [6] for natural language processing. In this item-wise soft attention model, at decoding step j, for each input code $\boldsymbol{c}_t$, a weight α_{tj} is calculated by:

$$e_{jt} = f_{att}(\boldsymbol{c}_t, \boldsymbol{h}_{j-1}) \tag{12}$$

and

$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{t=1}^{T} \exp(e_{jt})} \tag{13}$$

[1] For simplicity, in the following sections, we just use $\boldsymbol{c}$ to represents the intermediate code fed into the decoder, and ignore the superscript j.

where f_{att} usually is also a neural network within the attention module and calculates the unnormalized weight e_{jt}. Here the normalized weight α_{tj} is explained as the probability that how the code $\boldsymbol{c}_t$ is relevant to the output $\boldsymbol{y}_j$, or the importance should be assigned to the t-th input item when making the j-th prediction. Note here that Equation (13) is a simple softmax function transforming the scores e_{jt} to the scale from 0 to 1, which makes the α_{jt} can be interpreted as a probability.

Since now we have the probabilities, then the code $\boldsymbol{c}$ is just calculated by taken the expectation of all $\boldsymbol{c}_t$s with their probabilities α_{jt}s:

$$\begin{aligned} \boldsymbol{c} &= \phi_{W_{att}}\left(C, \boldsymbol{h}_{j-1}\right) \\ &= \mathbb{E}(\boldsymbol{c}_t) \\ &= \sum_{t=1}^{T} \alpha_{jt}\boldsymbol{c}_t \end{aligned} \tag{14}$$

Figure 7 gives a visual illustration of how the item-wise soft attention works at decoding step j, where the purple ellipse represents the Equation (12) and Equation (13), and the encoder network is not recurrent.

2.4.2 Item-wise hard attention

The item-wise hard attention is very similar to the item-wise soft attention. It still needs to calculate the weights for each code as shown from Equation (9) to Equation (13). As mentioned above, the α_{jt} can be interpreted as the probability that how the code $\boldsymbol{c}_t$ is relevant to the output $\boldsymbol{y}_j$, so instead of a linear combination of all codes in C, the item-wise hard attention stochastically picks one code based on their probabilities. In detail, an indicator l_j is generated from a categorical distribution at decoding step j to indicate which code should be picked:

$$l_j \sim \mathcal{C}\left(T, \{\alpha_{jt}\}_{t=1}^{T}\right) \tag{15}$$

where $\mathcal{C}()$ is a categorical distribution parameterized by the probabilities of the codes ($\{\alpha_{jt}\}_{t=1}^{T}$). And l_j works as an index in this case:

$$\boldsymbol{c} = \boldsymbol{c}_{l_j} \tag{16}$$

When the size of C is 2, the above categorical distribution in Equation (15) turns into a Bernoulli distribution:

$$l_j = \mathcal{B}\left(T, \alpha_{j1}\right) \tag{17}$$

2.4.3 Location-wise hard attention

As mentioned above, for some types of input X, like an image, it is not trivial to directly extract items from them. So the location-wise hard attention model is developed which accepts the whole feature map X as input, stochastically picks a sub-region from it, and uses this sub-region to calculate the intermediate code at each decoding step. This kind of location-wise hard attention is analyzed in many previous works [13, 28], while here we only

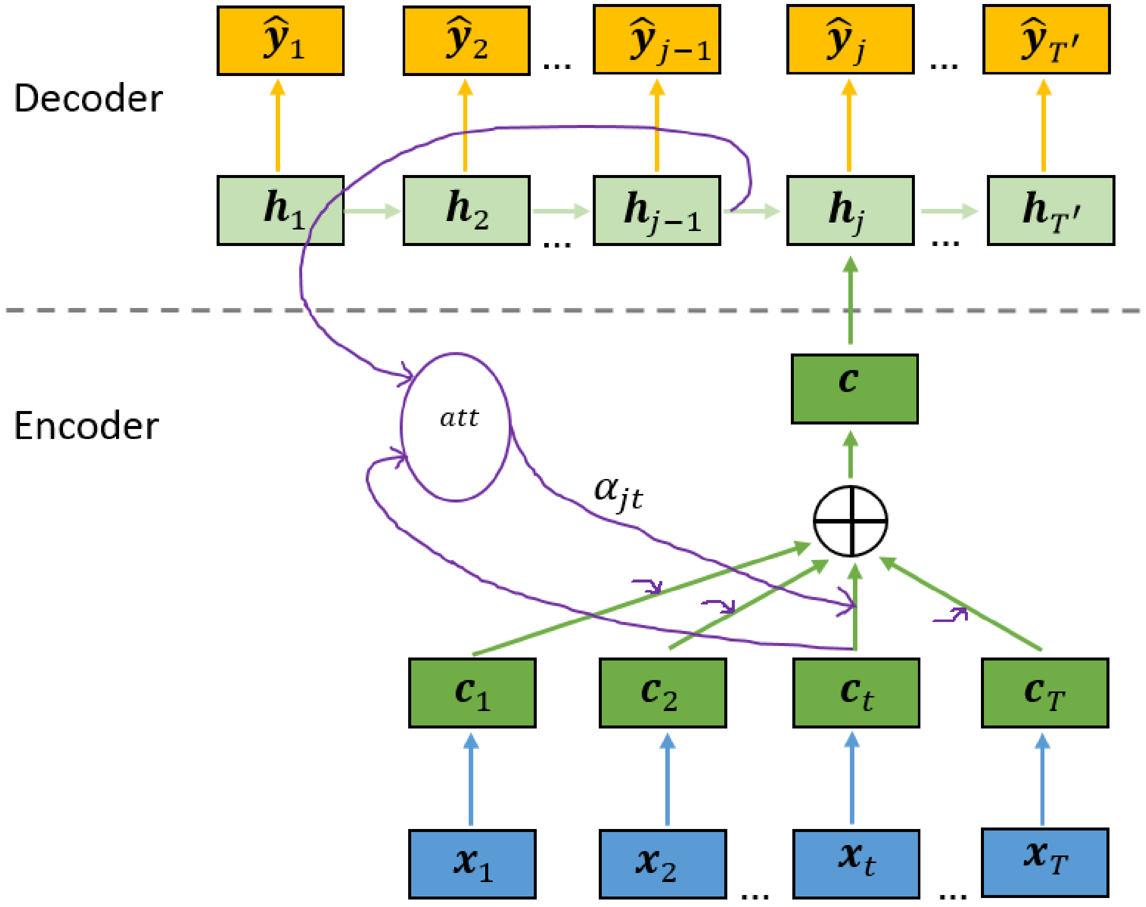


Figure 7: The item-wise soft attention based RNN model at decoding step j. The purple ellipse represents the Equation (12) and Equation (13). And the encoder is not recurrent.

focus on two recent proposed mechanisms in [39] and [4], because the attention models in those works are more general and can be trained end-to-end.

Since the location-wise hard attention model only picks a sub-region of the input X at the decoding step, it makes sense to estimate that the picked region may not correspond to the output item to be predicted. There are two potential solutions for this problem:

1. Pick only one input vector at each decoding step.

2. Make a few *glimpses* for each prediction, which means at each decoding step, pick some sub-regions, process them and update the model one by one, where the subsequent region (glimpse) is acquired based on the previous processed regions (glimpses). And the last prediction is treated as the ”real” prediction in this decoding step. The number of glimpses M either can be an arbitrary number (a hyperparameter) or automatically be decided by the model.

With a large enough training dataset, it is reasonable to estimate that method 1 can also find the optimal while the convergence may take longer time because of the limitation mentioned above. In testing, obviously method 1 is faster than method 2, but there may be a sacrifice on the performance. For method 2, it may have better performance in testing, while it also needs longer calculating time. Till now there are not so many works done on how to select an appropriate number of glimpses which leaves it an open problem. However, it is clear that method 1 is just an extreme case of method 2, and in the following sections, we treat the hard attention models as taking M glimpses at each step of prediction.

