The Encoder-Decoder Framework and Its Applications

Ahmad Asadi[1], Reza Safabakhsh[2]

[1]Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran
ahmad.asadi@aut.ac.ir
[2]Faculty of Computing Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran
safa@aut.ac.ir

Abstract. The neural encoder-decoder framework has advanced the state-of-the-art in machine translation significantly. Many researchers in recent years have employed the encoder-decoder based models to solve sophisticated tasks such as image/video captioning, textual/visual question answering, and text summarization. In this work we study the baseline encoder-decoder framework in machine translation and take a brief look at the encoder structures proposed to cope with the difficulties of feature extraction. Furthermore, an empirical study of solutions to enable decoders to generate richer fine-grained output sentences is provided. Finally, the attention mechanism which is a technique to cope with long-term dependencies and to improve the encoder-decoder performance on sophisticated tasks is studied.

Keywords: Encoder-decoder framework, machine translation, image captioning, video caption generation, question answering, long-term dependencies, attention mechanism

1 Introduction

The solution to a considerable number of the problems that we need to solve falls into the category of encoder-decoder based methods. We may wish to design exceedingly complex networks to face sophisticated challenges like automatically describing an arbitrary image or translating a sentence from one language to another. The neural encoder-decoder framework has recently been exploited to solve a wide variety of challenges in natural language processing, computer vision, speech processing, and even interdisciplinary problems. Some examples of problems that can be addressed by the encoder-decoder based models are machine translation, automatic image and video caption generation, textual and visual question answering, and audio to text conversion.

The encoder part in this model is a neural structure that maps raw inputs to a feature space and passes the extracted feature vector to the decoder. The decoder is another neural structure that processes the extracted feature vector to make decisions or generate appropriate output for the problem.

A wide variety of encoders are proposed to encode different types of inputs. Convolutional neural networks (CNNs) are typically used in encoding image and video inputs. Recurrent neural networks (RNNs) are widely used as encoders where the input is a sequence of structured data or sentence. In addition, more complex structures of different neural networks have been used to model complexities in inputs. Hierarchical CNN-RNN structures are examples of neural combinations which are widely used to represent temporal dependencies in videos which are used in video description generation.

Another potential issue with this baseline encoder–decoder approach is that the encoder has to compress all the necessary information of the input into a fixed-size tensor. This may make it difficult for the neural network to model temporal dependencies at both the input and the output. Attention mechanism is introduced to overcome the problem of fixed-length feature extraction as an extension to the encoder–decoder model. The distinguishing feature of this approach from the baseline encoder–decoder is that it does not attempt to encode a whole input into a single fixed-size tensor. Instead, it encodes the input into a sequence of annotation vectors and selects a combination of these vectors adaptively, while decoding and generating the output in each step.

Some of the tasks in which the encoder-decoder model is used to solve the problem are as follows:

1. Machine Translation

"Machine translation" (MT) is the task of generating a sentence in a destination language which has the same meaning as the given sentence from a source language. Two different approaches exist in machine translation.

The first approach, called "statistical machine translation" (SMT), is characterized by the use of statistical machine learning techniques in order to automatically translate the sentence from the source language to the destination language. In less than two decades SMT has come to dominate academic machine translation research [1].

The second approach is called "Neural Machine Translation" (NMT). In this category, the encoder-decoder framework was first proposed by Cho et al. [2] in 2014. In the Cho et al. [2]'s model a neural network is used to extract features from the input sentence and another neural network is used to generate a sentence word by word from the destination language using the extracted feature vector.

In the neural structures used in NMT, a neural network is trained to map the input sequence (the input sentence as a sequence of words) to the output sequence. This kind of learning is known as "Sequence to Sequence Learning".

Evaluations on the early models of NMT showed that although the generated translations are correct, the model faces extreme problems when translating long sentences [3]. The problem of modeling "long-term dependencies" is one of the most important challenges in the encoder-decoder models. We will drill into that and take a look at the proposed solutions, later in this chapter.

2. Image/Video Captioning

Image captioning and video captioning are the problems of associating a textual description to a given image or video which holistically describes the objects and events presented in the input. A wide variety of approaches have been proposed to solve these problems, including probabilistic graphical models (PGMs) and neural encoder-decoder based models.

Encoder-decoder based models for image captioning use a CNN as an encoder to extract a feature vector from the input image and pass it to an RNN as the decoder to generate the caption. The model architecture in this task is the same as that of machine translation except that the encoder uses a CNN to encode the image rather than an RNN.

In video captioning, also called "video description generation", a similar model based on the encoder-decoder architecture is employed to generate a caption for the input video. In video captioning models, the encoder typically consists of CNNs or combination of CNNs and RNNs to encode the input video and the decoder is the same as the decoder in machine translation and image captioning.

3. Textual/Visual Question Answering

Textual and visual question answering are the problems of generating an answer to a given question about an article and about an input image, respectively. Models proposed to solve these problems are supposed to generate a short or long answer, given an article or an image, and a question about it as the input. The base model architecture is then similar to that of machine translation, except that the encoder is required to extract a feature vector for a pair of inputs. The decoder is the same as the decoder in machine translation and image/video captioning because it is supposed to generate a sentence describing the meaning of the feature vector generated by the encoder.

4. Text Summarization

Proposed models for summarizing a text are supposed to generate a textual summary for the input text. The only constraint on the output is that it is required to describe the same meaning as the input text and its length should be shorter than that of the input. The base architecture of these models is the same as the architecture proposed in machine translation, except that the generated output here is from the same language as the input.

It can be easily seen that the baseline architecture proposed in machine translation is also used in other tasks with minor changes. In addition, the decoders of the models in different tasks are similar since most of them are used to generate a sentence word by word to describe the meaning of the input represented by the feature vector. On the other hand, a wide variety of encoders are used in order to extract appropriate feature vectors depending on the input types in different tasks.

The next section of this chapter discusses the baseline encoder-decoder model. The first encoder-decoder based model proposed in machine translation is introduced in section 3. Section 4, discusses different types of encoders and their applications in

details and makes a general perspective of the encoder structures in different problems. Section 5, provides a comprehensive study of the decoder structures, techniques of making deeper decoders, along with their applications in image/video caption generation. Section 6, introduces the attention mechanism and its usage in machine translation, Followed by an empirical study of the attention mechanism in other problems.

2 Baseline Encoder-Decoder Model

In this section, we introduce the very baseline encoder-decoder model. To give a clear picture of the idea, the basic structure for solving the machine translation task is presented in which the model is designed to translate a sentence from a source language to a destination one.

2.1 Background

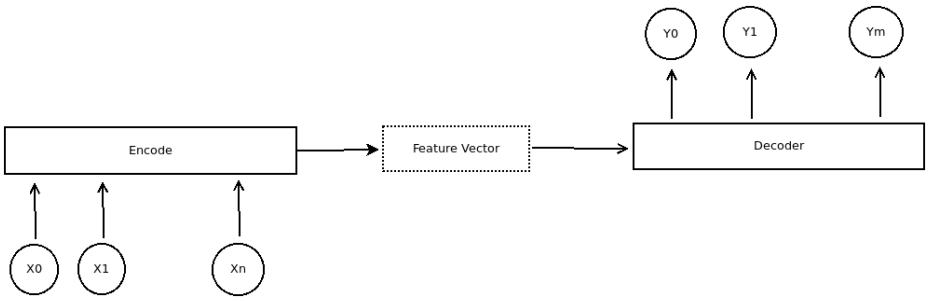
A wide range of problems in natural language processing, computer vision, speech recognition, and some multidisciplinary problems are solved by encoder-decoder based models. More specifically, some sophisticated problems in which generating an often-sequential output such as text is desired can be solved by models based on the encoder-decoder structure.

The main idea behind the framework is that the process of generating output can be divided into two subprocesses as follows:

1. Encoding phase: A given input is first projected into another space by a projection function, called "encoder", in order to provide a "good representation" of the input. The encoder can also be viewed as a feature extractor from the input and the projection process can be expressed by a feature extraction process.

2. Decoding phase: After encoding phase, a "latent vector" is generated for the given input that well represents its meaning. In the second phase, another projection function, called "decoder", is required to map the latent vector to the output space.

