Proceedings of the 5th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/22/20231220

Encoder-decoder models in sequence-to-sequence learning: A survey of RNN and LSTM approaches

Yunong Zhang

Civil Engineering College, Xi'an University of Architecture and Technology, Xi'an, China, 710054

zhang.yunong.0209@gmail.com

Abstract. In today's information age, from natural language processing to audio signal processing, from time series analysis to machine translation, the application of sequence data involves various fields. Encoder-decoder models, as a powerful approach to sequence modeling, have attracted extensive attention and research. This review paper aims to explore encoder-decoder models, focusing on the principles and operational steps of recurrent neural networks (RNN) and long short-term memory (LSTM), aiming to provide researchers and practitioners with a deep understanding of the fundamentals and applications of these models Condition. Through the analysis and summary of relevant literature, this study reveals the advantages of RNN and LSTM in sequence data processing; RNN structure is simple and effective, can process sequence input and output of any length, and can capture the time dependence in sequence data. But there is a problem of gradient disappearance, which makes it difficult to deal with long-term dependencies. To solve this problem, the LSTM model introduces different gating mechanisms, which effectively solve the gradient problem while better capturing long-term dependencies. Through the review of the principles and applications of the two, it provides a useful reference for further research and application.

Keywords: encoder-decoder model, RNN, LSTM.

1. Introduction

Encoder-Decoder model is an important deep learning model, which is widely used in natural language processing, machine translation and speech recognition and other fields. It uses two components, an encoder and a decoder, to achieve sequence-to-sequence conversion by encoding the input sequence and generating the decoding of the target sequence. In the encoder-decoder model, recurrent neural network (RNN) and long short-term memory (LSTM) are two core components and the key to how to implement the model. This article will comprehensively understand and explore the codec model and the principles and operation steps of RNN and LSTM involved in it by synthesizing various literatures. At the same time, this paper reviews recent advances in the research field and dissects the structure and composition of the model. Through a systematic review, it aims to reveal the characteristics, advantages and disadvantages of the codec model, and deeply discuss its application scenarios and development directions in natural language processing and other fields. With the continuous development of artificial intelligence and machine learning, the field of sequence data processing still faces many challenges and opportunities. How to further improve the performance of

DOI: 10.54254/2755-2721/22/20231220

the encoder-decoder model, how to deal with the processing of longer sequences, and how to combine other models and technologies are all important directions for future research. With a deep understanding of RNNs and LSTMs, researchers can explore these aspects and contribute to the development of the field of sequence data processing.

2. Introduction to encoder-decoder model

The Encoder-Decoder model is a common neural network architecture for processing sequence-to-sequence tasks such as machine translation and text generation. The model consists of two main components: an Encoder and a Decoder.

The encoder is responsible for converting the input sequence into a fixed-dimensional vector representation, capturing the semantic information of the input sequence. It can obtain a representation about the entire input sequence by stepping through each element in the input sequence and updating its hidden state. The decoder is responsible for gradually generating an output sequence based on the vector representation generated by the encoder. By using the vector representation of the encoder as the initial hidden state, and generating the output of the next time step based on the output of the previous time step and the current hidden state. The output of a decoder can be a sequence of discrete symbols (such as a word) or a continuous value (such as a speech signal).

Encoder-decoder models are widely used in natural language processing tasks, such as machine translation, text summarization, dialogue generation, etc. It can handle variable-length input and output sequences, and has certain contextual understanding and generation capabilities, enabling the model to understand and generate at the sequence level. Meanwhile, with proper design and improvement, the encoder-decoder model can also be applied to sequence-to-sequence tasks in other domains.

In the implementation of encoders and decoders, recurrent neural networks (RNN) or long short-term memory networks (LSTM) are often used as the base model; that is, the encoder-decoder model is an architecture based on these neural network structures. In order to better understand this architecture, in the second part of this article, we will introduce the principles and operation steps of RNN and LSTM in detail.

3. Principle of the basic neural network model

3.1. Model preparation

First of all, this paper needs to briefly introduce a traditional neural network model, the Multi-Layer Perceptron (MLP) model.