To keep the notation consistent, here we still use term l as the indicator generated by the attention model to show which sub-region should be extracted, i.e., in the case of location-wise hard attention, the l_j^m indicates the location of the center of the sub-region is about to be picked. In detail, at the m-th glimpse in the decoding step j, the attention module accepts the input X, the previous hidden states of the decoder ($\boldsymbol{h}_j^{m-1}$), and calculates l_j^m by:

$$l_j^m \sim \mathcal{N}\left(f_{att}\left(X, \boldsymbol{h}_j^{m-1}\right), s\right) \tag{18}$$

where $\boldsymbol{h}_j^0 = \boldsymbol{h}_{j-1}$ and $\mathcal{N}()$ is a normal distribution with a mean $f_{att}\left(X, \boldsymbol{h}_j^{m-1}\right)$ and a standard deviation s. And f_{att} usually is a neural network similar to the f_{att} in Equation (12). Then the attention module picks the sub-region centered at location l_j^m:

$$X_{out}^m = X_{l_j^m} \tag{19}$$

where $X_{l_j^m}$ indicates a sub-region of X centered at l_j^m, and X_{out}^m represents the output sub-region at glimpse m. Note here that the shape, size of the sub-region, and the standard deviation s in Equation (18) are all hyperparameters. One can also let the attention network generate these parameters as the generation of l_j^m in Equation (18) [59]. When X_{out}^m is generated, it is fed into the encoder in calculating the code $\boldsymbol{c}^m$ for the decoder, i.e.,

$$\boldsymbol{c}^m = \phi_{W_{enc}}\left(X_{out}^m\right) \tag{20}$$

and then we have

$$\begin{bmatrix} \hat{\boldsymbol{y}}_j^m \\ \boldsymbol{h}_j^m \end{bmatrix} = \phi_{W_{dec}}\left(\boldsymbol{c}^m, \hat{\boldsymbol{y}}_j^{m-1}, \boldsymbol{h}_j^{m-1}\right) \tag{21}$$

After M glimpses:

$$\boldsymbol{c} = \boldsymbol{c}^M \tag{22}$$

$$\boldsymbol{h}_j = \boldsymbol{h}_j^M \tag{23}$$

$$\hat{\boldsymbol{y}}_j = \hat{\boldsymbol{y}}_j^M \tag{24}$$

where $\boldsymbol{h}_j$ is the j-th hidden state of the decoder, and $\hat{\boldsymbol{y}}_j$ is the j-th prediction.

2.4.4 Location-wise soft attention

As mentioned above, the location-wise soft attention also accepts a feature map X as input, and at each decoding step, a transformed version of the input feature map is generated to calculate the intermediate code. It is firstly proposed in [25] called the spatial transformation network (STN). Instead of the RNN model, the STN in [25] is initially applied on a CNN model, but it can be easily transferred to a RNN model with tiny modifications. To make the explanations more clear, we use X_{in} to represent the input of the STN (where usually $X = X_{in}$), and X_{out} to represent the output of the STN. Figure 8 gives a visual example of how the STN is embedded into the whole framework, and how it works at decoding step j, in which the purple ellipse is the STN module. From Figure 8, we can see that after being processed by the STN, the X_{in} is transformed to X_{out}, and it is used to calculate the intermediate code $\boldsymbol{c}$ which is then fed to the decoder. The details of how the STN works will be illustrated in the following part of this section.

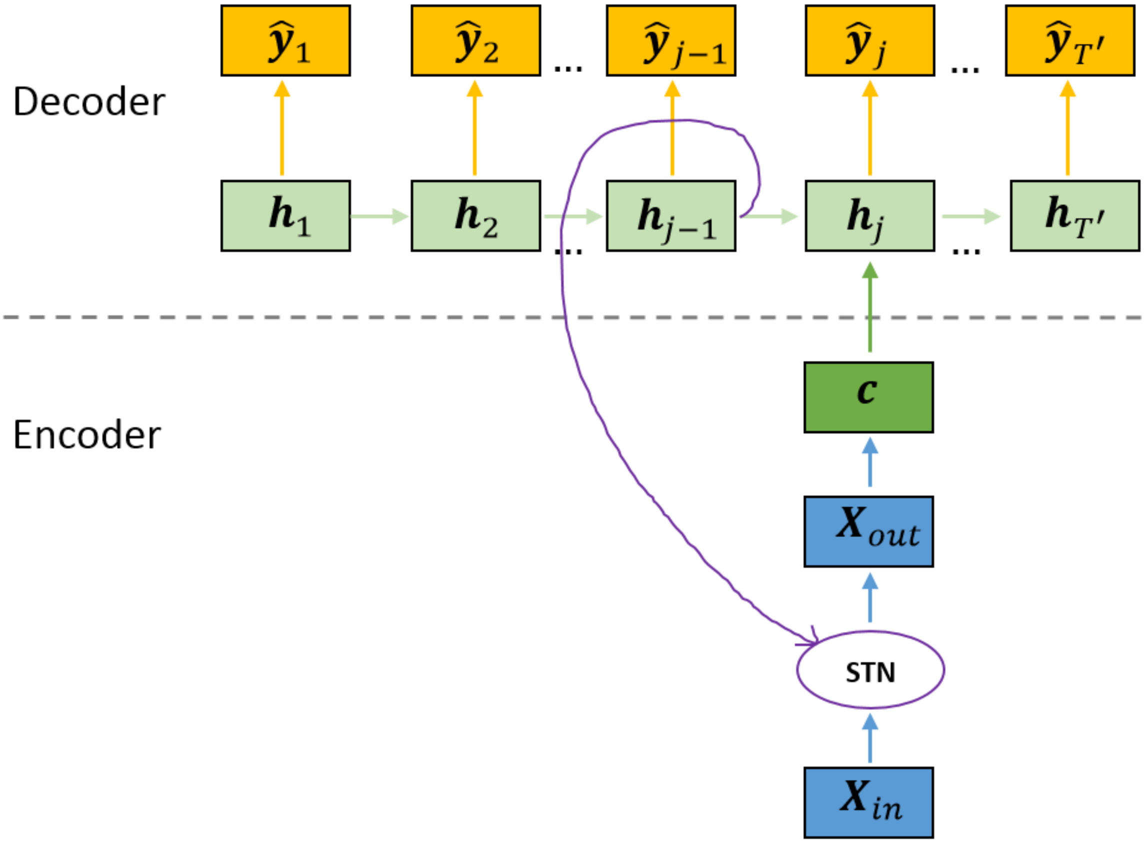


Figure 8: The location-wise soft attention (STN) based RNN model at decoding step j. The purple ellipse represents the STN module which will be described in detail later. X_{in} is the input feature map, and X_{out} is the output feature map.

Let the shape of X_{in} be $U_{in} \times V_{in} \times Q_{in}$ and the shape of X_{out} be $U_{out} \times V_{out} \times Q_{out}$, where U, V, Q represent the height, width, and number of channels respectively, so when

X_{in} is a color image, its shape should be $U_{in} \times V_{in} \times 3$. The STN works identically on each channel to keep the consistency between the channels, so the number of channels of the input and output are the same ($Q = Q_{in} = Q_{out}$). A typical STN network consists of three parts shown in Figure 9: localization network, grid generator, and the sampler.

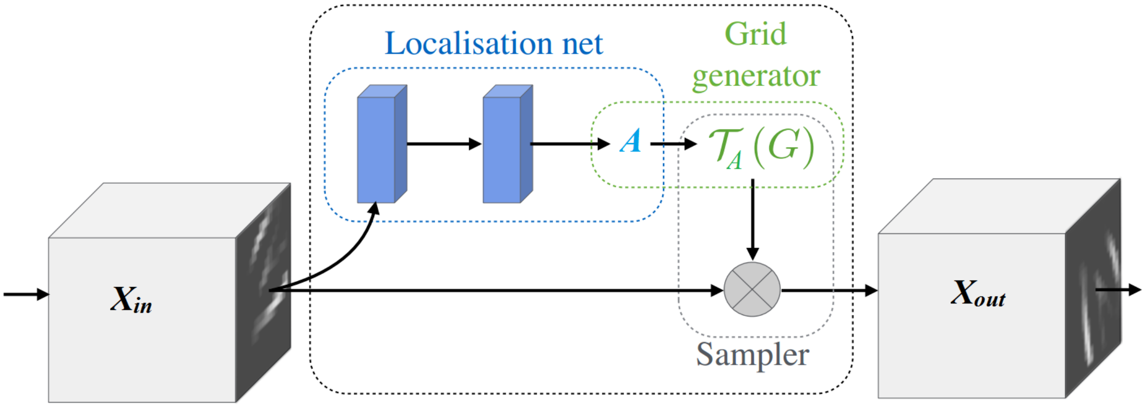


Figure 9: The structure of STN. The figure is taken from [25].