Figure 1 demonstrates the basic schema of the encoder-decoder framework. Let $X = \{X_0, X_1, \cdots, X_n\}$ denote the inputs and $Y = \{Y_0, Y_1, \cdots, Y_m\}$ denote the outputs of the problem. The decoder extracts a feature vector from the input and passes it to the decoder. The decoder then generates the output based on the features extracted by the encoder.



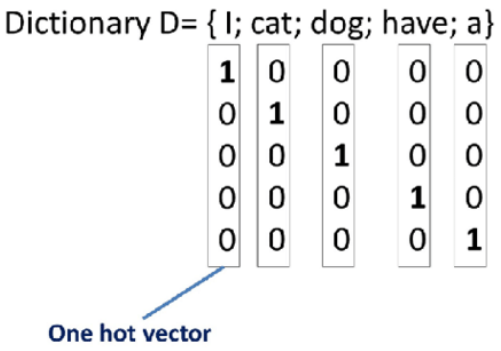
2.2 The encoder-decoder model for machine translation

Machine translation is the problem in which the encoder-decoder based models were originated and proposed first. The basic concepts of these models are shaped and presented in the machine translation literature. In this section we introduce the basic encoder-decoder structure proposed for machine translation by Cho et al. [2] to shed light on the model and its basics.

2.3 Formulation

Both the input and the output of machine translation models are sentences which can be formulated as a sequence of words. Let $X = \{X_0, X_1, \cdots, X_{Li}\}$ denote the input sentence, where x_i is the ith word in it, assuming that the input sentence has L_i words. Similarly, the output sentence could be formulated as $Y = \{y_0, y_1, \cdots, y_{Lo}\}$ in which y_i is the ith word in the output sentence assuming that it has L_o words. Furthermore, all of X_is and y_is are one-hot vectors created from a dictionary of all words in the input and the output datasets.

A one-hot vector is a vector whose components are all zero except for one of them. In order to create a one-hot vector for each word, first a dictionary of all possible words in the available datasets is created. Assuming N words in the dictionary, an N-dimensional zero vector for each word is created and the component with the same index as the word in the dictionary is set to 1. Figure 2 demonstrates the one-hot vector for each word in a sample dictionary. Assuming the dictionary D has 5 words “I”, “cat”, “dog”, “have”, “a” sequentially with the indices 0 to 4, one-hot vector for each word is displayed in the figure.



The translation process is divided into the following two subprocesses:

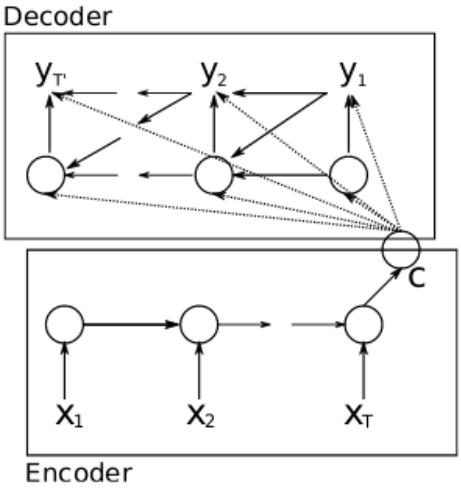
1. Encoding phase in MT

An RNN is used to extract a feature vector from the input sentence from the source language. All of the words in the input sentence are converted to one-hot vectors and passed to the RNN in the order of their presence in the sentence. The RNN then updates its hidden state and output vectors according to each word. The iteration is stopped when the End of Sentence (EOS) token is passed to the RNN. The EOS token is a token added manually to the end of input sentences to specify the end point of the sentence. The hidden state of the RNN after the EOS token is then used as the feature vector of the input sentence. One-hot vectors of the words in the input sentence are created using the dictionary of words from the source language.

2. Decoding phase in MT

Another RNN is used to generate the words of the output sentence in an appropriate order. The decoder RNN is designed to predict a probability distribution over all possible words in the dictionary of the source language words at each step. Then a word is selected with respect to the produced probability distribution as the next word in the sentence. The iteration is stopped when the EOS token is generated by the decoder or a predefined number of words are generated.

The structure of the model proposed by Cho et al. [2] [] is shown in figure 3. The context vector extracted by the encoder is denoted by C, which is the hidden state of the RNN encoder at the last step.



2.4 Encoders in machine translation (feature extraction)

An RNN is used as the encoder in the model proposed by Cho et al. [2] Let he denote the hidden state of the encoder RNN. This state vector is updated at each time step t according to equation (1) in which h_e^t is the hidden state of the encoder at time step t, $f_{encoder}$ is a nonlinear activation function that can be as simple as an element-wise logistic sigmoid function and as complex as a Long Short-Term Memory (LSTM), and X_t is the one-hot vector of the tth word in the input sentence.

$$h_e^t = f_{encoder}(h_e^{t-1}, x_t) \qquad (1)$$

Assuming that the input sentence has Li words, the encoder RNN should iterate on each word and update its hidden state vector at each step. The hidden state of the RNN after the L_ith word is then passed to the decoder as the context vector C. So, the context vector extracted by the encoder can be computed as in equation (2).

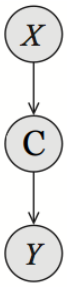
$$C = h_e^{L_i} \qquad (2)$$

2.5 Decoders in machine translation (language modeling)

The decoder is supposed to generate the output sentence word by word in a way that the meaning of the sentence is the same as the meaning of the input sentence represented by the context vector C. From another point of view, the decoder can be seen as an RNN that maximizes the likelihood of the translated sentence in the dataset for the input sentence and its generated context vector as expressed in (3), in which θ is the set of all trainable weights and the parameters of the model.

$$\Pr_{\theta}\{Y|X\} \tag{3}$$

On the other hand, according to the encoder-decoder structure, the random variable C directly depends on the random variable X, and the random variable Y directly depends on the random variable C. Figure 4 displays the dependency graph between these 3 random variables.



According to the dependencies displayed in figure 4, it can be shown that $Y \coprod X \mid C$. Since C is given, we can replace the likelihood expressed in (3) with the likelihood expressed in (4). Furthermore, assuming that each word in the sentence depends only on the meaning of the previous words in the sentence, the probability of a sentence can be replaced by the multiplication of the probabilities of its words given the previous ones.

$$\Pr_{\theta}\{Y|C\} = \Pi_{t=0}^{Lo}\, Pr\{y_t|y_{t-1}, y_{t-2}, \cdots, y_0, C\} \tag{4}$$

Consequently, the decoder is supposed to generate a probability distribution over each word at each step *t* given its previously generated words and the context vector extracted by the encoder. The probability distribution can be formulated by the RNN according to equations (5) and (6). Let h_d^t be the hidden state of the decoder at time step *t*. Let O_t be the decoder output at time step *t* (an *Lo* dimensional vector) which is generated by a nonlinear function *g* applied on the decoder's hidden state and the context vector C. The decoder's hidden state is also generated by the nonlinear function $f_{decoder}$ applied on the previously generated word, hidden state of the decoder at previous time step and the context vector according to (7). With applying a *Softmax* on the output, a vector with the same size is generated whose sum of components is equal to one and can be treated as the desired probability distribution.

$$O^t = g(h_d^t, y_{t-1}, C) \tag{5}$$

$$Pr\{y_t|y_{t-1}, y_{t-2}, \dots, y_0, C\} = SoftMax(O^t) \tag{6}$$

$$h_d^t = f_{decoder}\left(h_d^{t-1}, y_{t-1}, C\right) \tag{7}$$

At each time step, the probability distribution $\Pr\{y_t|y_{t-1}, y_{t-2}, \dots, y_0, C\}$ is generated by the decoder according to equation (6) and the next word is selected with re-

spect to this probability distribution over the words in the dictionary of the destination language.

