Enhancing Stack Neural Module Networks for NLVR2

Erez Sheffi [1 2] **Tal Lancewicki** [1 3] **Or Shahaf** [1 4]

Abstract

We focus on the task of reasoning about natural language and images. Specifically, we address the NLVR2 dataset in which we're tasked to classify a statement about two realistic images as true/false. In this paper we intend to adapt Stack Neural Module Networks to NLVR2 and enhance its performance.[1]

1. Introduction

Neural Module Networks (henceforth, NMN) were first introduced by Andreas et al. (2016) for the task of visual question answering. NMN dynamically assembles few neural networks, referred to as *modules*, to compose a layout with a suitable structure for a given question. The idea behind this method is that different models might be better suited for different kind of questions, and so it could be useful to determine the model's structure dependent on the question's type. At the same time, these different models might share some sub-structure and so it is more efficient to use the same "building blocks" (modules) in order to construct the models. That way, we train only few modules and effectively can construct infinite number of layouts from these modules. Andreas et al. (2016) use a pre-trained dependency parser that outputs structured representation of the question which is then used for constructing the layout using rule-based method.

Hu et al. (2017) build on the work of Andreas et al. and develop End-to-End Module Networks (henceforth, N2NMN) which learn the layout controller from scratch (without using a pre-trained model) using reinforcement learning. However, in order to achieve good performance, an expert-layout is required (i.e. a ground-truth parsing of the question). For that, they pre-train the layout controller using the expert-layout and then train the entire model while initializing the controller weights from the pre-training step.

Stack Neural Module Networks (henceforth, SNMN) (Hu et al., 2018) makes the model layout a continuous choice, instead of discrete as in NMN and N2NMN. This is done recurrently, where in each time step the layout controller assigns weights to each module, all modules are propagated, and the output is their weighted average. The output is then pushed to a stack-like differentiable memory pool so the modules in the next steps can pop it as their input. This results in an end-to-end differentiable model which can be optimized using gradient descent. The authors achieve near-state-of-the-art performance on CLEVER and VQA datasets even without using expert-layout

In this work, we adapt SNMN for the NLVR2 dataset (Suhr et al., 2018). Each sample in the dataset contains a pair of images and a related statement. The goal is to classify the statement as true/false. Each sentence appears in the dataset multiple times with different images and labels, with an almost perfect balance between labels for each sentence. This ensures that the correct label can not be predicted using the sentence only. Both sentence-only and image-only benchmark algorithms (i.e. algorithm that considers only the sentence or the images, respectively) achieve almost identical performance as simply assigning the most common label (approximately 51% accuracy on DEV and TEST sets). This is in contrast to some other related datasets (Manjunatha et al., 2019), such as VQA (Antol et al., 2015). Generally, NLVR2 is considered a challenging reasoning task and contains coordination between images, soft cardinality ("there are less than 8 bottles in total"), comparisons ("there are 4 balloons of different color"), and more. Many existing models that was designed for visual question answering and adapted to NLVR2 fail to perform well on the dataset (again, approximately 51% accuracy). This includes Film (Perez et al., 2018), MAC (Hudson & Manning, 2018) and N2NMN (Hu et al., 2017). We believed that those models straggle with this task for a few reasons. First, as mentioned before, NLVR2 contains some challenging statements that require complex reasoning skills. Second, the model should be adapted for handling two images, and the most trivial way of doing so is not necessarily the right way. Lastly, the binary signal of NLVR2's ground-truth labels is quite poor compared to other tasks of reasoning from image and text such as CLEVR (Johnson et al., 2017), VQA (Antol et al., 2015), GQA (Hudson & Manning, 2019) and others, which

[1]Tel-Aviv Univeristy [2]123456789 [3]305772394 [4]312491822. **AUTHORERR: Missing \icmlcorrespondingauthor.**

Final Project in Advanced Methods in Machine Learning and Natural Language Processing

[1]Code models publicly available at: https://github.com/sheffier/ml_nlp_vqa

makes the learning process more challenging. In this paper we try to address these three issues and achieve better performance results. We evaluate performance by training on the TRAIN set using Adam. After each epoch we evaluate the model's accuracy over the DEV set, and for last we take the model that achieved best accuracy on the DEV set and test it on the TEST set.

In Section 2 we briefly describe the original SNMN model. In section 3 we describe how we adapt it for handling two images. In section 4 we try to apply a curriculum learning method (Mao et al., 2019) as an attempt of dealing with the complexity of the statements. Section 6 give an idea for future work. Section 5 is conclusions.