At each decoding time j, the localization network $\phi_{W_{loc}}$ takes the input X_{in}, the previous hidden state of the decoder $\boldsymbol{h}_{j-1}$ as input, and generates the transformation parameter A_j as output:

$$A_j = \phi_{W_{loc}}(X_{in}, \boldsymbol{h}_{j-1}) \tag{25}$$[2]

Then the transformation τ parameterized by A_j is applied on a mesh grid G generated by the grid generator to produce a feature map S which indicates how to select pixels[3] from X_{in} and map them to X_{out}. In detail, the grid G consists of pixels of the output feature map X_{out}, i.e.,

$$G = \{G_i\} = \left\{ \left(x_{1,1}^{X_{out}}, y_{1,1}^{X_{out}} \right), \left(x_{1,2}^{X_{out}}, y_{1,2}^{X_{out}} \right), \ldots, \left(x_{U_{out},V_{out}}^{X_{out}}, y_{U_{out},V_{out}}^{X_{out}} \right) \right\} \tag{26}$$

where x and y represent the coordinates of the pixel, and $\left(x_{1,1}^{(X_{out})}, y_{1,1}^{(X_{out})} \right)$ is the pixel of X_{out} which locates at coordinate $(1, 1)$. And we have

$$S_i = \tau_{A_j}(G_i) \tag{27}$$

$$S = \{S_i\} = \left\{ \left(x_{1,1}^S, y_{1,1}^S \right), \left(x_{1,2}^S, y_{1,2}^S \right), \ldots, \left(x_{U_{out},V_{out}}^S, y_{U_{out},V_{out}}^S \right) \right\} \tag{28}$$

where each S_i shows the coordinates in the input feature map that defines a sampling position. Figure 10 gives a visual example illustrating how the transformation and the grid

[2] Since the original STN in [25] is applied on a CNN, there is no hidden state. As a result, in [25] the location network only takes X_{in} as input, and Equation (25) changes to $A_j = \phi_{W_{loc}}(X_{in})$.

[3] Here the pixel means the element of the feature map X_{in}, which does not have to be an image.

generator work. In Figure 10, the grid G consists of the red dots in the right part of the figure. Then a transformation (the dashed green lines in the figure) is applied on G to generate S, which is the red (and blue) dots in the left part of the figure. Then S indicates the positions to perform the sampling.

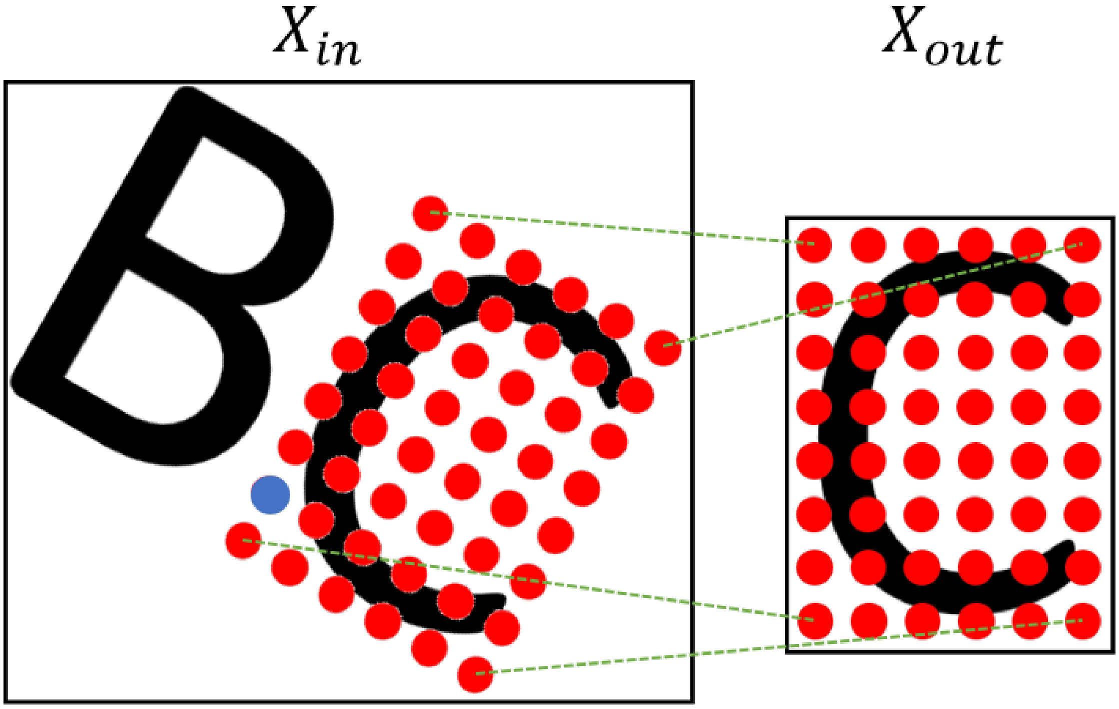


Figure 10: An example of the transformation $S = \tau_A(G)$, where the left part is X_{in}, the right part shows X_{out}. The red dots in the output image make up the grid generated by the grid generator as shown in Equation (26), and the red (and blue) dots in X_{in} are the elements of S as shown in Equation (28). The blue dot in X_{in} will be used to illustrate how the sampling works later.

Actually, τ can be any transformations, for example, the affine transformation, the plane projective transformation, or the thin plate spline. In order to make the descriptions above more clear, we assume τ is a 2D affine transformation, so A_j is a matrix consists of 6 elements:

$$A_j = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \tag{29}$$

Then Equation (27) can be rewritten as:

$$S_i = \begin{pmatrix} x_i^S \\ y_i^S \end{pmatrix} = \tau_{A_j}(G_i) = A_j \begin{pmatrix} x_i^{X_{out}} \\ y_i^{X_{out}} \\ 1 \end{pmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \begin{pmatrix} x_i^{X_{out}} \\ y_i^{X_{out}} \\ 1 \end{pmatrix} \tag{30}$$

The last step is the sampling which is operated by the sampler. Because S is calculated by a transformation, S_i does not always correspond exactly to the pixels in X_{in}, hence a sampling kernel is applied to map the pixels in X_{in} to X_{out}:

$$X_{out,i}^{q} = \sum_{u}^{U_{in}} \sum_{v}^{V_{in}} X_{in,u,v}^{q} k\left(x_i^S - v\right) k\left(y_i^S - u\right) \quad \forall i \in [1, 2, \ldots, U_{out}V_{out}] \quad \forall q \in [1, 2, \ldots, Q] \tag{31}$$

where k can be any kernel as long as it is partial differentiable with respect to both x_i^S and y_i^S, for example, one of the most widely used sampling kernel for images is the bilinear interpolation:

$$\begin{aligned} X_{out,i}^{q} = \sum_{u}^{U_{in}} \sum_{v}^{V_{in}} X_{in,u,v}^{q} \max\left(0, 1 - \left|x_i^S - v\right|\right) \max\left(0, 1 - \left|y_i^S - u\right|\right) \\ \forall i \in [1, 2, \ldots, U_{out}V_{out}] \quad \forall q \in [1, 2, \ldots, Q] \end{aligned} \tag{32}$$

when the coordinates of X_{in} and X_{out} are normalized, i.e., $\left(x_{1,1}^{X_{in}}, y_{1,1}^{X_{in}}\right) = (-1, -1)$ and $\left(x_{U_{in},V_{in}}^{X_{in}}, y_{U_{in},V_{in}}^{X_{in}}\right) = (+1, +1)$. A visual example of the bilinear interpolation is shown in Figure 11, where the blue dot represents a sampling position, and its value is calculated by a weighted sum of its surrounding pixels as shown in Equation (32). In Figure 11, the surrounding pixels of the sampling position are represented by four red intersection symbols.

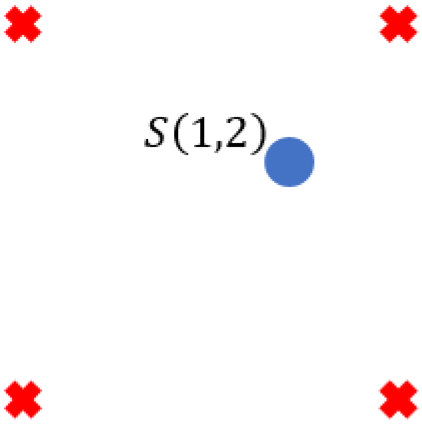


Figure 11: A visual example of the bilinear interpolation. The blue dot is an element in S which indicates the sampling position. In detail, it corresponds to the blue dot in Figure 10, where $S(1,2)$ means its coordinate is $(1,2)$. The four red intersection symbols represent the pixels in X_{in} which surround the sampling position.

Now if we review the whole process described above, the STN:

1. Accepts a feature map, and the previous hidden state of the decoder as input.

2. Generates parameters for a pre-defined transformation.

3. Generates the sampling grid and calculates the corresponding sampling positions in the input feature map based on the transformation.

4. Calculates the output pixel values by a pre-defined sampling kernel.

Therefore, for an input feature map X_{in}, an output feature map X_{out} is generated which has the ability to only focus on some interesting parts in X_{in}. For instance, the affine transformation defined in Equation (30) allows translation, rotation, scale, skew, and cropping, which is enough for most of the image related tasks. The whole process introduced above, i.e., feature map transformation, grid generation, and sampling, is inspired by the standard texture mapping in computer graphics.

At last the X_{out} is fed into the encoder to generate the intermediate code $\boldsymbol{c}$:

$$\boldsymbol{c} = \phi_{W_{enc}}(X_{out}) \tag{33}$$

2.5 Learning of the attention based RNN models

As mentioned in the introduction, the difference between the soft and hard attention mechanisms in optimization is that the soft attention module is differentiable with respect to its parameters so the standard gradient ascent/decent can be used for optimization. However, for the hard attention, the model is non-differentiable and the techniques from the reinforcement learning are applied for optimization.