The two components of the proposed model can be jointly trained to minimize the negative conditional log likelihood expressed in (8) in which *N* is the number of samples in the dataset, Y_n and X_n are the *n*th output and input pair in the dataset, θ is the set of all trainable parameters, and *Loss* is the loss function to be minimized.

$$Loss = -\frac{1}{N} \Sigma_{n=0}^{N} log\, Pr_{\theta}(Y_n | X_n) \qquad (8)$$

3 Encoder structure varieties

The baseline encoder-decoder architecture proposed by Cho et al. [2] in machine translation attracted the attention of many researchers in different fields. As explained before, almost all of the variants of the baseline architecture in different tasks share a similar decoder, but the structure of encoder varies based on the type of input. In this section, we will introduce the important structures of encoders to encode different input types.

3.1 Sentence as input

The simplest encoder for problems with sentences as inputs is an RNN. The first proposed encoder in machine translation is an LSTM which takes all words of the input sentence, processes them and returns the hidden state vector as the context vector.

Along with the RNNs, CNNs are employed to extract features from the source sentences in the encoding phase. As an instance, Gehring et al. proposed a convolutional encoder for machine translation in order to create better context vectors by taking nearby words into consideration using a CNN [4]. In this encoder, a CNN with a kernel size of k = 3 is used to extract a combination of each three nearby words' meaning in the sentence to generate the context vector.

In addition, different RNN cells are used as blocks of the encoder for sentence inputs. LSTMs [5] are widely used because of their ability to cope with long-term dependencies and remembering far history in the input sequence [6][7][8][9]. GRU [2] is also used in different proposed models due to its good performance and the fact that it can be assumed as a light-weighted version of LSTM [2][10][11][12].

3.2 Image as input

Encoder-Decoder based architectures form a majority of the proposed models to generate captions for images. In such models, the process of generating captions for the input image is divided into two steps. The first step is encoding in which a feature vector extracted from the image is returned as the context vector. The second step is

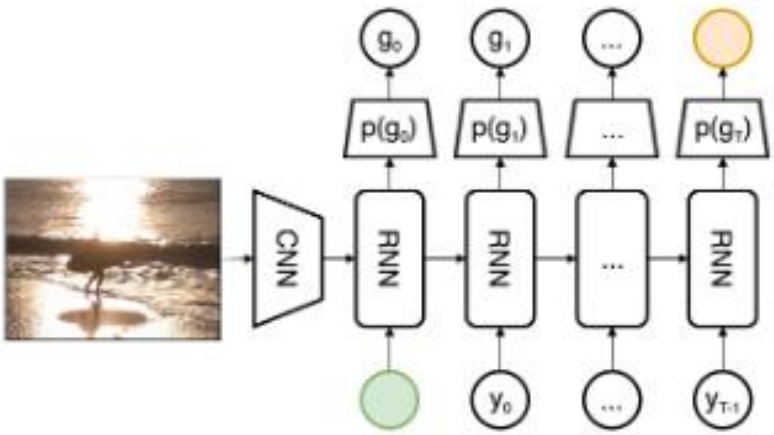
decoding in which the generated context vector is passed to a decoder to generate sentences describing the context.

The best choice for encoders in such problems is a CNN. Almost all of the proposed models for image captioning based on the encoder-decoder framework use different types of CNNs as the encoders. Neural encoder-decoder based approaches to image captioning share the same structure for decoder, while in most of them the encoder consists of a single CNN. So, the extracted feature vector from the image can be expressed in equation (9) in which X is the input image, CNN(X) is the output of the CNN network, and C is the context vector passed to the decoder.

$$C = CNN(X) \qquad (9)$$

A wide variety of CNNs are employed as encoders in the proposed models for image captioning. Since the pretrained versions of VGGNet [13] and AlexNet [14] on the ImageNet dataset [15] extract good features from images for different tasks and are available online, they have been used as encoders in different proposed image captioning models [16][17][18]. Furthermore, ResNet [19] has been widely used because of its good performance as the encoder in such models [20][21][22][23]. Google NIC Inception v3 [24] has also been used in proposed models because of its better image classification accuracy compared to ResNet [25][26][27][28].

Figure 5 illustrates the use of CNNs as the encoder in models based on the encoder-decoder framework for image captioning proposed by Vinyals et al. [27] Similar architectures are used to generate captions in other studies. As it is shown in the figure, the encoder part of the model consists of a CNN extracting a feature vector from the input image. The extracted feature vector is then passed to the decoder to generate the appropriate caption. The decoder consists of an RNN which generates the probability of the next word according to (4) at each step.



3.3 Video as input

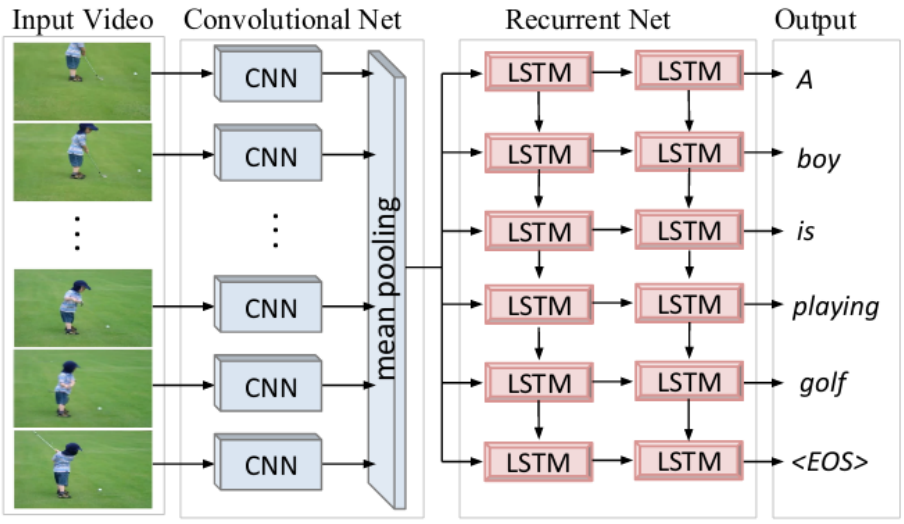
Another sort of problems that the encoder-decoder models play an important role in solving them are those with videos as the input and describing text as the output, also called "Video Description Generation" or "Video Captioning". Since creating a good representation is critical to the overall performance of video captioning models, a wide variety of encoders are proposed to cope with different difficulties and challenges of such systems. This section presents some examples of encoders proposed to deal with the challenges of extracting motion details from the video.

Assume an input video V consists of Li frames. We can present the video as in equation (10), in which v_I is a representation of the ith frame in the input video and v_{Li} is the end of video token (<EOV>). In fact, each v_I is the feature vector extracted by a CNN on the ith frame in the input video.

$$V = \{v_0, v_1, \cdots, v_{Li}\} \qquad (10)$$

Since in the baseline encoder-decode model, the encoder should return a "fixed length" context vector extracted from the input, an aggregation function is required to aggregate feature vectors from different frames in the video and pass it as the context vector to the decoder.

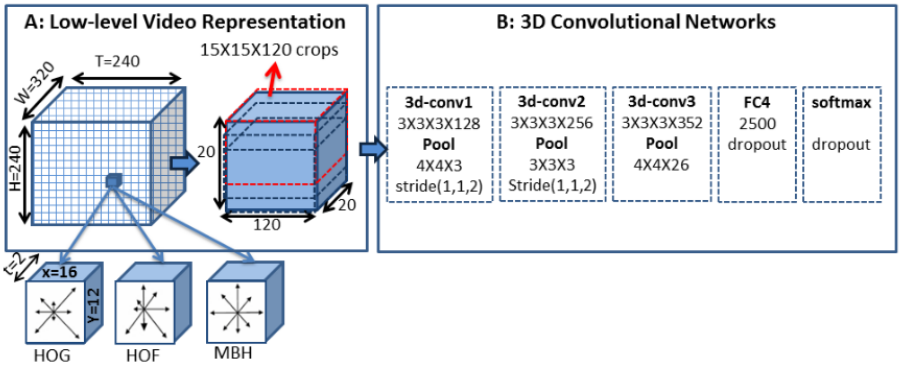
Different ideas have been employed to propose a good aggregation for video captioning. The first end-to-end encoder-decoder based approach in video description generation proposed by Venugopalan et al. in 2014 [29] used a mean pooling layer to create the fixed length context vector from the input video. In that model, first a CNN is applied to each frame of the input video. Then a mean pooling layer is applied to create an average feature vector over the set of feature vectors extracted from each frame. The average feature vector is then passed to the decoder to generate the sentence. A stacked RNN structure is used as the decoder.



