Review on "Learning to Reason: End-to-End Module Networks for Visual Question Answering"

Irdi Balla

31/10/2017

1 Introduction

For Visual Question Answering understanding both text and images in necessary. Deep networks have shown good performance but still they lack the ability to use compositional reasoning. One of the reasons is that they rely on the same network structure for all inputs. N2NMNs try to tackle that by predicting network architecture from the questions. This way N2NMNs create the structures for the language and also find the best layout.

2 Related Work

2.1 Neural Module Networks (NMN)

NMNs are recursive Neural Networks. Every input has a layout that provides a template for building the network. Later works on NMNs, Dynamic NMNs, rank several layouts instead of coming with a fixed one. N2NMNs on the other hand, perform better as they optimize over the full space of layouts.

2.2 Learning network Architecture

A line of research has focused on discovering architectures from data using Reinforcement Learning and Bayesian methods. The drawback to these methods is that they apply a fixed layout to every instance while N2NMNs dynamically predict a layout that suits best every individual instance.

2.3 Visual Question Answering (VQA)

VQA is often used to measure how well models reason on text and image at the same time. The approaches used so far seem to perform better every time but it is unsure if these model are reasoning or just learning the data or statistics about them. To address these problems new dataset are created such as SHAPES and CLEVR.

3 End-to-End Module Networks

N2NMNs consist of 2 main components: a set of co-attentive modules and a layout policy. The first one provide the parametrized functions to solve the subtasks while the layout policy predicts how the networks should be built.

3.1 Attentional neural modules

A module is a parametrized function. It takes tensors as input (none, one, or several) which together with the features from the image and question, and its internal parameters provide the output tensor.

$$y = f_m(a_1, a_2...; x_{vis}, x_{txt}; \theta_m) \tag{1}$$

The input is an image attention map while the output can either be an image attention map or a probability distribution. The features from the image is a spatial feature map obtained with a CNN, while the features of the question is a textual vector obtained from the question. Modules that can be use are: *find* (to localize objects), *relocate*(transforms the input attention map) *and/or* (take 2 tensors as input and return intersection/union of them), *filter*(reuses *find* and *and* to simplify the layout), *exist*, *count* and *describe* (used to infer answer from 1 attention map) while *eqcount*, *more*, *less* and *compare* are use when more that 1 attention map is given as input. For each module an attention map is predicted over the question words and a textual feature is obtained

$$x_{txt}^{(m)} = \sum_{i=1}^{T} \alpha_i^{(m)} \omega \tag{2}$$

3.2 Layout policy with sequence-to-sequence RNN

A layout policy outputs a probability distribution, where the highest probabilities belong to layouts that are effective for the task at hand. A layout can be mapped one-to-one to a linearized sequence. And after every layout is linearized the problem turns into a sequence-to-sequence learning problem. This problem is solved using attentional RNN. First every word is embedded into a vector, and a multi-layer LSTM is used as the encoder of the input question. The decoder is a similar LSTM but with different parameters. At every time-step the attention weights are calculated as follows:

$$\upsilon_{ti} = \nu^T tanh(W_1 h_i + W_2 h_t) \tag{3}$$

$$\alpha_{ti} = \frac{exp(\upsilon_{ti})}{\sum_{j=1}^{T} exp(\upsilon_{tj})} \tag{4}$$

A context vector is

$$c_t = \sum_{i=1}^{T} \alpha_{ti} h_i \tag{5}$$

We then use it to predict the next module token for which we construct the textual input as well. At test time the best layout is chosen and then we use it to assemble a neural network.

3.3 End-to-end training

During training, the layout policy and the parameters in each module are optimized by minimizing the loss. Since the layout is discrete the loss function is not fully differentiable. Back-propagation is used for the differentiable parts while for the rest policy gradient method is used. TO encourage exploration an entropy regularization is used. Optimizing the loss function from scratch is problematic. To solve that the model can be pre-trained using an expert policy(only at training, not during testing). But the optimal policy is not always optimal as the model should still be able to cover more layouts than those of the expert policy.