2.5.1 Soft attention

For the item-wise soft attention mechanism, when f_{att} in Equation (12) is differentiable with respect to its inputs, it is clear that the whole attention model is differentiable with respect to W_{att}, as well as the whole RNN model is differentiable with respect to $\theta = [W_{enc}, W_{dec}, W_{att}]$. As a result, the same sum of log-likelihood in Equation (8) is used as the objective function for the learning, and the gradient ascent is used to maximize the objective.

The location-wise soft attention module (STN) is also differentiable with respect to its parameters if the location network $\phi_{W_{loc}}$, the transformation τ and the sampling kernel k are carefully selected to ensure their gradients with respect to their inputs can be defined. For example, when using a differentiable location network $\phi_{W_{loc}}$, the affine transformation (Equation (29), Equation (30)), and the bilinear interpolation (Equation (32)) as the sampling kernel, we have:

$$\frac{\partial X^q_{out,i}}{\partial X^q_{in,u,v}} = \sum_u^{U_{in}} \sum_v^{V_{in}} \max\left(0, 1 - \left|x^S_i - v\right|\right) \max\left(0, 1 - \left|y^S_i - u\right|\right) \tag{34}$$

$$\frac{\partial X^q_{out,i}}{\partial x^S_i} = \sum_u^{U_{in}} \sum_v^{V_{in}} X^q_{in,u,v} \max\left(0, 1 - \left|y^S_i - u\right|\right) \begin{cases} 0 & \text{if } \left|v - x^S_i\right| \geq 1 \\ 1 & \text{if } v \geq x^S_i \\ -1 & \text{if } v < x^S_i \end{cases} \tag{35}$$

$$\frac{\partial X^q_{out,i}}{\partial y^S_i} = \sum_u^{U_{in}} \sum_v^{V_{in}} X^q_{in,u,v} \max\left(0, 1 - \left|x^S_i - v\right|\right) \begin{cases} 0 & \text{if } \left|u - y^S_i\right| \geq 1 \\ 1 & \text{if } u \geq y^S_i \\ -1 & \text{if } u < y^S_i \end{cases} \tag{36}$$

$$\frac{\partial x^S_i}{\partial a_{1,1}} = x^{X_{out}}_i \tag{37}$$

$$\frac{\partial x^S_i}{\partial a_{1,2}} = y^{X_{out}}_i \tag{38}$$

$$\frac{\partial x^S_i}{\partial a_{1,3}} = 1 \tag{39}$$

$$\frac{\partial y^S_i}{\partial a_{2,1}} = x^{X_{out}}_i \tag{40}$$

$$\frac{\partial y^S_i}{\partial a_{2,2}} = y^{X_{out}}_i \tag{41}$$

$$\frac{\partial y^S_i}{\partial a_{2,3}} = 1 \tag{42}$$

Now the gradients of W_{loc} ($\frac{\partial A_j}{\partial W_{loc}}$) can be easily derived from Equation (25). As a result, the entire STN is differentiable with respect to its parameters. As in the item-wise soft attention, the objective function is the sum of the log-likelihood, and the gradient ascent is used for optimization.

2.5.2 Hard attention

As shown in Section 2.4.2 and Section 2.4.3, the hard attention mechanism stochastically picks an item or a sub-region from the input. In this case, the gradients with respect to the picked item/region are zero because they are discrete. The zero gradients cannot be used to maximize the sum of the log-likelihood by the standard gradient ascent. In general, this is a problem of training a neural network with discrete values/units, and here we just introduced the method applied in [39, 4, 57] which employs the techniques from the reinforcement learning. The learning methods for the item-wise and location-wise hard attention are essentially the same, but the item-wise hard attention implicitly sets the number of glimpse as 1. In this section, we just make the number of glimpses as M.

Instead of the raw log-likelihood in Equation (7), a new objective L'_j is defined which is a variational lower bound on the log-likelihood $\log p\left(\boldsymbol{y}_i | X, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{i-1}, \theta\right)$ [4] in Equation (7) parameterized by θ ($\theta = [W_{enc}, W_{dec}, W_{att}]$). And we have

[4] For notational simplicity, we ignore the $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{i-1}, \theta$ in the log-likelihood later, so $\log p\left(\boldsymbol{y}_i | X, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{i-1}, \theta\right) = \log p\left(\boldsymbol{y}_i | X\right)$

$$\begin{aligned}\text{log-likelihood} = L_j &= \log p(\boldsymbol{y}_j|X) \\ &\geqslant \log \sum_{l_j} p(l_j|X) p(\boldsymbol{y}_j|l_j, X) \\ &= \sum_{l_j} p(l_j|X) \log p(\boldsymbol{y}_j|l_j, X) \\ &= \sum_{l_j} p(l_j|\boldsymbol{x}^l) \log p(\boldsymbol{y}_j|l_j, \boldsymbol{x}^l) = L'_j\end{aligned} \tag{43}$$

Then the derivative of L'_j is:

$$\begin{aligned}\frac{\partial L'_j}{\partial \theta} &= \sum_{l_j} \left(p(l_j|\boldsymbol{x}^l) \frac{\partial \log p(\boldsymbol{y}_j|l_j, \boldsymbol{x}^l)}{\partial \theta} + \log p(\boldsymbol{y}_j|l_j, \boldsymbol{x}^l) \frac{\partial p(l_j|\boldsymbol{x}^l)}{\partial \theta} \right) \\ &= \sum_{l_j} p(l_j|\boldsymbol{x}^l) \left(\frac{\partial \log p(\boldsymbol{y}_j|l_j, \boldsymbol{x}^l)}{\partial \theta} + \log p(\boldsymbol{y}_j|l_j, \boldsymbol{x}^l) \frac{\partial \log p(l_j|\boldsymbol{x}^l)}{\partial \theta} \right)\end{aligned} \tag{44}$$

As shown above, l_j is generated from a distribution (Equation (15), Equation (17), or Equation (18)), which indicates that $p(l_j|x^l)$ and $\frac{\partial \log p\left(l_j|x^l\right)}{\partial \theta}$ can be estimated by a Monte Carlo sampling as demonstrated in [56]:

$$\frac{\partial L'_j}{\partial \theta} \approx \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\partial \log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l)}{\partial \theta} + \log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l) \frac{\partial \log p(l_j^m|\boldsymbol{x}^l)}{\partial \theta} \right) \tag{45}$$

The whole learning process described above for the hard attention is equivalent to the REINFORCE learning rule in [56], and from the reinforcement learning perspective, after each step, the model can get a reward from the environment. In Equation (45), the $\log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l)$ is used as the reward R_j. But in the reinforcement learning, the reward can be assigned to an arbitrary value. Depending on the tasks to be solved, here are some widely used schemes:

- Set the reward to be exactly $R_j = \log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l)$.
- Set the reward to be a real value proportional to $\log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l)$, which means $R_j = \beta \log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l)$, and β is a hyperparameter.
- Set the reward as a zero/one discrete value:

$$R_j = \begin{cases} 1 & \boldsymbol{y}_j = \underset{\boldsymbol{y}_j}{\operatorname{argmax}} \log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l) \\ 0 & \text{otherwise.} \end{cases} \tag{46}$$

Then Equation (45) can be written as:

$$\frac{\partial L'_j}{\partial \theta} \approx \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\partial \log p(\boldsymbol{y}_j|l_j^m, \boldsymbol{x}^l)}{\partial \theta} + R_j \frac{\partial \log p(l_j^m|\boldsymbol{x}^l)}{\partial \theta} \right) \tag{47}$$

Now as shown in [39], although Equation (47) is an unbiased estimation of the real gradient of Equation (44), it may have high variance because of the unbounded R_j. As a result, usually a variance reduction item b is added to the equation:

$$\frac{\partial L'_j}{\partial \theta} \approx \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\partial \log p(\boldsymbol{y}_j | l_j^m, \boldsymbol{x}^l)}{\partial \theta} + (R_j - b_j) \frac{\partial \log p(l_j^m | \boldsymbol{x}^l)}{\partial \theta} \right) \tag{48}$$

where b can be calculated in different ways which are discussed in [4, 5, 39, 53, 57]. A common approach is to set $b_j = \mathbb{E}(R)$. By using the variance reduction variable b, the training becomes faster and more robust. At last, a hyperparameter λ can be added to balance the two gradient components:

$$\frac{\partial L'_j}{\partial \theta} \approx \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\partial \log p(\boldsymbol{y}_j | l_j^m, \boldsymbol{x}^l)}{\partial \theta} + \lambda (R_j - b_j) \frac{\partial \log p(l_j^m | \boldsymbol{x}^l)}{\partial \theta} \right) \tag{49}$$