Multi-layer perceptron is the most basic feed-forward neural network model, as shown in Figure 1, it consists of an input layer, a hidden layer and an output layer. Each neuron is connected to all neurons in the previous layer and has connections with weights. Each neuron receives the output of neurons in the previous layer as input, and performs a nonlinear transformation on the input through the activation function. The output of the hidden layer is used as the input of the next layer until the final output layer produces the prediction result of the model.

The MLP model treats each input and output as independent, which means that it cannot directly deal with time-dependent sequence data, such as natural language text or time series data. At the same time, the number of parameters in the MLP model increases with the number of layers and the number of neurons in each layer; for complex tasks and large-scale data sets, the MLP model may require a large number of parameters, resulting in a model that is too complex and difficult to train and inference is computationally expensive. To overcome these problems, recurrent neural networks and their variants, such as long short-term memory (LSTM), were developed, which can better handle sequential data and temporal dependencies.

DOI: 10.54254/2755-2721/22/20231220

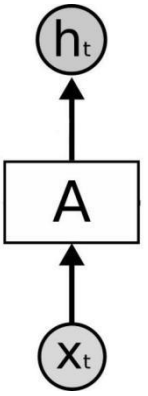

Figure 1. Schematic diagram of MLP basic unit.

3.2. RNN

RNN (Recurrent Neural Networks) is a classic neural network model, which is specially used to process sequence data modeling and prediction. On the basis of MLP, it adds the ability to deal with the time dependence of sequence data, that is, the information at the current time includes the information at the past time. This modeling of temporal dependencies of sequence data is very important in fields such as natural language processing, speech recognition, and machine translation.

The RNN model was first proposed by Elman in *Finding Structure in Time*, also known as the Elman network [1]. This model utilizes a recurrent structure to apply the network to time series forecasting tasks through time unwrapping, laying the foundation for subsequent RNN research.

As shown in Figure 2, a typical RNN unit consists of an input layer xt, a hidden layer A and an output layer ht. The input layer takes input data and combines it with the output of the hidden layer. The hidden layer contains a reflexive connection that allows information to be passed between different time steps. The output of the hidden layer is not only used as the input of the hidden layer of the next time step, but also can be used as the input of the output layer. By further connecting multiple RNN units together, a complete RNN model can be formed.

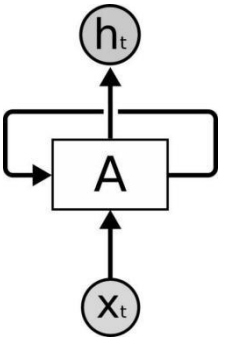

Figure 2. Schematic diagram of RNN basic unit.

However, traditional RNNs face the problems of vanishing and exploding gradients during training, which limits their ability to model long-term dependencies. In order to solve this problem, Williams et al. proposed an algorithm called RTRL (Real-Time Recurrent Learning) for training fully recurrent RNNs [2]. The algorithm enables RNNs to learn and predict in real time on continuous streams of data by computing precise gradient information.

In recent years, Cho et al. introduced the RNN encoder-decoder model and applied it to statistical machine translation tasks [3]. They proposed a framework using RNN as an encoder and decoder, using the encoder to encode the input sequence into a fixed-length vector representation, and then the decoder completes the translation by generating an output sequence.

DOI: 10.54254/2755-2721/22/20231220

Next, this article will briefly introduce the simplified process of the RNN model. For more detailed content, please refer to *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine* published by Kyunghyun Cho et al. in 2014 [3].

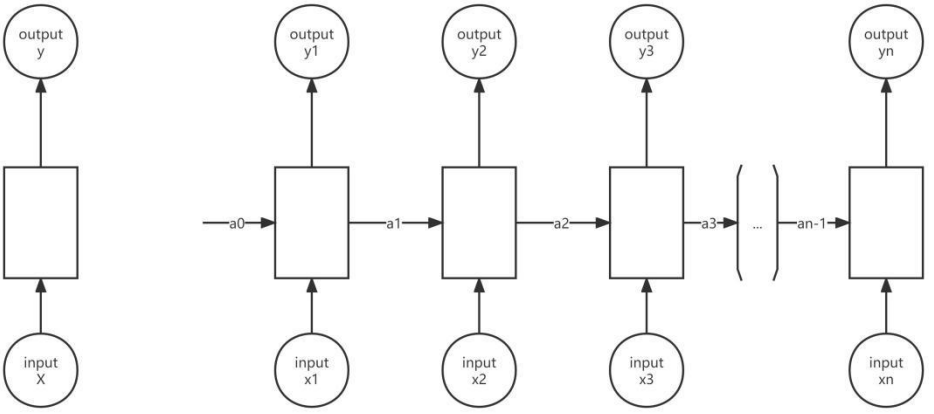

Figure 3. RNN information processing flow chart.