2. Background: Stack Neural Module Networks

This section briefly describes SNMNs. A detailed description can be found in (Hu et al., 2018). As mentioned before, SNMN runs sequentially on each sample, and consists of three components. The *layout controller* depends on the question, constructs a soft layout and supplies textual parameters to the modules at each time step, which represent a certain part of the question. The *Neural modules* are each in charge of reasoning sub-task. A differentiable *memory stack* that stores modules outputs which can be popped in next steps.

2.1. Layout Controller

At each time step t the layout controller makes a soft choice of which module to run, by assigning weight to each module. In addition, it generates a textual parameter c_t. It first executes a bi-directional LSTM on the input question to create a d-dimensional sequence $q = [h_1, ..., h_S]$ where S is the number of words in the input question. Then, it concatenates c_t with a time dependent linear transformation of q, and applies another linear transformation: $u = W_2 \left[W_1^{(t)} q + b_1; c_{t-1}\right] + b_2$ where u is d-dimensional. In order to compute module weights, a multi-layer perceptron (MLP) is then applied to u to predict module weights: $w^{(t)} = \text{softmax}\left(MLP\left(u\right)\right)$. Lastly, c_t is a linear combination of $h_1, ..., h_S$ where the coefficients defined by $cv_s = \text{softmax}\left(w^T \left(u \odot h_s\right)\right)$ where w is d-dimensional. So $c_t = \sum_{s=1}^{S} cv_s$.

2.2. Modules

There are several types of modules. Modules might use images' visual features (those are extracted using ResNet-152 (He et al., 2016)) or the textual parameter supplied by the controller. Modules might as well get, as an input, *image attention* which are outputs of modules from previous time step. The output is either an attention or scoring over possible answers. The module `FIND` for example, gets no attention and uses the visual features and textual parameter in order to output an attention that is related to the textual parameter. The module `COMPARE` gets two attentions (e.g. matching 2 bottles on one of the images), and uses the textual parameter (e.g. representing the word ”size”) and visual features to produce a distribution over possible answers (in NLVR2 - true or false; in this example: is the first dog larger than the second). A list of all modules and their detailed description appears in Table 1 in (Hu et al., 2018).

The final scoring function is summed across time-steps and averaged according to modules weights. Formally, if M_{ans} is the set of modules that output answers and $y_m^{(t)} \in [0,1]^2$ is the scoring for the answers of module m at time t, then $y = \sum_{t=0}^{T-1} \sum_{m \in M_{ans}} y_m^{(t)} w_m^{(t)}$ is the final score for each answer.

2.3. Differentiable memory stack

The stack consists of an array $A = \{A_i\}_{i=1}^{L}$ which represents L attentions (L is the stack size) and a stack-top pointer p which is an L-dimensional distribution. Push operation increments the stack pointer by $p = \text{1d\_conv}\left(p, [0,0,1]\right)$ and writes the new attention z to the stack by $A_i := \left(1 - p_i\right) A_i + z \cdot p_i$. Pop operation reads an attention $z = \sum_{i=1}^{L} A_i p_i$ and decrements the pointer by $p = \text{1d\_conv}\left(p, [1,0,0]\right)$. However, different modules may change the stack differently (e.g. `FIND` only pushes to the stack and `COMPARE` pops twice) and so we average the stack pointer and array according to modules weights at that time step. Formally, at time step $t = 0$, the stack's array is initialized to zero attention and p is initialized to point to the button of the stack (i.e. 1 at the first coordinate and 0 in all others). At time step $t > 0$, let $\left(A_m^{(t)}, p_m^{(t)}\right)$ be the stack after running module m. Then $A^{(t+1)} = \sum_{m \in M} A_m^{(t)} w_m^{(t)}$ and $p^{(t+1)} = \text{softmax}\left(\sum_{m \in M} p_m^{(t)} w_m^{(t)}\right)$.

3. Adapting Stack Neural Module Networks for NLVR2

In this section we deal with the fact that NLVR2 contains two images per sample. We start with a simple solution, taking it as a benchmark, and proceed with more complex methods.

3.1. Benchmark: Images Concatenation

As mentioned in Section 1, Suhr et al. (2018) tested the performance of N2NMN on NLVR2. They did by concatenating the two images into a single image. We take this solution as benchmark and apply it to SNMN. This results in 58.5% accuracy on the test set.