Different CNNs have been used to extract feature vectors from the frames of the input video. For instance, Yao et al. [23] used Inception V4 [30] for feature extraction from video frames. Majd et al. [31] proposed an extended version of the LSTM cells, called "C2LSTM" in which the motion data as well as the spatial features and the temporal dependencies are perceived by embedding a correlation modeling layer into the cell. Majd et al. [32] also proposed a novel network architecture using previously proposed C2LSTM as the encoder for human action recognition.

3D-CNNs

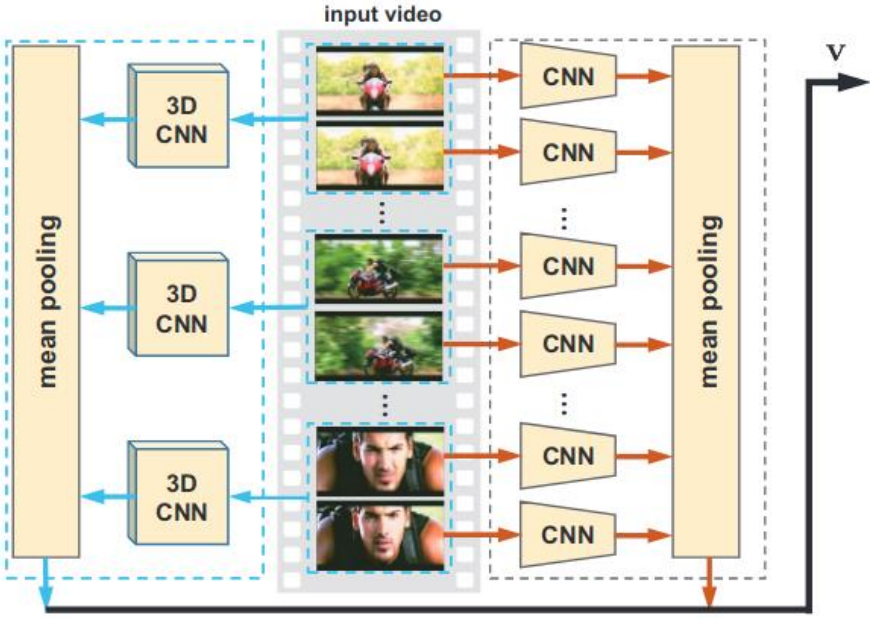
Extracting good features from the input video is a challenging task that can highly affect the performance of the proposed model. The extracted context vector from the input video should well express the detailed motions in the video. In order to create an encoder capable of extracting fine motion features from the video, Yao et al. [33] proposed a 3D-CNN as the encoder. The structure of this 3D-CNN is illustrated in figure [fig:3dcnn].



Actually, the proposed 3D-CNN models the spatio-temporal dependencies in the input video. The 3D-CNN is used to build a higher-level representation that preserves the local motion information from short frame sequences in the input video. This is accomplished by first dividing the input video clip into a 3D spatio-temporal grid of 16 * 12 * 2 (width * height * timesteps) cuboids. Each cuboid is represented by concatenating the histogram of oriented gradients (HOG), histogram of oriented flow (HOF) and motion boundary histogram (MBH) with 33 bins. This transformation ensures that the local temporal structures and motion features are well extracted. The generated 3D descriptor then is passed to 3 convolutional layers each followed by a max-pooling layer and one fully connected layer followed by a softmax layer as demonstrated in the figure 6. The output of the 3D-CNN is then passed to the decoder to generate an appropriate caption.

The 3D-CNN proposed by Yao et al. [33] is also used along-side the typical 2D-CNN in other works. Pan et al. [34] proposed a novel encoder-decoder architecture for video description generation and used the 3D-CNN and the typical 2D-CNN and applied a mean pooling layer to the set of features extracted by each of the CNNs and

concatenated the output to generate the context vector of the video. Figure 7 illustrates the encoder part of this model.



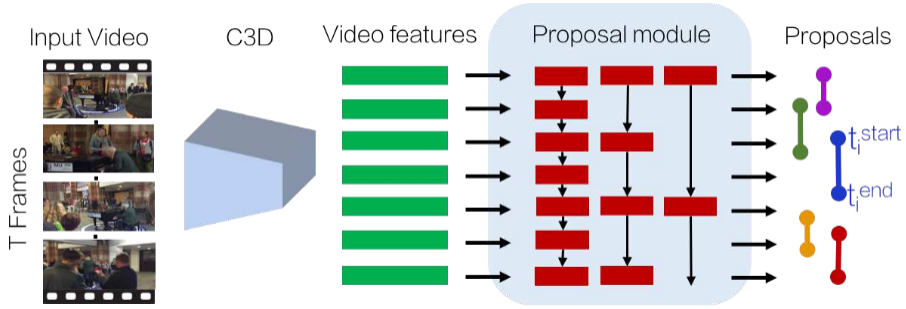
Dense video captioning

Another approach to video captioning includes those methods focusing on "Dense Video Captioning". Despite the models that generate a single sentence as the description of the input video, dense video captioning models first detect and localize the existing events in the input video and then generate a description sentence for each of the detected events.

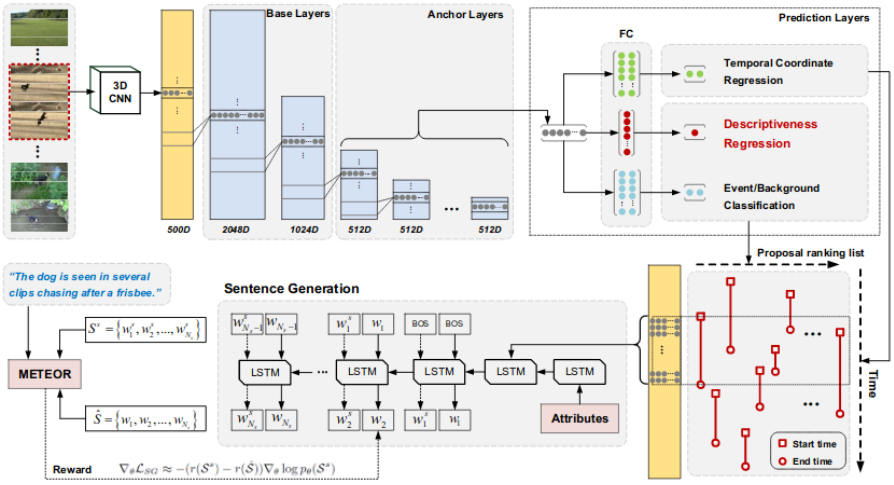
Encoders for dense video captioning are supposed to first detect all of the existing events in the input video. Then for each of the events a quadruple $< t_{start}, t_{end}, score, h >$ should be extracted. t_{start} and t_{end} are the starting and ending frame numbers of the specified event. score is the confidence score of the encoder for each of the events. If the score of an event is greater than a threshold, it is reported as an event and its quadruple is passed to the decoder for sentence generation; otherwise, it is ignored. Finally, h is the feature vector extracted from the range of frames between t_{start} and t_{end} which is used by the decoder as the context vector of the event to generate a sentence for the event [35].

The task of dense video captioning was proposed by Krishna et al. [36] first in 2017. The proposed encoder by Krishna et al. [36] for dense video captioning is able to identify events of the input video within a single pass while the proposed decoder simultaneously generates captions for each event detected and passed by the encoder.

Figure 8 illustrates the structure of encoder proposed by Krishna et al. [36] for dense video captioning. The proposed encoder is able to extract all events in the input video using a deep action proposal (DAP) module proposed by [37]. To do this, a 3D-CNN is applied to the input video frames to extract video features. These video features are passed to the DAP module. This module consists of different LSTMs that are applied to the video features sequence in different resolutions and are trained to detect starting and ending points of events. The confidence score of each event is also computed by DAP. The proposed event proposals are then sorted with respect to their ending points and passed sequentially to the decoder. The feature vector of each event is also the hidden state of the corresponding RNN in the DAP. The decoder then generates a sentence for each event using its feature vector as the encoder output.