As mentioned above, RNN considers the time dependence, that is, the output result after the processing of the front sequence will be used as the input information of the rear sequence. Figure 3 is a schematic diagram of the RNN processing flow. Specifically, we use a_0 and x_1 as the input information of the first part, which can be obtained after processing

$$\begin{aligned} a_1 &= g_a(w_{aa} \quad a_0 + w_{ax} \quad x_1 + b_a) \\ y_1 &= g_y(w_{ya} \quad a_0 + w_{yx} \quad x_1 + b_y) \end{aligned} \tag{1}$$

Then take the result a_1 obtained in the above formula as the input of the second part, then we have

$$\begin{aligned} a_2 &= g_a(w_{aa} \cdot a_1 + w_{ax} \cdot x_2 + b_a) \\ y_2 &= g_y(w_{ya} \cdot a_1 + w_{yx} \cdot x_2 + b_y) \end{aligned} \tag{2}$$

According to this recursive relationship, the expression of the n_{th} output result can be obtained

$$\begin{aligned} a_n &= g_a(w_{aa} \cdot a_{n-1} + w_{ax} \cdot x_n + b_a) \\ y_n &= g_y(w_{ya} \cdot a_{n-1} + w_{yx} \cdot x_n + b_y) \end{aligned} \tag{3}$$

Among them, $g_a(x)$ is the activation function for the output a, $g_y(x)$ is the activation function for the output y; w_{aa}, wax are the weight coefficients of a and x corresponding to the activation function of a, respectively, similarly, w_{ya}, w_{yx} respectively is the weight coefficient of a and x in the y activation function. Both b_a and b_y are constants.

When there is an input of x, the corresponding predicted output of y will be obtained, and the predicted result of y and the real result will be calculated as cross-entropy loss, and finally the parameters will be updated using the gradient descent method until the optimal result is obtained. Completed a core processing flow about RNN.

The above is an explanation of the simplified mathematical model of RNN. It is not difficult to see that RNN has the advantages of processing sequence data, long-term dependency modeling,

DOI: 10.54254/2755-2721/22/20231220

processing variable-length input and output, context understanding and sequence generation in natural language processing, making it suitable for NLP tasks. Have a better performance. However, RNN also has some problems, such as gradient disappearance, slow training speed, etc., and subsequent improved models such as LSTM and GRU have solved these problems to a certain extent.

3.3. LSTM

LSTM (Long Short-Term Memory) is a variant of Recurrent Neural Network (RNN), specifically designed to solve the gradient disappearance and gradient explosion problems in RNN.

In the ordinary RNN structure, the front sequence information is transmitted through a parameter, and the farther the transmission distance is, the more information is lost. This situation is called gradient disappearance. The gradient disappearance problem will lead to difficulties in modeling long-term dependencies, that is, the network cannot capture earlier information in the input sequence, thus affecting the performance of RNN in processing long sequence tasks.

In response to this defect, Hochreiter proposed the basic structure of LSTM and improved it for the long-term dependency problem [4]. LSTMs control the flow of information by introducing mechanisms called "gates". These gates include input gates, forget gates, and output gates, which control the transmission and preservation of information by learning to effectively handle long-term dependencies. According to the structure diagram of LSTM shown in Figure 4, this article will briefly explain each step.