With the gradients in Equation (49), now the hard attention based RNN model can also be trained by gradient ascent. For the whole sequence Y, we have the new objective function $L'(Y, X,)$:

$$\begin{aligned} L'(Y, X, \theta) &= \sum_{j=1}^{T'} \sum_{l_j} p\left(l_j | \boldsymbol{x}^l\right) \log p\left(\boldsymbol{y}_j | l_j, \boldsymbol{x}^l\right) \\ &= \sum_{j=1}^{T'} \sum_{l_j} p\left(l_j | X\right) \log p\left(\boldsymbol{y}_j | l_j, X\right) \\ &\leqslant \sum_{j=1}^{T'} \log \sum_{l_j} p(\boldsymbol{y}_j | X) \\ &= \sum_{j=1}^{T'} \log p(\boldsymbol{y}_j | X) \\ &= \sum_{j=1}^{T'} \log p\left(\boldsymbol{y}_j | X, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{j-1}, \theta\right) = \text{sum of the log-likelihood} \end{aligned} \tag{50}$$

2.6 Summary

In this section, we firstly give a brief introduction of the neural network, the CNN, and the RNN. Then we discuss the common RNN model for sequence to sequence problem. And the detailed descriptions of four types of attention mechanisms are given: item-wise soft attention, item-wise hard attention, location-wise hard attention, and location-wise soft attention, followed by how to optimize the attention based RNN models.

3. Applications of the attention based RNN model in computer vision

In this section, we firstly give a brief overview of the differences when one uses different attention based RNN models. And then two applications in computer vision which apply the attention based RNN model are introduced.

3.1 Overview

- As mentioned above, the item-wise attention model requires that the input sequence consists of explicit items, which is not trivial for some types of input, like an image input. A simple solution is to manually extract some patches from the input images and treat them as a sequence of item. For the extraction method, one can either extract some fixed size patches from some pre-defined locations, or apply other object proposal methods, like [49].

- The location-wise attention model can directly accept a feature map as input which avoids the ”items extraction” step mentioned above. And if compared to human perception, the location-wise attention is more appealing. Because when human being sees an image, he will not manually divide the image into some patches and calculate the weights for them before recognizing the objects in the image. Instead, he will directly focus on an object and its surroundings. That is exactly what the location-wise attention does.

- The item-wise attention model has to calculate the codes for all items in the input sequence, which is less efficient than the location-wise attention model.

- The soft attention is differentiable with respect to its inputs, so the model can be trained by optimizing the sum of the log-likelihood using gradient ascent. However, the hard attention is non-differentiable, usually the techniques from the reinforcement learning are applied to maximize a variational lower bound of the log-likelihood, and it makes sense to estimate there will be a sacrifice on the performance.

- The STN keeps both the advantages of the location-wise attention and the soft attention, where it can directly accept a feature map as input and optimize the raw sum of log-likelihood. However, it is just recently proposed, and originally applied on CNN. Currently, there are not enough applications of the STN based RNN model.

3.2 Image classification and object detection

Image classification is one of the most classical and fundamental applications in computer vision area, where each image has a (some) label(s), and the model needs to learn how to predict the label given a new input image. Now the most popular and successful model for image classification task is the CNN. While as mentioned above, the CNN takes one fixed size vector as input, which has some disadvantages:

- When the input image is larger than the input size accepted by the CNN, either the image needs to be rescaled to a smaller size to meet the requirements of the model, or the image needs to be cut into some subparts to be processed one by one. If the image is rescaled smaller, there are some sacrifices on the details of the image, which may harm the classification performance. On the other hand, if the image is cut into some patches, the amount of computation will scale linearly with the size of the image.

- The CNN has a degree of spatial transformation invariance build-in, while when the image is noisy or the object indicated by the label only occupies a small region of the input image, a CNN model may not still keep satisfied performance.

As a result, [39] and [4] propose the RNN based image classification models with the location-wise hard attention mechanism. As explained above, the location-wise hard attention based model stochastically picks a sub-region from the input image to generate the intermediate code. So the models in [39] and [4] can not only make prediction of the image label, but also localize the position of the object. This means the attention based RNN model integrates the image classification and object detection into a single end-to-end trainable model, which is another advantage compared to the CNN based object detection model. If one wants to apply a convolution network to solve the object detection problem, he has to use a separate model to propose the potential locations of the objects, like [49], which is expensive.

In this section, we will briefly introduce and analyze the models proposed in [39], [4], and their extensions. A brief structure of the RNN model in [39] is shown in Figure 12. In this model, *enc* is the encoder neural network taking a patch x of the input image to generate the intermediate code, where x is cut from the raw image based on a location value l generated by the attention network. In detail, l is generated by Equation (18), and x is extracted by Equation (19). Then the decoder accepts the intermediate code as input to make the potential prediction. Here in Figure 12 the decoder and the attention network are put in the same network $r^{(1)}$.

The experiments in [39] compare the performances of the proposed model with some non-recurrent neural networks, and the results show the superiority of the attention based RNN model to the non-recurrent neural networks with similar number of parameters, especially on the datasets with noise. For the details of the experiments, one can refer to Section 4.1 in [39].

However, the original model shown in Figure 12 is very simple, where *enc* is a two-layer neural network, and $r^{(1)}$ is a three-layer neural network. Besides, the first glimpse (l_1 in Figure 12) is assigned manually. And all the experiments are only conducted on some toy datasets. There is no evidence to prove the generalization power of the model in Figure 12 to some real-world problems. [4] proposes an extended version of model as shown in Figure 13 by firstly making the network deeper, and secondly using a context vector to obtain a better first glimpse.

The encoder (*enc*) in [4] is deeper, i.e., it consists of three convolutional layers and one fully connected layer. Another big difference between the model in Figure 13 and the model in Figure 12 is that the model in Figure 13 adds an independent layer $r^{(2)}$ as the attention network in order to make the first glimpse as accurate as possible, i.e., the model extracts a context vector from the whole image (I_{coarse} in the figure), and feeds it to the attention model to generate the first potential location l_1. The same context vector of the whole image is not fed into the decoder network $r^{(1)}$, because [4] observes that if so, the predicted

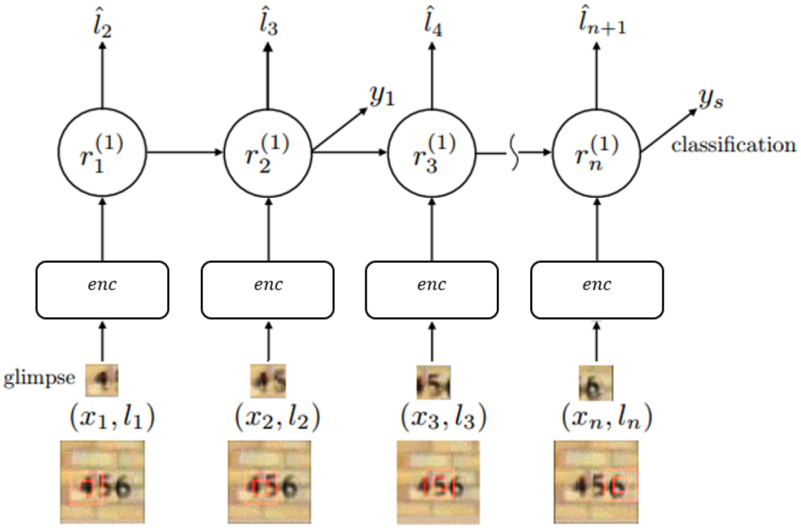


Figure 12: The model proposed in [39]. *enc* is the encoder, $r^{(1)}$ is the decoder and the attention model. $\hat{l}$ is equivalent to the output of $f_{att}()$ in Equation (18), which is used to generate l as shown in Equation (18). x is the sub-region of the whole image extracted based on l, and y_s is the predicted label for the image. The figure is taken from [4], and some modifications are made to fit the model in [39].

Table 1: Error rates on the MNIST pairs image classification dataset.

Model	Test error
[39]	9%
[4] without the context vector	7%
[4]	**5%**

label are highly influenced by the information of the whole image. Besides, both $r^{(1)}$ and $r^{(2)}$ are recurrent, while in Figure 12, only one hidden state exists.

Both two models introduced above are evaluated on a dataset called ”MNIST pairs” which is generated from the MNIST dataset. MNIST [29] is a handwritten digits dataset, which contains 70000 28×28 binary images (Figure 14a). The MNIST pairs dataset randomly puts two digits into a 100×100 image with some additional noise in the background [39] (Figure 14b). The results are shown in Table 1. It is clear that the performance of [4] is better, and the context vector indeed makes some improvements by suggesting a more accurate first glimpse.

The model in [4] are also tested on multi-digit street view house number (SVHN) [41] sequence recognition task, and the results show that with the same level of error rates, the number of parameters to be learned, and the training time of the attention based RNN model are much less than the state-of-the-art CNN methods.