Li et al. [35] proposed a novel end-to-end encoder-decoder based approach for dense video captioning which unified the temporal localization of event proposals and sentence generation [35]. Figure 9 illustrates the structure of the proposed model [35]. Here, instead of using an extra DAP module, a 12-layer convolutional structure is designed to extract features for action proposal over the output of the 3D-CNN. The first 3 layers of the convolutional structure (500D layer and base layers in figure 9) are designed to introduce nonlinearities and decrease the input dimension. The next 9 layers, which are called "Anchor layers", extract features from different resolutions to be used for event prediction. The "Prediction Layer" consists of three parallel fully connected layers to first regress temporal coordinates (t_{start} and t_{end}) of each event, then compute the descriptiveness of the event (score) and finally classify the event vs background. The prediction layer is applied to the output of all anchor layers to enable the model to detect events from different resolutions. The extracted proposals are then passed to the proposal ranking module which ranks event proposals with respect to their ending time. Finally, the events are passed to the decoder for sentence generation sequentially.



A wide variety of models are proposed to cope with the difficulties of the encoding phase in dense video captioning. Shen et al. [38] proposed a new CNN called "Lexical FCN" which is trained in a weakly supervised manner to detect events based on the captions in the dataset. Duan et al. also proposed a novel approach for dense video captioning based on the similar assumption "each caption describes one temporal segment, and each temporal segment has one caption" [39]. Xu et al. proposed an end-to-end encoder-decoder based model for dense video captioning which detects and describes events in the input video jointly and is applicable to dense video captioning on video streams. Zhou et al. also proposed an end-to-end approach with a masking network to localize and describe events jointly [40]. Wang et al. proposed a novel architecture to take both past and future frames into account while localizing the events in the input video using [41] bidirectional models.

4 Decoder structure varieties

In the encoder-decoder based models, decoders generate a sequential output for the given input. The generated output might be in the form of a descriptive text (the desired output in machine translation, image/video captioning, textual/visual question answering, and speech to text conversion), or a speech signal (the desired output in text to speech challenge). By the way, the output is a numerical sequence that is passed to the last layer in order to generate appropriate output for the given input. Therefore, the main structures of the decoders are similar in different tasks. This section, introduces different techniques proposed to make better decoders with better generated captions.

4.1 Long-term dependencies

One of the basic problems with RNNs is the problem of "long-term dependencies". Indeed, when the length of the input or the length of the desired output is too large, the gradients in these networks should propagate over many stages. When the gradient is propagated over a large number of stages, it tends to either vanish or explode.

In addition, the gradients in each backpropagation step are multiplied by small coefficients or small learning rates. Thus, the gradient in the early stages will be close to zero and might make no significant change in the weights of the early stage layers [42].

In this section we will discuss the approaches proposed to cope with the long-term dependency challenge in the decoders.

4.2 LSTMs

LSTMs have achieved excellent results on a variety of sequence modeling tasks thanks to their superior ability to preserve sequence information over time. The combination of the "*memory cell*" and the "*forget gate*" in the structure of LSTM improves its ability to model sequence information by training to forget the unnecessary information (using the forget gate) and keep the necessary information in the memory cell. Cho et al. [2], Bahdanau et al. [7], Luong et al. [43], Wu et al. [44], Johnson et al. [45] and Luong et al. [46] used LSTMs as both the encoder and decoder part of their models proposed for machine translation.

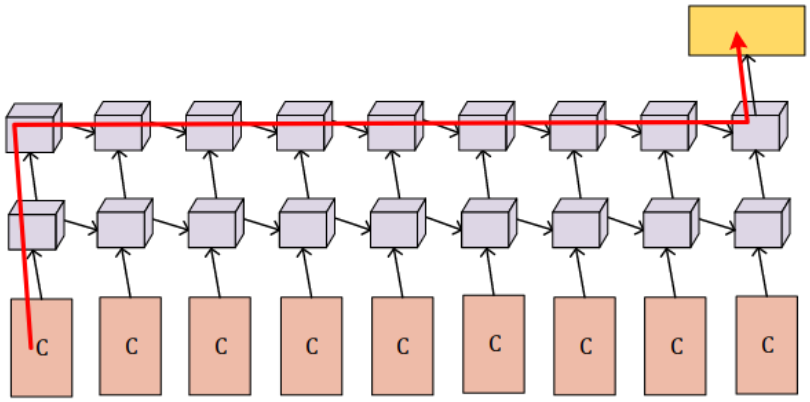
4.3 Stacked RNNs

As mentioned earlier, multi-staged decoders are hard to train due to the vanishing gradient problem. Thus, most of the proposed encoder-decoder based models use a single layer RNN as the decoder which results in difficulties to generate rich fine-grained sentences. Stacking multiple RNNs on top of each other is another way to enable decoders to generate sentences describing more details of the input image.

Donahue et al. [47] proposed an encoder-decoder based approach to image captioning which uses a stacked structure of LSTMs as the decoder in order to describe more details of the input image. In this method an LSTM is used on top of another one in a way that the first layer LSTM takes image features and the previously generated word embedding along with its previous hidden state vector as the input and generates a coarse low-level representation of the output sentence. In the next step, the hidden state of the low-level LSTM is passed to the next LSTM as the input along with its previous hidden state to generate the fine high-level representation of the output. A

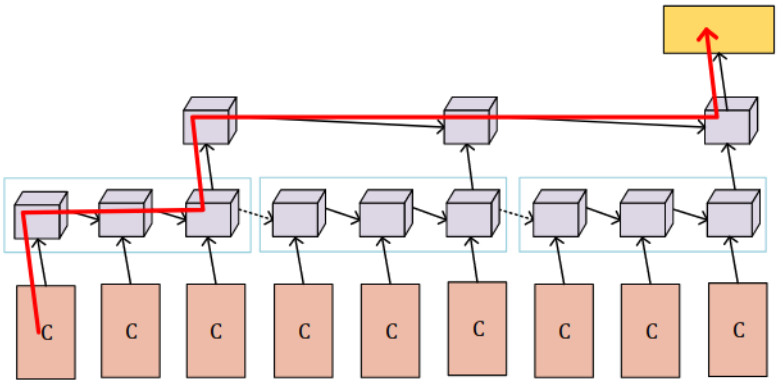
softmax layer is then applied to the generated high-level representation of the output to generate the probability distribution of the next word in the sentence. Gu et al. [48] also used encoder-decoder based model with a two-layer stacked LSTM as the decoder in order to enable the proposed model to generate better descriptions.

The idea of employing a stacked structure of RNNs as the decoder is also used in models proposed for video description generation. Venugopalan et al. [29] proposed the first decoder in neural encoder-decoder based approaches for video description generation with a simple stacked structure. Figure 10 demonstrates the architecture of a sample stacked decoder. Blocks tagged with "C" display the input at each step. The red line illustrates the shortest path from the first step to the output in the model. Since the length of the shortest path from the first step to the output correlates with the testing and the training time of the model, decreasing this length decreases the testing and the training time of the model.



Along with the methods using stacked RNNs as decoders, a category of models are proposed which follow a hierarchical fashion to arrange RNNs in decoders in order to enable the models to generate fine-grained output sequences.

In addition, hierarchical RNN structures are also used to enable encoders in problems with a sequential input to exploit and encode more detailed information from the input. Pan et al. [49] proposed an encoder-decoder based model for video description generation with a hierarchical encoder structure. In their model, two layers of different LSTMs are used. The first layer LSTM is applied to all sequence steps in order to exploit low-level features and the second layer LSTM is applied on the output of equally sized subsets of the input sequence steps to exploit the high-level features. Figure [fig:hrne] demonstrates this architecture. The illustrated red line, shows the shortest path from the first step to the output. Comparing structures displayed in figures [fig:stacked] and [fig:hrne] shows that the shortest path from the first step to the output in hierarchical models is much smaller than that in stacked models. Therefore, the efficiency of the hierarchical model is much higher than that of the stacked model.