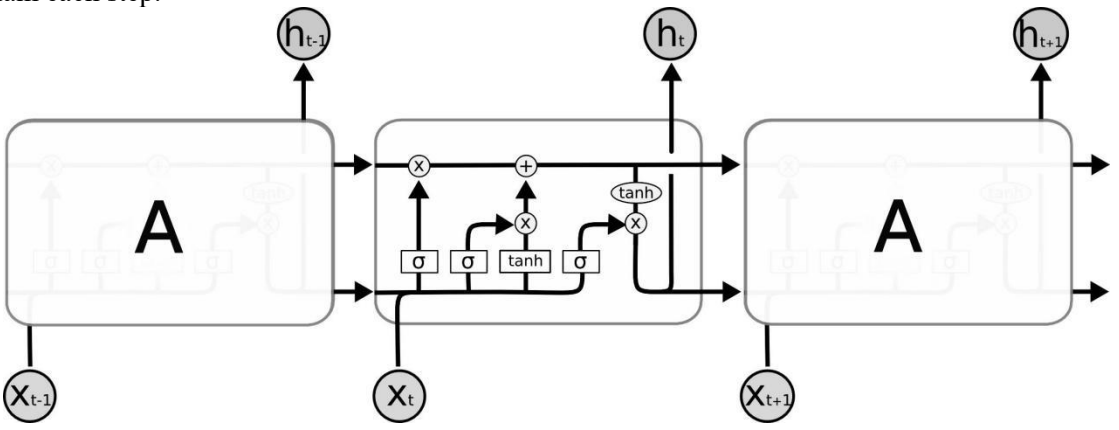

Figure 4. Schematic diagram of LSTM cell structure.

The first step in LSTM is to decide what to forget, that is, pass the forget gate. It uses a *sigmoid* function to output a value between 0 and 1, indicating the degree to which the corresponding element in each cell state is to be forgotten. The specific operation is as follows:

Step 1：Input the hidden state h_{t-1} of the previous moment and the input x_t of the current moment.

Step 2：Calculate the activation value f_t of the forget gate, and use the *sigmoid* function to act on the result of a linear transformation:

$$f_t = \sigma\left(w_f\left[h_{t-1}, x_t\right] + b_f\right) \tag{4}$$

Step 3：If f_t is close to 0, the information of the corresponding position will be completely forgotten; if f_t is close to 1, the information of the corresponding position will be completely retained. Through the forget gate, LSTM can selectively forget previously unimportant information.

The second step is to decide what to add, that is, through the input gate; the input gate consists of two parts, which are used to decide whether to update the cell state and update the value of the state. The *tanh* function is used, which can map the input between -1 and 1, and can effectively deal with the problem of gradient disappearance. The specific operation is as follows:

DOI: 10.54254/2755-2721/22/20231220

Step 1：Input the hidden state h_{t-1} of the previous moment and the input x_t of the current moment.

Step 2：Calculate the part that decides to update the cell state: use the *sigmoid* function to calculate the update gate u_t.

$$u_t = \sigma\left(w_u\left[h_{t-1}, x_t\right] + b_u\right) \tag{5}$$

Step 3：Compute new candidate cell states using the *tanh* function.

$$\tilde{C}_t = \tanh\left(w_c\left[h_{t-1}, x_t\right] + b_c\right) \tag{6}$$

Among them, w_u and b_u are the weight matrix and bias term of the update gate, and w_c and b_c are the weight matrix and bias term of the candidate cell states.

Step4：Combine the output of the two parts to update the cell state.

$$C_t = u_t \cdot \tilde{C}_t \tag{7}$$

In this process, the input gate decides whether to update the cell state by controlling the output of the update gate. If u_t is close to 0, it means that the cell state is not updated; if u_t is close to 1, it means that the cell state is completely updated. At the same time, the candidate cell state $\tilde{C}_t$ uses the tanh function to calculate a possible new cell state. Finally, the two are combined to obtain a new cell state C_t by element-wise multiplication, so that LSTM can selectively update the cell state to better capture the relevant information in the sequence.

The final step is to decide what is output, that is, through the output gate. It can weight the hidden state and decide which parts of the cell state will be output. The specific operation part is as follows:

Step 1：Input the hidden state h_{t-1} of the previous moment and the input x_t of the current moment.

Step 2：Calculate the output o_t of the output gate; use the *sigmoid* function to calculate the output gate o_t.

$$o_t = \sigma\left(w_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{8}$$

Step 3：Process the cell state C_t through the output gate o_t and *tanh* functions to obtain the hidden state h_t at the current moment.

$$h_t = o_t \cdot C_t \tag{9}$$

The output gate calculates a value o_t between 0 and 1 through the *sigmoid* function, indicating which parts of the cell state will be output. The *tanh* function maps the cell state C_t to a value ranging from -1 to 1, and then multiplies it with the output gate o_t by element-wise multiplication to obtain the hidden state h_t at the current moment.