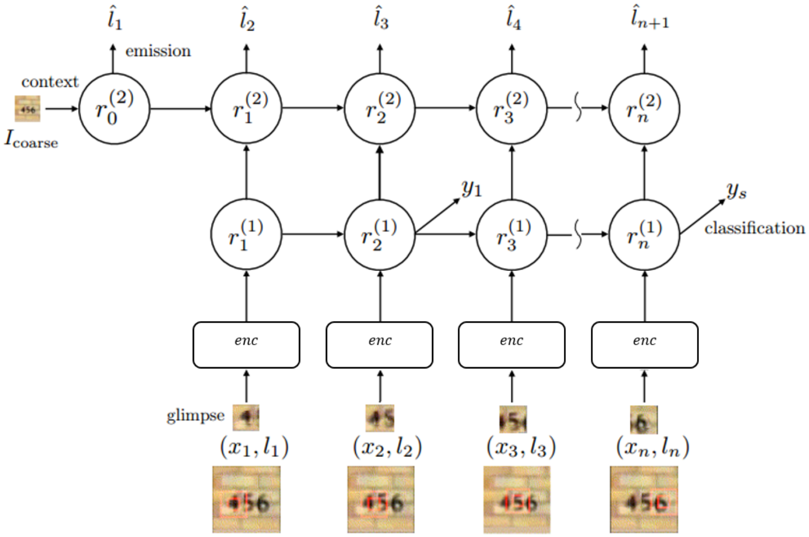
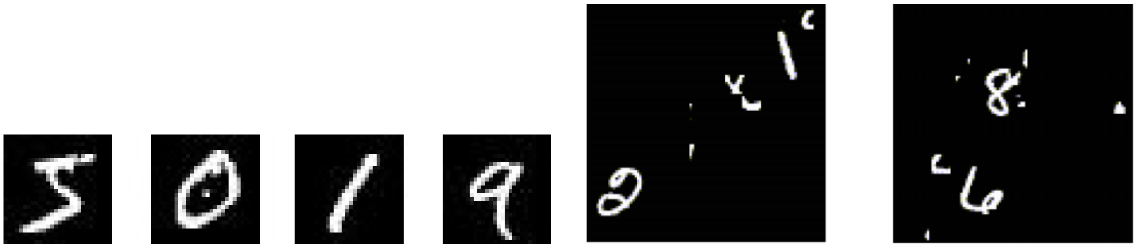


Figure 13: The model proposed in [4]. *enc* is the encoder, $r^{(1)}$ is the decoder network, and $r^{(2)}$ is the attention network. I_{coarse} is a context vector extracted from the whole input image to generate the location of the first glimpse. The other terms are the same as in Figure 12. The figure is taken from [4], and some modifications are made.



(a) Four examples taken from the MNIST dataset.

(b) Two examples taken from the MNIST pairs dataset [39].

Figure 14: Examples of MNIST and MNIST pairs datasets.

However nowadays both MNIST and SVHN are toy datasets with constraints environments, i.e., both of them are about digits classification and detection. Besides, their sizes are relatively small compared to other popular image classification/detection datasets, like the ImageNet. [45] further extends the model in [4] with only a few modifications and applies it to a real-world image classification task: Stanford Dogs fine-grained categorization task [27], in which the images have larger clutter, occlusion, and variations in pose. The biggest change made in [45] is that a part of the GoogLeNet [48] is used as the encoder, which is a pre-trained CNN on ImageNet dataset. The performance shown in Table 2 indicates that the attention based RNN model indeed can be applied to real-world datasets while still keep a competitive performance.

Table 2: The classification performance. * represents the model uses additional bounding boxes around the dogs in training and testing.

Model	Mean accuracy (MA) [7]
[58]*	0.38
[7]*	0.46
[18]*	0.50
GoogLeNet 96×96 (the encoder in [45])	0.42
RNN with location-wise hard attention [45]	**0.68**

3.3 Image caption

Image caption is a very challenging problem: by given an input image, the system needs to generate a natural language sentence to describe the content of the image. The classical way of solving this problem is by dividing the problem into some sub-problems, like object detection, objects-words alignment, sentence generation by the template, etc., and solving them independently. This process will obviously make some sacrifices on the performance. With the development and success of the recurrent network in machine translation area, the image caption problem can also be treated as a machine translation problem, i.e., the image can also be seen as a language, and the system just translate it to another language (natural language, like English). The RNN based image caption system is recently proposed in [51] (Neural Image Caption, NIC) which perfectly fits our encoder-decoder RNN framework without the attention mechanism.

The general structure of the NIC is shown in Figure 15. With an input image, the same trick mentioned above is applied: a pre-trained convolutional network is used as the encoder, and the output of a fully-connected layer is the intermediate code. Then it is followed by a recurrent network serving as the decoder to generate the natural language caption, and in NIC, the LSTM unit is used. Compared to the pervious methods which decompose the image caption problem into some sub-problems, the NIC is end-to-end, and is directly trained on the raw data, so the whole system is much simpler and can keep all information of the input data. The experimental results show the superiority of the NIC compared to the traditional methods.

However, one problem for the NIC is that only a single image representation (the intermediate code) is obtained by the encoder from the whole input image, which is counterintuitive, i.e., when human beings describe an image by natural language, usually some objects or some salient areas are focused on one by one. So it is natural to import the ability of ”focusing” by applying the attention based RNN model.

[57] adds the item-wise attention mechanisms into the NIC system. The encoder is still a pre-trained CNN, while instead of the fully-connected layer, the output of a convolutional layer is used to calculate the code set C in Equation (9). In detail, at each decoding step the whole image is fed into the pre-trained CNN, and the output of the CNN is a 2D feature map. Then some pre-defined 14×14 sub-regions of the feature map are extracted as the items in C. [57] implements both the item-wise soft and the item-wise hard attention

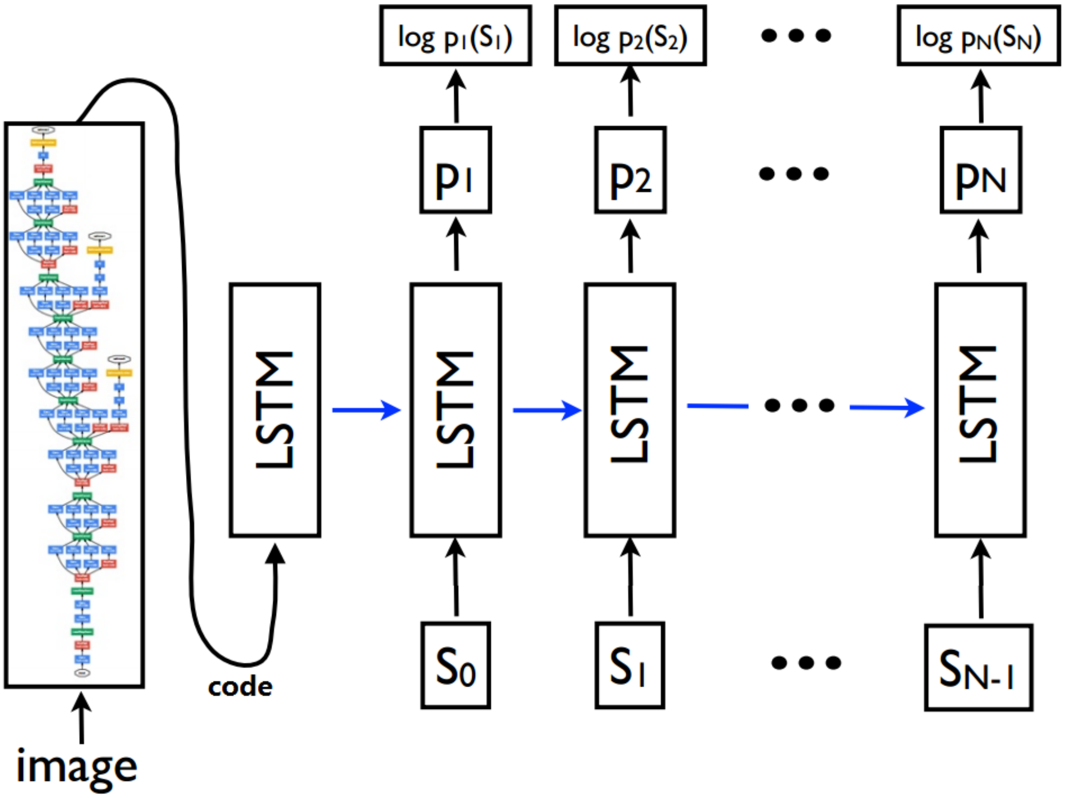


Figure 15: NIC. The image is firstly processed by a pre-trained CNN and a code is generated, and then the code is decoded by a LSTM unit. $S_i, i = 1, 2, \ldots, N$ are the word in the image caption sentence. $P_i, i = 1, 2, \ldots, N$ are the log-likelihoods of the predicted words and the ground truth words. The figure is taken from [51].

mechanisms. For the item-wise soft attention, at each decoding step a weight for each code in C is calculated by Equation (12) and Equation (13), and then an expectation of all codes in C (Equation (14)) is used as the intermediate code. For the item-wise hard attention, the same weights for all codes in C are calculated and the index of the code to be picked is generated by Equation (15).