More complex hierarchical structures are also proposed in the literature for different intents. Yu et al. [50] proposed a model with a hierarchical structure to generate a set of sentences arranged in a single paragraph as a description for the input video. The first layer in this model is a simple decoder to generate single sentences and the second layer is a "paragraph controller". The paragraph controller is another RNN which generates a feature vector given the last hidden state of the first layer RNN denoting the meaning of the next sentence to be generated. The first layer RNN then takes the feature vector generated by the second layer and concatenates it with other inputs to control the meaning of the next sentence.

4.4 Vanishing gradients in stacked decoders

Even though increasing the depth of the stacked decoder structure adds more non-linearities to the model and empowers it to generate fine-grained sentences, the number of layers in the stacked structures is strictly restricted. Most of stacked decoder structures use at most 2-layers of RNNs on top of each other [47][49][50].

Indeed, the most important issue restricting the number of layers in stacked structures is the problem of vanishing gradients in deeper decoders. The backpropagated gradients vanish as a result of two facts. First, the gradients in such architectures are multiplied by small multipliers and small learning rates at each stage. Second, since the loss function of the proposed decoders is based on the likelihood of the next word, decoders are supposed to predict a probability distribution over all the words in the dictionary. It means the decoder's output size is equal to the size of word dictionary. Furthermore, the sum of all components in the output layer is supposed to be equal to 1, which means the gradients computed at the last layer are numerically small. Summing up, the gradients vanish in stacked decoders since the computed gradients at the last layer are small and they are multiplied by small multipliers at each step.

Asadi et al. [51] proposed a novel approach to train the stacked decoders in a way that the gradients of the last layer are large enough to make significant changes in the weights of the first layers. The main idea is to use a word-embedding vector instead

of a one-hot vector representation as the decoder desired output. As a result, the optimization problem changes from predicting the conditional probability distribution of the next word to a word-embedding regression and the *Softmax* layer at the end of the decoder is removed. In this way, the limitations of the value of the computed gradients at the last step are resolved. In addition, the loss function of the decoder is changed from the cross-entropy to MSE of the word embedding of the next word.

4.5 Reinforcement learning

One of the problems of training the decoders using the loglikelihood model is that the performance of the model is highly different on the training and testing sets. This occurs since the optimization function for training is different from the evaluation metrics used in testing. Recently, reinforcement learning has been used to decrease the gap between training and testing performance of the proposed models. In other words, the main problem with the loglikelihood objective is that it does not reflect the task reward function as measured by the BLEU score in translation.

Wu et al. [44] proposed the first decoder trained by reinforcement learning for machine translation. After that, other researchers used reinforcement learning to train decoders in other tasks. Wang et al. [52] proposed the first decoder trained with reinforcement learning, taking CIDEr [53] score as the reward in video captioning. Li et al. [35] also trained a decoder in a reinforcement learning fashion using the METEOR [54] score as the reward to generate caption for the input videos.

Figure 11 illustrates the structure of the model proposed by Wang et al. [52] The decoder in this work consists of three different modules, namely a manager, a worker, and an internal critic. These three modules are trained using a reinforcement learning method. The manager operates at a lower temporal resolution and emits a goal when needed for the worker, and the worker generates a word for each time step by following the goal proposed by the manager. The internal critic determines if the worker has accomplished the goal and sends a binary segment signal to the manager to help it update goals.

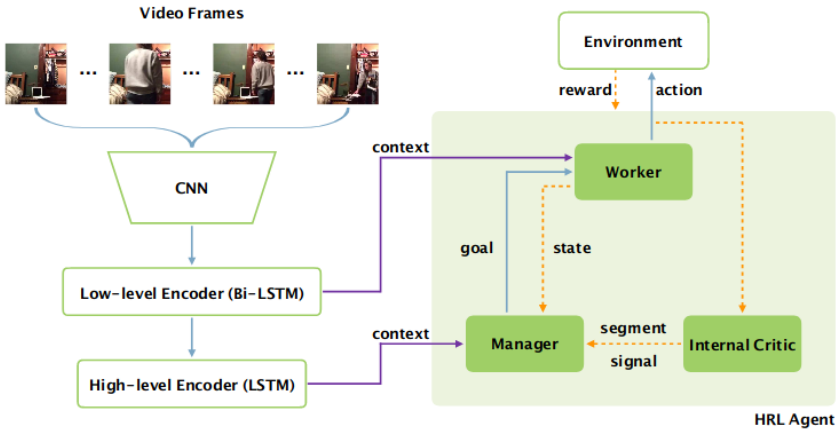
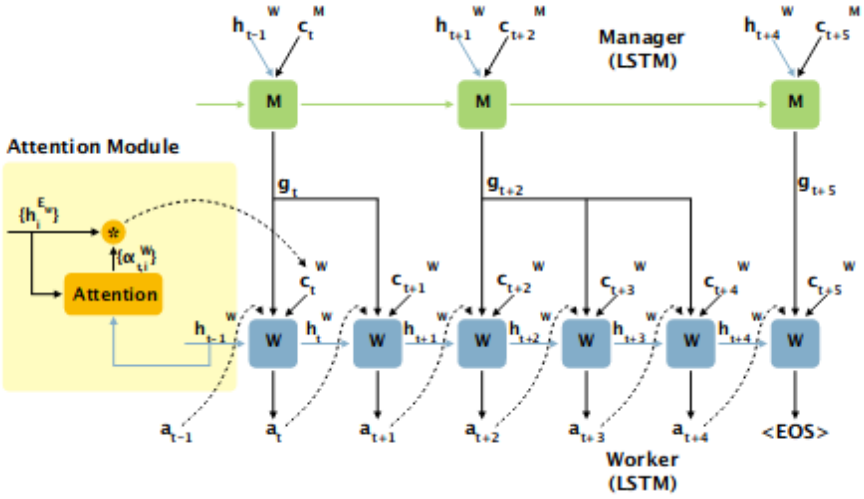


Figure 11 illustrates the unrolled decoder proposed by Wang et al. [52] The manager takes the context vector c_t^M at time step t and the feature vector of sentence generated at previous time step h_{t-1}^W as the input. An LSTM is used to model the extracted goal sequences. The LSTM takes the input and updates its hidden state h_t^M. The hidden state of the LSTM is then used to generate the next goal using the nonlinear function u_M according to (12).

$$h_t^M = LSTM^M(h_{t-1}^M, [c_t^M, h_{t-1}^W]) \quad (11)$$

$$g_t = u_M(h_t^M) \quad (12)$$

$LSTM^M$ denotes the LSTM function used in manager, u_M is the function projecting hidden states to the semantic goal, h_{t-1}^M is the hidden state of the manager LSTM at the previous time step, and g_t is the vector of semantic goal generated at time step t.



The worker then receives the generated goal g_t, takes the concatenation of $[c_t^W, g_t, \alpha_{t-1}]$ as the input, and outputs the probabilities π_t over all actions $\alpha t \in V$, where each action is a generated word according to equations (13) to (15)s.

$$h_t^W = LSTM^W(h_{t-1}^W, [c_t^W, g_t, \alpha_{t-1}]) \quad (13)$$

$$x_t = u_W(h_t^W) \quad (14)$$

$$\pi_t = SoftMax(x_t) \quad (15)$$

The internal critic is used to provide a good coordination between the manager and the worker. Internal critic is indeed a classifier to determine when the worker is done with generating an appropriate phrase for a given goal. When the worker is done, the internal critic sends an activation signal to the manager to generate a new goal. Let z_t be the binary signal of the internal critic, the probability $\Pr(z_t)$ is computed according to equations (16) and (17).

$$h_t^I = LSTM^I([h_{t-1}^I, \alpha_t]) \quad (16)$$

$$\Pr(z_t) = sigmoid(W_z h_t^I + b_z) \quad (17)$$

The objective of the worker is to maximize the discounted return in which θ_W is the set of trainable parameters of the worker, γ is the discount rate, and r_{t+k} is the reward at step $t + k$. Therefore, the loss function of the decoder can be written as (19).