The above are the core steps of LSTM. Through the adjustment of the forget gate, input gate and output gate, the LSTM model can effectively handle long-term dependencies and perform well in sequence modeling tasks. Each gate structure adaptively decides the retention and forgetting of information by learning parameters, so as to realize the modeling of long-term dependencies in sequences, which also enables it to better capture and utilize time dependencies when processing sequence data.

Graves et al. explored the use of RNNs and LSTMs for sequence labeling tasks [5]. He presented

DOI: 10.54254/2755-2721/22/20231220

experimental results using LSTMs for character-level annotation and speech recognition, and demonstrated the potential of LSTMs for sequence modeling. This paper further confirms the advantages of LSTM in processing sequence data and solving gradient problems.

Greff et al. conducted a comprehensive study and analysis of LSTM [6]. They provide an in-depth understanding of LSTM architecture and parameter settings by conducting experiments and analysis on the search space of LSTMs. The paper discusses the effects of various LSTM variants and parameter tuning, and how to optimize the performance of LSTM models.

4. Practical application

4.1. Advantages and disadvantages

4.1.1. Recurrent neural networks. RNN has the advantages of simple structure and processing sequence input and output of arbitrary length, which makes it suitable for various sequence modeling tasks. At the same time, RNN is able to capture temporal dependencies in sequence data, which makes it perform well in processing sequence data.

However, RNNs also have some disadvantages. The most notable of these is the vulnerability to vanishing or exploding gradients when dealing with long-term dependencies. For example, when training a language model to generate sentences, the model needs to capture long-term dependencies in the sentence, such as the topic or context information of the sentence. Due to the vanishing gradient problem of RNN, the relevance of distant words to the current position gradually weakens, making it difficult for the model to effectively capture these long-term dependencies, resulting in generated sentences that lack contextual consistency or coherence in meaning. Pascanu et al. explored the training difficulty of RNN, especially the problem of gradient disappearance and gradient explosion, and proposed some solutions [7].

4.1.2. Long short-term memory. LSTM effectively solves the gradient problem in traditional RNN by introducing the mechanism of input gate, forget gate and output gate, so that the gradient can be better transmitted in time; this enables LSTM to better capture long-term dependencies and is suitable for Tasks that require long-term memory.

Of course, as a powerful sequence modeling tool, LSTM usually needs to store a large number of parameters and intermediate states, so it takes up a large memory space. Memory requirements can be a challenge for devices with limited resources or environments that need to run multiple models simultaneously. Second, LSTM also requires long computation time in the inference stage (i.e. using the trained model for prediction or generation). Especially when the input sequence is long or real-time data needs to be processed, the reasoning time of LSTM may increase, which may have an impact on application scenarios with high real-time requirements. And when the training data is too small or there is noise, the LSTM model may over-adapt to the noise or irrelevant information in the training data, resulting in the problem of over-fitting.

4.2. Application scenario

4.2.1. RNN. RNN has a wide range of applications in language learning, language modeling, dynamic system modeling, speech recognition, handwriting recognition, named entity recognition and other tasks. It is able to process sequence data with temporal dependencies and capture structure and patterns in the sequence. For example, Graves et al. extended the application of RNNs to sequence labeling tasks [5]. This study proposes methods for supervised sequence annotation using RNNs, such as speech recognition, handwriting recognition, named entity recognition, etc. By applying RNN to the classification or labeling tasks of sequence data, the model can automatically capture the context information and time dependencies of the input sequence, thereby improving the accuracy and performance of sequence labeling.

DOI: 10.54254/2755-2721/22/20231220

Taken together, the capabilities of RNNs make them widely used in processing time-series data, natural language processing, speech recognition, and other tasks that require modeling sequence data.

4.2.2. LSTM. The LSTM model is also widely used in tasks such as sequence prediction, handwriting recognition, and speech recognition. Compared with the traditional RNN model, it has obvious advantages in dealing with long-term dependencies, sequence classification and labeling, and natural language processing tasks.