By using the attention mechanism, [57] reports improvements compared to the common RNN models without attention mechanism including NIC. However, both NIC and [57] have participated in the MS COCO Captioning Challenge 2015[5], and the competition gives opposite results. MS COCO Captioning Challenge 2015 is an image caption competition running on one of the biggest image caption datasets: Microsoft COCO [32]. Microsoft COCO contains 164k images in total where each image is labeled by at least 5 natural language captions. The performance is evaluated by human beings as well as some commonly used evaluation metrics (BLEU [42], Meteor [14], ROUGE-L [31] and CIDEr-D [50]). There

[5] MS COCO Captioning Challenge 2015, http://mscoco.org/dataset/#captions-challenge2015

Table 3: Rankings of some teams of the MS COCO Captioning Challenge. The rows are sorted by M1. For more details of the definitions of M1, and M2, one can refer to [5]. For team "Human", the captions are generated by human beings.

Model	Human		Other evaluation metrics						
	M1	M2	BLEU-1	BLEU-2	BLEU-3	BLEU-4	Meteor	ROUGE-L	CIDEr-D
Human	1	1	6	12	12	13	3	11	6
Google NIC	2	3	2	2	2	2	1	1	1
MSR [17]	3	2	5	5	5	5	4	5	4
[57]	4	5	9	9	9	9	6	8	9
MSR Captivator [15]	5	4	1	1	1	1	2	2	2
Berkeley LRCN [16]	6	6	10	7	8	8	7	6	8

are 17 teams participating the competition in total, and here we list the rankings of some top teams in Table 3 including [57].

Except for the team "Human" and the system proposed by [57], all other teams shown in Table 3 use models without the attention mechanism. We can notice that the attention based model performs worse than NIC on both the human evaluations and the automatic evaluation metrics, which suggests that there are still many spaces for improvements. In addition, by comparing the human and the automatic evaluation metrics, the attention based RNN model performs worse in the automatic evaluation metrics than in the human evaluations. This may suggest that, on one side, the current automatic evaluation metrics are still not perfect, and on the other side, the attention based RNN model can generate captions which are more "natural" for human beings.

In addition to the performance, another big advantage of the attention based RNN model is that it makes the visualization easier and much more intuitive as mentioned in Section 1.5. Figure 16 gives a visual example showing where the model attends in the decoding process. It is clear the attention mechanism indeed works, for example, when generating the word "people", the model focuses on the people, and when generating "water", the lake is attended.

The model in [57] uses fixed-size feature maps (14×14) to construct the code set C, and each feature map corresponds to a fixed-size patch of the input image. [26] claims that this design may harm the performance because some "meaningful scenes" may only occupy a small part of the image patch or cannot be cover by a single patch, where the meaningful scene indicates the objects/scenes correspond to the word is about to be predicted. [26] makes an improvement by firstly extract many object proposals [49] from an input image, which potentially contain the meaningful scenes, as the input sequence. And then each proposal is used to calculate the code in C. However, [26] neither directly compares its performance to [57], nor attends the MS COCO Captioning Challenge 2015. So here we cannot directly measure if and how the performance will be improved by using the object proposals to construct the input sequence. When analyzing in theory, we doubt if the improvements can really be obtained, because the currently popular object proposal methods only extract regions which probably contain some "objects". But a verb or an adjective in the ground truth image caption may not correspond to any explicit objects. Still, the problem proposed in [26] is very interesting and cannot be ignored: when the input of a RNN model is an

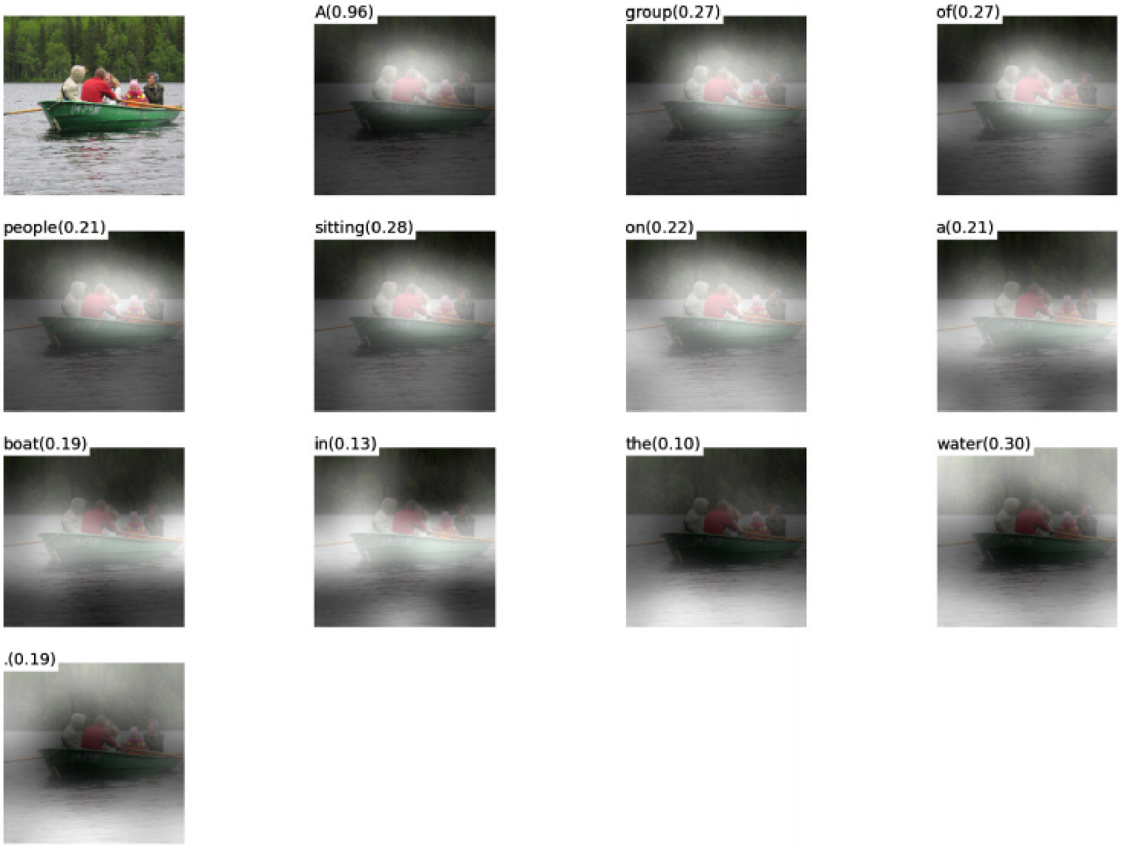


Figure 16: A visual example of generating the caption with the item-wise soft attention based RNN model, where the white area indicates the position the model focuses on. The word at the top left corner is the word generated by the model. The figure is taken from [57].

image, it is not trivial to obtain a sequence of items from it. One straightforward solution is to use the location-wise attention models like the location-wise hard attention or the STN.

4. Conclusion and future research

In this survey, we discuss the attention based RNN model and its applications in some computer vision tasks. Attention is an intuitive methodology by giving different weights to different parts of the input which is widely used in the computer vision area. Recently with the development of the neural network, the attention mechanism is also embedded to the RNN model to further enhance its ability to solve sequence to sequence problem. We illustrate how the common RNN works, and gives detailed descriptions of four different attention mechanisms, and their pros and cons are also analyzed.

- The item-wise attention based model requires that the input sequence contains explicit items. For some types of the input, like an image input, an additional step is needed to extract items from the input. Besides, the item-wise attention based RNN model needs to feed all items in the input sequence to the encoder, so on one side, it results in a slow training process, and on the other side, the testing efficiency is also not improved compared to the RNN without the attention module.

- The location-wise attention based model allows the input to be an entire feature map. At each time, the location-wise attention based model only focuses on a part of the input feature map, which gives efficiency improvements in both training and testing compared to the item-wise attention model.

- The soft attention module is differentiable with respect to its inputs, so the standard gradient ascent/decent can be used for optimization.

- The hard attention module is non-differentiable, and the objective is optimized by some techniques from the reinforcement learning.

At last, some applications in computer vision which apply the attention based RNN models are introduced. The first application is image classification and object detection which applies the location-wise hard attention mechanism. The second one is the image caption which uses both the item-wise soft and hard attention. The experimental results demonstrate that

- Considering the performance, the attention based RNN model is better than, or at least comparable to, the model without the attention mechanism.