$$R_t = \Sigma_{k=0}^{\infty} \gamma^k r_{t+k} \quad (18)$$

$$L(\theta_W) = -E_{\alpha_t \sim \pi_{\theta_W}}[R(\alpha_t)] \quad (19)$$

The gradient of the non-differentiable, reward-based loss function can be derived as:

$$\nabla_{\theta_W} L(\theta_W) = -E_{\alpha_t \sim \pi_{\theta_W}}[R(\alpha_t \nabla_{\theta_W} \log \pi_{\theta_W}(\alpha_t))] \quad (20)$$

Typically, the expectation of the loss function if estimated with a single sample, so the expectation term can be omitted. In addition, the reward can be subtracted with a baseline $b_t W$ in order to generalize the policy gradient.

$$\nabla_{\theta_W} L(\theta_W) \approx -(R(\alpha_t - b_t^W) \nabla_{\theta_W} log \pi_{\theta_W}(\alpha_t) \quad (21)$$

The manager is supposed to be trained in a way that it can compute goals to generate sentences with better BLEU scores. The action of the decoder is produced by the worker. So, the worker is assumed to be fully trained and used as a black box when training the manager. More specifically, the manager outputs a goal g_t at step t and the worker then runs c steps to generate the expected segmente$_t$, $c = \alpha_t \alpha_t + 1\alpha_t + 2 \cdots \alpha_t + c$ using the goal. Then the environment responds with a new state s_{t+c} and reward r(e_t, c). Following a similar math, the final gradients for training the manager can be derived as in (22).

$$\nabla_{\theta_M} L(\theta_M) = -(R(e_t, c) - b_t^M)\left[\Sigma_{i=t}^{t+c-1} \nabla g_t log \pi(\alpha_i)\right] \nabla_{\theta_M} \mu_{\theta_M}(s_t) \quad (22)$$

In which $\mu_{\theta_M}(s_t)$ is a noisy version of the generated goal and is used in order to empower exploration in the training of the model. Furthermore, rewards are defined as (23) and (24).

$$R(a_t) = \Sigma_{k=0}^{\infty} \gamma^k \left[CIDEr(sent + \alpha_{t+k}) - CIDEr(sent)\right] \quad (23)$$

$$R(e_t) = \Sigma_{n=0}^{\infty} \gamma^n \left[CIDEr(sent + e_{t+n}) - CIDEr(sent)\right] \quad (24)$$

Other metrics such as BLEU score can be used instead of CIDEr [55].

5 Attention mechanism

Models based on the encoder-decoder framework encode input to a "fixed length vector". The decoder in these models generates the output based on the information represented in the fixed length encoder output. Each element of the output may be more strongly related to a specific part of the input. In these cases, more detailed information about that specific part of the input is required, and the extra information from the other parts of the input could deceive the model.

Attention mechanism, first introduced by Bahdanau et al. [7] in machine translation, is a mechanism that allows the encoder-decoder models to pay more attention to a specific part of the input, while generating the output at each step. Furthermore, the mechanism enables decoders to cope with the long-term dependencies and generate more fine-detailed sentences and outputs.

In this section, first the basic idea of the attention mechanism proposed in machine translation is described. Then, the use of this mechanism in some encoder-decoder architectures proposed in various applications is discussed.

5.1 Basic mechanism

Bahdanau et al. [7] proposed the first encoder-decoder based model equipped with the attention mechanism in order to produce better translations. The encoder and the decoder parts of the proposed model are changed. The encoder is modified to generate a sequence of feature vectors called "annotation vectors" and an extra layer called "attention layer" is added in between the encoder and the decoder. The attention layer receives the annotation vectors generated by the encoder and creates a fixed length context vector at each step and passes it to the decoder in order to generate the probability of the next word in the sentence.

Based on these changes, the target probability distribution of the decoder can be expressed as in (25). It denotes the probability of the next word y_t at time step t, given all of the previously generated words and the context vector generated to predict the tth word. The decoder computes this probability at each step.

$$Pr(y_t|y_{t-1}, \cdots, y_0, C_t) \tag{25}$$

Let $L = \{l_0, l_1, \cdots, l_{N_i}\}$ be the set of generated annotation vectors by the encoder, and N_i be the number of generated annotations, the context vector C_t is then computed at each step using the equation (26). The coefficients α_k in (26) are called the "attention weights".

$$C_t = \Sigma_{k=0}^{N_i} \alpha_k^t l_k \tag{26}$$

The key point in generating the context vector at each step using the attention mechanism is to compute the attention weights at each step. Researchers have proposed different ways to compute the attention weights. One of the most used attention mechanisms, which is called "Soft Attention", is proposed by Xu et al. [9] The attention weights in soft attention are computed using equations (27) and (28).

$$\alpha_k^t = \frac{\exp(e_k^t)}{\Sigma_{j=0}^{N_i} \exp(e_j^t)} \tag{27}$$

$$e_k^t = f(h_{t-1}, l_j) \tag{28}$$

Equation (28) is an alignment model which scores how well the output at step t depends on the input section related to the annotation vector l_j. The function f in (28) measures the alignment between the output and the input. A simple candidate for implementing function f is an MLP which can be modeled as:

$$f(h_{t-1}, l_j) = W_2 tanh(W_h h_{t-1} + W_l l_j + b_1) + b_2 \tag{29}$$

In which W_2, W_h, and W_l are weight matrices and b_1 and b_2 are biases. All of these parameters can be trained jointly with other trainable model parameters.

Another version of the attention mechanism, called "Hard Attention", is also introduced by Xu et al. [9] in which at each step one of the attention weights is equal to 1 and the rest are equal to zero.

5.2 Extensions

Vaswani et al. [56] showed that the attention mechanism not only can be used instead of convolutional and recurrent layers in the network architecture, but also outperforms their functionality and decreases the computation complexity of the network training. Vaswani et al. [56] proposed a novel neural architecture in which all of the convolutional and recurrent layers of the networks are substituted with attention layers.

Attention mechanism is also used in other tasks. You et al. [57] proposed a semantic attention in image captioning. Lu et al. [20] also proposed an adaptive version of the attention mechanism in image captioning. Gao et al. [58] proposed an encoder-decoder based model using the attention mechanism and introduced a novel approach to train the attention weights to decrease the semantic gap of the generated caption.

Since the attention mechanism enables the models to focus more on a part of the input while generating the output, it can be used as a selector between multiple information sources. Wu et al. [59] proposed a novel encoder-decoder based model for video captioning which is used temporal features, audio features, motion features and semantic label features. The proposed attention-based hierarchical multi-modal fusion model (HATT) exploits the complementariness of multi-modal features. The model consists of three different attention layers: 1) low-level attention which deals with temporal, motion and audio features, 2) high-level attention which deals with semantic labels, and 3) sequential attention which aggregates the information from the other two layers of attention.

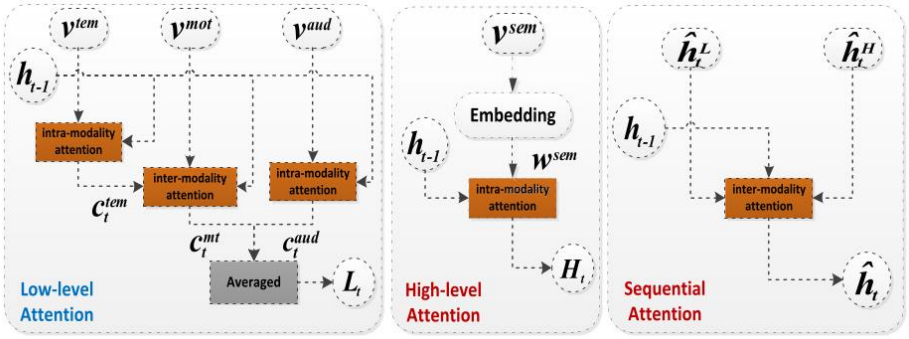
In the proposed model two different types of the attention mechanism is proposed:

1. Intra-modality attention: the soft-attention applied on a feature set V containing features from a single modality.

2. Inter-modality attention: the soft-attention applied on a feature set V containing features from multiple modalities.

Figure [fig:attnss] displays the structure of different attention layers proposed by Wu et al. [59]. In the low-level attention layer, an intra-modality attention is applied on the temporal features first to generate the context vector C_t^{tem}. Then an inter-modality attention selects between the temporal and the motion features generating C_t^{mt}. In parallel, an intra-attention modality is applied on the audio features, which generates the context vector C_t^{aud}. Finally, the average vector of the C_t^{mt} and the C_t^{aud} is computed and passed on. In the high-level attention layer, an intra-modality atten-

tion is used to generate the context vector of the semantic labels. Ultimately, in the sequential attention layer, an intra-modality attention is used to aggregate the context vectors from the low-level and the high-level attention layers.