Sundermeyer. et al. discussed the method and results of language modeling using long short-term memory (LSTM) neural network [8]. In this study, the author proposed a framework for language modeling using LSTM model and compared it with the traditional n-based models and traditional recurrent neural networks. They conducted experiments using a large-scale corpus, and the results showed that the LSTM model achieved significant improvements in language modeling tasks. Compared with the traditional n-gram model and the traditional recurrent neural network, the LSTM model can better model long-term dependencies and improve the accuracy of language modeling.

Besides, Karpathy et al. proposed a method using LSTM codec model to generate image captions [9]. The model works by encoding an image into a feature vector and then using an LSTM decoder to generate a textual description associated with the image. The LSTM model can handle the timing of the input sequence and learn the semantic correspondence between images and descriptions. Trained on a large-scale image and description dataset, the model is able to generate textual descriptions that are relevant to image content and somewhat creative.

These studies all demonstrate the application of LSTM models in processing sequence data generation tasks. By learning the context and dependencies of the input sequence, LSTM is able to generate a continuous sequence of text that matches the input. This application scenario is not limited to image description generation, but can also be extended to other text generation tasks, such as machine translation, dialogue systems, etc.

5. Future development

The first is deep learning. Deep learning has achieved great success in various fields. Applying deep learning methods to encoder-decoder models is also a promising direction. Design a deeper neural network structure, such as a deep LSTM or RNN model, to improve the expressiveness and learning ability of the model.

Second, the encoder-decoder model has potential in handling multimodal data (such as a combination of images and texts) and multi-task learning. Future research can explore how to integrate multiple input modalities or multiple tasks into an encoder-decoder model to increase model diversity and flexibility.

Finally, self-supervised learning and reinforcement learning are currently hot research areas and have great application potential in encoder-decoder models. By introducing the idea of self-supervised learning or reinforcement learning, the data utilization efficiency and learning ability of the model can be improved, thereby further improving the performance of the encoder-decoder model in various tasks.

In general, the encoder-decoder model has great development prospects in the fields of natural language processing and machine translation. Through continuous innovation, improvement and introduction of new technologies, we can expect the emergence of more powerful, efficient and adaptable encoder-decoder models for various tasks, bringing better effects and benefits to various practical applications.

6. Conclusion

This review paper mainly discusses the encoder-decoder model, and focuses on the principle and operation steps of recurrent neural network and long short-term memory. Through a comprehensive analysis of related literature, it concludes that RNN and LSTM, as the core components of the encoder-decoder model, can effectively handle long-term dependencies in sequence data. Their gating

DOI: 10.54254/2755-2721/22/20231220

mechanism and memory unit equip them with the ability to capture temporal dependencies and process long-term memory, making them excellent in natural language processing tasks. Future research can combine other deep learning techniques, such as attention mechanism and Transformer model, to further improve the ability of sequence modeling and generation. In conclusion, this paper presents a comprehensive overview of RNNs and LSTMs in encoder-decoder models, revealing their important roles and application prospects in sequence data processing. As the field of deep learning continues to evolve, we are confident that we will see more innovations and advancements in the future, driving the application of encoder-decoder models in natural language processing and other fields.

References

[1] Elman J L. Finding structure in time[J]. Cognitive science, 1990, 14(2): 179-211.
[2] Williams R J, Zipser D. Experimental analysis of the real-time recurrent learning algorithm[J]. Connection science, 1989, 1(1): 87-111.
[3] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.
[4] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.
[5] Graves A, Graves A. Supervised sequence labelling[M]. Springer Berlin Heidelberg, 2012.
[6] Greff K, Srivastava R K, Koutník J, et al. LSTM: A search space odyssey: IEEE Transactions on Neural Networks Learning Systems[J]. 2017.
[7] Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks[C]//International conference on machine learning. Pmlr, 2013: 1310-1318.
[8] Sundermeyer M, Schlüter R, Ney H. LSTM neural networks for language modeling[C]//Thirteenth annual conference of the international speech communication association. 2012.
[9] Karpathy A, Fei-Fei L. Deep visual-semantic alignments for generating image descriptions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 3128-3137.