3.2. Features Concatenation

Instead of concatenating the images and extracting features of the concatenated image, we believe that it makes more sense extracting features separately from each image and then concatenating the feature map, as the ResNet was trained in a single image and outputs 7×7 feature map. As a result, the setting of 3.1 ends up with a feature map that's effectively 7×3.5 for each image. Moreover, the middle column in each feature in 3.1 represents information that is a mix of pixel values from both images. By concatenating the features themselves we end up with a 7×14 feature map. Those features are treated now as single image and fed to the modules regularly. This results in 61% accuracy on the test set.

3.2.1. Adding left-image and right-image Encoding

Still, we do not have something that distinguishes the right image from the left one. A possible solution is to add a feature that indicates whether a pixel belongs to the left or right image. More precisely, the ResNet extracts 2048 features and those are fed into a ConvNet in our model that reduces it to 512 feature. So instead of reducing it to 512, we reduce it to 511 and add another feature that is 1 in all pixels that belong to the left image and -1 in all pixels that belong to the right image. This results in 59.1% accuracy on the test set, namely no improvement in performance is observed.

Let us examine the operation of one of the modules in this setting. Consider the module `FIND`:

$$a_{out} = \text{conv}\left(x \odot Wc\right)$$

$c \in \mathbb{R}^d$ is the textual parameter, $W \in \mathbb{R}^{512 \times d}$ is a weights matrix, $x \in \mathbb{R}^{7 \times 7 \times 512}$ are the image features after the ConvNet and the concatenation of the left-right encoding, $\odot$ is element-wise product with each pixel of x (each of those have 512 features), and *conv* is a ConvNet that outputs a 7×14 attention. That way, the last coordinate of Wc would contain the information of whether we need to refer to the left image, the right image or both. This is then multiplied by ± 1 and by its ConvNet's coefficient, and summed up together with the other features (multiplied by their coefficients). What we really want is some multiplication effect over the features rather than the summation effect that we have now. This leads us to the idea we implemented in the next sub-section (3.2.2).

3.2.2. Left-image and right-image Attention

Instead of distinguishing the left image from the right by adding a feature that indicates that, we calculate a scalar $p \in [0, 1]$ that depends on c and represents the attention that we want to give to the left image. We multiply each pixel in the left half of x by p and each pixel in the right part of x by $1-p$. More precisely, we add two vectors of variables, c_{left} and c_{right} which have the same dimension as c. Loosely speaking, c_{left} would represent the value of c when it is on the context of "left image"[2] and similarly for c_{right}. That way, whenever we're in the context of the left image, the value of $c^T c_{left}$ (a scalar) would be considerably high. We calculate p by

$$(p, 1-p) = \text{softmax}\left(c^T c_{left}, c^T c_{right}\right)$$

We applied this method to the `FIND` module and later also to the `TRANSFORM` module. Those achieved 60.9% and 60.1% accuracy on the test set, respectively. We believed that the rest of the modules should not use the left-right attention as those already get attention maps that were produced by `FIND` and `TRANSFORM`.

3.3. Double Stack Neural Module Networks

We designed Double Stack Neural Module Networks (henceforth, DSNMN) to separate the image features. Each module gets visual features of a single image (i.e. not concatenated as in Section 3.2). Basically, we duplicate all modules and the stack. At each time step t, temporary modules weights $\overline{w}^{(t)}$ are computed as before by the controller, but in a similar way we also calculate two scalars $w_{left}, w_{right} \in [0, 1]$ that decide the probability of each image to be processed in the current time step. The final module weights in the left stack are $w^{(t)}_{m,left} = \overline{w}^{(t)}_m \cdot w_{left}$ for $m \neq$ NoOp and $w^{(t)}_{\text{NoOp},left} = 1 - \sum_{m \neq \text{NoOp}} w^{(t)}_{m,left}$, and similarly for the right stack. Both sides run separately (each with its own stack) for T steps. The results of both sides are then passed through 2 fully connected layers that produce the final answer distribution.

The code for this model is at branch `dsnmn`.

4. Curriculum Learning

Many of NLVR2's statements are quite challenging to reason about. Take for example the sentence *"There are two rolled towels one blue and one pink"*. Before the model learned the visual concept (e.g. shapes, colors, etc.) of "towels" and "blue"/"pink", it is unlikely that the model will have the ability to learn something from this sample. Inspired from (Mao et al., 2019), we adopt a *Curriculum Learning* approach. This is motivated by human learning.

[2]For example, for the statement "The left image contains a dog", we wish to first run `FIND` with the context that represents ["left image", "dog"]. In that case we would expect that c will be similar to c_{left}.