- The attention based RNN model makes the visualization more intuitive.

The location-wise soft attention model, i.e., the STN, is not introduced in the applications, because it is firstly proposed on the CNN and currently there are not enough applications of the STN based RNN model.

Since the attention based RNN model is a new topic, it has not been well addressed yet. As a result, there are many problems which either need theoretical analyses or practical solutions. Here we list a few of them:

- A large majority of the current attention based RNN models are only applied on some small datasets. How to use the attention based RNN model to larger datasets, like the ImageNet dataset?

- Currently, most of the experiments are performed to compare the RNN models with/without the attention mechanism. More comprehensive comparisons of different attention mechanisms for different tasks are needed.

- In many applications the input sequences contain explicit items naturally, which makes them fit the item-wise attention model. For example, most of the applications in NLP treat the natural sentence as the input, where each item is a word in the sentence. Is it possible to convert these types of inputs to a feature map and apply the location-wise attention model?

- The current hard attention mechanism uses techniques from the reinforcement learning in optimization by maximizing an approximation of the sum of log-likelihood, which makes a sacrifice on the performance. Can the learning method be improved? [5] gives some insights but it is still an open question.
- How to extend the four attention mechanisms introduced in this survey? For example, an intuitive way of extension is to put multiple attention modules in parallel instead of only embedding one into the RNN model. Another direction is to further generalize an RNN and make it have a similar structure as the modern computer architecture [54, 20], e.g. a multiple cache system shown in Figure 1, where the attention module serves as a ”scheduler” or an addressing module.

In summary, the attention based RNN model is a newly proposed model, and there are still many interesting questions remaining to be answered.

References

[1] Central processing unit cache memory. http://www.pantherproducts.co.uk/Articles/CPU/CPU%20Cache.shtml.

[2] Radhakrishna Achanta, Francisco Estrada, Patricia Wils, and Sabine Süsstrunk. Salient region detection and segmentation. In *Computer Vision Systems*, pages 66–75. Springer, 2008.

[3] John R Anderson. *Cognitive psychology and its implications* . WH Freeman/Times Books/Henry Holt & Co, 1990.

[4] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[5] Jimmy Ba, Ruslan R Salakhutdinov, Roger B Grosse, and Brendan J Frey. Learning wake-sleep recurrent attention models. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Yuning Chai, Victor S. Lempitsky, and Andrew Zisserman. Symbiotic segmentation and part localization for fine-grained categorization. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 321–328, 2013. doi: 10.1109/ICCV.2013.47. URL http://dx.doi.org/10.1109/ICCV.2013.47.

[8] William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015.

[9] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014.

[10] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder-decoder networks. *Multimedia, IEEE Transactions on*, 17(11):1875–1886, 2015.

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255, 2009. doi: 10.1109/CVPRW.2009.5206848. URL http://dx.doi.org/10.1109/CVPRW.2009.5206848.

[13] Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural Computation*, 24(8):2151–2184, 2012. doi: 10.1162/NECO_a_00312. URL http://dx.doi.org/10.1162/NECO_a_00312.

[14] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, volume 6, 2014.

[15] Jacob Devlin, Hao Cheng, Hao Fang, Saurabh Gupta, Li Deng, Xiaodong He, Geoffrey Zweig, and Margaret Mitchell. Language models for image captioning: The quirks and what works. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*, pages 100–105, 2015. URL http://aclweb.org/anthology/P/P15/P15-2017.pdf.

[16] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Trevor Darrell, and Kate Saenko. Long-term recurrent convolutional networks for visual recognition and description. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 2625–2634, 2015. doi: 10.1109/CVPR.2015.7298878. URL http://dx.doi.org/10.1109/CVPR.2015.7298878.

[17] Hao Fang, Saurabh Gupta, Forrest N. Iandola, Rupesh K. Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C. Platt, C. Lawrence Zitnick, and Geoffrey Zweig. From captions to visual concepts and back. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1473–1482, 2015. doi: 10.1109/CVPR.2015.7298754. URL http://dx.doi.org/10.1109/CVPR.2015.7298754.

[18] Efstratios Gavves, Basura Fernando, Cees GM Snoek, Arnold WM Smeulders, and Tinne Tuytelaars. Fine-grained categorization by alignments. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1713–1720. IEEE, 2013.

[19] Abel Gonzalez-Garcia, Alexander Vezhnevets, and Vittorio Ferrari. An active search strategy for efficient object class detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3022–3031, 2015.

[20] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[22] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692, 2015.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[24] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (11):1254–1259, 1998.

[25] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2008–2016, 2015.

[26] Junqi Jin, Kun Fu, Runpeng Cui, Fei Sha, and Changshui Zhang. Aligning where to see and what to tell: image caption with region-based attention and scene factorization. *arXiv preprint arXiv:1506.06272*, 2015.

[27] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, 2011.

[28] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in neural information processing systems*, pages 1243–1251, 2010.

[29] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8, 2004.

[32] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, pages 740–755, 2014. doi: 10.1007/978-3-319-10602-1_48. URL http://dx.doi.org/10.1007/978-3-319-10602-1_48.

[33] Wang Ling, Chris Dyer, Alan W. Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1299–1304, 2015. URL http://aclweb.org/anthology/N/N15/N15-1142.pdf.

[34] Wang Ling, Yulia Tsvetkov, Silvio Amir, Ramon Fermandez, Chris Dyer, Alan W. Black, Isabel Trancoso, and Chu-Cheng Lin. Not all contexts are created equal: Better word representations with variable attention. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal,*

September 17-21, 2015, pages 1367–1372, 2015. URL http://aclweb.org/anthology/D/D15/D15-1161.pdf.

[35] Tie Liu, Zejian Yuan, Jian Sun, Jingdong Wang, Nanning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to detect a salient object. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(2):353–367, 2011.

[36] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Proceedings of AAAI*, 2016.

[37] Fandong Meng, Zhengdong Lu, Mingxuan Wang, Hang Li, Wenbin Jiang, and Qun Liu. Encoding source language with convolutional neural network for machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 20–30, 2015. URL http://aclweb.org/anthology/P/P15/P15-1003.pdf.

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[39] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2204–2212, 2014. URL http://papers.nips.cc/paper/5542-recurrent-models-of-visual-attention.

[40] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010. URL http://www.icml2010.org/papers/432.pdf.

[41] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5. Granada, Spain, 2011.

[42] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA.*, pages 311–318, 2002. URL http://www.aclweb.org/anthology/P02-1040.pdf.

[43] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal,*

September 17-21, 2015, pages 379–389, 2015. URL http://aclweb.org/anthology/D/D15/D15-1044.pdf.

[44] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. URL http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[45] Pierre Sermanet, Andrea Frome, and Esteban Real. Attention for fine-grained categorization. *arXiv preprint arXiv:1412.7054*, 2014.

[46] Søren Kaae Sønderby, Casper Kaae Sønderby, Henrik Nielsen, and Ole Winther. Convolutional LSTM networks for subcellular localization of proteins. In *Algorithms for Computational Biology - Second International Conference, AlCoB 2015, Mexico City, Mexico, August 4-5, 2015, Proceedings*, pages 68–80, 2015. doi: 10.1007/978-3-319-21233-3_6. URL http://dx.doi.org/10.1007/978-3-319-21233-3_6.

[47] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015. doi: 10.1109/CVPR.2015.7298594. URL http://dx.doi.org/10.1109/CVPR.2015.7298594.

[49] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. doi: 10.1007/s11263-013-0620-5. URL http://dx.doi.org/10.1007/s11263-013-0620-5.

[50] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 4566–4575, 2015. doi: 10.1109/CVPR.2015.7299087. URL http://dx.doi.org/10.1109/CVPR.2015.7299087.

[51] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3156–3164, 2015. doi: 10.1109/CVPR.2015.7298935. URL http://dx.doi.org/10.1109/CVPR.2015.7298935.

[52] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[53] Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001*, pages 538–545, 2001. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=141&proceeding_id=17.

[54] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[55] Wikipedia. Artificial neural network. https://en.wikipedia.org/wiki/Artificial_neural_network.

[56] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696. URL http://dx.doi.org/10.1007/BF00992696.

[57] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015. URL http://jmlr.org/proceedings/papers/v37/xuc15.html.

[58] Shulin Yang, Liefeng Bo, Jue Wang, and Linda G. Shapiro. Unsupervised template learning for fine-grained object recognition. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 3131–3139, 2012. URL http://papers.nips.cc/paper/4714-unsupervised-template-learning-for-fine-grained-object-recognition.

[59] Donggeun Yoo, Sunggyun Park, Joon-Young Lee, Anthony S Paek, and In So Kweon. Attentionnet: Aggregating weak directions for accurate object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2659–2667, 2015.