The proposed model by Wu et al. [59] employed the attention mechanism in a hierarchical structure to exploit both the long-term and the multi-modal feature dependencies from the input to generate fine-grained captions.

6 Conclusion

In this chapter, we presented the baseline encoder-decoder model first proposed in machine translation and then extended to other applications. The main idea in the encoder-decoder framework is to split the process of generating a textual output describing the input into two subprocesses. A feature vector is first extracted by an encoder from the input. Then a decoder is used to generate the output step by step using the feature vector extracted by the encoder.

The structure of the encoder varies based on the input type. Whenever the input is a text or a sequence, an RNN is used as the encoder. When the input is an image, CNNs can be used to encode the input image. If that the input is a video a combination of CNNs and RNNs is used to first extract features from all of the input frames and then model temporal dependencies between different frames. 3D-CNNs are also proposed to extract motion data from the input. More complex models to detect and localize events in the input video for dense video captioning are also discussed.

Extracting long-term dependencies and generating rich fine-grained sentences are the main problems of decoders in the encoder-decoder based models. LSTM cells are the simplest models that can cope with the problem of long-term dependencies, and are used in the proposed architectures for different tasks. Using stacked structures of RNNs is another approach to make deeper decoders. This allows the models to cope with the long-term dependency challenge and generate more detailed sentences. One of the most important problems of the stacked decoder structures is the problem of the vanishing gradients which can be solved by modifying the optimization problem of the decoder. Another approach to cope with the problem of long-term dependencies is

using the attention mechanism, which is a technique to focus more on different portions of the input, while generating outputs at each step.

References

1. Girvan, M., Newman, M.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences 99(12), 7821–7826 (2002)
2. Fortunato, S.: Community detection in graphs. Physics Reports 486(3–5), 75-174 (2010)
3. Newman, M. E. J., Girvan, M.: Finding and evaluating community structure in networks. Physics Rev. E 69(2), 026113 (2004)
4. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D.: Defining and identifying communities in networks. Proceedings of the National Academy of Sciences of USA 101(9), 2658-2663 (2004)
5. Clauset, A., Newman, M., Moore, C.: Finding community structure in very large networks. Phys. Rev. E 70(6), 066111 (2004)
6. Shi, C., Zhong, C., Yan, Z., Cai, Y., Wu, B.: A multi-objective optimization approach for community detection in complex network. In : IEEE Congress on Evolutionary Computation (CEC), Barcelona, Spain, pp.1-8 (2010)
7. Ehrgott, M.: Multicriteria Optimization 2nd edn. Springer, Berlin (2005)
8. Handl, J., Knowles, J.: An Evolutionary Approach to Multiobjective Clustering. IEEE Transactions on Evolutionary Computation 11(1), 56-76 (2007)
9. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
10. Srinivas, N., Deb., K.: Multiobjective optimization using nondominated sorting in genetic algorithms. Evolitionary Computation 2(3), 221–248 (1994)
11. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Parallel Problem Solving from Nature PPSN VI Lecture Notes in Computer Science 1917, 849-858 (2000)
12. Zitzler, E., Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation 3(4), 257-271 (Nov 1999)
13. Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation 8(2), 173-195 (2000)
14. Corne, D., Knowles, J., Oates, M.: The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization. In : Proceedings of the Parallel Problem Solving from Nature VI Conference, pp.839--848 (2000)
15. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Ltd, England (2001)
16. Leskovec, J., Lang, K., Mahoney, M.: Empirical Comparison of Algorithms for Network Community Detection. Proceeding of 19th international conference on World Wide Web (ACM WWW), 631-640 (2010)

17. Shi, C., Zhong, C., Yan, Z., Cai, Y., Wu, B.: On selection of objective functions in multi-objective community detection. In : Proceedings of the 20th ACM international conference on Information and knowledge management, Glasgow, Scotland, UK, 2301-2304 (2011)
18. Shi, J., Malik, J.: Normalized Cuts and Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 888--905 (1997)
19. Flake, G., Lawrence, S., Giles, C.: Efficient Identification of Web Communities. In : Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 150--160 (2000)
20. Pizzuti, C.: Ga-net: a genetic algorithm for community detection in social networks. In : Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X, Dortmund, Germany, 1081--1090 (2008)
21. Lancichinetti, A., Fortunato, S., Kertesz, J.: Detecting the overlapping and hierarchical community structure of complex networks. arXiv:0805.4770v2 (2008)
22. Aldecoa, R., Marín, I.: Deciphering Network Community Structure by Surprise. PLoS ONE 6, e24195 (September 2011)
23. Tasgin, M., Bingol, H.: Community detection in complex networks using genetic algorithm. arXiv:cond-mat/0604419 (2006)
24. Shi, C., Zhong, C., Yan, Z., Cai, Y., Wu, B.: A new genetic algorithm for community detection. Complex Sciences 5(2), 1298-1309 (2009)
25. Pizzuti, C.: Community detection in social networks with genetic algorithms. In : Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA, pp.1137-1138 (2008)
26. Firat, A., Chatterjee, S., Yilmaz, M.: Genetic clustering of social networks using random walks. Computational Statistics and Data Analysis 51(12), 6285–6294 (2007)
27. Pizzuti, C.: A multi-objective genetic algorithm for community detection in networks. In : 21st International Conference on Tools with Artificial Intelligence, pp.379-386 (2009)
28. Agrawal, R.: Bi-Objective Community Detection (BOCD) in Networks using Genetic Algorithm. Contemporary Computing, Communications in Computer and Information Science 168, 5-15 (2011)
29. Hafez, A. I., Ghali, N. I., Hassanien, A. E., Fahmy, A. A.: Genetic Algorithms for community detection in social networks. In : 12th International Conference on, Intelligent Systems Design and Applications (ISDA), 2012 pp.460-465 (2012)
30. Corne, D., Jerram, N., Knowles, J., Oates, M.: PESA-II: region-based selection in evolutionary multiobjective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp.283–290 (2001)
31. Park, Y., Song, M.: A genetic algorithm for clustering problem. In : Proceedings of the 3rd Annual Conference on Genetic Programming, pp.568–575 (1998)
32. Cormen, T., Leiserson, C., Rivest, R., Stein, a.: Introduction to Algorithms. MIT Press (2001)
33. Danon, L., Diaz-Guilera, A., Duch, J., Arenas, A.: Comparing community structure identification. Journal of Statistical Mechanics: Theory and Experiment 9, 09008 (2005)

34. In: Network DataSets. (Accessed 2013) Available at: http://www.personal.umich.edu/˜mejn/netdata
35. Zachary, W. W.: An information flow model for conflict and fission in small groups. Journal of Anthropological Research 33(4), 452-473 (1977)
36. Lusseau, D.: The emergent properties of dolphin social network. Proceedings of the Royal Society of London. Series B: Biological Sciences 270(Suppl 2), S186-S188 (2003)
37. In: Stanford Large Network Dataset Collection. (Accessed 2013) Available at: http://snap.stanford.edu/data/index.html
38. McAuley, J., Leskovec, J.: Learning to Discover Social Circles in Ego Networks. In : NIPS, pp.548-556 (2012)
39. Leskovec, J. In: Social Circles in Ego Networks. (Accessed 2013) Available at: http://snap.stanford.edu/socialcircles/
40. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An Open Source Software for Exploring and Manipulating Networks. In : International AAAI Conference on Weblogs and Social Media, http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154, .2009
41. Jacomy, M., Heymann, S., Venturini, T., Bastian, M.: ForceAtlas2, A continuous graph layout algorithm for handy network visualization. Medialab center of research (2011)