Without knowing how a towel looks like, it is specially hard to identify two of them and therefore we wish to first learn over simple questions in order to learn visual concepts, and only later train on the more complex samples. For that we heuristically prune the dataset, and omit all samples that contain numbers or ”all” or ”both” or ”single”. [3] This leaves us with approximately 25K TRAIN samples, and approximately 2K samples on each of the DEV and TEST sets. We pre-train on that dataset and train on the whole dataset, while initializing model's weight from the pre-training step. Pre-training on this dataset achieves 58.29% accuracy on DEV set, which is less than the best accuracy on the whole dataset. There are two effects: on one side, the training set is now significantly smaller which makes it harder for the model to generalize to samples that are not in TRAIN. On the other side, we kept only the ”easy” questions (in the DEV set as well) which we expect that should increase accuracy. However, it seems that the former's effect is more significant and so we did not expect, at this point, to obtain better performance after training. In fact, after training on the whole dataset we achieve only 56.5% and 56.7% accuracy on DEV set and TEST set, respectively. Which is smaller than the accuracy of training regularly. We find it quite peculiar, but it seems that the starting point of the pretraning lead us to a starting point that end up in a point that generalizes worse than the one we achieve from random initialization.

5. Conclusions

In this paper we adapt SNMN for NLVR2. Table 1 summarizes our results. The models outperformed all other models published in the NLVR2 leaderboard at the time the dataset was published. [4] However, we believe that more work could be done to boost performance.

6. Future Work

As we mentioned in Section 1, we believe that part of what makes this learning task challenging is the weak signal that the model gets from the ground-truth labels, as there are only two labels. Tan & Bansal (2019) deal with that by adding a pre-train step that allows them to train most of their model's weights on a different task. For example, one of the tasks that (Tan & Bansal, 2019) adopt, is reconstructing masked words from a sentence, using the image and the masked sentence itself. In figure 1 there is an example of such masked sentence and a pair of images from NLVR2. The fact that there are bottles in the images gives a clue that the masked word is ”bottles”. Hopefully, during pre-training the model will learn to transfer knowledge between textual and visual concepts and align their representations. We believe that a similar task can be found in order to pre-train our model, so that later at training step, weights will be initialized from the pre-training step, at point in which it has already learned some relation between visual and textual concepts.

[3] Originally, we restricted the statement to be up to 20 words long but this kept us with less than 10K which seems too small for training the neural net.

[4] The full leaderboard can be found here: http://lil.nlp.cornell.edu/nlvr/

Table 1. Model's best accuracy on VAL set and TEST accuracy

SETTING	BEST VAL	TEST
IMGCONCAT	58.1%	58.5%
FEATCONCAT	59.7%	61%
LEFT-RIGHT-ENCODING	58.8%	59.1%
IMATT	60.1%	60.9%
DSNMN	60.0%	61.0%
CURRICULUM	56.5%	56.7%

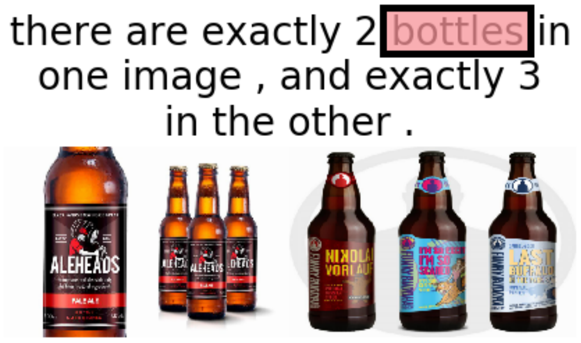
there are exactly 2 bottles in one image , and exactly 3 in the other .

Figure 1. An example of a pre-train sample for the task of reconstructing masked words. The word ”bottles” is masked at the input

References

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., and Parikh, D. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE* *conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 804–813, 2017.

Hu, R., Andreas, J., Darrell, T., and Saenko, K. Explainable neural computation via stack neural module networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 53–69, 2018.

Hudson, D. A. and Manning, C. D. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*, 2018.

Hudson, D. A. and Manning, C. D. Gqa: a new dataset for compositional question answering over real-world images. *arXiv preprint arXiv:1902.09506*, 2019.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2901–2910, 2017.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Manjunatha, V., Saini, N., and Davis, L. S. Explicit bias discovery in visual question answering models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9562–9571, 2019.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Suhr, A., Zhou, S., Zhang, I., Bai, H., and Artzi, Y. A corpus for reasoning about natural language grounded in photographs. *arXiv preprint arXiv:1811.00491*, 2018.

Tan, H. and Bansal, M. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490*, 2019.